CMSC 27200 - Spring 2024 — Lorenzo Orecchia
due 4/26/24 via Gradescope

H6.

- The assignment is due at Gradescope on 4/26/24.
- A LaTeX template will be provided for each homework. You are strongly encouraged to type your homework into this template using LaTeX. If you are writing by hand, please fill in the solutions in this template, inserting additional sheets as necessary. This will help facilitate the grading.
- You are permitted to discuss the problems with up to 2 other students in the class (per problem); however, *you must write up your own solutions, in your own words*. Do not submit anything you cannot explain. If you do collaborate with any of the other students on any problem, please list all your collaborators in the appropriate spaces.
- Similarly, please list any other source you have used for each problem, including other textbooks or websites.
- *Show your work.* Answers without justification will be given little credit.
- Your homework is *resubmittable*. Please refer to the course syllabus on Canvas for a more detailed description of this. For any problem that you have not changed from your last submission, please make sure to indicate this in your submission to help our graders grade faster.

Problem 1 (Scenic Route) *Lorenzo wants to drive from San Francisco to Chicago, taking the shortest walk he can. However, he also really wants to drive along the route that passes through Yellowstone National Park. Find the shortest walk that passes through the park.*
More formally, you are given a connected, directed, weighted graph $G = (V, E)$, where each $e \in E$ represents a different route, $w(e)$ is the length of the route e, and each $v \in V$ represents a points connecting some number of different routes. Furthermore, $s \in V$ represents San Francisco, and $t \in V$ represents Chicago, and there is some edge $(y_1, y_2) \in E$ representing the edge through Yellowstone that Lorenzo must take. Note that the edge (y_2, y_1) may exist, but is not the route that Lorenzo wants to take. Note also that it is possible that taking this edge forces Lorenzo to make a cycle - this is okay.

(a) *Provide pseudocode for an algorithm that finds the desired walk by running Dijkstra's a constant number of times. Sketch a proof of correctness. You do not need to analyze runtime, but it should not take longer than $O((|V| + |E|) \log |V|)$ (assuming a Binary heap implementation of Dijkstra's).*

(b) *Describe (in words) a way to modify G to make $G' = (V', E')$, such that running Dijkstra's algorithm (unmodified) a single time on G' lets you find the correct walk with a linear amount of processing. Your modifications should not affect the algorithm's total asymptotic runtime. Sketch a proof of correctness, and sketch a proof that the modifications do not affect the total runtime.*

Collaborators:

Solution:

(a) Pseudocode:

Algorithm 1 Dijkstra's With A Stop

```
   Input: $G, w_e, (y_1, y_2) \in E, s \in V, t \in V$
   Output: $P_{s \rightsquigarrow t} \in G$
1: $P_{s \rightsquigarrow y_1} \leftarrow Dijkstra(G, w_e, s, y_1)$
2: $P_{y_2 \rightsquigarrow t} \leftarrow Dijkstra(G, w_e, y_2, t)$
3: return $P_{s \rightsquigarrow y_1} \cup (y_1, y_2) \cup P_{y_2 \rightarrow t}$
```

The correctness of this algorithm follows from the correctness of Dijkstra's algorithm.

If $P_{s \rightsquigarrow t}$ is the shortest such path from s to t, all subpaths $p_{i \rightsquigarrow j} \in P_{s \rightsquigarrow t}$ between intermediate vertices i and j must be the shortest such path. Suppose there is a $p'_{i \rightsquigarrow j}$ from i to j that is shorter than $p_{i \rightsquigarrow j}$. Then we could we could get a shorter $P_{s \rightsquigarrow t}$ by splicing out $p_{i \rightsquigarrow j}$ for $p'_{i \rightsquigarrow j}$.

Now, $P_{s \rightsquigarrow t} = \{s, \ldots, y_1, y_2, \ldots t\}$ because Lorenzo wants to visit Yosemite. Running Dijkstra twice from s and y_2 gives the shortest subpaths $\{s, \ldots, y_1\}$ and $\{y_2, \ldots t\}$. The last subpath y_1, y_2 is trivial because $(y_1, y_2) \in E$. Thus, union of these subpaths, as computed by the algorithm, is a shortest path obeying the constraint.

(b) Let G_r be the reverse graph of G. Remove y_1 from G. Add an edge (y_2, y_{1r}) between G and G_r, forming a new, composite graph G'. Here, y_{1r} denotes a relabeling of y_1 in G_r.

Then, running Dijkstra on G' starting at y_2 gives the paths $P_{y_2 \rightsquigarrow s_r} = \{y_2, \ldots, s_r\}$ and $P_{y_2 \rightsquigarrow t}\{y_2, \ldots, t\}$.

The desired path $P_{s \rightsquigarrow t} = P_{s_r \rightsquigarrow y_2} \cup P_{y_2 \rightsquigarrow t}$, where $P_{s_r \rightsquigarrow y_2}$ is the reverse of $P_{y_2 \rightsquigarrow s_r}$.

Correctness

$P_{y_2 \rightsquigarrow s}$ is the correct shortest path out of y_2 in the reversed graph because we used Dijkstra. Thus, its reverse corresponds to the correct shortest path to y_2 in the original graph. It obeys the constraint because $P_{y_2 \rightsquigarrow s} = \{y_2, y_{1r}, \ldots, s_r\}$ due to the construction of G' with y_{1r} as a bridge.

$P_{y_2 \rightsquigarrow t}$ is correct because of Dijkstra.

Thus, the union is the correct path.

Runtime

Reversing the graph takes linear time.

Creating the G' takes linear time.

Running Dijkstra on G' takes twice the amount of computation as running Dijkstra of G, since G' has twice the number of edges and vertices.

Thus, asymptotic runtime is still the same.

Problem 2 (Homecoming) *Billy gets lost easily, so every time he goes anywhere in town, he has to stop by home first to check in. With that in mind, he wants a new understanding of how "far" locations in town are from each other.*
The town is modeled by a connected, weighted, directed graph $G = (V, E)$, and you are given some home vertex $h \in V$. Billy wants the matrix M of size $n \times n$, where

$$M[v, u] = \text{length of shortest walk from } v \text{ to } u \text{ that contains } h.$$

Even the entry $M[v, v]$ must correspond to a walk containing h.
Provide pseudocode, a formal proof of correctness, and a formal proof of runtime for an algorithm that outputs this matrix. For full credit your algorithm should run in time $O(|V|^2)$. You may assume that $|E| = O(|V|)$.

Collaborators:

Solution:

Algorithm 2 Billy's Algorithm

```
    Input: $G, h$
    Output: $M[v, u]$
1:  $O \leftarrow Dijkstra(G, h)$
2:  $I \leftarrow Dijkstra(reverse(G), h)$
3:  $M \leftarrow [[]]$
4:  for $v$ in $V$ do
5:      for $u$ in $V$ do
6:          $M[v, u] = I(v) + O(u)$
7:  return $M$
```

Correctness
The proof follows from the correctness of Dijkstra's algorithm.
Note that the desired lengths $|v \to h \to u| = |v \to h| + |h \to u|$, where all paths are the shortest such paths.
Running Dijkstra's on G with h as the source vertex gives the lengths of the all shortest $|h \to u|$.
Running Dijkstra's on the reverse of G with h as the source vertex gives the lengths of all shortest $|v \to h|$.
Then, we can just construct M by adding the appropriate lengths as specified and inserting them into the array. $\square$
Runtime
Dijkstra's ($O(|V| \log |V|)$) is run twice.
The graph is reversed ($O(|E|)$) once.
There are $|V|^2$ entries in M to fill.
The total runtime is thus $2 \cdot O(|V| \log |V|) + O(|E|) + O(|V|^2) = O(|V|^2)$. $\square$

Problem 3 (Colorful MST) *You are given access to a program P to find the MST of a weighted undirected graph with any edge weights. However you are now given as input a weighted undirected graph and some edges are colored red while the others are blue. You further know that all the edge weights in the input given to you are integers. Design an algorithm that takes only $O(|V| + |E|)$ additional time,* ***apart from the time taken in a single call to*** *P, to output a MST of your input graph that uses as few red edges as possible. Note that* ***you still want to output a minimum cost spanning tree****; so the goal is to minimize the total weight, but of the many possible spanning trees that minimize the weight, you want one that uses as few red edges as possible. Prove the correctness of your algorithm.*

Collaborators:

Solution:

Algorithm 3 Colorful MST

```
   Input: $G, w, c$
   Output: $\min_{w_e, c_e = R} [T \in G]$
1: for $e \in E$ do
2:     if $c(e) = R$ do
3:         $w(e) = w(e) + 0.1$
4: return $P(G, w)$
```

First, we prove a stronger version of the cut property.
Lemma
Let $(S, \bar{S})$ be a cut in G and e be the minimum weight edge across this cut. We claim that e is in *all* MSTs of G.
Suppose an MST, T, of G does not contain e. Then across the cut $(S, \bar{S})$ is another edge e'. However, given $w(e') > w(e)$, a spanning tree of lesser cost can be created by adding e to T to produce a cycle, and then removing e' to create a new MST with less cost. □
Proof
It follows from the lemma that the multiplicity of MSTs arise from the existence of multiple minimal edges across some cuts. Thus, we can minimize the number of red edges by avoiding red edges when we have a choice of minimal edges across a cut.
Since the given edge weights are integer, we can accomplish this simply by punishing red edges with some penalty < 1 to preserve the ordering of edge weights. Then, any MST algorithm will find the MST minimizing red edges. □

Problem 4 (Huffman Coding) *A geneticist wants to encode a genetic sequence, and plans to use Huffman Coding to do so.*

(a) Say the nucleotides A, C, G, T appear in the sequence with frequencies 31%, 20%, 9%, 40% respectively. What is the Huffman encoding of these four characters?

(b) The geneticist is an alien geneticist, and these extraterrestrial species have n different nucleotides in their genes. If the geneticist encoded their sequences, what is the longest possible code word? Provide a set of frequencies that achieves this legnth.

(c) Find the largest value p such that, if all nucleotide frequencies are less than p%, codewords are guaranteed to not have length 1. Sketch a proof.

Collaborators:

Solution:

(a) A: 10
C: 110
G: 111
T: 0

(b) If, for each iteration of creating the Huffman coding tree, the sum of the two smallest frequencies is less than or equal to next smallest frequency, the tree will be completely skewed to one side. This is the worst case. The tree will then have depth n so the longest word will be $n-1$.

We can achieve this with a exponentially decaying sequence:

$\{2^{-1}, 2^{-2}, \ldots, 2^{-(n-1)}, 2^{-(n-1)}\}$

Note that the last one does not adhere to the pattern to ensure unit measure. The principle still holds, however.

(c) Assume $n > 3$ otherwise codewords of length 1 are unavoidable.

If a nucleotide x has a codeword of length 1, then its sibling under the root node is a subtree. Let the children of that subtree be A and B. Necessarily, $f(x) \geq f(A), f(B)$. Now, $f(x) + f(A) + f(B) = 1$ so $f(x) > \frac{1}{3}$, otherwise either $f(A) > f(x)$ or $f(B) > f(x)$.

Thus by contrapositive, if the nucleotide frequencies are no more than $\frac{1}{3}$ then there are no length 1 codewords.