Assignment 3: Dynamic Programming

Due: In class, in the week Oct 14-18

Section B1

(as announced previously) Given two strings *s* and *t* with lengths *m* and *n* respectively, *and* match, mismatch and gap scores, implement dynamic programming algorithms to find optimal global sequence alignment and local sequence alignment of the two strings. Both algorithms must run in time O(*mn*).

For details on the algorithms and sample input-output, please see
https://en.wikipedia.org/wiki/Needleman%E2%80%93Wunsch_algorithm
https://en.wikipedia.org/wiki/Smith%E2%80%93Waterman_algorithm

Input format:
The first line of input file will contain string lengths and the scores separated by spaces. The next lines will contain two strings.

7 8 1 -1 -1
GCATGCT
GATTACAA

Output: The actual optimal alignments and the optimal scores.

Section B2

Change making: You are given n types of notes given by their denominations $d_1, \ldots, d_n$ and a value, V. In addition, you are given numbers $k_1, \ldots, k_n$ denoting the numbers of each type of note you have in hand. Give an algorithm to make change for V using fewest number of notes with the restriction that the i-th type of note can be used at most k_i times.

Input format:

The first line will contain the value, V and number of types of notes separated by spaces. Next two lines will contain denominations and multiplicities of the notes. For example:

80 9
1 2 5 10 20 50 100 500 1000
100 10 1 0 10 5 10 5 2

Output: The minimum number of notes and number of times each note type is used. For example, for the above input, output should be

4
20*4

If V cannot be made using the notes you have, print “not possible”.

Section A1

A mission-critical production system has n stages that have to be performed sequentially; stage i is performed by machine M_i. Each machine M_i has a probability r_i of functioning reliably and a probability $1 - r_i$ of failing (and the failures are independent). Therefore, if we implement each stage with a single machine, the probability that the whole system works is $r_1 \cdot r_2 \cdots r_n$. To improve this probability we add redundancy, by having m_i copies of the machine M_i that performs stage i. The probability that all m_i copies fail simultaneously is only $(1 - r_i)^{m_i}$, so the probability that stage i is completed correctly is $1 - (1 - r_i)^{m_i}$ and the probability that the whole system works is $\prod_{i=1}^{n}(1 - (1 - r_i)^{m_i})$. Each machine M_i has a cost c_i, and there is a total budget B to buy machines. (Assume that B and c_i are positive integers.)

Given the probabilities $r_1, \ldots, r_n$, the costs $c_1, \ldots, c_n$, and the budget B, find the redundancies $m_1, \ldots, m_n$ that are within the available budget and that maximize the probability that the system works correctly.

Sample Input 1:

Success probabilities: 0.7, 0.8, 0.9
Costs: 100, 100, 200
Budget: 800

Sample Output 1:

Maximum probability: 0.868694
Stage 1 redundancy: 3
Stage 1 redundancy: 3
Stage 1 redundancy: 1

Sample Input 2:

Success probabilities: 0.9, 0.8, 0.7
Costs: 100, 100, 200
Budget: 800

Sample Output 2:

Maximum probability: 0.864864
Stage 1 redundancy: 2
Stage 1 redundancy: 2
Stage 1 redundancy: 2

Input format: The first line of the input file will contain the budget and number of stages separated by spaces. The next lines will contain the probabilities and the costs of stages separated by spaces. For example:

1000 3
.2 .8 .95
200 100 500

Output: The multiplicities of the stages and the maximum probability. For example, output for the above input should be 2, 1, 1 and 0.2736.

Section A2

Matrix chain multiplication: (Cormen *et al.* 15.2) You are given a sequence of matrices $A_1, A_2, \ldots, A_n$. Find the order in which the matrices should be multiplied to minimize the total number of multiplications as well as the total number multiplications needed. Your algorithm should run in $O(n^3)$ time.

For example for the sample input,

```
A 40 20
B 20 30
C 30 10
D 10 30
```

your output should be

```
26000
((A (B C)) D)
```

Input format: The first line will contain the number of matrices, *n* followed by *n* lines each containing dimension of a matrix. For example

4
40 20
20 30
30 10
10 30

Output: Total number of multiplications needed and the order in which matrices are to be multiplied. You need to check whether the matrices can actually be multiplied.