BITS Pilani, Pilani Campus
2nd Sem. 2019-20
CS F211 Data Structures & Algorithms
===============================
Lab XII
===

Topic: Graphs

Programming Environment: C on Linux

Problem-1: Read the following problem and its possible solution. Since this is the last lab of the course and first lab on graphs, see the solution to understand how graphs are implemented. Then, go to implement Problem-2.

Consider a graph G = (V, E, w) - where w is an integer function on V - stored as an adjacency list. Write a best-first traversal procedure: starting at the root, pick the next vertex to be visited as one with the highest weight among the children of all vertices already visited. Given count of vertices, V, create an empty graph with adjacency list representation. The directed graph will be supplied as edge list, where each edge is represented by a pair of vertices which are integers in the range [0, V). Another function will then supply the weights to all the vertices of the graph.

Complete definition of following functions in `graph.c`. Write any additional functions and data structures in `extras.c` and `extras.h`. Consider `driver.c` as the driver program.

a) `Graph initGraph(int V)`
   Return an empty V-vertices adjacency list with all relevant fields properly initialized.

b) `void printAdjacencyList(Graph g)`
   Print data of each vertex in a line, with following format:
   <vertex id> <tab> ==> <adjacent vertex id 1> <adjacent vertex id 2>

c) `void insertEdge(Graph g, unsigned int u, unsigned int v)`
   Add an edge in the graph between u and v.

d) `void bestFirstTraverse(Graph g)`
   Traverse and print the graph in best first traversal order.

[Hint: You may implement a priority queue in extras.c and extras.h to identify the best adjacent neighbour to pick for expansion during traversal.]

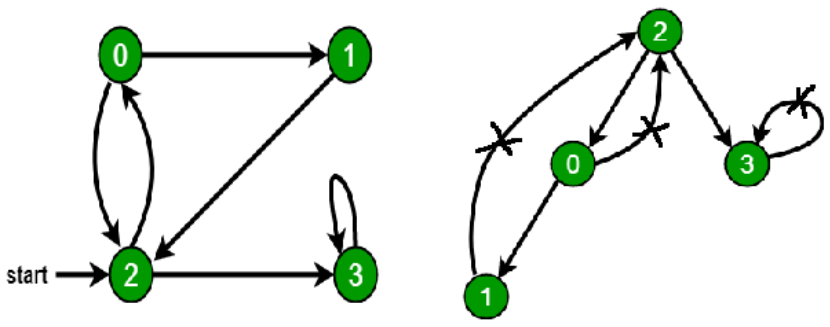
Sample Test Case:
Input: `15042018   10`
Output:
```
8 1
6 5
3 9
8 9
6 1
8 5
```

```
2 4
2 5
8 7
5 0
7 9
9 5
8 2
9 8
6 0
3 7
8 4
2 7
5 9
0 2

Weights: 70     90    70    40    90    60    30    80    10    20
Adjacency List:
0     (70)  ==> 2
1     (90)  ==>
2     (70)  ==> 7       5     4
3     (40)  ==> 7       9
4     (90)  ==>
5     (60)  ==> 9       0
6     (30)  ==> 0       1     5
7     (80)  ==> 9
8     (10)  ==> 4       2     7     5     9     1
9     (20)  ==> 8       5

Best First Traversal:
0     2     1     3     7     9     4     5     6     8
```

Problem-2: Depth First Search for a Graph

Depth First Traversal (or Search) for a graph is similar to Depth First Traversal of a tree. The only catch here is, unlike trees, graphs may contain cycles, so we may come to the same node again. To avoid processing a node more than once, we use a boolean visited array.

For example, in the following graph, we start traversal from vertex 2. When we come to vertex 0, we look for all adjacent vertices of it. 2 is also an adjacent vertex of 0. If we don't mark visited vertices, then 2 will be processed again and it will become a non-terminating process. A Depth First Traversal of the following graph is 2, 0, 1, 3.


$$\begin{tikzcd} 0 \arrow[r] \arrow[d, bend right] & 1 \arrow[dl] \\ \text{start} \arrow[r] & 2 \arrow[u, bend right] \arrow[r] & 3 \arrow[loop above] \end{tikzcd}$$

Store the graph as an adjacency list representation and find its DFS. Modularize the code yourself.