# Algorithm Design and Analysis Homework #2 Due: 2:20pm, Thursday, October 17, 2013 TA email: ada@csie.ntu.edu.tw

### === Homework submission instructions ===

- For Problem 1, commit your source code and a brief documentation to the SVN server (katrina.csie.ntu.edu.tw). You should create a new folder "hw2" and put these two files in it.
- The filenames of the source code, Makefile, and the documentation file should be "main.c" and "report.txt", respectively; you will get some penalties in your grade if your submission does not follow the naming rule. The documentation file should be in plain text format (.txt file). In the documentation file you should explain how your code works, and anything you would like to convey to the TAs.
- You have to login to judgegirl system (http://katrina.csie.ntu.edu.tw/judgegirl) to make it judge your program.
- For Problem 2 through 5, submit the answers via the SVN server (electronic copy) or to the TA at the beginning of class on the due date (hard copy).
- Except the programming assignment, each student may only choose to submit the homework in only one way; either all in hard copies or all via SVN. If you submit your homework partially in one way and partially in the other way, you might only get the score of the part submitted as hard copies or the part submitted via SVN.
- If you choose to submit the answers of the writing problems through SVN, please combine the answers of all writing problems into only ONE file in the pdf format, with the file name in the format of "hw2\_[student ID].pdf" (e.g. "hw2\_b01902888.pdf"); otherwise, you might only get the score of one of the files (the one that the grading TA chooses).
- Discussions with others are encouraged. However, you should write down your solutions by your own words. In addition, for each problem you have to specify the references (the Internet URL you consulted with or the people you discussed with) on the first page of your solution to that problem.
- NO LATE SUBMISSION IS ALLOWED for the homework submission in hard copies - no score will be given for the part that is submitted after the deadline. For submissions via SVN (including the programming assignment and electronic copies of the writing problems), up to one day of delay is allowed; however, the score of the part that is submitted after the deadline will get some penalties according to the following rule (the time will be in seconds):

LATE SCORE = ORIGINAL SCORE  $\times (1 - DelayTime/86400)$ 

# Problem 1. Giant Rectangle

#### (35%)

Hey guys, you have solved the problem in the monument and got a lot of treasure, and your reputation is resounded through the universe. Now the Rectangle King invites you to solve the ancestral "Giant Rectangle" problem.

The problem says, "We rectangles can be form by smaller rectangles. Once you know how you composed, you may gain the greatest power ever. Now given a giant N x M rectangle and lots of 1 x 2 rectangles, you have to tell me the maximum number of ways  $1 \ge 2$  small rectangles can form the giant rectangle"

Note: Every small rectangle can be rotated by 90 degrees. So you can either say you have lots of  $1 \ge 2$  and  $2 \ge 1$  rectangles.

Hint 1: The time complexity is  $O(M * N * O(2^{\min(M,N)}))$ . The array used for DP may be huge.

Hint 2: Try to do dynamic programming on every 1x1 block. Suggestion is to try all possibilities until you must fill the block.

## $\operatorname{Report}(10\%)$

Please write down how you derived the algorithm, how you implemented it, and analyze the running time and space usage of your algorithm using asymptotic notations.

## Coding(25%)

### Input format:

This problem contains several trials. The first line contains the total number (N) of trials  $(N \leq 50)$ . The following N lines contains each giant rectangle's size  $(m, n|1 \leq m, n < 14)$ .

### **Output format:**

Please output the maximum number of ways 1 x 2 small rectangles can form each giant rectangle in a single line. If the giant rectangle cannot be formed by small rectangles, please output 0.

```
Sample input:
```

```
2
2 2
1 1
Sample output:
2
0
```

**Problem** 2. All-Pairs Shortest-Paths Problem (15%)

Given a simple directed graph G = (V, E) where |V| = n and a weight function

 $w: V \times V \to \{x: x \ge 0\}$  that returns the weight of edges. All vertices in the graph are numbered from 1 to n; i.e.,  $V = \{1, 2, 3, ..., n\}$ . Note that when  $(u, v) \notin E$ ,  $w(u, v) = \infty$ ; when  $u \in V$ , w(u, u) = 0.

We want to design an algorithm to compute an  $n \times n$  matrix d where  $d_{ij}$  is the distance from vertex i to vertex j. We would achieve this by computing n+1  $n \times n$  matrices  $d^{(0)}, d^{(1)}, d^{(2)}, ..., d^{(n)}$  sequentially.  $d_{ij}^{(k)}$  is the shortest path from vertex i to vertex j where all intermediate vertices of the path (vertices other than start vertex and end vertex) are in  $\{1, 2, 3, ..., k\}$ . If there is no such path,  $d_{ij}^{(k)} = \infty$ .

Please answer the following problems.

- 1.  $\forall i \in V, j \in V, d_{ij}^{(0)} = ?$
- 2.  $\forall i \in V, j \in V, k \in V, d_{ij}^{(k)} = ?$
- 3.  $\forall i \in V, j \in V, d_{ij} = ?$
- 4. It won't be difficult to impelment the algorithm and make it run in  $O(n^3)$  time. However, it would also need  $O(n^3)$  space to store all the matrices.

Please give an pseudocode of an implementation for this algorithm and justify that it needs only  $O(n^2)$  space. You need to explain why your implementation still works.

### **Problem** 3. (15%)

Nowadays, we have different values of coins, the 1-dollar ones, the 5-dollar ones, the 10-dollar ones, and finally, the 50-dollar ones. Since coins are heavy, we always want to carry as few coins as possible.

One day Hsinmu just comes up with an idea: how about changing the denomination? If the denomination were changed to [1, 6, 10, 50], what would happen? Not within a second he finds out that, the minimum number of coins that would make up "12 dollars" would be two, with two 6-dollar coins, rather than three, with a 10-dollar coin plus two 1-dollar coins.

So here comes the questions for you:

- a. (8%) Given the total amount of money M to be made up, and a list L[1...n] that describes the values of the coins, your mission is to find the minimum number of coins that can make up M dollars. Design a dynamic-programming algorithm to solve the problem. Write down the pseudo-code of the algorithm and describe your idea. Also, analyze the running time of your algorithm.
- b. (7%) What if the numbers of the coins are limited? For example, if there are only three 1-dollar coins and ten 5-dollar coins, would it cause any differences? Describe where your algorithm needs to be modified, and analyze the running time. You can assume that

the total amount of money to be made up, M, the denomination, L[1...n], and the maximum number of coins that you can use for each kind, N[1...n], are provided.

### **Problem** 4. Palindrome(15%)

A palindrome is any string that is exactly the same as its reversal, like I, or LOL, or RACECAR, or AMANAPLANACATACANALPANAMA.

- 1. (5%) Design an algorithm to nd the length of the longest subsequence of a given string that is also a palindrome. For example, the longest palindrome subsequence of MAHDYNAM-ICPROGRAMZLETMESHOWYOUTHEM is MHYMRORMYHM, so given that string as input, your algorithm should output the number 11. Please show your psudeo code in detail to describe your algorithm and analyze the time compexity. (restriction:  $O(n^2)$ )
- (10%)Any string can be decomposed into a sequence of palindromes. For example, the string BUBBASEESABANANA (Bubba sees a banana.) can be broken into palindromes in the following ways (and many others):

$$\begin{split} &BUB + BASEESAB + ANANA\\ &B + U + BB + A + SEES + ABA + NAN + A\\ &B + U + BB + A + SEES + A + B + ANANA\\ &B + U + B + B + A + S + E + E + S + A + B + A + N + A + N + A \end{split}$$

Design an efficient algorithm to find the smallest number of palindromes that make up a given input string. For example, given the input string BUBBASEESABANANA, your algorithm would return the integer 3. Please show your psudeo code in detail to describe your algorithm and analyze the time comparity. (restriction:  $O(n^3)$ )

**Problem** 5. Birds in the tree (20%)

Given a tree T = (V, E), where |V| = n, there are many birds living on its tree nodes. Each tree node has its own capacity  $c_i, i \in V$ , which limits the number of birds on the node. These birds have a strange habit. The bird stays peace with other birds on its node. But If there is any other bird on the node which is directly connected to its node, they will fight together.

You are asked to design a dynamic programming algorithm to find the maximum number of birds in the tree without fight. The output should include one of bird allocation which maximize the number of birds. Please give the pseudo code of your algorithm in detail (14%), and justify that your algorithm runs in O(n)(6%).



Figure 1: an example

Figure 1 is an example. The numbers in nodes are the capacities. The maximum number of birds is 6, and the corresponding allocation is the set of yellow nodes.