Programming Design, Spring 2014
Homework 5

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

Submission. To submit your work, please upload the following two files to the online grading system at http://lckung.im.ntu.edu.tw/PD/.

1. A .pdf for Problems 1 to 3.
2. Your .cpp file for Problems 4 and 5. Note that only one .cpp file should be submitted.

Each student must submit her/his individual work. No hard copy. No late submission. The due time of this homework is ***8:00am, March 24, 2014***.

Problem 0

(0 point) Please read Sections 5.8–5.18 of the textbook.[1] In any case, I strongly suggest you to read the textbook thoroughly before you start to do this homework.

Problem 1

(10 points; 5 points each) Consider the following three files:

```
// main.cpp
#include <iostream>
#include "myHeader.h"
using namespace std;

int main ()
{
  int score[MAX_S_COUNT] = {0};
  inputScore (score);
  cout << maxScore (score);
  return 0;
}
```

```
// myHeader.h
const int MAX_S_COUNT = 1000;
int maxScore (int []);
void inputScore (int []);
```

```
// mySource.cpp
#include <iostream>
using namespace std;

int maxScore (int a[])
{
  int max = a[0];
  for (int i = 1; i < MAX_S_COUNT; i++)
  {
    if (a[i] > max)
      max = a[i];
  }
  return max;
}
void inputScore (int a[])
{
  for (int i = 0; i < MAX_S_COUNT; i++)
  {
    cin >> a[i];
    if (a[i] < 0)
      break;
  }
}
```

Suppose we put in the same folder and then compile them together.

(a) Explain why there will be a compilation error and how to fix it.

(b) Suppose it is typical to have a class whose size is much smaller than 1000, how to modify the three files to make them more efficient?

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

Problem 2

(10 points; 5 points each) Consider the following random number generating function:

```
void pseudorandom (int seed, int num, int a, int b, int c)
{
  for (int i = 0; i < num; i++)
  {
    seed = (seed * a + b) % c;
    cout << seed << " ";
  }
  cout << endl;
}
```

(a) Suppose we call this function by executing `pseudorandom (0, 10, 59, 37, 63);`, what will be the output? Do you think the last three arguments result in a good random number generator? Why or why not?

(b) Modify the function to give it the sixth parameter `rn`, an integer array whose length is `num`, so that after an execution those generated random numbers are stored in `rn`.

Problem 3

(10 points; 5 points each) Suppose you are given two points $s = (s_1, s_2) \in \mathbb{Z}^2$ and $t = (t_1, t_2) \in \mathbb{Z}^2$, where s is a starting point and t is a destination point. You are also given a line segment

$$[u, v] = \Big[(u_1, u_2), (v_1, v_2)\Big] := \Big\{(x_1, x_2) \in \mathbb{R}^2 \Big| x_i = u_i + k(v_i - u_i), k \in [0, 1], i = 1, 2\Big\}, \tag{1}$$

where $u = (u_1, u_2) \in \mathbb{Z}^2$, $v = (v_1, v_2) \in \mathbb{Z}^2$, and either $u_1 = u_2$ or $v_1 = v_2$. In other words, the line segment has integer points as its endpoints and is in either the north-south or east-west direction. The question we want to answer is: Is it possible to travel from s to t in at most one north-south move and at most another one east-west move without touching the line segment $[u, v]$?

As we are allowed to turn at most once, we now from s to t there are at most two possible routes that travel in north-south and east west directions. For example, if $s = (2, 3)$ and $t = (9, 6)$, then one possible route is composed by two line segments $[(2, 3), (9, 3)]$ and $[(9, 3), (9, 6)]$ and the other one is composed by $[(2, 3), (2, 6)]$ and $[(2, 6), (9, 6)]$ (cf. Figure 1). The question we want to answer is simply whether the given line segment touches both routes. For example, suppose the line segment is $[(5, 1), (5, 5)]$, the thick dashed line in Figure 2, then it is clear that only route 2 is available. Nevertheless, even if one route is blocked, as long as there is one route, the answer is "yes, we may travel from s to t in north-south and east-west directions with at most one turns and without touching the line segment." If the line segment is $[(2, 4), (10, 4)]$, however, both routes are blocked and the answer will become no.

For this problem, there are certainly multiple algorithms that may solve it. One interpretation of this problem is to treat the two possible routes as four line segments. An algorithm is then to divide this problem into four subproblems, in each we determine whether two north-south or east-west line segments intersect. In this case, we say that our original problem *reduces to* the problem of determining whether two north-south or east-west line segments intersect.

(a) Given two line segments $[a, b]$ and $[u, v]$ that are both in the north-south direction, describe an algorithm in pseudocodes that determines whether they intersect.[2]

(b) Given a north-south line segment $[a, b]$ and a east-west line segment $[u, v]$, describe an algorithm in pseudocodes that determines whether they intersect.[3]

[2] For $[a, b]$ to be north-south, it is clear that $a_1 = b_1$. Similarly, we have $u_1 = v_1$ for $[u, v]$.
[3] We have $a_1 = b_1$ and $u_2 = v_2$.

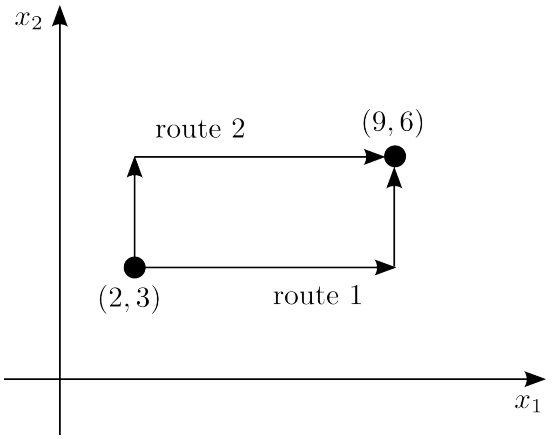

Figure 1: Two possible routes

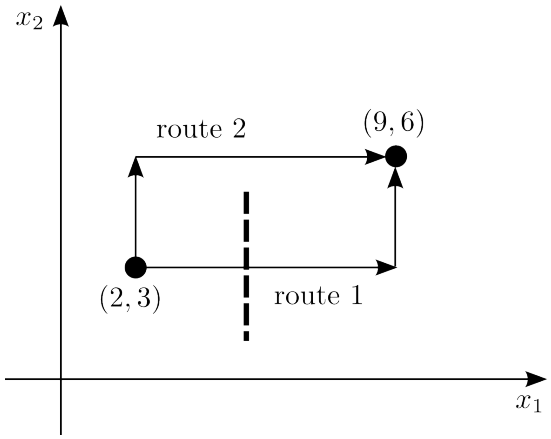

Figure 2: One route is blocked

Problem 4

(70 points) In IM city, all roads are built in the north-south or east-west directions. As a taxi driver, you need to drive on roads (of course) and thus you may also travel either north-south or east-west. In each day, some roads may be closed for maintenance or special events. Because turning takes more times then going straight, You are interested in knowing that, given any destination requested by a potential customers, whether it is possible to travel there with at most one turn.

You formulate the above problem abstractly as follows. Given a set of potential destinations T in a two-dimensional set

$$S = [-100, 100]^2 \cap \mathbb{Z}^2.$$

you ask whether you may travel from the origin to each destination with at most one turn. Certainly that depends on the given set of closed roads

$$B = \left\{[u^i, v^i]\right\}_{i=1,...,m},$$

where $[u^i, v^i]$ is the ith north-south or east-west line segment defined in (1). A target $t = (t_1, t_2) \in T$ can be traveled to from the origin if and only if

$$\left(\Big[(0,0),(0,t_2)\Big] \cup \Big[(0,t_2),(t_1,t_2)\Big]\right) \cap [u^i, v^i] = \emptyset \quad \forall i = 1, ..., m$$

or

$$\left(\Big[(0,0),(t_1,0)\Big] \cup \Big[(t_1,0),(t_1,t_2)\Big]\right) \cap [u^i, v^i] = \emptyset \quad \forall i = 1, ..., m.$$

When the first condition is true, you may first drive north-south and then east-west to get to t; when the second condition is true, you may first drive east-west and then north-south to get to t; if both conditions are false, you cannot get to t by turning at most once. Your question is: Among all destinations in T, which of them can be reached with at most one turn?

Input/output formats

The input will contain 35 lines, each with a sequence of numbers

$$(n, m, t_1^1, t_2^1, ..., t_1^n, t_2^n, u_1^1, u_2^1, v_1^1, v_2^1, ...u_1^m, u_2^m, v_1^m, v_2^m).$$

The first two numbers, n and m, are the numbers of destinations and closed roads, respectively. The next $2n$ numbers are the coordinates of the n destinations. In particular, (t_1^j, t_2^j) is the location of destination j, $j = 1, 2, ..., n$. The last $4m$ numbers are the coordinates of the $2m$ endpoints of the m closed roads. In

particular, (u_1^i, u_2^i) and (v_1^i, v_2^i) are the two endpoints of closed road i, $i = 1, 2, ..., m$. Two consecutive numbers are separated by a white space. All the numbers are integers, $n \leq 100$, $m \leq 100$, and all coordinates are within -100 and 100. Suppose destination t^j can be reached from the origin, j should be output. Your program should output the indices of all the reachable destinations in an ascending order, i.e., output j_1 before j_2 if $j_i < j_2$. Two consecutive outputs should be separated by a white space. If no destinations can be reached, your program should output a new line character directly.

For example, suppose a line contains

```
3 3 4 2 -1 4 -3 3 2 1 2 4 -4 1 -2 1 4 0 4 0
```

as an input line, we know there are three destinations $(4, 2)$, $(-1, 4)$, and $(-3, 3)$ and three closed roads $[(2, 1), (2, 4))$, $[(-4, 1), (-2, 1)]$, and $[(4, 0), (4, 0)]$. Note that it is possible for a closed "road" to be actually a closed "point". If you draw a figure, it is clear that the first destination is not reachable due to the first and third closed roads but the other two destinations can both be reached from the origin. Note that for one destination to be reachable, the existence of one single route is enough. Therefore, your program should output

```
2 3
```

and then a new line character. As another example, the output for the input line

```
3 3 2 2 2 0 -2 0 1 1 -1 1 -1 1 -1 -1 -1 -1 1 -1
```

should be

```
1 2
```

and then a new line character. Please also note that when a point is the endpoint of multiple closed roads, it will be defined in all those closed roads. Finally, if no destination is reachable, a single new line character should be output.

Designing your own algorithm

In this problem (and most problems you will see in the future), you need to design your own algorithm. The first thing you need to do is to create a correct algorithm. Once you have it, you may try to create a more efficient one. For most problems in this semester, we do not require you to design a fast algorithm or write a fast implementation. Typically only correctness is required. For this problem, in particular, we limit all points to be integer points within $[-100, 100]^2$ and the total number of destinations and closed roads to be no greater than 200. For this scale, most straightforward algorithms and implementations should be fast enough. One algorithm is provided below.[4] After the execution of this algorithm, the jth element of the vector R records whether destination t^j is reachable: It is reachable if R_j is true or not reachable otherwise.

[4]To implement the pseudocode as a C++ function, you may want to declare the return type as `void` and define R as a matrix parameter and "return R" by modifying the values of R in your function.

Algorithm findReachableDestinations (n, m, T, B)

create a Boolean vector R whose length is n
set R_j to ***false*** for all $j = 1, 2, ..., n$
for each destination $t^j \in T$
  set $isReachable_1$ and $isReachable_2$ to ***true***
  for each closed road $[x, y] \in B$
    if $[(0,0),(t_1^j,0)] \cap [x,y] \neq \emptyset$ or $[(t_1^j,0),(t_1^j,t_2^j)] \cap [x,y] \neq \emptyset$
      set $isReachable_1$ to ***false***
    if $[(0,0),(0,t_2^j)] \cap [x,y] \neq \emptyset$ or $[(0,t_2^j),(t_1^j,t_2^j)] \cap [x,y] \neq \emptyset$
      set $isReachable_2$ to ***false***
  if $isReachable_1$ or $isReachable_2$ is ***true***
    set R_j to ***true***

Obviously, you need a way to determine whether two given north-south or east-west line segments intersect. The algorithm your designed for Problem 3 is an alternative. In any case, implementing the algorithm as a function returning a Boolean value is suggested.

It is possible to enhance the efficiency of the above algorithm or even design a completely new one that is better. It is your discretion in choosing an algorithm.

What should be in your source file

For this problem, your .cpp source file should contain C++ codes that will both read testing data and complete the above task. You are welcome to use any technique you know. Finally, you should write relevant comments for your codes.

Grading criteria

- 70% of your grades 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 fully correct line of output gives you 2 points.
- 30% of your grades 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.

(Bonus) Problem 5

(20 points) Your program written for Problem 4 will be manually executed by the TAs for ten more lines of input. In each of these input lines, n and m can be at most 10000 and each coordinate is within -10000 and 10000. If your program can get the correct answer or a line of input, you get 1 bonus point. If your program get all correct answers and and run faster than the one written by the TAs (as the one that will be posted as the suggested solution), you will get 10 additional bonus points.