SAPTARSHI MUKHERJEE

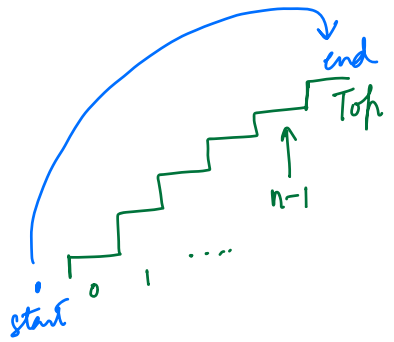
About Myself

→ SWE III @ Google
→ 3 times ACM-ICPC Regionals
→ 3 times in top 2000 in FB/Meta Hacker Cup
→ Master at Codeforces
→ 6 star at Codechef
→ BTech Hons. in CSE @ IIT Bhilai
→ Former member of National Blockchain Project, IIT-K
→ Author @ CCGrid '21

8) Scaler Adventure Park

Has a staircase that needs to be climbed.

n stairs → 0 to n-1.

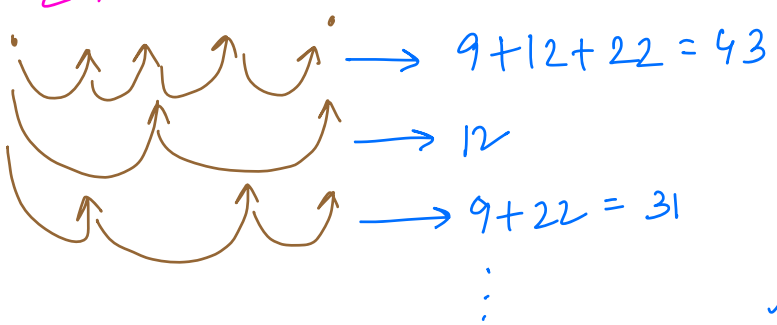


At a time, 1 or 2 stairs.

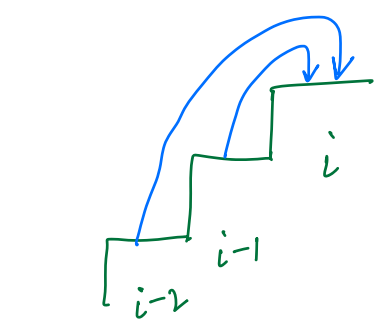
Each stair has a cost.

What is the min cost to reach the top?

A → {9, 12, 22}



A → {10, 20, 5, 8, 15, 25, 10, 12}



minCost(i) → minCost(i-1) + A[i]

minCost(i) → minCost(i-2) + A[i]

subproblems.

Sub-Problem

Problem → nth stair

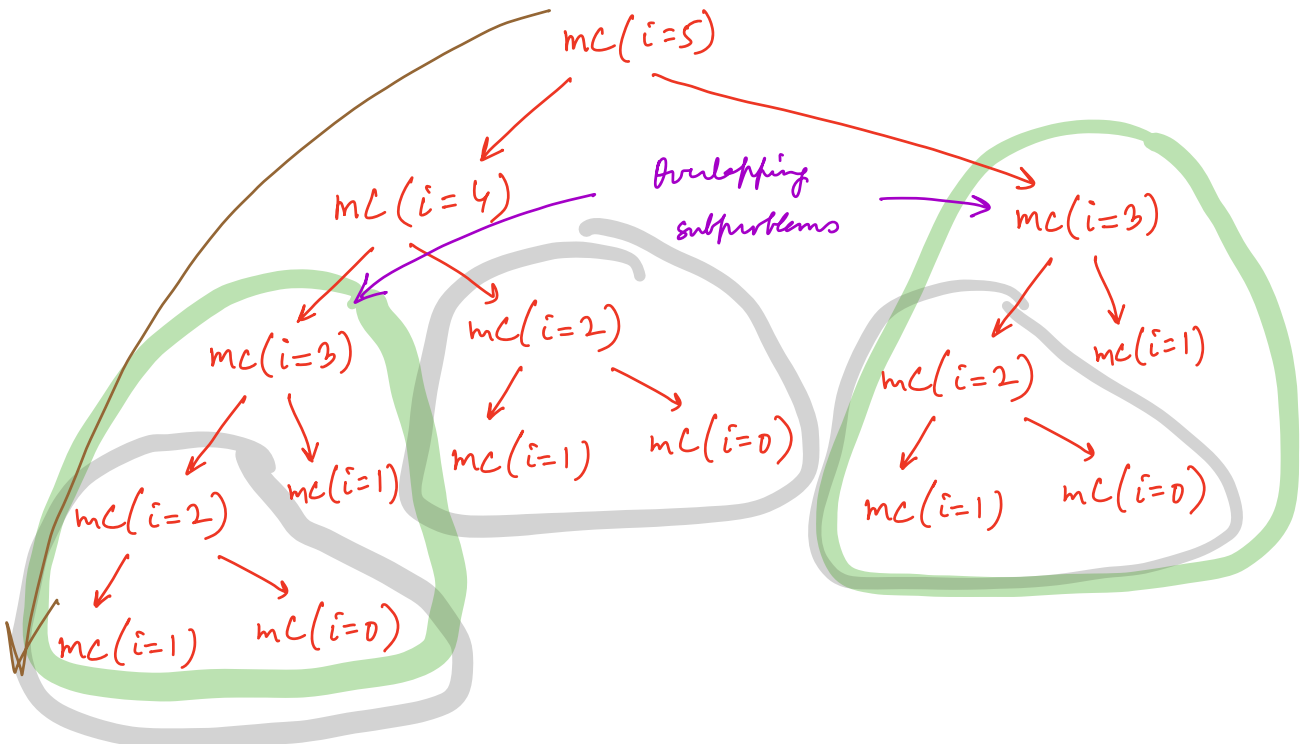
→ min(minCost(n-1), minCost(n-2)).

```
fn minCost(A[], i) {
    if (i < 2)
        return A[i]
    return min(minCost(A, i-1), minCost(A, i-2)) + A[i]
}
```

Ans → min(minCost(A, n-1), minCost(A, n-2))

$O(2^n)$ T.C

$O(n)$ S.C.



Optimal substructure → Answer to Problem can be computed from the answers to subproblems.

Overlapping subproblems → Some subproblems are repeating

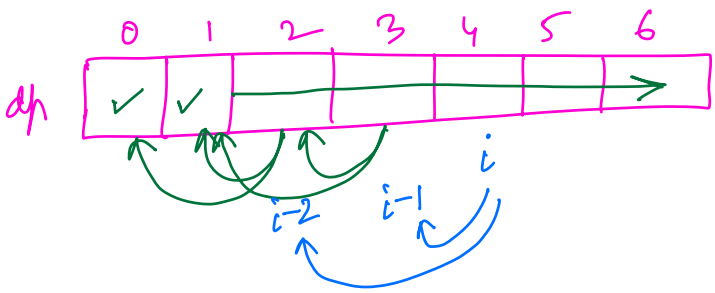
→ Dynamic Programming (DP).

Top Down DP (Memorization DP)

```
int dp[n] → {all -1}
fn minCost(A[], i) {
    if(i<2)
        return A[i]
    if(dp[i] != -1)
        return dp[i];
    dp[i] = min(minCost(A,i-1), minCost(A,i-2)) + A[i]
    return dp[i];
}
Ans → min(minCost(A,n-1), minCost(A,n-2))
```

O(n) T.C.
O(n) S.C.

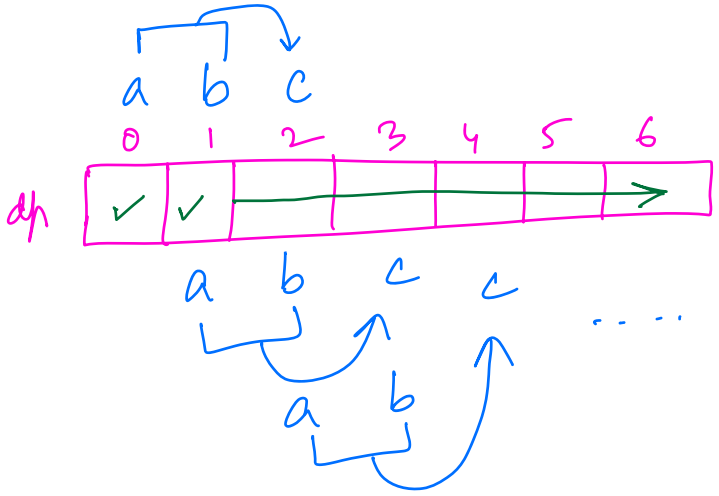
Bottom Up DP (Tabulation DP)




```
int dp[n] → {all -1}
dp[0] = A[0], dp[1] = A[1]
fn i → 2 to n-1 {
    dp[i] = min(dp[i-1], dp[i-2]) + A[i]
}
return min(dp[n-1], dp[n-2])
```

O(n) T.C
O(n) S.C

Space optimisation




```
cost1 = A[0]
cost2 = A[1]
for i → 2 to n-1 {
    cost = min(cost1, cost2) + A[i]
    cost1 = cost2
    cost2 = cost
}
return min(cost1, cost2)
```

i-2	i-1	i
cost1	cost2	cost
	cost1	cost2

O(n) T.C.
O(1) S.C.

[Break till 10:34 PM]

Graphs

Nodes connected by edges.

Tree vs Graphs

→ Tree has a hierarchy

→ Tree is acyclic, connected, with n−1 edges.

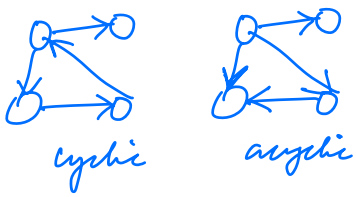
- Directed vs Undirected graphs

(Insta) (FB)

- Weighted vs Unweighted graphs
- Cyclic vs Acyclic graph

Undirected →

Directed →



How is graph given as I/P ?

#nodes (n) #edges (m)

u_1 v_1

u_2 v_2

u_3 v_3

⋮

u_m v_m

} m edges

4 6

1 2

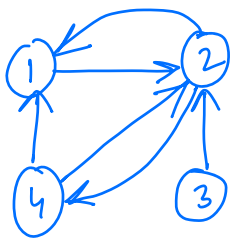
3 2

4 2

4 1

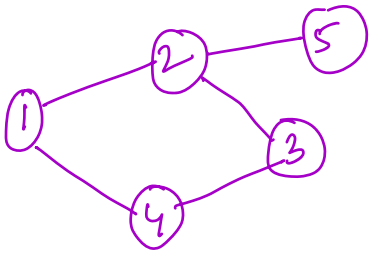
2 1

2 4

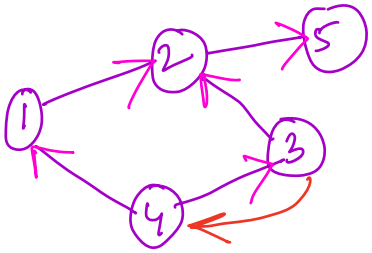


Ways to store a graph

A



B



Adjacency Matrix

M[i][j] = 1 if there is an edge from i to j.
0 otherwise.

A

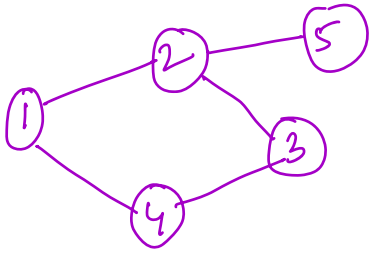
	0	1	2	3	4	5
0						
1		0	1	0	1	0
2		1	0	1	0	1
3		0	1	0	1	0
4		1	0	1	0	0
5		0	1	0	0	0

B

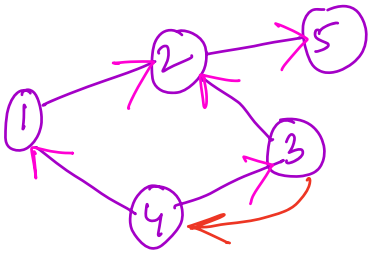
	0	1	2	3	4	5
0						
1		0	1	0	0	0
2		0	0	0	0	1
3		0	1	0	1	0
4		1	0	1	0	0
5		0	0	0	0	0

$O(n^2)$ space.

A



B



Adjacency list

Array of lists ⟶ adj[n]. adj[i] = list of neighbours of i.

A

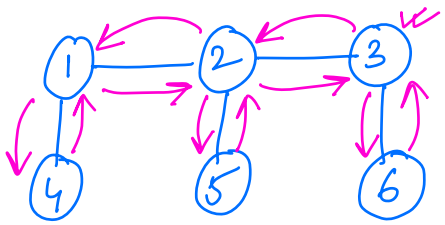
adj
0 →
1 → {2, 4}
2 → {1, 3, 5}
3 → {2, 4}
4 → {1, 3}
5 → {2}

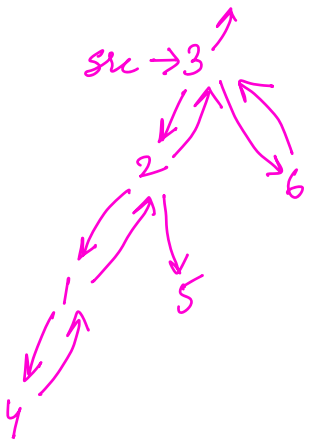
B

adj
0 →
1 → {2}
2 → {5}
3 → {2, 4}
4 → {1, 3}
5 → {}

O(n+m) space.

Depth First Traversal (DFS)






```
fn dfs(adj[], curr, vis){
    if(vis[curr])
        return;
    vis[curr] = true
    // Do anything
    for nbr in adj[curr] {
        dfs(adj, nbr, vis)
    }
}
```

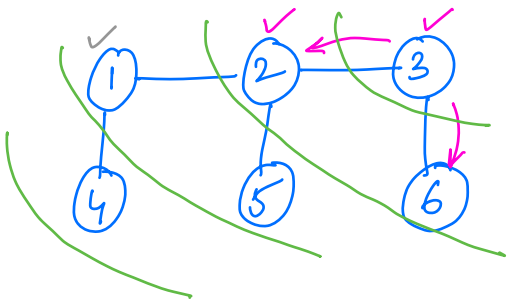
$O(n+m)$ T.C.

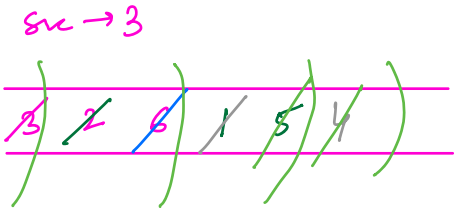
$O(n)$ S.C

↳ vis array.

```
fn dfs(adj, src){
    boolean vis[n] = {all false}
    dfs(adj, src, vis)
}
```

Breadth First Traversal (BFS)






```
fn bfs(src, adj) {
    boolean vis[n] = {all false};
    Queue q = { }
    q.add(src)
    vis[src] = true;
    while(!q.isEmpty()) {
        v = q.front()
        q.removeFront()  // or poll()
        // Do anything
        for(nbr in adj[v]) {
            if(!vis[nbr]) {
                q.add(nbr)
                vis[nbr] = True
            }
        }
    }
}
```

O(n+m) T.C.

O(n) S.C
↳ vis array.
+
queue