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

H6.Mark Xiong

- 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: the solution is as follows:

Algorithm 1 Scenic Route

(a) Input: G, s, t, y_1, y_2
Output: The shortest path from San Francisco to Chicago that passes through Yellowstone National Park

```
function Dijkstra(G, source) (helper function)
    distance = array of ∞
    distance[source] = 0
    Q = priority queue of vertices
    Q.add(source)
    while Q is not empty do
        u = Q.pop() min distance in Q
        for each neighbor v of u do
            if distance[u] + weight(u, v) < distance[v] then
                distance[v] = distance[u] + weight(u, v)
                Q.update(v, distance[v])
            end if
        end for
    end while
end function

function RouteST(G, s, t, y1, y2)
    D1 = Dijkstra(G, s)
    D2 = Dijkstra(G, y2)
    ylpath = w(y1, y2)
    return D1[y1] + ylpath + D2[t]
end function
```

Proof of correctness: Dijkstra's algorithm guarantees that the shortest path from a single source to any other vertex in a graph with non-negative weights is found. By running Dijkstra's algorithm twice, we can find the shortest path from San Francisco to y_1 and from y_2 to Chicago. The algorithm calculates the shortest path that passes through the yellow stone edge by combining the two shortest paths and the weight of the yellow stone edge. The two calls to Dijkstra's algorithm guarantee that the shortest path is found.

(b) To modify G to G', we can add a new vertex y to the graph and connect it to y_1 and y_2 with an edge of weight 0. The new graph G' is (V', E') where $V' = V \cup \{y\}$ and $E' = E \cup \{(y_1, y), (y, y_2)\}$. By adding the new vertex y and connecting it to y_1 and y_2, we can ensure that the shortest path from San Francisco to Chicago that passes through Yellowstone National Park can be found by running Dijkstra's algorithm once on G'.
The correctness of the algorithm is guaranteed by the fact that Dijkstra's algorithm finds the shortest path from a single source to any other vertex in a graph with non-negative weights. The zero-weight edges connecting y_1 and y_2 to y ensure the path is always chosen, and it does not increase the total weight of the path. As other weights remain unchanged, the algorithm still finds the shortest path.
The modifications do not affect the total runtime of the algorithm because the runtime of Dijkstra's algorithm is $O((|V| + |E|) \log |V|)$ and the addition of a single vertex and two edges will only increase minimal RT but does not affect the runtime in Big-O notation.

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: The solution is as follows:

Algorithm 2 Homecoming

Input: G, h
Output: The matrix M of size $n \times n$ where $M[v, u]$ is the length of the shortest walk from v to u that contains h

```
function Dijkstra(G, source) (helper function)
    distance = array of ∞
    distance[source] = 0
    Q = priority queue of vertices
    Q.add(source)
    while Q is not empty do
        u = Q.pop() min distance in Q
        for each neighbor v of u do
            if distance[u] + weight(u, v) < distance[v] then
                distance[v] = distance[u] + weight(u, v)
                Q.update(v, distance[v])
            end if
        end for
    end while
end function

function Homecoming(G, h)
    G' = reverse all edges in G
    DFromH = Dijkstra(G, h)
    DToH = Dijkstra(G', h)
    M = matrix of size |V| × |V|
    for each vertex v in V do
        for each vertex u in V do
            M[v, u] = DToH[v] + DFromH[u]
        end for
    end for
    return M
end function
```

Proof of correctness: The algorithm is based on on the principle that any shortest path from vertex v to vertex u that must pass through a third vertex h can be decomposed into two distinct shortest paths: v to h and h to u. This decomposition is valid due to the nature of shortest paths in graphs with non-negative weights. Running Dijkstra's algorithm from h on the graph G' with all edges on G reversed, we can find the shortest path to h from any vertex in G. Similarly, running Dijkstra's algorithm from h on the original graph G, we can find the shortest path from h to any vertex in G. By summing the two shortest paths, we can find the shortest path from any vertex v to any vertex u that passes through h. The final matrix $M[v, u]$ is constructed with loops that each iteration finds the shortest path from v to u that passes through h. The correctness of this matrix directly comes from the correctness of Dijkstra's algorithm.

Proof of runtime: The algorithm runs Dijkstra's algorithm twice, once on the original graph G and once on the reversed graph G'. The runtime of Dijkstra's algorithm using a binary heap is $O((|V| + |E|) \log |V|)$. Since $|E| = O(|V|)$, the runtime of Dijkstra's algorithm is $O(|V| \log |V|)$. The loops that construct the matrix M have a runtime of $O(|V|^2)$. The total runtime of the algorithm is $O(|V|^2) + O(|V| \log |V|) + O(|V| \log |V|) = O(|V|^2)$.

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: The solution is as follows:

Algorithm 3 Colorful MST

Input: G, P
Output: A MST of the input graph that uses as few red edges as possible

```
1: function ColorfulMST(G, P)
2:     ε = 1 × 10^−10
3:     for each edge e in G do
4:         weight(e) += ε if e is red
5:     end for
6:     MST = P(G) return MST
7: end function
```

More details are included: Proof of correctness: The algorithm works by adding a small value ϵ to the weight of each red edge in the graph. By adding this small value, the algorithm ensures that the MST algorithm P will preferentially select blue edges over red edges in cases where weights of two edges of different colors tie together because the modifications do not change the relative weights of edges with different originla weights, and the algorithm will still select the minimum weight connections needed to span all vertices without creating cycles, as required by the definition of an MST. In cases of identical weights, the algorithm will biases towards blue edges, and therefore minimize the number of red edges in the MST. Since the MST algorithm P is guaranteed to find the minimum cost spanning tree, the algorithm will select the MST that uses as few red edges as possible as blue edges are relatively lighter with added ϵ on red edges. The MST computed still adheres to the minimum weight principle because ϵ is too small to affect the selection of edges unless there's a tie by original weight. Thus, the tree remains optimal in terms of weight while 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: The solution is as follows:

(a) The Huffman encoding of the four characters looks like the following:

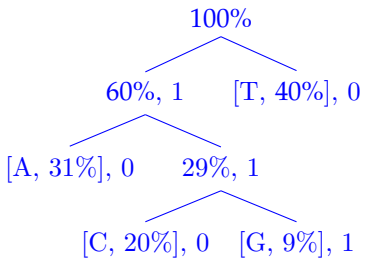

Figure 1: Huffman encoding of A, C, G, T

so we have T: 0, A: 10, C: 110, G: 111

(b) The longest possible code word is $n-1$. One senario that achieves this is as follows:

- The most frequent nucleotide has frequency of 50%
- The second most frequent nucleotide has frequency of 25%
- The third most frequent nucleotide has frequency of 12.5%
- Continue halving the frequency of the nucleotide until the last nucleotide has frequency of $1/2^n$%

(c) The largest value p is 33.333% or $\frac{1}{3}$. We could prove by contradiction. Assume that we have a full binary tree that does not have a codeword of length 1. This implies the root of the tree will have no direct frequencies as its children. As the largest frequency is no more than $\frac{1}{3}$, the other child would have a frequency greater than $\frac{2}{3}$, and its children would have a frequency greater or equal to $\frac{1}{3}$, which violates the contradiction.