Graph representations

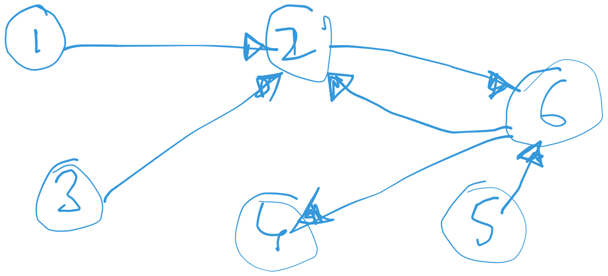
Representation
Traversal

① Adjacency matrix

cardinality
$|V| \times |V|$

$G = (V, E)$

	1	2	3	4	5	6
1		T				
2						T
3		T				
4						
5						T
6		T		T		



2-D matrix of Booleans

Undirected graph : Adj matrix is symmetric
entry (i, j) = entry (j, i)

Max. possible edges in a digraph : $|V|^2$

Undirected graph : $\frac{|V|^2}{2}$ Quadratic

⇒ Lot of storage overhead.

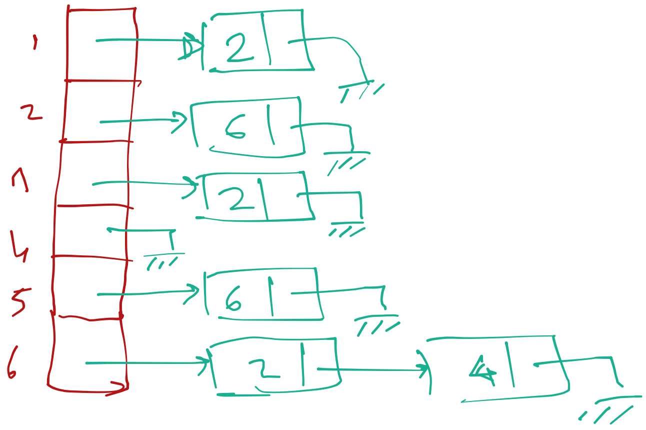
Planar graph : No edges are crossing.

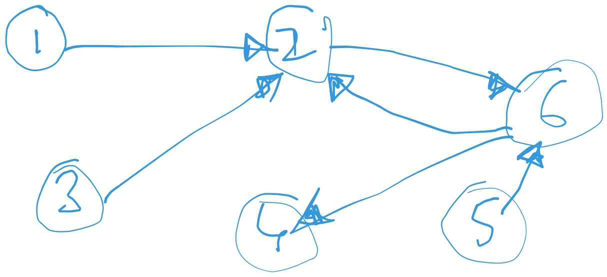
↳ Linear number of edges. $O(|V|)$

Sparse graphs : Far fewer edges than what is possible to store.

② Adjacency list : Collection of linked lists

Each vertex v has a linked list of edges that are outgoing from v.





Difficult to look up.

If vertices are represented by consecutive integers, use an array of lists.

Hash table appl'n.

If the vertices have names ("Boston"), use a hash table to map the strings to the list.

Key : Name of the vertex.
Value : List.

Adjacency list → More space and time efficient for a sparse graph BUT, less efficient for a complete graph
↳ Every possible edge is there

Graph traversals : Visit every vertex ONCE

→ Preorder

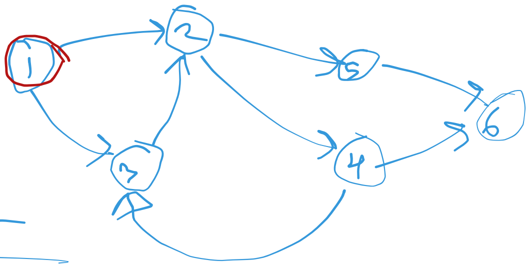
– Depth first Search (DFS)   trave
                                     in a
                                     tree

As deeply and as early as possible

Breadth first Search (BFS) → Level-order traversal

Level by level

Many ways to get from one node to another



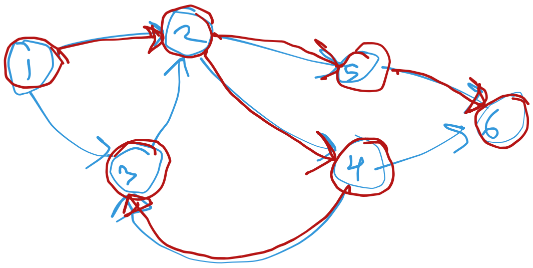
"visited"

Every vertex has a Boolean "visited" field. → tells whether we have visited the vertex before

OR

unordered_map <Node*, bool>

Depth-first -



```
class Graph {
    public:
        void dfs() {
            // Init the visited field for all
            //      the nodes to FALSE.
            for all nodes n in the graph
                if (!n->visited)
                    dfs(n);
        }
};
        void dfs(Node* n)
```

```
{
    n → visit();
    n → visited = true;
    for each vertex p such
      that (n, p) ∈ E
    {
        if (!p → visited) {
            dfs(p);
        }
    }
}
```

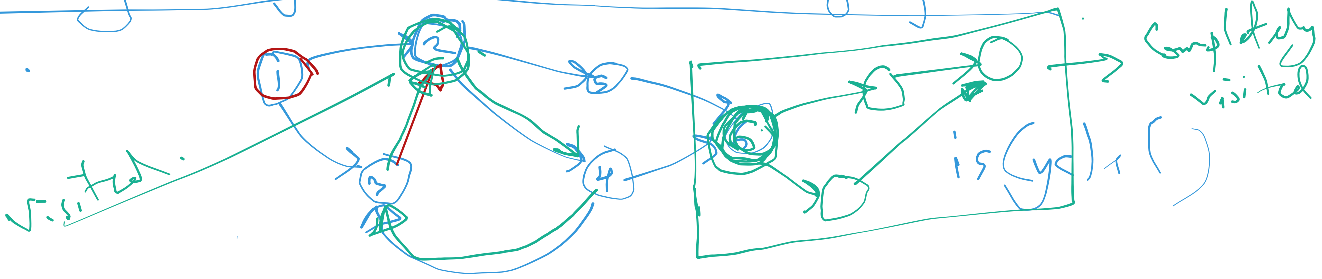
$\Theta(n)$ in adj matrix.

$\Theta(|V| + |E|)$ → Adj. list

$$\left(\frac{n(n-1)}{2}\right)$$

Adj. matrix. → $\Theta(|V|^2)$

Detecting a cycle in a directed graph



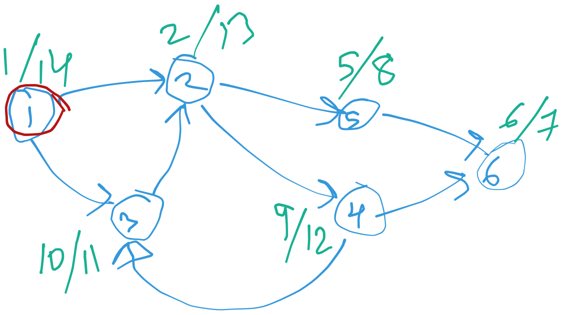
Idea : ~~If~~ a node is already visited, then declare a cycle.

Node (Unvisited) → Visited → Completely visited → all nodes reachable from self

isCycle()

self

Start / Finish times


1/14
2/13
5/8
6/7
9/12
10/11
1
2
3
4
5
6