University of New Mexico
Department of Computer Science
Midterm Examination II
CS362: Data Structure and Algorithms

Name:______________________________
Email:_______________________________

Instructions:

1. **Please silence your cell phones.**
2. Write your name and email address legibly in the space provided above.
3. Write your name legibly at the upper right hand corner on each page.
4. You are allowed to use a one page double-sided "cheating sheet" that you have brought to the exam and a calculator. Nothing else permitted.
5. You must not discuss the questions with anyone except the professor in charge.
6. Write your answers legibly.
7. There are **5 problems** in this exam.
8. For algorithmic problems, **do not give detailed pseudo-code**. Describe the key ideas and key steps of your solutions, correctness argument, and provide running time analysis. In the case, where you are using a well-known algorithm (e.g., merge sort or linear time selection) as a subroutine **without modification**, just cite the well-known algorithm and its running time. Do not repeat the steps of the well-known algorithms.
9. Don't spend too much time on any single problem. All questions are weighted equally. If you get stuck, move on to something else and get back later.
10. In the problems where you are asked to design an efficient algorithm, your points earned will depend on the efficiency of your algorithm. However, any solution is better than nothing.
11. Good luck, and enjoy the exam!

1. Suppose an arithmetic expression is given as a tree. Each leaf is a number and each internal node is one of the standard arithmetic operators $+, -, \times, \div$. For example, the expression: $1\times 2 \div 3 - 4\times(5+6)$ is the binary tree in the following figure. Design an efficient algorithm for evaluating such an expression. (You may assume the tree is a well-formatted binary tree, where the root is given, and each node has pointers to its left and right children.)

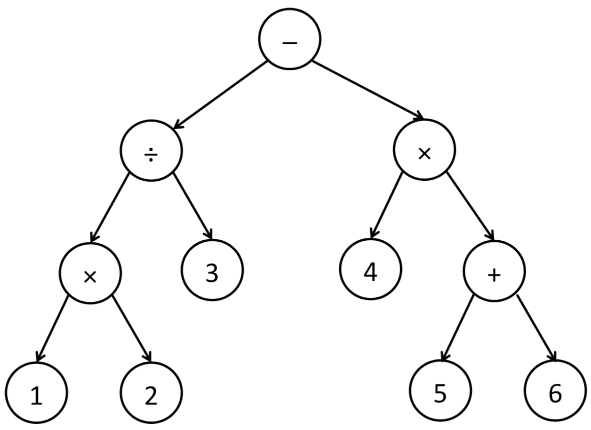

Figure 1

Solution: Uses the in-order traversal of trees.

2. The single destination shortest paths problem for a directed graph seeks the shortest path from every vertex to a specified destination vertex *d*. Give an efficient algorithm to solve the single-destination shortest paths problem on a general graph that may have negative edge costs and loops.

Solution:

Reverse the directions of all the edges, and then applying single source shortest paths.

3. A vertex cover of a graph $G(V, E)$ is a subset of vertices V' such that each edge in E is incident on at least one vertex of V' (See Figure 2 for illustrations). Give an efficient algorithm to find a minimum-sized vertex cover if G is a tree.

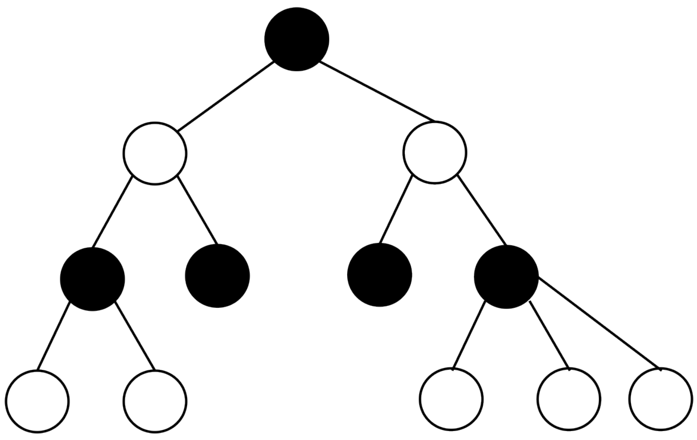

Figure 2 The vertex cover (shaded nodes) of a tree.

Solution:
Observe that the leaves can't be in the vertex cover.
Therefore, we should start the breadth first search from the leaves.
Make all of the immediate ancestors of the leaves to be part of the vertex cover.
Remove the edges they cover and repeat the process until done.

We can also solve this recursively.

4. Let *v* and *w* be two vertices in a directed graph *G*, where every edge has unit weight, i.e., the weight of every edge is 1. Design an efficient algorithm to find the number of different shortest paths between *v* and *w*. Note that two paths are different if they differ by at least one vertex or an edge.

Solution:

Notice that the shortest path length in a graph with unit edge weight is the number of edges on the path.

Breadth first search is the most efficient algorithm for calculating the shortest path in a unit weight graph. We will modify BFS to maintain the number of different shortest paths while exploring the graph.

For each vertex, we shall maintain two additional attributes, the shortest path length (L), and the number of different shortest paths (N).

The key is that when we update the shortest path length, we also update the number of different shortest path to a node.

```
Initialization:
For the source, L is set to 0, N is set to 1.
For all other vertices, L is set to ∞, N is set to 0.
Unmark all vertices and edges.
Mark source s as visited.
Enter the source into an empty queue Q.
While Q is not empty.
v = de_queue (Q)
For all vertices w adjacent to v:
    If w is unmarked, mark w as visited. L(w) = L(v) + 1. N(w) = N(w) + N(v).
    Enter w into Q.
```

5. Suppose you are asked to partition 2*n* players into two teams of *n* players each. Each player has a numerical rating that measures how good he/she is at the game.

(1) Suppose the goal is to divide the players as unfairly as possible, so as to create the maximum talent gap between the two teams. Show how this can be done efficiently.

For example, assume you are give 6 players with the following rating 1, 3, 4, 2, 7, 10. The solution will be to partition the players into the following two teams, where the first team include players with ratings 1, 3, 2, while the second team include players with ratings 4, 7, 10. The talent gap between the two teams is (1+ 3 + 2) – (4 + 7 + 10) = –15.

(2) Suppose the goal is to divide the players as fairly as possible, so as to create the minimum talent gap between the two teams. Show how this can be done efficiently

For example, assume you are give 6 players with the following rating 1, 3, 4, 2, 7, 10. The solution will be to partition the players into the following two teams, where the first team include players with ratings 1, 3, 10, while the second team include players with ratings 4, 2, 7. The talent gap between the two teams is (1+ 3 + 10) – (4 + 2 + 7) = 1

Solution:

(1) Sort all the players based on their rating. Put the half with the highest ratings into one team.

To be more efficient, we can use the medium selection algorithm to find the medium in linear time and then partition the players.

(2) Let S be the sum of the ratings of every player.

```
For $K$ = S/2 down to 1,
    Check if there is a knapsack solution with a knapsack size of $K$.
    If yes, break, and the players in the knapsack is in one team.
```