Laboratory 9: Graphs

CSC205A Data structures and Algorithms Laboratory B. Tech. 2015

Vaishali R Kulkarni

Department of Computer Science and Engineering
Faculty of Engineering and Technology
M. S. Ramaiah University of Applied Sciences
Email: vaishali.cs.et@msruas.ac.in
Tel: +91-80-4906-5555 (2212) WWW: www.msruas.ac.in

Introduction and Purpose of Experiment

- Graphs are important data structures used in many applications like network modelling applications, routing, traffic, water supply etc.
- This experiment introduces graphs and its representation.

Aim and objectives

Aim:

- To design and develop C program to represent a Graph using adjacency list and adjacency matrix

Objectives:

At the end of this lab, the student will be able to

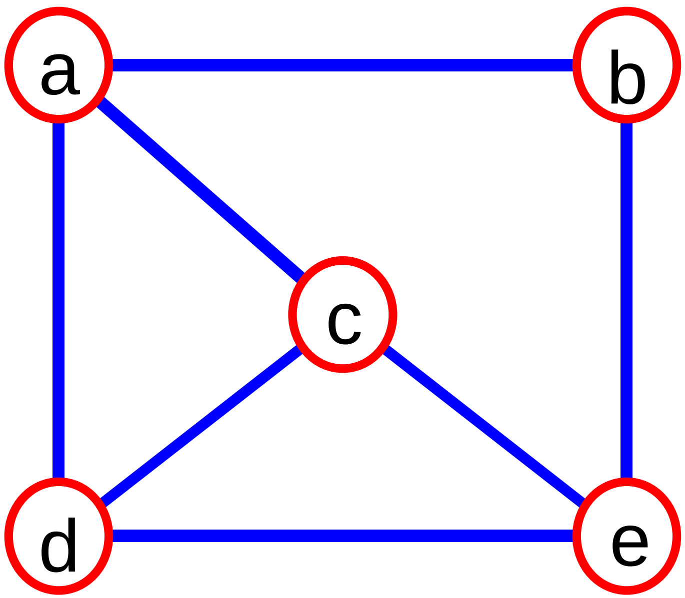
- Represent a graph data structure using adjacent list and adjacency list
- Compare and contrast between both the approaches.

What is a Graph?

- A graph G = (V,E) is composed of:

  V: set of vertices

  E: set of edges connecting the vertices in V
- An edge e = (u,v) is a pair of vertices
- Example:



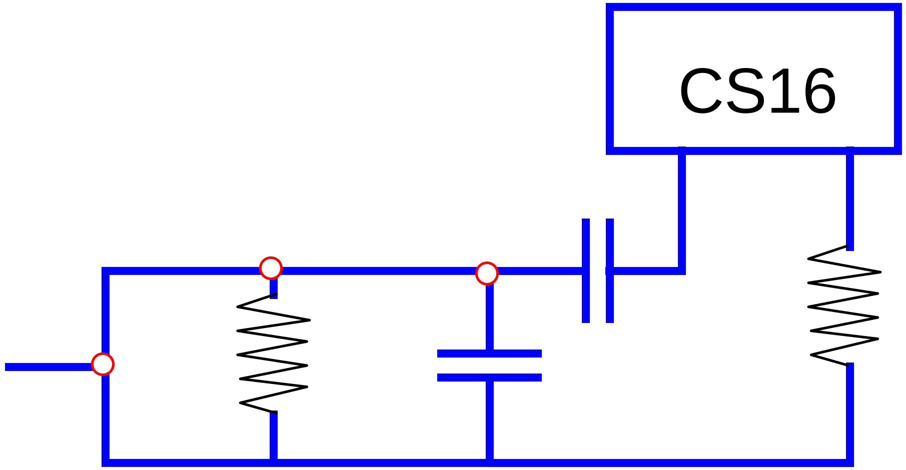
V= {a,b,c,d,e}

E= {(a,b),(a,c),(a,d),
(b,e),(c,d),(c,e),
(d,e)}

Some real life applications

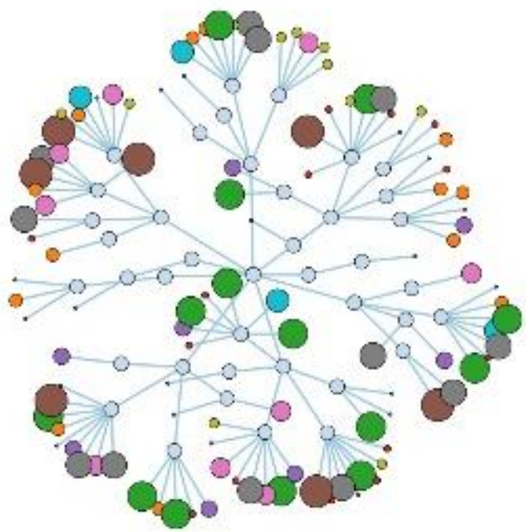
Applications

- Electronic circuits



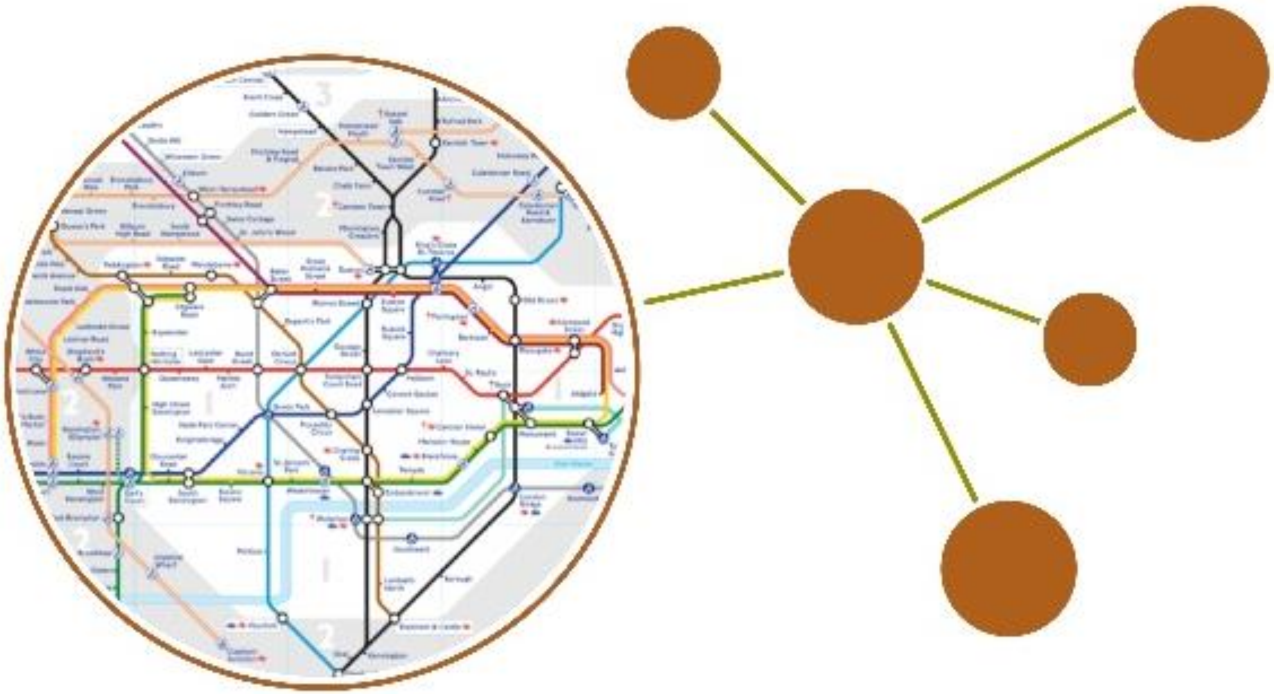
Graph Databases

- Store
- Manage
- Query

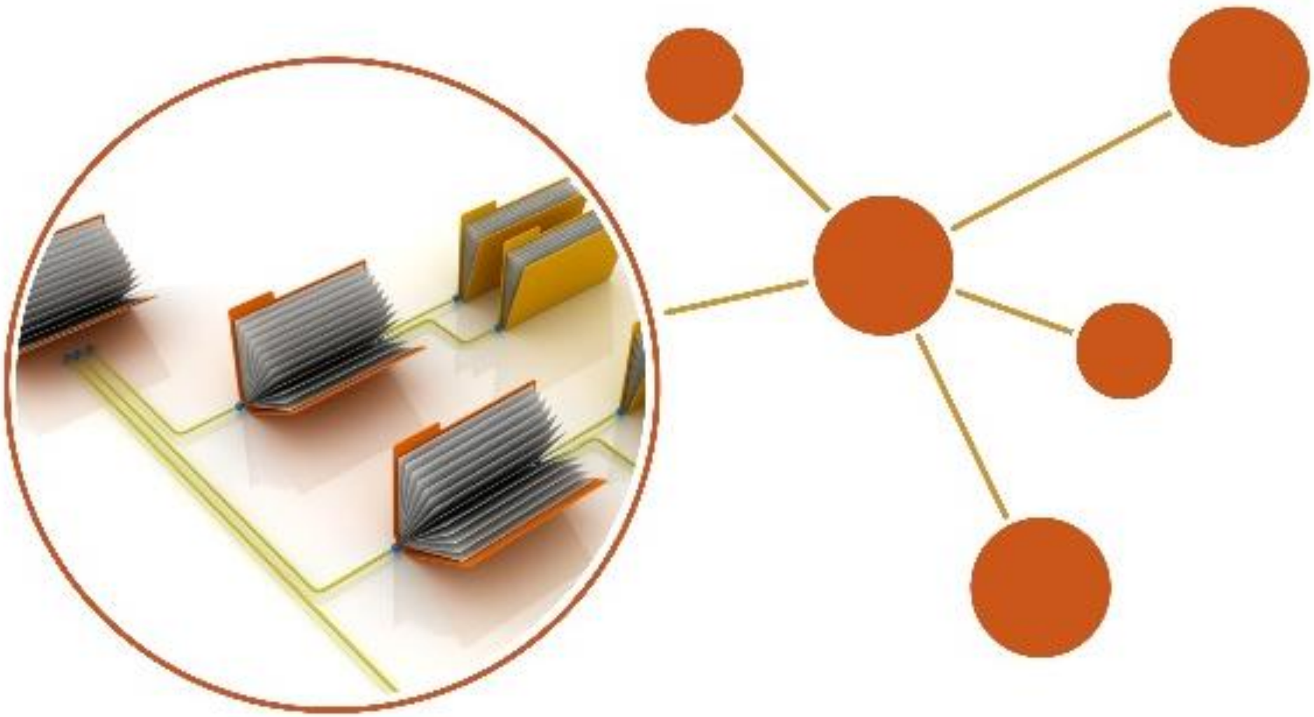


Social Network

Route Finding

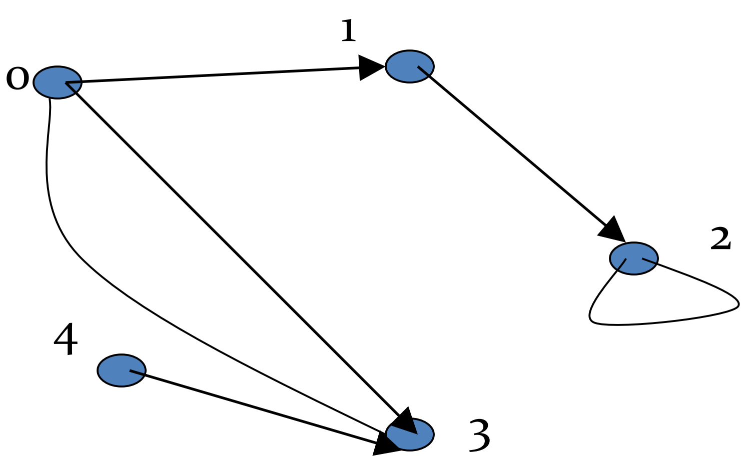


Access Control



Examples of Graphs

- V={0,1,2,3,4}
- E={(0,1), (1,2), (0,3), (3,0), (2,2), (4,3)}

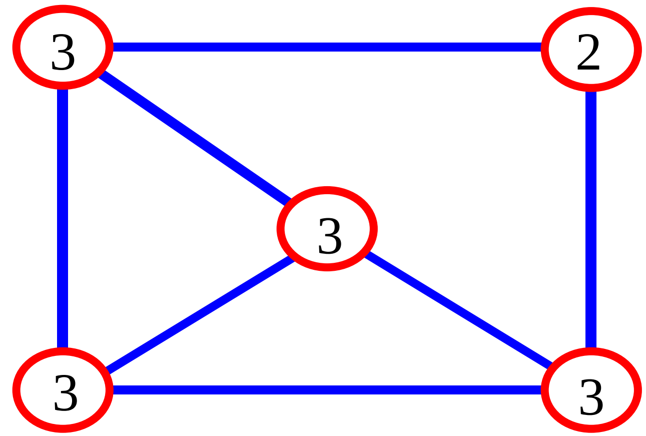


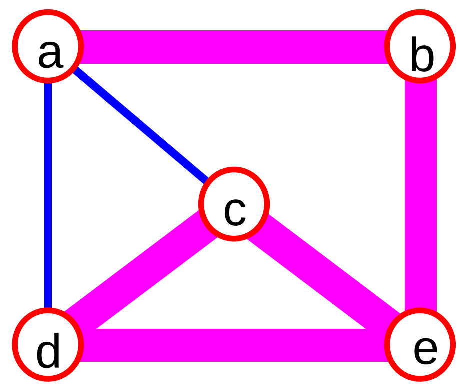
When (x,y) is an edge,
we say that x is *adjacent to* y, and y is *adjacent from* x.

0 is adjacent to 1.
1 is not adjacent to 0.
2 is adjacent from 1.

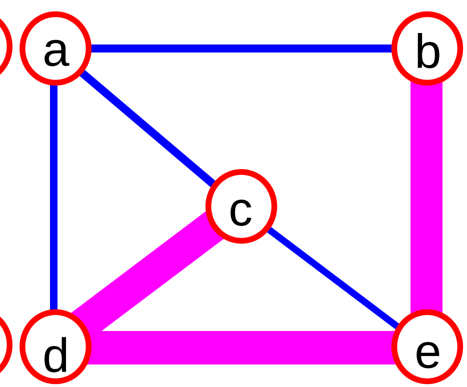
Terminology: Path

- Path: sequence of vertices $v_1, v_2, \ldots v_k$ such that consecutive vertices v_i and v_{i+1} are adjacent.





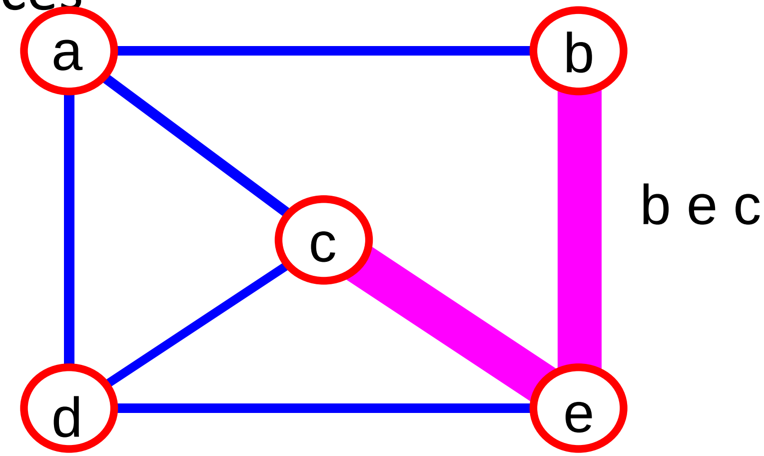
a b e d c



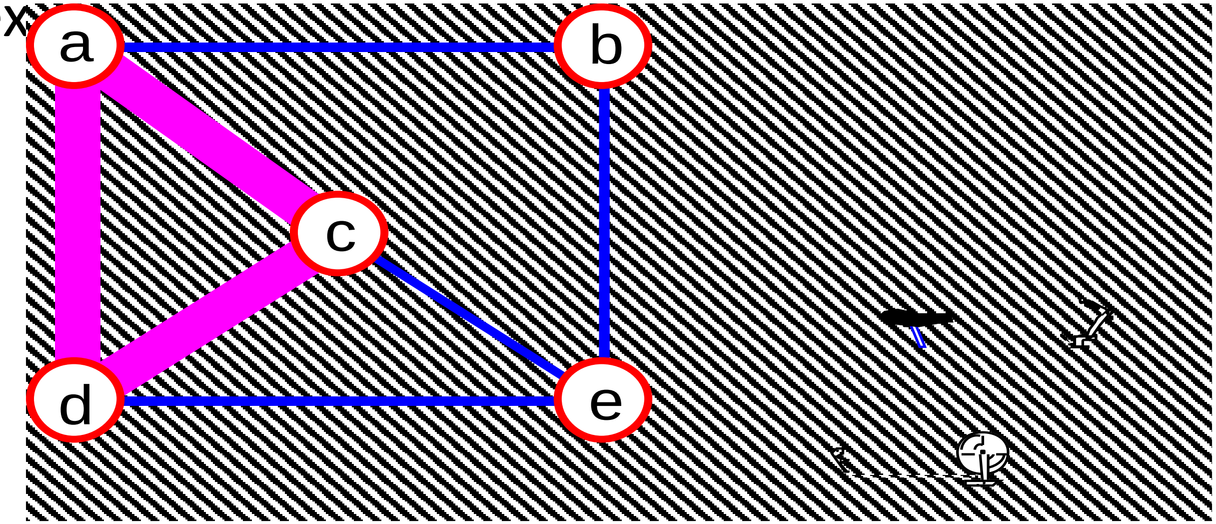
b e d c

More Terminology

- Simple path: no repeated vertices

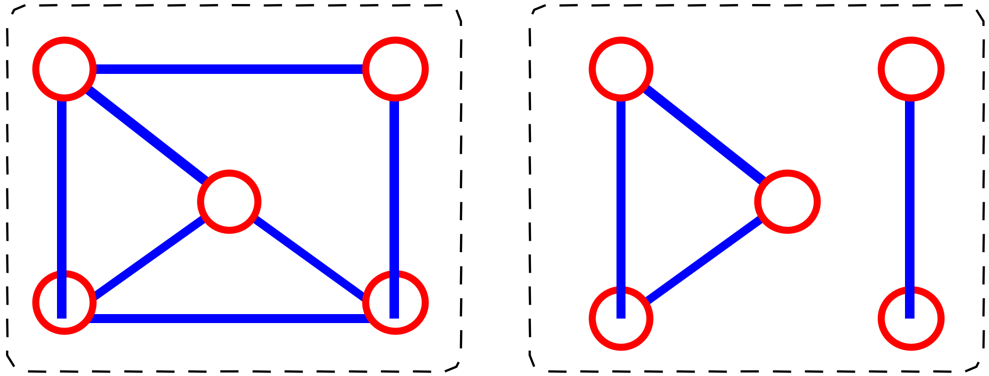


- Cycle: simple path, except that the last vertex is the same as the first vertex



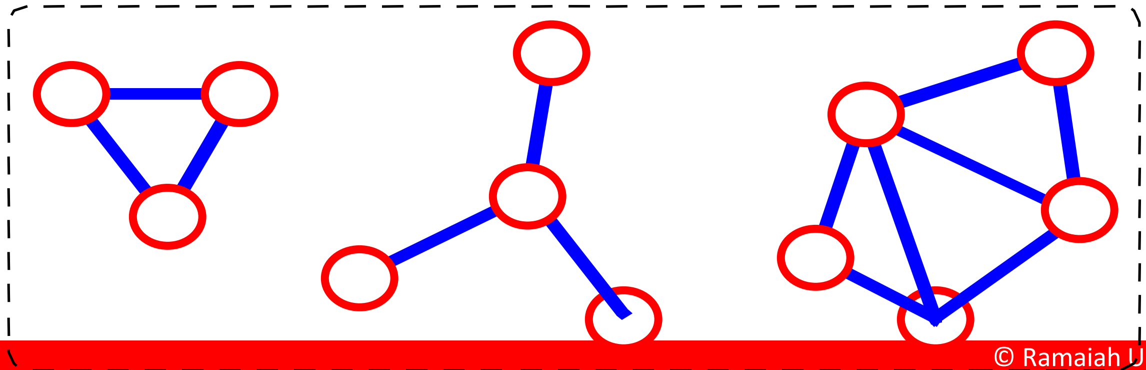
Even More Terminology

- Connected graph: any two vertices are connected by some path

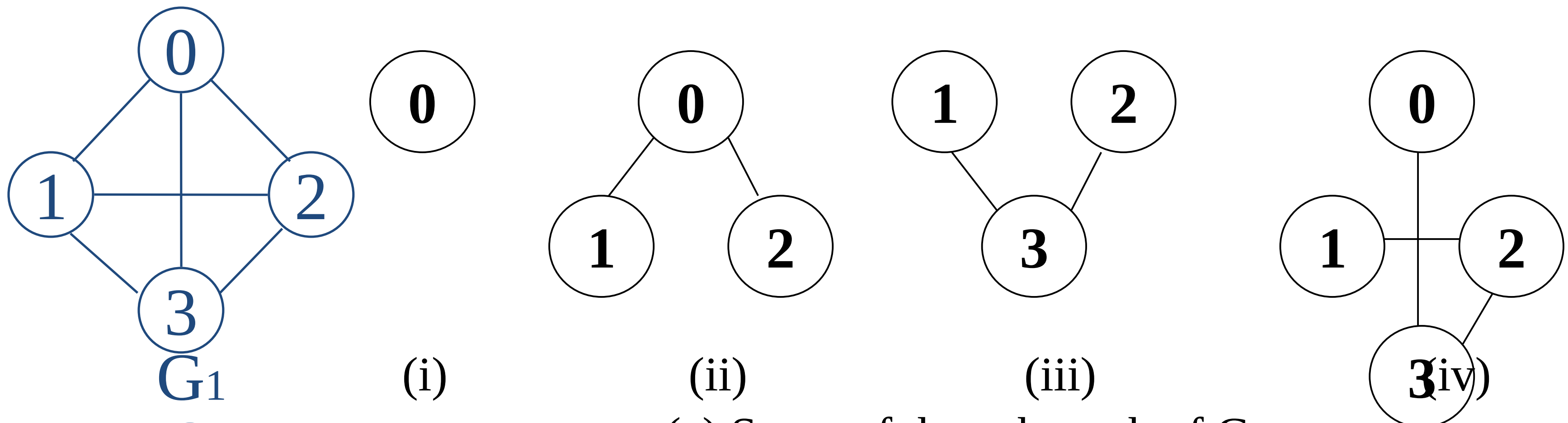


connected   not connected

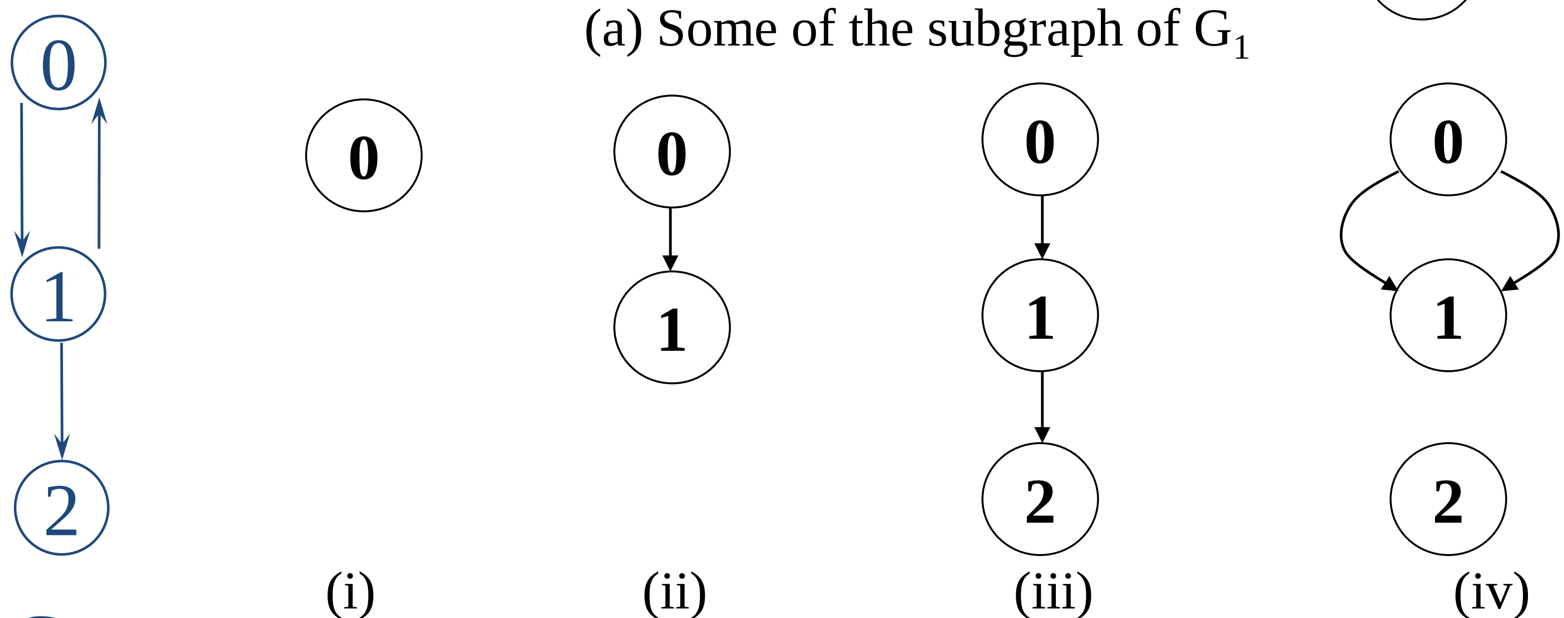
- Subgraph: subset of vertices and edges forming a graph
- Connected component: maximal connected subgraph. E.g., the graph below has 3 connected components.



Subgraphs Examples



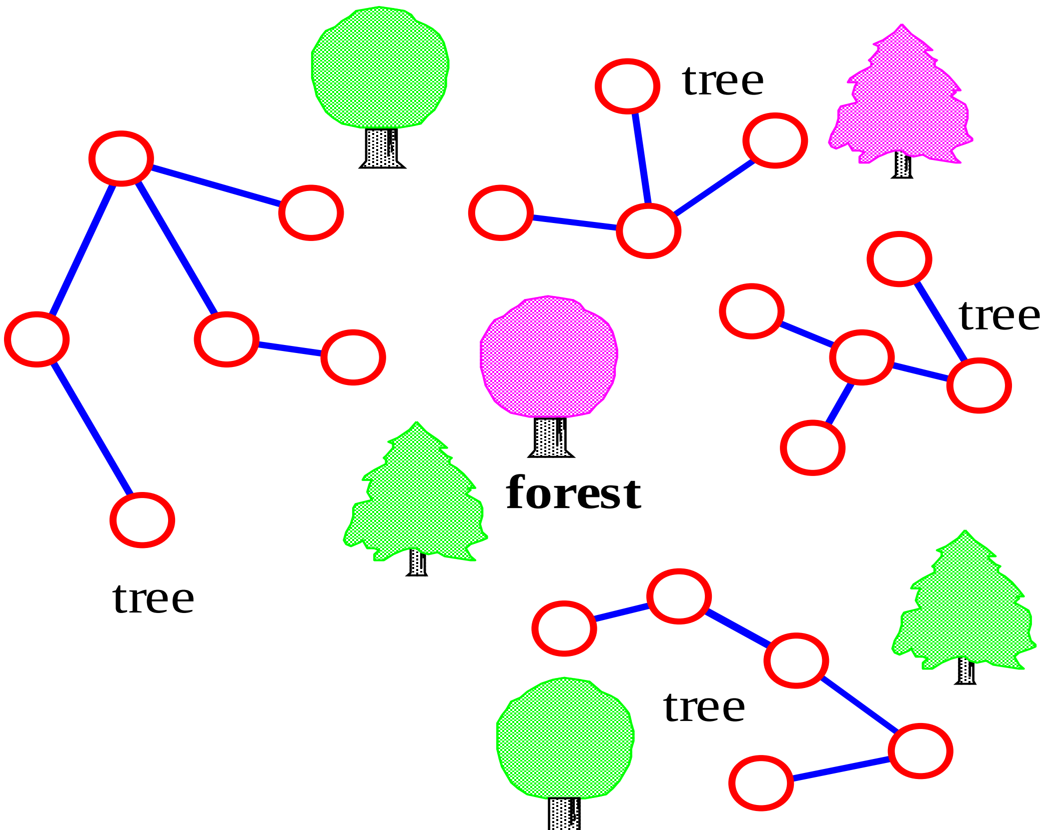
(a) Some of the subgraph of G_1



(b) Some of the subgraph of G_3

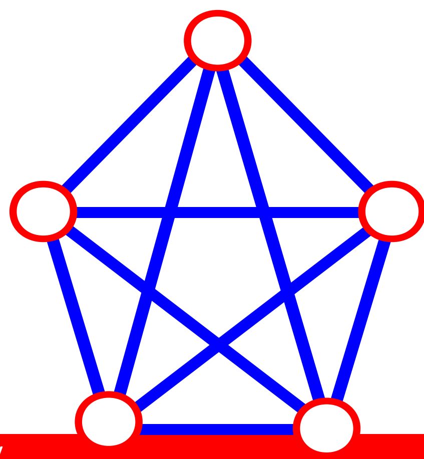
More…

- tree - connected graph without cycles
- forest - collection of trees



Connectivity

- Let **n** = #vertices, and **m** = #edges
- **A complete graph**: one in which all pairs of vertices are adjacent
- *How many total edges in a complete graph?*
  - Each of the n vertices is incident to **n**-1 edges, however, we would have counted each edge twice! Therefore, intuitively, m = **n**(**n** -1)/2.
- Therefore, if a graph is not complete, m < **n**(**n** -1)/2



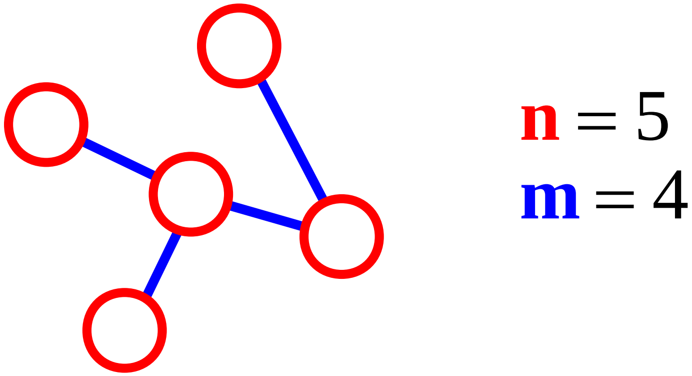
$\mathbf{n} = 5$
$\mathbf{m} = (5 * 4)/2 = 10$

More Connectivity

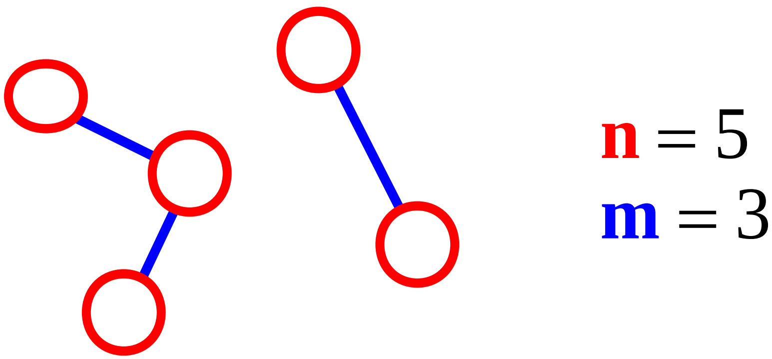
n = #vertices

m = #edges

- For a tree **m** = **n** - 1

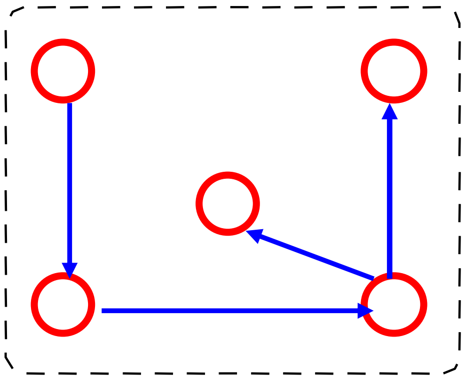


If **m** < **n** - 1, G is not connected



Oriented (Directed) Graph

- A graph where edges are directed



Directed vs. Undirected Graph

- An undirected graph is one in which the pair of vertices in a edge is unordered, $(v_0, v_1) = (v_1,v_0)$
- A directed graph is one in which each edge is a directed pair of vertices, $<v_0, v_1> != <v_1,v_0>$

Graph Representation

- For graphs to be computationally useful, they have to be conveniently represented in programs
- There are two computer representations of graphs:
  - Adjacency matrix representation
  - Adjacency lists representation

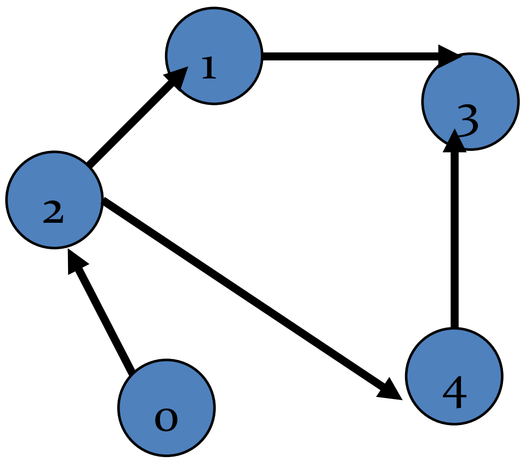
Graph Representations

- For graphs to be computationally useful, they have to be conveniently represented in programs
- There are two computer representations of graphs:
  - Adjacency Matrix
  - Adjacency Lists

Adjacency Matrix

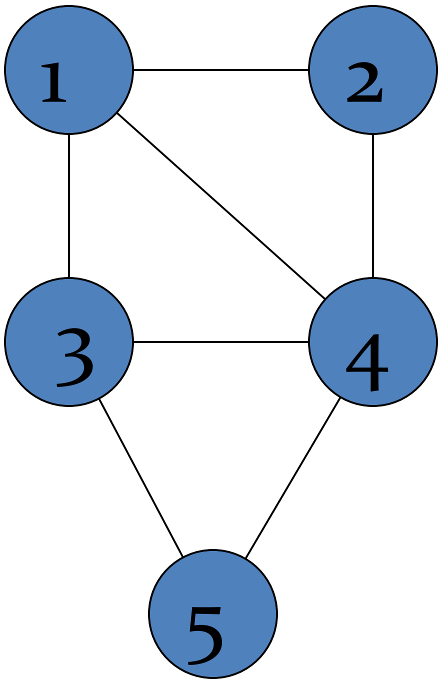
- A square grid of boolean values
- If the graph contains N vertices, then the grid contains N rows and N columns
- For two vertices numbered I and J, the element at row I and column J is true if there is an edge from I to J, otherwise false

Adjacency Matrix



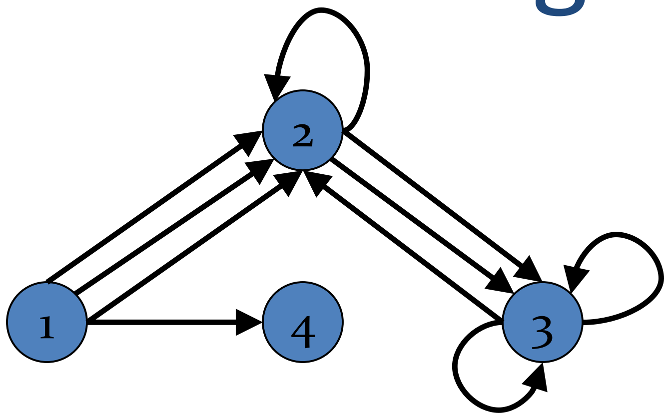
$$
\begin{array}{c c}
 & \begin{array}{ccccc} 0 & 1 & 2 & 3 & 4 \end{array} \\
\begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{array} &
\left(\begin{array}{ccccc}
\text{false} & \text{false} & \text{true} & \text{false} & \text{false} \\
\text{false} & \text{false} & \text{false} & \text{true} & \text{false} \\
\text{false} & \text{true} & \text{false} & \text{false} & \text{true} \\
\text{false} & \text{false} & \text{false} & \text{false} & \text{false} \\
\text{false} & \text{false} & \text{false} & \text{true} & \text{false}
\end{array}\right)
\end{array}
$$

Adjacency Matrix



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

Adjacency Matrix -Directed Multigraphs



A:

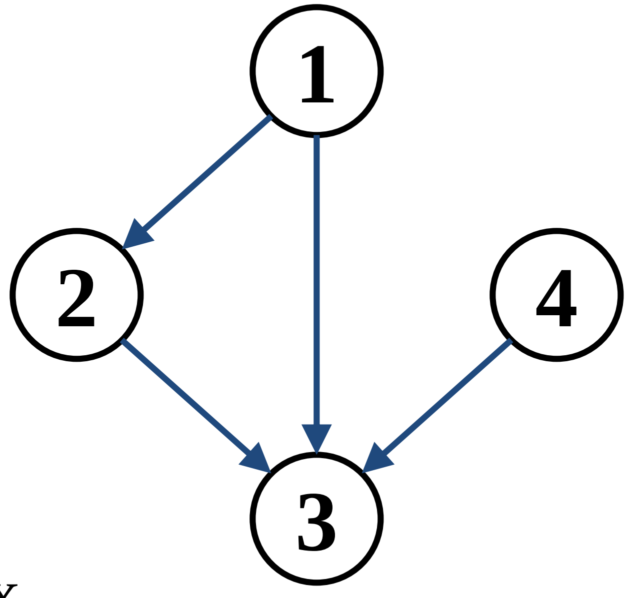
$$\begin{pmatrix} 0 & 3 & 0 & 1 \\ 0 & 1 & 2 & 0 \\ 0 & 1 & 2 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Adjacency Lists Representation

- A graph of n nodes is represented by a one-dimensional array L of linked lists, where
  - L[i] is the linked list containing all the nodes adjacent from node i.
  - The nodes in the list L[i] are in no particular order

Graphs: Adjacency List

- Adjacency list: for each vertex $v \in V$, store a list of vertices adjacent to v
- Example:
  - Adj[1] = {2,3}
  - Adj[2] = {3}
  - Adj[3] = {}
  - Adj[4] = {3}
- Variation: can also keep a list of edges coming *into* vertex



Graphs: Adjacency List

- How much storage is required?
  - The *degree* of a vertex v = # incident edges
    - Directed graphs have in-degree, out-degree
  - For directed graphs, # of items in adjacency lists is

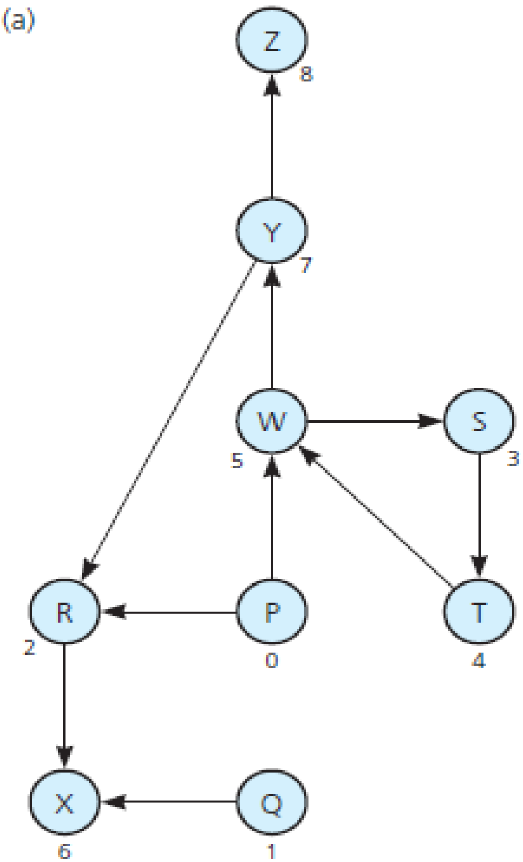
    $$\Sigma \text{ out-degree}(v) = |E|$$

    For undirected graphs, # items in adjacency lists is

    $$\Sigma \text{ degree}(v) = 2\,|E|$$
- So: Adjacency lists take O(V+E) storage

Implementing Graphs

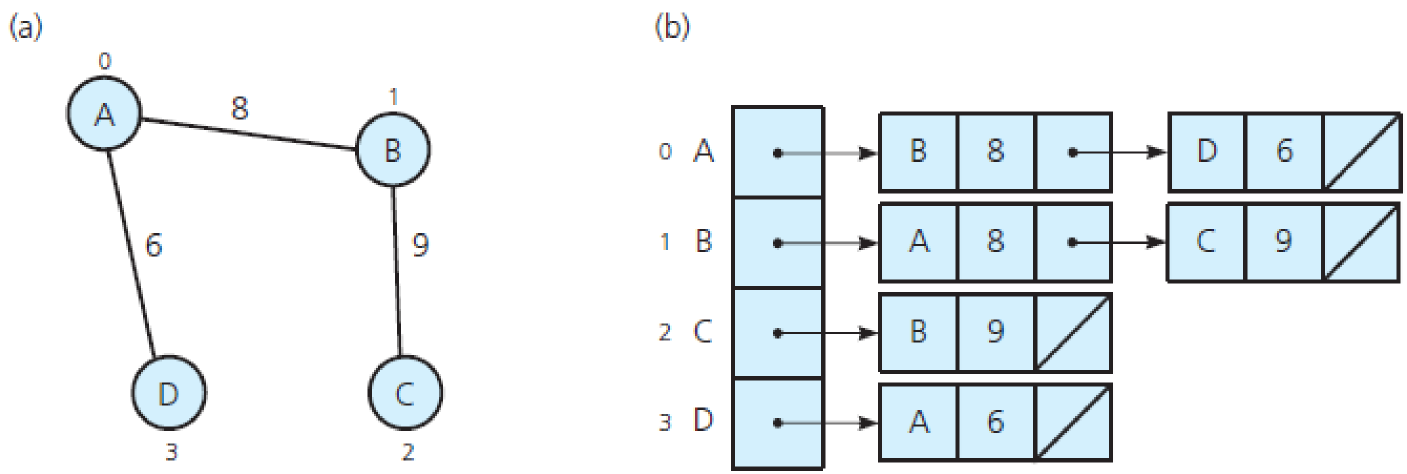
(a)



(b)

		0	1	2	3	4	5	6	7	8
		P	Q	R	S	T	W	X	Y	Z
0	P	0	0	1	0	0	1	0	0	0
1	Q	0	0	0	0	0	0	1	0	0
2	R	0	0	0	0	0	0	1	0	0
3	S	0	0	0	0	1	0	0	0	0
4	T	0	0	0	0	0	1	0	0	0
5	W	0	0	0	1	0	0	0	1	0
6	X	0	0	0	0	0	0	0	0	0
7	Y	0	0	1	0	0	0	0	0	1
8	Z	0	0	0	0	0	0	0	0	0

Implementing Graphs



- (a) A weighted undirected graph and
  (b) its adjacency list

Data structure for Adjacency list

```
// A structure to represent an adjacency list node
struct AdjListNode
{
    int dest;
    struct AdjListNode* next;
};
 // A structure to represent an adjacency list
struct AdjList
{
    struct AdjListNode *head;  // pointer to head node of list
};
// A structure to represent a graph. A graph is an array of adjacency lists.
// Size of array will be V (number of vertices in graph)
struct Graph
{
    int V;
    struct AdjList* array;
};
```

```c
// A utility function to create a new adjacency list node
struct AdjListNode* newAdjListNode(int dest)
{
    struct AdjListNode* newNode =
            (struct AdjListNode*) malloc(sizeof(struct AdjListNode));
    newNode->dest = dest;
    newNode->next = NULL;
    return newNode;
}

// A utility function that creates a graph of V vertices
struct Graph* createGraph(int V)
{
    struct Graph* graph = (struct Graph*) malloc(sizeof(struct Graph));
    graph->V = V;
     // Create an array of adjacency lists.  Size of array will be V
    graph->array = (struct AdjList*) malloc(V * sizeof(struct AdjList));
      // Initialize each adjacency list as empty by making head as NULL
    int i;
    for (i = 0; i < V; ++i)
        graph->array[i].head = NULL;
     return graph;
}
```

Complete the Code for addEdge Function

```
// Adds an edge to an undirected graph
void addEdge(struct Graph* graph, int src, int dest)
{
    1. Add an edge from src to dest.  A new node is added to the adjacency
       list of src.  The node is added at the begining

         write the code necessary for adding an edge from src to dest

    2. Since graph is undirected, add an edge from dest to src also

         write the necessary code to connect dest to src

}
```

Experimental Procedure

- Analyse the problem statement
- Design an algorithm for the given problem statement and develop a flowchart/pseudo-code
- Implement the algorithm in C language
- Compile the C program
- Design test cases and test the implemented program
- Document the Results
- Analyse and discuss the outcomes of your experiment

Exercise

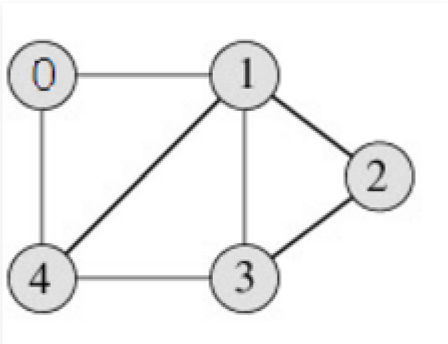
- Design an algorithm and write a program to create a hash table and use suitable hash function and demonstrate search operation for a given data. Tabulate the output for various inputs and verify against expected values. Analyse the efficiency of hash function used. Describe your learning along with the limitations of both, if any. Suggest how these can be overcome.

Key factors and discussion

- Implement adjacency matrix representation of graph
- Implement adjacency list representation of graph
  - Complete the code for addEdge function and write print graph function
  - Compare the efficiency of adjacency matrix and list representation of graph Collison resolution

Expected output for Adjacency Matrix

Input Graph

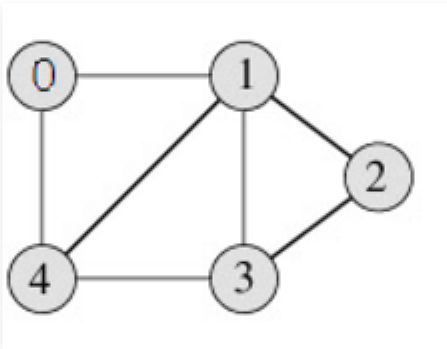


Output

	0	1	2	3	4
0	0	1	0	0	1
1	1	0	1	1	1
2	0	1	0	1	0
3	0	1	1	0	1
4	1	1	0	1	0

Expected output for Adjacency List

Input Graph



Output

```
Adjacency list of vertex 0
head -> 4-> 1

Adjacency list of vertex 1
head -> 4-> 3-> 2-> 0

Adjacency list of vertex 2
head -> 3-> 1

Adjacency list of vertex 3
head -> 4-> 2-> 1

Adjacency list of vertex 4
head -> 3-> 1-> 0
```

Results and Presentations

- Calculations/Computations/Algorithms

  The calculations/computations/algorithms involved in each program has to be presented

- Presentation of Results

  The results for all the valid and invalid cases have to be presented

- Analysis and Discussions

  how the data is manipulated or transformed, what are the key operations involved. Errors encounters and how they are resolved.

- Conclusions

  Summary

Comments

- Limitations of Experiments

  Outline the loopholes in the program, data structures or solution approach.

- Limitations of Results

  Present the test cases; justify if the program is tested correctly considering all the outcomes. Mention what is not tested, if any.

- Learning happened

  What is the overall learning happened

- Conclusions

  Summary

References

- Gilberg, R. F., and Forouzan, B. A. (2007): A Pseudocode Approach With C, 2nd edn. Cengage Learning