DS286 | 2016-10-26,28

L20-21: Graph Data Structure

Yogesh Simmhan
simmhan@cds.iisc.ac.in

Slides courtesy:
Venkatesh Babu, CDS, IISc

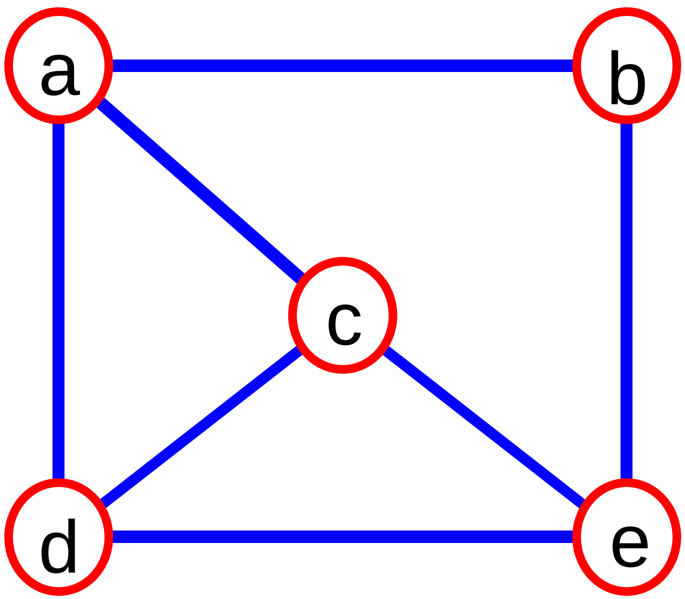
What is a Graph?

- 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:

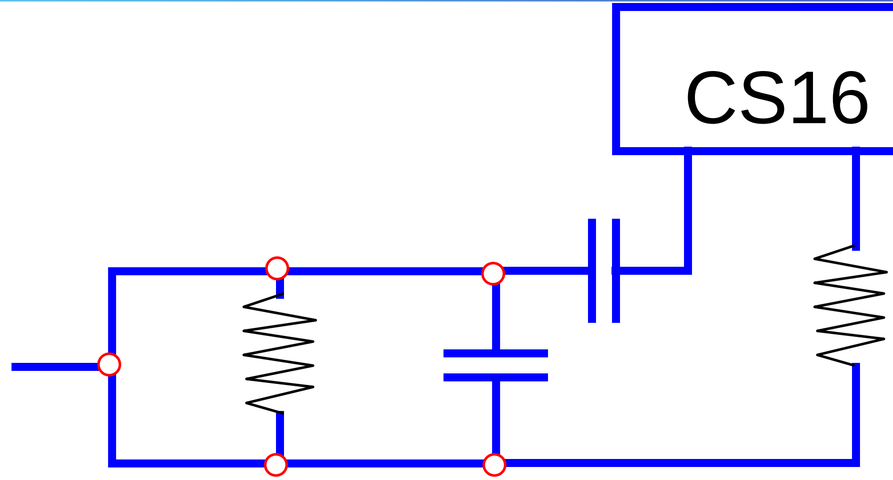


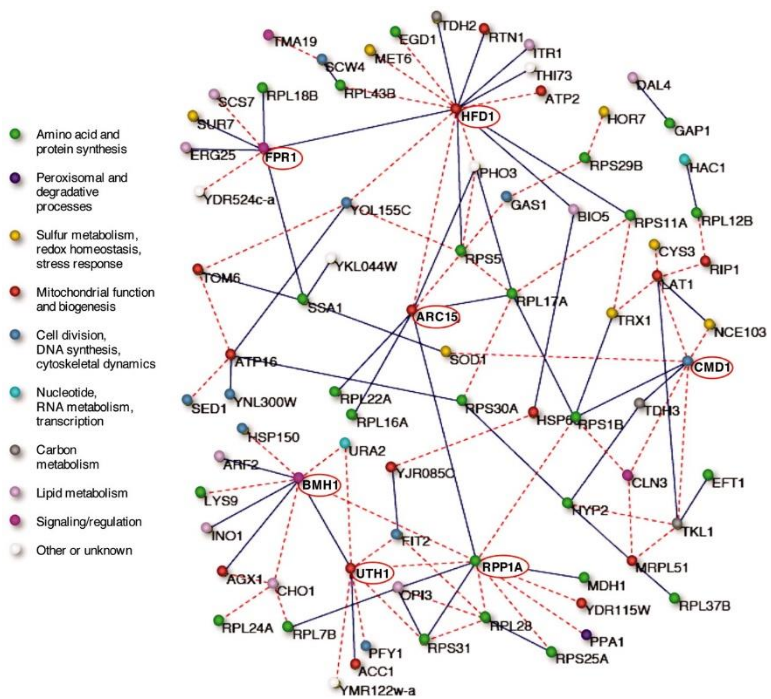
V= {a,b,c,d,e}

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

Applications

- Electronic circuit design
- Transport networks
- Biological Networks



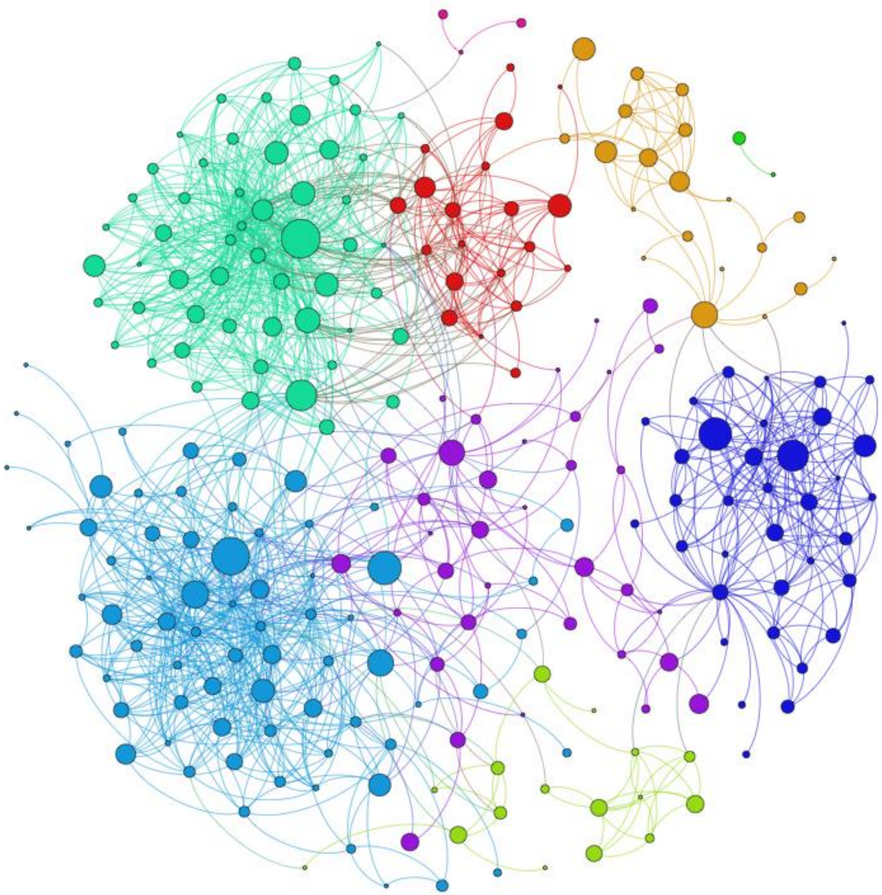


http://www.pnas.org/content/103/50/19033/F3.expansion.html

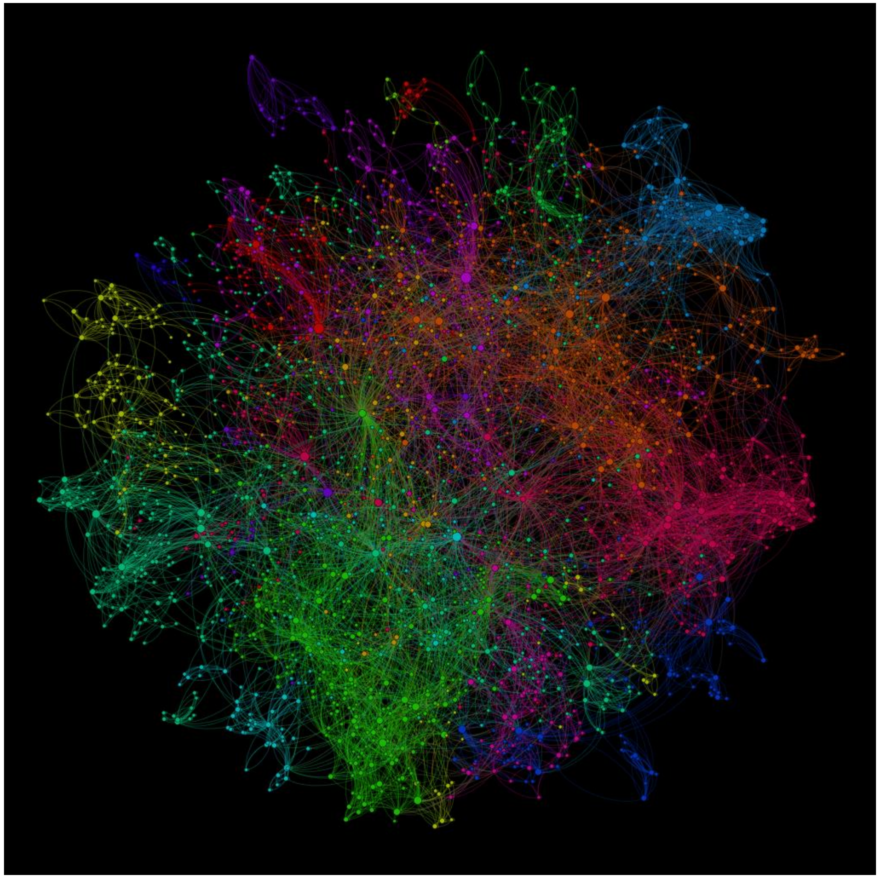
http://images.slideplayer.com/18/5684225/slides/slide_28.jpg

Applications

LinkedIn Social Network Graph



Java Call Graph for Neo4J



http://allthingsgraphed.com/2014/10/16/your-linkedin-network/

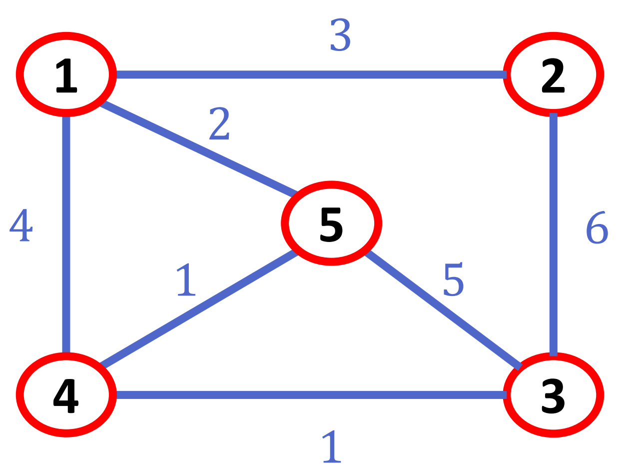
http://allthingsgraphed.com/2014/11/12/code-graphs-5-top-open-source-data-projects/

Terminology

- If (v_0, v_1) is an edge in an **undirected** graph,
  - v_0 and v_1 are adjacent, or are neighbors
  - The edge (v_0, v_1) is incident on vertices v_0 and v_1
- If $<v_0, v_1>$ is an edge in a **directed** graph
  - v_0 is adjacent to v_1, and v_1 is adjacent from v_0
  - The edge $<v_0, v_1>$ is incident on v_0 and v_1
  - v_0 is the source vertex and v_1 is the sink vertex

Terminology

- Vertices & edges can have **labels** that uniquely identify them
  - Edge label can be formed from the pair of vertex labels it is incident upon…*assuming only one edge can exist between a pair of vertices*
- Edge **weights** indicate some measure of distance or cost to pass through that edge



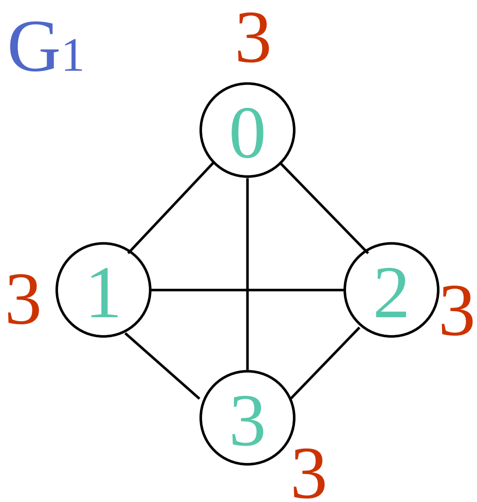
Terminology

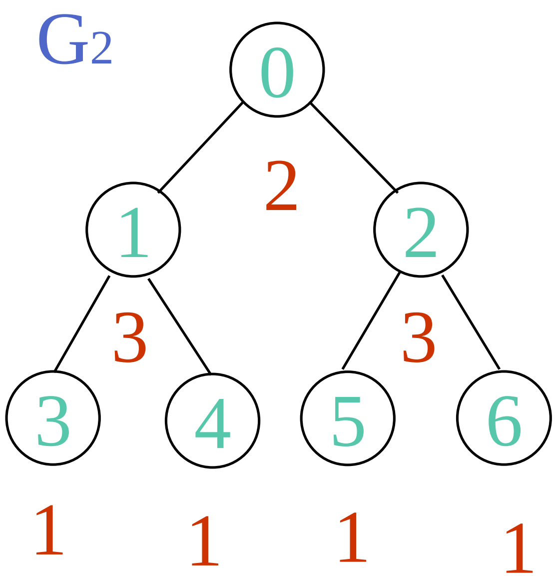
- The degree of a vertex is the number of edges incident to that vertex
- For directed graph,
  - the in-degree of a vertex *v* is the number of edges that have *v* as the sink vertex
  - the out-degree of a vertex *v* is the number of edges that have *v* as the source vertex
  - if d_i is the degree of a vertex *i* in a graph *G* with *n* vertices and *e* edges, the number of edges is

$$e = (\sum_{0}^{n-1} d_i) / 2$$

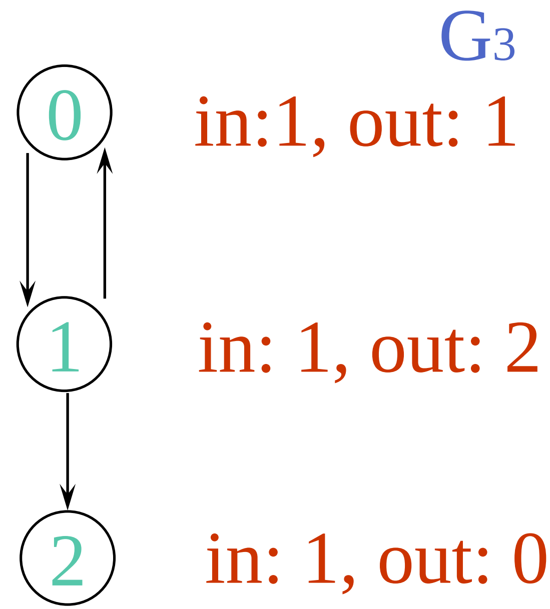
Why? Since adjacent vertices each count the adjoining edge, it will be counted twice

Examples





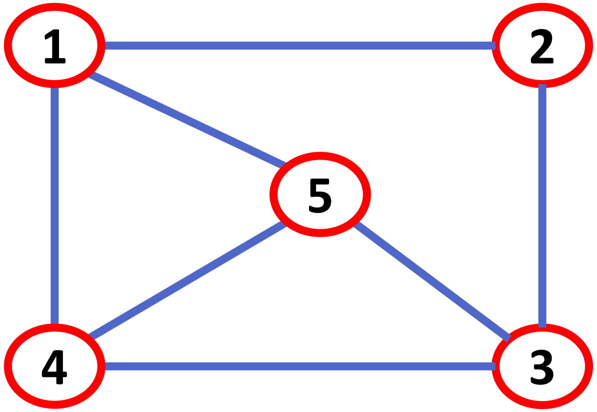
undirected graphs

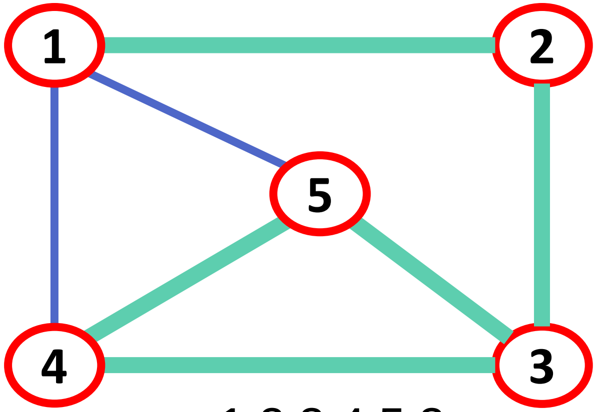


directed graph

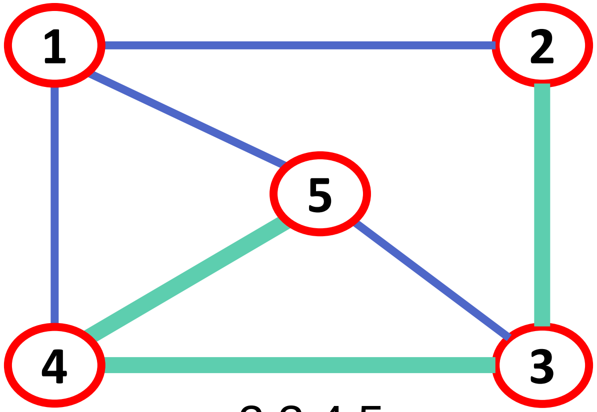
Terminology

- path is a sequence of vertices $<v_1,v_2,. . .v_k>$ such that consecutive vertices v_i and v_{i+1} are adjacent





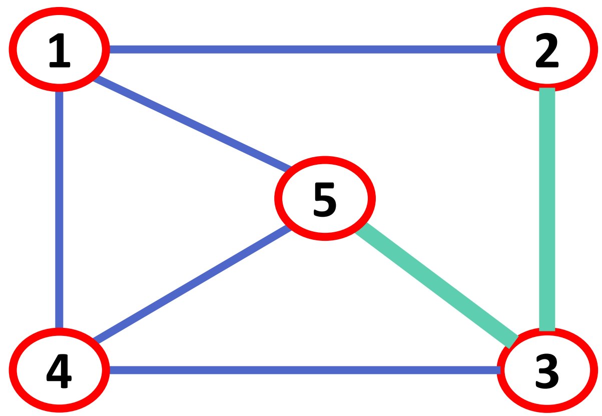
1 2 3 4 5 3



2 3 4 5

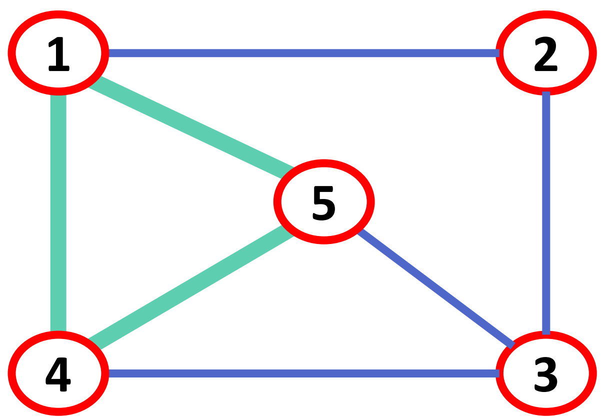
Terminology

- simple path: no repeated vertices



2 3 5

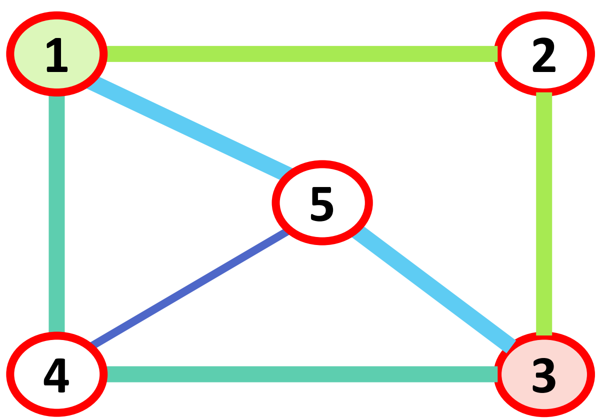
- cycle: simple path, except that the last vertex is the same as the first vertex



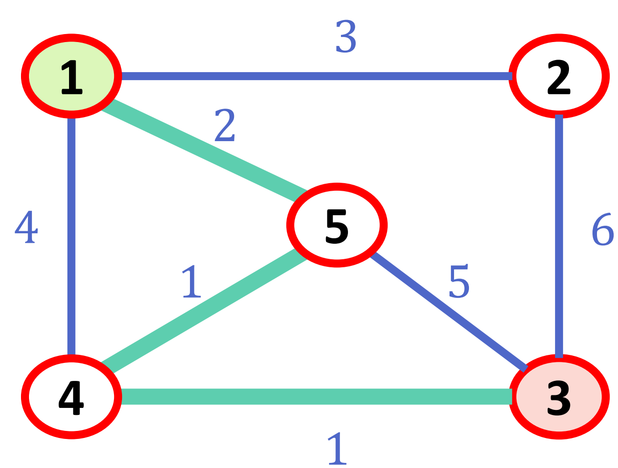
1 5 4 1

Terminology

- Shortest Path: Path between two vertices where the sum of the edge weights is the smallest
  - Has to be a simple path (why?)
  - Assume “unit weight” for edges if not specified



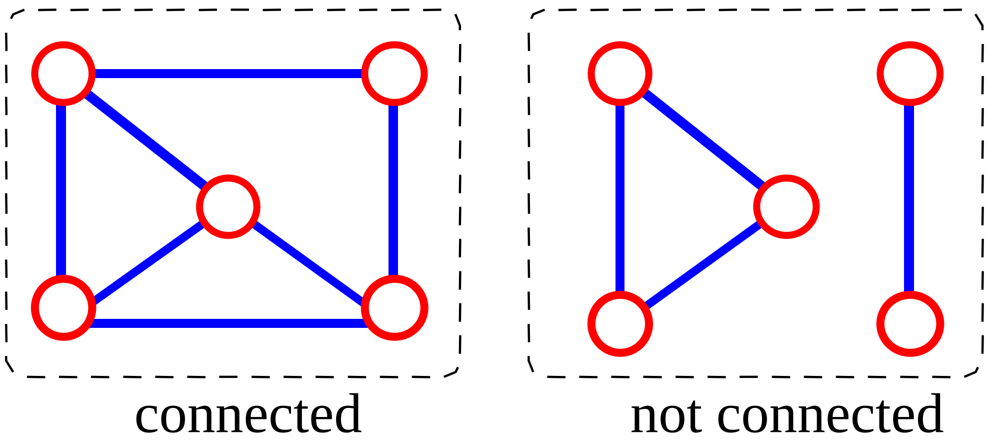
1 2 3
1 5 3
1 4 3



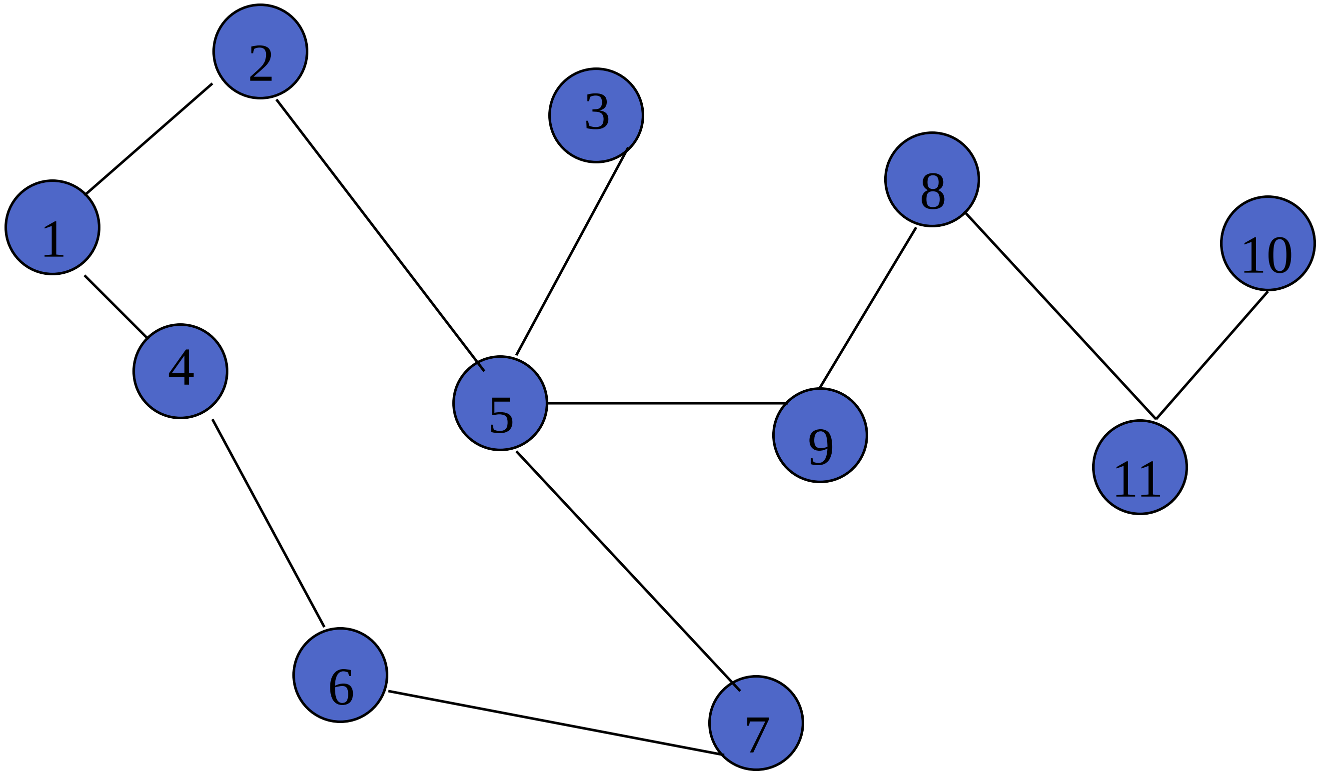
1 5 4 3

Connected Graph

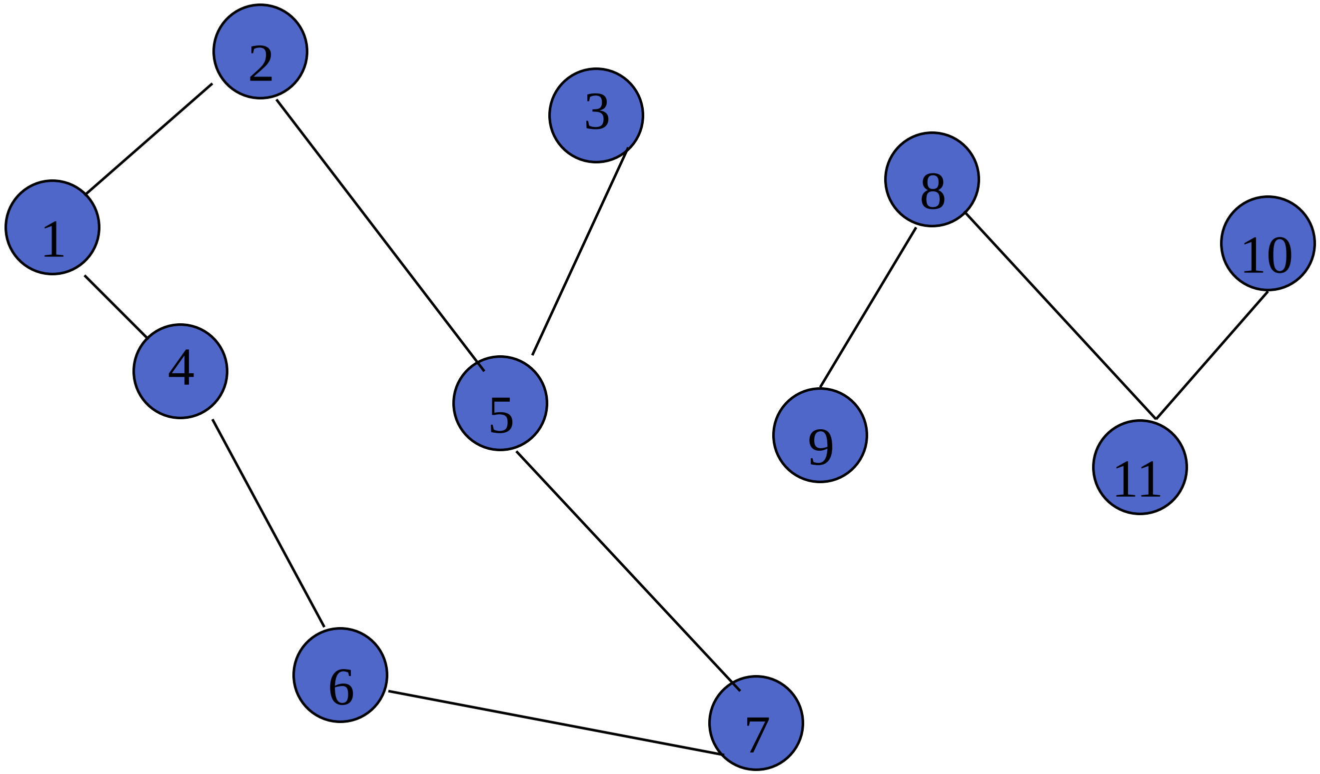
- connected graph: any two vertices are connected by some path



Connected Graph Example

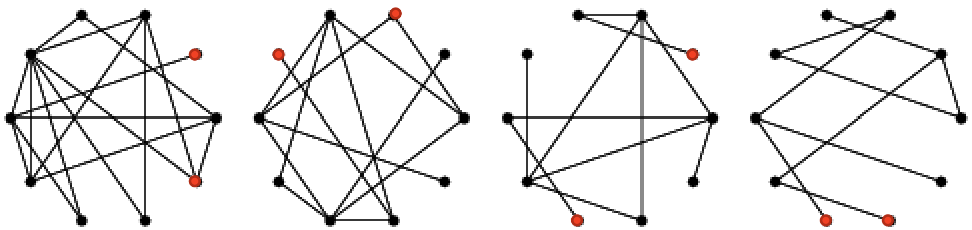


Example Of Not Connected



Graph Diameter

- A graph's dimeter is the distance of its *longest shortest path*
- if d(u,v) is the distance of the shortest path between vertices u and v, then:
- *diameter = Max(d(u,v))*, for all u, v in V
- A disconnected graph has an infinite diameter

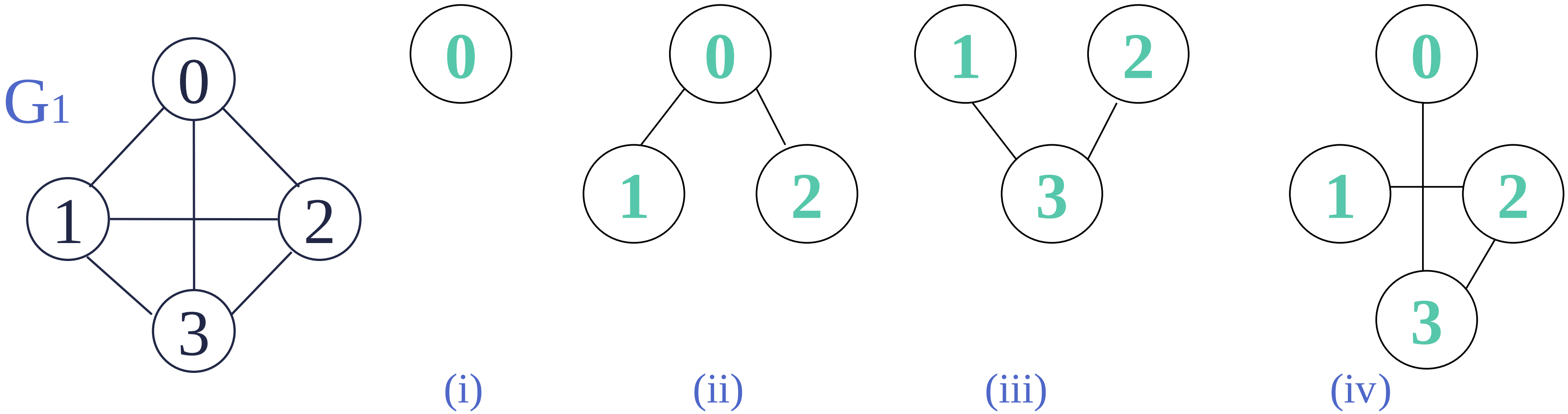


d=3 *d=4* *d=5* *d=7*

http://mathworld.wolfram.com/GraphDiameter.html

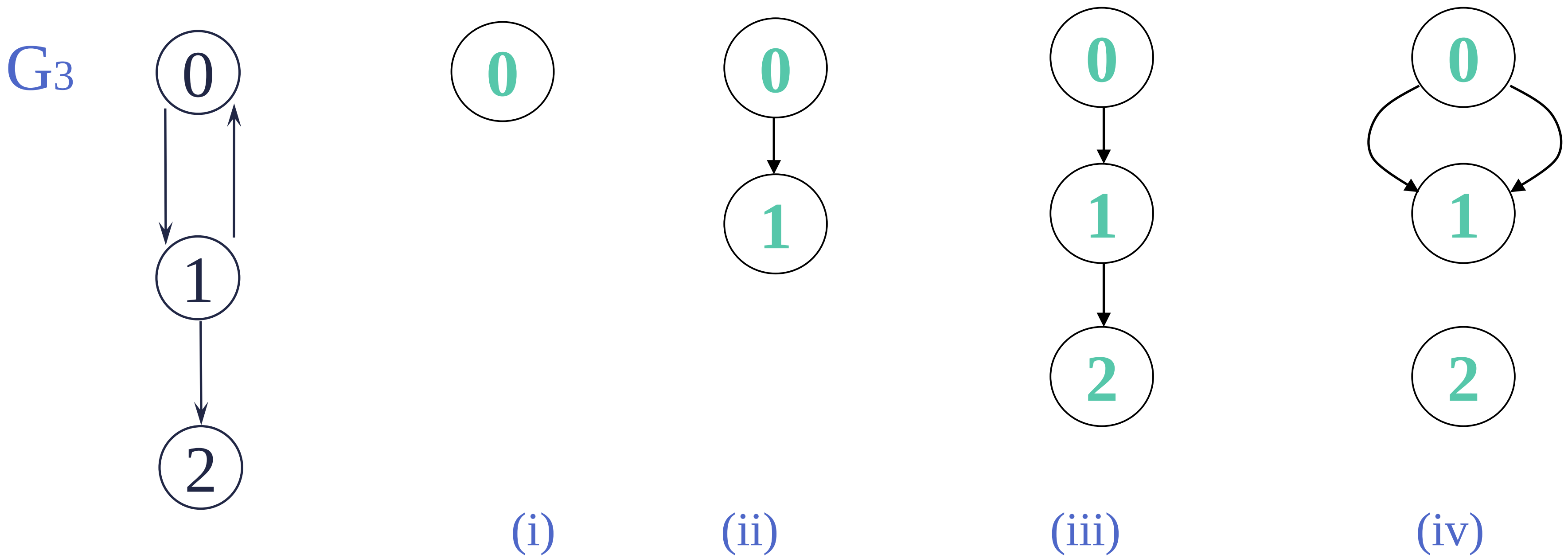
Subgraph

- subgraph: subset of vertices and edges forming a graph



(a) Some of the subgraph of G_1

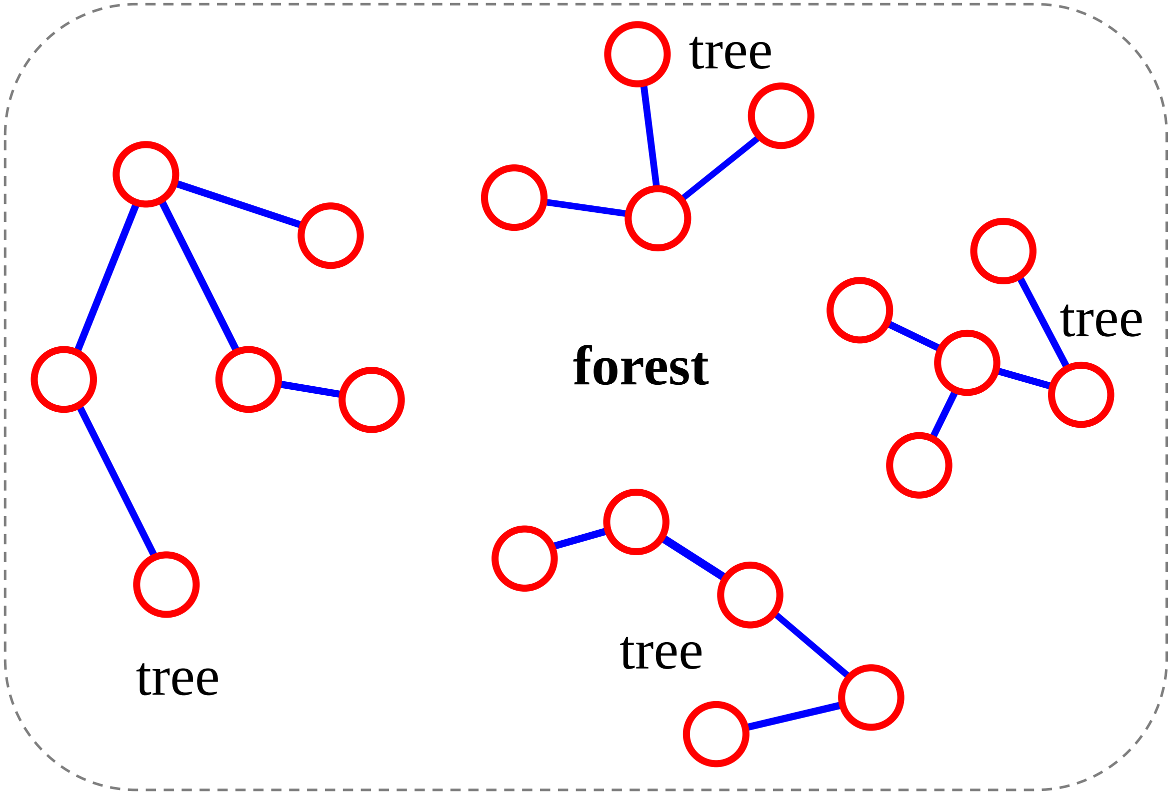
Subgraphs Examples



(b) Some of the subgraph of G_3

Trees & Forests

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

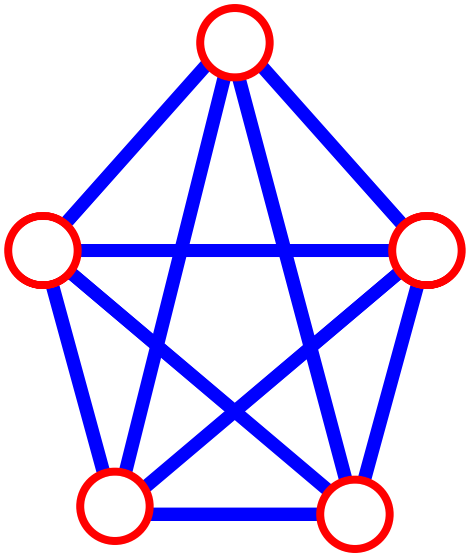


Fully Connected Graph

- Let **n** = #vertices, and **m** = #edges
- **Complete graph (or) Fully connected 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.

If a graph is not complete:

m < **n**(**n** -1)/2



n = 5

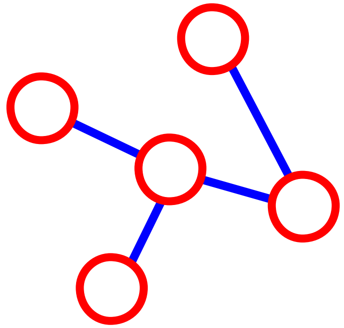
m = (5*4)/2 = 10

More Connectivity

n = #vertices

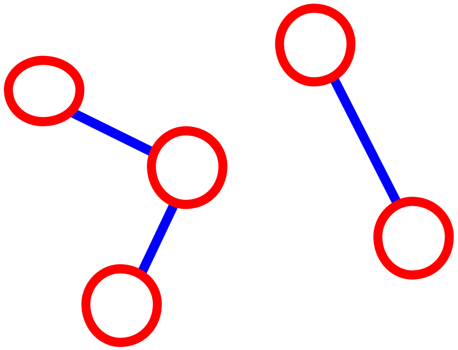
m = #edges

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



n = 5
m = 4

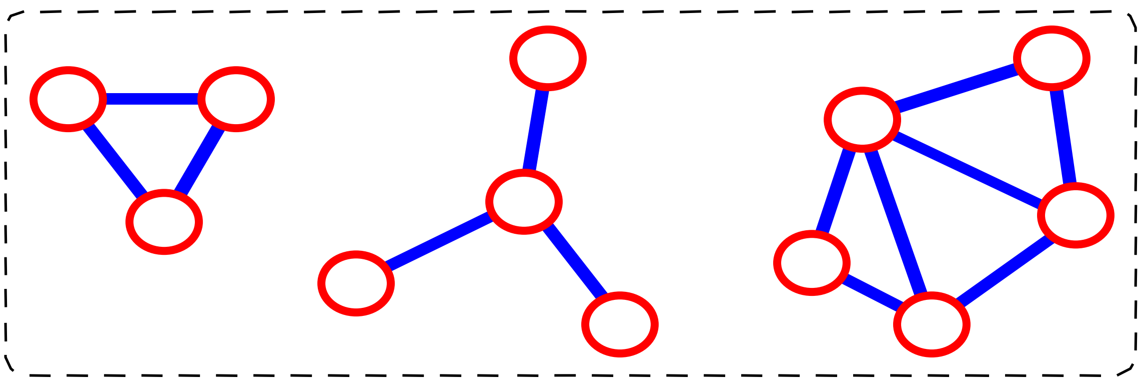
If **m** < **n** - 1, G is not connected



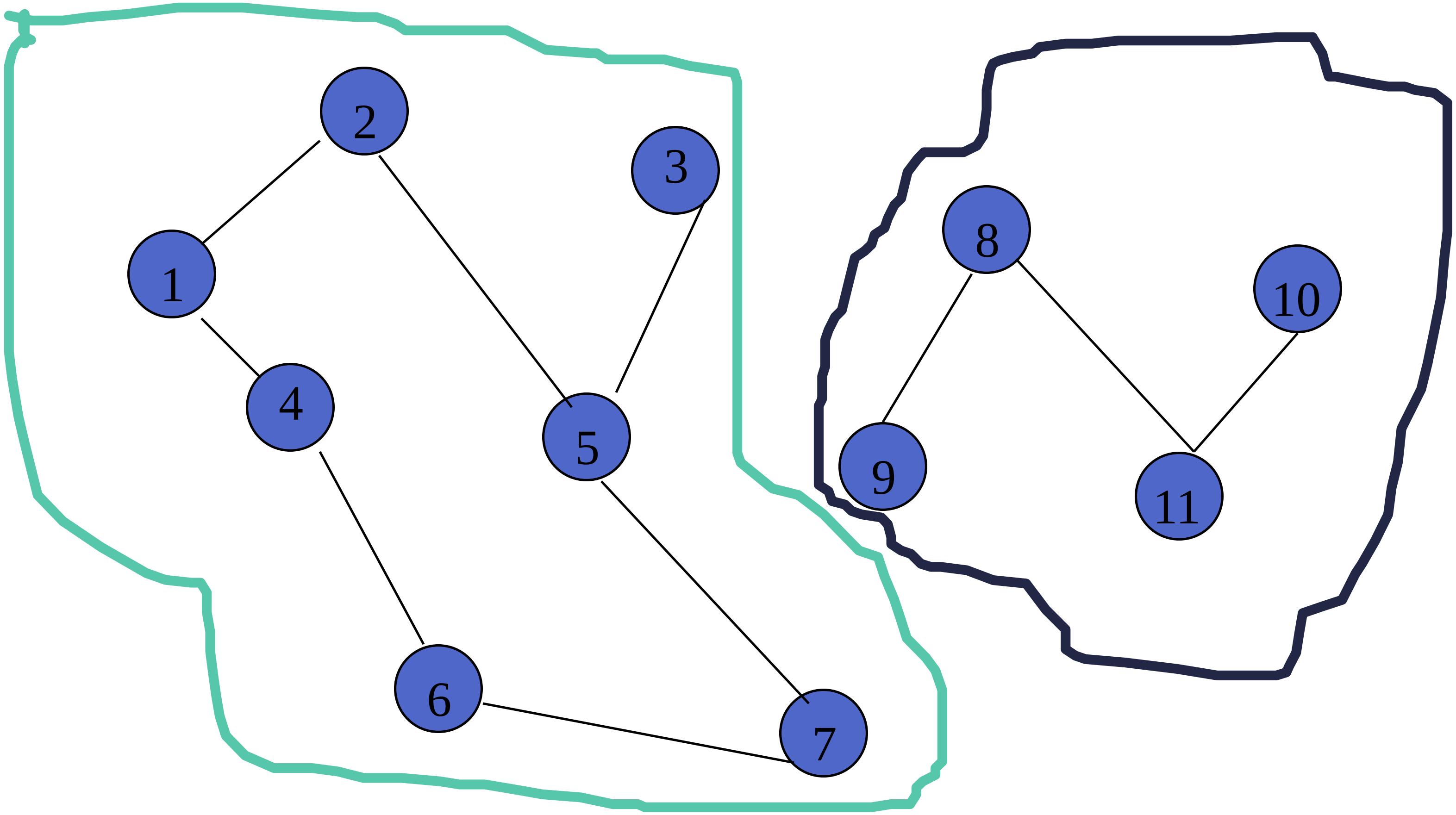
n = 5
m = 3

Connected Component

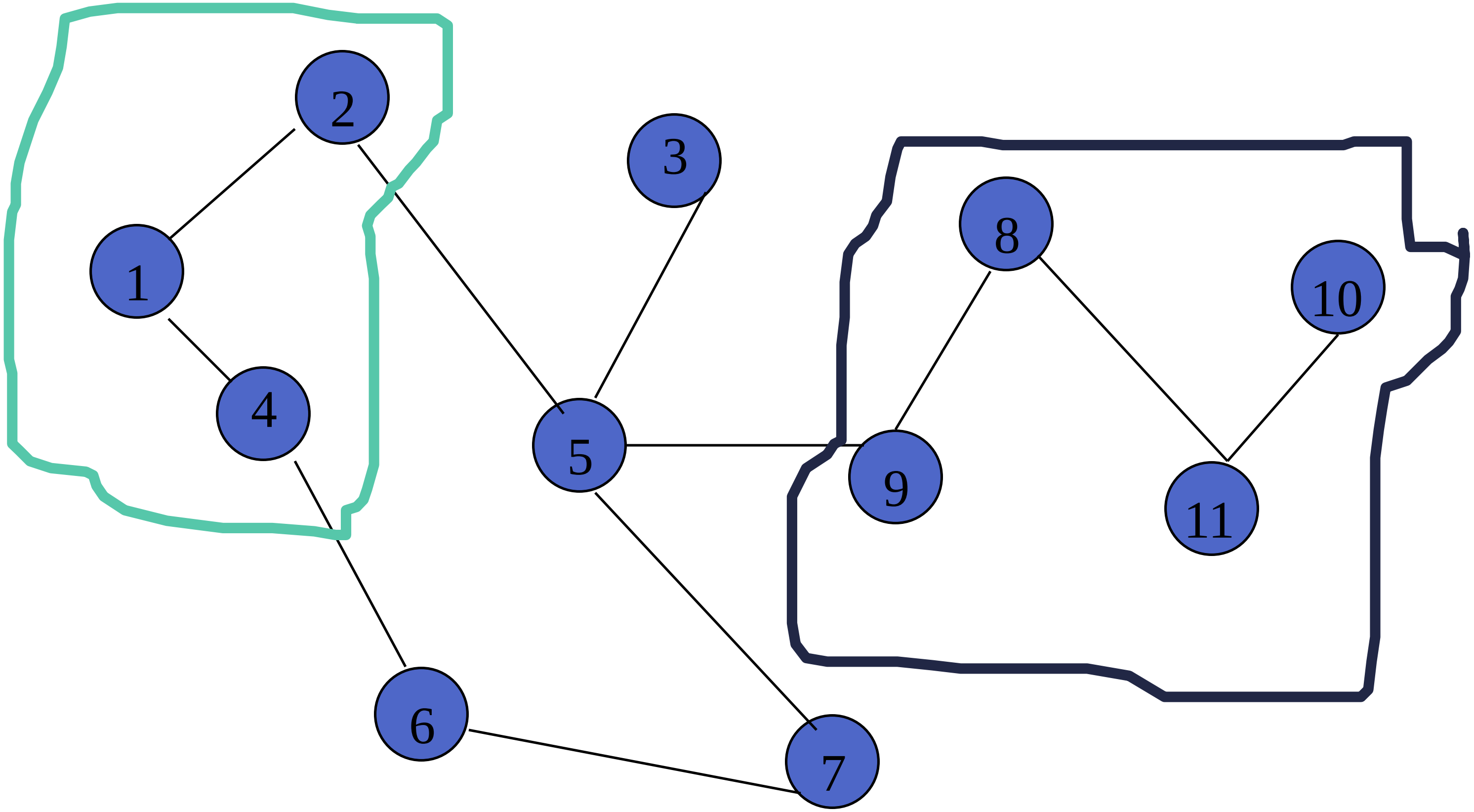
- A connected component is a maximal subgraph that is connected.
  - Cannot add vertices and edges from original graph and retain connectedness.
- A connected graph has exactly 1 component.



Connected Components

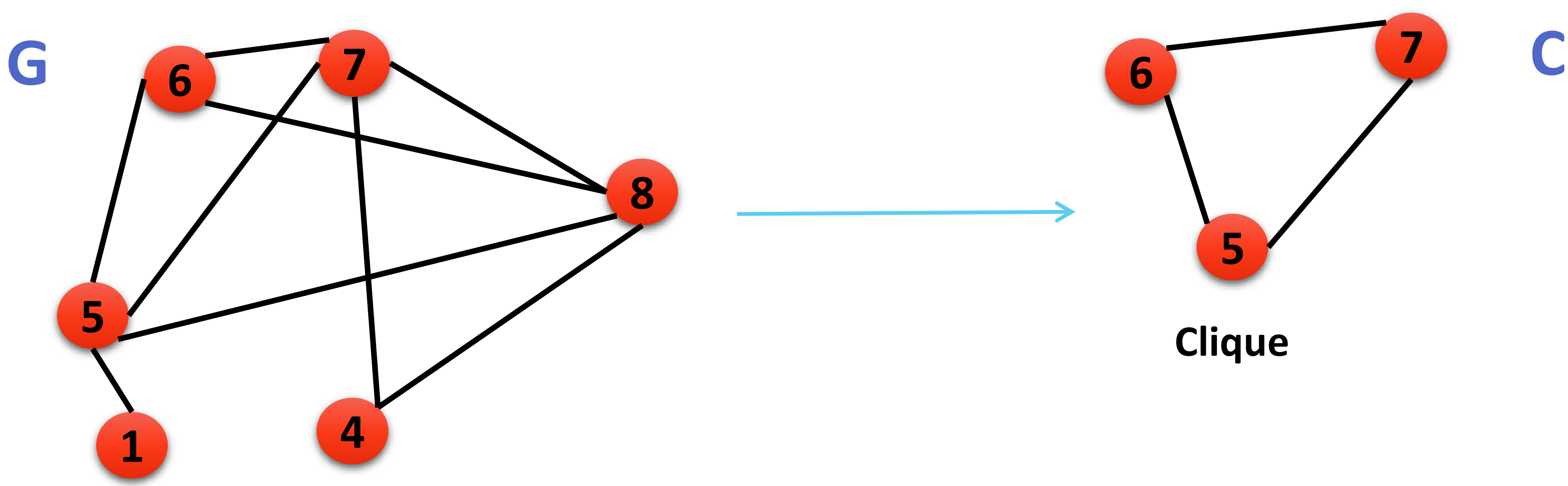


Not A Component



Clique

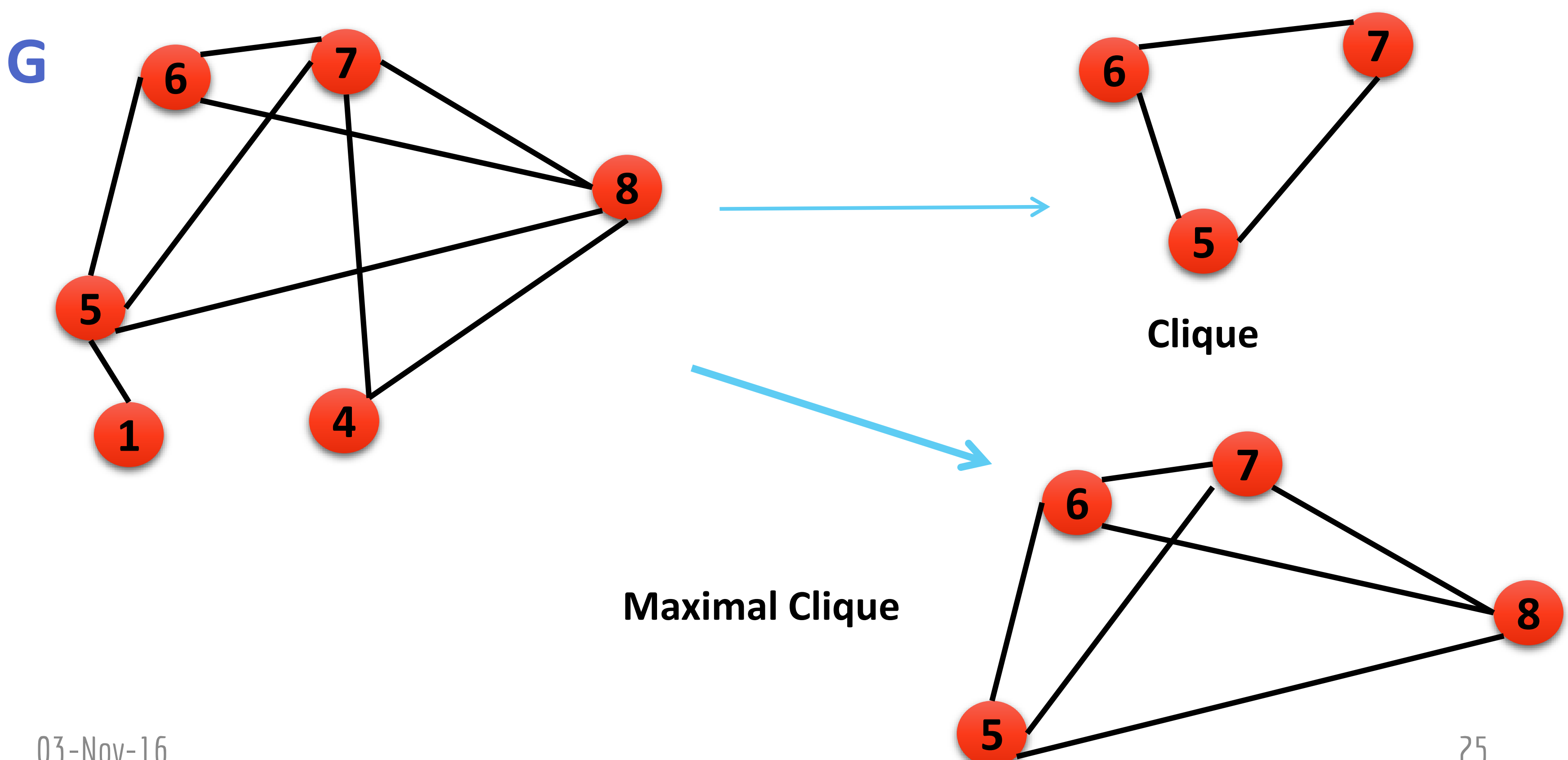
- A subgraph C of a graph G with *edges between all pairs of vertices*



https://www.csc.ncsu.edu/faculty/samatova/practical-graph-mining-with-R

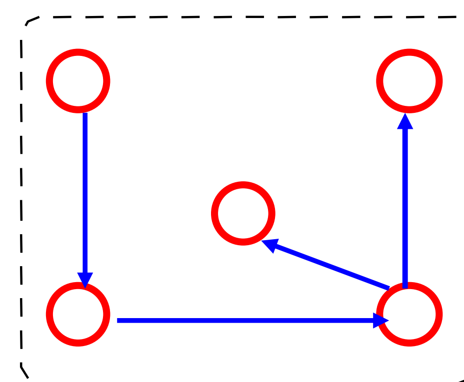
Maximal Clique

- A maximal clique is a clique that is not part of a larger clique



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 (or **Digraph**) is one in which each edge is a directed pair of vertices, $<v_0, v_1> \; != \; <v_1, v_0>$

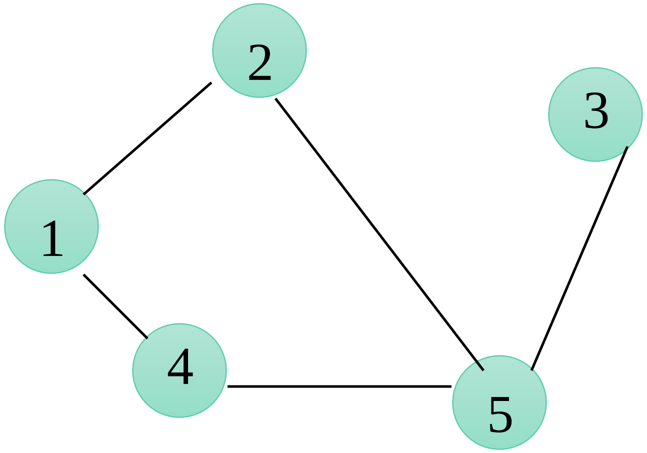


Graph Representation

- Adjacency Matrix
- Adjacency Lists
  - Linked Adjacency Lists
  - Array Adjacency Lists

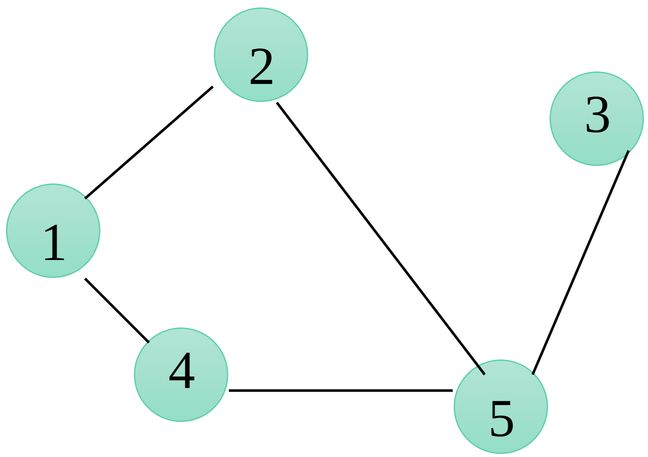
Adjacency Matrix

- 0/1 n x n matrix, where n = # of vertices
- A(i,j) = 1 iff (i,j) is an edge



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

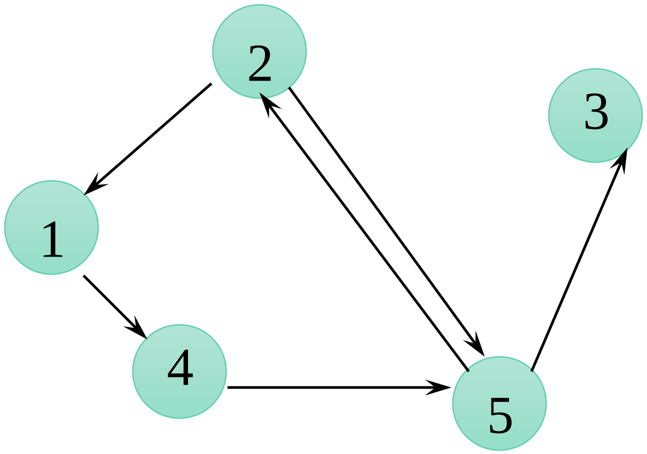
Adjacency Matrix Properties



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

- Diagonal entries are zero.
- Adjacency matrix of an *undirected graph* is *symmetric*.
  - A(i,j) = A(j,i) for all i and j.

Adjacency Matrix (Digraph)



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

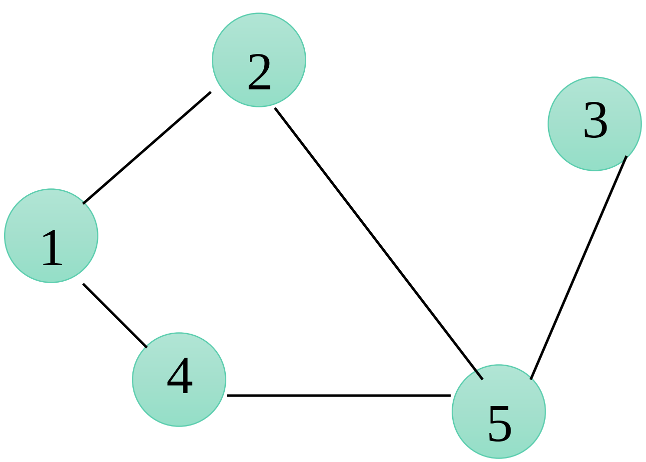
- Diagonal entries are zero.
- Adjacency matrix of a directed graph need not be symmetric.

Adjacency Matrix

- n^2 bits of space
- For an *undirected graph*, may store only lower or upper triangle (exclude diagonal)
  - $(n^2 - n)/2$ bits
- $O(n)$ time to find vertex degree and/or vertices adjacent to a given vertex.

Adjacency Lists

- Adjacency list for vertex *i* is a linear list of vertices adjacent from vertex *i*.
- An array of *n* adjacency lists.



aList[1] = (2,4)

aList[2] = (1,5)

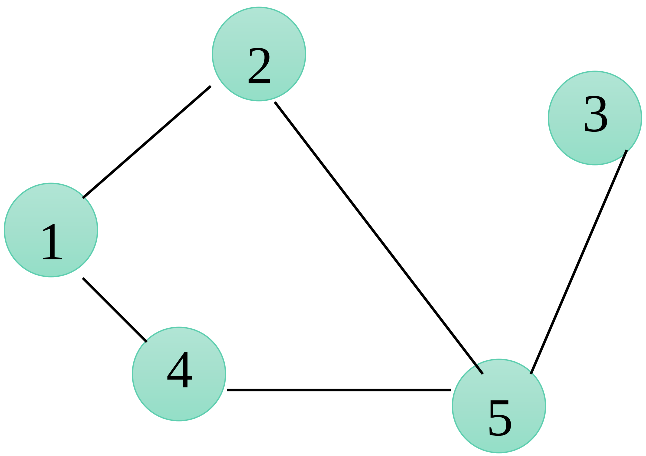
aList[3] = (5)

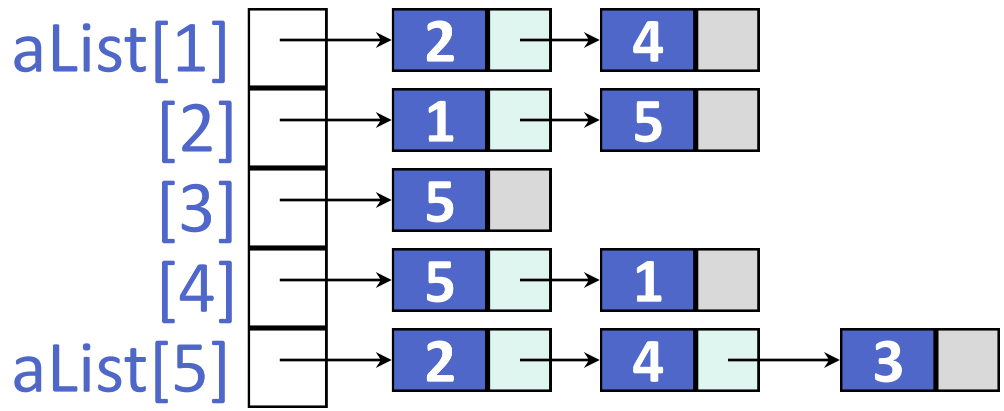
aList[4] = (5,1)

aList[5] = (2,4,3)

Linked Adjacency Lists

- Each adjacency list is a chain.

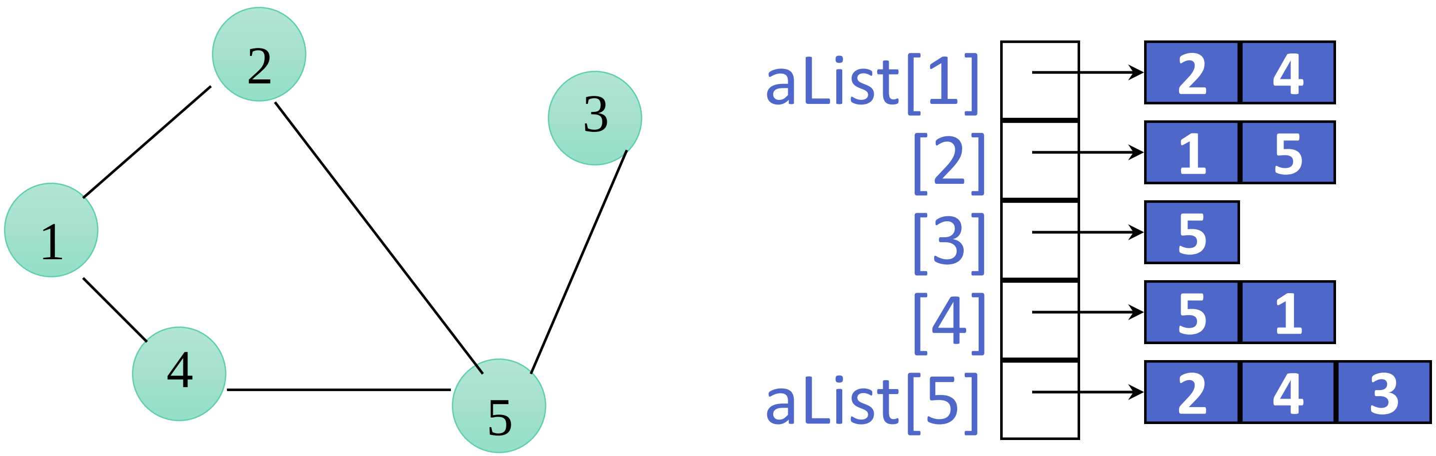




- Array Length = n
- # of chain nodes = 2e (undirected graph)
- # of chain nodes = e (digraph)

Array Adjacency Lists

- Each adjacency list is an array list.



- Array Length = n
- # of list elements = 2e (undirected graph)
- # of list elements = e (digraph)

Storing Weighted Graphs

- Cost adjacency matrix
  - C(i,j) = cost of edge (i,j) instead of 0/1
- Adjacency lists
  - Each list element is a pair (adjacent vertex, edge weight)

ADT for Graph

```
class Vertex<V,E> {
   int id;
   V value;
   int GetId();
   V GetValue();
   List<Edge<V,E>> Neighbors();
}
class Edge<V,E> {
   int id;
   E value;
   int GetId();
   E GetValue();
   Vertex<V,E> GetSource();
   Vertex<V,E> GetSink();
}
```

ADT for Graph

```
class Graph<V,E>{
   List<Vertex<V,E>> vertices;
   List<Edge<V,E>> edges;

   void InsertVertex(Vertex<V,E> v);
   void InsertEdge(Edge<V,E> e);

   bool DeleteVertex(int vid);
   bool DeleteEdge(int eid);

   List<Vertex<V,E>> GetVertices();
   List<Edge<V,E>> GetEdges();

   bool IsEmpty(graph);
}
```

Tasks

- Self study
  - **Read**: Graphs and graph algorithms (online sources)
- Finish Assignment 5 by Mon Nov 14 *(100 points)*
- Make progress on CodeChef (100 points)

Questions?