CST 370 – Fall B 2020
Homework 3
Due: 11/17/2020 (Tuesday) (11:55 PM)

How to turn in?

- Submit **three C++ or Java programs** on the iLearn. When you submit your homework programs, don't forget to include "Title", "Abstract", "ID", "Name", and "Date".
- Note that the **due time is 11:55(PM)**. This is the iLearn's timestamp, not your submission time. Since there could be a long delay between your computer and iLearn, you should **submit early**.

1. Write a program called **hw3_1.cpp (or hw3_1.java)** that reads a positive integer number from a user and reverse it. For the program, you can assume that the input number is a typical positive integer number. You don't need to consider a very large integer number.

Sample Run 0: Assume that the user typed the following number.

```
1234321
```

This is the correct output.

```
1234321
```

Sample Run 1: Assume that the user typed the following number.

```
425
```

This is the correct output.

```
524
```

Sample Run 2: Assume that the user typed the following number.

```
1200
```

This is the correct output. Note that the reverse of 1200 is not 0021. It should be 21 because we should remove the leading zeros.

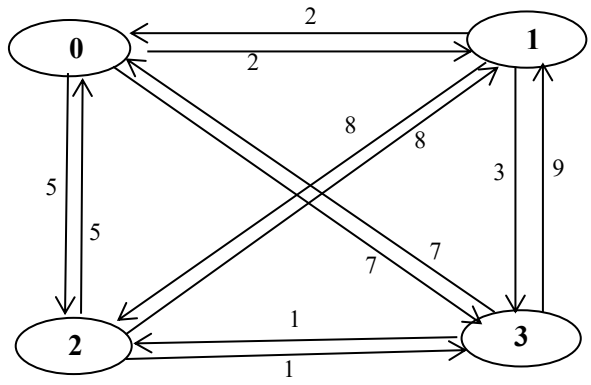
```
21
```

2. Write a C++ (or Java) program called **hw3_2.cpp or (hw3_2.java)** that reads an input graph data from a user. Then, it should present a path for the travelling salesman problem (TSP). In the assignment, you can assume that the **maximum number of vertices** in the input graph is **less than or equal to 20**.

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

```
4
12
0 1 2
0 3 7
0 2 5
1 0 2
1 2 8
1 3 3
2 0 5
2 1 8
2 3 1
3 0 7
3 1 9
3 2 1
0
```

The first line (= 4 in the example) indicates that there are four vertices in the graph. The next line (= 12 in the example) indicates the number of edges in the graph. The remaining 12 lines are the edge information with the "source vertex", "destination vertex", and "cost". The last line (= 0 in the example) indicates the starting vertex of the travelling salesman problem. This is the graph with the input information provided.



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

```
4
12
0 1 2
0 3 7
0 2 5
1 0 2
1 2 8
1 3 3
2 0 5
2 1 8
2 3 1
```

```
3 0 7
3 1 9
3 2 1
0
```

This is the correct output. Your program should present the path and total cost in separate lines.

```
Path:0->1->3->2->0
Cost:11
```

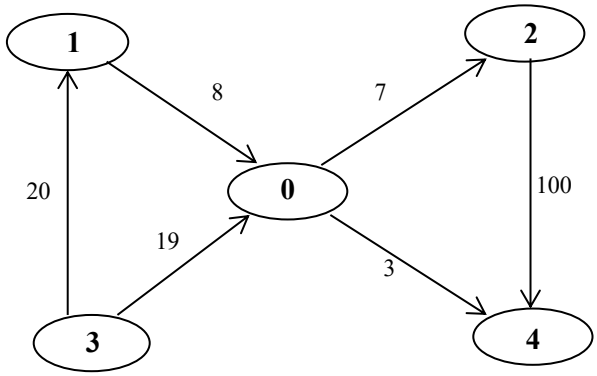
Sample Run 1: Assume that the user typed the following lines

```
5
6
0 2 7
3 1 20
0 4 3
1 0 8
2 4 100
3 0 19
3
```

This is the correct output.

```
Path:
Cost:-1
```

Note that if there is no path for the TSP, your program should **present empty path and -1 cost**.



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

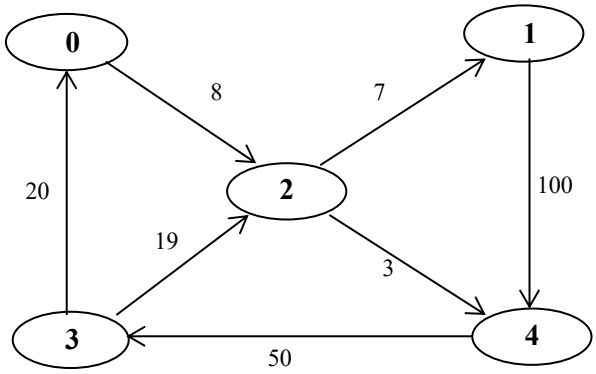
```
5
7
0 2 8
2 1 7
2 4 3
1 4 100
```

```
3 0 20
3 2 19
4 3 50
3
```

This is the correct output of your program.

```
Path:3->0->2->1->4->3
Cost:185
```

This is the directed graph of the input data:



[Hint]: To solve this problem, you can use all permutations of the vertices, except the starting vertex. For example, there are three vertices 1, 2, and 3, in the first sample run, except the starting vertex 0. This is all permutations with the three vertices

1, 2, 3
1, 3, 2
2, 1, 3,
2, 3, 1
3, 1, 2
3, 2, 1

For each permutation, you can calculate the cost. For example, in the case of the permutation "1, 2, 3", add the starting vertex city at the very beginning and end to the permutation which generates "0, 1, 2, 3, 0". This list corresponds to the path "0 → 1 → 2 → 3 → 0". Therefore, the cost of this path is "18". Similarly, the next permutation (= 1, 3, 2) corresponds to "0 → 1 → 3 → 2 → 0". The cost is 11. This way, you can check all possible paths for the TSP.

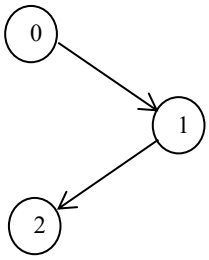
This is a sample C++ program to display permutations: https://repl.it/@YBYUN/permutationsusingSTL

3. Write a C++ (or Java) program called **hw3_3.cpp (**or **hw3_3.java)** that implements the ***Depth-First Search (DFS) algorithm*** as you learned in the class.

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

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

The first line (= 3 in the example) indicates that there are three vertices in the graph. For the homework, you can assume that the first vertex starts from the number 0. The second line (= 2 in the example) represents the number of edges, and following two lines are the edge information. This is the graph with the input information.



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

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

This is the correct output. Your program should display the mark array of DFS. For the problem, you can **assume that the starting vertex is always 0**. And also, you can **assume that the graph is connected**.

```
Mark[0]:1
Mark[1]:2
Mark[2]:3
```

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

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

This is the correct output.

```
Mark[0]:1
Mark[1]:2
Mark[2]:5
Mark[3]:3
Mark[4]:4
```

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

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

This is the correct output.

```
Mark[0]:1
Mark[1]:2
Mark[2]:4
Mark[3]:5
Mark[4]:3
```

Hint: In the lecture video, we use a stack to explain the operation of DFS. However, you **may not need to use an actual (or explicit) stack** to implement DFS. Read the following pseudocode of our textbook in which the function "dfs()"calls it recursively.

```
ALGORITHM DFS(G)
1.  //Implements a depth-first search traversal of a given graph
2.  //Input: Graph G = <V, E>
3.  //Output: Graph G with its vertices marked with consecutive integers
4.  //        in the order they are first encountered by the DFS traversal
5.  mark each vertex in V with 0 as a mark of being "unvisited"
6.  count ←0
7.  for each vertex v in V do
8.      if v is marked with 0
9.          dfs(v)
10.
11.
12. Function dfs(v)
13. //visits recursively all the unvisited vertices connected to vertex v
14. //by a path and numbers them in the order they are encountered
15. //via global variable count
16. count ←count + 1
17. mark v with count
18. for each vertex w in V adjacent to v do
19.     if w is marked with 0
20.         dfs(w)
```