Algorithms Homework 5*

Fundamental techniques: Dynamic programming and greedy algorithms

Reading: GT Chapter 5

1. How can we modify the dynamic programming algorithm for matrix multiplication from simply computing the most efficient way to multiply, to computing the specific order (parenthetisation) for this computation?

2. Same question for 0-1 knapsack: How can we modify the dynamic programming algorithm from simply computing the best value for the 0-1 knapsack problem, to computing the actual set that gives this value?

3. Argue that the following greedy strategies do not work for the 0-1 knapsack problem by giving a counter-example in each case:

   a. Select the items in order of their values; that is, select with largest value first, and so on.

   b. Select the items in order of their value-per-pound; that is, select the item with the largest value-per-pound first, and so on.

4. A game-board consists of a row of n fields, each consisting of two numbers. The first number can be any positive integer, while the second is 1, 2, or 3. An example of a board with $n = 6$ could be the following:

17	2	100	87	33	14
1	2	3	1	1	1

   The object of the game is to jump from the first to the last field in the row. The top number of a field is the cost of visiting that field. The bottom number is the maximal number of fields one is allowed to jump to the right from the field. The cost of a game is the sum of the costs of the visited fields.

   Let the board be represented in a two-dimensional array $B[n, 2]$. The following recursive procedure (when called with argument 1) computes the cost of the cheapest game:

*Collaboration is allowed, even encouraged, provided that the names of the collaborators are listed along with the solutions. Write up the solutions on your own.

```
Cheap(i)

     IF i>n THEN return 0
     x=B[i,1]+Cheap(i+1)
     y=B[i,1]+Cheap(i+2)
     z=B[i,1]+Cheap(i+3)
     IF B[i,2]=1 THEN return x
     IF B[i,2]=2 THEN return min(x,y)
     IF B[i,2]=3 THEN return min(x,y,z)

END Cheap
```

(a) Analyze the asymptotic running time of the procedure.

(b) Describe and analyze a more efficient algorithm for finding the cheapest game.

5. (C-5.10) Suppose we are given a collection $A = \{a_1, a_2, ..., a_n\}$ of n positive integers that add up to N. Design an $O(nN)$ -time algorithm for determining whether there is a subset B in A such that $\sum_{a_i inB} a_i = \sum_{a_i \in A-B} a_i$.