EE 441 HOMEWORK #1

Graph Traversals

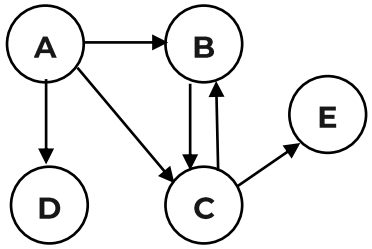
Due: November 11, 2018, 23:59
For questions: ccakmak@metu.edu.tr

Introduction: Graphs

A graph is a data structure that is composed of a number of vertices (single: vertex, nodes) and a number of edges (arcs) that connect them. Common use of graph data structures include routing algorithms for computer networks such as Dijkstra's shortest path algorithm and conditional dependence relations such as Bayesian networks. All map and navigation applications make use of graphs.

If the travel directions among connected vertices of a graph are restricted then the graph is a **directed graph**. In the directed graph shown below, for example, Vertex A has an edge to Vertex B but does not have an edge to A. B and C have edges from each other.

Graphs can be represented in a number of ways, one of which is the **adjacency matrix**. In this matrix, each row shows whether a certain vertex has an edge to other vertices or not. In the adjacency matrix for the example graph, the first row represents the edges from vertex A. It shows that Vertex A has edges to B, C and D since the respective columns are 1 whereas the other columns are 0.



	A	B	C	D	E
A	0	1	1	1	0
B	0	0	1	0	0
C	0	1	0	0	1
D	0	0	0	0	0
E	0	0	0	0	0

In this homework, we will implement Breadth-First Traversal and Depth-First Traversal algorithms. Both algorithms start from a given graph vertex and **visit all vertices in the graph exactly once**. **In this homework visiting a node means printing its name on the screen.**

Breadth-First Traversal (BFT)

After visiting a given vertex v, the breadth-first traversal (BFT) algorithm visits every unvisited vertex adjacent to v before visiting any other vertex. Thus, from the starting node, we follow all paths of length one. **The algorithm keeps track of the last visited vertices to reach the unvisited vertices that are reachable from these nodes.** Then we follow paths of length two that go to unvisited nodes. We continue increasing the length of the paths until there are no unvisited nodes along any of the paths.

The breath-first traversal algorithm does not completely specify the order in which it should visit the vertices adjacent to v. **The algorithm keeps track of the visited vertices not to visit them again.**

In this homework we implement Breadth-First traversal (BFT) as an iterative algorithm described as follows:

Iterative BFT Algorithm

Inputs: starting_vertex, graph

Required data structures:

- A queue to keep track of last visited vertices
- Some way of remember all visited vertices not to visit them again

Initialization:

Visit starting_vertex

Add starting vertex to the queue and remember that it is visited

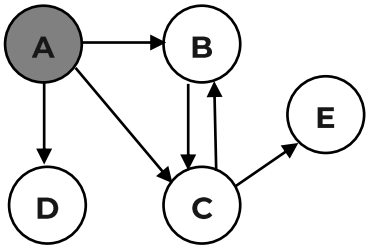
In a loop (iterative):

Delete the front vertex from the queue

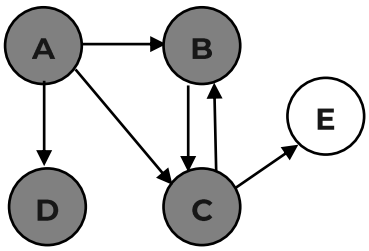
Visit all unvisited vertices that are reachable from the deleted vertex

Insert these new vertices to the queue

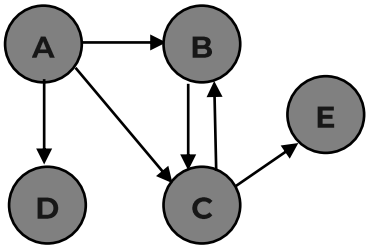
Here is an example for BFT with starting_vertex=A



Output: (A)



Output: (A B C D)



Output: (A B C D E)

The traversal output of BFT for the given graph, starting from Vertex A is "A B C D E".

Depth-First Traversal

For a given vertex v, the depth-first traversal (DFT) algorithm proceeds along a path from v as deeply into the graph as possible until we reach a dead end. We then **backtrack** until we reach a vertex with an edge to an unvisited vertex.

We take this edge and again follow it until we reach a dead end.

This process continues until we back up to the starting vertex and it has no edges to unvisited vertices. The depth-first traversal algorithm does not completely specify the order in which it should visit the vertices adjacent to v.

Iterative DFT Algorithm

Inputs: starting_vertex, graph

Required data structures:

- A stack for backtracking to when a dead-end is reached
- Some way of remember all visited vertices not to visit them again

Initialization:

Visit starting_vertex

Add starting vertex to the stack and remember that it is visited

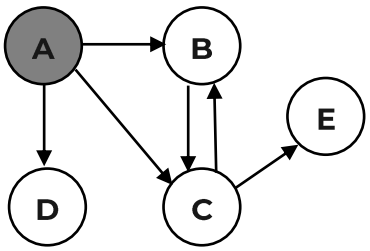
In a loop (iterative):

Visit an unvisited vertex reachable from the vertex at the top of the stack

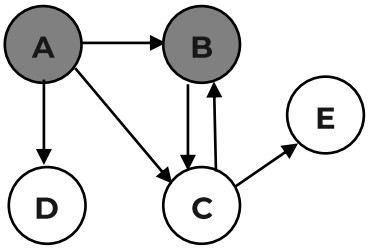
Push that vertex on to the stack

If all vertices reachable from the vertex at the top of the stack are visited, pop the stack

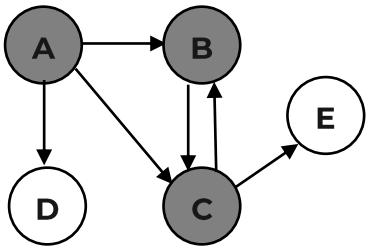
Here is an example for DFT with starting_vertex=A



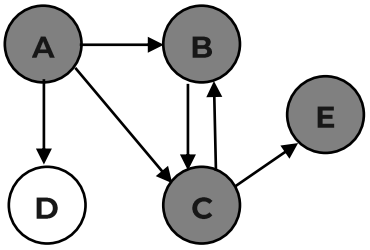
Output: (A)



Output: (A B)

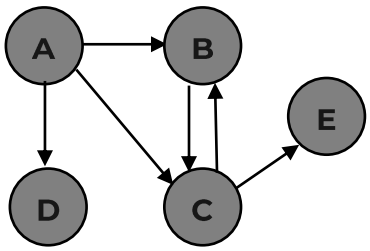


Output: (A B C)



Output: (A B C E)

- Backtrack to A



Output: (A B C E D)

The traversal output of DFT for the given graph, starting from Vertex A is "A B C E D".

Homework

Preparation: Complete the implementation of the Stack and Queue Classes covered in the Lecture Notes according to the comments. You can re-use all the code provided in the lecture notes.

The class declarations are as follows:

```
const int MaxStackSize=50;
template <class T>
class Stack
{
private:
T stacklist[MaxStackSize];
int top;
public:
Stack(void); // constructor to initialize top
//modification operations
void Push(const T& item);
T Pop(void);
void ClearStack(void);
//just copy top item without modifying stack contents
T Peek(void) const;
//check stack state returns top element value without removal
int StackEmpty(void) const;
//returns true if the stack is empty
int StackFull(void) const;
//returns true if the stack is full
};
```

```
# include <iostream>
# include <stdlib>

const int MaxQSize=50;

template <class T>
class Queue

{
private:

// queue array and its parameters

int front, rear, count;
T qlist[ MaxQSize] ;

public:

//constructor

Queue(void); // initialize data members
//queue modification operations
void QInsert(const T& item);

T QDelete(void);

void ClearQueue(void);
// queue access

T QFront(void) const;
// queue test methods

int QLength(void) const;
// returns the nuber of stored elements

int QEmpty(void) const;
//returns true if the queue is empty
int QFull(void) const;
//retuns true if the queue is full
};
```

Part 1) (70 pts) You will write a single program that implements both BFT and DFT algorithms to traverse a given graph (As defined in the previous sections of this assignment).

- Your input will be a text file named “E441HW1.txt”, in which resides an adjacency matrix representing a graph with exactly 6 vertices, which are ordered as (A, B, C, D, E, F). Your program should be able to read the text file and store the matrix correctly. An example input is given in the homework folder.
- The starting vertex for traversal is always “A”.

- You are expected to print out two separate results on the terminal: First the output of BFT then the output of DFT.

Part 2) (30 pts) You will write a single program (a modified version of your code in Part 1) that takes a *char* input from the user to set the destination vertex and prints out two paths from the starting vertex "A" to the given destination: One found with BFS algorithm and the other with DFS. For the example graph given in this homework manual, if the user enters Vertex "E" as the destination, the output of BFS should be "A C E" while the output of DFS should be "A B C E".

Run the code for the given text file in the homework folder and enter the destination as "F". Observe the resulting paths for both algorithms and try to understand the differences between these two approaches.

Regulations:

1. Use **Code::Blocks IDE** and choose GNU GCC Compiler while creating your project. Name your project as "e<student_ID>_HW1". Send the whole project folder compressed in a rar or zip file. You will not get full credit if you fail to submit your project folder as required.

2. The code you uploaded should be compilable and the built file should be executable. **We have to see some output to grade your homeworks.** So please make sure that you have uploaded a working version for your homework.

3. You should insert comments to your source code at appropriate places without including any unnecessary detail. Comments will be graded (On the condition that you have managed to send a working code). A code with insufficient/excessive comments is not a proper piece of work.

4. The homework is to be prepared individually. It is not allowed to prepare the homework as a group. METU honor code is essential. Do not share your code. **Any kind of involvement in cheating will result in a zero grade for all homeworks, for both givers and receivers.**

5. You have to give the links to any website you have benefitted from. You can add them to your code as comments in the beginning.

6. Late submissions are welcome, but penalized according to the following policy:

    - 1 day late submission: HW will be evaluated out of 70.
    - 2 days late submission: HW will be evaluated out of 50.
    - 3 days late submission: HW will be evaluated out of 30.
    - 4 or more days late submission: HW will not be evaluated.

Good Luck!