Graph(I)

Chin Ka Wang {rina__owo}

2025-02-24

Schedule

Part 1	Data structure: Vector
Part 2	Basic Graph Concept
Part 3	Storage of Graphs
Part 4	Trees

Part 1: Data structure: Vector

What data structure we've learnt?

- Array
- Stack
- Queue
- Linked list

```
int arr[10];
stack<int> S;
queue<int> Q;
```

Vector

A ***vector*** is basically same as array but it can adjust its length dynamically.

```cpp
vector<int> V;
int main() {
    V.push_back(3);
    V.push_back(9);
    cout << V[0] << " " << V[1];
}
```

輸出 完成 (0.001s)

```
3 9
```

Vector

```
vector<int> V[110];
vector<vector<int>> V;
```

An 2D array with the 2nd dimension be a vector
An 2D vector

Commonly used functions of vector

- push_back(): insert an element at the back of the vector
- pop_back(): delete the last element of the vector
- insert(): insert an element at a specific position in the vector
- erase(): erase an element by value / iterator in the vector
- clear(): clear the whole vector
- empty(): return a boolean showing whether the vector is empty
- size(): return the number of elements in the vector [be careful of its data type!]

Part 2: Basic Graph Concept

Given N cities numbered 1 to N.
Given some roads connecting different cities.
Find how many routes are there to go to city N from city 1.

輸入

```
4 5
1 2
1 3
2 3
2 4
3 4
```

Given N cities numbered 1 to N.
Given some roads connecting different cities.
Find how many routes are there to go to city N from city 1.

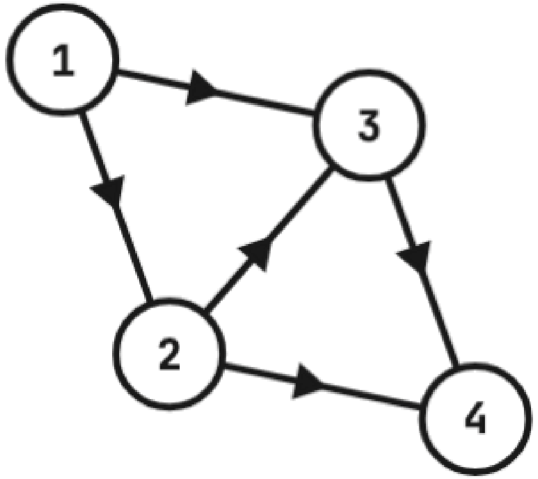
輸入

```
4 5
1 2
1 3
2 3
2 4
3 4
```

The first line consists of 2 integers N,M representing the number of cities and roads resp.

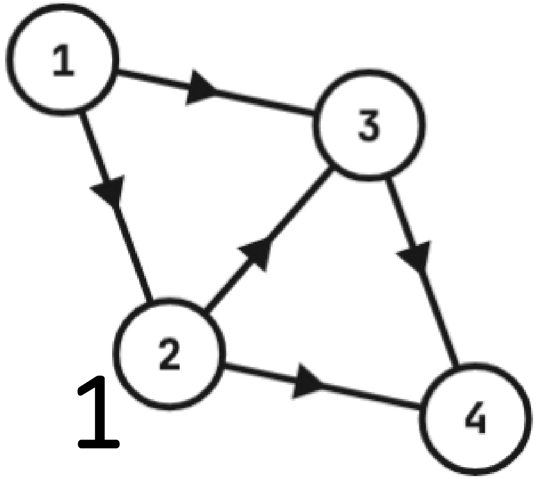
For the next M lines, each line consists of 2 integers u and v representing a road that allows you to go from city u to city v.

How many routes are there from 1 to 4?

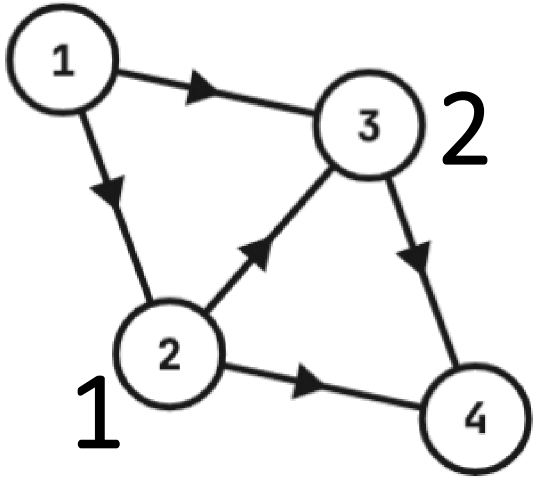


Now we can use the method of Dynamic Programming (DP)

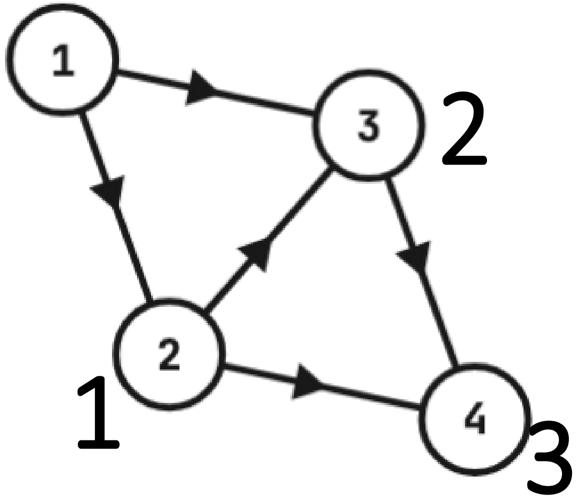
How many routes are there from 1 to 4?



How many routes are there from 1 to 4?



How many routes are there from 1 to 4?

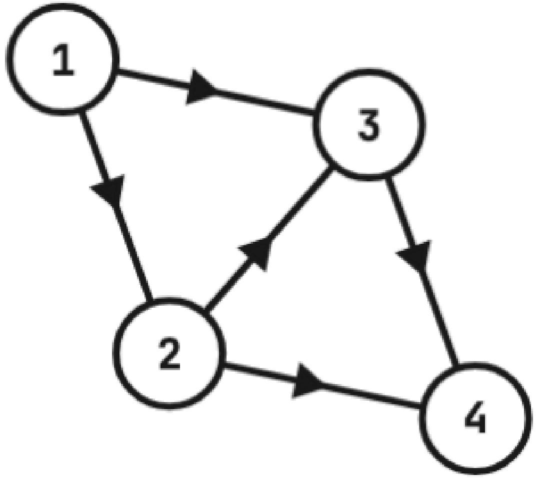


The answer is 3.

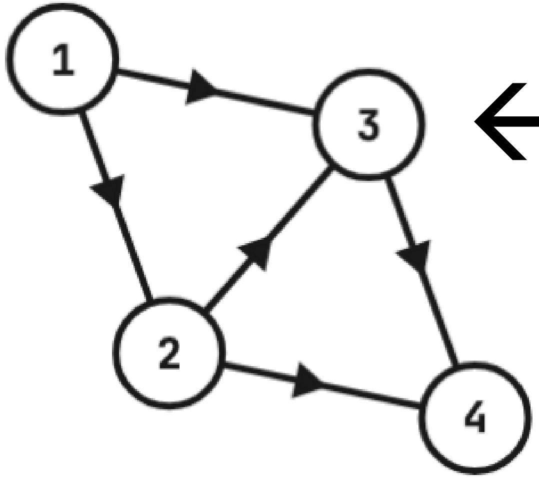
輸入

```
4 5
1 2
1 3
2 3
2 4
3 4
```

→

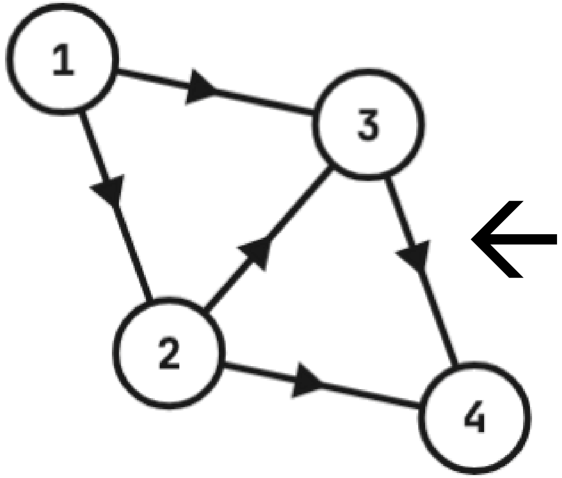


Graph



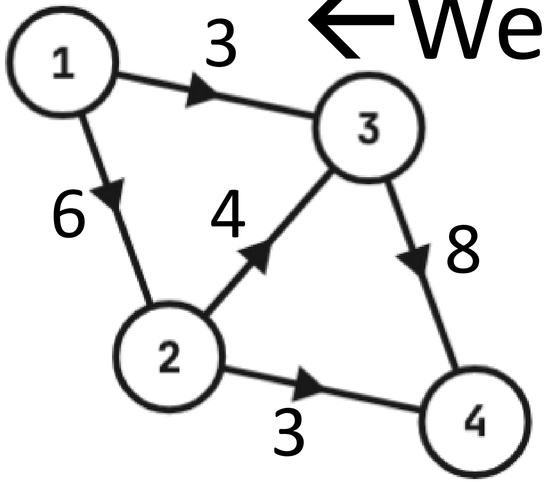
←Nodes/Vertices (Cities)

Graph



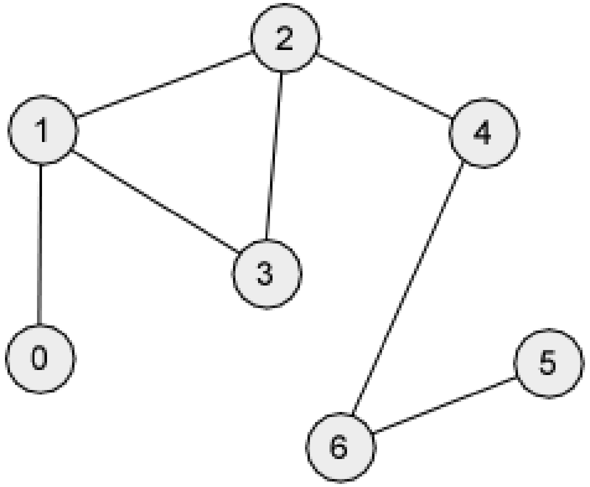
←Edges (Road)

Graph

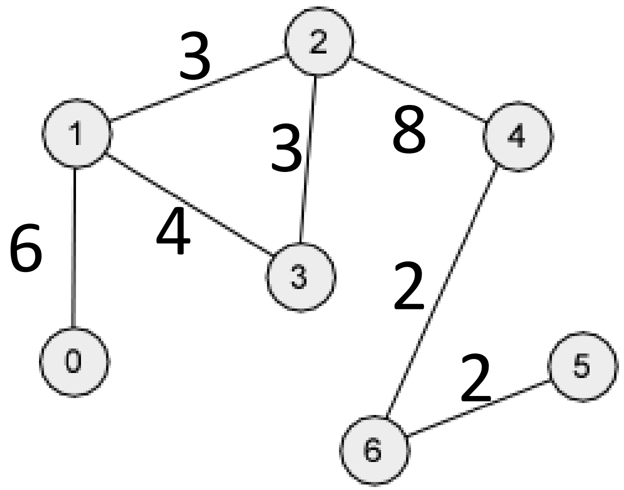


←Weights (Length of the road)

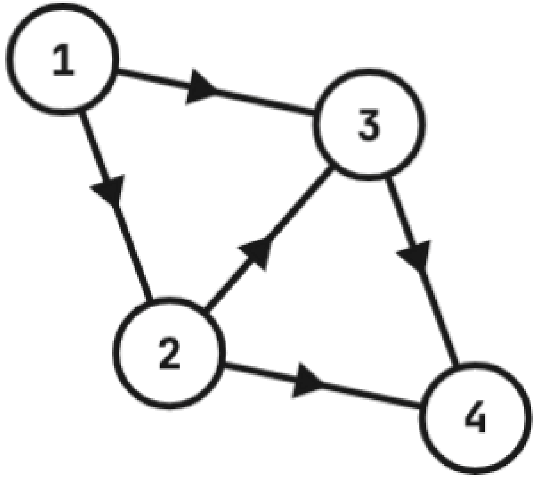
Unweighted Undirected Graph



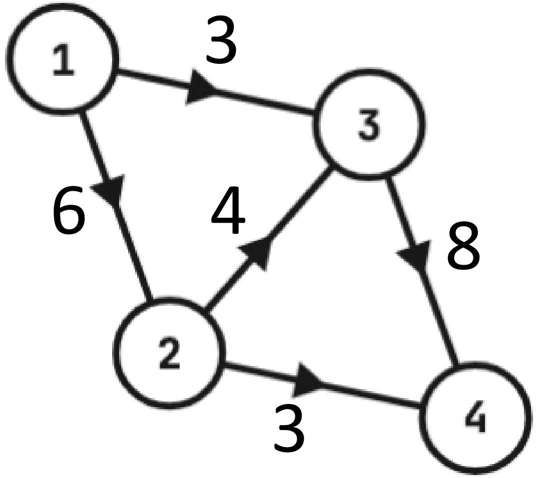
Weighted Undirected Graph



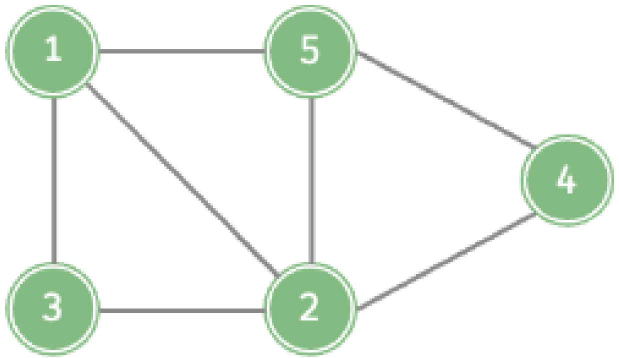
Unweighted Directed Graph



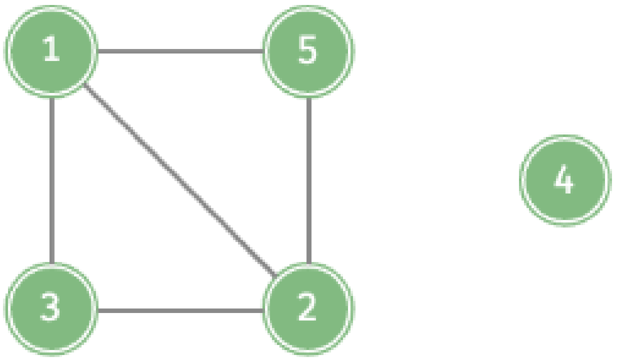
Weighted Directed Graph



Connectivity of Non-Directed Graph



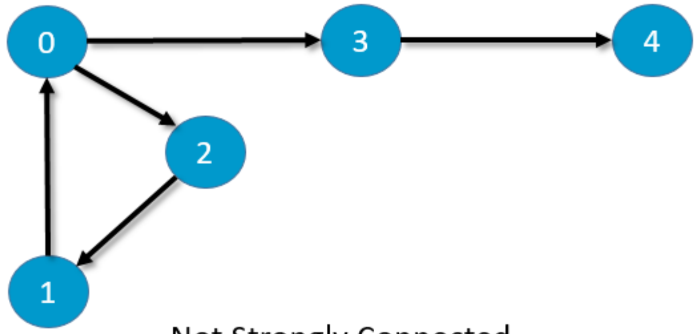
Connected graph



Disconnected graph

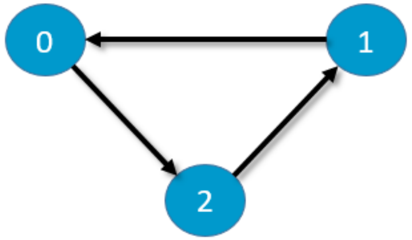
Connectivity of Directed Graph

Graph 1:



Not Strongly Connected

Graph 2:

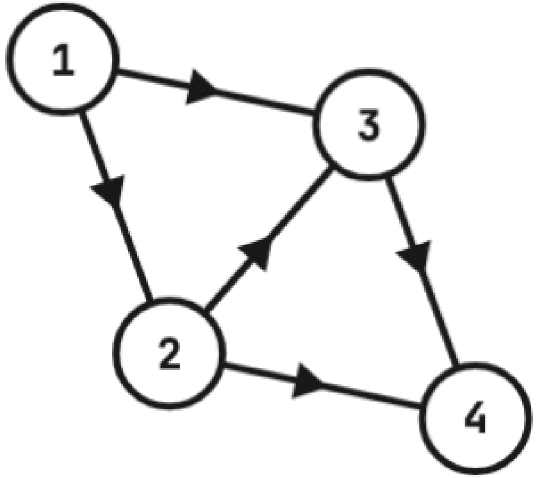


Strongly Connected

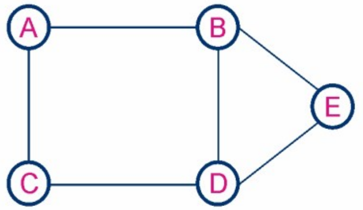
Part 3: Storage of Graphs

輸入

```
4 5
1 2
1 3
2 3
2 4
3 4
```

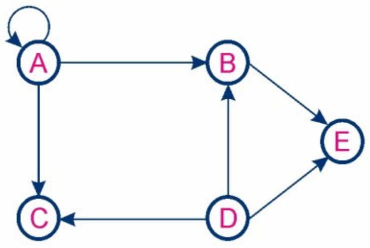


Adjacency Matrix



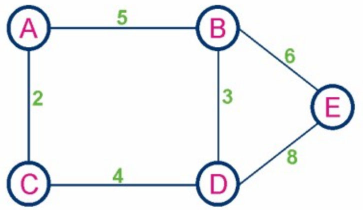
Undirected Graph

	A	B	C	D	E
A	0	1	1	0	0
B	1	0	0	1	1
C	1	0	0	1	0
D	0	1	1	0	1
E	0	1	0	1	0



Directed Graph

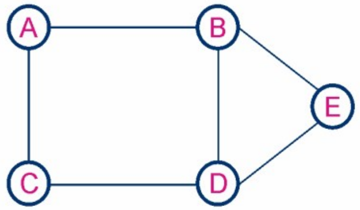
	A	B	C	D	E
A	1	1	1	0	0
B	0	0	0	0	1
C	0	0	0	0	0
D	0	1	1	0	1
E	0	0	0	0	0



Weighted Graph

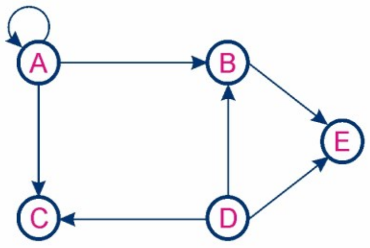
	A	B	C	D	E
A	0	5	2	0	0
B	5	0	0	3	6
C	2	0	0	4	0
D	0	3	4	0	8
E	0	6	0	8	0

Adjacency Matrix



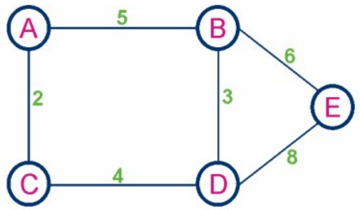
Undirected Graph

	A	B	C	D	E
A	0	1	1	0	0
B	1	0	0	1	1
C	1	0	0	1	0
D	0	1	1	0	1
E	0	1	0	1	0



Directed Graph

	A	B	C	D	E
A	1	1	1	0	0
B	0	0	0	0	1
C	0	0	0	0	0
D	0	1	1	0	1
E	0	0	0	0	0



Weighted Graph

	A	B	C	D	E
A	0	5	2	0	0
B	5	0	0	3	6
C	2	0	0	4	0
D	0	3	4	0	8
E	0	6	0	8	0

Just use a 2D array!

Adjacency Matrix

```
int N,M,u,v;
int adj[110][110];
int main() {
    cin >> N >> M;
    for (int i = 1; i <= M; ++i)
    {
        cin >> u >> v;
        adj[u][v] = 1;
    }
}
```

輸入

```
4 5
1 2
1 3
2 3
2 4
3 4
```

→

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

Adjacency Matrix

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

Lots of spaces are wasted!!

Adjacency Matrix

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

It consumes space of V^2 where V is the number of vertices.

Space complexity: O(V^2)

Adjacency Matrix

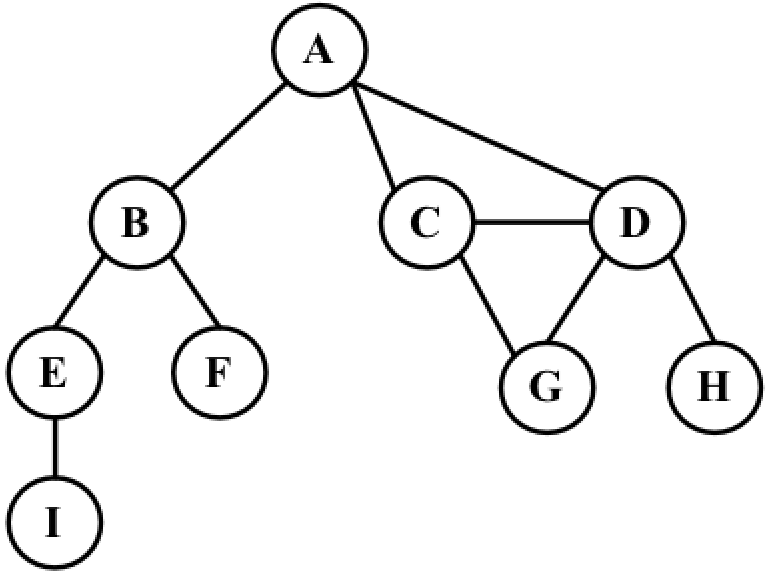
	1	2	3	4
1	0	1	1	0
2	0	0	1	1
3	0	0	0	1
4	0	0	0	0

However, the no. of edges are usually much fewer than square of vertices.

Adjacency List

An adjacency list stores the edges instead of listing out all relationships between each pair of vertices!

Adjacency List



A	B C D
B	A E F
C	A D G
D	A C G H
E	B I
F	B
G	C D
H	D
I	E

For each vertex, it stores what other vertices it can go.

Adjacency List

```cpp
int N,M,u,v;
vector<int> adj[110];
int main() {
    cin >> N >> M;
    for (int i = 1; i <= M; ++i)
    {
        cin >> u >> v;
        adj[u].push_back(v);
    }
}
```

輸入

```
4 5
1 2
1 3
2 3
2 4
3 4
```

→

1	2	3
2	3	4
3	4	
4		

Adjacency List

1	2	3
2	3	4
3	4	
4		

It only consumes space of E where E is the number of edges!

Space complexity: O(E)

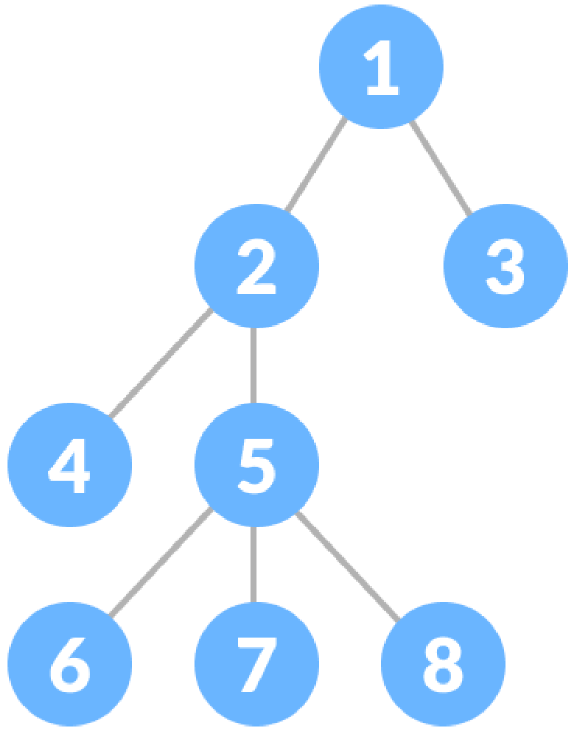
Part 4: Trees

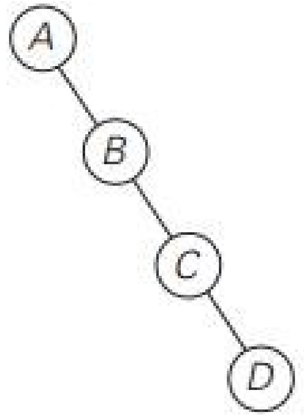
What is a tree?

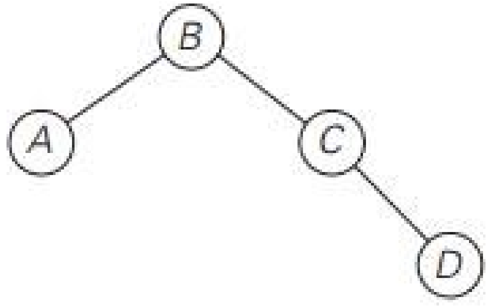
A tree is an undirected connected graph that doesn't have cycles.

If the tree has n vertices, the tree must have $n - 1$ edges.

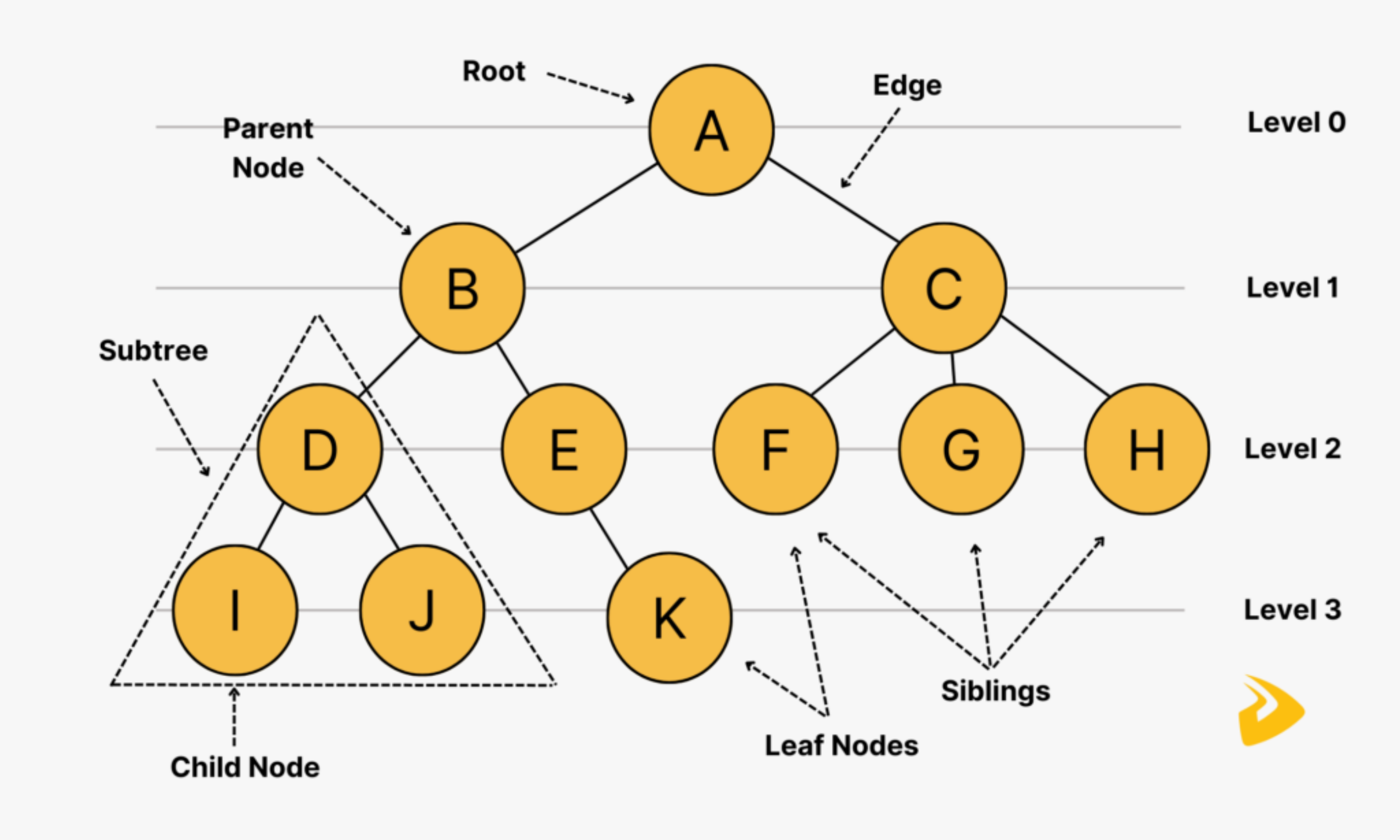
Examples of Trees



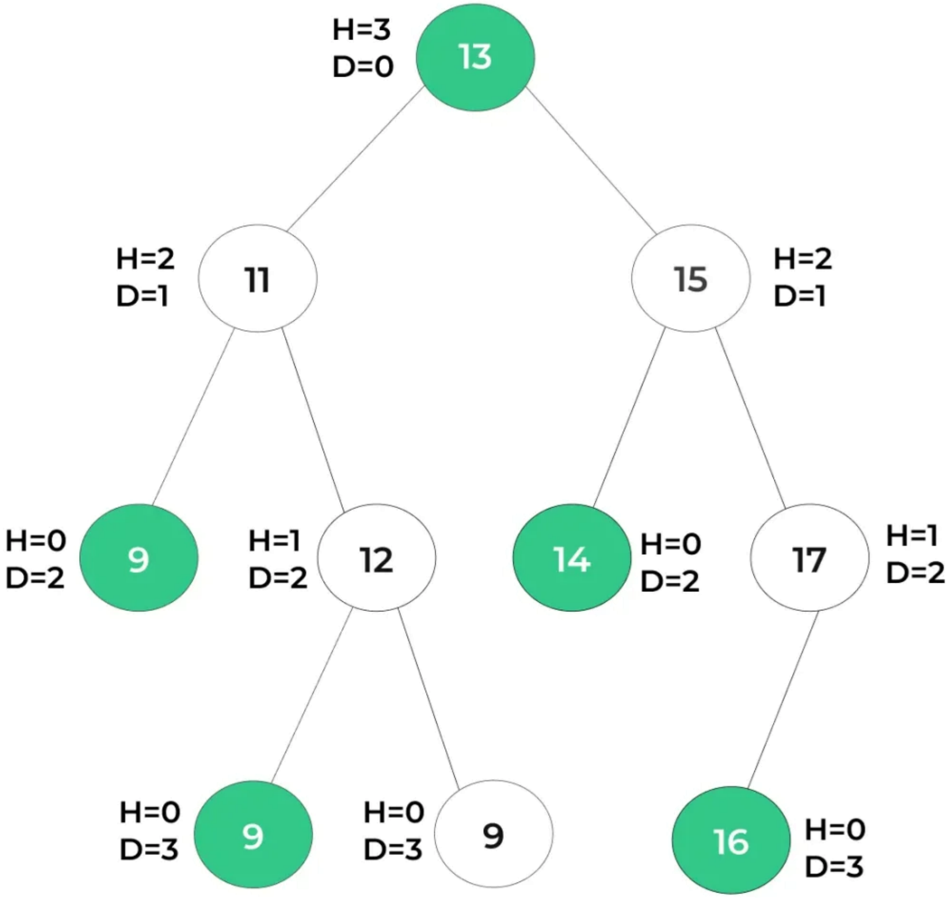




Structure of a Tree



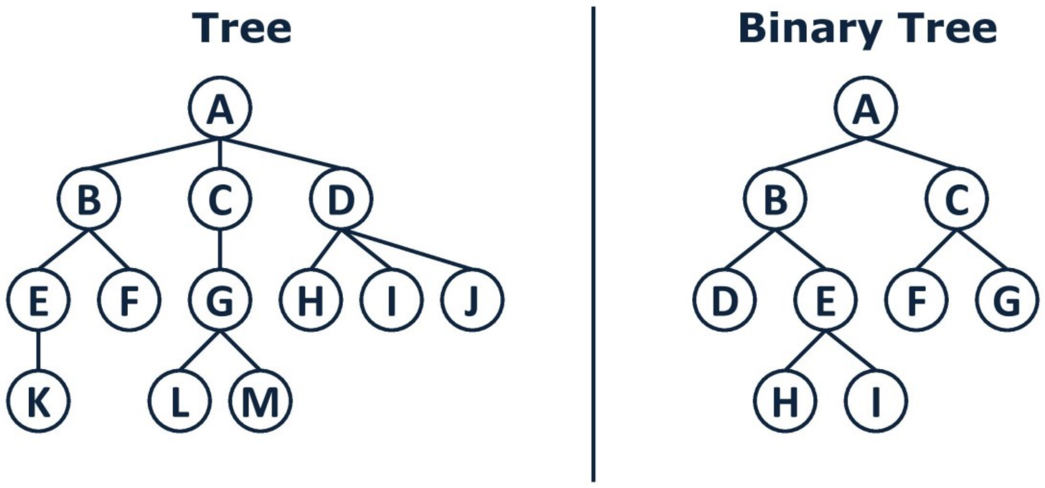
Structure of a Tree



Here,
H= Height of the Node
D=Depth of the Node

Binary Trees

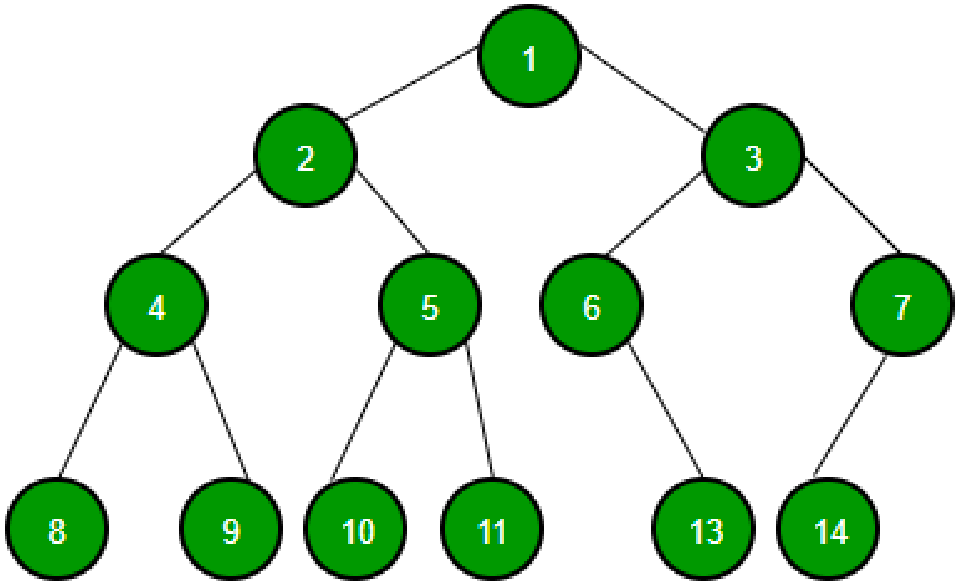
A binary tree is a tree data structure in which each node has at most two children, referred to as the left child and the right child.



Storage of Binary Trees

- Adjacency List
- Using Linear Array

How to use linear array to store a binary tree?

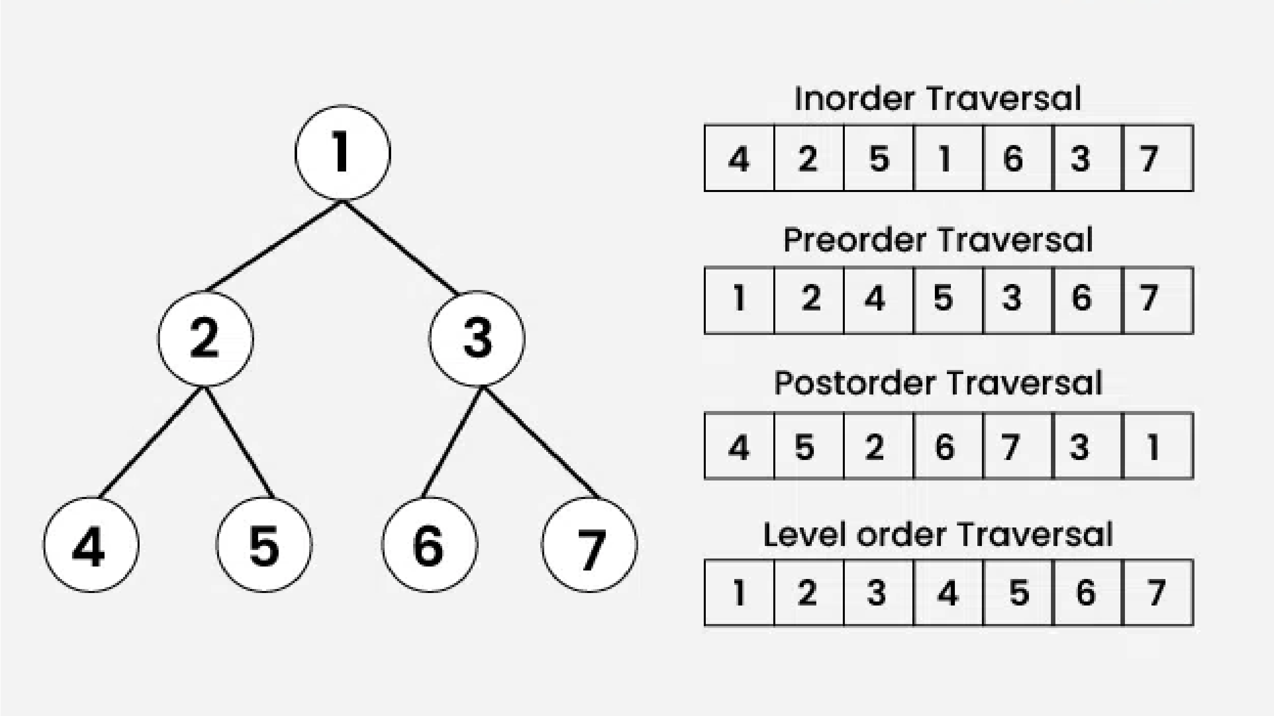


Note that for each node N,
N's left child = $2N$,
N's right child = $2N + 1$.

Traverse of Binary Trees

- Inorder Traversal 中序遍歷（左-根-右）
- Preorder Traversal 先序遍歷（根-左-右）
- Postorder Traversal 後序遍歷（左-右-根）
- Level order Traversal 層序遍歷

Traverse of Binary Trees



ANY QUESTIONS?

Practice Time