

CS5050 Homework 2, Chapter 6

Due on date specified by Eagle (due at class time) 10 points

Written homework provides an excellent framework for achieving the goals of obtaining a working knowledge of data structures, perfecting programming skills, and developing critical thinking strategies to aid the design and evaluation of algorithms. Since programming has a high overhead in terms of program entry and debugging, all important topics in this course cannot be covered via programming projects. Written homework exercises allow students to learn important material without a high time investment. Although the point value is low, the benefits are great. You can perfect your design skills without spending hours at the computer and can get feedback on your thinking skills from your study partners. Students who consistently do quality homework have far superior test scores.

Because assignments are done as a group and any questions are discussed in class or during office hours, written solutions to the homework will not be provided.

Note, these exercises may be done in groups of one, two, or three. If more than one person is involved, list all the names on ONE set of answers. Groups may change throughout the quarter. Answers should not be compared with others not in your group.

These are typed exercises, but you are certainly encouraged to actually code the programming segments if you have time.

1. Let's revisit a problem from assignment 1. Suppose we have a collection, A, of n positive integers that add to N. Design an efficient **dynamic programming** algorithm for determining whether there is a subset B of A such that the sum of the numbers in the subset equals the sum of the numbers of A which are not chosen for the subset. Show the pseudo-code solution. The handout given out in class which shows the code for the [matrix chain problem](#) may be of help. The trick for coding dynamic programming problems is to first find the recursion. Then, the matrix is created to store the answers to all the subproblems. So, for example, if your recursion was

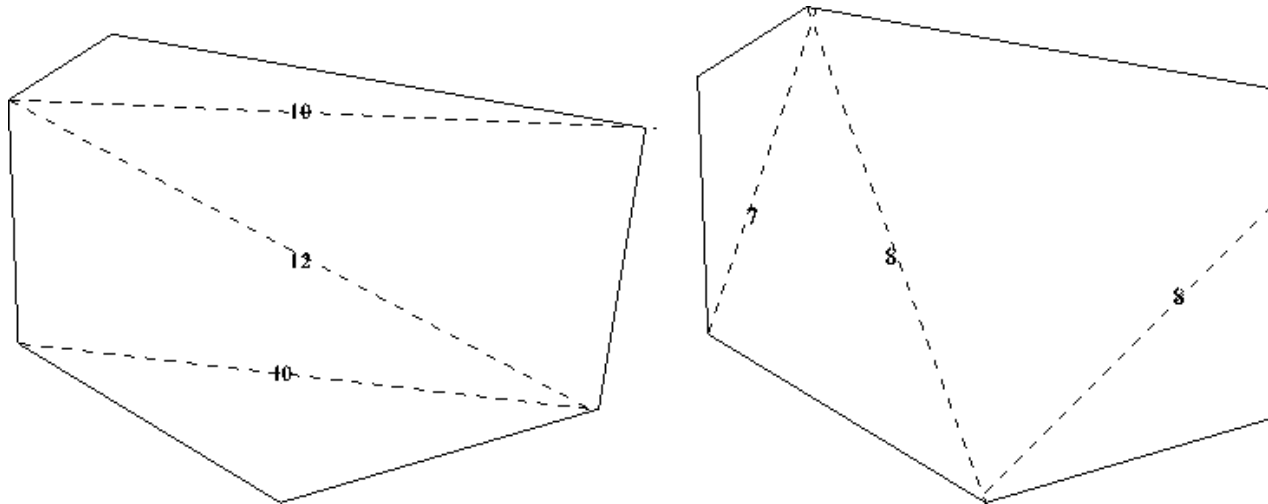
```
boolean doit (int beg, int end) // Notice this is NOT a suggestion for this
particular problem
```

Your matrix would have rows used as beg, columns used as end, and what was stored in the matrix would be booleans. The only thing left to do would be to decide the order in which the rows/columns/diagonals would be filled.

***HINT:** This problem is quite similar to the "making change" problem. You have to find a way that the problem gets smaller – either in the amount you are dealing with or the number of items you are allowed to use in the solution. Think "use it" or "don't use it".*

- Let's revisit another problem from assignment 1. A convex polygon is a polygon in which a line connecting any two vertices lies entirely within the polygon. A triangulation of a convex polygon, P , is an addition of diagonals connecting the vertices of P so that triangles are formed. The weight of a triangulation is the sum of the lengths of the diagonals. Give an efficient **dynamic programming** algorithm for computing a minimum weight triangulation. *Show the pseudo-code.*

The example below shows two possible triangulations of the polygon. Assuming the labels represent weight, the right triangulation is better.



- A computer network connects schools using a communication link that forms a free tree (a connected, acyclic, undirected graph). You want to find a node for the district office such that the length of the longest path from that node to any other node is minimized. Design an efficient algorithm to find a center node. Is the center unique? If not, how many centers can a free tree have?
- Given an [adjacency list](#) representation for a graph, write the pseudocode to perform a BFS traversal using a queue.
- In order to do video-phone communication, the number of links to transmit the video signal cannot exceed four. Suppose the network is represented by a graph. Design an efficient algorithm that computes, for each station, the set of stations it can reach using no more than four links.
- Write the pseudo code to do a DFS traversal of a **directed graph** such that each arc is identified as being a discovery, back, forward, or cross edge. **Hints: Note that your pseudo-code must be specific. You cannot assume a node "knows" its ancestors in the tree. Also, be careful about complexity. Repeatedly having the whole tree traversed (looking for a descendant) is terribly inefficient.**
- An Euler tour of a directed graph G with n vertices and m edges is a cycle that traverses each edge exactly once (according to its direction) and returns to the original location. Such a tour always exists if G is connected and the in-degree equals the out-degree of each vertex. Write the pseudo code for a $O(m+n)$ time algorithm for **finding** a Euler tour of such a digraph (rather than just determining there is one). In other words, you have to list the nodes in order that they are

visited such that all edges will be traversed. The Euler tour is a series of edges to traverse so that you go from the start node, traverse all the edges, and get back to the start node without traversing any edges more than once or missing any edges. Backtracking is normally very expensive; to stay within the $O(m+n)$ time, you will need to avoid backtracking.

Hints:

There is a special way of thinking about complexity that you need to understand. Consider the following “homey” example. Suppose I want my house to look neat when someone rings the doorbell. When the doorbell rings, I have to walk to the door to open it. Thus, my time to get there can’t be lessened. As I go to the door, if I see things out of place, I’m allowed to pick them up on the way to the door. This doesn’t cost me anything beyond a constant times my original cost of going to the door. So it might take me twice as long, but that is okay (because two is a constant). However, if I decide I need to wash my windows before opening the door, that is definitely going to slow me down. I can’t wash my windows “for free” in the complexity way of thinking– it changes the complexity of my algorithm. Instead of answering the door being $O(n)$ where n is the maximum distance from anywhere in my house to the front door, answering the door becomes $O(n+w)$ where w is the number of windows I have.

Consider the Euler tour. Suppose I want to mark each edge in some special way. Can I do that and stay in my $O(n+m)$ complexity allowed for a normal traversal? Sure. It is “on the way”. You had to visit each edge to traverse the graph, so if you do something else to each edge it doesn’t change the complexity (it just changes the constant multiplier). If you are trying for an $O(n+m)$ algorithm, you can touch the nodes and edges any constant number of times, but if you start to touch each node n times (for example), you change the complexity to $O(n^2)$.