CS-GY 6033: Design and Analysis of Algorithms I

Fall 2024

Instructor: Nairen Cao **NYU Tandon School of Engineering**

Problems	1	2	3	4	5 (ungraded)	Total
Max. Score	25	25	25	25	0	100

Table 1: Score Summary

Homework #5

Assigned: 11/12/2024 **Due: 11/26/2024**

General Instructions Write all solutions clearly, concisely, and legibly. It is okay to provide handwritten and scanned/photographed solutions as long as they are clearly legible. If typing, using LaTeX is recommended.

You should generally describe algorithms in words and high-level pseudocode. Specifically, emphasize the goal of each step in your algorithm rather than the coding details. After giving a high-level description, provide more detail on the implementation of each step where appropriate.

The purpose of having these problem sets is to enhance your mathematics and programming skills in the scope of designing algorithms. These problems may be quite challenging, so discussions with classmates are encouraged. However, you should spend at least half an hour thinking about the problems individually before discussing them. Although discussions (and consulting ChatGPT and WolframAlpha) are allowed, you need to write the solution in your own words and by yourself. Please acknowledge any person or reference you discuss with or consult from.

Special Instructions

- You can use Kruskal's algorithm, Prim's algorithm, Borůvka's algorithm, Dijkstra's algorithm, and Bellman-Ford algorithm as a black-box. All algorithms that appear on the slides can be used directly.
- **Theme for Homework:** Minimum Spanning Tree, Potential Function, Shortest Paths.

Problem 5-1. MST exercises

(a) Prove or disprove (by counterexample): if the edge weights are distinct, then there is a unique minimum spanning tree.

(b) Given an undirected graph $G = (V, E)$ and a minimum spanning tree T, suppose that we decrease the weight of one of the edges $(u, v) \in E$ that is not in T. Give an algorithm for finding the minimum spanning tree in the modified graph. Your algorithm should run in $O(V + E)$ time, i.e., not computing an MST from scratch. *Hint: can you find a cycle in* $T \cup \{(u, v)\}$*?*

(c) Suppose that a graph $G = (V, E)$ has a minimum spanning tree T already computed. How quickly can we update the minimum spanning tree if we modify G by adding a new vertex and its incident edges? In particular, can you come up with an algorithm that does not look at every edge in the graph?

Problem 5-2. MSTs on very sparse graphs

Suppose you are given as input a connected undirected graph $G = (V, E)$ with distinct real edge weights $w(u, v)$ for each edge $(u, v) \in E$. By distinct weights, I mean that no two edges have the same weight. Suppose also that the graph is extremely sparse, specifically that $|E| \leq n + 3$, where $n = |V|$. This level of sparsity means that the graph is very close to (only 4 edges away from) being a tree already.

(a) What are the worst-case asymptotic running times of Prim's and Kruskal's algorithms on this very sparse graph type as a function of $n = |V|$? Briefly justify your answer. *I'm really asking: given the restricted graph type, do these algorithms perform better than the general bounds we learned in class?*

The MST algorithms we learned all leverage the cut property, i.e., that the lightest edge crossing a cut belongs to some MST. For this problem, we'll instead leverage the cycle property:

Lemma 1 (Cycle Property) *Let $G = (V, E)$ be a connected undirected graph with distinct edge weights. Let C be any cycle in the graph, and let (u, v) be the edge with highest weight on the cycle C. Then (u, v) is* not *part of any MST.*

(b) Prove the cycle property (Lemma 1) as stated, i.e., with the assumption of distinctness.

(c) Design a new MST algorithm for this very sparse graph type that runs in $O(n)$ time. The algorithm should be fairly simple and make use of the previous problem parts.

Problem 5-3. Exercise on Shortest Paths

The following exercises are unrelated.

(a) Suppose that a weighted, directed graph (V, E) has a negative-weight cycle that is reachable from some designated source vertex $s \in V$. Give an efficient algorithm to list the vertices of one such cycle and analyze the running time. *If you want to use any algorithms you already know, such as Bellman-Ford or DFS, you should apply them as a black box (i.e., don't explain how they work). The algorithm I'm looking for should be quite short given appropriate subroutines. But it's not obvious. I'm not asking you to argue correctness, but you should think about how you would do so. If you are unable to come up with even a hand-wavy correctness argument, there's a good chance your algorithm doesn't work.*

(b) We are given a directed graph $G = (V, E)$ on which each edge $(u, v) \in E$ has a associate value $r(u, v)$, which is a real number in the range $0 < r(u, v) \leq 1$ that represents the reliability of a communication channel from vertex u to vertex v. We interpret $r(u, v)$ as the probability that the channel from u to v will NOT fail. The ***reliability*** of the path $p = \langle v_0, v_1, \ldots, v_k \rangle$ is the probability that none of the edges along the path fails. We assume that edges failing is independent, so the reliability of the path is given by $r(p) = \prod_{i=1}^{k} r(v_{i-1}, v_i)$. Conversely, the probability that some edge on the path fails is $1 - r(p) = 1 - \prod_{i=1}^{k} r(v_{i-1}, v_i)$.

Give an efficient algorithm to find the most reliable path between two given vertices, i.e., the path with lowest failure probability. *Model this as a shortest path problem. You have to specify (1) the graph (which should be trivial), (2) the edge weights, and (3) which shortest-path algorithm you should use.*

Figuring out the edge weights is the hard part here. You will want to leverage the following facts about logarithms: (1) log is monotonically increasing, i.e., if $x > y > 0$ then $\log(x) > \log(y)$, and (2) log of products becomes the sum of logs, i.e., $\log(xy) = \log(x) + \log(y)$. Keep in mind also that for $0 < x < 1$, $\log(x) < 0$.

(c) Suppose that we are given a weighted, directed graph $G = (V, E)$ in which edges that leave the source vertex s may have negative weights, all other edge weights are nonnegative, and there are no negative-weight cycles. Argue that Dijkstra's algorithm correctly finds shortest paths from s in this graph.

Problem 5-4. Balancing trees by rebuilding

This problem is doing a different balancing mechanism for binary search trees. Ignore anything you know about other balanced search trees, e.g., red-black trees, B-trees, AVL trees, etc. None of that is relevant here. However, you should know the basics of an (unbalanced) binary search tree as prerequisite for this course.

Consider an ordinary binary search tree augmented by adding to each node x the attribute $x.size$ giving the number of keys stored in the subtree rooted at x (inclusive). Let α be a constant in the range $1/2 \leq \alpha < 1$. We say that a given node x is α-***balanced*** if $(x.left).size \leq \alpha \cdot x.size$ and $(x.right).size \leq \alpha \cdot x.size$, i.e., neither subtree is more than an α fraction of the total. The tree as a whole is α-***balanced*** if every node in the tree is α-balanced. The following amortized approach to maintaining weight-balanced trees was suggested by G. Varghese.

(a) A $1/2$-balanced tree is, in a sense, as balanced as it can be. Given a node x in an arbitrary binary search tree, show how to rebuild the subtree rooted at x so that it becomes $1/2$-balanced. Your algorithm should run in time $\Theta(x.size)$, and it can use $O(x.size)$ auxiliary storage.

Hint: you'll probably want to use a size-$(x.size)$ array here as auxiliary storage. You'll also probably want to start your algorithm with an in-order tree walk.

(b) Show that an n-node α-balanced binary search tree has $O(\lg n)$ height. (Recall α is a constant.)

It would be equivalent to show that a search in an n-node α-balanced binary search tree takes $O(\lg n)$ time in the worst case, so you may do that if you prefer. However, it's more convenient to refer to height in a later problem part.

For the remainder of this problem, assume that the constant α is strictly greater than $1/2$. Suppose that we implement INSERT and DELETE as usual for an n-node binary search tree, except that after every such operation, if any node in the tree is no longer α-balanced, then we "rebuild" the subtree rooted at the highest such node in the tree so that it becomes $1/2$-balanced.

We shall analyze this rebuilding scheme using the potential method. For a node x in a binary search tree T, we define

$$\Delta(x) = |(x.left).size - (x.right).size| \ ,$$

and we define the potential of T as

$$\Phi(T) = c \sum_{x \in T : \Delta(x) \geq 2} \Delta(x) \ ,$$

where c is a sufficiently large constant that depends on α. Note that we are only summing over the nodes where there is at least a little imbalance (specifically $\Delta(x) \geq 2$), which is important.

(c) Argue that any binary search tree has nonnegative potential and that a $1/2$-balanced tree has potential 0.

(d) Suppose that m units of potential can pay for rebuilding an m-node subtree. How large must c be in terms of α in order for it to take 0 amortized time to rebuild a subtree that is not α-balanced? Don't use big-O notation here. *In other words, how large must c be in order for the change in potential caused by rebuilding to be sufficient to pay for the rebuild? Note the assumption that the tree is not α-balanced before the rebuild — that's important.*

(e) Show that inserting a node into an n-node α-balanced tree costs $O(\lg n)$ amortized time. *Hint: it's easier if you break up the insertion into steps: perform the regular BST insert, pay for an changes to potential, and pay for any subtree rebuilding.*

Problem 5-5. (Ungraded) Bottleneck spanning tree

Let $G = (V, E)$ be a connected undirected graph with edge weights. For any spanning tree T of G, the bottleneck of T is the maximum edge weight in T. A minimum bottleneck spanning tree (MBST) of G is a spanning tree with minimum bottleneck over all spanning trees of G. The bottleneck value of a graph G is the bottleneck value of an MBST of G.

(a) Argue that a minimum spanning tree is also a MBST.

(b) Give an $O(E)$-time algorithm that, given an input graph G and real value r, determines whether the bottleneck value of G is at most r.

(c) Suppose that each edge has a distinct integer weight in the range $1, 2, \ldots, |E|$. Use your algorithm from part (b) as a black-box subroutine to find the bottleneck value of G in $O(E \log E)$ time.

(d) Improve your algorithm from the previous part to find the bottleneck value of G in $O(E)$ time. You may use a subroutine for contracting a set of edges in linear time as a black box, e.g., as used in Borůvka's algorithm. Argue that your algorithm runs in $O(E)$ time.

(e) In the previous parts, you assumed edge weights were restricted to be 1) distinct and 2) integers in a small range. Let's now remove both of these restrictions. Describe how to change your algorithm so that it still runs in $O(E)$ time when the edge weights are arbitrary real numbers.