CS 170: Homework #3

Due on February 9, 2026 at 10:00pm

L. Chen and U. Vazirani, Spring 2026

Zachary Brandt

1 Skyline (Solo Question; 10 points)

Given a list of the n buildings in a city on flat ground, you want to compute the skyline. Each building has a rectangular cross-section specified by integers ℓ, r, h, where $\ell < r$ specify the left and right ends of the building, and h specifies the height.

The skyline, or the entire area covered by the buildings, may be specified by integers $p_0 < \cdots < p_m$ and $h_1, \ldots, h_m$, meaning that the tallest building between positions p_{i-1} and p_i has height h_i, and no buildings lie to the left of p_0 or to the right of p_m.

Give an efficient divide-and-conquer algorithm to compute the skyline, and analyze the running time. You may assume that each integer operation takes $O(1)$ time.

Hint: Think about merge sort. Can you efficiently (in linear time) merge two skylines?

Solution

2 Testing Ancestors (5 points)

You are given a binary tree $T = (V, E)$ (in adjacency list format), along with a designated root node $r \in V$. Recall that u is said to be an *ancestor* of v in the rooted tree if the path from r to v in T passes through u.

You wish to preprocess the tree so that queries of the form ”is u an ancestor of v?” can be answered in constant time. The preprocessing itself should take linear time. How can this be done?

Solution

3 Finding Ancestors (5 points)

You are given a tree $T = (V, E)$ with designated root node r, along with a positive integer K. Design an $O(|V|)$ time (not $O(K \cdot |V|)$ time) algorithm that outputs the Kth ancestor (if it exists) of every vertex. Recall that the Kth ancestor of vertex v is the vertex obtained by walking K steps from v towards the root.

Solution

4 Trees (10 points)

Let $T = (V, E)$ be an n-vertex tree, that is, a graph for which every pair of vertices is connected by a unique path.

(a) (5 points) If we remove a vertex of degree d, then T is fragmented into d connected subtrees. Give an $O(n)$ time algorithm that takes as input $T = (V, E)$, and outputs a vertex whose removal leaves no connected subtrees containing $> n/2$ vertices. Also explain why such a vertex always exists. Is it always unique?

Hint: think DFS.

(b) (5 points) Give an $O(n \log n)$ time algorithm that takes as input $T = (V, E)$ and an integer K, and outputs the number of pairs of vertices in V that are connected by a path of length K. You may assume that every vertex in T has degree ≤ 3 (but this assumption is not necessary!).

Hint: use your algorithm from part (a).

Solution

5 Vertex Separators (5 points)

Let $G = (V, E)$ be an undirected, unweighted graph with $n = |V|$ vertices. We call a set of vertices S a $u - v$ *separator* if (1) S does not contain u or v and (2) every path from u to v goes through some vertex in S.

Give an efficient algorithm that takes G, u, v as input and finds a $u - v$ separator of size at most $\frac{n-2}{d-1}$, where d is the length of the shortest path from u to v. Assume that u and v are connected in G and $d > 1$.

Hint: Is there a natural way to partition (some of) the vertices that aren't u, v into $d - 1$ sets?

Solution

6 Counting Shortest Paths (10 points)

Given a graph $G = (V, E)$ and a vertex s, let $p(v)$ be the number of distinct shortest paths from s to v; we let $p(s) = 1$. Design and analyze an efficient algorithm that outputs a list containing $p(v)$ for every vertex v, under the following assumptions:

(a) (5 points) G is undirected and unweighted.

(b) (5 points) G is directed with positive edge weights.

Your algorithm for part (a) should be more efficient than for part (b), assuming $|E|$ is significantly smaller than $|V|^2$.

Solution

7 Strongly Connected Components (7 points)

For full credit, you should do **one of the following two** questions. (You may solve the other for fun if you want!)

1. In the 2SAT problem, you are given a set of clauses, where each clause is the disjunction (OR) of two literals (a literal is a Boolean variable or the negation of a Boolean variable). You are looking for a way to assign a value true or false to each of the variables so that all clauses are satisfied – that is, there is at least one true literal in each clause. For example, here's an instance of 2SAT:

$$(x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_1 \vee x_2) \wedge (\overline{x_3} \vee x_4) \wedge (\overline{x_1} \vee x_4)$$

   Recall that $\vee$ is the logical-OR operator and $\wedge$ is the logical-AND operator and $\overline{x}$ denotes the negation of the variable x. This instance has a satisfying assignment: set x_1, x_2, x_3, and x_4 to `true, false, false, and true`, respectively.

   The purpose of this problem is to lead you to a way of solving 2SAT efficiently by reducing it to the problem of finding the strongly connected components of a directed graph. Given an instance I of 2SAT with n variables and m clauses, construct a directed graph $G_I = (V, E)$ as follows.

   - G_I has $2n$ nodes: one for each variable and its negation.
   - G_I has $2m$ edges: for each clause $(\alpha \vee \beta)$ of I (where α, β are literals), G_I has an edge from from $\overline{\alpha}$ to β, and one from the $\overline{\beta}$ to α.

   Note that the clause $(\alpha \vee \beta)$ is equivalent to each of the implications $\overline{\alpha} \implies \beta$ and $\overline{\beta} \implies \alpha$. In this sense, G_I records all implications in I.

   (a) (2 points) Show that if G_I has a strongly connected component containing both x and $\overline{x}$ for some variable x, then I has no satisfying assignment.

   (b) (2 points) Now show the converse of (a): namely, that if none of G_I's strongly connected components contain both a literal and its negation, then the instance I must be satisfiable.

   Hint: Pick a sink SCC of G_I. Assign variable values so that all literals in the sink are True. Why are we allowed to do this, and why doesn't it break any implications?

   (c) (2 points) Use the previoius parts to construct a linear-time algorithm for solving 2SAT.

   (d) (1 point) Try generalizing your algorithm to 3SAT, where each clause has three literals, e.g. $(x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee \overline{x_3} \vee x_4)$. Briefly explain why it works (or else what goes wrong).

2. Implement DFS and the SCC-finding algorithm covered in lecture. There are two ways that you can access the notebook and complete the problems:

   (a) **On Datahub**: click here and navigate to the `hw03` folder.

   (b) **On Local Machine**: `git clone` (or if you already cloned it, `git pull`) from the coding homework repo,

   https://github.com/Berkeley-CS170/cs170-sp26-coding

   and navigate to the `hw03` folder. Refer to the `README.md` for local setup instructions.

   Notes:

   - *Submission Instructions:* Please download your completed submission `.zip` file and submit it to the Gradescope assignment titled "Homework 3 Coding Portion".

- *Getting Help:* Conceptual questions are always welcome on Edstem and office hours; *note that support for debugging help during OH will be limited.* If you need debugging help first try asking on the public Edstem threads. To ensure others can help you, make sure to:

  (a) Describe the steps you've taken to debug the issue prior to posting on Ed.

  (b) Describe the specific error you're running into.

  (c) Include a few small but nontrivial test cases, alongside both the output you expected to receive and your function's actual output.

  If staff tells you to make a private Ed post, make sure to include *all of the above items* plus your full function implementation. If you don't provide them, we will ask you to provide them.

- *Academic Honesty Guideline:* We realize that code for some of the algorithms we ask you to implement may be readily available online, but we strongly encourage you to not directly copy code from these sources. Instead, try to refer to the resources mentioned in the notebook and come up with code yourself. That being said, we **do acknowledge** that there may not be many different ways to code up particular algorithms and that your solution may be similar to other solutions available online.

Solution