# CS/ECE 374 A ☠ Fall 2023

# Homework 9

Due Tuesday, October 31, 2023 at 9pm

---

This is the last homework before Midterm 2.

---

1. You are planning a hiking trip in Jasper National Park in British Columbia over winter break. You have a complete map of the park's trails, which indicates that hikers on certain trails have a higher chance of encountering a sasquatch. All visitors to the park are required to purchase a canister of sasquatch repellent. You can safely traverse a high-risk trail segment only by *completely* using up a *full* canister. The park rangers have helpfully installed several refilling stations around the park, where you can refill empty canisters at no cost. The canisters themselves are expensive and heavy, so you can only carry one. The trails are narrow, so each trail segment allows traffic in only one direction.

   You have converted the trail map into a directed graph $G = (V, E)$, whose vertices represent trail intersections, and whose edges represent trail segments. A subset $R \subseteq V$ of the vertices indicate the locations of the *R*epellent *R*efilling stations, and a subset $H \subseteq E$ of the edges are marked as *H*igh-risk. Each edge $e$ is labeled with the length $\ell(e)$ of the corresponding trail segment. Your campsite appears on the map as a vertex $s \in V$, and the visitor center is another vertex $t \in V$.

   (a) Describe and analyze an algorithm that finds the shortest *safe* hike from your campsite $s$ to the visitor center $t$. Assume there is a refill station at your campsite, and another refill station at the visitor center.

   (b) Describe and analyze an algorithm to decide if you can safely hike from *any* refill station to *any* other refill station. In other words, for *every* pair of vertices $u$ and $v$ in $R$, is there a safe hike from $u$ to $v$?

2. You are driving through the back-country roads of Tenkucky, desperately trying to leave the state before the state's annual Halloween Purge begins. Every road in the state is patrolled by a Driving Posse who will let you exercise your god-given right to drive as fast as you damn well please, provided you pay the appropriate speed tax. The faster you traverse any road, the more you have to pay. What's the fastest way to escape the state?

   You have an accurate map of the state, in the form of a directed graph $G = (V, E)$, whose vertices $V$ represent small towns and whose edges $E$ represent one-lane dirt roads between towns.[1] One vertex $s$ is marked as your starting location; a subset $X \subset V$ of vertices are marked as exits. Each edge $e$ has an associated value $\$(e)$ with the following interpretation.

   - If you drive from one end of road $e$ to the other in $m$ minutes, for any positive real number $m$, then you must pay road $e$'s Driving Posse a speed tax of $\lceil \$(e)/m \rceil$ dollars.

[1] Paved roads are far too expensive!

- Equivalently, if you pay road $e$'s Driving Posse a speed tax of $d$ dollars, for any positive integer $d$, you are allowed to drive the entire length of road $e$ in $\$(e)/d$ minutes, but no less.

In particular, any road you drive on *at all* will cost you *at least* one dollar. Anyone who violates this rule (for example, by running out of money) will be thrown in jail, which means almost certain death in the Purge.

The Driving Posses do not accept coins, credit cards, Venmo, Zelle, or any other mobile payment app—only cold hard American paper currency—and they do not give change. Fortunately, you are starting your journey with a pile of $D$ crisp new \$1 bills.

Describe and analyze and algorithm to compute the fastest possible driving route from $s$ to any exit node in $X$. The input to your algorithm consists of the map $G = (V, E)$, the start vertex $s$, the exit vertices $X$, and the positive integer $D$. Report the running time of your algorithm as a function of the parameters $V$, $E$, and $D$.

3. ***Practice only. Do not submit solutions.***

   After a grueling midterm at the See-Bull Center for Commuter Silence, you decide to take the bus home. Since you planned ahead, you have a schedule that lists the times and locations of every stop of every bus in Sham-Poobanana. Unfortunately, no single bus visits both the See-Bull Center and your home; you must change buses at least once. There are exactly $b$ different buses. Each bus starts at 12:00:01AM, makes exactly $n$ stops, and finally stops running at 11:59:59PM. Buses always run exactly on schedule, and you have an accurate watch. Finally, you are far too tired to walk between bus stops.

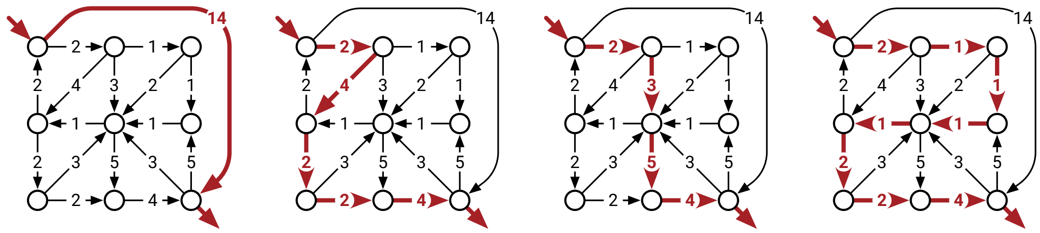
   (a) Describe and analyze an algorithm to determine a sequence of bus rides that gets you home as early as possible. Your goal is to minimize your *arrival time*, not the time you spend traveling.

   (b) Oh, no! The midterm was held on Halloween, and the streets are infested with zombies! Describe how to modify your algorithm from part (a) to minimize *the total time you spend waiting at bus stops*; you don't care how late you get home or how much time you spend on buses. (Assume you can wait inside the See-Bull Center until your first bus is just about to leave.)

   For both questions, your input consists of the exact time when the midterm ends See-Bull and two arrays $\mathit{Time}[1..b, 1..n]$ and $\mathit{Stop}[1..b, 1..n]$, where $\mathit{Time}[i, j]$ is the scheduled time of the $i$th bus's $j$th stop, and $\mathit{Stop}[i, j]$ is the location of that stop. Report the running times of your algorithms as functions of the parameters $n$ and $b$.

## Solved Problems

4. Although we typically speak of "the" shortest path from one vertex to another, a single graph could contain several minimum-length paths with the same endpoints.



Four (of many) equal-length shortest paths.

Describe and analyze an algorithm to compute the *number* of shortest paths from a source vertex $s$ to a target vertex $t$ in an arbitrary directed graph $G$ with weighted edges. Assume that all edge weights are positive and that any necessary arithmetic operations can be performed in $O(1)$ time each.

*[Hint: Compute shortest path distances from $s$ to every other vertex. Throw away all edges that cannot be part of a shortest path from $s$ to another vertex. What's left?]*

**Solution:** We start by computing shortest-path distances $dist(v)$ from $s$ to $v$, for every vertex $v$, using Dijkstra's algorithm. Call an edge $u{\to}v$ ***tight*** if $dist(u) + w(u{\to}v) = dist(v)$. Every edge in a shortest path from $s$ to $t$ must be tight. Conversely, every path from $s$ to $t$ that uses only tight edges has total length $dist(t)$ and is therefore a shortest path!

Let $H$ be the subgraph of all tight edges in $G$. We can easily construct $H$ in $O(V+E)$ time. Because all edge weights are positive, $H$ is a directed acyclic graph. It remains only to count the number of paths from $s$ to $t$ in $H$.

For any vertex $v$, let $NumPaths(v)$ denote the number of paths in $H$ from $v$ to $t$; we need to compute $NumPaths(s)$. This function satisfies the following simple recurrence:

$$NumPaths(v) = \begin{cases} 1 & \text{if } v = t \\ \displaystyle\sum_{v\to w} NumPaths(w) & \text{otherwise} \end{cases}$$

In particular, if $v$ is a sink but $v \neq t$ (and thus there are no paths from $v$ to $t$), this recurrence correctly gives us $NumPaths(v) = \sum \varnothing = 0$.

We can memoize this function into the graph itself, storing each value $NumPaths(v)$ at the corresponding vertex $v$. Since each subproblem depends only on its successors in $H$, we can compute $NumPaths(v)$ for all vertices $v$ by considering the vertices in reverse topological order, or equivalently, by performing a depth-first search of $H$ starting at $s$. The resulting algorithm runs in $O(V+E)$ time.

The overall running time of the algorithm is dominated by Dijkstra's algorithm in the preprocessing phase, which runs in $\boldsymbol{O(E \log V)}$ ***time***. ■

**Rubric:** 10 points = 5 points for reduction to counting paths in a dag (standard graph reduction rubric) + 5 points for the path-counting algorithm (standard dynamic programming rubric)

5. After moving to a new city, you decide to choose a walking route from your home to your new office. Your route must consist of an uphill path (for exercise) followed by a downhill path (to cool down), or just an uphill path, or just a downhill path. But you also want the *shortest* path that satisfies these conditions, so that you actually get to work on time.

   Your input consists of an undirected graph $G$, whose vertices represent intersections and whose edges represent road segments, along with a start vertex $s$ and a target vertex $t$. Every vertex $v$ has a value $h(v)$, which is the height of that intersection above sea level, and each edge $uv$ has a value $\ell(uv)$, which is the length of that road segment.

   (a) Describe and analyze an algorithm to find the shortest uphill–downhill walk from $s$ to $t$. Assume all vertex heights are distinct.

**Solution:** We define a new directed graph $G' = (V', E')$ as follows:

- $V' = \{v^\uparrow, v^\downarrow \mid V \in V\}$. Vertex $v^\uparrow$ indicates that we are at intersection $v$ moving uphill, and vertex $v^\downarrow$ indicates that we are at intersection $v$ moving downhill.
- $E'$ is the union of three sets:
  - Uphill edges: $\{u^\uparrow \to v^\uparrow \mid uv \in E \text{ and } h(u) < h(v)\}$. Each uphill edge $u^\uparrow \to v^\uparrow$ has weight $\ell(uv)$.
  - Downhill edges: $\{u^\downarrow \to v^\downarrow \mid uv \in E \text{ and } h(u) > h(v)\}$. Each downhill edge $u^\downarrow \to v^\downarrow$ has weight $\ell(uv)$.
  - Switch edges: $\{v^\uparrow \to v^\downarrow \mid v \in V\}$; each switch edge has weight 0.

We need to compute three shortest paths in this graph:

- The shortest path from $s^\uparrow$ to $t^\downarrow$ gives us the best uphill-then-downhill route.
- The shortest path from $s^\uparrow$ to $t^\uparrow$ gives us the best uphill-only route.
- The shortest path from $s^\downarrow$ to $t^\downarrow$ gives us the best downhill-only route.

$G'$ is a directed ***acyclic*** graph; we can get a topological ordering by listing the up vertices $v^\uparrow$, sorted by increasing height, followed by the down vertices $v^\downarrow$, sorted by decreasing height. Thus, we can compute the shortest path in $G'$ from any vertex to any other in $O(V' + E') = O(V + E)$ time by dynamic programming. (The algorithm is the same as the longest-path algorithm in the notes, except we use "min" instead of "max" in the recurrence, and define $\min \varnothing = \infty$.)

Our overall algorithm runs in $\boldsymbol{O(V+E)}$ ***time***. ■

**Rubric:** 5 points = 1 for vertices + 1 for edges + 1 for arguing $G'$ is a dag + 1 for algorithm + 1 for running time. This is not the only correct solution. Max 4 points for a correct reduction to Dijkstra's algorithm that runs in $O(E \log V)$ time.

(b) Suppose you discover that there is no path from $s$ to $t$ with the structure you want. Describe an algorithm to find a path from $s$ to $t$ that alternates between "uphill" and "downhill" subpaths as few times as possible, and has minimum length among all such paths. (There may be even shorter paths with more alternations, but you don't care about them.) Again, assume all vertex heights are distinct.

**Solution (Dijkstra, 5/5):** Let $L = 1 + \sum_{u \to v} \ell(u \to v)$. Define a new graph $G' = (V', E')$ as follows:

- $V' = \{v^\uparrow, v^\downarrow \mid V \in V\} \cup \{s, t\}$. Vertex $v^\uparrow$ indicates that we are at intersection $v$ moving uphill, and vertex $v^\downarrow$ indicates that we are at intersection $v$ moving downhill.
- $E'$ contains four types of edges:
  - Uphill edges: $\{u^\uparrow \to v^\uparrow \mid uv \in E \text{ and } h(u) < h(v)\}$. Each uphill edge $u^\uparrow \to v^\uparrow$ has weight $\ell(uv)$.
  - Downhill edges: $\{u^\downarrow \to v^\downarrow \mid uv \in E \text{ and } h(u) > h(v)\}$. Each downhill edge $u^\downarrow \to v^\downarrow$ has weight $\ell(uv)$.
  - Switch edges: $\{v^\uparrow \to v^\downarrow \mid v \in V\} \cup \{v^\downarrow \to v^\uparrow \mid v \in V\}$. Each switch edge has weight $L$.
  - Start and end edges $s \to s^\uparrow$, $s \to s^\downarrow$, $t^\uparrow \to t$, and $t^\downarrow \to t$, each with weight 0,

We need to compute the shortest path from $s$ to $t$ in $G'$; the large weight $L$ on the switch edges guarantees that this path with have the minimum number of switches, and the minimum length among all paths with that number of switches. Dijkstra's algorithm finds this shortest path in $O(E' \log V') = \boldsymbol{O(E \log V)}$ ***time***.

(Because $G'$ includes switch edges in both directions, $G'$ is not a dag, so we can't use dynamic programming directly.) ■

**Rubric:** 5 points, standard graph-reduction rubric. This is not the only correct solution with running time $O(E \log V)$.

**Solution (clever, extra credit):** Our algorithm works in two phases: First we determine the minimum number of switches required to reach every vertex, and then we compute the shortest path from $s$ to $t$ with the minimum number of switches. The first phase is can be solved in $O(V + E)$ time by a modification of breadth-first search; the second by computing shortest paths in a dag.

---

For the first phase, we define a new graph $G' = (V', E')$ as follows:

- $V' = \{v^\uparrow, v^\downarrow \mid V \in V\} \cup \{s, t\}$. Vertex $v^\uparrow$ indicates that we are at intersection $v$ moving uphill, and vertex $v^\downarrow$ indicates that we are at intersection $v$ moving downhill.
- $E'$ contains four types of edges:
  - Uphill edges: $\{u^\uparrow \to v^\uparrow \mid uv \in E \text{ and } h(u) < h(v)\}$. Each uphill edge has weight 0.
  - Downhill edges: $\{u^\downarrow \to v^\downarrow \mid uv \in E \text{ and } h(u) > h(v)\}$. Each downhill

edge has weight 0.

- Switch edges: $\{v^\uparrow \to v^\downarrow \mid v \in V\} \cup \{v^\downarrow \to v^\uparrow \mid v \in V\}$. Each switch edge has weight 1.
- Start and end edges $s \to s^\uparrow$, $s \to s^\downarrow$, $t^\uparrow \to t$, and $t^\downarrow \to t$, each with weight 0.

Now we compute the shortest path distance from $s$ to every other vertex in $G'$. We could use Dijkstra's algorithm in $O(E \log V)$ time, but the structure of the graph supports a faster algorithm.

Intuitively, we break the shortest-path computation into phases, where in the $k$th phase, we mark all vertices at distance $k$ from the source vertex $s$. During the $k$th phase, we may also discover vertices at distance $k+1$, but no further. So instead of using a binary heap for the priority queue, it suffices to use two bags: one for vertices at distance $k$, and one for vertices at distance $k+1$.

```
ZeroOneDijkstra(G, ℓ, s):
  s.dist ← 0
  for all vertices v ≠ s
      v.dist ← ∞
  curr ← new empty bag
  add s to curr
  for k ← 0 to V
      next ← new empty bag
      while curr is not empty
          take v from curr           ⟨⟨v.dist = k⟩⟩
          for all edges v→w
              if w.dist > v.dist + ℓ(v→w)
                  w.dist ← v.dist + ℓ(v→w)
                  if ℓ(v→w) = 0
                      add w to curr
                  else ⟨⟨if ℓ(v→w) = 1⟩⟩
                      add w to next
      curr ← next
```

This phase of the algorithm runs in $O(V' + E') = O(V + E)$ time.

---

Once we have computed distances in $G'$, we construct a second graph $G'' = (V', E'')$ with the same vertices as $G'$, but only a subset of the edges:

$$E'' = \left\{u' \to v' \in E' \;\middle|\; u'.dist + \ell(u' \to v') = v'.dist\right\}$$

Equivalently, an edge $u' \to v'$ belongs to $E''$ if and only if that edge is part of at least one shortest path in $G'$ from $s$ to another vertex. It follows (by induction, of course), that every path in $G''$ from $s$ to another vertex $v'$ is a shortest path in $G'$, and therefore a minimum-switch path in $G$.

We also reassign the edge weights in $G''$. Specifically, we assign each uphill edge $u^\uparrow \to v^\uparrow$ and downhill edge $u^\downarrow \to v^\downarrow$ in $G''$ weight $\ell(uv)$, and we assign every switch edge, start edge, and end edge weight 0. **Now we need to compute the shortest path from $s$ to $t$ in $G''$, with respect to these new edge weights.**

We can expand the definition of $E''$ in terms of the original input graph as follows:

$$\begin{aligned} E'' = &\left\{u^{\uparrow}\!\to v^{\uparrow} \;\middle|\; uv \in E \text{ and } h(u) < h(v) \text{ and } u^{\uparrow}.dist = v^{\uparrow}.dist\right\} \\ &\cup \left\{u^{\downarrow}\!\to v^{\downarrow} \;\middle|\; uv \in E \text{ and } h(u) > h(v) \text{ and } u^{\downarrow}.dist = v^{\downarrow}.dist\right\} \\ &\cup \left\{v^{\uparrow}\!\to v^{\downarrow} \;\middle|\; v \in V \text{ and } v^{\uparrow}.dist < v^{\downarrow}.dist\right\} \\ &\cup \left\{v^{\downarrow}\!\to v^{\uparrow} \;\middle|\; v \in V \text{ and } v^{\downarrow}.dist < v^{\uparrow}.dist\right\} \end{aligned}$$

We can topologically sort $G''$ by first sorting the vertices by increasing $v'.dist$, and then within each subset of vertices with equal $v'.dist$, listing the up-vertices by increasing height, followed by the down vertices by decreasing height. It follows that $G''$ is a dag! Thus, we can compute shortest paths in $G''$ in $O(V'' + E'') = O(V + E)$ time, using the same dynamic programming algorithm that we used in part (a).

The overall algorithm runs in $\boldsymbol{O(V + E)}$ ***time***. ■

**Rubric:** max 10 points =

- 5 for computing minimum-switch paths = 1 for vertices + 1 for edges (including weights) + 2 for 0/1 shortest path algorithm + 1 for running time.
- 5 for computing shortest minimum-switch paths = 1 for vertices + 1 for edges (including weights) + 1 for proving dag + 1 for dynamic programming algorithm + 1 for running time