CSE 101 Homework 3
Due Nov. 14
Backtracking and Dynamic Programming
100 points total = 10 %

Addition chains (20 points) An *addition-chain* for n is a sequence of numbers starting with 1 and ending with n so that any element of the sequence other than 1 is the sum of two (not necessarily distinct) earlier elements. For example, two addition-chains for 13 are $1, 2 = 1 + 1, 4 = 2 + 2, 8 = 4{+}4, 9 = 8{+}1, 13 = 9{+}4$ and $1, 2 = 1{+}1, 3 = 1{+}2, 5 = 3{+}2, 8 = 5{+}3, 13 = 8 + 5$. The *cost* of an addition-chain is the total number of additions, or, equivalently, the number of elements -1. So both addition chains above have cost 5. Describe and give a worst-case analysis of a back-tracking algorithm to find the minimum cost addition chain for input n.

Dominating Set (20 points) Let $G = (V, E)$ be an undirected graph. A *dominating set* in G is a set of vertices $D \subseteq V$ so that every vertex x is either in D or has a neighbor in D, i.e., either $x \in D$ or $N(x) \cap D \neq \emptyset$. Give a backtracking algorithm to find a minimum size dominating set in a graph G. (Hint: You may need to generalize the problem. Think of dividing the remaining vertices into R those that still require a covering element, and C, those that are already covered.) Give a (not necessarily tight) worst-case time bound for your algorithm.

Gizmos Consider the following problem. You wish to purchase (at least) n identical gizmos. Gizmos come in packages of different sizes and different prices. You can buy any number of packages of each size , as long as the total number is at least n. You wish to find the minimum total price of such a set of packages.

The input is given as n and an array $Packages[1..m]$, where each $Package[i]$ has a positive integer field $Package[i].size$ and a positive real field $Package[i].price$ giving the number of gizmos in the package and the price of the package.

A recursive algorithm to solve this problem is:

BestPrice[n : $positiveinteger$, $Packages[1..m]$: array of pairs ($size$: integer, $price$: real).]

1. $MinPrice \leftarrow$ inf;
2. For $d = 1$ to m do:
3. begin;
4. IF $Packages[d].size \geq n$ THEN $TempPrice \leftarrow Packages[d].price$
5. ELSE $TempPrice \leftarrow Packages[d].price + BestPrice(n - Packages[d].size, Packages)$;
6. IF $TempPrice < MinPrice$ THEN $MinPrice \leftarrow TempPrice$;

7. end;
8. Return $MinPrice$.

Part 1: 3 points Show the recursion tree of the above algorithm on the following input: $n = 6$, packages: buy 5 for \$ 12, 3 for \$8 or 2 for \$6.

Part 2: 3 points Give a bound on the worst-case number of recursive calls the recursive algorithm could make in terms of n and m.

Part 3: 8 points Give a dynamic programming version of the recurrence.

Part 4: 3 points Give a time analysis of this dynamic programming algorithm, in terms of n and m.

Part 5: 3 points Show the array that your algorithm produces on the above example.

Library storage (20 points) Consider the following problem. A library has n books that must be stored in alphabetical order on adjustable height shelves. Each book has a height and a thickness. The width of the shelf is fixed at W, and the sum of the thicknesses of books on a single shelf must be at most W. The next shelf will be placed on top, at a height equal to the maximum height of a book in the shelf. Give an algorithm that minimizes the total height of shelves used to store all the books. You are given the list of books in alphabetical order, $b_i = (h_i.t_i)$, where h_i is the height and t_i is the thickness. Give a back-tracking algorithm for this problem, based on the decision, which is the last book on the first shelf? (10 points). Then convert it to an efficient dynamic programming algorithm (10 points).

Solving an addition chains problem (20 points) For each member of your study group, convert his or her first and last initials into a number between one and 26, alphabetically (A is 1, B is 2, etc.). Add up all the converted initials for all members of your group, and add 40. For that value of n, give the smallest possible addition chain. You can implement the algorithm from part 1, or use pen and paper, but your answer must be correct , i.e., no smaller chain can exist. You must explain how you know no smaller chain exists, either by explaining the algorithm you used, or giving a case analysis.