CS29003: Algorithms Laboratory
Lab Test (ODD PC)
Date: October 25, 2018

IMPORTANT INSTRUCTIONS

- Make sure you have got the correct question paper based on your PC No. (Odd or Even).
- You will submit a single C file for the lab test. Name your file $< PCNo > \_ < RollNo > .c$, For example, if your roll no. is 17CS10002 and you are sitting in PC No. 11, name your file $11\_17CS10002.c$.
- The first few lines of your program must contain comments with the following information as shown:

```
NAME:
ROLL NO:
PC NO:
```

- **There are two alternative problems. Alternative Problem 1 is for 100% marks. Alternative Problem 2 is easier but is for 35% marks only. You should do EITHER 1 OR 2, and NOT both. If you do both, we will grade ONLY the part for Alternative Problem 1 and ignore the rest.**
- Do not use any global variables. Indent your program properly.

ALTERNATIVE PROBLEM 1 (for 100% marks)

Suppose that a project has N essential and independent components, numbered from 1 to N, all of which must be completed for the successful completion of the project. Each of these components has a probability of success which depends on the number of people working on the component. Thus the probability of success of the project is the product of the probability of success of each of the components.

Let $p(k, j)$ denote the probability of success of component k if j number of people are assigned to that component. It is given that $p(k, 0) = 0$ for all k, so at least one person must be assigned to each component. Also, for a component k, $p(k, j)$ values are non-decreasing with increasing values of j (i.e., the chance of success never decreases if more people are assigned to the component).

The project has a total of P people ($P \geq N$) that can be allocated to the different components; it is assumed that all people are equally competent in any of the components and therefore can be assigned to any component. You need to assign the P people to the N components so that the probability of success of the project is maximized.

You are required to design an $O(NP^2)$ time dynamic programming algorithm for the problem. The algorithm should print out the maxmimum probability of success of the project, and the assignment of the number of people to each component that gives the maximum success probability in the format shown in the sample output. There may be more than one assignment that gives the same maximum success probability; in such cases, you can print any one assignment that gives the maximum success probability.

[Part 1]: Write, as comments in your program, the following clearly:

1. The definition of the subproblem that you choose.
2. The recursive formulation used in the DP solution, including the base case. Write using notation as you have seen in text, do not write long sentences.

[Part 2]: Write a *main()* function to do the following (exactly in this order):

1. Read in N (assume $N < 10$).
2. Read in P (assume $P < 50$. You can also assume P will be entered as $\geq N$, no need to check).
3. Read in the $p(k, j)$ values for each component exactly in this order in a 2-d array (enter the values in each row in non-decreasing order):
   $p(1,1), p(1,2), p(1,3), \ldots, p(1,P)$
   $p(2,1), p(2,2), p(2,3), \ldots, p(2,P)$
   $p(3,1), p(3,2), p(3,3), \ldots, p(3,P)$
   $\vdots$
   $p(N,1), p(N,2), p(N,3), \ldots, p(N,P)$
4. Implement the DP algorithm to find the maximum success probability and the assignment of people to components (you can code the algorithm directly in main(), no separate function is needed. However, you are free to write any helper function you may want and call it from main() also). **You are NOT allowed to use Memoization.**

Sample Output

```
Enter N: 3
Enter P: 5
Enter the probabilities:
0.1 0.3 0.5 0.6 0.6
0.1 0.2 0.4 0.6 0.8
0.2 0.4 0.4 0.7 0.9
The maximum success probability is 0.012
Assignment of people for max success probability:
Component 1: 2
Component 2: 2
Component 3: 1
```

ALTERNATIVE PROBLEM 2 (for 35% marks)

Given two strings $A = a_1a_2a_3 \ldots a_m$ and $B = b_1b_2b_3 \ldots b_n$ of lengths m and n respectively, you have to implement a dynamic programming algorithm to find the length of the longest common subsequence (LCS) of A and B.

Write a *main()* function to do the following (exactly in this order):

1. Read in m and n (exactly in this order, assume $m, n < 50$).
2. Read in A, B (exactly in this order).
3. Implement the DP algorithm to find and print the length of the LCS.

You can assume that A and B are entered with the correct sizes (m and n), no need to check.

Sample Output

```
Enter m: 11
Enter n: 9
Enter A: algorithmic
Enter B: logarithm
The length of the  LCS of A and B is  7
```