DATA STRUCTURES – FALL 2021

LAB 13

Learning Outcomes

In this lab you are expected to learn the following:

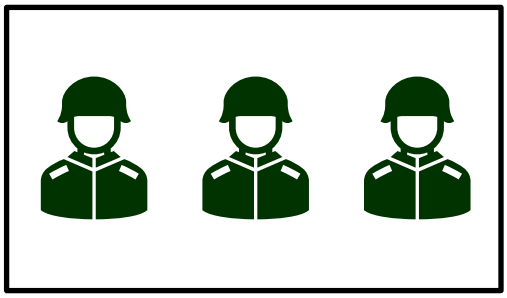
- To understand and implement the concepts of Graphs in datastructures
- Google UnitTest

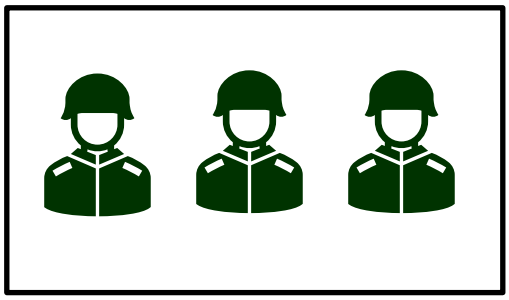
National University of Computer & Emerging Sciences (NUCES)
Islamabad,

Task 1

Troop West

Troop East





Consider 2 militant troops West and East. The analogy behind this scenario is both in a consensus wants to attack on some specific land to win. Each group consists of 1 General and 3 Captains. Every captain in a troop is assigned a specific task. Each troop is intended to notify another troop when to attack, or retreat. If a frontline captain from West troop decides to send message to the frontline captain of East troop about attack or retreat, what needs to be assured is that no other message should be transferred in the meanwhile in order to avoid any conflict in the information i.e., East troop sends a message to attack but meanwhile West troop decides to retreat, when both messages will be delivered, West will start the attack because of the message delivered to it, and the East will retreat because of the message delivered. To ensure the synchronized movement of both the troops, we need to be careful about sending message i.e., only one message should be sent from one troop and received by the other troop, no in parallel transference of message should be allowed. For this you need to design classes for Troops. Each troop will consist of Captains, a General, Message, A queue that receives messages from another troop, A queue that sends message to the other troop. A message should contain information regarding the troop it belongs to, the captain id who is intended to forward this message, the troop that is supposed to be receiving the message and the captain id who will be receiving it. Display “message sent FROM (captain id) TO (captain id)” right before removing message from outgoing queue. Display message “message received TO (captain id) FROM (captain id)” right after it has been inserted into the incoming queue.

Now design a function that is called when a Captain from a troop wants to send message to the Captain of the other troop. Get the time of the system and start counting for a second, meanwhile remove the message from the outgoing queue of troop 1 and insert it into the incoming queue of the troop 2. Since one message should be forwarded at a time, make sure no other captain from troop 2 makes a call to this function until the current message is received. This process should not take more than a second, you can use IF condition meanwhile to check if the outgoing message has been inserted into the ingoing queue of the other troop or not, if so, break the loop and resume all the operations. For this you can not use threads library, i.e. the creation of threads and handing them over the sending and receiving operations. You can use **Windows.h** or **ctime** library while working with visual studio. Design your own digital watch that uses time class that takes the current system time and work as a count down timer, meanwhile you have to make sure whether the message has been sent and received or not at the other end. For further details regarding using **Windows.h** and **ctime** library, you can visit this or this (last stack answer).

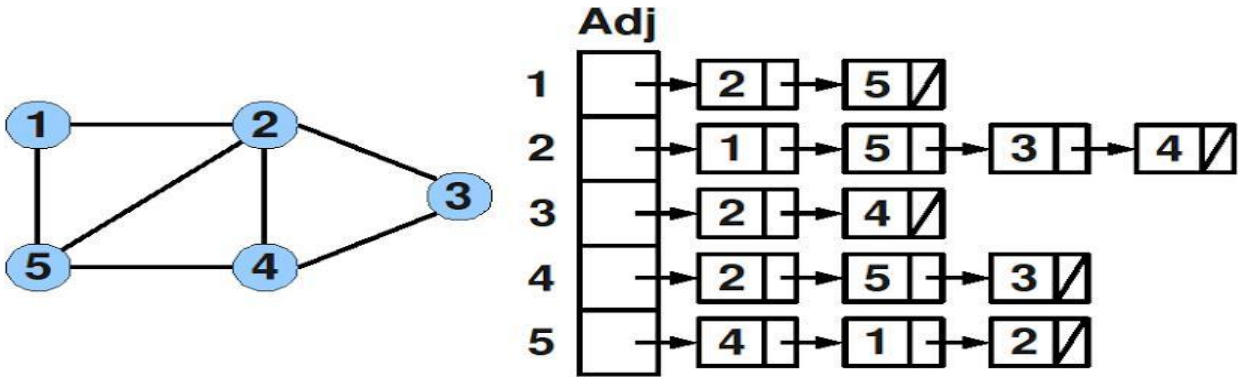
Task 2

Data Items

Each vertex in a graph has a label (of type int) that uniquely identifies it. Vertices may include additional data.

Structure

The relationship between the vertices in a graph is expressed using a set of directed/undirected Edges, where each edge connects one pair of vertices. Here is an example of undirected graph and its representation in the form of adjacency list.



Operations

Node:

```
class Node
{
 public:
    int data;
    Node * next;
};
```

Adjacency List:

```
class AdjList
{
 public:
    Node *head;
};
```

Graph (int maxVertices)
Results:
Constructor. Creates an empty graph. Allocates enough memory for a graph containing maxNumber vertices.

~ **Graph ()**
Results:
Destructor. Deallocates (frees) the memory used to store a graph.

Node createVertex (Node newVertex)
Results:
Create and return a Vertex.

void insertEdge (int src, int dest)
Requirements:
Graph includes vertices src and dest.

Results:
Inserts an undirected edge connecting vertices src and dest into a graph.

void showGraphStructure ()
Results:
Outputs a graph with the vertices in array form and the edges in adjacency list form. If the graph is empty, outputs "Empty graph". Note that this operation is intended for testing/debugging purposes only.

void BFS (int startNode)
Requirements:
Graph contains start node. You can create an array of visited nodes (size is same as size of graph).

Results:
Outputs nodes of graph in BFS.