Final Exam

Math 182, Fall 2021

INSTRUCTIONS: Answer each question in the space provided. If you run out of room, use the blank pages at the end. You may consult two double-sided pages of notes. You may not use a calculator, phone or computer. If you believe there is a typo in a question, you may raise your hand to ask about it.

If you cannot figure out how to solve a problem, you may receive partial credit for solving an easier version of the same problem, as long as you state clearly that that's what you are doing.

ADVICE: I recommend that early on in the exam, you skim all the questions. If you find yourself stuck on a question for a long time, try switching to a different question. For the algorithm design questions, you will receive most of the points as long as your main idea is correct, even if your pseudocode is missing or incorrect. So it may be a good idea to try to first solve and write main ideas for the algorithm design questions and only start writing pseudocode after you have solved or gotten stuck on all of them. And if you ever find yourself getting panicked or having trouble thinking productively about a question, take a moment to take a deep breath and relax.

Name: ______________________________

UID: ______________________________

Question	Points	Score
1	15	
2	2	
3	3	
4	10	
5	10	
6	12	
7	12	
8	12	
9	12	
10	12	
11	0	
12	0	
Total:	100	

Don't turn over this page until you are told to do so.

1 Short Answer

The following three questions are all True/False or short answer. No justification is required.

Question 1: True/False (15 points)

(a) For any function $f\colon \mathbb{N} \to \mathbb{N}$, $f(n) \in O(f(n)^2)$.

◯ True ◯ False

(b) Suppose you have a priority queue implemented using a binary heap and you perform a series of n `insert` operations and n `delete-min` operations in such a way that the binary heap never contains more than $\sqrt{n}$ elements. The total running time of all $2n$ operations is $O(n\log(n))$.

◯ True ◯ False

(c) If a directed graph can be topologically sorted then it has no cycles.

◯ True ◯ False

(d) Suppose you implement a hash table using external chaining. If the hash table contains n elements and no chain has length more than $O(\log(n))$ then checking whether the hash table contains a given item takes $O(\log(n))$ time in the worst case.

◯ True ◯ False

(e) Recall that the Independent Set problem asks whether a given graph has an independent set (a set of vertices with no edges between them) larger than some given size and the Reachability problem asks whether a given graph has a path between two given vertices. True or false: a reduction of the Independent Set problem to the Reachability problem is a polynomial time algorithm A that, when given a graph G and vertices s and t in G, produces a graph G' and a number m such that there is a path from s to t in G if and only if there is an indepndent set in G' of size larger than m.

◯ True ◯ False

Question 2: Topological Sort (2 points)

Find a topological sort of the following graph.

$$\begin{tikzcd}
A \arrow[d] & & D \arrow[ll] \arrow[lld] \arrow[dr] & \\
& & & E \arrow[llld] \arrow[dl] \\
B & & C &
\end{tikzcd}$$

Topological Sort: ______________________

Question 3: Minimum Cut (3 points)

Find a minimum s-t cut of the following graph and state its weight. Feel free to indicate the cut by simply circling the vertices which are on the same side of the cut as s.

$$\begin{tikzcd}
& A \arrow[d, "1"] \arrow[r, bend right, "2"'] \arrow[drrr, bend right, "3"] & D \arrow[l, bend right, "3"'] & E \arrow[l, "3"'] \arrow[r, "1"] & t \\
s \arrow[ur, "2"] \arrow[dr, "5"'] & & & & \\
& B \arrow[uurr, "2"] & & C \arrow[uur, "2"] \arrow[r, "2"'] & F \arrow[uu, "3"']
\end{tikzcd}$$

2 Correct or Not?

Both of the following two questions contain a description of a problem and an algorithm that is supposed to solve that problem. For each question you should determine whether or not the algorithm is correct. If it is correct, prove it. If it is not correct, provide an example of an input on which it fails to find the correct answer.

Question 4: Find a MAST (10 points)

Problem. If $G = (V, E)$ is a weighted, connected, undirected graph, a set $E' \subseteq E$ is called an *almost spanning tree* of G if (V, E') is connected and $|E'| = |V|$. The *weight* of an almost spanning tree is defined in the same way as the weight of a spanning tree—as the sum of the weights of the edges in E'.

Input. A connected graph $G = (V, E)$ such that $|E| \geq |V|$.
Output. An almost spanning tree of G whose weight is as small as possible.

Algorithm. First use Prim's algorithm to find a minimum spanning tree of G and then add to it the lightest edge in G that it does not yet contain.

This algorithm is:

◯ Correct. ◯ Not correct.

Proof of correctness or counterexample.

Question 5: Triple Matching (10 points)

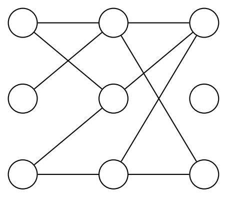
Problem. An undirected graph $G = (V, E)$ is called *strongly tripartite* if V can be partitioned into three sets V_1, V_2, V_3 such that every edge in E either has one endpoint in V_1 and one endpoint in V_2 or one endpoint in V_2 and one endpoint in V_3. The partition (V_1, V_2, V_3) is called a *tripartition* of G.

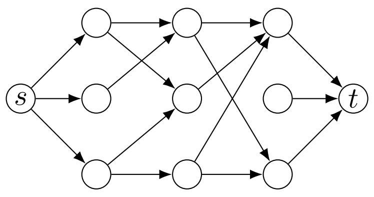
Suppose $G = (V, E)$ is a strongly tripartite graph with tripartition (V_1, V_2, V_3). A *triple matching* of G is a set $M \subseteq V^3$ of triples of vertices (i.e. lists of three vertices) in G such that no vertex occurs more than once in M and for each $(u, v, w) \in M$, $u \in V_1$, $v \in V_2$, $w \in V_3$ and $(u, v), (v, w) \in E$.

Input. A strongly tripartite graph $G = (V, E)$ and a tripartition (V_1, V_2, V_3) of G.
Output. The size of the largest possible triple matching of G.

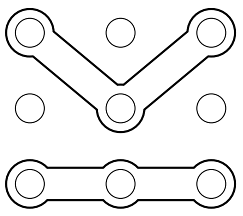
Algorithm. Create a network G' as follows. Start with V and add two new vertices, s and t. For each edge (u, v) in E such that $u \in V_1$ and $v \in V_2$, add a directed edge (u, v) to G' with weight 1. Similarly, for each edge (u, v) in E such that $u \in V_2$ and $v \in V_3$, add a directed edge (u, v) to G' with weight 1. Also add a directed edge of weight 1 from s to each vertex in V_1 and from each vertex in V_3 to t. Then use the Ford-Fulkerson algorithm to find a maximum flow on G'. Return the size of this flow as the size of the largest possible triple matching in G.

Example. Shown below is a graph G, the network G' formed from it as described above, and a triple matching of G.





Assume every edge has weight 1.



The largest triple matching has size 2

This algorithm is:

◯ Correct. ◯ Not correct.

Proof of correctness or counterexample.

3 Algorithm Design

Each of the next 5 questions asks you to design an efficient algorithm to solve some problem. For each problem, you should describe the main idea of your algorithm, give pseudocode for it and state its running time. You should give a brief justification of the running time (1-2 sentences should usually suffice). You do not need to provide a proof of correctness.

Partial credit. For each problem, you will receive at least 5 points (and sometimes more) if you give a correct algorithm which runs in polynomial time. Incorrect algorithms may or may not receive partial credit, depending on the details of the algorithm. Incorrect algorithms are more likely to receive partial credit if you state that you know the algorithm is incorrect and give an example of an input on which it fails.

Question 6: Painted Penguin Prefix (12 points)

You are the proud owner of n penguins, which are arranged in a line. You hired someone to paint all the penguins red, starting from penguin 1 and continuing in order until penguin n. However, the painter you hired was interrupted in the middle of their job and only painted the first few penguins. You want to find out how many of the penguins have already been painted. Design an efficient algorithm to solve this problem. For full credit, your algorithm should run in $O(\log(n))$ time.

Input. The number of penguins, n, and access to a function, `red`, which works as follows. For any $i \leq n$, `red`(i) returns `True` if and only if penguin i is painted red. You may assume this function runs in $O(1)$ time and that if `red`(i) returns `False` then so does `red`(j) for any $j \geq i$.

Output. The largest number i such that `red`(i) returns `True`.

Main idea.

Pseudocode.

Running time.

Question 7: Filling in an Array (12 points)

Suppose you play the following game with your friend. Your friend thinks of an array A of n numbers, which you will try to guess. To help you, your friend gives you a series of hints of the form "A[i] < A[j]" or "A[i] > A[j]" (where $1 \le i < j \le n$). After getting k such hints, you want to find an array B of real numbers (not necessarily integers) such that each hint you received about A is also true of B. Design an efficient algorithm to solve this problem.

Example. Suppose that $n = 5$, $k = 4$ and the hints you receive are A[1] > A[2], A[1] < A[3], A[2] < A[3], and A[4] < A[5]. Then one valid way to fill in B is `[5.5, 5, 6, 7.2, 7.3]`.

Input. A number n and arrays `L` and `G` such that for each $i \le n$, `L`[i] contains a list of all hints of the form "A[i] < A[j]" and `G`[i] contains a list of all hints of the form "A[i] > A[j]."

Output. A length n array B of real numbers such that for each hint of the form "A[i] < A[j]," B satisfies B[i] < B[j] and for each hint of the form "A[i] > A[j]," B satisfies B[i] > B[j].

Main idea.

Pseudocode.

Running time (in terms of both n and k).

Question 8: Road Trip (12 points)

You have decided to drive from Los Angeles to Boston for winter break and you want to figure out which hotels you should stay at along the way. Assume that there is only one highway from LA to Boston, that it is exactly n miles long and that there is exactly one hotel at each mile along this highway. Each day, you can drive at most k miles and then, if you are not yet at Boston, you must stop at a hotel for the night. Given the price of staying at each hotel for one night, you want to find the minimum possible total amount of money that you can spend on hotels during your trip.

Example. Suppose $n = 10$, $k = 3$ and the costs of the hotels, in order, are $3, 1, 4, 5, 2, 3, 7, 8, 1$ (note that there are only 9 hotels because after you have driven 10 miles, you have reached Boston and do not need a hotel). Then the minimum total cost is 7, which can be obtained by staying at the hotels at miles 2, 5, 6 and 9.

Input. An array A such that for each $i < n$, `A[`i`]` contains the cost to stay for one night at the hotel at mile i along the highway.

Output. The least number c such that there are indices $i_1 < i_2 < \ldots < i_l$ (indicating which hotels to stay at) which satisfy the two following constraints.

- $i_1 - 0$, $i_2 - i_1$, $i_3 - i_2$, …, $i_l - i_{l-1}$, $n - i_l$ are all less than or equal to k (i.e. you don't drive more than k miles per day, including the first and last days).
- $A[i_1] + A[i_2] + \ldots + A[i_l] = c$ (i.e. the total cost to stay at these hotels is c).

Note that you only need to return the minimum cost c and not the indices $i_1, \ldots, i_l$.

Hint. There is a dynamic programming solution with subproblems $C(i)$ defined as the minimum cost to travel the first i miles. You are free to use a different solution, however.

Main idea.

Pseudocode.

Running time (in terms of both n and k).

Question 9: Graph with Unreliable Edges (12 points)

Suppose $G = (V, E)$ is a directed graph and $s, t \in V$. Some of the edges in G have been marked as *unreliable*. You want to know if there is a path from s to t in G that uses at most 2 unreliable edges. Design an efficient algorithm to solve this problem.

Input. A directed graph $G = (V, E)$, vertices $s, t \in V$ and access to a function `unreliable`, such that for any edge $e \in E$, `unreliable(`e`)` returns `True` if and only if e is an unreliable edge. You may assume that this function runs in $O(1)$ time.

Output. `True` if there is a path from s to t in G that uses at most 2 unreliable edges and `False` otherwise.

Main idea.

Pseudocode.

Running time.

Question 10: Small Diameter Partition (12 points)

The *diameter* of an array of numbers is the difference between the largest element and the smallest element in the array. Note that if the array is sorted in increasing order then this is just the difference between the last and first elements.

Suppose A is an array of numbers, and $k > 0$. The *diameter* of a partition of A into k subarrays is the maximum diameter of any of those subarrays. Note that a partition of A into k subarrays is equivalent to a choice of k numbers $1 = i_1 < i_2 < \ldots < i_k$ indicating the beginning of each subarray (so the first subarray consists of $A[1], A[2], \ldots, A[i_2 - 1]$, the second consists of $A[i_2], A[i_2 + 1], \ldots, A[i_3 - 1]$ and so on). Furthermore, note that if A is sorted in increasing order then the diameter of the first subarray is $A[i_2 - 1] - A[i_1]$, the diameter of the second subarray is $A[i_3 - 1] - A[i_2]$, etc.

You want to design an efficient algorithm to solve the following problem: given a length n array of integers, A, sorted in increasing order and a number $k > 0$, find the minimum diameter of any partition of A into exactly k subarrays.

Example. Suppose $A =$ `[1, 5, 6, 8, 13, 15]` and $k = 3$. The minimum possible diameter of any partition of A into k subarrays is 3 and one partition with this diameter is `[1], [5, 6, 8], [13, 15]` (in other words, $i_1 = 1$, $i_2 = 2$, $i_3 = 5$).

Input. A length n array A of integers, sorted in increasing order, and a positive integer, k.

Output. The minimum possible diameter of any partition of A into k subarrays. In other words, the minimum possible value of $\max(A[n] - A[i_k], A[i_k - 1] - A[i_{k-1}], \ldots, A[i_2 - 1] - A[i_1])$ over all choices of indices $1 = i_1 < i_2 < \ldots < i_k \leq n$.

Main idea.

Pseudocode.

Running time (in terms of both n and k).

4 Extra Credit

The following two questions are extra credit. They are more challenging than the rest of the problems and are worth relatively few points, so I recommend that you only attempt them if you have finished the rest of the exam. For these questions, you do not need to provide pseudocode or an analysis of the running time, just the main idea of your algorithm.

Question 11: Faster Partitions (2 points (bonus))

Suppose that in question 10 the entries in A are all positive integers which are less than n^2. Show how to solve the problem in $O(n\log(n))$ time in this case.

Question 12: More Unreliable Edges (2 points (bonus))

Suppose that in the question 9, instead of being allowed to use at most 2 unreliable edges, you can use at most k unreliable edges. Describe an algorithm that runs in time $\Theta(|V| + |E|)$ no matter what k is. You may assume that the graph is undirected rather than directed.

Use this page if you run out of space on any problem. Be sure to indicate on the original page for the problem that your solution continues on a later page.

Use this page if you run out of space on any problem. Be sure to indicate on the original page for the problem that your solution continues on a later page.

Use this page if you run out of space on any problem. Be sure to indicate on the original page for the problem that your solution continues on a later page.

Use this page if you run out of space on any problem. Be sure to indicate on the original page for the problem that your solution continues on a later page.