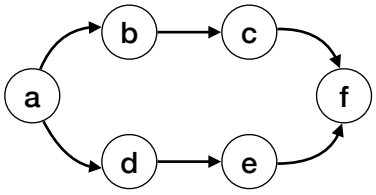
Prof. Salma Elmalaki
Fall 2022

Homework 2 - Solution Sketch

Practicing Chapters 4 (Greedy)

1. *(5 Points)* Chapter 3: Topological Order Consider the directed acyclic graph G in the figure below. How many topological orderings does it have? Explain how you get computed it. List the topological order.



Answer: As we saw in the text (or reasoning directly from the definition), the first node in a topological ordering must be one that has no edge coming into it. Analogously, the last node must be one that has no edge leaving it. Thus, in every topological ordering of G, the node a must come first and the node e must come last. Now we have to figure how the nodes b, c, d, e can be arranged in the middle of the ordering. The edge (b, c) enforces the requirement that b must come before c; similarly for (d, e). This exhausts all the possibilities, and so we conclude that there are four possible topological orderings:
a, b, c, d, e, f
a, b, d, c, e, f
a, d, e, b, c, f
a, d, b, e, c, f

2. *(25 Points)* **Chapter 4: Scheduling**

You have n distinct jobs, labeled $J_1, J_2, ..., J_n$, which can be performed completely independently of one another. Each job consists of two stages: first it needs to be preprocessed on a supercomputer, and then it needs to be finished on one of a local PCs. Let's say that job J_i needs p_i seconds of time on the supercomputer, followed by f_i seconds of time on a PC.

There are at least n PCs available on the premises, the finishing of the jobs can be performed fully in parallel–all the jobs can be processed at the same time. However, the supercomputer can only work on a *single* job at a time, so the system managers need to work out an order in which to feed the jobs to the supercomputer. As soon as the first job in order is done on the supercomputer, it can be handed off to a PC for finishing; at that point in time a second job can be fed to the supercomputer; when the second job is done on the supercomputer, it

can proceed to a PC regardless of whether or not the first job is done (since the PCs work in parallel); and so on.

(a) *(10 Points)* Design a schedule for the ordering of the jobs for the supercomputer that minimize the**completion time** of the schedule. The definition of the **completion time** of the schedule is the earliest time at which *all* jobs will have finished processing on the PCs.

(b) *(5 Points)* What is the running time of your algorithm?

(c) *(10 Points)* Prove that the schedule that you designed is optimal.

Hint: 1- Think of an order of the jobs with some rational explained and try to find a counter example until you find a good order. 2- You can use the exchange of argument to prove the optimality the same way that we did in for minimizing the max. lateness problem.

Answer:

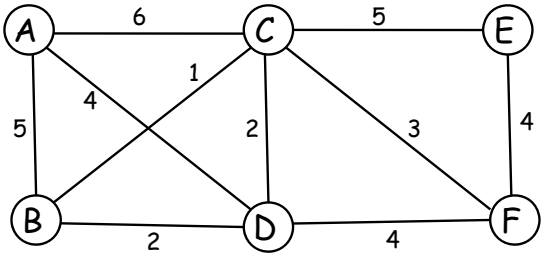
(a) From the question, we know that the working time of the supercomputer does not depend on the job orders. The supercomputer must run all the jobs and the timing is fixed. Clearly, the last job must have the shortest finishing time on PC. Using this reasoning we can suggest an algorithm P:
Schedule job P:
Run jobs in the order of decreasing finishing time f_i

(b) Running time is the $O(nlogn)$

(c) We can use an exchange argument to show that P is optimal, by showing that any schedule other than P can become P by swapping the adjacent job. Meaning that by swapping the adjacent jobs we can convert any other schedule to P without changing the total finishing time. Any schedule other than P must contain two jobs M; and N; so that N; runs directly after M, but the finishing time for the first job is less than the finishing time for the second one. We can optimize this schedule by swapping the order of these two jobs. Let Q be the schedule Q where we swap only the order of M; and N. It is clear that the finishing times for all jobs except M, and N, do not change. The job N; is now scheduled earlier, thus this job will finish earlier than in the original schedule. The job M, is scheduled for later, but the super-computer hands off M, to a PC in the new schedule at the same time as it would be handed off N; in the original schedule P. Since the finishing time for M, is less than the finishing time for N, the job M, will finish earlier in the new schedule than N; would finish in the original one. Hence our swapped schedule does not have a greater completion time. If we define an inversion to be a pair of jobs whose order in the schedule does not agree with the order of their finishing times, then such a swap decreases the number of inversions in Q while not increasing the completion time. Using a sequence of such swaps, we can therefore convert Q to P without increasing the completion time. Therefore the completion time for P is not greater than the completion time for any arbitrary schedule Q. Thus P is optimal.

3. *(20 Points)* **Chapter 4 - Trees on Graphs (Midterm, Fall 2011).**

Consider the undirected graph shown in the following figure. It consists of six nodes A,B,C,D,E,F and nine edges with the shown edge costs.



(a) *(5 Points)* Run Dijkstra's algorithm to find the shortest paths from node A to all other nodes. (Show the final answer and briefly describe the intermediate steps.)

Answer:

Dijkstra's algorithm computes a tree with the following edges:

$$(A, B), (A, D), (A, C), (D, F), (C, E)$$

During the execution, we break ties arbitrarily, which may lead to different SPTs. However, the shortest paths from A to all nodes have always the same cost (distance):

$$d(A) = 0, d(B) = 5, d(C) = 6, d(D) = 4, d(F) = 8, d(E) = 11$$

(b) *(5 Points)* Run an algorithm of your choice (*e.g.,* Kruskal, Prim, Reverse-Delete) and find a minimum spanning tree. (Show the final answer and briefly describe how you got there.)

Answer:

Let's running Kruskal's algorithm, which considers the edges in order of increasing cost:

$$(B, C), \quad (C, D), (B, D), \quad (C, F), \quad (D, F), (E, F), (A, D), \quad (A, B), (C, E) \quad (A, C)$$

Depending on how cycles appear and on how we break ties between edges with the same cost, we end up with different trees.

One MST1 consists of edges $(A, D), (B, C), (C, D), (C, F), (E, F)$ and has cost 14.

Another MST2 consists of edges $(A, D), (B, C), (B, D), (C, F), (E, F)$ and has cost 14.

Any other algorithm would be acceptable as well.

(c) *(5 Points)* Is the minimum spanning tree of this graph unique? Justify your answer, *i.e.,* if the answer is yes, provide a proof; if the answer is no, provide a counter-example and explain why this is the case.

Answer:

In the previous question we provided already 2 MSTs. This is no surprise, as there are several sets of edges with the same cost. Depending on how we break the ties between them, and in what order cycles tend to appear, we end up with different MSTs.

(d) *(5 Points)* Consider the average distance from A to all other nodes, first by following edges on the shortest path tree (a), let's call it d^{avg}_{SPT}; and then following edges on the minimum spanning tree found in (b), let's call it d^{avg}_{MST}. Which one is greater, d^{avg}_{SPT} or d^{avg}_{MST}? Does the same answer hold for any graph $G = (V, E)$ and node $A \in V$, or is it specific to this example?

Answer:

The distance of all shortest path between node A and all other nodes were calculated in (a), we have: $d_{SPT}^{avg} = (5+6+4+8+11)/5 = 34/5$. As found in (b), the MST of G is not unique, the distances of the shortest paths from A to all other nodes using edges in SPT, MST_1, and MST_2 (as named in (b)) are:

	SPT	MST_1	MST_2
A-B	5	7	6
A-C	6	6	7
A-D	4	4	4
A-E	11	13	14
A-F	8	9	10
Total	34	39	41

Hence $d_{MST_1}^{avg} = 39/5$ and $d_{MST_2}^{avg} = 41/5$ and, for graph G, the average distance using SPT is smaller that using any of the possible MSTs. In general, $d_{SPT}^{avg} \leq d_{MST}^{avg}$ since SPT will, by definition, find the shortest path from the node of origin to all other nodes.

4. *(15 points)* **Chapter 4 - Midterm - Fall 2019** - Consider the undirected graph below.

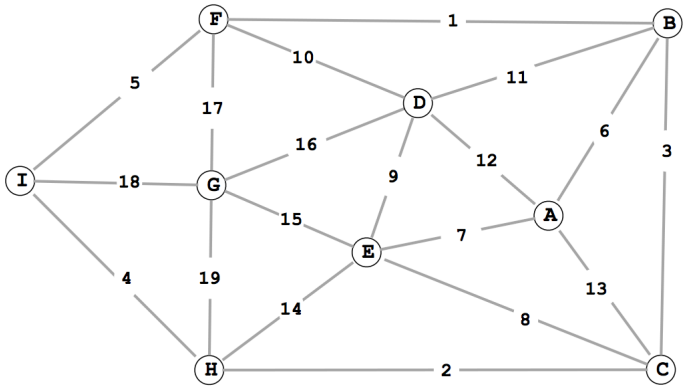
   (a) *(5 Points)*Use an algorithm of your choice to find a Minimum Spanning Tree (MST) for this graph. Show the final answer (draw the tree and write down its cost) and briefly describe how you got the answer (which algorithm you used and some intermediate steps).

   Answer:

   You can use any algorithm Use Kruskal's algorithm and arrange the edges in ascending order of cost:
   (F,B), add
   (H,C), add
   (B,C), add
   (I,H), add
   (I,F), creates a cycle
   (A,B), add
   (A,E), add
   (E,C), creates a cycle
   (E,D), add
   (F,D), creates a cycle
   (D,B), creates a cycle
   (D,A), creates a cycle
   (A,C), creates a cycle
   (H,E), creates a cycle
   (G,E), add
   (G,D), creates a cycle
   (F,G), creates a cycle
   (I,G), creates a cycle
   (G,H) creates a cycle
   Cost = 47

   (b) *(10 Points)* Is the MST for this graph unique? Justify your answer, i.e., if the answer is yes, provide a proof; if the answer is no, provide a second MST as a counter-example.

Answer: It is Unique: All edge costs are distinct (Proof by contradiction). Suppose there exist two MSTs for G, T and T*. Then there is some edge e in T that is not in T*. Now add edge e to T*. Now T*+e has a cycle. Consider the edge e' in this cycle with maximum cost (guaranteed to exist by distinct edge cost assumption). Now use the cycle property proved in lecture which is: the MST does not contain e'.



5. *(10 Points)* **Combinatorial Structure of Spanning Trees:** Let $\mathcal{G}$ be a connected graph, and $\mathcal{T}$ and $\mathcal{T}'$ two different spanning trees of $\mathcal{G}$. We say that $\mathcal{T}$ and $\mathcal{T}'$ are *neighbors* if $\mathcal{T}$ contains exactly one edge that is not in $\mathcal{T}'$, and $\mathcal{T}'$ contains exactly one edge that is not in $\mathcal{T}$.

   Now, from any graph $\mathcal{G}$, we can build a (large) graph $\mathcal{H}$ as follows. The nodes of $\mathcal{H}$ are the spanning trees of $\mathcal{G}$, and there is an edge between two nodes of $\mathcal{H}$ if the corresponding spanning trees are neighbors.

   Is it true that, for any connected graph $\mathcal{G}$, the resulting graph $\mathcal{H}$ is connected? Give a proof that $\mathcal{H}$ is always connected, or provide an example (with explanation) of a connected graph $\mathcal{G}$ for which $\mathcal{H}$ is not connected.

   Answer:

   Yes, H will always be connected. To show this, we prove the following fact:

   (1) Let $T = (V, F)$ and T'=(V,F') be the two spanning trees of G so that $|F - F'| = |F' - F| = k$. Then there is a path in H from T to T' of length k.

   What the above statement says is that we can transform T to T' in k steps, swapping one edge at a time. Also notice that this exercise is about spanning trees, *not* about minimum spanning trees.

   Proof. We prove this by induction on k. It is true for $k = 1$, by the definition of edges on H. Let's assume that it is true for $k - 1$. Now consider $|F - F'| = k > 1$ and choose an edge $f' \in F' - F$. The tree $T \cup \{f'\}$ contains a cycle C and this cycle must contain an edge $f \notin F'$. The tree $T \cup \{f'\} - \{f\} = T'' = (V, F'')$ has the property that $|F'' - F'| = |F' - F''| = k - 1$. Thus, by induction, there is a path of length $k - 1$ from T'' to T'. Since T and T'' are neighbors, it follows that there is a path of length k from T to T'.