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: Thomas Ren.

School Email: thomasren681@gmail.com.

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

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

✓

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

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.

If there is a cycle containing vertex v, then this cycle must be in a connected component of v. So, if we apply any SSSP algorithms on our graph, we only need to search a subgraph that is reachable from v.

We can BFS from v to return a connected component that contains v. Then we can apply a modified version of Bellman-Ford to detect a cycle with exactly k edges. To be more specific, we will modify the graph duplication to make it run in $O(k|E|)$-time.

1. Run BFS from v to return a searching tree. We construct a subgraph $G'(V', E', w')$, the structure of our graph is still the same as the original graph, but all vertices are reachable from v.
$|V'| = O(|V|)$ $|E'| = O(|E|)$ $|V'| = O(|E'|) = O(|E|)$ (connected graph).

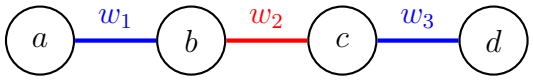
2. Graph duplication on G' k times. That is for every vertex v, we have v_i, $i = 0, 1, \dots, k-1$.
For every edge (u, v), we add (u_i, v_{i+1}), for $i = 0, \dots, k-2$.
We designate the duplicated graph as $G^*(V^*, E^*, w^*)$
$|V^*| = k|V'| = O(k|V|)$ $|E^*| = k|E'| = O(k|E|)$
$O(k|E|)$.
Note the difference between the graph duplication of Bellman-Ford is that we don't connect edge (v_i, v_{i+1}), so that we cannot stay at a node for one step.

3. DAG Relaxation from v_0 to v_{k-1}.
It costs linear time, so $O(k|E|)$.
Returns the shortest path of a cycle starting at 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.)

The subject tell us the following thing:

1. all edges have positive weights, which satisfies the dijkstra's condition.
2. edges are undirected, we can make each as reciprocal directed edges.
3. all edges have one of three state, namely {Red, Green, Blue}. we can construct a new graph which use three nodes to represent the three possible states
4. we only need to compute the shortest path from s to t, so solves a SSSP problem.

Graph Construction:

1. Initialize a Directed graph $G'(V', E')$.
2. for v in V, we append v_R, v_G, v_B to V', which leads to $|V'| = 3|V| = O(|V|)$.
3. for $e=(u,v,w)$ in E, w.log.o.g it's a red edge, we add a directed edge (u_R, v_R, w) and (v_R, u_R, w). $|E'| = 2|E| = O(|E|) = O(|V|)$ (connected graph).
4. for (v_R, v_G, v_B) in V', add directed edges: $\{(v_R, v_G, w_c), (v_G, v_R, w_c), (v_R, v_B, w_c), (v_B, v_R, w_c), (v_G, v_B, w_c), (v_B, v_G, w_c)\}$, so that if the color of edges got changed, we pay extra w_c.
   $|E'| = 2|E| + 6|V| = O(|E|) + O(|V|) = O(|V|)$.

Then the graph is well enough to implement Dijkstra.
Running time should be $O(|V'| \log |V'| + |E'|) = O(|V| \log |V|)$.

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 $O(nk)$.

Note that every town has at least one Ork, so # Orks $\geq$ # towns, i.e. $n = O(k)$.

we construct a graph as following:

1. Initialize an unweighted directed graph $G = (V, E)$.
2. for every town i we added r_i vertices, named $\tilde{v}_0, \tilde{v}_1, \cdots, \tilde{v}_{r_i-1}$
   $|V| = k$.
3. added unweighted, undirected edges $(\tilde{v}_0, \tilde{v}_1), \cdots, (\tilde{v}_{r_i-2}, \tilde{v}_{r_i-1})$ for every town i.
4. for every road (u, v), we add directed edges (u_{r_u-1}, v_1) and (v_{r_v-1}, u_1)
   Note we don't go directly to zero-index, cause we need to counterbalance the edge for road

$|E| = 3n - 2 + k$

$= O(n) + O(k) = O(k)$.

Then we can simply run BFS from Tina's Mirth to Riverdell. in $O(k)$-time, find the shortest path by reversely traverse the parent pointers.

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.

Observations:

1. every $v \in V$ is reachable from at most one simple negative-weight cycle.
   Count the number of strongly connected component that has a negative-weight cycle.
2. Negative-weight cycle can be witnessed only by Bellman-Ford for a general graph in $O(|V||E|)$-time.
Note this graph can be dense, so $|E|$ can only be bounded by $O(|V|^2)$.
So Bellman-Ford runs in $O(|V|^3)$-time.
3. Bellman-Ford only tells us which node is reachable from a negative-weight cycle, but doesn't tell the number of negative-weight cycles.
Use Bellman-Ford to process the graph. To be more specific, we delete the nodes and edges that are not reachable from negative-weight cycles.
Then modify the graph to run Full-DFS to count the number of connected components.

Implementations:

1. Add a supernode s with directed edges (s, v) for $v \in V$ with weight zero.
2. Apply Bellman-Ford to solve an SSSP problem from s.
3. If a vertex is reachable from a negative-weight cycle, $\delta(s, v) = -\infty$ or in our implementation, $\delta(s, v) =$ None.
4. Construct a new graph $G'(V', E')$, where $\delta(s, v') \neq$ None for all $v' \in V'$, i.e. we delete all vertices that are not reachable from a negative-weight cycle.
   This will also eliminate the difference of connected component and strongly connected component when we applying Full-DFS to our graph. G''
5. Construct an undirected graph that has the same V and E', remember to eliminate reciprocal edges to keep G'' simple.
6. Run Full-DFS on G'' to return the number of connected component.

[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. no negative-weight cycle.
- 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., $O(n^2)$.

Lemma 1: Bellham won't exhaust her car battery or exceed the capacity.

Lemma 2: There exists no negative-weight cycle.

Lemma 3: Our approach of reweighting the weight won't change the shortest path.

We will prove every lemma later. Now, we list observations that help us solve the subject.

Observations:

1. The map is a directed weighted graph.
2. We try to solve a single source shortest path problem (SSSP).
3. The $O(n \log n)$-time only allows Dijkstra algorithm.
4. To apply Dijkstra, we need to make the general graph non-negative.

Graph Construction:

Initialize a directed graph $G(V, E, w)$. For every location on the map, we add a corresponding node, for every road (u, v), we add a directed edge (u, v) with weight $J(x,y) - mg(h(y) - h(x))$.

Algorithm:

1. Run Dijkstra on our graph and return the shortest path's weight.
2. Add $(h(t) - h(s)) \cdot mg$ and return the maximum amount of energy Bellman can have.
3. The whole algorithm finishes in $O(|V| \log |V| + |E|)$-time
$|V| = n$ $|E| = O(n) \Rightarrow$ finishes in $O(n \log n)$-time.

(Proof continued on 51).

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.

Lemma 1 Proof:

For every road (u,v), the energy cost of Bellman's battery is $J(u,v) \le k$.

By Lemma 2 (Proof later), we know that no negative-weight cycle exists.

So all shortest path. from s to v is at most $|V|-1 = n-1$.

∴ the whole energy cost $= \sum_{u,v \in \pi} J(u,v) \le (n-1) \cdot J_{max}(u,v) = (n-1)k < nk$.

The initial energy stored in Bellman. is $\lfloor B/2 \rfloor \ge 2nk/2 = nk$.

∴ The initial energy Bellman has is sufficient for any simple shortest path's cost. □.

Lemma 2 Proof:

We prove by contradiction.

Assume that there exists a negative weight cycle $C = (c_0, \dots, c_{m-1})$

the weight for edge(u,v) is $J(u,v)$.

$W(C) = \sum J(u,v) > \sum mg(h(u)-h(v)) = mg[h(c_0)-h(c_1)+h(c_1)-\dots+h(c_{m-1})-h(c_{m-1})+h(c_{m-1})-h(c_0)] = 0$.

$\Rightarrow W(C) > 0$

So every cycle, if ever exists, has positive weight. □.

Lemma 3 Proof:

Assume, before reweighting, a simple path from s to t is $\pi = (s, v_0, \dots, v_{m-1}, t)$.

$W_\pi = J(s,v_0) + \dots + J(v_{m-1}, t)$.

After reweighting.

$W'_\pi = J(s,v_0) + mg(h(s)-h(v_0)) + \dots + J(v_{m-1},t) + mg(h(v_{m-1}) - h(t))$.

$= W_\pi + mg(h(s) - h(t))$.

So every simple path from s to t are changed by a fixed bias $mg(h(s)-h(t))$.

Thus the shortest path remain the shortest path. □.

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.

6.006 Introduction to Algorithms

Spring 2020