Name: Sindhuja Yerramalla
UUID: U00839259 / syrrmlla@memphis.edu.

HomeWork 2 - 7712-001

Problem-1 :- 3.24 from DPV

Steps for the Algorithm:

1. Run DFS on the given DAC graph G.
2. Find and remove the source vertex from G [Source is a vertex with highest post number]
3. Create a list and add the removed source to it.
4. Now find the next source vertex in the resulting graph again remove that second source vertex and add to the list and continue this process until all the vertices are processed.
5. The list is the linearized order of the graph. The linear time is O(|V| + |E|)

Algorithm

```
dfs(G)
create list L = [ ]
create list L' = [ ]
add source vertices found by DFS to L
while L is not empty:
    remove v from L
    add v to L'
    reduce the indegree of neighbour of v by 1
    find secondary source(v) whose indegree is now 0
    add that vertex to L
for each consecutive vertices i, i+1 in L':
    if there is no edge (i, i+1) in G:
        return False
return true.
```

Problem 2 :- 3.27 from DPV.

* Suppose there are $v_1, v_2, \ldots v_n$ number of odd degree vertices and any number of even-degree vertices in our Graph G
* Our Graph G is converted to G' in which every vertex is even-degree. This can be done by adding $n/2$ new vertices to G.
* These $n/2$ new vertices have a degree of 2 and connecting them to n odd-degree vertices makes all the vertices to even-degree.
* G' has all the vertices with even degree & G' contains a Eulerian cycle, This cycle visits each new vertex exactly once using the new edges. If the newly added vertices & their edges are removed from the graph G', it results in $n/2$ edge-disjoint paths.

∴ The possible paths obtained by pairing up the vertices of the odd-degree results in edge-disjoint paths.

Problem 3 : 3.28 from DPV

a) set x_1, x_2, x_3, x_4 to true, true, false, True respectively

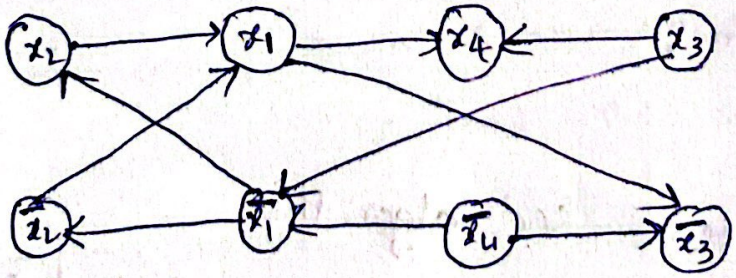
This is only other instance of satisfaction for given 2SAT problem other than true, false, false and True

b) $(x_1 \vee x_2) \wedge (x_1 \vee \bar{x}_2) \wedge (x_3 \vee x_4) \wedge (x_3 \vee \bar{x}_4) \wedge (\bar{x}_1 \vee \bar{x}_3)$

Given an instance I of 2sat with n variables & m clauses, Construct a directed graph $G_1 = (V, E)$ as follows

- G_1 has 2n nodes, one for each variable & its negation.
- G_1 has 2m edges : for each clause $(\alpha \vee \beta)$ of I. (where α, β are literals). G_1 has an edge from the negation of α to β, and one from the negation of β to α.

c)



d) If x and $\bar{x}$ are in the same SCC could we get $(x \Rightarrow \bar{x})$ and $(\bar{x} \Rightarrow x)$, that is $x = \bar{x}$.

e) Assign values to variables as follows.

* Repeatedly pick a sink SCC of G_I
* Assign value true to all literals in the sink, assign false to their negations and delete all of them.

Notice that an implication $(\bar{\alpha} \Rightarrow \beta)$ is always satisfied if literal β is true. Thus setting the literals L in the sink SCC to true ensures that all implications within this SCC are satisfied.

By the symmetry of edges in G_I for each edge in the sink SCC, there is another edge $(\bar{\beta} \Rightarrow \alpha)$ elsewhere. Thus the negations of the literals in the sink SCC form a source SCC.

Likewise, $(\bar{\beta} \Rightarrow \alpha)$ is always satisfied if $\bar{\beta}$ is false.

f) linear time algorithm:

- Build G_I.
- Run the linear-time SCC algorithm.
- For each variable x, check whether x and $\bar{x}$ are in the same SCC.

Problem 4: 3.31 from DPV

Let $G=(V,E)$ be an undirected graph.

a) Let e_1 & e_2 are edges of one cycle in G (C_1)
Let e_2 & e_3 are edges of one cycle (C_2) in G

As $e \sim e$, if $e = e$
$\therefore e_1 \sim e_1$, as $e_1 = e_1$

As $e_1 \sim e_2$, if $e_1 = e_2$
$\therefore e_1 \sim e_2$, as $e_1 = e_2$

so, '$\sim$' is reflexive relation

As $e \sim e'$ if e, e' are edges of one cycle.
we can say $e_1 \sim e_2$ and also $e_2 \sim e_1$ as e_1 and e_2 are edges of a cycle in undirected graph.

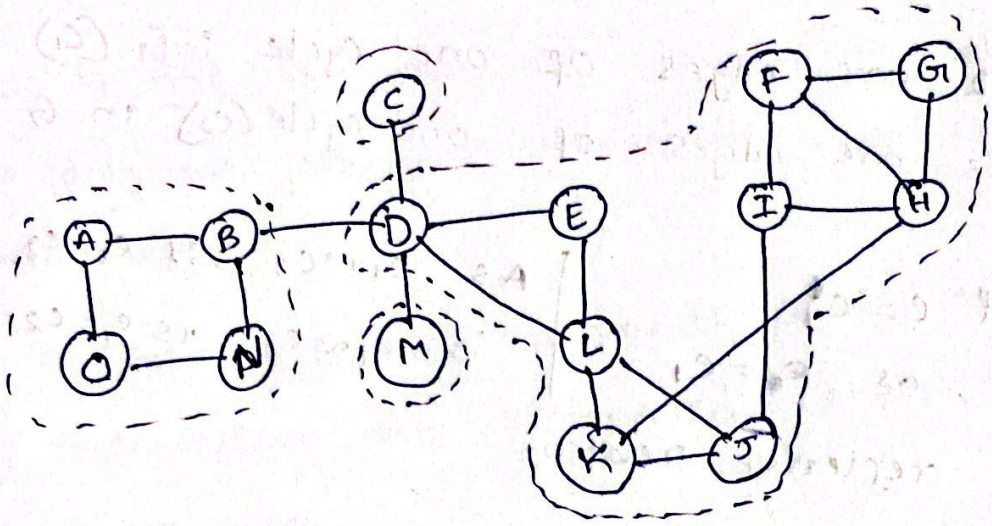
$\therefore$ '$\sim$' is a symmetric as $e_1 \sim e_2$ and $e_2 \sim e_1$

As e_2 is a edge in two cycles, we can say two cycles are having common edge e_2. So we can form one big cycle C_3, which contains edges e_1 & e_3 by removing common edge e_2.

C_1 contains e_1 & e_2 $\therefore e_1 \sim e_2$
e_2 contains e_2 & e_3 $\therefore e_2 \sim e_3$
e_3 contains e_1 & e_3 $\therefore e_1 \sim e_3$

$\Rightarrow$ '$\sim$' is a transitive relation

'$\sim$' is a reflexive, symmetric & transitive relation, it is equivalence relation.



b) Bridges : {BD, CD, DM}

Seperating vertex : {B, D, L}

c) Lets say there are two vertices in common in two biconnected components (C_1, C_2).

As one vertex of biconnected component is reachable to any other vertex of that biconnected component. There must be an edge between V_1 & V_2 because if V_1, V_2 don't share an edge, then e_1 & C_2 should be one biconnected component as the largest possible connected graph should be biconnected component.

If V_1 & V_2 doesn't share an edge, all vertices from C_1 should be reachable from every vertex in C_2, so then C_1 & C_2 are not valid biconnected

components, both will merge and make one valid biconnected components

If v_1 and v_2 share an edge, an edge should belong to only one biconnected component, so this case is also not valid.

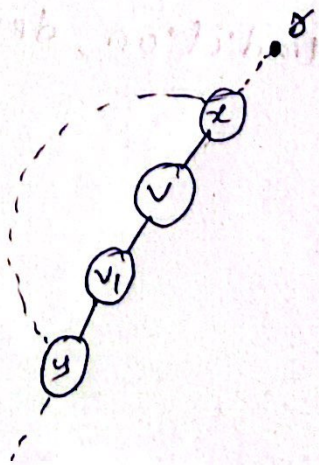
∴ Different biconnected components can't share two vertices.

By contradiction, two biconnected component share a single vertex or none.

d) If we convert all biconnected components into one single node and retain seperating vertices as a individual node. The derived graph will not have any cycles, as cycle existence in this derived graph would implies that any vertex of one biconnected component will be reachable from any vertex of another biconnected component. As mentioned in above answer, this will violate the definition of biconnected component. So we can conclude that cycles won't be there by contradiction, derived graph is tree.

e) Let's assume that D.F.S tree has only one child. This would implies that every vertex has been processed/explored from subsequent nodes in graph without back-tracing to source node/root node. So, if we delete root node 'q' from graph, every other vertex is reachable from other without traversing through rootnode. So, it is not a seperating vertex. By contradiction, root should have more than one children if it is a seperating vertex.

f) Let's say a non-root vertex 'v' has a child v_1 and descandants of v_1 has a backedge to ancestor of 'v'. so assume a backedge from 'y' to 'x'. If we remove 'v', then also we will have connected graph between angestor of 'v' and descendant of 'v'. so, it is not a separating vertex. By contradiction, we can say that non-root vertex 'v' is seperating vertex if and only if there is no backedge from decandants of 'v' to ansestors of 'v'.



g) while performing D.F.S, we will assign pre number to processed node, and move forward while storing previously processed node in any pointer. while in recursion if we find any back edge from 'v' to 'w', then we can assign the low value for vertex/node stored in previously processed node pointer by deriving minimum of pre(w) (which is already assigned) and Pre(v) which can be assigned at that point of instruction.

we have to intialize array for pre and low values before starting process.

As we are deriving low values during D.F.S itself, time taken will be $O(V+E)$ + c(operation to assign low value & previously processed)

so, it is linear time.

h) We perform the above D.F.S process and calculate low values once we calculate low value of a current node; and if pre(current node's parent) is less than the low value, low(current node) then the parent of this node is a seperating vertex.

If we encounter seperating vertex, the graph processed till now will be biconnected component.

If processed biconnected component just have one edge between two vertices, then that is a bridge.

So, we perform the D.F.S for the first time to calculate Pre and low values. Then, we will kick off D.F.S for one more time to derive bridges, biconnected components and seperating vertices.

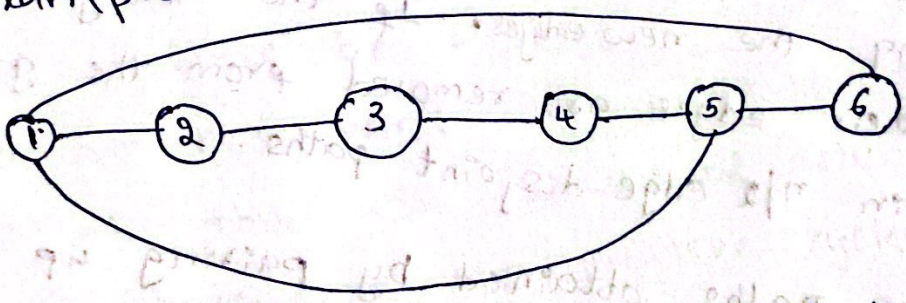
Problem 5 :- 4.4 from DPV book.

The given algorithm have following assumptions:

- Assume (v, w) is a back edge during DFS
- It forms a cycle with the edges of the tree from v to w and the length of the cycle can be calculated as level[v] - level[w] + 1

* In the given equation, the level of the node is its distance from the root node. But the given algorithm doesn't work if there is more than one back edge in the shortest cycle.

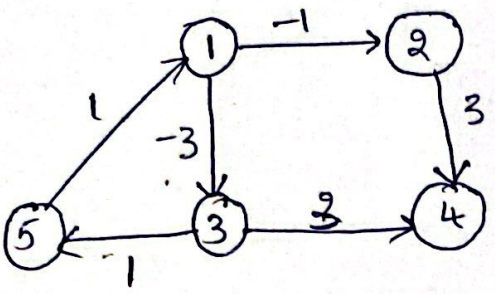
* For example let us see one example.



In the above figure, the shortest cycle would be $1 \to 5 \to 6 \to 1$, but the algorithm will detect $1-2-3-4-5 \to 1$

∴ we can say that the given algorithm fails when the shortest cycle consists of more than one back edge.

Problem 6 : 4.9 from DPV



If we take Node ① as source. The node1 will be set to zero at first and ① to ③ shortest distance will be -3 and from ① to ⑤ shortest distance will be -3+1=2. now from ⑤ to ① the distance be -3+1+1=-1 which is less than 0. So at this point the value of ① will be set to -1 this will continue and fell onto neverending loop. because of the negative cycles formed.

So if a source have an edge with negative. weighted away from source then there is a chance for negative cycle and the algorithm fall into neverending loop, so dijkstras algorithm fail in such case.