Service Restoration In Electric Power Distribution Networks

Magnus Bergstrand

Department of Automatic Control
Lund Institute of Technology
October 1989
SERVICE RESTORATION
IN
ELECTRIC DISTRIBUTION
NETWORKS

by
Magnus Bergstrand
Lund Institute of Technology
Lund Sweden
1989-10-01

ABSTRACT
Automatic service restoration in distributional networks has been studied. After a thorough examination of the literature in this field, one algorithm was selected and implemented as a Fortran program. This implementation was then proved to be effective. Further improvements to the algorithm were introduced and in three cases these proved to be successful. For example: An operational aid indicating roughly the order of the switching operations was added, the number of switching operations decreased in some cases and the priority of the zones was made more important.
Department of Automatic Control
Lund Institute of Technology
P.O. Box 118
S-221 00 Lund Sweden

Author(s)
Magnus Bergstrand

Supervisor
Rolf Johansson, LTH, Börje Nilsson, ABB

Sponsoring organisation

Title and subtitle
Service Restoration in Electric Power Distribution Networks.

Abstract
Automatic service restoration in distributional networks has been studied. After a thorough examination of the literature in this field, one algorithm was selected and implemented as a Fortran program. This implementation was then proved to be effective. Further improvements to the algorithm was introduced and in three cases these proved to be successful. For example: An operational aid indicating roughly the order of the switching operations was added, the number of switching operations decreased in some cases and the priority of the zones was made more important.

Key words

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title

Language | Number of pages | Recipient’s notes
--- | --- | ---
English | 143 | 

Security classification

The report may be ordered from the Department of Automatic Control or borrowed through the University Library 2, Box 1010, S-221 03 Lund, Sweden, Telex: 33248 lubbis lund.
ACKNOWLEDGEMENTS

This report was written as an examination paper at the Lund Institute Of Technology. The work was performed at ABB Network Control. The author wishes to thank Dr Rolf Johansson Lund Institute of Technology department of Automatic Control, Borje Nilsson ABB Network control and other personnel at ABB Network Control for their support and good advise.
# TABLE OF CONTENTS

1. BACKGROUND .................................................. 4  
   1.1 TASK .................................................. 4  
   1.2 HISTORY ............................................... 4  
   1.3 DISTRIBUTION NETWORKS .............................. 4  
   1.4 RESTORATION ......................................... 4  
   1.5 OPTIMIZATION ASPECTS ............................... 5  

2. INFORMATION RETRIEVAL ...................................... 6  
   2.1 EXPERT SYSTEM APPROACH ............................ 6  
   2.2 MATHEMATICAL APPROACH .......................... 7  
   2.3 SELECTION ........................................... 8  

3. IMPLEMENTATION ............................................ 9  
   3.1 ALGORITHM ........................................... 9  
   3.2 DATASTRUCTURE .................................... 14  
   3.3 NETWORK REPRESENTATION ........................ 14  
   3.4 COMPUTATIONS ..................................... 14  
   3.5 IMPROVEMENTS ....................................... 15  
   3.6 TESTRUNS ............................................ 16  

4. RESULTS .................................................... 20  

5. FUTURE ..................................................... 21  

6. REFERENCES ................................................ 22  

7. APPENDIX .................................................. 23  
   7.1 USER'S MANUAL ....................................... 24  
   7.2 EXAMPLES ............................................ 28  
   7.3 OPTIMIZATION ....................................... 34  
   7.4 STRUCTURE CHART ................................... 36
1. BACKGROUND

1.1 TASK

The task has been to study the existing methods of automatic service restoration in distribution networks. Furthermore one algorithm should be selected, and implemented in Fortran.

1.2 HISTORY

Automatic service restoration is a new area in electric power distribution. Nothing was published in this area until 1980. In 1980 Ross (1) defined the service restoration problem as the minimization of the weighted sum of unserviced customer hours and unserviced energy subject to line capacity constraints, transformer constraints and voltage drop constraints. Since then different methods have been developed. Today two different approaches stand out, one mathematical and one based on expert system technique.

1.3 DISTRIBUTION NETWORKS

The distribution network has a radial structure, looking similar to that of a tree. In the feeding substation, there is a transformer. The transformer is connected to a feeder via a bar and a feeder breaker. Branchpoints give the treelooking structure. Several switches (cut switches or tie switches) are included, creating zones (or nodes or sections) in the network. These sectionalizing switches makes it possible to isolate a fault so that a repair can be done. Because of the radiality of the network most of the faults not only causes the faulted zone to be deenergized but also all the nonfaulted zones further down viewed from the transformer. These zones can often be supplied from other transformers. Therefore service restoration is needed.

Distribution networks are often large. The normal size is about 100-1000 feeders, each one consisting of 10-500 nodes (varying much between different power companies). The voltage in these networks is in the range from 4-43.5 kV.

1.4 RESTORATION

When a fault has occurred a protective relay trips the circuit breaker. Next the fault has to be identified. One way to do this is by first opening all the sectionalizing switches and then closing the circuit breaker. The switches are then reclosed one by one in until the circuit breaker is tripped again. The faulted zone is identified and has to be isolated. At this point the operator must decide a sequence of switching operations that have to be performed
in order to restore the nonfaulted deenergized zones, without causing any
violation. This is done by hand calculation based on a loads forecast or by
the operators’ experience. Often is the sequence thus produced not optimal.
A better solution would certainly be reached if a computer would produce it.

1.5 OPTIMIZATION ASPECTS

Studies of the service restoration problem show that it is a large scale
combinatorial optimization problem. The only known method for solving com-
binatorial optimization problems is to test all possible solutions. Because of
the size of the problem this solution often cannot be reached in the lifetime of
a computer. In a network containing 1000 switches, there are $10^{300}$ possible
solutions.

The most common method for solving large scale combinatorial optimization
problems is to use a heuristic search method. This means that knowledge
about the problem is used to guide the search. In this case a possible rule might
be that all the circuit breakers are left closed unless the fault has occurred in
a transformer or a bus. By applying rules like this the number of possible
solutions reduce and the execution time becomes reasonable. By rejecting
possible solutions that seem to lead to a non-optimal solution one may loose
the true optimal solution. Instead an approximate optimal solution is reached.
This solution is often so good that it is unnecessary to find the true optimal
solution. Such is the case in service restoration. Heuristic search methods are
suitable both for ordinary programming languages such as fortran and symbol
manipulating languages such as lisp or prolog.
2. INFORMATION RETRIEVAL

During my information retrieval I found two main approaches, which will be dealt with separately.

2.1 EXPERT SYSTEM APPROACH

The most common approach to service restoration is an expert system although the other methods are faster. This is due to the 'higher' programming level in lisp or prolog. It is easier to make changes in your algorithm using these languages.

Morelato (2) used the expert system approach providing a framework for a whole family of algorithms concerning distribution network automation. He proposed a true expert system search approach, using a decision tree to solve different problems such as service restoration, the minimization of losses and costs, load balancing or combinations of the above. In fact, service restoration is a special case of load balancing. If the deenergized zones are connected to a support feeder creating a violation and then applies the load balancing algorithm, a service restoration algorithm would have been used. Any violations that may exist have to be cut off in the end. In an example he demonstrated how his service restoration algorithm works. This approach didn't seem right for fortran programming and besides there were some other weaknesses. For one thing voltage drop limits were discarded.

Lee and others(3) propose an algorithm that seems to be one of the best. They have certainly the highest level of ambition. Aspects like the service crews availability, traffic situations and such things are mentioned but not included in the algorithm. The algorithm is an extension of an earlier work (4) by the same authors. In their latest paper the authors criticize the mathematical approach for not considering such important factors as priority of zones (A hospital ought to have higher priority than a sports arena) and availability of crews. The algorithm is not a search algorithm. Instead it is something of trial and error. In short, the algorithm is as follows:

* Put all the deenergized non faulted zones in a list
* Determine priority of zones
* Try to restore the zone with highest priority first

This is done until no more zones can be restored. The restoration is done in three steps.

* First a whole group of feeders is restored. If the restoration causes a violation, the restoration is abandoned and the second step is tried.
* If the first step fails, only single zones are restored. If the restoration causes a violation, the restoration is abandoned and a third step is tried.
* The third step first tries to transfer load from the feeder that will take up the restored load so that when doing so, it will not be violated. Then the restored load will be transferred.
The last two steps seem to be quite simple but they are built by 250 rules. The proposed system may be used in a partly automated network (Some switches can be remote-controlled). This causes some switches to be preferred when it comes to switching operations and therefore the remotely controlled switches get a higher priority. The system not only produces a sequence of switching operations but also indicates the order of them. The execution time seems to be satisfactory (less than 5 seconds).

2.2 MATHEMATICAL APPROACH

This approach is the one used in the few existing physical test systems.

In (5) Castro and others show the results from a simulation of a proposed system. The simulation is a part of a project discarding economics to create a more effective distributional network. Communication links are thought to be installed in order to guide switches by remote control. He gives only hints of what the system contains and hardly anything of how it works. A major part of the paper describes the advantage of using a switchtable (an interface between pure algorithms and the real world). The system is thought to have two functions for network control.

First a load balancing algorithm which iterates a solution. Only one switch operation is suggested at a time until the network is balanced. If that is the case, the risk of overload in the network is reduced.

The second function is service restoration in case of emergency. The basic difference between these two algorithms according to Castro is that the service restoration produces the whole sequence of switching operations at one time and does not wait for the first operation to be completed.

This paper is perhaps more interesting for those who deal with the problem of installing a commercial system.

In (6) Kato and others also handles the case of a real system. Kato works for Tokyo Electric Power Co. and in the paper he describes their installed test system in Ginza. (It was installed in 1981) Much is said about communication etc. but a paragraph about service restoration is found.

Also here only hints are given about the algorithm. It works in two steps.

First all the loads that can be transferred directly without causing any violations are transferred. Then loads in those feeders who will be violated by further restoration is transferred to other feeders before they are able to take up new restored load.

Test runs (The system was put into trial application in 1984) show that non-faulted zones could be restored in five minutes, within about a tenth of the time for manual restoration.

Aoki and others (7),(8) have produced several papers in the field. Obviously he has studied Lee (3) because (7) is an improved algorithm compared with (8). He has introduced priority for the zones thereby showing that it is as easy to use priority in the mathematical approach as in the expert system approach. There are more improvements. Voltage drop constraints are included as well as the powerful effective gradient method (9),(10). The algorithm works in four steps.

1. After fault isolation the deenergized zones will be restored to the feeder with the largest violation margin, discarding any violations.
2. If violations occurred these are tried to be removed by transferring loads to adjacent feeders until no violations exist or no more load transfers can be made without creating new violations.

3. If violations still exist loads will be disconnected until no violations exist.

4. In the fourth step the previous deenergized loads (in step 3) will be reenergized one by one and step 2 will be tried once again among former violation feeders. If it fails, the load will be disconnected again.

The effective gradient method is used to select in which order the loads are tried to be transferred or cut off. This is the only algorithm which has been tested on a large network (1188 nodes) and it produced a solution in less than a second.

2.3 SELECTION

There were two main candidates for implementation, Aoki (7) and Lee(3). Lee seemed to be the best in some respects and Aoki in others. The greatest advantage with Lee's algorithm is that the priority of zones will be determined by the operator during execution and interpretations of the priority-measurements are postponed until this moment. (No file containing predecided priority exists). On the other hand seemed this advantage possible but not suitable for implementation in Fortran.

Aoki's algorithm on the other hand was obviously possible to implement in fortran and since it had no real disadvantages in any respect, this algorithm was selected.
3. IMPLEMENTATION

3.1 ALGORITHM

As it was previously stated the algorithm solves the problem in four steps. Below follows a flowchart of the algorithm.

figure 1 (by Aoki). Flowchart of algorithm.

Step 1
First, the fault is isolated. This creates groups of deenergized sections. If a fault occurs in a branch-point there will be more than one group. For each of these groups all possible connections are examined. The best connection, that is the one to a feeder with the biggest margin of getting violated, is chosen and the group is reenergized.

If the connection causes a violation, the magnitude of these violations is stored in a violation vector. For each feeder both voltage drop violations and line capacity violations are stored. The violation vector is later used in order to decide between which feeder pair a load transfer should be performed.

If no violations occurred the violation vector equals zero and a feasible solution is found. The algorithm stops.

Step 2
Step 2 is only used when at least one feeder is violated. It has two principal parts, first and second stage support. Both these parts try to eliminate violations by transferring load from a violated feeder to a nonviolated.
In the first stage support, load is tried to be transferred from a violated feeder to a support feeder if it is possible. One cut switch has to be chosen for load transfering and the algorithm first choses the one that will reduce the violation the most. This is done by solving the following optimization problem.

\[
\text{Max} \quad \sum_{j \in I_h} \frac{h_j}{\alpha_j(H_j + \beta)} y_j
\]

\[
\text{Sub.to} \quad \sum_j A_{ij} y_j \leq b_i (i = 1, 2, \ldots)
\]

The variable \(y_j\) is a 0-1 variable. It indicates the switch status. \(I_h\) is the set of possible cut switches that may be used for load transfer. \(H_j\) stands for remaining violation while \(h_j\) stands for the effective withdrawal of the violation. \(A\) is the capacity of the transformer, \(\alpha_j\) the node priority and \(\beta\) is only included for avoiding zero divide. This problem is solved by using the dual effective gradient method (9),(10) which states that the order in which the load transfers are to be tried, is in descending order of the measure

\[
\frac{h_j}{\alpha_j(H_j + \beta)}
\]

This procedure is repeated until no more violations exists or until no more transfers can be performed without causing any violations in the support feeder. A flowchart of the first stage support is given in figure 2.

\[\text{figure 2 (by Aoki). Flowchart of first stage support}\]
The second stage support is only tried when the first stage support fails to transfer load. It works similarly to the first stage support. The second stage support begins with a modified first stage support allowing violations in the support feeder. Then this violation is eliminated by performing a first stage support on the violated support feeder. Aoki suggested a slightly different measure for the first stage support of the violated feeder (not the violated support feeder). He suggests

\[
\frac{h_j}{a_j(a_j + \beta)}
\]

Here \(a_j\) is the load magnitude of the transferred node. The second stage support performed until violations in the feeder is eliminated or until no secondary support feeders is able to take up any more load. In figure 3 a flowchart of the second stage support is found.

Figure 3 (by Aoki) flowchart of second stage support
These two procedures are closely associated with each other and because of the similarity, a way of implementing them that could be hard to understand was selected. Some comments are needed.

First the candidate for a load transfer is to be selected. A file containing the pair of switches that is involved in the load transfer, the transferred load and an estimate of the reducement of the violations is stored in a file by a procedure called stage. This file is read by the procedure 'stagel' which calculates the different measures and selects a pair of switches able to perform a load transfer.

The two feeders involved in the load transfer is identified and all the different currents, node voltages and other data that could be changed due to a load exchange is first stored in a file called history. Then the actual load exchange is performed and new physical data is computed. If no new violations occur, all the 'backup'-files are deleted.

If new violations is created the former 'state' is read back in the net and a new trial is made until success or no more possible exchanges may be performed. Then 'stagel' terminates with ok=false, keeping all the 'backup'-files.

'Stage 2' then calls 'stagel' with ok=false. This causes 'stagel' to perform a load exchange discarding violations and as an outparameter you get 'outfee' which is the violated feeder. Then an ordinary stage1 is performed for this feeder.

The 'stack of backup files' thus created is difficult to handle and if someone ever needs to make any changes in these procedures they are advised to think at least twice, before making any changes in order to fully understand them. If any bad instruction occurs in these procedures, the algorithm will produce undesired solutions, and the fault might be hard to detect.

**Step 3**

If violations still remain after step 2 then step 3 is executed. In step 3 a load curtailment is performed. The dual effective gradient method is also this time used to produce a measurement helpful in selecting the end section that will be disconnected. The measurement will be

$$a_j(H_j + \beta)$$

For every turn of step 3 the section with the smallest value of the above will be selected. A small load causing a small remaining violation is chosen. When no more violations remain, step 4 is executed.

**Step 4**

In step 4 the previously curtailed loads are tried to be restored. This can work if more than one feeder were violated. Then there is a possibility of restoring loads between two former violation feeders. The dual effective gradient method produces this time the following measure

$$\frac{a_j}{(|R_j| + \beta)}$$

$R_j$ is the violation vector. When the cutswitch $j$ is closed, one feeder will be violated.
The curtailed sections will one by one be reenergized in decreasing order of the measurement above and a first stage support will be performed. If the first stage support fails the load will once again be disconnected. In figure 4, a flowchart of step 3 and 4 is found.

**Figure 4 (by Aoki) Flowchart of step 3 and 4**
3.2 DATASTRUCTURE

In the datastructure six different types are found. These are
1: transformer
2: feeder
3: cable
4: branchpoint,
6: switch

Every type is represented by matrices containing pointers to the closest neighbours and physical data. More details about the data structure is found in the appendix.

3.3 NETWORK REPRESENTATION

There are a few simple rules that must be followed when representing a network in order to make the algorithm work.

- The feeder is only used to provide a useful entrance to the network and to attach the violation vector.
- There must be a switch between the feeder and the first node in the network.
- A cable is used to connect elements of the other types and can therefore not be attached to another cable.
- The switch also represents the circuit breaker in the algorithm in order to reduce the size of the code.

3.4 COMPUTATIONS

The following data is presupposed to be known:

1. The network's topology
2. Cable impedances
3. Node impedances based on a load forecast
4. Transformer voltages

The transmission net was presumed to be strong so that the transformer voltage was kept fairly constant even if the load increased. Series impedances were used as a model of impedances. Knowing this and the fact that the network has a radial structure, the voltage and the current in an arbitrary point in the network could be computed. In figure 5 a radial structure is seen.

First the total impedance must be computed. This is done by traversing the 'tree' of the transformer. When a 'leaf' is found the computation begins. On the way up the computations proceed storing parallel impedances in matrice BGRAPH, when a branch point is reached. The search of a new leaf starts.

When the impedance is known the current is easily computed.

Then by applying the laws of Kirchhoff, all currents and voltages are easily computed by once again traversing the tree.
This simple move around method seemed to be the best in this case. More sophisticated methods taught in circuit theory seemed to be more difficult to handle in this case because the number of impedances in the local network (the one that is connected to a single transformer) is continuously changing. Too much work would be spent on setting up matrices.

There are some problems concerning numerics. After a leaf node the current often differs from zero but the fault is not big enough for considering double precision (about 0.1 percent of transformer current). It is necessary to remember that the load forecast probably contains some error.

3.5 IMPROVEMENTS

During the implementation of the algorithm, some possible improvements were discovered.

The multi stage support may be generalised. A third stage support can be performed tolerating violations in the secondary support feeder and a first stage performed on this violated feeder. Aoki mentions this but consider it unnecessary because "the distribution systems are often designed so that all de-energized loads in an island can be restored without creating any serious violations" (Island means a group of de-energized sections). This is certainly the case when the transformer carries a normal load but faults also occur in case of a peak load. A third stage support was implemented.

The measurement given in both the first and second stage supports could be improved. The suggested improvement was

\[
\frac{h_j}{\alpha_j(H_j + \beta)} \ast \frac{1}{a_j}
\]

This causes a small load causing small remaining violation to be selected and this seemed to be an even better measurement than the one by Aoki.

In the load curtailment the priority is discarded by Aoki. This seemed to be wrong. A high priority zone should not be selected for load curtailment if
another candidate with lower priority exists. Therefore a possible improvement might be
\[ \frac{a_j(H_j + \beta)}{\alpha_j} \]

In this algorithm a small numerical value of \( \alpha \) indicates a high priority zone.

The Aoki algorithm only indicates the switches that must be changed. It never tells you in which order to perform the switching operations. This may be fatal because it can create serious violations due to the fact that first violations are created and then these violations are tried to be reduced. Therefore some kind of indication of order is needed. It is only the switching operations that first reenergizes the isolated zones that can cause a violation, and therefore these must be performed in the end. An improvement of the algorithm would be that switching operations due to fault isolation would be indicated by the value 1. These operations must be performed first. Then the switching operations due to step 3 and 4 are performed so that no violations occur. Then those switching operations caused by the first and second stage support can be performed and eventually the switching operation caused by the reenergizing of isolated nodes.

3.6 TESTRUNS

In order to investigate if the suggested improvements really improved the result, a series of test runs was made on a small network taken from (7). This network is shown in figure 6.

figure 6. Topology of test system (by Aoki)

A number of different qualities were demanded for a good solution.

- The execution time has to be rather short, less than 10 seconds is preferred.
The number of switching operations need be as low as possible. This is due to the fact that today when almost every switch is manually controlled, it takes some time to perform the switching operations. (mostly due to the distances between the switches in the network)

The restored load ought to be as big as possible. It may be difficult to measure the restored load because it depends on the node voltage and this can be changed during execution. Different feeders have different loads and therefore different voltage drops.

The plain Aoki algorithm was tested. The test showed that this implementation was equally effective as the Aoki implementation (Execution time ranging from 5 ms to 1 s). The number of switching operations and the amount of restored load seemed to be equivalent. This implementation was then used as a reference.

One sequence of test runs of specially selected faults was run. The faults were severe causing major violations in order to see the differences more clearly.

The third stage support was tested. It was obvious that the execution time raised dramatically to about 5 seconds. This however was not so serious.

It was soon noticed that the third stage support had little effect on the amount of restored load unless the network was unbalanced with all primary and secondary support feeders carrying major loads while other feeders only carried minor to normal load. This was hardly a surprising result.

More important was the number of switching operations. It increased by a factor of approximately 1.5 (In average). When the second stage support has failed nothing is tried in order to reduce the violation until the load curtailment takes place.

In the load curtailment process only one single switch is changed. But if the third stage support is used successfully, then for every successful turn of this step six switches are changed. (Two switches connected to the primary support feeder, two to the secondary and two to the tertiary support feeder) The switches changed due to a successful third stage support can be seen in the figure 7.

```
F1 -- -- -- SW1 -- SW2 -- BP1 -- F2
  !
  SW3
  !
  SW4

F4 -- BP3 -- SW6 -- SW5 -- BP2 -- F3

F:Feeder  SW:Switch  BP:Branch Point
```

Figure 7. Topology typical for a third stage support.

Between switches and branch points an arbitrary number of sections may be found. The third support is applied to feeder F1 transferring load to the primary support feeder F2(SW1,SW2), causing a violation. The violation is
eliminated by a further load transfer to feeder F3 (SW3,SW4), which in its turn has to transfer load to F4 (SW5,SW6) before it can take up any more load. The load transfers in this case are carried out by opening the odd switches and closing the even.

This increased number of switching operations made the suggested improvement less desired because the number of switching operations was already large (about 10-15). The 'optimal' solution must have a chance of being reached before the repair of the faulted zone is finished.

During test runs of the third stage support, the suggested improvement of the measurement in the second and first stage support was detected. When a fault caused serious violations, often the most severe line capacity violation was in the 'root' cable. This could be reduced by transferring any load in the tree, and sometimes a node which had not been deenergized was selected. This cannot be desired. A load transfer should always be made among former deenergized loads in order to best reduce the violation.

It is a known fact that in the beginning (before the reenergizing of isolated sections) the violation feeder was non violated. Therefore it is desired to perform the load transfer in the reenergized sections because these created the violation. The reason for violation must have a greater attention than the size of load and this is done by dividing the proposed measurement (Aoki) by $H_j$. An example is shown in figure 8.

```
F — SW — BP — — — — N1
    N2
```

F stands for feeder, SW for switch, BP for branch point and N for node

Figure 8. Typical topology

The node N1 has been among the reenergized nodes. When it was reenergized, it caused a violation in feeder F. In this case the voltage for N1 is 20 V below the voltage drop limit but N2 is not violated. In the root cable, that is the cable from the feeder to the branchpoint, the current exceeds the line capacity with 40 A. In this case the violation vector would be (20,40) Only these two candidates (N1,N2) for load transfer exist. Both nodes have the same priority. The load of N1 is 60 A and the load of N2 is 40 A. Computing the measurement (by Aoki) for these nodes gives:

N1: $h_j=\sqrt{2000}$, $H_j=0$ and measurement=0.75
N2: $h_j=40$, $H_j=4.7$ and measurement=1.0

In this case N2 would have been selected. If the proposed measurement was to be used N1 would be selected instead. The new measurements would be:
It is obvious that the best choice for reducing the violations is transferring N1. Then both a voltage drop violation and a line capacity violation would be avoided. This was later proved to be an improvement, but not a major one. It reduced the number of switching operations with, in some cases, one or two. (The number of switching operations never increased but for the most cases it was the same.) The amount of restored load was unaffected (in average). The third suggested improvement to include the priority in the load curtailment was also an improvement in the sense that according to priority rules some previously wrong decisions were made right. An example of this is seen in figure 9.

![Figure 9. Typical configuration in case of load curtailment](image)

Here both nodes N1 and N3 are violated with a voltage equally large, 20 V below the voltage drop limit. N1 is a high priority zone ($\alpha_{N1}=1$) while N2 is a low priority zone ($\alpha_{N2}=10$). The loads are: N1=10 A, N2=20 A.

Because of the equality in violation both the zones cause equal reductions in the violation vector. In this case the measurement proposed by Aoki will only depend on the size of the load. Therefore the high priority zone will be selected.

This could cause many damages in the society. An important operation in a hospital may have to be interrupted or postponed.

According to the proposed measurement (priority included) the low priority zone would have been selected for load curtailment. Then it is up to the operator to decide the different priorities. In this case the number of switching operations and amount of restored load must be of second interest. Anyway no major changes emerged. This is due to the fact that the service restoration problem is a combinatorial optimization problem. A decision that perhaps is seen as a 'bad' decision in the sense that it restores less load could in fact be better than the decision suggested by the heuristic search approach. It is the combination of decisions taken that lead to the optimal solution.
4. RESULTS

* As a result of an information retrieval two major approaches to the service restoration problem was found (Expert System Approach and Mathematical Approach).
* The mathematical approach is the only one which has been tested in a physical system.
* The result from this test system was promising.
* The mathematical approach was selected due to the facts that code was to be written in fortran and the successful tests that others had done in the field with this approach.
* An algorithm was chosen for implementation, which fulfilled all possible requirements for a good performance such as a short execution time and a small number of switching operations.
* This algorithm was then implemented in fortran.
* The implementation was proven to be as effective as the original implementation.
* Four possible improvements were suggested.
* These suggested improvements caused a minor improvement in performance of the algorithm.
* An operator aid was added. Not only the switching operations were presented but also a rough indication of the order of these operations was given.
* In a physical system the use of a service restoration algorithm depends on the accuracy of the load forecast.
5. FUTURE

As it has been previously hinted, the service restoration algorithm is only one part of a group of algorithms thought to be used in an automatized distribution network in the future. The reason for the focus on the service restoration algorithm at the moment depends on the possibility to use it with locally controlled switches. It would be too expensive having service crews continuously driving around changing switches in order to improve the performance of the network. In case of an emergency however, it is essential to transfer load. In the future these other algorithms (the load balancing algorithm, the minimum loss and the minimum cost algorithm) as well as the service restoration algorithm will probably all be found in an ordinary automatized distribution network. Due to the high cost of installing communication links, hardly all sectionalizing switches will be remotely controlled. Instead, only those that often will be used for a load transfer will be supplied with remote control.
6. REFERENCES


(10) Toyoda, Y. "A simplified algorithm for obtaining approximate solutions to zero-one programming problems" Management Science, vol. 21, No. 12, 1975
7. APPENDIX

7.1 USER'S MANUAL

All the useful files are stored on node SNEV02 at ABB Network Control. The files will be found in directory EXARB ROOT: MBERGSTRAND.

In the file CONSTANTS.FOR are all the necessary constants declared and this file is included in all files in order to facilitate any changes in the structure.

This was the same reason for having the file COMMON.FOR. In this file all global variables needed were declared. This file is also included whenever it was necessary.

The constants MAXNRT, MAXNRF, MAXNRC, MAXNRB, MAXNRRN and MAXNRS have to be changed before the program is used on a new network.(nr of transformers, feeders,cables,branchpoints,nodes and switches) Another constant, BRANCH, may have to be changed. It indicates the maximum allowed branches for any branchpoint in the network.

When this is done all the files RESTORE, OUTFI, MOVING, TESTS, COMP, STEP1, STEP2, FAULT, FETCH and CHG must be recompiled and linked together. The main program is in file RESTORE.FOR.

Then an input file must be made, containing the physical data and the topology of the network.

All the different components (transformers, switches, feeders, cables, branch points and nodes) must be numbered.

The program uses formatted input. The procedures of interest with appropriate format is located in the file OUTFI.FOR. The procedures are called WRITF and INFI. Studies of the structure chart found later in the appendix is essential understanding WRITF and INFI. The structure chart tells what the different elements in the matrices used stand for.

In the file the different types is in the order

- Transformers
- Feeders
- Cables
- Branch Points
- Nodes
- switches.

The format used is as follows:

- All real values are written using exponents (0.100E+01).
- First two records are needed to define the transformers (common to all the different transformers).
- The first contains the constant MAXNRT (MAXimum NumbeR of Transformers ,integer) written in position 3-8
- The second contains the maximum voltage in the network (real) written in position 3-14
Then one record is used to represent every single transformer. This record is repeated MAXNRT times. It contains the following data:

- Maximum value of P in pos. 3-14 (real)
- Maximum value of Q in pos. 17-28 (real)
- A pointer to a cable in pos. 31-36 (integer)

The transformers, as well as any other types, have to be in order (Trafo 1, Trafo 2...).

**FEEDERS come after TRANSFORMERS**

- This is followed by one record containing the constant MAXNRF written in pos. 3-14 (integer)

Then one record containing the topology for each single feeder is repeated MAXNRF times. It contains of:

- A pointer to the cable that leads towards the transformer in pos. 3-8
- A pointer to the cable that leads 'down' towards the nodes in pos. 11-16

Both are integer values

The next type in the data file is CABLE.

- First one record containing the constant MAXNRC, written in pos 3-8 must be made. (integer)

That is followed by seven records, containing the physical data and the topology for each single cable. These seven records are repeated MAXNRC times. The records are shown below:

  Record 1. R pos. 3-14 real
  Record 2. X pos. 3-14 real
  Record 3. Imax pos. 3-14 real
  Record 4. Type 1 pos. 3-8 integer
  Record 5. Nr 1 pos. 3-8 integer
  Record 6. Type 2 pos. 3-8 integer
  Record 7. Nr 2 pos. 3-8 integer

Type and Nr are referring to the element, which the cable is connected to. BRANCH POINTS come after CABLES.

- One record containing the constant MAXNRB in position 3-8 (integer)

Then for each branch point:

- First the actual number of branches, less than the constant BRANCH, is written in one record in pos 3-8
- Then one record containing one pointer to a cable is written in position 3-8 for every branch.

This is repeated MAXNRB times.

An example:

In one network there are 2 branch points. The first branch point has 4 branches and the second has 3. The first branch point is connected to cables 6, 7, 8 and 9. The second is connected to cables 2, 3 and 4. This is how it appears in the data file.
NODES come after BRANCH POINTS.

- One record containing the constant MAXRN in pos. 3-8.
- Then the voltage drop limit is written in one record in pos. 3-14. The voltage drop limit is an actual voltage and not a percentage of the transformer voltage.

Then three records is used to represent one node.

For each node:

Record 1. Pointer to cable pos. 3-8 (int)
       Pointer to cable pos. 11-16 (int)
Record 2. R (Resistance) pos. 3-14 (real)
       X (Inductance) pos. 17-28 (real)
Record 3. Priority pos. 3-14 (real)

As an example, the physical data and the topology for one node, connected to the cables 3 and 4 is seen below as it must be written in the data file. The node's impedance is (543,j310) and the priority is 1.0.

```
  3
  4
  0.54300E+03  0.31000E+03
  0.10000E+01
```
The above is repeated for each node.
SWITCHES come after NODES.

- One record contains the constant MAXNRS in pos. 3-8.
The topology come before the status.
One record contains the topology.

- A pointer to a cable in pos. 3-8
- A pointer to a cable in pos. 11-16
   This is repeated for each switch.
Then follows the status for each switch in one record for each switch.

- Record: Switch status (0,1) in pos. 3-8
This is repeated MAXNRS times.

An example:
One network has two switches. The first switch is connected to cable 10 and 12, and it is opened (Status=0). The second switch is connected to cable 8 and 9, and it is closed (Status=1).
This is how it looks in the data file:

2
10
12
8
9
0
1

Following these instructions in order, a correct data file is produced.
All files of the type *.DAT* contains a network, with the name indicating whether the load is minor, medium or major.

During execution of the program all commands and answers must be written in CAPITAL LETTERS.

When the program starts its execution, first a check of the topology is made. Some faults can be detected and messages will be written on the screen telling the user what to do. The faults that will be detected is of the kind: "if you can 'go' to a neighbour but you can not go back to the starting point" it must be wrong in the network topology. Most of the faults are detected by this check but fault can also 'cooperate' so that two faults make it look like no fault. If errors of this kind occur in the network these faults most certainly are detected later causing messages like this: WRONG IN STEPDO.NODE, FORTRAN STOP.

If messages of this type appear, the program provides a way of easily finding these errors. When the first test of the topology is made, the program asks if the operator desires a visible track. If the answer is 'Y' the program will print out all the steps taken in the subroutine init. Then it is easy to follow the program until an error is found. Then the latest point must be remembered. In this point the error is found. Edit the data file and try again.
If the topology is correct the text INIT PASSED will be written on the screen.
Later in this section transcripts of possible test runs are showed with comments added afterwards.

The program will then ask if any changes are to be made in the network (concerning load). If you wish to do so the answer is Y. Then the program will ask if you wish to change the total load. If not, you have the possibility to change a single node impedance. If you choose to change the total load the program will ask for a factor with which every single node impedance in the network will be multiplied. As a guideline you can use 1.1 if you wish to reduce the load with 10 percent \( P = \frac{u^2}{R} \) but the system is not linear. In case of a change in a single node the old value of the impedance is written on the screen and then the new value is read. Then the program asks for a filename. The new load condition is saved for later use. The filename can be seven characters long. All names except HISTORY, STAGE and CURTAIN are allowed. Files with these names are used internally. See also in the transcripts.

Then the program will ask for a point in the network where the fault has occurred. It can be any point in the network. A point is indicated by its type and number (both integer values). Available types are

1: Transformors
2: Feeders
3: Cables
4: Branch Points
5: Nodes
6: Switches

Then the algorithm starts working, continuously writing data on the screen. A careful study of decisions taken can be done. In the end a menu is presented containing execution time, the switches that have changed status, the order of these switching operations, deenergized load and restored load. Then the program asks if some more faults (A new test case) is desired and if so the program restarts.

7.2 EXAMPLES

Four different examples are shown in order to demonstrate how the program works.
TRAFOS IS OK
FEEDER IS OK

FAULTED NETWORK
BRANCH-POINT NR 1 IS CONNECTED TO CABLE NR 30

CABLE NR 30 IS CONNECTED TO
TYPE= 5 NR= 6

AND
TYPE= 6 NR= 4

FAULT IN BRANCH-POINT(4) OR IN CABLE(3)?

THE FOLLOWING BRANCHES

1
2
30

THE FAULTING ONE IS (NOT) CONNECTED TO
THE BRANCH IS CONNECTED TO CABLE NR?

BRANCHPOINTS OK
NODES OK
SWITCH OK
SWITCH OK

TRAFOS IS OK
FEEDER IS OK
BRANCHPOINTS OK
NODES OK
SWITCH OK
SWITCH OK

THE NETWORK IS CORRECT
DO YOU WISH TO HAVE A VISIBLE TRACK? Y/N
DO YOU WISH TO MAKE ANY CHANGES IN THE NETWORK? Y/N
INIT PASSED

FAULT IN

1: TRANSFORMER
2: FEEDER (MAIN SUPPORT)
3: CABLE
4: BRANCHINGPOINT
5: NODE

WHICH TYPE OF COMPONENT?

WHICH NODE?
ENTERS STAGE 1 2
ENTERS STAGE 1
READS IN STAGE 1

IMPROVEMENT,FEEDER 1.614994
IMPROVEMENT,FEEDER 0.2358537
IMPROVEMENT,FEEDER 0.2358537

SAVENE

2 6
LEAVES STAGE1 AFTER A SUCCESSFUL SESSION, OK=T
ENTERS STAGE 1  2
ENTERS STAGE
READS IN STAGE1
IMPROVEMENT, FEEDER 5.272868 6
IMPROVEMENT, FEEDER 0.2615825 1
IMPROVEMENT, FEEDER 0.2615825 5
SAVENE

2  6
SAVENE
LEAVES STAGE1 AFTER A SUCCESSFUL SESSION, OK=T
ENTERS STAGE 1  2
ENTERS STAGE
READS IN STAGE1
IMPROVEMENT, FEEDER 6271.001 6
IMPROVEMENT, FEEDER 0.0000000E+00 1
IMPROVEMENT, FEEDER 0.0000000E+00 5
IMPROVEMENT, FEEDER 0.0000000E+00 6
SAVENE

2  6
SAVENE
LEAVES STAGE1 AFTER A SUCCESSFUL SESSION, OK=T
ENTERS STAGE
READS IN STAGE1
IMPROVEMENT, FEEDER 2805.759 6
IMPROVEMENT, FEEDER 0.0000000E+00 1
IMPROVEMENT, FEEDER 0.0000000E+00 5
IMPROVEMENT, FEEDER 0.0000000E+00 6
SAVENE

2  6
SAVENE
LEAVES STAGE1 AFTER A SUCCESSFUL SESSION, OK=T
TOTAL COMPUTATION TIME OF ALGORITHM 275 ms

THE FOLLOWING SWITCHES HAS CHANGED STATUS

37  4
38  3
45  1
46  1
50  3

TOTAL DEENERGIZED LOAD DUE TO FAULT
0.29190E+07

TOTAL RESTORED LOAD
0.29190E+07

ANOTHER FAULT? Y/N

FORTRAN STOP
COMMENTS

In the previous example the possibility to discover a single fault was demonstrated.

The error in the topology was identified and then the data concerning the topology for the two possible faulted elements were written on the screen. The operator then identifies the faulted element. (In this case the branch point) All connections for this erroneous branch point are written in the screen and the faulted connection is marked by the text "THE FAULTING ONE IS NOT CONNECTED TO". The program then asks for the number of the cable (the right one) that it is connected to. Then the whole test is done again so that no more faults occurred due to this 'new' topology.
TRAFOS IS OK
FEEDER IS OK
BRANCHPOINTS OK
NODES OK
SWITCH OK
SWITCH OK

THE NETWORK IS CORRECT
DO YOU WISH TO HAVE A VISIBLE TRACK? Y/N

DO YOU WISH TO MAKE ANY CHANGES IN THE NETWORK? Y/N
POINT IN IS 1
POINT IN FROM STEP DOWN IS 6 57

INDATA NEWDIR (POINT,DIR,BRA)
   6  57  0  4

OUTDATA NEWDIR (POINT,DIR,BRA)
   6  57  0  4

INDATA TKESTP (POINT,OLDP,DIR,BRA)
   6  57  1  1  0  4
POINT IN IS 6  57
POINT IN FROM STEP DOWN IS 4  1

OUTDATA TKESTP
   4  1  6  57  0  4

INDATA NEWDIR (POINT,DIR,BRA)
   4  1  0  4

OUTDATA NEWDIR (POINT,DIR,BRA)
   4  1  0  4

INDATA TKESTP (POINT,OLDP,DIR,BRA)
   4  1  6  57  0  4
POINT IN IS 4  1
POINT IN FROM STEP DOWN IS 2  1

OUTDATA TKESTP
   2  1  4  1  0  4

INDATA NEWDIR (POINT,DIR,BRA)
   2  1  0  4

OUTDATA NEWDIR (POINT,DIR,BRA)
   2  1  0  4

INDATA TKESTP (POINT,OLDP,DIR,BRA)
   2  1  4  1  0  4
POINT IN IS 2  1
POINT OUT FROM STEP DOWN IS 6  54

OUTDATA TKESTP
   6  54  2  1  0  4

INDATA NEWDIR (POINT,DIR,BRA)
   6  54  0  4

OUTDATA NEWDIR (POINT,DIR,BRA)
INDATA NEWDIR (POINT, DIR, BRA)  
5 1 1 5

OUTDATA NEWDIR (POINT, DIR, BRA)  
5 1 1 5

INDATA TKESTP (POINT, OLPD, DIR, BRA)  
5 1 6 1 1 5
POINT IN IS 5 1
CABUP IN NODE IS 2
CABLE UP IS 3
POINT OUT FROM STEPUP IS 6 54

OUTDATA TKESTP  
6 54 5 1 1 5

INDATA NEWDIR (POINT, DIR, BRA)  
6 54 1 5

OUTDATA NEWDIR (POINT, DIR, BRA)  
6 54 1 5

INDATA TKESTP (POINT, OLPD, DIR, BRA)  
6 54 5 1 1 5
POINT IN IS 6 54
POINT OUT FROM STEPUP IS 2 1

OUTDATA TKESTP  
2 1 6 54 1 5

INDATA NEWDIR (POINT, DIR, BRA)  
2 1 1 5

OUTDATA NEWDIR (POINT, DIR, BRA)  
2 1 1 5

INDATA TKESTP (POINT, OLPD, DIR, BRA)  
2 1 6 54 1 5
POINT IN IS 2 1
POINT OUT FROM STEPUP IS 4 1

OUTDATA TKESTP  
4 1 2 1 1 4

INDATA NEWDIR (POINT, DIR, BRA)  
4 1 1 4

OUTDATA NEWDIR (POINT, DIR, BRA)  
4 1 0 5

INDATA TKESTP (POINT, OLPD, DIR, BRA)  
4 1 2 1 0 5
POINT IN IS 4 1
WRONG IN STEPDO.BRA

FORTRAN STOP
COMMENTS

In this example the possibility to discover cooperating faults was demonstrated.

1 31 32
--- BP1 --- BP8 --- SW58

Figure 10. One part of the topology.

In branch point 1 (BP1) had a pointer to cable 32 been stored instead of a pointer to cable 31. This error was not detected by the first test. By demanding the program to show how it 'moves' (Answer Y to the question 'DO YOU WISH TO HAVE A VISIBLE TRACK') the last point before the program stops is shown in the screen. Then it is easy to make the proper change in the data file.

In this case the program stopped at point (4,1). That is branch point 1. Several pages in the middle were left out.
WHICH FILE IS TO BE READ?

TRAfos IS OK
FEEDER IS OK
BRANCHPOINTS OK
NODES OK
SWITCH OK
SWITCH OK

THE NETWORK IS CORRECT
DO YOU WISH TO HAVE A VISIBLE TRACK? Y/N

DO YOU WISH TO MAKE ANY CHANGES IN THE NETWORK? Y/N

CHANGE TOTAL LOAD? Y/N
ABB Network Control

CHANGE IN TYPE NR?
R IS  1009.800
X IS  408.0000
NEW R IS
NEW X IS

ON WHICH FILE DO YOU WISH TO STORE IT
INIT PASSED

FAULT IN

1: TRANSFORMER
2: FEEDER (MAIN SUPPORT)
3: CABLE
4: BRANCHINGPOINT
5: NODE

WHICH TYPE OF COMPONENT?

WHICH NODE?

TOTAL COMPUTATION TIME OF ALGORITHM  14 ms

THE FOLLOWING SWITCHES HAS CHANGED STATUS

41  4
42  1
48  1
49  1
50  4

TOTAL DEENERGIZED LOAD DUE TO FAULT
0.21720E+06

TOTAL RESTORED LOAD
0.21720E+06

ANOTHER FAULT? Y/N

FORTRAN STOP
COMMENTS

In the third example the possibility to change a single node impedance was shown. Then a fault causing no violation at all was chosen to show how fast a solution can be reached.
TRAPSO IS OK
FEEDER IS OK
BRANCHPOINTS OK
NODES OK
SWITCH OK
SWITCH OK

THE NETWORK IS CORRECT
DO YOU WISH TO HAVE A VISIBLE TRACK? Y/N

DO YOU WISH TO MAKE ANY CHANGES IN THE NETWORK? Y/N

CHANGE TOTAL LOAD? Y/N
IMPEDBANCES WILL BE MULTIPLIED WITH?

ON WHICH FILE DO YOU WISH TO STORE IT
INIT PASSED

FAULT IN

1: TRANSFORMER
2: FEEDER (MAIN SUPPORT)
3: CABLE
4: BRANCHINGPOINT
5: NODE

WHICH TYPE OF COMPONENT?

WHICH TRANSFORMER?

ENTERS STAGE 1 3
ENTERS STAGE1 6
READS IN STAGE1 6
IMPROVEMENT, FEEDER 0.000000E+00 6
OK=-.FALSE.

ENTERS STAGE 2
STAGE2 CALLS STAGE1 6
ENTERS STAGE 1 3
OK IS FALSE
BUSY READING

STAGE1 WITH OK=P HAD AN UNSUCCESSFULL COMPLETION
EXIT STAGE 2 OK=F

ENTERS STAGE 1 5
ENTERS STAGE1 6
READS IN STAGE1 6
IMPROVEMENT, FEEDER 112015.5 6
IMPROVEMENT, FEEDER 13.42432 4
IMPROVEMENT, FEEDER 0.1634084 4

SAVENE 5 6

CALLS RESNET 6

CALLS RESNET 6
IMPROVEMENT, FEEDER 112015.5 6
IMPROVEMENT, FEEDER 13.42432 4
IMPROVEMENT, FEEDER 0.1634084 4

SAVENE 5 4

SAVENE 5 4

LEAVES STAGE1 AFTER A SUCCESSFULL SESSION, OK=T

ENTERS STEP 3
The following switches has changed status:

<table>
<thead>
<tr>
<th>Switch</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td>11</td>
<td>3</td>
</tr>
<tr>
<td>16</td>
<td>4</td>
</tr>
<tr>
<td>37</td>
<td>4</td>
</tr>
<tr>
<td>54</td>
<td>1</td>
</tr>
<tr>
<td>57</td>
<td>1</td>
</tr>
<tr>
<td>58</td>
<td>1</td>
</tr>
</tbody>
</table>

Total deenergized load due to fault:
0.99335E+07

Total restored load:
0.99335E+07

Another fault? Y/N

FORTRAN STOP
COMMENTS

In the fourth example the total load was first changed providing the right answers in the proper places (Y to all but 'DO YOU WISH TO HAVE A VISIBLE TRACK?'). Then a serious fault was chosen (minor load). A fault in transformer 1 occurred. The topology of the test system is once again showed in figure 10. A closed switch is marked by 'x' and an open switch is surrounded by a circle.

The fault is in this case isolated by closing the circuit breakers (switch 54,57,58 or #1,f1,f2 in the figure) Isolated trees are created with roots in nodes 1 and 35. These are marked (1), (35) in figure 11.

![Figure 11. Topology](image)

Then for each tree all possible connections are examined. Possible switches for the tree with node 1 as the root are 4,14,16. Note that switch 8 is not a possible connection because this switch leads to another isolated section. Switch 16 is in this case the most suitable switch for reenergizing the 'tree' because feeder 5 has a bigger margin of getting violated than feeder 4. Switch 16 is then opened. The same examination is done for the tree with node 35 as root and in this case switch 37 is opened.

These two switching operations caused violations in feeders 3 and 5. First the violation in feeder 3 is tried to be decreased. The only possible load transfer is transferring node 72 from feeder 3 to feeder 6 (the switches 8,28 and 41 leads all to a violated feeder) but since this load transfer does not reduce the violation at all it is abandoned. There is a voltage drop violation in node 13. This violation would be eliminated by a load transfer of this node, but it is impossible to transfer this node since its neighbour's feeder also is violated. (The neighbour feeder is now feeder 5)

A first stage is then tried on feeder 5. The transcript shows that there are three possible switches for load transfer. These are switch nr: 43, 14, 4. Switch 43 cause the biggest reduction in the violation vector because switch 43 is opened and switch 33 is closed but then feeder 6 would be violated. Instead the possibility to transfer load by opening switch 11 and closing switch 4 is
examined. This load transfer turned out to be successful. Feeder 5 is not violated any more.

The next step in the algorithm is the load curtailment. Feeder 3 is still violated and therefore node 13 is deenergized by opening switch 9. Then no violations at all exist and the fourth step is performed. Node 13 is reenergized by closing switch 9 and a first stage support is tried. The load transfer succeeds and node 13 is transferred to feeder 4. (Node 12 is energized by feeder 4 since the previous first stage support of feeder 5) Then all loads are restored and the algorithm stops.

The switching operations that must be performed is compared with the NON-FAULTED system. So in this case the circuit breakers are already changed. They were tripped by the fault.

Realizing this the operator can then command the switching operations to be executed in order.
7.3 OPTIMIZATION

PROBLEM FORMULATION

The service restoration is formulated by Aoki as follows:

Maximize \[ \sum_i \sum_j a_{ij} x_{ij} \]

Subject to

(line capacity constraint)
\[ \sum_j a_{ij} x_{ij} \leq b_{ik} \]

(transformer capacity constraint)
\[ \sum_j a_{ij} x_{ij} \leq b_L \]

(voltage drop constraint)
\[ \sum_j \left( \sum_{k \in \mathcal{J}_{01}} s_{ik} x_{ij} \right) z_{il} \leq V_{ie} \]

Where,

\( X_{ij} \): such 0-1 variable as 0 if section \( j \) at feeder \( i \) is de-energized and 1 if it is energized. \( X_{ij} \) cannot arbitrarily be set to 1. \( X_{ij} = 1 \) only if the adjacent section is energized and the sectionalizing switch connected to it is closed. \( X_{ij} \) is a function of the sectionalizing switch status.

\( a_{ij} \): load magnitude of section \( j \) at feeder \( i \)
\( b_{ik} \): line capacity at \( k \)-th point of feeder \( i \)
\( b_L \): transformer capacity of transformer \( t \)
\( z_{il} \): impedance of section \( l \) at feeder \( i \)
\( s_{ik} \): \( s_{ik} = a_{ik} \) (if \( k \neq 1 \)), \( s_{ik} = a_{ik}/2 \) (if \( k = 1 \))
\( V_{ie} \): voltage drop limit at the \( i \)-th end of feeder \( i \)
\( \mathcal{J}_{01} \): index set of de-energized load sections on feeder \( i \)
\( \mathcal{J}_{ik} \): index set of load sections connected to the leaf side of point \( k \) of feeder \( i \)
\( \mathcal{J}_{il} \): index set of load sections which exist at the leaf side of section \( l \) (included) of the feeder \( i \)
\( \mathcal{J}_t \): index set of load sections \( ij \) connected to transformer \( t \)
\( \mathcal{T}_e \): index set of load sections which exist at the trunk of the tree between bus and section \( e \)
EFFECTIVE GRADIENT METHOD

The effective gradient method is used by some economists dealing with problems such as: Which orders will not be served due to a limited resource. The effective gradient method is very simple to understand. It says that if you have to exit from a restricted area with an unlimited number of steps, but wish to stay as close to the border as possible and the distance to the permitted area is not known you should take the shortest step possible in that direction.
7.4 STRUCTURE CHART

The network is represented by the following matrixes where the first index points out the number of the component in the network. The following map shows where the data is stored for every component.

**TRAFOS**

| P_{max} | Q_{max} | P | Q | V | R_{network} | X_{network} |

**MSFEED**

| Trafo Identity | Cable up | Cable down |

**CABTOP**

<table>
<thead>
<tr>
<th>First neighbour Type</th>
<th>Second neighbour Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
<td>Number</td>
</tr>
</tbody>
</table>

**CABPHY**

| R | X | I_{max} | I |

**BRAPOI**

| Actual nr of branches | Branch up | Branch 1 | Branch 2 | ... |

**NODETO**

| Up | Cable 1 | Cable 2 | Feeder identity |

**NODEPH**

| P | Q | V | R | X |

**SWITCH**

| Status | Up | Cable 1 | Cable 2 | Allowed change |
Some comments are required to understand the structure charts.
In up (cable up, branch up) the index pointing out the cable leading in the direction of the transformer is stored.
For open switches, up will be assigned a new value every time the switch will be 'visited'.
Allowed change is used to prevent a load transfer to the faulted zone.
In the program another matrix with three indexes, BGRAPH is used to store the impedance one sees if one look down a branch. This matrix have matched the two first indexes with BRAPOI.
PROGRAM MAIN

AUTHOUR MAGNUS BERGSTRAND

VERSION 1

DATE 1989-10-07

PURPOSE
WHEN A FAULT HAS OCCURRED IN AN ELECTRICAL DISTRIBUTION NETWORK
THIS FAULT MUST BE ISOLATED. THE NON FAULTED ZONES CAN OFTEN BE
RESTORED WITHOUT CAUSING ANY VIOLATIONS. THE PROGRAM SUGGEST SUCH
A POSSIBLE RESTORATION

METHOD
THE METHOD IS DESCRIBED IN 'SERVICE RESTORATION IN ELECTRIC
DISTRIBUTION NETWORKS' BY MAGNUS BERGSTRAND
THE METHOD WAS FIRST DESCRIBED BY AKI IN
'A NEW ALGORITHM FOR SERVICE RESTORATION IN DISTRIBUTION SYSTEMS'
IEEE/PES WINTER MEETING, NEW YORK 1989 PAPER 89 WM 085-2 PWRD

DESCRIPTION
FIRST IS THE FAULTED ZONE ISOLATED AND THE NON FAULTED ZONES
CONNECTED TO AN ADJACENT FEEDER. IF THIS CONNECTION DOES NOT
CAUSE ANY VIOLATION THE ALGORITHM STOPS AND A SOLUTION IS FOUND.
IF VIOLATIONS OCCURRED THESE ARE TRIED TO BE DECREASED IN STEP 2.
STEP 2 HAS TWO PRINCIPAL PARTS THE FIRST AND SECOND STAGE SUPPORT.
THE FIRST STAGE SUPPORT TRANSFERS LOAD TO AN ADJACENT FEEDER FROM
A VIOLATED FEEDER NOT ALLOWING ANY NEW VIOLATIONS.
THE SECOND STAGE SUPPORT TRANSFERS FIRST LOAD TO THE PRIMARY SUPPORT
FEEDER ALLOWING VIOLATIONS AND THEN TRANSFERS LOAD FROM THE PRIMARY
SUPPORT FEEDER TO A SECONDARY SUPPORT FEEDER NOT ALLOWING VIOLATIONS.
IF VIOLATIONS STILL EXIST THESE ARE REMOVED BY THE THIRD STEP (THE
LOAD CURTAILMENT).
AFTER THE LOAD CURTAILMENT THERE IS A POSSIBILITY TO RESTORE LOAD
AMONG FORMER VIOLATION FEEDERS AND THIS IS DONE IN THE FOURTH STEP.

SEE ALSO
FILES ROUTINES COMMENTS
CONSTANTS ALL CONSTANTS DECLARED
COMMON ALL COMMON VARIABLES DECLARED
OUTFI INFI READS THE NETWORK TOPOLOGY AND PHYSICAL DATA FROM
A DATA FILE
OUTFI WRITEF SAVES THE NETWORK TOPOLOGY AND PHYSICAL DATA ON
A DATA FILE
CHG INC CHANGES TOTAL LOAD
CHG CHANGE CHANGES PHYSICAL DATA IN A SINGLE ELEMENT
TESTS NETTES TESTS THE TOPOLOGY OF THE NETWORK
TESTS INIT INITIATES THE NETWORK
COMP FIPOW COMPUTES POWER
COMP COMPH COMPUTES IMPEDANCE
C Computes and fills in node voltages and cable currents
C Reads the faulted section
C Produces the result menu
C 'Garbage collector' cleans up among switching operations

INCLUDE 'CONSTANTS.FOR'
INCLUDE 'COMMON.FOR'

INTEGER FAULT(2)
INTEGER COU,PA
CHARACTER*7 CH2
LOGICAL VIO,READY,WR
INTEGER TIME,MSCPU,1,A
CHARACTER DATA

EXTERNAL MSCP

20 WRITE(*,*)' WHICH FILE IS TO BE READ?'
READ(*,300)CH2
CALL IMFI(CH2)
CALL NETTES(CH2)
10 WRITE(*,*)' DO YOU WISH TO HAVE A VISIBLE TRACK? Y/N'
READ(*,300)DATA
IF (DATA.EQ.'Y') THEN
   WR=.TRUE.
ELSEIF (DATA.EQ.'N') THEN
   WR=.FALSE.
ELSE
   GOTO 10
ENDIF
READY=.FALSE.
40 IF (.NOT. READY) THEN
   WRITE(*,100)' DO YOU WISH TO MAKE ANY CHANGES IN THE NETWORK? Y/N'
   READ(*,300)DATA
   IF (DATA.EQ.'Y' .OR. DATA.EQ.'y') THEN
      WRITE(*,100)' CHANGE TOTAL LOAD? Y/N'
      READ(*,300)DATA
      IF (DATA.EQ.'Y' .OR. DATA.EQ.'y') THEN
         CALL INC
      ELSE
         CALL CHANGE
      ENDIF
   ENDIF
   WRITE(*,100)' ON WHICH FILE DO YOU WISH TO STORE IT?'
   READ(*,300) CH2
   CALL WRITEF(CH2)
ELSEIF (DATA.NE.'N') THEN
   GOTO 40
ELSE
   READY=.TRUE.
   GOTO 40
ENDIF
ENDIF
CALL INIT(WR)
DO 50 COU=1,MAXNRT
   CALL FILM(COU)
50 CONTINUE
DO 60 COU=1,MAXNRF
   CALL FILM(COU,VIO)
60 CONTINUE
C The network is now initiated
C Identify fault and start clock
CALL FLTID(FAULT)
TIME=MSCPU()

C RUN ALGORITHM

CALL STEP1(FAULT)
CALL STEP2(VIO)
IF (VIO) THEN
   CALL STEP3
ENDIF

C STOP CLOCK

TIME=MSCPU()-TIME

C CLEAN UP AMONG SWITCHING OPERATIONS AND PRESENT RESULT

CALL GARBAGE
CALL RESULT(TIME)
WRITE(*,100)'ANOTHER FAULT? Y/N'
READ(*,300) DATA
READY=DATA.EQ.'N' .OR. DATA.EQ.'n'
IF (.NOT. READY) THEN
   GOTO 20
ENDIF

100 FORMAT('0',TR6,A)
200 FORMAT(' ',T10,A)
300 FORMAT(' ',T1,A)
STOP
END

SUBROUTINE RESULT(TIME)

C CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C AUTHOR: MAGNUS BERGSTRAND

C VERSION 1

C DATE: 1989-10-07

C PURPOSE

C PRESENTS THE RESULT

C VARIABLES

C TIME IS EXECUTION TIME IN ms

C CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

INCLUDE 'CONSTANTS.FOR'
INCLUDE 'COMMON.FOR'

INTEGER TIME,1
REAL SQRT

INTRINSIC SQRT

WRITE(*,100)'TOTAL COMPUTATION TIME OF ALGORITHM',TIME,' ms'
WRITE(*,*)
WRITE(*,200)'THE FOLLOWING SWITCHES HAS CHANGED STATUS'
WRITE(*,*)
DO 10 I=1,MAXNRS
   IF (CHGSWI(I)) THEN
      WRITE(*,300) I,SWINR(I)
   ENDIF
10 CONTINUE

STOP
END
ENDIF
10 CONTINUE
WRITE(*,*)
WRITE(*,200)'TOTAL DEENERGIZED LOAD DUE TO FAULT'
WRITE(*,500) SQRT(ISOLAT(1)**2+ISOLAT(2)**2)
WRITE(*,*)
WRITE(*,200)'TOTAL RESTORED LOAD'
WRITE(*,500) SQRT(RESTOR(1)**2+RESTOR(2)**2)
WRITE(*,*)
100 FORMAT('' ,TR6,A,IS,A)
200 FORMAT('' ,TR6,A)
300 FORMAT('' ,TR15,IS,TR3,12)
500 FORMAT('' ,TR6,E12.5)
RETURN
END

SUBROUTINE GARBAG

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

AUTHOR MAGNUS BERGSTRAND

VERSION 1

DATE 1987-09-07

PURPOSE
DECREASE THE NUMBER OF NECESSARY SWITCHING OPERATIONS

DESCRIPTION
WHEN FOR INSTANCE TWO SECTIONS HAS BEEN DEENERGIZED IN STEP 3
TWO SWITCHES HAVE CHANGED STATUS. THE SAME RESULT COULD MAYBE
BEEN GIVEN IF ONLY THE SWITCH NEAREST THE FEEDER CHANGED STATUS
ALL SWITCHES ARE EXAMINED
WHEN AN OPEN SWITCH IS FOUND THE SUBROUTINE 'MOVES' TOWARDS
THE FEEDER.
IF AN OPEN SWITCH IS FOUND ON THE WAY IT IS POSSIBLE TO REDUCE THE NUMBER
OF SWITCHING OPERATIONS WITH 1.
THE SWITCHING OPERATION PERFORMED ON THE STARTING SWITCH WAS UNNECESSARY

SEE ALSO
FILES ROUTINES COMMENTS
STEP2 STEPS THE LOAD CURTAILMENT
MOVING STESPUP TAKES ONE STEP UP

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

INCLUDE 'CONSTANTS.FOR'
INCLUDE 'COMMON.FOR'

INTEGER P(2)
INTEGER I
LOGICAL READY

DO 10 I=1,MAXNRS
  IF (SWITCH(I,1)).EQ.OPEN .AND. CHGSWI(I)) THEN
    P(1)=SMITYP
    P(2)=I
    READY=.FALSE.
  20   IF (.NOT. READY) THEN
    CALL STEPUP(P,.FALSE.)
    READY=(P(1).EQ.SMITYP .OR. P(1).EQ.FEETYP)
    READY=(READY .OR. P(I).EQ.TRATYP)
  GOTO 20
10 ENDIF
IF (X > E1) RETURN
END IF
END IF
RETURN
END

IF (X < E2) THEN
CLOSE
ELSE OPEN
END IF
CLOSE
END IF
END IF
IN THIS FILE ALL CONSTANTS USED ARE DECLARED

READ MAXNRx AS MAX NUMBER OF x

BRANCH IS THE MAXIMUM NUMBER OF BRANCHES IN ANY BRANCH POINT IN THE SYSTEM

PARAMETER (MAXNRT=4, MAXNRF=6, MAXNRN=78, MAXNRB=12)
PARAMETER (MAXNRC=177, BRANCH=4, MAXNRS=64)
PARAMETER (TRATYP=1, FEETYP=2, CABTYP=3)
PARAMETER (BRATYP=4, NODTYP=5, SWITYP=6)
PARAMETER (OPEN=0, CLOSED=1, UP=1, DOWN=0)
PARAMETER (NET=10)
PARAMETER (BETA=0.00001)
COMMON FOR

AUTHOR MAGNUS BERGSTRAND

VERSION 1

DATE 1987-09-30

READ ALSO 'SERVICE RESTORATION IN ELECTRICAL DISTRIBUTION NETWORKS'
BY MAGNUS BERGSTRAND (APPENDIX SECTION STRUCTURE CHART)

DESCRIPTION:
IN THIS FILE ALL COMMON VARIABLES IS DECLARED.
BELOW FOLLOWS A SHORT DESCRIPTION OF EACH VARIABLE.

ALFA IS THE NODE PRIORITY

IN 'SWINR' THE ORDER OF THE SWITCHING OPERATIONS IS STORED

IN ISOLAT AND RESTORE ARE THE AMOUNT OF ISOLATED LOAD AND RESTORED LOAD
STORED (W,VAR)

'TRAFOS' CONTAINS ALL INFORMATION ABOUT THE DIFFERENT TRANSFORMERS IN THE
NETWORK EXCEPT THE CABLE TO THE NETWORK WHICH IS STORED IN 'TRANEI'

MSFEED CONTAINS ALL DATA FOR THE FEEDERS

CABTOP CONTAINS TOPOLOGY FOR CABLES

CABPHY=PHYSICAL DATA FOR CABLES. CURRENTS IN AMPERE AND IMPEDANCES
IN OHM.

BRAPOI CONTAINS THE TOPOLOGY FOR BRANCH POINTS

NODETO CONTAINS THE TOPOLOGY FOR THE NODES

NODEPH CONTAINS THE PHYSICAL DATA FOR NODES
VOLTAGE IN VOLT, IMPEDANCE IN OHM AND POWER IN W

SWITCH CONTAINS ALL USEFUL INFORMATION ABOUT THE SWITCHES

IN BRAPH THE IMPEDANCE SEEN LOOKING DOWN A BRANCH IS STORED (OHM)

A CHART OF WHERE THE DATA IS STORED IN THE FOLLOWING MATRICES:
TRAFOS,MSFEED,CABTOP,CABPHY,BRAPOI,NODETO,NODEPH,SWITCH IS FOUND
IN THE REPORT 'SERVICE RESTORATION IN ELECTRIC DISTRIBUTION NETWORKS'

TRAMAX IS THE MAXIMUM VOLTAGE FOR ANY TRANSFORMER IN THE NETWORK

VIOVEC IS THE VIOLATION VECTOR.
THE FIRST INDEX INDICATES THE (VIOLATED) FEEDER

THE LOGICAL VECTOR CHGSWI INDICATES IF A SWITCH HAS CHANGED STATUS
COMPARED WITH THE INITIAL STATE

IMPLICIT LOGICAL (A-Z)
REAL ALFA(MAXNRM)
INTEGER SWINR(MAXNRS)
REAL ISOLAT(2),RESTOR(2)
REAL TRAFOS(1:MAXNRT,1:7),TRAMAX
INTEGER TRANEI(1:MAXNRT)
INTEGER MNSFED(1:MAXNRF,1:3)
INTEGER Cabtop(1:MAXNRC,1:4)
REAL CAPHY(1:MAXNRC,1:4)
INTEGER BRAPOI(1:MAXNRB,1:BRANCH+3)
INTEGER NODETO(1:MAXNRR,1:4)
REAL NODEPH(1:MAXNRR,1:5),VOLLIM
INTEGER SWITCH(1:MAXNRS,1:5)
REAL BRAPH(1:MAXNRB,3:BRANCH+2,1:2)
REAL VIOVEC(MAXNRF,2)
LOGICAL CHGSWI(MAXNRS)
COMMON TRAFOS,CAPHY,NODEPH
COMMON TRANEI,MNSFED,CABTOP
COMMON BRAPOI,NODETO,SWITCH
COMMON TRAMAX,VOLLIM,RESTOR
COMMON BRAPH,VIOVEC,ISOLAT
COMMON CHGSWI,SHNR,ALFA
SUBROUTINE CHANGE

AUTHOR MAGNUS BERGSTRAND

VERSION 1

DATE 1989-09-30

PURPOSE:
THIS PROCEDURE HANDLES THE CHANGE OF PHYSICAL
DATA IN A SINGLE ELEMENT

SEE ALSO:

FILE       ROUTINE       COMMENT
CONSTANTS   ALL CONSTANTS ARE DECLARED
COMMON      ALL COMMON VARIABLES ARE DECLARED
CHG        INC            CHANGES TOTAL LOAD

VARIABLES:

VOLTAGE IN VOLT
CURRENT IN AMPERE
IMPEDANCE IN OHM

INCLUDE 'CONSTANTS.FOR'
INCLUDE 'COMMON.FOR'

EXTERNAL WRITEF

INTEGER P(2), I

WRITE(*,100)
WRITE(*,*) ' CHANGE IN (TYPE , NR) ?'
READ(*,*) P(1), P(2)

THE POINT WHICH WILL BE CHANGED WAS READ

10 IF (P(1),EQ,TRATYP) THEN
    WRITE(*,*) ' NEW P MAX IS'
    READ(*,*) TRAFOS(P(2),1)
    WRITE(*,*) ' NEW Q MAX IS'
    READ(*,*) TRAFOS(P(2),2)
ELSEIF (P(1),EQ,CABYP) THEN
    WRITE(*,*) ' NEW R IS'
    READ(*,*) CABPHY(P(2),1)
    WRITE(*,*) ' NEW X IS'
    READ(*,*) CABPHY(P(2),2)
    WRITE(*,*) ' NEW IMAX IS'
    READ(*,*) CABPHY(P(2),3)
ELSEIF (P(1),EQ,NODYP) THEN
    WRITE(*,*) ' R IS',NODEPH(P(2),4)
SUBROUTINE INC

AUTHOR MAGNUS BERGSTRAND

VERSION 1

DATE 1989-09-30

PURPOSE:
READS A FACTOR P WHICH ALL THE NODE IMPEDANCES
IN THE NETWORK WILL BE MULTIPLIED WITH CAUSING
A CHANGE IN THE NETWORK LOAD

SEE ALSO:

FILE ROUTINE COMMENT
CONSTANTS ALL CONSTANTS ARE DECLARED
COMMON ALL COMMON VARIABLES ARE DECLARED
CHG CHANGES PHYSICAL DATA OF ANY
COMPONENT IN THE NETWORK

VARIABLES:

VOLTAGE IN VOLT
CURRENT IN AMPERE
IMPEDANCE IN OHM

INCLUDE 'CONSTANTS.FOR'
INCLUDE 'COMMON.FOR'

REAL P
INTEGER I,J

WRITE(*,*),' IMPEDANCES WILL BE MULTIPLIED WITH?'
READ(*,*) P
DO 10 I=1,MAXHRN
   NODEPH(I,4)=NODEPH(I,4)*P
   NODEPH(I,5)=NODEPH(I,5)*P
10 CONTINUE
RETURN
END
SUBROUTINE FLTID(FAULT)

FAULT (OUT) IS THE FAULTED POINT (TYPE, HR)

INTEGER FAULT(1:2)

READ(*,*) FAULT(1)
IF ((FAULT(1).LE.0).OR.(FAULT(1).GE.6)) GOTO 10
WRITE(*,*) ' WHICH TYPE OF COMPONENT?'
READ(*,*) TYPE
IF (FAULT(1).EQ.1) THEN
  WRITE(*,*) ' WHICH TRANSFORMER?'
ELSEIF (FAULT(1).EQ.2) THEN
  WRITE(*,*) ' WHICH FEEDER?'
ELSEIF (FAULT(1).EQ.3) THEN
  WRITE(*,*) ' WHICH CABLE?'
ELSEIF (FAULT(1).EQ.4) THEN
  WRITE(*,*) ' WHICH BRANCHINGPOINT?'
ELSEIF (FAULT(1).EQ.5) THEN
  WRITE(*,*) ' WHICH NODE?'
ENDIF
READ(*,*) HR
RETURN
END
SUBROUTINE WRITEF(N)

AUTHOR MAGNUS BERGSTRAND

VERSION 1

DATE 1987-10-08

PURPOSE
SAVES THE NETWORK TOPOLOGY AND PHYSICAL DATA ON A FILE

DESCRIPTION
USES FORMATTED OUTPUT
READ ALSO USER'S GUIDE IN 'SERVICE RESTORATION IN ELECTRICAL DISTRIBUTION NETWORKS' BY MAGNUS BERGSTRAND WHERE EXPLANATIONS AND EXAMPLES ARE FOUND.

SEE ALSO
FILES ROUTINES COMMENTS
OUTFI INF1 READS A FILE CONTAINING THE NETWORK TOPOLOGY
AND PHYSICAL DATA FOR A NETWORK USING THE SAME FORMAT

VARIABLES
THE STRING N CONTAINS THE NAME OF THE FILE

INCLUDE 'CONSTANTS.FOR'
INCLUDE 'COMMON.FOR'

INTEGER I,J
CHARACTER*7 N

OPEN(1,FILE=N,STATUS='NEW',ACCESS='SEQUENTIAL',FORM='FORMATTED')
WRITE(1,400) MAXNRT
WRITE(1,200) TRAHAX
DO 10 I=1,MAXNRT
   WRITE(1,300) TRAFOS(I,1),TRAFOS(I,2),TRANEI(I)
10   CONTINUE
WRITE(1,400) MAXNRF
DO 20 I=1,MAXNRF
   WRITE(1,400)MSFEED(I,1),MSFEED(I,3)
20   CONTINUE
WRITE(1,400) MAXNRC
DO 40 I=1,MAXNRC
   DO 50 J=1,3
      WRITE(1,200) CABPHY(I,J)
   50 CONTINUE
   DO 60 J=1,4
      WRITE(1,400)CABTOP(I,J)
   60 CONTINUE
40 CONTINUE
WRITE(1,400) MAXNRE
DO 70 I=1,MAXNRE
   WRITE(1,400) BRAPOI(I,1)
DO 80 J=3,(BRAPQI(I,1)+2)
   WRITE(1,600) BRAPQI(I,J)
80   CONTINUE
70 CONTINUE
WRITE(1,600) MAXNRN
WRITE(1,200) VOLLIM
DO 90 I=1,MAXNRN
   WRITE(1,400) NODETO(I,1),NODETO(I,2),NODETO(I,3)
   WRITE(1,500) NODEPH(I,4),NODEPH(I,5)
   WRITE(1,200) ALFA(I)
90 CONTINUE
WRITE(1,600) MAXNRS
DO 100 I=1,MAXNRS
   WRITE(1,400) SWITCH(I,1),SWITCH(I,2)
100 CONTINUE
DO 110 I=1,MAXNRS
   WRITE(1,600) SWITCH(I,1)
110 CONTINUE
CLOSE(1,STATUS='KEEP')
200 FORMAT']]['TR2,E12.5]
300 FORMAT(['']TR2,2(E12.5,TR2),16)
400 FORMAT(['']2(TR2,16))
500 FORMAT(['']2(TR2,E12.5))
600 FORMAT(['']TR2,16)
RETURN
END

SUBROUTINE INF1(N)
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
AUTHOR MAGNUS BERGSTRAND
C
VERSION 1
C
DATE 1989-10-08
C
PURPOSE
READS A FILE CONTAINING THE NETWORK TOPOLOGY AND PHYSICAL DATA
FOR A NETWORK.
C
DESCRIPTION
FORMATTED INPUT IS USED.
READ ALSO USER'S GUIDE IN 'SERVICE RECONSTRUCTION IN ELECTRICAL
DISTRIBUTION NETWORKS' BY MAGNUS BERGSTRAND WHERE EXPLANATIONS AND
EXAMPLES ARE FOUND.
C
SEE ALSO
FILES ROUTINES COMMENTS
OUTFI WRITEF SAVES A NETWORK ON A FILE USING THE SAME FORMAT
C
VARIABLES
THE STRING N IS THE NAME OF THE FILE
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
INCLUDE 'CONSTANTS.FOR'
INCLUDE 'COMMON.FOR'

INTEGER I,J,K
CHARACTER*7 N
OPEN(1,FILE=N,STATUS='OLD',ACCESS='SEQUENTIAL',FORM='FORMATTED')
REWIND 1
READ(1,600) I
IF (I .EQ. MAXNRT) THEN
  READ(1,200) TRAMAX
  DO 10 J=1,I
    READ(1,300) TRAFOS(J,1),TRAFOS(J,2),TRANEI(J)
  10 CONTINUE
ELSE
  WRITE(*,*)' WRONG NR OF TRAFOS',I,MAXNRT
  STOP
ENDIF
READ(1,2) I
IF (I .EQ. MAXNRF) THEN
  DO 20 J=1,I
    READ(1,400) MSPEED(J,1),MSPEED(J,2)
  20 CONTINUE
ELSE
  WRITE(*,*)' WRONG NR OF FEEDERS',I,MAXNRF
  STOP
ENDIF
READ(1,600) I
IF (I .EQ. MAXNRC) THEN
  DO 30 J=1,I
    DO 40 K=1,3
      READ(1,200) CABPHY(J,K)
    40 CONTINUE
    DO 50 K=1,4
      READ(1,600) CASTOP(J,K)
    50 CONTINUE
  30 CONTINUE
ELSE
  WRITE(*,*)' WRONG NR OF CABLES',I,MAXNRC
  STOP
ENDIF
READ(1,600) I
IF (I .EQ. MAXNRB) THEN
  DO 60 J=1,I
    READ(1,600) BRAPOI(J,1)
    DO 70 K=3,2*BRAPPOI(J,1))
      READ(1,600) BRAPOI(J,K)
    70 CONTINUE
  60 CONTINUE
ELSE
  WRITE(*,*)' WRONG NR BRANCHPOINTS',I,MAXNRB
  STOP
ENDIF
READ(1,600) I
IF (I .EQ. MAXNRRN) THEN
  READ(1,200) VOLLIM
  DO 80 J=1,I
    READ(1,400) NODETO(J,2),NODETO(J,3)
    READ(1,500) NODEPH(J,4),NODEPH(J,5)
    READ(1,200) ALFA(J)
  80 CONTINUE
ELSE
  WRITE(*,*)' WRONG NR OF NODES',I,MAXNRRN
  STOP
ENDIF
READ(1,600) I
IF (I .EQ. MAXNQRS) THEN
  DO 90 J=1,I
    READ(1,400) SWITCH(J,3),SWITCH(J,4)
  90 CONTINUE
  DO 100 J=1,I
    READ(1,600) SWITCH(J,1)
  100 CONTINUE
ELSE
...
SUBROUTINE LINES(NR)
C
C
C
C
INTEGER NR, I
DO 10 I = 1, NR
   WRITE(*,*)
10 CONTINUE
RETURN
END

SUBROUTINE TYPETABLE
C
C
C
C
WRITE(*,*)
WRITE(*,*) ' TYPES IN THE NETWORK'
WRITE(*,*)
WRITE(*,*) ' TRAFO ', 14
WRITE(*,*) ' FEEDER ', 12
WRITE(*,*) ' CABLE ', 13
WRITE(*,*)
WRITE(*,*)
WRITE(*,*)
WRITE(*,*)
WRITE(*,*)
RETURN
END
FETCH FOR

SUBROUTINE GETPOI(SWI,P)

AUTHOR MAGNUS BERGSTRAND

VERSION 1

DATE 1989-10-07

PURPOSE
FETCHES THE POINT (TYPE,NR) ON THE OTHER SIDE OF AN OPEN SWITCH

SEE ALSO
FILES ROUTINES COMMENTS
FETCH GETVOL GETS THE VOLTAGE ON THE OTHER SIDE OF AN OPEN SWITCH
FETCH GETFEE GETS THE FEEDER IDENTITY ON THE OTHER SIDE OF AN OPEN SWITCH
FETCH REALFE GETS THE FEEDER IDENTITY ON THE OTHER SIDE OF AN OPEN SWITCH AND TESTS THAT IT IS AN UNBROKEN CONNECTION
FETCH GETPLI COMPUTES THE MARGIN OF A TRANSFORMER GETTING VIOLATED (REAL POWER)
FETCH GETQLI COMPUTES THE MARGIN OF A TRANSFORMER GETTING VIOLATED (REACTIVE POWER)

CONSTANTS ALL CONSTANTS DECLARED
COMMON ALL COMMON VARIABLES DECLARED

VARIABLES
SWI IS THE NUMBER OF THE OPEN SWITCH
P (QUT) IS THE POINT (TYPE,NR)

INCLUDE 'CONSTANTS.FOR'
INCLUDE 'COMMON.FOR'

INTEGER SWI,P(2)

IF (SWITCH(SWI,2).EQ.3) THEN
  I=SWITCH(SWI,4)
ELSE
  I=SWITCH(SWI,3)
ENDIF

IF (GABTOP(I,1).EQ.SWITYP .AND. GABTOP(I,2).EQ.SWI) THEN
  P(1)=GABTOP(I,3)
  P(2)=GABTOP(I,4)
ELSE
  P(1)=GABTOP(I,1)
  P(2)=GABTOP(I,2)
ENDIF
RETURN
END

REAL FUNCTION GETVOL(SWI)
VERSION 1

DATE 1969-10-07

PURPOSE
GETS THE VOLTAGE ON THE OTHER SIDE OF AN OPEN SWITCH

DESCRIPTION
FIRST GETS THE POINT AND THEN THE VOLTAGE

SEE ALSO
FILES Routines Comments
CONSTANTS ALL CONSTANTS DECLARED
COMMON ALL COMMON VARIABLES DECLARED
FETCH GETPOI GETS THE POINT ON THE OTHER SIDE OF AN OPEN SWITCH
FETCH GETFEI GETS THE FEEDER IDENTIY ON THE OTHER SIDE OF AN OPEN SWITCH
FETCH REALFE GETS THE FEEDER IDENTIY ON THE OTHER SIDE OF AN OPEN SWITCH AND TESTS THAT IT IS AN UNBROKEN CONNECTION
FETCH GETFLI COMPUTES THE MARGIN OF A TRANSFORMER GETTING VIOLATED (REAL POWER)
FETCH GETQLI COMPUTES THE MARGIN OF A TRANSFORMER GETTING VIOLATED (REACTIVE POWER)

VARIABLES
SWI IS THE NUMBER OF THE OPEN SWITCH

INCLUDE 'CONSTANTS.FOR'
INCLUDE 'COMMON.FOR'

INTEGER P(2)
INTEGER SWI
REAL VOL

IF (SWITCH(SWI,1).EQ.CLOSED) THEN
  WRITE(*,*) ' WRONG IN GETVOL'
  STOP
ELSE
  CALL GETPOI(SWI,P)
  VOL=0.0
  IF (P(1).EQ.NODTP) THEN
    GETVOL=VOL+NODEP(H(P(2),3)-VOLLIM
  ELSEIF (P(1).EQ.BATYP) THEN
    I=BRAP(OI(P(2),2)
    VOL=VOL-CABPHY(I,4)*SQRT(CABPHY(I,1)**2+CABPHY(I,2)**2)
  CALL STEPUP(P,.FALSE.)
  GOTO 10
  ELSEIF (P(1).EQ.FEETY) THEN
    GETVOL=TRAMAX
  ELSE
    WRITE(*,*) ' WRONG IN GETVOL'
    GETVOL=0.0
  STOP
ENDIF
ENDIF
RETURN
END

INTEGER FUNCTION GETFEI(SWI)

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
AUTHOUR MAGNUS BERGSTRAND

VERSION 1

DATE 1989-10-07

PURPOSE
GETS THE FEEDER IDENTITY ON THE OTHER SIDE OF AN OPEN SWITCH

SEE ALSO
FILES ROUTINES COMMENTS
FETCH GETPO1 GETS THE POINT ON THE OTHER SIDE OF AN OPEN SWITCH
FETCH GETVOL GETS THE VOLTAGE ON THE OTHER SIDE OF AN OPEN SWITCH
FETCH REALFE GETS THE FEEDER IDENTITY ON THE OTHER SIDE OF AN OPEN
SWITCH AND TESTS THAT IT IS AN UNBROKEN CONNECTION
FETCH GETPLI COMPUTES THE MARGIN OF A TRANSFORMER GETTING VIOLATED
(REAL POWER)
FETCH GETQLI COMPUTES THE MARGIN OF A TRANSFORMER GETTING VIOLATED
(REACTIVE POWER)
MOVING STEPUP TAKES ONE STEP UP
CONSTANTS ALL CONSTANTS DECLARED
COMMON ALL COMMON VARIABLES DECLARED

VARIABLES
SWI IS THE NUMBER OF THE OPEN SWITCH

INCLUDE 'CONSTANTS.FOR'
INCLUDE 'COMMON.FOR'

INTEGER SWI,P(2)

IF (SWITCH(SWI,1).EQ.CLOSED) THEN
    WRITE(*,*)' WRONG IN GETFFE'
    STOP
ELSE
    CALL GETPO1(SWI,P)
    IF (P(1).EQ.MODTyp) THEN
        GETFFE=MODETO(P(2),4)
    ELSEIF (P(1).EQ.BRATYP) THEN
        CALL STEPUP(P, .FALSE.)
    GOTO 10
    ELSEIF (P(2).EQ.FEETYP) THEN
        GETFFE=P(2)
    ELSEIF (P(1).EQ.SWITYP) THEN
        CALL STEPUP(P, .FALSE.)
    GOTO 10
    ELSE
        WRITE(*,*)' DEGENERATED NET OR WRONG USE OF GETFFE'
    ENDIF
ENDIF
RETURN
END

SUBROUTINE REALFE(SWI,FE,FOUND)

AUTHOUR MAGNUS BERGSTRAND

VERSION 1
DATE 1989-10-07

PURPOSE
GETS THE FEEDER IDENTITY ON THE OTHER SIDE OF AN OPEN SWITCH AND TESTS IF IT IS AN UNBROKEN CONNECTION

DESCRIPTION
FIRST A CALL OF GETFEE IS MADE THEN STARTING FROM THE POINT ON THE OTHER SIDE OF THE SWITCH IT MOVES UPWARDS UNTIL A FEEDER OR AN OPEN SWITCH IS FOUND. IF A FEEDER IS FOUND THEN 'FOUND'=TRUE

SEE ALSO
FILES Routines COMMENTS
CONSTANTS ALL CONSTANTS DECLARED
COMMON ALL COMMON VARIABLES DECLARED
FETCH GETPO1 GETS THE POINT ON THE OTHER SIDE OF AN OPEN SWITCH
FETCH GETVOL GETS THE VOLTAGE ON THE OTHER SIDE OF AN OPEN SWITCH
FETCH GETFEE GETS THE FEEDER IDENTITY ON THE OTHER SIDE OF AN OPEN SWITCH
FETCH GETPLI COMPUTES THE MARGIN OF A TRANSFORMER GETTING VIOLATED (REAL POWER)
FETCH GETQLI COMPUTES THE MARGIN OF A TRANSFORMER GETTING VIOLATED (REACTIVE POWER)
MOVING STEPUP TAKES ONE STEP UP

VARIABLES
SWI IS THE NUMBER OF THE OPEN SWITCH
FEE (OUT) IS THE NUMBER OF THE FEEDER
FOUND (OUT) INDICATES WHETHER IT HAS AN UNBROKEN CONNECTION OR NOT

INCLUDE 'CONSTANTS.FOR'
INCLUDE 'COMMON.FOR'

INTEGER SWI,FEE
INTEGER P(2)
LOGICAL FOUND,READY
INTEGER GETFEE

EXTERNAL GETFEE

FEE=GETFEE(SWI)
CALL GETPOI(SWI,P)
READY=.FALSE.

10 IF (.NOT. READY) THEN
   CALL STEPUP(P,.FALSE.)
   IF (P(1),EQ.TRATYP) THEN
      WRITE(*,*) ' DEGENERATED NET.WRONG IN REALFE' STOP
   ELSEIF (P(1),EQ.FEETYP) THEN
      READY=.TRUE.
   ELSEIF (P(1),EQ.SWITYP) THEN
      IF (SWITCH(P(2),1),EQ.OPEN) THEN
         READY=.TRUE.
      ENDIF
   ENDIF
GOTO 10
ENDIF
FOUND=(FEE,EQ.P(2) .AND. P(1),EQ.FEETYP)
RETURN
END

REAL FUNCTION GETPLI(SWI)
AUTHOR: MAGNUS BERGSTRAND

VERSION 1

DATE: 1989-10-07

PURPOSE

GETS THE MARGIN OF A TRANSFORMER GETTING VIOLATED (REAL POWER)

SEE ALSO

FILES ROUTINES COMMENTS
CONSTANTS ALL CONSTANTS DECLARED
COMMON ALL COMMON VARIABLES DECLARED
FETCH GETPOL GETS THE POINT ON THE OTHER SIDE OF AN OPEN SWITCH
FETCH GETVOL GETS THE VOLTAGE ON THE OTHER SIDE OF AN OPEN SWITCH
FETCH GETFEED GETS THE FEEDER IDENTITY ON THE OTHER SIDE OF AN OPEN SWITCH
FETCH REALFE GETS THE FEEDER IDENTITY ON THE OTHER SIDE OF AN OPEN SWITCH AND TESTS THAT IT IS AN UNBROKEN CONNECTION
FETCH GETQLI COMPUTES THE MARGIN OF A TRANSFORMER GETTING VIOLATED (REACTIVE POWER)

VARIABLES

THE MARGIN IN W.
SWI IS THE NUMBER OF THE OPEN SWITCH

INCLUDE 'CONSTANTS.FOR'
INCLUDE 'COMMON.FOR'

INTEGER SWI,D(2)
INTEGER FEE,TRA

IF (SWITCH(SWI,1),EQ,CLOSED) THEN
   WRITE(*,99) 'MISUSED GETQLI'
   STOP
ELSE
   FEE=GETFEE(SWI)
   TRA=MSFEED(FEE,1)
   GETQLI=TRAFOS(TRA,1)-TRAFOS(TRA,3)
ENDIF
RETURN
END

REAL FUNCTION GETQLI(SWI)

AUTHOR: MAGNUS BERGSTRAND

VERSION 1

DATE: 1989-10-07

PURPOSE

GETS THE MARGIN OF GETTING VIOLATED FOR A TRANSFORMER ON THE OTHER SIDE OF AN OPEN SWITCH (REACTIVE POWER)

SEE ALSO

FILES ROUTINES COMMENTS
CONSTANTS ALL CONSTANTS DECLARED
COMMON

FETCH GETPO1 GETS THE POINT ON THE OTHER SIDE OF AN OPEN SWITCH

FETCH GETPO1 GETS THE VOLTAGE ON THE OTHER SIDE OF AN OPEN SWITCH

FETCH GETPO1 GETS THE FEEDER IDENTITY ON THE OTHER SIDE OF AN OPEN SWITCH

FETCH REALFE GETS THE FEEDER IDENTITY ON THE OTHER SIDE OF AN OPEN SWITCH AND TESTS THAT IT IS AN UNBROKEN CONNECTION

FETCH GETPLI COMPUTES THE MARGIN OF A TRANSFORMER GETTING VIOLATED (REAL POWER)

FETCH GETQLI COMPUTES THE MARGIN OF A TRANSFORMER GETTING VIOLATED (REACTIVE POWER)

VARIABLES

THE MARGIN IN VA.

SWI IS THE NUMBER OF THE OPEN SWITCH

INCLUDE 'CONSTANTS.FOR'

INCLUDE 'COMMON.FOR'

INTEGER SWI,F(2)
INTEGER FEE,TRA

IF (SWITCH(SWI,1).EQ.CLOSED) THEN
  WRITE(*,*)' MISUSED GETQLI'
  STOP
ELSE
  FEE=GETFEE(SWI)
  TRA=NSFEED(FEE,1)
  GETQLI=TRAFOS(TRA,2)-TRAFOS(TRA,4)
ENDIF
RETURN
END
SUBROUTINE STEP1(FAULT)

AUTHOR MAGNUS BERGSTRAND

VERSION 1

DATE 1989-10-07

PURPOSE

ISOLATES A FAULTED ZONE AND CONNECTS DEENERGIZED NON FAULTED ZONES TO ADJACENT FEEDERS.

METHOD

THE METHOD IS DESCRIBED IN 'SERVICE RESTORATION IN ELECTRIC DISTRIBUTION NETWORKS' BY MAGNUS BERGSTRAND

THE METHOD WAS FIRST DESCRIBED BY AOKI IN 'A NEW ALGORITHM FOR SERVICE RESTORATION IN DISTRIBUTION SYSTEMS' IEEE/PES WINTER MEETING, NEW YORK 1989 PAPER B9 WM 885-2 PWRD

DESCRIPTION

FAULT IS THE FAULTED POINT (TYPE,NR)

THERE ARE TWO BASIC TYPES OF FAULTS.

FAULTS AMONG THE ZONES OR FAULTS "ABOVE THE FEEDER" (IN A TRANSFORMER)

IF A NODE IS FAULTED THIS IS ISOLATED BY FIRST FINDING THE FIRST SWITCH ABOVE THE NODE. CLOSE IT AND THEN CLOSING ALL FIRST SWITCHES BELOW THE FAULTED NODE. IT CAN BE MORE THAN ONE SWITCH DUE TO BRANCH POINTS.

WHEN A TRANSFORMER IS FAULTED THE FAULT IS ISOLATED BY THE FAULT ITSELF.

ALL THE CIRCUIT BREAKERS ARE TRIPED BY THE FAULT.

AFTER THE FAULT HAS BEEN ISOLATED THE ROOTS OF THE ISOLATED TREES ARE WRITTEN ON A FILE AND THEN READ BACK. STARTING FROM THESE ROOTS ALL POSSIBLE CONNECTIONS ARE EXAMINED AND THE ONE WITH THE LEAST CHANCE OF CAUSING ANY VIOLATION IS SELECTED REENERGIZING THE TREE.

SEE ALSO

FILES ROUTINES COMMENTS

STEP2 STEP2 SECOND STEP IN THE ALGORITHM

STEP2 STEP3 THIRD AND FOURTH STEP IN THE ALGORITHM

STEP1 ABOVEF LOGICAL FUNCTION, IS THE FAULT LOCATED ABOVE ANY FEEDER

MOVING MOVE TAKES ONE STEP UP OR DOWN

MOVING DIREC COMPUTES NEW DIRECTIONS

MOVING STEPUP TAKES ONE STEP UP

MOVING STEPDO TAKES ONE STEP DOWN

COMP CON CONNECTS DEENERGIZED ZONES

CONSTANTS ALL CONSTANTS

COMMON ALL COMMON VARIABLES ARE DECLARED

VARIABLES

FAULT IS THE FAULTED POINT (TYPE,NR)

INCLUDE 'CONSTANTS.FOR'

INCLUDE 'COMMON.FOR'
INTEGER FAULT(1:2)
INTEGER TRAFFEEDER,ISOISW
INTEGER CURPOI(2),P(2),I
INTEGER DIR,BRA,OLDP(2),Y,J
LOGICAL READY,MR,CHG,ABOVEF,OK

PARAMETER(TRASH=1)
EXTERNAL STEPUP,STEPDO
EXTERNAL MOVE,DIREC
EXTERNAL CON,ABOVEF

OPEN(STATUS='SCRATCH',ACCESS='SEQUENTIAL')
CURPOI(1)=FAULT(1)
CURPOI(2)=FAULT(2)
WR=.FALSE.
FEEDER=0
ISOISW=0
ISOLAT(1)=0.0
ISOLAT(2)=0.0
IF (.NOT. ABOVEF(CURPOI)) THEN

C FAULT AMONG SECTIONS (BELOW FEEDER)

READY=CURPOI(1).EQ.TRATYP
10 IF (.NOT. READY) THEN

C FIND FIRST SWITCH

IF (CURPOI(1).EQ.CABTYP) THEN
    CALL STEPDO(CURPOI,BRA,WR)
ENDIF
CALL STEPUP(CURPOI,WR)
READY=CURPOI(1).EQ.SWITYP .OR. CURPOI(1).EQ.TRATYP
GOTO 10

ENDIF
DIR=DOWN
BRA=3
P(1)=CURPOI(1)
P(2)=CURPOI(2)
READY=.FALSE.
CHG=.FALSE.

20 IF (.NOT. READY) THEN

C FIND ALL SWITCHES BELOW THE FAULTED ZONE
C THESE SWITCHES ARE OPENED
C THEY MUST BE OPENED IN THIS PERHAPS ODD WAY BECAUSE OTHERWISE WILL
C NEITHER STEPUP NOR STEPDO WORK CORRECTLY

IF (CHG) THEN
    SWITCH(Y,1)=CLOSED
ENDIF
CALL MOVE(P,OLDP,DIR,BRA,WR)
IF (CHG) THEN
    SWITCH(Y,1)=OPEN
ENDIF
READY=AND. P(1).EQ.CURPOI(1))
IF (READY) THEN
GOTO 20
ENDIF
CHG=.FALSE.
IF (P(1).EQ.SWITYP) THEN
    I=P(2)
    IF (SWITCH(I,1).EQ.CLOSED) THEN
        SWITCH(I,1)=OPEN
        SWITCH(I,5)=0
    ENDIF
ENDIF

SWINR(1)=1
CHGSWI(1)=.NOT. CHGSWI(1)
CHG=.TRUE.
I=1
IF (SWITCH(I,2).EQ.3) THEN
  J=SWITCH(I,4)
ELSE
  J=SWITCH(I,3)
ENDIF

SAVE THE POINT BELOW THE SWITCH THAT HAS ISOLATED THE FAULT
THIS POINT IS THEN USED AS ROOT IN A TREE IN ORDER TO DECIDE
THE BEST WAY TO REENERGIZE THE NON FAULTED ZONES.

IF (CABTOP(J,1).EQ.SWITYP .AND. CABTOP(J,2).EQ.1) THEN
  WRITE(TRASH,*) CABTOP(J,3),CABTOP(J,4)
ELSEIF (CABTOP(J,3).EQ.SWITYP .AND. CABTOP(J,4).EQ.1) THEN
  WRITE(TRASH,*) CABTOP(J,1),CABTOP(J,2)
ELSE
  WRITE(‘*,*) ' WRONG IN STEP 1', STOP
ENDIF
ELSEIF (SWITCH(I,1).EQ.OPEN) THEN
  SWITCH(I,5)=0
ELSE
  WRITE(‘*,*) ' WRONG IN STEP1‘,SWITCH(I,1),OPEN,CLOSED
  STOP
ENDIF
ENDIF
CALL DIREC(P,OLDP,DIR,BRA)
GOTO 20
ENDIF

IF (CURPOI(1).EQ.SWITYP) THEN
  SWITCH(CURPOI(2),1)=OPEN
  SWITCH(CURPOI(2),5)=0
  SWINR(CURPOI(2))=1
  CHGSWI(CURPOI(2))=.NOT. CHGSWI(CURPOI(2))
ENDIF
ENDFILE TRASH
REWIND TRASH
ELSE

THE FAULT WAS IN A TRANSFORMER (ABOVE THE FEEDER)
START FROM THE TRANSFORMER MOVING UPWARDS
OPENING ALL SWITCHES ON THE WAY UNTIL YOU FIND A FEEDER. STOP.
WRITE THE NUMBER OF THE FEEDER ON A FILE
DO THIS FOR ALL POSSIBLE BRANCHES

OPEN(3,STATUS=’SCRATCH’,ACCESS=’SEQUENTIAL’)
READY=CURPOI(1).EQ.TRATYP
100 IF (.NOT. READY) THEN
  IF (CURPOI(1).EQ.CABTOP) THEN
    CALL STEPUP(CURPOI,BRA,WR)
  ENDIF
  CALL STEPUP(CURPOI,WR)
  READY=(CURPOI(1).EQ.SWITYP .OR. CURPOI(1).EQ.TRATYP)
GOTO 100
ENDIF
IF (CURPOI(1).EQ.SWITYP) THEN
  WRITE(3,’*')CURPOI(2)
ENDIF
DIR=DOWN
BRA=3
P(1)=CURPOI(1)
P(2)=CURPOI(2)
READY=.FALSE.
IF (.NOT. READY) THEN
CALL MOVE(P,OLDP,DIR,BRA,.FALSE.)
READY=(P(2).EQ.CURPOI(2).AND. P(1).EQ.CURPOI(1))
ENDIF
GOTO 200
ENDIF
IF (P(1).EQ.FEETYP) THEN
TRAFO=MBFEED(P(2),1)

C ISOLATED LOAD

ISOLAT(1)=TRAFO(TRAFO,3)
ISOLAT(2)=TRAFO(TRAFO,4)
DIR=UP
P1(1)=FEETYP
P1(2)=P(2)
CALL STEPDO(P1,3,.FALSE.)
IF (P1(1).EQ.SWITYP) THEN
I=P1(2)
IF (SWITCH(I,1).EQ.CLOSED) THEN
SWITCH(I,1)=OPEN
SWITCH(I,5)=0
SWINR(I)=1
CHGSI(I)=.NOT. CHGSI(I)
IF (SWITCH(I,2).EQ.3) THEN
J=SWITCH(I,4)
ELSE
J=SWITCH(I,3)
ENDIF
IF (CABTOP(J,1).EQ.SWITYP .AND. CABTOP(J,2).EQ.I) THEN
WRITE (TRASH,*) CABTOP(J,3),CABTOP(J,4)
ELSEIF (CABTOP(J,3).EQ.SWITYP .AND. CABTOP(J,4).EQ.I) THEN
WRITE (TRASH,*) CABTOP(J,1),CABTOP(J,2)
ELSE
WRITE (*,*) 'WRONG IN STEP 1'
STOP
ENDIF
ELSEIF (SWITCH(I,1).EQ.OPEN) THEN
SWITCH(I,5)=0
ENDIF
ENDIF
ELSEIF (P(1).EQ.SWITYP .AND. DIR.EQ.DOWN) THEN
IF (SWITCH(P(2),1).EQ.CLOSED) THEN
WRITE (3,*) P(2)
ENDIF
CALL DIREC(P,OLDP,DIR,BRA)
GOTO 200
ENDIF
ENDIF
IF (CURPOI(1).EQ.SWITYP) THEN
SWITCH(CURPOI(2),1)=OPEN
SWITCH(CURPOI(2),5)=0
SWINR(CURPOI(2))=1
CHGSI(CURPOI(2))=.NOT. CHGSI(CURPOI(2))
ENDIF
ENDIF
ENDFILE TRASH
REWIND TRASH
ENDFILE 3
REWIND 3
READ(3,*,ERR=210) I

C READ THE NUMBER OF THE SWITCHES THAT WILL BE OPENED TO ISOLATE THE
C FAULT

SWITCH(I,1)=OPEN
SWITCH(I,5)=0
SWIHR(I)=1
CHGSWI(1)=.NOT. CHGSWI(1)
GOTO 205

210 ENDIF
30 READ(TRAN.*END=40) CURPO1(1),CURPO1(2)

C READS AND CONNECTS THE DEENERGIZED NODES
CALL CON(CURPO1,OK)
IF (.NOT. OK) THEN
    RESTOR(1)=0,0
    RESTOR(2)=0,0
ENDIF
GOTO 30
40 CLOSE(TRAN,STATUS='DELETE')
IF (OK) THEN
    RESTOR(1)=ISOLAT(1)
    RESTOR(2)=ISOLAT(2)
ENDIF
RETURN
END

LOGICAL FUNCTION ABOVEF(POI)

C CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C AUTHOR: MAGNUS BERGSTRAND
C VERSION 1
C DATE 1967-10-07
C PURPOSE
C INDICATES IF A FAULT HAS OCCURRED ABOVE A FEEDER
C DESCRIPTION
C MOVES UPWARDS UNTIL A FEEDER OR A TRANSFORMER IS FOUND.
C IF A TRANSFORMER IS FOUND THEN TRUE
C SEE ALSO
C FILES ROUTINES COMMENTS
C STEP1 STEP1 ISOLATES THE FAULT AND CONNECTS THE NON FAULTED ZONES.
C CALLS ABOVEF
C MOVING STEPUP TAKES ONE STEP UP
C CONSTANTS ALL CONSTANTS DECLARED
C COMMON ALL COMMON VARIABLES DECLARED

C VARIABLES
C POI IS THE FAULTED POINT (TYPE,NR)
C CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C INCLUDE 'CONSTANTS.FOR'
C INCLUDE 'COMMON.FOR'
C INTEGER POI(2),P(2)
C LOGICAL READY
P(1)=POI(1)
P(2)=POI(2)
READY=(P(1).EQ.FEETYP .OR. P(1).EQ.TRATYP)
10 IF (.NOT. READY) THEN
    CALL STEPUP(P,.FALSE.)
READY = (P(1) .EQ. FEETYP .OR. P(1) .EQ. TRATYP)
GOTO 10
ENDIF
ABOVEF = (P(1) .EQ. TRATYP)
RETURN
END
SUBROUTINE STEPUP(POINT,WR)

AUTHOR MAGNUS BERGSTRAND

VERSION 1

DATE 1989-10-07

PURPOSE
TAKES ONE STEP UP

DESCRIPTION
A CENTRAL SUBROUTINE.
NOTICE THAT POINT CAN BE OF ANY TYPE EXCEPT CABLE.
IN MSFEED, NODETO, BRAPOI AND SWITCH A POINTER UP IS FOUND.
IT POINTS OUT AN INDEX TO THE CABLE LEADING TOWARDS THE TRANSFORMER.
IN CABTOP (CABLE TOPOLOGY) THE TWO END POINTS OF THE CABLE ARE STORED.
THIS SUBROUTINE MOVES 'POINT' TO THE OTHER END POINT IN DIRECTION UP

SEE ALSO
FILES ROUTINES COMMENTS
MOVING STEPDO TAKES ONE STEP DOWN
MOVING MOVE TAKES ONE STEP UP OR DOWN
MOVING DIREC COMPUTES NEW DIRECTION
CONSTANT ALL CONSTANTS DECLARED
COMMON ALL COMMON VARIABLES DECLARED

VARIABLES
POINT IS THE CURRENT POINT (TYPE,NR)
IF WR=TRUE THE CURRENT POINT WILL BE PRINTED ON THE SCREEN

INCLUDE 'CONSTANTS.FOR'
INCLUDE 'COMMON.FOR'

INTEGER POINT(1:2),I
LOGICAL WR

IF (WR) THEN
WRITE(*,'') POINT IN IS',POINT(1),POINT(2)
ENDIF
IF (POINT(1),EQ., BRATYP) THEN
  I=BRAPOI(POINT(2),1)
  IF (CABTOP(1,1),EQ., POINT(1)) AND. CABTOP(1,2),EQ., POINT(2)) THEN
    POINT(1)=CABTOP(1,3)
    POINT(2)=CABTOP(1,4)
  ELSE
    POINT(1)=CABTOP(1,1)
    POINT(2)=CABTOP(1,2)
  ENDIF
ELSEIF (POINT(1),EQ., NODETP) THEN
  I=NODETO(POINT(2),1)
IF (WR) THEN
  WRITE(*,*) ’ CABUP IN NODE IS’, I
ENDIF
I=MODETO(POINT(2),I)
IF (WR) THEN
  WRITE(*,*) ’ CABLE UP IS’, I
ENDIF
IF (CABTOP(I,1).EQ.POINT(1) .AND. CABTOP(I,2).EQ.POINT(2)) THEN
  POINT(1)=CABTOP(I,3)
  POINT(2)=CABTOP(I,4)
ELSE
  POINT(1)=CABTOP(I,1)
  POINT(2)=CABTOP(I,2)
ENDIF
ELSEIF (POINT(1).EQ.SWITYP) THEN
  I=SWITCH(POINT(2),2)
  I=SWITCH(POINT(2),1)
  IF (CABTOP(I,1).EQ.POINT(1) .AND. CABTOP(I,2).EQ.POINT(2)) THEN
    POINT(1)=CABTOP(I,3)
    POINT(2)=CABTOP(I,4)
  ELSE
    POINT(1)=CABTOP(I,1)
    POINT(2)=CABTOP(I,2)
  ENDIF
ELSEIF (POINT(1).EQ.FEETYP) THEN
  I=SFED(POINT(2),2)
  IF (CABTOP(I,1).EQ.FEETYP .AND. CABTOP(I,2).EQ.POINT(2)) THEN
    POINT(1)=CABTOP(I,3)
    POINT(2)=CABTOP(I,4)
  ELSE
    POINT(1)=CABTOP(I,1)
    POINT(2)=CABTOP(I,2)
  ENDIF
ELSE
  WRITE(*,*) ’ WRONG TYPE IN (SUBROUTINE STEPUP)’
ENDIF
IF (WR) THEN
  WRITE(*,*) ’ POINT OUT FROM STEPUP IS’, POINT(1), POINT(2)
ENDIF
RETURN
END

SUBROUTINE STEPDO(POINT,BRANMR,WR)
C
C CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C AUTHER MAGNUS BERGSTRAND
C
C VERSION 1
C
C DATE 1989-10-07
C
C PURPOSE
C TAKES ONE STEP DOWN
C
C DESCRIPTION
C THIS PROCEDURE TAKES A STEP IN THE DIRECTION "NOT DOWN"
C IT WORKS FOR ALL TYPES INCLUDING CABLES,
C WHEN POINT IS A BRANCH POINT STEPDO TAKES A STEP DOWN IN THE
C BRANCH POINTED OUT BY 'BRANMR'
C NOTICE THAT NO CHECK IS MADE THAT THIS BRANCH LEADS DOWNWARDS
C VARIABLES
C POINT IS THE CURRENT POINT (TYPE, NR)
C BRANMR IS THE NUMBER OF THE DOWNWARDS LEADING BRANCH THAT WILL BE VISITED
C IF NR=TRUE THEN THE CURRENT POINT WILL BE WRITTEN ON THE SCREEN
C
C CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

INCLUDE 'CONSTANTS.FOR'
INCLUDE 'COMMON.FOR'

INTEGER BRANMR, POINT(2)
LOGICAL WR
INTEGER I, J, K, L,
IF (NR) THEN
  WRITE(*,*)' POINT IN IS ',POINT(1), POINT(2)
ENDIF

IF (POINT(1).EQ.FEETYP) THEN
  I=HSFEED(POINT(2), 3)
  IF (CABTOP(I,1).EQ.FEETYP .AND. CABTOP(I,2).EQ.POINT(2)) THEN
    POINT(1)=CABTOP(I,3)
    POINT(2)=CABTOP(I,4)
  ELSEIF (CABTOP(I,3).EQ.FEETYP .AND. CABTOP(I,4).EQ.POINT(2)) THEN
    POINT(1)=CABTOP(I,1)
    POINT(2)=CABTOP(I,2)
  ELSE
    WRITE(*,*)' WRONG IN STEPDO.FEEDER'
    STOP
  ENDIF
ENDIF

ELSEIF (POINT(1).EQ.BRATYP) THEN
  I=BRAPOI(POINT(2), BRANMR)
  IF (CABTOP(I,1).EQ.BRATYP .AND. CABTOP(I,2).EQ.POINT(2)) THEN
    POINT(1)=CABTOP(I,3)
    POINT(2)=CABTOP(I,4)
  ELSEIF (CABTOP(I,3).EQ.BRATYP .AND. CABTOP(I,4).EQ.POINT(2)) THEN
    POINT(1)=CABTOP(I,1)
    POINT(2)=CABTOP(I,2)
  ELSE
    WRITE(*,*)' WRONG IN STEPDO.BRA'
    STOP
  ENDIF
ENDIF

ELSEIF (POINT(1).EQ.TRATYP) THEN
  I=TRAEOF(POINT(2))
  IF (CABTOP(I,1).EQ.TRATYP .AND. CABTOP(I,2).EQ.POINT(2)) THEN
    POINT(1)=CABTOP(I,3)
    POINT(2)=CABTOP(I,4)
  ELSEIF (CABTOP(I,3).EQ.TRATYP .AND. CABTOP(I,4).EQ.POINT(2)) THEN
    POINT(1)=CABTOP(I,1)
    POINT(2)=CABTOP(I,2)
  ELSE
    WRITE(*,*)' WRONG INSTEPDO.TRAFO'
    WRITE(*,*)' TRAFO', Point(2)
    WRITE(*,*)CABTOP(I,1), CABTOP(I,2), CABTOP(I,3), CABTOP(I,4)
    STOP
  ENDIF
ELSEIF (POINT(1).EQ.CABTYP) THEN
  THE USED METHOD FOR CABLES
C
LOOK AT ONE NEIGHBOUR
IF CABLE UP IS IDENTICAL WITH THE CURRENT CABLE GO TO THAT NEIGHBOUR
OTHERWISE GO TO THE OTHER NEIGHBOUR

K=POINT(2)
IF (CABTOP(K,1).EQ.SWITYP .OR. CABTOP(K,3).EQ.SWITYP) THEN
  IF (CABTOP(POINT(2),1).EQ.SWITYP) THEN
    I=1
  ELSE
    I=3
  ENDIF
J=MOD((I+2),4)
IF (CABTOP(POINT(2),J).EQ.TRATYP) THEN
  POINT(1)=CABTOP(POINT(2),1)
  POINT(2)=CABTOP(POINT(2),I+1)
ELSEIF (CABTOP(POINT(2),J).EQ.FEETYYP) THEN
  K=CABTOP(POINT(2),J+1)
  IF (POINT(2).EQ.MSFEED(K,2)) THEN
    POINT(1)=FEETYYP
    POINT(2)=CABTOP(POINT(2),J+1)
  ELSE
    POINT(1)=SWITYYP
    POINT(2)=CABTOP(POINT(2),I+1)
  ENDIF
ELSEIF (CABTOP(POINT(2),J).EQ.BRATYP) THEN
  K=CABTOP(POINT(2),J+1)
  L=BRAPAI(K,2)
  IF (POINT(2).EQ.BRAPAI(K,L)) THEN
    POINT(1)=BRATYP
    POINT(2)=CABTOP(POINT(2),J+1)
  ELSE
    POINT(1)=SWITYYP
    POINT(2)=CABTOP(POINT(2),I+1)
  ENDIF
ELSEIF (CABTOP(POINT(2),J).EQ.NODITYP) THEN
  K=CABTOP(POINT(2),J+1)
  L=NODETO(K,1)
  IF (POINT(2).EQ.NODETO(K,L)) THEN
    POINT(1)=NODITYP
    POINT(2)=CABTOP(POINT(2),J+1)
  ELSE
    POINT(1)=SWITYYP
    POINT(2)=CABTOP(POINT(2),I+1)
  ENDIF
ENDIF
ELSEIF (CABTOP(POINT(2),1).EQ.TRATYP) THEN
  POINT(1)=CABTOP(POINT(2),3)
  POINT(2)=CABTOP(POINT(2),4)
ELSEIF (CABTOP(POINT(2),1).EQ.FEETYYP) THEN
  K=CABTOP(POINT(2),2)
  IF (POINT(2).EQ.MSFEED(K,2)) THEN
    POINT(1)=FEETYYP
    POINT(2)=CABTOP(POINT(2),2)
  ELSE
    POINT(1)=CABTOP(POINT(2),3)
    POINT(2)=CABTOP(POINT(2),4)
  ENDIF
ELSEIF (CABTOP(POINT(2),1).EQ.CABTYYP) THEN
  WRITE(*,*) '   DEGENERATED NETWORK'
  STOP
ELSEIF (CABTOP(POINT(2),1).EQ.BRATYP) THEN
  K=CABTOP(POINT(2),2)
  L=BRAPAI(K,2)
  IF (POINT(2).EQ.BRAPAI(K,L)) THEN
    POINT(1)=BRATYP
    POINT(2)=CABTOP(POINT(2),2)
ELSE
    POINT(1) = CABTOP(POINT(2), 3)
    POINT(2) = CABTOP(POINT(2), 4)
ENDIF
ELSEIF (CABTOP(POINT(2), 1).EQ. NODTYPE) THEN
    K = CABTOP(POINT(2), 2)
    L = NODETO(K, 1)
    IF (POINT(2).EQ. NODETO(K, L)) THEN
        POINT(1) = NODTYPE
        POINT(2) = CABTOP(POINT(2), 2)
    ELSE
        POINT(1) = CABTOP(POINT(2), 3)
        POINT(2) = CABTOP(POINT(2), 4)
    ENDIF
ENDIF

HERE THE SECTION CONCERNING CABLES ENDS

ELSEIF (POINT(1), EQ. NODTYPE) THEN
    IF (NODETO(POINT(2), 1).EQ. 2) THEN
        I = 3
    ELSEIF (NODETO(POINT(2), 1).EQ. 3) THEN
        I = 2
    ELSE
        WRITE(*,*) ' WRONG IN STEPD0.NODE'
        STOP
ENDIF

J = NODETO(POINT(2), 1)
IF (CABTOP(J, 1), EQ. NODTYPE .AND. POINT(2), EQ. CABTOP(J, 2)) THEN
    POINT(1) = CABTOP(J, 3)
    POINT(2) = CABTOP(J, 4)
ELSEIF (CABTOP(J, 3), EQ. NODTYPE .AND. POINT(2), EQ. CABTOP(J, 4)) THEN
    POINT(1) = CABTOP(J, 1)
    POINT(2) = CABTOP(J, 2)
ELSE
    WRITE(*,*) ' WRONG IN STEPD0.NODE'
    STOP
ENDIF
ELSEIF (POINT(1), EQ. SWITYP) THEN
    IF (SWITCH(POINT(2), 1), EQ. CLOSED) THEN
        IF (SWITCH(POINT(2), 2), EQ. 3) THEN
            I = 4
        ELSE
            I = 3
        ENDIF
        J = SWITCH(POINT(2), 1)
        IF (CABTOP(J, 1), EQ. SWITYP .AND. CABTOP(J, 2), EQ. POINT(2)) THEN
            POINT(1) = CABTOP(J, 3)
            POINT(2) = CABTOP(J, 4)
        ELSEIF (CABTOP(J, 3), EQ. 6 .AND. CABTOP(J, 4), EQ. POINT(2)) THEN
            POINT(1) = CABTOP(J, 1)
            POINT(2) = CABTOP(J, 2)
        ELSE
            WRITE(*,*) ' WRONG IN STEPD0.SWITCH'
            WRITE(*,*) 'SWITYP,POINT(2)
            WRITE(*,*) CABTOP(J, 1), CABTOP(J, 2), CABTOP(J, 3), CABTOP(J, 4)
            STOP
        ENDIF
    ENDIF
ENDIF

IF (HR) THEN
    WRITE(*,*) 'POINT OUT FROM STEP DOWN IS', POINT(1), POINT(2)
ENDIF
RETURN
END
SUBROUTINE MOVE(POINT,OLD,DIR,BRA,WR)

AUTHOR MAGNUS BERGSTRAND

VERSION 1

DATE 1989-10-07

PURPOSE
TAKES ONE STEP UP OR DOWN. USED IN COOPERATION WITH NEWDIR IT CAN BE
USED TO TRAVERSE A TREE.

DESCRIPTION
DIR INDICATES THE DIRECTION (UP,DOWN)
IF DIRECTION IS DOWN AND POINT IS BRANCH POINT THE VARIABLE BRA IS VALID.
IT INDICATES THE BRANCH THAT WILL BE 'VISITED'
THIS SUBROUTINE WORKS IF THE NETWORK TOPOLOGY IS ALREADY INITIATED.
NOTICE THAT THIS PROCEDURE ASSIGN A NEW VALUE TO 'CABLE UP' IF A
CLOSED SWITCH IS FOUND. UP WILL BE ASSIGNED THE DIRECTION TOWARDS OLD.

SEE ALSO
FILES ROUTINES COMMENTS
MOVING STEPUP TAKES ONE STEP UP
MOVING STEPDO TAKES ONE STEP DOWN
MOVING DIREC COMPUTES NEW DIRECTION
CONSTANT ALL CONSTANTS DECLARED
COMMON ALL COMMON VARIABLES DECLARED
TESTS TKESTP TAKES ONE STEP UP OR DOWN. WORKS IN AN UNINITIATED
NETWORK.

VARIABLES
POINT IS THE CURRENT POINT (TYPE,NR)
OLD IS THE FORMER POINT (TYPE,NR)
DIR INDICATES THE DIRECTION (UP,DOWN)
BRA IS THE NUMBER OF THE BRANCH
IF WR=TRUE THEN THE CURRENT POINT WILL BE WRITTEN ON THE SCREEN

INCLUDE 'CONSTANTS.FOR'
INCLUDE 'COMMON.FOR'

INTEGER POINT(2),OLD(2)
INTEGER DIR,BRA
INTEGER TYP,NR,P(2)
LOGICAL WR

TYP=POINT(1)
NR=POINT(2)
OLD(1)=POINT(1)
OLD(2)=POINT(2)
IF (DIR.EQ.DOWN) THEN
  IF (TYP.EQ.SWITYP) THEN
    IF (SWITCH(NR,1).EQ.CLOSED) THEN

IT IS ONLY POSSIBLE TO TAKE ONE STEP DOWN IF THE SWITCH IS CLOSED

CALL STEPDO(POINT,BRA,WR)
IF (POINT(1).EQ.SWITYP) THEN
  IF (SWITCH(POINT(2),1).EQ.OPEN) THEN

THE RESULT FROM THIS STEP WAS AN OPEN SWITCH
IN THIS SWITCH ASSIGN UP A NEW VALUE TOWARDS OLD.

I=SWITCH(POINT(2),3)
IF (CABTOP(1,1).EQ.OLD(1) .AND. CABTOP(1,2).EQ.OLD(2)) THEN
SWITCH(POINT(2),2)=3
ELSEIF (CABTOP(1,3).EQ.OLD(1) .AND. CABTOP(1,4).EQ.OLD(2)) THEN
SWITCH(POINT(2),2)=3
ELSE
SWITCH(POINT(2),2)=4
ENDIF
ENDIF
ENDIF
ELSE
CALL STEPDO(POINT,BRA,WR)
IF (POINT(1).EQ.SWITYP) THEN
IF(SWITCH(POINT(2),1).EQ.OPEN) THEN

THE RESULT FROM THIS STEP WAS AN OPEN SWITCH
IN THIS SWITCH ASSIGN UP A NEW VALUE TOWARDS OLD.

I=SWITCH(POINT(2),3)
IF (CABTOP(1,1).EQ.OLD(1) .AND. CABTOP(1,2).EQ.OLD(2)) THEN
SWITCH(POINT(2),2)=3
ELSEIF (CABTOP(1,3).EQ.OLD(1) .AND. CABTOP(1,4).EQ.OLD(2)) THEN
SWITCH(POINT(2),2)=3
ELSE
SWITCH(POINT(2),2)=4
ENDIF
ENDIF
ENDIF
ELSEIF (DIR.EQ.UP) THEN
CALL STEPUP(POINT,WR)
ENDIF
RETURN
END

SUBROUTINE DIREC(POINT,OLDP,DIR,BRA)

AUTHOR MAGNUS BERGSTRAND

VERSION 1

DATE 1989-10-07

PURPOSE
COMPUTES A NEW DIRECTION

DESCRIPTION
IS INTENDED TO COOPERATE WITH MOVE. THEN IT CAN BE USED TO TRAVERSE
A TREE WITH EITHER AN OPEN SWITCH OR A 'LEAF NODE' (ONLY ONE FEEDING
CABLE ATTACHED) AS LEAFS.

SEE ALSO
FILES ROUTINES COMMENTS
MOVING STEPUP TAKES ONE STEP UP
MOVING STEPDO TAKES ONE STEP DOWN
MOVING MOVE TAKES ONE STEP UP OR DOWN
TESTS NEWDIR COMPUTES A NEW DIRECTION IN A NON INITIATED NETWORK
CONSTANT ALL CONSTANTS DECLARED
COMMON ALL COMMON VARIABLES DECLARED
VARIABLES
POINT IS THE CURRENT POINT (TYPE,NR)
OLDP IS THE FORMER POINT (TYPE,NR)
DIR IS THE DIRECTION
BRA IS THE NUMBER OF THE BRANCH

INCLUDE 'CONSTANTS.FOR'
INCLUDE 'COMMON.FOR'

INTEGER POINT(2),OLDP(2)
INTEGER DIR,BRA,NXTBRA
INTEGER I,0
LOGICAL READY

IF (DIR.EQ.DOWN) THEN
  IF (POINT(1).EQ.NODTYP) THEN
    IF (NODETO(POINT(2),2).EQ.0 .OR. NODETO(POINT(2),3).EQ.0) THEN
      IF THE NODE IS A LEAF NODE CHANGE DIRECTIONS!
      DIR=UP
      ENDIF
  ELSEIF (POINT(1).EQ.SWITYP) THEN
    IF (SWITCH(POINT(2),1).EQ.OPEN) THEN
      IF AN OPEN SWITCH IS FOUND CHANGE DIRECTION!
      DIR=UP
      ENDIF
  ELSEIF (POINT(1).EQ.BRATYP) THEN
    IF CABLE UP=4 THEN BRA=4 ELSE BRA=3
    IF (BRAPOI(POINT(2),2).EQ.3) THEN
      BRA=4
    ELSE
      BRA=3
    ENDIF
  ENDIF
ELSEIF (DIR.EQ.UP) THEN
  IN THIS CASE THE ONLY TIME WHEN THE DIRECTION IS CHANGED IS WHEN
  A BRANCH POINT IS REACHED.
  THEN THE NEXT BRANCH VISITED WILL BE BRA+1 IF THIS BRANCH EXISTS
  AND IF IT IS NOT THE BRANCH UP

  IF (POINT(1).EQ.BRATYP) THEN
    BRA=2
    I=POINT(2)
    READY=.FALSE.
    10 IF (.NOT. READY) THEN
      BRA=BRA+1
      C=BRAPOI(I,BRA)
      IF (CABTOP(C,1).EQ.OLDP(1) .AND. CABTOP(C,2).EQ.OLDP(2)) THEN
        READY=.TRUE.
      ELSEIF (CABTOP(C,3).EQ.OLDP(1) .AND. CABTOP(C,4).EQ.OLDP(2)) THEN
        READY=.TRUE.
      ENDIF
      GOTO 10
    ENDIF
  ENDIF
  NXTBRA=BRA+1
  IF (NXTBRA.LE.(2+BRAPOI(I,1)) .AND. NXTBRA.NE.BRAPOI(I,2)) THEN
    DIR=DOWN
    BRA=NXTBRA
ELSEIF (NXTBRA.LT.BRAPOI(1,1)+2 .AND. (NXTBRA+1).NE.BRAPOI(1,2)) THEN
   BRA=BRA+2
   DIR=DOWN
ENDIF
ENDIF
RETURN
END
SUBROUTINE CABFOR(J)

AUTHOR MAGNUS BERGSTRAND

VERSION 1

DATE 1989-10-06

PURPOSE

PRODUCES A MENU

SEE ALSO

FILES ROUTINES COMMENTS

CONSTANTS ALL CONSTANTS DECLARED

COMMON ALL COMMON VARIABLES DECLARED

TESTS TRTEST TEST OF TOPOLOGY (TRANSFORMERS)

TESTS FETEST TEST OF TOPOLOGY (FEEDERS)

TESTS BRTEST TEST OF TOPOLOGY (BRANCH POINTS)

TESTS NTTEST TEST OF TOPOLOGY (NODES)

TESTS SWTEST TEST OF TOPOLOGY (SWITCHES)

THE ABOVE CALLS CABFOR

INCLUDE 'CONSTANTS.FOR'

INCLUDE 'COMMON.FOR'

INTEGER J

WRITE(*,200) 'CABLE NR',J,' IS CONNECTED TO '
WRITE(*,300) 'TYPE=',CABTOP(J,1),', NR=',CABTOP(J,2)
WRITE(*,300) ' AND '
WRITE(*,300) 'TYPE=',CABTOP(J,3),', NR=',CABTOP(J,4)
200 FORMAT(' ',TR6,A,I5,A)
300 FORMAT(' ',TR6,A,I2,A,I6)

RETURN
END

SUBROUTINE WRICAB(INDEX)

AUTHOR MAGNUS BERGSTRAND

VERSION 1

DATE 1989-10-06

PURPOSE

PRODUCES A MENU

SEE ALSO

FILES ROUTINES COMMENTS
CONSTATS
COMMOM
TESTS TRTEST
TESTS FETEST
TESTS BTTEST
TESTS NOSTEST
TESTS SWTTEST
THE ABOVE CALLS WRICAB

INCLUDE 'CONSTANTS.FOR'
INCLUDE 'COMMON.FOR'

INTEGER INDEX

WRITE(*,'(A,A)')' CABLE IS CONNECTED TO: TYPE? NR?'
READ(*,*) CABTOP(INDEX,1),CABTOP(INDEX,2)
WRITE(*,'(A,A)')' AND TO: TYPE? NR?'
READ(*,*) CABTOP(INDEX,3),CABTOP(INDEX,4)
WRITE(*,'(A,A)')' R? X? IMAX?'
READ(*,*) CABPHY(J,1),CABPHY(J,2),CABPHY(J,3)
RETURN
END

SUBROUTINE TRTEST(OK)

AUTHOR MAGNUS BERGSTRAND

VERSION 1

DATE 1987-10-06

PURPOSE
TESTS THE TOPOLOGY REGARDING TRANSFORMERS

DESCRIPTION
IF A POINTS NEIGHBOURS NEIGHBOUR IS NOT THE POINT ITSELF
A FAULT IN THE TOPOLOGY IS DETECTED

SEE ALSO
FILES ROUTINES COMMENTS
CONSTANTS ALL CONSTANTS DECLARED
COMMOM ALL COMMON VARIABLES DECLARED
TESTS TRTEST TEST OF TOPOLOGY (TRANSFORMERS)
TESTS FETEST TEST OF TOPOLOGY (FEEDERS)
TESTS BTTEST TEST OF TOPOLOGY (BRANCH POINTS)
TESTS NOSTEST TEST OF TOPOLOGY (NODES)
TESTS SWTTEST TEST OF TOPOLOGY (SWITCHES)

CALL LINES(3)
DO 10 I=1,MAXNRT

EXTERNAL LINES
J=TRANEI(I)
K=1

20 IF (CABTOP(J,1).NE.TRATYP .AND. CABTOP(J,2).NE.K) THEN
   IF (CABTOP(J,3).NE.TRATYP .AND. CABTOP(J,4).NE.K) THEN
      WRITE(*,*)' FAULT IN NETWORK'
      CALL LINES(2)
      OK=FALSE.
      WRITE(*,100)' TRAFO NR' ,I,' IS CONNECTED TO CABLE NR',J
      CALL CABFOR(J)
      WRITE(*,*)' WHICH ONE IS THE WRONG ONE? TYPE?'
      READ(*,*) TYPE
      IF (TYPE.EQ.TRATYP) THEN
         WRITE(*,**) ' TRAFO CONNECTED TO WHICH CABLE?'
         READ(*,**) TRANEI(I)
      ELSEIF (TYPE.EQ.CABTYP) THEN
         CALL WRICAB(J)
      ELSE
         WRITE(*,**) ' WRONG TYPE'
         GOTO 30
      ENDIF
      GOTO 20
   ENDIF
   GOTO 20
ENDIF

10 CONTINUE
100 FORMAT(‘,TRA,A,I3,A,I5)
WRITE(*,*)' TRAFOS IS OK'
RETURN
END

SUBROUTINE FETEST(OK)

C

C CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C

C AUTHOR MAGNUS BERGSTRAND

C

C VERSION 1

C

C DATE 1987-10-06

C

C PURPOSE

C TESTS THE TOPOLOGY REGARDING FEEDERS

C

C DESCRIPTION

C IF A POINTS NEIGHBOURS NEIGHBOUR IS NOT THE POINT ITSELF
C A FAULT IN THE TOPOLOGY IS DETECTED

C

C SEE ALSO

C FILES ROUTINES COMMENTS
C CONSTANTS ALL CONSTANTS DECLARED
C COMMON ALL COMMON VARIABLES DECLARED
C TESTS TRTEST TEST OF TOPOLOGY (TRANSFORMERS)
C TESTS FETEST TEST OF TOPOLOGY (FEEDERS)
C TESTS BRTEST TEST OF TOPOLOGY (BRANCH POINTS)
C TESTS NOTEST TEST OF TOPOLOGY (NODERS)
C TESTS SUTEST TEST OF TOPOLOGY (SWITCHES)
C TESTS NETTES TEST OF NETWORK TOPOLOGY

C CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C

C INCLUDE 'CONSTANTS.FOR'
C INCLUDE 'COMMON.FOR'

C

LOGICAL OK
INTEGER I,J,K,TY1,TY2,NR1,NR2,TYPE
EXTERNAL LINES

DO 10 I=1,MAXNRF
   DO 20 J=2,3
      K=MSFEED(I,J)
      IF (CABTOP(K,1).NE.FEETYP .AND. CABTOP(K,2).NE.1) THEN
         IF (CABTOP(K,3).NE.FEETYP .AND. CABTOP(K,4).NE.1) THEN
            TY1=CABTOP(K,1)
            TY2=CABTOP(K,3)
            NR1=CABTOP(K,2)
            NR2=CABTOP(K,4)
            OK=.FALSE.
            CALL LINES(3)
            WRITE(*,*)' FAULTED NETWORK'
            WRITE(*,200)'FEEDER NR',I,' IS CONNECTED TO CABLE NR',K
            WRITE(*,*)
            CALL CABFOR(K)
            WRITE(*,*)
            WRITE(*,*)' FAULT IN (TYPE) ?'
            READ(*,*) TYPE
            IF (TYPE.EQ.FEETYP) THEN
               WRITE(*,*)' TO WHICH CABLE IS THE FEEDER CONNECTED?'
               READ(*,*) MSFEED(I,J)
            ELSEIF (TYPE.EQ.CABTOP) THEN
               CALL WRICAB(K)
            ELSE
               WRITE(*,*)' WRONG TYPE!'
               GOTO 40
            ENDIF
            GOTO 30
         ENDIF
      ENDIF
   GOTO 10
20 CONTINUE
10 CONTINUE
200 FORMAT(' ',TR6,A,3,A,4)
      WRITE(*,*)' FEEDER IS OK'
      RETURN
      END

SUBROUTINE BRTEST(OK)

C
    CCCC CCC CCC CCC CCC CCC CCC CCC CCC CCC CCC CCC CCC CCC CCC CCC CCC
C
    AUTHOR MAGNUS BERGSTRAND
C
    VERSION 1
C
    DATE 1989-10-06
C
    PURPOSE
    C
    TESTS THE TOPOLOGY REGARDING BRANCH POINTS
C
    DESCRIPTION
    C
    IF A POINTS NEIGHBOURS NEIGHBOUR IS NOT THE POINT ITSELF
    C
    A FAULT IN THE TOPOLOGY IS DETECTED
C
    SEE ALSO
C
    FILES ROUTINES COMMENTS
C
    CONSTANT ALL CONSTANTS DECLARED
C
    COMMON ALL COMMON VARIABLES DECLARED
C
    TESTS TRTEST TEST OF TOPOLOGY (TRANSFORMERS)
INCLUDE 'CONSTANTS.FOR'
INCLUDE 'COMMON.FOR'

LOGICAL OK
INTEGER I,J,K,L,TYPE

EXTERNAL LINES

DO 10 I=1,MAXMRB
  DO 20 J=3,(BRAPOI(I,1)+2)
    K=BRAPOI(I,J)
    IF (CABTOP(K,1).NE.BRATYP .AND. CABTOP(K,2).NE.I) THEN
      IF (CABTOP(K,3).NE.BRATYP .AND. CABTOP(K,4).NE.I) THEN
        OK=.FALSE.
        CALL LINES(3)
        WRITE(*,*)' FAULTED NETWORK'
        WRITE(*,200)' BRANCH-POINT NR',I,' IS CONNECTED TO CABLE NR',K
        WRITE(*,*)
        CALL CABFOR(K)
        WRITE(*,*)' FAULT IN BRANCH-POINT(4) OR IN CABLE(3) ?'
        READ(*,*) TYPE
        IF (TYPE.EQ.BRATYP) THEN
          WRITE(*,*)' THE FOLLOWING BRANCHES'
          WRITE(*,*)
          DO 50 L=3,(2+BRAPOI(I,1))
            WRITE(*,*)',BRAPOI(I,L)
          CONTINUE
          WRITE(*,*)
          WRITE(*,*)' THE FAULTING ONE IS (NOT) CONNECTED TO',BRAPOI(I,J)
          WRITE(*,*)' THE BRANCH IS CONNECTED TO CABLE NR ?'
          READ(*,*) BRAPOI(I,J)
        ELSEIF(TYPE.EQ.CABTP) THEN
          CALL WRICAB(K)
          ELSE
            WRITE(*,*)' WRONG TYPE!'!
          ENDIF
        ENDIF
      ENDIF
  ENDIF
  CONTINUE
  20 CONTINUE
  10 CONTINUE
  200 FORMAT(' TR6,A13,A14')
WRITE(*,*)' BRANCHPOINTS OK'
RETURN
END

SUBROUTINE NOTEST(OK)

AUTHOUR MAGNUS BERGSTRAND

VERSION 1
DATE 1989-10-06

PURPOSE
TESTS THE TOPOLOGY REGARDING NODES

DESCRIPTION
IF A POINTS NEIGHBOURS NEIGHBOUR IS NOT THE POINT ITSELF
A FAULT IN THE TOPOLOGY IS DETECTED

SEE ALSO
FILES Routines COMMENTS
CONSTANTS ALL CONSTANTS DECLARED
COMMON ALL COMMON VARIABLES DECLARED
TESTS TRTEST TEST OF TOPOLOGY (TRANSFORMERS)
TESTS FETEST TEST OF TOPOLOGY (FEEDERS)
TESTS BRTEST TEST OF TOPOLOGY (BRANCH POINTS)
TESTS NOTEST TEST OF TOPOLOGY (NODES)
TESTS SUTEST TEST OF TOPOLOGY (SWITCHES)
TESTS NETTES TEST OF NETWORK TOPOLOGY

cccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

INCLUDE 'CONSTANTS.FOR'
INCLUDE 'COMMON.FOR'

LOGICAL OK
INTEGER I,J,K,TYPE

EXTERNAL LINES

DO 10 I=1,MAXRNM
    J=NODETO(I,2)
    K=NODETO(I,3)
    IF (.NOT. (J.EQ.0)) THEN
20       IF (CABTOP(J,1).NE.NODTYP .AND. CABTOP(J,2).NE.1) THEN
            IF (CABTOP(J,3).NE.NODTYP .AND. CABTOP(J,4).NE.1) THEN
                OK=.FALSE.
                CALL LINES(3)
                WRITE(*,*)' FAULTED NETWORK'
                WRITE(*,*)
                WRITE(*,200)'NODE NR',I,'IS CONNECTED TO CABLE NR',J
                WRITE(*,*)
                CALL CABFOR(J)
                WRITE(*,*)
                CALL CABFOR(K)
                WRITE(*,*)
                WRITE(*,*)' FAULT IN NODE (5) OR IN CABLE (3)'
                READ(*,*) TYPE
                IF (TYPE.EQ.NODTYP) THEN
                    WRITE(*,*)' TO WHICH CABLE IS THE NODE CONNECTED?'
                    READ(*,*) NODETO(I,2)
                ELSEIF (TYPE.EQ.CABTYP) THEN
                    CALL WRICAB(J)
                ELSE
                    WRITE(*,*)' WRONG TYPE!'
                    GOTO 30
                ENDIF
            GOTO 20
        ENDIF
    ENDIF
ENDIF

IF (.NOT. (K.EQ.0)) THEN
40       IF (CABTOP(K,1).NE.NODTYP .AND. CABTOP(K,2).NE.1) THEN
                IF (CABTOP(K,3).NE.NODTYP .AND. CABTOP(K,4).NE.1) THEN
                    OK=.FALSE.
                    CALL LINES(3)
WRITE(*,'(A,20)') 'NODE NR I IS CONNECTED TO CABLE NR K'
WRITE(*,*)
CALL CABFOR(J)
WRITE(*,*)
CALL CABFOR(K)
WRITE(*,*)
WRITE(*,'(A)') 'FAULT IN NODE (S) OR IN CABLE (3)'
READ(*,*)
TYPE
IF (TYPE.EQ.NODTYP) THEN
   WRITE(*,'(A)') 'TO WHICH CABLE IS THE NODE CONNECTED?'
   READ(*,*) NODETO(I,J)
ELSEIF (TYPE.EQ.CABTYP) THEN
   CALL WRICAB(K)
ELSE
   WRITE(*,'(A)') 'WRONG TYPE!'
   GOTO 50
ENDIF
GOTO 40
ENDIF
ENDIF
CONTINUE
100 FORMAT('I')
200 FORMAT('I,TR6,A,I4,A,I4')
WRITE(*,*)
RETURN
END

SUBROUTINE SWICAB(A,OK)

C
C CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C AUTHOR MAGNUS BERGSTRAND
C
C VERSION 1
C
C DATE 1989-10-06
C
C PURPOSE
C TESTS THE TOPOLOGY REGARDING 'A HALF SWITCH'
C
C DESCRIPTION
C IF A POINTS NEIGHBOURS NEIGHBOUR IS NOT THE POINT ITSELF
C A FAULT IN THE TOPOLOGY IS DETECTED
C
C SEE ALSO
C FILES ROUTINES COMMENTS
C CONSTANTS ALL CONSTANTS DECLARED
C COMMON ALL COMMON VARIABLES DECLARED
C TESTS TRTEST TEST OF TOPOLOGY (TRANSFORMERS)
C TESTS FETEST TEST OF TOPOLOGY (FEEDERS)
C TESTS BRTEST TEST OF TOPOLOGY (BRANCH POINTS)
C TESTS NGETEST TEST OF TOPOLOGY (NODES)
C TESTS SUTEST TEST OF TOPOLOGY (SWITCHES)
C TESTS NETTEST TEST OF NETWORK TOPOLOGY
C
C CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
INCLUDE 'CONSTANTS.FOR'
INCLUDE 'COMMON.FOR'

INTEGER A,I,J,TYPE
LOGICAL OK

EXTERNAL LINES

DO 10 I=1,MAXNRS
   J=SWITCH(I,A)
20   IF (CABTOP(J,1).NE.SWITYP .AND. CABTOP(J,2).NE.I) THEN
      IF (CABTOP(J,3).NE.SWITYP .AND. CABTOP(J,4).NE.I) THEN
         OK=.FALSE.,
         CALL LINES(3)
      FAULTED NETWORK'
      WRITE(*,*)
      WRITE(*,200) 'SWITCH Nr.' ,I,' IS CONNECTED TO CABLE Nr.' ,J
      WRITE(*,*)
      CALL CABFOR(J)
   ENDIF
      FAULT IN SWITCH (6) OR IN CABLE (3) ?'
      READ(*,*) TYPE
      IF (TYPE.EQ.SWITYP) THEN
         WRITE(*,*) ' TO WHICH CABLE IS THE SWITCH CONNECTED?'
         READ(*,*) SWITCH(I,A)
      ELSEIF (TYPE.EQ.CABTY) THEN
         CALL WRICAB(J)
      ELSE
         WRITE(*,*) ' WRONG TYPE!'
         GOTO 30
      ENDIF
   GOTO 20
10 CONTINUE
GOTO 20
END

SUBROUTINE SWTSTOK
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
AUTHOUR MAGNUS BERGSTRAND
C
VERSION 1
C
DATE 1989-10-06
C
PURPOSE
C
TESTS THE TOPOLOGY REGARDING SWITCHES
C
DESCRIPTION
C
IF A POINTS NEIGHBOURS NEIGHBOUR IS NOT THE POINT ITSELF
C
A FAULT IN THE TOPOLOGY IS DETECTED
C
SEE ALSO
C
FILES ROUTINES COMMENTS
C
CONSTANTS ALL CONSTANTS DECLARED
C
COMMON ALL COMMON VARIABLES DECLARED
C
TESTS TRTST TEST OF TOPOLOGY (TRANSFORMERS)
C
TESTS FETST TEST OF TOPOLOGY (FEEDERS)
C
TESTS NTST TEST OF TOPOLOGY (NODES)
C
TESTS SWTST TEST OF TOPOLOGY (SWITCHES)
C
TESTS NETTST TEST OF NETWORK TOPOLOGY
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

INCLUDE 'CONSTANTS.FOR'
INCLUDE 'COMMON.FOR'

LOGICAL OK

CALL SWICAB(3,OK)
CALL SWICAB(4,OK)
RETURN
END

SUBROUTINE NETTES(CH2)

AUTHOR MAGNUS BERGSTRAND

VERSION 1

DATE 1989-10-06

PURPOSE
TEST THE NETWORK TOPOLOGY

DESCRIPTION
ANY FAULT THAT WILL BE DETECTED WILL BE CORRECTED

SEE ALSO
FILES Routines Comments
CONSTANTS ALL Constants Declared
COMMON ALL COMMON VARIABLES DECLARED
TESTS TRTEST TEST OF TOPOLOGY (TRANSFORMERS)
TESTS TRTEST TEST OF TOPOLOGY (FEEDERS)
TESTS BRTEST TEST OF TOPOLOGY (BRANCH POINTS)
TESTS NOTEST TEST OF TOPOLOGY (NODES)
TESTS SWTETEST TEST OF TOPOLOGY (SWITCHES)

CHARACTER*7 CH2
LOGICAL OK

EXTERNAL LINES

10 IF (.NOT. OK) THEN
     CALL WRITEF(CH2)
     GOTO 10
ENDIF

WRITE(*,*)' THE NETWORK IS CORRECT'
RETURN
END

SUBROUTINE TKETSP(POINT,OLDP,DIR,BRA,WR)
AUTHOUR MAGNUS BERGSTRAND

VERSION 1

DATE 1989-10-06

PURPOSE
CAN BE USED TO TRAVERSE A NETWORK IN COOPERATION WITH NEWDIR

DESCRIPTION
THIS SUBROUTINE IS ONLY TO BE CALLED FROM INIT
TKESTP MANAGES TO TRAVERSE AN UNINITIATED NETWORK
IT INITIATES THE NETWORK AT THE SAME TIME

SEE ALSO
FILES ROUTINES COMMENTS
CONSTANTS ALL CONSTANTS DECLARED
COMMON ALL COMMON VARIABLES DECLARED
MOVING STEPUP TAKES ONE STEP UP
MOVING STEPDO TAKES ONE STEP DOWN
MOVING MOVE TAKES ONE STEP UP OR DOWN (IN AN INITIATED NETWORK)
MOVING DIREC COMPUTES NEW DIRECTION
TESTS NEWDIR COMPUTES NEW DIRECTION. IS INTENDED TO COOPERATE
WITH TKESTP

VARIABLES
POINT IS THE CURRENT POINT. (TYPE,NR)
OLDP IS THE FORMER POINT. (TYPE,NR)
DIR IS THE DIRECTION (UP OR DOWN)
BRA IS THE NUMBER OF BRANCH IN CASE OF A BRANCH POINT
IF WR=TRUE THEN TEXT IS WRITTEN ON THE SCREEN INDICATING THE CURRENT
POINT

INCLUDE 'CONSTANTS.FOR'
INCLUDE 'COMMON.FOR'

INTEGER POINT(2),OLDP(2),DIR,BRA
INTEGER IN,I,J
LOGICAL WR

IF (WR) THEN
  WRITE(*,*)
  WRITE(*,*)' INDATA TKESTP (POINT,OLDP,DIR,BRA)'
  WRITE(*,*)POINT(1),POINT(2),OLDP(1),OLDP(2),DIR,BRA
ENDIF

IF (DIR.EQ.UP) THEN
  OLPD(1)=POINT(1)
  OLPD(2)=POINT(2)
  CALL STEPDOP(POINT,WR)

ONE STEP UP WAS TAKEN

IF (POINT(1).EQ.BRATYP) THEN
  IF

DO 20 J=3,(2+BRAPOL(POINT(2),I))
  I=BRAPOL(POINT(2),J)
  IF (CABTOP(I,1).EQ.OLDP(1) .AND. CABTOP(I,2).EQ.OLDP(2)) THEN
    BRA=J
  ELSEIF (CABTOP(I,3).EQ.OLDP(1) .AND. CABTOP(I,4).EQ.OLDP(2)) THEN
    BRA=J
  ENDIF
  20 CONTINUE

FIND OUT IN WHICH BRANCH OLPD IS.
CONTINUE
ENDIF
ELSEIF (DIR.EQ.DOWN) THEN
IF (POINT(1).EQ.BRATYP) THEN
I=POINT(2)
ENDIF
ELSEIF (BRAPOI(I,2).EQ.0) THEN
C 'UP' IS NOT KNOWN
C UP IS IN THE DIRECTION OF OLDP!

DO 30 J=3,BRAPOI(I,1)+2
   K=BRAPOI(I,J)
   IF (CABTOP(K,1).EQ.OLDP(1) .AND. CABTOP(K,2).EQ.OLDP(2)) THEN
      BRAPOI(I,2)=J
   ELSEIF (CABTOP(K,3).EQ.OLDP(1) .AND. CABTOP(K,4).EQ.OLDP(2)) THEN
      BRAPOI(I,2)=J
   ENDIF
30 CONTINUE
ENDIF
ELSEIF (BRAPOI(I,2).EQ.3) THEN
C SINCE THE DIRECTION IS DOWN
   BRA=4
ELSE
   BRA=3
ENDIF
ENDIF
OLDP(1)=POINT(1)
OLDP(2)=POINT(2)
CALL STEPDO(POINT,BRA,WR)
C ONE STEP DOWN WAS TAKEN
C HEREAFTER UP WILL BESSIGNED A VALUE IN CASE OF SWITCH OR NODE

IF (POINT(1).EQ.BWITYP) THEN
   J=POINT(2)
   I=SWITCH(J,3)
   IF (CABTOP(I,1).EQ.OLDP(1) .AND. CABTOP(I,2).EQ.OLDP(2)) THEN
      SWITCH(J,2)=3
   ELSEIF (CABTOP(I,3).EQ.OLDP(1) .AND. CABTOP(I,4).EQ.OLDP(2)) THEN
      SWITCH(J,2)=3
   ELSE
      SWITCH(J,2)=4
   ENDIF
ELSEIF (POINT(1).EQ.NOQTYPO) THEN
   I=NODETO(POINT(2),2)
   IF (CABTOP(I,1).EQ.OLDP(1) .AND. CABTOP(I,2).EQ.OLDP(2)) THEN
      NODETO(POINT(2),1)=2
   ELSEIF (CABTOP(I,3).EQ.OLDP(1) .AND. CABTOP(I,4).EQ.OLDP(2)) THEN
      NODETO(POINT(2),1)=2
   ELSE
      NODETO(POINT(2),1)=3
   ENDIF
ENDIF
ENDIF
IF (WR) THEN
   WRITE(*,*)
   WRITE(*,*)' OUTDATA TKESTP'
   WRITE(*,*)POINT(1),POINT(2),OLDP(1),OLDP(2),DIR,BRA
ENDIF
RETURN
END

SUBROUTINE NEWDIR(POINT,DIR,BRA,WR)
AUTHOR MAGNUS BERGSTRAND

VERSION 1

DATE 1987-10-06

PURPOSE
COMPUTES THE NEW DIRECTION.

DESCRIPTION
IS USED IN COOPERATION WITH TKESTP IN ORDER TO TRAVERSE AN
UNINITIALIZED NETWORK

SEE ALSO
FILES ROUTINES COMMENTS
CONSTANTS ALL CONSTANTS DECLARED
COMMON ALL COMMON VARIABLES DECLARED
MOVING STEPUP TAKES ONE STEP UP
MOVING STEPD0 TAKES ONE STEP DOWN
MOVING MOVE TAKES ONE STEP UP OR DOWN (IN AN INITIATED NETWORK)
MOVING DIREC COMPUTES NEW DIRECTION
TESTS TKESTP TAKES ONE STEP UP OR DOWN. IS INTENDED TO COOPERATE
WITH NEWDIR

VARIABLES
POINT IS THE CURRENT POINT. (TYPE,NR)
DIR IS THE DIRECTION (UP OR DOWN)
BRA IS THE NUMBER OF BRANCH IN CASE OF A BRANCH POINT
IF WR=TRUE THEN TEXT IS WRITTEN ON THE SCREEN INDICATING THE CURRENT
POINT

INCLUDE 'CONSTANTS.FOR'
INCLUDE 'COMMON.FOR'

INTEGER POINT(2)
INTEGER DIR,BRA
INTEGER 1
LOGICAL WR

IF (WR) THEN
  WRITE(*,*)
  WRITE(*,*)' INDATA NEWDIR (POINT,DIR,BRA)'
  WRITE(*,*) POINT(1),POINT(2),DIR,BRA
ENDIF

IF (POINT(1).EQ.BRATYP) THEN
  IF (DIR.EQ.UP) THEN
    IN THIS CASE THE ONLY POSSIBLE CHANGE OF DIRECTION IS WHENEVER
    A BRANCH POINT IS REACHED. THEN THE NEW DIRECTION MAY BE DOWN
    IN THE NEXT BRANCH
    
    I=POINT(2)
    IF (BRAPOINT(1,1).GT.(BRA-2) .AND. BRAPOINT(1,2).NE.(BRA+1)) THEN
      BRA=BRA+1
      DIR=DOWN
    ELSEIF (BRAPOINT(1,2).EQ.(BRA+1)) THEN
      IF (BRA+2 .LE. 2+BRAPOINT(1,1)) THEN
        BRA=BRA+2
        DIR=DOWN
      ENDIF
    ENDIF
  ENDIF
ENDIF
ELSEIF (POINT(1).EQ.SWITYP) THEN
  C
  THE DIRECTION IS DOWN
  C
  IF AN OPEN SWITCH IS FOUND CHANGE DIRECTION
  IF (SWITCH(POINT(2),1).EQ.OPEN) THEN
    DIR=UP
  ENDF
ELSEIF (POINT(1).EQ.NODYP) THEN
  C
  IF A NODE IS FOUND AND THIS NODE HAS ONLY ONE FEEDING CABLE
  C
  CHANGE DIRECTION
  IF (NODETO(POINT(2),2).EQ.0) THEN
    DIR=UP
  ELSEIF (NODETO(POINT(2),3).EQ.0) THEN
    DIR=UP
  ENDF
ENDIF
ENDIF
IF (MR) THEN
  WRITE(*,*)
  WRITE(*,*)'OUTDATA NENDIR (POINT,DIR,BRA)'
  WRITE(*,*) POINT(1),POINT(2),DIR,BRA
ENDIF
RETURN
END

SUBROUTINE FILLIN(POINT,DIR,BRA,TRA,FEE)
  C
  CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
  C
  AUTHOR MAGNUS BERGSTRAND
  C
  VERSION 1
  C
  DATE 1989-10-06
  C
  PURPOSE
  C
  ASSIGNS FEEDER AND TRANSFORMER IDENTITY.
  C
  COMPUTES IMPEDANCES.
  C
  DESCRIPTION
  C
  IS ONLY TO BE CALLED FROM INIT
  C
  THE IMPEDANCE WILL BE COMPUTED BY A CALL OF UPDATE
  C
  SEE ALSO
  C
  FILES  ROUTINES  COMMENTS
  C
  CONSTANTS  ALL CONSTANTS DECLARED
  C
  COMMON  ALL COMMON VARIABLES DECLARED
  C
  TESTS  INIT  INITIATES THE NETWORK
  C
  TESTS  UPDATE  COMPUTES IMPEDANCE
  C
  MOVING  STEPUP  TAKES ONE STEP UP
  C
  VARIABLES
  C
  POINT IS THE CURRENT POINT (TYPE,NUMBER)
  C
  DIR IS THE DIRECTION (UP OR DOWN)
  C
  BRA IS THE NUMBER OF BRANCH
  C
  TRA IS THE TRANSFORMER IDENTITY
  C
  FEE IS THE FEEDER IDENTITY
  C
  CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
INCLUDE 'CONSTANTS.FOR'
INCLUDE 'COMMON.FOR'

INTEGER POINT(2), P(2)
INTEGER DIR, BRA, TRA, FEE
INTEGER OLD(2)
REAL CUR
LOGICAL FIRST

IF (DIR.EQ.DOWN) THEN
  IF (POINT(1).EQ.PEETYP) THEN
    FEE=POINT(2)
    MSPEED(POINT(2),1)=TRA
  ELSEIF (POINT(1).EQ.SWITYP) THEN
    SWITCH(POINT(2), 5)=1
  ELSEIF (POINT(1).EQ.NODITYP) THEN
    I=POINT(2)
    NODETO(I, 4)=FEE
    CUR=(NODEP(1, 1)**2+NODEP(1, 2)**2)/(NODEP(1, 4)**2+NODEP(1, 5)**2)
    FIRST=.TRUE.
    P(1)=POINT(1)
    P(2)=POINT(2)
    OLD(1)=0
    OLD(2)=0
  ENDIF
IF (P(1).NE.TRATYP) THEN
  CALL UPDATE(P, OLD, FIRST, CUR, NODEP(POINT(2), 4), NODEP(POINT(2), 5))
  OLD(1)=P(1)
  OLD(2)=P(2)
  CALL STEPUP(P, .FALSE.)
GOTO 10
ENDIF
CALL UPDATE(P, OLD, FIRST, CUR, 1.0, 1.0)
ENDIF
RETURN
END

SUBROUTINE UPDATE(POINT, OLD, FIRST, CUR, NODEP, NODEX)

AUTHOUR MAGNUS BERGSTRAND

VERSION 1

DATE 1789-10-06

PURPOSE
COMPUTES IMPEDANCES

DESCRIPTION
COMPUTES IMPEDANCES WHILE MOVING FROM A NODE TO A TRANSFORMER
COMPUTATIONS ARE BASED ON THE FACT THAT THE NETWORK HAS A RADIAL
STRUCTURE.
THE RESULT Z=(RES, j*REA) ARE SAVED BETWEEN CALLS.
THE PARALLELL IMPEDANCE BETWEEN THE OLD VALUE AND A NEW NODE IS COMPUTED
THEN IS THE CABLE IMPEDANCE ADDED

SEE ALSO
FILES ROUTINES COMMENTS
CONSTANTS ALL CONSTANTS DECLARED
COMMON ALL COMMON VARIABLES DECLARED
MOVING STEPUP TAKES ONE STEP UP
COMP COMPIMP COMPUTES IMPEDANCE. INITIATED NETWORK
C COMP BRAPAR COMPUTES EQUIVALENT PARALLEL IMPEDANCE FOR A BRANCH POINT
C COMP PARALL COMPUTES EQUIVALENT PARALLEL IMPEDANCE

VARIABLES
POINT IS THE CURRENT POINT (TYPE, NR)
OLD IS THE FORMER POINT (TYPE, NR)
FIRST INDICATES FIRST CALL.
CUR IS THE CURRENT (AMPERE)
NODER IS NODE RESISTANCE (OHM)
NODEX IS NODE REACTANCE (OHM)

C CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

INCLUDE 'CONSTANTS.FOR'
INCLUDE 'COMMON.FOR'

INTEGER POINT(2), OLD(2)
INTEGER I, J, K
LOGICAL FIRST
REAL NODER, NODEX, CUR, F(2), G(2)
REAL RES, REA, SUMR, SUMX

SAVE RES, REA

IF (FIRST) THEN
  RES=NODEPH(POINT(2), 4)
  REA=NODEPH(POINT(2), 5)
  FIRST=.FALSE.
ELSEIF (.NOT. FIRST) THEN
  IF (POINT(1).EQ.TRATYP) THEN
    TRAFOS(POINT(2), 6)=RES
    TRAFOS(POINT(2), 7)=REA
  ELSEIF (POINT(1).EQ.FEETYP) THEN
    I=MSFEED(POINT(2), 3)
    RES=RES+CABPHY(I, 1)
    REA=REA+CABPHY(I, 2)
  ELSEIF (POINT(1).EQ.BRATYP) THEN
    I=POINT(2)
    DO 30 J=3, (BRAPOI(I, 1)+2)
      K=BRAPOI(I, J)
      IF (CABTOP(K, 1).EQ. OLD(1) .AND. CABTOP(K, 2).EQ. OLD(2)) THEN
        Braph(I, J, 1)=RES
        Braph(I, J, 2)=REA
      ELSEIF (CABTOP(K, 3).EQ. OLD(1) .AND. CABTOP(K, 4).EQ. OLD(2)) THEN
        Braph(I, J, 1)=RES
        Braph(I, J, 2)=REA
      ENDIF
  ENDIF
  30 CONTINUE
  CALL BRAPAR(POINT(2), RES, REA)
  RES=CABPHY(K, 1)+RES
  REA=CABPHY(K, 2)+REA
ELSEIF (POINT(1).EQ.NODTYP) THEN
  F(1)=RES
  F(2)=REA
  G(1)=NODEPH(POINT(2), 4)
  G(2)=NODEPH(POINT(2), 5)
  CALL PARALL(F, G, RES, REA)
  I=NODETO(POINT(2), 1)
  I=NODETO(POINT(2), 1)
  RES=CABPHY(I, 1)+RES
  REA=CABPHY(I, 1)+REA
ELSEIF (POINT(1).EQ.SWITYP) THEN
  I=SWITCH(POINT(2), 2)
SUBROUTINE INIT(WR)

C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
AUTHOR MAGNUS BERGSTRAND
C
VERSION 1
C
DATE 1986-10-06
C
PURPOSE
C INITIATES THE NETWORK
C
DESCRIPTION
C FIRST ALL PARALLELL IMPEDANCES IN BRANCH POINTS ARE ASSIGNED A
C LARGE VALUE
C THEN FOR EACH TRANSFORMER THE TOPOLOGY IS INITIATED AND IMPEDANCES COMPUTED
C BY A CALL OF FILLIN
C
SEE ALSO
C FILES ROUTINES COMMENTS
C CONSTANTS ALL CONSTANTS DECLARED
C COMMON ALL COMMON VARIABLES DECLARED
C TESTS FILLIN INITIATE THE NETWORK AND COMPUTES IMPEDANCES
C TESTS TKSTP TAKES A STEP UP OR DOWN IN A NONINITIATED NETWORK
C TESTS NewDiR COMPUTES NEW DIRECTION IN A NONINITIATED NETWORK
C MOVING STEPDO TAKES ONE STEP DOWN
C
VARIABLES
C IF 'WR'=TRUE THEN TEXT SHOWING THE CURRENT POINT WILL BE WRITTEN
C ON THE SCREEN
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

INCLUDE 'CONSTANTS.FOR'
INCLUDE 'COMMON.FOR'

INTEGER POINT(2),OLDP(2),DIR,BRA
INTEGER TRA,FEE
INTEGER I,J,K
LOGICAL WR

DO 10 I=1,MAXNRB
  BRAPOI(I,2)=0
  DO 20 J=3,(2*BRAP0I(I,1))
    BRAPH(I,J,1)=1.0E+20
    BRAPH(I,J,2)=0.0
  20    CONTINUE
10    CONTINUE
DO 30 I=1,MAXNRS
  CHGSMI(I)=.FALSE.
30    CONTINUE
DO 50 I=1,MAXNRT

TRA=1
TRAFOS(TRA,6)=1.001E+06
TRAFOS(TRA,7)=1.001E+06
POINT(1)=TRATYP
POINT(2)=TRA
OLDP(1)=TRATYP
OLDP(2)=TRA
DIR=DOWN
BRA=4
CALL STEPDO(POINT,BRA,WR)
IF (POINT(1).EQ.SWITYP) THEN
  K=SWITCH(POINT(2),3)
  IF (CABTOP(K,1).EQ.OLDP(1).AND. CABTOP(K,2).EQ.OLDP(2)) THEN
    SWITCH(POINT(2),2)=3
  ELSEIF (CABTOP(K,3).EQ.OLDP(1).AND. CABTOP(K,4).EQ.OLDP(2)) THEN
    SWITCH(POINT(2),2)=3
  ELSE
    SWITCH(POINT(2),2)=4
  ENDIF
ELSE
  WRITE(*,*),' DEGENERATED NET SWITCH EXPECTED'
  STOP
ENDIF
CALL FILLIN(POINT,DIR,BRA,TRA,FEE)
CALL NEWDIR(POINT,DIR,BRA,WR)
60 IF (.NOT. (POINT(1).EQ.TRAYPH .AND. POINT(2).EQ.TRA)) THEN
  CALL TKESTP(POINT,OLDP,DIR,BRA,WR)
  CALL FILLIN(POINT,DIR,BRA,TRA,FEE)
  CALL NEWDIR(POINT,DIR,BRA,WR)
GOTO 60
ENDIF
50 CONTINUE
WRITE(*,*),' INIT PASSED'
RETURN
END
SUBROUTINE CON(POINT,OK)

AUTHOR MAGNUS BERGSTRAND

VERSION 1

DATE 1989-09-30

PURPOSE:
CONNECTS A GROUP OF SECTIONS TO THE FEEDER
WITH THE LARGEST MARGIN OF GETTING VIOLATED

DESCRIPTION:
STARTS A SEARCH FROM THE ROOT OF A TREE AND
EXAMINES ALL POSSIBLE LOAD TRANSFERS, WHEN
THE BEST SWITCH FOR LOAD TRANSFER IS FOUND
THIS SWITCH IS OPENED
IF HOWEVER NO POSSIBLE CONNECTION IS AVAILABLE
OK EQUALS FALSE.

VARIABLES:
POINT IS THE ROOT OF THE TREE (TYPE,NR)

SEE ALSO:

FILE ROUTINE COMMENT

CONSTANTS ALL CONSTANTS DECLARED
COMMON ALL COMMON VARIABLES DECLARED
FETCH GETPOI GETS A POINT ON THE OTHER SIDE OF SWITCH
FETCH GETVOL GETS THE VOLTAGE ON THE OTHER SIDE
FETCH GETFEE GETS THE NUMBER OF THE FEEDER ON THE OTHER
SIDE OF THE SWITCH
FETCH GETPLI GETS THE TRANSFORMERS MARGIN OF GETTING
VIOLATED REGARDING ACTIVE POWER
FETCH GETQLI GETS THE TRANSFORMERS MARGIN OF GETTING
VIOLATED REGARDING REACTIVE POWER
MOVING MOVE TAKES ONE STEP UP OR DOWN
MOVING DIREC COMPUTES NEW DIRECTION
COMP FORBAC CHANGES DIRECTIONS IN THE TOPOLOGY DUE
TO THE LOAD TRANSFER
COMP COMPIM COMPUTES NEW IMPEDANCE FOR A FEEDER
COMP FILPOM COMPUTES NEW POWER FOR A TRANSFORMER
COMP FILVOL COMPUTES ALL NODE VOLTAGES AND CABLE
CURRENTS FOR A FEEDER

INCLUDE 'CONSTANTS.FOR'
INCLUDE 'COMMON.FOR'

INTEGER POINT(2),PL(2),OLD(2)
LOGICAL READY,NEG
INTEGER FEEDER,GETFEE,FEE
INTEGER TRAFO, DIR, BRA, SWI
LOGICAL V10
REAL BEST, TEMP, VOL, P, Q
REAL ABS, GETVOL
REAL GETPLI, GETQLI
LOGICAL FOUND, VAL, OK

EXTERNAL GETP0I, GETV0L, GETFEE
EXTERNAL GETPLI, GETQLI
EXTERNAL MOVE, DIREC

VAL=.FALSE.
P1(1)=POINT(1)
P1(2)=POINT(2)
READY=.FALSE.
DIR=DOWN
IF (P1(1).EQ.BRATYP) THEN
  IF (BRAP0I(P1(1),2).EQ.3) THEN
    BRA=4
  ELSE
    BRA=3
  ENDIF
ENDIF
BEST=-1.0E30
IF (P1(1).EQ.NODTYP) THEN
  IF (NOVETO(P1(2),2).EQ.0 .OR. NOVETO(P1(2),3).EQ.0) THEN
    GOTO 20
  ENDIF
ENDIF
10 IF (.NOT. READY) THEN
  IF (P1(1).EQ.NODTYP .AND. DIR.EQ.DOWN) THEN
C ADDS UP THE MAGNITUDE OF ISOLATED LOAD

  ISOLAT(1)=ISOLAT(1)+NODEPH(P1(2),1)
  ISOLAT(2)=ISOLAT(2)+NODEPH(P1(2),2)
ENDIF
CALL MOVE(P1,OLD,DIR,BRA,.FALSE.)
CALL DIREC(P1,OLD,DIR,BRA)
IF (P1(1).EQ.SWITYP) THEN
  IF (SWITCH(P1(2),1).EQ.OPEN) THEN
    CALL REALFE(P1(2),FEE,FOUND)
  IF (FOUND) THEN
C A FEEDER THAT CAN TAKE UP LOAD IS FOUND
C IT IS POSSIBLE TO MAKE A CONNECTION
C BELOW Follows Computations in ORDER TO
C DECIDE THE BEST CONNECTION

  VAL=.TRUE.
  VOL=GETVOL(P1(2))
  P=GETPLI(P1(2))
  Q=GETQLI(P1(2))
  NEG=(VOL.LT.0.0 .OR. P.LT.0.0)
  NEG=(NEG .OR. Q.LT.0.0)
  IF (NEG) THEN
    TEMP=-(ABS(VOL)+ABS(P)+ABS(Q))
  ELSE
    TEMP=VOL+P+Q
  ENDIF
  IF (TEMP.GT.BEST) THEN
C THE BEST CONNECTION IS STORED FOR LATER USE

    BEST=TEMP
    FEEDER=FEE
SWI=P1(2)
ENDIF
ENDIF
ENDIF
ENDIF
READY=(P1(1).EQ.POINT(1).AND. P1(2).EQ.POINT(2).AND. DIR.EQ.UP)
GOTO 10
ENDIF
OK=VAL
20 IF (VAL) THEN

A CONNECTION IS FOUND AND THE BEST SWITCHING OPERATION IS PERFORMED

SWINR(SWI)=4
SWITCH(SWI,1)=CLOSED
CHGSWI(SWI)=.NOT. CHGSWI(SWI)
TRAFO=MSFEED(FEEDER,1)
CALL FORBAC(SWI,TRAFO,FEEDER)
CALL COMPIM(FEEDER)
CALL FILPON(TRAFO)
CALL FILVOL(TRAFO,VO)
ELSE

NO POSSIBLE CONNECTION EXIST AND NO LOAD WAS RESTORED

WRITE(*,*)' IT IS NOT POSSIBLE TO MAKE ANY CONNECTION'
RESTOR(1)=0.0
RESTOR(2)=0.0
ENDIF
RETURN
END

SUBROUTINE UDAT(SWI,TRAFO,FEEDER)

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

AUTHOR: MAGNUS BERGSTRAND

VERSION 1

DATE 1989-09-30

PURPOSE:
FILLS IN FEEDER AND TRANSFORMER IDENTITY IN PROPER PLACES
WHEN A LOAD TRANSFER HAS CHANGED THE TOPOLOGY

DESCRIPTION:
IS ONLY TO BE CALLED BY SUBROUTINE FORBAC
SIMPLE ASSIGNMENT OF VARIABLES TRAFO AND FEEDER

VARIABLES:
TRAFO, INTEGER, THE NUMBER OF THE 'NEW' TRANSFORMER OF THE FEEDER
FEEDER, INTEGER, THE NUMBER OF THE 'NEW' FEEDER A NODE BELONGS TO

SEE ALSO:

FILE ROUTINE COMMENT
CONSTANTS ALL CONSTANTS DECLARED
COMMON ALL COMMON VARIABLES DECLARED
COMH FORBAC CHANGES THE 'DIRECTIONS' OF THE TOPOLOGY
DUE TO A LOAD TRANSFER
MOVING MOVE TAKES ONE STEP UP OR DOWN
MOVING DIREC COMPUTES NEW DIRECTION

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
INCLUDE 'CONSTANTS.FOR'
INCLUDE 'COMMON.FOR'

INTEGER P(2), OLD(2)
LOGICAL READY
INTEGER DIR,BRA

P(1)=SWITYP
P(2)=SWI
READY=.FALSE.
DIR=DOUN
BRA=3
10 IF (.NOT. READY) THEN
   CALL MOVE(P,OLD,DIR,BRA,.FALSE.)
   CALL DIREC(P,OLD,DIR,BRA)
   IF (DIR.EQ.UP) THEN
      IF (P(1).EQ.NODTYP) THEN
         NODETO(P(2),4)=FEEDER
      ENDIF
      ELSEIF (P(1).EQ.FEETYP) THEN
         MSFEED(P(2),1)=TRAFO
      ENDIF
   ENDIF
   READY=(P(1).EQ.SWITYP .AND. P(2).EQ.SWI)
   GOTO 10
ENDIF
RETURN
END

SUBROUTINE FORBAC(SWI,TRAFO,FEEDER)

AUTHOUR MAGNUS BERGSTRAND

VERSION 1

DATE 1987-09-30

PURPOSE:
CHANGES DIRECTIONS.
WHEN A LOAD TRANSFER IS MADE THE FORMER DIRECTION 'DOWN'
WILL BECOME 'UP'

DESCRIPTION:
STARTING AT THE SWITCH THAT WILL BE OPENED THIS ROUTINE
MOVES TOWARDS THE SWITCH THAT WILL BE CLOSED CHANGING
DIRECTIONS. THE ACTUAL CHANGE IN POINTERS IS DONE BY A
CALL OF SUBROUTINE CHGDIR

VARIABLES:
SWI IS THE NUMBER OF THE SWITCH THAT WILL BE OPENED
TRAFO IS THE NUMBER OF THE 'NEW' TRANSFORMER THAT A
FEEDER POSSIBLY WILL BE ATTACHED TO
FEEDER IS THE NUMBER OF THE 'NEW' FEEDER THAT A NODE
WILL BE ATTACHED TO

SEE ALSO:

FILE ROUTINE COMMENT

CONSTANTS ALL CONSTANTS DECLARED
COMMON ALL COMMON VARIABLES DECLARED
COMP UDAT ASSIGNS NEW VALUES OF TRAFO,FEEDER
COMP CHGDIR CHANGES POINTERS DUE TO A LOAD TRANSFER
MOVING STEPUP TAKES ONE STEP UP
MOVING STEPDO TAKES ONE STEP DOWN
INCLUDE 'CONSTANTS.FOR'
INCLUDE 'COMMON.FOR'

INTEGER TRAFO, FEEDER, I
INTEGER SWI, DIR, BRA
INTEGER POI(2), OLD(2)
LOGICAL READY

POI(1) = SWITYP
POI(2) = SWI
OLD(1) = SWITYP
OLD(2) = SWI
READY = .FALSE.
CALL STEPUP(POI, .FALSE.)
I = SWITCH(SWI, 3)
IF (CABTOP(I, 1), EQ, POI(1), AND, CABTOP(I, 2), EQ, POI(2)) THEN
  SWITCH(SWI, 2) = 4
ELSEIF (CABTOP(I, 3), EQ, POI(1), AND, CABTOP(I, 4), EQ, POI(2)) THEN
  SWITCH(SWI, 2) = 4
ELSE
  SWITCH(SWI, 2) = 3
ENDIF
10 IF (.NOT. READY) THEN
   CALL CHGDIR(POI, OLD, BRA, TRAFO, FEEDER)
   OLD(1) = POI(1)
   OLD(2) = POI(2)
   CALL STEPD0(POI, BRA, .FALSE.)
   IF (POI(1), EQ, SWITYP) THEN
     IF (SWITCH(POI(2), 1), EQ, OPEN) THEN
       READY = .TRUE.
     I = SWITCH(POI(2), 3)
     IF (CABTOP(I, 1), EQ, OLD(1), AND, CABTOP(I, 2), EQ, OLD(2)) THEN
       SWITCH(POI(2), 2) = 3
     ELSEIF (CABTOP(I, 3), EQ, OLD(1), AND, CABTOP(I, 4), EQ, OLD(2)) THEN
       SWITCH(POI(2), 2) = 3
     ELSE
       SWITCH(POI(2), 2) = 4
     ENDIF
     I = SWITCH(POI(2), 2)
   ENDIF
  GOTO 10
ENDIF
GOTO 10
ENDIF
CALL UDAT(SWI, TRAFO, FEEDER)
RETURN
END

SUBROUTINE CHGDIR(POI, OLD, BRA, TRAFO, FEEDER)

AUTHOUR MAGNUS BERGSTRAND

VERSION 1

DATE 1989-09-30

PURPOSE:
WHEN A LOAD TRANSFER IS PERFORMED CHANGES MUST BE MADE IN THE
TOPOLOGY. THIS SUBROUTINE CHANGES POINTER ACCORDING TO THE
NEW TOPOLOGY
DESCRIPTION:
IT IS ONLY TO BE CALLED BY SUBROUTINE FORBAC
STANDING AT THE POINT POI CHANGES IS MADE ACCORDING TO THE TYPE
OF THIS POINT.

VARIABLES:
POI IS THE CURRENT POINT IN THE NETWORK
OLD IS THE PREVIOUSLY VISITED POINT IN THE NETWORK
BRA IS AN OUTPARAMETER, IT INDICATES THE OLD VALUE OF UP.
TRAFO IS THE NUMBER OF THE TRANSFORMER WHICH ALL OTHER
ELEMENTS ARE ATTACHED TO.
FEEDER IS THE NUMBER OF THE FEEDER ENERGIZING THE TRANSFERRED LOAD

SEE ALSO:

FILE ROUTINE COMMENT
CONSTATS ALL CONSTANTS DECLARED
COMMON ALL COMMON VARIABLES DECLARED
COMP FORBAC CHANGES THE NETWORK TOPOLOGY

INCLUDE 'CONSTANTS.FOR'
INCLUDE 'COMMON.FOR'

INTEGER TRAFO, FEEDER, BRA
INTEGER POI(2), OLD(2)
INTEGER I, J

IF (POI(1).EQ.SWITYP) THEN
   I=SWITCH(POI(2),3)
   IF (CABTOP(I,1).EQ.OLD(1) .AND. CABTOP(I,2).EQ.OLD(2)) THEN
      SWITCH(POI(2),2)=3
   ELSEIF (CABTOP(I,3).EQ.OLD(1) .AND. CABTOP(I,4).EQ.OLD(2)) THEN
      SWITCH(POI(2),2)=3
   ELSE
      SWITCH(POI(2),2)=4
   ENDIF
ELSEIF (POI(1).EQ.NOOTYP) THEN
   I=NODETO(POI(2),2)
   IF (CABTOP(I,1).EQ.OLD(1) .AND. CABTOP(I,2).EQ.OLD(2)) THEN
      NODETO(POI(2),1)=2
   ELSEIF (CABTOP(I,3).EQ.OLD(1) .AND. CABTOP(I,4).EQ.OLD(2)) THEN
      NODETO(POI(2),1)=2
   ELSE
      NODETO(POI(2),1)=3
   ENDIF
I=NODETO(POI(2),1)
NODETO(POI(2),4)=FEEDER
ELSEIF (POI(1).EQ.BRATYP) THEN
   BRA=BRAPOL(POI(2),2)
   DO 10 J=3, (2+BRAPOL(POI(2),1))
      I=BRAPOL(POI(2),J)
      IF (CABTOP(I,1).EQ.OLD(1) .AND. CABTOP(I,2).EQ.OLD(2)) THEN
         BRAPOL(POI(2),2)=J
      ELSEIF (CABTOP(I,3).EQ.OLD(1) .AND. CABTOP(I,4).EQ.OLD(2)) THEN
         BRAPOL(POI(2),2)=J
      ENDIF
   CONTINUE
ELSEIF (POI(1).EQ.FEETYP) THEN
   I=MSFEED(POI(2),2)
   MSFEED(POI(2),2)=MSFEED(POI(2),3)
   MSFEED(POI(2),3)=I
   MSFEED(POI(2),1)=TRAFO
ELSE
   WRITE(*,*)'   WRONG TYPE IN CHGOIR',POI(1),POI(2)
   STOP
ENDIF
OLD(1)=POI(1)
OLD(2)=POI(2)
RETURN
END

SUBROUTINE COMPIM(FEEDER)

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

AUTHOR MAGNUS BERGSTRAND

VERSION 1

DATE 1989-09-30

PURPOSE:
COMPUTES THE IMPEDANCE FOR FEEDER 'FEEDER'

METHOD:
COMPUTATIONS IS BASED ON THE FACT THAT THE DISTRIBUTION NETWORK
HAS A RADIAL STRUCTURE. THEREFORE PARALLEL IMPEDANCES WITH CABLE
IMPEDANCES WILL BE COMPUTED. SEE ALSO 'SERVICE RESTORATION IN
ELECTRIC DISTRIBUTION NETWORKS' BY MAGNUS BERGSTRAND (SECTION
COMPUTATIONS)

DESCRIPTION:
TRAVERSES THE TREE WITH 'FEEDER' AS ROOT.
STORES PARALLEL IMPEDANCES IN EVERY BRANCH POINT
COMPUTATIONS IS ONLY MADE WHEN MOVING TOWARDS THE TOP
WHEN THE ROOT IS REACHED THE SUBROUTINE CONTINUES COMPUTING
IMPEDANCES UNTIL THE TOP IS REACHED

VARIABLES:
FEEDER IS THE NUMBER OF THE FEEDER FOR WHICH THE IMPEDANCE IS COMPUTED
IMPEDANCES IN OHM

SEE ALSO:

FILE ROUTINE COMMENT
CONSTANTS ALL CONSTANTS DECLARED
COMMON ALL COMMON VARIABLES DECLARED
MOVING MOVE TAKES ONE STEP UP OR DOWN
MOVING DIREC COMPUTES NEW DIRECTION
COMP THE ABOVE ARE USED TO TRAVERSE THE TREE
PARALL COMPUTES THE EQUIVALENT TO TWO PARALLEL
IMPEDANCES
BRAPAR COMPUTES THE EQUIVALENT IMPEDANCE FOR A
BRANCH POINT

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

INCLUDE 'CONSTANTS.FOR'
INCLUDE 'COMMON.FOR'

INTEGER POI(2),OLD(2)
LOGICAL READY,FIRST
INTEGER A,DIR,BRA,I,J
REAL SUMR,SUMX,SUM,RES,REA,F(2),G(2)

POI(1)=FEETYP
POI(2)=FEEDER
READY=.FALSE.
OLD(1)=POI(1)
OLD(2)=POI(2)
DIR=DOWN
BRA=3
10 IF (.NOT. READY) THEN
    CALL MOVE(POI,OLD,DIR,BRA,.FALSE.)
    CALL DIREC(POI,OLD,DIR,BRA)
    IF (DIR.EQ.UP) THEN
        IF (POI(1).EQ.NODTyp) THEN
            IF (FIRST) THEN

C A LEAF IS FOUND. COMPUTATIONS START

    RES=NODeph(POI(2),4)
    REA=NODeph(POI(2),5)
    FIRST=.FALSE.
    ELSE

C THE NODE IS NOT A LEAF. PARALLEL IMPEDANCE IS COMPUTED INCLUDING
C CABLE IMPEDANCE

    G(1)=RES
    G(2)=REA
    F(1)=NODeph(POI(2),4)
    F(2)=NODeph(POI(2),5)
    CALL PARALL(F,G,RES,REA)
    I=NODEto(POI(2),1)
    I=NODEto(POI(2),1)
    RES=CABphy(I,1)+RES
    REA=CABphy(I,2)+REA
    ENDIF
    ELSEIF (POI(1).EQ.SWIType) THEN
        IF (.NOT. FIRST) THEN

C ADDS UP CABLE IMPEDANCE

            I=SWITCH(POI(2),2)
            I=SWITCH(POI(2),1)
            RES=RES+CABphy(I,1)
            REA=REA+CABphy(I,2)
            ELSE
                RES=1.0E+30
                REA=1.0E+30
            ENDIF

    ELSEIF (POI(1).EQ.TRATyp) THEN

C STORE IMPEDANCE FOR LATER USE

        TRAFos(POI(2),6)=RES
        TRAFos(POI(2),7)=REA
        ELSEIF (POI(1).EQ.BRATyp) THEN

C COMPUTES THE PARALLEL IMPEDANCE AND SAVES IT IN BRAPH FOR LATER USE

        DO 20 I=3,(2+BRAPoi(POI(2),1))
        J=BRAPoi(POI(2),1)
        IF (CABTop(J,1).EQ.OLD(1) .AND. CABTop(J,2).EQ.OLD(2)) THEN
            BRAPh(POI(2),1,1)=RES
            BRAPh(POI(2),1,2)=REA
        ELSEIF (CABTop(J,3).EQ.OLD(1) .AND. CABTop(J,4).EQ.OLD(2)) THEN
            BRAPh(POI(2),1,1)=RES
            BRAPh(POI(2),1,2)=REA
        ENDIF
    20 CONTINUE
CALL BRAPAR(POI(2), RES, REA)
I=BRAPOI(POI(2), 2)
I=BRAPOI(POI(2), 1)
RES=RES+CABPHY(I, 1)
REA=REA+CABPHY(I, 2)
ELSEIF (POI(1).EQ.FEETYP) THEN
  I=MSFEED(POI(2), 2)
  RES=RES+CABPHY(I, 1)
  REA=REA+CABPHY(I, 2)
ELSE
  WRITE(*,*) ' WRONG TYPE IN CompIm', POI(1), POI(2)
  STOP
ENDIF
ELSEIF (DIR.EQ.DOWN) THEN
C IF A BRANCH POINT IS REACHED AND A NEW DIRECTION DOWN HAS BEEN
C COMPUTED RES AND REA CONTAINING THE IMPEDANCE OF THE BRANCH THAT
C HAS BEEN VISITED MUST BE SAVED FOR LATER USE (I=(RES, J#REA))
IF (POI(1).EQ.BRATYP) THEN
  DO 30 I=3, (2*BRAPOI(POI(2), 1))
  A=BRAPOI(POI(2), I)
  IF (CABTOP(A, 1).EQ.OLD(1) .AND. CABTOP(A, 2).EQ.OLD(2)) THEN
    J=I
  ELSEIF (CABTOP(A, 3).EQ.OLD(1) .AND. CABTOP(A, 4).EQ.OLD(2)) THEN
    J=I
  ENDIF
  CONTINUE
  IF (J.NE.BRAPOI(POI(2), 2)) THEN
    BRAPH(POI(2), J, 1)=RES
    BRAPH(POI(2), J, 2)=REA
  ENDIF
30  ENDIF
IF (.NOT. READY) THEN
C HERE THE SAME COMPUTATIONS AS THE ABOVE IS PERFORMED UNTIL
C THE TRANSFORMER IS REACHED
OLD(1)=POI(1)
OLD(2)=POI(2)
CALL STEPSUM(POI, FALSE.)
IF (POI(1).EQ.SWITYP) THEN
  IF (.NOT. FIRST) THEN
    I=SWITCH(POI(2), 2)
    I=SWITCH(POI(2), 1)
    RES=RES+CABPHY(I, 1)
    REA=REA+CABPHY(I, 2)
  ENDIF
ELSEIF (POI(1).EQ.TRATYP) THEN
  TRAFOS(POI(2), 6)=RES
  TRAFOS(POI(2), 7)=REA
ELSEIF (POI(1).EQ.BRATYP) THEN
  DO 40 I=3, (2*BRAPOI(POI(2), 1))
    J=BRAPOI(POI(2), I)
    IF (CABTOP(J, 1).EQ.OLD(1) .AND. CABTOP(J, 2).EQ.OLD(2)) THEN
      BRAPH(POI(2), J, 1)=RES
      BRAPH(POI(2), J, 2)=REA
    ELSEIF (CABTOP(J, 3).EQ.OLD(1) .AND. CABTOP(J, 4).EQ.OLD(2)) THEN

SUBROUTINE FILPON(TRAFO)

C CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C AUTHOR MAGNUS BERGSTRAND

C VERSION 1

C DATE 1989-09-30

C PURPOSE:
C COMPUTES AND STORES THE POWER OF THE TRANSFORMER 'TRAFO'

C DESCRIPTION:
C FIRST THE IMPEDANCE FOR THE TRANSFORMER MUST BE COMPUTED
C USING COMPIM
C THEN THE POWER IS COMPUTED USING I=I/Z, P=Z*I**2

C VARIABLES:
C TRAFO IS THE NUMBER OF THE TRANSFORMER
C POWER IN VA
C VOLTAGE IN VOLT
C IMPEDANCE IN AMPERE

C SEE ALSO:

C FILE ROUTINE COMMENT
C CONSTANTS ALL CONSTANTS DECLARED
C COMMON ALL COMMON VARIABLES DECLARED
C COMP COMPIM COMPUTES THE IMPEDANCE FOR ONE FEEDER

C CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

INCLUDE 'CONSTANTS.FOR'
INCLUDE 'COMMON.FOR'

INTEGER TRAFO
REAL CUR

CUR=TRAMAX/SQRT(TRAFOS(TRAFO,6)**2+TRAFOS(TRAFO,7)**2)
TRAFOS(TRAFO,3)=TRAFOS(TRAFO,6)*CUR**2
TRAFOS(TRAFO,4)=TRAFOS(TRAFO,7)*CUR**2
RETURN
END
SUBROUTINE COMPV0(R,R,CUR,CURF, VOL, FI)
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
AUTHOR MAGNUS BERGSTRAND
C
VERSION 1
C
DATE 1989-09-30
C
PURPOSE:
C COMPUTES NODE VOLTAGES
C
DESCRIPTION:
C USES THE FORMULA V2 = V1 - Z*1
C
VARIABLES:
C THE IMPEDANCE Z IS REPRESENTED BY (R, j*1) BOTH IN OMS
C THE CURRENT I IS EXPONENTIAL FORM I = CUR*EXP(CURF)
C CUR IN AMPERE, CURF IN RADIANS
C THE VOLTAGE ALSO IN EXPONENTIAL FORM (BOTH IN AND OUT)
C VOLT IN VOLTS AND FI IN RADIANS
C
SEE ALSO:
C
FILE ROUTINE COMMENT
C CONSTANTS ALL CONSTANTS DECLARED
C COMMON ALL COMMON VARIABLES DECLARED
C COMP COMPUCU COMPUTES THE CABLE CURRENT
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
REAL R,R,CUR,CURF,VOLT,FI
REAL ZABS, ZFI
REAL REA, IM
ZABS = SQRT(R**2 + X**2)
ZFI = ATAN(X/R)
REA = VOLT*COS(FI) - CUR*ZABS*COS(CURF + ZFI)
IM = VOLT*SIN(FI) - CUR*ZABS*SIN(CURF + ZFI)
VOLT = SQRT(REA**2 + IM**2)
FI = ATAN(IM/REA)
RETURN
END

SUBROUTINE COMPCU(BRAF, BRACUR, CURREN, FI, POI, BRA)
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
AUTHOR MAGNUS BERGSTRAND
C
VERSION 1
C
DATE 1989-09-30
C
PURPOSE:
C COMPUTES THE CABLE CURRENT IN TWO CASES:
C 1. AFTER A NODE
C 2. THE CURRENT DOWN IN THE BRANCH 'BRA' IF 'POI' IS A BRANCH POINT
C
DESCRIPTION:
C 1. SINCE THE VOLTAGE IS KNOWN THE CURRENT IS DECREMENTED BY U/Z
2. All impedences for the different branches are known as well as the voltage the current is easily computed using the laws of Kirchhoff.

VARIABLES:
BRACUR in amperes (current from feeder to branchpoint)
BRAFI in radian (phase)
CURREN in amperes (1. in and out, 2. only out)
FI in radian (1. in and out, 2. only out)

SEE ALSO:
FILE ROUTINE COMMENT

CONSTANTS ALL CONSTANTS DECLARED
COMMON ALL COMMON VARIABLES DECLARED
COMP COMPUTE IMPEDANCE OF A FEEDER
COMPIM COMPUTES IMPEDANCE OF A FEEDER
COMPVO COMPUTES VOLTAGE
COMP FILVOL COMPUTES AND FILLS IN NODE VOLTAGES
AND CABLE CURRENTS FOR ONE FEEDER

INCLUDE 'CONSTANTS.FOR'
INCLUDE 'COMMON.FOR'

REAL BRAFI, BRACUR
REAL CURREN, FI
INTEGER POI(2), BRA
COMPLEX ZP, Z, CUR
REAL RES, REA, VO
REAL ALFA1, ZABS

IF (POI(1), EQ, BRATYP) THEN
    CALL BRAFAR (POI(2), RES, REA)
    ZP = CMPLX (RES, REA)
    Z = CMPLX (BRAPOI(2), BRA, 2)
    CUR = CMPLX (BRACUR * COS (BRAFI), BRACUR * SIN (BRAFI))
    CUR = CUR * ZP / Z
    CURREN = ABS (CUR)
    IF = ATAN (REAL (CUR)) / (REAL (CUR))
ELSE IF (POI(1), EQ, NODTYP) THEN
    VO = NODEPH (POI(2), 3)
    RES = NODEPH (POI(2), 4)
    REA = NODEPH (POI(2), 5)
    ALFA1 = ATAN (REA / RES) - FI
    ZABS = SQRT (RES**2 + REA**2)
    CURREN = CURREN - COS (ALFA1) * VO / ZABS
    FI = FI + ATAN (VO * SIN (ALFA1) / CURREN / ZABS)
ELSE
    WRITE ('***', 'WRONG TYPE IN COMPCU')
    STOP
ENDIF
RETURN
END

SUBROUTINE FILVOL (FEEDER, VO)

AUTHOUR MAGNUS BERGSTRAND
VERSION 1
DATE 1989-09-30
PURPOSE:
COMPUTES AND FILLS IN NODE VOLTAGES AND CABLE CURRENTS FOR ONE FEEDER

DESCRIPTION:
TRaverses the tree with 'feeder' as root.
When moving downwards computing new cable currents and node voltages
If any violations occur 'vio' becomes true and the common declared
viovec (violation vector) differs from zero
Only the biggest line capacity violation is stored in the first element
The sum of voltage drop constraints are stored in second element

VARIABLES:
Feeder is the number of the feeder
vio indicates if a violation occurred

SEE ALSO:

FILE ROUTINE COMMENT
CONSTANTS ALL CONSTANTS DECLARED
COMMOLN ALL COMMON VARIABLES DECLARED
MOVING MOVE TAKES ONE STEP UP OR DOWN
MOVING DIREC COMPUTES NEW DIRECTION
MOVING STEPUP TAKES ONE STEP UP
MOVING STEPD0 TAKES ONE STEP DOWN
COMP COMPVO COMPUTES NODE VOLTAGE
COMP COMPCU COMPUTES CURRENT

INCLUDE 'CONSTANTS.FOR'
INCLUDE 'COMMON.FOR'

INTEGER FEEDER
REAL R,X,VOL,BRAVOL(MAXNB),RSUM,XSUM,SUM,CURREN
INTEGER I,J,K,A,B
INTEGER WAY(MAXNB)
INTEGER BRA,DIA,P01(2),OLD(2)
LOGICAL READY,VIO,ENDNOD
REAL CUR,BRACUR(MAXNB)
REAL SQRT
REAL NODFI(MAXNB),BRAFI(MAXNB)
REAL CURFI,VOLFI,BCURFI(MAXNB)

INTRINSIC SQRT

VIO=.FALSE.
VI0VEC(FEEDER,1)=0.0
VI0VEC(FEEDER,2)=0.0
PO1(1)=FEETYP
PO1(2)=FEEDER
READY=.FALSE.
I=1
10 IF (.NOT. READY) THEN

MOVES TO TRANSFORMER MEMORIZING PATH

OLD(1)=PO1(1)
OLD(2)=PO1(2)
CALL STEPUP(PO1,.FALSE.)
IF (PO1(1).EQ.BRATYP) THEN
   DO 20 J=3,2+BRAPO1(PO1(2),1)
      K=BPAP0I(PO1(2),J)
      IF (CABTOP(K,1).EQ.OLD(1) .AND. CABTOP(K,2).EQ.OLD(2)) THEN
         WAY(I)=J
         I=I+1
   20 CONTINUE
   CALL STEPD0(PO1,.FALSE.)
ELSEIF (CABTOP(K,3).EQ.0LD(1) .AND. CABTOP(K,4).EQ.OLD(2)) THEN
    WAY(I)=J
    I=I+1
ENDIF

CONTINUE
ENDIF
READY=(POI(1).EQ.TRATYP)
GOTO 10
ENDIF
READY=.FALSE.

C
INITIATE

R=0.0
X=0.0
VOL=TRAMAX
VOLFI=0.0
DO 25 J=1,MAXNRB
   BRAVOL(J)=0.0
   BRACUR(J)=0.0
   BRAFI(J)=0.0
   SCURFI(J)=0.0
END
CONTINUE

C
STARTS COMPUTING ON THE WAY BACK TO FEEDER

CUR=TRAMAX/SQRT(TRAfos(POI(2),6)*2+TRAfos(POI(2),7)*2)
CURFI=ATAN(TRAfos(POI(2),7)/TRAfos(POI(2),6))
BRA=4

30
IF (.NOT. READY) THEN
   CALL STEP0(Poi,BRA,.,FALSE.)
   IF (POI(1).EQ.BRATYP) THEN
   I=I-1
   BRA=WAY(I)
   J=BRAPoi(POI(2),2)
   J=BRAPoi(POI(2),J)
   R=R+CABPHY(J,1)
   X=X+CABPHY(J,2)
   IF (CUR.GT.CABPHY(J,3)) THEN
      VIO=.TRUE.
      IF ((CUR-CABPHY(J,3)).GT.VIOVEC(FEEDER,1)) THEN
         VIOVEC(FEEDER,1)=VIOVEC(FEEDER,1)+CUR-CABPHY(J,3)
      ENDIF
   ENDIF
   CABPHY(J,4)=CUR
   BRACUR(POI(2))=CUR
   BSCURFI(POI(2))=CURFI
   CALL COMPV0(R,X,CUR,CURFI,VOL,VOLFI)
   BRAVOL(POI(2))=VOL
   BRAFI(POI(2))=VOLFI
   IF (VOL.LT.VOLLIM) THEN
      VIO=.TRUE.
   ENDIF
   R=0.0
   X=0.0
   CALL COMPCU(BCURFI(POI(2)),BRACUR(POI(2)),CUR,FI,POI,BRA)
   ELSEIF (POI(1).EQ.SWITYP) THEN
   J=SWITCH(POI(2),2)
   J=SWITCH(POI(2),J)
   R=R+CABPHY(J,1)
   X=X+CABPHY(J,2)
   CABPHY(J,4)=CUR
   IF (CUR.GT.CABPHY(J,3)) THEN
      VIO=.TRUE.
      IF ((CUR-CABPHY(J,3)).GT.VIOVEC(FEEDER,1)) THEN
         VIOVEC(FEEDER,1)=VIOVEC(FEEDER,1)+CUR-CABPHY(J,3)
ELSEIF (POI(1).EQ.FEETYP) THEN
    J=MSFEED(POI(2),2)
    R=R+CABPHY(POI(2),1)
    X=X+CABPHY(POI(2),2)
ELSE
    WRITE(*,'(A,A)') 'DEGENERATED NETWORK.FILVOL'
    STOP
ENDIF
READY=.POI(1).EQ.FEETYP .AND. POI(2).EQ.FEEDER
GOTO 30
ENDIF
C
C NOW THE FEEDER IS REACHED
C
C FROM HERE ON THE TREE IS TRAVERSED COMPUTING CURRENTS AND VOLTAGES
READY=.FALSE.
DIR=DOWN
40 IF (.NOT. READY) THEN
    CALL MOVE(POI,OLD,DIR,BRA,.FALSE.)
    IF (DIR.EQ.DOWN) THEN
        ENDNO=.TRUE.
        IF (POI(1).EQ.SWITYP) THEN
            IF (SWITCH(POI(2),1).EQ.CLOSED) THEN
                J=SWITCH(POI(2),2)
                J=SWITCH(POI(2),J)
                R=R+CABPHY(J,1)
                X=X+CABPHY(J,2)
                IF (CUR.GT.CABPHY(J,3)) THEN
                    VIO=.TRUE.
                    IF ((CUR-CABPHY(J,3)).GT.VIOVEC(FEEDER,1)) THEN
                        VIOVEC(FEEDER,1)=VIOVEC(FEEDER,1)+CUR-CABPHY(J,3)
                    ENDIF
                ENDIF
                CABPHY(J,4)=CUR
            ELSEIF (POI(1).EQ.NODITYP) THEN
                J=NODETO(POI(2),1)
                J=NODETO(POI(2),J)
                R=R+CABPHY(J,1)
                X=X+CABPHY(J,2)
                CABPHY(J,4)=CUR
                CALL COMPVO(R,X,CUR,CURFI,VOL,VOLFI)
                NODEP(P=POI(2),3)=VOL
                IF (VOL.LT.VOLLIM) THEN
                    VIO=.TRUE.
                ENDIF
                IF (CUR.GT.CABPHY(J,3)) THEN
                    VIO=.TRUE.
                    IF ((CUR-CABPHY(J,3)).GT.VIOVEC(FEEDER,1)) THEN
                        VIOVEC(FEEDER,1)=VIOVEC(FEEDER,1)+CUR-CABPHY(J,3)
                    ENDIF
                ENDIF
                ENDIF
        ENDIF
        ENDNO=.FALSE.
        R=VOLT/SQRT(NODEP(P=POI(2),4)**2+NODEP(P=POI(2),5)**2)
        NODEP(P=POI(2),1)=NODEP(P=POI(2),4)*CUR**2
        NODEP(P=POI(2),2)=NODEP(P=POI(2),5)*CUR**2
        CALL COMPUCU(1.0,1.0,CUR,CURFI,POI,BRA)
        R=0.0
        X=0.0
ELSEIF (POI(1).EQ.BRATYP) THEN
    J=BRAPOI(POI(2),2)
    J=BRAPOI(POI(2),J)
    R=R+CABPHY(J,1)
    X=X+CABPHY(J,2)
    CALL COMPVO(R,X,CUR,CURFI,VOL,VOLFI)
BRAVOL.PO(2)=VOL
BRAFI.PO(2)=VOLFI
IF (CUROLT.CABPHY(J,3)) THEN
  VIO=TRUE.
  IF ((CUR-CABPHY(J,3)) .GT. VIOMC(FEEDER,1)) THEN
    VIOMC(FEEDER,1)=VIOMC(FEEDER,1)+CUR-CABPHY(J,3)
  ENDIF
ENDIF
CABPHY(J,4)=CUR
BCURFI.PO(2)=CURFI
BRACUR.PO(2)=CUR
ELSE
  WRITE(*,*) ' WRONG IN NETWORK.FILVOL'
  WRITE(*,*) ' POINT IS',POI(1),POI(1)
  STOP
ENDIF
ENDIF
CALL DIREC.PO(0,0,DIR,0)
IF (DIR.EQ.DOWN .AND. POI(1).EQ.0) THEN
  CALL COMPCU.BCURS(POI(2), BRACUR.PO(2), CUR, CURFI, POI, BRA)
  VOL=BRAVOL.PO(2)
  VOLFI=BCRAFI.PO(2)
  R=0.0
  X=0.0
ENDIF
IF (DIR.EQ.UP .AND. ENDMOD) THEN
  IF (POI(1).EQ.NODTYP) THEN
    THIS IS THE ONLY PLACE NECESSARY TO CHECK THE VOLTAGE DROP CONSTRAINT
    ANY VIOLATION WILL ALWAYS OCCUR IN A LEAF NODE
    IF (MODEPH.PO(2),3).LT.VOLLIM) THEN
      VIOMC(FEEDER,2)=VIOMC(FEEDER,2)+VOLLIM-MODEPH.PO(2),3
    ENDIF
    ENDMOD=.FALSE.
  ENDIF
ENDIF
READY=0.PO(1).EQ.FEETYP
GOTO 40
ENDIF
RETURN
END

SUBROUTINE PARALL(A,B,RES,REA)

CCCCCCCCCCCCC

AUTHOR MAGNUS BERGSTRAND

VERSION 1

DATE 1989-09-30

PURPOSE:
COMPUTES THE IMPEDANCE (RES,j*REA)=A//B

VARIABLES:
A AND B ARE VECTORS (OHN, j*OHN)
RES IS RESISTANCE (OUT) IN OHM
REA IS REACTANCE (OUT) IN OHM

SEE ALSO:

FILE ROUTINE COMMENT
CONSTANTS ALL CONSTANTS DECLARED
COMMON BRAPAR

ALLOCMN VARIABLES DECLARED

COMPUTES PARALLEL IMPEDANCE WHEN POINT
IS A BRANCH POINT

REAL A(2), B(2), RES, REA
COMPLEX A1, B1, C1, C2

A1 = CMPLX(A(1), A(2))
B1 = CMPLX(B(1), B(2))
C1 = CMPLX(1, 0)
C2 = C1/A1 + C1/B1
C2 = C1/C2
RES = REAL(C2)
REA = AIMAG(C2)
RETURN
END

SUBROUTINE BRAPAR(NR, RES, REA)

AUTHOUR MAGNUS BERGSTRAND

VERSION 1

DATE 1989-09-30

PURPOSE:

COMPUTES THE PARALLEL IMPEDANCE FOR BRANCH POINT 'NR'

VARIABLES:

THE IMPEDANCE (RES, j*REA) IN OHMS (OUT)

SEE ALSO:

FILE ROUTINE COMMENT
CONSTANTS ALL CONSTANTS DECLARED
COMMON ALL COMMON VARIABLES DECLARED
COMP PARALL COMPUTES THE PARALLEL IMPEDANCE BETWEEN
TWO VECTORS

INCLUDE 'CONSTANTS.FOR'
INCLUDE 'COMMON.FOR'

INTEGER NR
REAL RES, REA
INTEGER I, J
COMPLEX A, B, C
LOGICAL READY

IF (BRAPOI(NR, 2).EQ.3) THEN
  I=4
ELSE
  I=3
ENDIF

A = CMPLX(BRAPH(NR, I, 1), BRAPH(NR, I, 2))
B = CMPLX(1, 0)
A = B/A

IF (I.LT. (BRAPOI(NR, 1)+2)) THEN
I=I+1
IF (I.NE.BRAPOI(NR,2)) THEN
   C=CMPLX(GRAPH(NR,I,1),BRAPH(NR,I,2))
   A=A+B/C
ENDIF
GOTO 10
ENDIF
A=B/A
RES=REAL(A)
REA=AIMAG(A)
RETURN
END
SUBROUTINE STAGE (FEEDER, OK)

AUTHOR MAGNUS BERGSTRAND

VERSION 1

DATE 1989-10-05

PURPOSE
WHEN A LOAD TRANSFER IS TO BE PERFORMED THIS SUBROUTINE
EXAMINES DIFFERENT POSSIBILITIES AND MAKES A FILE 'STAGE'
CONTAINING DATA ABOUT POSSIBLE SWITCHES

DESCRIPTION
TRAVERSES THE TREE WITH THE FEEDER 'FEEDER' AS ROOT.
WHEN A CLOSED SWITCH IS FOUND THE ESTIMATED REDUCTION
OF THE VIOLATION VECTOR IS COMPUTED AND STORED IN THE
FILE 'STAGE'. ALSO THE NUMBERS OF THE SWITCHES INVOLVED
IN THE NODES PRIORITY AS WELL AS THE NUMBER OF THE FEEDER
THAT WILL TAKE UP ANY LOAD WILL BE STORED.
THIS SUBROUTINE IS ONLY TO BE CALLED FROM 'STAGE1'

SEE ALSO
FILES Routines Comments
CONSTANTS ALL CONSTANTS DECLARED
COMMON ALL COMMON VARIABLES DECLARED
STEP2 FIRST STAGE SUPPORT. TRANSFERS LOAD TO
STAGE1 AN ADJACENT FEEDER.
STEP2 SECOND STAGE SUPPORT. TRANSFERS LOAD TO
STAGE2 AN ADJACENT FEEDER WHICH IN ITS TURN
TRANSFERS LOAD TO A SECONDARY SUPPORT
FEEDER.
STEP2 IMP COMPUTES THE ESTIMATED REDUCTION OF THE
STAGE3 VIOLATION VECTOR
MOVING MOVE TAKES ONE STEP UP OR DOWN
MOVING DIREC COMPUTES NEW DIRECTION
FETCH GETFEE GETS THE NUMBER OF THE FEEDER ON THE OTHER
SIDE OF AN OPEN SWITCH

VARIABLES
FEEDER IS THE NUMBER OF THE FEEDER THAT WILL TRANSFER LOAD
OK INDICATES IF ANY POSSIBLE LOAD TRANSFERS WERE FOUND

INCLUDE 'CONSTANTS.FOR'
INCLUDE 'COMMON.FOR'

INTEGER FEEDER
LOGICAL OK
INTEGER 1
INTEGER POI(2), OLD(2)
INTEGER DIR, BRA, SW12, FEE
REAL SUM, LOAD, ALPHA
INTEGER GETFEE
LOGICAL VIOLTE

EXTERNAL GETFEE
EXTERNAL VIOLTE

WRITE(*,*) 'ENTERS STAGE', STATUS='NEW', ACCESS='SEQUENTIAL'
OPEN(1,FILE='STAGE', STATUS='NEW', ACCESS='SEQUENTIAL')
POI(1)=FEETYP
POI(2)=FEEDER
DIR=DOWN
BRA=3
CALL STEPD0(POI, BRA, .FALSE.)
OK=.FALSE.

10 IF (POI(1).NE.FEETYP) THEN

C TRAVERSES THE TREE

CALL MOVE(POI, OLD, DIR, BRA, .FALSE.)
CALL DIREC(POI, OLD, DIR, BRA)
IF (POI(1).EQ.SWITYP) THEN
  IF (SWITCH(POI(2), 1).EQ.OPEN .AND. SWITCH(POI(2), 5), EQ.1) THEN

C A POSSIBLE LOAD TRANSFER IS FOUND

  FEE=GETFEE(POI(2))
  IF (.NOT. VIOLTE(FEE)) THEN

C THE FEEDER THAT WILL TAKE UP LOAD IS NOT VIOLATED

C COMPUTE THE IMPROVEMENT AND STORE DATA

  OK=.TRUE.
  CALL IMP(POI(2), SW12, SUM, LOAD, ALPHA)
  WRITE(1,*) POI(2), SW12, SUM, FEE, LOAD, ALPHA
ENDIF
ENDIF
ENDIF
GOTO 10
ENDIF
CLOSE(1, STATUS='KEEP')
RETURN
END

SUBROUTINE IMP(SWI, SW12, SUM, LOAD, ALPHA)

AUTHOR MAGNUS BERGSTRAND

VERSION 1

DATE 1989-10-05

PURPOSE
COMPUTES THE ESTIMATED REDUCTION OF THE VIOLATION VECTOR IF
THE SWITCH 'SW1' WOULD BE CLOSED AND 'SW12' OPENED.

DESCRIPTION
STARTING AT SWITCH 'SW1' THIS SUBROUTINE MOVES TO SWITCH 'SW12'
CHECKING VIOLATIONS ON THE WAY.

SEE ALSO
FILES Routines Comments
INCLUDE 'CONSTANTS.FOR'
INCLUDE 'COMMON.FOR'

INTEGER SW1, SW2
REAL SUM, CUR, LOAD, ALPHA
INTEGER POI(2), OLD(2), P(2)
INTEGER DIR, BRA
REAL SORT
LOGICAL EMDMOD, FIRST

INTRINSIC SORT

ALPHA=100
POI(1)=SW1TYP
POI(2)=SW1
CALL STEPUU(POI, .FALSE.)
10 IF (POI(1).NE.SW1TYP) THEN
   CALL STEPUU(POI, .FALSE.)
   GOTO 10
ENDIF
DIR=DOWN
SW2=POI(2)

SW2 IS FOUND
ONE HAVE TO START FROM THIS SWITCH BECAUSE OF THE REDUCTION
DEPENDS ON POSSIBLE BRANCH POINTS. ONE SWITCHING OPERATION
CAN TRANSFER LARGE LOADS IN DIFFERENT BRANCHES

SUM=0.0
BRA=.5
FIRST=.TRUE.
P(1)=POI(1)
P(2)=POI(2)
OLD(1)=0
OLD(2)=0
EMDMOD=.TRUE.
CALL STEP0O(POI, BRA, .FALSE.)
20 IF (POI(1).NE.P(1) .OR. POI(2).NE.P(2)) THEN
   IF (DIR.EQ.UP) THEN
      CALL MKESUM(POI, SUM, LOAD, EMDMOD, FIRST, ALPHA)
ELSE
    ENDMOD=.TRUE.
ENDIF
CALL MOVE(POI,OLD,DIR,ENDMOD,FALSE.)
CALL DIREC(POI,OLD,DIR,ENDMOD)
GOTO 20
ENDIF
ENDMOD=.FALSE.
30 IF (POI(1).NE.FEETYP) THEN
   
C THE BIGGEST LINE CAPACITY VIOLATION CAN BE IN THIS CABLE
   
   CALL STEPUP(POI,FALSE.)
   CALL MKESUM(POI,SUM,LOAD,ENDMOD,FIRST,ALPHA)
   GOTO 30
ENDIF
RETURN
END

SUBROUTINE MKESUM(POI,SUM,LOAD,ENDMOD,FIRST,ALPHA)

AUTHOUR MAGNUS BERGSTRAND

VERSION 1

DATE 1989-10-05

PURPOSE

COMPUTES THE MEASURE SUM

DESCRIPTION

IF THE CURRENT POINT IS A NODE A CHECK IS DONE TO SEE IF
THE NODE VOLTAGE IS BELOW THE MAXIMUM VOLTAGE DROP.
THEN A CHECK OF THE LINE CAPACITY CONSTRAINTS IS PERFORMED.
IF ANY VIOLATIONS IS FOUND SUM WILL BE INCREASED.

SEE ALSO

FILES

ROUTINES

COMMENTS

CONSTANTS ALL CONSTANTS DECLARED

COMMON ALL COMMON VARIABLES DECLARED

STEP2 IMP COMPUTES THE ESTIMATED REDUCTION OF THE
   VIOLATION IN CASE OF FIRST OR SECOND STAGE
   SUPPORT BY A CALL OF MKESUM

STEP2 EVAL COMPUTES THE ESTIMATED REDUCTION OF THE
   VIOLATION IN CASE OF LOAD CURTAILMENT
   BY A CALL OF IMPROV.

STEP2 IMPROV COMPUTES THE REDUCTION OF THE VIOLATION
   VECTOR IN CASE OF A LOAD CURTAILMENT

VARIABLES

POI IS THE CURRENT POINT (TYPE,NR)

SUM IS THE ESTIMATED REDUCTION

LOAD IS THE MAGNITUDE OF TRANSFERRED LOAD IN VA.

ENDMOD INDICATES THAT A NODE IS A LEAF

FIRST INDICATES FIRST CALL FOR A FEEDER

ALPHA IS THE PRIORITY

INCLUDE 'CONSTANTS.FOR'
INCLUDE 'COMMON.FOR'
REAL SUM,CUR,SQRT,MIN,VOLSUM,LOAD,LOAD1,ALPHA
INTEGER POI(2),I
LOGICAL ENDNOD,FIRST

INTRINSIC SQRT,MIN

C CUR (REDUCTION IN LINE CAPACITY VIOLATIONS), VOLSUM (REDUCTION IN
C VOLTAGE DROP VIOLATIONS) AND LOAD1 (MAGNITUDE OF TRANSFERED LOAD)
C ARE SAVED BETWEEN CALLS

SAVE CUR
SAVE VOLSUM
SAVE LOAD1

IF (FIRST) THEN

C INITIATE

    CUR=0.0
    VOLSUM=0.0
    FIRST=.FALSE.
    LOAD1=0.0

ENDIF

C THE INDEX OF THE CABLE LEADING TOWARDS THE FEEDER IS STORED IN '1'

IF (POI(1).EQ.SWITYP) THEN
    I=SWITCH(POI(2),2)
    I=SWITCH(POI(2),1)
ELSEIF (POI(1).EQ.NODETYP) THEN
    I=NODETO(POI(2),1)
    I=NODETO(POI(2),1)
    IF (ENDNOD) THEN
        LOAD1=LOAD1+SQRT(NODEPH(POI(2),1)**2+NODEPH(POI(2),2)**2)
    IF (VOLLIM.GT.NODEPH(POI(2),3)) THEN

C VOLTAGE DROP VIOLATION

    VOLSUM=VOLSUM+VOLLIM-NODEPH(POI(2),3)
    ALPHA=MIN(ALFA(POI(2)),ALPHA)
    ENDF
    ENDNOD=.FALSE.
ENDIF
ELSEIF (POI(1).EQ.BRATYP) THEN
    I=BRAPOI(POI(2),2)
    I=BRAPOI(POI(2),1)
ENDIF
IF (CABPHY(I,4).GT.CABPHY(I,3)) THEN

C LINE CAPACITY VIOLATION

    IF ((CABPHY(I,4)-CABPHY(I,3)).GT.CUR) THEN
        CUR=CABPHY(I,4)-CABPHY(I,3)
    ENDF
ENDIF
LOAD=LOAD1
SUM=SQRT((CUR**2)+(VOLSUM**2))
RETURN
ENDIF

LOGICAL FUNCTION VIOLTE(FEE)

C CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C AUTHOR MAGNUS BERGSTRAND
VERSION 1

DATE 1989-10-05

PURPOSE

CHECKS IF THE FEEDER 'FEE' IS VIOLATED

DESCRIPTION

IF THE FEEDER 'FEE' IS VIOLATED THEN

THIS LOGICAL FUNCTION EQUALS TRUE

VARIABLES

FEE IS THE NUMBER OF THE FEEDER

INCLUDE 'CONSTANTS.FOR'

INCLUDE 'COMMON.FOR'

VIOLTE=(VI0VEC(FEE,1).GT.0.0 .OR. VI0VEC(FEE,2).GT.0.0)

RETURN

END

SUBROUTINE STAGE1(FEE,OK,FIRST,OUTFEE)

AUTHOUR MAGNUS BERGSTRAND

DATE 1989-10-05

PURPOSE

TRANSFERS LOAD TO AN ADJACENT NON VIOLATED FEEDER.

METHOD

THE METHOD IS DESCRIBED IN 'SERVICE RESTORATION IN ELECTRIC
DISTRIBUTIONAL NETWORKS' BY MAGNUS BERGSTRAND

THE METHOD WAS FIRST DESCRIBED BY AOKI IN

'A NEW ALGORITHM FOR SERVICE RESTORATION IN DISTRIBUTION SYSTEMS'

IEEE/PES WINTER MEETING, NEW YORK 1989 PAPER 89 WM 085-2 PWRD

DESCRIPTION

STARTING AT A VIOLATED FEEDER ALL POSSIBLE LOAD TRANSFERS TO

A NON VIOLATED FEEDER IS EXAMINED. THE LOAD TRANSFER THAT WILL

REDUCE THE VIOLATION VECTOR THE MOST WILL BE TRIED FIRST. IF THIS

LOAD TRANSFER CAUSES A VIOLATION IT IS ABANDONED AND THE NEXT LOAD

TRANSFER IS TRIED. IF NO LOAD TRANSFER IS POSSIBLE WITHOUT ANY NEW

VIOLATIONS OK EQUALS FALSE.

STAGE1 HAS ANOTHER FUNCTION. WHEN A FIRST STAGE SUPPORT HAS FAILED

A SECOND STAGE SUPPORT IS TRIED. FIRST IN THIS SECOND STAGE A CALL

OF STAGE1 IS MADE WITH OK.EQ..FALSE.. THIS CAUSES THE FIRST STAGE

TO ALLOW VIOLATIONS IN THE PRIMARY SUPPORT FEEDER.

SEE ALSO

FILES

ROUTINES

COMMENTS

ALL CONSTANTS DECLARED

ALL COMMON VARIABLES DECLARED

IS THE SECOND STEP OF THE ALGORITHM

HANDLES THE CALLS OF STAGE1 AND STAGE2

MAKES A FILE CONTAINING DATA ABOUT THE

DIFFERENT LOAD TRANSFERS.
STEP2 STAGE2

THE SECOND STAGE SUPPORT, TRANSFERS FIRST LOAD
TO A PRIMARY SUPPORT FEEDER ALLOWING VIOLATIONS
THese VIOLATIONS ARE THEN REMOVED BY A LOAD
TRANSFER TO A SECONDARY SUPPORT FEEDER.

VARIABLES

FEE IS THE NUMBER OF VIOLATED FEEDER (IN)
IF OK IS FALSE A VIOLATION IN THE SUPPORT FEEDER IS ALLOWED
FIRST INDICATES IF IT IS THE FIRST TIME A LOAD TRANSFER IS TRIED.
IF A PREVIOUS LOAD TRANSFER HAS FAILED THE FILE CONTAINING DATA OF
POSSIBLE LOAD TRANSFERS IS KEPT.
OUTFEE IS AN OUTPARAMETER INDICATING THE NUMBER OF THE SWITCH NOW
VIOLATED DUE TO A LOAD TRANSFER. IT IS ONLY VALID WHEN THE CALL WAS
MADE WITH OK=FALSE.

#include 'CONSTANTS.FOR'
#include 'COMMON.FOR'

LOGICAL FIRST
INTEGER FEE,TRAFO,OUTFEE
LOGICAL VI0,OK
INTEGER SWI,SWI2,FEEDER
INTEGER BSWI,BSWI2,BFEE
REAL SUM,BSUM,SLASK,VIOSUM,LOAD
REAL MAX,MAX2

SAVE MAX,MAX2

EXTERNAL VIOSUM

WRITE(*,*)'ENTERS STAGE 1',FEE
IF (OK) THEN
   CALL STAGE(FEE,OK)
IF (OK) THEN
   WRITE(*,*)'READS IN STAGE1 **************************************************'
   OPEN(1,FILE='STAGE',STATUS='OLD',ACCESS='SEQUENTIAL')
   BSUM=0.0
   MAX=1.0E+30
   BSWI=MAXNRS+1
   10 READ(1,*,ERR=20) SWI,SWI2,SUM,FEEDER,LOAD,ALPHA
   SLASK=0.01+VIOSUM(FEE)-SUM
   SLASK=SUM/(SLASK+Beta)/LOAD/ALPHA
   WRITE(*,*)'IMPROVEMENT,FEEDER',SLASK,FEEDER
   IF (SLASK.GT.BSUM .AND. SLASK.LT.MAX) THEN
      BSUM=SLASK
      BSWI=SWI
      BSWI2=SWI2
      BFEE=FEEDER
   ENDF
   GOTO 10
   20 IF (BSWI.LT.MAXNRS+1) THEN
   THEN A POSSIBLE LOAD TRANSFER WAS FOUND
   THEN THE NETWORKS STATE IS SAVED
   WRITE(*,*)'SAVENE**************************************************************************'
   CALL SAVENE(BFEE)
   WRITE(*,*)FEE,BFEE
   WRITE(*,*)'SAVENE**************************************************************************'
   CALL SAVENE(FEE)
THE LOAD TRANSFER IS PERFORMED

SWITCH(BSWI2,1)=OPEN
SWITCH(BSWI,1)=CLOSED
SHINR(BSWI)=3
CHGSWI(BSWI2)=.NOT. CHGSWI(BSWI2)
SHINR(BSWI2)=3
CHGSWI(BSWI)=.NOT. CHGSWI(BSWI)
TRA=MSFEED(BFEE,1)
CALL FORBAC(BSWI,TRA,BFEE)

THE NEW STATE IS COMPUTED

CALL COMPIF(BFEE)
IF (SWITCH(BSWI2,2),EQ,3) THEN
  SWITCH(BSWI2,2)=4
ELSE
  SWITCH(BSWI2,2)=3
ENDIF
CALL COMPIF(FEE)
CALL FILPOW(TRA)
TRA=MSFEED(FEE,1)
CALL FILPOW(TRA)
CALL FILVOL(FEE,VIO)
CALL FILVOL(BFEE,VIO)
IF (VIO) THEN

SINCE A VIOLATION OCCURED THE OLD STATE MUST BE READ BACK

REWIND 1
WRITE(*,*)'CALLS RESNET***************'
REWIND 1
CALL RESNET
WRITE(*,*)'CALLS RESNET***************'
CALL RESNET

TAKE NEXT LOAD TRANSFER AND TRY AGAIN

MAX=BSUM
BSUM=0.0
BSWI=MAXNRS+1
GOTO 10
ELSEIF (.NOT. VIO) THEN

DELETE FILE AFTER A SUCCESSFUL LOAD TRANSFER

CLOSE(1,STATUS='DELETE')
WRITE(*,*)'LEAVES STAGE1 AFTER SUCCESSFUL SESSION, OK=T'
ENDIF
ELSE

NO POSSIBLE LOAD TRANSFER WAS FOUND.

SAVE FILE FOR LATER USE IN THE SECOND STAGE SUPPORT

CLOSE(1,STATUS='KEEP')
WRITE(*,*)'OK=.FALSE.'
OK=.FALSE.
ENDIF
ELSEIF (.NOT. OK) THEN

ALLOWS VIOLATION IN THE SUPPORT FEEDER

WRITE(*,*)'OK IS FALSE'
IF (FIRST) THEN
BSUM=0.0
MAX2=1.0E+30
ELSE
A PREVIOUS UNSUCCESSFUL SECOND STAGE HAS BEEN MADE.
THE OLD STATE MUST BE READ BACK
CALL RESNET
CALL RESNET
ENDIF
BSWI=MAXNRS+1
READ, COMPUTE AND IDENTIFY THE BEST LOAD TRANSFER
OPEN(1,FILE='STAGE', STATUS='OLD', ACCESS='SEQUENTIAL')
READ(1,*,ERR=30) SWI, SWI2, SUM, FEEDER, LOAD, ALPHA
SLASK=0.01+VIO*SUN(FEE)-SUM
SLASK=SUM/(SLASK+BETA)/LOAD/ALPHA
IF (SLASK.GT.BSUM .AND. SLASK.LT.MAX2) THEN
  BSUM=SLASK
  BSWI=SWI
  BSWI2=SWI2
  BFEED=FEEDER
ENDIF
GOTO 25
30 IF (BSWI.LT.MAXNRS+1) THEN
A POSSIBLE LOAD TRANSFER WAS FOUND
SAVE OLD STATE
WRITE(*,*)'SAVENE******************************************************************************'
CALL SAVENE(BFEED)
WRITE(*,*)FEED,BFEED
WRITE(*,*)'SAVENE******************************************************************************'
CALL SAVENE(FEE)
PERFORM LOAD TRANSFER
SWITCH(BSWI2,1)=OPEN
SWITCH(BSWI2,5)=0
SWITCH(BSWI,1)=CLOSED
SWINR(BSWI2)=3
SWINR(BSWI)=3
CHGSWI(BSWI2)=.NOT. CHGSWI(BSWI2)
CHGSWI(BSWI)=.NOT. CHGSWI(BSWI)
TRA=MSFEED(BFEED,1)
CALL FORRAC(BSWI,TRA,BFEED)
COMPUTE NEW STATE
CALL COMPIM(BFEED)
CALL COMPIM(FEE)
CALL FILPOW(TRA)
TRA=MSFEED(FEE,1)
CALL FILPOW(TRA)
CALL FILVOL(FEE,VIO)
CALL FILVOL(BFEED,VIO)
OK=.TRUE.
MAX2=BSUM
CLOSE(1,STATUS='KEEP')
OUTFEE=BFEED
ELSE
NO POSSIBLE LOAD TRANSFER WAS FOUND
WRITE(*,*)'STAGE1 WITH OK=F HAD AN UNSUCCESSFUL COMPLETION'
SUBROUTINE SAVNET(FEEDER)

AUTHOR MAGNUS BERGSTRAND

VERSION 1

DATE 1989-10-05

PURPOSE

Saves the former state in the network when a load transfer is to be made.

DESCRIPTION

Traverses a tree with the feeder 'FEEDER' as root saving everything in the network topology that could be changed due to a load transfer in the file 'HISTORY.DAT'.

A stack of backup files is created.

SEE ALSO

FILES

CONSTANTS

COMMON

MOVING

MOVING

MOVING

MOVING

STEP2

STEP2

Routines

All constants declared

All common variables declared

Takes one step up

Takes one step down

Takes one step up or down

Computes new direction

Reads back the former state from the file 'HISTORY.DAT'

Creates a stack of backup files by calls of SAVNET.

VARIABLES

'FEEDER' is the number of the feeder that will be saved.

INCLUDE 'CONSTANTS.FOR'

INCLUDE 'COMMON.FOR'

INTEGER FEEDER

INTEGER POI(2),Old(2)

INTEGER I,J

INTEGER DIR,BRA

LOGICAL READY

OPEN(2,FILE='HISTORY',STATUS='NEW',ACCESS='SEQUENTIAL')

POI(1)=FEETYP

POI(2)=FEEDER

WRITE(2,*)FEEDER,VIOVEC(FEEDER,1),VIOVEC(FEEDER,2)

DIR=DOWN

BRA=3

OLD(1)=POI(1)

OLD(2)=POI(2)

READY=.FALSE.
10 IF (.NOT. READY) THEN
C TRAVES THE TREE

CALL MOVE(POI, OLD, DIR, BRA, .FALSE.)
CALL DIREC(POI, OLD, DIR, BRA)
IF (DIR.EQ.UP) THEN

C FIRST IS THE POINT WRITTEN.
C THEN DIFFERENT DATA THAT MAY BE CHANGED ARE WRITTEN DEPENDING
C ON THE TYPE OF THE POINT

WRITE(2,*) POI(1), POI(2)
IF (POI(1).EQ.SWITYP) THEN
  WRITE(2,*) SWITCH(POI(2),1), SWITCH(POI(2),2), SWITCH(POI(2),5)
  I=SWITCH(POI(2),2)
  I=SWITCH(POI(2),1)
ELSEIF (POI(1).EQ.NODETYP) THEN
  WRITE(2,*) NODETO(POI(2),1), NODETO(POI(2),4)
  WRITE(2,*) NODEPH(POI(2),1), NODEPH(POI(2),2), NODEPH(POI(2),3)
  I=NODETO(POI(2),1)
  I=NODETO(POI(2),1)
ELSEIF (POI(1).EQ.BRATYP) THEN
  WRITE(2,*) BRAPOL(POI(2),2)
  DO 20 J=3,(2+BRAPOI(POI(2),1))
    WRITE(2,*) BRAPH(POI(2),J,1), BRAPH(POI(2),J,2)
  CONTINUE
  WRITE(2,*) BRAPOL(POI(2),2)
  I=BRAPOL(POI(2),2)
  I=BRAPOL(POI(2),1)
ELSEIF (POI(1).EQ.FEETYP) THEN
  I=MSFED(POI(2),2)
ELSEIF (POI(1).EQ.TRATYP) THEN
  WRITE(2,*) TRAFOS(POI(2),3), TRAFOS(POI(2),4)
  WRITE(2,*) TRAFOS(POI(2),5), TRAFOS(POI(2),6)
  WRITE(2,*) TRAFOS(POI(2),7)
ELSE
  WRITE(*,*) ' WRONG TYPE IN SAVEN2'
  STOP
ENDIF
IF (POI(1).NE.TRATYP) THEN
  WRITE(2,*) CABPHY(1,4)
ENDIF
ENDIF
READY=(POI(1).EQ.FEETYP)
GOTO 10
ENDIF
DIR=UP
READY=.FALSE.
30 IF (.NOT. READY) THEN
C EVEN THE TOPOLOGY ABOVE THE FEEDER IS SAVED

CALL STSEPUP(POI, .FALSE.)
IF (DIR.EQ.UP) THEN
  WRITE(2,*) POI(1), POI(2)
  IF (POI(1).EQ.SWITYP) THEN
    WRITE(2,*) SWITCH(POI(2),1), SWITCH(POI(2),2), SWITCH(POI(2),5)
    I=SWITCH(POI(2),2)
    I=SWITCH(POI(2),1)
  ELSEIF (POI(1).EQ.BRATYP) THEN
    WRITE(2,*) BRAPOI(POI(2),2)
    DO 40 J=3,(2+BRAPOI(POI(2),1))
      WRITE(2,*) BRAPH(POI(2),J,1), BRAPH(POI(2),J,2)
    CONTINUE
    WRITE(2,*) BRAPOI(POI(2),2)
  ELSEIF (POI(1).EQ.FEETYP) THEN
    WRITE(2,*) FEETYP(POI(2),2)
    I=FEETYP(POI(2),1)
  ELSEIF (POI(1).EQ.TRATYP) THEN
    WRITE(2,*) TRAFOS(POI(2),3), TRAFOS(POI(2),4)
    WRITE(2,*) TRAFOS(POI(2),5), TRAFOS(POI(2),6)
    WRITE(2,*) TRAFOS(POI(2),7)
  ELSE
    WRITE(*,*) ' WRONG TYPE IN SAVEN2'
    STOP
ENDIF
IF (POI(1).NE.TRATYP) THEN
  WRITE(2,*) CABIYTH(1,4)
ENDIF
ENDIF
READY=(POI(1).EQ.FEETYP)
GOTO 30
I=BRAPOI(POI(2),2)
I=BRAPOI(POI(2),I)
ELSEIF (POI(1).EQ.FEETYP) THEN
   I=MSPEED(POI(2),2)
ELSEIF (POI(1).EQ.TRATYP) THEN
   WRITE(2,*)TRAFOS(POI(2),3),TRAFOS(POI(2),4)
   WRITE(2,*)TRAFOS(POI(2),5),TRAFOS(POI(2),6)
   WRITE(2,*)TRAFOS(POI(2),7)
ELSE
   WRITE(*,*)' WRONG TYPE IN SAVENE' STOP
ENDIF
IF (POI(1).NE.TRATYP) THEN
   WRITE(2,*)CABPHY(I,4)
ENDIF
READY=POI(1).EQ.TRATYP
GOTO 30
ENDIF
CLOSE(2,STATUS='KEEP')
RETURN
END

SUBROUTINE RESNET

AUTHOR MAGNUS BERGSTRAND

VERSION 1

DATE 1989-10-05

PURPOSE
READS BACK A FORMER STATE WHEN A LOAD TRANSFER HAS FAILED.

DESCRIPTION
OPENS AND READS THE FILE 'HISTORY.DAT' (ALWAYS THE TOP OF THE STACK
OF BACKUP FILES)

SEE ALSO
FILES ROUTINES COMMENTS
CONSTANTS ALL CONSTANTS DECLARED
COMMON ALL COMMON VARIABLES DECLARED
STEP2 SAVENE SAVES A FORMER STATE IN THE FILE 'HISTORY.DAT'

INCLUDE 'CONSTANTS.FOR'
INCLUDE 'COMMON.FOR'

INTEGER POI(2),I,J

OPEN(2,FILE='HISTORY',STATUS='OLD',ACCESS='SEQUENTIAL')
READ(2,*)I,VIOVEC(I,1),VIOVEC(I,2)
10 READ(2,*,ERR=40)POI(1),POI(2)
IF (POI(1).EQ.SWITYP) THEN
   J=SWITCH(POI(2),1)
   READ(2,*)SWITCH(POI(2),1),SWITCH(POI(2),2),SWITCH(POI(2),5)
   I=SWITCH(POI(2),2)
   I=SWITCH(POI(2),I)
   IF (J.NE.SWITCH(POI(2),1)) THEN
      CHG5WI(POI(2))=.NOT.,CHG5WI(POI(2))
   ENDIF
ELSEIF (POI(1).EQ.NODYTP) THEN

READ(2,*)NODETO(POI(2),1),NODETO(POI(2),4)
READ(2,*)NODEPH(POI(2),1),NODEPH(POI(2),2),NODEPH(POI(2),3)
I=NODETO(POI(2),1)
I=NODETO(POI(2),1)
ELSEIF (POI(1).EQ.BRATYP) THEN
READ(2,*) BRAPOI(POI(2),2)
J=BRAPOI(POI(2),2)
J=BRAPOI(POI(2),2)
I=J
DO 20 J=J,(2+BRAPOI(POI(2),1))
   READ(2,*)BRAPHO(POI(2),J,1),BRAPHO(POI(2),J,2)
20 CONTINUE
READ(2,*)BRAPOI(POI(2),2)
I=BRAPOI(POI(2),2)
I=BRAPOI(POI(2),2)
ELSEIF (POI(1).EQ.FEETYP) THEN
   I=NSFEED(POI(2),2)
ELSEIF (POI(1).EQ.TRATYP) THEN
   READ(2,*)TRAFOS(POI(2),3),TRAFOS(POI(2),4)
   READ(2,*)TRAFOS(POI(2),5),TRAFOS(POI(2),6)
   READ(2,*)TRAFOS(POI(2),7)
ELSE
   WRITE(*,*)' WRONG TYPE IN RESNET'
STOP
ENDIF
IF (POI(1).NE.TRATYP) THEN
   READ(2,*)CABPHY(I,4)
ENDIF
GOTO 10
40 CLOSE(2,STATUS='DELETE')
RETURN
END

SUBROUTINE STAGE2(FEE,OK,FIRST,OUTFEE,FEE2,1)

AUTHOR MAGNUS BERGSTRAND

VERSION 1

DATE 1989-10-05

PURPOSE
TRANSFERS LOAD TO A SECONDARY SUPPORT FEEDER VIA A PRIMARY SUPPORT FEEDER.

METHOD
THE METHOD IS DESCRIBED IN 'SERVICE RESTORATION IN ELECTRIC DISTRIBUTION NETWORKS' BY MAGNUS BERGSTRAND.
THE METHOD WAS FIRST DESCRIBED BY AOKI IN 'A NEW ALGORITHM FOR SERVICE RESTORATION IN DISTRIBUTION SYSTEMS' IEEE/PES WINTER MEETING, NEW YORK 1987 PAPER 87 WM 085-2 PWRD

DESCRIPTION
LOAD IS TRANSFERED TO A PRIMARY SUPPORT FEEDER CAUSING A VIOLATION.
THIS VIOLATION IS THEN REMOVED BY TRANSFERED LOAD TO A SECONDARY SUPPORT FEEDER

SEE ALSO
FILES Routines Comments
CONSTANTS ALL CONSTANTS DECLARED
COMMON ALL COMMON VARIABLES DECLARED
STEP2 STEP2 PROVIDES THE FRAMEWORK TO STAGE1 AND STAGE2
C  STEP2  STAGE1  FIRST STAGE SUPPORT CAN BE FORCED TO ALLOW
C  STEP2  DELHIS  VIOLATION IN THE SUPPORT FEEDER.
C
C VARIABLES
C FEE IS THE NUMBER OF THE VIOLATED FEEDER
C OK=FALSE INDICATES THAT VIOLATION IS ALLOWED IN THE SECONDARY SUPPORT
C FEEDER. (THE ALGORITHM CAN BE GENERALIZED WITH A THIRD, FOURTH...
C STAGE SUPPORT)
C FIRST INDICATES FIRST CALL FOR FEEDER 'FEE'
C OUTFEE (OUTPARAMETER) IS THE NUMBER OF THE VIOLATED PRIMARY
C SUPPORT FEEDER. IT IS ONLY VALID WHEN OK=F.
C FEE2 (OUTPARAMETER) IS THE NUMBER OF THE VIOLATED SECONDARY
C SUPPORT FEEDER. IT IS ONLY VALID WHEN OK=F.
C 'I' IS THE NUMBER OF SUCCESSFUL CALLS OF FIRST STAGE SUPPORT.
C 2*I IS THE NUMBER OF BACKUP FILES IN THE STACK.
C
C CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C INCLUDE 'CONSTANTS.FOR'
C INCLUDE 'COMMON.FOR'
C
INTEGER FEE,OUTFEE,I
LOGICAL OK,FIRST
INTEGER FEE2,SLASK
LOGICAL READY,FIRST1,FIRST2,VIOLTE

EXTERNAL VIOLTE

WRITE(*,'**ENTERS STAGE 2**')
IF (.NOT. OK) THEN

C NO VIOLATION IS ALLOWED IN THE SECONDARY SUPPORT FEEDER
C
FIRST1=.TRUE.
WRITE(*,'**STAGE2 CALLS STAGE1 **')

C ALLOWS VIOLATION IN THE PRIMARY SUPPORT FEEDER
C
OK=.FALSE.
CALL STAGE1(FEE,OK,FIRST1,FEE2)
FIRST2=.FALSE.
FIRST1=.FALSE.
READY=.NOT. OK
I=0
10 IF (.NOT. READY) THEN

C DURING THIS LOOP THE VIOLATION IN THE PRIMARY SUPPORT FEEDER
C IS TRIED TO BE ELIMINATED
C THE VARIABLE I IS INCREMENTED FOR EACH SUCCESSFUL LOAD TRANSFER
C TO A SECONDARY SUPPORT FEEDER.
C
WRITE(*,'**LOOP IN STAGE 2 **')
OK=.TRUE.
CALL STAGE1(FOE2,OK,FIRST2,SLASK)
IF (.NOT. OK) THEN

C IT WAS NOT POSSIBLE TO TRANSFER LOAD FROM THE PRIMARY SUPPORT FEEDER
C TO ANY SECONDARY SUPPORT FEEDER
C READ BACK THE FORMER STATE
C TRY A NEW PRIMARY SUPPORT FEEDER
C
CALL DELHIS(I)
I=0
OPEN(1,FILE='STAGE',STATUS='OLD',ACCESS='SEQUENTIAL')
CLOSE(1,STATUS='DELETE')
CALL STAGE1(FEE,OK,FIRST1,FEE2)
    READY=.(.NOT. OK)
ELSE
    READY=.NOT. VIOLTE(FEE2)
    I=I+1
ENDIF
GOTO 10
ENDIF
ELSEIF (FIRST) THEN

C OK IS FALSE. VIOLATIONS ARE ALLOWED IN THE SECONDARY SUPPORT FEEDER

FIRST=.FALSE.
FIRST1=.TRUE.
OK=.FALSE.
CALL STAGE1(FEE,OK,FIRST1,FEE2)

C VIOLATIONS ARE ALLOWED IN THE PRIMARY SUPPORT FEEDER

FIRST2=.TRUE.
FIRST1=.FALSE.
READY=.(.NOT. OK)
I=0
40 IF (.NOT. READY) THEN
    WRITE(*,*)'LOOP IN STAGE 2 *************'
    OK=.TRUE.
    CALL STAGE1(FEE2,OK,FIRST2,OUTFEE)
ENDIF

C VIOLATIONS IN THE SECONDARY SUPPORT FEEDER WAS ALLOWED

IF (.NOT. OK) THEN

C NO POSSIBLE LOAD TRANSFER TO ANY SECONDARY SUPPORT FEEDER WAS FOUND
C READ BACK FORMER STATE
C FIND A NEW POSSIBLE PRIMARY SUPPORT FEEDER

READY=.TRUE.
OK=.FALSE.
CALL STAGE1(FEE2,OK,FIRST2,OUTFEE)
IF (.NOT. OK) THEN
    CALL DELHIS(I)
    I=0
    OPEN(1,FILE='STAGE',STATUS='OLD',ACCESS='SEQUENTIAL')
    CLOSE(1,STATUS='DELETE')
    CALL STAGE1(FEE,OK,FIRST1,FEE2)
ENDIF

C A NEW VIOLATED SECONDARY SUPPORT FEEDER WAS TRIED

READY=.(.NOT. OK)
ELSE
    I=I+1
ENDIF
ELSE

C THE ROUTINE MANAGED TO TRANSFER LOAD TO A NOW VIOLATED SECONDARY
C SUPPORT FEEDER

READY=.NOT. VIOLTE(FEE2)
I=I+1
ENDIF
GOTO 40
ENDIF
ELSE

C THERE HAVE BEEN AN UNSUCCESSFUL THIRD STAGE SUPPORT
C TRY TO TRANSFER LOAD TO A NEW SECONDARY SUPPORT FEEDER
FIRST2=.FALSE.
FIRST1=.FALSE.
READY=.FALSE.
I=0
50 IF (.NOT. READY) THEN
   OK=.FALSE.
   CALL STAGE1(FEE,OK,FIRST2,SLASK)
   LOAD WAS TRANSFERRED TO A PRIMARY SUPPORT FEEDER
   FIRST2=.FALSE.
   IF (.NOT. OK) THEN
   THE LOAD TRANSFER TO ANY PRIMARY SUPPORT FEEDER WAS
   UNSUCCESSFUL.
   READY=.TRUE.
   ELSE
   CALL STAGE1(SLASK,OK,FIRST1,FEE2)
   IF (OK) THEN
   LOAD WAS TRANSFERRED TO THE SECONDARY SUPPORT FEEDER
   READY=.TRUE.
   ELSE
   LOAD COULD NOT BE TRANSFERRED TO ANY SECONDARY SUPPORT FEEDER
   READ BACK OLD STATE AND TRY AGAIN
   CALL DELHIS(I)
   I=0
   ENDIF
   GOTO 50
   ENDIF
   ENDIF
   ENDIF
WRITE(*,*)'EXIT STAGE 2 OK=',OK
RETURN
END

SUBROUTINE STAGE3(FEE,OK,FIRST,OUTFEE)
CCC
AUTHOUR MAGNUS BERGSTRAND
VERSION 1
DATE 1989-10-05
PURPOSE
TRANSERS LOAD TO TERTIARY SUPPORT FEEDER
DESCRIPTION
FIRST A FIRST STAGE SUPPORT IS TRIED ALLOWING VIOLATIONS
IN THE SUPPORT FEEDER, THEN IS SECOND STAGE SUPPORT APPLIED
TO THIS VIOLATED SUPPORT FEEDER.
SEE ALSO
FILES ROUTINES COMMENTS
CONSTANTS ALL CONSTANTS DECLARED
COMMON ALL COMMON VARIABLES DECLARED
VARIABLES

FEED IS THE NUMBER OF THE VIOLATED FEEDER
OK INDICATES THAT NO VIOLATION IS ALLOWED IN THE TERTIARY SUPPORT
FEEDER
FIRST INDICATES WHETHER A FORMER CALL WITH FEEDER=FEED HAS BEEN MADE
OUTFEED IS THE NUMBER OF THE TERTIARY SUPPORT FEEDER
THESE LAST VARIABLES ARE WITHOUT MEANING IF NOT A FOURTH STAGE
SUPPORT IS IMPLEMENTED

C C C C C C C C C C C C C C C C C C C C C C C C C C C C C C C C C C C C C C

INCLUDE 'CONSTANTS.FOR'
INCLUDE 'COMMON.FOR'

INTEGER FEE,OUTFEED
LOGICAL OK,FIRST

IF (OK) THEN
   FIRST1=.TRUE.
   OK=.FALSE.
   CALL STAGE2(FEE,OK,FIRST1,OUTF,FEE2,NR)
10   IF (.NOT. READY) THEN
        CALL STAGE1(OUTF,OK,.TRUE.,SLASK)
   IF (.NOT. OK) THEN
        CALL DELHIS(NR)
   NR=0
   OPEN(1,FILE='STAGE',STATUS='OLD',ACCESS='SEQUENTIAL')
   CLOSE(1,STATUS='DELETE')
   CALL STAGE2(FEE,FIRST1,OUTF,FEE2,NR)
   READY=(.NOT. OK)
   ENDIF
   GO TO 10
   ENDIF
RETURN
END

SUBROUTINE DELHIS(I)

AUTHOR MAGNUS BERGSTRAND

VERSION 1

DATE 1989-10-05

PURPOSE
READS BACK THE STATE OF NETWORK AS IT WAS BEFORE AN UNSUCCESSFUL
LOAD TRANSFER

SEE ALSO
FILES       ROUTINES       COMMENTS
CONSTANTS   ALL CONSTANTS DECLARED
COMMON     ALL COMMON VARIABLES DECLARED
STEP2      RESNET       READS THE FILE 'HISTORY.DAT' CONTAINING
                      PARTS OF THE STATE

VARIABLES
'I' INDICATES THE SIZE OF THE STACK OF BACK UP FILES.
2*I IS THE NUMBER OF FILES
INTEGER I,J

DO 10 J=1,NR
   CALL REGNET
   CALL REGNET
10 CONTINUE
RETURN
END

SUBROUTINE STEP2(VIO)

AUTHOUR MAGNUS BERGSTRAND

VERSION 1

DATE 1989-10-05

PURPOSE

PROVIDES THE FRAMEWORK FOR THE FIRST AND SECOND STAGE SUPPORT

METHOD

THE ALGORITHM IS DESCRIBED IN 'SERVICE RESTORATION IN ELECTRIC
DISTRIBUTION NETWORKS' BY MAGNUS BERGSTRAND

THE METHOD WAS FIRST DESCRIBED BY AOKI IN
'A NEW ALGORITHM FOR SERVICE RESTORATION IN DISTRIBUTION SYSTEMS'
IEEE/PES WINTER MEETING, NEW YORK 1989 PAPER 89 WM 085-2 PWRD

DESCRIPTION

ALL FEEDERS ARE CHECKED IN ORDER IF THEY ARE VIOLATED
FOR ANY VIOLATED FEEDER:
FIRST STAGE SUPPORT
IF FAILURE: SECOND STAGE SUPPORT
IF SUCCESS REPEAT IT

SEE ALSO

FILES
CONSTANTS ALL CONSTANTS DECLARED
COMMON ALL COMMON VARIABLES DECLARED
STEP2 STAGE1 FIRST STAGE SUPPORT
STEP2 STAGE2 SECOND STAGE SUPPORT
STEP2 VIOLTE LOGICAL FUNCTION. CHECKS IF A FEEDER IS VIOLATE!

VARIABLES

VIO IS AN OUTPARAMETER. IT INDICATES WHETHER ANY FEEDER IS VIOLATED.

INCLUDE 'CONSTANTS.FOR'
INCLUDE 'COMMON.FOR'

LOGICAL OK,VIO
INTEGER FEE,OUTFEE, I,NR,J
LOGICAL VIOLTE

EXTERNAL VIOLTE

VIO=.FALSE.
DO 10 I=1,MAXNF
FEE=I
IF (VIOLTE(FEE)) THEN

C A VIOLATED FEEDER IS FOUND

OK=.TRUE.
IF (OK) THEN
CALL STAGE1(FEE,OK,.TRUE.,OUTFEE)
IF (.NOT. OK) THEN

C FIRST STAGE SUPPORT FAILED
C TRY A SECOND STAGE SUPPORT

OK=.TRUE.
CALL STAGE2(FEE,OK,.TRUE.,OUTFEE,NR)
IF (OK) THEN
DO 30 J=1,2*NR

C REMOVE USED FILES

OPEN(3,FILE='HISTORY',STATUS='OLD',ACCESS='SEQUENTIAL')
CLOSE(3,STATUS='DELETE')

30 CONTINUE
OPEN(3,FILE='STAGE',STATUS='OLD',ACCESS='SEQUENTIAL')
CLOSE(3,STATUS='DELETE')
ENDIF
ELSE

C REMOVE USED FILES

OPEN(1,FILE='HISTORY',STATUS='OLD',ACCESS='SEQUENTIAL')
CLOSE(1,STATUS='DELETE')
OPEN(1,FILE='HISTORY',STATUS='OLD',ACCESS='SEQUENTIAL')
CLOSE(1,STATUS='DELETE')
ENDIF
OK=(OK .AND. VIOLTE(FEE))
GOTO 20
ENDIF
ENDIF
VIO=VIO .OR. VIOLTE(FEE)

10 CONTINUE
RETURN
END

SUBROUTINE IMPROV(FEE,SWI)

AUTHOUR MAGNUS BERGSTRAND

VERSION 1

DATE 1989-10-05

PURPOSE
EXAMINES DIFFERENT POSSIBLE SWITCHES SUITABLE FOR A LOAD TRANSFER
THE BEST SWITCH, THAT IS THE ONE THAT WILL CAUSE A SMALL REDUCTION
OF THE VIOLATION VECTOR WILL BE CHOSEN

DESCRIPTION
TRAVERSES THE TREE WITH FEEDER 'FEE' AS ROOT.
WHEN A LEAF IS FOUND A CALL OF EVAL IS PERFORMED.

SEE ALSO
VARIABLES

'FEE' IS THE NUMBER OF THE VIOLATED FEEDER CAUSING THE LOAD CURTAILMENT

SWI (OUTPARAMETER) IS THE NUMBER OF THE SWITCH TO OPEN FOR THE

BEST LOAD CURTAILMENT

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

INCLUDE 'CONSTANTS.FOR'
INCLUDE 'COMMON.FOR'

INTEGER FEE,SWI,P(2),OLD(2)
INTEGER DIR,BRA
LOGICAL READY,ENDS,FIRST

P(1)=FEE
P(2)=FEE
DIR=DOWN
BRA=3
FIRST=.TRUE.
READY=.FALSE.

10 IF (.NOT. READY) THEN
   IF (DIR.EQ.DOWN) THEN
      ENDS=.TRUE.
      ENDF
      CALL MOVE(P,OLD,DIR,BRA,.FALSE.)
      CALL DIREC(P,OLD,DIR,BRA)
      IF (DIR.EQ.UP .AND. ENDS) THEN
         CALL EVAL(P,SWI,FIRST,FEE)
         ENDS=.FALSE.
      ENDF
      READY=P(1).EQ.FEETYP
      GOTO 10
   ENDF
RETURN
END

SUBROUTINE EVAL(P,SWI,FIRST,FEE)

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

AUTHOUR MAGNUS BERGSTRAND

VERSION 1

DATE 1989-10-05

PURPOSE

COMPUTES THE NUMBER OF THE SWITCH MOST SUITABLE FOR A LOAD
CURTAILMENT SO FAR

DESCRIPTION

STARTING FROM A LEAF THE SUBROUTINE FINDS THE FIRST SWITCH
TOWARDS THE TOP. WHEN A SWITCH IS FOUND THIS IS USED AS A ROOT IN
A TREE. THE TREE IS TRAVERSED AND IN THE MEAN TIME THE SUBROUTINE
COMPUTES THE MAGNITUDE OF THE WITHDRAWAL OF THE VIOLATION VECTOR.
BEGINING AT THE ROOT IS NECESSARY DUE TO THE EXISTANCE OF BRANCH POINTS
LOAD IN MORE THAN ONE BRANCH MAY BE CUT OFF
THE BEST MAGNITUDE OF VIOLATIONS WITHDRAWAL AND THE
NUMBER OF THE SWITCH CAUSING THE WITHDRAWAL ARE SAVED BETWEEN CALLS

SEE ALSO

FILES Routines Comments

CONSTANTS ALL CONSTANTS DECLARED

COMMON ALL COMMON VARIABLES DECLARED

MOVING STEPUP TAKES ONE STEP UP

MOVING STEPDO TAKES ONE STEP DOWN

MOVING MOVE TAKES ONE STEP UP OR DOWN

MOVING DIREC COMPUTES NEW DIRECTION

STEP2 MKSEGUN DECIDES THE NUMBER OF THE SWITCH DUE TO A STAGE SUPPORT

VARIABLES

P IS THE LEAF (TYPE, NR)

SWI (OUTPARAMETER) IS THE NUMBER OF THE SWITCH SO FAR

FIRST INDICATES FIRST CALL IN NEW STATE

FEE IS THE NUMBER OF THE VIOLATED FEEDER

INCLUDING 'CONSTANTS.FOR'

INCLUDING 'COMMON.FOR'

INTEGER POI(2), P(2), SWI, FEE, TEMP, DIR, BRA, OLD(2)

LOGICAL FIRST, READY

REAL BEST, VOLSUM, SUM, CURSUM, LOAD, SLASK

REAL VIOSUM, SQRT, I, MIN, ALPHA

REAL CUR1

INTRINSIC SQRT, MIN

EXTERNAL VIOSUM

SAVE TEMP

SAVE BEST

IF (FIRST) THEN

BEST = 1.0E+30

FIRST = .FALSE.

TEMP = 0

ENDIF

VOLSUM = 0.0

CURSUM = 0.0

CUR1 = 0.0

LOAD = 0.0

POI(1) = P(1)

POI(2) = P(2)

READY = .FALSE.

IF (.NOT. READY) THEN

CALL STEPUP(POI, .FALSE.)

READY = POI(1).EQ.SWITYP

GOTO 5

ENDIF

5 THE ROOT IS FOUND

TEMP = POI(2)

DIR = DOWN

BRA = 3

READY = .FALSE.

10 IF (.NOT. READY) THEN

IF (DIR.EQ.UP) THEN

THE NUMBER OF A CABLE IS STORED
IF (POI(1).EQ.SWITYP) THEN
  I=SWITCH(POI(2),2)
  I=SWITCH(POI(2),I)
ELSEIF (POI(1).EQ.NODTYPO) THEN
  I=NODETO(POI(2),I)
  I=NODETO(POI(2),I)
ENDIF

C NODE PRIORITY, CURRENT REDUCTION, LOAD REDUCTION AND
C REDUCTION IN VOLTAGE DROP IS COMPUTED

ALPHA=ALFA(POI(2))
Z=SQR((NODEPH(POI(2),4)**2)+(NODEPH(POI(2),5)**2))
CUR1=CUR1+NODEPH(POI(2),3)/Z
LOAD=LOAD+SQR((NODEPH(POI(2),1)**2+NODEPH(POI(2),2)**2)
IF (VOLLIM.GT.NODEPH(POI(2),3)) THEN
  VOLSUM=VOLSUM+VOLLIM-NODEPH(POI(2),3)
ENDIF
ELSEIF (POI(1).EQ.BRATYP) THEN
  I=BRAPOI(POI(2),2)
  I=BRAPOI(POI(2),I)
ENDIF
IF (CABPHY(I,4).GT.CABPHY(I,3)) THEN
  C CHECK OF LINE CAPACITY CONSTRAINTS
  IF ((CABPHY(I,4)-CABPHY(I,3)).GT.CURSUM) THEN
    CURSUM=MIN(CUR1,(CABPHY(I,4)-CABPHY(I,3))
  ENDIF
ENDIF
TRAVERSES THE TREE

CALL MOVE(POI,OLD,DIR,BRA,.FALSE.)
CALL DIREC(POI,OLD,DIR,BRA)
IF (POI(1).EQ.SWITYP .AND. POI(2).EQ.TEMP) THEN
  READY=.TRUE.
ENDIF
GOTO 10
ENDIF
READY=.FALSE.
20 IF (.NOT. READY) THEN

THE SAME TEST IS THEN MADE UP TO THE FEEDER

IF (POI(1).EQ.SWITYP) THEN
  I=SWITCH(POI(2),2)
  I=SWITCH(POI(2),I)
ELSEIF (POI(1).EQ.NODTYPO) THEN
  I=NODETO(POI(2),I)
  I=NODETO(POI(2),I)
  ALPHA=ALFA(POI(2))
ELSEIF (POI(1).EQ.BRATYP) THEN
  I=BRAPOI(POI(2),2)
  I=BRAPOI(POI(2),I)
ENDIF
IF (CABPHY(I,4).GT.CABPHY(I,3)) THEN
  IF ((CABPHY(I,4)-CABPHY(I,3)).GT.CURSUM) THEN
    CURSUM=MIN(CUR1,(CABPHY(I,4)-CABPHY(I,3))
  ENDIF
ENDIF
CALL STEPUP(POI,.FALSE.)
IF (POI(1).EQ.FEETYP) THEN
  READY=.TRUE.
ENDIF
GOTO 20
ENDIF

COMPUTE AND DECIDE

SUM=SQR((CURSUM**2)+(VOLSUM**2))
SLASK=3.01+VOLSUM*FEE-SUM
SLASK=LOAD*(SLASK+BETA)*ALPHA
IF (SLASK.LT.BEST .AND. SLASK.GT.0.0) THEN
  BEST=SLASK
  SWI=TEMP
ENDIF

FINAL REMARK
BEST AND SWI IS SAVE

RETURN
END

SUBROUTINE STEP3

AUTHOUR MAGNUS BERGSTRAND

VERSION 1

DATE 1989-10-05

PURPOSE
WHEN NO MORE LOAD TRANSFER IS POSSIBLE LOAD MUST BE CUT OFF.
AFTER A LOAD CURTAILMENT THERE IS A POSSIBILITY TO RESTORE
LOAD AMONG FORMER VIOLATION FEEDERS.

METHODE
THE METHODE IS DESCRIBED IN 'SERVICE RESTORATION IN ELECTRIC
DISTRIBUTION NETWORKS' BY MAGNUS BERGSTRAND.
THE METHOD WAS FIRST DESCRIBED BY AOKI IN
'A NEW ALGORITHM FOR SERVICE RESTORATION IN DISTRIBUTION SYSTEMS'
IEEE/PES WINTER MEETING, NEW YORK 1989 PAPER B5 WM 085-2 PWRD

DESCRIPTION
THIS SUBROUTINE HANDLES BOTH STEP3 AND STEP4 IN THE ALGORITHM
(LOAD CURTAILMENT AND RESTORATION OF CURTAILED LOADS)
THE SUBROUTINE WILL SEARCH FOR A VIOLATED FEEDER,
WHEN A VIOLATED FEEDER IS FOUND A CALL OF IMPROV IS MADE, THUS
PRODUCING THE NUMBER OF THE BEST SWITCH FOR A LOAD CURTAILMENT.
SWITCHING OPERATION IS PERFORMED AND THE NEW STATE WILL BE COMPUTED
THE NUMBER OF THE SWITCH IS SAVED ON THE FILE 'CURTAIN.DAT' FOR LATER
USE
A DESCRIPTION OF THE FOURTH STEP IN THE ALGORITHM IS FOUND FURTHER DOWN

SEE ALSO
FILES
  ROUTINES
  COMMENTS
  ALL CONSTANTS DECLARED
  ALL COMMON VARIABLES DECLARED
  TAKES ONE STEP UP
  TAKES ONE STEP DOWN
  TAKES ONE STEP UP OR DOWN
  COMPUTES NEW DIRECTION
  COMPUTES THE NUMBER OF THE SWITCH THAT WILL BE
  OPENED DUE TO A LOAD CURTAILMENT
  COMPUTES IMPEDANCE
  COMPUTES AND FILLS IN NODE VOLTAGES AND
C 
C CABLE CURRENTS 
C COMPUTES POWER 
C SAVEME SAVES THE STATE OF THE NETWORK 
C RESEND READS BACK FORMER STATE 
C FIRST STAGE SUPPORT 
C 
C INCLUDE 'CONSTANTS.FOR' 
C INCLUDE 'COMMON.FOR' 
C 
INTEGER I, FEE, OUTFEE, SWI, SWI2 
LOGICAL VIOLTE 
LOGICAL OK, FIRST, READY 
REAL 51, 52, BSUM, SUM, MINI, SLASK, VIOSUM, LOAD 
INTEGER BSWI, BSWI2, BFEE, FEEDER, TRA 
INTEGER POI(2), P(2), OLD(2), DIR, BRA 
C 
EXTERNAL VIOSUM 
EXTERNAL VIOLTE 
C 
WRITE(*,*) \ 'ENTERS STEP 3' 
OPEN(2, FILE='CURTAIN', STATUS='NEW', ACCESS='SEQUENTIAL') 
DO 10 I=1, MAXNRF 
FEE=I 
OK=.TRUE. 
FIRST=.TRUE. 
READY=.NOT. VIOLTE(FEE) 
15 IF (.NOT. READY) THEN 
C 
A VIOLATED FEEDER IS FOUND 
C 
WRITE(*,*) \ 'STAGE 3 WILL BE CUTTING LOAD IN FEEDER', FEE 
CALL IMPROV(FEE, SWI) 
C 
PERFORM SWITCHING OPERATIONS 
C 
SWITCH(SWI, 1)=OPEN 
SWINR(SWI)=2 
CHGSWI(SWI)=.NOT. CHGSWI(SWI) 
C 
SAVE NUMBER OF THE SWITCH ON 'CURTAIN.DAT' 
WRITE(2, *) SWI 
C 
COMPUTE NEW STATE 
C 
CALL COMPIM(FEE) 
TRA=MSFEED(FEE, 1) 
CALL FILPOW(TRA) 
CALL FILVOL(FEE, VI) 
READY=.NOT. VIOLTE 
GOTO 15 
ENDIF 
C 
CONTINUE 
ENDFILE 2 
REWIND 2 
C 
DESCRIPTION 
C HERE STEP 4, THE RESTORATION OF CURTAILED LOADS, BEGINS. 
C ONE AT A TIME THE CURTAILED LOADS ARE REENERGIZED IF IT IS 
C POSSIBLE. ONE CURTAILED SECTION MAY LACK ENERGIZED 
C NEIGHBOURS DUE TO REPEATED LOAD CURTAILMENTS IN THE SAME 
C BRANCH. 
C THEN A FIRST STAGE SUPPORT IS TRIED. THE LOAD
C WILL ONCE AGAIN BE CUT OFF IF THIS FAILS

S1=0.0
S2=0.0
40 READ(2,*,ERR=60)SWI

C READ IN 'CURTAIN.DAT'

POI(1)=SWITYP
POI(2)=SWI

C THEN A SEARCH FOR THE FIRST SWITCH ABOVE THE READ SWITCH STARTS
C IF THIS NEW SWITCH IS CLOSED IT IS POSSIBLE TO REENERGIZE THE
C FORMER DEENERGIZED ZONE

CALL STEPUP(POI, .FALSE.)
READY=POI(1).EQ.SWITYP
50 IF (.NOT. READY) THEN
   CALL STEPUP(POI, .FALSE.)
   READY=POI(1).EQ.SWITYP
   GOTO 50
ENDIF
IF (SWITCH(POI(2),1).EQ.CLOSED) THEN
   READY=.FALSE.
   DIR=DOWN
   P(1)=POI(1)
   P(2)=POI(2)
55 IF (.NOT. READY) THEN
   CALL MOVE(P, OLD, DIR, BRA, .FALSE.)
   IF (P(1).EQ.NODRTYP .AND. DIR.EQ.UP) THEN
      FEEDER=NODETO(P(2),4)
      S1=S1+NODEPH(P(2),1)
      S2=S2+NODEPH(P(2),2)
   ENDIF
   CALL DIREC(P, OLD, DIR, BRA)
   READY=(P(1).EQ.POI(1) .AND. P(2).EQ.POI(2))
   GOTO 55
ENDIF

C SAVE OLD STATE

CALL SAVENE(FEEDER)

C PERFORM SWITCHING OPERATION

TRA=MSFEED(FEEDER,1)
SWITCH(SWI,1)=CLOSED
CHGSWI(SWI)=.NOT. CHGSWI(SWI)

C COMPUTE NEW STATE

CALL COMPIM(FEEDER)
CALL FILPOW(TRA)
CALL FILVOL(FEEDER,V10)
OK=.TRUE.
FIRST=.TRUE.

C TRY A FIRST STAGE SUPPORT

CALL STAG1(FEEDER,OK,FIRST,OUTFE)
IF (.NOT. OK) THEN

C READ BACK OLD STATE

CALL RESNET
RESTOR(1)=RESTOR(1)-S1
RESTOR(2)=RESTOR(2)-S2
ENDIF
ELSE
THE SWITCH WAS CLOSED
FIND OUT HOW BIG THE LOAD WAS THAT COULD NOT BE RESTORED
READY=.FALSE.
DIR=DOWN
P(1)=POI(1)
P(2)=POI(2)
58 IF (.NOT. READY) THEN
CALL MOVE(P,OLD,DIR,BRA,.FALSE.)
IF (P(1).EQ.NODTPY .AND. DIR.EQ.UP) THEN
FEEDER=NODETO(P(2),4)
S1=S1+NODEPH(P(2),1)
S2=S2+NODEPH(P(2),2)
ENDIF
CALL DIREC(P,OLD,DIR,BRA)
READY=(P(1).EQ.POI(1) .AND. P(2).EQ.POI(2))
GOTO 58
ENDIF
RESTOR(1)=RESTOR(1)-S1
RESTOR(2)=RESTOR(2)-S2
ENDIF
GOTO 40
60 CLOSE(2,STATUS=’DELETE’)
RETURN
END

REAL FUNCTION VIOSUM(FEE)
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
AUTHOR MAGNUS BERGSTRAND
C
VERSION 1
C
DATE 1989-10-05
C
PURPOSE
THIS FUNCTION COMPUTES THE EFFECTIVE LENGTH OF THE VIOLATION
C
FILES ROUTINES COMMENTS
C
STEP2 VIOLTE INDICATES WHETHER A FEEDER IS VIOLATED OR NOT
C
VARIABLES
C
‘FEE’ IS THE NUMBER OF THE (PERHAPS) VIOLATED FEEDER
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
INCLUDE ’CONSTANTS.FOR’
INCLUDE ’COMMON.FOR’

INTEGER FEE
REAL SQRT
INTRINSIC SQRT

VIOSUM=SQRT(VIOVEC(FEE,1)**2+VIOVEC(FEE,2)**2)
RETURN
END