Stephen Woodbury 5/22/17 CMPS 102
I have read and agree to the collaboration policy. Stephen Woodbury.
Collaborators: none

Assignment 3_1 : Graphs – Software Project Management

Assumptions:

1) Our Graph is connected.

Components for our Algorithm:

inProgress: List of tasks currently being worked on – init to nothing
cost: Array of costs of each task (task # is the index #) : defaulty init'd
VneedsU's: Array of Arrays: v by(u's needed for said v to start)
UneededbyV's: Array of Arrays: u by (v's that rely on said u)
[note: u and v represent projects]
UCount: Array of # of projects needed per project to start.
completeTime: Array of completion times for every project : all init to 0
CriticalProjects: List of projects that are critical : init to nothing
CritsToAnalyze: Queue of Critical Projects to scan through : init to nothin
time: int count of current time: init to 0.
projectsComplete: int count of projects completed, init to 0.
totalProjects: int count of projects to complete total: init to "n"
pTA: int, project To Analyze

Algorithm:

```
1 Set up VneedsU's, UneededbyV's, Ucount accordingly.
2 Add all projects that aren't dependent on other projects to inProgress
3 While(projectsComplete < totalProjects)
4   time++
5   for(each member m of inProgress)
6     if(--cost[m] <= 0)
7       completeTime[m] = time
8       projectsComplete++
9       take m out of inProgress
10       for(each project p in UneededbyV's[m])
11        if(--UCount[p] <=0)
12          add project p to inProgress
13Add project with latest compl.time to CritsToAnalyze and CriticalProjects
14while(CritsToAnalyze != nothing)
15  pTA = CritsToAnalyze.dequeue()
16  Add project w/latest compl. Time of all projects in VneedsU's[pTA] to CritsToAnalyze
17  Add project w/latest compl. Time of all projects in VneedsU's[pTA] to CriticalProjects
  [Notes: if there's a tie, add all tied projects, if project already
  exists in CriticalProjects or in CritsToAnalyze, don't add it again].
18Return completeTime and CriticalProjects
```

Algorithm Description:

Our Algorithm takes our project list and project dependencies and puts the necessary information into *VneedsU's, UneededbyV's, Ucount.* We start all projects that aren't dependent on other projects and start doing our projects. As our while loop iterates, our *time* increases. Every time *time* increases, we decrease the cost of all projects currently being ran as they are being worked on. Once a project's cost reaches 0, increase our projects completed, take the project off of projects being ran, set completion time, check all projects that require the completed project as a prereq, decrease the prereq count for that project, if there are no longer any prereqs for

that project, start it. Once we're out of the while loop, all projects should be completed as early as they could have been. Find the project with the latest completion time (in case of tie, take all ties), make them critical projects, add them to crit proj's to analyze. While we have crit proj's to analyze, take a crit proj from the crit proj's to analyze, remove it from projects to analyze, and inspect all projects that are a pre req of this project. Find the pre-req project with the latest completion time (if a tie, add all ties), make a critical project, add to crit proj's to analyze pull another project off of proj's to analyze... Once done with second while loop, we should have a list of critical projects. Return the list of critical projects and the array of earliest completion times for all projects.

Proof of Correctness : Termination

claim: Algorithm terminates.

Pf. For the first While loop, every iteration increases our time by 1 and the cost of every project in progress will decrease by 1. There will always be a project in progress as there will always be a project with its pre req's completed. There will always be projects with its pre-reqs completed as there are no cycles. At the start, there are no pre-reqs for projects started at time 0 as there are no cycles. Once the cost of a project has decreased to 0, it is removed from the inProgress list and cannot be added to the inProgress list again as it is completed. In cannot be added again as if it was, that would imply there is a cycle, which is a contradiction. We also know every project has a finite duration, which means the cost will reach zero no matter what. Therefore the first while loop must terminate. For the Second loop, every iteration removes a project from our CriticalProjects to analyze. There can only be a total of n projects to analyze, therefore our second while loop must terminate.

Proof of Correctness : Every Project has earliest completion time

Claim: Every project finishes as early as it can.

If a project has no project dependencies, it runs until completion. If the project is not running during duration of algorithm, it must be that it is dependent on a project not yet finished. We can have an infinite number of parallel projects running so the only thing inhibiting a project from running is if a project it's dependent on isn't complete, which can't be helped. Therefore, every project finishes as early as it can.

RunTime Analysis:

1)No counting as part of cost 2)O(n) 3)O(T) 4)O(1) 5)O(n) 6-9)O(1) 10)O(n) 11-12) O(1) 13-14)O(n) 15)O(1) 16-17)O(n) 18) O(1)

[Notes: n = # projects, T = all durations added together] total: O(n) + O(T)*(7*O(1) + 2*O(n)) + O(n) + O(n)*(O(1) + 2*O(n)) + O(1)

= ***O(Tn) [T>=n].***

Space Analysis:

inProgress, cost, Ucount, completeTime, criticalProjects, CritsToAnalyze: O(n),

VneedsU's, UneededbyV's: O(n^2) [def less than this]

time, projectsComplete, totalProjects, pTA: O(1)

total: ***On^2).***