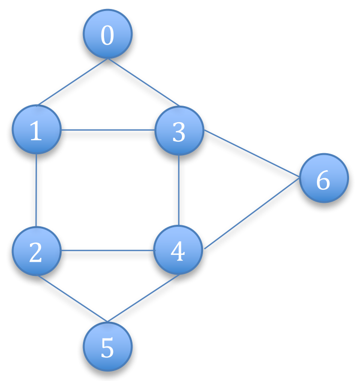
DSA 5005 Data Structures – Summer 2017 – Programming Project 3
Due August 11, 2017 – 11:59 PM

Objectives

1. [30 pts] Graph Class with Bit Representation of Adjacency Matrix
2. [10 pts] Add Edge
3. [10 pts] Using Queue Standard Template Library for use as part of BFS (see below)
4. [10 pts] Ostream operator to print the graph in the edge list format
5. [10 pts] Implement the copy constructor and the overloaded = (assignment) operator
6. [30 pts] Breadth First Search (BFS) and output the parent array representation of the BFS tree. This function will take as input the Graph object and the start vertex p.
7. [20 pts Bonus] Perform Depth First Search (DFS) and output the parent array representation of the DFS tree. This function will take as input the Graph object and the start vertex p.
8. Document your project thoroughly as the examples in the textbook. This includes but not limited to header comments for all classes/methods, explanatory comments for each section of code, meaningful variable and method names, and consistent indentation. Program that lack or weak in documentation will receive a *deduction* of up to 30% of the grade.

Project Description

A graph G = (V, E) consists of vertex set V and the edge set E. We will assume the graph G is undirected implying that the edges have no direction. We will also assume that the graph is connected (there exists a path between every pair of nodes).



```
7       Number of nodes
2 4     Edge from 2 to 4
0 1
3 0
3 6
6 4
3 4
1 3
1 2
2 4
5 2
4 5
```

In the above we have provided a graph and an edge list representation of the graph. The above graph can be stored in a matrix which is declared as follows. This matrix is called the adjacency matrix.

bool AdjacencyMatrix[7][7];

The matrix element AdjacencyMatrix[i][j] is set to true if there is an edge between nodes numbered i and j. For example, for the graph above AdjacencyMatrix[4][3] is set to true, since there is an edge between nodes 4 and 3. Since the edges no direction (undirected graph), the matrix element corresponding to edge (3,4) AdjacencyMatrix[3][4] is also set to true. We will assume that AdjacencyMatrix[i][i] for all i, is set to false.

We will now show the entire adjacency matrix for the above graph in the following (note that a 0 represents false and 1 represents true).

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

In C/C++ programming language each bool takes 8 bits. The total size consumed by the matrix is 7*7*8 bits = 49 Bytes.

Bit Representation of Adjacency Matrix

We can represent each row of the above matrix by an unsigned char (8 bits). Note that we need only seven bits, but we can assume that the last bit as a zero. For example, the bit 8 bits corresponding to the first row will be 01010000 which corresponds to the letter P. The second rows bits are 10110000 and it corresponds to the character ° (degree symbol). Note that in this case, the first bit is actually the most significant bit, and the least significant bit is the unused 8th bit (in other words, all of your values will be even numbers). In this representation, the total number of bits will 8*8 bits = 8 bytes.

Now, let us say that we have 10 nodes in a graph. The size of AdjacencyMatrix structure using the bool data type will be 10*10*8 = 100 bytes. For the bit representation of the matrix, we need two unsigned character per row. The 6 lower order bits of the second unsigned character will be all 0's. The will give us a total size of 10*2 bytes (one byte for each unsigned character).

In general, if the number of nodes is *n*, then the total number of unsigned characters per row will be $\left\lceil \frac{n}{8} \right\rceil$ (ceiling of n/8). In C++, you can declare the bit representation of the matrix as follows:

```
#include <iostream>
using namespace std;

int main () {

  int n = 10;
  int size;

  // Create the bit Adjacency Matrix structure
  if (n%8 == 0)
    size = n/8;
  else
    size = (int) (n/8) + 1;

  unsigned char** bitAdjacencyMatrix;
  bitAdjacencyMatrix = new unsigned char* [n];
  for (int i=0; i < n; i++) {
    bitAdjacencyMatrix[i] = new unsigned char[size];
  }

  //Initialize the bit Adjacency Matrix structure
  for (int i=0; i < n; i++)
    for (int j=0; j < size; j++)
       bitAdjacencyMatrix[i][j] = 0x00 << 8;
  }
}
```

Bit Manipulation in C/C++

Given an unsigned char and a position k, we can set the bit at position k to be 1, set the bit at position k to be 0, or check bit at position k for 0 or 1. Note that the bits are numbered 0 through 7 from right to left.

```
void printBits (unsigned char s) {

 //print the bit pattern
 for (int i=0; i < 8; i++)
   if (s & (1 << (7-i)))
       cout << 1;
   else
       cout << 0;
}

int main () {

  unsigned char l = 0x01 << 1; // set l to be equal to 2;
  cout << "Printing the bits before changing: ";
  printBits(l);
  cout << endl;

  //set the third bit from right to be a 1
  l |= 0x01 << 2;
  cout << "Printing the bits after setting the third bit from the right to be a 1: ";
  printBits(l);
  cout << endl;

  //set the second bit from the right to be a 0
  l &= ~(0x01 << 1);
  cout << "Printing the bits after setting the second bit from the right to be a 0: ";
  printBits(l);
  cout << endl;

 //check the second bit from the right
 if (l & (1 << 1))
   cout << "Second bit from right is set" << endl;
 else
   cout << "Second bit from right is not set" << endl;
}
```

The Graph Class

As part of this project you will design a graph class that stores the graph using the bitAdjacencyMatrix data structure described earlier. Your class structure should look something like this:

```
class BitMatrixGraph {
     protected:
       unsigned char** bitAdjacencyMatrix;
       int n; //number of nodes in the graph
      public:
       BitMatrixGraph();
       BitMatrixGraph(int numNodes);
       BitMatrixGraph(const BitMatrixGraph& otherGraph)
       //ostream operator
       //overloaded = operator
       void addEdge (int u, int v);
       bool isAnEdge (int u, int v);
       //other that you want and will be useful
};
```

You will create an instance of the BitMatrixGraph after reading the first line of the input file that contains the number of nodes in the input graph. The input structure is described in the next section.

Input File

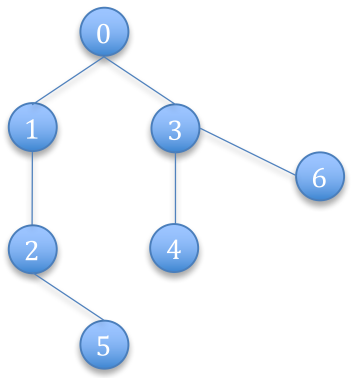
The first line of the input contains an integer that corresponds to the number of nodes in the input graph. Followed by this integer, there will be unknown number of lines of input. Each line of input will contain two integer u and v, indicating that there is an edge between vertices u and v.

Processing and Output

In the main program create an object of BitMatrixGraph after reading the number of vertices and calling the appropriate constructor. Next, after reading each edge u and v, call the addEdge method on the graph. This method will set the bit v corresponding to row u and bit u corresponding to row v. After all edges re processed, use the printBits function as part of the ostream operator to print data structure that stores the graph (bitAdjacencyMatrix).

After the graph is created you need to write C++ statements to demonstrate copy constructor and overloaded = operator.

After the graph is created you will call a function that will perform the Breadth First Search. This algorithm (and C++ method) is provided in your textbook. The output of this function is will be a parent array to store the parent of each node in the Breadth First Search. If we assume that vertex 0 is the start vertex, the BFS search tree and the parent representation is shown below.



```
 0  1 2 3 4 5 6 <- Nodes
-1 0 1 0 3 2 3 <- Parent Number
```

Constraints

1. This project must be implemented in C++. The only header file you will have <iostream>, <queue>, and <stack> standard template libraries if needed.
2. None of the projects will be a group project. Consulting with other members of this class on programming projects is strictly not allowed and plagiarism charges will be imposed on students who do not follow this.