CMPS 102 — Quarter Spring 2017 – Homework 3

VLADOI MARIAN

May 18, 2017

I have read and agree to the collaboration policy.Vladoi Marian
I want to choose homework heavy option.
Name of students I worked with: Victor Shahbazian and Mitchell Etzel

Solution to Problem 1: Graphs

A) 1. We will represent the n projects as DAG.
2. Each node has the following data members :
a. name; b. array of pointers to the succesors nodes; c. D= duration of the project; d. ES = early time to start the project; EF =early time to finish the project; LS = late time to start the project; LF = late time to finish the project;
3. EF = ES + D, and LS = LF - D.
4. We will create a dummy node , s, the start node This will be the parent node of all nodes that do not have parent nodes. ES(s) = 0; D(s) = 0; EF(s) = 0; 5. 4. We will create a dummy node , f, the final node.This will be the children node of all nodes that do not have children node. LS(f) = 0; D(f) = 0; LF(s) = 0; 6. We will initialize all the nodes with the duration, ES = 0, EF = 0; LS = 0; LF = ∞; 4. In order to calculate the earliest possible completion time c(v) I will modify Disktra algorithm.

```
DIJKSTRA Modified (Graph,s)
1. Create vertex set Q = empty.
2. Start with s , the start node .
3. Add the start node s to Q.
4. While Q is not empty
5.     u = node in Q with maximum duration
6.     remove u from the Q
7.     for each children v of u
8.         ES(v) = EF(u) (Early start of v = Early finish of u)
9          New early finish time of v NewEF (v) = ES(v) + D(v) (Update the Early finish of v by addiging the duration)
10.        if ( NewEF(v) > EF(v) ) (If we find that the v project has to wait longer for a different dependent project)
11.            then we update EF(v) = NewEF(v)
12     add the node v to the Q
13. return the earliest possible completion time for all nodes in the graph.
```

B) 1. After we applyed DIJKSTRA Modified, on the same graph we will apply the following algorithm.
1. To find the critical projects we will reverse all the edges of the Graph. The start node becomes a finish node and the finish node becomes a start node .

```
Reverse DIJKSTRA Modified (Graph,s)
1. Create vertex set Q = empty.
2. Create a critical nodes list L = empty
3. Start with s , the start node .
```

```
4. LF(s) = EF(s) (late finist time of s = early finish time of s)
5. LS(s) = LF(s) - D.(s) (calculating the late start time by subtracting the duration)
6. Add the start node s to Q.
7. **While** Q is not empty
8.     u = node in Q with maximum Late Finish Time
9.     remove u from the Q
10.     **for** each children v of u
11.          New Late finish time NewLF(v) = LS(u) ( New late finish of v = late start of u)
12.          **if** ( NewLF(v) $<$ LF(v)) (we choose the smallest late finish time)
13.               then LF(v) = NewLF(v) (update the late finish time)
14.          LS (v) = LF(v) - D(v) (Update the Late start time of v by subtracting the duration
15.          **if**(EF(v) == LF(v) ) ( if early finish time == late finish time)
16.               then add the node v to L ( we founded a critical nod and added to the List)
17          add the node v to the Q
18. return the List L with all critical nodes.
```

Proof:

Both Algorithms used modified version of Dijkstra.
For part A of the problem , we find all the biggest paths from start node to all nodes in the graph. We want to compute what is the earliest complition time for each project. The earlies finish time of nodes is the earliest complition time for each project. At the end of the algorithm we calculated the earliest complition time for all nodes in the Graph. The final node tell us the earliest complition time of the entire project. For part B we use againg Dijkstra twice. Once we calculated the earliest finish time we go in reverse in order to calculate the latest finish time. If the earliest finish time == latest finish time it means that the node is a critical node (increasing its duration by 1 would delay the completion of the entire product)
DIJKSTRA Modified (Graph,s) :
1. We maintain a set of explored nodes S whose longest path distance d(u) from s to u is known.
2. Initialize S = $\{s\}$ and d(s) = 0.
3. Repeatedly choose unexplored node v which maximize the earliest finish time.
4. We add the node v to the set S and we calculate the earliest finish time.
5. For each node u in S, d(u) is the length of the biggest path from s to u.
6. It was proved in class by induction that the statement 5 is true.
7. If this algorithm is implemented with a Fibonacci heap the running time is:
Each EXTRACT-MIN takes O(1) amortized time.
There are O(V) other operations, taking O(lg V) amortized time each.
Therefore, time is O(V lg V + E).
8. Space complexity of DIJKSTRA Modified algorithm is O(V^2)

Reverse DIJKSTRA Modified (Graph,s)
1. We maintain a set of explored nodes S whose shortest late starting time path distance d(u) from s to u is known.
2. Initialize S = $\{s\}$ and d(s) = 0.
3. Repeatedly choose unexplored node v which minimize the late starting time.
4. We add the node v to the set S and we calculate the late satrting time.
5. For each node u in S, d(u) is the length of the smallest late starting time path from s to u.
6. It was proved in class by induction that the statement 5 is true.
7. If this algorithm is implemented with a Fibonacci heap the running time is:
Each EXTRACT-MIN takes O(1) amortized time.
There are O(V) other operations, taking O(lg V) amortized time each.
Therefore, time is O(V lg V + E).
8. Space complexity of Reverse DIJKSTRA Modified algorithm is O(V^2)