Programming Design, Spring 2015
Homework 3

Instructor: Ling-Chieh Kung
Department of Information Management
National Taiwan University

To submit your work, please upload a PDF file for Problems 1 and 2 and two CPP files for Problems 3 and 4 (optional) to PDOGS at http://pdogs.ntu.im/judge/. Each student must submit her/his individual work. No hard copy. No late submission. The due time of this homework is ***8:00am, March 23, 2014***. Please answer in either English or Chinese.

Before you start, please read Sections 6.1–6.4 of the textbook.[1] Sections 6.7 and 6.8 are also helpful. If you are wondering what are "functions" in C++, you may skim through the first few pages of Chapter 5, which will formally introduced later in this semester.

Problem 1

(10 points) Use your own words to describe a good timing for one to use constant variables. Explain the reasons.

Problem 2

(20 points) Consider the following problems regarding algorithms and pseudocodes.

1. (5 points) Modify the pseudocode on Page 28 of the slides for March 16 to make all the indices correct.
2. (5 points) Write a pseudocode for the implementation on Page 29 of the slides for March 16.
3. (10 points) Write a pseudocode that completes the following task: Given an array of n integers (may be positive or negative), find the length of the *longest positive sequence*. For example, if the given array contains $(9, 4, -3, 0, 4, 2, 8, 3, -2, 3)$, the longest positive sequence is $(4, 2, 8, 3)$ and thus the output should be 4, the length of this sequence.

Problem 3

(70 points) You are given n items. The weight and value of item i is w_i kg and v_i dollars, respectively. Given a knapsack (i.e., a backpack) that can carry at most B kg, which items should you choose to maximize the total value without breaking the knapsack? Please note that none of these items is divisible!

The above problem is the well-known *knapsack* problem in Computer Science. Mathematically, we may formulate the problem as follows. Let x_i be 1 if we choose an item or 0 otherwise. Then the knapsack problem is

$$\begin{aligned} \max \quad & \sum_{i=1}^{n} v_i x_i \\ \text{s.t.} \quad & \sum_{i=1}^{n} w_i x_i \leq B \\ & x_i \in \{0,1\} \quad \forall i = 1, ..., n. \end{aligned}$$

[1] The textbook is *C++ How to Program: Late Objects Version* by Deitel and Deitel, seventh edition.

As an example, suppose you have 4 items of weights 2, 3, 4, 3 and values 2, 4, 5, 3, respectively, and a backpack of capacity 9, then the best thing to do is to choose items 1, 2, and 3. In this way, the total weight $2 + 3 + 4 = 9$ is okay for the knapsack and the total value $2 + 4 + 5 = 11$ is the highest that we may attain. This is called an *optimal solution* for this problem.

In general, solving a knapsack problem with many items is hard.[2] In this problem, we will only ask you to compare several candidate solutions and pick a best one. Consider the above example again. In this problem, we will not ask you to find an optimal solution; instead, what we may ask is "among $(x_1, x_2, x_3, x_4) = (1, 1, 0, 1)$, $(1, 0, 1, 1)$, and $(0, 1, 1, 1)$, which one is optimal?" Then you report that $(1, 0, 1, 1)$ is the best among the three candidates. Note that $(0, 1, 1, 1)$ is *infeasible* because the total weight is larger than the knapsack capacity. An infeasible solution of course cannot be optimal. Your task is to identify a feasible solution which gives you the largest value. It is known that among all candidate solutions, there is at least one that is feasible.

Input/output formats

There are 15 input files. In each file, there are $k+4$ lines of values, where $k \in \{3, 4, ..., 100\}$ is the number of candidate solutions for you to evaluate. Below is the description of an input file:

- The first line contains a positive integer n, a white space, a positive integer B, and a newline character. $n \in \{1, 2, ..., 1000\}$ is the number of items and $B \in \{1, 2, ..., 10000\}$ is the knapsack value.
- The second line contains n positive integers w_1, w_2, ..., and w_n, followed by a newline character. Each two values are separated by a white space. $w_i \in \{1, ..., 100\}$ is the weight of item i.
- The third line contains n positive integers v_1, v_2, ..., and v_n, followed by a newline character. Each two values are separated by a white space. $v_i \in \{1, ..., 100\}$ is the value of item i.
- The fourth line contains a positive integer k and a newline character. k is the number of candidate solutions for you to evaluate.
- Each of the last k lines contains a positive integer $m \in \{1, 2, ..., k\}$, a sequence of n binary values $x_i \in \{0, 1\}$, $i = 1, ..., n$, followed by a newline character. Each two values are separated by a white space. m is the index of the candidate solution and x_i is 1 if item i is selected and 0 otherwise. Candidates are labeled in the order of 1, 2, 3, ..., and k. In other word, m is increasing from the fifth line to the last line.

For the above example, the input file will be

```
4 9
2 3 4 3
2 4 5 3
3
1 1 1 0 1
2 1 0 1 1
3 0 1 1 1
```

Given the input file, you output three positive integers in a single line in your output file: the index of your reported best solution and the total weight and value of the selected items in the solution. Each two values should be separated by a white space. For the above example, the output file should contain

```
2 9 10
```

Suppose more than one candidate solutions are optimal, report ONLY the one with the SMALLEST index.

[2] In Computer Science, we say that the knapsack problem is *NP-hard*, which means most researchers in the world believe that there is no "efficient" method for finding an optimal solution.

Grading criteria

- 45 points for this program will be based on the correctness of your output. PDOGS will compile your program, feed testing data into your program, and check the correctness of your outputs. Each correct output gives you 3 points.
- 5 points for this program depends on whether you declare your arrays *dynamically*. Pay attention to the syntax provided on Page 32 of the slides for March 16. Declare your arrays dynamically by using `int*` and `new`. If you fail to do this, you lose the 10 points.
- 20 points for this program will be based on how you write your program, including the logic and format. Please try to write a robust, efficient, and easy-to-read program.

For this problem, you are NOT allowed to use techniques not covered in lectures. You should write relevant comments for your codes.

Bonus: Problem 4

(20 points) You are given an $n \times n$ integer matrix A. Your task is to determine whether there is any $m \times m$ square submatrix B such that

$$\sum_{i=1}^{m}\sum_{j=1}^{m} B_{ij} = 0$$

for some $m \leq n$. We call this kind of sqaure submatrix a *zero-sum* matrix. You need to find the number of zero-sum square submatrices. For example, given

$$A = \begin{bmatrix} 1 & 2 & -1 \\ -1 & -1 & 1 \\ 0 & 1 & -1 \end{bmatrix},$$

we can see that there is one 1×1 zero-sum square submatrix $\begin{bmatrix} 0 \end{bmatrix}$, one 2×2 zero-sum square submatrix $\begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix}$, and no 3×3 zero-sum square submatrix. The total number of zero-sum square submatrice is thus 2. In this problem, you are given that $n = 8$.[3]

Input/output formats

There are 10 input files. In each file, there is an 8×8 integer square matrix A contained in 8 lines. In line i, we have the values $A_{i,1}$, $A_{i,2}$, ..., and $A_{i,8}$. Each of two values are separated by a white space. Each line is ended with a newline character.

Given the input file, you output one nonnegative integers in a single line in your output file: the total number of zero-sum square submatrices.

Grading criteria

All points for this program will be based on the correctness of your output. PDOGS will compile your program, feed testing data into your program, and check the correctness of your outputs. Each correct output gives you 2 points. For this problem, you are allowed to use any technique. Though suggested, comments are not required.

[3]With this restriction, you may declare a static two-dimensional array by just the syntax provided in the slides for March 16. Read the section about multi-dimensional arrays by yourselves, if you want to work on this bonus problem. That section will be covered on March 23.