Agenda

1) Max subsequence without adjacent elements.
2) count of paths in grid
3) count of paths in grid with blocked cells.
4) Minimum sum path

Q.1 Given an A[], find **maximum subsequence sum** such that no two consecutive elements are picked.

A = 9 14 3 ans = 14

A = 13 14 2 ans = 15

A = 4 3 2 1 ans = 6

A = -4 -3 -2 -1 ans = 0

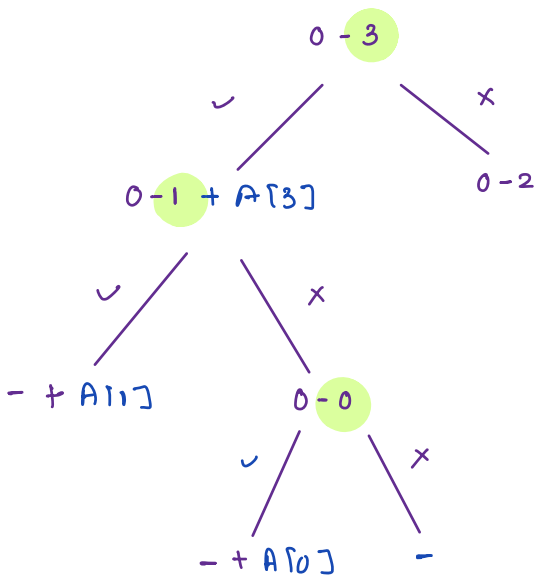
A = 9 4 13 24 ans = 33

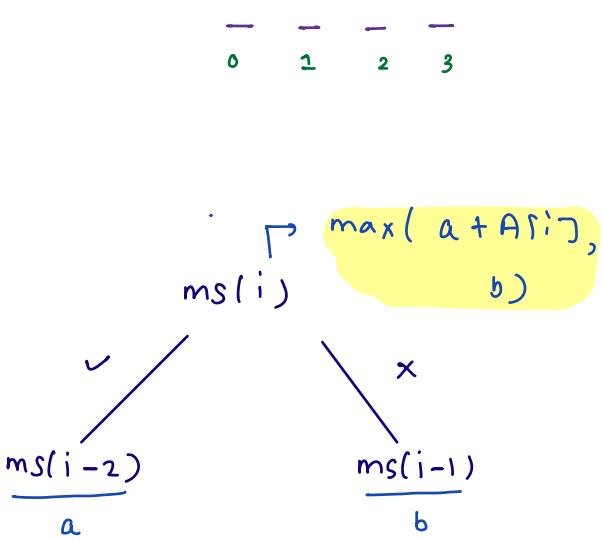
Alternative sum idea even_sum & odd_sum won't work.

idea 1 : Going on on all subseq, in base case check if subseq. is containing non-consecutive ele if yes than sum of this subseq. can be considered for updation of max ans till now.

TC: $O(2^n \times n)$

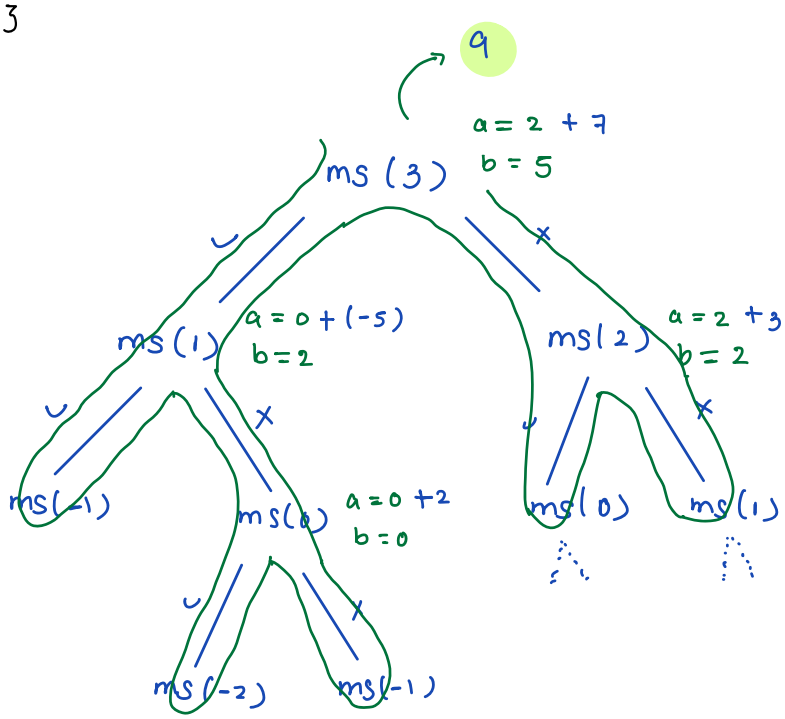
idea 2





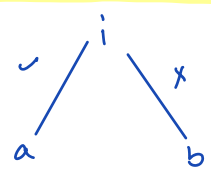
```
int solve (int [] A) {
    int n= A.length;
    return maxSubseqSum( A, n-1 );
}

int maxSubseqSum ( int [] A, int i ) {
    if( i<0) {
        return 0;
    }
    int a = maxSubseqSum ( A, i-2 );    // pick ith element
    int b = maxSubseqSum ( A, i-1 );    // don't pick ith element
    int ans = Math.max (a+A[i], b);
    return ans;
}
```

```
int maxSubseqSum ( int [] A, int i ) {
    if ( i<0 ) {
        return 0;
    }
    int a = maxSubseqSum ( A, i-2 );
    int b = maxSubseqSum ( A, i-1 );
    int ans = Math.max ( a+A[i], b );
    return ans;
}
```



A = 2 -5 3 7
0 1 2 3

int ans = Math.max (a+A[i], b)



as repeated sub-problems are there, DP can be applied.

```
int [] dp = new int [n];
```

→ fill it with -1

```
int maxSubseqSum ( int [] A, int i ) {
    if ( i<0 ) {
        return 0;
    }
    if ( dp [i] != -1 ) {
        return dp [i];
    }
    int a = maxSubseqSum ( A, i-2 );
    int b = maxSubseqSum ( A, i-1 );
    int ans = Math.max ( a+A[i], b );
    dp [i] = ans;
    return ans;
}
```

TC : O(n)

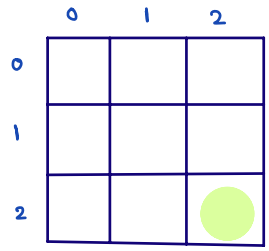
SC : O(n)

Q.2 Count total no. of ways to go from (0, 0) to (n-1, m-1)

TL BR

movement allowed → R or ↓ D

A =



r r d d

r d r d

r d d r

d r r d

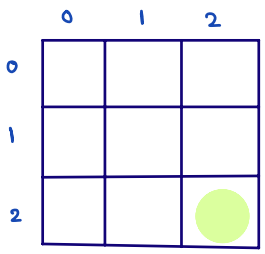
d r d r

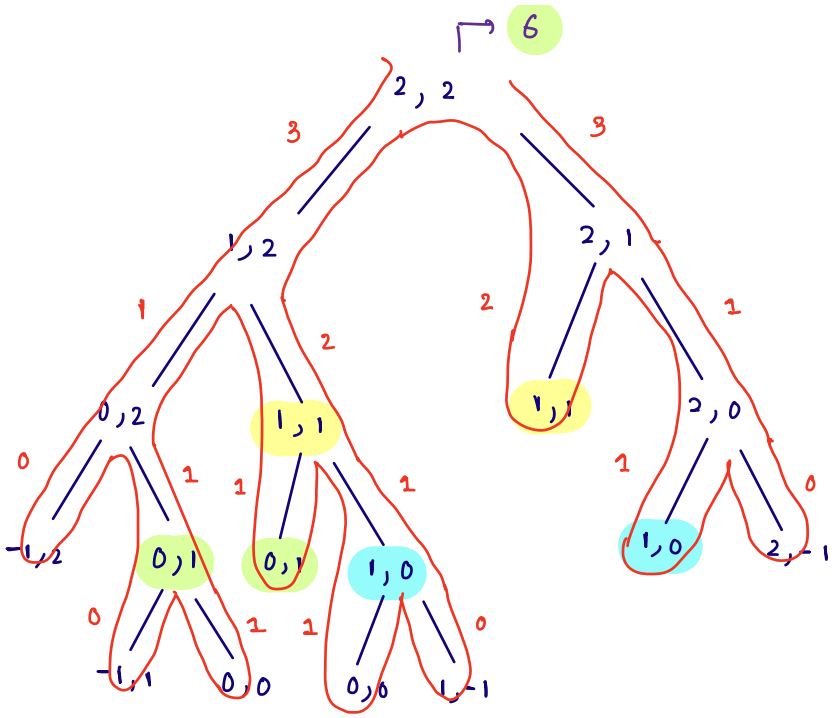
d d r r

ways (i, j) = ways (i-1, j) + ways(i, j-1)

no. of ways from 0,0 to (i, j) = no. of ways from 0,0 to (i-1, j) + no. of ways from 0,0 (i, j-1)

A =





ways (i, j) = ways (i-1, j) + ways(i, j-1)

```
int solve ( int n, int m) {
    dp = new int [n][m];
    // travel dp and fill it with -1
    return ways (n-1, m-1);
```

3

`int [][] dp;` (dimension : `n x m` and fill it with -1)

```
int ways (int i, int j) {
    if (i<0 || j<0) { return 0; }
    if (i==0 && j==0) { return 1; }
    if (dp[i][j] != -1) {
        return dp[i][j];
    }
    int a = ways(i-1, j);
    int b = ways(i, j-1);
    dp[i][j] = a+b;
    return a+b;
}
```

TC : O(N*m)

SC : O(N*m)

Q.3 Count total no. of ways to go from (0, 0) to (n-1, m-1)

Note: if mat[i][j] == 1, it is a blocked cell, else it is unblocked.

→ R ↓ D

A =

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

logic: last ques logic with base case condition c slight change.

```
int solve ( int [ ] [ ] A , int n, int m ) {
    dp = new int [n] [m];
    // travel dp and fill it with -1
    return ways ( A, n-1, m-1);
}
```

3

```
int ways (int [][]A, int i, int j) {
    if (i< 0 || j <0) { return 0; }

    if (A[i][j] == 1) {return 0; }

    if (i == 0 && j==0) {return 1; }

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

      int a = ways ( A, i-1, j );

      int b = ways (A, i, j-1);

      dp[i][j] = a+b;

      return a+b;

}
```

Q.4 **min path sum**

Given a 2d matrix, filled with non-negative no., find the minimum cost path from (0,0) to (n-1, m-1).

→ R ↓ D

A =

	0	1	2
0	2	3	5
1	1	6	4
2	9	2	7

mincost = 18

A =

	0	1	2
0	2	2	100
1	10	100	200
2	1	1	3

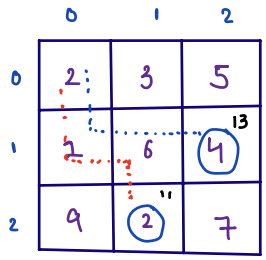
mincost = 17

A =

	0	1	2
0	2	2	100
1	10	100	200
2	1	1	3

Greedy soln won't work.

A =



a = mincost(i-1, j);

b = mincost(i, j-1);

ans = min(a, b) + A[i][j];

Code on IDE

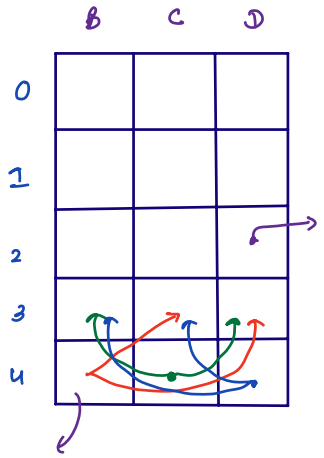
Doubts
=

A = [1 5 -3 4 -2]
0 1 2 3 4

B = 2

C = 1

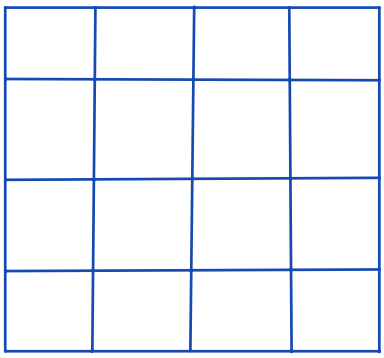
D = -1

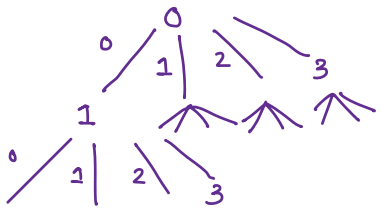


→ ans till 2 such that 2nd indexed is involved with D.

ans till 4 such that 4 is involved with B

max
- ans(i,B) => Max (ans(i-1,C), ans(i-1,D)) + A[i]*B
- ans(i,C) => Max (ans(i-1,B), ans(i-1,D)) + A[i]*C
- ans(i,D) => Max (ans(i-1,B), ans(i-1,C)) + A[i]*D





TC: $O(N^N)$

contest: syllabus is till backtracking.