Graph - Data structure

- Basics
- BFS & applications
- DFS & applications

Basic Definitions

A graph G is a pair of sets (V, E), where V is an arbitrary non-empty set and E is a set of pairs of elements of V.

Elements of V and E are called vertices (nodes) and edges (links) of G respectively.

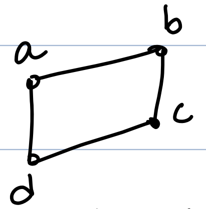
In an undirected graph, the edges are unordered pairs (sets of size two). We use uv instead of {u, v} to denote the undirected edge between vertices u and v.

In a directed graph, the edges are ordered pairs of vertices. We use (u, v) denote the directed edge from u to v.

Usually n - denotes # of vertices.

m - " " " edges.

Eg 1

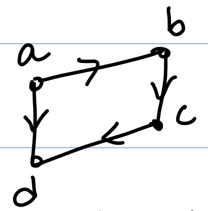


Undirected graph

$V = \{a, b, c, d\}$

$E = \{\{a,b\}, \{b,c\}, \{c,d\}, \{d,a\}\}$

Eg 2



directed graph

$V = \{a, b, c, d\}$

$E = \{(a,b), (b,c), (c,d), (a,d)\}$

Real life examples: Road networks, Social networks, etc let $G = (V, E)$ be an undirected graph.

Path: A Path P is a Sequence of distinct vertices $v_1, v_2, \ldots v_k$ such that each Consecutive Pair v_i, v_{i+1} is joined by an edge in G.
P is called a Path from v_1 to v_k.

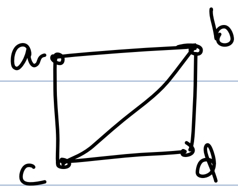
Cycle: A cycle is a Path $v_1, v_2 \ldots v_k$, $(k > 2)$ with $v_1 = v_k$.

Connected: An undirected graph is Connected if for every Pair of nodes u and v there is a Path from u to v.

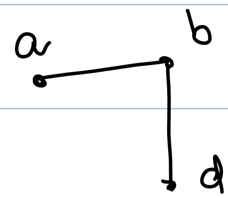
Tree: An undirected graph is a tree if it is Connected and does not Contain a cycle.

Subgraph: A Subgraph of a graph G is another graph formed from a subset of the vertices and edges of G.

Example:



G



H

H is a Subgraph of G.

Exercise

Prove that

① Every tree on n vertices has exactly $n-1$ edges

Other Terminology

1. adjacent
2. degree
3. Subgraph, induced subgraph
4. Walk, Path, Cycle, length of the path/cycle
5. Connected graph, Components, Strongly Connected
6. Tree, Forest, DAG
7. distance in the graphs (weighted vs unweighted)

Representation of Graphs

There are several ways to represent graphs, each with its advantages and disadvantages.

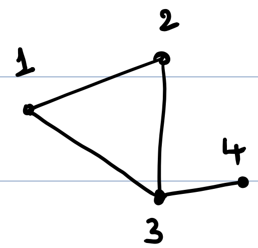
In this course we will see three ways to represent graphs.

1. Edge lists
2. Adjacency matrices
3. Adjacency Lists

Edge list

An edge list is a list or array of all the edges of the graph.

E.g.:



Array = [[1 2] [2 3] [1 3] [3 4]]

Total space for an edge list is $O(E)$

disadvantage:

If we want to find whether the graph contains a particular edge, we have to search through the edge list, In the worst case we have to search through $|E|$ edges.

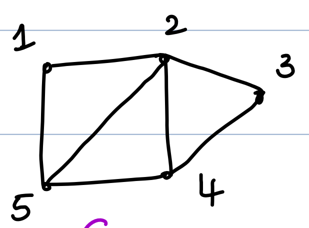
Adjacency matrices

We assume that the vertices are numbered $1, 2, \ldots |V|$ in some arbitrary manner.

The adjacency matrix of a graph G is a $V \times V$ matrix $A = (a_{ij})$ of 0's and 1's such that

$$a_{ij} = \begin{cases} 1 & \text{if } ij \in E \\ 0 & \text{otherwise} \end{cases}$$

Example:



G

$$A = \begin{array}{c|c} & \begin{matrix} 1 & 2 & 3 & 4 & 5 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{matrix} & \begin{bmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

Adjacency matrix representation of G.

Remark: For an undirected graph, the adjacency matrix is symmetric.

For a directed graph the adjacency matrix need not be symmetric.

We can find out whether an edge is Present in constant time by just looking up the corresponding entry in the matrix

<u>disadvantage</u> :

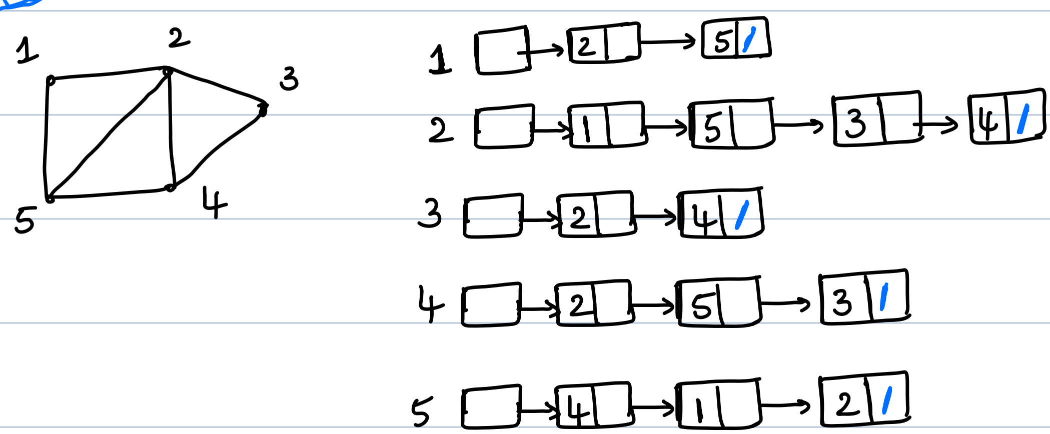
① Adjacency matrix representation of a graph requires $\Theta(n^2)$ space, irrespective of the # of edges in the graph.

② Finding which vertices are adjacent to a given vertex i, we have to look at all $|V|$ entries in row i, even if only a small number of vertices are adjacent to a vertex i

Adjacency Lists :

An adjacency list is an array of lists, each containing the neighbors of one of the vertices (out-neighbors if the graph is directed).

Example:



This representation uses an array indexed by vertex number, in which the array cell for each vertex points to a singly linked list of the neighboring vertices.

① The overall space required for an adjacency list is $O(n+m)$ { Some books write $O(V+E)$ }

n → # of vertices, m → # of edges

or $O(|V|+|E|)$

② The standard way to implement the adjacency list is using a array (list) of single-linked lists, where the head of the linked list is the vertex and all the connected linked lists are the vertices to which it is connected.

③ We can find neighbors of a vertex i in $\Theta(d_i)$ time, where d_i is the degree of vertex i.

Comparison :

		Adjacency list	Adjacency matrix
①	Space	$\Theta(V+E)$	$\Theta(V^2)$
②	Test if $uv \in E$	$O(1+\min\{\deg(u),\deg(v)\})$ $\parallel$ $O(V)$ (worst case)	$O(1)$
③	Test if $(u,v) \in E$ ↓ directed edge	$O(1+\deg(u)) = O(V)$	$O(1)$
④	List v's (out)-neighbors	$\Theta(1+\deg(v)) = O(V)$	$\Theta(V)$
⑤	List all edges	$\Theta(V+E)$	$\Theta(V^2)$
⑥	Insert edge uv	$O(1)$	$O(1)$
⑦	Delete edge uv	$O(\deg(u)+\deg(v)) = O(V)$	$O(1)$
⑧	Adding a vertex	$O(1)$	$O(V^2)$ (Storage space must be increased.
⑨	Adding an edge	$O(1)$	$O(1)$
⑩	Delete a vertex	$O(V+E)$	$O(V^2)$

Relationship b/w n & m:

If G is undirected & Connected & without Parallel edges & self loops

then $$n-1 \le m \le \binom{n}{2}$$

Sparse vs Dense Graphs

In most applications m is $\Omega(n)$ and $O(n^2)$

(Roughly)

- In a Sparse graph m is $O(n)$ or close to it
- In a dense graph m is close to $\Theta(n^2)$

Adjacency Lists (We use this representation in this course)

if a graph is sparse, then most of entries in adjacency matrix are zero (which is a waste of space). Hence, we generally use adjacency list representation for sparse graphs.

* If G has n vertices & m edges then Adjacency list representation requires $\Theta(m+n)$ space.

Q: Which is better? adjacency matrix or adjacency list.

Ans: Depends on density of the graph and operations we need to form.