Mock interviews ⇒ Simulation of real interviews!

Structure

60 mins → 2 Questions

Flow : → 3-5 mins to introduce

Problems → 20-25 mins.

Feedback → 5 mins

Questions

- → Take a moment to process the question (Example, Dry)
- → Basic Sol^n (Brute force) → Optimisation → Code!!
- → Quality of Code

DSA ⇒ (PSP ≥ 50) ⇒ ≥ 90%.

Clear Mock Interviews

DSA ⇒ PSP >, 55 ⇒ 70% Clear

↓ 30 day

Contest (13th) — (14th March) — Discussion — (14th April)

Graphs

Collection of Nodes & Edges.

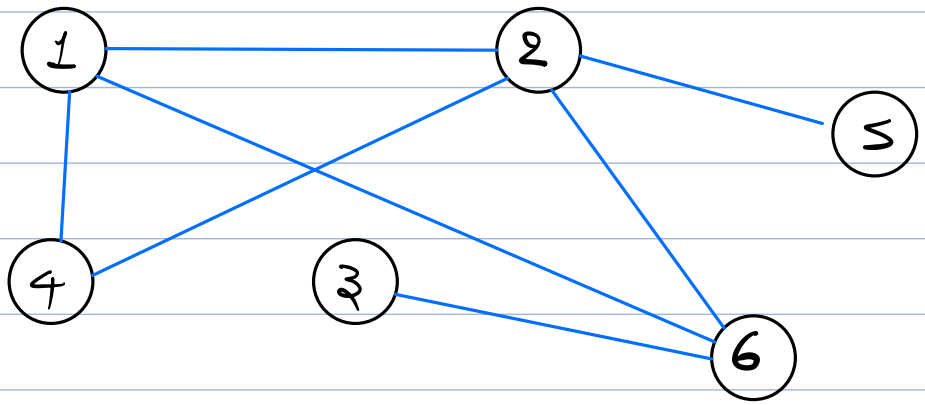
Real life Examples

1) Social Media
   Nodes → Users
   Edges → Connection b/w users.

2) A network of computers.

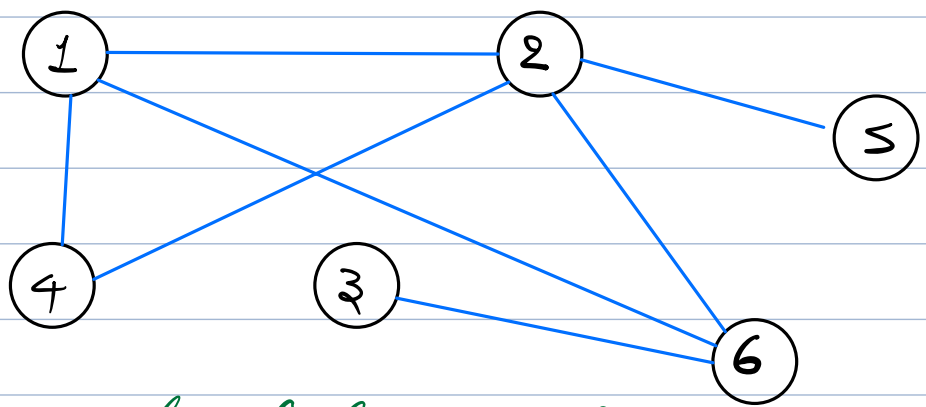
3) Google Maps
   Nodes → landMark
   Edges → Roads connecting them.

Nodes (N) → 1, 2, 3, 4, ... N.



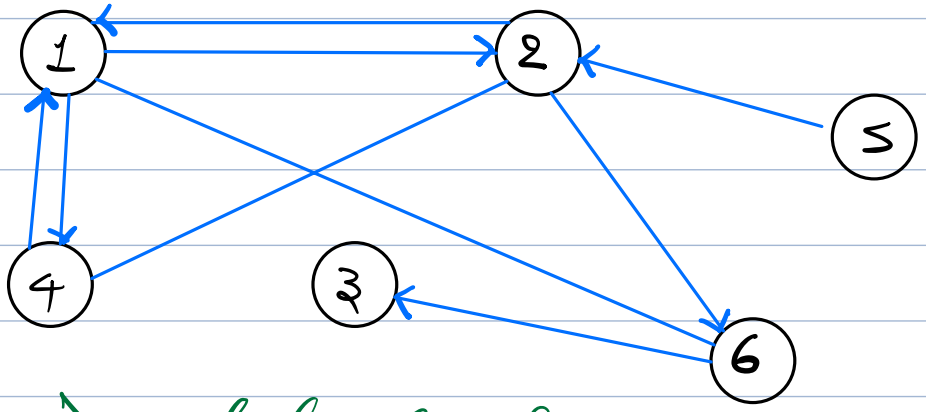
Types of Graphs

Facebook

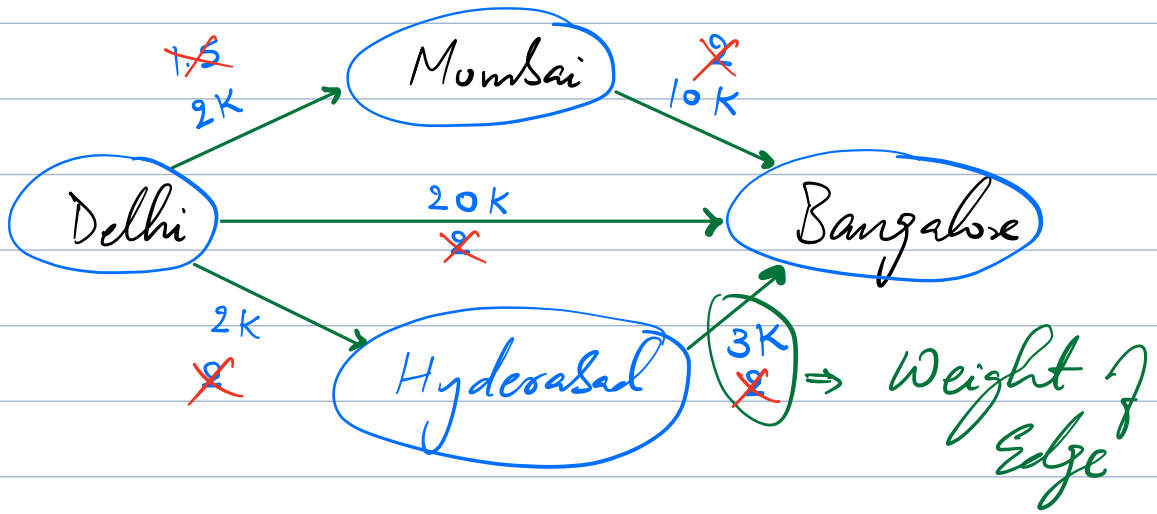


Undirected Graphs

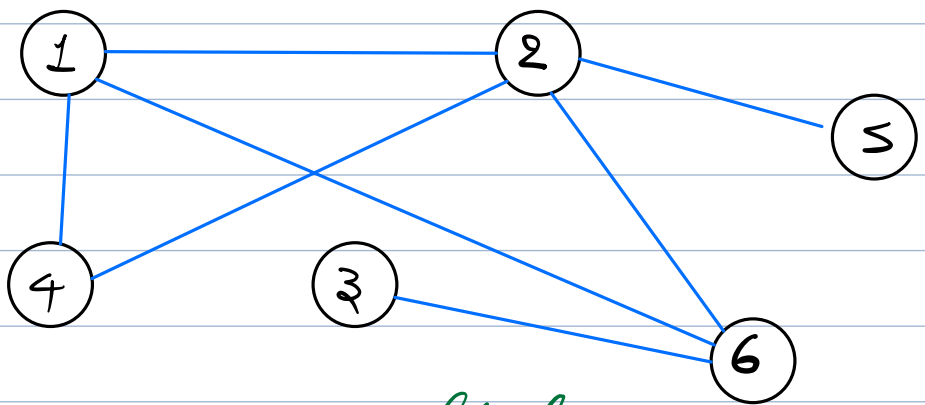
Instagram



Directed Graphs



Weighted graphs.



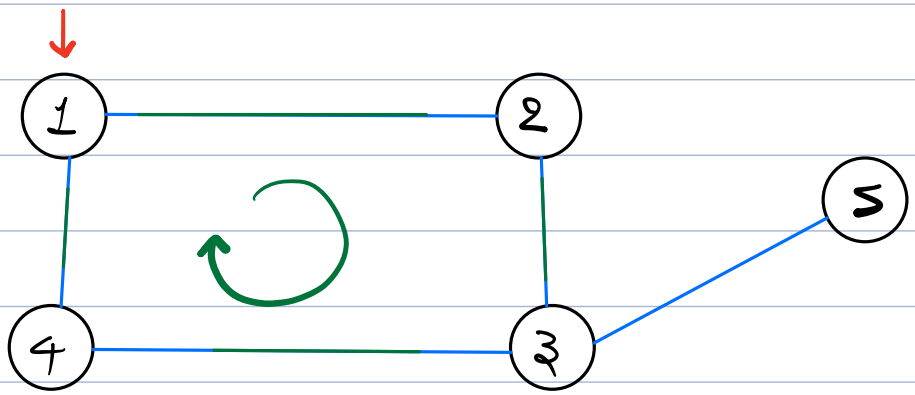
Unweighted graphs.

Undirected Unweighted graph

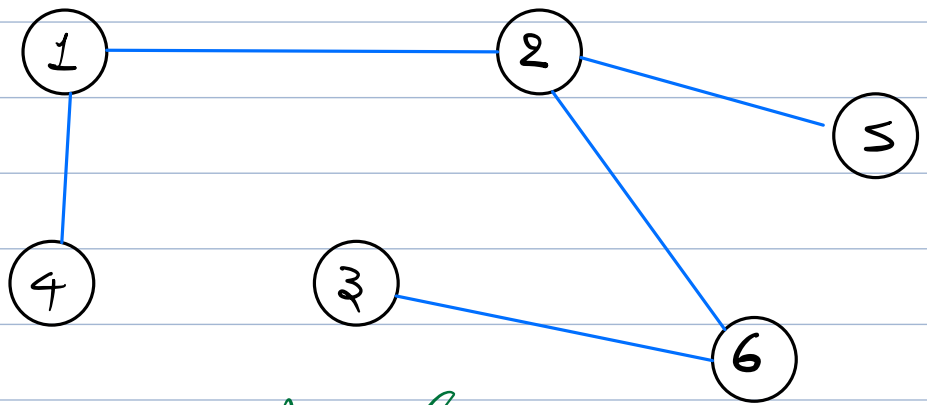
Undirected Weighted graph

directed Unweighted graph

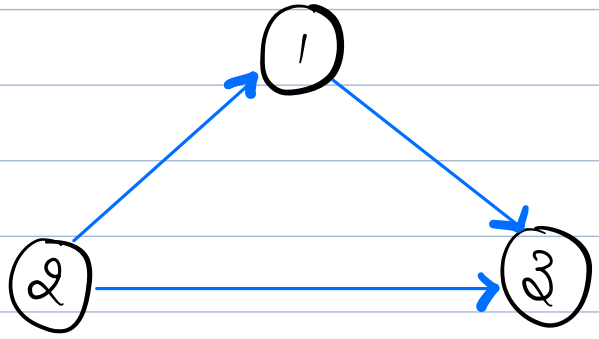
directed Weighted graph



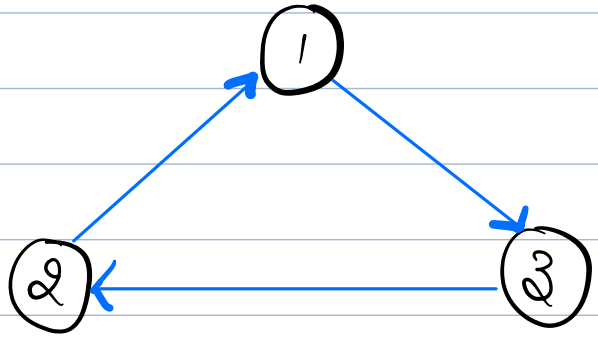
Cyclic Graph (If you can come back to any node of graph w/o repeating an edge)



Acyclic graph



Directed Acyclic graph

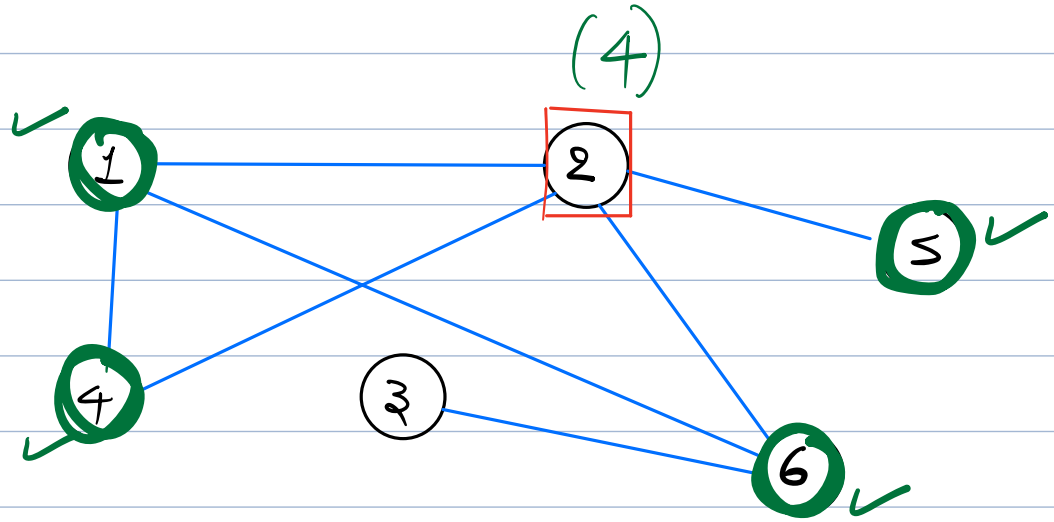


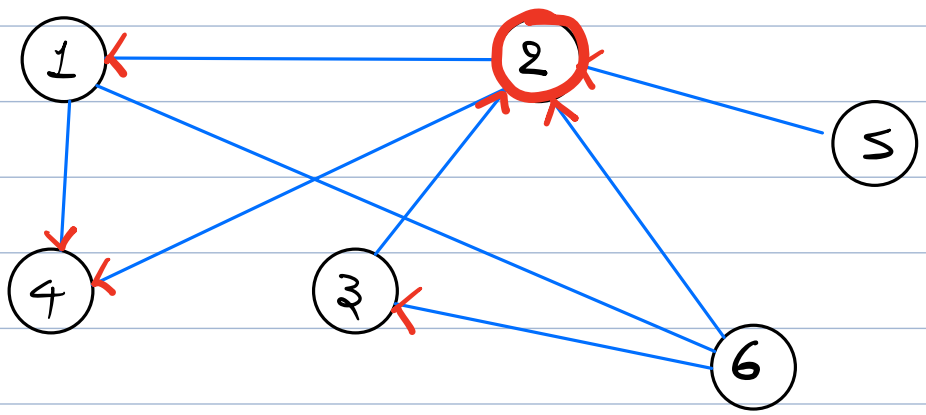
Directed Cyclic Graph

Repeated ✗

In Degree ≠ Out Degree

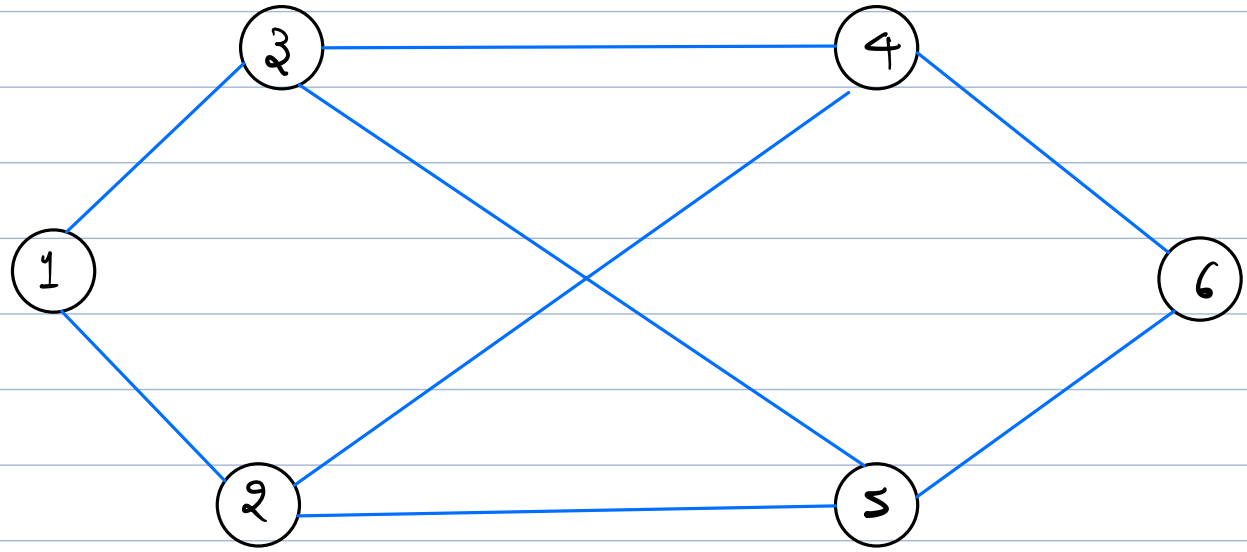
Degree ⇒ No. of adjacent nodes present





indegree = 3

outDegree = 2



Input

N = 6
E = 8

0 1
u — v — w

3, 4
3, 5
4, 6
5, 6
2, 1
1, 3
4, 2
5, 2

1) No. of Nodes = N

2) No. of edges = E

3) E lines of I/p

M[E][2] Unweighted

M[E][3] Weighted.

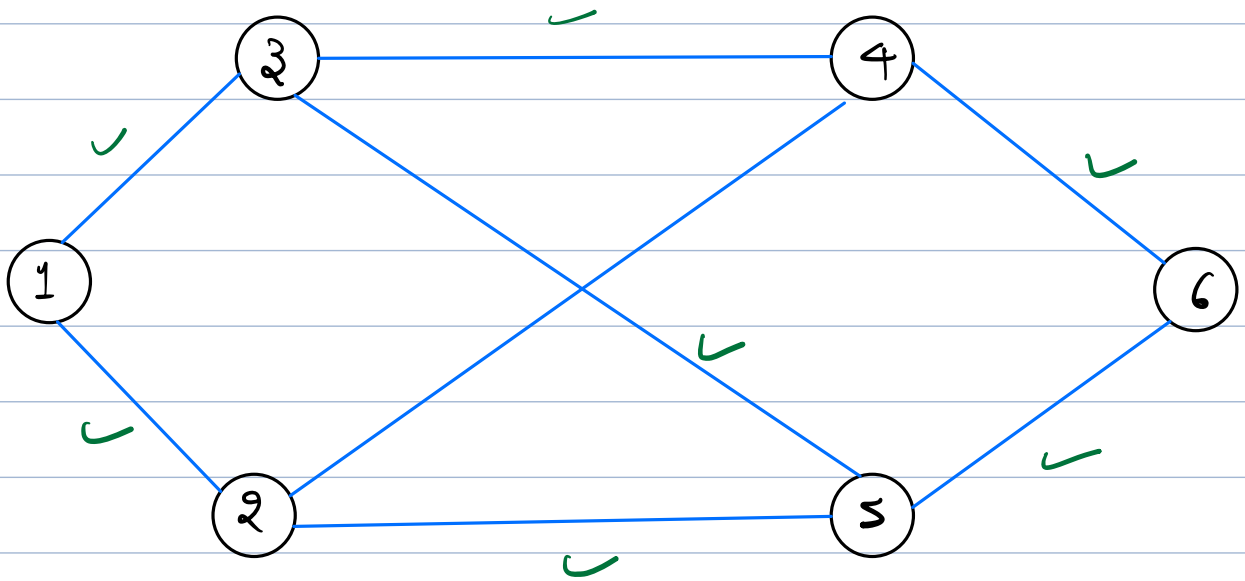
How to store a graph ??

1) Adjacency Matrix

Max no. of Edges
- ${}^{N}C_2 \rightarrow \frac{(N)(N-1)}{2}$ (Undirected)
- ${}^{N}C_2 \times 2 \rightarrow (N)(N-1)$ (Directed)

$O(N^2)$

Mat[N][N]
- if (Mat[i][j] == 1) { Edge from i to j }
- if (Mat[i][j] == 0) { No Edge from i to j }



	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	0	1	1	0	0	0
2	0	1	0	0	1	1	0
3	0	1	0	0	1	1	0
4	0	0	1	1	0	0	1
5	0	0	1	1	0	0	1
6	0	0	0	0	1	1	0

NOTE : for undirected ⇒ Mat[i][j] = Mat[j][i]

Code

E ⇒ No. of Edges.

I/P ⇒ A[E][2] ⇒ Edges

N ⇒ No. of Nodes

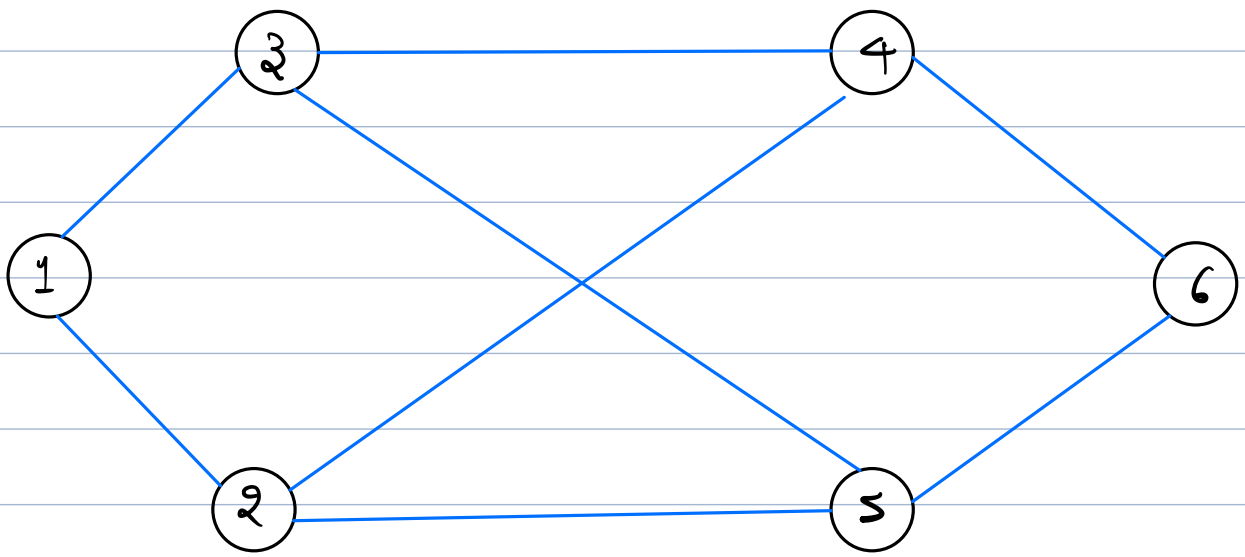
```
Mat[N+1][N+1] = {0};        // N^2

for (i=0; i < A.size(); i++) {  } // E

    s = A[i][0];
    d = A[i][1];

    Mat[s][d] = 1;
    Mat[d][s] = 1;
```

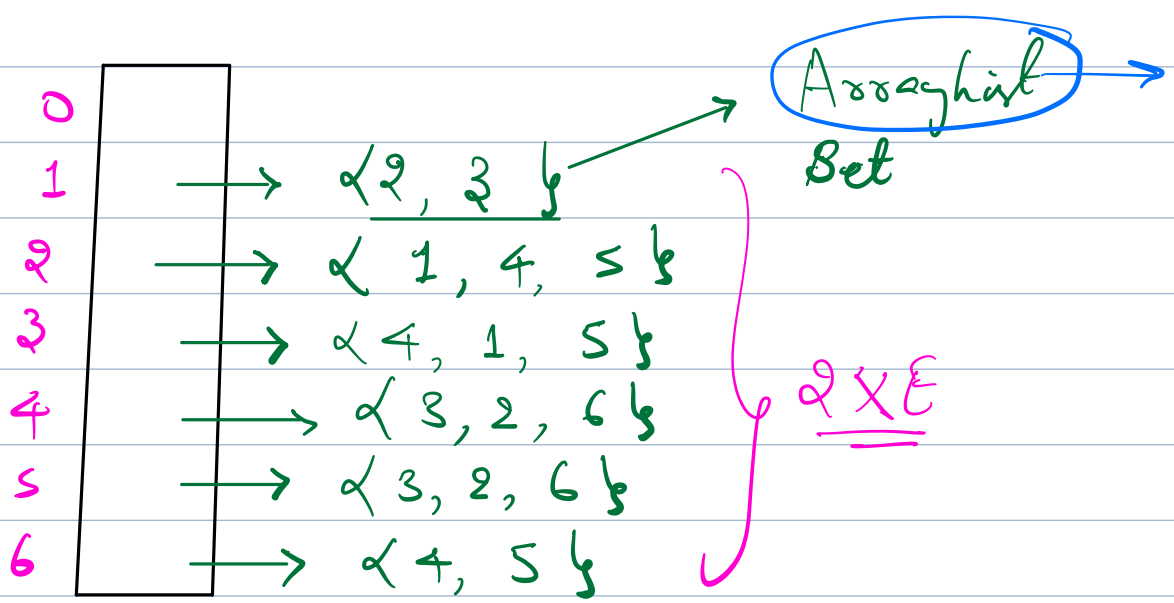
$T.C. = O(N^2)$

$S.C. = O(N^2)$

---

2) Adjacency List



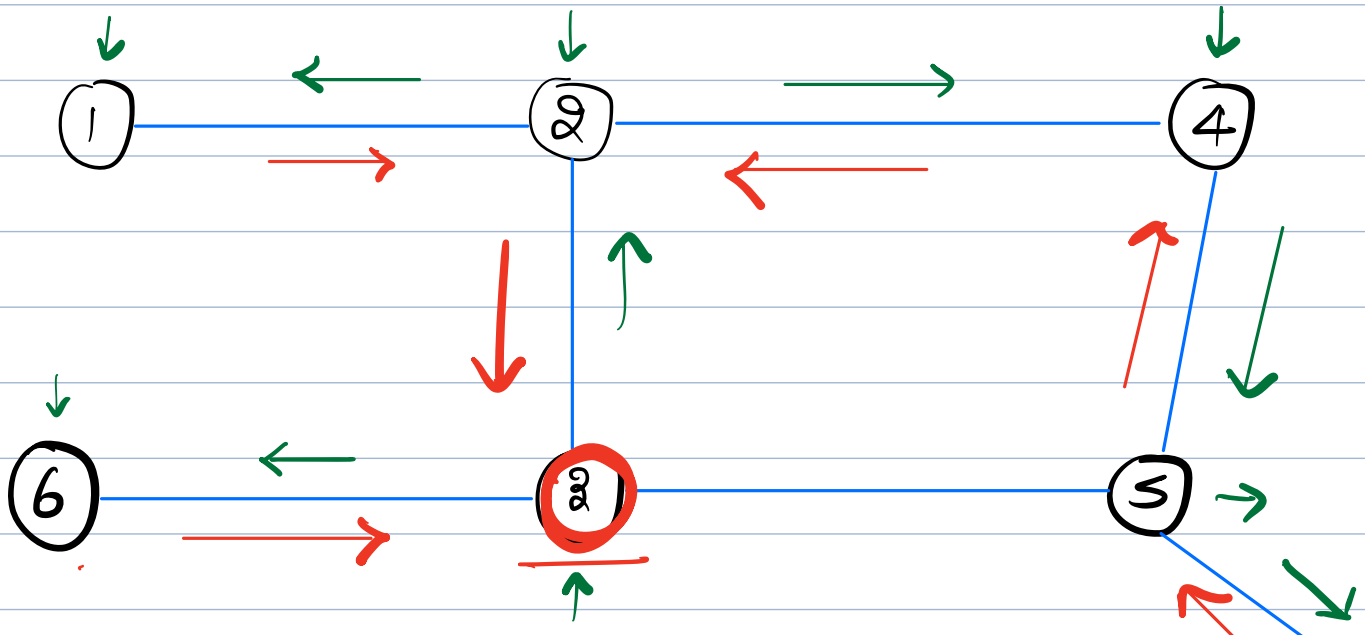


```
List<int> graph[N+1]

for (i=0; i<A.size(); i++) {          } // E

    s = A[i][0];
    d = A[i][1];

    graph[s].add(d);
    graph[d].add(s);                  } 2
}
```

T.C. = $O(E)$ — Total no. of edges.

S.C. = $O(V+E)$

Depth first Search (DFS)



0	1	2	3	4	5	6	7
F	T	T	T	T	T	T	T

2 → { ~~0~~, ~~1~~, ~~5~~ }

6 → { ~~3~~ }

1 → { ~~2~~ }

4 → { ~~2~~, ~~5~~ }

5 → { ~~3~~, ~~4~~, ~~7~~ }

7 → { ~~5~~ }

Steps

dfs(source)

1) Decide Source & start exploring

2) Mark Node as visited

3) Go to all unvisited neighbours & call dfs recursively.

4) Backtrack.

Code

```
bool visited[N+1] = {false};

void dfs(source) {
    visited[source] = true;
    for(all nodes u present in adj. list of
        source) {
        if(visited[u] == false) {
            dfs(u);
        }
    }
}
```

$T.C. = O(V+E)$

Given a graph. Check if it contains a cycle.

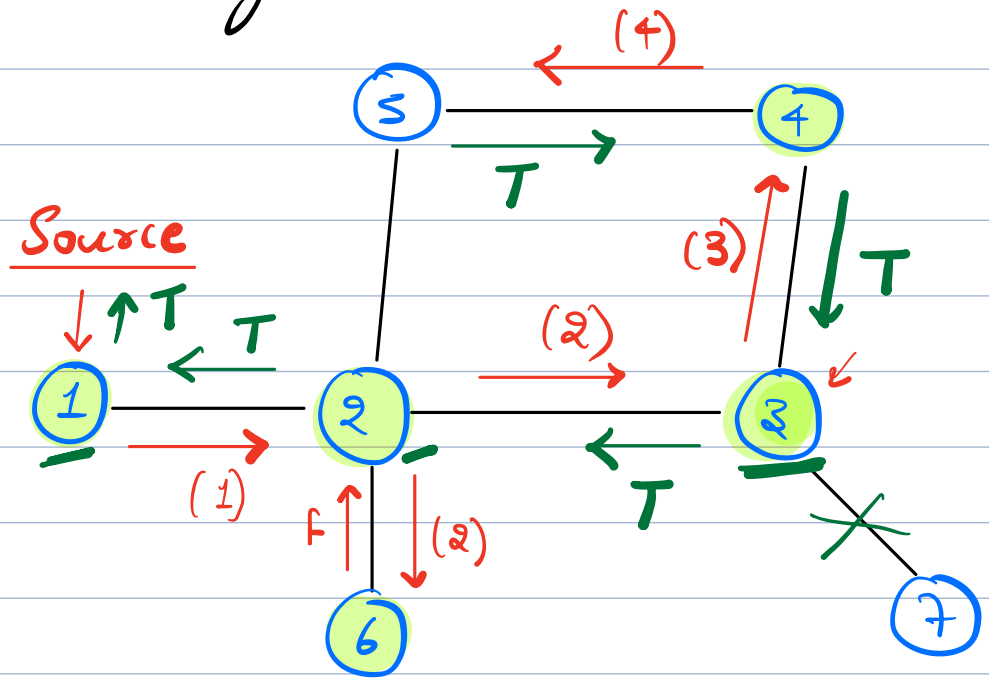
(1) 2 → {~~1~~, ~~6~~, 3, ~~5~~}

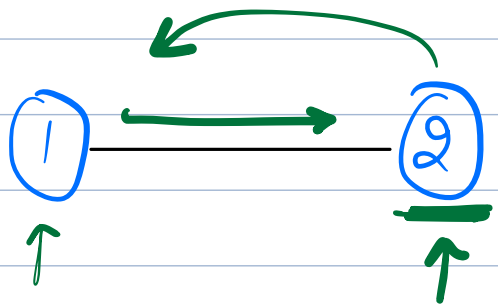
(2) 6 → {~~2~~}

(2) 3 → {~~2~~, 4, ~~7~~}

(3) 4 → {~~3~~, 5}

(4) 5 → {~~4~~, 2}





0	1	2
F	T	~~F~~ T

Code

} Connected Component

NULL

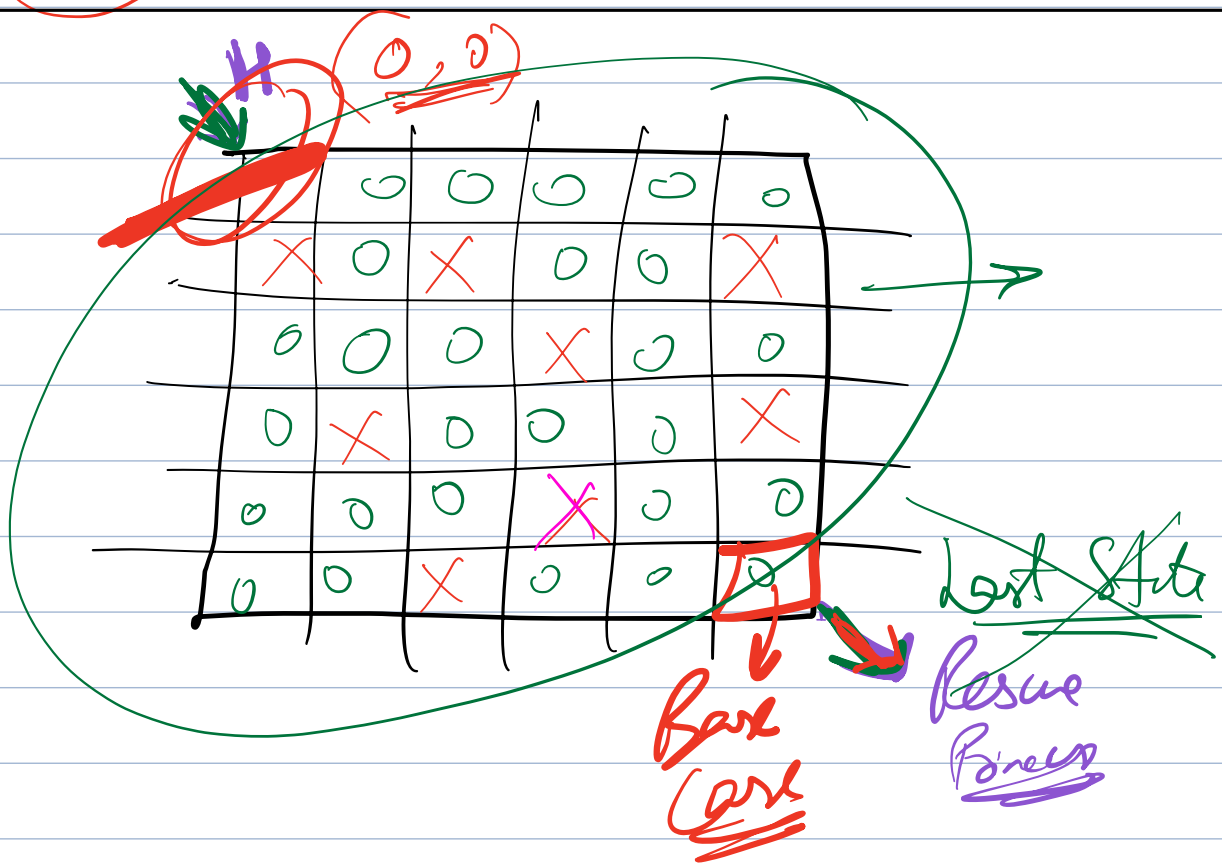
```
bool isCyclic(source, parent){
    visited[source] = True;
    for(all nodes u connected to source){
        if(visited[u] == True){
            if(u != parent){
                return True;
            }
        }else{
            bool cycleFound = dfs(u, source);
            if(cycleFound){
                return True;
            }
        }
    }
    return false;
}
```

$$T.C = O(V+E)$$

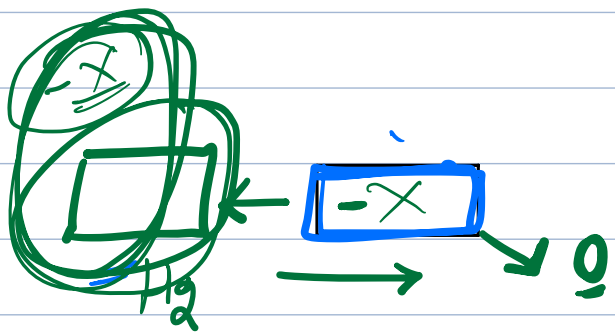
$$S.C = O(V)$$

Binary Search → Questions

Dungeon Princess

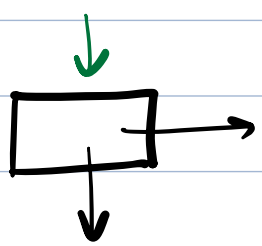


1

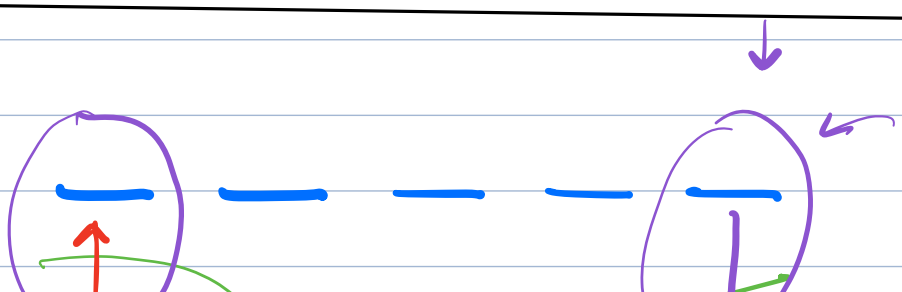


$H_2 - x = 0$

$H_2 = x$



N Digit



Sum = 7

N == 1

2 digit

q 1-9 ?

+0

61
52
43
34
25
16
07 X

q 0-9 ?

10

Sum of digits = X

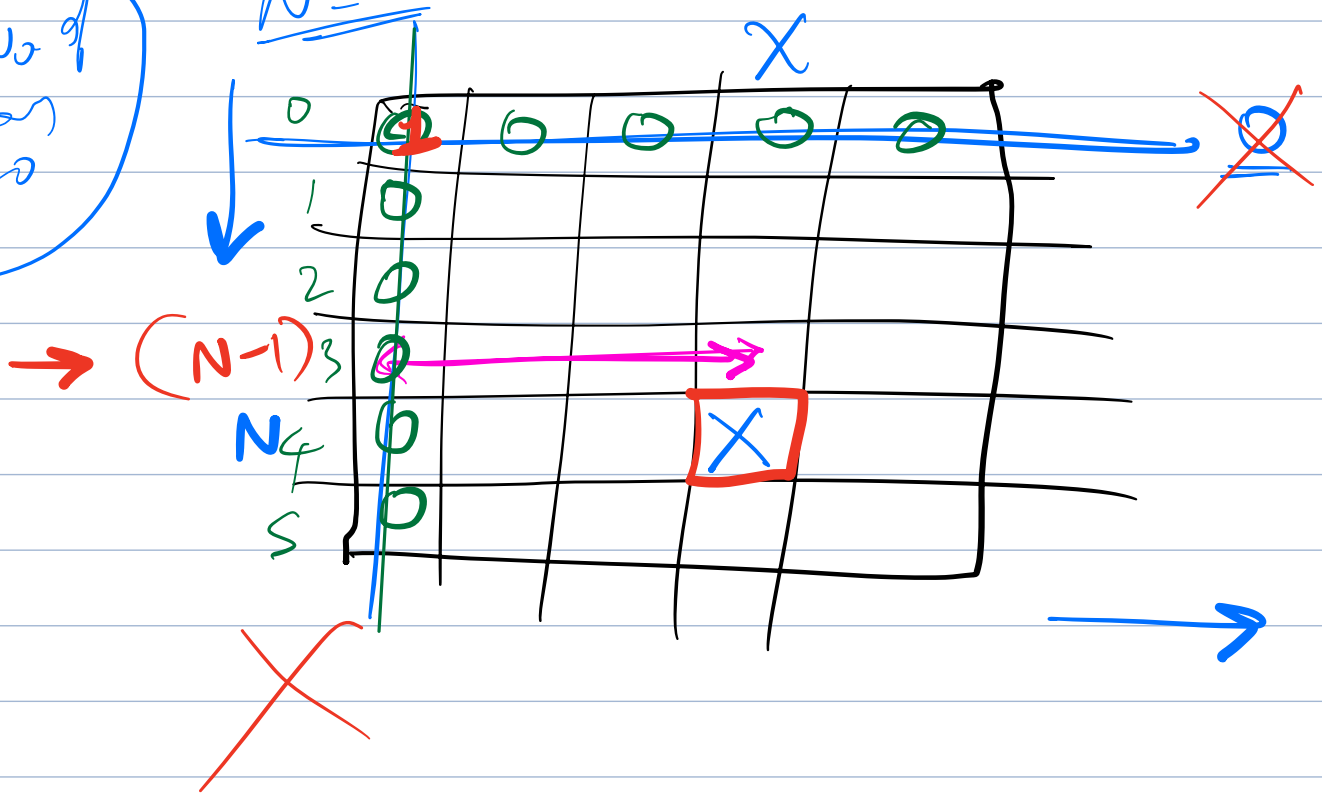
$(N, X) \rightarrow \sum_{i=0}^{9} (N-1, X-i)$

0

0,9

No ? wrong

N = 0



Flip

A = [-A[i] _ _ _ _]

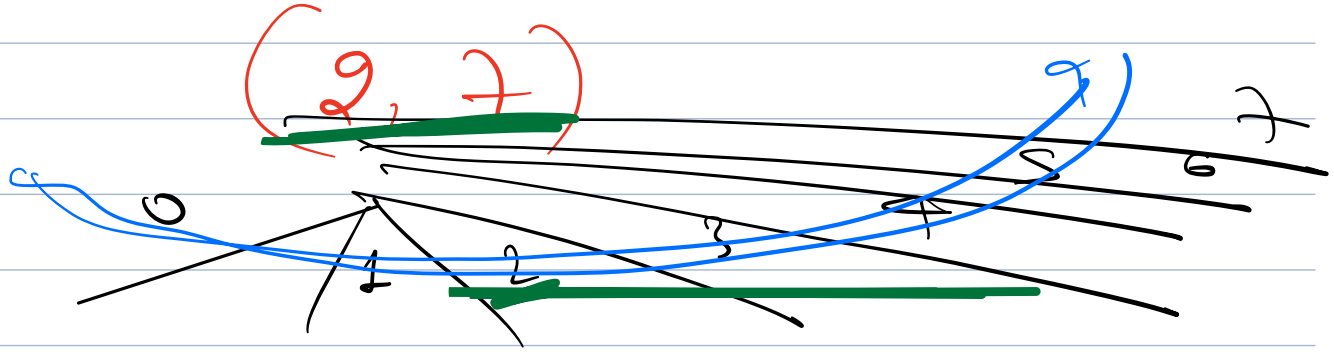
Min no. of flips

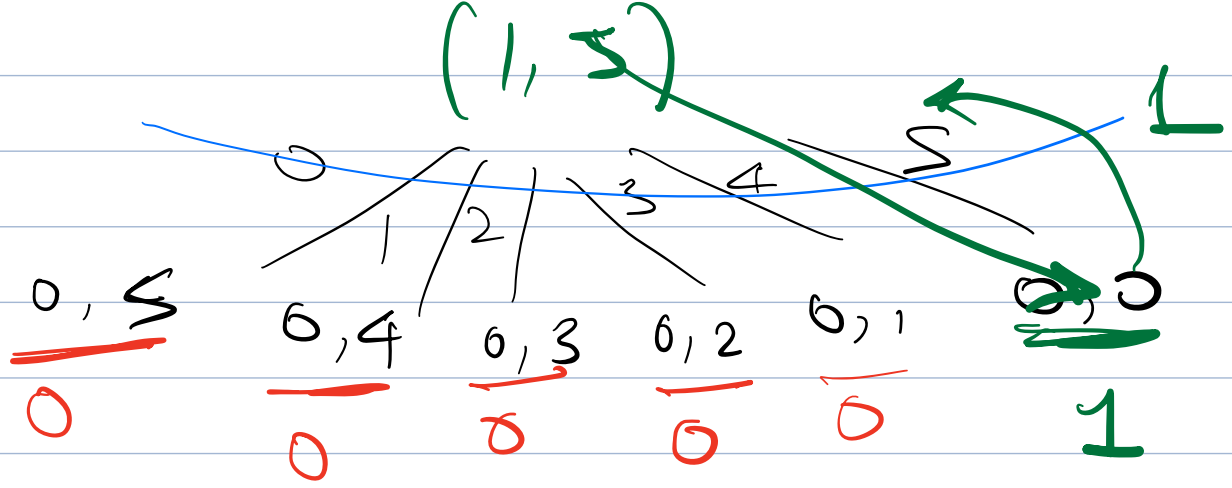
Sum after flip is close to zero as possible

Total Sum = $\sum$

Max Sum of elements that can be flipped = $\sum$ / 2 → Capacity

$\sum$ → Min no. of elements





$N = N$

$M = C = Sum/2$

$(N+1)(C+1)$