Todays Content:

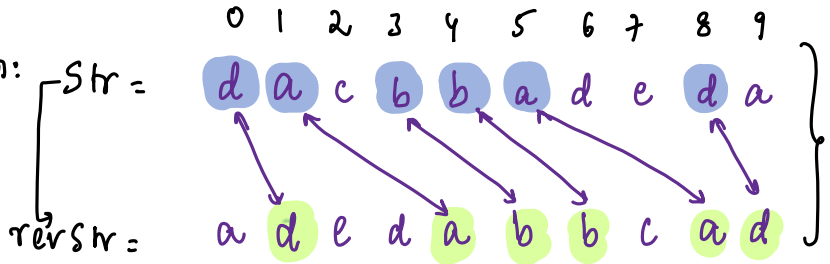
a. longest palindromic subsequence

b. N^{th} step:

a. longest palindrome subsequence:

Given a string, return length of longest palindrome subsequence.

Ex:



Q: longest common subsequence:

= d a b b a d ?

longest palindrome subsequence.

b. Min Cost to reach N^{th} step:

Cost to land at each staircase is given.

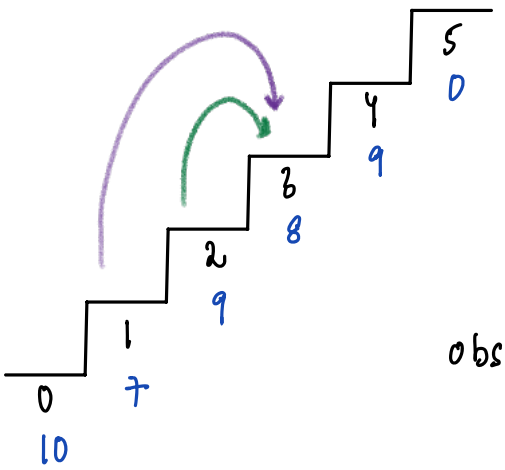
Stairs start at 0^{th} step, calculate min cost to reach N^{th} step:

Note: Cost to land at N^{th} step = 0

Eg: N = 5: Mincost to land at 5^{th} step

Cost to land at each step from 0..4 is given

	0	1	2	3	4
Cost[5] = {	10	7	9	8	9 }



minCost(5)

minCost(4) + cost[5]

minCost(3) + cost[5]

minCost(2) + cost[4]

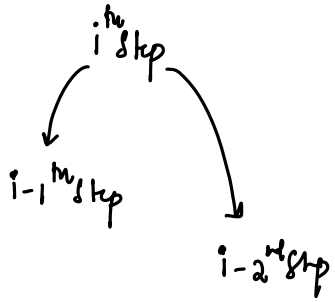
minCost(3) + cost[4]

minCost(1) + cost[3]

minCost(2) + cost[3]

obs: recursion + overlapping subProblems

Ass: Min Cost to reach i^{th} step:



	$i-1$	$i-2$	
i^{th} $i==0$:	-1	-2	x
$i==1$:	0	-1	*

```
int fun(int A[], int i) {
    if ( i==0 || i==1) {
        return A[i]
    }
    int c=0;
    if( i != A.length) { c= A[i]}
    return min( fun(A, i-1) , fun(A, i-2)) + c
}
```

```
int solve(int A[]) {
    int N= A.length; // We need to get min cost to reach N^th step
    return fun(A, N)
}
```

→

dpstate:

```
int dp[N+1] = -1;    dp[i] = Min Cost to reach i^th step, Final ans = dp[N]
int fun(int A[], int i) {
    if ( i==0 || i==1) {   → // Edge Cases
        return A[i]
    }
    if(dp[i] == -1) {
        int c=0;
        if( i != A.length) { c= A[i]}   → // Edge Case
        dp[i] = min( fun(A, i-1) , fun(A, i-2)) + c
    }
    return dp[i];
}
```