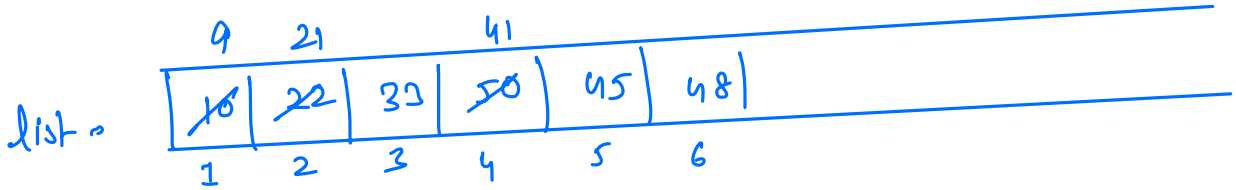
- ✓ LIS in NlogN
- ✓ Interleaving Strings
- ✓ Regular Expression - II
- ✓ Poisnous graph.
- ✓ matrix and Absolute Difference.
- → Auto-complete (Hint)

Contest. → Revise assignments.

5 Questions. → 2 G
→ 2 M
→ 1 Hard.

① Longest Increasing Subsequence

Eg → [10 (0) 22 (1) 9 (2) 33 (3) 21 (4) 50 (5) 41 (6) 45 (7) 48 (8)]



code →

```
list<int> lis;
lis.add(arr[0]);
for(i=1; i<N; i++){
    if(arr[i] > list.get(list.size()-1)){
        list.add(arr[i]);
    }
    else{
        // find idx of the element which is (Binary search)
        // just greater than arr[i];
        list[idx] = arr[i];
    }
}
return list.size();
```

T.C → O(N log N)
S.C → O(N)

② Interleaving Strings →

Given A, B, C find whether C is formed by the interleaving of A and B.

A = "aabcc"
B = "dbbca"
C = "aadbbcbcac"

ans → true.

a a b c c

d b b c a

a a d b b c c a c a

A → a a b c c
0 1 2 3 4 ← i

B → d b b c a
0 1 2 3 4 ← j

C → a a d b b c b c a c
0 1 2 3 4 5 6 7 8 9
↑
i+j

s1, s2, s3, i, j

s1[i] == s3[i+j] → s1, s2, s3, i+1, j

s2[j] == s3[i+j] → s1, s2, s3, i, j+1

code →

dp[n][m], ∀ i,j dp[i][j] = -1

```
int interleaving( s1, s2, s3, i, j, int[][] dp ){
    if( i == s1.length() && j == s2.length() ){ return 1 }
    if( dp[i][j] != -1 ){ return dp[i][j]; }
    f1 = 0 , f2 = 0 ;
    if( s1[i] == s3[i+j] ){
        f1 = interleaving( s1, s2, s3, i+1, j, dp ) ;
    }
    if( s2[j] == s3[i+j] ){
        f2 = interleaving( s1, s2, s3, i, j+1, dp);
    }
    dp[i][j] = Max( f1, f2);
    return Max(f1, f2);
}
```

-1 ↓ N.C. 0 ↓ f 1 ↓ t

T.C → O(n×m)
S.C → O(n×m)

[Bottom-up → #todo]

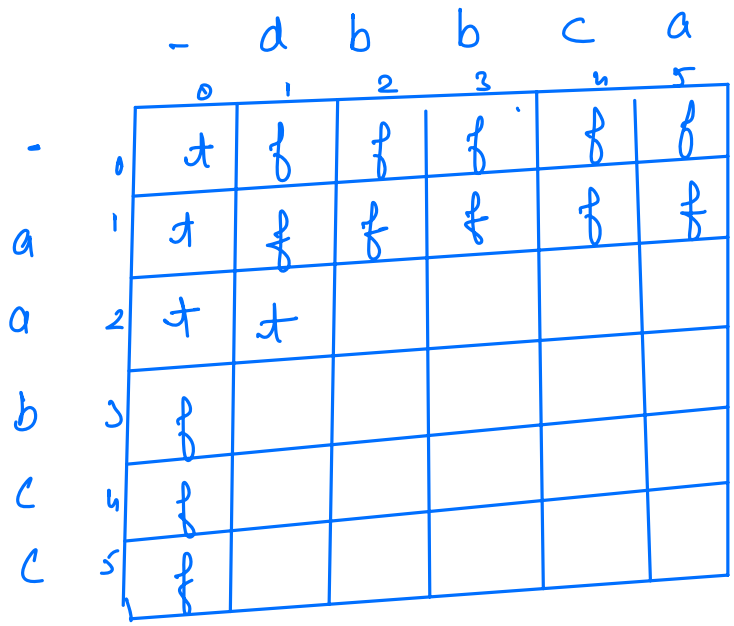
Bottom-up idea

A = "aabcc"

B = "dbbca"

C = "aadbbcbcac"

ans → true.



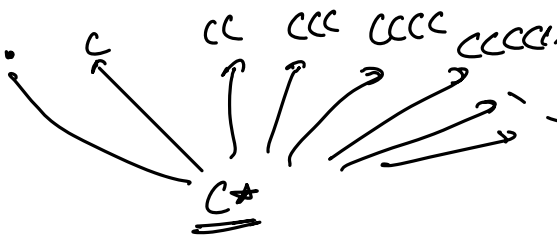
[T.C → O(N×m)
S.C → O(N×m)] ✓

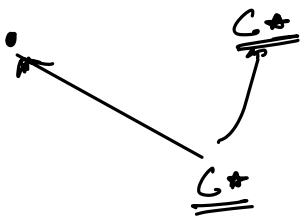
③ Regular Expression - II

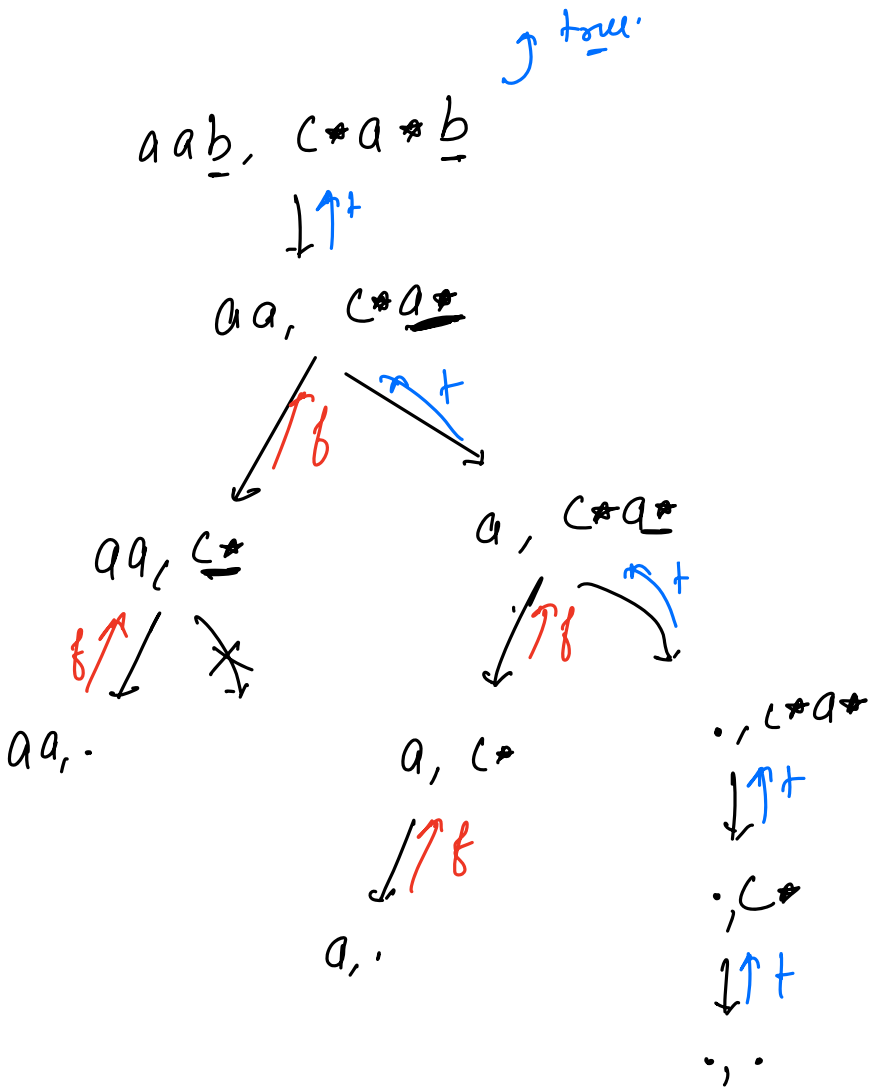
Implement wildcard pattern matching with support for ' ? ' and ' * ' for strings A and B.

' . ' : Matches any single character.
' * ' : Matches zero or more of the preceding element.
The matching should cover the entire input string (not partial).

A = "aab"
B = "c*a*b"







s1 → a b c b b x y t

s2 → a b * c . * t

		-	a	b	*	c	.	*	t
		0	1	2	3	4	5	6	7
-	0	t	f	f	f	f	f	f	f
a	1	f	t	f	t	f	f	f	f
b	2	f	f	t	t	f	f	f	f
c	3	f	f	f	f	t	f	t	f
b	4	f	f	f	f	f	t	t	f
b	5	f	f	f	f	f	f	t	f
x	6	f	f	f	f	f	f	t	f
y	7	f	f	f	f	f	f	t	f
t	8	f	f	f	f	f	f	t	t

i

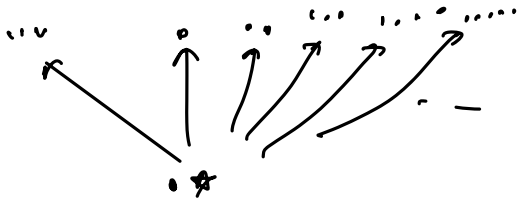
j

* → dp[i][j-2]

s1 → a b c d d m

s2 → a b . * m

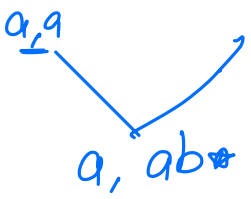
true



s1 → a b c c d e m n t y

s2 → . *

true



code. → dp[N+1][M+1]; ∀ i,j dp[i][j] = false;

```
for( i=0; i<=N; i++){
    for( j=0; j<=M; j++){
        if(i==0 && j==0) { dp[i][j] = true; }
        else if( j==0) { dp[i][j] = false; }
        else if (i==0){
            if( s2[j-1] == '*'){
                dp[i][j] = dp[i][j-2];
            }else{
                dp[i][j] = false;
            }
        }else{
            if( s1[i-1] == s2[j-1] || s2[j-1] == '.'){
                dp[i][j] = dp[i-1][j-1];
            }else if( s2[j-1] == '*'){
                f1 = dp[i][j-2];
                f2 = false;
                if( s2[j-2] == '.' || s2[j-2] == s1[i-1]){
                    f2 = dp[i-1][j];
                }
                dp[i][j] = (f1 || f2);
            }
        }
    }
}
return dp[N][M];
```

T.C → O(N×M)
S.C → O(N×M)

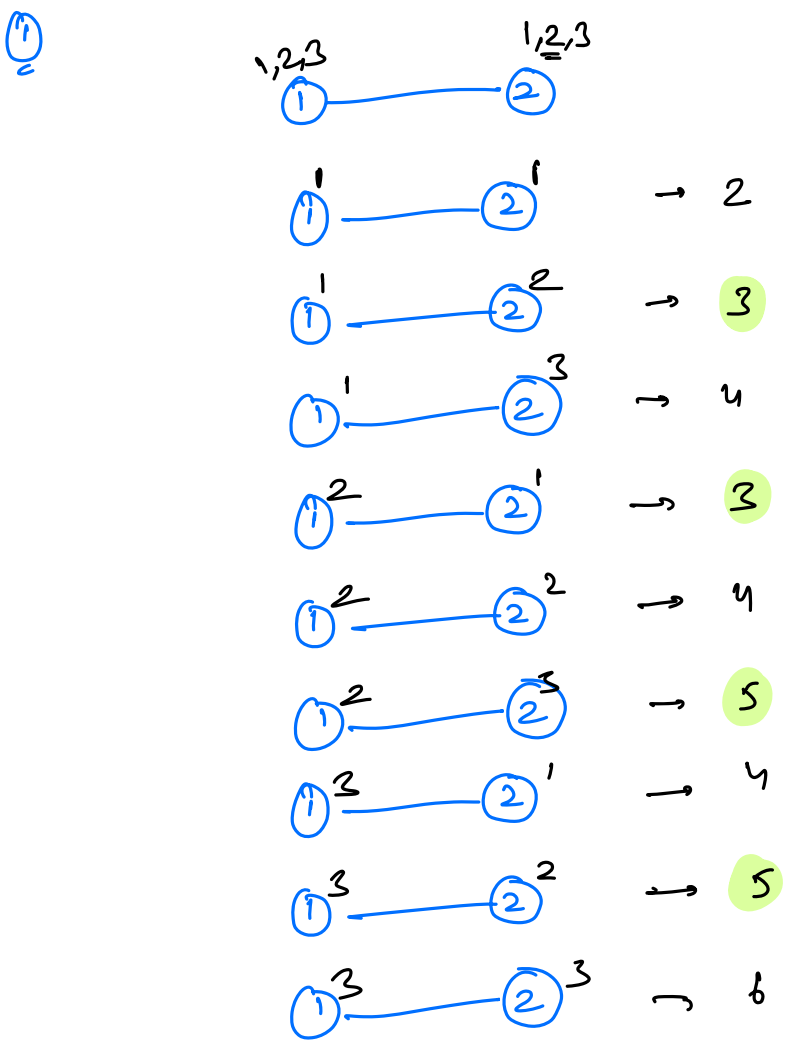
Break → 10:41 → 10:48

(4) Poisnous Graph

You are given an undirected unweighted graph consisting of A vertices and M edges given in a form of 2D Matrix B of size M x 2 where (B[i][0], B][i][1]) denotes two nodes connected by an edge.

You have to write a number on each vertex of the graph. Each number should be 1, 2 or 3. The graph becomes Poisonous if for each edge the sum of numbers on vertices connected by this edge is odd.

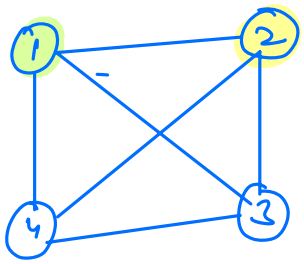
Calculate the number of possible ways to write numbers 1, 2 or 3 on vertices so the graph becomes poisonous. Since this number may be large, return it modulo 998244353.



Ans = 4

(2) 4

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

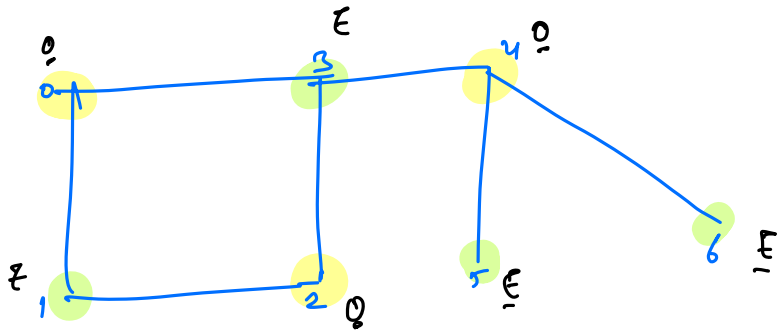


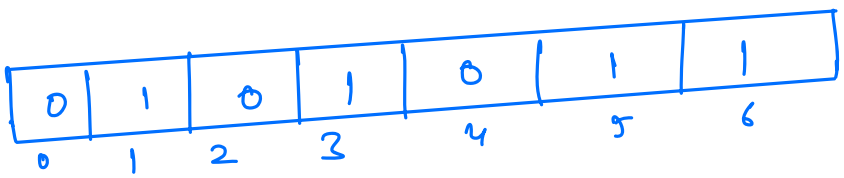
ans = 0

observation

→ if graph is non bi-partite, then ans → 0

→ if graph is bi-partite.





yellow → 3 , green → 4

[1,2,3]

yellow colored nodes → E and green colored nodes → O

or

yellow colored nodes → O and green colored nodes → E

$\Rightarrow 1^3 \times 2^4 + 2^3 \times 1^4 = \underline{2^4 + 2^3}$

[∴ ans = $2^{\text{yellow-colored nods}} + 2^{\text{green colored nods}}$]

[T.C → O(N+E)
S.C → O(N)]

(4) Matrix and Absolute Difference -

Given a matrix C of integers, of dimension A x B.

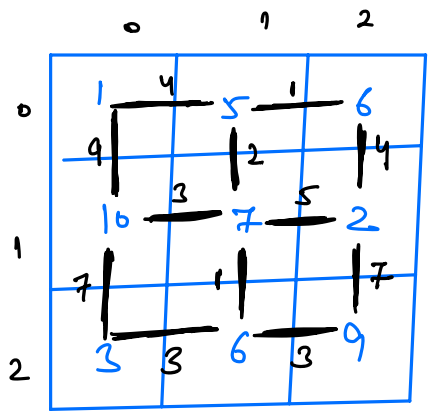
For any given K, you are not allowed to travel between cells that have an absolute difference greater than K.

Return the minimum value of K such that it is possible to travel between any pair of cells in the grid through a path of adjacent cells.

NOTE:

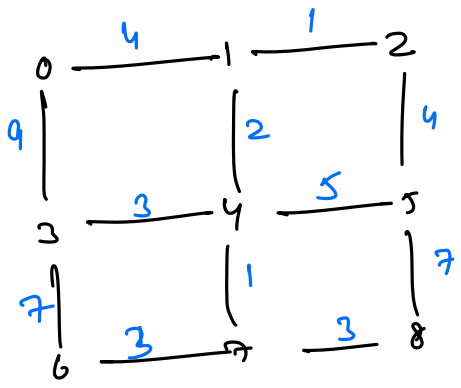
Adjacent cells are those cells that share a side with the current cell.

Ex:1



⇒ Minimum Spanning Tree.

Prim's Algo Kruskal's Algo



1, 2, 1 ✓

1, 4, 2 ✓

4, 7, 2 ✓

6, 7, 3 ✓

3, 4, 3 ✓

7, 8, 3 ✓

2, 5, 4 ✓

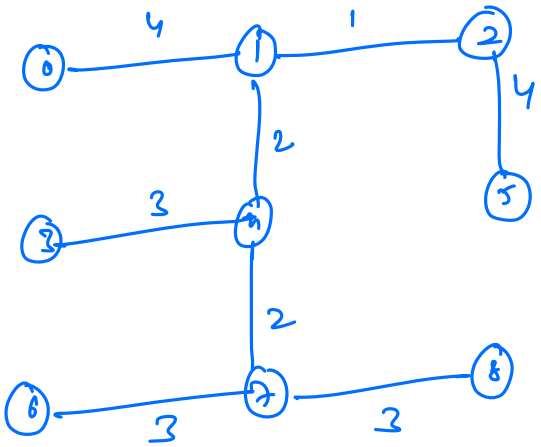
0, 1, 4 ✓

4, 5, 5 ✗

3, 6, 7 ✗

5, 8, 7 ✗

0, 3, 9 ✗



ans = 4.