University of Illinois at Urbana-Champaign
Department of Computer Science

Third Examination

CS 225 Data Structures and Software Principles
Sample Exam 1
75 minutes permitted

Print your name, netID, and lab section day/time neatly in the space provided below; print your name at the upper right corner of every page.

Name:
NetID:
Lab Section (Day/Time):

- This is a **closed book** and **closed notes** exam. In addition, you are not allowed to use any electronic aides of any kind.
- You should have 6 sheets total (the cover sheet, plus numbered pages 1-10). The last sheet is scratch paper; you may detach it while taking the exam, but must turn it in with the exam when you leave.
- Unless otherwise stated in a problem, assume the best possible design of a particular implementation is being used.
- Unless the problem specifically says otherwise, (1) assume the code compiles, and thus any compiler error is an exam typo (though hopefully there are not any typos), and (2) assume you are NOT allowed to write any helper methods to help solve the problem, nor are you allowed to use additional arrays, lists, or other collection data structures unless we have said you can.

Problem	Points	Score	Grader
1	20		
2	20		
3	20		
4	30		
Total	90		

1. **[Ghost of Euler – 20 points].**

   You have the following `EdgeNode` class:

```
class EdgeNode {
public:
   int index;  // index of target vertex
   EdgeNode* next;  // ptr to next edge
};
```

   Furthermore, you have a variable of type `Array<EdgeNode*>` that is indexed from `1` to the size of the array (given by the `Size()` method in the `Array` class). The array is an adjacency list implementation of a graph, of the kind we first discussed in lecture; the vertices have indices from `1` to `Size()`.

   You want to write a method **`numOddDegree`** which has one parameter, a reference to an `Array<EdgeNode*>` as described above. You can assume this adjacency list represents an undirected graph. You want to return the number of vertices with odd degree. (The degree of a vertex in an undirected graph, is the number of edges which that vertex is an endpoint for. So, you want to know how many vertices are endpoint to an odd number of edges.) You can assume the graph has no self-loops (i.e. no vertex has an edge to itself.)

```
int numOddDegree(Array<EdgeNode*>& graph) {
   // your code goes here
```

(Ghost of Euler, continued)

2. **[Vertices Nearby – 20 points].**

   You are given an adjacency matrix implementation of an unweighted, directed graph – the graph has vertices labelled with indices `0` through `n-1`, and you are given the value `n` and a two-dimensional array with `n` rows and `n` columns, both indexed from `0` through `n-1`. You want to write a method that takes those two values (the integer `n` and an `int**` to the two-dimensional array) and returns a `1` if the distance from any vertex to any other vertex is always `2` or less, and returns a `0` if there is at least one vertex for which the minimum distance to some other vertex is `3` or more. You are allowed to create additional one-dimensional arrays if you need to.

```
int DistanceCheck(int** graph, int n) {
   // your code goes here
```

(Vertices Nearby, continued)

3. **[Analysis – 20 points].**

   (a) You are given a graph which is implemented by an adjacency matrix. This matrix has `V` rows and `V` columns, and represents a graph with `V` vertices and `E` edges. What is the order of growth of the worst-case running time for adding a vertex to the graph? Express your answer in Big-$\mathcal{O}$ notation and explain convincingly why your answer is correct.

   (b) You are given a graph which is implemented via an adjacency list. This adjacency list is of the kind we first discussed in class – i.e. an array of vertices, with each cell of the array pointing to a linked list of edge nodes. You want to find the vertex with the greatest number of departing edges. What will be the worst-case running time of this algorithm, in terms of `V` and `E`? Express your answer in big-$\mathcal{O}$ notation and explain convincingly why your answer is correct.

4. [**Algorithms – 30 points (6 points each)**].

   (a) Give an example of a undirected, weighted graph that has two edges of equal weight, for which there is still a unique minimum spanning tree.

   (b) In the depth-first-search-based implementation of topological sort, if we moved the "post-visit" code to instead be "pre-visit" code (i.e. if we took the code we run after we explore a vertex's neighbors, and instead run it *before* we explore a vertex's neighbors), the algorithm will no longer work. Explain why.

(c) When we implement breadth-first-search, it is important that we mark a vertex as "encountered" *before* it is enqueued, rather than *after* it is *dequeued.* Explain why it is important – i.e. explain why we couldn't instead choose to mark a vertex "encountered" after dequeueing it.

(d) Explain cycle detection for Kruskal's algorithm. That is, explain why disjoint sets are used for this part of the algorithm (i.e. explain what the equivalence relation is that we are trying to model with our disjoint sets structure), and explain how the disjoint sets structure gets used to figure out whether to accept or reject an edge that is being considered.

(e) For the given graph, run Dijkstra's algorithm, indicating in the table below the distances at each vertex at the end of each step (d_v), and whether or not the vertex has been marked known yet at the end of each step (k_v).

	A	B	C	D	E	F	G
A	0	12	0	0	0	0	18
B	0	0	0	0	5	0	1
C	0	17	0	0	4	0	12
D	0	0	3	0	0	13	0
E	9	0	0	5	0	10	0
F	3	8	0	0	0	0	0
G	2	0	0	0	0	0	0

V	d_v	k_v	d_v	k_v	d_v	k_v	d_v	k_v	d_v	k_v	d_v	k_v	d_v	k_v	d_v	k_v
A	∞	0														
B	∞	0														
C	∞	0														
D	∞	0														
E	0	0														
F	∞	0														
G	∞	0														
-	Start		Step 1		Step 2		Step 3		Step 4		Step 5		Step 6		Step 7	

(scratch paper)