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 120 minutes to earn a maximum of 120 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 **briefly** 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.

Problem	Parts	Points
0: Information	2	2
1: Connect 4ward	2	8
2: Counting Blobs	1	10
3: HAM DAG	1	10
4: Long Paths	1	15
5: Unicycles	1	15
6: Doh!-nut	2	30
7: Selfish Coins	1	15
8: Alternating Sums	1	15
Total		120

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.

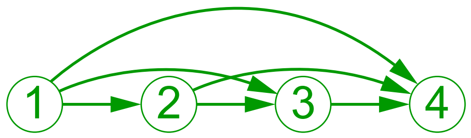
Solution: OK!

Problem 1. [8 points] **Connect 4ward** (2 parts)

The **super-cool** graph of order n is a graph on vertices $v_1, v_2, \ldots, v_n$ with a directed edge from vertex v_i to vertex v_j whenever $i < j$.

(a) [2 points] Draw the super-cool graph of order 4.

Solution:



Rubric:

- 2 points for a correct drawing

(b) [6 points] Given a super-cool graph of order n, run depth-first search from vertex v_1, always searching neighbors with lower index first (i.e., if vertex v_i and v_j are both neighbors with $i < j$, explore v_i before v_j). In the boxes below, write the number of tree, back, forward, and cross edges in the resultant DFS tree **<u>in terms of n</u>**.

Tree	Back	Forward	Cross
$n-1$	0	$\frac{(n-1)(n-2)}{2}$	0

Solution: The DFS tree is a connected chain of n adjacent vertices, so there are no cross edges. In addition, the graph has no cycles, so there are no back edges. There are always $n-1$ tree edges, and there are $n(n-1)/2$ edges total, so there are $n(n-1)/2 - (n-1) = (n-1)(n-2)/2 = (n^2 - 3n + 2)/2$ forward edges.

Rubric:

- 1 point for back edges
- 1 point for cross edges
- 2 points for tree edges
- 2 points for forward edges

Problem 2. [10 points] **Counting Blobs** (1 part)

An **image** is a 2D grid of black and white square pixels where each white pixel is contained in a **blob**. Two white pixels are in the same blob if they share an edge of the grid. Black pixels are not contained in blobs. Given an $n \times m$ array representing an image, describe an $O(nm)$ algorithm to count the number of blobs in the image.

Solution: Construct a graph with a vertex per white pixel, with an undirected edge between two vertices if the pixels associated with them are both white and share an edge of the grid. This graph has size at most $O(nm)$ vertices and at most $O(nm)$ edges (as pixels share edges with at most four other pixels), so can be constructed in $O(nm)$ time. Each connected component of this graph corresponds to a blob, so run breadth-first search or depth-first search to count the number of connected components in $O(nm)$ time.

Rubric:

- 8 points for a correct $O(nm)$ time algorithm
- 2 points for running time analysis
- Partial credit may be awarded

Problem 3. [10 points] **HAM DAG** (1 part)

A **Hamiltonian path** is a path in a directed graph that visits every vertex exactly once. Describe a linear time algorithm to determine whether a directed **acyclic** graph $G = (V, E)$ contains a Hamiltonian path. (**Hint:** It might help to draw a DAG which contains a Hamiltonian path.)

Solution: A directed Hamiltonian path would constitute a topological sort ordering, as any edge directed against the topological sort would constitute a cycle in the graph. Further, if a Hamiltonian path exists, it constitutes the only valid topological sort ordering, as any other ordering would violate some directed edge from the Hamiltonian path. So the algorithm is to find a topological sort order using depth-first search, and check whether every adjacent vertex is connected by an edge. If so, there is a Hamiltonian path. Depth-first search runs in $O(|V| + |E|)$, linear time.

Alternatively, find the longest path in the graph starting from any vertex, by adding an auxiliary vertex s with a directed edge from it to every vertex of the graph, assigning a weight of -1 to all edges, and find the minimum weight path to every vertex from s using topological sort relaxation in $O(|V| + |E|)$ time. If a minimum weight path to any vertex has weight $-|V|$, then such a path starts from the auxiliary vertex and visits every vertex of the original graph, constituting a Hamiltonian path in the original graph.

Rubric:

- 8 points for a correct linear time algorithm
- 2 points for running time analysis
- Partial credit may be awarded

Problem 4. [15 points] **Long Paths** (1 part)

A directed graph is **strongly-connected** if there exists a path from every vertex to every other vertex. Given a weighted strongly-connected directed graph $G = (V, E)$ containing both positive and negative edge weights, describe an $O(|V||E|)$ time algorithm to determine whether there exists a (not necessarily simple) path from vertex s to vertex t that has path weight greater than w.

Solution: Find a longest path from s to t using a similar algorithm to Problem Set 8 Question 3. Negate all edge weights and run Bellman-Ford from s in $O(|V||E|)$ time. If a negative weight cycle is not detected, return whether the minimum weight path to t is less than or equal to $-w$. Otherwise, if a negative weight cycle exists, there exists a path from s to t with weight greater than w: traverse any path from s to t that includes a vertex on the cycle (which exists because the graph is strongly connected), and then splice in as many trips around the cycle as necessary to make the path weight greater than w.

Rubric:

- 12 points for a correct $O(|V||E|)$ time algorithm
- 3 points for running time analysis
- Partial credit may be awarded

Problem 5. [15 points] **Unicycles** (1 part)

Given a **connected** weighted undirected graph $G = (V, E)$ having only positive weight edges containing **exactly one cycle**, describe an $O(|V|)$ time algorithm to determine the minimum weight path from vertex s to vertex t.

Solution: Given two vertices in a weighted tree containing only positive weight edges, there is a unique simple path between them which is also the minimum weight path. A depth-first search from a source vertex s in the tree results in a directed DFS tree in $O(|V|)$ time (since $|E| = |V| - 1$). Then relaxing edges in topological sort order of the directed DFS tree computes minimum weight paths from s in $O(|V|)$ time. Since G has one cycle, our strategy will be to break the cycle by removing an edge, and then compute the minimum weight path from s to t in the resultant tree.

First, we find the vertex v on the cycle closest to s by running depth-first search from s in $O(|V|)$ time (since $|E| = |V|$). To find v, one edge of the cycle will be the only DFS back edge e_1, and will terminate at vertex v, with the other edge of the cycle incident to v being a single outgoing DFS tree edge e_2. If s is on the cycle, $v = s$; otherwise the unique path from s to v does not contain e_1 or e_2. A shortest path from s to t cannot traverse both edges e_1 and e_2, or else the path would visit v at least twice, traversing a cycle of positive weight. Removing either e_1 or e_2 results in a tree, at least one of which contains the minimum weight path from s to t. Thus, find the minimum weight path from s to t in each tree using the algorithm described above, returning the minimum of the two in $O(|V|)$ time.

Rubric:

- 12 points for a correct $O(|V|)$ time algorithm
- 3 points for running time analysis
- Partial credit may be awarded

Problem 6. [30 points] **Doh!-nut** (2 parts)

Momer has just finished work at the FingSprield power plant (at location p), and needs to drive to his home (at location h). Momer knows that if his driving route comes within a one mile drive of a donut shop, he will stop and eat donuts, and his wife, Harge, will be angry. Momer knows the layout of FingSprield, which can be modeled as a set of n locations, with two-way roads of known length connecting some pairs of locations (you may assume that no location is incident to more than five roads), as well as the locations of the d donut shops and g grocery stores in the city. Note that for this problem, the two parts can be solved independently.

(a) [15 points] Describe an $O(n \log n)$ algorithm to find the shortest driving route from the power plant back home that avoids driving **within a one mile drive** of a donut shop (or determine that no such route exists). Please specify any graphs you may construct.

Solution: Construct a graph G with a vertex for each of the n city locations, and a undirected edge between two locations if there is a road connecting them, with each edge weighted by the positive length of its corresponding road. Each vertex has bounded degree, so the number of edges in G is $O(n)$. First, we identify vertices that are within a one mile drive of a donut shop location: create an auxiliary vertex x with a weight 0 outgoing edge from x to every donut shop location, and run Dijkstra from x. Remove every vertex from the graph whose shortest path to x is less than or equal to one mile, resulting in graph $G' \subset G$. If either p or h are not in G', then no route exists. Otherwise, run Dijkstra from p in G'. If no path exists to h, then no valid route exists. Otherwise, Dijkstra finds a shortest path from p to h, so return it (via parent pointers). This algorithm runs Dijkstra twice. Since the number of edges is $O(|V|)$, Dijkstra can run in $O(|V| \log |V|) = O(n \log n)$ time (e.g. using a binary heap to implement a priority queue).

Rubric:

- 3 points for graph pruning
- 8 points for a correct $O(n \log n)$ algorithm to prune graph
- 2 points for $O(n \log n)$ shortest path finding (Dijkstra)
- 2 points for running time analysis
- Partial credit may be awarded

(b) [15 points] Just as Momer is getting ready to leave, Harge calls to tell Momer to pick up some groceries on his way home. A grocery store will satisfy Momer's cravings, so he no longer needs to avoid passing close to donut shops on his drive from the power plant back home. Describe an $O(n \log n)$ algorithm to find the shortest driving route from the power plant that passes at least one grocery store location on the way home (or determine that no such route exists). Please specify any graphs you may construct.

Solution: Construct a graph G'' with two vertices $(v, 0)$ and $(v, 1)$ for each vertex v in G from part (a). Vertices $(v, 0)$ and $(v, 1)$ correspond respectively to either having not visited or having already visited at least one vertex corresponding to a grocery store location prior to arriving at city location v. For each edge $\{u, v\}$ in G, connect $(u, 0)$ to $(v, 0)$ if u does not contain a grocery store; and connect both $(u, 0)$ to $(v, 1)$ and $(u, 1)$ to $(v, 1)$ if u does contain a grocery store. Running Dijkstra from $(p, 0)$ to $(h, 1)$ will find the shortest driving route from p to h that passes at least one grocery store along the way (or determine that no such path exists if Dijkstra from p does not reach h). Since the size of G'' is at most twice as large as G, this application of Dijkstra can also be performed in $O(n \log n)$ time.

An alternative solution runs Dijkstra twice on G, computing shortest paths to each vertex from both h and p. Then loop through each grocery store vertex, and find a grocery store s whose shortest paths to h and p have minimum total weight. Return the shortest path from p to s then s to h. Again, this algorithm takes $O(n \log n)$ time.

Rubric:

- 12 points for a correct $O(n \log n)$ time algorithm
- 3 points for running time analysis
- Partial credit may be awarded

Problem 7. [15 points] **Selfish Coins** (1 part)

You are given a row of n coins, where each coin c_i has a different value v_i. You would like to select coins from the row to maximize the value of chosen coins. You may choose any subsequence of coins from left to right, except that if you select any two adjacent coins, you may not select either of the next two coins on their right (adjacent pairs of coins are **selfish**). For example, if the values of coins in the row are $(6, 7, 1, 8, 2)$, then choosing coins $\{7, 8, 2\}$ would be a subset achieving the maximum value sum 17 (if you had instead selected coins 6 and 7, the selfish condition would forbid you from also picking coins 1 or 8).

Given a row of coin values v_i for $i \in [1, n]$, describe a dynamic program that determines the maximum value of any subset of coins subject to the selfish constraint. Be sure to define a set of subproblems, relate the subproblems recursively, argue the relation is acyclic, provide base cases, construct a solution from the subproblems, and analyze running time. Correct but inefficient polynomial time dynamic programs will be awarded significant partial credit.

Solution:

1. **Subproblems**
   - $x(i)$: maximum value of any selfish subset from coin i to n (suffix)
2. **Relate**
   - Either does not use coin i, uses i and not $i + 1$, or uses both i and $i + 1$ (Guess!)
   - $x(i) = \max(x(i + 1), v_i + x(i + 2), v_i + v_{i+1} + x(i + 4))$ for $i \in [1, n - 1]$
3. **DAG**
   - Subproblems $x(i)$ only depend on strictly larger i, so acyclic
   - Solve in order of decreasing i
   - Base case: $x(n) = v_n$, $x(i) = 0$ for $i > n$ (no coins, no value!)
4. **Evaluate**
   - Solve subproblems via recursive top down or iterative bottom up
   - The maximum value of any selfish subset is $x(1)$.
5. **Analysis**
   - (# subproblems: $O(n)$) $\times$ (work per subproblem: $O(1)$) $= O(n)$

Rubric:

- 2 points for subproblem description
- 5 points for a correct recursive relation
- (1, 2) points for (acyclic, base cases)
- 1 point for specifying solution
- (2, 2) points, running time is (correct, efficient $O(n)$)

Problem 8. [15 points] **Alternating Sums** (1 part)

Given an array A containing n integers, describe a dynamic programming algorithm to find an increasing subsequence of array indices $B = (b_0, \dots, b_{m-1})$, with $0 \le b_0 < b_1 < \dots < b_{m-1} < n$, that maximizes the alternating subsequence sum:

$$\sum_{i=0}^{m-1} (-1)^i A[b_i] = A[b_0] - A[b_1] + A[b_2] - A[b_3] + \dots$$

For example, if $A = (5, 2, 7, 1, 3, 8)$, then $B = (0, 1, 2, 3, 5)$ would achieve the maximum alternating sum $5 - 2 + 7 - 1 + 8 = 17$. Be sure to define a set of subproblems, relate the subproblems recursively, argue the relation is acyclic, provide base cases, construct a solution from the subproblems, and analyze running time. Correct but inefficient polynomial time dynamic programs will be awarded significant partial credit.

Solution:

1. **Subproblems**
   - $x(i, j)$: maximum sum of any alternating subsequence from integers i to n, assuming the sum starts with a plus $j = +1$ or minus $j = -1$ operator
2. **Relate**
   - Either first integer is in alternating sum or not (Guess!)
   - $x(i, j) = \max(j \cdot A[i] + x(i + 1, -j), x(i + 1, j))$ for $i \in [1, n], j \in \{+1, -1\}$
3. **DAG**
   - Subproblems $x(i, j)$ only depend on strictly larger i, so acyclic
   - Solve in order of decreasing i, and j in either order
   - Base case: $x(n + 1, j) = 0$ for $j \in \{+1, -1\}$ (no integers, no sum!)
4. **Evaluate**
   - Solve subproblems via recursive top down or iterative bottom up
   - The maximum alternating sum is $x(1, +1)$.
   - Reconstruct maximizing sequence by storing parent pointers
5. **Analysis**
   - (# subproblems: $O(n)$) $\times$ (work per subproblem: $O(1)$) $= O(n)$

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 refer to the scratch paper next to the question itself.

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 refer to the scratch paper next to the question itself.

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 refer to the scratch paper next to the question itself.

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 refer to the scratch paper next to the question itself.