Solution: Practice Quiz 2

- You have 120 minutes to complete the exam. The exam consists of 100 points and is designed to be completed in 100 minutes. You have a further 20 minutes to type and clean your solutions. Do not spend too much time on any one problem. The points are a rough guide of how many minutes you should spend on a problem.
- Do not spend time re-deriving facts that we have presented in lecture or recitation. Simply cite them.
- If you need assistance clarifying a question in the exam, post a *private* post in Piazza. If you post a Zoom link, we will meet you there.
- This is an open book exam: you can use CLRS, your notes from this class, or any material released by us this term. Internet usage is restricted to getting help for KaTeX/LaTeX, to access Piazza, to access Canvas, and to use Gradescope. Use of any material not released by us this term is *strictly* forbidden.
- Any form of collaboration is *strictly* forbidden. Any violations will result in a grade of 0 on the quiz and may result in a possible disciplinary action.
- If we have to make any announcement during the quiz, we will use Piazza. Thus, please monitor Piazza periodically.
- Unless you have special accommodations, you *must* make your final submission no later than two hours after the start of the exam window (regardless of when you started your exam).

Problem 1. [15 points] **Summer Travel** (2 parts)

Alyssa P. Hacker has decided to drive from Cambridge to Sacramento during the month of August. Assume that the US road network can be modeled as a set of n cities c_0 through c_{n-1} with m pairs of cities connected by one-way roads. Road (c_i, c_j) leads from city c_i to city c_j and the toll on the road costs d_{ij} dollars where d_{ij} is a positive integer smaller than 10. Alyssa also has a set of cities with average high temperature in August greater than $90F$; this set is given as C. You can assume that Cambridge and Sacramento are not in C.

(1) [5 points] Describe an $O(n + m)$ algorithm that finds the cheapest route from Boston to Sacramento that Alyssa can take without stopping at **any** city in C, or reports that none exists.

Solution: Remove all vertices corresponding to cities in C and incoming and outgoing edges. Run BFS on the resulting graph.

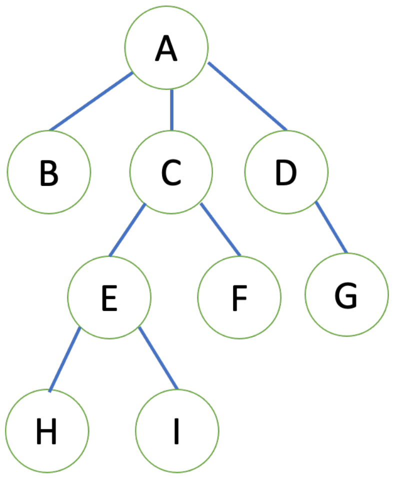
(2) [10 points] Describe an $O(k(n + m))$ algorithm that finds the cheapest route from Boston to Sacramento that Alyssa can take while stopping at **at most** k cities in C, or reports that none exists.

Solution: Make $k + 1$ duplicates where the index of the duplicate corresponds to the number of $> 90F$ cities encountered along the path. Edges from hot cities from the first copy go to the corresponding vertices in the second copy, and so on. The $k + 1^{th}$ copy has all the hot cities deleted. Find shortest path using BFS. Since there are $O(kn)$ nodes and $O(km)$ edges, the overall complexity is $O(k(n + m))$ for BFS.

Problem 2. [20 points] **How Many Edges?** (3 parts)

Assume for all parts that the G we refer to does not have any self-loop edges. Show your work so partial credit can be assigned.

(1) [6 points] You are given the DFS tree shown below. This DFS tree came from an undirected, connected graph G that you accidentally deleted! Provide *numeric* values for the **minimum** and **maximum** number of edges such a G could have had, including the edges in the DFS tree.



Solution: For an undirected graph, we can only have back edges and tree edges. Thus, when maximizing the number of edges, we can have an edge between each vertex and all of its ancestors. In that case, the maximal number of edges is the sum of the depths, which is $0 + 1 + 1 + 1 + 2 + 2 + 2 + 3 + 3 = 15$. On the other hand, the graph might just be the given tree and thus the minimum number of edges is simply 8.

(2) [10 points] Now, suppose the tree above corresponds to a BFS tree of a graph G. Provide *numeric* values for the minimum and maximum number of edges such a G could have had, including the edges in the BFS tree.

Solution: Note that for an undirected graph, all edges are within the same level in the BFS tree or between consecutive levels. The number of edges that can exist within a level is simply $\binom{1}{2}+\binom{3}{2}+\binom{3}{2}+\binom{2}{2} = 0+3+3+1 = 7$. The number of edges between consecutive levels is a bit trickier. Since the children are visited from left to right as drawn in the tree, there can be edges between a vertex and the children of all of its siblings that are to its left. In other words, we could have the edges (D, E), (D, F), (F, H), (F, I), (G, H), and (G, I) resulting in a maximum total of $8 + 7 + 6 = 21$. Edges such as (B, E) cannot exist as in that case E would have been a child of B.

Since the problem did not explicitly mention that vertices are visited from left to right, full credit was awarded to solutions that assumed that they could be visited in any order. In that case, the edges (B, E), (B, F), (B, G), and (C, G) could potentially exist. However, note that all of them cannot exist since in that case each of B, C, and D would have edges to all of E, F, and G and thus E, F, and G would have the same parent. However, we can have the edges (B, E), (B, F), and (B, G) if B was visited after C and D giving us a total of 24 edges. Moreover, there are modified versions of BFS that do not process the edges vertex by vertex in which case we can have 25 edges.

As before, the graph may simply be the given tree resulting in a minimum of 8 edges.

(3) [4 points] You realize that the tree you have is in fact both a BFS tree and a DFS tree for the **same** G. Provide *numeric* values for the minimum and maximum number of edges such a G could have had, including the edges in the tree.

Solution: Note that all back edges go up by at least 2 levels, which is not allowed for a BFS tree. Hence, there can only be tree edges and thus both the minimum and the maximum are 8.

Problem 3. [15 points] **Average Cycle** (2 parts)

(1) [10 points] Given weight $W > 0$ and a weighted directed graph $G(V, E, w)$ with n vertices and m edges that can have **positive or negative** weights, describe an $O(nm)$ algorithm for determining if there is some cycle with **average** weight strictly less than W. Note: If edges $e_1, \cdots, e_k$ are in a cycle, the average weight of the cycle is given by $\frac{\sum_{i=1}^{k} w(e_i)}{k}$.

Solution: Subtract W from all edge weights and look for a negative weight cycle. If one exists return YES, otherwise return NO. The existence of a negative weight cycle can be checked in $O(nm)$ time using Bellman-Ford from a supernode connected to all vertices. This is correct since subtracting the same amount from all weights decreases the average weight of any cycle by exactly that amount. Thus, any cycle of weight less than W in the original graph has negative weight after modifying the weights. Similarly, any negative weight cycle in the modified graph must have had average weight less than W in the original one.

(2) [5 points] Suppose the weights in G are **non-negative** integers in the range $[0, n^3]$. Describe an $O(nm \log(n))$ algorithm for determining the **minimum average weight** of any cycle in the graph. You can report an integer within ± 1 of the answer, and assume that you have a correct algorithm for Part 1.

Solution: Note that the optimal W must be in the range $[0, n^3]$. Thus, we binary search on the answer using the previous part to get which direction to recurse on. We stop when the range has width less than 1. This will require using the algorithm from the previous part at most $O(\log(n^3)) = O(\log(n))$ time and thus the total runtime would be $O(nm \log(n))$.

Problem 4. [20 points] **Orkscapade**

Ranger Raargorn needs to deliver a message from her home town of Tina's Mirth to the town of Riverdell, but the towns of Midgard have been overrun by an army of k Orks. Raargorn has a map of the n towns and $3n$ roads in Midgard, where each road connects a pair of towns in both directions. Scouts have determined the number of Orks $r_i \geq 1$ stationed in each town i (there is **at least one Ork** stationed in each town). Describe an $O(k)$-time algorithm to find a path from Tina's Mirth to Riverdell on which Raargorn will encounter the fewest total Orks in towns along the way. **Partial credit** will be awarded for slower correct algorithms, e.g., $O(k \log k)$ or $O(nk)$.

Solution: Construct a graph $G = (V, E)$ with:

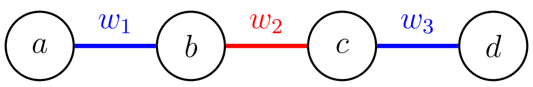
- a chain of r_i vertices $(v_1, \ldots, v_{r_v})$ connected by $r_v - 1$ edges for each town v, i.e., unweighted directed edge (v_i, v_{i+1}) for all $i \in \{1, \ldots, r_v - 1\}$; and
- two unweighted directed edges (u_{r_u}, v_1) and (v_{r_v}, u_1) for each road between towns u and v.

Graph G has $\sum_v r_v = k$ vertices and $2(3n) + \sum_v (r_v - 1) = 5n + k$ edges. Since there is at least one Ork in each town, $k \geq n$, so G has size $O(k)$. Let s and t correspond to the towns of Tina's Mirth and Riverdell respectively. Graph G has the property that any path from s_1 to t_{r_t} corresponds to a path from Tina's Mirth to Riverdell crossing edges equal to the number of Orks encounters in towns along the way, since for any road connecting towns u and v, going from u_1 to v_1 requires traversing r_v edges in G. So solve unweighted SSSP from s_1 to t_{r_t} using BFS in $O(k)$ time, and return the sequence of towns visited along the found shortest path by following parent pointers.

Problem 5. [25 points] **Color Cost**

A **3-color labeling** of a graph maps each edge to either red, green, or blue. The **color cost** of a 3-color labeled path is its path weight plus a positive integer w_c every time the path changes color.

For example, in the graph below (having four vertices $\{a, b, c, d\}$, a blue edge (a, b) with weight w_1, a red edge (b, c) with weight w_2, and another blue edge (c, d) with weight w_3) the path (a, b, c, d) has color cost $w_1 + w_2 + w_3 + 2w_c$, because the path changes color twice.



Given a 3-color labeling $c : E \to \{\text{red}, \text{green}, \text{blue}\}$ of a connected, weighted, undirected graph $G = (V, E, w)$ containing only **positive edge weights**, and given two vertices $s, t \in V$, describe an **efficient** algorithm to return a path from s to t having minimum color cost.
(By "efficient", we mean that faster correct algorithms will receive more points than slower ones.)

Solution: Construct a new graph $G' = (V', E')$ with:

- 3 vertices for each vertex $v \in V$: specifically v_i for $i \in \{\text{red}, \text{green}, \text{blue}\}$ corresponding to arriving at vertex v via an edge with color i;
- (vertex-edges) 3 undirected edges for each vertex $v \in V$: specifically $\{v_{\text{red}}, v_{\text{blue}}\}$, $\{v_{\text{green}}, v_{\text{red}}\}$, and $\{v_{\text{blue}}, v_{\text{green}}\}$ of weight w_c; and
- (edge-edges) 1 undirected edge for each undirected edge $\{u, v\} \in E$ of weight w and color $c(u, v)$: specifically undirected edge $\{u_{c(u,v)}, v_{c(u,v)}\}$ with weight w.

Graph G' has $3|V|$ vertices and $3|V| + |E|$ edges, and has the property that the minimum weight of any path in G' from any vertex s_i to any vertex t_j for $i, j \in \{\text{red}, \text{green}, \text{blue}\}$ is equal to the minimum color cost of any 3-color labeled path in G from s to t, as switching colors at a vertex requires traversing an edge of weight w_c. So solve SSSP three times, once from each vertex s_i and find the minimum weight of any path to any t_j, and then return a minimum path by constructing parent pointers as shown in lecture. Since this graph only has positive edge weights, we can solve SSSP using Dijkstra in $O(|V| + |E| + |V| \log |V|) = O(|E| + |V| \log |V|)$ time.

Note that you can avoid running Dijkstra three times via a supernode, but this only reduces work by a constant factor. Also, one can also avoid adding vertex-edges by adding three edges for each edge connected and weighted appropriately, but these edges will need to be **directed** toward a vertex labeled with the same color as the corresponding edge.