Alex Iacob

Prof. Haller

CSCI 261 Section 2

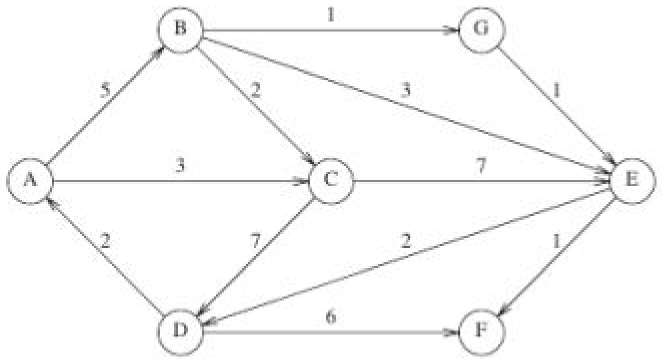
March 11, 2021

Homework #3

E.1: *Suppose you are given a set S={j 1, j 2,..., j n } of jobs, where job j_i requires t_i units of time to complete once it is started. You have one resource for running these jobs, so only one job can be run at a time. Let f_i be the finish time of job j_i. Your goal is to minimize the average finish time. For example, suppose there are two jobs, j_1 and j_2 with processing times t_1=3 and t _2=5. Consider the schedule in which j_2 runs first followed by j_1, then f 2=5 and f 1=8, and the average finish time is (5+8)/2 = 6.5. If we schedule the jobs in the opposite order, then f 1=3 and f 2=8 and the average finish time is (3+8)/2 = 5.5. Give an algorithm that minimizes the average completion time and state the running time.*

To minimize the average completion time, we must sort the jobs in the list from longest completion time to shortest completion time. We do this because if the last element (job with the shortest time) is removed first, we won't have to reindex the other elements in the list. Now we just have to iterate through the list until there are no more jobs left to do. This means that this algorithm must sort the list, then iterate through all of the elements in the list. This completion time would be O(n log(n))

E.2:



a) *Find the shortest path from **A** to all other nodes in the graph. Show work.*

	V	**Known**	**d_v**	**P_v**
A	A	F **-> T**	**0**	**0**
A -> B	B	F **-> T**	∞ -> **5**	0 -> **A**
A -> C	C	F **-> T**	∞ -> **3**	0 -> **A**
A -> B -> G	G	F **-> T**	∞ -> **6**	0 -> **B**
A -> B -> G -> E	E	F **-> T**	∞ -> **7**	0 -> C -> **G**
A -> B -> G -> E -> F	F	F **-> T**	∞ -> **8**	0 -> **E**
A -> B -> G -> E -> D	D	F **-> T**	∞ -> **9**	0 -> C -> **E**

b) *Find the shortest unweighted path from **B** to all other nodes in the graph. Show work.*

	V	**Known**	**d_v**	**P_v**
B -> C -> D -> A	A	F **-> T**	∞ -> **3**	null -> **D**
B	B	F **-> T**	**0**	**null**
B -> C	C	F **-> T**	∞ -> **1**	null -> **B**
B -> C -> D	D	F **-> T**	∞ -> **2**	null -> **C**
B -> E	E	F **-> T**	∞ -> **1**	null -> **B**
B -> E -> F	F	F **-> T**	∞ -> **2**	null -> **E**
B -> G	G	F **-> T**	∞ -> **1**	null -> **B**

Text 4.3: Page 189: *Prove that, for a given set of boxes with specified weights, the greedy algorithm currently in use actually minimizes the number of trucks that are needed. Your proof should follow the type of analysis we used for the Interval Scheduling Problem: it should establish the optimality of this greedy packing algorithm by identifying a measure under which it "stays ahead" of all other solutions.*

Base Case: k = 1 truck This case is clear. The greedy algorithm fits as many boxes as possible into the first truck in the order they arrive.

Induction step: Assume this holds true for k trucks, k≥1. That is, greedy fits j' boxes into the first k trucks and the other solution fits i' boxes into the first k trucks, and i'≤ j'.

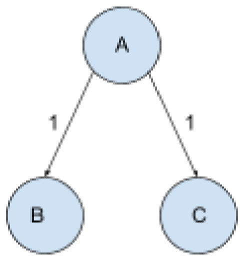
In order for the greedy algorithm to remain ahead of the other possible solutions, the total weight of the first truck of the greedy algorithm must be greater than or equal to the total weight of the first truck of another algorithm. $T_{\text{greedy-weight}} >= T_{\text{other-weight}}$.

Next, we assume that this is the case of the induction step. This means that the total weight for all k trucks for the greedy algorithm must be greater than or equal to the weight of all k trucks for the other algorithms. Also, since we would like to send as much weight as possible, Truck_{k+1} must weigh more than Truck_k.

This means that for every k trucks, the greedy packing algorithm must send the same amount of weight or more as the other algorithms. This also holds true for k + 1 trucks; the greedy algorithm must send an equal or greater amount of weight as the other solution(s).

E.3: *Prove whether it is always, never, or sometimes true that the order in which the nodes are added to the shortest-path tree is the same as the order in which they are encountered in a breadth-first traversal.*

It is sometimes true that the order in which the nodes are added to the shortest path tree is the same order as they are encountered in a BFS.



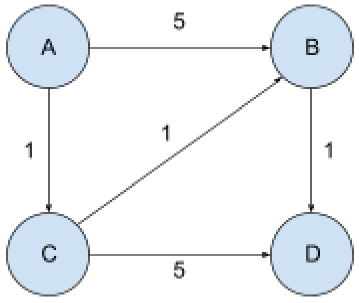
The shortest path from A to B is A -> B

BFS from A to B is A -> B with a cost of 1

The shortest path from A to C is A -> C

BFS from A to C is A -> C with a cost of 1

This situation proves the statement.



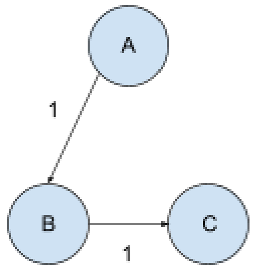
The shortest path from A to D is A -> C -> B -> D with a cost of 3

BFS from A to D is A -> B -> D with a cost of 6

This situation disproves the statement

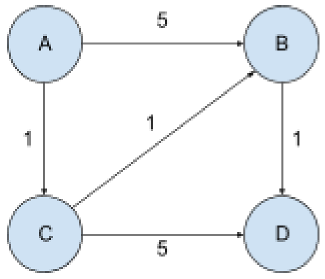
E.4. *Prove whether it is always, never, or sometimes true that the order in which the nodes are added to the shortest-path tree is the same as the order in which they are encountered in a depth-first traversal.*

It is sometimes true that the order in which the nodes are added to the shortest path tree is the same order as they are encountered in a DFT.



The shortest path from A to B is A -> B with a cost of 2

DFT from A to C is A -> B -> C with a cost of 2



The shortest length path from A to D is A -> B -> D with a cost of 6

DFT from A to D is A -> C -> B -> D with a cost of 3