DSA → 4.2

→ Revision of DSA Topics

→ Maths → inverse mod & problems

→ Backtracking

→ Trie → 2

→ String Matching

→ Dp → 2

→ Graphs → 2

→ Contest.

Today's Agenda →

→ Recursion

→ D.P

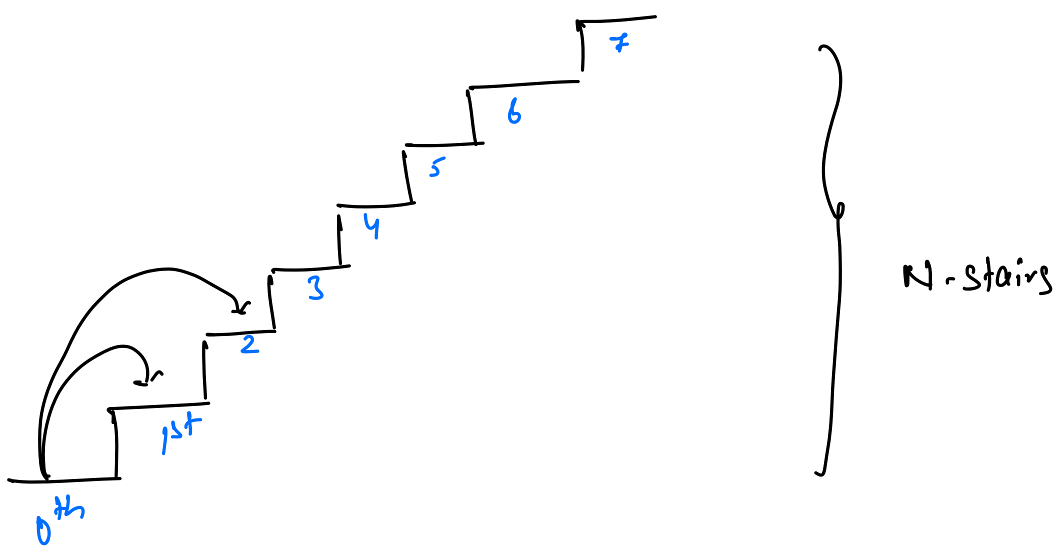
→ Graphs [B.F.S & D.F.S]

1

Scaler has opened an Adventure Park and there is a staircase. Every staircase has a cost associated that you need to pay when you claims it.

You can either start from the 0th staircase or the 1st staircase.

Goal: What is the least amount of money you can spend to reach the top.



Eg → [10, 20, 5, 8, 15, 25, 10, 12]

0	1	2	3	4	5	6	7
10	20	5	8	15	25	10	12

ans = 52

idea 1 → Recursion

Min. Amount(N) → MinAmount(N-1), MinAmount(N-2)

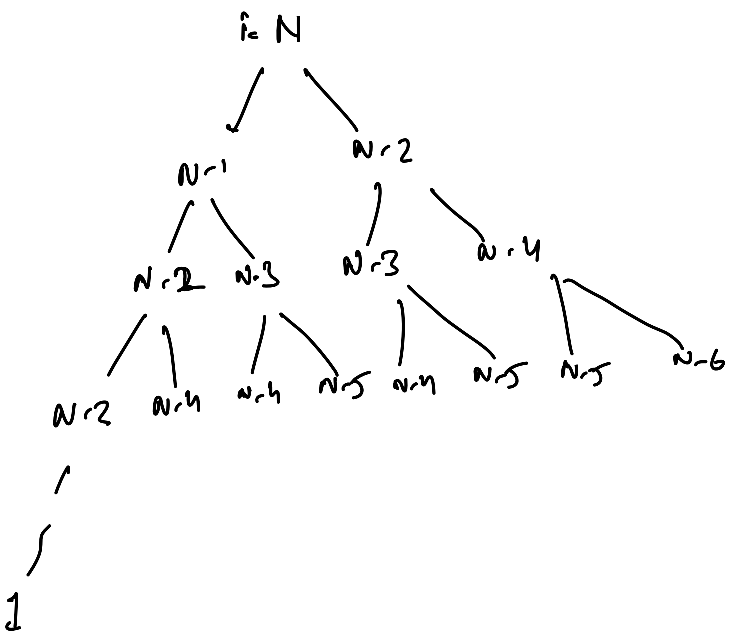
→ Optimal substructure
→ Overlapping sub-problems

→ D.P

Recursive code →

```
int mincost ( int[] arr, int i ){
    if (i == 0 || i == 1) { return arr[i] }

    return Min ( mincost( arr, i-1) , mincost( arr, i-2)) + arr[i] ;
}
```

$$\boxed{\begin{array}{l} T.C \rightarrow O(2^N) \\ S.C \rightarrow O(N) \end{array}}$$



Optimisation

→ Memoization (Top-down Approach)

① → decide storage

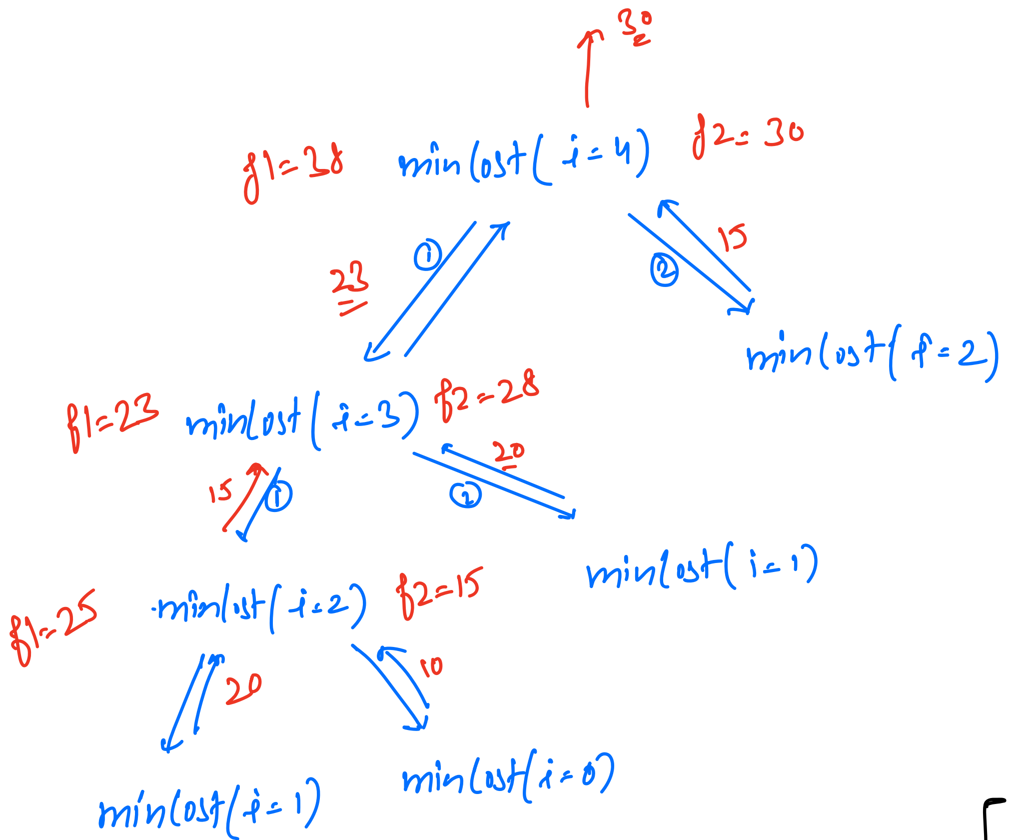
② → store your answer in storage before returning it.

③ → check if answer is pre-calculated.

```
int dp[N];     ∀i, dp[i] = -1

int mincost( int[] arr, int i , int[] dp ){
    if (i == 0 || i == 1) { return arr[i] }

    if( dp[i] != -1) { return dp[i] }

    int f1 = mincost(arr, i-1 , dp) + arr[i];   — ①

    int f2 = mincost(arr, i-2, dp ) + arr[i];   — ②

    dp[i] = min( f1, f2);

    return min( f1, f2);
}
```

$$arr[\,] \rightarrow \begin{array}{ccccc} [10, & 20, & 5, & 8, & 15] \\ 0 & 1 & 2 & 3 & 4 \end{array}$$

$$dp[\,] \rightarrow \begin{array}{ccccc} & & 15 & 23 & 30 \\ [-1 & -1 & \cancel{-1} & \cancel{-1} & \cancel{-1}] \\ 0 & 1 & 2 & 3 & 4 \end{array}$$



$$\begin{bmatrix} T.C \rightarrow O(N) \\ S.C \rightarrow O(N) \end{bmatrix}$$

Tabulation → (bottom-up Approach)

$$arr[\,] \rightarrow \begin{array}{ccccc} [10, & 20, & 5, & 8, & 15] \\ 0 & 1 & 2 & 3 & 4 \end{array}$$

$$dp[\,] \rightarrow \begin{array}{ccccc} [10 & 20 & 15 & 23 & 30] \\ 0 & 1 & 2 & 3 & 4 \end{array}$$

$dp[i]$ → min cost required to reach i^{th} - stair.

code →

```
int dp[N];

dp[0] = arr[0],   dp[1] = arr[1];

for( i=2; i<N ; i++){
        dp[i] = Min( dp[i-1], dp[i-2]) + arr[i];
}

return dp[N-1];
```

$\left[\begin{array}{l} T.C \rightarrow O(N) \\ S.C \rightarrow O(N) \end{array}\right]$

Further S.C. optimisation →

```
int a = arr[0], b = arr[1];

for( i=2; i<N; i++){
        c = min(a,b) + arr[i];
        a = b;
        b = c;
}

return b;
```

$\left[\begin{array}{l} T.C \rightarrow O(N) \\ S.C \rightarrow O(1) \end{array}\right]$

Graphs.

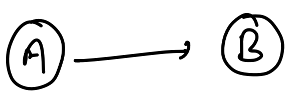
↳ collection of nodes & edges.

Difference b/w tree & graph →

1. Trees are hierarchical unlike graphs.
2. Trees always have root node.
3. Cycle will not be there in trees.
4. In trees, if there are N nodes, then no. of edges will be N-1.

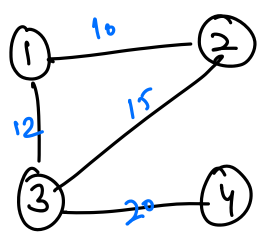
Types of Graph →

1. Directed

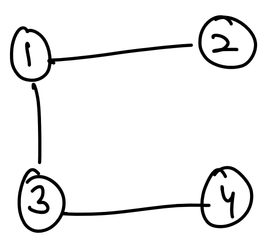


un-directed

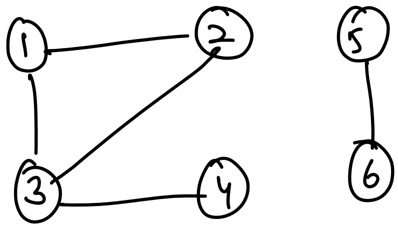
2. weighted



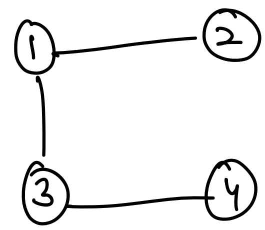
Un-weighted



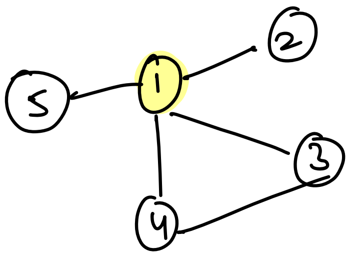
3 **Cyclic**



Acyclic



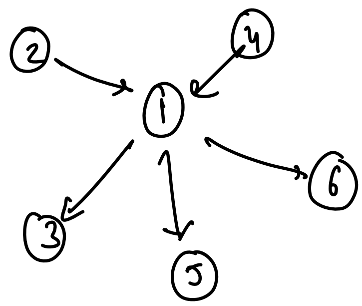
4 **Degree**



degree(1) = 4

degree(4) = 2

in/out - degree



in-degree(1) = 2

out-degree(1) = 3

How graph is given as i/p. ⇒

i/p → 1st line. ⇒ no. of nodes (N) no. of edges. (E)

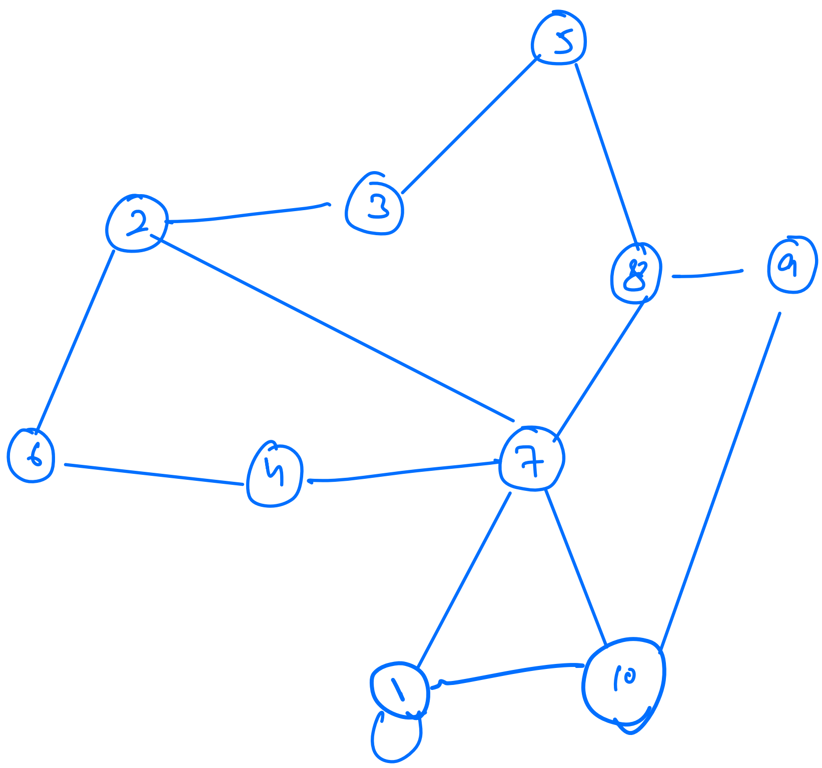
followed by E lines. u, v.

An edge between u and v.

Eg →

10 14 (N → 10, E → 14)

→ 2 3 ✓
→ 4 7 ✓
→ 8 9 ✓
→ 2 7 ✓
→ 7 8 ✓
→ 10 1 ✓
→ 4 6 ✓
→ 5 8 ✓
→ 2 6 ✓
→ 10 9 ✓
→ 7 10 ✓
→ 3 5 ✓
→ 7 1 ✓
→ 1 1



→ Adjacency matrix
→ Adjacency list

How to store a graph →

① Adjacency matrix →

N = 5 E = 6

1 2

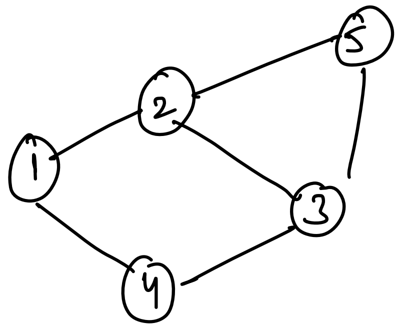
1 4

2 3

3 4

2 5

3 5



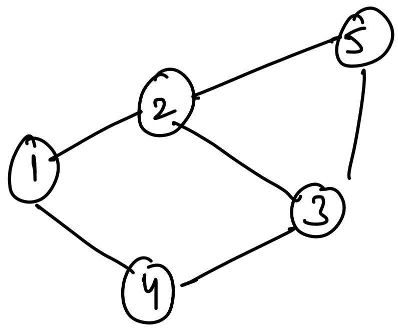
	1	2	3	4	5
1	0	1	0	1	0
2	1	0	1	0	1
3	0	1	0	1	1
4	1	0	1	0	0
5	0	1	1	0	0

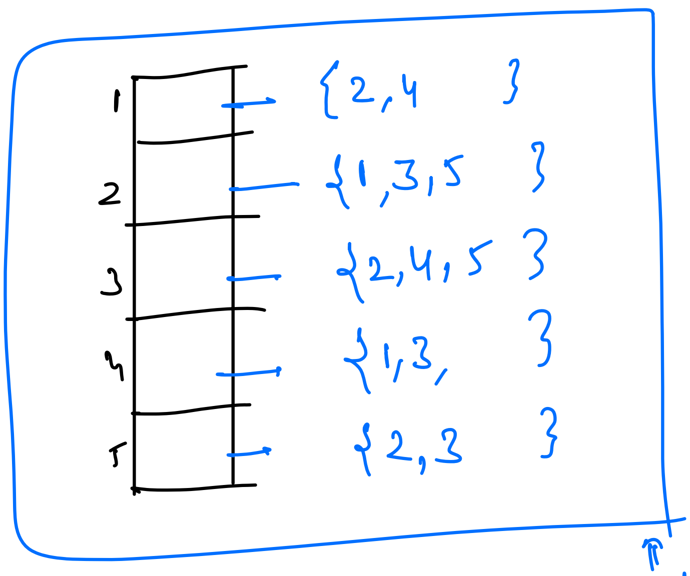
mat[N][N]

② Adjacency list →

N = 5 E = 6

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





adjacency list

int arr [N]

List <integer> graph [N];

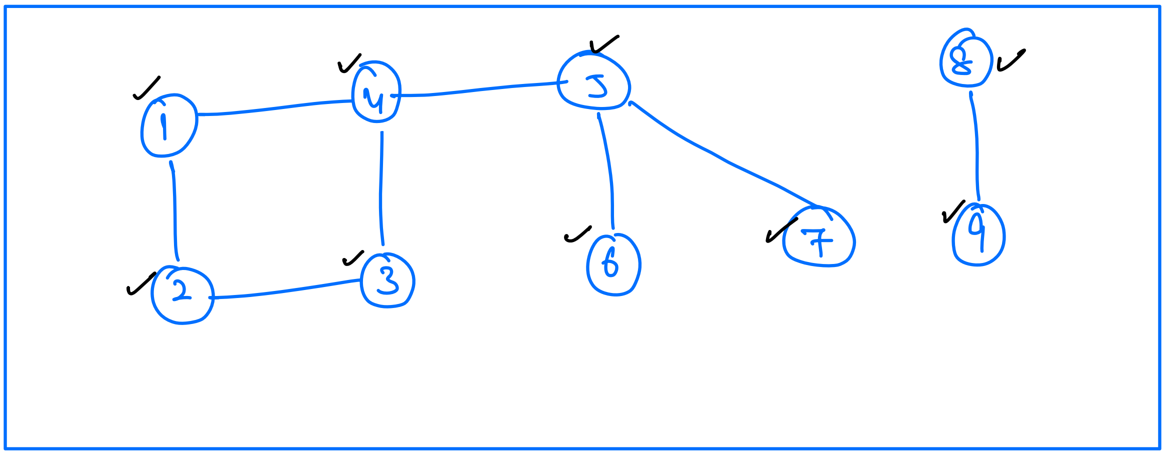
for weighted graph

List <Pair <int, int> > graph [N];

Traversals in a graph →

① D.F.S. (Depth First Search) (Pre, Post, In)

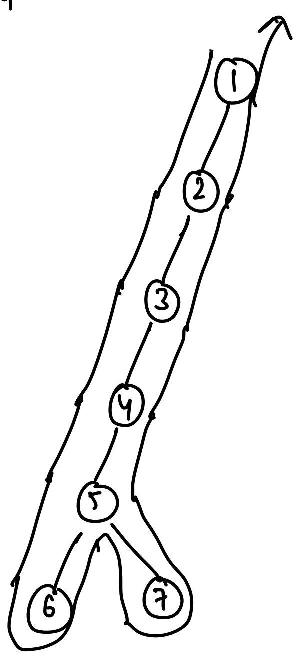
N = 9, E = 8

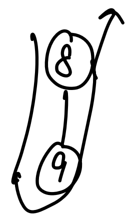


visited[] → [

	1	2	3	4	5	6	7	8	9
	t	t	t	t	t	t	t	t	d
	~~f~~	~~f~~	~~f~~	~~f~~	~~f~~	~~f~~	~~f~~	~~f~~	~~f~~

]





code →

→ List<int>[] graph; (input)

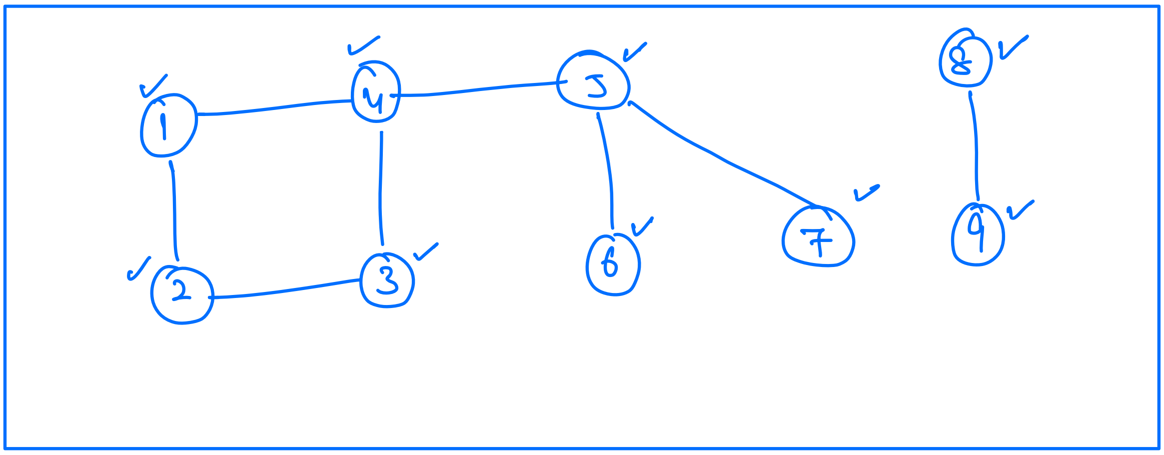
boolean visited[N+1], ∀i, visited[i] = false;

```
for( i = 1; i <= N; i++){
    if( visited[i] == false){
        dfs( graph, i, visited);
    }
}

void dfs( List<int>[] graph, int src, boolean[] visited){
    print(src);
    visited[src] = true;
    for( int nbr : graph[src]){
        if( !visited[nbr]){
            dfs( graph, nbr, visited);
        }
    }
}
```

T.C → $O(N+E)$
S.C → $O(N)$

Breadth First Search (B.F.S) → level order traversal

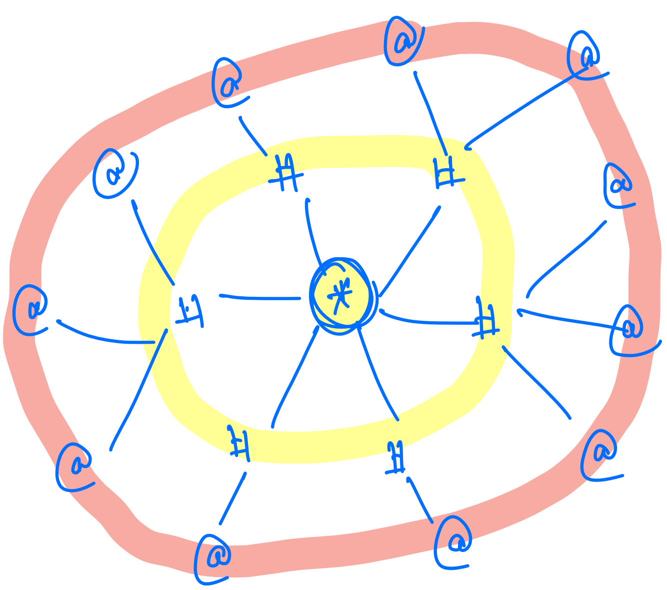
N = 9, E = 8



visited[] → [$\overset{t}{\cancel{f}}_1$ $\overset{t}{\cancel{f}}_2$ $\overset{t}{\cancel{f}}_3$ $\overset{t}{\cancel{f}}_4$ $\overset{t}{\cancel{f}}_5$ $\overset{t}{\cancel{f}}_6$ $\overset{t}{\cancel{f}}_7$ $\overset{t}{\cancel{f}}_8$ $\overset{t}{\cancel{f}}_9$]

| ~~1~~ | ~~2~~ | ~~4~~ | ~~3~~ | ~~5~~ | ~~6~~ | ~~7~~ | ~~8~~ | ~~9~~ | |

B.F.S. → 1, 2, 4, 3, 5, 6, 7, 8, 9



⇒ B.F.S grows radially.

code:→

```
→   List<int> [] graph;        (input)

boolean visited[N+1],   ∀i, visited[i] = false;

for( i = 1; i ≤ N; i++){
    if( visited[i] = = false){
        bfs (graph, i, visited);
    }
}

void  bfs( List<int> [] graph, int src, boolean[] visited){
    Queue <int> q;
    q.enqueue(src),  visited[src] = true;
    print(src);
    while(!q.isEmpty()){
        int rn = q.dequeue();
        for( int nbr : graph[rn]){
            if( visited(nbr) == false){
                visited[nbr] = true;
                q.enqueue(nbr);
                print(nbr);
            }
        }
    }
}
```

$\boxed{\begin{array}{l} T.C \to O(N+E) \\ S.C \to O(N) \end{array}}$

---x---x---

3 10 4 20 7 11 30 18 29 50

3 4 7 10 11 18 20 29 30 40 50

7 → (4,7) → 3

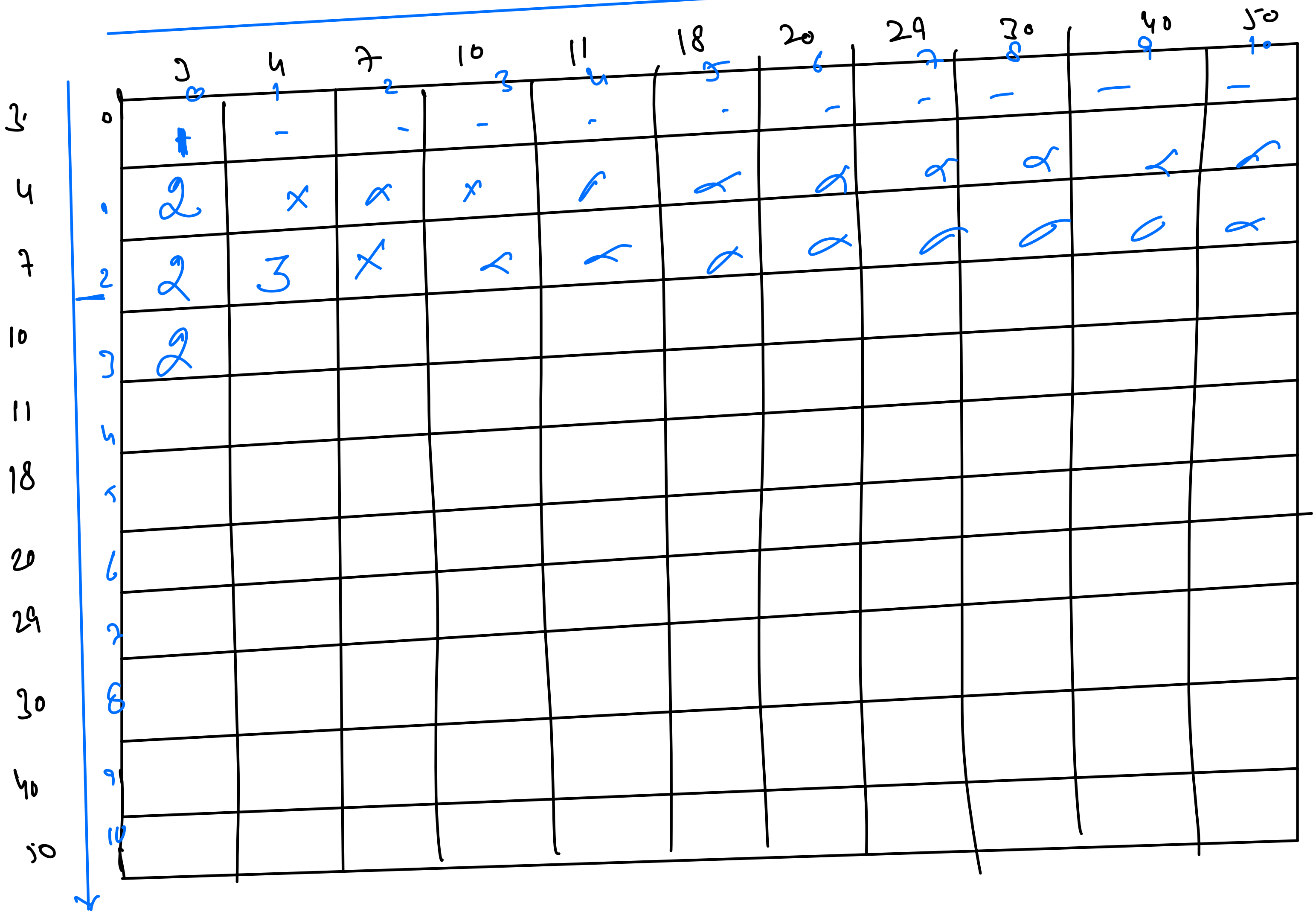
10 → (7,10) → 3

11 → (7,11) → 4

18 → (7,11) → 5

29 → (18,29) → 6

3 4 7 10, 11 18 20 29 30 40, 50



{ length of longest fib like subsequence where last 2 elements are

_, arr(i) }

arr[j], arr[i].