Problem Solving 3

01. Maximal Rectangle in Matrix
02. 0-1 knapsack (space optimised)
03. Count Binary strings with no consecutive 0s / Ways to give signal
04. Friends pairing / Lets party
05. Distinct subsequences (string DP)

* Find maximum area of rectangle in a histogram

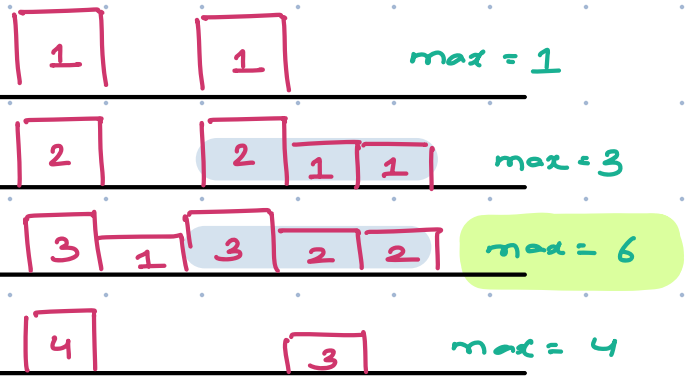
```
int rectangularArea(int[] ht)
    int[] left = nearestSmallerIdxOnLeft(ht); // default val = -1
    int[] right = nearestSmallerIdxOnRight(ht); // default val = n & not -1
    int area = 0, ans = 0
    for (i=0; i<n; i++){
        int h = ht[i];
        P1 = left[i]
        P2 = right[i]
        area = h * (P2 - P1 - 1)
        ans = Math.max(ans, area)
    }
    return ans;
}
```

TC : O(n)
SC : O(n)

Maximal Rectangle

Given a 2D binary matrix filled with 0's & 1's, find largest rectangle containing only ones

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




```
int maximalRectangle (int [][] A){
    if (n==0) return 0;
    int [] ht = new int [A[0].length];
    int area = 0
    for (i=0; i<n; i++){
        for(j=0; j<m; j++){
            int val= A[i][j];
            if (val == 1){
                ht[j] += val;
            }
            else ht[j]= 0
        }
        area = Max (area, rectangularArea(ht));
    }
}
```

TC : O(n*m)
SC : O(m)

* 0-1 KnapSack

```
int [][] dp = new int[n+1][cap+1];

for (i=0; i<=n; i++){
    for (j=cap; j>=0; j--){
        if (i==0 || j==0) dp[i][j] = 0;
        else {
            rej = dp[i-1][j]
            sel = 0
            if (wt[i-1] <= j)
                sel = dp[i-1][j-wt[i-1]] + val[i-1];
            dp[i][j] = max(sel, rej);
        }
    }
}
return dp[N][cap];
```

TC : O(n*cap)

SC : O(n*cap)

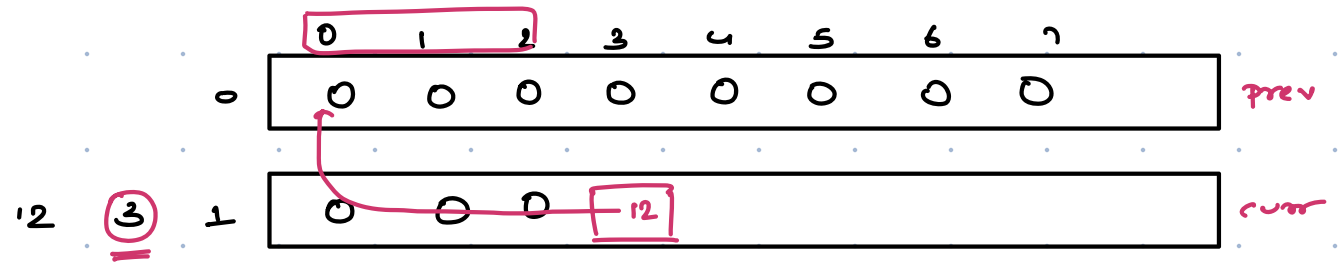
val	wt	Ele ↓ / Cap	0	1	2	3	4	5	6	7
		0	0	0	0	0	0	0	0	0
12	3	1	0	0	0	12	12	12	12	12
20	6	2	0	0	0	12	12	12	20	20
15	5	3	0	0	0	12	12	15	20	20
6	2	4	0	0	6	12	12	18	20	21 ✓
10	4	5	0	0	6	12	12	18	20	22

```
int [] prev = new int [cap+1]
int [] curr = new int [cap+1];

for (i=0; i≤n; i++){
    for (j=cap; j≥0; j--){
        if (i==0 || j==0)   curr[j] = 0;
        else {
            rej= prev[j];
            sel = 0
            if (wt[i-1] ≤ j)
                sel = prev[j- wt[i-1]] + val[i-1];
            curr[j] = max(sel, rej);
        }
    }
    prev = curr;
}
return curr[c];
```

TC : O(n*cap)

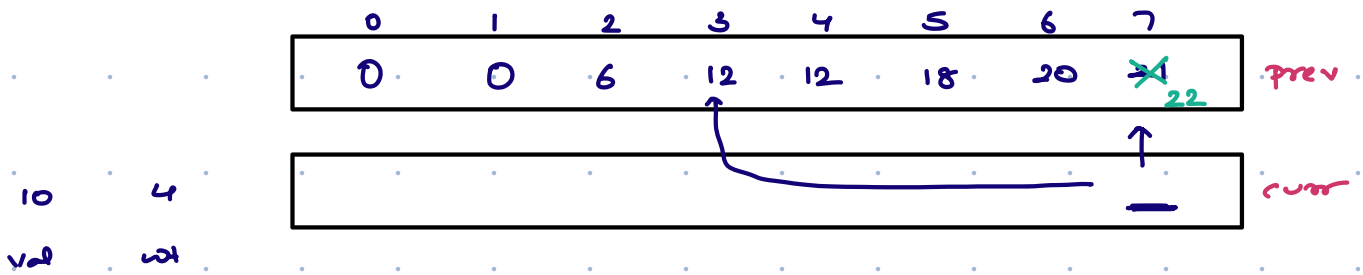
SC : O(2*cap)



j = 3

sel = prev[j - wt[i-1]] + val[i-1]

* Optimise to a single Arr



rej = prev[j] = 21

sel = prev[3] + 10 = 12+10 = 22

```
int [] prev = new int [cap+1]

for (i=0; i≤n; i++){
    for (j=cap; j≥0; j--){
        if (i==0 || j==0) prev[j]=0
        else {
            rej= prev[j];
            sel = 0
            if (wt[i-1]≤j)
                sel= prev[j-wt[i-1]] + val[i-1];
            prev[j] = max(sel, rej);
        }
    }
}
return prev[c];
```

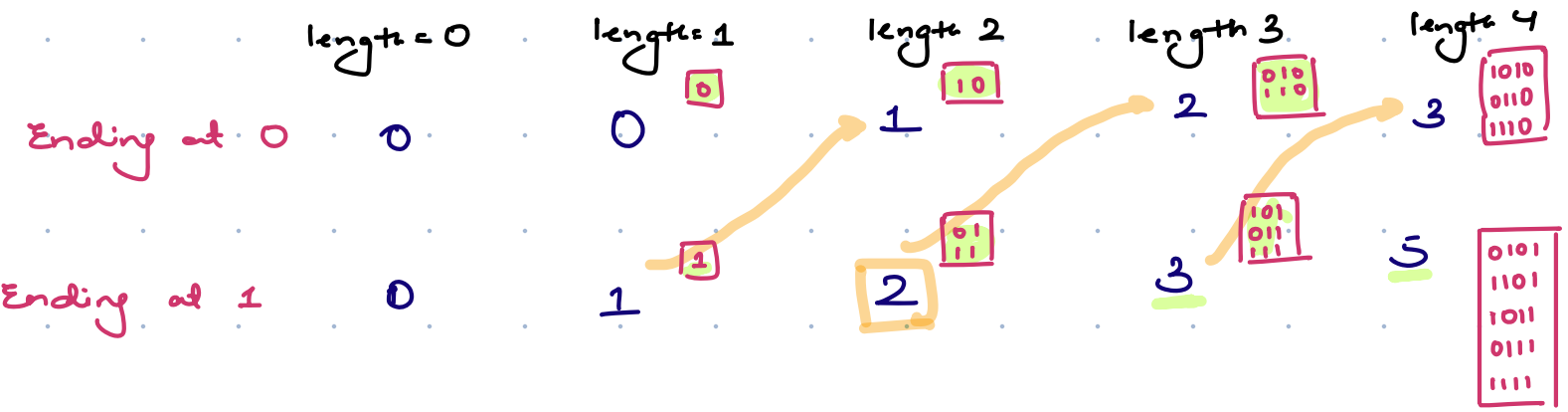
TC : O(n*m)
SC : O(m)

* **Count Binary strings with no consecutive 0's**

Given a no. n, count no. of binary strings of length n with no consecutive 0's

Similar questions → ways to send signal
→ House robber
→ Arrange buildings
→ Paint house
→ Paint fence
→ Friends pairing

If we have a binary string, then it can either ends at 0 or 1.



	0	1	2	3	4	5	6	n
dp0[n+1] =	0	1	1	2	3	5	8	13
dp1[n+1] =	0	1	2	3	5	8	13	21

for n = 7

Ans = 34

```
int [] dp0 = new int [n+1];
int [] dp1 = new int [n+1];
dp0[1] = 1
dp1[1] = 1
for (i=2; i<=n; i++){
    dp0[i] = dp1[i-1];
    dp1[i] = dp0[i-1] + dp1[i-1];
}
return dp0[n] + dp1[n];
```

* Recursive code

```
int helper (int n, int ending) {
    if (n == 1) return 1
    int count = 0
    if (endwith == 0) {
        count += helper(n-1, 1);
    }
    else {
        count += helper(n-1, 0) + helper(n-1, 1);
    }
}
```

* Friends pairing / Let's party (DP 4)

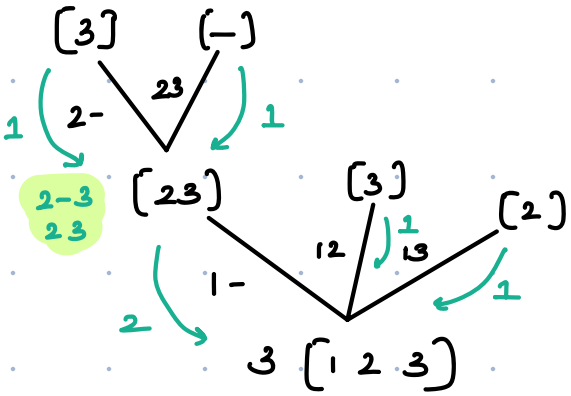
1. You are given a number n, representing the number of friends.
2. Each friend can stay single or pair up with any of it's friends.
3. You are required to print the number of ways in which these friends can stay single or pair up.

E.g.

1 person can stay single or pair up in 1 way.

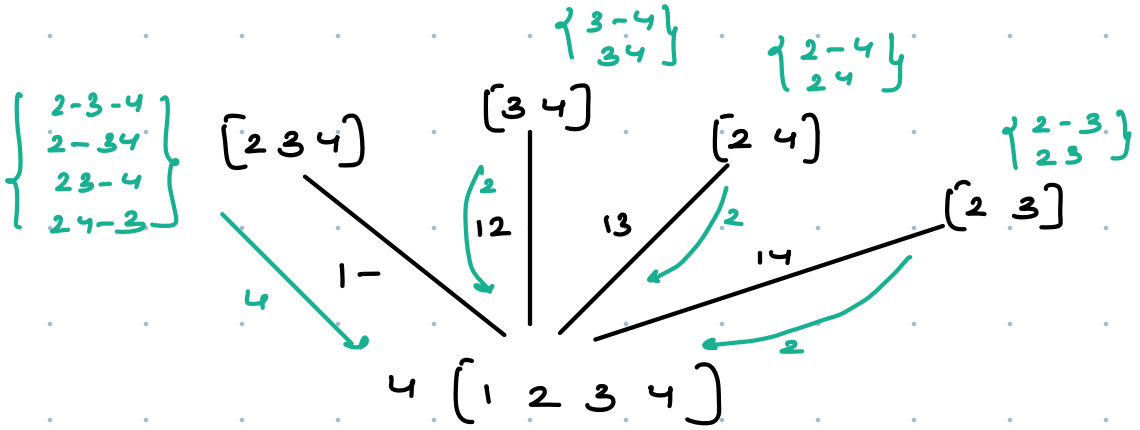
2 people can stay singles or pair up in 2 ways. 12 = 1-2, 12.

3 people (123) can stay singles or pair up in 4 ways. 123 = 1-2-3, 12-3, 13-2, 23-1.

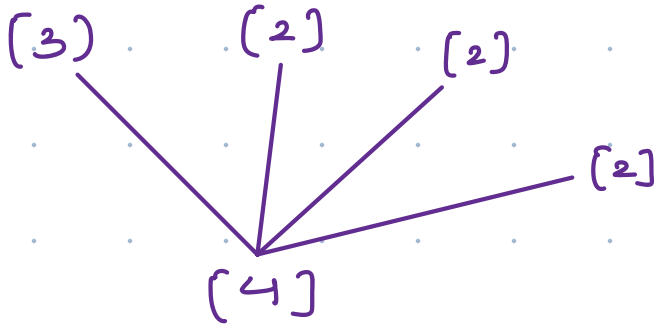


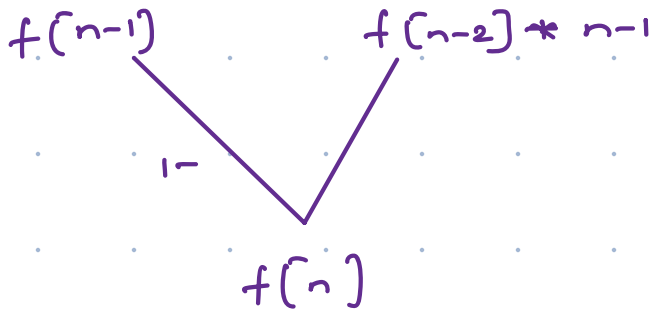
Ans=4

[1-2-3, 1-23, 12-3, 13-2]



Ans= 10






```
dp[1] = 1
dp[2] = 2

for (i=3; i≤n; i++) {
    dp[i] = dp[i-1] + dp[i-2] * (i-1)
}
return dp[n];
```

* Distinct Subsequences (Dp on string)

Given two sequences A & B, count no. of unique ways in sequence A to form a subsequence identical to B.

str_1 = b a b g b a g
0 1 2 3 4 5 6

str_2 = b a g

Subseq

0 1 3
0 1 6
0 5 6
2 5 6
4 5 6

Ans = 5

str_1 = r a b b b i t
0 1 2 3 4 5 6

str_2 = r a b b i t

subseq

0 1 2 3 5 6
0 1 2 4 5 6
0 1 3 4 5 6

Ans = 3

str_1 = b a b g b a g , b a g

(str1, str2, i, j)

if (str1[i] ≠ str2[j])

(str1, str2, i-1, j)

if (str1[i] == str2[j])

(str1, str2, i-1, j-1)

+

(str1, str2, i-1, j)

Ans = 3

(rabbbit, rabbit)

3

0

(rabbbi, rabbi)

(rabbbi, rabbit)

3

(rabbb, rabbi)

(rabbb, rabbi)

0

(rabbb, rabbit)

(i < j) return 0

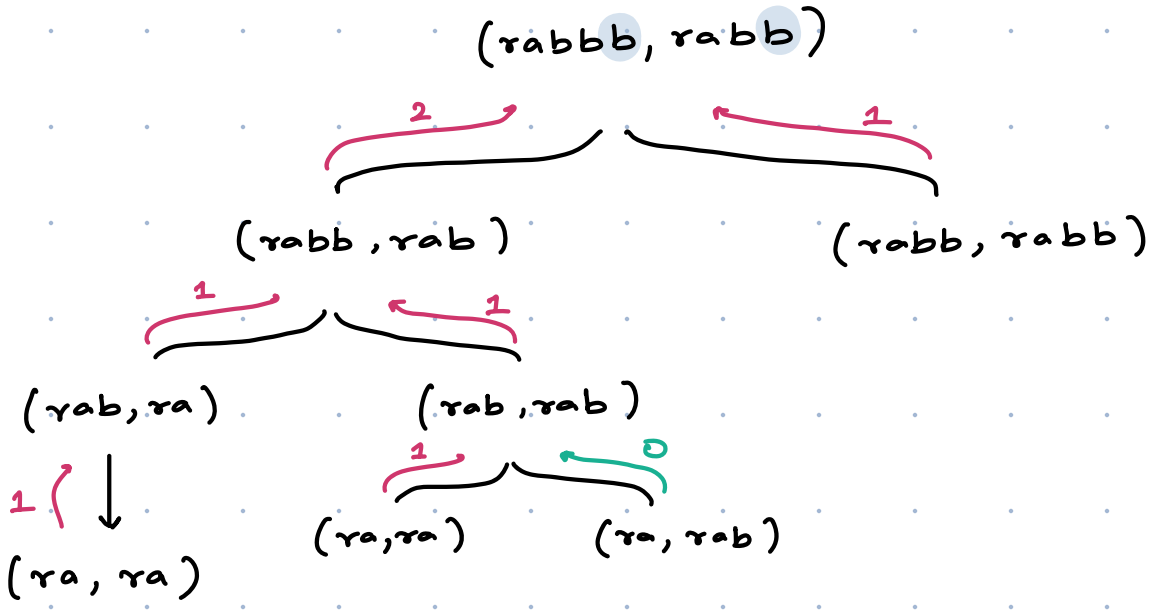
(rabb, rabb)

(rabb, rabbi)

0

(rabb, rabbi)

Expanded below



n-1 m-1

```
int distinct ( str1 , str2 , i , j ){

    if ( j == -1 ) return 1 ;

    if ( i < j ) return 0;

    int x = distinct ( str1 , str2 , i-1, j-1 );

    int y = distinct ( str1 , str2 , i-1, j );

    if ( str1[i] == str2[j] ) return x+y

    else return y;

}
```

```java
class Solution {
    public int numDistinct(String s, String t) {
        int n=s.length();
        int m=t.length();
        int[][]dp=new int[n+1][m+1];
        for(int[]d:dp)Arrays.fill(d,-1);
        return distinct(s,t,n,m,dp);
    }
    public int distinct(String s,String t,int n,int m,int[][]dp){
        if(m==0) return dp[n][m]=1;
        if(n<m) return dp[n][m]=0;

        if(dp[n][m]!=-1){
            return dp[n][m];
        }

        int a=distinct(s,t,n-1,m-1,dp);
        int b=distinct(s,t,n-1,m,dp);

        if(s.charAt(n-1)==t.charAt(m-1)){
            return dp[n][m]=a+b;
        }else{
            return dp[n][m]=b;
        }
    }
}
```

* N chocolates

Problem Description

You have **N** Chocolates arranged in a queue. Each chocolate **A[i]** is one of the **B** types. His friend likes when chocolates of the same type are together.

To make your friend happy you decided to eat some chocolates from the queue but your friend restricted that you can eat **atmost C** chocolates from the queue. After eating, the remaining chocolates are joined together(so that the gaps are closed in the queue).

You must find the maximum number of chocolates of the same type in the queue after eating at most **C** chocolates.

Note: Chocolates of the same type that are adjacent are counted in answer

Example Input

Input 1:

```
A = [1, 2, 1, 1, 2, 1, 2, 2, 2]
B = 2
C = 2
```

Input 2:

```
A = [1, 2, 1, 1, 2]
B = 2
C = 2
```

Example Output

Output 1:

```
4
```

Output 2:

```
3
```

Example Explanation

Explanation 1 -

```
It is optimal if you eat the chocolates at positions 2 and 5 then queue will look like this
[1, 1, 1, 1, 2, 2, 2] and one more way is to eat chocolate at position 6 then the queue will
look like [1, 2, 1, 1, 2, 2, 2, 2]. In both case we will get the optimal answer which is 4.
```

Explanation 2 -

```
It is optimal if you eat the chocolates at positions 2 then queue will look like this [1, 1, 1, 2].
In this case we will get the optimal answer which is 3.
```

B = 2

C = x x ~~0~~ ~~1~~ 0

{ 1 2 1 1 2 1 2 2 2 }

O(n)

length = 6 − C = 6 − 2 = 4

A[ ] : { 1 2 1 1 2 }

```
public class Solution {
    public int solve(ArrayList<Integer> A, int B, int C) {
        List<List<Integer>> pos = new ArrayList<>();
        for (int i = 0; i <= B; i++) {
            pos.add(new ArrayList<>());
        }
        for (int i = 0; i < A.size(); i++) {
            pos.get(A.get(i)).add(i);
        }
        int ans = 0;
        for (List<Integer> i : pos) {
            int temp = 0;
            int left = 0, right = 1;
            while (right < i.size()) {
                if (temp + i.get(right) - i.get(right - 1)
- 1 <= C) {
                    temp += i.get(right) - i.get(right - 1)
- 1;
                    right++;
                } else {
                    ans = Math.max(ans, right - left);
                    temp -= i.get(left + 1) - i.get(left) -
1;
                    left++;
                }
            }
            ans = Math.max(ans, right - left);
        }
        return ans;
    }
}
```