CS 6363: Homework 3

Version 1.0

Instructor: Benjamin Raichel

Due Wednesday November 7, at 11:59 pm

Homework is to be submitted online through eLearning by the above deadline. Typeset solutions (in LaTeX) are strongly preferred, but not required. If you chose to hand wright and scan your solutions, if what is written is illegible you will be marked down accordingly. Please make sure your name is on each page of the submission. You do not need to restate the problem, just the problem number.

Explanations are to be given for each problem, unless the problem specifically states it is not necessary. Explanations should be clear and concise. Rambling and/or imprecise explanations will be marked down accordingly.

Problem 1. Greedy Algorithms (22 Points)

You are a thief planning to steal a collection of valuable jewels on display at the museum. There are n different jewels in total, each has a different size and black market value. The sizes are given in an array $S[1 \dots n]$ and the values are given in an array $V[1 \dots n]$. Let $[n] = \{1, \dots, n\}$ denote the set of jewels. Unfortunately you cannot steal all the jewels, and instead can only take what will fit in your robber's bag which has size b (and a big dollar sign on the front of course). Specifically, the sum of the sizes of the items you choose should be less than or equal to b. Naturally, you wish to steal a subset $J \subseteq [n]$ of jewels which maximizes the total sum of values subject to fitting in your bag. You can assume b as well all values in S and V are positive.

At first you are worried as you recall this is the well known NP-hard Knapsack problem (or rather you will recall, once we cover NP-hardness). Then you remember you have a handy dandy jewel cutter, which cuts jewels to any desired fraction. Specifically, for any $c \geq 1$, if you take a $1/c$ fraction of the ith jewel then the size is $S[i]/c$ and the value is $V[i]/c$.

(a) (4 points) Give a short description of the greedy algorithm for the jewel thief problem when you are allowed to take any fractional amount of each jewel (or the whole jewel).

(b) (7 points) Prove that your greedy algorithm is correct.

Recall the class scheduling problem, where one is given starting and ending times of all the available classes on a given day, and the goal is to select the largest number of classes whose scheduled times do not overlap. In class we proved that the greedy algorithm which selects the class ending first, removes classes conflicting with this selection, and then repeats, is optimal. Instead consider the greedy strategy which selects the class which conflicts with the fewest other classes, removes classes conflicting with this selection, and then repeats.

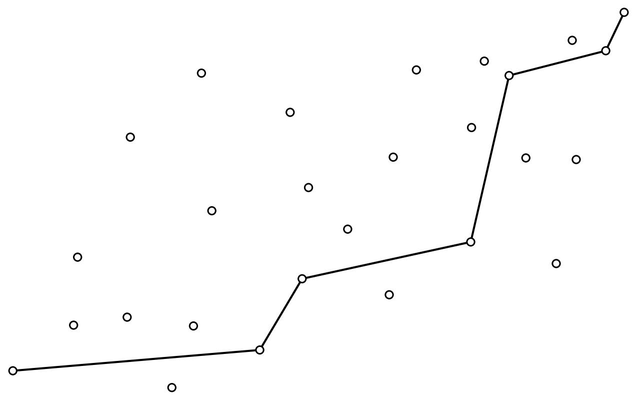
(c) (6 points) Prove this greedy strategy is NOT optimal, by giving the starting and ending times of a set of classes where this strategy fails to produce an optimal class schedule. You must clearly state for your set of classes what the optimal solution is and what classes the greedy algorithm selects.

You are give a text file containing only the characters $\{a, b, c, d, e, f\}$. Let $F(x)$ denote the frequency of a character x. Suppose that: $F(a) = 13$, $F(b) = 4$, $F(c) = 6$, $F(d) = 17$, $F(e) = 2$, and $F(f) = 11$.

(d) (5 points) Give a Huffman code for the above set of frequencies, i.e. specify the binary encoding for each of the six characters.

Problem 2. Longest Monotonic Polygonal Path (24 points)

A polygonal path is a sequence of line segments joined end-to-end; the endpoints of these line segments are called the vertices of the path. The length of a polygonal path is the sum of the lengths of its segments. A polygonal path with vertices $(x_1, y_1), (x_2, y_2), \ldots, (x_k, y_k)$ is monotonically increasing if $x_i < x_{i+1}$ and $y_i < y_{i+1}$ for ever index i. Informally, each vertex of the path is above and to the right of its predecessor.



Suppose you are given a set S of n points in the plane, represented as two arrays $X[1 \ldots n]$ and $Y[1 \ldots n]$. Describe and analyze an algorithm to compute the length of the maximum-length monotonically increasing path with vertices in S. Assume you have a constant time subroutine $LENGTH(x, y, x', y')$ that returns the length of the segment from (x, y) to (x', y').

Hint:

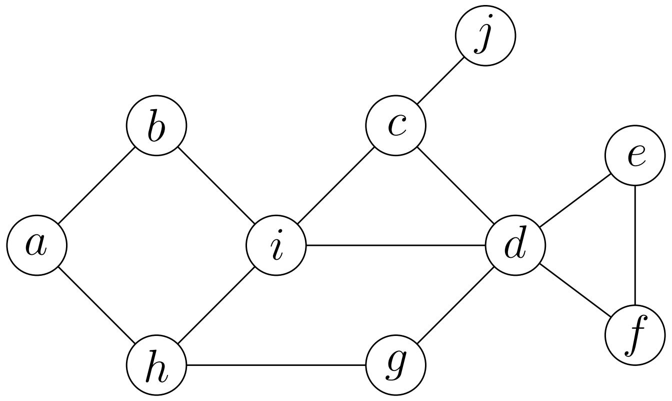
- Model the problem as a graph.
- Write a recursive function which given any point returns the length of the maximum length monotonically increasing path stating at that point.
- Argue your graph has a nice structure that allows you to easily and efficiently apply dynamic programming to your recursive function.

Problem 3. Running the Algorithms (18 points)

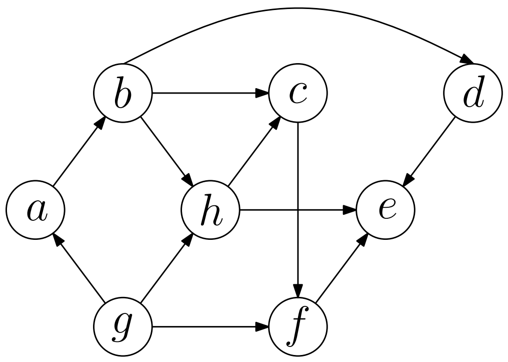
No explanation is required for any part of this problem.

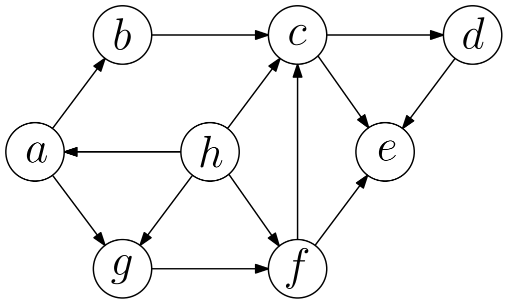
Some parts of this problem require drawing graphs. The figures below were created using the ipe drawing editor, and I recommend this for your figures since it is free, easy to use, and makes nice figures. Otherwise, feel free to hand draw and scan your graphs.

(a) (4 points) Draw the DFS and BFS tree of the following graph, when called from vertex a. There is ambiguity in the order which neighboring vertices are processed, so to make the trees consistent, process neighbors in alphabetical order. In particular, for BFS neighbors are put in the queue in alphabetical order, and for DFS recursive calls to neighbors are made in alphabetical order.

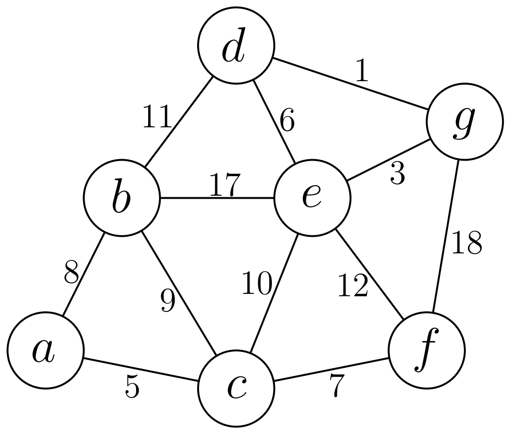


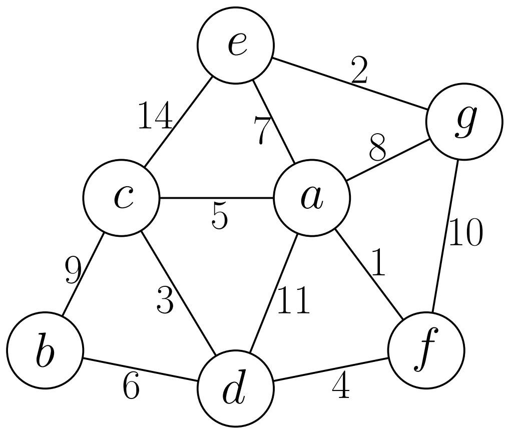
(b) (4 points) Give a topological ordering of the vertices for each of the following two graphs. You can either draw each graph on a horizontal line with vertices in topological order and all edges going from the left to right, or if it easier you can just simply list the vertices in topological order.





(c) (4 Points) Draw the MST of the following two graphs with the edges labeled by weight. Alternatively, you can just list the set of weights of the edges in the MST.





(d) (3 Points) Give the weight of the fifth edge added when running Prim's algorithm for each of the two graphs above, when starting from vertex a.

(e) (3 points) Give the weight of the fifth edge added when running Kruskal's algorithm for each of the two graphs above.

Problem 4. Bottleneck Paths (24 points)

You are given an undirected, weighted, and connected graph $G = (V, E)$. For simplicity assume all edge weights are distinct. Consider a path between two vertices s and t. The *width* of this path is maximum weight of any edge in the path. The *bottleneck shortest path* between s and t is a path with smallest possible width, and the width of such a path is called the *bottleneck distance* between s and t.

(a) Prove that the minimum spanning tree of G contains a bottleneck shortest path between every pair of vertices in G.

(b) Give an $O(|V| + |E|)$ time algorithm which answers the following question. Given a target weight W, is the bottleneck distance between s and t at most W?

(c) Let B be the bottleneck distance between s and t.

(i) Prove that deleting any edge with weight $> B$ does not change the bottleneck distance between s and t.

(ii) Prove that contracting (see definition below) any edge with weight $< B$ does not change the bottleneck distance between s and t.

(d) Give an $O(|V| + |E|)$ time algorithm to compute the bottleneck shortest path between s and t. [Hint: Start by finding the median weight edge in G]

Definition: Given an edge uv in G, *contracting* uv is the operation which adds a new vertex w, deletes vertices u and v, and sets the adjacent vertices of w to be those which u and v were adjacent to. That is, for any vertex $z \neq u, v$, the edge wz is added if and only if either uz or vz was in G before the contraction. For an edge wz which is added, we set its weight to be $weight(uz)$ if only uz was in G, $weight(vz)$ if only vz was in G, and $\min\{weight(uz), weight(vz)\}$ if both uz and vz were in G.

You can assume for the problem above that you have a subroutine $contract(F, G)$, which given any subset $F \subseteq E$, contracts all edges in F in $O(|V| + |E|)$ time.

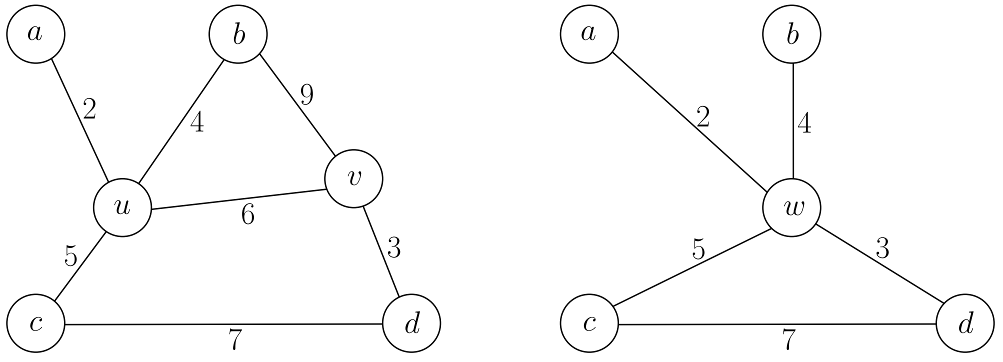


Figure 1: Left: original graph. Right: graph after contracting uv.

Problem 5. Updating Minimum Spanning Trees (12 points)

Problem 6b in Chapter 20 of Jeff's notes:

Let G be an edge-weighted, undirected, and connected graph. You are already given the MST, T, of G. Give an algorithm to update the MST when the weight of a single edge $e \in G$ is increased, producing a new graph G'. Specifically, your are given the edge e and its new weight, and your algorithm should modify T so that it is an MST in G', and do so in $O(|V| + |E|)$ time.

Useful Fact from class: The min weight edge across any vertex bi-partition must be in the MST. Moreover, (the removal of) any edge e in the MST defines a vertex bi-partition, call it $B(e)$, and as e is the only edge from the MST which goes across $B(e)$, e must be the minimum weight edge across $B(e)$ in G.