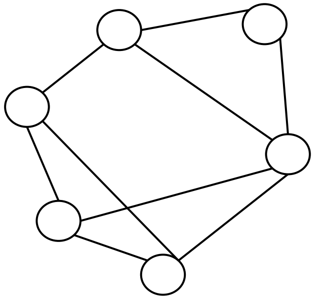
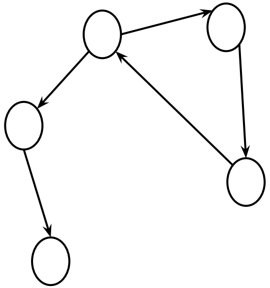
Graphs

Graphs

- Another data structure including vertices and edges
- Although trees are quite flexible, they have an inherent limitation in that they can only express hierarchical structures
- Fortunately, we can generalize a tree to form a graph, in which this limitation is removed
- Informally, a graph is a collection of nodes(vertices) and the connections between them(edges)





Graphs

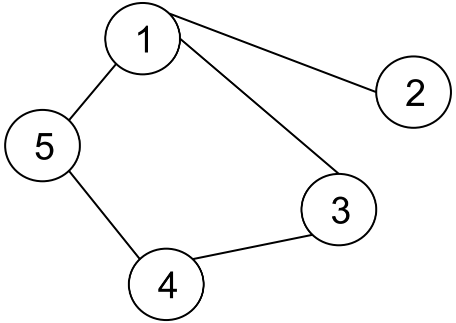
- Let’s consider some definitions:
- A simple graph G = (V, E) consists of a (finite) set denoted by V, and a collection E, of unordered pairs {u, v} of distinct elements from V
- Each element of V is called a vertex or a point or a node, and each element of E is called an edge or a link
- The number of vertices, the cardinality of V, is called the order of graph and devoted by |V|
- The cardinality of E, called the size of graph, is denoted by |E|

Graphs

- A path between vertices v_1 and v_n is a sequence of edges denoted v_1, v_2, …, v_{n-1}, v_n
- If $v_1 = v_n$, the path is actually a cycle

Graphs

- A path between vertices v_1 and v_n is a sequence of edges denoted $v_1, v_2, \ldots, v_{n-1}, v_n$
- If $v_1 = v_n$, the path is actually a cycle



1→ 3→ 4→ 5→ 1 is a cycle

Graphs

- Two vertices are adjacent or neighbor if there is an edge between them
- The number of edges incident with a vertex v, is the degree of the vertex; if the degree is 0, v is called isolated

Graph Representation

- Graphs can be represented in a number of ways, two of which are:

1. ***Adjacency List:*** One of the simplest is an adjacency list, where each vertex adjacent to a given vertex is listed

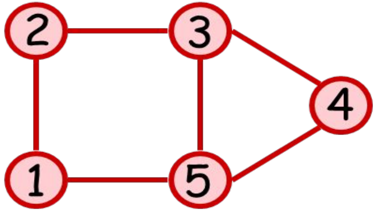
2. ***Adjacency Matrix*** is a |V| x |V| binary matrix where:

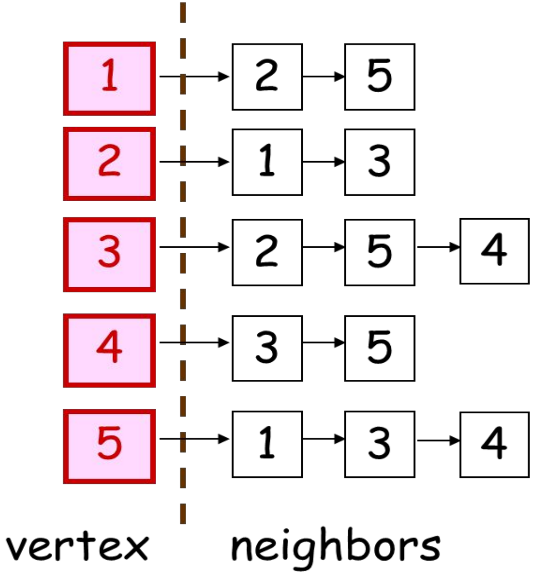
$$a_{ij} = \begin{cases} 1 & \text{if there exists an edge } \left(v_i v_j\right) \\ 0 & \text{otherwise} \end{cases}$$

Graph Representation

Adjacency List

- For each vertex u, store its neighbors in a linked list

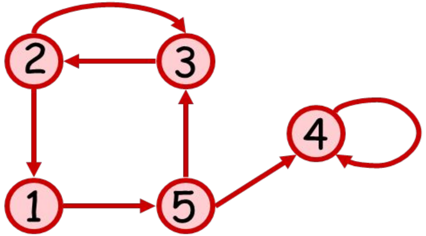


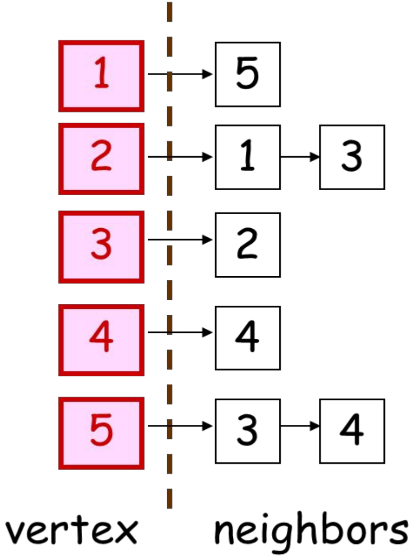


Graph Representation

Adjacency List

- For each vertex u, store its neighbors in a linked list





Graph Representation

Adjacency List:

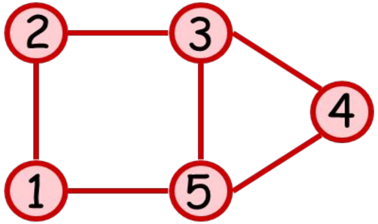
- Let G = (V, E) be an input graph
- Using Adjacency List representation :
- Space : O(|V| + |E|)
  - Excellent when |E| is small
- Easy to list all neighbors of a vertex
- Takes O(|V|) time to check if a vertex u is a neighbor of a vertex v

Graph Representation

Adjacency Matrix

- Use a |V| × |V| matrix A such that

$A(u,v) = 1$ if (u,v) is an edge
$A(u,v) = 0$ otherwise



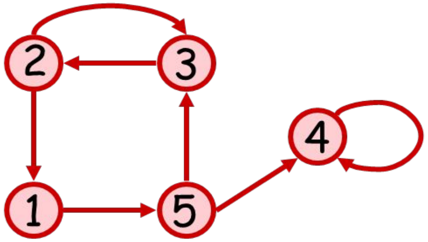
	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	0	0
3	0	1	0	1	1
4	0	0	1	0	1
5	1	0	1	1	0

Graph Representation

Adjacency Matrix

- Use a |V| × |V| matrix A such that

$$A(u,v) = 1 \quad \text{if} \quad (u,v) \text{ is an edge}$$
$$A(u,v) = 0 \quad \text{otherwise}$$



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

Graph Representation

Adjacency Matrix:

- Let G = (V, E) be an input graph
- Using Adjacency Matrix representation :
- Space : O($|V|^2$)
  - Bad when |E| is small
- O(1) time to check if a vertex u is a neighbor of a vertex v
- (|V|) time to list all neighbors

Graph Representation

How to implement adjacency list using **list** container?

```
#include <list>

class Graph
{
    int V;    // No. of vertices

    // Pointer to an array containing adjacency
    // lists
    list<int> *adj;
}
```

Graph Representation

How to implement adjacency list using **list** container?

```
#include <list>

class Graph
{
    int V;
    list<int> *adj;
}

Graph::Graph(int V)
{
    this->V = V;
    adj = new list<int>[V];
}
```

Graph Representation

How to implement adjacency matrix?

Graph Representation

How to implement adjacency matrix? It can be implemented using a 2D array only!