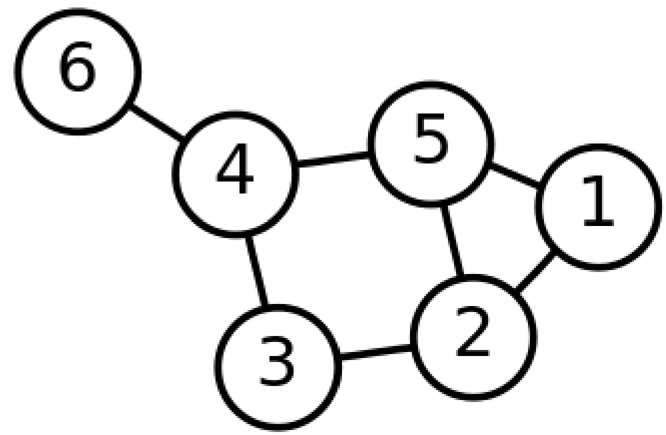
Graph Theory Crash Course

2015

Graph

- Model pairwise relations between objects
  - Objects = “Nodes”
  - Relations = “Edges”
- Examples:
  - Road map
    - Intersections = Nodes
    - Roads = Edges
  - Chess
    - Game board states = Nodes
    - Available moves = Edges

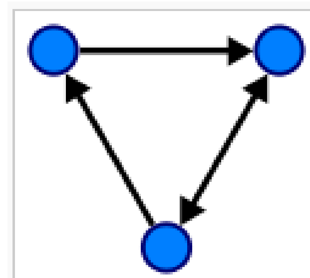


Nodes

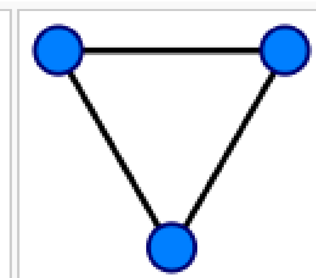
- Objects in the graph, related by edges
- Can contain information
- Example: Road map
  - Intersection nodes, number of cars / hr
- Example: Chess
  - Game state nodes, position of all the pieces

Edges

- Can be directed or undirected
- Undirected edges:
  - Drawn as lines
  - Edge from node a to b is same as from b to a
  - No orientation
- Directed edges:
  - Drawn as arrows
  - Edge from node a to b cannot be traversed from b to a
  - Orientation from src to dst



A directed graph.



A simple undirected graph with three vertices and three edges. Each vertex has degree two, so this is also a regular graph.

Edges

- Can store information, usually in the form of an edge “weight”.
- Example: Road map
  - Edge weight = length of road from intersection a to intersection b

Graph Representation

- How can we model a collection of pairwise relations in code?
- Edge list: simply list the relations
- Adjacency matrix
- Adjacency list

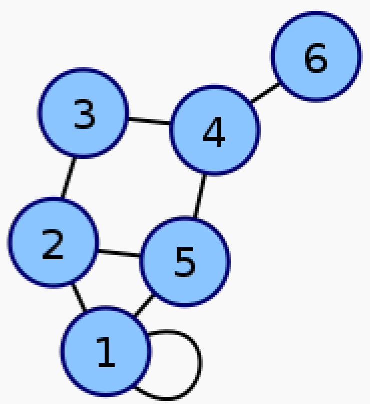
Edge list

- Simply make a list (or vector) of pairwise relations.

```
struct Edge {
    int a, b;
};

vector<Edge> EdgeList;

// Edge from 0 to 1
Edge e1;
e1.a = 0; e1.b = 1;

// Edge from 1 to 2
Edge e2;
e2.a = 1; e2.b = 2;

EdgeList.push_back(e1);
EdgeList.push_back(e2);
```

Adjacency Matrix

- Make a table. Rows correspond to the source node, columns to the destination node.
- A 1 in row R and column C means that the edge R->C exists.



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

Coordinates are 1-6.

Adjacency Matrix

- Undirected graph: if a->b exists, so does b->a
  - Therefore, matrix symmetric.
- Weighted graph
  - May replace 1 with the edge weight.

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

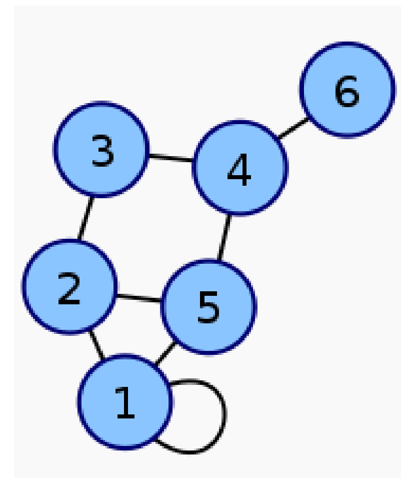
Coordinates are 1-6.

Adjacency Matrix

```
int Graph[10][10];  // 10 nodes, 0-9

Graph[2][1] = 1;    // Create edge 2->1
Graph[3][2] = 1;    // Create edge 3->2
```

Adjacency Lists

- Each node stores the edges that extend from that node.
- Example:
  - Node 1 stores:
    - Edge to 5
    - Edge to 1
    - Edge to 2



- For undirected graphs, we need to be careful to add the reverse edges too.

Adjacency Lists

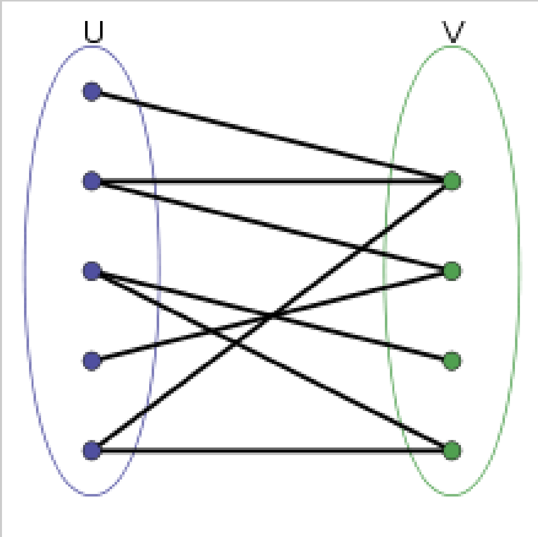
```
struct Node {
    // Store endpoint of edges from this node
    vector<int> Edges;
};

vector<Node> Graph(10); // 10 nodes, 0-9

// Insert an undirected edge between nodes 0 and 1
Graph[0].Edges.push_back(1);    // Edge from 0 to 1
Graph[1].Edges.push_back(0);    // Edge from 1 to 0
```

Special Graphs

- Regular graph: Each node has the same number of neighbours
- Complete graph: Every pair of nodes is joined by an edge; every possible edge exists.
- Connected graph: Always possible to move from a to b for any {a,b}.
- Strongly connected graph: Directed path from a to b exists for all {a,b}.

Special graphs

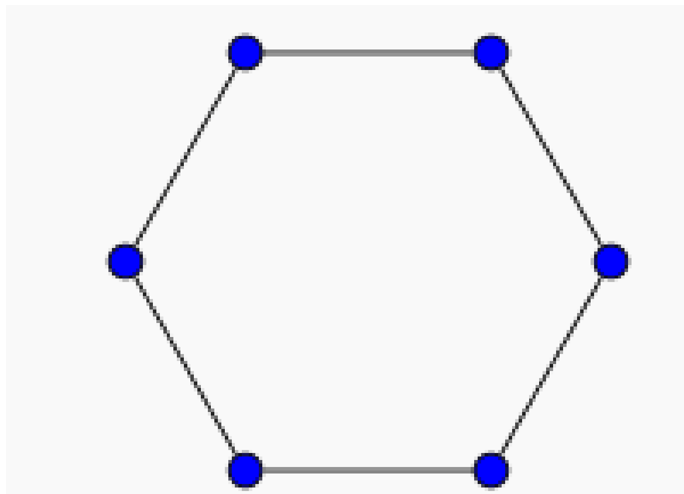
- Bipartite graph: Can separate vertices into two groups such that edges only cross between the groups.



Example of a bipartite graph without cycles

Cycle

- A path that leads you back where you started.



- Some graphs contain cycles, others do not.
- Cycle-free graphs are called “acyclic”.

Trees

- Tree: Connected graph with no cycles