Programming Design, Spring 2016
Homework 8

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

Please upload one PDF file for Problem 1 and three CPP files for Problems 2, 3, and 4 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, April 18, 2016***. Please answer in either English or Chinese.

Before you start, please read Sections 7.1–7.9 and 7.11 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) Answer the following questions with respect to the following function:

```
int print(int** arr, int r)
{
  for(int i = 0; i < r; i++)
  {
    for(int j = 0; j <= i; j++)
      cout << arr[i][j] << " ";
    cout << "\n";
  }
}
```

(a) Determine whether the following function is equivalent to the given function `print`. Explain why.

```
int print(int** arr, int r)
{
  for(int i = 0; i < r; i++)
  {
    for(int j = 0; j <= i; j++)
      cout << *(*(arr + i) + j) << " ";
    cout << "\n";
  }
}
```

(b) Determine whether the following function is equivalent to the given function `print`. Explain why.

```
int print(int** arr, int r)
{
  for(int i = 0; i < r; i++)
  {
    int* p = *(arr + i);
    for(int j = 0; j <= i; j++)
      cout << *p << " ";
    p++;
    cout << "\n";
  }
}
```

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

(c) Add appropriate `delete` statements to the following program to release dynamically allocated space.

```
int main()
{
  int r = 3;
  int** array = new int*[r];
  for(int i = 0; i < r; i++)
  {
    array[i] = new int[i + 1];
    for(int j = 0; j <= i; j++)
      array[i][j] = j + 1;
  }
  print(array, r);
  return 0;
}
```

(d) Explain why there is a compilation error at the function invocation:

```
int main()
{
  int array[3][3] = {0};
  print(array, r);
  return 0;
}
```

Problem 2

(20 points) A police station is open seven days a week. The numbers of police officers needed each day are different. Each police officer works for five consecutive days and then have two days off. Because one may start to work on any day, there are seven different types of work schedules (Monday to Friday, Tuesday to Saturday, ..., and Sunday to Thursday). How to minimize the total number of police officers needed to fulfill the demands of all days?

As an example, suppose the demands are given in Table 1, where day 1 means Monday, day 2 means Tuesday, ..., and day 7 means Sunday. One feasible schedule is the following: 18 officers starts to work on day 1, 6 on day 3, 3 on day 5, and 7 on day 6. Let x_i be the number of officers starting to work on day i, $i = 1, ..., 7$, this schedule can be expressed as $x = (18, 0, 6, 0, 3, 7, 0)$. In total 34 officers are required. As we may see in Table 2, all the demands are satisfied, though on some days there are some excess supply.

Day	1	2	3	4	5	6	7
Demand	18	12	24	19	27	16	14

Table 1: An example demand distribution

To minimize the total number of police officers needed, we should try to cut down excess supply. It can be verified that another schedule $x = (10, 0, 14, 0, 8, 0, 0)$ is better, as it can satisfy all demands with only 32 officers. However, it is not so clear whether this is an optimal schedule, i.e., a feasible schedule that uses the minimum number of officers.

While a polynomial-time algorithm exists for finding an optimal schedule, in this problem you are only required to implement a given algorithm. The algorithm runs in the following way. Let D_i^1 be the demand of day i, $i = 1, ..., 7$. In iteration j, $j = 1, 2, ...$, we schedule D_i^j officers to start working on day

Day	1	2	3	4	5	6	7
Demand	18	12	24	19	27	16	14
Supply 1	18	18	18	18	18	0	0
Supply 2	0	0	6	6	6	6	6
Supply 3	3	3	0	0	3	3	3
Supply 4	7	7	7	0	0	7	7
Total supply	28	28	31	24	27	16	16

Table 2: A feasible schedule

i, where i is the smallest index such that day i still needs officers. We then update D^j to D^{j+1} to reflect the impact of adding D_i^j officers into the schedule. We keep doing so until all demands are fulfilled.

As an example, consider $D^0 = (18, 12, 24, 19, 27, 16, 14)$ as the initial demand distribution in Table 1. This requires us to set $x_1 = D_1^0 = 18$, as day 1 is currently the first day that still needs officers. We then update D^0 to $D^1 = (0, 0, 6, 1, 9, 16, 14)$ to represent the unfulfilled demands. This requires us to set $x_3 = D_3^1 = 6$. We then have $D^2 = (0, 0, 0, 0, 3, 10, 8)$. By setting $x_5 = 3$, we have $D^3 = (0, 0, 0, 0, 0, 7, 5)$. After we set $x_6 = 7$, we get $D^4 = (0, 0, 0, 0, 0, 0, 0)$ and may stop. The answer is $x = (18, 0, 6, 0, 3, 7, 0)$.

Input/output formats

There are 10 input files. In each file, there are seven integers D_1, D_2, ..., and D_7. Two consecutive integers are separated by a white space. It is known that $0 \leq D_i \leq 10000$. Given this input, your program should run the given algorithm to find a feasible schedule x. The schedule should be printed out as seven integers, x_1, x_2, ..., and x_7. Each two consecutive integers should be separated by a white space.

For example, for the input

```
18 12 24 19 27 16 14
```

the output should be

```
18 0 6 0 3 7 0
```

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

20 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.

Problem 3

(30 points) Continue from Problem 2. Note that the algorithm always starts on day 1, which may result in a bad schedule. A simple idea to improve the algorithm is to run it for seven times, one to start from

a different day. For the above example, if we start from day 2, we will have:

$$\begin{aligned} D^1 &= (18, 12, 24, 19, 27, 16, 14) \quad \rightarrow \quad x_2 = 12 \\ D^2 &= (18, 0, 12, 7, 15, 4, 14) \quad \rightarrow \quad x_3 = 12 \\ D^3 &= (18, 0, 0, 0, 3, 0, 2) \quad \rightarrow \quad x_5 = 3 \\ D^3 &= (15, 0, 0, 0, 0, 0, 0) \quad \rightarrow \quad x_1 = 15. \end{aligned}$$

The schedule is $(15, 12, 12, 0, 3, 0, 0)$, which requires 42 officers in total. This shows that the starting day does have an impact on the schedule quality, and trying other starting days may help find a better schedule. Of course, we have no idea whether this modified algorithm will give us an optimal schedule or not, but at least it is better than the original one (though the running time is seven times longer). Implement this algorithm and report the number of officers required by the seven schedules generated by the seven starting days.

Input/output formats

There are 10 input files. The input format is the same as that in Problem 2. Given this input, your program should run the given algorithm to find a starting day i that results in the best schedule x (among the seven schedules generated by the seven starting days). If multiple starting days result in the same best schedule, we call the one with the smallest index the best starting day. Your program should print out the the number of officers required by the seven schedules generated by the seven starting days, first for starting on day 1, second for starting day 2, ..., and last for starting on day 7. Each two consecutive integers should be separated by a white space.

For example, for the input

```
18 12 24 19 27 16 14
```

the output should be

```
34 42 42 51 51 43 41
```

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

- 20 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.

Problem 4

(40 points) Recall that a graph may be represented by an adjacency matrix A, where $A_{ij} = 1$ if there is an edge connecting nodes i and j and 0 otherwise. For the graph in 1, the adjacency matrix is

$$A = \begin{bmatrix} 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}.$$

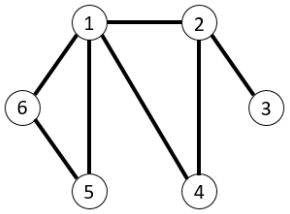

Figure 1: A graph

While adjacency matrices are intuitive and easy to understand, it may be inefficient in some cases. In particular, using an adjacency matrix to store an n-node graph requires n^2 bytes (if each entry of the matrix is stored by a Boolean variable), which can be wasting a lot of spaces if the number of edges is much smaller than n. For example, the graph in Figure 2 has 53 node and only 56 edges. An adjacency matrix will have $53^2 = 1809$ entries, in which only $56 \times 2 = 112$ entries are 1. This is an example of a *sparse matrix*, and an adjacency matrix may be wasting too many spaces.

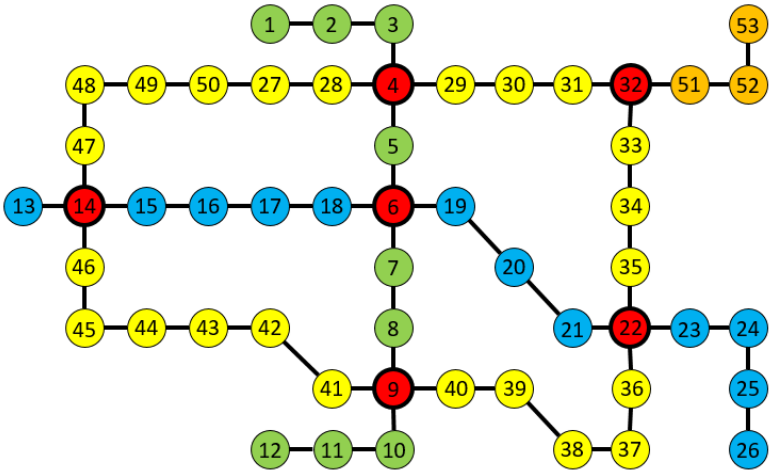

Figure 2: A subway map

An alternative data structure is an *adjacency list*.[2] The idea is still simple: We should have n single-dimensional arrays whose lengths are different, where array i records the nodes that is adjacent to node i. For the graph in Figure 1, array 1 will be $\{2, 4, 5, 6\}$, array 2 will be $\{1, 3, 4\}$, etc. This obviously saves spaces when the number of edges is much smaller than n^2. To implement these variable-length arrays in C++, we may use dynamic arrays in the following way. First, we should have an array of n pointers. Pointer i points to a dynamic array of d_i integers, where d_i is the number of nodes adjacent to node i.

[2]Some people define an adjacency list by using linked list. In my opinion, it is not needed. Here I am just using the idea, not any specific way of implementation.

In this problem, you need to build an adjacency list to store a given graph. Then you will be given a path $(v_1, v_2, ..., v_k)$. You need to find the number of edges that need to be added into the graph to make the path exist. For the graph in Figure 1, if the given path is $(1, 2, 3, 4, 5)$, the number of missing edges is 2 (from 3 to 4 and from 4 to 5). For $(5, 1, 2, 4)$, it is 0.

Input/output formats

There are 15 input files. In each file, there are $m + 2$ lines of integers. The first lines contains n and m, where n is the number of nodes and m is the number of edges. Nodes are labeled as 1, 2, ..., and n. Each of the next m lines contains two integers u and v, two node IDs. This represents an edge between u and v. The last line contains a path $(v_1, v_2, ..., v_k)$ (which may not really exist in the graph). Any two consecutive integers in one line are separated by a white space. It is known that $n \leq 10000$, $m \leq 30000$, no two given edges are the same, $k \leq n$, $v_i \neq v_j$ if $i \neq j$, and all the information contained in the input file is consistent and valid. For example, the following input

```
6 7
2 3
1 2
5 1
1 6
1 4
2 4
6 5
1 2 3 5 4
```

represents the graph in Figure 1.

Given the input, your program should construct an adjacency list to store the graph. It should then identifies missing edges in the given path. These edges should be printed according to the order in the path. Each missing edge should be printed as two integers in one line, where each integer is a node ID, and the smaller node ID should be printed first. For the input above, the output should be

```
3 5
4 5
```

which means that, along the path, first the edge between 3 and 4 and then that between 4 and 5 are missing. Note that the second line should be `4 5`, not `5 4`; the smaller ID should be printed first.

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 any technique. Moreover, you MUST implement an adjacency list. One who fails to do so will get no point for this problem. Finally, you should write relevant comments for your codes.

Grading criteria

If you do not implement an adjacency list, you will get 0 point. If you do, you will be graded in the following way:

- 30 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.