CS 580 Algorithm Design, Analysis, And Implementation
Fall 2021 Vassilis Zikas HW 2

Due Friday Oct. 1 at 11:59 p.m.

1. (a) (5 + 5+ 5 points) Decide whether each of the following is true or false. If it is true, give a short explanation. If it is false, give a counterexample.

   i. Suppose we are given an instance of the Shortest $s-t$ Path Problem on a directed graph G. We assume that all edge costs are positive and distinct. Let P be a minimum-cost $s-t$ path for this instance. Now suppose we replace each edge cost c_e by its square, c_e^2, thereby creating a new instance of the problem with the same graph but different costs. True or false? P must still be a minimum-cost $s-t$ path for this new instance.

   ii. Suppose we are given an instance of the Minimum Spanning Tree Problem on a graph G, with edge costs that are all positive and distinct. Let T be a minimum spanning tree for this instance. Now suppose we replace each edge cost c_e by its square, c_e^2, thereby creating a new instance of the problem with the same graph but different costs. True or false? T must still be a minimum spanning tree for this new instance.

   iii. Consider the Interval Scheduling Problem from class. We know that we can obtain the optimal solution by scheduling jobs in increasing order of finish times. Say you are given a schedule R, and you've found an optimal solution for the same, let us call this optimal solution as S. Now suppose that the interval durations of each of the jobs is squared (we do this by extending the finish times of each job), let us call the resulting schedule as R'. Would the optimal solution S for the schedule R still be an optimal solution for the new schedule R'?

   (b) (20 points) Alex is playing a video game in which he is in a big maze filled with m passages (edges) denoted by $[1,\ldots,m]$. He has to get from the entrance to the exit using these passages. However, each such passage has a possibility of disappearing before the start of the game, in which case the passage can no longer be used by Alex during the game. For each passage i we associate a number $p_i \in [0,1]$ which denotes the probability that the passage will *not* disappear during the game. The layout of the maze and these passage probabilities are known to Alex at the start of the game, help Alex by finding the most promising path i.e. a path from the entrance to the exit with the highest probability of not disappearing. Note that once the game is started, the passages no longer change/disappear.
   Hint: Let a path from entrance to exit consist of the passages $(2,4,m-1)$. Then, probability that the path does not disappear during the game is $p_2 p_4 p_{m-1}$.

Answer:

(a) **False**. Let G have edges $(s,v),(v,t),(s,t)$, where the first two of the these edges have cost 3 and the third has cost 5. Then the shortest path is the single edge (s,t), but after squaring the costs the shortest path would go through v.

(b) **True**. If we feed the costs c_e^2 into Kruskal's algorithm, it will sort them in the same order, and hence put the same subset of edges in the MST. We only care about the relative order of the costs, and not their actual values.

(c) **False**. No S would not also be an optimal solution to the new schedule R'. We can easily construct a counter-example to show the same.

(d) Our goal is to find a path from the entrance to the exit with the highest probability of not disappearing. Let us denote passage i as a edge e_i with weight p_i. Intersection of two or more passages form vertices in the graph. Consider any path P formed by the passages $e_{k_1}, e_{k_2}, \ldots e_{k_k}$. Then probability that this path does not disappear is

$$\prod_{i=1}^{k} p_{k_i}$$

Our goal is to maximize this probability over all paths from entrance to exit. Let S denote the set of all such paths, then we need to find the following

$$\text{argmax}_{P \in S} \prod_{e \in P} p_e.$$

This is equivalent to

$$\text{argmax}_{P \in S} \log \prod_{e \in P} p_e = \text{argmax}_{P \in S} \sum_{e \in P} \log p_e = \text{argmin}_{P \in S} \sum_{e \in P} -\log p_e$$

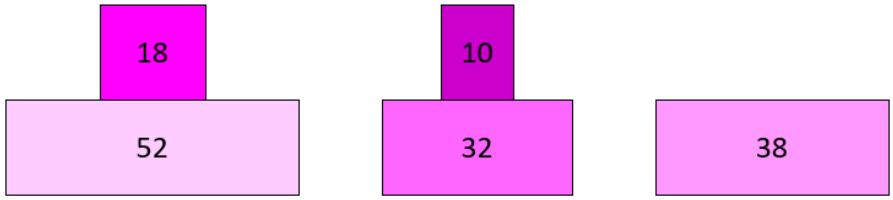
i.e. we need to find the shortest path in the given graph (maze) but with the edge weights equal to $-\log p_e$.

Now since $\forall e,\ p_e \in [0,1),\ -\log p_e \in [0,\infty)$. Therefore the edge weights of the transformed graph are positive and we can apply Dijkstra's algorithm from s to find the shortest path. Time complexity would be same as Dijkstra.

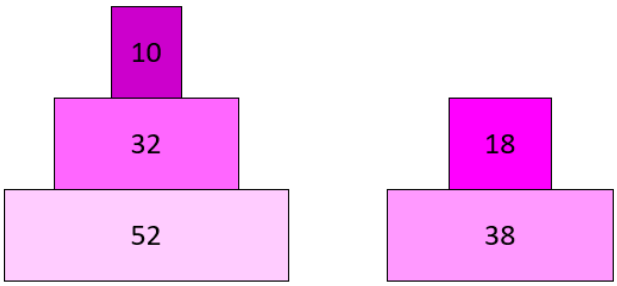
2. (15 + 20 points) **Greedy Algorithm.** Jane is a baker and she bakes cakes of varying sizes for her customers. She has to pack the cakes in boxes and send them out for delivery at the end of the day. She bakes the cakes in order of when the customer places their order, and she must pack the cake before starting the next cake. She can use as many boxes as she needs, but the delivery cost is dependent on the number of boxes she uses. She can stack several cakes in the same box, but she cannot stack a larger cake onto a smaller cake or it will collapse. Our objective is to help Jane minimize her deliverly cost, in other words the number of boxes she needs to use.

More precisely the input to our algorithm is an ordered list of cakes' sizes e.g., [52, 32, 38, 10, 18]. The output is a list of valid stacks of cakes. A valid stack should be non-collapsing e.g., the stack $(32, 38)$ would collapse since the larger cake (size 38) is on top of the smaller one (size 32). Similarly, a valid stack should respect the original ordering of cakes e.g., the stack $(38, 32)$ is invalid because the cake with size 38 would have been baked/packed after the cake with size 32. The goal is to minimize the number of stacks of cakes. For this problem we will assume that there is no limit on the height of a stack.

Since cakes must be stacked in order, we could, for example, stack cake 1 on box 1, cake 2 on box 2, cake 3 on box 3, cake 4 on box 2, and cake 5 on box 1 as shown below.



However, the optimal solution is 2 boxes as follows.



(a) Devise a greedy algorithm which returns a packing of the cakes that minimizes her delivery cost. Analyze the time and space complexity of your algorithm.

(b) Prove the correctness of your algorithm i.e. prove that the number of boxes used by your greedy algorithm is no more than number of boxes used by any optimal algorithm.

Answer: Greedy choice: Minimize number of stacks. For each cake, put the cake on the first stack in which it is smaller than the top cake, or put it on a new stack if no such stack exists. Keep doing this until all cakes are stacked. This is making a situationally optimal choice at each step, thus it is a greedy solution.

The greedy solution would take $\Theta(n \log n)$ time in the worst-case with n being the number of cakes, as a binary search will be performed in each round to select the stack for each cake (the top of the stacks are guaranteed to be in sorted order), and there could potentially be $\Omega(n)$ stacks. If binary search is not used, then the run-time would be $O(n^2)$. (Best Case Running Time: If the cakes are given in sorted order then the greedy solution time complexity would run in time $O(n)$ as all cakes are stacked in one box. This observation is not required for full credit.)

The algorithm would use $\Theta(n)$ space because we would need space to store the stacks, which would contain n elements in total corresponding to the cakes.

Proof of correctness: First, we note that in our algorithm, we never return an invalid stacking, as we will never stack a cake on top of a smaller cake as described in the algorithm.

We also note that using our algorithm, the top cakes of the stacks will be in sorted ascending order at any point in the algorithm. We can show this inductively. A single box with a single cake is sorted, so the base case is covered. Assuming that the tops of the stacks in the stacking of the first k cakes is sorted, we know that our algorithm places the $k+1$st cake on the first pile on which it fits, so it will be larger than the tops of all piles to the left and smaller than the tops of all piles to the right. Hence, the stacking with $k+1$ cakes is also sorted. By induction, this means the top cakes of the stacks will always be sorted.

Now we show that our algorithm is optimal (returns the stacking using the least number of boxes). Suppose our algorithm returns a solution, S, which differs from the optimal solution, S_{OPT}. That is, S_{OPT} contains a stacking of the cakes that is different from the stacking in S. The cakes must be stacked in order, so let c be the first cake that is stacked differently in S than S_{OPT}. We consider two cases:

(a) If c is placed in a new box in S, then by the algorithm, it could not have fit on any of the existing stacks, so S_{OPT} cannot have placed c on an existing stack. Hence, this case is impossible.

(b) If c is placed on an existing stack, b_i, in S, but it was placed on a different stack, b_j, in S_{OPT}, then that means b_j must be to the right of b_i. This is because if b_j was to the left of b_i, then our algorithm would have placed c on b_j instead since it places the cake on the first stack on which it fits. However, since the tops of the stacks are in sorted ascending order, either b_j is a new stack or the top cake of b_j would be at least as large as the top cake of b_i, so any subsequent cakes placed on top of b_i in S_{OPT} could also fit on b_j in S. Hence, we could swap c in S_{OPT} to stack b_i and the solution will still be optimal.

Since in every case, the first differing placement of a cake in S_{OPT} could be swapped to match the placement in S for an equal or more optimal solution, we could swap every cake placement in S_{OPT} to match S. This means S is at least as optimal as S_{OPT}, so our algorithm is optimal. □

3. (30 points) Consider a *directed* graph $G : (V,E)$ with non negative edge weights in which each edge is also assigned a color, either white, black, or pink. Given $s,t \in V$, our objective is to find the length of the *shortest walk* (least cost) from s to t such that edges in the path have colors repeated in the following order: (white, black, pink). For instance, a s-t walk in which edges have the colors (white, black, pink, white, black) or (black, pink, white, black, pink, white) is valid. Note that a walk (as opposed to a path) allows repetition of vertices.

Hint: Create a new graph $G' : (V',E')$ such that V' includes three copies of each vertex in V and $|E'| = |E|$. Think about how to add edge $(u,v) \in E$ to E' based on its color.
Answer: Create the graph H by transforming G as follows. First create three copies of V denoted by V_1, V_2, V_3. u_i refers to the copy of $u \in V$ in V_i. For each directed edge $e : (u,v) \in E$,

(a) If e is white, add the edge (u_1, v_2) with weight same as e.

(b) If e is black, add the edge (u_2, v_3) with weight same as e.

(c) If e is pink, add the edge (u_3, v_1) with weight same as e.

Now connect a new node s_0 to the nodes s_1, s_2 and s_3 by directed edges of weight 0 and the nodes t_1, t_2 and t_3 to a new node t_0 with edges of weight 0. Required shortest s-t walk is the shortest s_0-t_0 path in H. This can be seen as follows. Consider a shortest walk in G which follows the given color restriction. Then we can create a s_0-t_0 path in H as dictated by the edges and their colors in the walk. This path cannot be shorter than the shortest path in H i.e. shortest s-t walk in G that follows the color restriction must be greater than or equal to shortest path in s_0-t_0 path in H. Similary we can show the other direction i.e. shortest s-t walk in G that follows the color restriction must be less than or equal to shortest path in s_0-t_0 path in$H$. So, we run Dijkstra from s_0 in graph H. Correctness of the algorithm follows from correctness of Dijkstra. Time complexity is same as Dijkstra – $O(|E| + |V| \log |V|)$.

4. **(*) 1st Bonus Problem.** (10 + 20 points)—*Recall that this problem will be graded separately, and the grade will not count towards the grade of this homework. The solution to this problem, along with two more bonus problems to be included in later homeworks, will count towards an extra 10% on the final grade of the class.*

(a) Let T_1 and T_2 be two spanning trees, and let $e \in T_1 \setminus T_2$ be an edge in T_1 but not T_2. Prove that there exists an edge $d \in T_2 \setminus T_1$ such that $T_2 - d + e$ and $T_1 - e + d$ are both spanning trees. (Here "+" denotes adding an edge and "-" denotes deleting an edge.)

(b) Let T_1 and T_2 be two spanning trees. Show that there exists a one-to-one mapping $f : (T_2 \setminus T_1) \to (T_1 \setminus T_2)$ such that for every $e \in T_2 \setminus T_1$, the three $T_1 - f(e) + e$ is a spanning tree.

Answer:

(a) Let T_1 and T_2 be two distinct spanning trees. For ease of exposition let $S_1 = T_1 \setminus T_2$ and $S_2 = T_2 \setminus T_1$ (note that these are sets of edges). Since T_1 and T_2 are distinct, we have that $S_1, S_2 \neq \phi$ i.e. are non-empty. Consider any $e \in S_1$. The subgraph spanned by $T_1 \setminus \{e\}$ consists of two disconnected components (otherwise there would have been a cycle contradicting the definition of a tree). Consider the cut formed by these disconnected components say (A, B) (note that A and B are disjoint vertex sets). Now, since T_2 is a spanning tree of the same graph as T_1, there must exist an edge $e' \in T_2$ such that e' has one endpoint in A and the other in B (otherwise T_2 would not be spanning). Since $e \in S_1$, $e' \neq e$. Furthermore, e is the only edge in T_1 that goes across the (A, B) cut, thus $e' \notin T_1$. This implies that $e' \in S_2$. $T_1 \setminus \{e\} \cup \{e'\}$ thus gives us a spanning tree.

(b) We will prove the existence of a one-to-one mapping $\Phi : (T_2 \setminus T_1) \to (T_1 \setminus T_2)$ by induction on $|T_2 \setminus T_1|$.

Base case: $|T_2 \setminus T_1| = 0$. The claim is trivially true.

Inductive Hypothesis: For any two spanning trees T_1 and T_2, if $|T_2 \setminus T_1| = k - 1$, then there exists a one-to-one map $\Phi : (T_2 \setminus T_1) \to (T_1 \setminus T_2)$ such that for every $e \in T_2$, the tree $T_1 - e + \Phi(e)$ is a spanning tree.

Let T_1 and T_2 be spanning trees such that $|T_2 \setminus T_1| = k$. consider any $e \in T_2 \setminus T_1$. By problem 1, we know that there exists an edge $d \in T_1 \setminus T_2$ such that $T_3 = T_2 - e + d$ is a spanning tree. Now, $|T_3 \setminus T_1| = k - 1$ by construction. Thus, by our inductive hypothesis, there exists a one-to-one mapping $\Phi : (T_3 \setminus T_1) \to (T_1 \setminus T_3)$ such that for every edge $e' \in T_1$, the tree $T_1 - e' + \Phi(e')$ is a spanning tree. Define a new mapping $\Phi^* : (T_2 \setminus T_1) \to (T_1 \setminus T_2)$ as follows:

$$\Phi^*(e') = \begin{cases} \Phi(e') & \dots \text{ if } e' \neq e \\ d & \dots \text{ if e' = e} \end{cases}$$

Φ^* is a one-to-one mapping as $e \notin (T_3 \setminus T_1)$ i.e. the domain of Φ, $d \notin T_1 \setminus T_3$ i.e. the range of Φ, and Φ is a one-to-one mapping.

By our above discussion, we already have that e and $\Phi^*(e) = d$ are an exchangeable pair. Furthermore, for all $e' \in (T_2 \setminus T_1)$ such that $e' \neq e$, we have that $T_2 - e' + \Phi^*(e') = T_2 - e' + \Phi(e')$ is a spanning tree by our inductive hypothesis. The claim follows by the principle of mathematical induction.

5. **2 points** Have you assigned pages on Gradescope? **Answer:** Yes!