配电网可靠性和故障软自愈研究

摘要

电网是全国民生、工业基础，由发、输、变、配、用各环节构成统一整体。运行需要由较高的稳定性保证供电质量，并且在时空拓扑结构不断变化的情况下，要对故障能够进行判断和处理。利用图数据结构与实际电网在结构上保持一致，从而变成图论问题，自然、直观处理整体拓扑，并且能够在业务流动过程中，不断并入新的电网，实现业务拓展，图拓扑结构拓展。自然也就能够对可靠性进行图上分析，并且实现基于数据的智能化配电网自愈控制。

针对问题一：本文采用模型 + 指标进行可靠性评估：首先给出评估指标，然后在图数据结构的基础下，使用等值最小路混合的方法，将元件可靠性分为最小路上以及非最小路上，并且最终邓加到最小路上，然后根据公式计算出可靠性。最后，对于拓扑信息进行了定性的灵敏度分析，给出了算例比较结果。

针对问题二：本题根据电网的拓扑关系，首先推导出故障发生时电网元件之间动作的因果关系，构建因果网络和规则矩阵 R。然后根据电网警告信息建立状态矩阵 T，通过 R 的转置与 T 的计算得出故障信息向量 T'。最后手动输入可能故障的节点 F，通过 T' 与 F 相与，得到最终故障节点向量 F'，找到故障节点。

针对问题三：本文首先将馈线以及开关进行图论抽象化，然后根据边权值检测推测出故障点，接着关闭邻接开关进行故障自动隔离，然后进行无故障段的恢复供电，通过连通分量个数的判断得出孤岛个数，以及边集的转化，实现孤岛的消除，进而完成负荷转移恢复供电。最后，给出了“软自愈”和“硬自愈”的成本差异估计。

关键字：最小路　等值法　因果网络　孤岛　连通分量　BFS

一、 问题重述

1.1 问题背景

配电网是供电系统的关键组成部分，是决定供电质量的重要因素，90% 的断电都是因为配电网配置不合理导致的。配电网的可靠性是高质量供电的重要指标。连续时空拓扑信息，是一个利用发展的眼光研究配电系统业务的重要角度。而对于电网故障，故障自愈是一种事故恢复的快速自动化方法，能实现线路故障的自动判断、自动隔离和负荷转移，对于提升供电质量有很大的促进作用。

1.2 问题提出

问题一: 针对配电网，建立一种基于连续时空电网拓扑的可靠性评估模型，并且要对连续时空拓扑信息差异引起的模型可靠性差异进行说明，也就是对拓扑信息进行灵敏度分析。

问题二: 建立一种基于连续时空电网拓扑的配电网故障检测模型，并举例配电网信息差异引起的故障件检测差异。

问题三: 基于“业务内生电网拓扑信息”，研究设计电网的“软自愈”方案，实现同类线路故障的自动判断、自动隔离、负荷转移恢复供电的算法，最后估算“软自愈”与“硬自愈”方案的成本差异，并且结合中压 (10kv) 电网故障图说明。

二、 问题分析

2.1 问题一分析

要求建立可靠性评估模型，并且是依赖于连续时空电网拓扑的，已经明确了自变量。对于可靠性评估，采用指标 + 模型的评估思路。先通过模型，分析拓扑信息，进一步量化指标，对可靠性进行一个合理评估。最后对自变量——电网拓扑信息进行灵敏度分析。

2.2 问题二分析

随着配电网的不断智能化，配电网可以通过 SCADA 系统[?]进行数据监测和控制。通过收集和处理配电网的监测信息，运用故障发生时配电网中各种警告信息，能够逆向推断配电网的错误位置，进行配电网的故障检测。

2.3 问题三分析

软自愈实现方案：先将配电网抽象成图，之后根据异常电流值确定问题开关，进而确定故障点，实现自动判断。然后将故障点周围开关关闭，使其孤岛化，实现自动隔离。最后，对于因此产生的孤岛正常节点，将其并入存在电源的连通分支，即实现了恢复供电。 估算与硬自愈的成本差异，

三、 模型假设

假设 1 假设配电网仅存在弱环电网或者无环拓扑结构。

假设 2 配电网有完备的数据监控采集设备，能够根据记录的时间来判断监控设备开关动作的时序。可以用于查找故障主因，便于故障的后续分析。

假设 3 配电网中包括三种类型的节点：故障节点、保护动作节点和短路器跳闸节点。

例: 故障节点线路 LI 上发生错误，则保护节点 $OP1$ 将动作使断路器 $CB1$ 跳闸。

假设 4 此模型不讨论含有分布式电源 DG 的情况

四、 符号说明

表 1　符号表

变量	说明	量纲
λ	负荷点平均故障停运率	次/a
U	负荷点平均停运时间	h/a
$SAIFI$	系统平均停电频率指标	次 (户 · a)
$SAIDI$	系统平均停电持续时间	h(户 · a)
$ASAI$	平均供电可靠率	%
$ASUI$	平均供电不可靠率	%
$AIHC$	用户平均停电时间	h
$AITC$	用户平均停电次数	次

五、 模型的建立与求解

5.1 问题一

配电网的可靠性评估方法大致分为：模拟法和解析法[?]。解析法模型准确、原理简单，并且便于对不同元件引起的拓扑结构差异进行灵敏度分析，所以我们以解析法中的网络法为基础，采用基于等值法和最小路法的混合算法评估模型[?]，并且使用 IEEE 所提出的标准指标进行量化评价，然后根据这些指标，对拓扑信息的灵敏度进行分析。

5.1.1 指标建立

配电网的可靠性指标分为负荷点和系统的可靠性指标。

负荷点的可靠性指标包括：1) 负荷点平均故障停运率 λ(次/a)；2) 负荷点平均停运时间 U (h/a)；系统的可靠性指标: 系统平均停电频率指标 $SAIFI$; 系统平均停电持续时间 $SAIDI$; 平均供电可靠率 $ASAI$; 平均供电不可靠率 $ASUI$;

针对电网的最基础串并联结构，我们给出相关指标计算公式：

N 个串联可修复元件，计算采用如下公式：

$$\lambda = \sum_{i=1}^{n} \lambda_i \tag{1}$$

$$U = \sum_{i=1}^{n} \lambda_i r_i \tag{2}$$

N 个并联可修复元件，计算采用如下公式：

$$\lambda = (\prod_{i=1}^{n} \lambda_i)(\sum_{i=1}^{n} r_i) \tag{3}$$

对于系统可靠性指标，都可以通过负荷点的三个基本指标计算得到, 计算采用如下公式：

$$\begin{cases} SAIFI = \dfrac{\sum \lambda_i N_i}{\sum N_i} \quad ; \quad SAIDI = \dfrac{\sum U_i N_i}{\sum N_i} \\ ASAI = 1 - \dfrac{\sum U_i N_i}{8760 \sum N_i} \quad ; \quad ASUI = 1 - ASAI \end{cases} \tag{4}$$

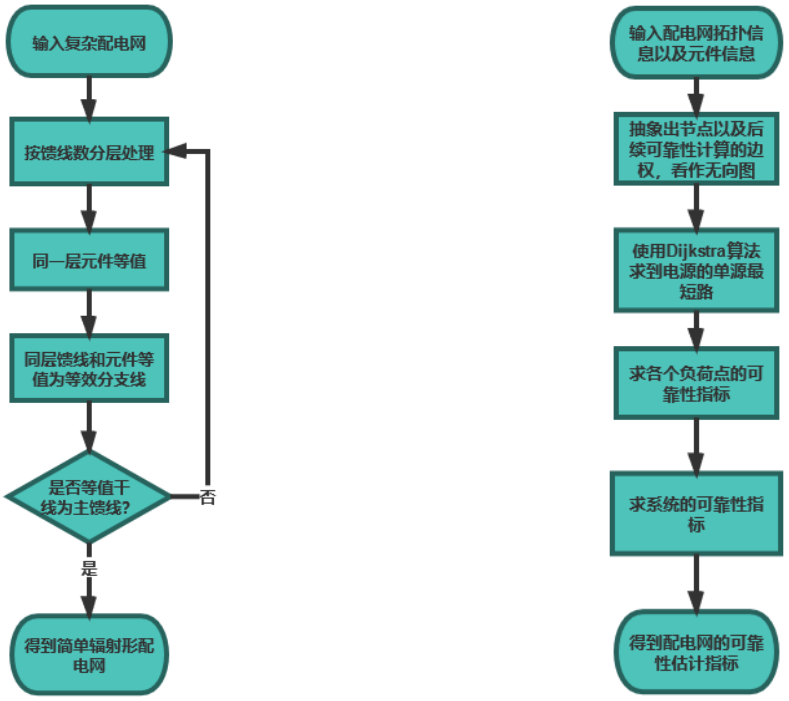
公式中 $\lambda_i' r_i$ 为负荷点 i 的年平均故障率;r_i 是负荷点 i 的年平均故障时间,U 是负荷点的单次平均停运时间,N_i 代表用户数。

5.1.2 等值法简化电网

结构复杂的配电网，可以通过可靠性等值的方法[?]将其等价为简单的辐射形配电网，简化拓扑信息，方便计算。实际配电系统一般由主副馈线组成，根据馈线数量分层

处理，每一条馈线以及所连接的元件同属一层，之后等消除一条相应的等效分支线，自底而上逐层等价，最后得出简单拓扑结构的电网。

等值法的计算流程如图 1(a) 所示。



(a) 等值算法流程　　**(b) 最小路算法流程**

图 1　算法流程图

5.1.3 Dijkstra 最小路法

配电网通常采用多闭环的设计思路，而运行过程中采用开环的辐射状结构。其次复杂配电系统等值之后，都可以简化为简单的辐射性网络。作为可靠性评估，我们要看重不同元件的可靠度对负荷点的影响，从而进一步估计系统的影响。所以，需要对元件进行分类，采用最小路法，把元件分成最小路元件以及非最小路元件，非最小路对负荷点的影响可以折算到最小路上。

基本思路：将配电网抽象成图论上的无向图，具体图化方式为：将负荷母线以及变电站、变压器看作节点，线路、分段开关看作边，边权视为 1，配电网可以抽象为图 $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ 表示，自然得出图的邻接矩阵，电源看作最短路的源点，化成单源无向图。然后求每一个负荷点到电源最短路，即使用 Dijstra 求单源最短路。之后，把非最小路的元件故障影响折算到最小路，进行可靠性指标计算。Dijkstra 最小路法算法流程如图 1(b) 所示。

5.1.4 灵敏度分析

配电网的可靠性水平与连续时空电网拓扑息息相关。而电网拓扑本质上是取决于电网的布线以及元件位置参数、性质参数，进而导致拓扑结构的位置不同。因此，配电网的可靠性灵敏度分析是要对元件、设备的参数进行偏分，从而反应设备参数变化引起的时空拓扑变化对可靠性的影响程度。使用参数[?]用户平均停电时间 $AIHC$ 以及用户平均停电次数 $AITC$ 来计算灵敏度，公式如下：

用户平均停电时间对元件的故障率 λ 的灵敏度:

$$\frac{\partial AIHC}{\partial \lambda_i} = \frac{\sum_{j\in i} r_{ij} N_j}{\sum_{j=1}^{n} N_j} \tag{5}$$

用户平均停电时间对元件的故障时间 r 的灵敏度:

$$\frac{\partial AIHC}{\partial r_i} = \frac{\sum_{j\in i} \lambda_{ij} N_j}{\sum_{j=1}^{n} N_j} \tag{6}$$

用户平均停电次数对元件的故障率 λ 的灵敏度:

$$\frac{\partial AITC}{\partial \lambda_i} = \frac{\sum_{j\in i} N_j}{\sum_{j=1}^{n} N_j} \tag{7}$$

5.1.5 举例说明

图 2 是一个较为普遍的辐射形配电网系统为例，以次结构说明配电网拓扑信息对于可靠性的影响以及灵敏度分析。经过拓扑抽象图化后变为一个图论上的无向图，如图 3 所示。$V0$ 是电源点，$V2\,V6\,V7\,V8$ 分别对应负荷点 $a\,b\,c\,d$，$V1\,V3\,V4\,V5$ 是连接点，主馈线和分支线的交点。$E1-E8$ 是图的边。

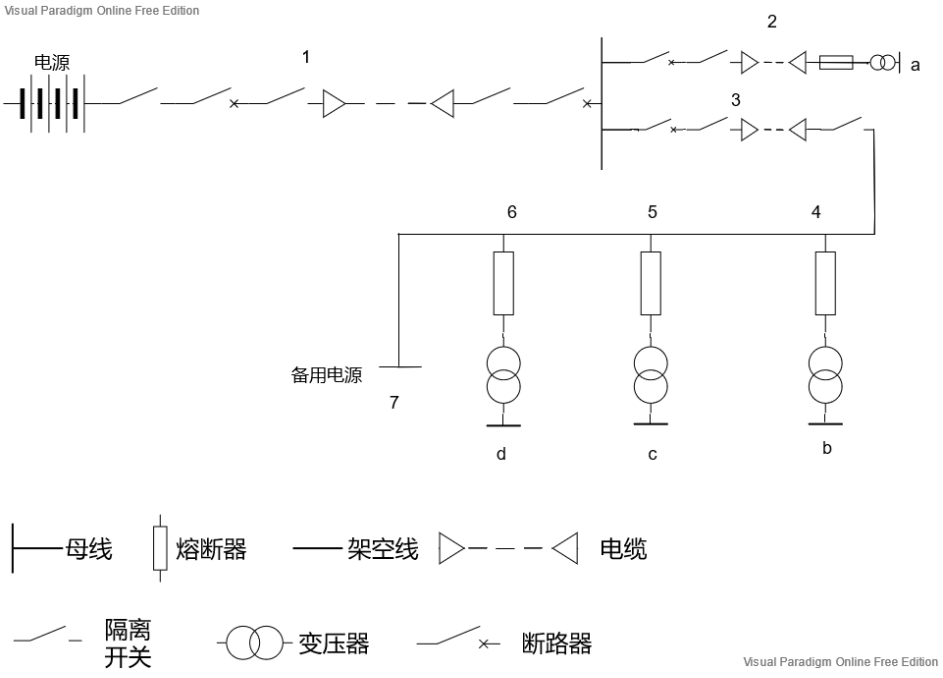


图 2　简单配电网拓扑图

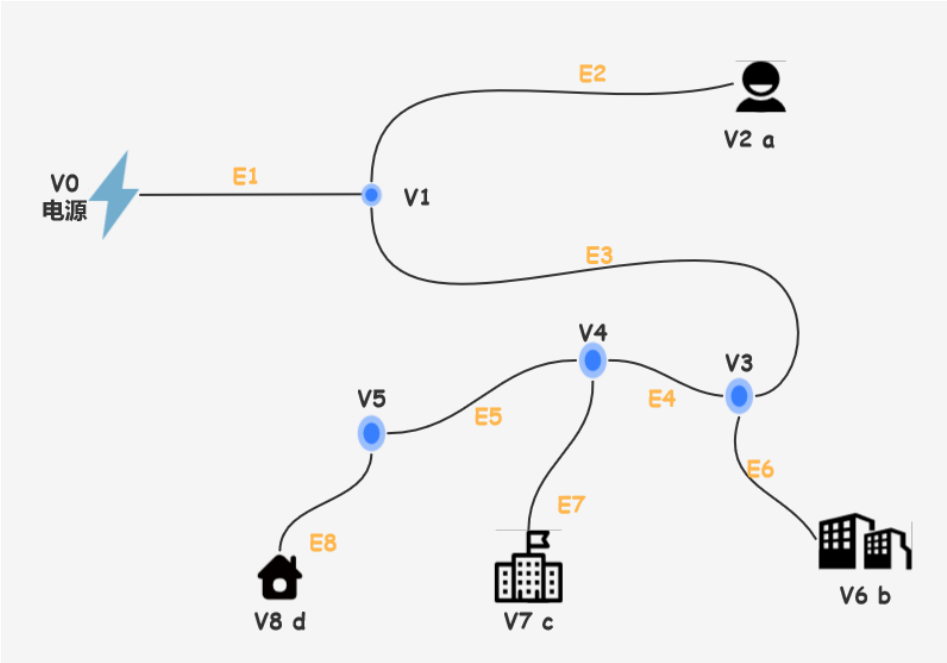


图 3　抽象简化电网图

首先对整个拓扑图，使用 Dijstra 算法，得到各个负荷点到电源 $V0$ 的最短路长度以及路径，结果如表 2。为了简化计算，可以把元件的故障率都等效到边上，作为边的等

表 2　最短路

顶点	最短跳数	路径
V1	1	V0 → V1
V2	2	V0 → V1 → V2
V3	2	V0 → V1 → V3
V4	3	V0 → V1 → V3 → V4
V5	4	V0 → V1 → V3 → V4 → V5
V6	3	V0 → V1 → V3 → V6
V7	4	V0 → V1 → V3 → V4 → V7
V8	5	V0 → V1 → V3 → V4 → V5 → V8

效故障率。简化处理后的边故障信息如表 3(a) 所示。

之后，基于最短路首先进行负荷点可靠性指标评估，结果如表 3(b) 所示。再对系统的可靠性进行评估，结果如表 5。

边	等效故障率	等效故障时间
E1	0.20	2 小时
E2	0.10	5 小时
E3	0.15	5 小时
E4	0.20	5 小时
E5	0.25	3 小时
E6	0.20	3 小时
E7	0.20	2 小时
E8	0.10	1 小时

(a) 线路故障信息

负荷点	平均故障率	平均故障时间
a	0.30	0.90 小时
b	0.55	1.75 小时
c	0.75	2.55 小时
d	0.90	3.00 小时

(b) 负荷点可靠性指标

表 3　拓扑信息

表 4　系统可靠性指标

$ASAI$%	$ASUI$ %	$SAIDI$(h(户 · a))	$SAIFI$(次 (户 · a))
99.9899	0.010073	0.882353	0.264706

最后，基于不同的时空拓扑结构进行灵敏度的定性分析，如图 4 的拓扑结构，配电网只有一条主干线路，多条分支线路，没有分段开关，所以可以对 8 台配电变压器进行灵敏度分析，基于拓扑结构，给出定性分析如表 5。

表 5　灵敏度分析

影响排序	影响负荷点数量	负荷点编号	解释
1	8	a、b、c、d、f	没有熔断器、断路器或其他保护装置
2	1	g	有断路器, 对其他元件影响弱
3	1	e	有熔断器, 对其他元件影响极微弱
4	1	h	有熔断器和断路器, 只影响本身

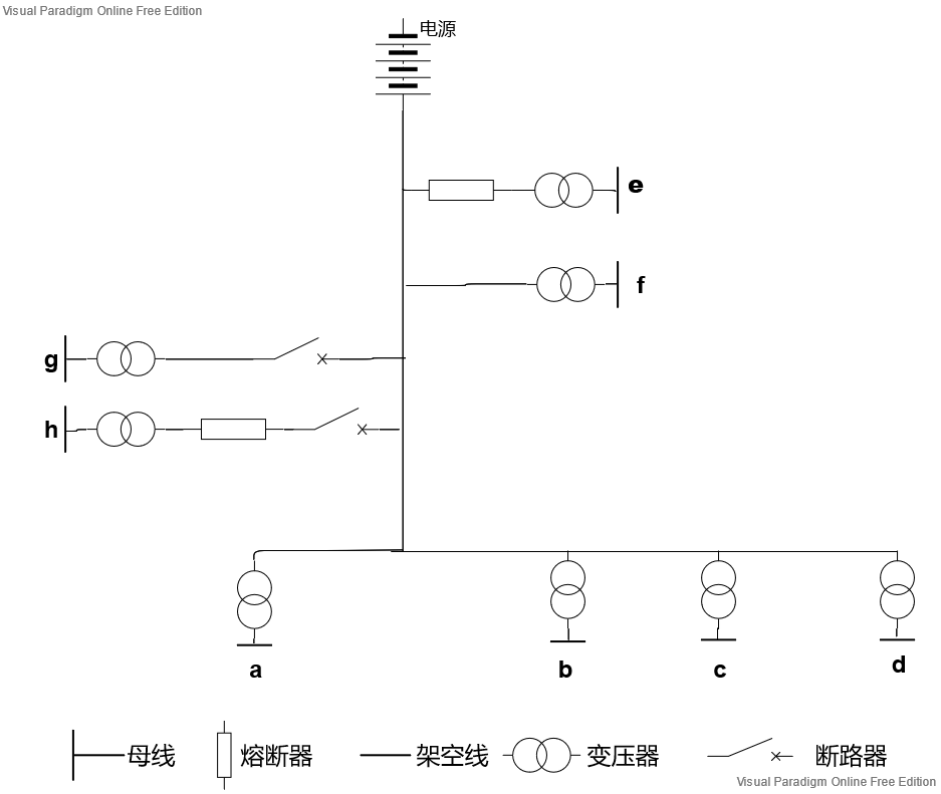


图 4　基于不同元件拓扑的电网图

5.2 问题二

通过收集的配电网中各种警告信息，来逆向推断故障原因。电网中的元件存在一定的拓扑关系，通过拓扑关系我们可以推导出其动作之间的因果关系，并且可以画出简单因果网络模型[?]，来描述电网中各元件之间的物理拓扑关系以及错误警告的依赖关系。通过警告信息，推断出可能存在错误的节点，结合实际问题分析对可能存在问题的节点进行筛选，最终得到故障节点。

5.2.1 建立规则矩阵

依据 $R[i,j]=1$, 表明 P_j 是 P_i 发生的因，P_i 是 P_j 发生的后果

$$R[i,j]=\begin{cases}1, & P_j \Rightarrow P_i \\ 0, & else\end{cases} \tag{8}$$

根据上述定义，建立网络的规则矩阵

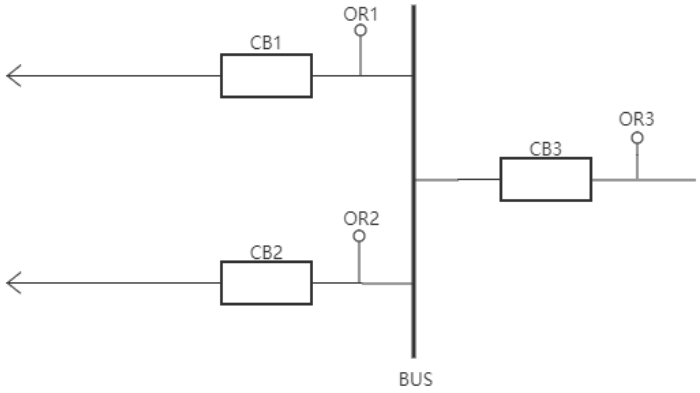


图 5　简单配电网络

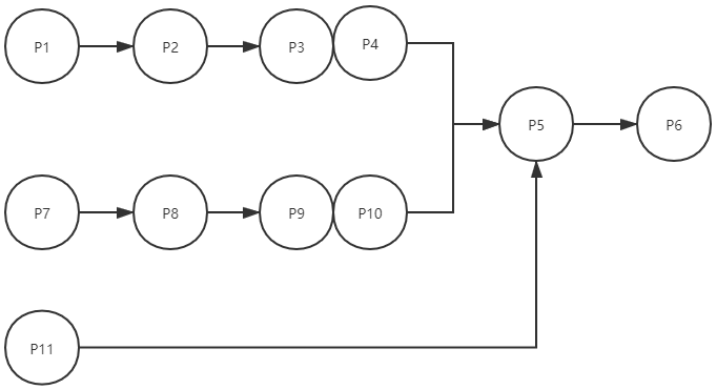


图 6　因果关系网络

表 6　简单配电网络模型节点含义表

节点	含义	节点	含义
P_1	线路 L_1 故障	P_7	线路 L_2 故障
P_2	保护 OR_1 动作	P_8	保护 OR_2 动作
P_3	断路器 CB_1 跳闸	P_9	断路器 CB_2 跳闸
P_4	OR_1 动作 CB_1 拒动	P_{10}	OR_2 动作 CB_2 拒动
P_5	保护 OR_3 动作	P_{11}	母线 BUS 故障
P_6	断路器 CB_3 跳闸		

$$
R = \begin{array}{c} \\ P_1 \\ P_2 \\ P_3 \\ P_4 \\ P_5 \\ P_6 \\ P_7 \\ P_8 \\ P_9 \\ P_{10} \\ P_{11} \end{array}
\begin{array}{c}
\begin{array}{ccccccccccc} P_1 & P_2 & P_3 & P_4 & P_5 & P_6 & P_7 & P_8 & P_9 & P_{10} & P_{11} \end{array} \\
\begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{pmatrix}
\end{array}
$$

5.2.2 建立输入向量

警告信息向量 T：反映了故障发生时，所有保护动作和断路器的动作信息。当某一结点 P_i 发生动作，系统应收到对应的警告信息，则 T(i) = 1，否则 T(i) = 0。

$$
T(i) = \begin{cases} 1, & P_i\text{为真} \\ 0, & \text{否则} \end{cases} \tag{9}
$$

可能故障元件向量 F：依据实际情况，反映了可能存在故障的位置，用来缩小故障范围。当 P_i 是可能存在故障的元件时 F(i) = 1，否则 F(i) = 0。

$$
F(i) = \begin{cases} 1, & P_i\text{属于故障节点} \\ 0, & \text{否则} \end{cases} \tag{10}
$$

5.2.3 计算故障向量 T' 和结果向量 F'

在规则矩阵 R 中 $R[i,j] = 1$，表明 P_j 是 P_i 的因，P_j 是 P_i 的果，对 R 进行转置，则矩阵 R^T 的第 j 行中所有为 1 的元素，表示以 P_j 为错误的因所能引发的结果。依据输入的 T，T 中反应了电网故障的结果。R^T 的第 j 行与 T 对应相乘，若结果为 1，则证明，可能存在 P_j 故障。

通过 R^T 与 T 进行逻辑乘，得到 T'，即反映了所有错误信息的可能原因。将 T' 与之前定义的 F 进行对应位置的与运算，筛选错误原因，即可得到电网的故障结果向量 F'。

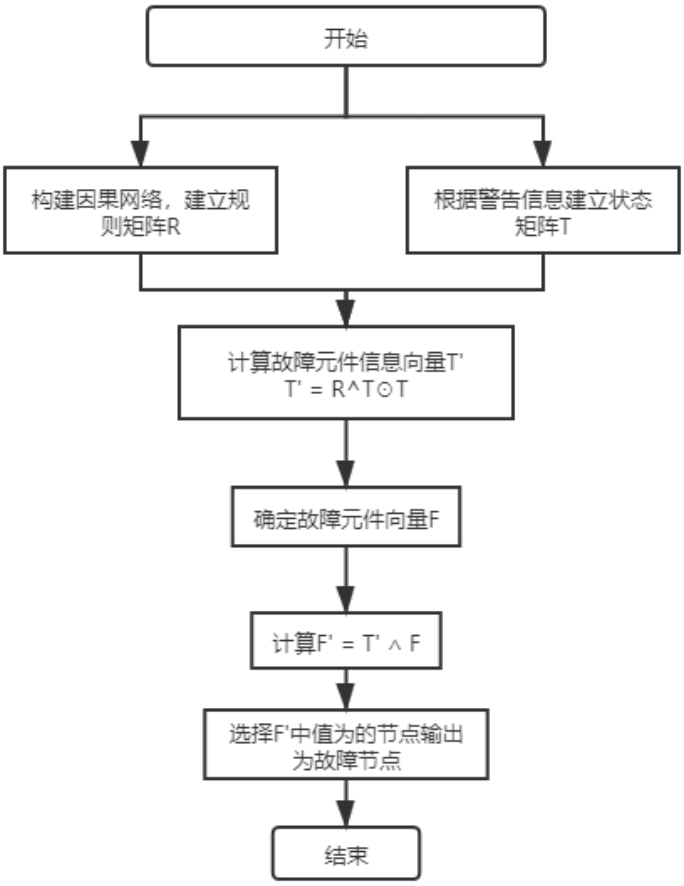


图 7　算法流程

5.2.4 算法举例

简单的配电系统录像图如图**??**所示。该线路继电保护系统由三段式电流保护 OR, 和断路器 CB 组成。假设 L_1 上存在故障，组保护 OR_1 将动作使断路器 CB_1 跳闸。若此时断路器拒动，则后备保护 OR_3 将动作并使断路器 CB_3 跳闸从而将故障隔离。依据三个基本节点的定义，故障元件节点、保护节点、断路器动作节点数量分别为 3 个，3 个，5 个。通过上述过程，可以建立如图**??**所示电网因果网络模型和如表**??**所示规则矩阵 R

故障时，各设备动作如下：

1. 保护动作：OR_2, OR_3
2. 断路器跳闸：CB_3
3. 断路器拒动：CB_2

依据故障时各元件动作，易知 P_5, P_6, P_8, P_{10} 为真, 则 $T = [0\,0\,0\,0\,1\,1\,0\,1\,0\,1\,0]$, 由实际警告信息得 $F = [0\,1\,0\,0\,0\,0\,0\,1\,0\,0\,0\,1]$, 进行逆向推理计算得 $T' = R^T \bigodot T$，通过 $F' = T' \bigwedge F$ 进行筛选得 $F' = [0\,0\,0\,0\,0\,0\,0\,1\,0\,0\,0\,1]$。有 F' 可知，故障元件可能为 L_1 或 BUS

5.2.5 配电网信息差异故障检测差异

基于上述例子中的规则矩阵，当电网的警告信息出现差异，即 CB_2 的跳闸结果丢失时, 各设备动作如下：

1. 保护动作：OR_2, OR_3
2. 断路器跳闸：CB_3

得 $T = [00001101000], F = [010000010001]$, 依据算法，计算出最终错误信息向量 $F' = [00000010001]$。由 F' 可知，故障元件可能为 L_2 和 BUS , 检测结果依然正确

基于上述例子中的规则矩阵，当电网的警告信息出现差异，即 OR_2 的保护信息丢失时, 各设备动作如下：

1. 保护动作：OR_3
2. 断路器跳闸：CB_3
3. 断路器拒动：CB_2

得 $T = [00001100010], F = [010000010001]$, 依据算法，计算出最终错误信息向量 $F' = [00000000001]$。由 F' 可知，故障元件可能为 BUS , 检测结果出现偏差

5.3 问题三

基于动态模式以及时空角度的拓扑结构，自然需要软自愈方案来实现配电网的故障方案解决，并且提前判断故障征兆，从而实现稳定供电；此外，通过软自愈，能够通过实时检测，优化负荷分配，实现负荷转移。

5.3.1 建立抽象拓扑结构

将配电网中的联络开关看作无向边 e，从而得到所有边的集合 E；根据开关的开闭状态将 E 分为 $E1$ 以及 $E2$，分别表示处于开状态的开关以及闭状态的边，两个集合的元素可以动态转化。在进行故障判断、故障预知以及负荷调整时，给予边 e 一个权值 $Cost_m(I, t)$，表示 t 时刻某边 e_m 的电流值为 I，从而实现基于连续时空拓扑信息 + 电流的判断。之后将配电网的开关之间的馈线看作顶点，组成顶点集 V，得到 $G = (V, E)$ 的无向图。

对于题目中图二进行抽象拓扑得到的图，如图 8 所示:

5.3.2 故障判断

需要通过电流值变化判断故障点, 所以我们根据中压配电网的常规电流值给出故障电流阈值 I_{bound} 以及优化电流阈值 $I_{PreBound}$。如果超过 I_{bound} 则说明该开关附近已经发

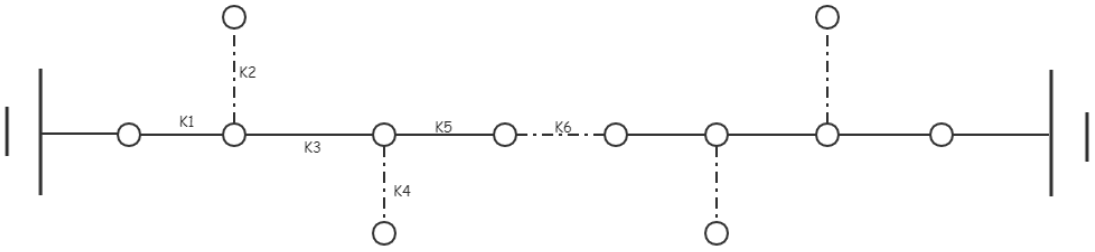


图 8　抽象化样例图

生了线路故障，造成了电流异常; 如果超过 $I_{PreBound}$ 则说明附近线路负载过大，需要进行负载平衡优化。根据中压配电网特性设置[?] $I_{bound} = 800A, I_{PreBound} = 600A$, 从而根据阈值即可判断出异常的开关，而开关不会对电流产生影响，如果有故障馈线，则开关两侧的线路可能存在故障点，即可以得出结论:

$$\text{若馈线存在故障} \iff \text{馈线所直连的所有开关电流值都异常}$$

所以我们根据图 G 异常电流的边，自然确定出公共点，从而定位到出问题的馈线位置。

5.3.3 故障隔离

实现故障判断后，进行开关动作进而完成故障隔离。实现故障隔离要关闭开关。容易得出结论：关闭故障点所邻接的开关使得故障范围最小。基于此，对于图 G 只需要把出问题的点 J 所有的边 e_j 都从 $E1$ 删除转移到 $E2$ 中，即可实现故障点的孤岛化处理以及故障隔离。

5.3.4 恢复供电

因为故障隔离的开关动作会导致除故障点之外的孤岛产生，判断方法是:

$$\text{连通分支数目} > \text{变电站个数} + \text{故障点个数} \iff \text{存在额外无故障段孤岛}$$

恢复供电就是要将此类节点并入无故障的电网，从而变成正常的电网。通过使用 BFS 算法以及标号的思想，将每次从电源点 BFS 到的所有点都标一个相同的号，从而得出所有的连通分支; 进而遍历孤岛节点的边，选择添加未合上并且能够并入带有电源的连通分支的开关 (边)，即实现了恢复供电! 题目所给的例图的解决方案如图 9:

5.3.5 优化及负载均衡

由于软自愈系统的计算能力远超过硬自愈系统，所以除了电网故障自愈还能实现线路正常状态下的运行优化，即负载均衡。当一部分供电网络负载过大，很可能出现故

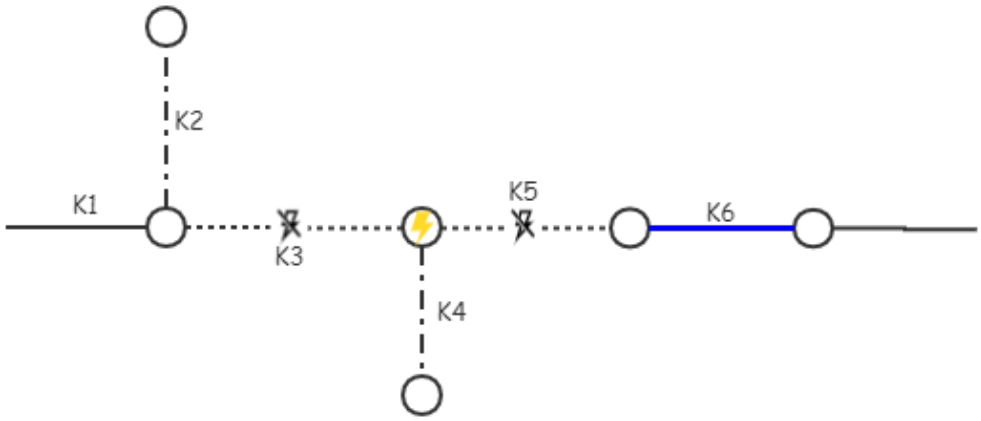


图 9　抽象化样例图

障，所以把负载过大的地区转移到负载较小的网络，就是实现负载均衡。

因为配电网电压恒定，所以功率 $P \propto I$，故仍可以通过电流过大判断是否要进行负载调整。根据故障判断中的 $I_{PreBound}$ 使用同样的算法可以定位到负载较大的馈线段。之后，通过 BFS 从需要优化的节点开始扩展到最远叶节点，然后对此叶节点进行负荷转移，即通过开关的状态变化更换供电站。

5.3.6 成本差异估计

”硬自愈” 是依靠专用电子设备装置接受系统的实时监测数据，通过自动化系统 (硬件)，从而实现线路开关的控制完成故障自愈。

软自愈是通过计算机系统接受实时监测数据，通过软件算法完成线路故障控制的推测，然后再发出开关动作指令。还能够实现线路优化、负载均衡、提前预测等功能。

成本差异体现在：

1. 设备价格：硬自愈需要更多的成本去铺设基础设施，软自愈则可以通过仿真计算减少成本。
2. 维护成本：硬自愈需要日常维护硬件设施的完整程度，才能保证运行正常，而软自愈则不需要。
3. 用户体验：

六、 模型评价与推广

6.1 模型的综合评价

问题一: 指标清晰，对于可靠性评估目标进行了具体的量化分析; 所有问题都基于了配电网拓扑信息，并且可以通过等值法简化配电网，简化了问题，便于统一模型。从

PDO 角度出发，关注一段时间之内的配电网可靠性评估。

问题二：由于多种故障情形都可能得到相同得警告信息。例如在计算举例中，当 L1 故障，OR1 执行保护动作，CB1 拒跳，导致 OR3 执行保护动作，CB3 跳闸。当 L1 与 BUS 同时发生故障时，可能会得到相同的警告信息，导致计算的故障结果存在冗余。

问题三：此模型能根据开关的电流值，根据电流值是否超出阈值，判断电网中是否发生跳闸故障。并且根据电网中的负载量，自动重构电网，以降低故障发生的可能性。再电网故障判断的过程中，依赖于各个开关的电流输入，以判断开关是否跳闸，对于开关据动等情况没有充分的考虑。对于电网重构优化，没有过考虑对于负载最优的重新分配。当电网中的某一条线路中的过大的负载，自动将其分配到其他的新的线路中，但未分配结果的最优性。

6.2 模型的改进

问题一：还需要对具体不同元件等效到最小路的方法进行区分，才能更精确地评估出可靠性。

问题二：由于在同一通路上，两点同时发生故障的概率较低，依据故障点的拓扑信息，进行溯源，得到最源头的故障原因，减少结果的冗余。

问题三：根据周围多个监测点的电流值，充分考虑开关据动和元件损坏等情况，提高系统的可靠性，降低数据变量的灵敏度。

七、 参考文献

参考文献

[1] 文向南. 基于因果网络的配电网故障诊断研究 [D]. 青岛大学,2017.

[2] 赵华, 王主丁, 谢开贵, 李文沅. 中压配电网可靠性评估方法的比较研究 [J]. 电网技术,2013,37(11):3295-3302.

[3] 别朝红, 王秀丽, 王锡凡. 复杂配电系统的可靠性评估 [J]. 西安交通大学学报,2000(08):9-13.

[4] Billintoo R, Wang P. Reliability-network-equivalent approach to distribution-system-reliability evalation [J].IEEE Proc,1998,145(2):149 153.

[5] 高亚静, 林琳, 刘建鹏. 油田配电网的可靠性与灵敏度分析 [J]. 电力科学与工程,2013,29(12):36-40.

[6] http://www.abcbxw.com/news/8139.html 黔西南州人民政府网—兴义供电局 2018 年第一批 10kv 线路

八、 附录

问题一: 计算 C++ 程序:

```
#include<bits/stdc++.h>
using namespace std;

/*假设一般规模的配电网图*/
const int N=100; int n;const int INF=100000;
int p[N][N],d[N],path[N];//path数组用于输出路径
map<int,string> road; //记录路径


//Input datas:
/*-------------------------------------------------*/
float probably[N][N];//等效故障率
float Outage_durations[N][N];//等效每次故障平均停电持续时间
float users[N];//负荷点的用户数(K)
bool Lps[N];//是不是负荷点
/*-------------------------------------------------*/


//Output datas:
/*-------------------------------------------------*/
    //负荷点:
    float Lp_break_proba[N];//负荷点故障率
    float Lp_break_times[N];//负荷点故障持续时间

    //系统平均停电持续时间(SAIDI) 平均停电频率(SAIFI)
    //系统平均供电可用率(ASAI)  平均供电不可用率(ASUI)
    float SAIDI=0,SAIFI=0,ASAI = 0,ASUI = 0;
/*-------------------------------------------------*/
```

```
//最短路程序
void dijkstra(int sec,int n)//sec为出发节点，n表示图中节点总数
{
    int i,j,min,min_num;
    int vis[N]={0,};
    for(i=0;i<n;i++)
    {
        d[i]=p[sec][i];
    }
    vis[sec]=1;d[sec]=0;
    for(i=1;i<n;i++){
        min=INF;
        for(j=0;j<n;j++)
        {
            if(!vis[j]&&d[j]<min)
            {
                min=d[j];
                min_num=j;
            }
        }
        vis[min_num]=1;
        for(j=0;j<n;j++)
        {
            if(d[j]>min+p[min_num][j])
            {
                path[j]=min_num;
                d[j]=min+p[min_num][j];
            }
        }
    }
}

//打印路径
void print(int sec,int n)//sec为出发节点，n表示图中节点总数
```

```
{
    int i,j;
    //由于记录的中途节点是倒序的，所以使用栈，获得正序
    stack<int> q;
    for(i=1;i<n;i++)
    {//打印从出发节点到各节点的最短距离和经过的路径
        j=i;
        while(path[j]!=-1)          //如果j有中途节点
        {
            q.push(j);              //将j压入堆
            j=path[j];              //将j的前个中途节点赋给j
        }
        q.push(j);
        printf("%d=>%d, length:%d, path: %d ",sec,i,d[i],sec);
        while(!q.empty())           //先进后出，获得正序
        {
            printf("%d ",q.top());//打印堆的头节点
            q.pop();                //将堆的头节点弹出
        }
        printf("\n");
    }
}

void calculate_lp(){
    //计算 各负荷点的可靠性指标

    int pre=0,now;
    for(int i =1;i<=n;i++){
        for(auto v:road[i]){
            if(v == ' ' ) continue;
            else if(v == '0') continue;
            else {
                now = v - '0';
                Lp_break_proba[i] += probably[pre][now];
                Lp_break_times[i] += probably[pre][now]*
```

```
                                        Outage_durations[pre][now];
                pre = now;
            }
        }
        pre = 0;
    }

    for(int i =1;i<=N;i++){
        if(Lps[i]) {
            cout<<"V"<<i<<"负荷点平均故障率: ";
            printf("%.2f",Lp_break_proba[i]);
            cout<<" 平均故障时间: ";
            printf("%.2f\n",Lp_break_times[i]);
        }
    }
}

void calculate_sys(){
    //计算 系统的的可靠性指标

    float users_num = 0;
    for(int i=1;i<=N;i++){
        if(Lps[i]){
            users_num += users[i];
            SAIDI = Lp_break_times[i]*users[i];
            SAIFI = Lp_break_proba[i]*users[i];
        }
    }

    ASAI = 1 - (SAIDI/(users_num*8760));
    ASUI = 1 -ASAI;
    SAIDI = SAIDI / users_num;
    SAIFI = SAIFI / users_num;
    cout<<"ASAI /%: "<<ASAI*100<<"\n";
    cout<<"ASUI /%: "<<ASUI*100<<"\n";
```

```
    cout<<"SAIDI/(h/(户·a)): "<<SAIDI<<"\n";
    cout<<"SAIFI/(次/(户·a)): "<<SAIFI<<"\n";
}

int main()
{
    memset(path,-1,sizeof(path));//将path数组初始化为-1
    int i,j; n=9;//点个数
    for(i=0;i<n;i++){//初始化
        for(j=0;j<n;j++){
            p[i][j]=(i==j?0:INF);
        }
    }

//Graph:
    //V0 是电源点  V2 V6 V7 V8是负荷点a b c d
    Lps[2]=Lps[6]=Lps[7]=Lps[8]=true;
    //其余是 主馈线和分支线的交点 用来分隔电路
    p[0][1]=1;p[1][2]=1;p[1][3]=1;p[3][4]=1;p[4][5]=1;
    p[3][6]=1;p[4][7]=1;p[5][8]=1;
    dijkstra(0,n); //求从节点0出发到各节点的最短距离
    print(0,n);    //打印从节点0出发到各节点的最短距离和路径


//Input datas:
    //假设的供电线的可靠性参数
    //等效故障率:
    probably[0][1]=0.2;probably[1][2]=0.1;probably[1][3]=0.15;
    probably[3][4]=0.2;probably[4][5]=0.25;probably[3][6]=0.2;
    probably[4][7]=0.2;probably[5][8]=0.1;
    //故障停电时间
    Outage_durations[0][1]=2;Outage_durations[1][2]=5;
    Outage_durations[1][3]=5;Outage_durations[3][4]=5;
    Outage_durations[4][5]=3;Outage_durations[3][6]=3;
    Outage_durations[4][7]=2;Outage_durations[5][8]=1;
```

```
    //用户数
    users[2] = 3;users[6]= 4;users[7]=5;users[8]=5;
    //已知最短路的情况下进行计算:
    road[1] = "0 1";          road[2] = "0 1 2";
    road[3] = "0 1 3";        road[4] = "0 1 3 4";
    road[5] = "0 1 3 4 5";    road[6] = "0 1 3 6";
    road[7] = "0 1 3 4 7";    road[8] = "0 1 3 4 5 8";


//Output datas:
    calculate_lp();
    calculate_sys();

    return 0;
}
```

问题二: 故障检测函数

```
    function [F1] = TopoErrorDetect(R,T,F)
    %Inputs:
        %R: 规则矩阵
        %T: 警告信息向量
        %F: 可能故障元件向量
            %Outputs:
        %F1: 检测结果
    T1 = logical(R' * T');%依据输入信息, 逆推可能各章的元件
    F1 = logical(F' .* T1);%依据实际, 筛选故障元件
    end
```

```
    #include<bits/stdc++.h>
#include<algorithm>
#include"queue"

using namespace std;
struct Edge{
    int u,v,w,nxt;
    bool open;//open代表开关是否打开
```

```
};
class elecNet{
public:
    Edge * Edges;
    int* head;
    int tot;
    int iBound;//电流上界，超出界限则认为发生故障，下一步将进行跳闸.
    int iPreBound;
    vector<int> lable;
    int s1 ;
    int s2;
    int n;
    int m;
    vector<int> brokenNode;
    vector<int> vis;
    vector<int> alwaysClose;
    vector<int> overPre;
    vector<int> overPre_1;
    vector<int> overPre_2;
    vector<int > removeNodes;
    void findRemoveNode();
    void remove();
    elecNet(int n0,int m0,int sa,int sb) {
         n = n0;
         m = m0;
        vis = vector<int>(n+1,0);
        Edges = new Edge[2*m+2];
        head  =  new int [n+1];
        lable = vector<int>(n+1,0);
        lable[sa] = 1;
        lable[sb] = 2;
        s1 = sa;
        s2 = sb;
        tot = 0;
        memset(head , sizeof(head), 0);
```

```
        memset(Edges, sizeof(Edges),0);
        for(int i = 0;i <=2*m+1;i++ )
            Edges[i].nxt = 0;
        for(int i = 0;i <= n;i++)
            head[i] = 0;
        iBound = 10;
        iPreBound = 8;
    }

    void addEdge(int u,int v,bool open);//初始化
    void addAlwaysClose(vector<int> a);
    void inputEdge(vector<int>&);//每一个时刻重新输入电流
    void connect();
    void bfs(int s,int flag);
};
void elecNet:: addEdge(int u,int v,bool open){
    tot++;
    Edges[tot].u = u;
    Edges[tot].v = v;
   // Edges[tot].w = w;
    Edges[tot].open = open;
    Edges[tot].nxt = head[u];
    head[u] = tot;
}

void elecNet::inputEdge(vector<int> & a) {
    vector<int> preBroken;
    overPre_1 =  vector<int>() ;
    overPre_2 =  vector<int>() ;
    brokenNode = vector<int>();
    for(int j = 1;j<= m;j++){

        int i = j*2;
        Edges[i].w = a[j];
        Edges[i-1].w = a[j];
```

```
        if(a[j] > iBound) {//电流
            Edges[i].open = false;//关闭电闸
            Edges[i-1].open = false;//关闭电闸
            cout << "switch " << j <<": close"<<endl;
            if (find(preBroken.begin(), preBroken.end(),
            Edges[i].u) != preBroken.end())
                brokenNode.emplace_back((Edges[i].u));
            else preBroken.emplace_back(Edges[i].u);
            if (find(preBroken.begin(), preBroken.end(),
            Edges[i].v) != preBroken.end())
                brokenNode.emplace_back((Edges[i].v));
            else preBroken.emplace_back(Edges[i].v);
        }
        else if(Edges[i].w > iPreBound){
            if(!Edges[i].open)continue;
           if(find(overPre_1.begin(),overPre_1.end(),
           Edges[i].u) != preBroken.end())
               overPre_2.emplace_back(Edges[i].u);
           else overPre_1.emplace_back(Edges[i].u);
            if(find(overPre_1.begin(),overPre_1.end(),
            Edges[i].v) != preBroken.end())
                overPre_2.emplace_back(Edges[i].v);
            else overPre_1.emplace_back(Edges[i].v);
            for(int k = 0;k < overPre_1.size();k++){
                if(find(overPre_2.begin(),overPre_2.end(),
                overPre_1[k]) == overPre_2.end())
                    overPre.emplace_back(overPre_1[k]);
            }
        }
    }
    for(int i = 0;i < brokenNode.size();i++)
    {  cout <<"broken node : "<<brokenNode[i]<<endl;
        alwaysClose.emplace_back(brokenNode[i]);
    }
    return ;
```

```
}

void elecNet::connect() {


    for(int i = 0;i <= lable.size();i++)
        lable[i]  = 0;
     lable[s1] = 1;
     lable[s2] = 2;
     for(int i = 0; i< vis.size();i++)
        vis[i] = 0;
     bfs(s1, 1);
     bfs(s2, 2);
     list<int> island;
     for(int i=1;i <= n; i++){
        if(!lable[i]) {
            if (find(brokenNode.begin(), brokenNode.end(), i)
             == brokenNode.end())
                island.push_back(i);
        }

     }
     bool flag = true;
     while(flag){
        flag = false;
        for(auto it = island.begin();it != island.end(); it++){
            int u = *it;
            if(find(alwaysClose.begin(),alwaysClose.end(),u)
            != alwaysClose.end()) continue;
            for(int i = head[u];i;i = Edges[i].nxt){
                int v = Edges[i].v;
                if(lable[v]){
                    lable[u] = lable[v];//并入电网
                    Edges[(i+1)/2*2].open = true;//打开电闸
                    Edges[(i+1)/2*2-1].open = true;
```

```
                    cout << "switch " << (i+1)/2 <<": open"
                    <<endl;
                    it = island.erase(it);//从孤岛中删除
                    flag = true;
                    break;
                }

            }
        }
    }
    return ;
}

void elecNet::bfs(int s,int flag) {
    queue<int> q;
    lable[s] = flag;
    q.push(s);
    vis[s] = 1;
    while(!q.empty()){
        int u = q.front();
        q.pop();
        for(int i = head[u];i;i = Edges[i].nxt){
            int v = Edges[i].v;
            if(vis[v])continue;
            if(lable[v] == 0 && Edges[i].open){
                lable[v] = flag;
                q.push(v);
                vis[v] = 1;
            }
        }
    }
    return ;
}

void elecNet::addAlwaysClose(vector<int> a) {
```

```
    alwaysClose = vector<int>(a);
}

void elecNet::findRemoveNode() {
    for(int i = 0;i < overPre.size();i++){
        queue<int > q;
        vector<int> vis(n,0);
        q.push(overPre[i]);
        vis[overPre[i]];
        while(!q.empty()){
            int u = q.front();
            q.pop();
            int cnt = 0;
            for(int e = head[u];e;e = Edges[e].nxt){
                int v = Edges[e].v;
                if(find(overPre_2.begin(),overPre_2.end(),v)
                 != overPre_2.end()) continue;
                if(!Edges[e].open) continue;
                cnt++;
                q.push(v);
            }
            if(cnt == 0){
                removeNodes.emplace_back(u);
                break;
            }

        }

    }
}

void elecNet::remove() {
    for(int i = 0;i < removeNodes.size();i++){
        int u = removeNodes[i];
        int u1 = u;
```

```
        int eClose = 0;
        for(int e = head[u]; e; e = Edges[e].nxt){
            if(Edges[e].open) {
                u1 = Edges[e].v;
                eClose = e;
                continue;
            }
        }
        for(int e = head[u];e;e = Edges[e].nxt){
            if(Edges[e].open) continue;
            int v = Edges[e].v;
            if(lable[v] == lable[u] || lable[v] == 0)
             continue;
            //并到另一个树上
            int j = (eClose+1)/2*2;
            Edges[j].open = false;
            Edges[j-1].open = false;
            cout <<"cut node "<< u << " from "<<lable[u]
            <<" to "<<lable[v]<<endl;
            cout <<"switch "<<j/2 << ": close"<<endl;
            int k = (e+1)/2*2;
            Edges[k].open = true;
            Edges[k-1].open = true;
            cout <<"switch " <<k/2 <<": open"<<endl;
            lable[u] = lable[v];
            break;
        }

    }
}


int main() {
     freopen("a.in","r",stdin);
    freopen("a.out","w",stdout);
```

```
    int n0,m0,s1,s2;
    cin >>n0>>m0>>s1>>s2;
    elecNet en(n0,m0,s1,s2);
    int t;
    cin >>t;
    vector<int> close;
    for(int i = 0; i < t;i ++){
        int tmp;
        cin >> tmp;
        close.emplace_back(tmp);
    }
    en.addAlwaysClose(close);
    for(int i = 1;i <= m0;i++){
        int u,v;
        bool open;
        cin >> u>>v>>open;
        en.addEdge(u,v,open);
        en.addEdge(v,u,open);
    }

    vector<int> a;
    a.emplace_back(0);
    for(int i = 1;i <= m0;i++){
        int w;
        cin >> w;
       a.emplace_back(w);
     //   a.emplace_back(w);
    }
    en.inputEdge(a);
    en.connect();
    en.findRemoveNode();
    en.remove();
    return 0;
}
//input:
```

```
//14  13  1  2
//4  5  7  11  14
//1  3  1
//3  4  1
//4  5  0
//4  6  1
//6  8  1
//6  7  0
//8  9  0
//9  10  1
//10  12  1
//10  11  0
//12  14  0
//12  13  1
//2  13  1
//1
//2
//0
//11
//11
//0
//0
//2
//2
//0
//0
//2
//3
```