Name : Omkar Dixit

GWID : G42396299

Home Work III



**Formulas :**

Cij = mini+1 <= k <= j{Ci,k-1 + Ckj + Wij}   
Wij = SUMjs=i+1 ps + SUMjs=i qs

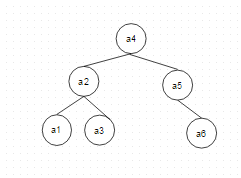
Table for Wij.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| W00 = 0 | W11 = 1/20 | W22 = 1/20 | W33 = 1/20 | W44 = 1/20 | W55 = 1/20 | W66 = 0 |
| W01 = 4/20 | W12 = 3/20 | W23 = 4/20 | W34 = 6/20 | W45 = 5/20 | W56 = 3/20 |
| W02 = 6/20 | W13 = 6/20 | W24 = 9/20 | W35 = 10/20 | W46 = 7/20 |
| W03 = 9/20 | W14 = 11/20 | W25 = 13/20 | W36 = 12/20 |
| W04 = 14/20 | W15 = 15/20 | W26 = 15/20 |
| W05 = 18/20 | W16 = 17/20 |
| W06 = 20/20 |

Table for Cij.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| C00 = 0 | C11 =0 | C22 = 0 | C33 = 0 | C44 = 0 | C55 = 0 | C66 = 0 |
| C01 = 4/20  R01=1 | C12 = 3/20  R12=2 | C23 = 4/20  R23=3 | C34 = 6/20  R34=4 | C45 = 5/20  R45=5 | C56 = 3/20  R56=6 |
| C02 = 9/20  R02=1 | C13 = 9/20  R13= 3 | C24 = 13/20  R24=4 | C35 = 15/20  R35=4 | C46 = 10/20  R46=5 |
| C03 = 17/20  R03=2 | C14 = 20/20  R14=3 | C25 = 22/20  R25=4 | C36 = 21/20  R36=5 |
| C04 = 29/20  R04=3 | C15 = 29/20  R15=4 | C26 = 29/20  R26=4 |
| C05 = 40/20  R05=4 | C16 = 36/20  R16=4 |
| C06 = 47/20  R06=4 |

Optimal Binary Search Tree can be shown as follows: c5 and 6.



2.

2. a.

Notation : as there is no probability, we will assume all the probabilities are same

Notations:

Let Vo will go to a from b. At i-1 it will go to b and it will go to a only if signmai is equal to G(B,a)

Probability of going to a node from b node can be shown as

Pi,a = Pi-1,b \* ((G(b,a)) == signami)

Proof for principle of optimality :

We will prove by contradiction.

Suppose Pi-1,b is not optimal solution that means. There is better path to go to a without going to a. that means

P’(I,a) = P(i-1,b) \* G\*(b,a)) == sigmai ) > Pi,a = Pi-1,b \* ((G(b,a)) == signami) but it can’t happen.

Algorithm :

G is matrix of edges and Vo is node 1.

function find\_right\_sound(G[][],sigma[] ,Vo, output)

Begin

Int p[][];

Int q[][];

For (i=1;I <= n; i++ )

Do

For (j=1;I <= n; i++ )

do

For (k=1;I <= n; i++ )

do

p[k][i]=max(p(i-1,b)\*G[k][i]) //if G[k][i] is equal to sigma[i]

output [index]=k;

index++;

done

return output;

done

find\_right\_sound end;

b)

Here p[I,j] is included it will return following :

function find\_right\_sound(G[][],sigma[] ,Vo, output)

Begin

Int p[][];

Int q[][];

For (i=1;I <= n; i++ )

Do

For (j=1;I <= n; i++ )

do

For (k=1;I <= n; i++ )

do

p[k][i]=max(p(i-1,b)\*G[k][i]) //if G[k][i] is equal to sigma[i]

output [index]=k;

index++;

done

return output;

done

find\_right\_sound end;

3.

:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Points | 1 | 2 | 3 | 4 | 5 |
| 1 | 0 | infinity | 1 | 4 | 4 |
| 2 | infinity | 0 | infinity | 6 | 2 |
| 3 | 1 | infinity | 0 | infinity | 1 |
| 4 | 4 | 6 | infinity | 0 | 10 |
| 5 | 4 | 2 | 1 | 10 | 0 |

:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Points | 1 | 2 | 3 | 4 | 5 |
| 1 | 0 | infinity | 1 | 4 | 4 |
| 2 | infinity | 0 | infinity | 6 | 2 |
| 3 | 1 | infinity | 0 | 5 | 1 |
| 4 | 4 | 6 | 5 | 0 | 8 |
| 5 | 4 | 2 | 1 | 8 | 0 |

:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Points | 1 | 2 | 3 | 4 | 5 |
| 1 | 0 | infinity | 1 | 4 | 4 |
| 2 | infinity | 0 | infinity | 6 | 2 |
| 3 | 1 | infinity | 0 | 5 | 1 |
| 4 | 4 | 6 | 5 | 0 | 8 |
| 5 | 4 | 2 | 1 | 8 | 0 |

:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Points | 1 | 2 | 3 | 4 | 5 |
| 1 | 0 | infinity | 1 | 4 | 2 |
| 2 | infinity | 0 | infinity | 6 | 2 |
| 3 | 1 | infinity | 0 | 5 | 1 |
| 4 | 4 | 6 | 5 | 0 | 6 |
| 5 | 2 | 2 | 1 | 6 | 0 |

:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Points | 1 | 2 | 3 | 4 | 5 |
| 1 | 0 | 10 | 1 | 4 | 2 |
| 2 | 10 | 0 | 11 | 6 | 2 |
| 3 | 1 | 11 | 0 | 5 | 1 |
| 4 | 4 | 6 | 5 | 0 | 6 |
| 5 | 2 | 2 | 1 | 6 | 0 |

:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Points | 1 | 2 | 3 | 4 | 5 |
| 1 | 0 | 4 | 1 | 4 | 2 |
| 2 | 4 | 0 | 3 | 6 | 2 |
| 3 | 1 | 3 | 0 | 5 | 1 |
| 4 | 4 | 6 | 5 | 0 | 6 |
| 5 | 2 | 2 | 1 | 6 | 0 |

4. a)

Logic:

Assumption :

If graph is not connected and Graph can be viewed as a spanning tree and spanning tree is a binary tree then graph can be viewed as binary tree.

We will modify bfs algorithm as follows :

1. Check whether graph is cyclic or not if it is cyclic then rerutn false.
2. Init counter to 0.
3. Start BFS with minimum element.
4. Before de queue will check counter. (if it is 0 or 2, proceed or return that graph is not binary tree.
5. After every de- queue operation, reset counter.
6. Increment counter after visiting unvisited node.
7. Return that graph is binary tree at the end. (If program reaches end, it has checked all the nodes)
8. If number of nodes == in tree then

**** Procedure BFS(input: graph G;output)

Begin

For every edge of graph // check cycle

If edge makes graph then return -1;

End;

Integer counter=0;

Queue Q;

Integer s,x;

while (G has an unvisited node) do

s := an unvisited node;

visit(s);

Enqueue(s,Q);

While (Q is not empty) do

If(counter = 0 or 2)then

x := Dequeue(Q);

counter=0;

For (unvisited neighbor y of x) do

visit(y);

Enqueue(y,Q);

Counter++;

Endfor

Else

Output = false

Return -1;

End if;

endwhile

endwhile

output=true;

return 0;

end

Time Complexity :

Check cycles : edge e=(x,y) creates a cycle if both x and y belong to the same tree in the forest.

It is same as of BFS.

*  Every node is visited onece. Also, every edge (x,y) is "crossed" once when node y is checked from x to see if it is visited (if not visited, then y would be visited from x).
* Therefore, the time of BFS is O(n+|E|) + O(|E|log n)

4b)

Logic :

Will group elements I n BFS as follows :

If both nodes(I,j) are unvisited then

Put I in group 1 and j in group 2.

Change group for next edge ie if j is in group 2, for edge k (j,k) , k should go in group 1.

 Procedure check\_bipartite(input: graph G)

begin

Queue Q;

Integer s,x;

Integer val[];

while (G has an unvisited node) do

s := an unvisited node;

val[s]= 0; // as it is unvisited

visit(s);

Enqueue(s,Q);

While (Q is not empty) do

x := Dequeue(Q);

For (all neighbors y of x) do

If (y is visited ) then ;

If val[y] = val[x] then // both are in same group

Return false;

Else // put unvisited node in 0 if current node is in 1

// put unvisited node in 1 if current node is in 0

If(val [x] = 0) then val[y]=1;

If(val [x] = 1) then val[y]=0;

Enqueue(y,Q);

endfor

endwhile

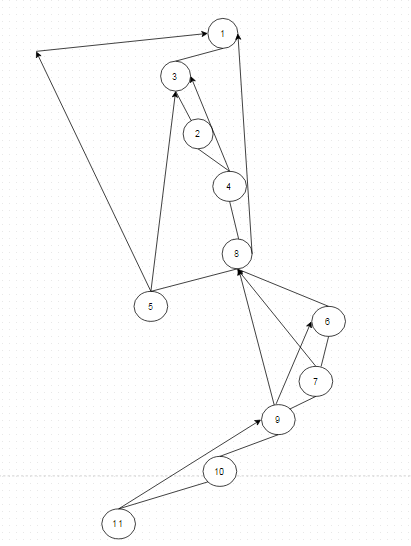
endwhile

end

5.

Spanning tree after applying bi connectivity algorithm

1. Using DFS,



Note : Backtracking paths are shown as



And paths for spanning tree are shown as :



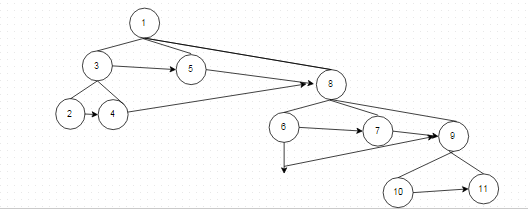
|  |  |  |  |
| --- | --- | --- | --- |
| Node | DFN | L[i] | Articulation point or no |
| 1 | 1 | 1 | No |
| 2 | 3 | 1 | No |
| 3 | 2 | 1 | No |
| 4 | 4 | 1 | No |
| 5 | 6 | 1 | No |
| 6 | 7 | 5 | No |
| 7 | 8 | 5 | No |
| 8 | 5 | 1 | Yes |
| 9 | 9 | 5 | Yes |
| 10 | 10 | 9 | No |
| 11 | 11 | 9 | No |

1. As

**L[y] >= DFN[x]**

Point 8 and point 9 are articulation points.

1. BFS:



Distances :

(1,3 1,5 1,8) = 1

(1,2, 1,4 1,6 1,7 1,9) =2

(1,10 1,11) =3