22/10/2023

* **House Robber II (leetcode - 213)**

Houses are arranged in circle i.e. 1st house is neighbour to last one. Condition is cannot rob in adjacent houses.
Given array nums representing money of each house, return maximum amount you can rob.

I/P → nums = [10, 20, 30, 40, 50]

O/P → 80

Explanation → Maximum amount can be robbed →

Rob house 3 (money = 30)
Rob house 5 (money = 50)

Total amount = 80

Case 1:

0	1	2	3	4
10	20	30	40	50

$e \in (0, n-2)$

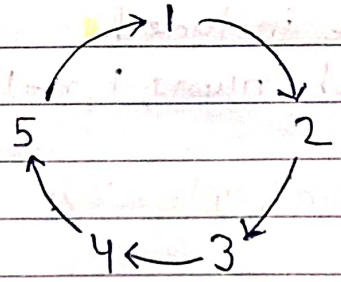
→ 10+30=40
→ 20+40=60
max → 60

Case 2:

0	1	2	3	4
10	20	30	40	50

$e \in (1, n-1)$

→ 20+40=60
→ 30+50=80
max → 80



Overall max = max(60, 80) = 80

Houses are arranged in circle. That's why 1 and 5 are neighbour to each other.

Code →

```
#include <bits/stdc++.h>
using namespace std;

int rob (vector<int>&nums , int s, int e){
        // base case
        if(s>e)  return 0;

// include
int option1 = nums[s] + rob (nums, s+2, e);
// exclude
int option2 = 0 + rob (nums, s+1, e);

return max (option1, option2);
}

int range (vector<int>&nums){
        int n = nums.size();
// case when there is single element only
        if (n==1)  return nums[0];
// when 1st house included
int option1 = rob(nums, 0, n-2);
// when last house included
int option2 = rob(nums, 1, n-1);

return max (option1, option2);
}
int main(){
  vector<int>nums{10, 20, 30, 40, 50};
  int ans = range(nums);
  cout << ans << endl;
}
```

* **Count Dearrangement →**

Permutation of n elements such that no element appears in its original position.

Base cases → • if (n==1) return 0;
for single element, let {2}, there is no dearrangement possible. That's why returning 0.

• if (n==2) return 1;
for two elements, let {2,3}, there is one dearrangement possible only that is {3,2}. That's why returning 1.

Logic → let say, there are n elements →

{1, 2, 3, 4, 5, ----, i, ---n-2, n-1, n}

for element 1, there are n-1 dearrangements possible.
Now, there are 2 possible cases →

Case 1: When element 1 is swapped with element i. Now, their positions are interchanged. We have fixed their position. Now, we are left with (n-2) elements for dearrangement.

{1, 2, 3, 4, 5, ---, i, ----, n-2, n-1, n}

Swapped and fixed

Recursive relation →
(n-1) * f(n-2)

Case 2: When element 1 is placed at position of element i but element i is placed at some other position rather than element 1's position.

{1, 2, 3, 4, 5, ---, i, ---, n-2, n-1, n}

new position of element 1 → ith element position

new position of element i → any position other than element 1.

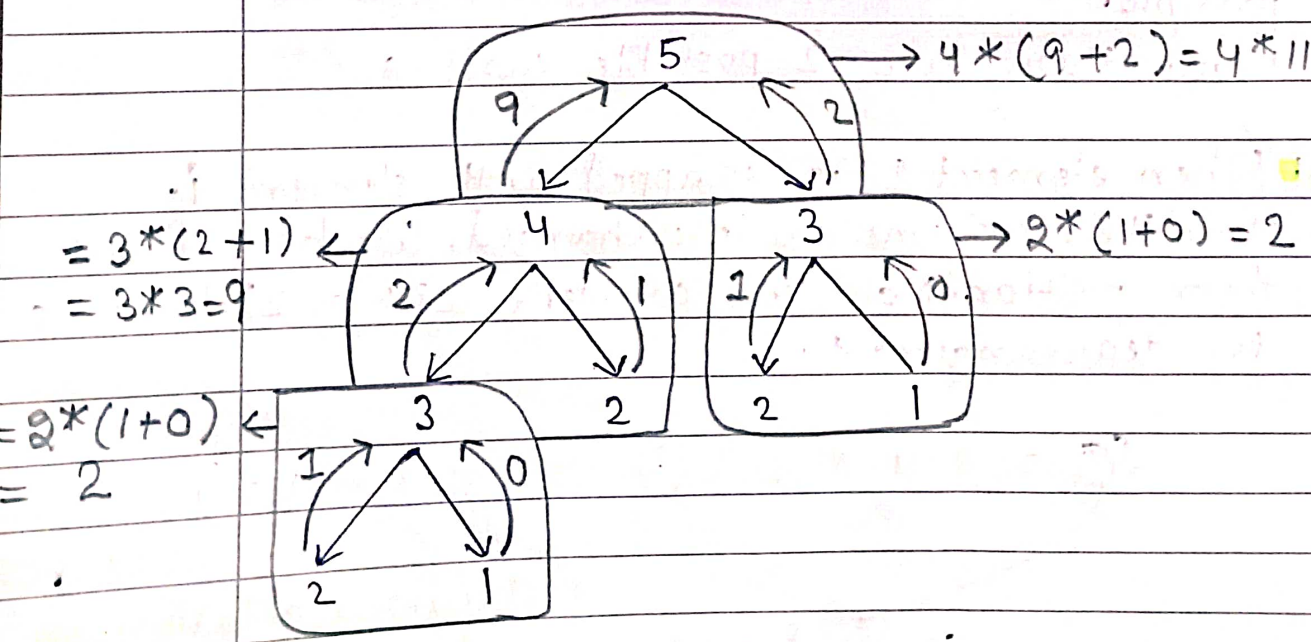
Now, we are left with (n-1) elements to dearrange.

Recursive relation →
$(n-1) * f(n-1)$

Overall Recursive relation →

$$(n-1) * (f(n-1) + f(n-2))$$

* let (recursive tree) for n = 5 →



* Let for another example $n = 4$

Recursive relation → $ans = (n-1) * (countD(n-1) + countD(n-2))$

countD(4)

ans = 3 * (countD(3) + countD(2)) = 3*(2+1) = 9

2 1

ans = 2 * (countD(2) + countD(1))

1 0

= 2 * (1+0) = 2

Code →

```
#include<bits/stdc++.h>
using namespace std;

int countD(int n){
    // base case
    if (n==1) return 0;
    if (n==2) return 1;

    // recursive call
    return (n-1) * (countD(n-1) + countD(n-2));
}
int main(){
    int n = 5;
    int countIs = countD(n);
    cout << countIs << endl;
}
```

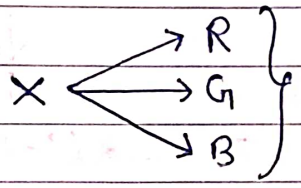
I/P → 5

O/P → 44

* **Painting Fence →**

Given a fence with n posts and k colors, find out the number of ways of painting the fence so that not more than two consecutive posts have same colors.

⇐ let for $n = 1$, $k = 3$ (R, G, B)



Possible ways = 3

Generalising, for $n = 1$, there are k possible ways.

⇐ for $n = 2$, $k = 3$

X X

f(2)

Same — Different

3 i.e K: RR, GG, BB

RG, RB, GR, GB, BG, BR

6 i.e 3×2

Generalising, for $n = 2$, when same color = k possible ways
when different color = $K*(K-1)$ possible ways.

Note → Jb possible ways find krne hai same color ke case mai, then last ke two posts se start krte hue same banane hai like X(XX) → This one
not this one ← (XX)X

For $n=3$, $k=3$

X X X

f(3)

same | Different

6	RGG	RRG	RGR	RGB	24
	RBB	GGB	RBR	RBG	
	GRR	BBG	GRG	GRB	
	GBB	RRB	GBG	GBR	
	BGG	GGR	BGB	BGR	
	BRR	BBR	BRB	BRG	

Generalising, for $n=3$, when same color = $f(n-2)*(k-1)$
when different color = $f(n-1)*(k-1)$ possible ways

Brief of above three cases → for k colors

Total ways = same + different

- $n = 1$
  Total ways = K

- $n = 2$
  same → K
  different → $K*(K-1)$
  Total ways = $K + K*(K-1)$

- $n = 3$
  same → $K*(K-1)$
  different → $(K+K*(K-1))*(K-1)$
  Total ways = $K*(K-1)+(K+K*(K-1))*(K-1)$

So, we can conclude that →

$f(n)_{same} = f(n-1)_{different}$

$f(n-1)_{different} = f(n-2)_{total} * (k-1)$

[If A = B, B = C, then, A = C]

That's why $f(n)_{same} = f(n-2)_{total} * (k-1)$

for same = $f(n) = f(n-2) * (k-1)$

When posts are n-1,

$f(n-1)_{different} = f(n-1-1)_{total} * (k-1)$

When posts are n,

$f(n)_{different} = f(n-1)_{total} * (k-1)$

For different = $f(n) = f(n-1) * (k-1)$

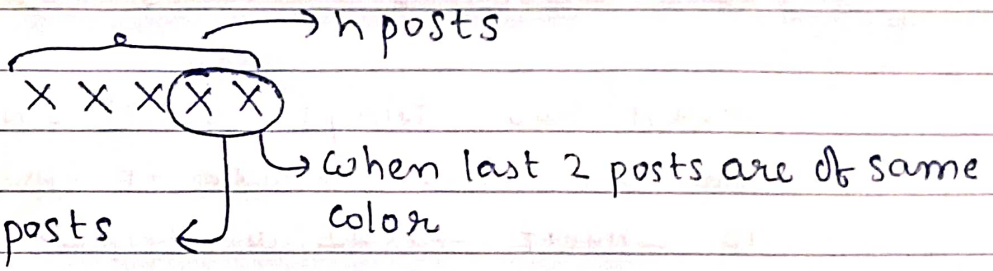
Overall recursive relation →

Total = same + different

$f(n) = f(n-2) * (k-1) + f(n-1) * (k-1)$

$f(n) = (k-1) * (f(n-2) + f(n-1))$

- Same or different here means,



Code →

```
#include <bits/stdc++.h>
using namespace std;

int getPaintWays (int n, int k){

    // base case
    if (n==1)   return k;
    if (n==2)   return k + k*(k-1);

    // recursive relation
int ans = (k-1)*(getPaintWays(n-2,k) + getPaintWays(n-1,k));
return ans;
}

int main(){
    int n=4;
    int k=3;

    int ans = getPaintWays(n,k);
    cout << ans << endl;
}
```

I/P = n=4, k=3

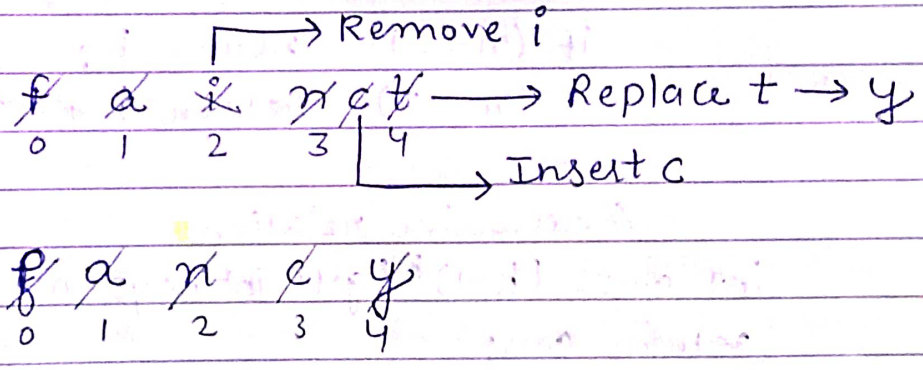
O/P = 66

* Edit Distance (leetcode - 72)

Given two strings word1 and word2, return the minimum number of operations required to convert word1 to word2.

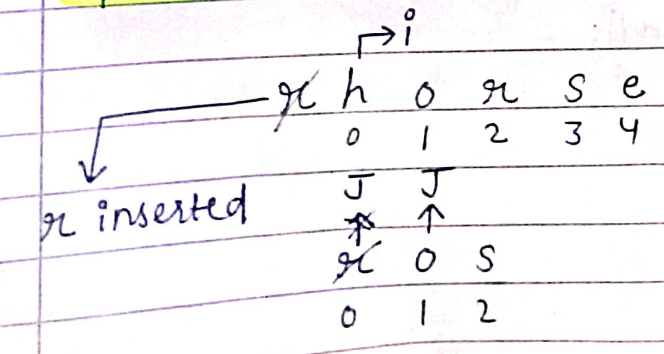
Operations permitted →
- Insert a character
- Delete a character
- Replace a character

Example → word1 = "faint", word2 = "fancy"
Output = 3



- f a int → fant (Remove 'i')
- fant → fanct (Insert 'c')
- fanct → fancy (Replace 't' with 'y')

Operation 1 → insert

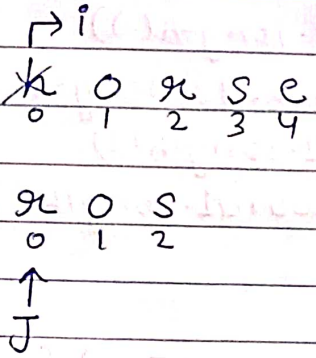


Jb we insert r in string horse. Now, it is rhorse. rhorse ka r and ros ka r cancel ho gye. lekin h cancel nhi hua. That's why i+1 nhi kiya and J+1 kiya.

So, insert operation mai i ko increment nhi karna and J ko karna hai.

i.e (i, J+1)

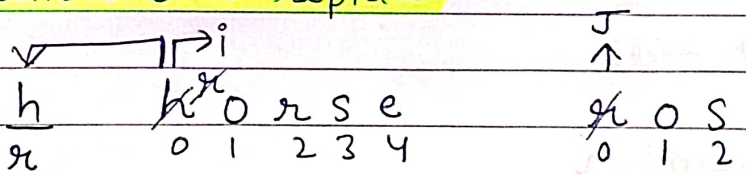
operation 2 → delete



Jb we deleted h from string horse. Now, it is orse while another string is as it is i.e. ros. Now, we compare o of string orse which is at index i+1 and r of string ros which is at index J.

So, delete operation mai i ko increment karna hai and J ko nhi karna hai. i.e. (i+1, J)

operation 3 → replace



Now, horse is rorse. rorse ka r and ros ka r cancel out ho gaye. That's why i and J both are incremented i.e. (i+1, J+1)

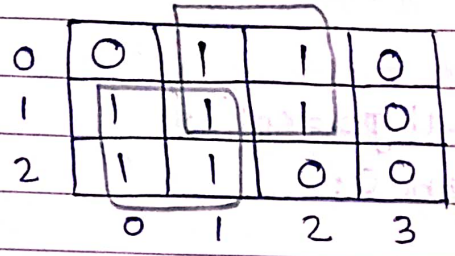
Code →

```
#include <bits/stdc++.h>
using namespace std;

int minOperations (string &word1, string &word2,
                   int i, int J)
{
    //base case
    if (i >= word1.length())
        return word2.length() - J;
    if (J >= word2.length())
        return word1.length() - i;

int ans= 0;
    if (word[i] == word[J]){ //match
    ans = 0 + minoperations(word1, word2, i+1, J+1);
    }
    else{
    // not matched, operations perform kro
//insert int op1 = 1 + minOperations(word1, word2, i, J+1);
//delete int op2 = 1 + minOperations(word1, word2, i+1, J);
//replace int op3 = 1 + minoperations(word1, word2, i+1, J+1);

    ans = min(op1, min(op2, op3));
}
return ans;
}
int main(){
   string word1 = "faint";
   string word2 = "fancy";
    int i=0;
    int J=0;
   cout << minOperations(word1, word2, i, J);
}
```

* **Maximal square (Leetcode - 221)**

Given an **mxn** binary matrix filled with '0's and '1's. Find the largest square containing only '1's and return its area.

Input →



Output → 4

= **Base case →**

Jb i and J out of bound chale jaye

= **Recursive calls →**

har cell ke liye 3 calls jayengi →

- towards right
- towards diagonal
- towards down

} Accordingly i and J change honge depends kis side ki call aayi hai

= If cell contains character '1', tbhi square consider krna hai else return 0.

= Jitne bhi square bn rhe honge, unme se jo maximum side ka square hai, vo return krna hai.

Code →

```
#include <bits/stdc++.h>
using namespace std;

int maximalSq(vector<vector<char>>& matrix, int i,
              int j, int row, int col, int &maxi){

    //base case
    if(i>=row || j>=col){
        return 0;
    }

    //recursive calls->
int right= maximalSq(matrix, i, j+1, row, col, maxi);
int diagonal= maximalSq(matrix, i+1, j+1, row, col, maxi);
int down= maximalSq(matrix, i+1, j, row, col, maxi);

 if (matrix[i][j]=='1'){
int ans= 1+ min(right, min(diagonal, down));
maxi= max(maxi, ans);
return ans;
}
else{
    //Jb character '0' hai
    return 0;
  }
}
```

```
int main(){

    vector<vector<char>> matrix
    {
      {'0','1','1','0'},
      {'1','1','1','0'},
      {'1','1','0','0'}
    };

    int i=0;
    int j=0;
    int row= matrix.size();
    int col= matrix[0].size();
    int maxi=0;
    int ans= maximalSq(matrix, i, j, row, col, maxi);
    cout<< maxi * maxi <<endl;
}
```

O/P → 4