

Dynamic Programming



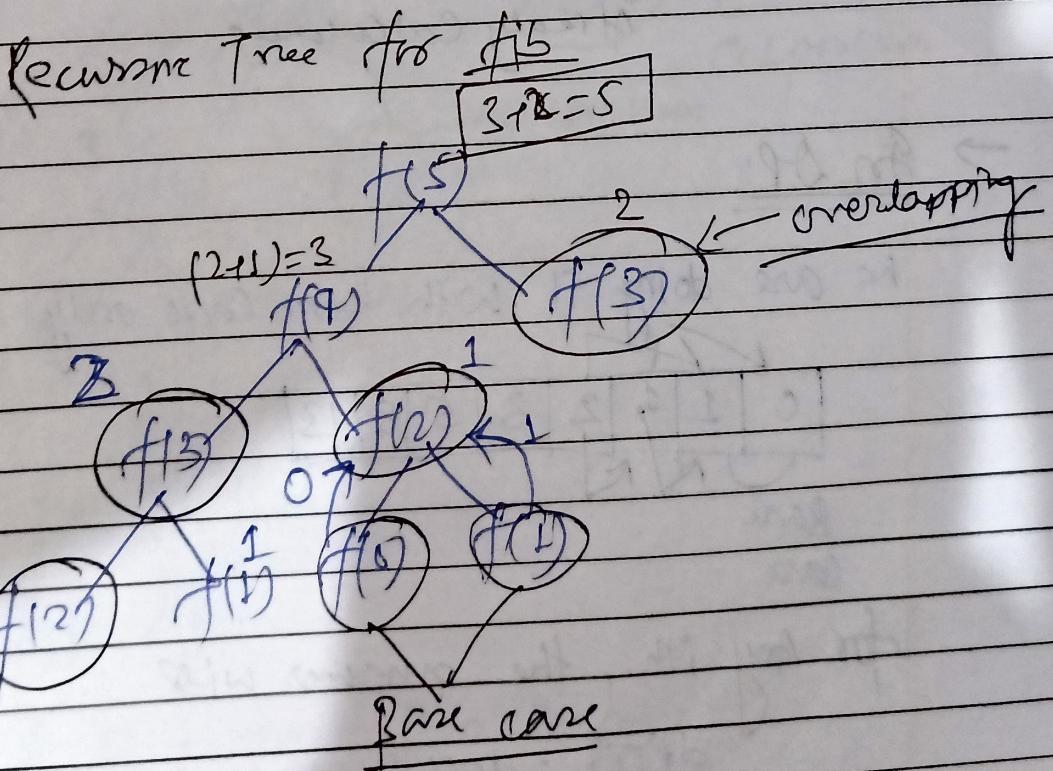
Not EASY BUT BUILD Logic

- i) Tabularim :- Bottom UP (from leaf)
- ii) Memorizatim :- TOP - DOWN

1. Fibonacci NO

0 1 1 2 3 5 8 13 21 ...

$$f(n) = f(n-1) + f(n-2) \quad (\text{Recursive Relation})$$



Tabulation.

Try to go from Base Case to the required Answer

In Memorization, What we are doing

- Initially filled the array with -1 And during iteration if array is filled with some other value then we have already calculated for that index and simply return that.

-1	-1	-1	-1	-1	-1
↓ after some iteration					
-1	-1	2	5	4	12
↓ already calculated					

- By DP.

we are doing it with base case only

0	1	1	2	3	5	8	13

Bare
case

for Any i^{th} , the answer will

$$dp[i] = dp[i-1] + dp[i-2]$$

6/28/22, 4:55 PM

fibonacci.cpp

```
1 #include<bits/stdc++.h>
2 using namespace std;
3
4
5
6 int recursiveFib(int n,vector<int>&memo)
7 {
8     if(n<=1)
9     {
10         return 1;
11     }
12     if(memo[n]!=-1)
13     {
14         return memo[n];
15     }
16
17     return recursiveFib(n-1,memo)+recursiveFib(n-2,memo);
18 }
19 /*
20
21 In tabulation approach
22
23 we need to calculate our answer through give base case
24
25 */
26 int dpFib(int n)
27 {
28     int dp[n+1];
29     dp[0]=1;
30     dp[1]=1;
31
32     for(int i=2;i<=n;i++)
33     {
34         dp[i]=dp[i-1]+dp[i-2];
35     }
36     return dp[n];
37 }
38
39
40 int main()
41 {
42     int n;
43     cin>>n;
44     vector<int>memo(n+1,-1);
45     int ans1=recursiveFib(n,memo);
46     int ans2=dpFib(n);
47     cout<<ans1<<" "<<ans2<<endl;
48 }
49
50
51 /*
52
53 In recursive + memoization
54
55
56 time complexity: O(n)
57 space complexity: O(n) for array + O(n) because of recursion stack
58
59
```

6/28/22, 4:55 PM

fibonacci.cpp

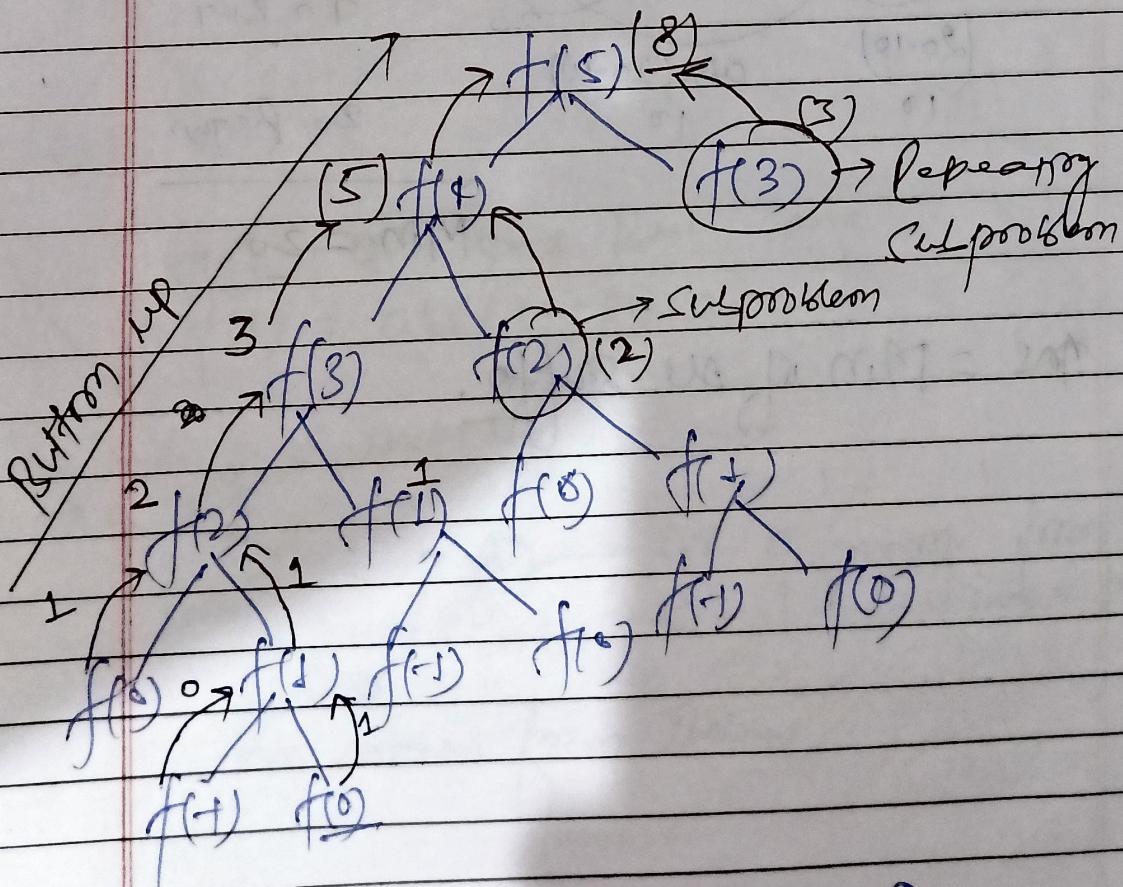
```
60 In dp
61
62
63 time complexity: O(n)
64 space complexity: O(n)
65 */
66 */
```

2. Count No of ways to reach N by taking 1 or 2 steps

→ Exactly same as fibonacci number calculation.

→ If $n=0$, that means you can not move anywhere so number of moves will be 1.

Recursion Tree for $n=3$



→ if n is neg then return 0.

6/28/22, 4:59 PM

```
countWaysToReachN.cpp

1 /*
2 you have to take only 1 or 2 steps
3
4 then count number of ways to reach N and return it ;
5
6 */
7 #include<bits/stdc++.h>
8 using namespace std;
9
10 int recursiveCount(int n,vector<int>&memo)
11 {
12     if(n==0)
13     {
14         return 1;
15     }
16     if(n<0)
17     {
18         return 0;
19     }
20     if(memo[n]!=-1)
21     {
22         return memo[n];
23     }
24
25     return recursiveCount(n-1,memo)+recursiveCount(n-2,memo);
26 }
27
28 int dpCount(int n )
29 {
30     int dp[n+1];
31     dp[0]=1;
32     for(int i=1;i<=n;i++)
33     {
34         dp[i]=dp[i-1]+ (i-2>=0?dp[i-2]:0);
35     }
36     return dp[n];
37 }
38
39
40 int main()
41 {
42     int n;
43     cin>>n;
44     vector<int>memo(n+1,-1);
45     int ans1= recursiveCount(n,memo);
46     int ans2= dpCount(n);
47
48     cout<<ans1<<" "<<ans2<<endl;
49 }
50
51 /*
52
53 In recursive + memoization
54
55
56 time complexity: O(n)
57 space complexity: O(n) for array + O(n) because of recursion stack
58
59
```

localhost:4649/?mode=clike

6/28/22, 4:59 PM

```
countWaysToReachN.cpp

60 In dp
61
62
63 time complexity: O(n)
64 space complexity: O(n)
65
66 */
```

1/2

localhost:4649/?mode=clike

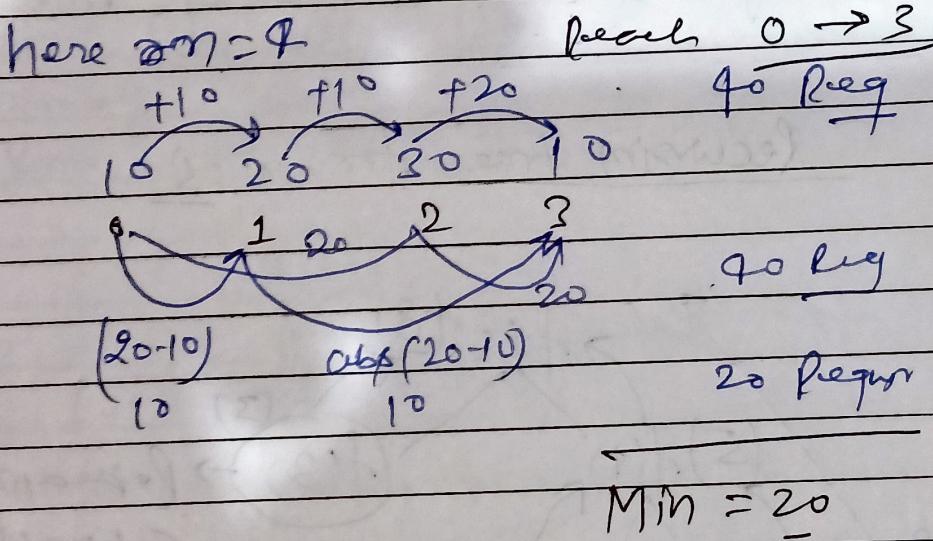
2/2

3. Frog jump

a_i = Energy

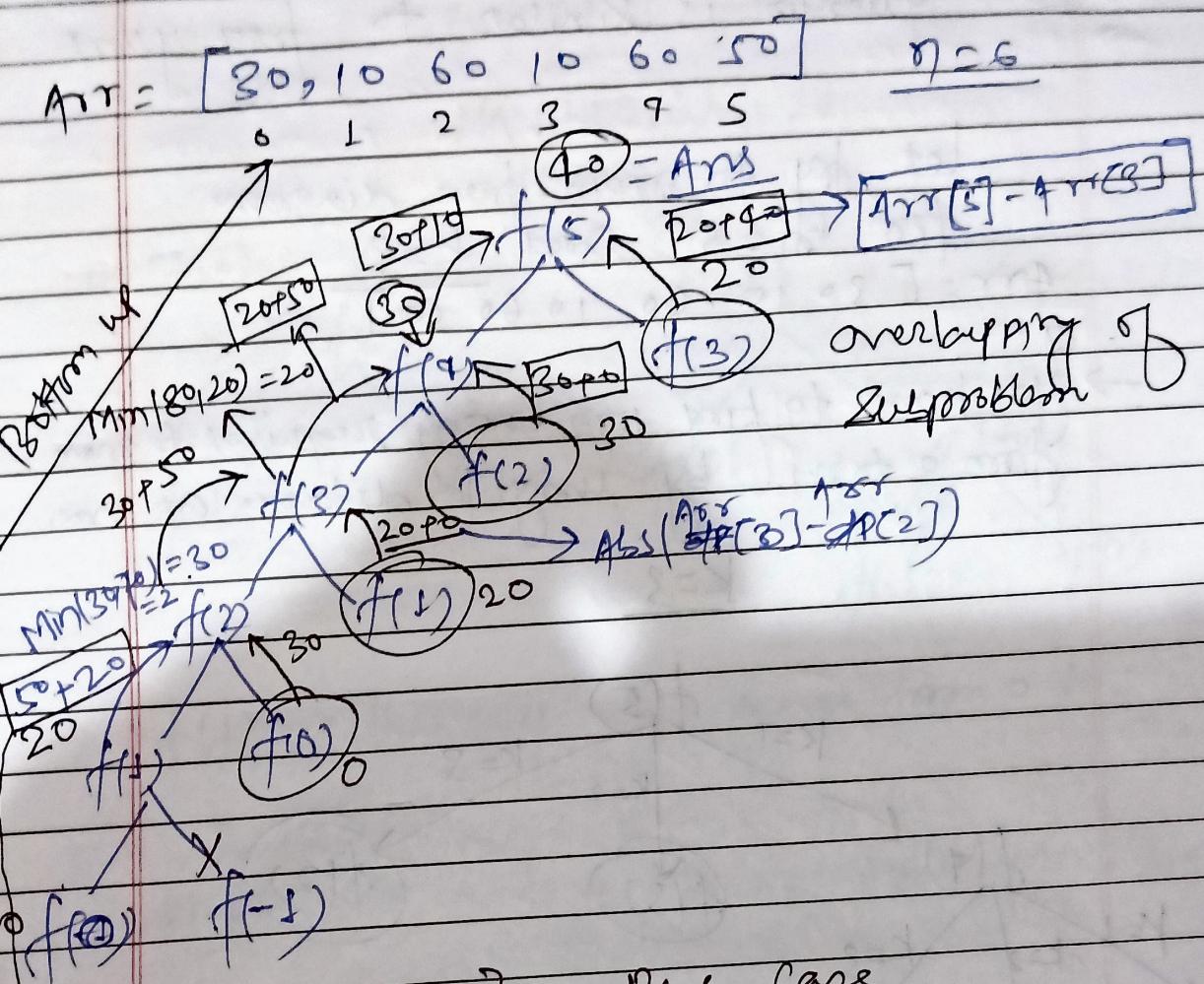
you have to frog from energy requirement
to reach from 0 to 7-1 by taking
1 or ~~2~~ 2 steps ahead

If you are at $a_i \rightarrow$ then you can move
 a_{i+1} or a_{i+2} .

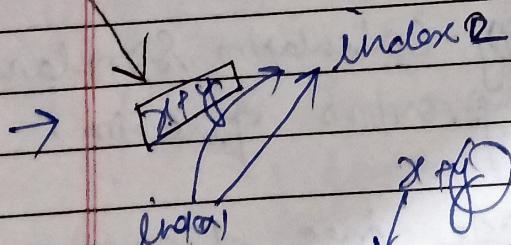


$\text{Ans} = \text{Min of all stuff}$.

Recursion Tree Diagram



$$\begin{cases} f(0) = 0 \\ f(1) = \text{abs}(10 - 30) = 20 \end{cases} \rightarrow \text{Base Case}$$



cost of moving from index 1 to index 2
 min shows from below

6/28/22, 11:14 AM

FrogJump.cpp

```
1 /*
2 Ways to write recursive code
3
4 1. try to represent the problem in a terms of index
5
6 2.Do all possible stuffs on that index and accordingly to the problem statement
7
8 3. if question asking COUNT ALL WAYS ==> Sum of all the stuffs
9  if question asking MIN OR MAX ==> Then we will take max or min from the
calculated stuffs
10 */
11
12 #include<bits/stdc++.h>
13 using namespace std;
14
15 int recursiveJump(vector<int>&input,vector<int>&memo,int n)
16 {
17     if(n==0)
18     {
19         return 0;
20     }
21     if(n<0)
22     {
23         return INT_MAX;
24     }
25
26     if(memo[n]!=-1)
27     {
28         return memo[n];
29     }
30
31     int oneStep= recursiveJump(input,memo,n-1)+ abs(input[n]-input[n-1]);
32
33     int twoStep= n-2>=0?recursiveJump(input,memo,n-2)+abs(input[n]-input[n-
2]):INT_MAX;
34
35     return min(oneStep,twoStep);
36 }
37
38 int dpJump(vector<int>&input)
39 {
40     int dp[input.size()];
41     dp[0]=0;
42     for(int i=1;i<input.size();i++)
43     {
44         dp[i]=min(dp[i-1]+abs(input[i]-input[i-1]),i-2>=0?dp[i-2]+abs(input[i]-
input[i-2]):INT_MAX);
45     }
46     return dp[input.size()-1];
47 }
48
49 int main()
50 {
51     int n;
52     cin>>n;
53 /*
54 6
55 30 10 60 10 60 50
56
```

6/28/22, 11:14 AM

FrogJump.cpp

```
57 output=40
58 */
59 vector<int>v(n,0);
60 for(int i=0;i<n;i++)
61 {
62     cin>>v[i];
63 }
64 vector<int>memo(n+1,-1);
65 int ans1= recursiveJump(v,memo,n-1);
66 int ans2= dpJump(v);
67 cout<<ans1<<" "<<ans2<<endl;
68 }
69
70
71 /*
72 In recursive + memoization
73 time complexity: O(n)
74 space complexity: O(n) for array + O(n) because of recursion stack
75
76 In dp
77 time complexity: O(n)
78 space complexity: O(n)
79
80
81 */
82 */
```

4. Frog jump with K distance

→ This question is similar to frog jump

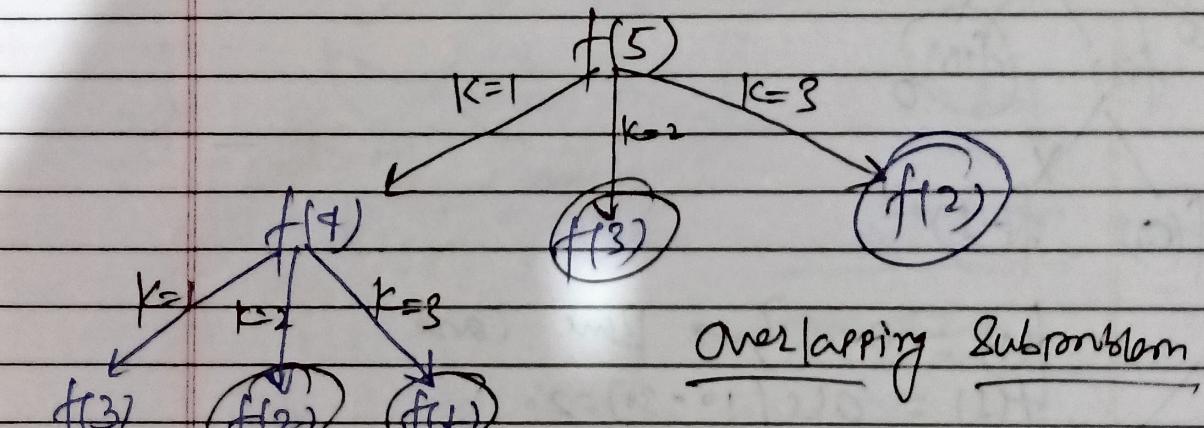
let try to make tree diagram

for $n=6$ and $k=3$

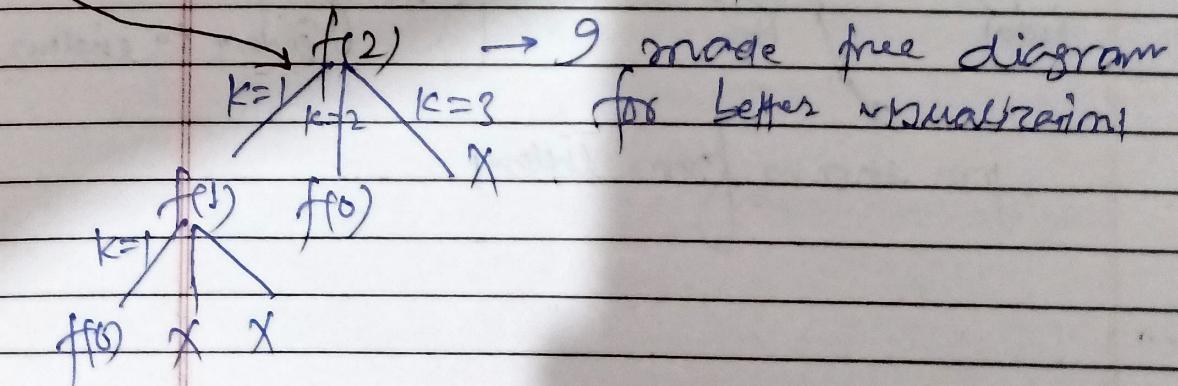
$$Arr = [30 \ 10 \ 60 \ 10 \ 60 \ 50]$$

→ You have to find min energy required to move from 0 to n by using k distance at max.

$K=3$



→ Energy calculation similar to previous question.



6/28/22, 11:15 AM

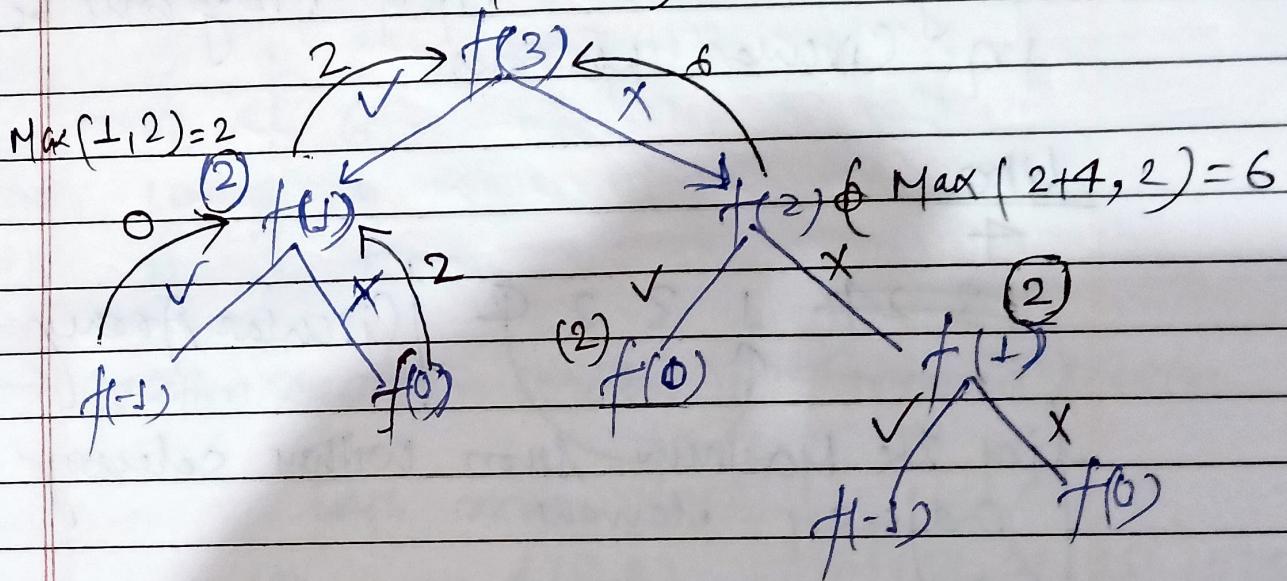
FrogJumWithKdistance.cpp

```
1 /*
2     Ways to write recursive code
3
4     1. try to represent the problem in a terms of index
5
6     2.Do all possible stuffs on that index and accordingly to the problem
7     statement
8
9     3. if question asking COUNT ALL WAYS == > Sum of all the stuffs if question
10    asking MIN OR MAX ==
11        > Then we will take max or
12        min from the calculated stuffs
13 */
14
15 #include<bits/stdc++.h>
16 using namespace std;
17 /*
18 this problem is extension of frogJump
19 */
20
21 int recursiveJump(vector<int> &input, vector<int> &memo, int n, int k)
22 {
23     if (n == 0)
24     {
25         return 0;
26     }
27     if(memo[n]!=-1)
28     {
29         return memo[n];
30     }
31     int minAns=INT_MAX;
32 //   we need to check for each k distance from the current index
33     for(int i=1;i<=k;i++)
34     {
35         if(n-i>=0)
36         {
37             int ans=recursiveJump(input,memo,n-i,k)+abs(input[n]-input[n-i]);
38             minAns=min(minAns,ans);
39         }
40     }
41     return memo[n]= minAns;
42 }
43
44
45 int dpJump(vector<int>&input,int k)
46 {
47     int dp[input.size()];
48     dp[0]=0;
49     for(int i=1;i<input.size();i++)
50     {
51         dp[i]=INT_MAX;
52         for(int j=1;j<=k;j++)
53         {
54             if(i-j>=0)
55             {
56                 dp[i]=min(dp[i],dp[i-j]+abs(input[i]-input[i-j]));
57             }
58         }
59     }
60     return dp[input.size()-1];
61 }
62
63 int main()
64 {
65     int n;
66     cin>>n;
67     int k;
68     cin>>k;
69     vector<int>v(n,0);
70     for(int i=0;i<n;i++)
71     {
72         cin>>v[i];
73     }
74     vector<int>memo(n+1,-1);
75     int ans1= recursiveJump(v,memo,n-1,k);
76     int ans2= dpJump(v,k);
77     cout<<ans1<<" "<<ans2<<endl;
78 }
79 */
80
81 /*
82 In recursive + memoization
83
84 time complexity: O(n*k)
85 space complexity: O(n) for array + O(n) for recursion stack
86
87 In DP
88
89 time complexity: O(n*k)
90 space complexity: O(n)
91
92 */
93 */
```

5. MAXIMUM SUM OF NON-ADJACENT ELEMENT

$$A_{xx} = [2, 1, 4, 9] \quad n = 4$$

$$\text{Max}(2+9, 6) = 11$$



$\rightarrow f(1) \rightarrow$ This will give Answer from 0 to 1 index
Ans $[0 \dots 1] = 2$

$f(2) \rightarrow$ This will give Answer from 0 to 2th index
Ans $[0 \dots 2] = 6$

$f(3) \rightarrow$ This will give Answer from 0 to 3th index $[0 \dots 3] = 11$

$f(4) \rightarrow$ This will give Answer from 0 to 4th index $[0 \dots 4] = 17$

6/28/22, 11:16 AM

maximumSumOfNonAdjecentElement.cpp

```
/*
    Ways to write recursive code

    1. try to represent the problem in a terms of index

    2.Do all possible stuffs on that index and accordingly to the problem
statement

    3. if question asking COUNT ALL WAYS == > Sum of all the stuffs if question
asking MIN OR MAX ==
        > Then we will take max or
min from the calculated stuffs
*/
#include<bits/stdc++.h>
using namespace std;
int recursiveSum(vector<int> &input, vector<int> &memo, int n)
{
    if(n==0)
    {
        //n==0 , only reach if we have not taken n=1 element , if we have taken then this
will be -1 ;
        return input[0];
    }
    if(n<0)
    {
        return 0;
    }
    if(memo[n]!=-1)
    {
        return memo[n];
    }
    // taken
    int taken=0;
    int notTaken=0;

    taken= input[n]+recursiveSum(input,memo,n-2);
    notTaken= 0 + recursiveSum(input,memo,n-1);

    return memo[n]=max(taken,notTaken);
}
int countDp(vector<int>&input)
{
    int dp[input.size()];
    dp[0]=input[0];

    for(int i=1;i<input.size();i++)
    {
        dp[i]= max(dp[i-1],(i-2<0?0:dp[i-2])+input[i]);
    }
    return dp[input.size()-1];
}
int main()
{
    int n;
    cin>>n;
```

localhost:4649/?mode=clike

6/28/22, 11:16 AM

maximumSumOfNonAdjecentElement.cpp

```
vector<int>v(n,0);
for(int i=0;i<n;i++)
{
    cin>>v[i];
}
vector<int>memo(n+1,-1);

int ans1= recursiveSum(v,memo,n-1);
int ans2= countDp(v);

cout<<ans1<<" "<<ans2<<endl;
}

/*
In recursive + memoization
time complexity : O(n)
space complexity : O(n) for array + O(n) because of recursion stack

In dp
time complexity : O(n)
space complexity : O(n)
*/

```

1/2

localhost:4649/?mode=clike

2/2

6. House Robber

→ In this problem you can not Select 2 adjacent element and array will be in Circular Manner

like

4

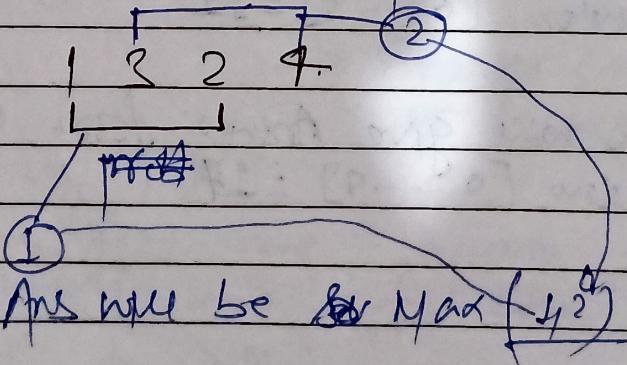
~~1 3 2 4~~ ↴ 3 2 4 | Circular Array

find the Maximum sum without selecting
Adjacent element

→ Obs

1st element and Last element can not participate in solution because they were adjacent

Now Break the problem into 2 problem



Ans will be ~~Max~~ $\text{Max}(\text{1}^1)$

→ Apply the last question approach and you will get the answer

6/28/22, 11:21 AM

HouseRobber2.cpp

```

1 /*
2 question:
3
4 Mr. X is a skilled burglar who intends to loot houses along a street. Each home has
5 a specific amount of money tucked away. All of the residences in this area are
6 grouped in a circle. That is, the first home is next to the final one. Meanwhile,
7 nearby houses are connected to a security system, which will immediately inform the
8 police if two adjacent houses are broken into on the same night.
9
10 Given an integer array nums indicating the amount of money in each house. Return the
11 greatest amount of money Mr. X can rob without notifying the cops.
12
13 Example:
14 Input:
15 [2, 3, 2]
16 Output:
17 3
18 */
19
20 /*
21 Ways to write recursive code
22
23     1. try to represent the problem in a terms of index
24
25     2. Do all possible stuffs on that index and accordingly to the problem
26 statement
27
28     3. if question asking COUNT ALL WAYS == > Sum of all the stuffs if question
29 asking MIN OR MAX ==
30         > Then we will take max or
31         min from the calculated stuffs
32 */
33 #include<bits/stdc++.h>
34 using namespace std;
35
36 int recursiveRobber(vector<int>&input,int n,vector<int>&memo)
37 {
38     if(n==0)
39     {
40         return input[0];
41     }
42     if(n<0)
43     {
44         return 0;
45     }
46     if(memo[n]!=-1)
47     {
48         return memo[n];
49     }
50     int taken=0;
51     int notTaken=0;
52     taken= input[n]+recursiveRobber(input,n-2,memo);
53     notTaken= 0 + recursiveRobber(input,n-1,memo);
54     return memo[n]=max(taken,notTaken);
55 }
```

6/28/22, 11:21 AM

HouseRobber2.cpp

```

53 }
54
55 int dpRobber(vector<int>&input)
56 {
57     int dp[input.size()];
58     dp[0]=input[0];
59     for(int i=1;i<input.size();i++)
60     {
61         dp[i]= max(dp[i-1],(i-2<0?0:dp[i-2])+input[i]);
62     }
63     return dp[input.size()-1];
64 }
65
66 int main()
67 {
68     int n;
69     cin>>n;
70
71     vector<int>v(n,0);
72     vector<int>input1;
73     vector<int>input2;
74     // due to circular nature , 1st and last can not include in the answer tharts why
75     // need to call the function by taking 1st element and not taking last elmenet and vice
76     // versa
77     for(int i=0;i<n;i++)
78     {
79         cin>>v[i];
80         if(i!=0)
81         {
82             input1.push_back(v[i]);
83         }
84         if(i!=n-1)
85         {
86             input2.push_back(v[i]);
87         }
88     }
89     vector<int>memo(n+1,-1);
90     vector<int>memo1(n+1,-1);
91     int
92     ans1=max(recursiveRobber(input1,input1.size()-1,memo),recursiveRobber(input2,input2.
93     size()-1,memo1));
94     int ans2=max(dpRobber(input1),dpRobber(input2));
95
96     cout<<ans1<<" "<<ans2<<endl;
97 }
98 /*
99 In recusive + memoization
100
101 time complexity: O(n)
102 space complexity: O(n) for array + O(n) because of recursion stack
103
104 In dp
105 time complexity: O(n);
106 space complexity: O(n);
107 */

```

7. Ninja TRAINING

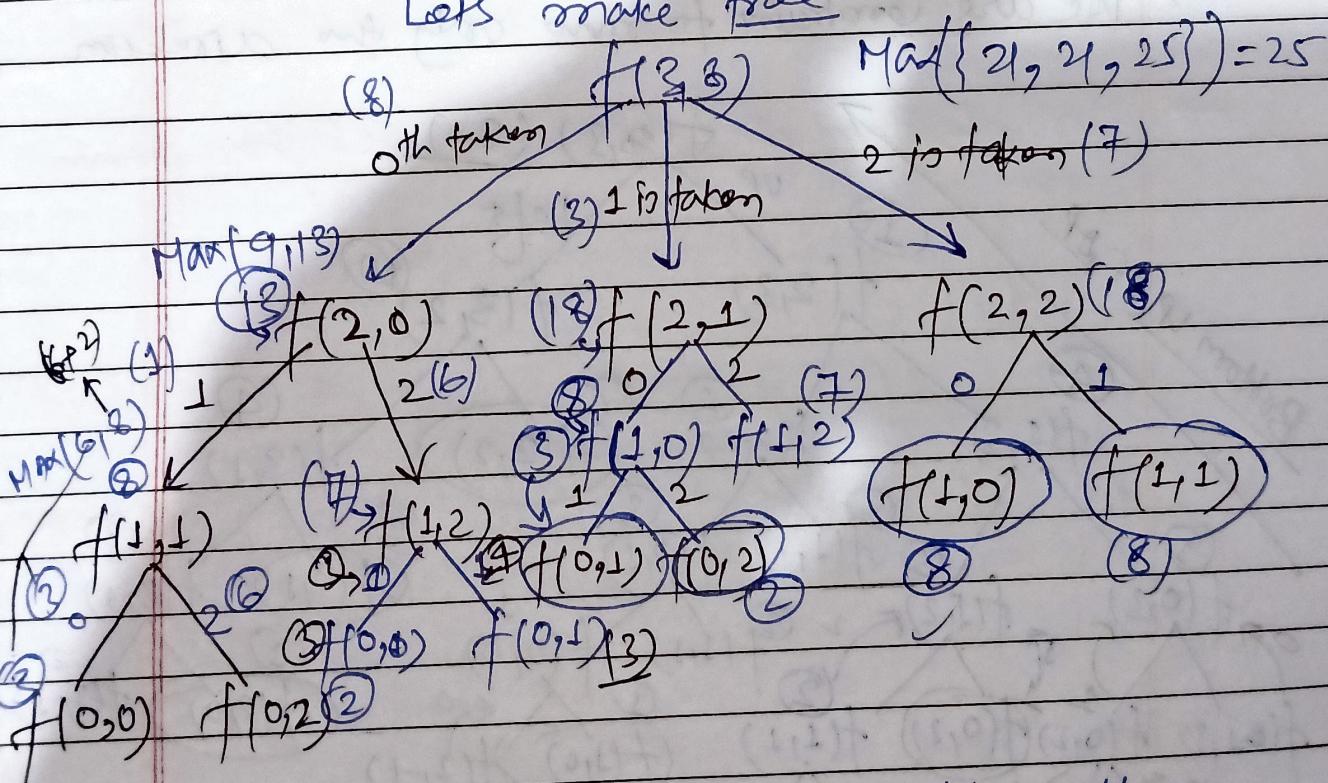
Input

$$n=9$$

do	2	1	3
di	-3	4	6
d2	10	1	6
d3	8	3	7

→ question description written in Comment section.

Let's make tree



→ There is overlapping of subproblem, then we need to do memoization

→ What is $f(2, 0)$

You have to calculate ~~for 2 in~~ from 0 to 279 index provided for 309 index we have slept off items.

6/28/22, 11:24 AM

NinjaTraining.cpp

```
1 /*
2 question:
3 Ninja is planning this 'N' days-long training schedule. Each day, he can perform any
4 one of these three activities. (Running, Fighting Practice or Learning New Moves).
5 Each activity has some merit points on each day. As Ninja has to improve all his
6 skills, he can't do the same activity in two consecutive days. Can you help Ninja
7 find out the maximum merit points Ninja can earn?
8
9 You are given a 2D array of size N*3 'POINTS' with the points corresponding to each
10 day and activity. Your task is to calculate the maximum number of merit points that
11 Ninja can earn.
12 input:
13 3
14 10 40 70
15 20 50 80
16 30 60 90
17 output=210
18
19
20 int recursiveTraining(vector<vector<int>>&input,int n,vector<vector<int>>&memo,int k)
21 {
22     if(n==0)
23     {
24         int maxe=0;
25         for(int i=0;i<3;i++)
26         {
27             if(k!=i)
28             {
29                 maxe=max(maxe,input[n][i]);
30             }
31         }
32
33         return maxe;
34     }
35
36     if(memo[n][k]!=-1)
37     {
38         return memo[n][k];
39     }
40     for(int i=0;i<3;i++)
41     {
42         // we can not take previously taken activity
43         if(k!=i)
44         {
45             memo[n][k]=max(memo[n][k],input[n][i]+recursiveTraining(input,n-
46             1,memo,i));
47         }
48     }
49     return memo[n][k];
50 }
51 int dpTraining(vector<vector<int>>&input)
52 {
```

localhost:4649/?mode=clike

6/28/22, 11:24 AM

NinjaTraining.cpp

```
53     int n=input.size();
54     vector<vector<int>>dp(n,vector<int>(4,0));
55
56     dp[0][0]=max(input[0][1],input[0][2]);
57     dp[0][1]=max(input[0][0],input[0][2]);
58     dp[0][2]=max(input[0][0],input[0][1]);
59     dp[0][3]=max({input[0][0],input[0][1],input[0][2]});
60
61     for(int i=1;i<n;i++)
62     {
63         for(int last=0;last<4;last++)
64         {
65             for(int task=0;task<3;task++)
66             {
67                 if(last!=task)
68                 {
69                     dp[i][last]=max(dp[i][last],input[i][task]+dp[i-1][task]);
70                 }
71             }
72         }
73     }
74
75     return dp[n-1][3];
76 }
77 int main()
78 {
79     int n;
80     cin>>n;
81     vector<vector<int>> v(n,vector<int>(3));
82     for(int i=0;i<n;i++)
83     {
84         for(int j=0;j<3;j++)
85         {
86             cin>>v[i][j];
87         }
88     }
89
90     vector<vector<int>> memo(n,vector<int>(4,-1));
91
92     int ans1= recursiveTraining(v,n-1,memo,3);
93     int ans2= dpTraining(v);
94     cout<<ans1<<" "<<ans2<<endl;
95
96 }
```

1/2

localhost:4649/?mode=clike

2/2

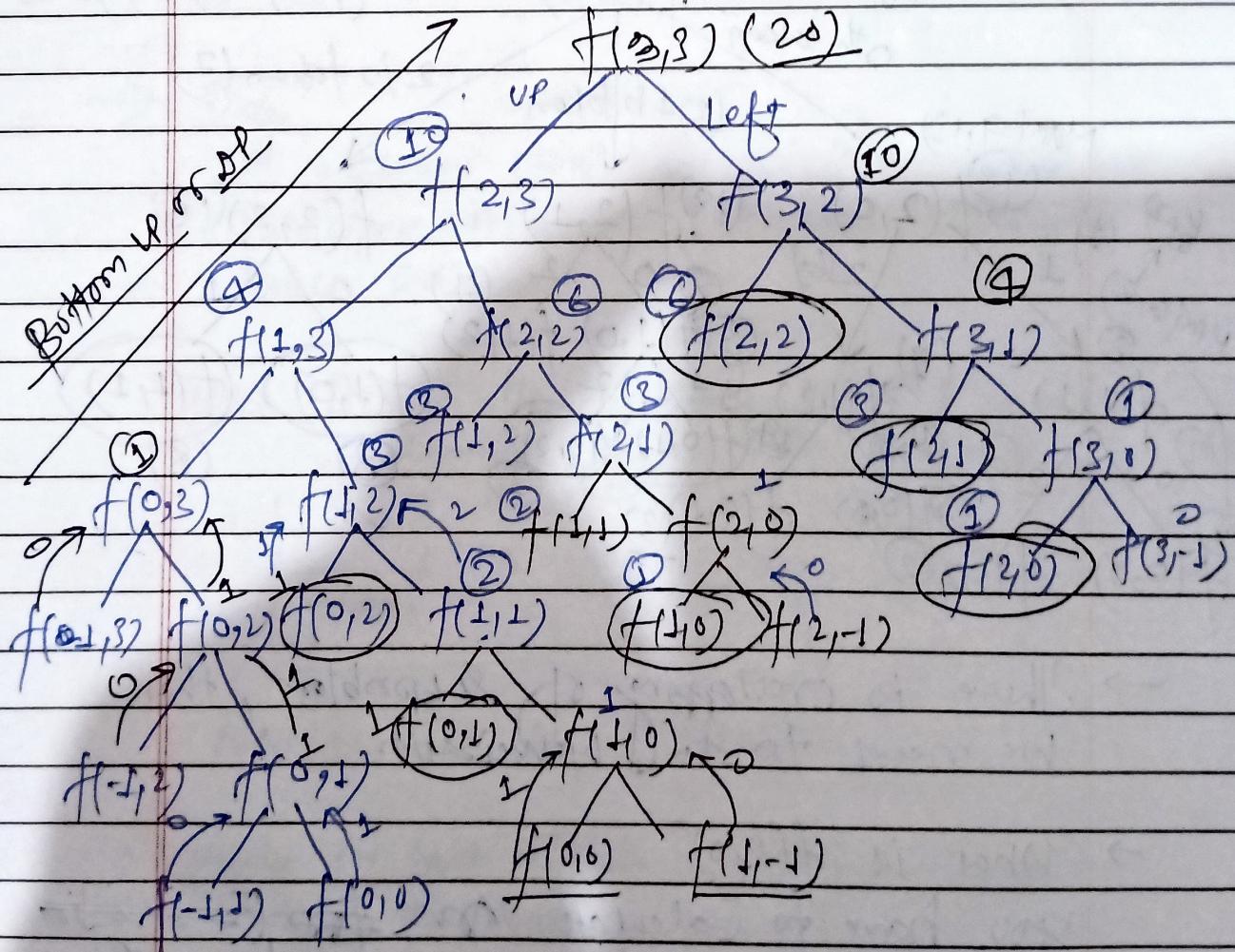
8. DP on Grid / 2D Matrix

→ Memoization to DP Conversion

- i) Declare the Base Case
- ii) Express all states in for loops
- iii) Copy the recurrence relation.

lets make recursive tree for $n=3, m_3$

→ We are constraint to move only two direction

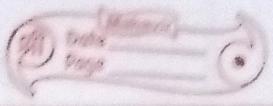


Circle are overlapping of Δ transformation.

If $n=0$ and $m=0$ then return will be 1 because only position are there to move.

→ Here overlapping come in the scene than memoization will take place followed by bottom-up (dp).

9. Unique path 2.



This is slightly different from problem 2.

For this case we need to keep one extra base case of dead cell.

→ In recursion you can put

if ($\text{arrG}[j] == -1$) } Dead cell Condition
return 0 ; }

→ In bottom-up approach we can put

if ($\text{arrG}[j] == -1$)
 $\text{dp}[i][j] = 0;$

And time and space complexity going to same.

```

1 /*
2 Given a 'N' * 'M' maze with obstacles, count and return the number of unique paths
3 to reach the right-bottom cell from the top-left cell. A cell in the given maze has
4 a value '-1' if it is a blockage or dead-end, else 0. From a given cell, we are
5 allowed to move to cells (i+1, j) and (i, j+1) only. Since the answer can be large,
6 print it modulo 10^9 + 7.
7
8 For Example :
9 Consider the maze below :
10
11 0 0 0
12 0 -1 0
13 0 0 0
14
15 There are two ways to reach the bottom left corner -
16
17 (1, 1) -> (1, 2) -> (1, 3) -> (2, 3) -> (3, 3)
18 (1, 1) -> (2, 1) -> (3, 1) -> (3, 2) -> (3, 3)
19
20 Hence the answer for the above test case is 2.
21 */
22 #include <bits/stdc++.h>
23 using namespace std;
24
25 // top down approach
26 int recursiveCount(int m, int n, vector<vector<int>>&input, vector<vector<int>>&memo)
27 {
28     if(n>=0 and m>=0 and input[n][m]==-1)
29     {
30         return 0;
31     }
32     if (n == 0 and m == 0)
33     {
34         return 1;
35     }
36     if (n < 0 or m < 0)
37     {
38         return 0;
39     }
40
41     if (memo[n][m] != -1)
42     {
43         return memo[n][m];
44     }
45     int upSideCall = recursiveCount(m, n - 1, input, memo);
46     int leftSideCall = recursiveCount(m - 1, n, input, memo);
47
48     memo[n][m] = upSideCall + leftSideCall;
49
50 /* for bottom up approach
51 1. Declare the base case
52 2. express all the states(changing variable) in for loops
53 3. copy the recurrence relation
54 */

```

```

55 int dpCount(int m, int n, vector<vector<int>>&input)
56 {
57     vector<vector<int>> dp(n + 1, vector<int>(m + 1, 0));
58     // dp[0][0]=1;
59
60     for (int i = 0; i <=n; i++)
61     {
62         for (int j = 0; j <= m; j++)
63         {
64             if (input[i][j]==-1)
65             {
66                 dp[i][j] = 0;
67             }
68             else if(i==0 and j==0)
69             {
70                 dp[i][j]=1;
71             }
72             else
73             {
74                 dp[i][j] = (j - 1 >= 0 ? dp[i][j - 1] : 0) + (i - 1 >= 0 ? dp[i - 1]
75 [j] : 0);
76             }
77         }
78     }
79     return dp[n][m];
80 }
81
82 int main()
83 {
84     int m, n;
85     cin >> m >> n;
86     vector<vector<int>>input(n, vector<int>(m,0));
87     for(int i=0;i<n;i++)
88     {
89         for(int j=0;j<m;j++)
90         {
91             cin>>input[i][j];
92         }
93     }
94     vector<vector<int>> memo(n + 1, vector<int>(m + 1, -1));
95
96     int ans1 = recursiveCount(m - 1, n - 1, input, memo);
97     int ans2 = dpCount(m - 1, n - 1, input);
98
99     cout << ans1 << " " << ans2 << endl;
100    // cout<<ans1<<endl;
101 }
102 /*
103 In recursive + memoization
104 time complexity: O(mn)
105 space complexity: O(mn) for memoization and O(n+m) recursion stack
106
107 In dp
108 time complexity: O(mn)
109 space complexity: O(mn)
110 */

```

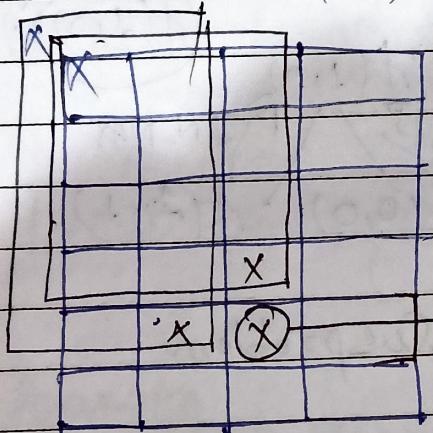
10. Minimum Path Sum

→ For writing recurrence we need to face.

Care of three things

- i) Explore (i,j)
- ii) Explore all path
- iii) face the power part

$f(m-1, m-1) \rightarrow$ this will give last from (0,0) to $(\underline{m-1}, \underline{m-1})$.

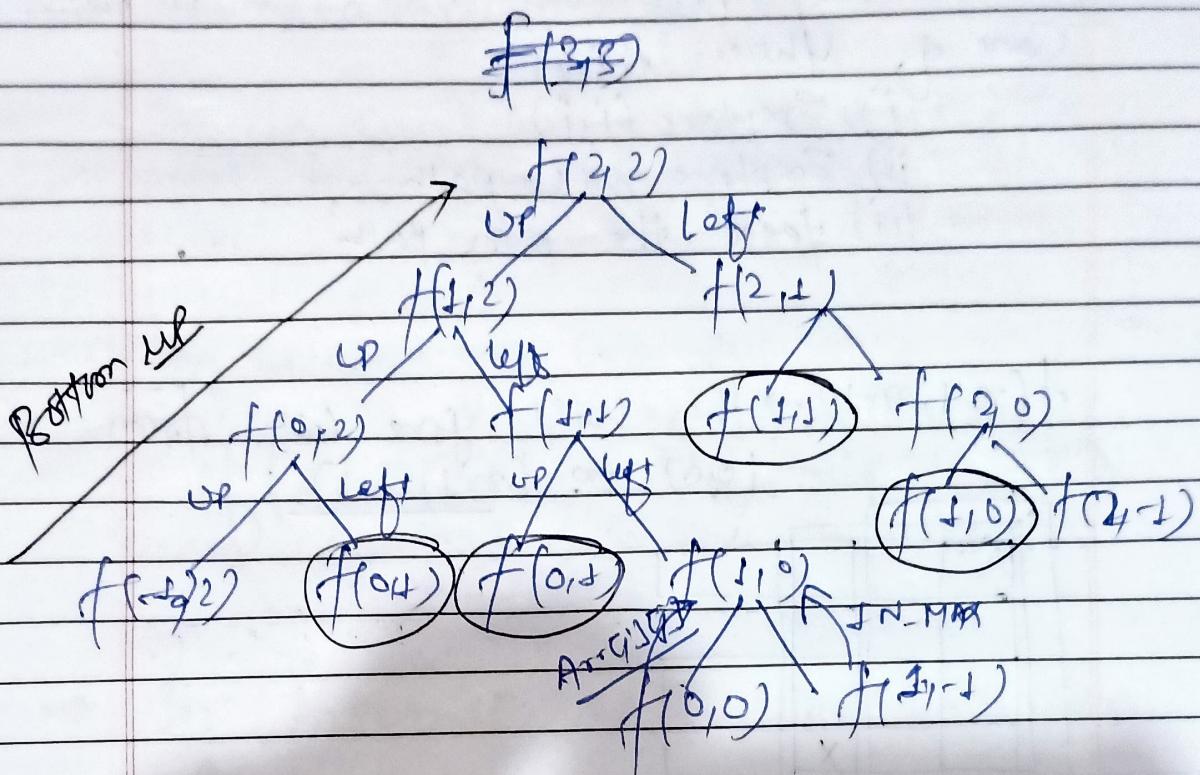


→ if you are calculating
for this
Value at point \oplus from (both sides)

→ This problem less bit similar to previous one..

→ Time and Space Complexity ~~can remain~~ same as previous one.

Recursive Tree visualisation



○ overlapping of subproblems

→ from both branches we have option to choose min(left, right) and add the right value.

6/28/22, 11:31 AM

MinimumPathSum.cpp

```
1 /*
2 Given a m x n grid filled with non-negative numbers, find a path from top left to
3 bottom right, which minimizes the sum of all numbers along its path.
4
5 Note: You can only move either down or right at any point in time.
6 Example 1:
7
8 Input: grid = [[1,3,1],[1,5,1],[4,2,1]]
9 Output: 7
10 Explanation: Because the path 1 → 3 → 1 → 1 → 1 minimizes the sum.
11 */
12 #include <bits/stdc++.h>
13 using namespace std;
14
15 int recursiveMinimuCost(vector<vector<int>> &input, int n, int m, vector<vector<int>>
16 &memo)
17 {
18     if (n == 0 and m == 0)
19     {
20         return input[n][m];
21     }
22     if (n < 0 or m < 0)
23     {
24         return 1e9;
25     }
26     if (memo[n][m] != -1)
27     {
28         return memo[n][m];
29     }
30     int left = recursiveMinimuCost(input, n, m - 1, memo);
31     int up = recursiveMinimuCost(input, n - 1, m, memo);
32     memo[n][m] = min(left, up) + input[n][m];
33     return memo[n][m];
34 }
35
36 int dpMinimumCost(vector<vector<int>> &input)
37 {
38     int n = input.size();
39     int m = input[0].size();
40     vector<vector<int>> dp(n, vector<int>(m, 0));
41     dp[0][0] = input[0][0];
42     for (int i = 0; i < n; i++)
43     {
44         for (int j = 0; j < m; j++)
45         {
46             if(i==0 and j==0)
47             {
48                 dp[i][j]=input[i][j];
49             }
50             else
51             {
52                 dp[i][j] = min((i - 1) >= 0 ? dp[i - 1][j] : 1e9, (j - 1) >= 0 ? dp[i]
53 [j - 1] : 1e9) + input[i][j];
54             }
55         }
56     }
57 }
```

localhost:4649/?mode=clike

6/28/22, 11:31 AM

MinimumPathSum.cpp

```
57     return dp[n - 1][m - 1];
58 }
59
60 int main()
61 {
62     int n, m;
63     cin >> n >> m;
64     vector<vector<int>> input(n, vector<int>(m, 0));
65     for (int i = 0; i < n; i++)
66     {
67         for (int j = 0; j < m; j++)
68         {
69             cin >> input[i][j];
70         }
71     }
72     vector<vector<int>> memo(n, vector<int>(m, -1));
73     cout << recursiveMinimuCost(input, n - 1, m - 1, memo) << endl;
74     cout << dpMinimumCost(input) << endl;
75 }
76
77 /*
78 * In recursive + memoization
79
80 time complexity: O(mn)
81 space complexity: O(mn) for memoization and O(n+m) recursion stack
82
83 In dp
84
85 time complexity: O(mn)
86 space complexity: O(mn)
87
88 */
89 */
```

1/2

localhost:4649/?mode=clike

2/2

11. Triangle

Starting point

①

2 3

3 6 7

8 9 6 10

Move



Not fixed ending point here.

→ find the minimum path from top to bottom.

→ Tabulation is opposite of recursion

if recursion starts from (0,0) then tabulation
will start from (m+1, m). just opposite

Recursion

1 → Starting point for recursion

2 3

3 6 7

8 9 6 10



Starting point for tabulation because it is
totally opposite of recursion.

You can make this problem
easy &

6/28/22, 11:52 AM

Traingle.cpp

```
/*
2 Problem Statement
3 You are given a triangular array/list 'TRIANGLE'. Your task is to return the minimum
path sum to reach from the top to the bottom row.
4 The triangle array will have N rows and the i-th row, where 0 <= i < N will have i +
1 elements.
5 You can move only to the adjacent number of row below each step. For example, if you
are at index j in row i, then you can move to i or i + 1 index in row j + 1 in each
step.
6 For Example :
7 If the array given is 'TRIANGLE' = [[1], [2,3], [3,6,7], [8,9,6,1]] the triangle
array will look like:
8
9 1
10 2,3
11 3,6,7
12 8,9,6,10
13
14 For the given triangle array the minimum sum path would be 1->2->3->8. Hence the
answer would be 14.
15 */
16 #include <bits/stdc++.h>
17 using namespace std;
18
19 /*
20 In this question , in recursive we will start from top to bottom
21 */
22
23 int recursiveTraingle(vector<vector<int>> &input, int row, int col,
vector<vector<int>> &memo)
{
    if (col > row)
    {
        return 1e9;
    }
    if (row == input.size() - 1)
    {
        return input[row][col];
    }

    if (memo[row][col] != -1)
    {
        return memo[row][col];
    }

    int down = recursiveTraingle(input, row + 1, col, memo);
    int diag = recursiveTraingle(input, row + 1, col + 1, memo);
    memo[row][col] = min(down, diag) + input[row][col];
    return memo[row][col];
}

int dpTraingle(vector<vector<int>> &input)
{
    int n = input.size();

    /*
    As this is opposite of recursion then this will start from bottom to top

    Now we have to check States in Dp because of variable ending point .

```

localhost:4649/?mode=clike

6/28/22, 11:52 AM

Traingle.cpp

```
for  i=3 then possible j= 0 1 2 3
for i=2 then possible j= 0 1 2
for i=1 then possible j= 0 1
for i=0 then possible j= 0
    for different i or row we have different states of j .
*/
//    now write base case from bottom
vector<vector<int>> dp(n, vector<int>(n, 0));
for (int i = 0; i < n; i++)
{
    dp[n - 1][i] = input[n - 1][i];
}
for (int i = n - 2; i >= 0; i--)
{
    for (int j = 0; j <= i; j++)
    {
        int down = dp[i + 1][j];
        int diag = dp[i + 1][j + 1];
        dp[i][j] = min(down, diag) + input[i][j];
    }
}
return dp[0][0];
}

int main()
{
    int n;
    cin >> n;
    vector<vector<int>> input(n);
    for (int i = 0; i < n; i++)
    {
        input[i].resize(i + 1);
        for (int j = 0; j <= i; j++)
        {
            cin >> input[i][j];
        }
    }
    vector<vector<int>> memo(n, vector<int>(n, -1));

    int ans1 = recursiveTraingle(input, 0, 0, memo);
    int ans2 = dpTraingle(input);
    cout << ans1 << " " << ans2 << endl;
}
```

1/2

localhost:4649/?mode=clike

2/2

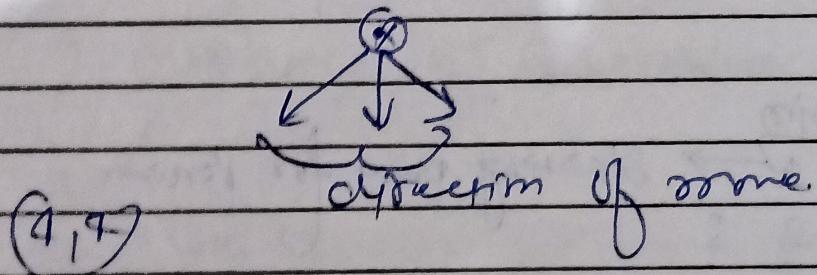
12. Max/Min Falling Sum.

→ Max Path Sum

From any cell → 1st Row }
to
any cell → Last Row }

→ In this question starting and ending point are not fixed - we can start any cell from 1st row and similar if end at cell in last row.

For this we can come in three ways



(1,1)

→ 1 2 10 4
10 3 2 1
1 1 20 2

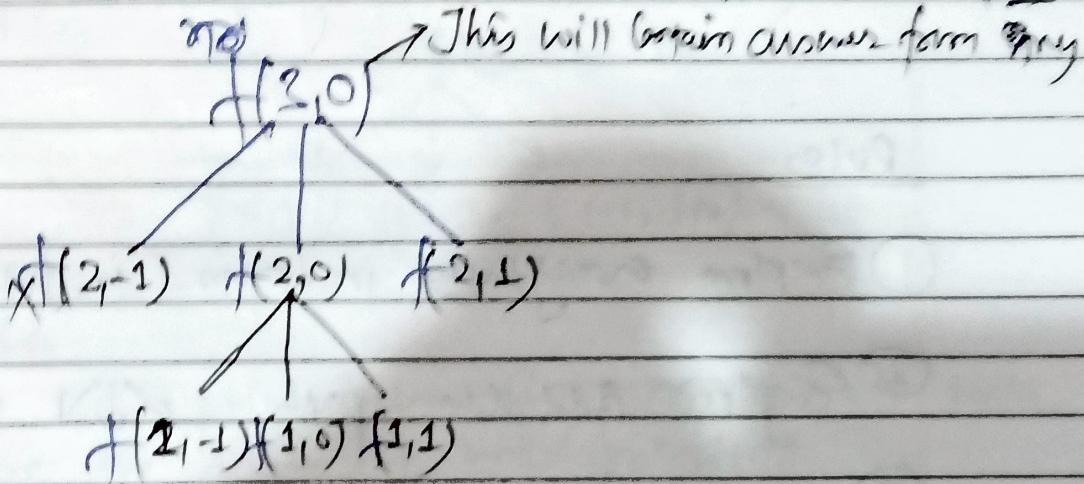
(n-1, m) 1 2 2 1
x x x x }

Any cell can be starting point.

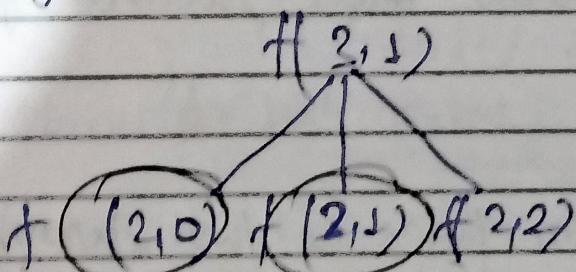
So we need to explore all path from last start column.

Let's see overlapping of subproblem.

→ If we start from $f(3,0)$ return form it on to $f(3,0)$.



→ If we start from $f(3,1)$.



overlapping form calling it $f(3,0)$.
That's why memoization will come into game.

$f(3,1) \rightarrow$ It will contain answer, starting from
any column form left to ending at
 $f(2,1)$ in last row.

6/28/22, 12:01 PM

FallingPathSum.cpp

```

1 /*
2 You have been given an N*M matrix filled with integer numbers, find the maximum sum
3 that can be obtained from a path starting from any cell in the first row to any cell
4 in the last row.
5 */
6
7 #include <bits/stdc++.h>
8 using namespace std;
9
10 int recursiveFallsumUTIL(vector<vector<int>> &mat, int n, int m, vector<vector<int>>
11 &memo)
12 {
13     if (m < 0 or m >= mat[0].size())
14     {
15         return -1e8;
16     }
17     if (n == 0)
18     {
19         return mat[0][m];
20     }
21     if (memo[n][m] != -1)
22     {
23         return memo[n][m];
24     }
25     // we have to move upper , upper right diagonal , upper left diagonal
26     int up, up_right, up_left;
27     up = up_right = up_left = INT_MIN;
28
29     up = recursiveFallsumUTIL(mat, n - 1, m, memo) + mat[n][m];
30
31     up_right = recursiveFallsumUTIL(mat, n - 1, m + 1, memo) + mat[n][m];
32
33     up_left = recursiveFallsumUTIL(mat, n - 1, m - 1, memo) + mat[n][m];
34
35     memo[n][m] = max(max(up, up_right), up_left);
36 }
37 /*
38 In this recursive approach i need to traverse every col in last row for getting
39 answer.
40 In this question starting or ending point not fixed.
41 */
42 int recursiveFallingSum(vector<vector<int>> &input, int n, int m,
43 vector<vector<int>> &memo)
44 {
45     int ans = INT_MIN;
46
47     for (int i = 0; i <= m; i++)
48     {
49         ans = max(ans, recursiveFallsumUTIL(input, n, i, memo));
50     }
51     return ans;
52 }
```

localhost:4649/?mode=clike

1/3

FallingPathSum.cpp

```

6/28/22, 12:01 PM
53 int dpFallingSum(vector<vector<int>>&input)
54 {
55     int n = input.size();
56     int m = input[0].size();
57     vector<vector<int>> dp(n, vector<int>(m, 0));
58
59     /*
60     Since tabulations completely opposite of recursion , here we start from starting
61
62     lets think about states in the dp
63
64     for i=0 what would be possible of j =0, 1,2.....m;
65     for i=1 what would be possible of j =0, 1,2.....m;
66     for i=2 what would be possible of j =0, 1,2.....m;
67     for i=3 what would be possible of j =0, 1,2.....m;
68
69     */
70     // base case ==> destination + outofbound;
71
72     for(int j=0;j<m;j++)
73     {
74         dp[0][j]=input[0][j];
75     }
76     for(int i=1;i<n;i++)
77     {
78         for(int j=0;j<m;j++)
79         {
80             int up, up_right, up_left;
81             up = up_right = up_left = INT_MIN;
82             up = dp[i-1][j] + input[i][j];
83             if(j+1<m)
84             {
85                 up_right = dp[i-1][j+1] + input[i][j];
86             }
87             if(j-1>=0)
88             {
89                 up_left = dp[i-1][j-1] + input[i][j];
90             }
91             dp[i][j] = max(max(up,up_right),up_left);
92         }
93     }
94     int ans=INT_MIN;
95     for(int i=0;i<m;i++)
96     {
97         ans=max(ans,dp[n-1][i]);
98     }
99     return ans;
100 }
101
102 int main()
103 {
104     int n, m;
105     cin >> n >> m;
106     vector<vector<int>> mat(n, vector<int>(m));
107     for (int i = 0; i < n; i++)
108     {
109         for (int j = 0; j < m; j++)
110         {
111             cin >> mat[i][j];
112         }
113     }
114 }
115 vector<vector<int>> memo(n, vector<int>(m, -1));
116
117 int ans1 = recursiveFallingSum(mat, n - 1, m - 1, memo);
118 int ans2= dpFallingSum(mat);
119 cout<<ans1<< " "<<ans2<<endl;
120 // cout << ans1 << endl;
121 }
122
123 /*
124 In recursive + memoization
125
126     time complexity = O(n*m)
127     space complexity = O(n*m)for memoization array + O(n) for recursion stack
128
129
130 In dp
131
132     time complexity = O(n*m)
133     space complexity = O(n*m)
134
135 */
136 */


```

localhost:4649/?mode=clike

2/3

6/28/22, 12:01 PM

FallingPathSum.cpp

```

113 }
114 }
115 vector<vector<int>> memo(n, vector<int>(m, -1));
116
117 int ans1 = recursiveFallingSum(mat, n - 1, m - 1, memo);
118 int ans2= dpFallingSum(mat);
119 cout<<ans1<< " "<<ans2<<endl;
120 // cout << ans1 << endl;
121 }
122
123 /*
124 In recursive + memoization
125
126     time complexity = O(n*m)
127     space complexity = O(n*m)for memoization array + O(n) for recursion stack
128
129
130 In dp
131
132     time complexity = O(n*m)
133     space complexity = O(n*m)
134
135 */
136 */


```

localhost:4649/?mode=clike

3/3

13. Cherry picker II 3D DP

- Fixed starting point
- Variable ending point

Rules

- ① Explore everything in form $(i_1, i_2, \Delta(i_2, j_2))$
- ② Explore all the path $\leftarrow \downarrow \uparrow$
- ③ Max sum

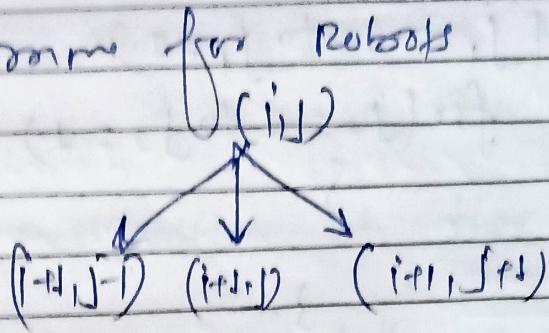
If we have only one starting point we should start recursion from fixed point.

→ In this question there is two start points

Robot 1	→ 3	1	1 4	Robot 2
	2	5	1	
	1	5	5	
	2	1	1	

→ If Robot 1 Coordinate will be (i_1, j_1)
and if Robot 2 Coordinate will be (i_2, j_2)

→ Possible moves for Robots



Ques

At every point value of i or row no will be the same for both robots?

initially $f(i_1, i_2, j_2)$

\downarrow now
 $f(i, j_1, j_2)$ because now no will

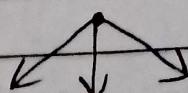
same for both robot.

→ from one index one robot can move 3 steps
in next row and same for other

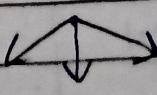
(Robot 1 moves)



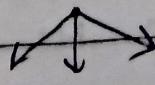
(Robot 2 moves) Moves



3



3



9 moves

All coordinate of these ~~is~~ 9 moves,
we will get. Use this

$\text{for } (j_1 = -1 \text{ to } j_1 = 1)$

$\text{for } (j_2 = -1 \text{ to } j_2 = 1)$

3

Rare Case

- ① Out of Bound error
 - ② If $j_1 = j_2$ then pick one.
 - ③ $j_1 \neq j_2$ then return $\min(j_1, j_2)$, $\max(j_1, j_2)$

for only recognition

$$T_C = \underline{(3^n \times 3^n)}$$

Aff2 Memorization

$$T_C = O(N \times M \times n) \times q$$

Total money.

→ for States in tabulation

for $i = m-1$ | j_1 and j_2 can be anywhere
 in last row in the $[0, m-1]$
 that's why we are using selected strip.

6/28/22, 12:06 PM

CherryPickup.cpp

```

1 /*
2 You are given a rows x cols matrix grid representing a field of cherries where
3 grid[i][j] represents the number of cherries that you can collect from the (i, j)
4 cell.
5
6 Robot #1 is located at the top-left corner (0, 0), and
7 Robot #2 is located at the top-right corner (0, cols - 1).
8 Return the maximum number of cherries collection using both robots by following the
9 rules below:
10
11 From a cell (i, j), robots can move to cell (i + 1, j - 1), (i + 1, j), or (i + 1, j
12 + 1).
13 When any robot passes through a cell, It picks up all cherries, and the cell becomes
14 an empty cell.
15 When both robots stay in the same cell, only one takes the cherries.
16 Both robots cannot move outside of the grid at any moment.
17 Both robots should reach the bottom row in grid.
18 */
19 #include <bits/stdc++.h>
20 using namespace std;
21
22 int recursiveCherryPickup(vector<vector<int>> &input, int i1, int j1, int j2,
23 vector<vector<vector<int>> &memo)
24 {
25     if (i1 < 0 || i1 >= input.size() || j1 < 0 || j1 >= input[0].size() || j2 < 0 ||
26     j2 >= input[0].size())
27     {
28         return -1e8;
29     }
30     if (i1 == input.size() - 1)
31     {
32         if (j1 == j2)
33         {
34             return input[i1][j1];
35         }
36         else
37         {
38             return (input[i1][j1] + input[i1][j2]);
39         }
40     }
41     if (memo[i1][j1][j2] != -1)
42     {
43         return memo[i1][j1][j2];
44     }
45     int ans = INT_MIN;
46     for (int jj1 = -1; jj1 <= 1; jj1++)
47     {
48         for (int jj2 = -1; jj2 <= 1; jj2++)
49         {
50             int newj1 = j1 + jj1;
51             int newj2 = j2 + jj2;
52             int temp = 0;
53             if (j1 == j2)
54             {
55                 temp = recursiveCherryPickup(input, i1 + 1, newj1, newj2, memo) +
56                 input[i1][j1];
57             }
58             else
59             {
60                 temp = dpCherryPickup(input, i1 + 1, newj1, newj2, memo) +
61                 input[i1][j1];
62             }
63             ans = max(ans, temp);
64         }
65         dp[i1][j1][j2] = ans;
66     }
67     memo[i1][j1][j2] = ans;
68 }
69
70 int result = 0;
71 for (int j1 = 0; j1 < m; j1++)
72 {
73     for (int j2 = 0; j2 < m; j2++)
74     {
75         result = max(result, dp[0][j1][j2]);
76     }
77 }
78 return result;
79
80 int main()
81 {
82     int n, m;
83     cin >> n >> m;
84
85     vector<vector<int>> input(n, vector<int>(m));
86     for (int i = 0; i < n; i++)
87     {
88         for (int j = 0; j < m; j++)
89         {
90             cin >> input[i][j];
91         }
92     }
93     vector<vector<vector<int>>> memo(n, vector<vector<vector<int>>(m, vector<vector<int>>(m, -1)));
94
95     int ans1 = recursiveCherryPickup(input, 0, 0, m - 1, memo);
96     cout << ans1 << endl;
97     int ans2 = dpCherryPickup(input);
98     cout << ans2 << endl;
99 }

```

localhost:4649/?mode=clike

CherryPickup.cpp

1/3

6/28/22, 12:06 PM

```

52     }
53     else
54     {
55         temp = recursiveCherryPickup(input, i1 + 1, newj1, newj2, memo) +
56         input[i1][j1] + input[i1][j2];
57     }
58     ans = max(ans, temp);
59 }
60 memo[i1][j1][j2] = ans;
61 return ans;
62 }
63 bool isValid(int j1, int j2, int n, int m)
64 {
65     if (j1 < 0 || j1 >= m || j2 < 0 || j2 >= m)
66     {
67         return false;
68     }
69     return true;
70 }
71
72 int dpCherryPickup(vector<vector<vector<int>> &input)
73 {
74     int n = input.size();
75     int m = input[0].size();
76     vector<vector<vector<int>> dp(n, vector<vector<int>>(m, vector<int>(m, -1)));
77
78 // first we need to write base , it will be motivated from recursion base case
79 /*
80 states like for i==n-1 then possible value of j1 and j2 may be they both point
81 to same point or they may be on different point in same row or either they exchange
82 their positions.
83 */
84 for (int j1 = 0; j1 < m; j1++)
85 {
86     for (int j2 = 0; j2 < m; j2++)
87     {
88         if (j1 == j2)
89         {
90             dp[n - 1][j1][j2] = input[n - 1][j1];
91         }
92         else
93         {
94             dp[n - 1][j1][j2] = input[n - 1][j1] + input[n - 1][j2];
95         }
96     }
97 // now total three different states are there i,j1,j2 then three loops will be
98 for (int i = n - 2; i >= 0; i--)
99 {
100     for (int j1 = 0; j1 < m; j1++)
101     {
102         for (int j2 = 0; j2 < m; j2++)
103         {
104             int ans = INT_MIN;
105             for (int jj1 = -1; jj1 <= 1; jj1++)
106             {
107                 for (int jj2 = -1; jj2 <= 1; jj2++)
108                 {
109                     int newj1 = j1 + jj1;
110                     int newj2 = j2 + jj2;
111                     if (!isValid(newj1, newj2, n, m))
112                     {
113                         continue;
114                     }
115                     int temp = 0;
116                     if (j1 == j2)
117                     {
118
119                         temp = dp[i + 1][newj1][newj2] + input[i][j1];
120                     }
121                     else
122                     {
123
124                         temp = dp[i + 1][newj1][newj2] + input[i][j1] + input[i]
125 [j2];
126                     }
127
128                     ans = max(ans, temp);
129                 }
130             }
131             dp[i][j1][j2] = ans;
132         }
133     }
134 }
135
136 int result = 0;
137 for (int j1 = 0; j1 < m; j1++)
138 {
139     for (int j2 = 0; j2 < m; j2++)
140     {
141         result = max(result, dp[0][j1][j2]);
142     }
143 }
144 return result;
145
146 int main()
147 {
148     int n, m;
149     cin >> n >> m;
150
151     vector<vector<int>> input(n, vector<int>(m));
152     for (int i = 0; i < n; i++)
153     {
154         for (int j = 0; j < m; j++)
155         {
156             cin >> input[i][j];
157         }
158     }
159     vector<vector<vector<int>>> memo(n, vector<vector<vector<int>>(m, vector<vector<int>>(m, -1)));
160
161     int ans1 = recursiveCherryPickup(input, 0, 0, m - 1, memo);
162     cout << ans1 << endl;
163     int ans2 = dpCherryPickup(input);
164     cout << ans2 << endl;
165 }

```

localhost:4649/?mode=clike

2/3

localhost:4649/?mode=clike

3/3

14. DP on subset | Subsequence & target

Rules

- (i) Express every index in terms of (index, target).
- (ii) Explore possibilities of that index.
either Arr[i] will be the part of our Subset or Subsequence.
- (iii) return T or F according to cond'n

Question Name:- Subset equal to target

$$\text{Arr} = [3, 1, 1, 4] \quad \text{target} = 4$$

$f(3, 4)$ → This will give the result whether from index 0 to 3, there is target = 4 or not.

→ On every index, there is two possibility. Whether we should include it in our answer or not. And accordingly need to handle the target.

Let's See recursive Tree

→ Base Case will be

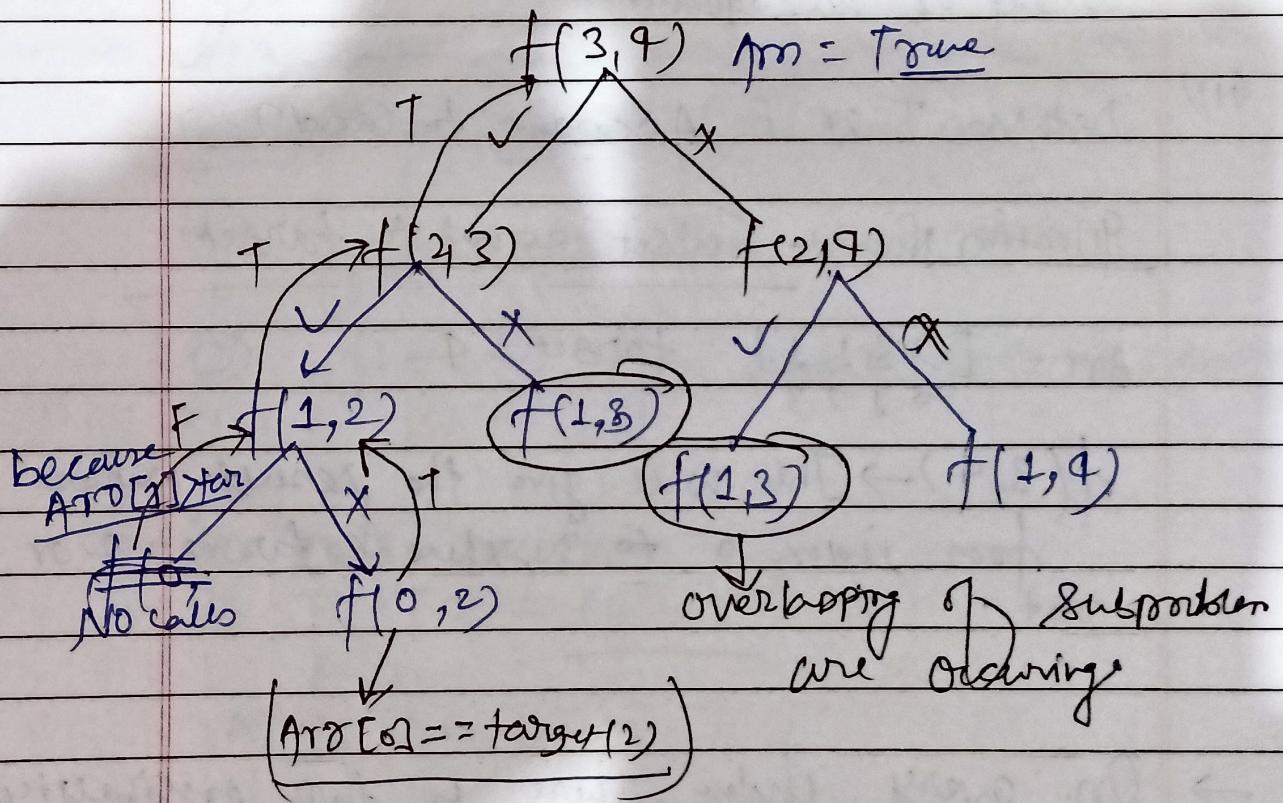
when $i == j == target$

$i == 0$

return $\{arr[0] == target\}$;

because $i == 0$ means only one element in the array.

$arr = [2, 3, 4, 1]$



$$f(\underline{1}, \underline{2}) = ?$$

This function indicates that from index 0 to 1, there is $\text{target} = 2$ or not.

$\rightarrow [2, 3], \text{target} = 2$

o 1

6/28/22, 12:07 PM

SubsetSumEqualToTarget.cpp

```
1 /*
2 You are given an array/list 'ARR' of 'N' positive integers and an integer 'K'. Your
3 task is to check if there exists a subset in 'ARR' with a sum equal to 'K'.
4 Note: Return true if there exists a subset with sum equal to 'K'. Otherwise, return
5 false.
6 For Example :
7 If 'ARR' is {1,2,3,4} and 'K' = 4, then there exists 2 subsets with sum = 4. These
8 are {1,3} and {4}. Hence, return true.
9 */
10 #include<bits/stdc++.h>
11 using namespace std;
12
13 /*
14 In recursive we are starting from index n-1 to 0 that is top down approach .
15 f(n-1,target)=this will give the answer whether any subset available equal to target
16 or not .
17
18 In subset or subsequence problem use inclusion or exclusion method .
19 */
20 int recursiveTarget(vector<int>&input,int n ,int target)
21 {
22     if(target==0)
23     {
24         return 1;
25     }
26     if(n==0)
27     {
28         return input[n]==target?1:0;
29         // Agar array in ek hi element hai to oo simply target ke hi equal hi hona
30         chahiye otherwise return false
31     }
32     int taken=0;
33     int notaken=0;
34     // for any particular , we have two option whether we take or not take the
35     // element
36     notaken=recursiveTarget(input,n-1,target);
37
38     if(input[n]<=target)
39     {
40         taken=recursiveTarget(input,n-1,target-input[n]);
41     }
42
43     return taken||notaken;
44 }
45
46 int tabulationCheck(vector<int> &input, int target)
47 {
48     int n=input.size();
49     vector<vector<int>>dp(n,vector<int>(target+1,0));
50     /*
51      dp[i][j]= the will give the answer by taking index [0....i]. any subset equal to
52      target(j) or not
53
54      for base case
55      if target ==0 , its is possible everywhere because empty subset .
56
57      if(i==0) then dp[0][arr[0]]=true; because of only one element are there so that
58      element will be only true.
59      */
60 }
```

localhost:4649/?mode=clike

6/28/22, 12:07 PM

SubsetSumEqualToTarget.cpp

```
52     for(int i=0;i<n;i++)
53     {
54         dp[i][0]=1;
55     }
56     dp[0][input[0]]=1;
57     for(int i=1;i<n;i++)
58     {
59         for(int j=1;j<=target;j++)
60         {
61             bool nottaken = dp[i - 1][j];
62             bool taken = false;
63             if (input[i] <= j)
64             {
65                 taken = dp[i - 1][j - input[i]];
66             }
67             dp[i][j] = taken || nottaken;
68         }
69     }
70     return dp[n-1][target];
71 }
72
73
74 int main()
75 {
76     int n;
77     cin>>n;
78     vector<int>input(n);
79     for(int i=0;i<n;i++)
80     {
81         cin>>input[i];
82     }
83     int target;
84     cin>>target;
85
86     int ans1= recursiveTarget(input,n-1,target);
87     cout<<ans1<<endl;
88     int ans2= tabulationCheck(input,target);
89     cout<<ans2<<endl;
90 }
91
92 /*
93 * In recursion + memoization
94
95 time complexity: O(n*target)
96 Space complexity: O(n*target) + O(n) for recursion stack memory
97
98 In tabulation
99
100 time complexity: O(n*target)
101 Space complexity: O(n*target)
102
103
104 */
105 */
```

1/2

localhost:4649/?mode=clike

2/2

15. Partition equal subset

Arr = [2, 3, 3, 9, 7, 5] target = 10

→ Divide it into two subset such that
sum of S_1 = sum of S_2 .

$$\boxed{S_1 = S_2 = S_{\text{total}}}$$

→ If $\text{sum}(\text{Arr})$ is odd then Ans will be false

If sum is even then we should search for $\text{sum}/2$ as target in Arr.

→ same question as previous

• ① Array is given

② target = $\text{sum}/2$

6/28/22, 12:10 PM

EqualSumPartition.cpp

```
1 /*
2 Given a non-empty array nums containing only positive integers, find if the array can
3 be partitioned into two subsets such that the sum of elements in both subsets is
4 equal.
5
6 Example 1:
7
8 Input: nums = [1,5,11,5]
9 Output: true
10 Explanation: The array can be partitioned as [1, 5, 5] and [11].
11 */
12 #include <bits/stdc++.h>
13 using namespace std;
14
15 int recursiveTarget(vector<int> &input, int n, int target)
16 {
17     if (target == 0)
18     {
19         return 1;
20     }
21     if (n == 0)
22     {
23         return input[n] == target ? 1 : 0;
24     }
25     int taken = 0;
26     int notaken = 0;
27
28     notaken = recursiveTarget(input, n - 1, target);
29
30     if (input[n] <= target)
31     {
32         taken = recursiveTarget(input, n - 1, target - input[n]);
33     }
34
35     return taken || notaken;
36 }
37
38 int tabulationCheck(vector<int> &input, int target)
39 {
40     int n = input.size();
41     vector<vector<int>> dp(n, vector<int>(target + 1, 0));
42
43     for (int i = 0; i < n; i++)
44     {
45         dp[i][0] = 1;
46     }
47     dp[0][input[0]] = 1;
48     for (int i = 1; i < n; i++)
49     {
50         for (int j = 1; j <= target; j++)
51         {
52             bool nottaken = dp[i - 1][j];
53             bool taken = false;
54             if (input[i] <= j)
55             {
56                 taken = dp[i - 1][j - input[i]];
57             }
58             dp[i][j] = taken || nottaken;
59         }
60     }
61 }
```

localhost:4649/?mode=clike

6/28/22, 12:10 PM

EqualSumPartition.cpp

```
58     }
59     return dp[n - 1][target];
60 }
61
62 int main()
63 {
64     int n;
65     cin >> n;
66     vector<int> input(n);
67     int sum=0;
68     for (int i = 0; i < n; i++)
69     {
70         cin >> input[i];
71         sum+=input[i];
72     }
73     int target;
74
75     if(sum%2==1)
76     {
77         cout<<0<<endl;
78         return 0;
79     }
80     else
81     {
82         target=sum/2;
83     }
84     int ans1 = recursiveTarget(input, n - 1, target);
85     cout << ans1 << endl;
86     int ans2 = tabulationCheck(input, target);
87     cout << ans2 << endl;
88 }
89 /*
90 complexity will be same as previous question
91 */
92 }
```

1/2

localhost:4649/?mode=clike

2/2

16.

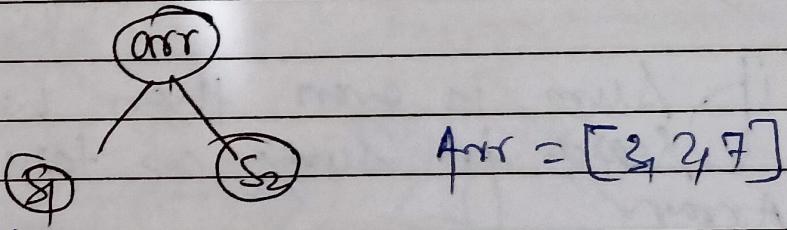
Partition a set into two subsets with minimum absolute difference.

$$\text{Arr} = [1, 2, 3, 4]$$

$$\begin{matrix} \textcircled{3} & \textcircled{7} \\ \{1, 2\} & \{3, 4\} \end{matrix} = |7 - 3| = 4$$

$$\begin{matrix} \textcircled{4} & \textcircled{6} \\ \{4, 3\} & \{2, 1\} \end{matrix} = |6 - 4| = 2$$

$$\checkmark \begin{matrix} \textcircled{5} & \textcircled{5} \\ \{4, 3\} & \{2, 1\} \end{matrix} = |5 - 5| = 0$$



abs $|S_1 - S_2|$ is minimal

	0	1	2	3	4	5	6	7	8	9	10	11	12
fill 0th	T												
fill 1st	T												
fill 2nd	T	F	T	T	X	T	X	T	X	T	T	X	T

Here I have made dp[7][sums]

I have filled last row to get the answer

In Array [3 2 1 7] Sum = 12

from dp table we see like
Subset

Possible sum using whole array : last row
of dp array

→	0	1	2	3	4	5	6	7	8	9	10	11	12
	✓	✗	✓✓	✗	✗	✗	✗	✓	✓	✗	✗	✓	✓

→ Possible sum

$S_1 = 0, 2, 3, 5, 7, 9, 10, 12$

→ If $S_1 = 0, S_2 = 12 - 0 = 12$

S_1	S_2	(Sum - S_1)	Ans
0	12	12	12
2	10	8	
3	9	6	
5	7	2	2 → min
7	5	2	
9	3	6	
10	2	8	
12	0	12	

If sequence getting repeat we found free
half the sum only

→ It's just extension of problem 14.
make $dp[n][sum]$ And Col. min from
last row in dp table
int ans = INT_MAX

for (int i = 0; i < sum; i++)

ans = min(ans, abs(sum - dp[n-1][i]))

Code part

6/28/22, 12:12 PM

```
PartitionWithMinimumDifference.cpp

1 /*
2 Given a set of integers, the task is to divide it into two sets S1 and S2 such that
3 the absolute difference between their sums is minimum.
4 If there is a set S with n elements, then if we assume Subset1 has m elements,
5 Subset2 must have n-m elements and the value of abs(sum(Subset1) - sum(Subset2))
6 should be minimum.
7 */
8 #include <bits/stdc++.h>
9 using namespace std;
10
11 int tabulationCheck(vector<int> &input, int target)
12 {
13     int n = input.size();
14     vector<vector<int>> dp(n, vector<int>(target + 1, 0));
15
16     for (int i = 0; i < n; i++)
17     {
18         dp[i][0] = 1;
19     }
20     dp[0][input[0]] = 1;
21     for (int i = 1; i < n; i++)
22     {
23         for (int j = 1; j <= target; j++)
24         {
25             bool nottaken=dp[i-1][j];
26             bool taken=false;
27             if(input[i]<=j)
28             {
29                 taken=dp[i-1][j-input[i]];
30             }
31             dp[i][j]=taken||nottaken;
32         }
33
34         int ans = INT_MAX;
35         target--;//we have already done target=sum+1; that is why we are decreasing it .
36
37         for (int i = 0; i <= target / 2; i++)
38         {
39             if (dp[n - 1][i])
40             {
41                 int s1=i;
42                 int s2=(target-i);
43                 ans= min(ans,abs(s1-s2));
44             }
45         }
46
47         return ans;
48     }
49
50 int main()
51 {
52     int n;
53     cin >> n;
54     vector<int> input(n);
55     int sum = 0;
56     for (int i = 0; i < n; i++)
```

localhost:4649/?mode=clike

6/28/22, 12:12 PM

```
PartitionWithMinimumDifference.cpp

57     {
58         cin >> input[i];
59         sum += input[i];
60     }
61     /*
62      As described above , we will use the last row in Dp table and out target will
63      shift to sum+1; j++
64
65      that means
66
67      1. array will be there
68      2. target= sum+1;
69
70      */
71     int target = sum + 1;
72
73     int ans2 = tabulationCheck(input, target);
74     cout << ans2 << endl;
75 }
```

1/2

localhost:4649/?mode=clike

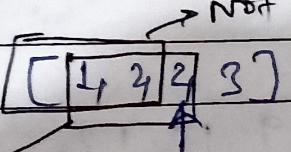
2/2

17. Number of Subsets

[1, 2, 3] target = 3

→ Count the number of subset in array
 $\{1, 2\}, \{1, 3\}, \{3\} = 3$ subsets

Pick and not pick

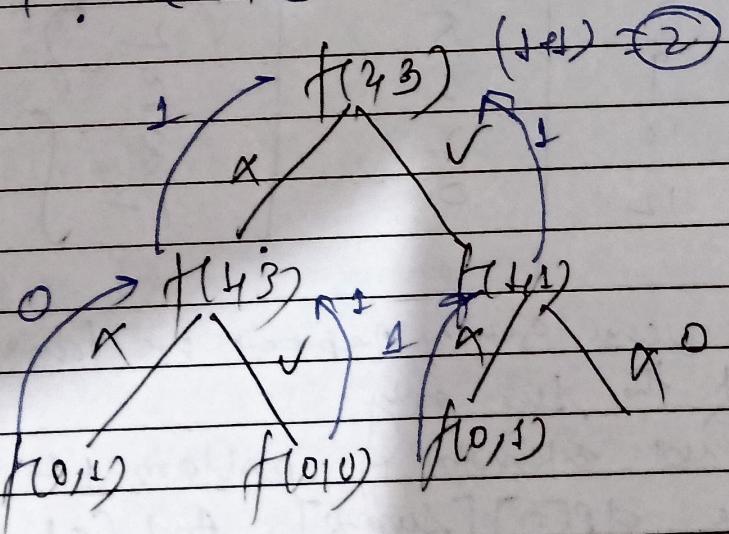


pick for index = 2 decision like pick and not pick.

→ for every element two choices

then Complexity of $O(2^n)$

Arr = [1, 2, 3] target = 3



6/28/22, 12:14 PM

CountPartitionWithGivenDiff.cpp

```
1 /*
2 Given an array 'ARR', partition it into two subsets (possibly empty) such that their
3 union is the original array. Let the sum of the elements of these two subsets be 'S1'
4 and 'S2'.
5
6 Given a difference 'D', count the number of partitions in which 'S1' is greater than
7 or equal to 'S2' and the difference between 'S1' and 'S2' is equal to 'D'. Since the
8 answer may be too large, return it modulo '10^9 + 7'.
9 */
10 #include <bits/stdc++.h>
11 using namespace std;
12
13 int recursiveTarget(vector<int> &input, int n, vector<vector<int>> &memo, int target)
14 {
15     if (target == 0)
16     {
17         return 1;
18     }
19     if (n == 0)
20     {
21         return input[n] == target;
22     }
23     if (memo[n][target] != -1)
24     {
25         return memo[n][target];
26     }
27     int nottaken = 0;
28     int taken = 0;
29     nottaken = recursiveTarget(input, n - 1, memo, target); // that means 0 to n-1th
30     index tak kitna subset hai
31     if (input[n] <= target)
32     {
33         taken = recursiveTarget(input, n - 1, memo, target - input[n]); // that means
34         nth index kitna subset hai
35     }
36     memo[n][target] = taken + nottaken;
37
38     return memo[n][target];
39 }
40
41 int tabulationCheck(vector<int> &input, int target)
42 {
43     int n = input.size();
44     vector<vector<int>> dp(n, vector<int>(target + 1, 0));
45     for (int i = 0; i < n; i++)
46     {
47         dp[i][0] = 1;
48     }
49     dp[0][input[0]] = 1;
50     for (int i = 1; i < n; i++)
51     {
52         for (int j = 1; j <= target; j++)
53         {
54             int nottaken = dp[i - 1][j];
55             int taken = 0;
56             if (input[i] <= j)
57             {
58                 taken = dp[i - 1][j - input[i]];
59             }
60             dp[i][j] = taken + nottaken;
61         }
62     }
63     return dp[n - 1][target];
64 }
65
66 int main()
67 {
68     int n;
69     cin >> n;
70     vector<int> input(n);
71     int difference;
72     int s1pluss2=0;
73     for (int i = 0; i < n; i++)
74     {
75         cin >> input[i];
76         s1pluss2+=input[i];
77     }
78     cin >>difference;
79     int s1minus2=difference;
80
81     int s1= s1minus2+s1pluss2;
82     if(s1%2)
83     {
84         cout<<0<<" "<<0<<endl;
85         return 0;
86     }
87     int target=s1/2;
88     vector<vector<int>> memo(n, vector<int>(target + 1, -1));
89
90     int ans1 = recursiveTarget(input, n - 1, memo, target);
91
92     int ans2 = tabulationCheck(input, target);
93     cout << ans1 << " " << ans2 << endl;
94 }
```

6/28/22, 12:14 PM

CountPartitionWithGivenDiff.cpp

```
1 }
2     dp[i][j] = taken + nottaken;
3 }
4
5 return dp[n - 1][target];
6 }
7
8 int main()
9 {
10    int n;
11    cin >> n;
12    vector<int> input(n);
13    int difference;
14    int s1pluss2=0;
15    for (int i = 0; i < n; i++)
16    {
17        cin >> input[i];
18        s1pluss2+=input[i];
19    }
20    cin >>difference;
21    int s1minus2=difference;
22
23    int s1= s1minus2+s1pluss2;
24    if(s1%2)
25    {
26        cout<<0<<" "<<0<<endl;
27        return 0;
28    }
29    int target=s1/2;
30    vector<vector<int>> memo(n, vector<int>(target + 1, -1));
31
32    int ans1 = recursiveTarget(input, n - 1, memo, target);
33
34    int ans2 = tabulationCheck(input, target);
35    cout << ans1 << " " << ans2 << endl;
36 }
```

18. Count partition with given difference

for a given array we have to
find

$$S_1 - S_2 = q$$

$$S_1 + S_2 = \text{sum of all elements}$$

$$S_1 - S_2 = q$$

$$S_1 + S_2 = \text{Sum}$$

$$2S_1 = \frac{\text{Sum} + q}{2}$$

$$\Rightarrow S_1 = \left(\frac{\text{Sum} + q}{2} \right)$$

So, now our question is like finding
 $\frac{\text{Sum} + q}{2}$ in Array.

Input:

$$\text{Array } \left\{ \frac{\text{Sum} + q}{2} \right\}$$

$$\text{target} = \left(\frac{\text{Sum} + q}{2} \right)$$

just apply below question concept

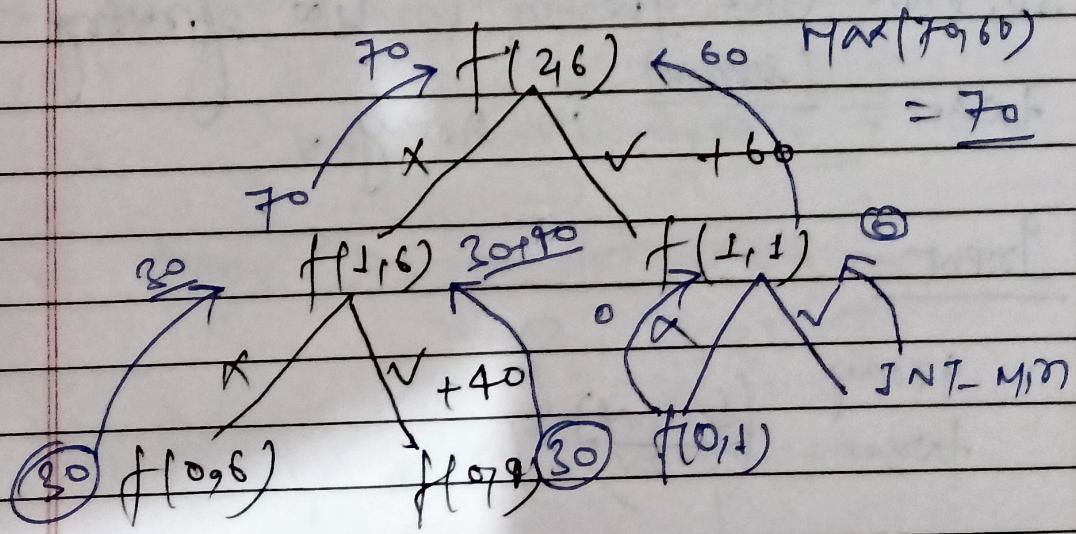
19. 0/1 Knapsack

Rules

- Explore in terms of (i, w) .
- Explore all possible pick and not picked.
- Maximize probability
- for all possible combination we use recursion

Example

wt \rightarrow 3 2 5 weight = 6
 Val \rightarrow 30 40 60
 0 1 2



Rules for Tabulation

1. Base Case
2. Use for loops according to the states
3. Copy recurrence relation

6/28/22, 12:21 PM

ZeroOrOneKnapsack.cpp

```
1 /*  
2 Problem Statement  
3  
4 A thief is robbing a store and can carry a maximal weight of W into his knapsack.  
5 There are N items and the ith item weighs wi and is of value vi. Considering the  
6 constraints of the maximum weight that a knapsack can carry, you have to find and  
7 return the maximum value that a thief can generate by stealing items.  
8 */  
9 #include<bits/stdc++.h>  
10 using namespace std;  
11  
12 int recursiveTarget( vector<int>&weight, vector<int>&value, int n  
13 , vector<vector<int>>&memo, int capacity)  
14 {  
15     if(n==0)  
16     {  
17         if(weight[n]>capacity)  
18             return 0;  
19         else  
20             return value[n];  
21     }  
22     if(capacity==0)  
23         return 0;  
24  
25     if(memo[n][capacity]!=-1)  
26         return memo[n][capacity];  
27  
28     int Exclude = recursiveTarget(weight,value,n-1,memo,capacity);  
29     int Include = 0;  
30     if(weight[n]<=capacity)  
31         Include = value[n] + recursiveTarget(weight,value,n-1,memo,capacity-  
32         weight[n]);  
33     memo[n][capacity] = max(Exclude,Include);  
34     return memo[n][capacity];  
35 }  
36  
37 int tabulationCheck(vector<int>&weight, vector<int>&value, int capacity)  
38 {  
39     int n = weight.size();  
40     vector<vector<int>> dp(n, vector<int>(capacity+1, 0));  
41  
42     // Base case according to the recursive one  
43     for(int i=weight[0]; i<=capacity; i++)  
44     {  
45         dp[0][i] = value[0];  
46     }  
47     if(capacity==0)  
48     {  
49         return 0;  
50     }  
51  
52     for(int i=1; i<n; i++)  
53     {  
54         for(int j=0; j<=capacity; j++)  
55         {  
56             int taken=0;  
57             int notTaken=0;  
58             if(weight[i]<=j)  
59                 taken = value[i] + dp[i-1][j-weight[i]];  
60             notTaken = dp[i-1][j];  
61             dp[i][j] = max(taken,notTaken);  
62         }  
63     }  
64 }  
65  
66 int main()  
67 {  
68     int n;  
69     cin >> n;  
70     vector<int> weight(n);  
71     vector<int> value(n);  
72     for(int i=0; i<n; i++)  
73     {  
74         cin >> weight[i];  
75     }  
76     for(int i=0; i<n; i++)  
77     {  
78         cin >> value[i];  
79     }  
80     int capacity;  
81     cin >> capacity;  
82     vector<vector<int>> memo(n, vector<int>(capacity+1, -1));  
83     int ans=recursiveTarget(weight,value,n-1,memo,capacity);  
84     cout << ans << endl;  
85     int ans2=tabulationCheck(weight,value,capacity);  
86     cout << ans2 << endl;  
87     return 0;  
88 }
```

6/28/22, 12:21 PM

ZeroOrOneKnapsack.cpp

```
55     {  
56         taken = value[i] + dp[i-1][j-weight[i]];  
57     }  
58     notTaken = dp[i-1][j];  
59     dp[i][j] = max(taken,notTaken);  
60 }  
61  
62 return dp[n-1][capacity];  
63 }  
64  
65  
66 int main()  
67 {  
68     int n;  
69     cin >> n;  
70     vector<int> weight(n);  
71     vector<int> value(n);  
72     for(int i=0; i<n; i++)  
73     {  
74         cin >> weight[i];  
75     }  
76     for(int i=0; i<n; i++)  
77     {  
78         cin >> value[i];  
79     }  
80     int capacity;  
81     cin >> capacity;  
82     vector<vector<int>> memo(n, vector<int>(capacity+1, -1));  
83     int ans=recursiveTarget(weight,value,n-1,memo,capacity);  
84     cout << ans << endl;  
85     int ans2=tabulationCheck(weight,value,capacity);  
86     cout << ans2 << endl;  
87     return 0;  
88 }
```

20.

Minimum Coins

$$\text{arr} = [1, 4, 3]$$

$$\text{target} = 7$$

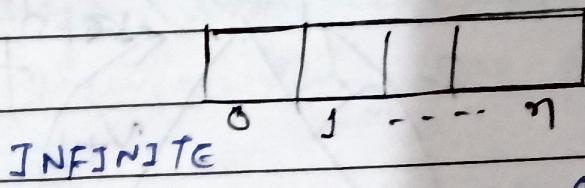
$$\begin{cases} (3, 3, 1) \rightarrow \text{len} = 3 & \{ \text{Min} = 3 \} \\ (4, 3, 1) \rightarrow \text{len} = 4 & \end{cases}$$

→ Try all possible Combs, then return
min of all possible Combs

→ Recursion will come if

→ find out min coins required to represent
the target

$f(n, t) \rightarrow$ What is minimum no of
coins required to represent $\$$
Counting in given array from index 0 to n .
The coins



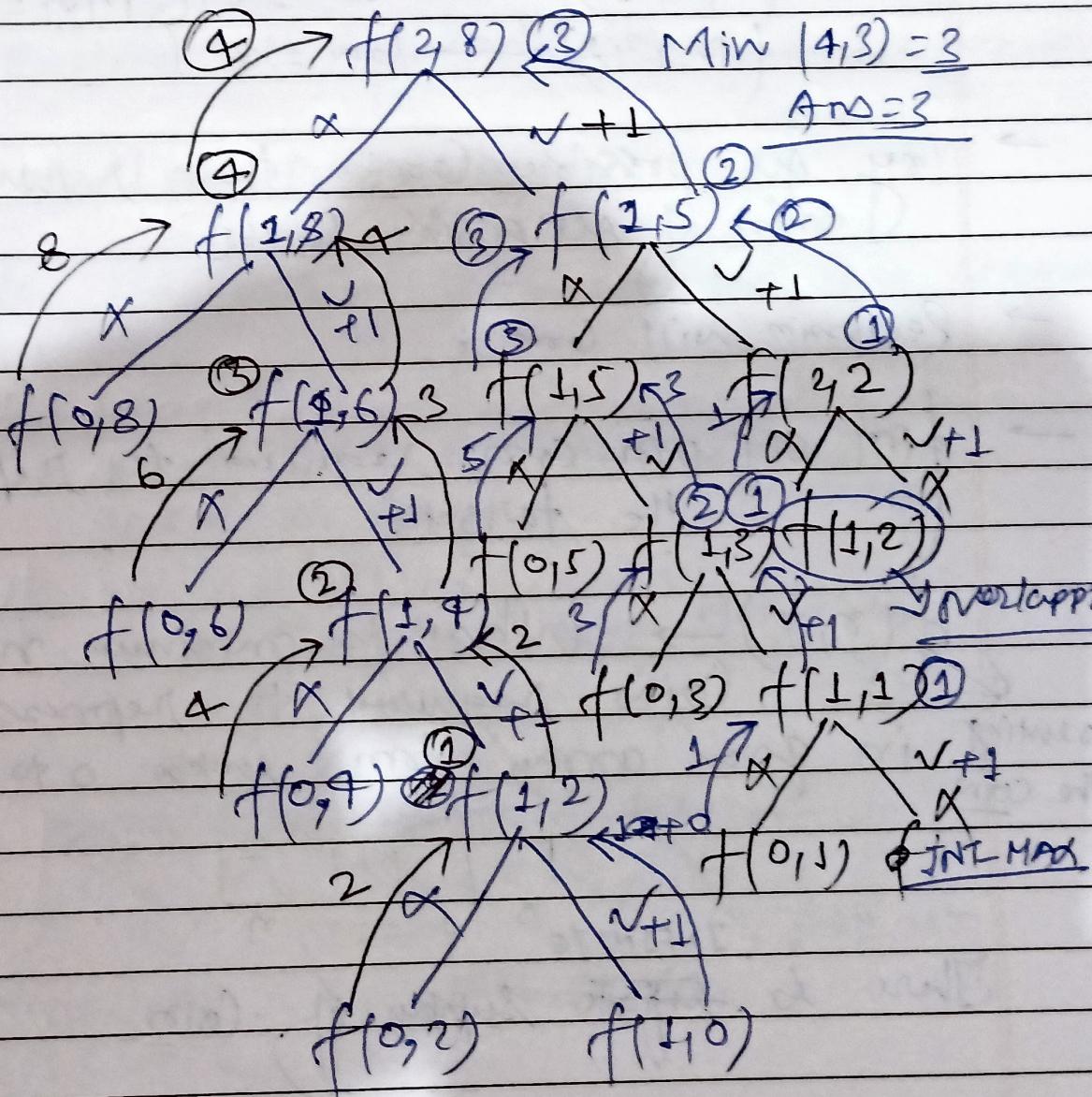
There is infinite supply of coins

→ For the Base case always think about
last element / single element.

Let's make free diagram

$$ar = \{443\}$$

target = 8



→ This may ~~log~~ look confused but if you go through the tree you will get the force properly

6/28/22, 12:29 PM

MinimumCoinsToTarget.cpp

```
/*
2 Bob went to his favourite bakery to buy some pastries. After picking up his favourite
3 pastries his total bill was P cents. Bob lives in Berland where all the money is in
4 the form of coins with denominations {1, 2, 5, 10, 20, 50, 100, 500, 1000}.
5
6 Bob is not very good at maths and thinks fewer coins mean less money and he will be
7 happy if he gives minimum number of coins to the shopkeeper. Help Bob to find the
8 minimum number of coins that sums to P cents (assume that Bob has an infinite number
9 of coins of all denominations).
10 */
11 #include<bits/stdc++.h>
12 using namespace std;
13
14 int recursiveCount(vector<int> &value, int target, int n, vector<vector<int>>&memo)
15 {
16     // we are starting from last element then always think about first element for
17     // the base case ;
18     if(n==0)
19     {
20         if(target%value[n]==0)
21         {
22             return target/value[n];
23         }
24         else
25         {
26             return 1e9;
27         }
28     }
29     if(memo[n][target]!=-1)
30     {
31         return memo[n][target];
32     }
33
34     int taken=1e9;
35     int notTaken=1e9;
36
37     notTaken = recursiveCount(value, target, n-1,memo);
38     if(value[n]<target)
39     {
40         taken = recursiveCount(value, target - value[n], n,memo) + 1;
41     }
42     return memo[n][target]= min(taken,notTaken);
43 }
44
45 int tabulationCount(vector<int>&value,int target)
46 {
47     int n = value.size();
48     int dp[n][target+1];
49     // think about the last element ;
50     for(int i=0;i<=target;i++)
51     {
52         dp[0][i] = (i%value[0]==0)?(i/value[0]):1e9;
53     }
54     // we have already processed the 0th element then we need to start i==1
55     for(int i=1;i<n;i++)
56     {
57         for(int j=0;j<=target;j++)
58         {
59             // After writing the different states of DP then just copy the recurrence
60             relation
61         }
62     }
63 }
```

6/28/22, 12:29 PM

MinimumCoinsToTarget.cpp

```
int taken = 1e9;
int notTaken = 1e9;

notTaken = dp[i-1][j];
if (value[i] <= j)
{
    taken = dp[i][j-value[i]] + 1;
}
dp[i][j]= min(taken, notTaken);
}

// due to bottom up result will store at (n-1) and target
return dp[n-1][target];

}

int main()
{
    int target;
    cin>>target;
    vector<int>value{1, 2, 5, 10, 20, 50, 100, 500, 1000};
    int n= value.size();
    vector<vector<int>>memo(n, vector<int>(target+1,-1));
    int ans1= recursiveCount(value,target,n-1,memo);
    cout<<ans1<<endl;

    int ans2=tabulationCount(value,target);
    cout<<ans2<<endl;
}

/*
In recursion + memoization
time complexity : O(n*target);
space complexity : O(n*target) + recursion stack
In tabulation
time complexity : O(n*target);
space complexity : O(n*target)
*/

```

21. Target Sum



$$Arr[] = \{1, 4, 2, 1\}$$

$$\text{target} = 3$$

You have to assign $\pm 1, -1$ to every element to get target $= 3$.

$$\begin{array}{cccc} + & + & - & - \\ 1 & 2 & 3 & 1 \end{array} = -1$$

$$\begin{array}{cccc} - & + & + & - \\ & & & \end{array} = 3$$

- Q. How many ways we can assign the sign to achieve the given target.

→ We have already done this question before.
∴ Count the subset with given difference

Divide them into 2 subset named S_1 and S_2
Every element the part of S_1 and S_2

then problem resolve like

$$\left[\text{Sum}(S_1) - \text{Sum}(S_2) = \text{target} \right]$$

Such type of question we have already solved

22. Coin change 2

→ for In a given Array you need
to Count the number of ways
to represent target poimted Supply
if the Coin is infinite

$$\text{Arr} = [1, 2, 3, 4]$$

target = 4.

Here again we use principle of inclusion
and exclusion

→ if we are adding one element to our
answer then we can include same
element in the answer again.

→ if we are not including ^{to} answer then
we can move to next index.

You can see in explanation in the
Code section

6/28/22, 12:31 PM

CoinChangeii.cpp

```
1 /*
2 You are given an integer array coins representing coins of different denominations
3 and an integer amount representing a total amount of money.
4
5 Return the number of combinations that make up that amount. If that amount of money
6 cannot be made up by any combination of the coins, return 0.
7
8 You may assume that you have an infinite number of each kind of coin.
9 */
10
11 #include<bits/stdc++.h>
12 using namespace std;
13
14 int recursiveCount(vector<int>&value,int target,int n,vector<vector<int>>&memo)
15 {
16     // base case just think about the case when n==0;that means only one element are
17     // there .
18     if(n==0)
19     {
20         return target%value[0]==0;
21     }
22
23     if(memo[n][target]!=-1)
24     {
25         return memo[n][target];
26     }
27     int notinclude=0;
28     int include=0;
29     notinclude=recursiveCount(value,target,n-1,memo);
30     if(value[n]<=target)
31     {
32         include=recursiveCount(value,target-value[n],n,memo);
33     }
34     return memo[n][target]=(notinclude+include);
35 }
36
37 int tabulationCount(vector<int>&input,int target)
38 {
39     int n=input.size();
40     vector<vector<int>>dp(n,vector<int>(target+1,0));
41     for(int i=0;i<=target;i++)
42     {
43         dp[0][i]=(i%input[0]==0);
44     }
45
46     for(int i=1;i<n;i++)
47     {
48         for(int j=0;j<=target;j++)
49         {
50             int notinclude=0;
51             int include=0;
52             notinclude=dp[i-1][j];
53             if(input[i]<=j)
54             {
55                 include=dp[i][j-input[i]];
56             }
57             dp[i][j]=notinclude+include;
58         }
59     }
60 }
61
62 int main()
63 {
64     int n;
65     cin>>n;
66     vector<int>value(n);
67     for(int i=0;i<n;i++)
68     {
69         cin>>value[i];
70     }
71     int target;
72     cin>>target;
73     vector<vector<int>>memo(n,vector<int>(target+1,-1));
74     cout<<recursiveCount(value,target,n-1,memo)<<endl;
75     cout<<tabulationCount(value,target)<<endl;
76     return 0;
77 }
```

6/28/22, 12:31 PM

CoinChangeii.cpp

```
57     return dp[n-1][target];
58 }
59
60 int main()
61 {
62     int n;
63     cin>>n;
64     vector<int>value(n);
65     for(int i=0;i<n;i++)
66     {
67         cin>>value[i];
68     }
69     int target;
70     cin>>target;
71     vector<vector<int>>memo(n,vector<int>(target+1,-1));
72     cout<<recursiveCount(value,target,n-1,memo)<<endl;
73     cout<<tabulationCount(value,target)<<endl;
74     return 0;
75 }
```

6/28/22, 12:35 PM

UnboundedKnapSack.cpp

```
1 /*
2 You are given 'N' items with certain 'PROFIT' and 'WEIGHT' and a knapsack with weight
3 capacity 'W'. You need to fill the knapsack with the items in such a way that you get
4 the maximum profit. You are allowed to take one item multiple times.
5 For Example
6 Let us say we have 'N' = 3 items and a knapsack of capacity 'W' = 10
7 'PROFIT' = { 5, 11, 13 }
8 'WEIGHT' = { 2, 4, 6 }
9
10 We can fill the knapsack as:
11 1 item of weight 6 and 1 item of weight 4.
12 1 item of weight 6 and 2 items of weight 2.
13 2 items of weight 4 and 1 item of weight 2.
14 5 items of weight 2.
15 The maximum profit will be from case 3 i.e '27'. Therefore maximum profit = 27.
16 */
17 #include<bits/stdc++.h>
18 using namespace std;
19
20 int recursiveCal(vector<int>&value, vector<int>&wt, vector<vector<int>>&memo, int n, int w)
21 {
22     if(n==0)
23     {
24         int ans=(w/wt[0])*value[0];
25         return ans;
26     }
27     if(memo[n][w]!=-1)
28     {
29         return memo[n][w];
30     }
31     int include=0;
32     int exclude=0;
33     exclude=recursiveCal(value,wt,memo,n-1,w);
34     if(w>=wt[n])
35     {
36         include=recursiveCal(value,wt,memo,n,w-wt[n])+value[n];
37     }
38     memo[n][w]=max(include,exclude);
39     return memo[n][w];
40 }
41
42 int tabulationCal(vector<int>&value, vector<int>&wt, int w)
43 {
44     int n=value.size();
45     vector<vector<int>>dp(n, vector<int>(w+1, 0));
46     for(int i=0;i<=w;i++)
47     {
48         dp[0][i]=(i/wt[0])*value[0];
49     }
50     for(int i=1;i<n;i++)
51     {
52         for(int j=0;j<=w;j++)
53         {
54             int include = 0;
55             int exclude = 0;
56             exclude = dp[i- 1][j];
```

6/28/22, 12:35 PM

UnboundedKnapSack.cpp

```
57     if (j >= wt[i])
58     {
59         include = dp[i][j-wt[i]]+value[i];
60     }
61     dp[i][j]=max(include,exclude);
62 }
63 }
64 return dp[n-1][w];
65 }
66
67 int main()
68 {
69     int n,w;
70     cin>>n>>w;
71     vector<int> v(n);
72     vector<int> wt(n);
73     for(int i=0;i<n;i++)
74     {
75         cin>>v[i];
76     }
77     for(int i=0;i<n;i++)
78     {
79         cin>>wt[i];
80     }
81     vector<vector<int>>memo(n, vector<int>(w+1, -1));
82
83     int ans1= recursiveCal(v,wt,memo,n-1,w);
84     cout<<ans1<<endl;
85
86     int ans2= tabulationCal(v,wt,w);
87     cout<<ans2<<endl;
88
89     return 0;
90 }
91 }
```

6/28/22, 12:36 PM

rodCuttingProblem.cpp

```
1 /*
2 Problem Statement
3 Given a rod of length 'N' units. The rod can be cut into different sizes and each
4 size has a cost associated with it. Determine the maximum cost obtained by cutting
5 the rod and selling its pieces.
6
7 Note:
8 1. The sizes will range from 1 to 'N' and will be integers.
9
10 2. The sum of the pieces cut should be equal to 'N'.
11
12 3. Consider 1-based indexing.
13 */
14 #include <bits/stdc++.h>
15 using namespace std;
16
17 int recursiveCal(vector<int> &value, int index, int capacity, vector<vector<int>>
18 &memo)
19 {
20     // here length will index+1 because index is 0 based that is why we are doing +1;
21     /*
22     Here we are trying to make different combination of length to equal to capacity .
23     That means reverse of the problem given;
24     */
25     // base case if you have only one element then ? if you capacity=10 and n=0
26     // then you have only one choice to cut the rod in length equal to 1. because index=0 is
27     // equal to length 1;
28     if (index == 0)
29     {
30         return value[0] * capacity;
31     }
32
33     int include = 0;
34     int exclude = 0;
35     if (memo[index][capacity] != -1)
36     {
37         return memo[index][capacity];
38     }
39     exclude = 0 + recursiveCal(value, index - 1, capacity, memo);
40     int length = index+ 1;
41     if (capacity >= length)
42     {
43         include = recursiveCal(value, index, capacity - length, memo) + value[index];
44     }
45     return memo[index][capacity] = max(include, exclude);
46 }
47
48 int tabulationCheck(vector<int>&value)
49 {
50     int n = value.size();
51     int capacity = n + 1;
52     vector<vector<int>>dp(n, vector<int>(capacity, 0));
53     for(int i=0;i<=n;i++)
54     {
55         dp[0][i]=value[0]*i;
56     }
57
58     for(int i=1;i<n;i++)
59     {
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88 }
```

localhost:4649/?mode=clike

6/28/22, 12:36 PM

rodCuttingProblem.cpp

```
54     // states in dp , just think at i , what could be possible value of capacity
55     // i thin it would be 0 to n .
56     for(int j=0;j<=n;j++)
57     {
58         int include = 0;
59         int exclude = 0;
60         exclude = 0 + dp[i-1][j];
61         int length = i + 1;
62         if (j >= length)
63         {
64             include = dp[i][j-length]+value[i];
65         }
66         dp[i][j] = max(include, exclude);
67     }
68     return dp[n-1][n];
69 }
70
71 int main()
72 {
73     int n;
74     cin >> n;
75     vector<int> v(n);
76     for (int i = 0; i < n; i++)
77     {
78         cin >> v[i];
79     }
80
81     vector<vector<int>> memo(n, vector<int>(n + 1, -1));
82
83     int ans1 = recursiveCal(v, n - 1, n, memo);
84     cout << ans1 << endl;
85
86     int ans2 = tabulationCheck(v);
87     cout << ans2 << endl;
88 }
```

1/2

localhost:4649/?mode=clike

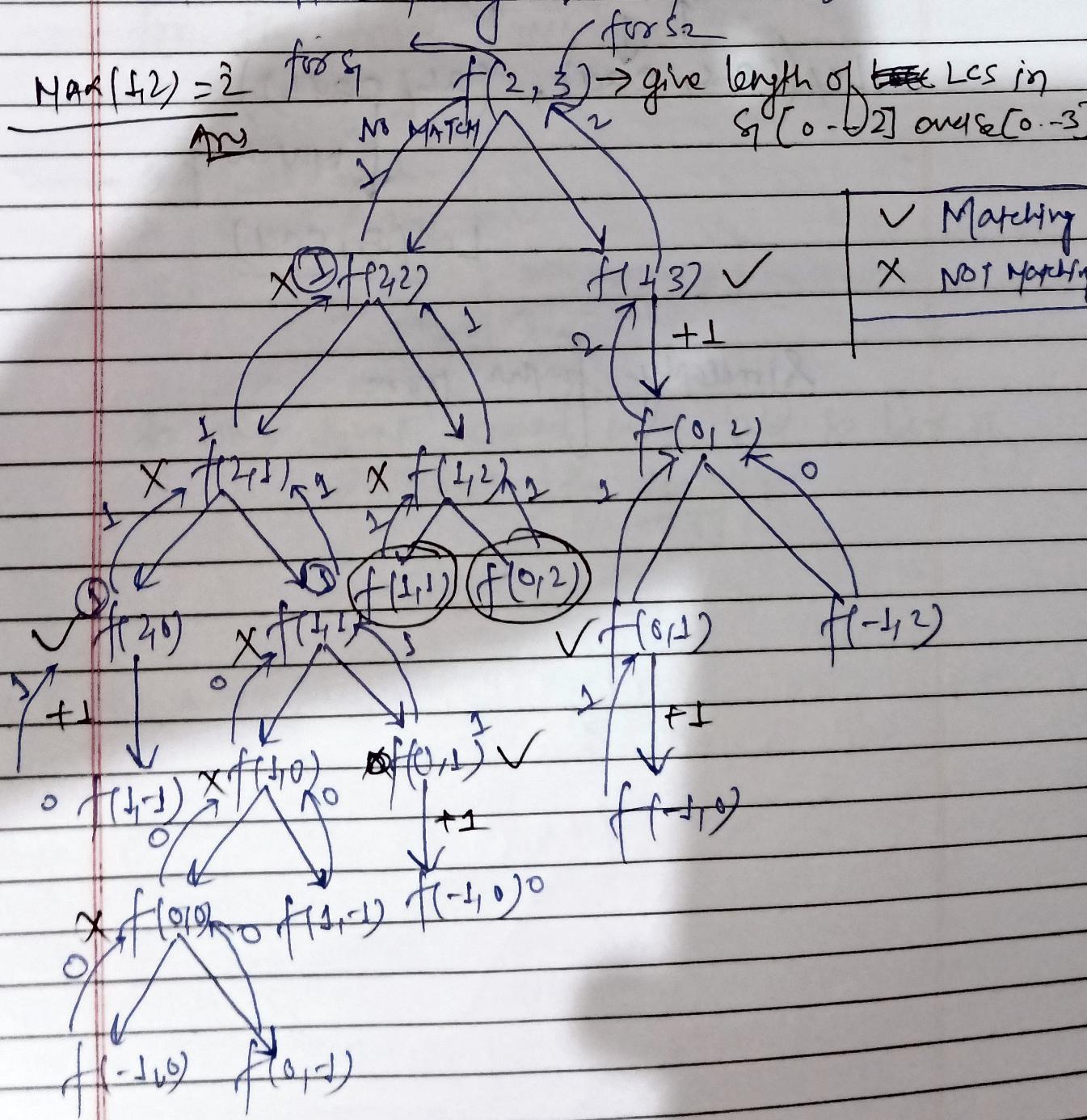
2/2

25. Longest Common Subsequence

$$\rightarrow S_1 = abc \quad S_2 = cadb$$

$$LCS(S_1, S_2) = \underline{ab} \quad \underline{\text{length} = 2} \quad \text{Ans}$$

Let's explore through the tree diagram.

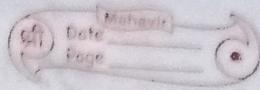


If our strings matching at particular index then we decrement both of the index otherwise leave it

$$\begin{array}{c}
 \text{Lcs}(abc, cadb) \\
 \uparrow \quad \uparrow \\
 \text{Lcp}((abc, ca), (ab, cadb)) \\
 \downarrow \quad \downarrow \text{Matching} \\
 \underline{\text{Lcs}(a, ca)}
 \end{array}$$

Similarly further more

26. Point LCP



$$S_1 = abc \quad S_2 = \underset{m}{cadb}$$

We are making DP of ~~not~~, or ~~not~~ 2nd because of ~~not~~ negative index base case. That's why we are shifting it by 1 right side.

for the table will look like

		cadb				
		0	0	0	0	0
a		0	0	1	1	1
b	0	0	1	1	2	1
c	0	1	1	1	2	1

NOT Matching X
[3, 4] index

So our final answer will be 5 as LCP of both the strings.

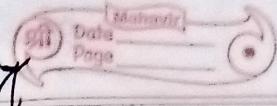
6/28/22, 12:39 PM

LongestCommonSubsequence.cpp

LongestCommonSubsequence.cpp

```
56         }
57     }
58 }
59 return dp[n][m];
60 }
61 }
62
63 int main()
64 {
65     string input1, input2;
66     cin >> input1 >> input2;
67     vector<vector<int>> dp(input1.length() + 1, vector<int>(input2.length() + 1,
68 -1));
69     int ans1 = recursiveLcs(input1, input2, input1.size() - 1, input2.size() - 1,
70 dp);
71     cout << ans1 << endl;
72
73     int ans2= tabulationLcs(input1,input2);
74     cout << ans2 << endl;
75 }
```

27. Longest Common Substring

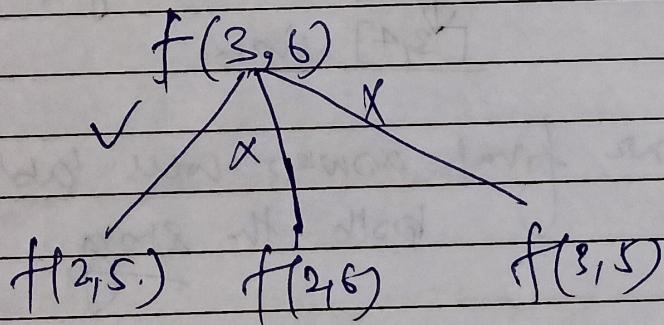


$$\rightarrow S_1 = \underset{0}{a} \underset{1}{b} \underset{2}{c} \underset{3}{d} \underset{4}{x} \underset{5}{y} \underset{6}{z}$$

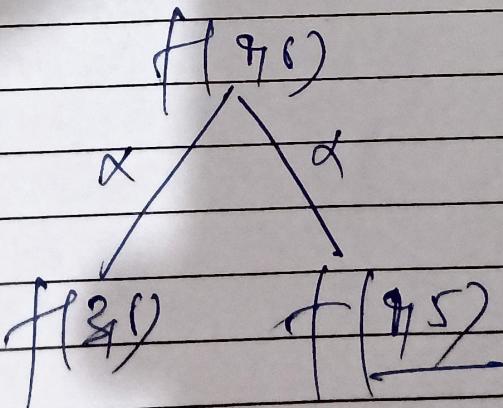
$$S_2 = \underset{0}{x} \underset{1}{y} \underset{2}{z} \underset{3}{a} \underset{4}{b} \underset{5}{c} \underset{6}{d}$$

then the longest common substring will
of length 4 (a b c d)

In recursive i/f matching, then we need
to call in three branch



\rightarrow If not matching then we need to call
in only two branches



In q/p you need to do
 $q[i] == p[j]$ if not matching.

6/28/22, 12:40 PM

LongestCommonSubstring.cpp

```
1 /*
2 Given two strings 'X' and 'Y', find the length of the longest common substring.
3 
4 Examples :
5 
6 Input : X = "GeeksforGeeks", y = "GeeksQuiz"
7 Output : 5
8 Explanation:
9 The longest common substring is "Geeks" and is of length 5.
10 */
11 #include <bits/stdc++.h>
12 using namespace std;
13 
14 int recursiveSubstring(string input1, string input2, int n, int m,
15 vector<vector<int>> &dp, int count)
16 {
17     if (n < 0 || m < 0)
18     {
19         return count;
20     }
21     if (dp[n][m] != -1)
22     {
23         // return dp[n][m];
24     }
25     int temp = 0;
26     if (input1[n] == input2[m])
27     {
28         temp = recursiveSubstring(input1, input2, n - 1, m - 1, dp, count + 1);
29 
30         temp = max(temp, max(recursiveSubstring(input1, input2, n - 1, m, dp, 0),
31 recursiveSubstring(input1, input2, n, m - 1, dp, 0)));
32     }
33     return dp[n][m] = temp;
34 }
35 
36 int tabulationSubstring(string input1, string input2)
37 {
38     int n = input1.length();
39     int m = input2.length();
40     vector<vector<int>> dp(n + 1, vector<int>(m + 1, 0));
41     for (int i = 0; i <= n; i++)
42     {
43         dp[i][0] = 0;
44     }
45     for (int i = 0; i <= m; i++)
46     {
47         dp[0][i] = 0;
48     }
49     int ans = 0;
50     for (int i = 1; i <= n; i++)
51     {
52         for (int j = 1; j <= m; j++)
53         {
54             if (input1[i - 1] == input2[j - 1])
55             {
56                 dp[i][j] = 1 + dp[i - 1][j - 1];
57                 ans = max(ans, dp[i][j]);
58             }
59         }
60     }
61 }
62 }
63 }
64 }
65 }
```

localhost:4649/?mode=clike

6/28/22, 12:40 PM

LongestCommonSubstring.cpp

```
58     else
59     {
60         dp[i][j] = 0;
61     }
62 }
63 }
64 return ans;
65 }
66 
67 int main()
68 {
69     string input1, input2;
70     cin >> input1 >> input2;
71     vector<vector<int>> dp(input1.length() + 1, vector<int>(input2.length() + 1,
72 -1));
73     int ans1 = recursiveSubstring(input1, input2, input1.length() - 1,
74 input2.length() - 1, dp, 0);
75     cout << ans1 << endl;
76 
77     int ans2 = tabulationSubstring(input1, input2);
78     cout << ans2 << endl;
79 }
```

1/2

localhost:4649/?mode=clike

2/2

6/28/22, 12:42 PM

MinimumInsertionToMakePalindrom.cpp

```
1 /*
2 Given a string s. In one step you can insert any character at any index of the
3 string.
4 Return the minimum number of steps to make s palindrome.
5 */
6 A Palindrome String is one that reads the same backward as well as forward.
7 */
8 #include <bits/stdc++.h>
9 using namespace std;
10 int recursiveLcs(string input1, string input2, int n, int m, vector<vector<int>> &dp)
11 {
12     if (n < 0 || m < 0)
13         return 0;
14     if (dp[n][m] != -1)
15         return dp[n][m];
16
17     if (input1[n] == input2[m])
18         return dp[n][m] = 1 + recursiveLcs(input1, input2, n - 1, m - 1, dp);
19     else
20         return dp[n][m] = max(recursiveLcs(input1, input2, n - 1, m, dp),
21                               recursiveLcs(input1, input2, n, m - 1, dp));
22 }
23
24 int tabulationLcs(string input1, string input2)
25 {
26     int n = input1.length();
27     int m = input2.length();
28     vector<vector<int>> dp(n + 1, vector<int>(m + 1, 0));
29
30     for (int i = 0; i <= n; i++)
31     {
32         dp[i][0] = 0;
33     }
34     for (int i = 0; i <= m; i++)
35     {
36         dp[0][i] = 0;
37     }
38     for (int i = 1; i <= n; i++)
39     {
40         for (int j = 1; j <= m; j++)
41         {
42             if (input1[i - 1] == input2[j - 1])
43             {
44                 dp[i][j] = 1 + dp[i - 1][j - 1];
45             }
46             else
47             {
48                 dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);
49             }
50         }
51     }
52     return dp[n][m];
53 }
54
55 int main()
56 {
57     string input1, input2;
```

localhost:4649/?mode=clike

6/28/22, 12:42 PM

MinimumInsertionToMakePalindrom.cpp

```
58     cin >> input1;
59     input2 = input1;
60     reverse(input2.begin(), input2.end());
61     vector<vector<int>> dp(input1.length() + 1, vector<int>(input2.length() + 1,
62     -1));
63     int ans1 = recursiveLcs(input1, input2, input1.size() - 1, input2.size() - 1,
64     dp);
65     cout << input1.size() - ans1 << endl;
66
67     int ans2 = tabulationLcs(input1, input2);
68     cout << input1.size() - ans2 << endl;
69 }
```

1/2

localhost:4649/?mode=clike

2/2

28. Longest Palindrome Subsequence (LPS)

→ Let suppose you have given string

input = "raushan";

input = "raushan";

For palindrome subsequence, you just need
to reverse the input string and call
for Lcs. That length will be LPS.

Lps-Lcs

LPS = LCS (real input, reversed input);

34. Agar ek g Mil v gaya toh v hongy
Second g ~~I~~ phone S Mein Search ~~not~~

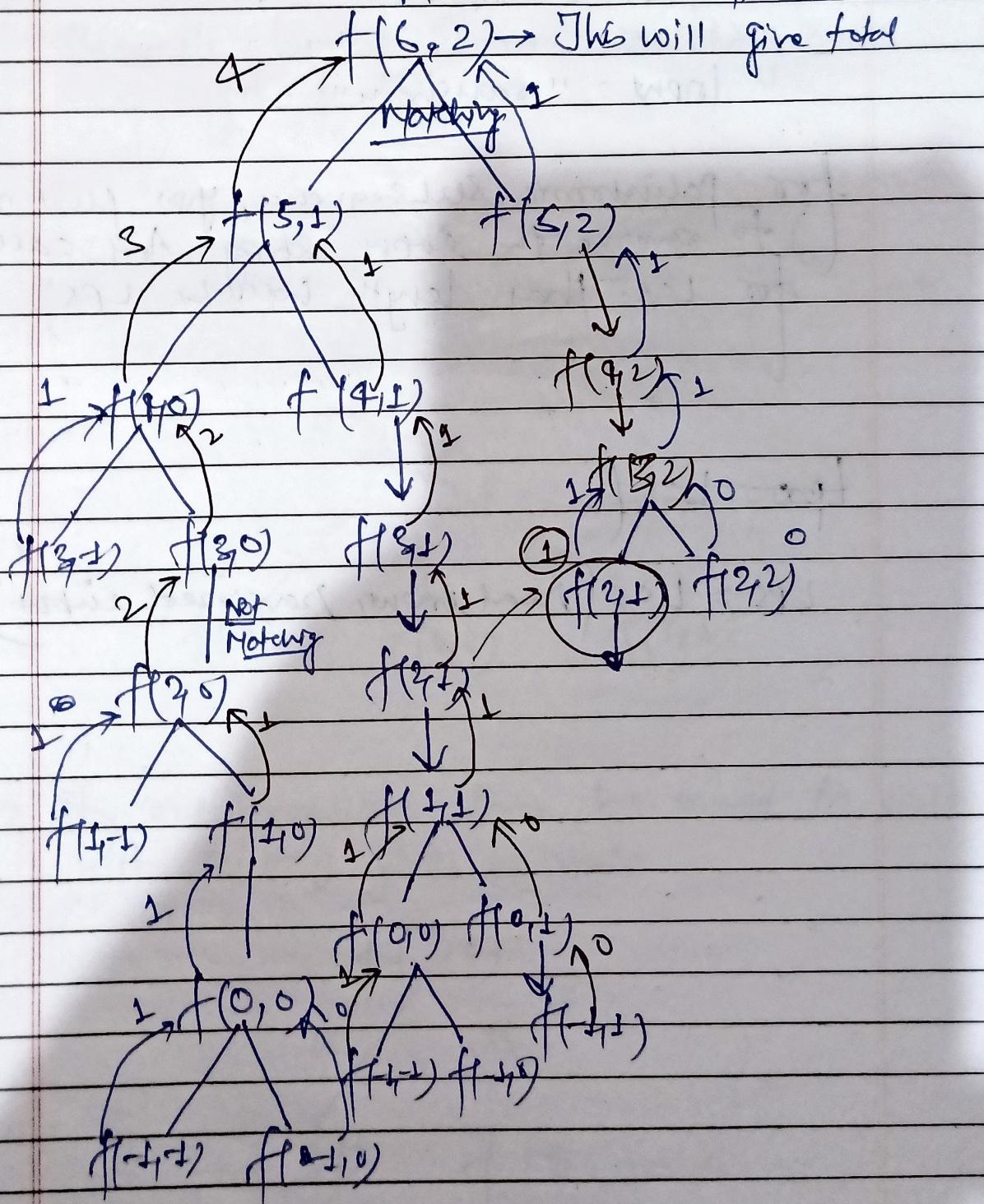
33. Distinct Sub-segments

$\Sigma = \{a, b, g, g\}$

$$f = \text{bag}^{\text{obj2}}$$

$$4+1=5$$

Amber Subseen



6/28/22, 12:43 PM

DistinctSequence.cpp

```
1 /*
2 Input: s = "babgbag", t = "bag"
3 Output: 5
4 Explanation:
5 As shown below, there are 5 ways you can generate "bag" from S.
6 babgbag
7 babgbag
8 babgbag
9 babgbag
10 babgbag
11 */
12
13 #include <bits/stdc++.h>
14 using namespace std;
15
16 int recursiveDistinctSequence(string input1, string input2, int n, int m,
17     vector<vector<int>> &dp)
18 {
19     if (m < 0)
20     {
21         return 1;
22     }
23     if (n < 0)
24     {
25         return 0;
26     }
27     if (dp[n][m] != -1)
28     {
29         return dp[n][m];
30     }
31     int ans = 0;
32     if (input1[n] == input2[m])
33     {
34         ans = (recursiveDistinctSequence(input1, input2, n - 1, m - 1, dp) +
35             recursiveDistinctSequence(input1, input2, n - 1, m, dp));
36     }
37     else
38     {
39         ans = recursiveDistinctSequence(input1, input2, n - 1, m, dp);
40     }
41     return dp[n][m] = ans;
42 }
43
44 int tabulation(string input1, string input2)
45 {
46     int n = input1.length();
47     int m = input2.length();
48     vector<vector<int>> dp(n + 1, vector<int>(m + 1, 0));
49
50     for (int i = 0; i <= m; i++)
51     {
52         dp[0][i] = 0;
53     }
54     for (int i = 0; i <= n; i++)
55     {
56         // for any i whose j==0 then we should always put dp[i][0]=1;
57         dp[i][0] = 1;
58     }
```

6/28/22, 12:43 PM

DistinctSequence.cpp

```
58 for (int i = 1; i <= n; i++)
59 {
60     for (int j = 1; j <= m; j++)
61     {
62         int ans = 0;
63         if (input1[i - 1] == input2[j - 1])
64         {
65             ans = dp[i - 1][j - 1] + dp[i - 1][j];
66         }
67         else
68         {
69             ans = dp[i - 1][j];
70         }
71         dp[i][j] = ans;
72     }
73 }
74 return dp[n][m];
75 }
76
77 int main()
78 {
79     string input1, input2;
80     cin >> input1 >> input2;
81
82     int n = input1.length();
83     int m = input2.length();
84     vector<vector<int>> dp(n + 1, vector<int>(m + 1, -1));
85
86     int ans1 = recursiveDistinctSequence(input1, input2, n - 1, m - 1, dp);
87     cout << ans1 << endl;
88
89     int ans2 = tabulation(input1, input2);
90     cout << ans2 << endl;
91 }
```

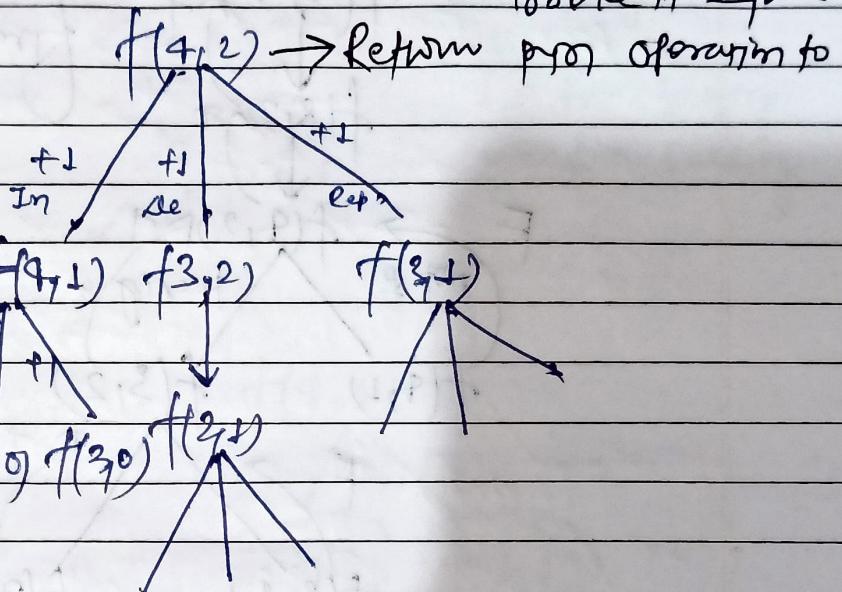
89 Edit Distance

→ we have to change it from word1 → word2

word1 = horse
0 1 2 3 4

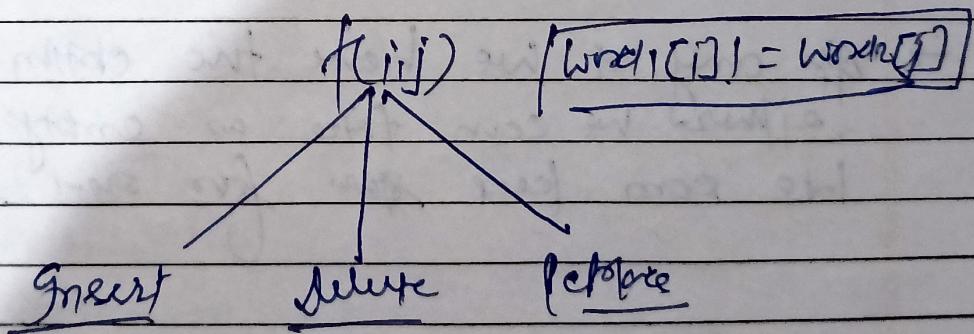
word2 = dog
0 1 2

- ~~Op~~
Insert
- Delete
- Replace



+
E(f(4,1)) f(3,0) f(3,1)

→ Boundary if character is not match for any given operation either then there is three option open.



If match then go straight down by this
(i-1, j-1) match

6/28/22, 12:44 PM

EditDistance.cpp

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

You have the following three operations permitted on a word:
Insert a character
Delete a character
Replace a character

Example 1:
Input: word1 = "horse", word2 = "ros"
Output: 3
Explanation:
horse -> rorse (replace 'h' with 'r')
rorse -> rose (remove 'r')
rose -> ros (remove 'e')
*/
#include<bits/stdc++.h>
using namespace std;

int recursiveEditDistance(string input1, string input2, int n, int m, vector<vector<int>>&dp)
{
    if(n<0 and m<0)
    {
        return 0;
    }
    if(n<0 and m>=0)
    {
        return m+1;
    }
    if(n>=0 and m<0)
    {
        return n+1;
    }
    if(dp[n][m]!=-1)
    {
        return dp[n][m];
    }
    int ans=0;
    if(input1[n]==input2[m])
    {
        ans=recursiveEditDistance(input1, input2, n-1, m-1, dp);
    }
    else
    {
        ans=min(recursiveEditDistance(input1, input2, n-1, m, dp),
                min(recursiveEditDistance(input1, input2, n, m-1, dp),
                    recursiveEditDistance(input1, input2, n-1, m-1, dp)))+1;
    }
    return dp[n][m]=ans;
}

int tabulationEditDistance(string input1, string input2)
{
    vector<vector<int>>dp(input1.length()+1, vector<int>(input2.length()+1, 0));

```

localhost:4649/?mode=clike

6/28/22, 12:44 PM

EditDistance.cpp

```
int n= input1.size();
int m= input2.size();
for(int i=0;i<=n;i++)
{
    dp[i][0]=i+1;
}
for(int i=0;i<=m;i++)
{
    dp[0][i]=i+1;
}
dp[0][0]=0;

for(int i=1;i<=n;i++)
{
    for(int j=1;j<=m;j++)
    {
        int ans=0;
        if(input1[i-1]==input2[j-1])
        {
            ans=dp[i-1][j-1];
        }
        else
        {
            ans=min(dp[i-1][j], min(dp[i][j-1], dp[i-1][j-1]))+1;
        }
        dp[i][j]=ans;
    }
}
return dp[n][m];

int main()
{
    string input1, input2;
    cin>>input1>>input2;
    vector<vector<int>> dp(input1.length()+1, vector<int>(input2.length()+1, -1));
    int
ans=recursiveEditDistance(input1, input2, input1.length()-1, input2.length()-1, dp);
cout<<ans<<endl;
ans=tabulationEditDistance(input1, input2);
cout<<ans<<endl;
}
```

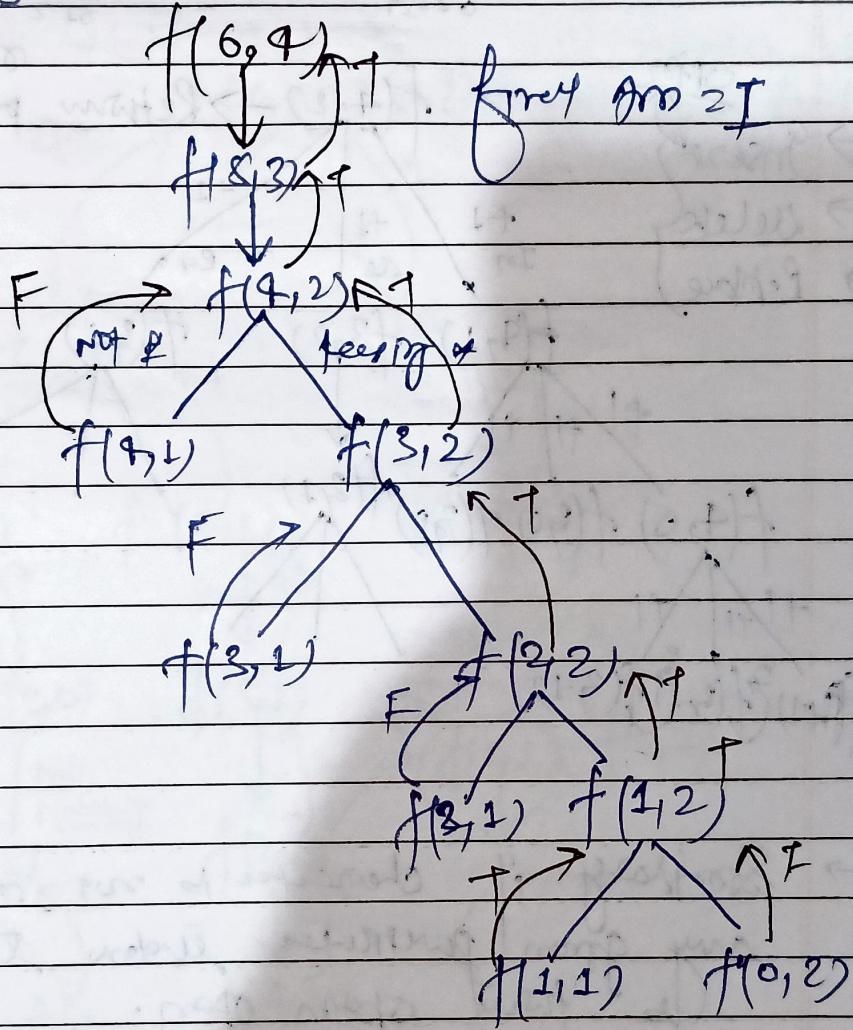
1/2

localhost:4649/?mode=clike

2/2

35 Wild Card matching

String $a b c d e f g$ pattern $a^* b^? c^d$



→ If any of we have two option either we can take as empty or we can keep that for next segment

6/28/22, 12:48 PM

WildCardMatching.cpp

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 /*
4 Given an input string (s) and a pattern (p), implement wildcard pattern matching
with support for '?' and '*' where:
5 '?' Matches any single character.
6 '*' Matches any sequence of characters (including the empty sequence).
7 The matching should cover the entire input string (not partial).
8 Example 1:
9 Input: s = "aa", p = "a"
10 Output: false
11 Explanation: "a" does not match the entire string "aa".
12 */
13 int recursiveWildCard(string input, string required, int n, int m,
vector<vector<int>> &dp)
14 {
15     if (n < 0 and m < 0)
16         return 1;
17     if (m >= 0 and n < 0)
18     {
19         return 0;
20     }
21     if (n >= 0 and m < 0)
22     {
23         bool flag = true;
24         for (int i = 0; i <= m; i++)
25         {
26             if (input[i] != '*')
27             {
28                 flag = false;
29                 break;
30             }
31         }
32         if (flag)
33             return 1;
34         else
35             return 0;
36     }
37     if (dp[n][m] != -1)
38         return dp[n][m];
39
40     if (input[n] == required[m] or input[n] == '?')
41     {
42         dp[n][m] = recursiveWildCard(input, required, n - 1, m - 1, dp);
43     }
44
45     else if (input[n] == '*')
46     {
47         dp[n][m] = recursiveWildCard(input, required, n - 1, m, dp) ||
recursiveWildCard(input, required, n, m - 1, dp);
48     }
49     else
50     {
51         dp[n][m] = 0;
52     }
53     return dp[n][m];
54 }
55 int tabulationWildCard(string input, string required)
56 {
```

localhost:4649/?mode=clike

6/28/22, 12:48 PM

WildCardMatching.cpp

```
57     int n = input.length();
58     int m = required.length();
59     vector<vector<int>> dp(n + 1, vector<int>(m + 1, 0));
60     for(int i=0;i<=m;i++)
61     {
62         dp[0][i]=0;
63     }
64     for(int i=0;i<=n;i++)
65     {
66         int flag = 1;
67         for (int j = 0; j <= i; j++)
68         {
69             if (input[j] != '*')
70             {
71                 flag = 0;
72                 break;
73             }
74         }
75         if (flag)
76             dp[i][0] = 1;
77         else
78             dp[i][0] = 0;
79     }
80     dp[0][0]=1;
81     for(int i=1;i<=n;i++)
82     {
83         for(int j=1;j<=m;j++)
84         {
85             if (input[i] == required[j] or input[i] == '?')
86             {
87                 dp[i][j] = dp[i - 1][j - 1];
88             }
89             else if (input[i] == '*')
90             {
91                 dp[i][j] = dp[i - 1][j] || dp[i][j - 1];
92             }
93             else
94             {
95                 dp[i][j] = 0;
96             }
97         }
98     }
99     return dp[n][m];
100 }
101 int main()
102 {
103     string input;
104     string required;
105     cin >> input;
106     cin >> required;
107     int n = input.length();
108     int m = required.length();
109     vector<vector<int>> dp(n + 1, vector<int>(m + 1, -1));
110     int ans = recursiveWildCard(input, required, n - 1, m - 1, dp);
111     int ans1= tabulationWildCard(input,required);
112     if (ans1 and ans)
113         cout << "YES";
114     else
115         cout << "NO";
116 }
```

localhost:4649/?mode=clike

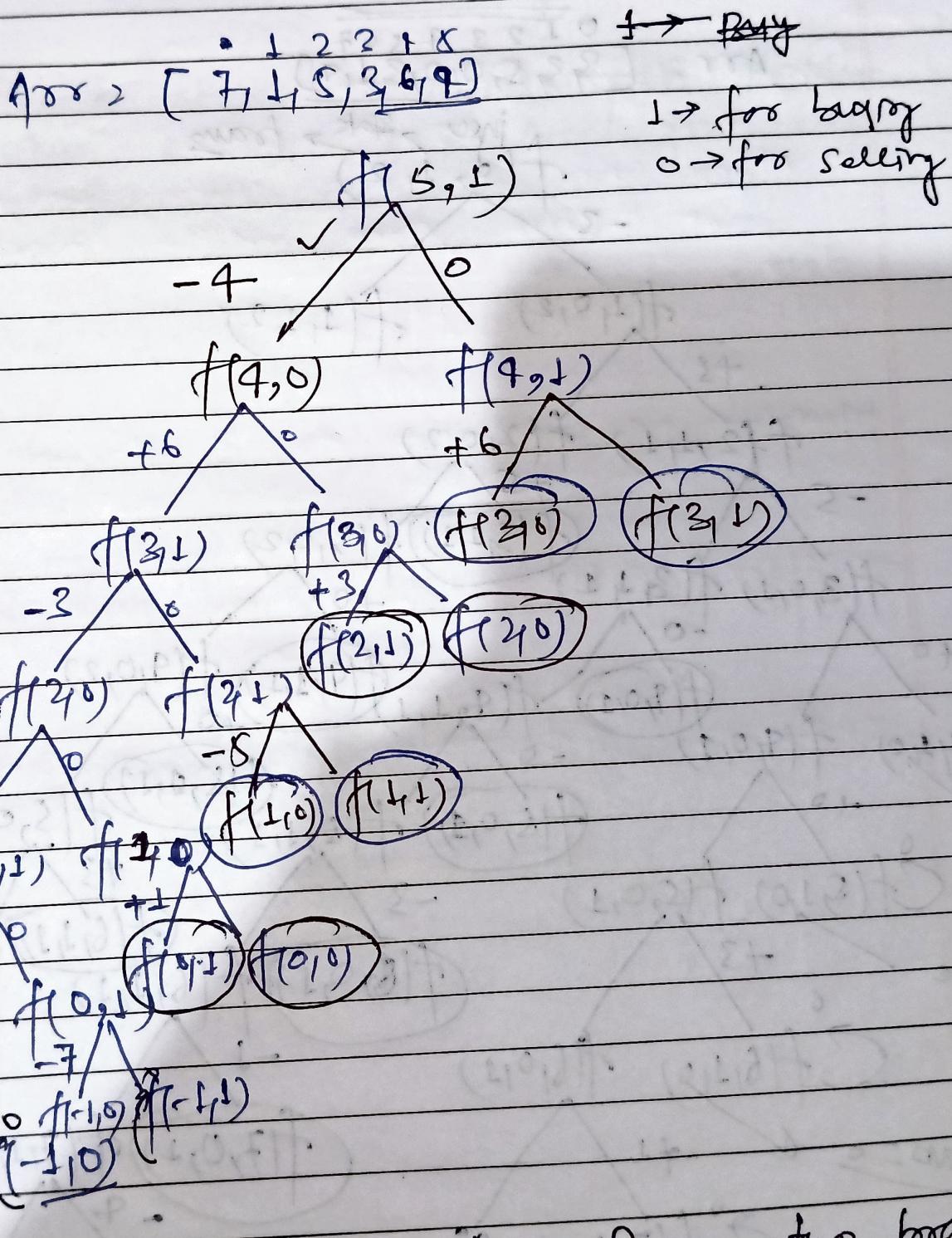
1/3

2/3

```
1 /*
2 You are given an array prices where prices[i] is the price of a given stock on the
3 ith day.
4
5 You want to maximize your profit by choosing a single day to buy one stock and
6 choosing a different day in the future to sell that stock.
7
8 */
9 #include<bits/stdc++.h>
10 using namespace std;
11 int BestTimeToBuyAndSellStock(vector<int>&prices)
12 {
13     int n = prices.size();
14
15     int maxe = prices[0];
16     int mine = prices[0];
17
18     int maxecost = 0;
19
20     for (int i = 0; i < n; i++)
21     {
22         maxe = prices[i];
23
24         maxecost = max(maxecost, maxe - mine);
25
26         mine = min(mine, prices[i]);
27     }
28
29     return maxecost;
30 }
31
32 int main()
33 {
34     int n;
35     cin>>n;
36     vector<int>input(n);
37     for(int i=0;i<n;i++)
38     {
39         cin>>input[i];
40     }
41     int ans=BestTimetoBuyandSellStock(input);
42     cout<<ans;
43 }
44
45 /*
46 time complexity: O(n)
47 space complexity: O(1)
48 */
49
```

36. Buy and Sell Stock II

→ Every stock has two choices either sell or buy



→ Because of two choices I am two branches from every stock

6/28/22, 12:54 PM

bestTimeToBuyAndSellStockii.cpp

```
1 /*
2 You are given an integer array prices where prices[i] is the price of a given stock
3 on the ith day.
4
5 On each day, you may decide to buy and/or sell the stock. You can only hold at most
6 one share of the stock at any time. However, you can buy it then immediately sell it
7 on the same day.
8
9 Find and return the maximum profit you can achieve.
10 */
11 #include<bits/stdc++.h>
12 using namespace std;
13
14 int calculateans(vector<int> &price, int buyorsell, int index, vector<vector<int>>
15 &dp)
16 {
17     if (index == price.size())
18     {
19         return 0;
20     }
21     if (dp[index][buyorsell] != -1)
22     {
23         return dp[index][buyorsell];
24     }
25     if (buyorsell)
26     {
27         int ans = max(-price[index] + calculateans(price, !buyorsell, index + 1, dp),
28 calculateans(price, buyorsell, index + 1, dp));
29         return dp[index][buyorsell] = ans;
30     }
31     else
32     {
33         int ans1 = max(price[index] + calculateans(price, !buyorsell, index + 1, dp),
34 calculateans(price, buyorsell, index + 1, dp));
35
36         return dp[index][buyorsell] = ans1;
37     }
38 }
39
40 int tabulation(vector<int> &price)
41 {
42     int n = price.size();
43     // vector<vector<int>>dp(n+1, vector<int>(2,0));
44     vector<int> ahead(2, 0);
45     vector<int> curr(2, 0);
46     for (int index = n - 1; index >= 0; index--)
47     {
48         for (int buyorsell = 0; buyorsell <= 1; buyorsell++)
49         {
50             if (buyorsell)
51             {
52                 int ans = max(-price[index] + ahead[!buyorsell], ahead[buyorsell]);
53                 curr[buyorsell] = ans;
54             }
55             else
56             {
57                 int ans1 = max(price[index] + ahead[!buyorsell], ahead[buyorsell]);
58                 curr[buyorsell] = ans1;
59             }
60         }
61     }
62 }
```

localhost:4649/?mode=clike

6/28/22, 12:54 PM

bestTimeToBuyAndSellStockii.cpp

```
54     }
55     ahead = curr;
56 }
57
58 return ahead[1];
59 }
60
61 int main()
62 {
63     int n;
64     cin >> n;
65     vector<int> price(n);
66     for (int i = 0; i < n; i++)
67     {
68         cin >> price[i];
69     }
70     vector<vector<int>> dp(n + 1, vector<int>(2, -1));
71     int ans = calculateans(price, 1, 0, dp);
72     int ans1 = tabulation(price);
73     cout << ans << " " << ans1;
74     return 0;
75 }
76
77 /*
78 In recursive + memoization
79     time complexity= O(n*m)
80     space complexity= O(n*m) + recursive Stack
81
82 In dp
83     time complexity= O(n*2)
84     space complexity= O(2)
85 */
86 */
```

1/2

localhost:4649/?mode=clike

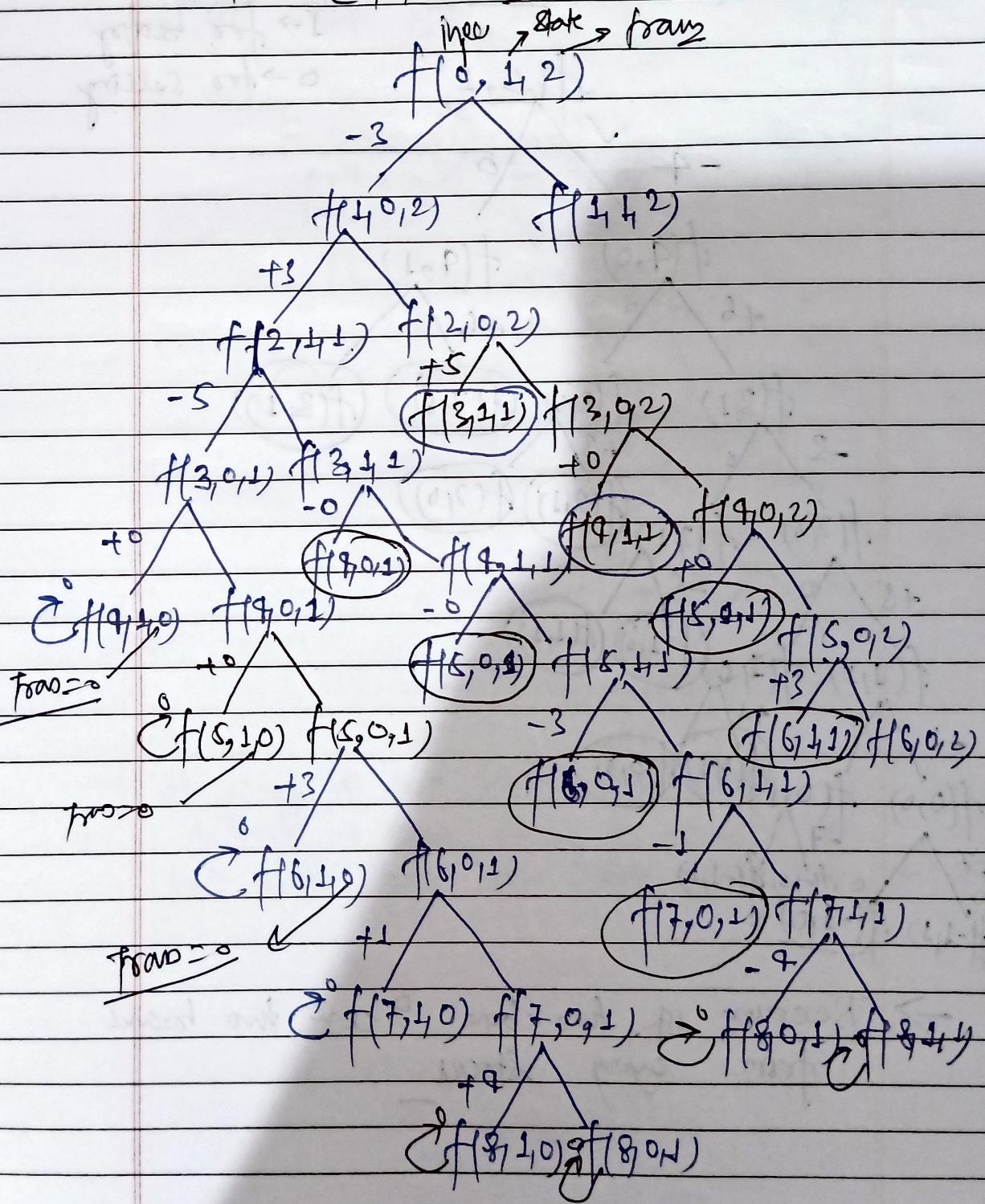
2/2

37.

Boat fence to buy and sell III

→ Where we need to take at most 2 transaction

$$Arr = [3, 8, 5, 0, 1, 3, 4, 7]$$



6/28/22, 12:55 PM

bestTimeToBuyAndSellStockCooldown.cpp

```
1 /*
2 You are given an array prices where prices[i] is the price of a given stock on the
3 ith day.
4 Find the maximum profit you can achieve. You may complete as many transactions as you
5 like (i.e., buy one and sell one share of the stock multiple times) with the
6 following restrictions:
7 After you sell your stock, you cannot buy stock on the next day (i.e., cooldown one
8 day).
9 Note: You may not engage in multiple transactions simultaneously (i.e., you must sell
10 the stock before you buy again).
11 */
12 #include<bits/stdc++.h>
13 using namespace std;
14 int calculateans(vector<int> &price, int buyorsell, int index, vector<vector<int>>
15 &dp)
16 {
17     if (index >= price.size())
18     {
19         return 0;
20     }
21     if (dp[index][buyorsell] != -1)
22     {
23         return dp[index][buyorsell];
24     }
25     if (buyorsell)
26     {
27         int ans = max(-price[index] + calculateans(price, !buyorsell, index + 1, dp),
28 calculateans(price, buyorsell, index + 1, dp));
29         return dp[index][buyorsell] = ans;
30     }
31     else
32     {
33         int ans1 = max(price[index] + calculateans(price, !buyorsell, index + 2, dp),
34 calculateans(price, buyorsell, index + 1, dp));
35
36         return dp[index][buyorsell] = ans1;
37     }
38 }
39
40 int tabulation(vector<int> &price)
41 {
42     int n = price.size();
43     vector<vector<int>> dp(n + 2, vector<int>(2, 0));
44     // space optimization
45     // vector<int>ahead(2,0);
46     // vector<int>curr(2,0);
47     for (int index = n - 1; index >= 0; index--)
48     {
49         for (int buyorsell = 0; buyorsell <= 1; buyorsell++)
50         {
51             if (buyorsell)
52             {
53                 int ans = max(-price[index] + dp[index + 1][!buyorsell], dp[index +
54 1][buyorsell]);
55                 dp[index][buyorsell] = ans;
56             }
57         }
58     }
59 }
```

localhost:4649/?mode=clike

6/28/22, 12:55 PM

bestTimeToBuyAndSellStockCooldown.cpp

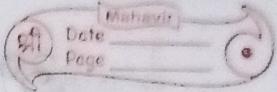
```
1         {
2             int ans1 = max(price[index] + dp[index + 2][!buyorsell], dp[index +
3 1][buyorsell]);
4             dp[index][buyorsell] = ans1;
5         }
6         // ahead=curr;
7     }
8
9     return dp[0][1];
10 }
11 int main()
12 {
13     int n;
14     cin>>n;
15     vector<int>input(n);
16     for(int i=0;i<n;i++)
17     {
18         cin>>input[i];
19     }
20     vector<vector<int>> dp(prices.size(), vector<int>(2, -1));
21     int ans=calculateans(input,0,0,dp);
22     cout<<ans<<endl;
23     int ans1=tabulation(input);
24     cout<<ans1<<endl;
25 }
26 */
27 /* In recursion +memoization
28 time complexity: O(n*2);
29 space complexity: O(n*2)+ recursion stack;
30
31 In tabulation
32     time complexity: O(n*2);
33     space complexity: O(n*2);
34 */
35 
```

1/2

localhost:4649/?mode=clike

2/2

38. Buy and Stock Sell IN



→ In this problem you need to do 1k transactions at most

→ It is almost similar to previous problem just change 2 to 1. And you will get the answer.

6/28/22, 12:56 PM

```
bestTimeToBuyAndSellStockiii.cpp

1 /*
2 You are given an array prices where prices[i] is the price of a given stock on the
3 ith day.
4 Find the maximum profit you can achieve. You may complete at most two transactions.
5
6 Note: You may not engage in multiple transactions simultaneously (i.e., you must sell
7 the stock before you buy again).
8 */
9 #include<bits/stdc++.h>
10 using namespace std;
11
12 int recursive(vector<int> &prices, int index, int trans, int buyorsell,
13 vector<vector<vector<int>>& dp)
14 {
15     if (index == prices.size() or trans == 0)
16     {
17         return 0;
18     }
19
20     if (dp[index][trans][buyorsell] != -1)
21     {
22         return dp[index][trans][buyorsell];
23     }
24
25     if (buyorsell)
26     {
27         int ans = 0;
28         ans = max(-prices[index] + recursive(prices, index + 1, trans, !buyorsell,
29         dp), recursive(prices, index + 1, trans, buyorsell, dp));
30         return dp[index][trans][buyorsell] = ans;
31     }
32     else
33     {
34         int ans = 0;
35         ans = max(prices[index] + recursive(prices, index + 1, trans - 1, !buyorsell,
36         dp), recursive(prices, index + 1, trans, buyorsell, dp));
37         return dp[index][trans][buyorsell] = ans;
38     }
39
40     int tabulation(vector<int> &prices)
41     {
42         int n = prices.size();
43         // vector<vector<vector<int>>> dp(n+1, vector<vector<int>>(3, vector<int>(2, 0)));
44         /* here my matrix size is of n*3*2;
45         we can matrix of size 3*2
46
47         */
48         vector<vector<int>> ahead(3, vector<int>(2, 0));
49         vector<vector<int>> curr(3, vector<int>(2, 0));
50
51         for (int index = n - 1; index >= 0; index--)
52         {
53             for (int trans = 1; trans < 3; trans++)
54             {
55                 for (int buyorsell = 0; buyorsell < 2; buyorsell++)
56                 {
```

6/28/22, 12:56 PM

```
bestTimeToBuyAndSellStockiii.cpp

55         if (buyorsell)
56         {
57             int ans = 0;
58             ans = max(-prices[index] + ahead[trans][!buyorsell], ahead[trans]
59             [buyorsell]);
60             curr[trans][buyorsell] = ans;
61         }
62         else
63         {
64             int ans = 0;
65             ans = max(prices[index] + ahead[trans - 1][!buyorsell],
66             ahead[trans][buyorsell]);
67             curr[trans][buyorsell] = ans;
68         }
69         ahead = curr;
70     }
71
72     return curr[2][1];
73 }
74
75 int main()
76 {
77     int n;
78     cin >> n;
79     vector<int> input(n);
80     for (int i = 0; i < n; i++)
81     {
82         cin >> input[i];
83     }
84     vector<vector<vector<int>>> dp(n + 1, vector<vector<int>>(3, vector<int>(2, -1)));
85     int ans = recursive(input, 0, 3, 1, dp);
86     cout << ans << endl;
87     int ans1 = tabulation(input);
88     cout << ans1 << endl;
89 }
90
91 /*
92 In recursion + memoization
93 time complexity= O(n*3*2);
94 space complexity= O(n*3*2)+ recursive stack
95
96 In tabulation
97     time complexity= O(n*3*2);
98     space complexity= O(3*2);
99 */
```

39. Buy and sell stock with cooldown

→ This problem just similar to Buy and Sell stock II.

Only diff difference

→ After you sell your stock, you can not buy stock on the next day.

~~modification~~ modification to its second version

II

→ After selling the stock just do index+2.

index+2 because of cooldown after selling the stocks

6/28/22, 12:58 PM

```
bestTimeToBuyAndSellStockIV.cpp

1 /*
2 You are given an integer array prices where prices[i] is the price of a given stock
3 on the ith day, and an integer k.
4
5 Find the maximum profit you can achieve. You may complete at most k transactions.
6 Note: You may not engage in multiple transactions simultaneously (i.e., you must
7 sell the stock before you buy again).
8
9 Example 1:
10 Input: k = 2, prices = [2,4,1]
11 Output: 2
12 Explanation: Buy on day 1 (price = 2) and sell on day 2 (price = 4), profit = 4-2 =
13 2.
14 */
15 #include<bits/stdc++.h>
16 using namespace std;
17
18 int recursive(vector<int> &prices, int index, int trans, int buyorsell,
19 vector<vector<vector<int>>& dp)
20 {
21     if (index == prices.size() or trans == 0)
22     {
23         return 0;
24     }
25
26     if (dp[index][trans][buyorsell] != -1)
27     {
28         return dp[index][trans][buyorsell];
29     }
30
31     if (buyorsell)
32     {
33         int ans = 0;
34         ans = max(-prices[index] + recursive(prices, index + 1, trans, !buyorsell,
35         dp), recursive(prices, index + 1, trans, buyorsell, dp));
36         return dp[index][trans][buyorsell] = ans;
37     }
38     else
39     {
40         int ans = 0;
41         ans = max(prices[index] + recursive(prices, index + 1, trans - 1,
42         !buyorsell, dp), recursive(prices, index + 1, trans, buyorsell, dp));
43         return dp[index][trans][buyorsell] = ans;
44     }
45
46     int tabulation(vector<int> &prices, int k)
47     {
48         int n = prices.size();
49
50         vector<vector<int>> ahead(k + 1, vector<int>(2, 0));
51         vector<vector<int>> curr(k + 1, vector<int>(2, 0));
52
53         for (int index = n - 1; index >= 0; index--)
54         {
55             for (int trans = 1; trans < k + 1; trans++)
56             {
57                 if (buyorsell)
58                 {
59                     int ans = 0;
60                     ans = max(-prices[index] + ahead[trans][!buyorsell],
61                     ahead[trans][buyorsell]);
62                     curr[trans][buyorsell] = ans;
63                 }
64                 else
65                 {
66                     int ans = 0;
67                     ans = max(prices[index] + ahead[trans - 1][!buyorsell],
68                     ahead[trans][buyorsell]);
69                     curr[trans][buyorsell] = ans;
70                 }
71             }
72             ahead = curr;
73         }
74     }
75
76     return curr[k][1];
77 }
78
79 int main()
80 {
81     int n;
82     cin>>n;
83     vector<int> prices(n,0);
84
85     for(int i=0; i<n; i++)
86     {
87         cin>>prices[i];
88     }
89     int k;
90     cin>>k;
91
92     vector<vector<vector<int>>> dp(n, vector<vector<int>>(k+1, vector<int>(2,-1)));
93     int ans1= recursive(prices,0,k,1,dp);
94     int ans = tabulation(prices, k);
95     cout<<ans1<<endl;
96     cout<<ans<<endl;
97
98     /* In recursion +memoization
99      time complexity: O(n*k*2);
100     space complexity: O(n*k*2)+ recursion stack;
101
102     In tabulation
103     time complexity: O(n*k*2);
104     space complexity: O(k*2);
105 */
106 }
```

localhost:4649/?mode=clike

6/28/22, 12:58 PM

```
bestTimeToBuyAndSellStockIV.cpp

54 for (int buyorsell = 0; buyorsell < 2; buyorsell++)
55 {
56     if (buyorsell)
57     {
58         int ans = 0;
59         ans = max(-prices[index] + ahead[trans][!buyorsell],
60         ahead[trans][buyorsell]);
61         curr[trans][buyorsell] = ans;
62     }
63     else
64     {
65         int ans = 0;
66         ans = max(prices[index] + ahead[trans - 1][!buyorsell],
67         ahead[trans][buyorsell]);
68         curr[trans][buyorsell] = ans;
69     }
70     ahead = curr;
71 }
72
73 return curr[k][1];
74 }

int main()
75 {
76     int n;
77     cin>>n;
78     vector<int> prices(n,0);
79
80     for(int i=0; i<n; i++)
81     {
82         cin>>prices[i];
83     }
84     int k;
85     cin>>k;
86
87     vector<vector<vector<int>>> dp(n, vector<vector<int>>(k+1, vector<int>(2,-1)));
88     int ans1= recursive(prices,0,k,1,dp);
89     int ans = tabulation(prices, k);
90     cout<<ans1<<endl;
91     cout<<ans<<endl;
92
93     /* In recursion +memoization
94      time complexity: O(n*k*2);
95      space complexity: O(n*k*2)+ recursion stack;
96
97      In tabulation
98      time complexity: O(n*k*2);
99      space complexity: O(k*2);
100 */
101 }
```

1/2

localhost:4649/?mode=clike

2/2

6/28/22, 1:00 PM

bestTimeToBuyAndSellStockWithTrans

```
1 /**
2 * You are given an array prices where prices[i] is the price of a given stock on the
3 * ith day, and an integer fee representing a transaction fee.
4 *
5 * Find the maximum profit you can achieve. You may complete as many transactions as you
6 * like, but you need to pay the transaction fee for each transaction.
7 *
8 * Note: You may not engage in multiple transactions simultaneously (i.e., you must sell
9 * the stock before you buy again).
10 *
11 * Example 1:
12 * Input: prices = [1,3,2,8,4,9], fee = 2
13 * Output: 8
14 * Explanation: The maximum profit can be achieved by:
15 * - Buying at prices[0] = 1
16 * - Selling at prices[3] = 8
17 * - Buying at prices[4] = 4
18 * - Selling at prices[5] = 9
19 * The total profit is ((8 - 1) - 2) + ((9 - 4) - 2) = 8.
20 */
21 using namespace std;
22 int calculateans(vector<int> &price, int buyorsell, int index, vector<vector<int>>
&dp, int fee)
23 {
24     if (index == price.size())
25     {
26         return 0;
27     }
28     if (dp[index][buyorsell] != -1)
29     {
30         return dp[index][buyorsell];
31     }
32     if (buyorsell)
33     {
34         int ans = max(-price[index] + calculateans(price, !buyorsell, index + 1, dp,
fee), calculateans(price, buyorsell, index + 1, dp, fee));
35         return dp[index][buyorsell] = ans;
36     }
37     else
38     {
39         int ans1 = max(price[index] - fee + calculateans(price, !buyorsell, index +
1, dp, fee), calculateans(price, buyorsell, index + 1, dp, fee));
40
41         return dp[index][buyorsell] = ans1;
42     }
43 }
44
45 int tabulation(vector<int> &price, int fee)
46 {
47     int n = price.size();
48     // vector<vector<int>>dp(n+1, vector<int>(2,0));
49     // space optimization
50     vector<int> ahead(2, 0);
51     vector<int> curr(2, 0);
52     for (int index = n - 1; index >= 0; index--)
53     {
```

6/28/22, 1:00 PM

bestTimeToBuyAndSellStockWithTrans

```
54 for (int buyorsell = 0; buyorsell <= 1; buyorsell++)
55 {
56     if (buyorsell)
57     {
58         int ans = max(-price[index] + ahead[!buyorsell], ahead[buyorsell]);
59         curr[buyorsell] = ans;
60     }
61     else
62     {
63         int ans1 = max(price[index] - fee + ahead[!buyorsell],
ahead[buyorsell]);
64         curr[buyorsell] = ans1;
65     }
66 }
67 ahead = curr;
68 }
69
70 return curr[1];
71 }
72
73 int main()
74 {
75     int n;
76     cin >> n;
77     vector<int> input(n);
78     for (int i = 0; i < n; i++)
79     {
80         cin >> input[i];
81     }
82     int fee;
83     cin >> fee;
84     int ans=calculateans(input,1,0,vector<vector<int>>(n+1,vector<int>(2,-1)),fee);
85     cout<<ans<<endl;
86     int ans1=tabulation(input,fee);
87     cout<<ans1<<endl;
88 }
89 /*
90 * In recursion +memoization
91 * time complexity: O(n*2);
92 * space complexity: O(n*2)+ recursion stack;
93
94 In tabulation
95     time complexity: O(n*2);
96     space complexity: O(2);
97 */
98 */
```

Mahavir

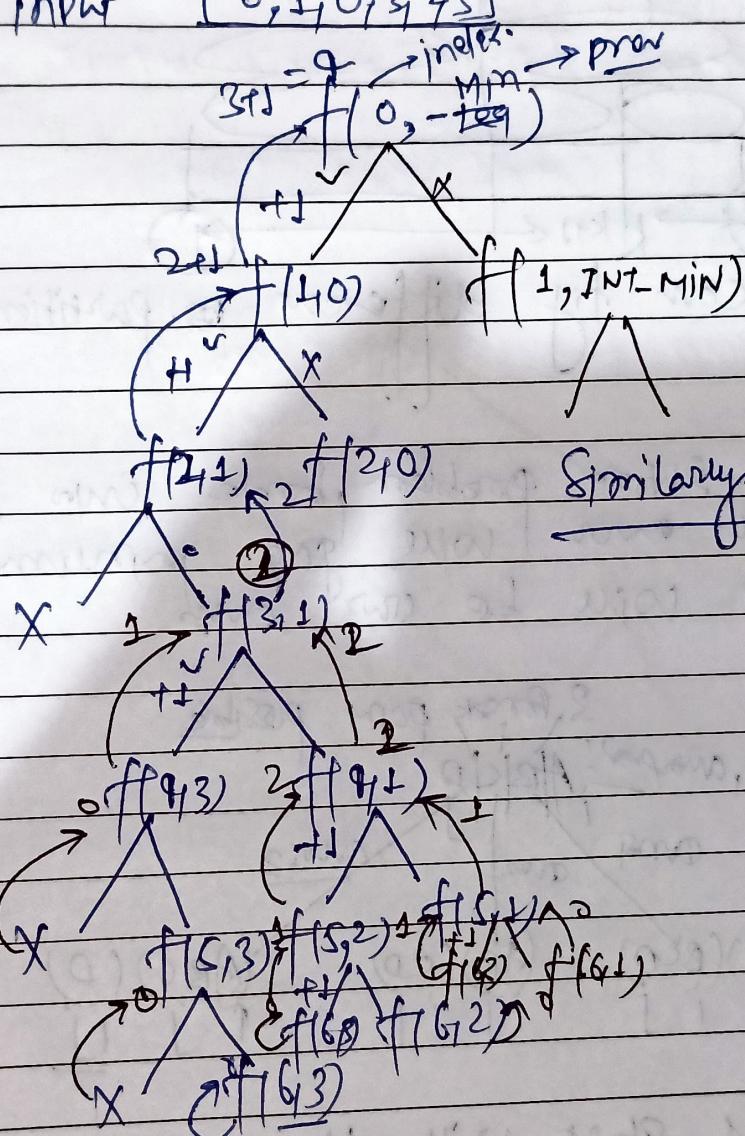
Date _____ Page _____

4.0. Longest Increasing Subsequence

→ There are three methods to solve it.

All three method are mentioned in the code section.

input $\left[\begin{smallmatrix} 0 & 1 & 2 & 3 & 4 & 5 \\ 0 & 4 & 0 & 3 & 4 & 3 \end{smallmatrix} \right]$



Penning method in code section. Please read carefully.

```
1 /*
2 Given an integer array nums, return the length of the longest strictly increasing
3 subsequence.
4 A subsequence is a sequence that can be derived from an array by deleting some or no
5 elements without changing the order of the remaining elements. For example,
6 [3,6,2,7] is a subsequence of the array [0,3,1,6,2,2,7].
7 */
8 #include<bits/stdc++.h>
9 using namespace std;
10 int calculateans(vector<int> &nums, int index, int prev, vector<vector<int>> &dp)
11 {
12     if (index == nums.size())
13     {
14         return 0;
15     }
16     int include = 0;
17     int exclude = 0;
18     if (dp[index][prev + 1] != -1)
19     {
20         return dp[index][prev + 1];
21     }
22     exclude = calculateans(nums, index + 1, prev, dp);
23     if (prev == -1 or nums[index] > nums[prev])
24     {
25         include = 1 + calculateans(nums, index + 1, index, dp);
26     }
27
28     return dp[index][prev + 1] = max(include, exclude);
29 }
30
31 int tabulation(vector<int> &nums)
32 {
33     int n = nums.size();
34     vector<vector<int>> dp(n + 1, vector<int>(n + 1, 0));
35     // here we are shifting the second parameter by 1 right because of
36     // prev=-1 , thats why we are doing +1 in every second parameter in the dp array
37     for (int index = n - 1; index >= 0; index--)
38     {
39         for (int prev = index - 1; prev >= -1; prev--)
40         {
41             int include = 0;
42             int exclude = 0;
43
44             exclude = dp[index + 1][prev + 1];
45             if (prev == -1 or nums[index] > nums[prev])
46             {
47                 include = 1 + dp[index + 1][index + 1];
48             }
49
50             dp[index][prev + 1] = max(include, exclude);
51         }
52     }
53
54     return dp[0][0];
55 }
```

```
56 int arraySolution(vector<int> &nums)
57 {
58     int n = nums.size();
59     vector<int> dp(n, 1);
60     vector<int> parent(n, 0);
61     for (int i = 0; i < n; i++)
62     {
63         parent[i] = i;
64     }
65     dp[0] = 1;
66     int ans = 1;
67     for (int i = 1; i < n; i++)
68     {
69         for (int j = 0; j < i; j++)
70         {
71             if (nums[j] < nums[i])
72             {
73                 dp[i] = max(dp[i], dp[j] + 1);
74             }
75         }
76         ans = max(ans, dp[i]);
77     }
78
79     return ans;
80 }
81
82 void printLcs(vector<int> &nums)
83 {
84     int n = nums.size();
85     vector<int> dp(n, 1);
86     vector<int> parent(n, 0);
87     for (int i = 0; i < n; i++)
88     {
89         parent[i] = i;
90     }
91     dp[0] = 1;
92     int ans = 1;
93     int maxe = 0;
94     int index = -1;
95     for (int i = 1; i < n; i++)
96     {
97         for (int j = 0; j < i; j++)
98         {
99             if (nums[j] < nums[i])
100             {
101                 // dp[i]=max(dp[i],dp[j]+1);
102                 if (dp[j] + 1 > dp[i])
103                 {
104                     dp[i] = dp[j] + 1;
105                     parent[i] = j;
106                 }
107             }
108         }
109         if (dp[i] > maxe)
110         {
111             maxe = dp[i];
112             index = i;
113         }
114         ans = max(ans, dp[i]);
115     }
}
```

```
116     cout << nums[index] << " ";
117
118     while (index != parent[index])
119     {
120         cout << nums[parent[index]] << " ";
121         index = parent[index];
122     }
123     cout << endl;
124 }
125
126
127 int LcsNlogN(vector<int> &nums)
128 {
129     vector<int> dp;
130     int n = nums.size();
131     dp.push_back(nums[0]);
132     for (int i = 1; i < n; i++)
133     {
134         if (nums[i] > dp.back())
135         {
136             dp.push_back(nums[i]);
137         }
138         else
139         {
140             auto it = lower_bound(dp.begin(), dp.end(), nums[i]) - dp.begin();
141             dp[it] = nums[i];
142         }
143     }
144     return dp.size();
145 }
146
147 int lengthOfLIS(vector<int> &nums)
148 {
149
150     int n = nums.size();
151
152     // vector<vector<int>>dp(n, vector<int>(n+1, -1));
153     // int ans= calculateans(nums,0,-1,dp);
154     // int ans= tabulation(nums);
155     // int ans= arraySolution(nums);
156     // printLcs(nums);
157     int ans = LcsNlogN(nums);
158     return ans;
159 }
160
161 int main()
162 {
163     int n;
164     cin>>n;
165     vector<int> input(n);
166     for(int i=0;i<n;i++)
167     {
168         cin>>input[i];
169     }
170
171     cout<<lengthOfLIS(input)<<endl;
172 }
```

6/28/22, 1:04 PM

LongestStringChain.cpp

```
1 /*
2 You are given an array of words where each word consists of lowercase English
letters.
3
4 wordA is a predecessor of wordB if and only if we can insert exactly one letter
anywhere in wordA without changing the order of the other characters to make it equal
to wordB.
5
6 For example, "abc" is a predecessor of "abac", while "cba" is not a predecessor of
"bcad".
7 A word chain is a sequence of words [word1, word2, ..., wordk] with k >= 1, where
word1 is a predecessor of word2, word2 is a predecessor of word3, and so on. A single
word is trivially a word chain with k == 1.
8
9 Return the length of the longest possible word chain with words chosen from the given
list of words.
10
11 Example 1:
12
13 Input: words = ["a", "b", "ba", "bca", "bda", "bdca"]
14 Output: 4
15 Explanation: One of the longest word chains is ["a", "ba", "bda", "bdca"].
16 */
17 #include<bits/stdc++.h>
18 using namespace std;
19 const bool static comp(string &input1, string &input2)
20 {
21     return input1.size() < input2.size();
22 }
23 bool isValid(string &larger, string &smaller)
24 {
25     int nl = larger.size();
26     int ns = smaller.size();
27     if (abs(nl - ns) != 1)
28     {
29         return false;
30     }
31
32     int i = 0;
33     int j = 0;
34     while (i != nl)
35     {
36         if (larger[i] == smaller[j])
37         {
38             i++;
39             j++;
40         }
41         else
42         {
43             i++;
44         }
45     }
46     return j == ns;
47 }
48 int longestStrChain(vector<string> &words)
49 {
50
51     sort(words.begin(), words.end(), comp);
52     int n = words.size();
```

localhost:4649/?mode=clike

6/28/22, 1:04 PM

LongestStringChain.cpp

```
53
54     vector<int> dp(n, 1);
55     int maxe = 0;
56     for (int i = 0; i < n; i++)
57     {
58         for (int j = 0; j < i; j++)
59         {
60             if (isValid(words[i], words[j]))
61             {
62                 if (dp[i] < dp[j] + 1)
63                 {
64                     dp[i] = dp[j] + 1;
65                 }
66             }
67         }
68         maxe = max(maxe, dp[i]);
69     }
70     return maxe;
71 }
72 int main()
73 {
74     int n;
75     cin >> n;
76     vector<string> words(n);
77     for (int i = 0; i < n; i++)
78     {
79         cin >> words[i];
80     }
81     cout << longestStrChain(words) << endl;
82     return 0;
83 }
```

1/2

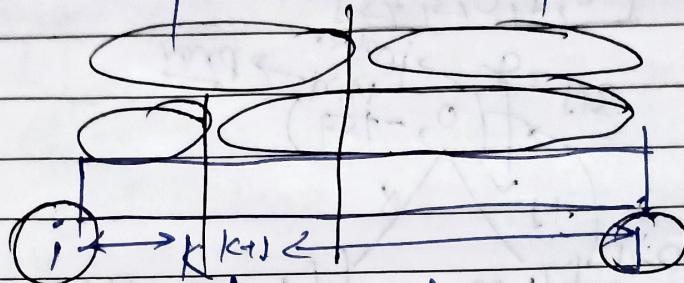
localhost:4649/?mode=clike

2/2

4.1. Matrix chain multiplication (MCM)

→ Partitions DP

→ Solve problem in pattern



We can try different partitions.

→ face above problem, break into smaller which ever will give minimum that will be our aim

3. Break point possible

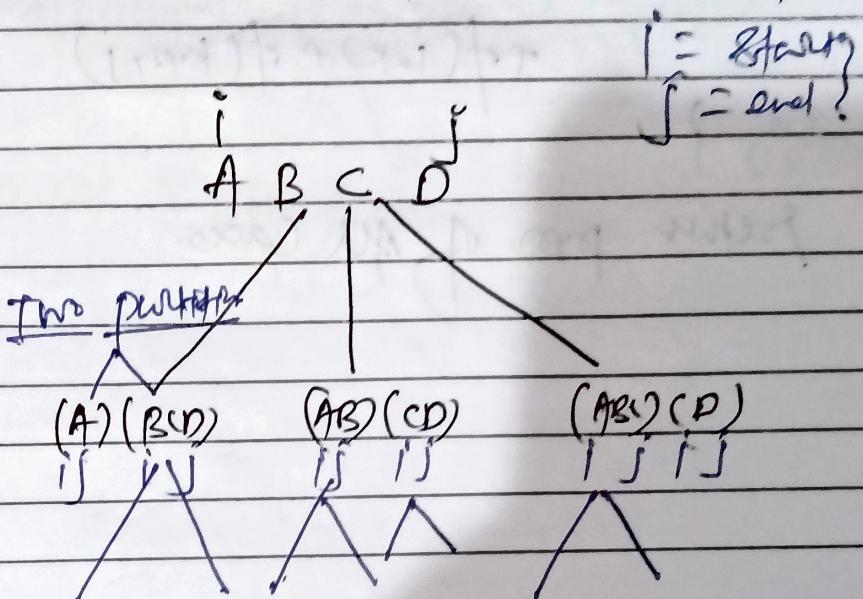
result pos (ans1, ans2),
ans1, ans2, AB|CD

(A)(BCD) (AB)(CD) (ABC)(D)

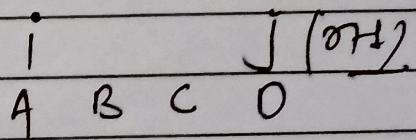
→ Always start with the entire problem
j will be the start point and
will be the end point and k
will be its partition.

Parties

- i) Start with entire block (array of i, j)
- ii) Try all partitions (from a low to try all partition ...)
- iii) Return the best possible for 2 partition



\rightarrow Let suppose you have given array $[10, 20, 20, 90, 50]$



$f(1, 8) \rightarrow$ return the min multiplicans to multiply matrix from $(1 \text{ to } 8)$

Then our loop will run for

$$f(k \rightarrow i \rightarrow j \rightarrow)$$

self calculation

$$\text{steps} = \left[A[j-1] \times \text{row}[k] \times \text{row}[j] \right]$$

$$+ f(i, k) + f(k+1, j)$$

}

Recur form of all cases.

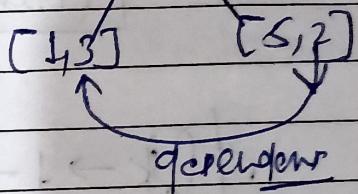
```
1 #include<bits/stdc++.h>
2 using namespace std;
3
4 int calculate(int i, int j, int *input, vector<vector<int>> &dp)
5 {
6     if (i == j)
7     {
8         return 0;
9     }
10
11    if (dp[i][j] != -1)
12    {
13        return dp[i][j];
14    }
15
16    int ans = INT_MAX;
17    for (int k = i; k <= j - 1; k++)
18    {
19        int leftans = calculate(i, k, input, dp);
20        int rightans = calculate(k + 1, j, input, dp);
21        int additionalans = input[i - 1] * input[k] * input[j];
22
23        ans = min(ans, (leftans + rightans + additionalans));
24    }
25    return dp[i][j] = ans;
26}
27 int matrixMultiplication(int n, int arr[])
28 {
29     // code here
30
31     vector<vector<int>> dp(n, vector<int>(n, -1));
32
33     int result = calculate(1, n - 1, arr, dp);
34
35     return result;
36 }
37
38
39 int main()
40 {
41     int n;
42     cin >> n;
43     int arr[n];
44     for (int i = 0; i < n; i++)
45     {
46         cin >> arr[i];
47     }
48     int result = matrixMultiplication(n, arr);
49     cout << result;
50     return 0;
51 }
```

50. Minimum Cost to Cut the Sticks

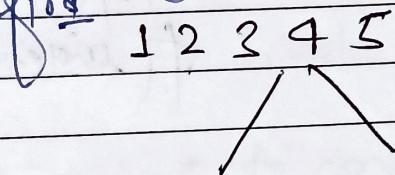
$$\text{arr}[] = [4, 3, 9, 5] \quad n=4$$

→ In this problem we just need to solve subproblem independent of each other.

if our array like $[1, 3, 4, 2]$



for solving independent we need to solve them first.



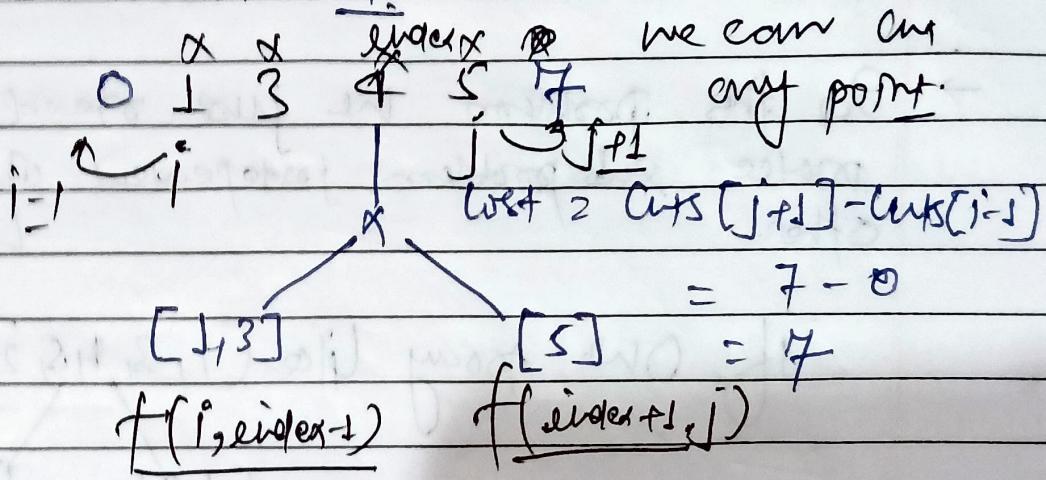
$[1, 2, 3] \quad [5]$

Both independent of each other

→ Now how to find the length of sticks

Cost \Rightarrow length of their stick in which they meet part

for calculating length we will move
0 only $\frac{m}{n}$.



$f(k \rightarrow l \rightarrow m)$

$$\text{cost} = \text{cost}[j+1] - \text{cost}[i-1] + f(i, \text{index}-1) + f(\text{index}+1, j)$$

}

if $(i > j)$
return 0

$$TC = (M^2 \times M) = M^3$$

for i, j for a loop

6/28/22, 1:07 PM

minimumCostToCutTheStick.cpp

```
1 /*
2 Given a wooden stick of length n units. The stick is labelled from 0 to n. For
3 example, a stick of length 6 is labelled as follows:
4
5 Given an integer array cuts where cuts[i] denotes a position you should perform a cut
6 at.
7
8 You should perform the cuts in order, you can change the order of the cuts as you
9 wish.
10
11 The cost of one cut is the length of the stick to be cut, the total cost is the sum
12 of costs of all cuts. When you cut a stick, it will be split into two smaller sticks
13 (i.e. the sum of their lengths is the length of the stick before the cut). Please
14 refer to the first example for a better explanation.
15
16 Return the minimum total cost of the cuts.
17 Example 1:
18 Input: n = 7, cuts = [1,3,4,5]
19 Output: 16
20 Explanation: Using cuts order = [1, 3, 4, 5] as in the input leads to the following
21 scenario:
22
23 The first cut is done to a rod of length 7 so the cost is 7. The second cut is done
24 to a rod of length 6 (i.e. the second part of the first cut), the third is done to a
25 rod of length 4 and the last cut is to a rod of length 3. The total cost is 7 + 6 + 4
26 + 3 = 20.
27 Rearranging the cuts to be [3, 5, 1, 4] for example will lead to a scenario with
28 total cost = 16 (as shown in the example photo 7 + 4 + 3 + 2 = 16).
29 */
30 #include<bits/stdc++.h>
31 using namespace std;
32
33 int calculate(vector<int> &cuts, int start, int end, vector<vector<int>> &dp)
34 {
35     if (start > end)
36     {
37         return 0;
38     }
39     if (dp[start][end] != -1)
40     {
41         return dp[start][end];
42     }
43
44     int ans = INT_MAX;
45     for (int k = start; k <= end; k++)
46     {
47         int leftans = calculate(cuts, start, k - 1, dp);
48         int rightans = calculate(cuts, k + 1, end, dp);
49         int additionalans = cuts[end + 1] - cuts[start - 1];
50         ans = min(ans, leftans + rightans + additionalans);
51     }
52
53     return dp[start][end] = ans;
54 }
55
56 int minCost(int n, vector<int> &cuts)
57 {
58
59 }
```

6/28/22, 1:07 PM

minimumCostToCutTheStick.cpp

```
49 int m = cuts.size();
50 vector<vector<int>> dp(m + 1, vector<int>(n + 1, -1));
51 sort(cuts.begin(), cuts.end());
52 cuts.push_back(n);
53 cuts.insert(cuts.begin(), 0);
54 int ans = calculate(cuts, 1, m, dp);
55
56 return ans;
57 }
58
59 int main()
60 {
61     int n;
62     cin >> n;
63     vector<int> cuts(n);
64     for (int i = 0; i < n; i++)
65     {
66         cin >> cuts[i];
67     }
68     int result = minCost(n, cuts);
69     cout << result;
70     return 0;
71 }
```

Q. Burst Balloons

→ place ~~for~~ sub problem steps independent is important.

$$\text{Ans} \quad [3, 4, 5, 8] \quad n=4$$

$$1 \times 2 \times 1 = 3$$

$$[4, 5, 8]$$

$$1 \times 1 \times 5 = 5$$

$$[5, 8]$$

$$1 \times 5 \times 8 = 40$$

$$[8]$$

$$1 \times 8 \times 1 = 8$$

$$\underline{56}$$

We can burst any balloon at ~~first~~.

We need to maximize it.

$$[b_1, b_2, b_3, b_4, b_5, b_6]$$

X

$$[b_1, b_2, b_3] + [b_4, b_5] + [b_6]$$

subsets

rest

subset

This will not give correct result.

$$\rightarrow \text{first strategy } [b_1, b_2, b_3, b_4, b_5, b_6]$$

X

$$(b_2 \times b_3 \times b_5)$$

depending on b_5

Dependent on each other.

Then we need to remove it independent

→ for that we go from last element

$$\begin{array}{r} x \\ \times 358 \\ \hline 13 \end{array} \quad 1 \times 3 \times 5 = 15$$

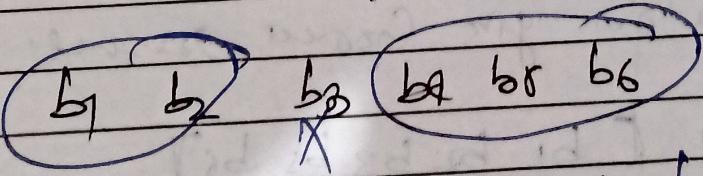
for avoid coordination

$$[3, 5, 8] \rightarrow 3 \times 5 \times 8 = 120$$

$$\begin{array}{r} \times \\ \boxed{3} 87 \\ \hline 1 \times 3 \times 8 = 24 \end{array}$$

last element [8] → $1 \times 8 \times 1 = 8$

Affixing this is last element to solve.



$$a_{i,j} \times a_{[e:i]} \times a_{[j-e]} + f(j, \min\{i-j\}) + \\ f(e+i, j)$$

6/28/22, 1:08 PM

BurstBalloons.cpp

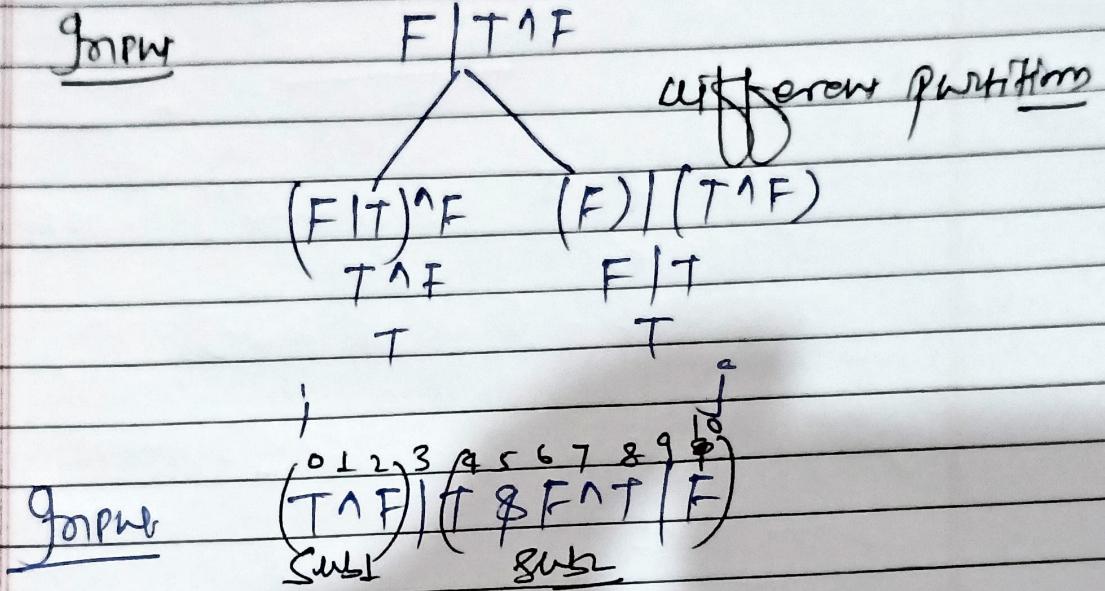
```
1 /*
2 You are given n balloons, indexed from 0 to n - 1. Each balloon is painted with a
3 number on it represented by an array nums. You are asked to burst all the balloons.
4 If you burst the ith balloon, you will get nums[i - 1] * nums[i] * nums[i + 1] coins.
5 If i - 1 or i + 1 goes out of bounds of the array, then treat it as if there is a
6 balloon with a 1 painted on it.
7 */
8 #include<bits/stdc++.h>
9 using namespace std;
10
11 int calculate(vector<int> &nums, int start, int end, vector<vector<int>> &dp)
12 {
13     if (start > end)
14     {
15         return 0;
16     }
17
18     int ans = INT_MIN;
19     // we are starting from bottom that means there will be only one element in the
20     // array and their left and right side will be
21     if (dp[start][end] != -1)
22     {
23         return dp[start][end];
24     }
25     for (int k = start; k <= end; k++)
26     {
27         int leftans = calculate(nums, start, k - 1, dp);
28         int rightans = calculate(nums, k + 1, end, dp);
29
30         int additional = nums[start - 1] * nums[k] * nums[end + 1];
31
32         ans = max(ans, leftans + rightans + additional);
33     }
34     return dp[start][end] = ans;
35 }
36
37 int maxCoins(vector<int> &nums)
38 {
39     int n = nums.size();
40     nums.push_back(1);
41     nums.insert(nums.begin(), 1);
42     vector<vector<int>> dp(n + 1, vector<int>(n + 1, -1));
43     int ans = calculate(nums, 1, n, dp);
44
45     return ans;
46 }
47
48 int main()
49 {
50     int n;
51     cin >> n;
52     vector<int> nums(n);
53     for (int i = 0; i < n; i++)
54     {
55         cin >> nums[i];
```

6/28/22, 1:08 PM

BurstBalloons.cpp

```
56     }
57     int result = maxCoins(nums);
58     cout << result;
59     return 0;
60 }
61
62
63 // This question based on Matrix chain multiplication
64
```

52. Evaluate to true



$f(0, 10)$ and $f(2, 10)$ + Groove calculation

$f(0, 4)$ and $f(6, 10)$ + Groove calculation

We need to try every possible partition

$$f(i, \text{index}-1) + c + f(\text{index}+1, j)$$

\times

$$if (\text{pos}[\text{index}] == \$)$$

$$\text{Total way} = \text{ways}$$

6/28/22, 1:09 PM

EvaluateBooleanExpressionToTrue.cpp

```
1 /*
2 Given an expression, A, with operands and operators (OR , AND , XOR), in how many
3 ways can you evaluate the expression to true, by grouping in different ways?
4 Operands are only true and false.
5
6 Return the number of ways to evaluate the expression modulo 103 + 3.
7
8 Input Format:
9
10 The first and the only argument of input will contain a string, A.
11
12 The string A, may contain these characters:
13   '|' will represent or operator
14   '&' will represent and operator
15   '^' will represent xor operator
16   'T' will represent true operand
17   'F' will false
18 */
19 #include <bits/stdc++.h>
20 using namespace std;
21 long long calculate(string input, int start, int end, int istrue,
22 vector<vector<vector<long long>>> &dp)
23 {
24     if (start > end)
25     {
26         return 0;
27     }
28     if (start == end)
29     {
30         if (istrue)
31         {
32             return input[start] == 'T';
33         }
34         else
35         {
36             return input[start] == 'F';
37         }
38     }
39     if (dp[start][end][istrue] != -1)
40     {
41         return dp[start][end][istrue];
42     }
43     int ways = 0;
44     long long mode = 1e3 + 3;
45     for (int k = start + 1; k <= end - 1; k += 2)
46     {
47         long long lefttrue = calculate(input, start, k - 1, 1, dp);
48         long long leftfalse = calculate(input, start, k - 1, 0, dp);
49         long long righttrue = calculate(input, k + 1, end, 1, dp);
50         long long rightfalse = calculate(input, k + 1, end, 0, dp);
51
52         // appan calculation
53         if (input[k] == '&')
54         {
55             if (istrue)
56                 ways = (ways + (lefttrue * righttrue) % (mode)) % (mode);
57             else
58             {
```

6/28/22, 1:09 PM

EvaluateBooleanExpressionToTrue.cpp

```
58             ways = (ways + (lefttrue * rightfalse) % (mode) + (leftfalse * righttrue) % (mode) + (leftfalse * rightfalse) % (mode)) % (mode);
59         }
60     }
61     else if (input[k] == '|')
62     {
63         if (istrue)
64         {
65             ways = (ways + (lefttrue * righttrue) % (mode) + (lefttrue * rightfalse) % (mode) + (leftfalse * righttrue) % (mode)) % (mode);
66         }
67         else
68         {
69             ways = (ways + (leftfalse * rightfalse) % (mode)) % (mode);
70         }
71     }
72     else
73     {
74         if (istrue)
75         {
76             ways = (ways + (lefttrue * rightfalse) % (mode) + (leftfalse * righttrue) % (mode)) % (mode);
77         }
78         else
79         {
80             ways = (ways + (lefttrue * righttrue) % (mode) + (leftfalse * rightfalse) % (mode)) % (mode);
81         }
82     }
83 }
84 return dp[start][end][istrue] = ways;
85 }
86
87 int cntrue(string input)
88 {
89
90     long long n = input.size();
91
92     vector<vector<vector<long long>>> dp(n + 1, vector<vector<long long>>(n + 1,
93     vector<long long>(2, -1)));
94
95     long long ans = calculate(input, 0, n - 1, 1, dp);
96
97     return ans;
98 }
99
100 int main()
101 {
102     int n;
103     cin >> n;
104     string input;
105     cin >> input;
106     int result = cntrue(input);
107     cout << result;
108     return 0;
109 }
```

6/28/22, 1:11 PM

PalindromicPartition.cpp

```
1 /*
2 Given a string s, partition s such that every substring of the partition is a
3 palindrome.
4 Return the minimum cuts needed for a palindrome partitioning of s.
5 Example 1:
6 Input: s = "aab"
7 Output: 1
8 Explanation: The palindrome partitioning ["aa","b"] could be produced using 1 cut.
9 */
10 #include<bits/stdc++.h>
11 using namespace std;
12
13 bool ispalindrome(string &input, int start, int end)
14 {
15     while (start < end)
16     {
17         if (input[start] != input[end])
18         {
19             return false;
20         }
21         start++;
22         end--;
23     }
24     return true;
25 }
26
27 int calculate(string &input, int start, vector<int> &dp)
28 {
29     if (start == input.size())
30     {
31         return 0;
32     }
33
34     int ans = INT_MAX;
35     if (dp[start] != -1)
36     {
37         return dp[start];
38     }
39
40     for (int k = start; k < input.size(); k++)
41     {
42         if (ispalindrome(input, start, k))
43         {
44             int nextcall = calculate(input, k + 1, dp);
45             int ans1 = 1 + nextcall;
46             ans = min(ans, ans1);
47         }
48     }
49     dp[start] = ans;
50 }
51
52 int minCut(string s)
53 {
54     int n = s.size();
55     vector<int> dp(n + 1, -1);
```

localhost:4649/?mode=clike

6/28/22, 1:11 PM

PalindromicPartition.cpp

```
59     int result = calculate(s, 0, dp);
60
61     return result - 1;
62 }
63
64 int main()
65 {
66     string s;
67     cin >> s;
68     int result = minCut(s);
69     cout << result;
70     return 0;
71 }
```

1/2

localhost:4649/?mode=clike

2/2

6/28/22, 1:12 PM

```
PartitionArrayMaximumSum.cpp

1 /*
2 Given an integer array arr, partition the array into (contiguous) subarrays of length
3 at most k. After partitioning, each subarray has their values changed to become the
4 maximum value of that subarray.
5
6 Example 1:
7
8 Input: arr = [1,15,7,9,2,5,10], k = 3
9 Output: 84
10 Explanation: arr becomes [15,15,15,9,10,10,10]
11 */
12 #include<bits/stdc++.h>
13 using namespace std;
14 int calculate(vector<int> &arr, int start, int k, vector<int> &dp)
15 {
16     if (start == arr.size())
17     {
18         return 0;
19     }
20     int ans = INT_MIN;
21
22     int maxe = INT_MIN;
23     int len = 0;
24     if (dp[start] != -1)
25     {
26         return dp[start];
27     }
28     for (int k1 = start; k1 < min(start + k, (int)arr.size()); k1++)
29     {
30         maxe = max(maxe, arr[k1]);
31         len++;
32         int nextcall = calculate(arr, k1 + 1, k, dp);
33
34         ans = max(ans, nextcall + len * maxe);
35     }
36
37     return dp[start] = ans;
38 }
39
40 int maxSumAfterPartitioning(vector<int> &arr, int k)
41 {
42
43     int n = arr.size();
44     vector<int> dp(n, -1);
45     int ans = calculate(arr, 0, k, dp);
46
47     return ans;
48 }
49
50 int main()
51 {
52     int n;
53     cin >> n;
54     vector<int> arr(n);
55     for (int i = 0; i < n; i++)
56     {
```

6/28/22, 1:12 PM

```
PartitionArrayMaximumSum.cpp

57         cin >> arr[i];
58     }
59     int k;
60     cin >> k;
61     int result = maxSumAfterPartitioning(arr, k);
62     cout << result;
63 }
```