CSC263 Fall 2017 Assignment 4

Name: Fan Shen, Make Zhang
SN: 1002172187, 1002376555

Q1. SOLUTION

a) For solving this question, we will use a directed graph that named G with N vertices, and each vertex will contain one statement from the given group of statements and numbered from 1 to N. If one vertex(i.e. vertex a) contains the statement about another statement(i.e. vertex b), we will build a edge called (a,b) between them. And we also will assign different weight to the edges. For example, if the a said b is True, we will assign 1 for that. And if a said b is False, we will assign 0.

b) If the N statements do form a paradox, there must be at least one cycle that has odd total weight edges. Since it shows there must be a conflict happened.(i.e. 1 said 2 is True, and 2 said 1 is False and the total weight in the cycle will be 1 that is a odd number) and vice-versa. We could know that if there is no paradox for those statements when there is no cycle has odd total weight.(i.e. 1 said 2 is True, and 2 said 1 is True. The total weight will be 2 that is a even number)

c) First, we need to construct a directed graph formed by given statements. Since we has all the properties above, we could apply the DFS. As the search progress, we will looking for the cycle or we can say back edge, and we will stop and count the total weight in that cycle. If that's a odd number, we will return "Paradox Found" and end the search. Else, we need to continue our search until the next cycle appear. At the end, if we visit all the vertex in the graph, and there is still no odd-weight graph, we will return "NO Paradox" and end the program.

d) In the worst case, we would search the whole graph and there is no Paradox formed by those statements. Therefore, the total run time for worst case would be the time of constructing the graph plus the time of applying the depth first search, which equals to $O(N) + O(|V| + |E|) \in O(N)$ since in worst case, |V| will be N and the largest |E| will also be |N| (i.e. there are two edges between each two vertices)

Q2. SOLUTION

a) We will model this problem using a data structure of directed graph. Each kid is a vertex. If there are n ill-behaved children, there are n vertex. Each statement is a edge. If "kid i hates kid j", create a directed edge from kid i to kid j. In total, there are n vertexes and m edges.

b) It is not always possible to find a valid arrangement. If there is a circle in the directed graph, it cannot form a valid arrangement.

For example, kid 1 hates kid 2 and kid 2 hates kid 1. It is a simple cycle.

If put kid 1 before kid 2, kid 2 will throw something at kid 1. If put kid 2 before kid 1, kid 1 will throw something at kid 2.

As a result, It is impossible to arrange kid 1 and kid 2 to avoid throwing.

c) Our algorithm is use Topological sort of a directed acyclic graph.

```
ARRANGE-IN-LINE(G):
for each u ∈ V do
        u.color = White
        u.π = NIL
time = 0
list = linked-list(NIL)
for each u ∈ V do
        if u.color = white
            DFS-VISIT(G, u)
return list
DFS-VISIT(G, u):
u.color = grey
time = time + 1
u.d = time
for each v ∈ Adj[u] do
        if v.color = white
            v.π = u
            DFS-VISIT(G, v)
u.color = black
time = time + 1
u.f = time
u.next = list.header
list.header = u
```

The running time is $O(V + E)$ which is $O(n + m)$ in this question.

This is because it is just DFS of a graph and add to a linked list if the vertex ends. A DFS for a graph had a running time of $O(V + E)$ and the linked list take constant time to add to the header.

d) ARRANGE-IN-ROWS(G):

```
list = ARRANGE-IN-LINE(G)
```

```
u = list.header
line = 1
u.line = 1
while u.next != NIL do
        if u.d < u.next.f
            if u.line < line
                u.next.line = u.line + 1
            else
                line = line + 1
                u.next.line = line
        else
            u.line = 1
        u =u.next
return line
```

Simple explanation: The algorithm is using the straight line from question c. If vertex's discovered time is smaller than its' next vertex's finished time, these two vertexes should be in two rows. If vertex's line is smaller than the total line, just set its' next vertex's line as one more. Else need to add one more for the total line.

The running time of this algorithm is $O(V + E)$ which is $O(n + m)$ in this question.

This is because the ARRANGE-IN-LINE(G) algorithm runs in $O(n+m)$ and then this algorithm go through the list which contains n vertexes. In the while loop, it is all simple assignment, which take constant time. So in total, the worst-case running time is $O(n + m + n)$ which is $O(n + m)$.

Q3. SOLUTION

Use the disjoint set with union-by-rank and path compression to make the algorithm. There are Make-Set(a), Find-Set(a), and Union(a, b) functions for disjoint set.

For the abstract date type DEPR:

Initialize the date with n + 1 Make-Set for each element in $\{0\} \bigcup \{a, ..., b\}$ and set its min-int as itself which present the minimum int in the tree. (e.g. $a.min - int = a$). This will take $O(n + 1)$ time.

Each Delete(S, i) operation is similar to Union operation for disjoint set. First find the element in the disjoint forest, if not in the disjoint, then do nothing; if it is a then Union(0, a) and update the min-int of a to be 0; if its min-int is itself then Union$(i, i - 1)$ (with union-by-rank and path compression) and update the min-int of i, else do nothing.

In this case, S only contain the element that its min-int is itself. The delete element i is not in S but its min-int is still in S as the biggest element in S that is smaller than the delete element i.

Each Predecessor(S, i) operation is similar to Find-Set operation for disjoint set. If i is smaller than a, then return 0. If i is bigger than b, then call Find-Set(b) and return the min-int of b. If i is between a and b, then call Find-Set$(i - 1)$ and return the min-int of $i - 1$.

In this case, it only return min-int of an element or 0. From Delete operation, the min-int of i is the predecessor of i and is in S. So the return value is either in S or 0 and is predecessor of i.

In total, each Delete(S, i) operation will call at most one Union with some constant assignments and each Predecessor(S, i) will call at most one Find-set with some constant assignments. The worst case running time for m operations of disjoint set with union-by-rank and path compression is $O(mlog^*m)$, by aggregate method, each operation needs $O(log^*m)$ amortized cost $(\frac{T(m)}{m} = \frac{O(mlog^*m)}{m} = O(log^*m))$, the amortized cost per operation for m Delete and Predecessor is also $O(log^*m)$.

Q4. SOLUTION

NOTE: All Redlines represent the connection pathes

a) False, the counter example is Figure(a.2). We have also draw the graph named Figure(a.1) that follows the given instrcution. In that graph, p2 must connect with p1 since the question said pi must connect with its closest neighbour among p1...p(i-1) and p3 connects to p1 also. But in our counter example, it's simple to see that the distance between p2 and p3 is smaller than p1 to p2. Beacuse $\angle$ p2p3p1 is greater than 90 degree and the hypotenuse must longger than the other side. Therefore, Figure(a.1) is not a MST.|

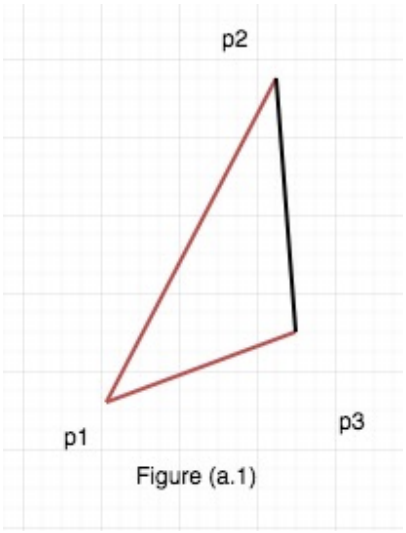

Figure (a.1)

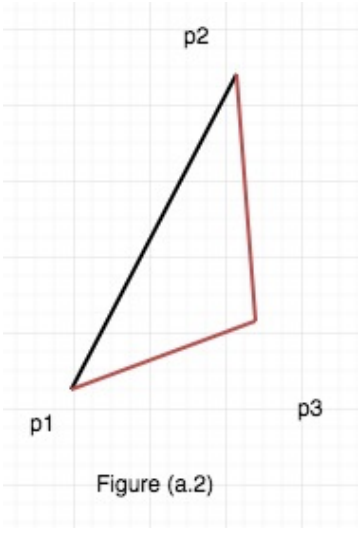

Figure (a.2)

b) False, the counter example is Figure(b.2). We have also draw the graph named Figure(b.1) that follows the given instrcution. In that graph, A must connect with B in their seperated part and C connects to D also. But in our counter example, it's simple to see that the distance between the two short sides is smaller than the long sides. And we would prefer to connect AC and BD in order to make the MST. Therefore, Figure(b.1) is not a MST.|

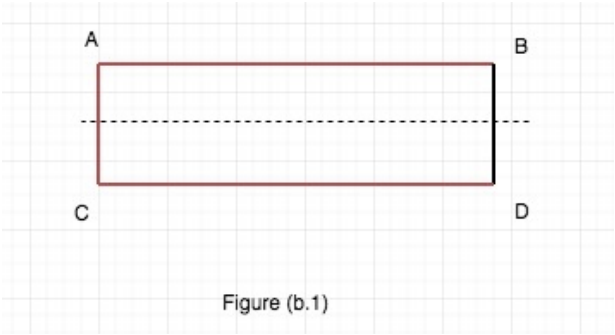

Figure (b.1)

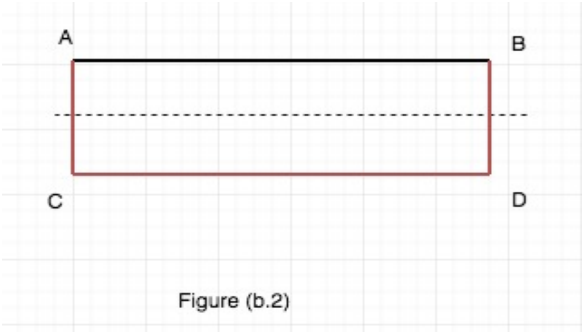

Figure (b.2)

NOTE: Dotted line represent how we seperate the graph to different parts

Q5. SOLUTION

Using contradiction to prove that for a connected undirected weighted graph, if there is no edges have same weight, there is only one MST.

Assume there is two MST.

Let edge e_1 be the lightest edge in the graph that is selected in one of the MST but not the other one. Let e_1 in the first MST but not in the second MST.

If add e_1 to second MST, it will have a cycle. So there is an edge e_2 that is in second MST but not in first MST since MST should not have cycle.

Since that e_1 is the lightest edge that is in one of the MST and all edges are distinct, $w(e_1) < w(e_2)$.

So if we replace e_2 in second MST with e_1, we get a smaller total weight, which contradict the fact that the second tree is also a MST.

So for a connected undirected weighted graph, if there is no edges have same weight, there is only one MST.