Theoritical Assignment 4

Sahil Dhull

160607

Ques 1:

1.
If G is unique path graph, then the DFS tree T_v of vertex v doesn't have (1) Forward Edges and (2) Cross Edges. Back edges are possible.
Necessary and Sufficient Condition:
G is a unique path graph iff $\forall$ v $\in V_G$, T_v (DFS Tree of vertex v) does not have forward and cross edges.
Proof:
One Way: If G is a unique path graph, then $\forall$ v $\in V_G$, T_v does not have forward or cross edges. Lets assume for some v $\in V_G$, T_v have forward or cross or both type of edges.
CASE 1: T_v has only forward edges:
Let the forward edge be between v and u i.e. v $\rightarrow$ u. And in the tree T_v, there will be some path v $\rightarrow\rightarrow$ u ($\rightarrow\rightarrow$ denotes path having some edges in between, but $\rightarrow$ denotes direct edge) that consists of only tree edges (because it is DFS tree of v).
So there exist 2 paths from v to u, one direct edge and the other through tree edges. Contradiction!! Hence this case is not possible.
CASE 2: T_v has only cross edges:
Let the DFS Tree T_v be such that $\exists$ cross edge between some nodes x and y. And $\exists$ v $\rightarrow\rightarrow$ x in DFS that uses only tree edges and $\exists$ v $\rightarrow\rightarrow$ y in DFS that uses only tree edges.
But there also exists a path from v to y such that v $\rightarrow\rightarrow$ $x \rightarrow y$ which contains exactly one cross edge. So it is no more unique path graph as there are 2 paths from v to y. Contradiction!! Hence this case is also not possible.
CASE 3: T_v has both forward and cross edges:
This case is union of above 2 cases, both of which are not possible. Hence this case is also not possible.
So all the cases are not possible, hence the assumption is wrong. So $\forall$ v $\in V_G$, T_v does not have forward or cross edges.
Other Way: If $\forall$ v $\in V_G$, T_v does not have forward or cross edges, then G is a unique path graph.
Assumption: Let G be not a unique path graph, But T_v satisfies the above condition.
Now since G is not a unique path graph, $\exists$ at least one vertex set (v,u) such that there are 2 paths from v to u. Consider T_v (DFS Tree with vertex x). It is given that T_v has no forward or cross edge. Since u is reachable from v, it must be present in its T_v and it must be descendant of v in T_v. This means removing back edges from T_v doesn't affect the reachability of u from v. After removing the back edges, T_v now becomes proper tree with k-1 edges, where k is number of nodes in T_v. Now we know that a tree is unique path graph, implying $\exists$ a unique path from v to u. But it is not the case, hence Contradiction!!
Assumption is false. Implies G is a unique path graph.
Hence Proved.

2.
Modification: Simply call DFS for each vertex v in G (By each time initializing array of visited nodes), as soon as any cross edge or forward edge is detected, exit the code and print "G is not unique path", else it is unique path graph. For detecting cross edge and forward edge, use start time and finish time of nodes, and apply conditions on them.
Pseudo Code:

```
main{
    flag=0 // for testing if G is unique path or not
    for all v in G {
        for all v in G{
            mark unvisited
            start[v]=0
            end[v]=0
```

```
        }
        s_index=0 // for start time
        e_index=0 // for end time
        for all v in G{
            if v is unvisited and flag=0
                DFS(v)
        }
    }
    if flag=0
        Print "G is unique Path Graph"
    else
        Print "G is not unique path graph"
}
DFS(v){
    mark v as visited
    start[v]=s_index
    increment s_index
    for all w in adj[v]{
        if flag=0{
            if w is unvisited{
                DFS(w)
            }
            else if start[w]>start[v]{
                //Forward Edge is present
                //Exit the loop and recursion
                set flag=1
            }
            else if start[w]<start[v] and end[w]<end[v]{
                //Cross Edge is present
                //Exit the loop and recursion
                set flag=1
            }
        }
    }
    end[v]=e_index
    increment e_index
}
```

Time Complexity:
$| V |$ = m and $| E |$ = m, so the loop in main is executed m times. Each time first all vertices are assigned unvisited and start and end time of all vertices as 0, this again occurs for m times making time as O(m^2) till now. In the next loop, dfs is called. Now lets calculate how much time each dfs will take. DFS take O($| V | + | E |$) time with no modification, so without modification time would have been O(n) for DFS.
But the modification is such that whenever any cross edge or forward edge is encountered, dfs stops. So $| E |$ here accounts for only tree edges and back edges (beacuse dfs will traverse at max all tree edges and back edges in worst case and cross or forward edges are not traveleld at all).
Now in DFS, number of tree edges will be m-1 i.e. O(m). Task is to count number of back edges.
Note: $| E_{back} |$ = O(m) [Proof at the end of ques]
So $| E |$ = O(m) (in our case), implying time for dfs is O($| V | + | E |$) = O(m+m) = O(m). Hence the second loop (DFS traversal) also takes O(m) time. So total time will be $\mathbf{O(}m^2\mathbf{)}$.
Proof: (that $| E_{back} |$ = O(m))
When $| E_{back} |$ = m-1 for some T_v, there is only one case when G is unique path graph and that is when corresponding to each tree edge, there is exactly one back edge i.e. b $\rightarrow$ a (back edge) whenever a $\rightarrow$ b (tree edge). Any arrangement other than this would lead to G not being unique path graph, because then atleast one back edge will point to some ancestor a of some vertex b, such that a is not parent of b (let x be the parent of b) and hence there will be 2 paths from x to a (one being back edge from x to a and other being x $\rightarrow$ b $\rightarrow$ a).
So if $| E_{back} | \geq$ m, then atleast one back edge will be similar like discussed above and hence G will not be unique path graph.
Hence $| E_{back} |$ = O(m).

Ques 2:

1. Given E in CNF form: E = $a_1 \wedge a_2 \wedge a_3 ... \wedge a_K$
   where each a_i for i from 1 to K is a clause such that $a_i = x_i \vee y_i$ where x_i *or* $\overline{x_i}$ $\in$ X and y_i *or* $\overline{y_i}$ $\in$ X
   We know, a $\vee$ b $\equiv \overline{a} \Rightarrow$ b,
   also a $\vee$ b $\equiv$ b $\vee$ a $\equiv \overline{b} \Rightarrow$ a.
   So each clause a $\vee$ b will be replaced by: $(\overline{a} \Rightarrow$ b) $\wedge$ $(\overline{b} \Rightarrow$ a)
   Hence E = $(\overline{x_i} \Rightarrow y_i) \wedge (\overline{y_i} \Rightarrow x_i) \wedge ... \wedge (\overline{x_K} \Rightarrow y_K) \wedge (\overline{y_K} \Rightarrow x_K)$
   Now consider constructing a graph where each clause $a \Rightarrow b$ means there are 2 nodes a and b and a directed edge from a to b in the graph. This is done for all clauses in E.
   So, Vertex Set = all x_i, $\overline{x_i}$, y_i, $\overline{y_i}$ for i=1 to K
   Edge Set = all $(\overline{x_i}, y_i)$ *and* $(\overline{y_i}, x_i)$ for i=1 to K

2. For E to be satisfiable, there must be atleast one assignment of truth values to the variables such that E is true.
   Since E is in CNF form, this is only true when all clauses of E are true i.e. all expressions of form a $\Rightarrow$ b must be true. There are 3 cases possible: a(T) and b(T) or a(F) and b(F) or a(F) and b(T). Note that a $\Rightarrow$ b means directed edge from a to b in graph.
   Similarly there might be an expression b $\Rightarrow$ c for which also there are 3 options. But as soon as there exists cycles like a $\Rightarrow$ b $\Rightarrow$ c $\Rightarrow$ a, the possibilities diminish and only 2 cases are possible, either all must be T or all must be F, because lets say A is F and C is T, then c $\Rightarrow$ a means a must also be true.
   More importantly, this is true for an SCC i.e. in any SCC all nodes must have same truth value.
   Proof: Consider a, b and c to be part of a single SCC, then
   Case1: if a is T, then b is T, c is T, and so on, all nodes of SCC are true.
   Case2: if a is F, b has 2 options, T or F. Say b takes T, hence c must be T, and all reachable nodes from c must be T. But a is also reachable from c, hence a is T. Contradiction!!
   Hence b is F. Similarly c is F and so on all nodes of SCC must be F.
   Hence proved that all nodes in SCC must have same truth value.
   Now if all nodes in SCC have same truth value, as soon as we encounter x and $\overline{x}$ in same SCC, it becomes unsatisfiable.
   STEPS TO FOLLOW:

   - First run dfs to push nodes onto stack.
   - Reinitialize the visit array of nodes.
   - Flip the edges of graph.
   - Run dfs 2nd time using top elements of stack to find SCCs of the graph G. (Till here is the algorithm for finding SCC of graph G as covered in class)
   - While doing the dfs second time, store the component number for each node.
   - Check if component number of any x and $\overline{x}$ is same, for all x in V.
   - If they turn out to be same (implying x and $\overline{x}$ lie in same SCC) , print E is unsatisfiable and exit.
   - Else E is satisfiable.

3. - If E is satisfiable, means that all expressions of form u $\Rightarrow$ v are true in E. So one among the following must be true: u is T and v is T or u is F and v is F or u is F and v is T.
   - Since E is satisfiable, implies all SCCs in G must have atleast one assignment value for which everything suits. So consider the graph X which basically contains node for each SCC in G and edge from node 1 to node 2, if there is an edge from SCC1 to SCC2 (An edge from SCC1 to SCC2 means there are u in SCC1 and v in SCC2 such that there is edge form u to v). Now X will be just like any other directed graph.
   - Now since all nodes inside any SCC are having same truth value in G, we can consider assigning that value to corresponding node in graph X.
   - Consider 2 SCCs SCC_a and SCC_b in G such that edge exists from SCC_a to SCC_b through some nodes u in SCC_a to v in SCC_b. Corresponding to them there are nodes a and b in X.
   - So there will again be 3 cases for truth values of u and v. And same 3 cases wil be for truth values of SCC_a and SCC_b and by the argument above, same 3 cases will be for a and b in X.
   - By this argument if we assign truth values to nodes in X, we will get truth values for all variables in G, i.e. for all literals in E.

- So now we need to assign values to nodes of graph X, in order to make it satisfiable (knowing that X is satisfiable).
- Consider a node a in X, and all edges incident on a. If a is assigned T, then all nodes on the other sides of these incident edges can either be assigned T or F values. But if these nodes are also assigned T, then all nodes from which edges on these nodes are incident can be assigned T or F. But if they are also assigned T and so on, we can assign truth values to nodes in X.
- Inspecting carefully, we get that nodes should be assigned truth values in reverse topological order because if we start with any random node m, then all nodes to which edge goes from m must also have T value, which may not always be possible. Hence this assignment should start from node that is at bottom-most level in topological order, and continue upwards and then we can assure the truth values can be assigned to make E satisfiable.
- Also if some node comes in this path towards upward topological nodes such that it can't be assigned T value (beacuse of assignment of T to some node earlier) then there will be 2 cases:
  (a) Either mark it F, and continue assignment and check if possible. If not 2nd case must be true (because E is satisfiable).
  (b) Mark that earlier node F and then restart assignment from that earlier node.
- So in this way truth values are assigned to nodes in X.
- After this truth values can be assigned to corresponding SCC in G. Assigning truth value to SCC means assign that value to all those nodes which are not assigned any value earlier and check for other nodes which were already assigned values (due to assignment of their negation in any other SCC), check that truth value is compatible with their already assigned values. If not compatible, then the case like above arises where 2 cases are possible and both are checked.
- In this way truth values are assigned to all literals in E to make E satisfiable.