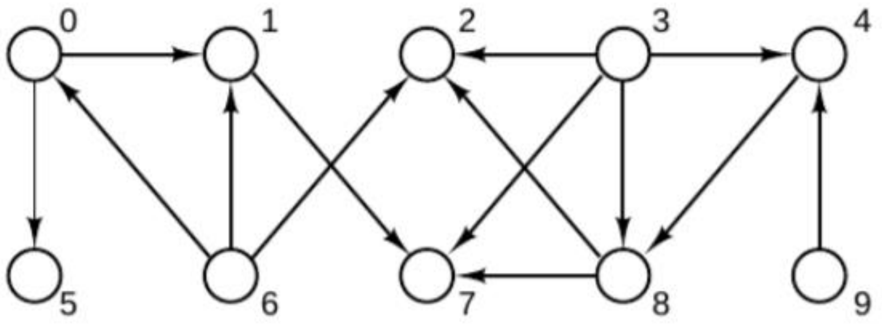
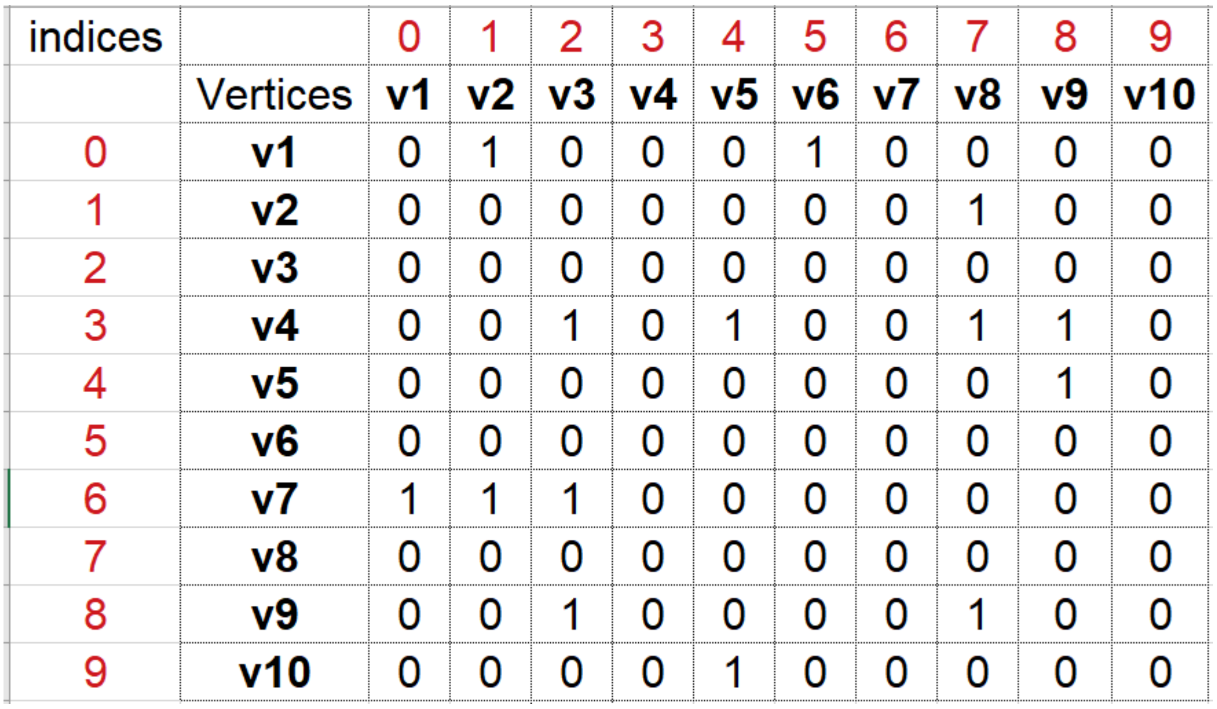
This document written **by Tevhit Karslı** to informal design document my Breadth-first sorting coding.

This is the graph:



This is graph representation to top graph.



indices		0	1	2	3	4	5	6	7	8	9
	Vertices	**v1**	**v2**	**v3**	**v4**	**v5**	**v6**	**v7**	**v8**	**v9**	**v10**
0	**v1**	0	1	0	0	0	1	0	0	0	0
1	**v2**	0	0	0	0	0	0	0	1	0	0
2	**v3**	0	0	0	0	0	0	0	0	0	0
3	**v4**	0	0	1	0	1	0	0	1	1	0
4	**v5**	0	0	0	0	0	0	0	0	1	0
5	**v6**	0	0	0	0	0	0	0	0	0	0
6	**v7**	1	1	1	0	0	0	0	0	0	0
7	**v8**	0	0	0	0	0	0	0	0	0	0
8	**v9**	0	0	1	0	0	0	0	1	0	0
9	**v10**	0	0	0	0	1	0	0	0	0	0

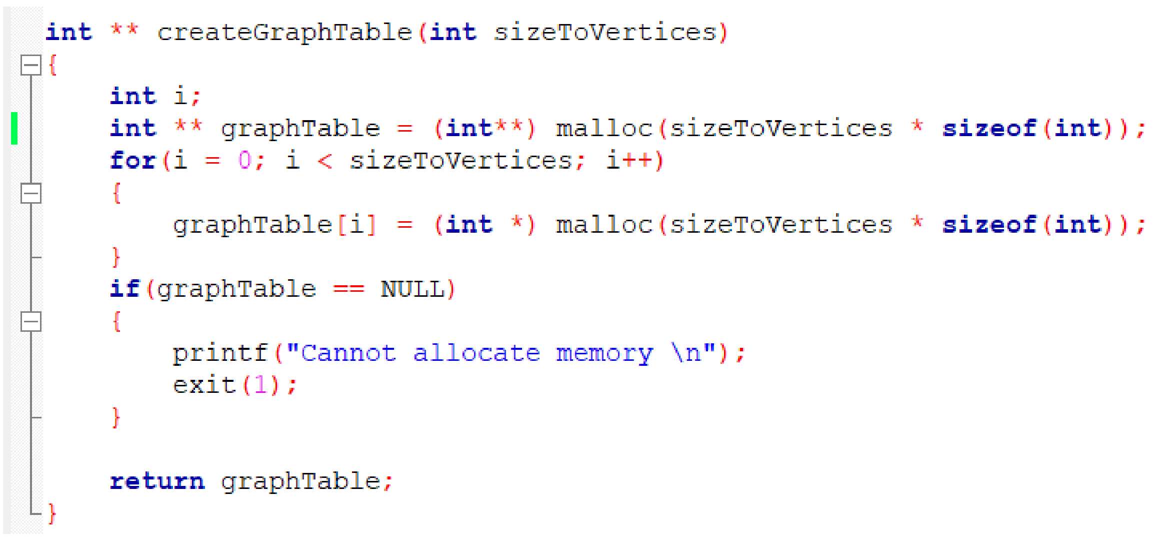
```
#define VERTICIES 10
#define EDGES_COUNT 14
```

I defined constants according to the graph and graph representation. The "VERTICIES" constant is 10, because the graph has 10 vertices. The "EDGES_COUNT" is 14, because the graph has 14 edges.

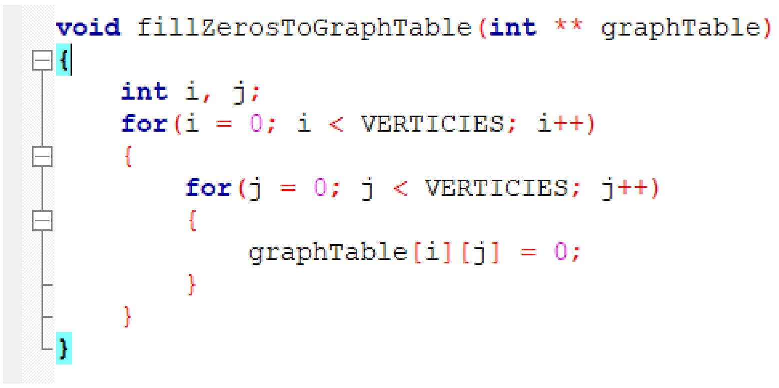
In my main code, I firstly created an adjacency table/graph table. Then, I continue with it. When you look "main" function, you will see this command firstly.

```c
int ** myGraphTable = createGraphTable(VERTICIES);

fillZerosToGraphTable(myGraphTable);
```

This commands created a graph table(matrix) and fill the matrix columns and rows with zero. The "createGraphTable" and "fillZerosToGraphTable" functions provides this operation. This functions are;

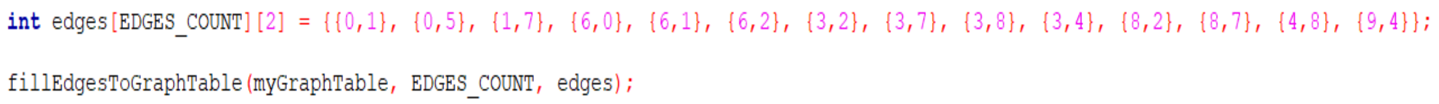


```c
int ** createGraphTable(int sizeToVertices)
{
    int i;
    int ** graphTable = (int**) malloc(sizeToVertices * sizeof(int));
    for(i = 0; i < sizeToVertices; i++)
    {
        graphTable[i] = (int *) malloc(sizeToVertices * sizeof(int));
    }
    if(graphTable == NULL)
    {
        printf("Cannot allocate memory \n");
        exit(1);
    }

    return graphTable;
}
```



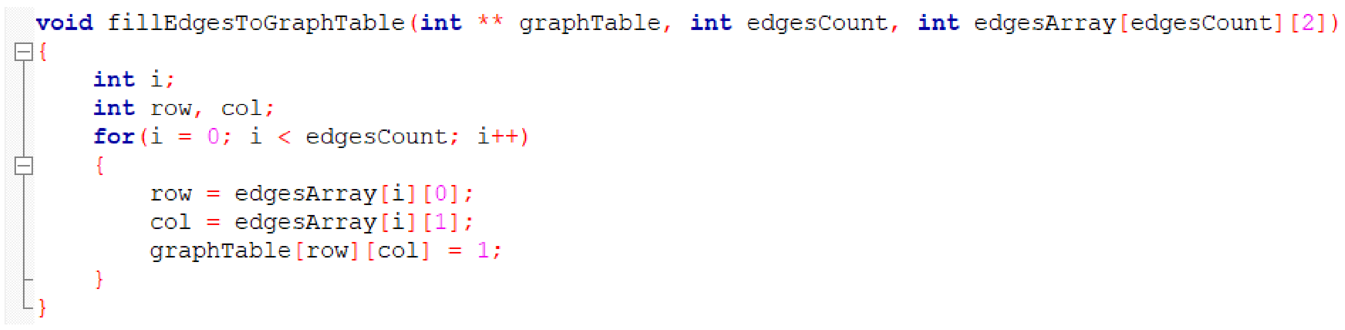
```c
void fillZerosToGraphTable(int ** graphTable)
{
    int i, j;
    for(i = 0; i < VERTICIES; i++)
    {
        for(j = 0; j < VERTICIES; j++)
        {
            graphTable[i][j] = 0;
        }
    }
}
```

Secondly, I created a matrix named “edges” to store the edges according to the graph. I added the edges to matrix manuelly.



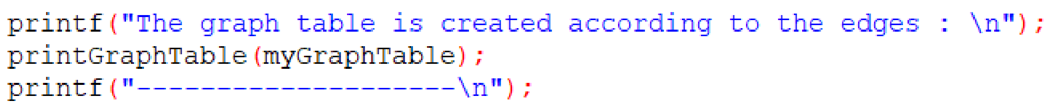
```
int edges[EDGES_COUNT][2] = {{0,1}, {0,5}, {1,7}, {6,0}, {6,1}, {6,2}, {3,2}, {3,7}, {3,8}, {3,4}, {8,2}, {8,7}, {4,8}, {9,4}};

fillEdgesToGraphTable(myGraphTable, EDGES_COUNT, edges);
```

Then I fill graph table with “edges” matrix. The “fillEdgesToGraphTable” function provides filling graph table(matrix) with edges.

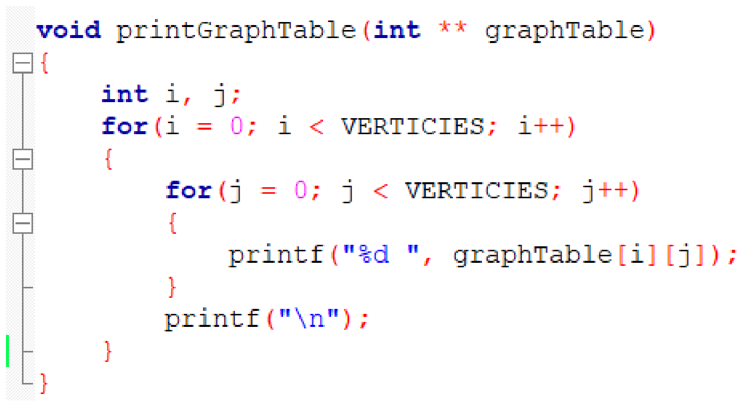


```
void fillEdgesToGraphTable(int ** graphTable, int edgesCount, int edgesArray[edgesCount][2])
{
    int i;
    int row, col;
    for(i = 0; i < edgesCount; i++)
    {
        row = edgesArray[i][0];
        col = edgesArray[i][1];
        graphTable[row][col] = 1;
    }
}
```

Thirdly, I printed the graph table on screen with “printGraphTable” function.

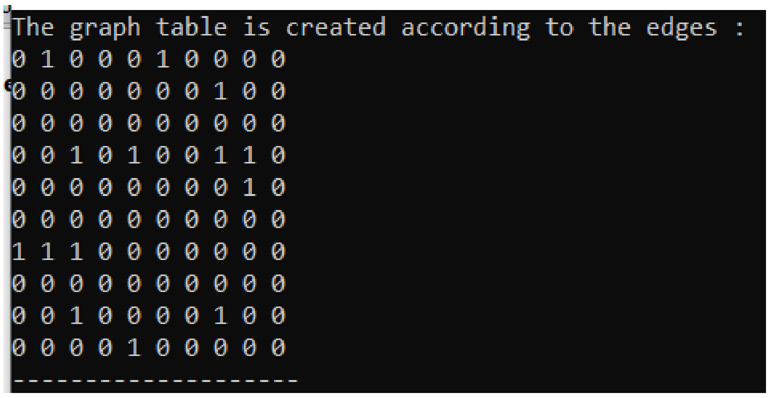


```
printf("The graph table is created according to the edges : \n");
printGraphTable(myGraphTable);
printf("--------------------\n");
```



```
void printGraphTable(int ** graphTable)
{
    int i, j;
    for(i = 0; i < VERTICIES; i++)
    {
        for(j = 0; j < VERTICIES; j++)
        {
            printf("%d ", graphTable[i][j]);
        }
        printf("\n");
    }
}
```

The “printGraphTable” function provides the print screen the matrix which took in paramater. So, the screen is that:



```
The graph table is created according to the edges :
0 1 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 0
0 0 1 0 1 0 0 1 1 0
0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0 0 0
1 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 1 0 0
0 0 0 0 1 0 0 0 0 0
--------------------
```

Fourthly, I send graph table and its vertices count to “BFS” function. The function provides sorting graph table with breadth-first algorithm then return an array that sorted vertex number. Here, the returned array is assigned to the “BFS_Array” array.

```
int * BFS_Array = BFS(myGraphTable, VERTICIES);
```

```
int * BFS(int ** graphTable, int verticesCount)
{
    int * bfsArray = (int*) malloc(verticesCount * sizeof(int));

    Queue * queue = createQueue(MAX);

    int * indegrees = findFirstIndegree(graphTable, verticesCount);

    int i, j;

    int state[verticesCount];
    for(i = 0; i < verticesCount; i++) {
        state[i] = 0;
    }

    int vertex;
    for(i = 0; i < verticesCount; i++)
    {
        for(j = 0; j < verticesCount; j++)
        {
            if(indegrees[j] == 0 && state[j] == 0)
            {
                enqueue(queue, j);
                state[j] = 1;
            }
        }
        dequeue(queue, &vertex);
        bfsArray[i] = vertex;
        for(j = 0; j < verticesCount; j++)
        {
            if(graphTable[vertex][j] == 1)
                indegrees[j]--;
        }
    }
    destroyQueue(queue);

    return bfsArray;
}
```

When we, look the “BFS” function:

The function take graph table(matrix) and its vertices count in parameter. In the function, I create an array named “bfsArray” to store sorted graph vertex. I take indegrees values of graph with “findFirstIndegree” function(*the function description is bottom*). So, I obtained initial indegrees in “indegrees” array.

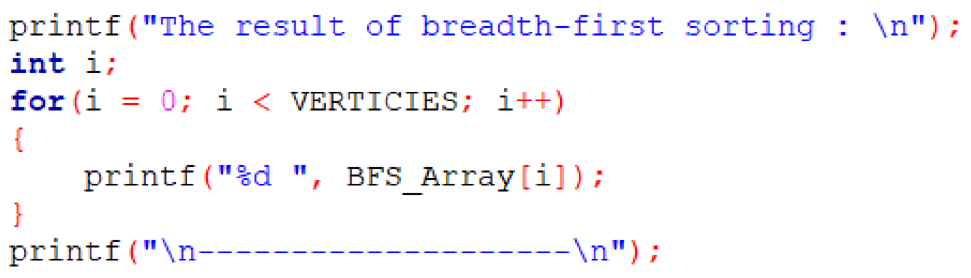
I created an array named “state”. I determine zero initial values of array elements. Every array elements is to vertex in graph. I check vertices every step if it operated or not.

I created a queue to make breadth-first sorting algorithm. I created a for loop to make enqueue and dequeue operation for queue. This operation maket he value of “verticesCount” times. So, I determined the loop “verticesCount” times.

In loop,

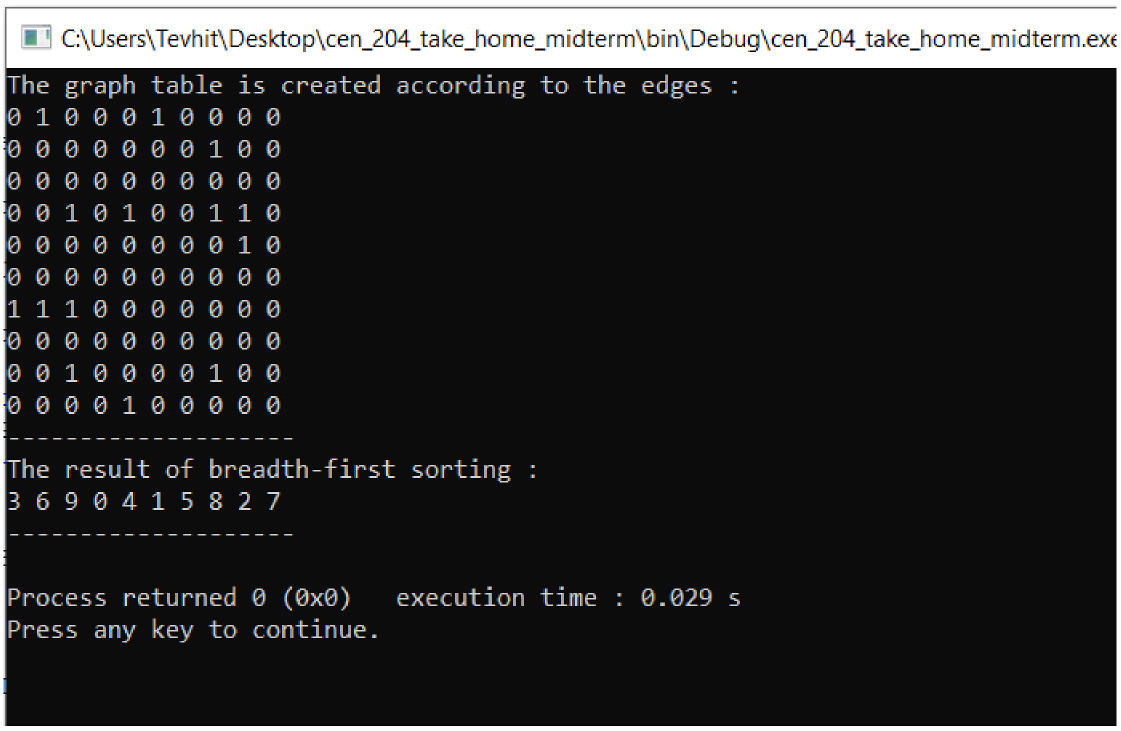
- I make enqueue operation according to the indegrees. I make enqueue indegree value of 0. (*initially 3-6-9 vertex is zero indegree, so, the code make 3 enqueue*). Then I determined 1 value of “state”. So, another time, it will not enqueue.
- I make dequeue operation for my “queue”. Then it will be sorted first vertex. I add “bfsArray” to vertex.
- To second and then steps, I decrease one of the value indegrees of edges to dequeue’ded value.

So, I obtained sorted array according to the breadth-first sorting algorithm in “bfsArray” array. I returned main function this array.



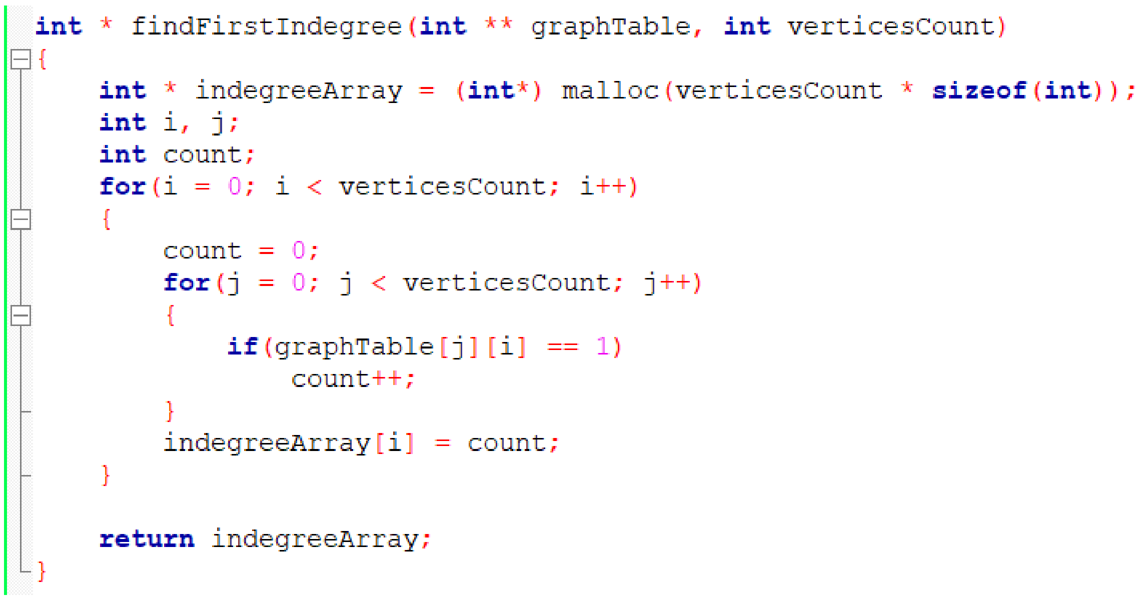
```
printf("The result of breadth-first sorting : \n");
int i;
for(i = 0; i < VERTICIES; i++)
{
    printf("%d ", BFS_Array[i]);
}
printf("\n--------------------\n");
```

In main function; finally, I print screen to sorted vertex number which is in “BFS_Array” array. So, I obtained the following screen output:



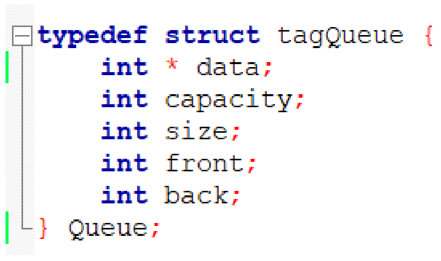
```
C:\Users\Tevhit\Desktop\cen_204_take_home_midterm\bin\Debug\cen_204_take_home_midterm.exe
The graph table is created according to the edges :
0 1 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 0
0 0 1 0 1 0 0 1 1 0
0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0 0 0
1 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 1 0 0 0
0 0 0 0 1 0 0 0 0 0
--------------------
The result of breadth-first sorting :
3 6 9 0 4 1 5 8 2 7
--------------------

Process returned 0 (0x0)   execution time : 0.029 s
Press any key to continue.
```

The following “findFirstIndegree” function is provides to obtain firs indegree of vertices according to the edges of graphs. It returns an array.



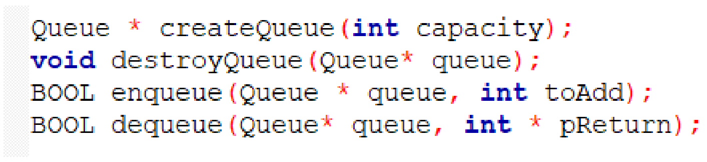
```c
int * findFirstIndegree(int ** graphTable, int verticesCount)
{
    int * indegreeArray = (int*) malloc(verticesCount * sizeof(int));
    int i, j;
    int count;
    for(i = 0; i < verticesCount; i++)
    {
        count = 0;
        for(j = 0; j < verticesCount; j++)
        {
            if(graphTable[j][i] == 1)
                count++;
        }
        indegreeArray[i] = count;
    }

    return indegreeArray;
}
```

Also, I have a Queue struct and I have its functions to use the queue.



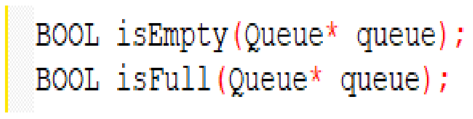
```c
typedef struct tagQueue {
        int * data;
        int capacity;
        int size;
        int front;
        int back;
} Queue;
```

The struct is created to stored “Queue”. I use queue to BFS algorithm code.



```c
Queue * createQueue(int capacity);
void destroyQueue(Queue* queue);
BOOL enqueue(Queue * queue, int toAdd);
BOOL dequeue(Queue* queue, int * pReturn);
```

These functions provide the use queue operation. I can create a queue, destroy a queue and put-get to queue.



```c
BOOL isEmpty(Queue* queue);
BOOL isFull(Queue* queue);
```

These functions provide the check queue is ful or is empty. I can check queue to determine is it empty or is it full when I make put-get operation to queue.

Tevhit Karslı

16 07 06 017

tevhitkarsli@hotmail.com