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, S4) 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
0: Information	2	2
1: Through Cycle	1	10
2: Consistent Colors	1	10
3: Useless Vertices	1	15
4: Restaurant Relocation	1	15
5: Disjoint Dimensions	1	18
6: Military Movements	1	20
Total		90

Name: _____

School Email: _____

Problem 0. [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 1. [10 points] **Through Cycle**

Given a directed graph $G = (V, E)$ and a vertex $v \in V$, describe an $O(|E|)$-time algorithm to determine whether G contains a cycle through v.

Solution: There is a cycle in G through v if and only if some incoming neighbor of v is reachable from v. Run either depth-first search or breadth-first search from v to mark all vertices reachable from v, which can be done in $O(|E|)$ time since the number of vertices reachable from v cannot be more than $|E|$. Then, return whether any of the $O(|E|)$ incoming neighbors of v is reachable.

Common Mistakes:

- Running DFS or BFS without starting from vertex v
- Checking for any back-edge, instead of a back-edge to v
- Assuming $|V| = O(|E|)$ instead of search from v only explores $O(|E|)$ vertices

Problem 2. [10 points] **Consistent Colors**

A **consistent coloring** of an undirected graph is an assignment of vertices to colors such that two vertices have the same color if they share an edge. Describe a linear-time algorithm to determine the maximum number of distinct colors in any consistent coloring of a given undirected graph.

Solution: Vertices in the same connected component of G have the same color, as any two vertices in the same connected component are connected to each other by a sequence of edges, while two vertices from different connected components may be assigned different colors. Then the maximum number of distinct colors in any consistent coloring is equal to the number of connected components in G. We can count the connected components of G in linear-time using either full breath-first-search or full depth-first search.

Common Mistakes:

- Not specifying a full DFS/BFS (as opposed to from a single source)
- Not connecting differently colored components with maximization
- Not realizing linear-time means $O(|V| + |E|)$ for graphs, trying for $O(|V|)$ or $O(|E|)$ instead

Problem 3. [15 points] **Useless Vertices**

A directed weighted graph is **special** if: for every ordered pair of vertices there exists a **unique** minimum weight path between them (having finite total weight). A vertex in a special graph is **useless** if it is not on the unique minimum weight path between any other pair of vertices. Given a special directed weighted graph $G = (V, E)$, possibly containing positive and negative weights, describe an $O(|V|^3)$-time algorithm to determine for every vertex $v \in V$ whether v is useless.

Solution: If we knew the shortest distance between every pair of vertices, then for each vertex v, we could check whether the shortest distance between other vertices would change if v were removed, certifying that v is useless; specifically, check whether $d(a, b) = d(a, v) + d(v, b)$ for every ordered pair of vertices a, b. This check is sufficient because the special property ensures the shortest path from a to b is unique, so if the equality holds, removing v would increase the shortest path distance from a to b. This procedure checks every vertex against all other pairs of vertices, so if all pairs shortest path distances are already known, this procedure runs in $O(|V|^3)$ time. To compute all pairs shortest path distances in G, running Bellman-Ford $|V|$ times will be too slow (it would be $O(|V|^4)$), so run Johnson's algorithm in $O(|V||E| + |V|^2 \log |V|) = O(|V|^3)$ time. Note that Floyd-Warshall could also be used for an $O(|V|^3)$ running time, but this algorithm has not yet been covered.

Common Mistakes:

- Finding APSP without asserting a lack of negative cycles
- Using a graph algorithm that does not apply to this graph
- Analyzing running time of APSP, but not of post-processing
- Checking vertices for inclusion in paths, but naively in $O(|V|^4)$
- Efficiently maintain hash table of vertices on paths, but incorrectly (paths include endpoints)

Problem 4. [15 points] **Restaurant Relocation**

Katé Srill, the owner of a food truck in Cambridge, wants to open a restaurant in San Francisco. She decides to drive there and sell food in cities along the way. She gets out her map of all US roads, and marks:

- each of the r roads directly connecting two cities on the map with a positive integer estimating the **cost of tolls** to travel between the two cities along that road, and
- each of the c cities on the map with a positive integer estimating the **profit** she thinks she can make by purchasing ingredients and selling food in that city for one day (she won't spend more than one day **consecutively** in any city).

Given her marked map, describe an efficient algorithm to determine whether Katé can start at Cambridge with some amount of money, and reach San Francisco with at least \$1000 **more** than when she started (for her restaurant down payment).

Solution: This problem is somewhat vague on whether Katé sells in Cambridge or San Francisco. This solution assumes selling in Cambridge before leaving and never selling in San Francisco (she must have \$1000 more whenever she first arrives). Other assumptions are possible, but solutions are similar, so we were lenient when grading.

Let $t(a, b)$ represent the cost of tolls to traverse the road between cities a and b, and let $p(a)$ be the profit obtained by selling food in city a. Construct graph G with vertices being the c cities, and for each road connecting directly two cities a and b, add directed edges (a, b) with weight $t(a, b) - p(a)$ and (b, a) with weight $t(a, b) - p(b)$, corresponding to selling food in a then going to b or selling food in b then going to a respectively (but no edges leaving San Fransisco). Then if the shortest path distance from Cambridge to San Francisco in G is smaller than -1000, then Katé can reach San Francisco with \$1000 more than when she started. G may have positive or negative weights, so run Bellman-Ford from Cambridge to find the shortest path distance to San Francisco in $O(rc)$ time.

Common Mistakes:

- Graph duplication on state of money does not have bounded size
- (Katé may have unbounded amount of money along the way)
- Using undirected edges (or not specifying whether edges are directed or undirected)
- Incorrectly assuming a graph had non-negative weights or no cycles

Problem 5. [18 points] **Disjoint Dimensions**

An **interval** (a, b, w) is a triple of integers with left endpoint a, right endpoint $b > a$, and weight w.

- Two intervals (a_1, b_1, w_1) and (a_2, b_2, w_2) are **disjoint** if $b_1 \leq a_2$ or $b_2 \leq a_1$, and a set of intervals are **pairwise disjoint** if each pair of intervals in the set is disjoint.
- The **dimension** of a set of intervals is the sum total of their weights.

Given a set of n intervals, describe an efficient algorithm to determine the maximum dimension of any subset of the intervals that is pairwise disjoint.

Solution:

$O(n \log n)$ time: Construct graph G with vertices being the distinct integers that appear as endpoints in the input set (let s be the smallest of these endpoints); and with

1. a directed edge (a, b) with weight zero when b is the smallest endpoint greater than a, and
2. a directed edge (a, b) with weight $-w$ for each interval (a, b, w).

G has at most $2n = O(n)$ vertices and at most $2n - 1 + n = O(n)$ edges. We can construct G efficiently by first finding the set of unique endpoints in $O(n)$ time using a hash table, and then sorting the endpoints in $O(n \log n)$ time, adding edges between adjacent pairs. This sorted order constitutes a topological sort of G by construction, so G is acyclic. Then any path starting from s corresponds to a subset of intervals that is pairwise disjoint (corresponding to the type 2 edges above), and any subset that is pairwise disjoint corresponds to a path in G starting from s (by connecting the edges associated with the subset). Thus, we can find the maximum dimension of any pairwise disjoint subset by running DAG relaxation from s in $O(n)$ time, returning the negative of the minimum distance of any vertex from s. This algorithm runs in $O(n \log n)$ time in total.

$O(n^2)$ time: Construct graph G with vertices being the n left endpoints a_i and another node s; and with directed edge (a_i, a_j) with weight $-w_j$ if $b_i \leq a_j$ for all pairs $a_i, a_j j$, and directed edge (s, a_i) with weight w_i for each a_i. We can construct this graph naively in $O(n^2)$ time: for each of the n intervals, add an edge from s in $O(1)$ time, and for each of the $O(n^2)$ pairs of weighted intervals, check whether to add an edge between them in $O(1)$ time. Then by construction, any path in G from s corresponds to a subset of intervals that are pairwise disjoint, and any subset of intervals that are pairwise disjoint corresponds to a path in G from s by sorting the intervals in the subset by left endpoint. G is acyclic because sorting the a_is (with s first) corresponds to a topological sort of G, so run DAG relaxation from s and return the negative weight of the shortest path found to any other node, also in $O(n^2)$ time.

Common Mistakes:

- Using an undirected graph or trying to solve via connected components
- Not realizing a correctly constructed graph is acyclic
- Weighting edges incorrectly or forgetting to negate edge weights for maximization

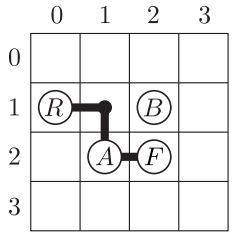
Problem 6. [20 points] **Military Movements**

Uncivilized is a turn-based game where armies move on the tiles of a game board consisting of an $n \times n$ grid of square **tiles**.

- Each **turn** an army may make at most $k = \Theta(\sqrt{n})$ sequential moves between edge-adjacent tiles on the board.
- Exactly n of the tiles contain **cities**; an army must end each turn on a tile containing a city so the army may rest (before the next turn can be taken). Each city will charge an army a positive integer **lodging fee** for resting there.

Given the name, location, and lodging fee of each city on the board, describe an $O(n^2)$-time algorithm to determine the minimum total fee that must be paid by an army that starts rested in **R**eemeen city, and moves to the city of **F**interwell to rest, using as many turns as needed.

For example, if $k = 2$ and the board contains $n = 4$ cities: R at $(0, 1)$ with fee 4, A at $(1, 2)$ with fee 2, B at $(2, 1)$ with fee 3, and F at $(2, 2)$ with fee 1, the minimum total fee for an army at R to move to F would be 3: by making two moves to rest at A on the first turn, and making one move to rest at F on the second turn (note that F is not reachable from R on the first turn).



Solution: Let $f(a)$ be lodging fee of tile a for any tile a that contains a city. First note that movement is limited each turn, and an army must start and end each turn in a tile containing a city. The overall strategy will be to find cities reachable from another city in a single turn from each city in an unweighted graph G, use this information to construct a weighted graph G' on only the cities, and then use Dijkstra to find a shortest path corresponding to minimizing total fees. Here are two methods for finding all reachable cities from each city in $O(k^2) = O(n)$ time.

Method (1): Construct graph G where vertices are the n^2 tiles of the board. For each directed pair of adjacent tiles a and b, add the undirected edge $\{a, b\}$. G has $O(n^2)$ edges, since each tile has at most four adjacencies, so G can be constructed naively in $O(n^2)$ time. Then run breadth-first search from each city to find cities reachable in at most k edges. The number of tiles reachable from any city in k edges is at most $O(k^2) = O(n)$, so searching from each city takes $O(n)$ time, and searching from all cities in this way takes $O(n^2)$ time in total.

Method (2): Given the locations of two cities (x_1, y_1) and (x_2, y_2), we can compute whether one is reachable from another in k moves in constant time by evaluating whether $k \leq |x_1 - x_2| + |y_1 - y_2|$. So evaluating this for each of the $O(n^2)$ pairs of cities takes $O(n^2)$ time.

Now construct graph G' where vertices are the n cities, with a directed edge from city a to city b with weight $f(a)$ if b is reachable from a in a single turn (found by the searches in G). G' has at most $O(n^2)$ edges and can be constructed in as much time. Then every path from Reemeen to Finterwell in G' corresponds to a sequence of cities where adjacent cities are reachable in a single turn, and the weight of the path corresponds to the fees paid along the way. Thus, we can run Dijkstra from Reemeen to determine the minimum fee to Finterwell in $O(n^2 + n \log n) = O(n^2)$ time.

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:

- Not stating an algorithm to finding cities reachable from other cities
- Incorrectly claiming BFS from a city is $O(n)$ without stopping search early
- (or stopping search at $\sqrt{n}$ instead of k)
- Incorrectly assuming $O(|E|) = O(n)$ in a constructed city-graph G'
- Incorrectly assuming city-graph G' is acyclic
- Removing positions not reachable from any city in k moves, then running SSSP

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.

SCRATCH PAPER 4. 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 S4" on the problem statement's page.