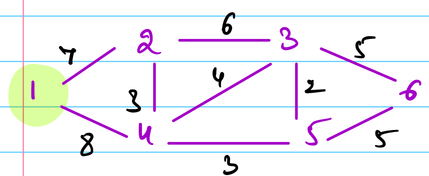
21/9/2023

Graphs-4

Dijkstra's algo → Single source shortest path algorithm for weighted graph with +ve weights

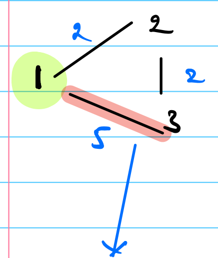
Q There are N cities in a country. You are living in city 1. Find min distance to reach every city from city 1.

$c_1 \overset{dist}{\rule{2cm}{0.4pt}} c_2$



Ans → $d[i], \forall i$

$d[1] = 0$ (source)

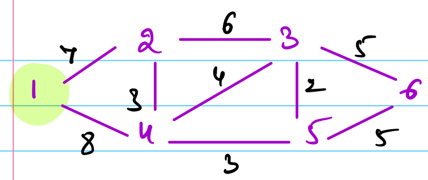


$d[1] = 0$
$d[2] = 2$
$d[3] = 4 \ (2+2)$

Relaxing an edge

intermediate node

```
if ( d[u_w] > d[u_v] + d(v_w] )
    d[u_w] = d[u_v] + d[v_w];
```

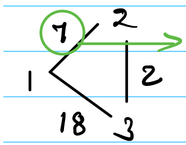


Ans = d[i], ∀i

d[1] = 0 (source)

d[i] = INT_MAX (∀ i>1)

min distance to reach city 2 from city 1 = 7

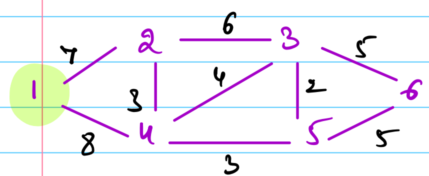


→ min wt edge

$u \xrightarrow{wt} v$

(~~wt~~, u, v)

d[u]+wt



~~(7,1,2)~~, ~~(8,1,4)~~

~~(10,2,4)~~, ~~(13,2,3)~~

~~(12,4,3)~~, ~~(11,4,5)~~

~~(13,5,3)~~, ~~(16,5,6)~~

~~(17,3,6)~~

min heap.

	1	2	3	4	5	6
d =	0	7	12	8	11	16
p =	-1	1	4	1	4	5

6 → 5 → 4 → 1

1) Insert all edges connected to source in the heap.

2) Get min dist path from heap, update distance of destination if it is INT_MAX

3) Insert all edges connected to the popped out element which are not directed to nodes for which we already have answer, i.e. d[x] == INT_MAX

TC: O(E log E) , SC: O(N+E)

1)
```
for (v: adj[u]) {
    pq.add( wt, u, v);
}
```

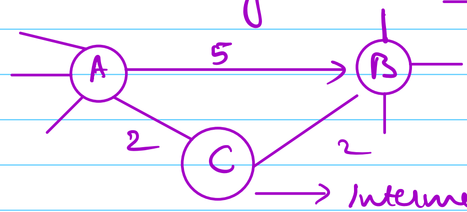
2)
```
{wt, u, v} = pq.top();
if (d[v] == INT_MAX) {
    d[v] = wt;
    pre[v] = u;

    for (neigh: adj[v] ) {
        pq.add (d[v] + wt[i], v, neigh);
    }
}
```

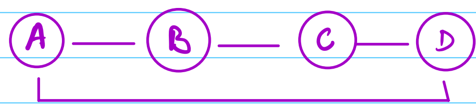
Q. Find shortest distance from every node to every other node

Ans ⇒ 2D array.

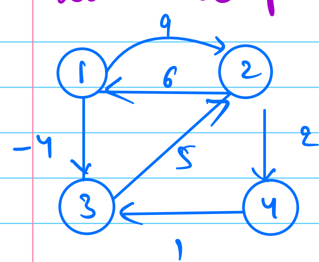
Floyd warshall algo



→ Intermediate node.



∀ nodes, consider it as intermediate node & try all alternate paths.



Adj' matrix

A =

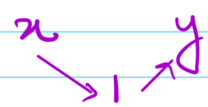
	1	2	3	4
1	0	9	-4	0
2	6	0	0	2
3	0	5	0	0
4	0	0	1	0

D =

	1	2	3	4
1	0	9	-4	∞
2	6	0	∞	2
3	∞	5	0	∞
4	∞	∞	1	0

D[i][j] = min distance from i to j

1 → intermediate node



D1 =

	1	2	3	4
1	0	9	-4	∞
2	6	0	2	2
3	∞	5	0	∞
4	∞	∞	1	0

$\underset{\infty}{d(2\to 3)} \to d(2\to 1\to 3)$
$\Rightarrow 6+(-4) = 2$

$\underset{2}{d(2\text{-}4)} \to d(2\to 1\to 4)$
$\to 6+\infty = \infty$

$\underset{5}{d(3\to 2)} = d(3\to 1\to 2)$
$\Rightarrow \infty + \Rightarrow \infty$

$x \longrightarrow 2 \nearrow y$

$$D2 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 9 & -4 & 11 \\ 2 & 6 & 0 & 2 & 2 \\ 3 & 11 & 5 & 0 & 7 \\ 4 & \infty & \infty & 1 & 0 \end{array}$$

2 → intermediate node.

$d(1 \to 3) \to d(1 \to 2 \to 3)$

$-4 \to 9 + 2 \to 11$

$d(1 \to 4) \to d(1 \to 2 \to 4)$

$\infty \to d(9 + 2)$

$d(3 \to 1)$

$\to 5 + 6 \to 11$

Similarly take 3 & 4 as intermediate nodes

```
for k = 1 to N {  → intermediate node.
    for i → 1 to N {
        for j → 1 to N {
            if (d[i][j] > d[i][k] + d[k][j]) {
                d[i][j] = d[i][k] + d[k][j]
            }
        }
    }
}
return d;
```

TC: $O(N^3)$

SC: $O(N^2/1)$

Q Given a matrix of integers containing,

0 → empty

1 → fresh orange

2 → rotten orange

(r-1, c)

(r, c-1) ← (r, c) → (r, c+1)

(r+1, c)

Every minute, any fresh orange adjacent to a rotten orange becomes rotten.

In how many minutes, will all oranges become rotten?

If not possible, return -1.

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

BFS traversal

Multi Source BFS

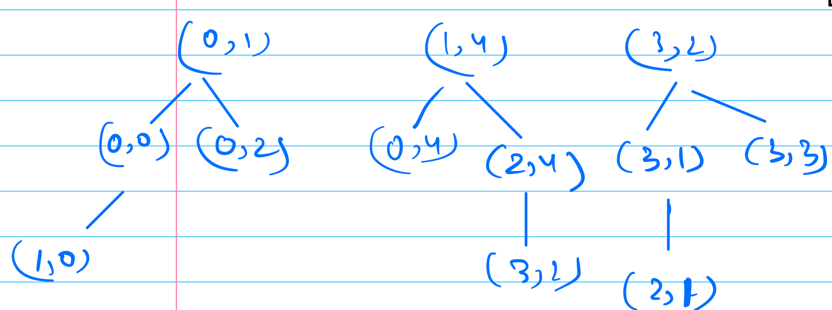
N*M

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

Time

~~(0,1)~~, ~~(1,4)~~, ~~(3,2)~~, ~~(0,0)~~, ~~(0,2)~~, ~~(0,4)~~, ~~(2,4)~~

~~(3,1)~~, ~~(3,3)~~, ~~(1,0)~~, ~~(3,2)~~, ~~(2,1)~~



if T != ∞ ⇒ rotten orange

if ($\forall i, j$, s.t $A[i][j] = 1$, we have $T[i][j] != \infty$)

Ans = max time of any orange ($!= \infty$)

else

Ans = -1

TC: $O(N*M)$
SC: $O(N*M)$

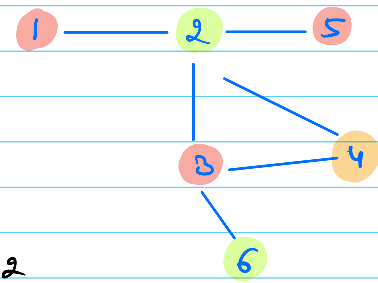
Meet at 8 : 48 am IST

Graph Coloring

Color all the nodes of graph such that no two adjacent nodes are of same color & count of distinct colors is minimum.

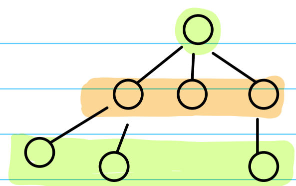
Chromatic Number

(N-P hard)



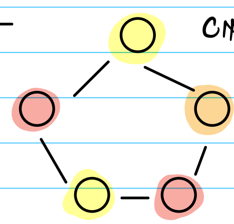
Special cases

1) Tree

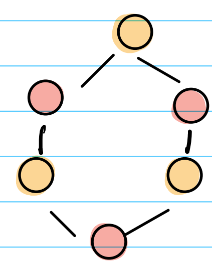


CN = 2

2) Cycle graph



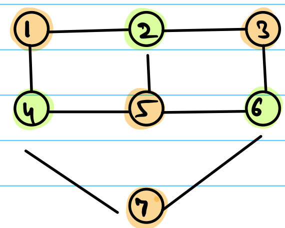
CN = 3
(odd)



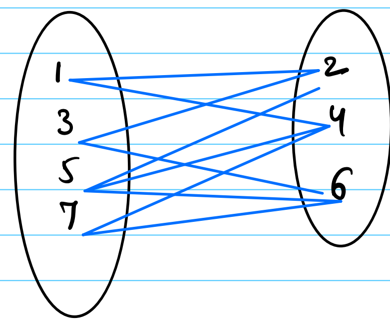
CN = 2
(even)

$CN = 2 + (N \% 2)$

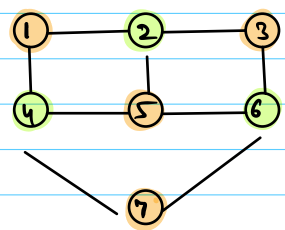
Bipartite graph (CN = 2)



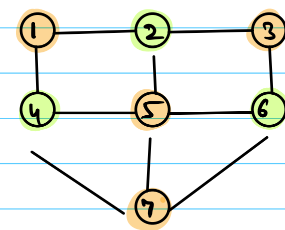
Graph can be divided into 2 sets, s.t. there is no edge b/w nodes of same set.



Q Check if the given graph is bipartite



True



false.

Soln :- Travel & color all nodes s.t. no 2 adj' nodes have same color. If not possible, return false.

$\forall_i$: col[i] = -1
col[s] = 0 // 0 or 1 → color.

```
boolean dfs (s) {
    for (v : adj[s]) {   // s → v
        if ( col[v] == -1) {
            col[v] = 1 - col[s];
            if (!dfs(v)) return false;
        }
        else if (col[s] == col[v]) {
            return false;
    }
    return true;
}
```

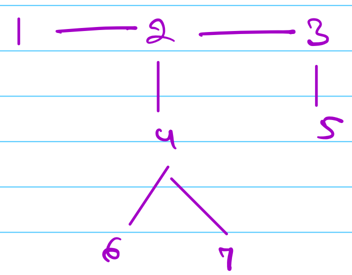
TC: O(N+E)
SC: O(N)
recursion + col

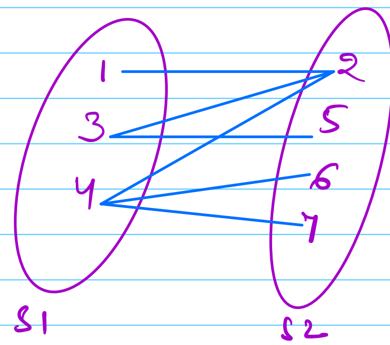
Q A country consists of $\overbrace{N \text{ cities connected by } (N-1) \text{ roads}}^{\text{tree}}$.
King of that city wants to construct max. roads s.t
the cities can be divided into 2 sets & there is no road b/w cities in same set $\rightarrow$ Bipartite graph

Find max # new roads King can construct.

Eg

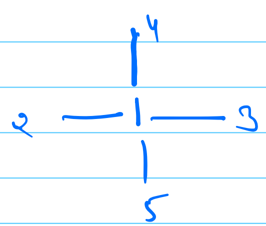
$N = 7$





cnt1 ⇒ # nodes in S1 , total roads = cnt1 * cnt2
cnt2 ⇒ # nodes in S2

Ans ⇒ total − (N−1)



Is Ans = 0, possible ?

Yes.

```
for i → 1 to N {
    if (col[i] == 1) cnt1++;
    else   cnt2++;
}
```

TC: O(N+E)
SC: O (N)

Doubts

1. Bellman ford

2.