Final Notes For DSA

DP on Squares:-

Q1 Maximum Rectangle Area with All 1's

Ans:-

```
class Solution {
public:
   int largestRectangleArea(vector<int>& arr) {

        int n=arr.size();

        vector<int>v1(n,-1);
        vector<int>v2(n,n);
        stack<int>st,st1;
        // st.push(0);
        for(int i=0;i<n;i++){

         if(st.size()!=0 && arr[i]>arr[st.top()]){
                 v1[i]=st.top();
            }
            else if(st.size()!=0 && arr[i]<=arr[st.top()]){
                while(st.size()!=0 &&
arr[i]<=arr[st.top()]){
                    st.pop();
                }
                if(st.size()==0){
                    v1[i]=-1;
                }
                else{
                    v1[i]=st.top();
                }
            }
            st.push(i);
        }
        st=st1;
           for(int i=n-1;i>=0;i--){
```

```
        if(st.size()!=0 && arr[i]>arr[st.top()]){
                v2[i]=st.top();
            }
            else if(st.size()!=0 && arr[i]<=arr[st.top()]){
                while(st.size()!=0 &&
arr[i]<=arr[st.top()]){
                    st.pop();
                }
                if(st.size()==0){
                    v2[i]=n;
                }
                else{
                    v2[i]=st.top();
                }
            }
            st.push(i);
        }
        int d=0;
        for(int i=0;i<n;i++){
               d=max((i-v1[i]+v2[i]-i-1)*arr[i],d);
        }
        return d;

    }
    int maximalRectangle(vector<vector<char>>& arr) {
        int n=arr.size();
        int m=arr[0].size();
        int d=0;
        vector<int>v(m,0);
        for(int i=0;i<n;i++){
            for(int j=0;j<m;j++){
                  if(arr[i][j]=='1')
{
    v[j]+=1;
}
else{
    v[j]=0;
```

```
}
      }
      d=max(largestRectangleArea(v),d);
        }
        return d;
    }
};
```

Here the pre-requisites to solve the ques is , to know about how to solve the question of “Largest rectangle in a histogram” and this question is solved by using this logic:-
pehle left se wo index dhundo jo ki current index ki value se chota h , agr aisa koi bhi index nhi h to whn (jo vector v1 banaya h) pe -1 ki value rakhdo. Similarly ek dusra vector banao v2 , aur whn pe store karo ki right side se kn sa index h jo ki current index ki value se chota h , agr nhi h to uss index ke lie vector me ans store karo n(size of arrar).

Ab to total length of rectange ke liye, for loop lagao 0 se n tk , left side se kitna part required rectangle ke liye hoga wo niklega isse => i-v1[i] which also include the width of current index (every index is a width of 1) ab right side ke liye => v2[i]-i-1 , isme current index ki width ko consider nhi krna h.

Waise maine ye kia , tum koi bhi algo laga skte ho jis left side kitna length le jo ki rectangle ka part h aur

right se kitna length rectangle ka part h.
complete algo:-

```
class Solution {
public:
    int largestRectangleArea(vector<int>& arr) {

        int n=arr.size();

        vector<int>v1(n,-1);
        vector<int>v2(n,n);
        stack<int>st,st1;
        // st.push(0);
        for(int i=0;i<n;i++){

         if(st.size()!=0 && arr[i]>arr[st.top()]){
                 v1[i]=st.top();
            }
            else if(st.size()!=0 && arr[i]<=arr[st.top()]){
                while(st.size()!=0 &&
arr[i]<=arr[st.top()]){
                    st.pop();
                }
                if(st.size()==0){
                    v1[i]=-1;
                }
                else{
                    v1[i]=st.top();
                }
            }
            st.push(i);
        }
        st=st1;
           for(int i=n-1;i>=0;i--){

         if(st.size()!=0 && arr[i]>arr[st.top()]){
                 v2[i]=st.top();
            }
```

```
            else if(st.size()!=0 && arr[i]<=arr[st.top()]){
                while(st.size()!=0 &&
arr[i]<=arr[st.top()]){
                    st.pop();
                }
                if(st.size()==0){
                    v2[i]=n;
                }
                else{
                    v2[i]=st.top();
                }
            }
            st.push(i);
        }
        int d=0;
        for(int i=0;i<n;i++){
               d=max((i-v1[i]+v2[i]-i-1)*arr[i],d);
        }
        return d;

    }
};
```

Q2 Count the square submatrices containing all 1's ?

Ans:- here first we have to create an 2D dp array of the same size as of the array. Now fill the first row and aolum as same as the given matrix. Now start filling from the element present in second row and 2nd column , now filling is done by this formula:-

Min(dp[i-1][j],dp[[i-1,j-1],dp[i][j-1])+1;
after creating DP array just add all the element fro that

array, and that will be your answer.

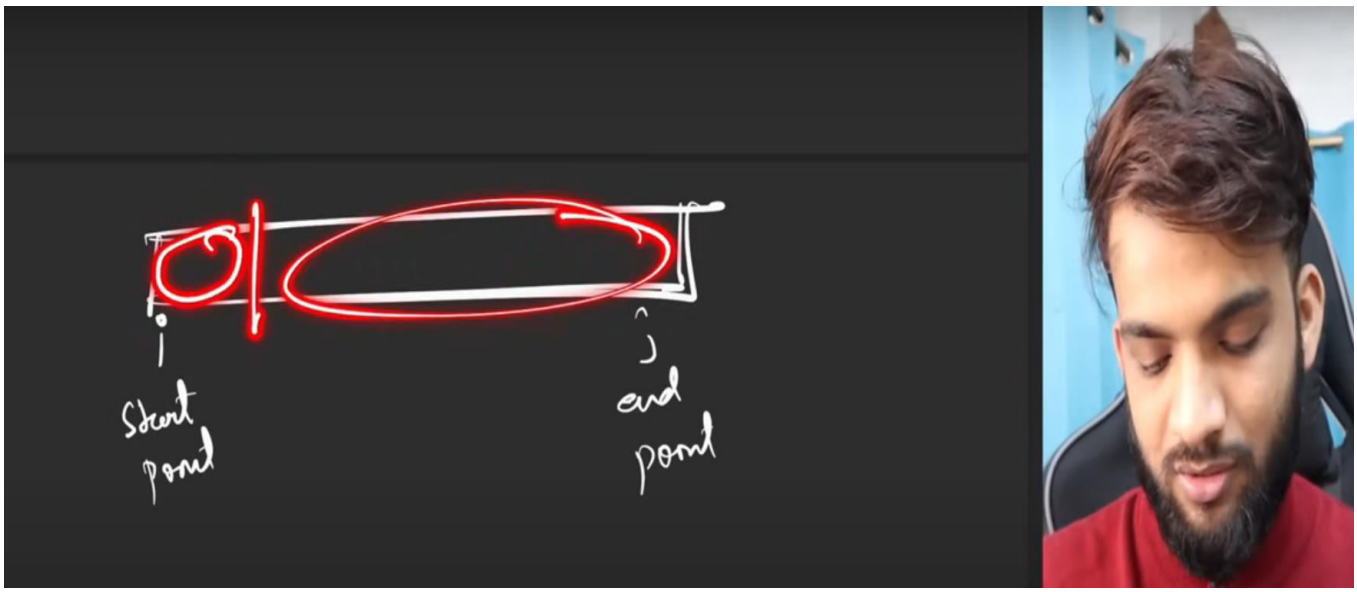
code:-

```
class Solution {
public:
    int countSquares(vector<vector<int>>& arr) {
        int n=arr.size();
        int m=arr[0].size();
        vector<vector<int>> dp(n, vector<int>(m, 0));

    for (int j = 0; j < m; j++) dp[0][j] = arr[0][j];
    for (int i = 0; i < n; i++) dp[i][0] = arr[i][0];

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

    int sum = 0;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            sum += dp[i][j];
        }
    }
    return sum;
    }
};
```

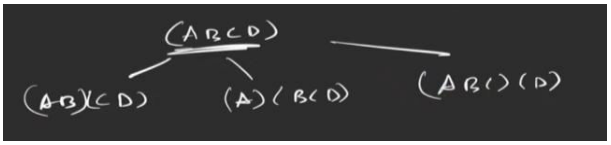
Matrix Chain Multiplication|Partition DP:-

Whenever there a multiple ways to solve the question so that we get a different answer or we have a choice to make partitions then we use partition dp.

Rules for Solving MCM:-

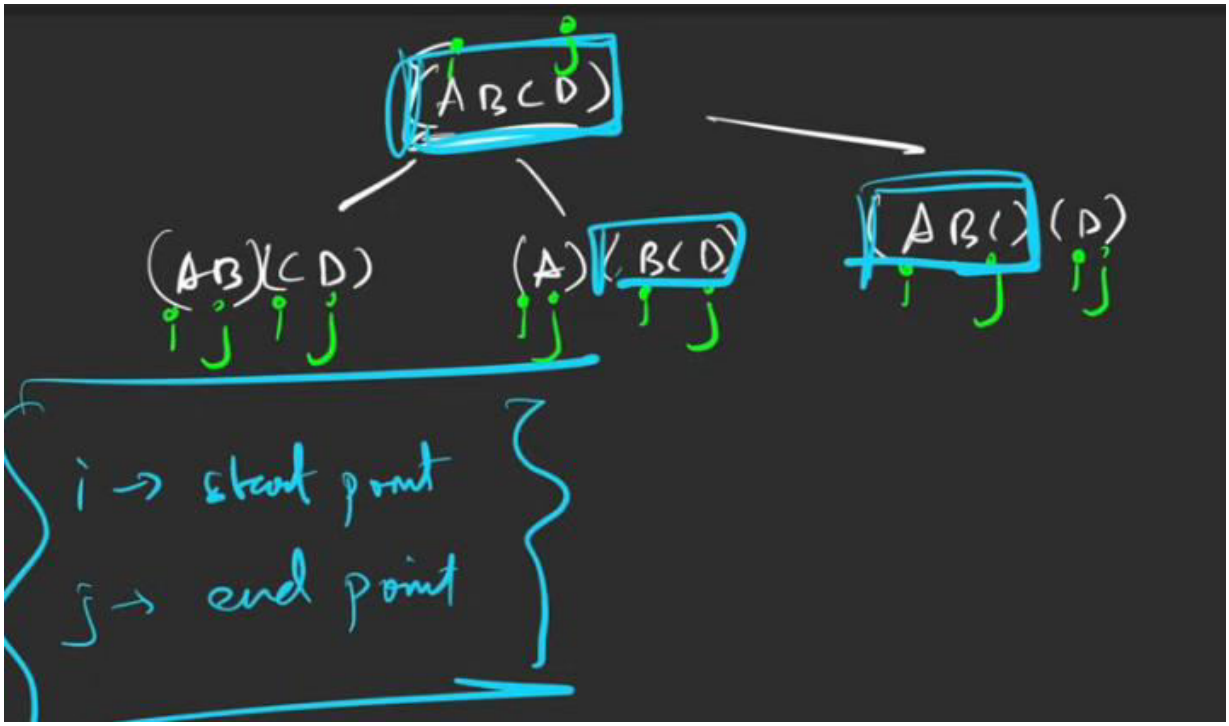


1) An would be given where we have to put partitions, and then solve the particular partition.
2) We have to mark two points for the start and end points.
3) A given array of size N resembles the size of N-1 matrices.
4) Start with the entire array, which means give me the answer without making any partitions.
5) Now from step 4 break the problem into smaller problems.

6) 

The array contains while ABCD now ABCD is broken down into the smaller problems.

7) Run a loop to try all the partiotions.



8) Return the ans of the best two partitions means the left and right sides of a partition.

Q1 Matrix Chain Multiplication?

Ans:-

```
int fn(vector<vector<int>>&dp,int i,int j,int arr[],int n){
    if(i==j){
        return 0;
    }
    else if(dp[i][j]!=-1){
        return dp[i][j];
    }
    else{
        int ans=1e9;
```

```
        for(int p=i;p<j;p++){
            // temp ans1 from i to p and temp ans2 from p+1
to j.
            int d=fn(dp,i,p,arr,n)+fn(dp,p+1,j,arr,n)+arr[i-
1]*arr[p]*arr[j];
            ans=min(d,ans);
        }
        return dp[i][j]=ans;
    }
}
    int matrixMultiplication(int n, int arr[])
    {
        // code here
        vector<vector<int>>dp(n+1,vector<int>(n+1,-1));
        return fn(dp,1,n-1,arr,n);
    }
```

Here taking i from 1 not from 0 and Ai represents a matrix, and if i==0 then this matrix won't have dimensions because of arr[-1] is absurd.

Q2 Minimum cost to cut the stick?

Ans:-

```
class Solution {

public:

int fn(vector<vector<int>>&dp,vector<int> arr,int i,int j){
   if(i>j){
    return 0;
   }
  else if(dp[i][j]!=-1){
    return dp[i][j];
   }

    int ans=1e9;
    for(int p=i;p<=j;p++){
```

```
int mini=fn(dp,arr,p+1,j)+fn(dp,arr,i,p-1)+arr[j+1]-arr[i-
1];
ans=min(mini,ans);
    }
    return dp[i][j]=ans;

}
    int minCost(int n, vector<int>& arr) {
        int k=arr.size();
        arr.insert(arr.begin(),0);
        arr.push_back(n);
        sort(arr.begin(),arr.end());
        vector<vector<int>>dp(k+1,vector<int>(k+1,-1));
        return fn(dp,arr,1,k);
    }
};
```

isme bus arr[j+1]-arr[i-1] wali cheez pata krlo aur partition kisme lagana h bus hogya partition dp.

Q3 Burst Ballon?

Ans:- here we have used a bottom-up approach like there is a single ballon , then two then 3 ,,, and so on,,,

```
class Solution {
public:

int fn(vector<int>& arr,vector<vector<int>>&dp,int i,int j){
    if(i>j){
        return 0;
    }
    else if(dp[i][j]!=-1){
        return dp[i][j];
    }
    else{
        int ans=-1e9;
          for(int k=i;k<=j;k++){
```

```
                int mini=fn(arr,dp,k+1,j)+fn(arr,dp,i,k-
1)+arr[i-1]*arr[k]*arr[j+1];
                ans=max(ans,mini);
            }
            return dp[i][j]=ans;
    }
}
    int maxCoins(vector<int>& arr) {

        int n=arr.size();
        arr.insert(arr.begin(),1);
        arr.push_back(1);
        vector<vector<int>>dp(n+1,vector<int>(n+1,-1));
        return fn(arr,dp,1,n);


    }
};
```

Q4 Partition array for maximum sum?

Ans:-

```
class Solution {

public:
int fn( vector<int>&dp,vector<int>& arr, int k,int p,int n){
    if(p==n){
        return 0;
    }
    if(dp[p]!=-1){
        return dp[p];
    }
    else{
        int maxi=-1e9,l=0,d=-1e9;
        for(int i=p;i<min(p+k,n);i++){
            l++;
            d=max(d,arr[i]);
            int mini=(l*d)+fn(dp,arr,k,i+1,n);
```

```
            maxi=max(maxi,mini);
        }
        return dp[p]=maxi;
    }

}
    int maxSumAfterPartitioning(vector<int>& arr, int k) {
        int n=arr.size();
        vector<int>dp(n+1,-1);
        return fn(dp,arr,k,0,n);

    }
};
```

Q5 Palindrome Partitioning?

Ans:-

```
class Solution {

public:
    int dp[2001][2001];

    int helper(string &s, int i,int j){
        if(i>=j || isPallindrome(s,i,j)){
            dp[i][j] = 0;
            return 0;
        }

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

        int mn = INT_MAX;

        for(int k = i; k<=j; k++){
            if (isPallindrome(s,i,k)){
                mn = min(mn,1+helper(s,k+1,j));
            }
```

```
        }
        return dp[i][j] = mn;


    }
    bool isPallindrome(string &s,int i , int j){
        while(i<=j){
            if (s[i]!=s[j]){
                return false;
            }
            else{
                j--;
                i++;
            }
        }
        return true;
    }

    int minCut(string s) {
        memset(dp,-1, sizeof(dp));
        return helper(s,0,s.length()-1);
    }
};
```

Format for MCM:-

1) We are given an array or strings.
2) We have to take two variables i and j for start and end respectively.
3) Then we have to take a variable k which moves from i to j which is depicted as a partition point.

4) And from the two partitions we get our temporary answers.
5) Taking at which I and j start and from where k will iterate is a tedious task.
6) Base condition is always like if(i>j){return 0;}.
7) Temporary ans will consist ans of two partitions from i to k and from k+1 to j.
8) Then final ans={any function like min/max}(temporary ans,final ans); inside the for loop only.
9) Then return the final answer out of the for loop.

In the case of MCM, we have to take i from 1 not from 0 because Ai defines the ith matrix of size arr[i-1]*arr[i] that's why we have to take i from 1.

In base condition when i==j then return 0, as we are considering there is only a single element/matrix and a single matrix can't be there for matrix multiplication or we have to make matrix multiplications from matrix I to matrix j but if matrix I and matrix j are the same then there would not be any possible multiplication so we have to return 0.

We are solving for I to k and k+1 to j , now if we take k till j , then one ans I to k will give the matrices but from k+1 to j , here k+1 will exceed j and there is a case

where k+1 exceed the array . so we will take k till j-1, as from k+1 to j , there may a case occur of k+1==j, which can be solved. Basically k+1 is depicting as a starting point for next recursive call and starting point cant be outside of array.

To calculate the cost of current matrix multiplication sets like cost of multiplying (AD)*(CD) , is arr[i-1]*arr[k]*arr[j].

Longest Increasing Subsequence

Q1 Longest Increasing Subsequence?

Ans:- first I have tried this :-

```
class Solution {
public:

int fn(vector<int> arr,vector<vector<int>>&dp,int a,int b){
    int n=arr.size();
    if(b==n){
        return 0;
    }
    else if(dp[a][b]!=-1){
        return dp[a][b];
    }
    else{
        int p=-1e9,q=-1e9;
        if(arr[b]>arr[a] || b==a){
            p=1+fn(arr,dp,b,b+1);
        }

        q=fn(arr,dp,a,b+1);
```

```
        return dp[a][b]=max(p,q);
    }
}

    int lengthOfLIS(vector<int>& arr) {
        int n=arr.size();
        vector<vector<int>>dp(n,vector<int>(n,-1));
        return fn(arr,dp,0,0);

    }
};
```

I this approach , I have taken arr[b]>arr[a] , now this condition is correct but it counts only 1 element in the case where we encounter arr[b]>arr[a] , but there should be 2 elements hence this approach was wrong. So I tried this:-

```
class Solution {
public:

int fn(vector<int>& v,vector<vector<int>>&dp,int b,int x,int
n){
    if(b<0){
        return 0;
    }
    else if(dp[b][x-1]!=-1){
        return dp[b][x-1];
    }
    else{
        int p=0,q=0;
        if( x==n || v[x]>v[b]){
            p=1+fn(v,dp,b-1,b,n);
        }
```

```
            q=fn(v,dp,b-1,x,n);

        return dp[b][x-1]=max(p,q);

    }
}
    int lengthOfLIS(vector<int>& v) {
        int n=v.size();
        vector<vector<int>>dp(n,vector<int>(n+1,-1));
     fn(v,dp,n-1,n,n);
    return dp[n-1][n-1];
    }
};
```

Here I have taken a variable x, which starts from n and if there is a case where arr[x]>arr[b] ,we still increment 1 but the arr[b]th is already calculated when x was n. so this approach resolves the ambiguity. Rest approach was same as take and not take approach. And the storage is taken as dp[v][x-1] see, store in whatever manner you want to.

Q2 Printing LIS?

Ans:-

1D DP

Q1 Climbing Stairs?

Ans:- In this question, we have to determine the maximum no. of ways in which we can go to any no. of stairs by climbing to 1 or 2 stairs ahead.

So in this, we have to care about the maximum no. of ways to go to any stair x which lies between 1 to n. here we do not care about how we reach that particular stair x, which means we do not have to store that by climbing 1 or 2 we have reached there. So the only thing we care about is the no. of ways we can get to that stair x.

So the DP array would be 1D.

```
class Solution {
public:
```

```
int fn(vector<int>&dp,int n,int a){
    if(a<=2){
        return dp[a]=a;
    }
    else if(dp[a]!=-1){
        return dp[a];
    }
    else{
        int p=0,q=0;
     if(a-2>=0){
         p=fn(dp,n,a-2)*1;
    }
    if(a-1>=0){
        q=fn(dp,n,a-1)*1;
    }
    return dp[a]=p+q;
    }
}
    int climbStairs(int n) {
        vector<int>dp(n+1,-1);
        return fn(dp,n,n);
    }
};
```

in this question we have to go to stair x , and we can go to stair x from stair (x-2) and (x-1) , as we know my mathematics when we have to calculate the number of ways to go from point A to point B , we do this:-

(no. of ways to go point A)*(number of ways to go point B) so we get the total no. of ways

so here point A is either (x-2) or (x-1) and no. of ways to go from point A to point B is 1.

Then we have to add the no. of ways from (x-2) and (x-1) as the no. of ways to go from (x-1) and (x-2) to x is independent.

We created 1D dp , and stores the value, there might be a question what if the values are updated later, or what if a new maximum is there? So there would not be a case of update because dp[1] and dp[2] is fixed so as dp[3] if we have these fixed values, if dp[3] is fixed and we know that dp[3] represent the maximum values. So when we move on to dp[4] = d[3]+dp[2] . hence dp[4] is also fixed which cant be updated by other no. because it consist of maximum value. Hence we do not need 2D dp. 2D dp is needed when want to store the value for particular index and the method by which we get this value. But here in 1D we do not care about by which method we get this value, we have to store the value.

Q2 Frog Jump?

```
#include <bits/stdc++.h>
int fn(vector<int> &h,vector<int> &v,int i,int n){
if(i==0){
    return v[i]=0;
}
  else if(i==1){
          return v[i]=abs(h[i]-h[0]);
      }
      else if(i==2){
          return v[i]=min(abs(h[i]-h[0]),abs(h[i]-
h[1])+abs(h[1]-h[0]));
      }
else if(v[i]!=-1){
    return v[i];
}


else{


     return v[i]=min((abs(h[i]-h[i-1])+fn(h,v,i-
1,n)),(abs(h[i]-h[i-2])+fn(h,v,i-2,n)));

}



}
int frogJump(int n, vector<int> &h)
{
    vector<int>v(n,-1);
    fn(h,v,n-1,n);
    return v[n-1];
}
```

Whenever we approach for 1D DP , we already know the solutions for 2 to 3 cases ,

and these solutions are fixed which conclude to give fixed solutions for further cases too like, there would not be a case where dp[4] will have two solutions.

And lastly we write function which is need to be calculated.

Q3 Frog Jump Upto K distances?

Ans:-

```
int fn(vector<int>&dp,vector<int> &arr,int a,int k){
      if(a==0){
          return dp[a]=0;
      }
      else if(dp[a]!=-1){
          return dp[a];
      }
      else{
          int p=1e9;
          for(int i=k;i>=1;i--){
              if(a-i>=0){
                  p=min(p,abs(arr[a]-arr[a-i])+fn(dp,arr,a-
i,k));
              }

          }
          return dp[a]=p;

      }
  }

    int minimizeCost(vector<int>& arr, int n, int k) {
        // Code here
        vector<int>dp(n,-1);
        fn(dp,arr,n-1,k);
```

```
        return dp[n-1];
    }
```

Just another variation.

Q3 House Robbers?

Ans:-

```
class Solution {
public:

   int fn(vector<int>&arr,vector<int>&dp,int a){
    if(a<0){
        return 0;
    }
    else if(a==0){
        return dp[0]=arr[a];
    }
    else if( a==1){
     dp[a]=max(arr[1],arr[0]);
     return dp[a];
    }
    else if(dp[a]!=-1){
        return dp[a];
    }
    else{
        int p=-1e9,q=-1e9;
            p=arr[a]+fn(arr,dp,a-2);
            q=fn(arr,dp,a-1);
            return dp[a]=max(p,q);

    }
   }
```

```
    int rob(vector<int>& arr) {
        int n=arr.size();
        vector<int>dp(n,-1);
         fn(arr,dp,n-1);
        return dp[n-1];
    }
};
```

Here two operations were taken and not taken rather than the previous versions of X-1 or X-2.

2D/3D DP And DP on Grids

Q1 Ninja Training?

Ans:- In 2D/3D DP we are provided with a 2D matrix in a question. In these cases we do not know for the answer for particular index like in 1D DP because we can not generalize, as there is a update of values like if I choose this index and last time I chose that index so my answer is this , but my answer can change depending upon which index I have chosen last time. Whereas in 1D DP we know the ans

for indexes 1,2, or 3 but here we do not know the answer for 1,2 or 3rd index, if we want to calculate manually for index 1,2,3 then we make things tedious as what is the matrix size is increased and we can't perform manually but in 1D we can perform easily because dp[0] , dp[1] find kaafi easy hota h. so hum 2D me abhi ka ans + fn() lagate h taaki dp[i][j] pata lag sake aur cheez sabhi values ke lie h , unlike 1D me jhn dp[0] , d[1] hume malum hi hota h.But manually calculating is still not possible because here the answer is dependent upon 2 things, 1st one is what we have chosen now, and 2nd one is what we have chosen before/which method we have used before, so our answer is this.

Code:-

```
  int fn(vector<vector<int>>& arr,vector<vector<int>>&
dp,int i,int j){
      if(i<0){
          return 0;
      }
      else if(dp[i][j]!=-1){
          return dp[i][j];
      }
```

```
      else{
          int p=-1e9;
          for(int k=0;k<3;k++){
              if(k!=j){
                  p=max(p,arr[i][k]+fn(arr,dp,i-1,k));
                  // mereko max batao agr maine isko chuna
aur pehle
                  //kuch bhi chuna ho to mila jula ke max
kya milega

                  // maanlo row 2 ne 2nd element ko chuna h,
to 1st row
                  // mereko 2nd element ke alawa jo max ho
wo lake dega.
                  // similarly 3dr row 2nd row mese wo
chunega jo consecutive,
                  // na ho aur maximum bhi return kare
filhal ke situation ke hisab se.
              }
          }
          return dp[i][j]=p;
      }
  }

    int maximumPoints(vector<vector<int>>& arr, int n) {
        // Code here
        vector<vector<int>>dp(n+1,vector<int>(3,-1));
        int d=-1e9;
        for(int i=0;i<3;i++){
            d=max(d,fn(arr,dp,n-1,i));
        }
        return d;
    }
```

So see here answer depends upon, what I have chosen now and what I have chosen before, You may think what I chose in earlier stages should also be considered, but these things are already being stored,

for example, in the 2nd row I have chosen 2nd element so p in that case return the maximum value, for which my current chosen value plus whatever I chose in 1st row, except for the same element as in 2nd row.

So the fixed answer is stored in terms of 2D like for the 2nd row if I chose the 2nd element I will get this answer as dp[1][1]=p (fixed answer) similarly for other elements of the 2nd row we get some fixed values, which is not updated further, so similarly if I chose 1 element of 3rd row, and I chose 2nd element of 2nd row, I do not have to care about what I have chosen in 1 row, because the maximum possible answer from 1st row to 2nd row, if I chose 2nd element of 2nd row, is already calculated. So for the 3rd row, it becomes easy to calculate. In short manlo 2nd row me kuch bhi chuna ho, to 1st row se aisa kya chunenge ki arr[1][1]+previous element , maximum ho , mane ki agr koi 3rd row se se 2nd row ka 2nd element chone to usko 1 row me kya chunna h uski tension na ho , because dp[1][1] ne pehle hi calculate krlia h ki agr tum mereko lete ho to , arr[1][1] ko mila ke , 1row me kya add karen ki (arr[1][1] +previous element) maximum hojae.

To 3rd row ko bus dekhna h dp[1][1] kya dera h aur usko sach maan kea age bhadna h.

Q2 Unique Paths?

Ans:-

```
class Solution {
public:

int fn(vector<vector<int>>&dp,int i,int j){
    if(i<0 || j<0){
        return 0;
    }
    else if(i==0 && j==0){
        return dp[0][0]=1;
    }
    else if(dp[i][j]!=-1){
        return dp[i][j];
    }
    else{
        int p=0,q=0;
        p=fn(dp,i-1,j)*1;// ek just uper phonchne me kitna
tarika laga
        q=fn(dp,i,j-1)*1;// ek just left pohonchne me kitna
tarika laga
        // *1 islie kaaya kyuki left aur uper se final jgh
pohonchne ke liye ek hi tarika h.
        return dp[i][j]=p+q;
    }
}

    int uniquePaths(int m, int n) {
        vector<vector<int>>dp(m,vector<int>(n,-1));
        fn(dp,m-1,n-1);
        return dp[m-1][n-1];
    }
```

```
};
```

Exact climbing stairs jaisa hi ques h , bus isme , 2D matrix ke indexes me store krna h ki udhar aane me kitna tarika laga simple.

Aur udhar aane me kitna tarika laga depend krta h ki tum kitne tarike se aarhe ho.

Ab dp[1][0] aur dp[0][1] ye dono fixed answers denge, so dp[1][1] =dp[1][0]+dp[0][1], bhi fixed hua similarly, is fixed values ko use krke , sare indexes fixed falues hi store karenge in 2D array, agr 1D array lete aur bolte is row me ye maximum h to galat hota , because uss row me kis column se maximum aaya h wo bhi matter krta h , islie humne 2D array lia h.

Simplified NOTE :- 2D DP me hum answer store krte h answer for the indexes present in the 2D vector. Aur wo answer desired answer hota, taaki tumhe peeche jaane ki jarurat na

pade in 2D DP(aur wo calculation question to question depend krta h , jisse hum ye pata laga payen ki kaise us particular index ke liye answer lana h jisme pehle kya lia tha abhi kya liya sb mila jula ke particular index ke liye answer compute krna h(like approach in ques 1 and ques 2 is different but are storing answer in our 2D vector named as dp)). we can not calculate answer for some indexes because it can become tedious.

And similarly in 1D DP , hum answer store krte h in indexed of 1D array, taak hume peeche jane ki zaroorat na pade. And in 1D we already know the solutions of some indexes by calculating manually.

Q3 Minimum Path Sum in a Grid?

Ans:-

```
class Solution {
public:

int fn(vector<vector<int>>& arr,vector<vector<int>>& dp,int
i,int j){
    if(i<0 || j<0){
        return 1e9;
```

```
    }
    else if(i==0 && j==0){
        return arr[0][0];
    }
    else if(dp[i][j]!=-1){
        return dp[i][j];
    }
    else{
        int p=1e9;

            p=min(p,min(arr[i][j]+fn(arr,dp,i-
1,j),arr[i][j]+fn(arr,dp,i,j-1)));
        return dp[i][j]=p;
    }
}

    int minPathSum(vector<vector<int>>& arr) {
        int m=arr.size();
        int n=arr[0].size();
        vector<vector<int>>dp(m,vector<int>(n,-1));
        int d=1e9;

               d=fn(arr,dp,m-1,n-1);

            return d;
    }
};
```

Here we are using a 2D vector for storing the answer and logic for storing the answer depends upon what we have chosen now and what we have chosen before, and computed the desired answer. For complete theory how

we have computed the answer by considering what we taken now and what we have taken before and we have computed the answer, then read Q1 theory and the depth logic like arr[i][j]+fn(dp,arr,i-1,j) kaise kia , to ye sb intuition aur practice se aaega.

Q4 3D DP question:- Chocolates Pick up?

Ans:- here we have to answer in 3D vector.

Because our answer depends upon the 3 parameters.

Code:-

```
        int
fn(vector<vector<vector<int>>>&dp,vector<vector<int>>&
arr,int i,int j1,int j2,int m,int n){
            if(i==n || j1<0 || j1==m || j2<0 || j2==m){
                return -1e9;
            }
            else if(dp[i][j1][j2]!=-1){
                return dp[i][j1][j2];
            }
               else if(i==n-1){
                if(j1==j2){
                    return arr[i][j1];
                }
                else{
                    return arr[i][j1]+arr[i][j2];
```

```
                }
            }
            else{
                int p=0;
                if(j1==j2){
                    for(int k=-1;k<=1;k++){
                        for(int l=-1;l<=1;l++){
                            p=max(arr[i][j1]+fn(dp,arr,i+1,
j1+k,j2+l,m,n),p);
                        }
                    }
                }
                else{
                       for(int k=-1;k<=1;k++){
                        for(int l=-1;l<=1;l++){
                            p=max(arr[i][j1]+arr[i][j2]+fn(
dp,arr,i+1,j1+k,j2+l,m,n),p);
                        }
                    }
                }
                return dp[i][j1][j2]=p;
            }
         }

    int solve(int n, int m, vector<vector<int>>& arr) {
        // code here
        vector<vector<vector<int>>> dp(n,
vector<vector<int>>(m, vector<int>(m, -1)));
        fn(dp,arr,0,0,m-1,m,n);
        return dp[0][0][m-1];
    }
```

DP on Subsequence

In this case, generally, 1 or 2 1D arrays are given and asked to find the maximum or minimum or ask if the element satisfies the criterion or not, by choosing some of the elements from the array, and the criterion for choosing the elements will vary from question to question. In most of cases, we will apply a pick and not pick algorithm where our recursion will consider the element and do not consider the element simultaneously.

Q1 Subset Sum?

Ans:-

```
int fn(vector<int>&arr,vector<vector<int>>&dp,int k,int a){
    if(a<0){
        return (k==0)?1:0;
    }
    else if(k==0){
        return dp[a][k]=1;
    }
    else if(dp[a][k]!=-1){
        return dp[a][k];
    }
    else{
        int p=0,q=0;
        if(k-arr[a]>=0){
            p=fn(arr,dp,k-arr[a],a-1);
```

```
        }
        q=fn(arr,dp,k,a-1);
        return dp[a][k]=p|q;
    }
}

    bool isSubsetSum(vector<int>arr, int sum){
        // code here
        int n=arr.size();
        vector<vector<int>>dp(n,vector<int>(sum+1,-1));
        fn(arr,dp,sum,n-1);
        return dp[n-1][sum];
    }
```

Here we have taken a 2D array, to store the result of If we have chosen this index and my current sum is k, what result I would expect if I recursively iterate further. Means agr tumhara sum k h aur tum arr[a] ko pick kroge to, dp[a][k] btaega ki kya tumhe k==0 mil skta h ya nhi, agr 1D lete to mushkil hota because kisi index pe true hoga yan hi depend krta h ki k kya h uss time , agr arr[a]==k hua to true deta, nhi false deta , to ye ambiguity ko solve krne ke liye 2D DP lia.

Q2 Rod Cutting Problem?

Ans:-

```
int fn(int arr[],vector<vector<int>>&dp,int a,int l)
 {
     if(a<0|| l<=0){
         return 0;
     }
     else if(dp[a][l]!=-1){
         return dp[a][l];
     }
     else{
         int p=0,q=0;
         if(a+1<=l){
         p=arr[a]+fn(arr,dp,l-a-1,l-(a+1));
         }
         q=fn(arr,dp,a-1,l);
         return dp[a][l]=max(p,q);
     }
 }

   int cutRod(int arr[], int n) {
       //code here
       int s=0;
       for(int i=0;i<n;i++){
           s+=arr[i];
       }
       vector<vector<int>>dp(n,vector<int>(n+1,-1));
       fn(arr,dp,n-1,n);
       return dp[n-1][n];

   }
```

Here we have taken a 2D vector to store the result, at a particular index, if are left with l length of the rod then what would be our

maximum profit in that case. And if you are thinking ki aise kaise pata chala, to ye DP on 2D ke topics me maine puri theory di h padhlo ki kaise pata chalega, aur waise bhi intuition building practice se aajata h. bus itna pata karlo ki kya store karna h , aur kaise calculate hoga, fir recursion laga do , aur heap of faith rakho.

```
p=arr[a]+fn(arr,dp,l-a-1,l-(a+1));
```

ye maine kia ki abhi mai arr[a] le lia , to mereko batao ki peeche agr mai element leta to lya profit aata , aise karte karte maan lo mai rod of length pe gya to whn se ,mereko ek fixed ans milta because 1 hi length bacha , so

so dp[a][1] mereko ek fixed ans deta depend krta h a==0 h ya 1 because rod of length a==1 ke liye arr[0] value deta, agr a<0 hota to 0 deta , but dp[a][k] ka ans milta jo ki max hota agr mai rod of length 1 ko consider krta yap hir nhi krta, so similarly , mere paas ek fixed ans aaya dp[a][l==1] ke liye ab agr maid dp[a][2] ke liye calculate krta to to bhi isme

recursion lagta ki ya to mai 2 ko pura lelun ya 1 , 1 krke kaatun aur jo bhi max hota wo ans hota , to base case se return hone ke baat ek tarah se maanlo ki fixed ans generate hone lagta h , chahe max ke liye nikal rhe hoy a min ke liye , about doubt ye h ki kya para ans update horha ho , to cheez resolve hoti jhi tum dp array knsa le rhe ho, ab maanlo tumhara ans 2 factor pe depend krta h , agr dp array tumhara dono factors ke ans store krr rha h to aisa ho hi nhi skta ki , uss ans me update ki zaroorta pade because tume calculation ka logoc jo bhi dia ho, is hisab se undono factors ke liye jo best ans ho skta tha wo 2D dp dedeta h. agr factor 2 hai 1D dp h to dikkat h. par ye patthar ki lakeer h ki agr factor 2 h aur 2D dp h to wo best optimal hi , dega because usko update krne ke liye koi teesra factor h hi nhi , jis tarike se dp[i][j] ans dia h dubara wo usi tarike se whi ans dega , until and unless koi teesra factor role play krr rha ho.

DP on Strings

In this category, we are with 1 or 2 strings and asked to find the longest or shortest / maximum or minimum kind of thing that can be solved by picking or not picking the algorithm of DP. For this category we have to take 2D vector dp for ans storing because our ans depends upon 2 factors.

Q1 Longest Common Subsequence?

Ans:-

```
class Solution {
public:

    int fn(vector<vector<int>>&dp,string &s1,string &s2,int
a,int b){
        if(a<0 || b<0){
            return 0;
        }
        else if(dp[a][b]!=-1){
            return dp[a][b];
        }
        else{
            int p=0,q=0;
            if(s1[a]==s2[b]){
                p=1+fn(dp,s1,s2,a-1,b-1);
            }

                q=max(fn(dp,s1,s2,a-1,b),fn(dp,s1,s2,a,b-
1));
```

```
            return dp[a][b]=max(p,q);
        }
    }

    int longestCommonSubsequence(string s1, string s2) {
        int n=s1.length();
        int m=s2.length();
        vector<vector<int>>dp(n,vector<int>(m,-1));
        return fn(dp,s1,s2,n-1,m-1);
        // return dp[n-1][m-1];
    }
};
```

Here we have taken 2D dp to store the answer because of our answer depends upon two factors which are the indexes of string 1 and string 2. Logic is coded according to the demands of the question.

Q2 Print All Longest Common Subsequence?

Ans:-

first I tried this code:-

```
      int fn(vector<vector<int>>&dp,string &s1,string
&s2,int a,int b){
        if(a<0 || b<0){
            return 0;
        }
        else if(dp[a][b]!=-1){
            return dp[a][b];
        }
```

```
        else{
            int p=0,q=0;
            if(s1[a]==s2[b]){
                p=1+fn(dp,s1,s2,a-1,b-1);
            }

                q=max(fn(dp,s1,s2,a-1,b),fn(dp,s1,s2,a,b-
1));

            return dp[a][b]=max(p,q);
        }
    }
      int gn(vector<vector<int>>&ap,string &s1,string
&s2,int a,int b,int c,vector<string>&v,string s){

        if(s.length()==c){
                        reverse(s.begin(), s.end());

            v.push_back(s);
        }
        if(a<0 || b<0){
            return 0;
        }

        else if(ap[a][b]!=-1){
            return ap[a][b];
        }
        else{
            int p=0,q=0;
            if(s1[a]==s2[b]){
                p=1+gn(ap,s1,s2,a-1,b-1,c,v,s+s1[a]);
            }

                q=max(gn(ap,s1,s2,a-
1,b,c,v,s),gn(ap,s1,s2,a,b-1,c,v,s));

            return ap[a][b]=max(p,q);
        }
```

```
    }

        vector<string>
all_longest_common_subsequences(string s, string t)
        {
            // Code here
            int n=s.length(),m=t.length();
            vector<vector<int>>dp(n,vector<int>(m,-1));
            vector<vector<int>>ap(n,vector<int>(m,-1));

            int d=fn(dp,s,t,n-1,m-1);
            vector<string>v,v1;
            string h="";
            int e=gn(ap,s,t,n-1,m-1,d,v,h);
            set<string>st;

            for(auto &p:v){
                st.insert(p);
            }
            for(auto &p:st){
                v1.push_back(p);
            }
return v1;

        }
```

The problem In this code is try to find the length of lcs first then applied gn function so that whichever string s has length c , then I will store it on my vector, but the problem is, not all my strings were getting to the point where their length == c, because before that ap 2D vector send the answer, so we never to

that point. My functions gn and fn are the same, I am just storing the answer in the gn function.

Now I will do this:-

In this, we have to use the tabulation method instead of the memoization method to find lcs.

Method to convert memoization code to tabulation code:-

```
class Solution {
public:
    int longestCommonSubsequence(string s1, string s2) {
        int n=s1.length();
        int m=s2.length();
        vector<vector<int>>dp(n+1,vector<int>(m+1,-1));
        for(int i=0;i<=n;i++){
            dp[i][0]=0;
        }
        for(int i=0;i<=m;i++){
            dp[0][i]=0;
        }
        for(int i=1;i<n+1;i++){
               for(int j=1;j<m+1;j++){
                if(s1[i-1]==s2[j-1]){
                    dp[i][j]=1+dp[i-1][j-1];
                }
                else{
                    dp[i][j]=max(dp[i-1][j],dp[i][j-1]);
                }
               }
```

```
        }
        return dp[n][m];
    }
};
```

The above code for finding lcs.

First create a 2D vector, then assign the values in 2D array just like we assigned in the base condition, the replace fn(i-1,j) to dp[i-1][j]. and apply two for loop because there we 2 factors. Then send off the answer by the way you can compare the codes of memorization and tabulation and can easily figure out how we have done.

Now back to answer our question:-

Here the way to get to our answer we just have to backtrack, which means we move from top to bottom to get the answer, now we will move from bottom to up.

```
#include <bits/stdc++.h>

using namespace std;

void lcs(string s1, string s2) {

  int n = s1.size();
  int m = s2.size();
```

```
  vector < vector < int >> dp(n + 1, vector < int > (m + 1,
0));
  for (int i = 0; i <= n; i++) {
    dp[i][0] = 0;
  }
  for (int i = 0; i <= m; i++) {
    dp[0][i] = 0;
  }

  for (int ind1 = 1; ind1 <= n; ind1++) {
    for (int ind2 = 1; ind2 <= m; ind2++) {
      if (s1[ind1 - 1] == s2[ind2 - 1])
        dp[ind1][ind2] = 1 + dp[ind1 - 1][ind2 - 1];
      else
        dp[ind1][ind2] = 0 + max(dp[ind1 - 1][ind2],
dp[ind1][ind2 - 1]);
    }
  }

  int len = dp[n][m];
  int i = n;
  int j = m;

  int index = len - 1;
  string str = "";
  for (int k = 1; k <= len; k++) {
    str += "$"; // dummy string
  }

  while (i > 0 && j > 0) {
    if (s1[i - 1] == s2[j - 1]) {
      str[index] = s1[i - 1];
      index--;
      i--;
      j--;
// jaise aae the upper se neeche ab neeche se upper jar he
h  waise hi.
```

```
    } else if (dp[i - 1][j] > dp[i][j-1]) {
      i--;
    } else j--;
  }
  cout << str;
}

int main() {

  string s1 = "abcde";
  string s2 = "bdgek";

  cout << "The Longest Common Subsequence is ";
  lcs(s1, s2);
```

Q3 Edit Distance?

Ans:-

```
class Solution {
public:

int fn(string s1, string s2,int i,int
j,vector<vector<int>>&dp){
    if(i<0){
        return j+1;
    }
    else if(j<0){
        return i+1;
    }
    else if(dp[i][j]!=-1){
        return dp[i][j];
    }
    else{
        int p=1e9,q=1e9;
        if(s1[i]==s2[j]){
```

```
            p=fn(s1,s2,i-1,j-1,dp);
        }
        else{
        q=1+min({fn(s1,s2,i,j-1,dp),fn(s1,s2,i-1,j-
1,dp),fn(s1,s2,i-1,j,dp)});
        }
        return dp[i][j]=min(p,q);
    }
}

    int minDistance(string s1, string s2) {
        int n=s1.length(),m=s2.length();
        vector<vector<int>>dp(n,vector<int>(m,-1));
        if(n==0){
            return m;
        }
        else if(m==0){
            return n;
        }
        return fn(s1,s2,n-1,m-1,dp);
        // return dp[n-1][m-1];
    }
};
```

DP on Stocks

In this category, we are given with 1D array which depicts the price of stocks on a particular ith day, and then we have to find the max or min answer based on the question.

Q1 Best time to buy and sell stock 1?

Ans:- the easiest problem of this category.

Have simple implementation.

```
class Solution {
public:
    int maxProfit(vector<int>& arr) {

        int n=arr.size();
        vector<int>v;
        stack<int>st;
        int d=0;
        for(int i=n-1;i>=0;i--){
            if(arr[i]>=d){
                v.push_back(-1);
            }
            else{
                v.push_back(d);
            }
            d=max(d,arr[i]);
        }

        reverse(v.begin(),v.end());
        int e=0;
         for(int i=0;i<n;i++){
            e=max(v[i]-arr[i],e);
         }
         return e;
    }
};
```

Q2 Buy and Sell Stock 2?

Ans:- In this case, there are two scenarios where the 1st Is we buy and the second one in

we sell, so at each element of the vector there are two possible scenarios whether we buy or sell, so take and not take a decision is their for two cases, like in before topics of DP, there is a single scenario whether we will take the element of not but here there are two.

```
class Solution {
public:


      int fn(vector<int>& arr,vector<vector<int>>&dp,int
t,int a,int n){
        if(a==n){
            return 0;
        }
        else if(dp[t][a]!=-1){
            return dp[t][a];
        }
        else{
            int p=0,q=0;
            if(t==1){
                p=max(-
arr[a]+fn(arr,dp,0,a+1,n),fn(arr,dp,1,a+1,n));
            }
            else{
                q=max(arr[a]+fn(arr,dp,1,a+1,n),fn(arr,dp,0,
a+1,n));
            }
                    return dp[t][a]=max(p,q);

        }
      }

    int maxProfit(vector<int>& arr)
```

```
    {
        int n=arr.size();
        vector<vector<int>>dp(2,vector<int>(n,-1));
        return fn(arr,dp,1,0,n);
    }
};
```

Bus take aur not take wala h har question me.

Q3 Buy and Sell Stock 3?

Ans:- this is just a modified version of the above problem, because there is is a restriction, that at most we can only buy 2 times.

```
class Solution {
public:

    int fn(vector<int>&
arr,vector<vector<vector<int>>>&dp,int a,int t,int d,int n){
        if(d==0 || a==n){
            return 0;
        }
        else if(dp[a][t][d]!=-1){
            return dp[a][t][d];
        }
        else{
            int p=0,q=0;
            if(t){
             p=max(-
arr[a]+fn(arr,dp,a+1,0,d,n),fn(arr,dp,a+1,1,d,n));
            }
```

```
            else{
                q=max(arr[a]+fn(arr,dp,a+1,1,d-
1,n),fn(arr,dp,a+1,0,d,n));
            }
            return dp[a][t][d]=max(p,q);

        }
    }

    int maxProfit(vector<int>& arr) {
        int n=arr.size();
        vector<vector<vector<int>>>dp(n,vector<vector<int>>(
2,vector<int>(3,-1)));
        return fn(arr,dp,0,1,2,n);

    }
};
```

Here we have used 3D dp because here the answer depends upon the 3 factors which are:- 1) whether we have to take or not 2) whether we have to pick or not pick, for selling or buying 3) if we are selling or buying, so do we have any transaction before or not, which also contributes to the answer.

Q4 Buy and Sell Stock 4?

Ans:- same as the above question, instead of 2 there are k transactions.

```
class Solution {
public:

    int fn(vector<int>&
arr,vector<vector<vector<int>>>&dp,int a,int t,int d,int n){
        if(d==0 || a==n){
            return 0;
        }
        else if(dp[a][t][d]!=-1){
            return dp[a][t][d];
        }
        else{
            int p=0,q=0;
            if(t){
             p=max(-
arr[a]+fn(arr,dp,a+1,0,d,n),fn(arr,dp,a+1,1,d,n));
            }
            else{
                q=max(arr[a]+fn(arr,dp,a+1,1,d-
1,n),fn(arr,dp,a+1,0,d,n));
            }
            return dp[a][t][d]=max(p,q);

        }
    }
    int maxProfit(int k, vector<int>& arr) {
         int n=arr.size();
        vector<vector<vector<int>>>dp(n,vector<vector<int>>(
2,vector<int>(k+1,-1)));
        return fn(arr,dp,0,1,k,n);
    }
};
```

Q5 Buy and Sell Stock With Cool Down?

Ans:-

This is just an advanced version of 2nd question where after selling we have to move a+2 rather than a+1.

```
class Solution {
public:

 int fn(vector<int>& arr,vector<vector<int>>&dp,int t,int
a,int n){
        if(a>=n){
            return 0;
        }
        else if(dp[t][a]!=-1){
            return dp[t][a];
        }
        else{
            int p=0,q=0;
            if(t==1){
                p=max(-
arr[a]+fn(arr,dp,0,a+1,n),fn(arr,dp,1,a+1,n));
            }
            else{
                q=max(arr[a]+fn(arr,dp,1,a+2,n),fn(arr,dp,0,
a+1,n));
            }
                    return dp[t][a]=max(p,q);

        }
      }
    int maxProfit(vector<int>& arr) {
        int n=arr.size();
       vector<vector<int>>dp(2,vector<int>(n,-1));
```

```
        return fn(arr,dp,1,0,n);
    }
};
```

Q6 Buy and Sell Stock with Transaction Fee?

Ans:- this is a modified version of Q2 here we have to reduce the fee after selling the stock.

```
class Solution {
public:
   int fn(vector<int>& arr,vector<vector<int>>&dp,int t,int
a,int n,int k){
        if(a==n){
            return 0;
        }
        else if(dp[t][a]!=-1){
            return dp[t][a];
        }
        else{
            int p=0,q=0;
            if(t==1){
                p=max(-
arr[a]+fn(arr,dp,0,a+1,n,k),fn(arr,dp,1,a+1,n,k));
            }
            else{
                q=max(arr[a]-
k+fn(arr,dp,1,a+1,n,k),fn(arr,dp,0,a+1,n,k));
            }
                    return dp[t][a]=max(p,q);

        }
      }
    int maxProfit(vector<int>& arr, int k) {
        int n=arr.size();
       vector<vector<int>>dp(2,vector<int>(n,-1));
```

```
        return fn(arr,dp,1,0,n,k);
    }
};
```

GRAPH

Learning Graph

Q1 Find the no. of undirected graphs from n vertices?

Ans:-

```
long long count(int n) {
        // your code here
         long long int d=pow(2,n*(n-1)/2);
         return d;
    }
```

Q2 Graph Representation?

Ans:-

```
vector<vector<int>> printGraph(int n, vector<pair<int,int>>arr) {
        // Code here

        vector<vector<int>>v(n);
        for(int i=0;i<arr.size();i++){
            v[arr[i].first].push_back(arr[i].second);
            v[arr[i].second].push_back(arr[i].first);
        }
        return v;

    }
```

Kuch nhi bus isne vertices ke beech me relations diye hue to bus usme se adjacency matrix banana tha , jo ki cumulative tarike se batata h ki ek particular node kis kis node se mila hua h.

Q3 (BREADTH-FIRST-SEARCH) BFS?

Ans:-

Breadth-First Search (BFS) is a fundamental algorithm in graph theory, commonly used for traversing or searching tree or graph data structures. It explores the vertices of a graph in layers, starting from a specified source vertex and visiting all its neighbors at the present depth level before moving on to nodes at the next depth level.

Why We Perform BFS:

1. **Shortest Path in Unweighted Graphs:** BFS is particularly useful for finding the shortest path from a source vertex to all other vertices in an unweighted graph. Since it explores all neighbors at the current depth level before moving deeper, the first time it reaches a vertex, it guarantees that the path is the shortest.

2. **Level Order Traversal:** BFS is ideal for scenarios where processing nodes in order of their distance from the source vertex is necessary. This is akin to level-order traversal in trees.

3. **Connected Components:** BFS can be used to find all connected components in an undirected graph. By running BFS from every unvisited vertex, you can identify all distinct connected subgraphs.

4. **Cycle Detection in Undirected Graphs:** BFS can be utilized to detect cycles in an undirected graph. If during the traversal, a vertex is found that has already been visited and is not the parent of the current vertex, a cycle exists.

5. **Network Broadcasting:** BFS can simulate scenarios like broadcasting information across a network where you want to ensure that every node gets the message with minimal delay.

6. **Bipartite Graph Check:** BFS helps in checking whether a graph is bipartite. By attempting to color the graph using two colors while traversing, one can determine if it is possible to divide the graph into two sets of vertices with no edges within the same set.

When We Perform BFS:

1. **Finding Shortest Path in Unweighted Graphs:** When you need to determine the shortest path from a source vertex to all other vertices in an unweighted graph, BFS is the algorithm of choice.

2. **Exploring All Nodes within a Certain Distance:** BFS is appropriate when you need to explore all nodes that are within a certain distance from the source node, such as in peer-to-peer networks, social networks, and web crawlers.

3. **Level Order Traversal:** When dealing with problems requiring processing of nodes level by level, BFS is used. This is common in many hierarchical or layered structures.

4. **Identifying Connected Components:** BFS helps in identifying and labeling connected components in a graph, which is useful in network analysis and understanding sub-structures within a graph.

5. **Cycle Detection:** In undirected graphs, to check for the presence of cycles, BFS can be employed.

6. **Checking Bipartiteness:** When you need to determine if a graph is bipartite, BFS helps by trying to color the graph using two colors while traversing.

```
7.  vector<int> bfsOfGraph(int n, vector<int> adj[]) {
8.          // Code here
9.          vector<int>v;
10.              // v.push_back(0);
11.              queue<int>q;
12.              q.push(0);
13.              vector<int>vis(n,0);
14.              vis[0]=1;
15.              while(q.size()!=0){
16.                  int t=q.front();
17.                  q.pop();
18.                  v.push_back(t);
```

```
                for(int i=0;i<adj[t].size();i++){
                    if(vis[adj[t][i]]!=1){
                        vis[adj[t][i]]=1;
                        q.push(adj[t][i]);

                    }
                }
            }

            return v;
        }
```

Here I have done vis[adj[t][i]]=1 inside the for loop not below where I have done q.pop() because there may be a possibility that the same node comes in queue two times because it is not marked visited before coming under the consideration of attachment with another node.

Matlb maanlo node 2, 4 ke sath aur 6 ke sath juda h , jb 2 pe aaya tab tum 4 le liye , lekin queue me 4 ka baari aaya hi nhi , kyu ki 2 se le ke 4 tk ke beech me bht nodes tha to pehle wo pop hua, par 6 tha wala pehle aagya aur usme 6 aur 4 ka connection bhi tha , to 4 to mark hua nhi tha to isliye 4 dubara queue me aagya, to isliye visited mark for loop ke

aandari kia h taaki usi time consider hojae ki knsa node visited aur kaunsa nhi.

Q4 DEPTH-FIRST-SEARCH (DFS)?

Ans:-

Depth-First Search (DFS) is a fundamental graph traversal algorithm used to explore vertices and edges of a graph. It starts at a given vertex and explores as far as possible along each branch before backtracking. DFS can be implemented using either a stack data structure or recursively.

Why We Use DFS:

1. **Pathfinding:** DFS is useful for pathfinding and solving maze-like problems where you need to explore all possible paths from a starting point to find a specific node or to check if a path exists.

2. **Topological Sorting:** In a Directed Acyclic Graph (DAG), DFS is used to perform topological sorting, which orders vertices linearly based on their dependencies.

3. **Connected Components:** DFS helps in finding all connected components in an undirected graph. Each DFS run identifies one connected component.

4. **Cycle Detection:** DFS can detect cycles in both directed and undirected graphs by checking for back edges (edges that point back to an ancestor in the DFS tree).

5. **Graph Coloring and Bipartiteness Check:** DFS can be used to check if a graph is bipartite by attempting to color it with two colors.

6. **Finding Bridges and Articulation Points:** In a graph, DFS can be used to find bridges (edges whose removal increases the number of connected components) and articulation points (vertices whose removal increases the number of connected components).

When We Perform DFS:

1. **When the entire graph needs to be explored:** DFS is used when you need to visit every vertex and edge in a graph, such as when searching for a specific vertex or when you want to examine all possible paths.

2. **For Problems Involving Path Finding:** DFS is well-suited for scenarios where all paths need to be explored, such as puzzle-solving, navigating mazes, and certain game AI implementations.

3. **In Tree Structures:** DFS is inherently recursive and mimics tree traversal methods like pre-order, in-order, and post-order traversals.

4. **To Check Graph Properties:** DFS is used to check for connectivity, presence of cycles, and bipartiteness in graphs.

```
5.     void
   dfs(vector<int>&v,vector<int>&vis,vector<int>adj[],int
   t)
6.     {
7.         v.push_back(t);
8.        // vis[t]=1;
9.         for(int i=0;i<adj[t].size();i++){
10.                if(vis[adj[t][i]]!=1){
11.                    vis[adj[t][i]]=1;
12.                    dfs(v,vis,adj,adj[t][i]);
13.                }
14.            }
15.            return;
16.         }
17.
18.         vector<int> dfsOfGraph(int n, vector<int>
   adj[]) {
19.            // Code here
20.            vector<int>v;
21.            vector<int>vis(n,0);
22.            vis[0]=1;
23.            dfs(v,vis,adj,0);
24.            return v;
25.         }
```

here we have marked visited inside the for loop and 0 is marked visited inside the main function, but here we can be marked visited before the for loop because, this traversal goes to depth and then moves further, if any node is future has a connection with the first lineage of the node, so we do not have to care about as it is already marked visited, so if we have to apply before for loop so we have to write just after taking into the vector v.

and we are pushing the node inside the vector into the beginning of dfs because it will print the desired result as becase in depth searching whenever we move down from a particular node so we have to take it inside the vector v. means jaise hi dfs call hua kisi element ke liye to us element ko collect karlo.

Ab direct collect krna h to seedha top me likhdo , otherwise for loop me likh ke kisi condition andar likhne ka koi faeda hi nhi h

kyu ki usko collect krne ke liye koi condition boli hi nhi h.

Kyu maanlo tumne for loop ke andar likh dia ,likh dia agr if condition ke andar likha to last node me jb jaega to usme if condition kaam hi nhi krega, to wo store hi nhi hoga, ab maanlo else me likha to usme jo visit hochuka hoga usko lelega bina dfs aur usme dfs lgega bhi nhi, agr for loop ke last me likha to wo last se collect karna shuru krega joki desired h nhi.

Q5 No. of Connect Components?

Ans:- In a graph, it is not necessary that there is a connection between all the nodes, there may be a possibility where some of the nodes are combined to form a graph and some other nodes have combined to form another graph so these individual graphs are known as components.

So if there is another component in a given graph, then by traversal technique all nodes would not be marked visited in a single go of

BFS or DFS. If we have to do BFS or DFS one more time then definitely there is another component and the no. of time BFS is applied is equal to the no. of components.

```
 void
dfs(vector<int>&v,vector<int>&vis,vector<vector<int>>&adj,in
t t)
    {
       // vis[t]=1;
        for(int i=0;i<adj[t].size();i++){
            if(vis[adj[t][i]]!=1){
                vis[adj[t][i]]=1;
                dfs(v,vis,adj,adj[t][i]);
            }

        }
        return;
    }
    int findNumberOfGoodComponent(int e, int v,
vector<vector<int>> &arr) {
        // code here
        vector<vector<int>>adj(v+1);
                int t=0;
        for(int i=0;i<arr.size();i++){
            adj[arr[i][0]].push_back(arr[i][1]);
            adj[arr[i][1]].push_back(arr[i][0]);
        }
        vector<int>v9;
        vector<int>vis(v+1,0);
        for(int i=1;i<=v;i++){
            if(vis[i]!=1){
                vis[i]=1;
                t++;
                dfs(v9,vis,adj,i);
            }
```

```
        }
        return t;
    }
```

Problems with BFS/DFS

Q1 Number of Provinces?

Ans:- Just same as to find the no. of components.

```
class Solution {
public:
    int findCircleNum(vector<vector<int>>& arr) {
        int n=arr.size();
        vector<vector<int>>adj(n+1);
        for(int i=0;i<n;i++){
            for(int j=0;j<n;j++){
                if(arr[i][j]==1 && i!=j){
                    adj[i+1].push_back(j+1);
                    adj[j+1].push_back(i+1);
                }
            }
        }
        int t=0;
        vector<int>vis(n+1,0);
        for(int i=1;i<=n;i++){
            if(vis[i]!=1){
                    queue<int>q;
                t++;
                q.push(i);
                vis[i]=1;
                while(q.size()!=0){
                    int t=q.front();
                    q.pop();
                    for(int j=0;j<adj[t].size();j++){
```

```
                    if(vis[adj[t][j]]==0){
                        vis[adj[t][j]]=1;
                        q.push(adj[t][j]);
                    }
                }
            }
        }
        return t;
    }
};
```

Q2 Rotten Oranges?

Ans:- In this, we are thing of traversal because the rotten ones are the reason of more rotten oranges, so we have to travel from the rotten oranges to non rotten ones to get the time of the state where all oranges get rotten.

Here there is no case of optimizations, which means if pure orange is near rotten then it will definitely get rotten. So we do not need dynamic programming here.

Maanlo ek orange ki wjh se sb rotten hogae, to uska time note krlo, fir dusre orange ki wjh

se dekho, agr wo use kam time me rotten krr rha h to usko update krdo fir.

Code:-

```
class Solution {
public:
    int orangesRotting(vector<vector<int>>& arr) {
        int n=arr.size();
        int m=arr[0].size();
        vector<vector<int>>v(n,vector<int>(m,1e9));
        vector<vector<int>>vis(n,vector<int>(m,0));
        queue<pair<int,int>>q;
        int t=0;
        for(int i=0;i<n;i++){
            for(int j=0;j<m;j++){
                if(arr[i][j]==2){
                    q.push({i,j});
                    v[i][j]=0;
                    vis[i][j]=1;
                }
            }
        }
        int dx[4]={0,0,-1,1};
        int dy[4]={1,-1,0,0};
        while(q.size()!=0){
        int x=q.front().first;
        int y=q.front().second;
        q.pop();
        for(int i=0;i<4;i++){
            int nx=x+dx[i];
            int ny=y+dy[i];
            if(nx>=0 && ny>=0 && nx<n && ny<m &&
arr[nx][ny]==1 && vis[nx][ny]==0){
                if(v[nx][ny]>v[x][y]+1){
                    v[nx][ny]=v[x][y]+1;
                }
```

```
                vis[nx][ny]=1;
                q.push({nx,ny});
                arr[nx][ny]=2;
            }
        }
        }
        for(int i=0;i<n;i++){
            for(int j=0;j<m;j++){
                if(arr[i][j]!=0){
                if(arr[i][j]==1){
          return -1;
                }

                t=max(t,v[i][j]);
                }
            }
        }
        return t;
    }
};
```

This is my way of doing it.

Q3 Cycle detection in an undirected graph?

Ans:- here if one node is visited then it should not be encountered while traversing further until and unless it is a parent of the code node from which we are traveling further.

```
   int dfs(vector<int> adj[],vector<int>&vis,int t,int p){
        vis[t]=1;
        for(int i=0;i<adj[t].size();i++){
            if(vis[adj[t][i]]==0){
```

```
                if(dfs(adj,vis,adj[t][i],t)==1){
                    return 1;
                }
            }
            else{
                if(adj[t][i]!=p){
                    return 1;
                }
            }
        }
        return 0;
    }

    bool isCycle(int n, vector<int> adj[]) {
        // Code here
        vector<int>vis(n,0);
        for(int i=0;i<n;i++){
            if(vis[i]==0){
                vis[i]=1;
                int f=dfs(adj,vis,i,-1);
                if(f==1){
                    return 1;
                }
            }
        }
return 0;

    }
```

Q4 Bipartite Graph?

Ans:- in this graph, every node has a distinct color from the adjacent node.

```
class Solution {
public:
```

```cpp
bool isBipartite(vector<vector<int>>& arr) {
    int n=arr.size();
    map<int,int>m;
    queue<int>q;
    // q.push(0);
    // m[0]=0;
    vector<int>vis(n,0);
    // vis[0]=1;
    int y=0;
    for(int i=0;i<n;i++){
        if(vis[i]==0){
            q.push(i);
            m[i]=0;
            vis[i]=1;

    while(q.size()!=0){
        int t=q.front();
        q.pop();
        int d=m[t];
        for(int i=0;i<arr[t].size();i++){
            if(vis[arr[t][i]]!=1){
                vis[arr[t][i]]=1;
                q.push(arr[t][i]);
                int e=!d;
                m[arr[t][i]]=e;
            }
            else{
                if(m[arr[t][i]]==d){
                    return 0;
                }
            }
        }

    }
        }
    }

    return 1;
```

```
    }
};
```

Q5 Cycle Detection in Directed Graph (DFS)?

Ans:- The same algorithm as done for undirected graphs will not work. For detail explanation see this :- https://www.youtube.com/watch?v=9twcmtQj4DU

Here in this, if moving in the same path we visit the node again then we can say the graph has a cycle because the connection between the nodes is unidirectional, so here the child can not go back to his parent until and unless the cycle is present.

Here we will take a visited and path-visited array.

This question will be solved by DFS.

We mark visited and path-visited simultaneously but going back means after dfs

we will unmark path-visited and other nodes may also come to this node through some other direction, but we will still make it visited because we do not want to call dfs for the same node again.

```
bool dfs(vector<int>& vis, vector<int>& pvis, vector<int>
adj[], int t) {
    vis[t] = 1;
    pvis[t] = 1;
    for (int i = 0; i < adj[t].size(); i++) {
        int neighbor = adj[t][i];
        if (vis[neighbor] == 0) {
            if (dfs(vis, pvis, adj, neighbor)) {
                return true;
            }
        } else if (pvis[neighbor] == 1) {
            return true;
        }
    }
    pvis[t] = 0; // Reset the parent visit array here after
all neighbors are processed
    return false;
}

bool isCyclic(int n, vector<int> adj[]) {
    vector<int> vis(n, 0), pvis(n, 0);
    for (int i = 0; i < n; i++) {
        if (vis[i] == 0) {
            if (dfs(vis, pvis, adj, i)) {
                return true;
            }
        }
    }
    return false;
```

```
}
```

Pvis[] ko for loop ke andar unmarked nhi krskte bcoz, if condition ke andar jb last code hota, to if codition me jaata hi nhi , to back-track ke waqt wo unmarked hota hi nhi, and else condition me agr 1 h to ans hi mil gya other wise 0 to wo waise bhi h, so jab bhi back-track krta ho, to function ke last me krte h mane function ke completion se pehle taaki pvis[] ko shuru me jo mark kia tha to ab usko unmark krdo.

Toposort and Problems

Q1 Kahn's Algorithm (DFS method – Topological sort)?

Ans:-

Topological sorting (toposort) is an ordering of the vertices in a directed acyclic graph (DAG) such that for every directed edge $u \rightarrow v$, vertex u comes before vertex v in the ordering. It is a linear ordering of vertices that respects the direction of the edges.

Why Do We Use Toposort?

1. **Dependency Resolution:** Topological sorting is commonly used to resolve dependencies. For example, if you have a set of tasks where certain tasks depend on the completion of others, a topological sort will provide an order in which to complete the tasks.

2. **Build Systems:** In software development, toposort can determine the order in which files should be compiled, given the dependencies between them.

3. **Scheduling:** It can be used in scheduling problems where certain jobs must be completed before others.

4. **Precedence Constraints:** It helps in organizing tasks, courses, or activities that have precedence constraints.

Where Do We Use Toposort?

1. **Course Prerequisites:** When planning a course schedule, where some courses have prerequisites, a topological sort can provide a valid order in which to take the courses.

2. **Task Scheduling:** In project management, tasks with dependencies can be scheduled using toposort to ensure that all dependencies are respected.

3. **Compilation Order:** In programming, determining the order in which modules or files should be compiled when some files depend on others.

4. **Package Management:** In package management systems, installing packages in an order that respects dependencies between packages.

How to Perform Toposort:

There are two main algorithms to perform topological sorting:

1. **Kahn's Algorithm (BFS-Based):**

    - Find all vertices with no incoming edges and add them to a queue.
    - Remove a vertex from the queue, append it to the topological order, and remove its outgoing edges.
    - If removing an edge leaves another vertex with no incoming edges, add that vertex to the queue.

- Continue until the queue is empty. If all vertices are processed, the graph is a DAG, and the topological order is valid. If not, the graph has at least one cycle and hence no valid topological order.

2. **Depth-First Search (DFS-Based):**

- Perform a DFS traversal of the graph.
- On finishing the traversal of a vertex, add it to the front of a list (or push it onto a stack).
- The reverse of the list (or the content of the stack) will give the topological order.

```
3.     vector<int> topoSort(int n, vector<int> adj[])
4.     {
5.         // code here
6.         vector<int>v(n,0);
7.         queue<int>q;
8.         vector<int>v1;
9.         for(int i=0;i<n;i++){
10.
11.                 for(int j=0;j<adj[i].size();j++){
12.                     v[adj[i][j]]++;
13.                 }
14.
15.             }
16.             for(int i=0;i<n;i++){
17.                 if(v[i]==0){
18.                     q.push(i);
19.                 }
20.             }
21.             while(q.size()!=0){
22.                 int t=q.front();
23.                 q.pop();
24.                 v1.push_back(t);
25.                 for(int i=0;i<adj[t].size();i++){
26.                     v[adj[t][i]]--;
27.                     if(v[adj[t][i]]==0){
28.                         q.push(adj[t][i]);
29.                     }
30.                 }
```

```
31.             }
32.             return v1;
33.         }
```

This is the first case where the visited array is not used anyway this is the BFS way or My way of doing it, here we have stored the nodes that have 0 incoming edges, then in the while loop, we store 0 vertices with 0 incoming nodes and subtract the incoming node from the other nodes and then collect 0 incoming nodes again.

Q2 (DFS method – Topological sort)?

Ans:-

```
    void dfs(int node, int vis[], stack<int> &st,
             vector<int> adj[]) {
        vis[node] = 1;
        for (auto it : adj[node]) {
            if (!vis[it]) dfs(it, vis, st, adj);
        }
        st.push(node);
    }
public:
    //Function to return list containing vertices in
Topological order.
    vector<int> topoSort(int V, vector<int> adj[])
    {
        int vis[V] = {0};
```

```
        stack<int> st;
        for (int i = 0; i < V; i++) {
            if (!vis[i]) {
                dfs(i, vis, st, adj);
            }
        }

        vector<int> ans;
        while (!st.empty()) {
            ans.push_back(st.top());
            st.pop();
        }
        return ans;
    }
```

Q3 Cycle Detection in Directed Graph (BFS)?

Ans:-

Simple toposort krlo BFS se Chahe DFS se, agr toposort vector ka size == no. of nodes then cycle is not present otherwise cycle is present.

SHORTEST PATH ALGORITHMS AND PROBLEMS

Q1 Shortest Path in UDG with Unit Weights?

Ans:- Here we have to do a traversal technique, to go to a particular node, then

just extra thing we have to do is to calculate the distance from going one node to another, here it is mentioned as 1 unit from going one node to another node.

Now in these categories of questions, we generally do not take visited arrays, because we have to update information about the node.

NOTE:- think of both side's like is DP is beneficial or Graph Algorithms whichever suits best just try that only.

```
   vector<int> shortestPath(vector<vector<int>>& arr, int
n,int m, int s){
        // code here
        // first create the adjacency matrix.
        vector<vector<int>>adj(n);

        for(int i=0;i<m;i++){
            adj[arr[i][0]].push_back(arr[i][1]);
            adj[arr[i][1]].push_back(arr[i][0]);
        }
        vector<int>v(n,1e9);
        queue<pair<int,int>>q;
        q.push({s,0});
        v[s]=0;
        while(q.size()!=0){
            int t=q.front().first;
            int d=q.front().second;
            v[t]=d;
```

```
            q.pop();
            for(int i=0;i<adj[t].size();i++){
                if(v[adj[t][i]]>v[t]+1){
                    v[adj[t][i]]=v[t]+1;
                    q.push({adj[t][i],d+1});
                }
            }
        }
        for(auto &p:v){
            if(p==1e9){
                p=-1;
            }
        }
        return v;
    }
```

Here we have taken a node inside the queue only when it gets updated because, from this node we get new shorter distances for other nodes that are connected to that particular node which is updated just now.

Q2 Shortest Path in DAG with given weights?

Ans:-

```
 vector<int> shortestPath(int n,int m, vector<vector<int>>& ar
r){

        // code here
        // lets create an adjacency matrix with the given weig
hts

        vector<vector<pair<int,int>>>adj(n);
        for(int i=0;i<m;i++){
            adj[arr[i][0]].push_back({arr[i][1],arr[i][2]});
        }
        vector<int>v(n,1e9);
```

```
        v[0]=0;
        priority_queue<pair<int,int>>q;
        q.push({0,0});
        while(q.size()!=0){
            int t=q.top().first;
            int d=q.top().second;
            q.pop();
            v[t]=d;
            for(int i=0;i<adj[t].size();i++){
                if(v[adj[t][i].first]>v[t]+adj[t][i].second){
                    v[adj[t][i].first]=v[t]+adj[t][i].second;
                    q.push({adj[t][i].first,adj[t][i].second+v
[t]});
                }
            }
        }
        for(auto &p:v){
            if(p==1e9){
                p=-1;
            }
        }
        return v;
    }
```

In directed graph jab sirf normal queue lia tha, aur line 16 me v[t]=d kia tha to gala tans aarha tha kyuki aisa ho skta h ki for particular t, queue me 2 ans store ho maanlo {2,18}, and {2,35}, ab tum kya krr rhe ho ki d ek baar 18 h , to tum v[2]=18 krdiye, fir whn se kahin gae, to udhar se distance dekhne lage, pr jb dubara tum t=2 pe gae to d=35 milaa , to tum v[t]=35 update krdiye ye jane bina ki wo value chota h

ya bada, isliye tumhare test cases me for a particular value galat aarha h.

Aur jn priority queue liya to but obvious h ki max distance pehle upper aaega aur jb last update hoga to wo chote value se hoga.
pr bina priority queue ke krna h to aise krna hoga chahe UG hoy a DG:-

```
 vector<int> shortestPath(int n,int m, vector<vector<int>>& ar
r){
        // code here
        // lets create an adjacency matrix with the given weig
hts

        vector<vector<pair<int,int>>>adj(n);
        for(int i=0;i<m;i++){
            adj[arr[i][0]].push_back({arr[i][1],arr[i][2]});
        }
        vector<int>v(n,1e9);
        v[0]=0;
        queue<pair<int,int>>q;
        q.push({0,0});
        while(q.size()!=0){
            int t=q.front().first;
            int d=q.front().second;
            q.pop();
            if(v[t]>d){
            v[t]=d;
            }
            for(int i=0;i<adj[t].size();i++){
                if(v[adj[t][i].first]>v[t]+adj[t][i].second){
                    v[adj[t][i].first]=v[t]+adj[t][i].second;
                    q.push({adj[t][i].first,adj[t][i].second+v
[t]});
                }
```

```
            }
        }
        for(auto &p:v){
            if(p==1e9){
                p=-1;
            }
        }
        return v;
    }
```

Bus ek update daal dena in case agr queue me dubara aagya h to update condition ke hisab se krna. Because agr galat value se update krdoge to sirf usi node ke liye galat aaega, because us particular node ki wjh se, baaki jgh smallest way se gae hoge becoz pehle smallest than na, to dusri jgh jana optimize hoga bus jb dubara update kia tha v[t]=d, tb sirf us particular node ke liye galti hogya tha.

So UG hoy a DG agr queue use krr rhe ho to condition lagana padega.

Q3 Dijkstra's Algorithm?

Ans:- This is the same as Q1, the only change is that the weights are not 1 here.

This question is going to be solved by using the method used in q1.

Dijkstra's algorithm is a classic algorithm for finding the shortest paths between nodes in a graph. Specifically, it calculates the shortest path from a single source node to all other nodes in a weighted graph with non-negative edge weights. The algorithm is named after the Dutch computer scientist Edsger Dijkstra, who proposed it in 1956.

Why Dijkstra's Algorithm is Used

1. **Shortest Path Calculation**: The primary use of Dijkstra's algorithm is to find the shortest path from a source node to a destination node in a graph.
2. **Routing**: In networking, Dijkstra's algorithm is used in routing protocols to find the shortest path for data packets.
3. **Mapping and Navigation**: GPS systems use Dijkstra's algorithm to find the shortest route between two locations.
4. **Pathfinding in AI**: It is used in AI for pathfinding, such as in game development for characters to find the shortest path to their targets.

```
5.   vector <int> dijkstra(int n, vector<vector<int>>
adj[], int s)
6.     {
7.         // Code here
8.         queue<pair<int,int>>q;
9.         q.push({s,0});
10.            vector<int>v(n,1e9);
11.            while(q.size()!=0){
12.                int t=q.front().first;
13.                int d=q.front().second;
14.                q.pop();
15.                if(v[t]>d){
16.                    v[t]=d;
17.                }
18.                for(int i=0;i<adj[t].size();i++){
19.                    if(v[adj[t][i][0]]>v[t]+adj[t][i][1
]){
20.                        v[adj[t][i][0]]=adj[t][i][1]+v[
t];
```

```
                        q.push({adj[t][i][0],adj[t][i][
  1]+v[t]});
                    }
                }
            }
            for(auto &p:v){
                if(p==1e9){
                    p=-1;
                }
            }
            return v;
        }
```

Q4 Bellman Ford Algorithm?

Ans:-

The Bellman-Ford algorithm is used to find the shortest paths from a single source vertex to all other vertices in a weighted graph. Unlike Dijkstra's algorithm, Bellman-Ford can handle graphs with negative weight edges, making it more versatile.

Key Features and Uses of Bellman-Ford Algorithm

1. **Handles Negative Weights**:
    - Bellman-Ford can process graphs where edges may have negative weights, which Dijkstra's algorithm cannot handle correctly.
2. **Detects Negative Weight Cycles**:
    - The algorithm can detect if there is a negative weight cycle in the graph (a cycle whose total weight is negative). If such a cycle exists, it indicates that there is no shortest path because the total path cost can be infinitely reduced by traversing the cycle.
3. **Applications**:
    - **Currency Arbitrage**: In financial markets, Bellman-Ford can be used to detect arbitrage opportunities where a cycle of currency conversions yields a profit.

- **Network Routing**: In computer networks, it is used to find the shortest path when there might be negative weights in the cost metrics.
- **Dynamic Graphs**: When dealing with graphs where edge weights can change dynamically, Bellman-Ford is useful because it recalculates paths efficiently even with negative weights.

Only worked in directed graph.

Just apply Dijkstra n-1 time where n is no. of nodes.

```
    vector<int> bellman_ford(int V, vector<vector<int>>& edges, int S) {
        vector<int> dist(V, 1e8);
        dist[S] = 0;
        for (int i = 0; i < V - 1; i++) {
            for (auto it : edges) {
                int u = it[0];
                int v = it[1];
                int wt = it[2];
                if (dist[u] != 1e8 && dist[u] + wt < dist[v]) {
                    dist[v] = dist[u] + wt;
                }
            }
        }
        // Nth relaxation to check negative cycle
        for (auto it : edges) {
            int u = it[0];
            int v = it[1];
            int wt = it[2];
            if (dist[u] != 1e8 && dist[u] + wt < dist[v]) {
```

```
                return { -1};
            }
        }


        return dist;
    }
```

Q5 Floyd Warshall Algorithm?

Ans:- In this algorithm, we have to determine the distance from any to any node.

So we have to find out every possible network where node 1 can travel to node 2 by any possible way.

This is the Algorithm where dynamic programming comes into the picture.

Works in DG and also in UDG when it is converted to DG.

If the arr[i][i]<0 then the graph contains a cycle.

```
void shortest_distance(vector<vector<int>>&arr){
        // Code here
```

```
        int n=arr.size();
        for(int i=0;i<n;i++){
            for(int j=0;j<n;j++){
                if(arr[i][j]==-1){
                    arr[i][j]=1e9;
                }
                if(i==j){
                    arr[i][j]=0;
                }
            }
        }

        for(int k=0;k<n;k++){// this for is loop is for via
where node
        //i will travel to node j via node k.
        for(int i=0;i<n;i++){

            for(int j=0;j<n;j++){
     // if i am moving from i to j, then i making sure that
i consider the
     // least possible weigths to go from i to j.
                arr[i][j]=min(arr[i][j],arr[i][k]+arr[k][j])
;
// here we are travelling via node k, so first we go from
node i to node k
// which is arr[i][k], then we travel form node k to node j,
which is arr[k][j].

            }
        }

        }
           for(int i=0;i<n;i++){
            for(int j=0;j<n;j++){
                if(arr[i][j]==1e9){
                    arr[i][j]=-1;
                }
```

```
            }
        }

    }
```

If you are thinking ki wo kaise store krr rha to dekho who kisi na kisi loop ki wjh se store kr le rhaa h agr wo nhi krr rha h to whn connection h hi nhi fir.

Q6 Find the city with the smallest numbers of neighbours in a threshold distance?

Ans:-

```
class Solution {
public:
    int findTheCity(int n, vector<vector<int>>& arr, int d)
{
        vector<vector<int>>adj(n,vector<int>(n,1e9));
        for(int i=0;i<arr.size();i++){
            adj[arr[i][0]][arr[i][1]]=arr[i][2];
            adj[arr[i][1]][arr[i][0]]=arr[i][2];
        }
    for(int i=0;i<n;i++){
        adj[i][i]=0;
    }
           for(int k=0;k<n;k++){
            for(int i=0;i<n;i++){
                for(int j=0;j<n;j++){
                    adj[i][j]=min(adj[i][j],adj[i][k]+adj[k]
[j]);
                }
            }
```

```
            }

            int t=1e9,z=0,f=0;
            for(int i=0;i<n;i++){
             z=0;
             for(int j=0;j<n;j++){
                 if( i!=j && adj[i][j]<=d){
                     z++;
                 }
             }
             if(t>=z){
                    f=i;
                    t=z;
             }
            }
           return f;
    }
};
```

Just implemented floyed warshall on a question.

MINIMUM SPANNING TREE/DISJOINT SET AND PROBLEMS

Q1 Minimum Spanning Tree?

Ans:-

A Minimum Spanning Tree (MST) is a subset of the edges of a connected, undirected graph that connects all the vertices together, without any cycles and with the minimum possible total edge weight. Here's a detailed explanation of what MST is, why it is used, and its applications:

What is a Minimum Spanning Tree?

1. **Definition**:
   - A spanning tree of a graph is a subgraph that includes all the vertices of the original graph, is connected, and contains no cycles.
   - A minimum spanning tree is a spanning tree with the smallest possible total edge weight.
2. **Properties**:
   - It includes all the vertices of the graph.
   - It has exactly $V-1$ edges, where V is the number of vertices.
   - It minimizes the sum of the weights of the edges included.

Why Use a Minimum Spanning Tree?

1. **Cost Minimization**:
   - MST is used to minimize the cost in systems where connections have weights (such as costs, distances, or lengths). The MST provides the cheapest way to connect all nodes.
2. **Efficiency**:
   - It provides a way to efficiently connect all nodes in a network with the least amount of wiring or connection cost.

Where is MST Used?

1. **Network Design**:
    - **Telecommunication Networks**: Designing a network that connects a set of locations with the minimum cost for laying cables.
    - **Computer Networks**: Designing a backbone to connect different LANs or WANs with minimum total cable length.
    - **Electrical Grids**: Minimizing the cost of wiring when connecting a set of substations.
2. **Clustering**:
    - MST can be used in clustering algorithms to find clusters in a set of points by removing the longest edges in the MST.
3. **Approximation Algorithms**:
    - Used in approximation algorithms for problems like the Traveling Salesman Problem (TSP).
4. **Image Processing**:
    - Used in image segmentation where the goal is to partition an image into regions.
5. **Civil Engineering**:
    - Designing road networks or water supply networks to ensure the minimum cost of construction.

How to Find an MST?

Two of the most common algorithms to find the MST of a graph are:

1. **Kruskal's Algorithm**:
    - Sort all the edges in the graph by their weight in ascending order.
    - Add edges one by one, starting from the smallest weight, ensuring no cycle is formed, until $V-1$ edges are included.
    - Uses a disjoint-set (union-find) data structure to detect cycles efficiently.
2. **Prim's Algorithm**:

- Start from an arbitrary vertex and grow the MST one edge at a time by adding the smallest edge that connects a vertex in the MST to a vertex outside the MST.
- Uses a priority queue to select the minimum weight edge efficiently.

Method 1:- Prims Algorithm.

Here we will take priority queue for storing the node, it weight,and the node parent and we will take visited array.

```
        int spanningTree(int V, vector<vector<int>> adj[])
        {
                priority_queue<pair<int, int>,
                                vector<pair<int, int> >,
greater<pair<int, int>>> pq;

                vector<int> vis(V, 0);
                // {wt, node}
                pq.push({0, 0});
                int sum = 0;
                while (!pq.empty()) {
                        auto it = pq.top();
                        pq.pop();
                        int node = it.second;
                        int wt = it.first;

                        if (vis[node] == 1) continue;
                        // add it to the mst
                        vis[node] = 1;
```

```
                    sum += wt;
                    for (auto it : adj[node]) {
                        int adjNode = it[0];
                        int edW = it[1];
                        if (!vis[adjNode]) {
                            pq.push({edW, adjNode});
                        }
                    }
                }
            return sum;
        }
```

We have not marked visited inside the for loop as we have to take the smallest one, so there may be a case of a newly updated answer we only marked visited when it comes through the priority queue, there may be a different approach to solve this, but to perform prims algorithms we have to follow this approach.

If it is visited then we do not run for loop.

Method 2:- Kruskal Algorithm

Disjoint Set Union By Rank:-

It is used to determine whether 1 or 2 are from the same component or not with O(1) time complexity.

```
class DisjointSet {
    vector<int> rank, parent;
public:
    DisjointSet(int n) {
        rank.resize(n + 1, 0);
        parent.resize(n + 1);
        for (int i = 0; i <= n; i++) {
            parent[i] = i;
        }
    }

    int findUPar(int node) {
        if (node == parent[node])
            return node;
        return parent[node] = findUPar(parent[node]);
    }

    void unionByRank(int u, int v) {
        int ulp_u = findUPar(u);
        int ulp_v = findUPar(v);
        if (ulp_u == ulp_v) return;
        if (rank[ulp_u] < rank[ulp_v]) {
            parent[ulp_u] = ulp_v;
        }
```

```
        else if (rank[ulp_v] < rank[ulp_u]) {
            parent[ulp_v] = ulp_u;
        }
        else {
            parent[ulp_v] = ulp_u;
            rank[ulp_u]++;
        }
    }
};
int main() {
    DisjointSet ds(7);
    ds.unionByRank(1, 2);
    ds.unionByRank(2, 3);
    ds.unionByRank(4, 5);
    ds.unionByRank(6, 7);
    ds.unionByRank(5, 6);
    // if 3 and 7 same or not
    if (ds.findUPar(3) == ds.findUPar(7)) {
        cout << "Same¥n";
    }
    else cout << "Not same¥n";

    ds.unionByRank(3, 7);

    if (ds.findUPar(3) == ds.findUPar(7)) {
        cout << "Same¥n";
    }
    else cout << "Not same¥n";
```

```
    return 0;
}
```

Disjoint Set Union By Size:-

```
class DisjointSet {
    vector<int> rank, parent;
public:
    DisjointSet(int n) {
        rank.resize(n + 1, 0);
        parent.resize(n + 1);
        for (int i = 0; i <= n; i++) {
            parent[i] = i;
        }
    }

    int findUPar(int node) {
        if (node == parent[node])
            return node;
        return parent[node] = findUPar(parent[node]);
    }

    void unionByRank(int u, int v) {
        int ulp_u = findUPar(u);
        int ulp_v = findUPar(v);
        if (ulp_u == ulp_v) return;
        if (rank[ulp_u] < rank[ulp_v]) {
            parent[ulp_u] = ulp_v;
        }
```

```
        else if (rank[ulp_v] < rank[ulp_u]) {
            parent[ulp_v] = ulp_u;
        }
        else {
            parent[ulp_v] = ulp_u;
            rank[ulp_u]++;
        }
    }
};
int main() {
    DisjointSet ds(7);
    ds.unionByRank(1, 2);
    ds.unionByRank(2, 3);
    ds.unionByRank(4, 5);
    ds.unionByRank(6, 7);
    ds.unionByRank(5, 6);
    // if 3 and 7 same or not
    if (ds.findUPar(3) == ds.findUPar(7)) {
        cout << "Same¥n";
    }
    else cout << "Not same¥n";

    ds.unionByRank(3, 7);

    if (ds.findUPar(3) == ds.findUPar(7)) {
        cout << "Same¥n";
    }
    else cout << "Not same¥n";
```

```
    return 0;
}
```

Now come on to the Kruskal Algorithm:-

First of all sort all the edges according to weight.

Then apply DSU according to the edges listed in sorted order of weights, where we will update the ultimate parents of nodes.

We will add only those nodes that have different ultimate parents.
finally we will get MST because we have sorted the edges according to weight, so we storing the edges with minimum weight from the start.

```
vector<int>rank,par;

    public:
    //Function to find sum of weights of edges of the Minimum Spanning Tree.
    void fn(int n){
        rank.resize(n+1);
        par.resize(n+1);
        for(int i=0;i<=n;i++){
            rank[i]=0;
            par[i]=i;
        }
```

```
        return;
    }

    int findpar(int node){
        if(par[node]==node){
            return node;// yhi ultimate h , becoz iske uper ko
i h hi nhi.
            // becoz shuruat me sb node ka parent wo khud hi t
ha.
        }
        else{
            return par[node]=findpar(par[node]);// iske just u
per h usko pucho
            // kya wo ultimate h
            // bhejna to node ka parent thana.
            //pehle jab return node krr rhe to node update ho
rha tha uska parent nhi..lol
        }
    }

    void unionp(int a,int b){
        int ua=findpar(a); // ultimate parent of a;
        int ub=findpar(b);// ultimate parent of b;
        if(ua==ub){
            return; // ab ek hi ultimate parent h to union nhi
 hoga.
        }
        else if(rank[ua]>rank[ub]){
            // ab hum attach krenge, low rank ko high rank se
            // to low rank ka parent high rank wala bnjaega,
            // to jb findpar() use kroge to ans bhi usi hisab
se milega.
            par[ub]=ua;
            // rank bhadane ki jaroorat nhi h becoz wo waise b
hi bada h ,
            // pr bhadane ko bhada skte ho.
        }
        else if(rank[ub]>rank[ua]){
            par[ua]=ub;
        }
        else if(rank[ua]==rank[ub]){
```

```
            // ab rank bhadana padega kyuki, ab diffrentiation
 krna hoga na
            // ki kaun ultimate parent bnega, waise banane ko
kisi ko bhi
            // bana skte ho.
            par[ua]=ub;
            rank[ub]++;
            // jo parent banega rank usi ka bhadega.
        }
    }

    int spanningTree(int n, vector<vector<int>> adj[])
    {
        // code here
        fn(n);
        vector<vector<int>>arr;
        for(int i=0;i<n;i++){

            for(int j=0;j<adj[i].size();j++){
                vector<int>v;
                v.push_back(adj[i][j][1]);
                v.push_back(i);
                v.push_back(adj[i][j][0]);
                arr.push_back(v);
            }
        }
        sort(arr.begin(),arr.end());
        int s=0;
        for(int i=0;i<arr.size();i++){
            if(findpar(arr[i][1])!=findpar(arr[i][2])){
                s+=arr[i][0];
                unionp(arr[i][1],arr[i][2]);
            }
        }
        return s;
    }
```