Algorithms

Graph Reduction

We call the reduction of the graph, the operation of removing from the graphs all nodes and edges that cannot appear in the minimal path set. We have two different algorithms for graph reduction based on the type of graph. For directed graph it is REDUCE-DIRECTED-GRAPH and for undirected it is REDUCE-UNDIRECTED-GRAPH. Before we reduce the given graph, we connect an artificial source ‘s’ to the source nodes and an artificial terminal ‘t’ to the terminals nodes. In case of directed graph artificial source node ‘s’ only have outgoing edges and artificial terminal node ‘t’ have only incoming nodes.

Algorithm1: REDUCE-DIRCTED-GRAPH(G,s,t)

```
for all v in G.V do
    if card(IN_N(v)) == 0 and n is not s then
        To_delete_in<- To_delete_in U v     // storing all those nodes which cannot be reached
    end if
    if card(OUT_N(v)) == 0 and n is not s then
        To_delete_out<- To_delete_out U v   // storing all those nodes from where we cannot move further
    end if
end for
while To_delete_in is not empty do
    x<-element from to To_delete_in
    To_delete_in<- To_delete_in – x
    for all y in OUT_N(x) do
        if card(IN_N(y)) == 1 then
            To_delete_in<- To_delete_in U y
        end if
    end for
    G<-G/x
end while
while To_delete_out is not empty do
    x<-element from to To_delete_out
    To_delete_out<- To_delete_out – x
    for all y in IN_N(x) do
        if card(OUT_N(y)) == 1 then
            To_delete_out<- To_delete_out U v
        end if
    end for
    G<-G/x
end while
return G
```

Algorithm2: REDUCE-UNDIRCTED-GRAPH(G,s,t)

```
for all v in G.V do
    if card(L[v] == 1 and n is not s then
        To_delete<- To_delete U v     // storing all those nodes from where we cannot move further
    end if
end for
while To_delete is not empty do
    x<-element from to To_delete
    To_delete <- To_delete– x
```

```
    for all y in L[x] do
        if card(L[y]) == 1 then
            To_delete <- To_delete U y
        end if
    end for
    G<-G/x
end while
return G
```

Finding paths

Once the graph is reduced, we call the finding path algorithm to compute the minimal path set. We have two different algorithms for finding the path sets, FIND-PATH-DIRECTED and FIND-PATH-UNDIRECTED. Finding paths is called after the graph is reduced. Before finding path in undirected graph we convert it into directed graph and then call FIND-PATH-UNDIRECTED. While finding path in undirected graph we maintain a stack where all the visited nodes are stored.

Algorithm 3: FIND-PATH-DIRECTED(G,S,T)

```
REDUCE-DIRECTED-GRAPH(G,s,t)
l<-[]
PATH(G,S,T)
    for all x in S do
        NEXT_NODE(G,S,T,x,R)
        l<-[]
    end for
NEXT-NODE(G,S,T,n,R)
    if n in T then
        ADD-NODE(n)
        l<- l - n
    else
        ADD-NODE(n)
        for x in OUT_N(n) do
            if x not in S then
                NEXT-NODE(G,S,T,x,R)
            end if
        end for
        l<- l - x
    end if
ADD-NODE(n)
    l<-l U n
```

Algorithm 4: FIND-PATH-UNDIRECTED(G,S,T)

```
REDUCE-UNDIRECTED-GRAPH(G,s,t)
CONVERT-UNDIRECTED(G,s,t)
l<-[]
visited_node<-[]
PATH(G,S,T)
    for all x in S do
        NEXT_NODE(G,S,T,x,R)
        l<-[]
        visited_node<-[]
    end for
```

```
NEXT-NODE(G,S,T,n,R)
    if n in T then
        ADD-NODE(n)
        I<- I – n
        visited_node<- visited_node-n
    else
        ADD-NODE(n)
        for x in OUT_N(n) do
            if x not in visited_node and x not in s then
                NEXT-NODE(G,S,T,x,R)
            end if
        end for
        I<- I – x
        visited_node<-visited_node-x
    end if
ADD-NODE(n)
    I<-I U n
    visited_node<- visited_node U n
```

Converting Graph

We call the converting graph, the operation of converting the undirected graph to directed graph. Converting Graph is called after the undirected graph is reduced.

- Source nodes only have outgoing edges.
- Terminal nodes only have incoming edges.

Algorithm 5: CONVERT-UNDIRECTED-TO-DIRECTED(G,S,T)

```
for x in G.V do
    if x in S then
        for y in L(x) do
            OUT_N(x)<- OUT_N(x) U y
        end for
    else if x in T then
        for y in L(x) do
            IN_N(x)<- IN_N(x) U y
        end for
    else
        for y in G.V do
            if y in T then
                OUT_N(x)<- OUT_N(x) U y
            else if y in S then
                IN_N(x)<- IN_N(x) U y
            else
                IN_N(x)<- IN_N(x) U y
                OUT_N(x)<- OUT_N(x) U y
            end if
    end if
end for
```