Name: Jinggui Tan & Siddhant Barua
CW-ID - 10427081 & 10439929

<u>Question 2.</u>

Describe (do not implement) how you would update the above implemented *random_graph* method to generate a graph *G=(V,G)* that does not contain a negative-weight cycle. You are given a function that can determine whether or not an edge completes a negative-weight cycle.

1. Generate random graph first as what we have implemented. So there could be cycle and then negative-weight cycle exists in this graph.
2. Because we allow negative weight for edges, we use Bellman-Ford algorithm to see if there are negative-weight cycle. If we got true from Bellman-Ford, then DONE. If we get false from Bellman-Ford algorithm, we go to step 3.
3. We find a vertex that distance has changed after last RELAX, then go from this vertex via it's ancestor until we find a cycle.
4. Change the weight of some edges from this cycle until the total weight is greater than 0.(we have a lot of ways to do this, for example increase weight of the smallest edge)
5. Use provided function to test this cycle.
6. If this cycle is not a negative-weight cycle anymore, go to step 2.

<u>Question 3.</u>

Θ(V*E) or O(V*E) is the running time.

```
for(int i=1; i<=no_vert-1;i++){
  for(int j=0; j<no_vert; j++){
    for(int k=0; k<no_vert; k++){
      if(m_edge[j][k]!=INT_MAX){
        if(distance[k]>distance[j]+m_edge[j][k]){
          //cout <<j << " " <<k << " "<< distance[j] <<"  " << m_edge[j][k] << endl;
          if(distance[j] != INT_MAX ){
            distance[k]=distance[j]+m_edge[j][k];
          }else{
            // in case of integer overflow
            distance[k]=distance[j];
          }
          pred[k]=j;
        }
      }
    }
  }
}
```

If we look at the code, it looks like we would run n^3 times because there are 3 nested loops. Then the time complexity would be O(n^3). But actually, the codes would only be executed when there is edge.

Name: Jinggui Tan & Siddhant Barua
CW-ID - 10427081 & 10439929

So, the RELAX process will run |V|-1 loops, and in each loop when there is an edge, we run the RELAX. We got E edges. In total, we run the codes for (|V|-1)*E times. Strictly speaking, the time complexity is $\Theta(V*E)$, and $\Theta(V*E)$ is also $O(V*E)$.

Codes:

For the command input, you need input all arguments including n->number of vertices, m-> number of edges, w-> weight interval [-w, w] and s->source node.

Here in the graph.cpp we have a set_edge(int **) function.
If you want to manually test it, you could just input the whole adjacency matrix to set_edge(int **) and then call Bellman_Ford or Floyd_Warshall algorithm.

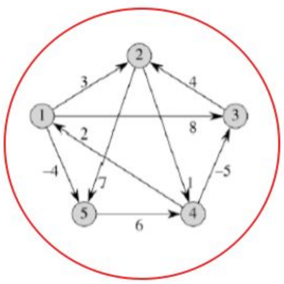
For example :

```
// manully testing
  int testMatrix[5][5] ={
// A B C D E
// 1 2 3 4 5
  {0,3,8,INT_MAX,-4},//A 1
  {INT_MAX,0,INT_MAX,1,7},//B 2
  {INT_MAX,4,0,INT_MAX,INT_MAX},//C 3
  {2,INT_MAX,-5,0,INT_MAX},//D 4
  {INT_MAX,INT_MAX,INT_MAX,6,0},//E 5
};

int ** edge = new int*[5];
for(int i=0; i<5; i++){
  edge[i] = testMatrix[i];
}
g.set_edge(edge);
g.output();

if(!g.BF(s)){
  cout << "exist negetive-weight cycle!" << endl;
}
g.FW();
return 0;
```

Name: Jinggui Tan & Siddhant Barua
CW-ID - 10427081 & 10439929

Now we do this test for the following graph and the expected output is as follows

Example:



$$D^{(0)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(0)} = \begin{pmatrix} \text{NIL} & 1 & 1 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & \text{NIL} & \text{NIL} \\ 4 & \text{NIL} & 4 & \text{NIL} & \text{NIL} \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(1)} = \begin{pmatrix} \text{NIL} & 1 & 1 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & \text{NIL} & \text{NIL} \\ 4 & 1 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(2)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(2)} = \begin{pmatrix} \text{NIL} & 1 & 1 & 2 & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & 2 & 2 \\ 4 & 1 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(3)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(3)} = \begin{pmatrix} \text{NIL} & 1 & 1 & 2 & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & 2 & 2 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(4)} = \begin{pmatrix} 0 & 3 & -1 & 4 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \quad \Pi^{(4)} = \begin{pmatrix} \text{NIL} & 1 & 4 & 2 & 1 \\ 4 & \text{NIL} & 4 & 2 & 1 \\ 4 & 3 & \text{NIL} & 2 & 1 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ 4 & 3 & 4 & 5 & \text{NIL} \end{pmatrix}$$

$$\boxed{D^{(5)} = \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \quad \Pi^{(5)} = \begin{pmatrix} \text{NIL} & 3 & 4 & 5 & 1 \\ 4 & \text{NIL} & 4 & 2 & 1 \\ 4 & 3 & \text{NIL} & 2 & 1 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ 4 & 3 & 4 & 5 & \text{NIL} \end{pmatrix}}$$

And our output is as follows :

```
[[      0       3       8       Inf     -4      ]
[       Inf     0       Inf     1       7       ]
[       Inf     4       0       Inf     Inf     ]
[       2       Inf     -5      0       Inf     ]
[       Inf     Inf     Inf     6       0       ]
]
Bellman-Ford:
Vertex          Distance from 2 Pre
1               0               NIL
2               1               3
3               -3              4
4               2               5
5               -4              1

Floyd-Warshell: weight matrix
0       1       -3      2       -4
3       0       -4      1       -1
7       4       0       5       3
2       -1      -5      0       -2
8       5       1       6       0

Floyd-Warshell: predcessor matrix
NIL     3       4       5       1
4       NIL     4       2       1
4       3       NIL     2       1
4       3       4       NIL     1
4       3       4       5       NIL
```

Name: Jinggui Tan & Siddhant Barua
CW-ID - 10427081 & 10439929

NOTE:: For testing in main , comment out necessary parts, as if we create a random graph and soon after create a custom graph which we input via the adjacency matrix, the new custom graph re-writes the properties of the random graph and this leads to an unfavorable output.