Programming Design, Spring 2016
Homework 5

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

Please upload one PDF file for Problem 1 and two CPP files for Problems 2 and 3 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 ***2:00 am, March 28, 2016***. Please answer in either English or Chinese.

Before you start, please read Sections 5.1–5.7 and 6.5-6.8 of the textbook.[1]

The TA who generates the testing data and grades this homework is ***Wei-Hung Liao***.

Problem 1

(20 points; 5 points each) Consider the following program:

```
#include<iostream>
using namespace std;

const int ATTRIB_CNT = 3;

int f(int [][ATTRIB_CNT], int, int []);

int main()
{
  int person[5][ATTRIB_CNT] = {1, 7, 4, 9, 7, 6, 4, 1,
                               5, 4, 4, 4, 8, 6, 2};
  int max[1] = {0};

  cout << f(person, 5, max) << " ";
  cout << max[0] << "\n";
  cout << f(person + 2, 3, max) << " ";
  cout << max[0] << "\n";

  return 0;
}

int f(int person[][ATTRIB_CNT], int n, int max[])
{
  int maxIndex = 0;
  max[0] = 0;

  for(int i = 0; i < n; i++)
  {
    int sum = 0;
    for(int j = 0; j < ATTRIB_CNT; j++)
      sum += person[i][j];

    if(sum > max[0])
    {
```

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

```
        maxIndex = i;
        max[0] = sum;
      }
    }
    return maxIndex;
}
```

(a) Which variable is a global variable? Is it a good idea to use this global variable? Explain why.

(b) What is the physical meaning of the return value of the function `f`? Please explain its meaning based on `person`, the input matrix.

(c) Explain what the last parameter `max` does. Note that it is an array of length 1, which may only store one element. Why not declare it as a single integer?

(d) What are the outputs of this program? Explain why the outputs are like that. In particular, explain what happens when we execute `f(person + 2, 3, max)`.

Problem 2

(30 points) Please redo Problem 3 in Homework 4 with functions. Use functions to make your program easier to read. The TAs will compare your programs to see whether you modified your own program submitted for Homework 4. Your new program will be graded based on the logic and readability.

Problem 3

(50 points) n jobs are to be scheduled on one machine to minimize the *makespan*, i.e., the completion time of the last job. There are two kinds of works that require machine time. First, the machine needs to spend a *processing time* p_j to complete job j. Second, if job j is scheduled right before job k, there is a *setup time* s_{jk} that needs to be spent before the machine can start to process job k. This makes the order of processing jobs matters in minimizing the makespan. We say this kind of scheduling problems are *sequencing* problems.

As an example, suppose there are $n = 4$ jobs, jobs 1, 2, 3, and 4. We may represent their processing times and setup times as a vector p and a matrix S:

$$p = \begin{bmatrix} 3 \\ 4 \\ 5 \\ 6 \end{bmatrix} \quad \text{and} \quad S = \begin{bmatrix} 0 & 0 & 3 & 2 \\ 2 & 0 & 1 & 2 \\ 3 & 0 & 0 & 4 \\ 2 & 0 & 2 & 0 \end{bmatrix}.$$

This means that $p_1 = 3$, $s_{23} = 1$, etc. A schedule can be represented by an order of job indices. For example, $x = (1, 4, 2, 3)$ means to process these jobs in the order of job 1 first, then job 4, then job 2, and lastly job 3. If we do so, the makespan is

$$3 + \boxed{2} + 6 + \boxed{0} + 4 + \boxed{1} + 5 = 21,$$

where the three numbers in boxes are setup times and the four numbers not in boxes are processing times. Obviously, we have no idea whether this is an optimal schedule.

While finding an optimal schedule is hard, in this problem we will only ask you to implement a specific method. We first define the concept *neighbor* for this problem. Two schedules x and y are neighbors if there exists two indices i and j such that $x_i = y_j$, $x_j = y_i$, and $x_k = y_k$ for all $k \neq i, k \neq j$. For example, $x = (1, 2, 3, 4)$ and $y = (1, 3, 2, 4)$ are neighbors, but x and $z = (1, 4, 2, 3)$ are not. In other words, two schedules are neighbors if one can become the other by switching two jobs and vice versa. Between a

pair of neighbors, we also define their *sum of switching indices* as $x_i + x_j$, i.e., the sum of indices of the two switched jobs.

Now, given n jobs, we will always start from the schedule $x^0 = (1, 2, ..., n)$. Then in iteration i, $i = 1, 2, ...$, we check whether there is any *neighbor* of x^{i-1} that has a strictly smaller makespan. If yes, we switch to that neighbor by setting x^i to be that neighbor. If no, we say we have reached a *local optimum*. We then stop and then report this schedule as our proposed schedule. If multiple neighbors all have the same minimum makespan, we move to the one whose sum of switching indices is the smallest. If there is still a tie, we move to the one of which the smaller index of the switched jobs is the smallest. For example, suppose we are currently at $(2, 3, 4, 1)$. If $(4, 3, 2, 1)$ and $(2, 4, 3, 1)$ are equally good, we pick $(4, 3, 2, 1)$ because $2 + 4 < 3 + 4$. If $(3, 2, 4, 1)$ and $(2, 3, 1, 4)$ are equally good, we pick $(2, 3, 1, 4)$ because $2 + 3 = 1 + 4$ but $1 < 2$.

In this problem, you will be given the information of n, p, and S. Your program should print out the schedule by following the method defined above.

Input/output formats

There are 20 input files. In each file, there are $n + 2$ lines of nonnegative integers. The first line contains n, the number of jobs. The second line contains n integers p_1, p_2, ..., and p_n, the processing times of the n jobs. The last n lines contains an $n \times n$ matrix. In particular, the $(i + 2)$th line contains $s_{i,1}$, $s_{i,2}$, ..., and $s_{i,n}$. For example, the following input

```
4
3 4 5 6
0 0 3 2
2 0 1 2
3 0 0 4
2 0 2 0
```

describes the example shown above. Two consecutive integers are separated by a white space. It is assumed that $n \leq 100$, $p_j \leq 100$, and $s_{ij} \leq 50$.

Given this input, your program should follow the method defined in this problem to propose a schedule. The jobs in your proposed schedule should be printed in the processing order, where two consecutive job indices are separated by a white space. After the sequence of job indices, print out a white space and then the makespan of your schedule. For the example above, suppose that you want to propose $(1, 4, 2, 3)$, your output should be

```
1 4 2 3 21
```

What should be in your source file

Your .cpp source file should contain C++ codes that will both read testing data and complete the above task. For this problem, you are allowed to use only techniques covered so far. NO other techniques are allowed. Finally, you should write relevant comments for your codes.

Grading criteria

- 40 points 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 fully correct set of outputs gives you 2 points.
- 10 points 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.