Solution: Quiz 2

- Do not open this quiz booklet until directed to do so. Read all the instructions on this page.
- When the quiz begins, write your name on the top of every page of this quiz booklet.
- You have 90 minutes to earn a maximum of 90 points. Do not spend too much time on any one problem. Skim them all first, and attack them in the order that allows you to make the most progress.
- **You are allowed two double-sided letter-sized sheet with your own notes**. No calculators, cell phones, or other programmable or communication devices are permitted.
- Write your solutions in the space provided. Pages will be scanned and separated for grading. If you need more space, write "Continued on S1" (or S2, S3) and continue your solution on the referenced scratch page at the end of the exam.
- Do not waste time and paper rederiving facts that we have studied in lecture, recitation, or problem sets. Simply cite them.
- When writing an algorithm, a **clear** description in English will suffice. Pseudo-code is not required. Be sure to argue that your **algorithm is correct**, and analyze the **asymptotic running time of your algorithm**. Even if your algorithm does not meet a requested bound, you **may** receive partial credit for inefficient solutions that are correct.
- **Pay close attention to the instructions for each problem**. Depending on the problem, partial credit may be awarded for incomplete answers.

Problem	Parts	Points
1: Information	2	2
2: Exact Edges	1	16
3: Color Cost	1	18
4: Orkscapade	1	18
5: Count Cycles	1	18
6: Bellham's Fjord	1	18
Total		90

Name: _____

School Email: _____

Problem 1. [2 points] **Information** (2 parts)

(a) [1 point] Write your name and email address on the cover page.
Solution: OK!

(b) [1 point] Write your name at the top of each page.
Solution: OK!

Problem 2. [16 points] **Exact Edges**

Given a weighted, directed graph $G = (V, E, w)$ with positive and negative edge weights, and given a particular vertex $v \in V$, describe an $O(k|E|)$-time algorithm to return the minimum weight of any cycle containing vertex v that also has exactly k edges, or return that no such cycle exists. Recall that a cycle may repeat vertices/edges.

Solution: Assume all vertices in G are reachable from v so that $|V| = O(|E|)$; otherwise, run BFS or DFS to solve single source reachability from v, and replace G with the subgraph reachable from v in $O(|E|)$ time. Construct a new graph $G' = (V', E')$ with:

- $k + 1$ vertices for each vertex $v \in V$: specifically v_i for all $i \in \{0, \ldots, k\}$; and
- k edges for each edge $(u, v) \in E$: specifically edges (u_{i-1}, v_i) for all $i \in \{1, \ldots, k\}$.

Graph G' has $(k + 1)|V| = O(k|E|)$ vertices and $k|E|$ edges in $k + 1$ layers, and has the property that paths from v_0 to v_k have a one-to-one correspondence with cycles through v in G of the same weight containing exactly k edges, since exactly one edge is traversed with each increase in layer. So solve SSSP from v_0 in G' to return the minimum weight path to v_k. Since edges in G' always increase in subscript, G' is a DAG, so we can solve SSSP using DAG relaxation in linear time with respect to the size of G'. So together with initial pruning, this algorithm takes $O(k|E|)$ time in total.

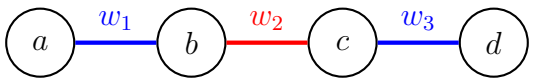
Common Mistakes:

- Using BFS to detect cycles in a directed graph (need DFS)
- Trying to enumerate all paths or cycles of length k (may be exponential)
- Using Bellman-Ford without removing zero-weight edges (may use fewer than k edges)
- Finding cycles using $k - 1$ edges, not k edges
- Trying to find negative-weight cycles (wrong problem)
- Not running a reachability algorithm to prune to graph reachable from v

Problem 3. [18 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.

Common Mistakes:

- Incorrectly trying to modify Dijkstra to keep track of state
- Failing to identify (or identifying incorrectly) a source from which to run SSSP
- Weighting or directing edges in a duplicated graph incorrectly

Problem 4. [18 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.

Common Mistakes:

- Not directing edges with vertex weight, or otherwise not clearly defining a graph
- Expanding weights on all edges, leading to an $O(nk)$ expansion
- (e.g., a town with $\Theta(k)$ Orks may connect to $\Theta(n)$ roads)
- Using Dijkstra without modification to achieve an $O(k \log k)$-time algorithm
- Expanding vertex weights incorrectly (path length r_i instead of $r_i - 1$)
- Finding shortest paths in a BFS or DFS tree (may not contain shortest paths)

Problem 5. [18 points] **Count Cycles**

A **cycle-sparse** graph is any weighted directed simple graph $G = (V, E, w)$ for which every vertex $v \in V$ is reachable from at most one simple[1] negative-weight cycle in G. Given a cycle-sparse graph, describe an $O(|V|^3)$-time algorithm to return the number of negative-weight cycles in G.

Solution: Construct a new graph G' by adding a supernode x to G with a zero-weight directed edge (x, v) for each $v \in V$. Then run SSSP from x in G' using Bellman-Ford to label each vertex $v \in V$ with its shortest path distance $\delta(x, v)$. For each $v \in V$, $\delta(x, v) = -\infty$ if and only if v is reachable from a negative-weight cycle in G (since adding x does not add or remove any cycles). Further, for any directed edge (u, v), if $\delta(x, u) = \delta(x, v) = -\infty$, then both u and v are each reachable from the same simple negative-weight cycle (since v is reachable from u and each vertex is reachable from at most one simple negative-weight cycle).

So, construct a new graph G'' on only the vertices $v \in V$ where $\delta(x, v) = -\infty$ in G', with an **undirected** edge between u and v in G'' if they share a directed edge in G. Graph G'' has the property that the number of connected components in G'' equals the number of negative-weight cycles in G, so count and return the number of connected components in G'' using Full-BFS or Full-DFS. This algorithm takes $O(|V|+|E|)$ time to construct G', $O(|V||E|)$ time to run Bellman-Ford, $O(|V|+|E|)$ time to construct G'', and then $O(|V|+|E|)$ time to count connected components in G'', leading to an $O(|V||E|) = O(|V|^3)$ running time in total.

Common Mistakes:

- General lack of precision when describing algorithm
- Trying to enumerate all paths or cycles (may be exponential)
- Repeatedly running Bellman-Ford, generally yielding $|V| \cdot O(|V||E|) = O(|V|^4)$ time
- Stating that $|E| = O(|V|)$
- Confusing connected components with strongly connected components

[1] Recall a cycle is simple if visits any vertex at most once.

Problem 6. [18 points] **Bellham's Fjord**

Gralexandra Bellham wants to drive her electric car in Norway from location s in Oslo to a scenic Fjord at location t. She has a map of the n locations in Norway and the $O(n)$ one-way roads directly connecting pairs of them.

- Each location x is marked with its (positive or negative) integer **height** $h(x)$ above sea-level.
- Her car has **regenerative braking** allowing it to generate energy while going downhill. Each road from location x to y is marked with the integer energy $J(x, y)$ that the electric car will either spend (positive) or generate (negative) while driving along it.
- By the laws of physics, $J(x, y)$ is always strictly greater than the difference in **potential energy** between locations x and y, i.e., $J(x, y) > m \cdot g \cdot (h(y) - h(x))$, where m and g are the mass of the car and the acceleration due to gravity respectively.

Her car battery has very large **energy capacity** $b > 2nk$ where k is the maximum $|J(x, y)|$ of any road. Assuming she departs s at half capacity, $\lfloor b/2 \rfloor$, describe an $O(n \log n)$-time algorithm to determine the maximum amount of energy Bellham can have in her battery upon reaching t.
Partial credit will be awarded for slower correct algorithms, e.g., $O(n^2)$.

Solution: Construct graph G with a vertex for each of the n locations in Norway and a directed edge for each of the $O(n)$ roads: specifically for each road from location u to v, add directed edge (u, v) weighted by $J(u, v)$. Then $\lfloor b/2 \rfloor$ minus the weight of a minimum-weight path from s to t in G would correspond to the maximum energy Bellham could have upon reaching t; or at least it would be if she did not either exceed or exhaust her tank along the way.

First we show that every minimum-weight path from s to t in G is simple. It suffices to show that every directed cycle in G has positive weight. Consider cycle $(c_0, \ldots, c_{k-1}, c_k = c_0)$. C has weight $\sum_{i=1}^{k} J(c_{i-1}, c_i) > \sum_{i=1}^{k} mg(h(c_i) - h(c_{i-1})) = 0$, as desired.

Any simple path in G traverses at most $n - 1$ edges, so the magnitude of its weight is at most $(n-1)k < b/2$. Thus $\lfloor b/2 \rfloor$ minus the weight of any simple path in G will always be > 0 and $< b$ (so Bellham cannot exhaust or exceed her tank by driving on a simple path from s to t).

Lastly, we find the weight of a minimum-weight path from s to t by solving SSSP. Unfortunately using Bellman-Ford takes $O(n^2)$ time which is too slow. However, we can re-weight edges in G to be positive while preserving shortest paths by exploiting the provided vertex potentials, similar to Johnson's algorithm. Specifically, create new graph G', identical to G, except change the weight of each edge (u, v) to $J(u, v) - mg(h(v) - h(u)) > 0$. This transformation preserves shortest paths since the weight of each path from, e.g., a to b changes by the same amount, namely by $mg(h(b) - h(a))$. So run Dijkstra from s to find the minimum weight D of any path to t in G', and return $\lfloor b/2 \rfloor - (D - mg(h(b) - h(a)))$.

Constructing G takes $O(n)$ time, reweighting to G' also takes $O(n)$ time, and then running Dijkstra from s in G' takes $O(n \log n)$ time, leading to $O(n \log n)$ time in total.

Common mistakes continued on S1.

SCRATCH PAPER 1. DO NOT REMOVE FROM THE EXAM.

You can use this paper to write a longer solution if you run out of space, but be sure to write "Continued on S1" on the problem statement's page.

Common Mistakes: (for Problem 6)

- Using Bellman-Ford directly yielding $O(n^2)$-time algorithm (eligible for half the points)
- Running Bellman-Ford to find new weights, still leading to $O(n^2)$-time
- Using $-J$ on weights instead of J
- b-times graph duplication (inefficient, b may be much larger than n)

SCRATCH PAPER 2. DO NOT REMOVE FROM THE EXAM.

You can use this paper to write a longer solution if you run out of space, but be sure to write "Continued on S2" on the problem statement's page.

SCRATCH PAPER 3. DO NOT REMOVE FROM THE EXAM.

You can use this paper to write a longer solution if you run out of space, but be sure to write "Continued on S3" on the problem statement's page.