CST 370 – Spring (B) 2022
Homework 6
Due: 4/19/2022 (Tuesday) (11:55 PM)

How to turn in: Write **two programs** in **either C++ or Java** and submit them on Canvas before the due.

- Note that **each problem** will have **8 test cases**, and **one test case** will have the value of **1.5 points**, not 1 point. So, the full points of homework 6 will be 30 points as other homework assignments. For detail of grading, refer to the **"rubric"** on Canvas.
- You **can submit** your programs **multiple times** before the due. However, the **last submission will be used for grading**.
- You have to **submit two programs together**, especially at your last submission. **If you submit, for example, only one program** at the last submission, **we are able to see only that program when we grade** your homework.
- Due time is 11:55(PM). Since there could be a long delay between your computer and Canvas, you should submit it early.
- When you submit your homework program, don't forget to include "Title", "Abstract", "ID", "Name", and "Date".

1. Write a C++ (or Java) program called **hw6_1.cpp (**or **hw6_1.java)** to collect maximum number of coins on an *n* x *m* board which was covered in the class.

Input format: This is a sample input from a user.

```
5 6
0 0 0 0 1 0
0 1 0 1 0 0
0 0 0 1 0 1
0 0 1 0 0 1
1 0 0 0 1 0
```

The first line (= 5 and 6 in the example) indicates that the board has 5 rows and 6 columns. From the second line, the configuration of the board is presented. The number 1 indicates that there is a coin on the cell, while the number 0 means no coin. For the homework, you can assume that the **board size** is **less than or equal to 25 x 25**.

Sample Run 0: Assume that the user typed the following lines

```
5 6
0 0 0 0 1 0
0 1 0 1 0 0
0 0 0 1 0 1
0 0 1 0 0 1
1 0 0 0 1 0
```

This is the correct output. Note that the coordinate of the starting cell is (1,1) and the destination is (5,6). Your program should display maximum coins and path to collect them. When **backtracking** from the destination cell, if there is **more than one optimal path**, your solution should always **pick the path from the left, not from the top**.

```
Max coins:5
Path:(1,1)->(2,1)->(2,2)->(2,3)->(2,4)->(3,4)->(3,5)->(3,6)->(4,6)->(5,6)
```

Sample Run 1: Assume that the user typed the following lines. Again, when **backtracking** from the destination spot, your solution should always **pick the path from the left, not from the top**, if there is **more than one optimal path**.

```
4 5
0 0 0 1 0
0 1 1 1 0
1 1 1 0 1
0 0 0 0 0
```

This is the correct output.

```
Max coins:4
Path:(1,1)->(2,1)->(3,1)->(3,2)->(3,3)->(3,4)->(3,5)->(4,5)
```

Sample Run 2: Assume that the user typed the following lines

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

This is the correct output.

```
Max coins:3
Path:(1,1)->(1,2)->(2,2)->(3,2)
```

2. Write a C++ (or Java) program called **hw6_2.cpp (or hw6_2.java)** that implements the **Floyd algorithm** to display all-pairs shortest paths as we covered in the class.

Input format: This is a sample input from a user.

```
4
0 -1 3 -1
2 0 -1 -1
-1 7 0 1
6 -1 -1 0
```

The first line (= 4 in the example) indicates that there are four vertices in the input graph. Then the following 4 lines present the distances among the vertices. Note that the value **–1** indicates the **infinity**. For this homework, you can **assume that the number of vertices is less than 25**.

Sample Run 0: Assume that the user typed the following lines

```
4
0 -1 3 -1
2 0 -1 -1
-1 7 0 1
6 -1 -1 0
```

This is the correct output. In the class, we drew all five matrices such as $D^{(0)}$, $D^{(1)}$, $D^{(2)}$, $D^{(3)}$, and $D^{(4)}$. For the homework, just present the last matrix (= $D^{(4)}$).

```
0 10 3 4
2 0 5 6
7 7 0 1
6 16 9 0
```

Sample Run 1: Assume that the user typed the following lines

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

This is the correct output.

```
0 2 4
4 0 2
2 4 0
```

Sample Run 2: Assume that the user typed the following lines

```
4
0 10 -1 -1
-1 0 15 -1
-1 -1 0 20
50 -1 -1 0
```

This is the correct output.

```
0 10 25 45
85 0 15 35
70 80 0 20
50 60 75 0
```

[Hint]: This is the pseudocode of Floyd algorithm.

ALGORITHM *Floyd*$(W[1..n, 1..n])$

```
//Implements Floyd's algorithm for the all-pairs shortest-paths problem
//Input: The weight matrix W of a graph with no negative-length cycle
//Output: The distance matrix of the shortest paths' lengths
D ← W  //is not necessary if W can be overwritten
for k ← 1 to n do
    for i ← 1 to n do
        for j ← 1 to n do
            D[i, j] ← min{D[i, j], D[i, k] + D[k, j]}
return D
```