MANIPAL INSTITUTE OF TECHNOLOGY
Manipal – 576 104

DEPARTMENT OF INFORMATION & COMMUNICATION TECHNOLOGY

CERTIFICATE

This is to certify that Ms./Mr. …………………...…………………………………… Reg. No. …..…………………… Section: …. Roll No: ………… has satisfactorily completed the lab exercises prescribed for Algorithm Lab [ICT-2263] of Fourth Semester B. Tech. (IT/CCE) Degree at MIT, Manipal, in the academic year 2019-2020.

Date: ……....................................

Signature	Signature
Faculty in Charge	Head of the Department

CONTENTS

Course Objectives

- To implement basic algorithm designing techniques.
- To understand basic approximation algorithm.
- To apply the appropriate designing technique for a given problem.

Course Outcomes

- Implement an algorithm to find path between any two vertices in the given graph.
- Apply the knowledge of shortest path algorithms for real world problems.
- Implement Greedy, Divide and Conquer, Dynamic Programming, Back tracking and Branch and Bound techniques to solve different problems.
- Implement approximation algorithm for travelling sales person and vertex cover problem.

Evaluation Plan

- Internal Assessment Marks : 60%
  - ✓ Continuous evaluation component (biweekly):10 marks
  - ✓ The assessment will depend on punctuality, program execution, maintaining the observation note and answering the questions in viva voce
- End semester assessment of 2 hour duration: 40%

INSTRUCTIONS TO THE STUDENTS

Pre- Lab Session Instructions

1. Students should carry the lab manual and the required stationery to every lab session
2. Be in time and follow the institution dress code
3. Must sign in the log register provided
4. Make sure to occupy the allotted seat and answer the attendance
5. Adhere to the rules and maintain the decorum

In- Lab Session Instructions

- Follow the instructions on the allotted exercises
- Show the program and results to the instructors on completion of experiments
- On receiving approval from the instructor, copy the program and results in the lab manual
- Prescribed textbooks and class notes can be kept ready for reference if required.

General Instructions for the exercises in Lab

- Implement the given exercise individually and not in a group.
- The programs should meet the following criteria:
    - Programs should be interactive with appropriate prompt messages, error messages if any, and descriptive messages for outputs.
    - Programs should perform input validation (data type, range error, etc.) and give appropriate error messages and suggest corrective actions.

  - Comments should be used to give the statement of the problem and every function should indicate the purpose of the function, inputs and outputs.
  - Statements within the program should be properly indented.
  - Use meaningful names for variables and functions.
  - Make use of constants and type definitions wherever needed.
- Plagiarism (copying from others) is strictly prohibited and would invite severe penalty in evaluation.
- The exercises for each week are divided under three sets:
  - Solved exercise
  - Lab exercises : to be completed during lab hours
  - Additional Exercises - to be completed outside the lab or in the lab to enhance the skill
- If a student misses a lab class then he/she must ensure that the experiment is completed during the repetition class in case of genuine reason (medical certificate approved by HOD) with the permission of the faculty concerned
- Questions for lab tests and examination are not necessarily limited to the questions in the manual, but may involve some variations and / or combinations of the questions.
- A sample note preparation is given as a model for observation.

THE STUDENTS SHOULD NOT

- Bring mobile phones or any other electronic gadgets to the lab.
- Go out of the lab without permission.

LAB NO: 1 **Date:**

SEARCHING AND SORTING

Objectives:

1. Understand Algorithm and Algorithm Design Techniques
2. Apply the technique to searching and sorting using step count method
3. Analyze the complexity for different input sizes

1. <u>Algorithm and Algorithm Analysis</u>

Algorithm

Algorithm is a sequence of unambiguous instructions for solving a problem i.e for obtaining required output for any legitimate input in a finite amount of time.

It is procedural solution to problem

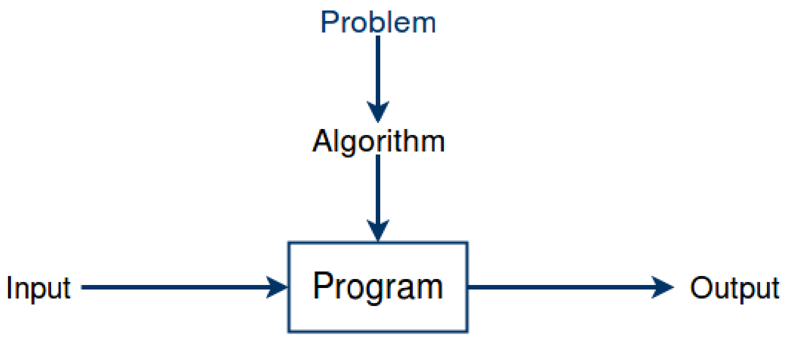


Figure 1: Notion of an Algorithm

2. Algorithm Analysis

Algorithm design technique is a general approach to solving problems algorithmically that is applicable to variety of problems from different areas of computing

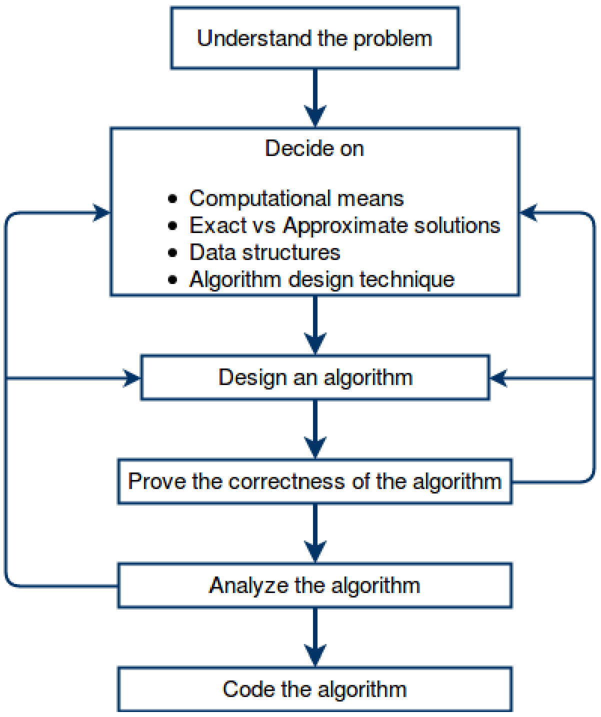


Figure 2: Algorithmic Problem Solving

3. Performance of a Program

Performance analysis of an algorithm depends upon two factors i.e. amount of memory used and amount of compute time consumed on any CPU. Formally they are notified as complexities in terms of:

- Space Complexity
- Time Complexity

Space Complexity of an algorithm is the amount of memory it needs to run to completion i.e. from start of execution to its termination. Space need by any program is the sum of following components:

Fixed Component: This is independent of the characteristics of the inputs and outputs. This part includes: Instruction Space, Space of simple variables, fixed size component variables, and constants variables.

Variable Component: This consist of the space needed by component variables whose size is dependent on the particular problems instances(Inputs/Outputs) being solved, the space needed by referenced variables and the recursion stack space is one of the most prominent components. Also this included the data structure components like Linked list, heap, trees, graphs etc.

Therefore the total space requirement of any algorithm 'A' can be provided as

Space(A) = Fixed Components(A) + Variable Components(A)

Time complexity: Time Complexity of an algorithm(basically when converted to program) is the amount of computer time it needs to run to completion. The time taken by a program is the sum of the compile time and the run/execution time . The compile time is independent of the instance(problem specific) characteristics. following factors effect the time complexity:

- Characteristics of compiler used to compile the program.
- Computer Machine on which the program is executed and physically clocked
- Multiuser execution system
- Number of program steps.

The number of steps is the most prominent instance characteristics. The number of steps any program statement is assigned depends on the kind of statement like comments count as zero steps, an assignment statement which does not involve any calls to other algorithm is counted as one step, for iterative statements the steps count is considered only for the control part of the statement etc.

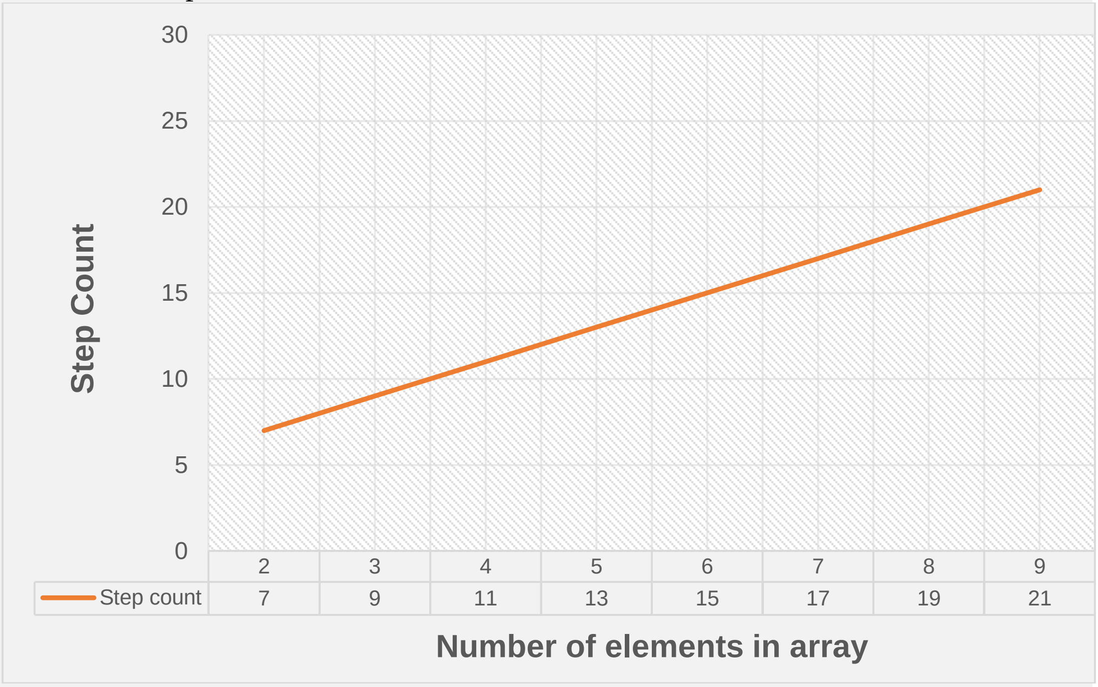
Therefore to calculate total number of program steps we use following procedure. For this we build a table in which we list the total number of steps contributed by each statement. This is arrived at by first determining the number of steps per execution of the statement and the frequency of each statement executed. This procedure is explained using an Example1.

Solved Exercises:

1. **To print step count of sum function (to add elements of an array) and plot number of elements and step counts**

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

int sum(int arr[], int n)
{
    int i,s=0;
    count++;
    for (i = 0; i < n; i++)
    {
        count++;
        count++;
        s=s+arr[i];

    }
    count++;
    count++;
    return s;
}

int main(void)
{
    int arr[25], n, x;
    count=0;
    cout<<"enter no. of elements";
    cin>>n;
    cout<<"enter "<< n <<" elements";
    for(int i=0;i<n;i++)
    cin>>arr[i];
    int result = sum(arr, n);
    cout<<"Sum of elements "<<result;
    cout<<endl;
    cout<<"Number of steps for sum function "<<count;
   return 0;
}
enter no. of elements 5
enter 5 elements1 2 3 4 5
Sum of elements 15
```

```
Number of steps for sum function 13

enter no. of elements 8
enter 8 elements 11 12 33 21 43 22 43 67
Sum of elements 252
Number of steps for sum function 19

Number of steps : 2n+3
```



	2	3	4	5	6	7	8	9
Step count	7	9	11	13	15	17	19	21

2. To print the time taken by the sum function

```
#include <iostream>
#include <chrono>
using namespace std;
using namespace std::chrono;
int count;
int sum(int arr[], int n)
{

    int i,s=0;
    count++;
    for (i = 0; i < n; i++)
    {
        count++;
        count++;
        s=s+arr[i];

    }
    count++;
    count++;

    return s;
}

int main(void)
{
    int arr[1000], n, x;
    count=0;
    cout<<"enter no. of elements";
    cin>>n;
    //cout<<"enter "<< n <<" elements";
    for(int i=0;i<n;i++)
    arr[i]=i;
    auto start = high_resolution_clock::now();
    int result = sum(arr, n);
    auto stop = high_resolution_clock::now();
    cout<<"Sum of elements "<<result;
    cout<<endl;
    auto duration = duration_cast<microseconds>(stop - start);
    cout << "time taken to run sum fucntion:  "<<duration.count() << endl;
    cout<<"Number of steps for sum function "<<count << microseconds;
   return 0;
}
```

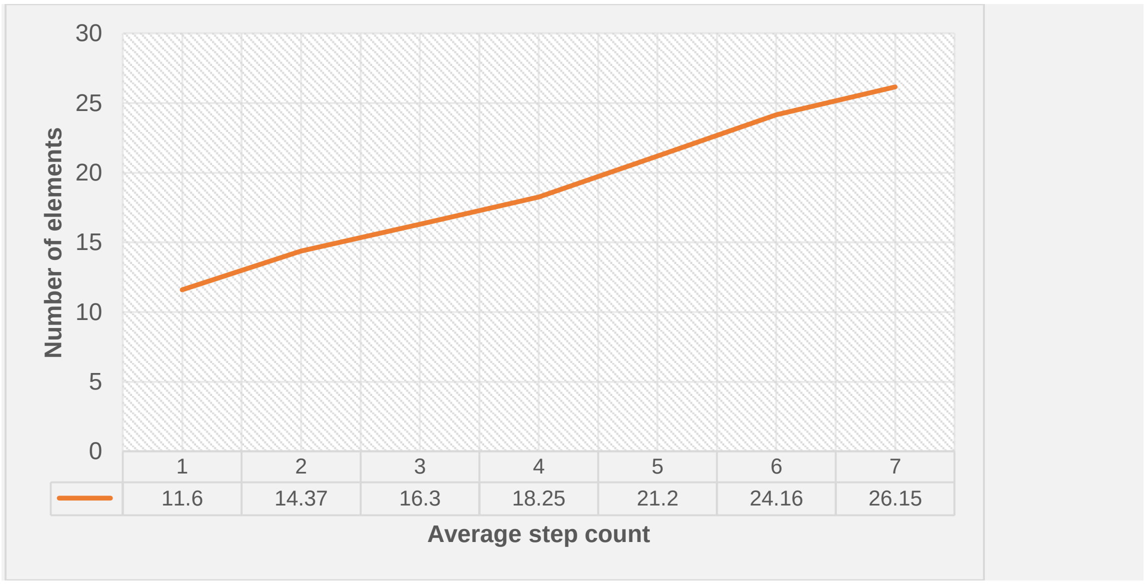
Output

```
enter no. of elements500
Sum of elements 124750
time taken to run sum function:  2 microseconds
Number of steps for sum function 1003
```

3. To print step counts of linear search function and plot average step counts

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

int search(int arr[], int n, int x)
{
    int i;
    count++;
    for (i = 0; i < n; i++)
    {
        count++;
        count++;
        if (arr[i] == x)
          {
           count++;
            return i;
           }
     }
    count++;
    count++;
    return -1;
}

int main(void)
{
    int arr[25], n, x;
    count=0;
    cout<<"enter no. of elements";
    cin>>n;
    cout<<"enter "<< n <<" elements";
    for(int i=0;i<n;i++)
    cin>>arr[i];
    cout<<"enter the element to be searched";
    cin>> x;
    int result = search(arr, n, x);
   (result == -1)? cout<<"Element is not present in array"
                 : cout<<"Element is present at index " <<result;
    cout<<endl;
    cout<<"Number of seteps for search function "<<count;
   return 0;
}
enter no. of elements 6
enter 6 elements 1 3 2 5 4 6
enter the element to be searched 3
Element is present at index 1
Number of steps for search function 6

enter no. of elements 6
enter 6 elements 1 3  2 5 4 6
enter the element to be searched 4
Element is present at index 4
Number of steps for search function 12

enter no. of elements 6
enter 6 elements 1 3 2 5 4 6
enter the element to be searched 10
Element is not present in array
Number of steps for search function 15

enter no. of elements 6
enter 6 elements 1 3 2 5 4 6
enter the element to be searched56
Element is not present in array
Number of steps for search function 15
```

Number of steps :
If lement found in the ith position : 2(i)+2
If element not found : 2n+3
Hence total number of intsnces : n+1
Average case complexity = $\frac{\sum_{i=1}^{n}(2i+2)+2n+3}{n+1}$



	1	2	3	4	5	6	7
—	11.6	14.37	16.3	18.25	21.2	24.16	26.15

Lab Exercises

Write a Program to perform the following and find the time complexity using step count method

1. Binary Search (both iterative and recursive)
2. Bubble Sort
3. Selection Sort
4. Insertion Sort

Additional Exercises

1. Find the substring in a given string without using String handling functions.
2. Implement rank sort algorithm.

LAB NO: 2 **Date:**

GRAPHS-I

Objectives:

1. Understand the graph representations
2. Understand Breadth First Search (BFS) and Depth-First Search (DFS), graph traversal algorithms.
3. Apply the BFS and DFS to find sequence and analyze the complexities based on the graph representaions

1. Graph Representations

Graph is a data structure that consists of following two components:

1. A finite set of vertices also called as nodes.

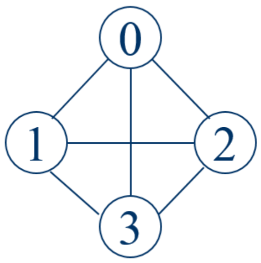
2. A finite set of ordered pair of the form (u, v) called as edge. The pair is ordered because (u, v) is not same as (v, u) in case of a directed graph(di-graph). The pair of the form (u, v) indicates that there is an edge from vertex u to vertex v. The edges may contain weight/value/cost.

Following two are the most commonly used representations of a graph.

1. Adjacency Matrix

2. Adjacency List

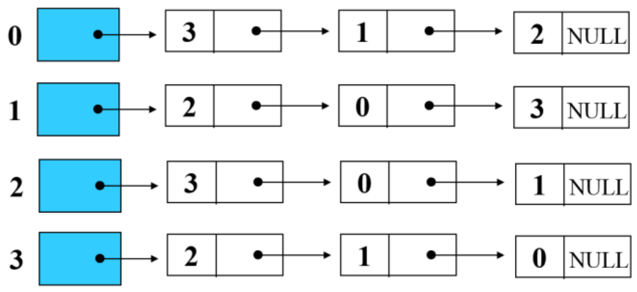
Adjacency Matrix:



Adjacency Matrix is a 2D array of size V x V where V is the number of vertices in a graph. Let the 2D array be adj[][], a slot adj[i][j] = 1 indicates that there is an edge from vertex i to vertex j. Adjacency matrix for undirected graph is always symmetric. Adjacency Matrix is also used to represent weighted graphs. If adj[i][j] = w, then there is an edge from vertex i to vertex j with weight w.

0	1	1	1
1	0	1	1
1	1	0	1
1	1	1	0

Adjacency List:
An array of lists is used. Size of the array is equal to the number of vertices. Let the array be array[]. An entry array[i] represents the list of vertices adjacent to the ***i***th vertex. This representation can also be used to represent a weighted graph. The weights of edges can be represented as lists of pairs. Following is adjacency list representation of the above graph.



2. Breadth First Search

.B x.readth-first search (BFS) is an algorithm for traversing or searching tree or graph data structures. It starts at the tree root (or some arbitrary node of a graph, sometimes referred to as a 'search key') and explores the neighbor nodes first, before moving to the next level neighbors.

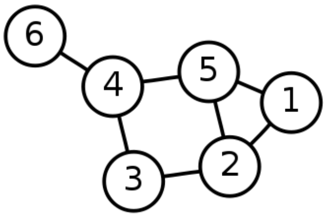
Input: A graph and a starting vertex root of Graph

Output: All vertices reachable from root labeled as explored.

Example:

BFS will visit the sibling vertices before the child vertices using this algorithm:

```
Mark the starting node of the graph as visited and enqueue it into the queue
While the queue is not empty
        Dequeue the next node from the queue to become the current node
        While there is an unvisited child of the current node
            Mark the child as visited and enqueue the child node into the queue
```



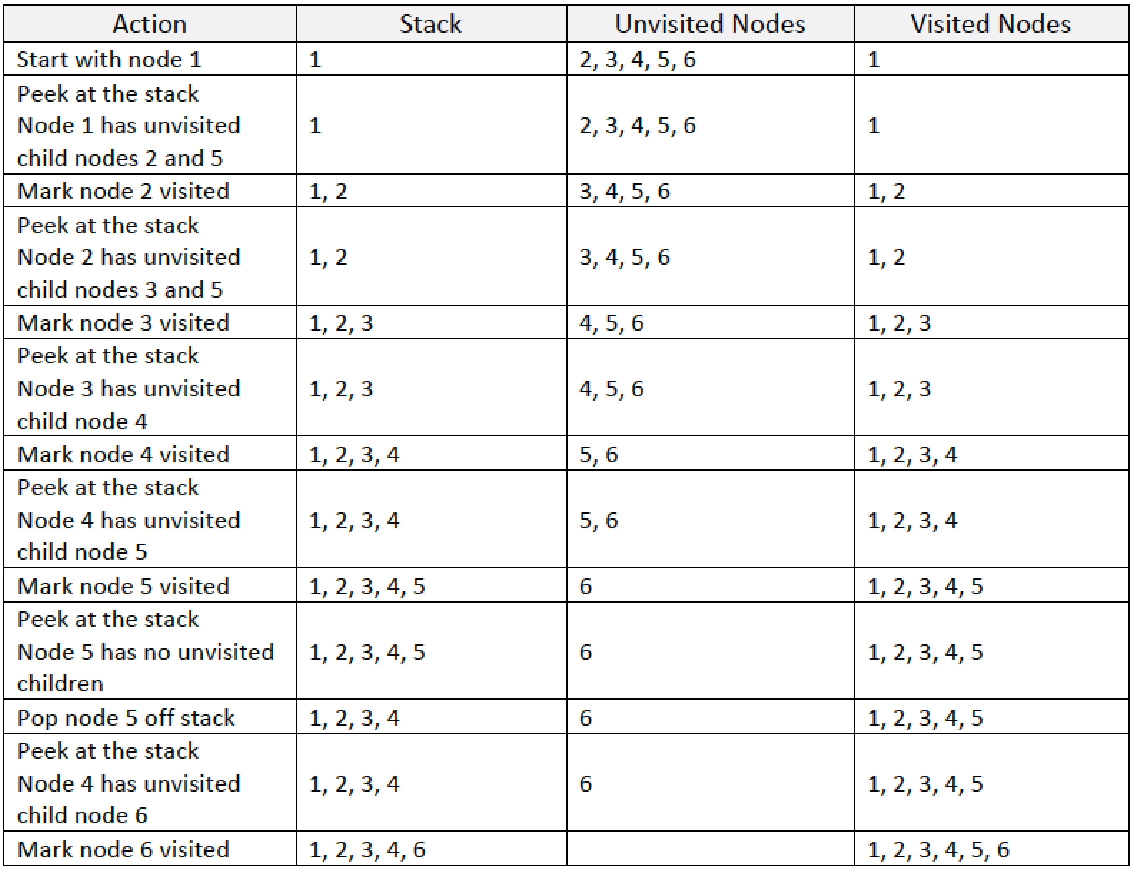
The queue operation enqueue adds to the left and dequeue removes from the right.

Action	Current Node	Queue	Unvisited Nodes	Visited Nodes
Start with node 1		1	2, 3, 4, 5, 6	1
Dequeue node 1	1		2, 3, 4, 5, 6	1
Node 1 has unvisited children nodes 2 and 5	1		2, 3, 4, 5, 6	1
Mark 2 as visited and enqueue into queue	1	2	3, 4, 5, 6	1, 2
Mark 5 as visited and enqueue into queue	1	5, 2	3, 4, 6	1, 2, 5
Node 1 has no more unvisited children, dequeue a new current node 2	2	5	3, 4, 6	1, 2, 5
Mark 3 as visited and enqueue into queue	2	3, 5	4, 6	1, 2, 5, 3
Node 2 has no more unvisited children, dequeue a new current node 5	5	3	4, 6	1, 2, 5, 3
Mark 4 as visited and enqueue into queue	5	4, 3	6	1, 2, 5, 3, 4
Node 5 has no more unvisited children, dequeue a new current node 3	3	4	6	1, 2, 5, 3, 4
Node 3 has no more unvisited children, dequeue a new current node 4	4		6	1, 2, 5, 3, 4
Mark 6 as visited and enqueue into queue	4	6		1, 2, 5, 3, 4, 6

2. Depth-First Search :

Considering a given node as the parent and connected nodes as children, DFS will visit the child vertices before visiting siblings using this algorithm.

```
Mark the starting node of the graph as visited and push it onto the stack
While the stack is not empty
Peek at top node on the stack (look at the top element on the stack, but do not remove it)
If there is an unvisited child of that node
        Mark the child as visited and push the child node onto the stack
Else
        Pop the top node off the stack
```



Action	Stack	Unvisited Nodes	Visited Nodes
Start with node 1	1	2, 3, 4, 5, 6	1
Peek at the stack Node 1 has unvisited child nodes 2 and 5	1	2, 3, 4, 5, 6	1
Mark node 2 visited	1, 2	3, 4, 5, 6	1, 2
Peek at the stack Node 2 has unvisited child nodes 3 and 5	1, 2	3, 4, 5, 6	1, 2
Mark node 3 visited	1, 2, 3	4, 5, 6	1, 2, 3
Peek at the stack Node 3 has unvisited child node 4	1, 2, 3	4, 5, 6	1, 2, 3
Mark node 4 visited	1, 2, 3, 4	5, 6	1, 2, 3, 4
Peek at the stack Node 4 has unvisited child node 5	1, 2, 3, 4	5, 6	1, 2, 3, 4
Mark node 5 visited	1, 2, 3, 4, 5	6	1, 2, 3, 4, 5
Peek at the stack Node 5 has no unvisited children	1, 2, 3, 4, 5	6	1, 2, 3, 4, 5
Pop node 5 off stack	1, 2, 3, 4	6	1, 2, 3, 4, 5
Peek at the stack Node 4 has unvisited child node 6	1, 2, 3, 4	6	1, 2, 3, 4, 5
Mark node 6 visited	1, 2, 3, 4, 6		1, 2, 3, 4, 5, 6

Solved Exercise: Representing graph using adjacency matrix and printing indegree of vertices

```
#include <iostream>
using namespace std;
int count;
int indegree(int arr[][10], int p, int n)
{
    int c1=0;

    for (int i = 1; i <= n; i++)
    {
        if(arr[i][p]==1)
        c1++;
    }

    return c1;
}

int main(void)
{
    int A[10][10], n, m,x;
    count=0;
    cout<<"enter no. of vertices";
    cin>>n;
    cout<<"enter number of edges";
    cin>>m;
    for(int i=1;i<=n;i++)
    for(int j=1;j<=n;j++)
    A[i][j]=0;
    int p,q;

    for(int i=1;i<=m;i++)
    {
       cout<<"enter Source";
       cin>> p;
       cout<<"enter destination ";
       cin>>q;
       A[p][q]=1;
    }

    for(p=1;p<=n;p++)
```

```
   {
   int result = indegree(A,p,n);
   cout<<endl;
   cout<<"Indegree of "<< p <<" is:" << result;
    }
   return 0;
}

Output
enter no. of vertices4
enter number of edges5
enter Source 1
enter destination  3
enter Source1
enter destination 4
enter Source3
enter destination 4
enter Source4
enter destination 2
enter Source2
enter destination 1

Indegree of 1 is:1
Indegree of 2 is:1
Indegree of 3 is:1
Indegree of 4 is:2
```

Lab Exercises

Write Programs to perform the following and find the time complexity using step count method:

1. Depth First Search (DFS)
2. Breadth First Search (BFS)
3. To find mother vertex in a graph
4. To find transpose of a given graph

Additional Exercises

Write a Program to input a graph from the user (nodes should be given numbers 1, 2,.. upto n) and perform the following:

1. Display the nodes with even number using BFS.
2. Display the nodes with odd number using DFS.

LAB NO: 3 **Date:**

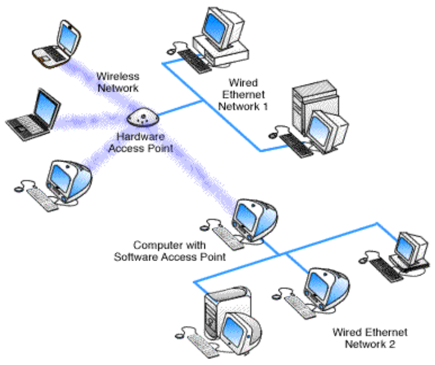
GRAPHS-II

Objectives:

1. Apply graph traversal algorithms to find the path, check the graph connected or not etc.
2. Analyze the time complexity.

1. Applications

In a network of computers it is possible to find whether one machine is communicating with another machine or not. The network of computers can be represented by a graph, with each vertex representing one of the machines in the network and each edge representing a communication wire between two machines. Either a BFS or a DFS can be used to determine whether a path exists between two vertices any two computers 'u' and 'v'.



Lab Exercises

Write a Program to perform the following and find the time complexity using step count method:

1. Finding a path in the graph
2. Finding a cycle in the graph
3. Check whether the given graph is connected or not.

Additional Exercises

1. Write a program to find the components of the given graph also analyze the complexity.
2. Write a program to find the strongly Connected Components of a graph also analyze the complexity.