ADD ONE TO LINKEDLIST

```
class Solution
{
  public:
  Node* reverse(Node * pt)
  {
    Node* curr=pt, *prev=NULL, *nxt=NULL;
    while(curr!=NULL)
    {

      nxt = curr->next;
      curr->next = prev;
      prev = curr;
      curr = nxt;
    }
    return prev;
  }
  Node* addOne(Node *head)
  {
    // Your Code here
    // return head of list after adding one
    Node *ptr=reverse(head);
    Node* cur=ptr;
    Node* prev=NULL;
    while(cur)
    {
      if(cur->data<9){
      cur->data+=1;
      return reverse(ptr);
      }
    else
    {
      cur->data=0;
      prev=cur;
      cur=cur->next;
    }
    }
    Node* temp=new Node(1);
    prev->next=temp;
    return reverse(ptr);
  }
```

BALANCED TREE OR NOT

```
bool isBalanced(node* root)
{
   int lh;
   int rh;
   if (root == NULL)
      return 1;

   lh = height(root->left);
   rh = height(root->right);

   if (abs(lh - rh) <= 1 && isBalanced(root->left) && isBalanced(root->right))
      return 1;
   return 0;
}
```

BOTTOM VIEW OF A TREE

```
 vector <int> bottomView(Node *root) {
      // Your Code Here
       vector<int>ans;
      if(root==NULL) return ans;
      map<int,int>mp;
      queue<pair<Node*,int>>q;
      q.push({root,0});
      while(!q.empty())
      {
         auto it=q.front();
         q.pop();
         Node* node=it.first;
         int line=it.second;
         mp[line]=node->data;
         if(node->left!=NULL)
         q.push({node->left,line-1});
         if(node->right!=NULL)
         q.push({node->right,line+1});
      }
      for(auto it:mp)
      {
         ans.push_back(it.second);
      }
      return ans;
   }
};
```

BOUNDARY TRAVERSAL OF A TREE

```
class Solution {
public:
bool isLeaf(Node* root)
{
   if(root->left==NULL && root->right==NULL)
   return true;
   return false;
}
void addLeftBoundary(Node* root, vector<int> &v)
{
   Node* curr=root->left;
   while(curr)
   {
      if(!isLeaf(curr))
      v.push_back(curr->data);
      if(curr->left)
      curr=curr->left;
      else
      curr=curr->right;
   }
}
void addLeaves(Node* root, vector<int> &v)
{
   if (isLeaf(root)) {
         v.push_back(root->data);
         return;
      }
      if (root->left) addLeaves(root->left, v);
      if (root->right) addLeaves(root->right, v);
}
void addRightBoundary(Node* root, vector<int> &v)
{
    Node* cur = root->right;
      vector<int> tmp;
      while (cur) {
         if (!isLeaf(cur)) tmp.push_back(cur->data);
         if (cur->right) cur = cur->right;
         else cur = cur->left;
      }
      for (int i = tmp.size()-1; i >= 0; --i) {
         v.push_back(tmp[i]);}
}
   vector <int> boundary(Node *root)
   {
      //Your code here
```

```
        vector<int>v;
        if(!root)return v;
        if(!isLeaf(root))
        v.push_back(root->data);
        addLeftBoundary(root,v);
        addLeaves(root,v);
        addRightBoundary(root,v);
        return v;
    }
};
```

CHECK BST

```
bool isBSTUtil(Node * root,int min,int max)
    {
        if(root==NULL)
        return true;
        if(root->data<min || root->data>max)
        return false;
        return isBSTUtil(root->left,min,root->data-1)&&isBSTUtil(root->right,root->data+1,max);
    }
    bool isBST(Node* root)
    {
          return isBSTUtil(root,INT_MIN,INT_MAX);
    }
```

CONVERT TREE INTO BST

```
void inorder(Node* root,vector<int>&v)
    {
        if(!root) return;
        inorder(root->left,v);
        v.push_back(root->data);
        inorder(root->right,v);
    }
    void inorderbst(Node* root,vector<int>v,int &i)
    {
        if(!root) return;
        inorderbst(root->left,v,i);
        root->data=v[i];
        i++;
        inorderbst(root->right,v,i);


    }
    Node *binaryTreeToBST (Node *root)
    {
```

```
    //Your code goes here
    vector<int>v;
    inorder(root,v);
    sort(v.begin(),v.end());
    int i=0;
    inorderbst(root,v,i);
    return root;
  }
```

COUNT NO OF NODES IN A GIVEN RANGE

```
int inorder(Node* root, int l, int h)
{
  if(!root) return 0;
  if(root->data<l) return inorder(root->right,l,h);
  if(root->data>h) return inorder(root->left,l,h);
  return 1+inorder(root->left,l,h)+inorder(root->right,l,h);
}
int getCount(Node *root, int l, int h)
{
 // your code goes here
 int c=0;
 c=inorder(root,l,h);
 return c;
}
```

BST Kth LARGEST ELEMENT

```
int ans=0;
  void solve(Node* root,int K,int &idx)
  {
    if(!root) return;
    solve(root->right,K,idx);
    if(K==idx)
    {
      ans=root->data;
      idx++;
      return;
    }
    else
    idx++;
    solve(root->left,K,idx);
  }
  int kthLargest(Node *root, int K)
  {
    //Your code here
```

```
    int idx=1;
    ans=-1;
    solve(root,K,idx);
    return ans;
  }
```

BST Kth SMALLEST ELEMENT

```
int ans=0;
  void solve(Node* root,int K,int &idx)
  {
    if(!root) return;
    solve(root->left,K,idx);
    if(K==idx)
    {
      ans=root->data;
      idx++;
      return;
    }
    else
    idx++;
    solve(root->right,K,idx);
  }
  int printKthSmallest(Node *root, int K)
  {
    //Your code here
    int idx=1;
    ans=-1;
    solve(root,K,idx);
    return ans;
  }
```

BINARY SEARCH MAGNETS

```
 double distance(double low, double high, double magnets[], double n)
  {
    while(low<high)
    {
      double mid=(low+high)/2;
      double total_force=0;
      for(int i=0;i<n;i++)
      {
        total_force+=(1/(mid-magnets[i]));
      }
      if(abs(total_force)<0.000001)
      {
```

```
            return mid;
        }
        else if(total_force<0)
        {
            high=mid;
        }
        else
        {
            low=mid;
        }
    }
  }
    void nullPoints(int n, double magnets[], double getAnswer[])
  {
    // Your code goes here
    for(int i=0;i<n-1;i++)
    {
        getAnswer[i]=distance(magnets[i],magnets[i+1],magnets,n);
    }
  }
```

BST DEADEND

```
bool solve(Node* root, int min, int max)
{
   if(!root) return false;
   if(min==max) return true;
   return solve(root->left,min,root->data -1) or solve(root->right,root->data +1,max);
}
bool isDeadEnd(Node *root)
{
   //Your code here
   return solve(root,1,INT_MAX);
}
```

LARGEST BST IN A BINARY TREE

```
 vector<int> solve(Node* root)
   {
      if(!root) return {1,0,INT_MAX,INT_MIN};
      if(!root->left and !root->right) return {1,1,root->data,root->data};
      vector<int> l= solve(root->left);
      vector<int> r= solve(root->right);
      if(l[0] and r[0])
      {
         if(root->data>l[3] and root->data<r[2]){
         int x=l[2];
```

```
        int y=r[3];
        if(x==INT_MAX) x=root->data;
        if(y==INT_MIN) y=root->data;
        return {1,l[1]+r[1]+1,x,y};
      }
      }
      return{0,max(l[1],r[1]),0,0};
  }
  int largestBst(Node *root)
  {
        //Your code here
        vector<int> ans = solve(root);
        return ans[1];
  }
```

BST LCA

```
Node* LCA(Node *root, int n1, int n2)
{
  //Your code here
  if(root==NULL)
  return NULL;
  if(root->data>n1 && root->data>n2)
  return LCA(root->left,n1,n2);
  if(root->data<n1 && root->data<n2)
  return LCA(root->right,n1,n2);
  return root;
}
```

BST MIN MAX

```
int minValue(struct Node *root) {
   // your code here
   if(root==NULL)
   return -1;
   while(root->left!=NULL)
   {
      root=root->left;
   }
   return root->data;
}
```

```
int maxValue(struct Node *root) {
```

```
    // your code here
    if(root==NULL)
    return -1;
    while(root->right!=NULL)
    {
        root=root->right;
    }
    return root->data;
}
```

BST PREDECESSOR AND SUCCESSOR

```
Node* inpre(Node * root)
{
    Node* p=root->left;
    while(p->right) p=p->right;
    return p;
}
Node* insuc(Node * root)
{
    Node* p=root->right;
    while(p->left) p=p->left;
    return p;
}
void findPreSuc(Node* root, Node*& pre, Node*& suc, int key)
{

// Your code goes here
if(!root)
return ;
if(root->key==key)
{
    if(root->left) pre=inpre(root);
    if(root->right) suc=insuc(root);
    return;
}
if(key>root->key)
{
    pre=root;
    findPreSuc(root->right,pre,suc,key);
}
else if(key<root->key)
{
    suc=root;
    findPreSuc(root->left,pre,suc,key);
```

```
}


}
```

BUY AND SELL STOCK AT MOST ONE FOR MAXIMUM PROFIT

```
int maxProfit(vector<int>& prices) {
       int buy=prices[0], maxprofit=0;
       for(int i=1;i<prices.size();i++)
       {
          if(buy>prices[i])
          {
             buy=prices[i];
          }
          else if(prices[i]-buy>maxprofit)
             maxprofit=prices[i]-buy;
       }
       return maxprofit;
   }
```

BUY AND SELL STOCK ANY NO OF TIME FOR MAX PROFIT

```
int maxProfit(vector<int>& prices) {
       int n=prices.size();
       int ans=0;
       for(int i=1;i<n;i++)
       {
          if(prices[i]-prices[i-1]>0)
             ans+=prices[i]-prices[i-1];
       }
       return ans;
   }
```

return the index of days

```
vector<vector<int>>v1;
  for(int i=1;i<A.size();i++)
  {
     if(A[i]>A[i-1])
     {
        vector<int>v;
        v.push_back(i-1);
```

```
        while(A[i]>A[i-1] && i<n)
        {
           i++;
        }

        i--;
        v.push_back(i);
        v1.push_back(v);

     }
  }
```

CATALAN NO

```
        int prefixStrings(int n)
        {
        int mod=1000000007;
    unsigned long int catalan[n + 1];
        catalan[0] = catalan[1] = 1;

   for (int i = 2; i <= n; i++) {
      catalan[i] = 0;
      for (int j = 0; j < i; j++)
         catalan[i] += ((catalan[j]%mod) * (catalan[i - j - 1])%mod)%mod;
   }
   return catalan[n];
        }
```

CHECK FOR SUM TREE IN TREE

```
class Solution
{
   public:
   int f=1;
   int solve(Node* root)
   {
      if(!root)
      return 0;
      if(!root->left && !root->right)
      return root->data;
      if(f==0)
      return 0;
      int a=solve(root->left);
      int b=solve(root->right);
      if(a+b!=root->data)
```

```
        f=0;
        return a+b+root->data;
    }
    bool isSumTree(Node* root)
    {
         // Your code here
         f=1;
         solve(root);
         return f;
    }
};
```

CIRCULAR TOUR

```
 int tour(petrolPump p[],int n)
    {
      //Your code here
        int start=0;
        int reqfuel=0;
        int extrafuel=0;
        for(int i=0;i<n;i++)
        {
            extrafuel+=p[i].petrol-p[i].distance;
            if(extrafuel<0)
            {
                reqfuel+=extrafuel;
                start=i+1;
                extrafuel=0;
            }
        }
        if(extrafuel+reqfuel>=0)
        return start;
        return -1;
    }
```

DP BINOMIAL COFF

```
    int mod=1000000007;
    int C[n + 1][r + 1];
    int i, j;
    if(r>n)
    return 0;
    for (i = 0; i <= n; i++) {
        for (j = 0; j <= min(i, r); j++) {
            // Base Cases
            if (j == 0 || j == i)
```

```
            C[i][j] = 1;


        // Calculate value using previously
        // stored values
        else
            C[i][j] = (C[i - 1][j - 1]%mod + C[i - 1][j]%mod)%mod;
    }
  }


  return C[n][r];
  }
```

DP COIN CHANGE RECURSIVE

```
    if(n==0)
    return 1;
    if(n<0)
    return 0;
    if(m<=0 && n>0)
    return 0;
    return count(S,m,n-S[m-1]) +  count(S,m-1,n);


          DP
            long long int i,j,x,y;
    long long int arr[n+1][m];
    for(i=0;i<m;i++)
    arr[0][i]=1;
    for(i=1;i<n+1;i++)
    {
      for(j=0;j<m;j++)
      {
        x=(i-S[j]>=0)?  arr[i-S[j]][j] : 0;
        y=(j>=1)?  arr[i][j-1] : 0;
        arr[i][j]=x+y;
      }
    }
    return arr[n][m-1];
```

DP COIN GAME WINNER WITH THREE CHOICES

```
int findWinner(int N, int X, int Y)
{
        // Your code goes here
        int dp[N+1];
        dp[0]=0;
        dp[1]=1;
        for(int i=2;i<=N;i++)
        {
           if(i-1>=0  &&  dp[i-1]==0)
           dp[i]=1;
           else if (i-X>=0 && dp[i-X]==0)
           dp[i]=1;
           else if(i-Y>=0  &&  dp[i-Y]==0)
           dp[i]=1;
           else
           dp[i]=0;
        }
        return dp[N];
}
```

COUNT AND SAY

```
if (n==1)
    return "1";
  if(n==2)
    return "11";
  string s="11";
  for(int i=3;i<=n;i++)
  {
    string t="";
    s=s+&;
    c=1;
    for(int j=1;j<s.length();j++)
    {
      if(s[j]!=s[j-1])
      {
        t=t+to_string(c);
        t=t+s[j-1];
```

```
            c=1;
          }
          else
          {
            c++;
          }
          s=t;
        }
      }

      return s;
    }
```

COUNT PALENDROMIC SUBSCQUENCE RECURSIVE

```
long long int fun(int i,int j,string s)
  {
    if(i>j)
    return 0;
    if(i==j)
    return 1;
    if(dp[i][j]!=-1) return dp[i][j]%1000000007;
    if(s[i]==s[j]) return dp[i][j]=(fun(i+1,j,s)%1000000007 + fun(i,j-1,s)%1000000007 + 1)%1000000007;
    else
    return dp[i][j]=(fun(i+1,j,s)%1000000007+fun(i,j-1,s)%1000000007-fun(i+1,j-1,s)%1000000007)%1000000007;
  }
  long long int  countPS(string S)
  {
    //Your code here

    long long int n=S.length();
    dp[n][n];
    memset(dp,-1,sizeof(dp));
    return fun(0,n-1,S);
  }
```

DP

```
long long int  countPS(string S)
  {
    long long int len=S.length();
    long long int dp[len+1][len+1];
    for(long long int i=0;i<=len;i++)
    {
      for(long long int j=0;j<=len;j++)
      {
```

```
            if(i==j)
            dp[i][j]=1;
            else
            dp[i][j]=0;
          }
      }
      long long int j=1;
      while(j<=len)
      {
         for(long long int i=0;i<=len-j;i++)
      {
         if(S[i]==S[i+j-1])
         dp[i][i+j]=dp[i+1][i+j]+dp[i][i+j-1];
         else
         dp[i][i+j]=dp[i+1][i+j]+dp[i][i+j-1]-dp[i+1][i+j-1];
      }
      j++;
      }
      return dp[0][len]-1;
   }
```

DP COUNT SET BITS FROM 1 TO N

```
 vector<int> countBits(int n) {
       vector<int>ans(n+1);
       ans[0]=0;
       for(int i=1;i<=n;i++)
       {
          ans[i]=ans[i/2]+i%2;
       }
       return ans;

   }
```

DP DISTINCT PALINDROMIC SUBSTRING

```
unordered_set<string> m;
   int solve(string &s,int i,int j,vector<vector<int>> &a)
   {
      if(i>=j) return 1;
      if(a[i][j]!=-1) return a[i][j];
      if(s[i]!=s[j]) return a[i][j]=0;
      return a[i][j]=solve(s,i+1,j-1,a);
   }
    int palindromeSubStrs(string s) {
```

```
    // code here
    vector<vector<int>> a(s.size(),vector<int>(s.size(),-1));
    m.clear();
    for(int i=0;i<s.size();i++)
    {
       for(int j=i;j<s.size();j++)
       {
          if(solve(s,i,j,a))
          {
             m.insert(s.substr(i,j-i+1));
          }
       }
    }
    return m.size();
  }
```

DP EDIT DISTANCE

```
int editDistance(string s, string t) {
    int sl,tl;
    sl=s.length();
    tl=t.length();
    int arr[sl+1][tl+1];

    for(int i=0;i<=sl;i++)
    {
       for(int j=0;j<=tl;j++)
       {
          if(i==0)
          arr[i][j]=j;
          else if(j==0)
          arr[i][j]=i;
          else if(s[i-1]==t[j-1])
          arr[i][j]=arr[i-1][j-1];
          else
          arr[i][j]=  1+min(arr[i-1][j-1],min(arr[i][j-1],arr[i-1][j]))  ;
       }
    }

    return arr[sl][tl];
  }
```

DP EGG DROP

```
int eggDrop(int n, int k)
```

```
{


    int eggFloor[n + 1][k + 1];
int res;
int i, j, x;

for (i = 1; i <= n; i++) {
  eggFloor[i][1] = 1;
  eggFloor[i][0] = 0;
}



for (j = 1; j <= k; j++)
  eggFloor[1][j] = j;



for (i = 2; i <= n; i++) {
  for (j = 2; j <= k; j++) {
    eggFloor[i][j] = INT_MAX;
    for (x = 1; x <= j; x++) {
      res = 1 + max(
              eggFloor[i - 1][x - 1],
              eggFloor[i][j - x]);
      if (res < eggFloor[i][j])
        eggFloor[i][j] = res;
    }
  }
}
return eggFloor[n][k];
}
```

DP FRIENDS PAIRING PROBLEM

```
int countFriendsPairings(int n)
{
   // code here
  dp[n+1];
  dp[0]=0;
  dp[1]=1;
  dp[2]=2;
  int mod=1000000007;
  for(long long int i=3;i<=n;i++)
  {
    dp[i]=(dp[i-1]%mod+((i-1)%mod*dp[i-2]%mod))%mod;
  }
```

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

DP INTERLEAVED STRING

```
int dp[1001][1001];
  /*You are required to complete this method */
  bool solve(string A,string B, string C,int n,int m,int len)
  {
     if(len==0)
     return 1;
     if(dp[n][m]!=-1)
     return dp[n][m];
     int a=0,b=0;
     if(n-1>=0 && A[n-1]==C[len-1])
     a=solve(A,B,C,n-1,m,len-1);
     if(m-1>=0 && B[m-1]==C[len-1])
     b=solve(A,B,C,n,m-1,len-1);
     return dp[n][m]=a || b;
  }
  bool isInterleave(string A, string B, string C)
  {
     int n=A.length();
     int m=B.length();
     int len=C.length();
     if(m+n!=len)
     return 0;
     dp[n][m];
     memset(dp,-1,sizeof(dp));
     return solve(A,B,C,n,m,len);
  }
```

DP 0 1 KNAPSACK(RECURSIVE)

```
if(W==0 || n==0)
     return 0;
     int ans=0;
     if(W-wt[0]>=0)
     {
        int call1=knapSack(W-wt[0],wt+1,val+1,n-1)+val[0];
        ans=max(ans,call1);
     }
     int call2=knapSack(W,wt+1,val+1,n-1);
```

```
    ans=max(call2,ans);
    return ans;
```

DP Solution

```
            int knapSack(int W, int wt[], int val[], int n)
{
    int arr[n+1][W+1];
    for(int i=0;i<W+1;i++)
    arr[0][i]=0;
    for(int i=0;i<n+1;i++)
    arr[i][0]=0;
    for(int i=1;i<=n;i++)
    {
       for(int j=1;j<=W;j++)
       {
          if(j<wt[i-1])
          arr[i][j]=arr[i-1][j];
          else
          {
             arr[i][j]=max(arr[i-1][j],val[i-1]+arr[i-1][i-wt[j]]);
          }
       }
    }

    return arr[n][W];
}
```

DP LCS

```
int lcs(int x, int y, string s1, string s2)
{
    int arr[x+1][y+1];
    for(int i=0;i<=x;i++)
    {
       for(int j=0;j<=y;j++)
       {
          if(i==0 || j==0)
          arr[i][j]=0;
          else if(s1[i-1]==s2[j-1])
          arr[i][j]=1+arr[i-1][j-1];
          else
          arr[i][j]=max(arr[i-1][j],arr[i][j-1]);
       }
    }
```

```
    return arr[x][y];
  }
```

DP LONGEST PALENDROMIC SUBSTRING

```
string longestPalindrome(string S){
    // code here
    int len=S.length();
    int dp[len][len];
    for(int i=0;i<len;i++)
    {
       for(int j=0;j<len;j++)
       {
          dp[i][j]=0;;
       }
    }
    for(int i=0;i<len;i++)
    {
       dp[i][i]=1;
    }
    for(int i=0;i<len-1;i++)
    {
       if(S[i]==S[i+1])
       dp[i][i+1]=1;
       else
       dp[i][i+1]=0;
    }
    int j=2;
    while(j<=len-1)
    {
       for(int i=0;i<len-2;i++)
       {
          if(S[i]==S[i+j]  &&  dp[i+1][i+j-1]==1)
          {
             dp[i][i+j]=1;
          }
          else
          dp[i][i+j]=0;
       }
       j++;
    }
    j=len-1;
    int start=0;
```

```
        int end=0;
        while(j>=0)
        {
            for(int i=0;i<=len;i++)
            {
                if(dp[i][j+i]==1)
                {
                    start=i;
                    end=i+j;
                    j=0;
                    break;
                }
            }
            j--;
        }
         string st=S.substr(start,end-start+1);
        return st;
    }
```

DP LONGEST PALENDROMIC SUBSTRING

```
int longestPalinSubseq(string A) {
        string B=A;
        reverse(A.begin(),A.end());
        int m=A.length();
        int n=B.length();
        int t[m+1][n+1];

        //Base case
        for(int i=0; i<m+1; i++){
            for(int j=0; j<n+1; j++){
                if(i==0 || j==0){
                    t[i][j]=0;
                }
            }
        }

        //cycle detection
        for(int i=1; i<m+1; i++){
            for(int j=1; j<n+1; j++){
                if(A[i-1]==B[j-1]){
```

```
            t[i][j]=1+t[i-1][j-1];
          }
          else{
            t[i][j]=max(t[i][j-1],t[i-1][j]);
          }
        }
      }
      return t[m][n];
    }
```

DP LONGEST REPETING SUBSCQUENCE

```
    int n = str.length();
    int dp[n+1][n+1];
    for (int i=0; i<=n; i++)
      for (int j=0; j<=n; j++)
        dp[i][j] = 0;

    for (int i=1; i<=n; i++)
    {
      for (int j=1; j<=n; j++)
      {

        if (str[i-1] == str[j-1] && i != j)
          dp[i][j] =  1 + dp[i-1][j-1];

        else
          dp[i][j] = max(dp[i][j-1], dp[i-1][j]);
      }
    }
    return dp[n][n];
                        }
```

DP LONGEST SUBSCQUENCE WITH ADJECENT DIFF ONE

```
int longestSubsequence(int N, int A[])
    {
      // code here
      int dp[N];
      for(int i=0;i<N;i++)
      dp[i]=1;
      for(int i=1;i<N;i++)
      for(int j=0;j<i;j++)
      {
        if(A[i]==A[j]+1  ||  A[i]==A[j]-1)
```

```
            dp[i]=max(dp[i],dp[j]+1);
        }
        int ans=0;
        for(int i=0;i<N;i++)
        ans=max(ans,dp[i]);
        return ans;
    }
```

DP MAXIMUM SUM WITH NO TWO ADJECENT ELEMENTS

```
    int findMaxSum(int *arr, int n) {
        // code here
        int dp[n];
        for(int i=0;i<n;i++)
        {
            dp[i]=0;
        }
        dp[0]=arr[0];
        dp[1]=max(arr[0],arr[1]);
        for(int i=2;i<n;i++)
        {
            dp[i]=max(dp[i-1],dp[i-2]+arr[i]);
        }
        return dp[n-1];
    }
```

MAXIMUM SUM SUCH THAT NO THREE SM ARE CONSECUTIVE

```
int main()
{
    int arr[5]={1,2,3,4,5};
    int dp[5];
    dp[0]=arr[0];
    dp[1]=arr[0]+arr[1];
    dp[2]=max(arr[0]+arr[1],max(arr[0]+arr[2],arr[1]+arr[2]));
    for(int i=3;i<5;i++)
    {
        dp[i]=max(dp[i-1],max(dp[i-2]+arr[i],dp[i-3]+arr[i]+arr[i-1]));
    }
    int ans;
    for(int i=0;i<5;i++)
    ans=max(ans,dp[i]);
    cout<<ans;


    return 0;
}
```

DP MAXIMUM SUM INCREASING SUBSCQUENCE

```
int maxSumIS(int arr[], int n)
{
   // Your code goes here
   int dp[n];
   for(int i=1;i<n;i++)
   dp[i]=0;
   dp[0]=arr[0];
   for(int i=1;i<n;i++)
   {
      for(int j=0;j<i;j++)
      {
         if(arr[j]<arr[i])
         {
            dp[i]=max(dp[i],arr[i]+dp[j]);
         }
         else
         {
            dp[i]=max(dp[i],arr[i]);
         }
      }
   }
   int ans=0;
   for(int i=0;i<n;i++)
   {
      ans=max(ans,dp[i]);
   }
   return ans;
}
```

DP MAXIMIZE THE CUT SEGMENTS

```
int maximizeTheCuts(int n, int x, int y, int z)
   {
      //Your code here
       int d[n+1];
     d[0]=0;
     int a[]={x,y,z};

     for(int i=1;i<=n;i++){
        d[i]=-1;

        if(i-x>=0 && d[i-x]!=-1){
           d[i]=max(d[i],d[i-x]+1);
```

```
        }
        if(i-y>=0 && d[i-y]!=-1){
            d[i]=max(d[i],d[i-y]+1);
        }
        if(i-z>=0 && d[i-z]!=-1){
            d[i]=max(d[i],d[i-z]+1);

        }
    }
    if(d[n]==-1)
    return 0;
    return d[n];
}
```

DP NUMERIC KEYPAD

```
long long getCount(int N)
{
        // Your code goes here
        long long dp[10][N+1];
        for(long long i=0;i<=9;i++)
        for(long long j=0;j<=N;j++)
        {
           dp[i][j]=0;
        }
        for(long long i=1;i<=N;i++)
        {
           for(long long j=0;j<=9;j++)
           {

              if(i==2)
              {
                 if(j==0)
                 dp[j][i]=2;
                 if(j==1)
                 dp[j][i]=3;
                 if(j==2)
                 dp[j][i]=4;
                 if(j==3)
                 dp[j][i]=3;
                 if(j==4)
                 dp[j][i]=4;
                 if(j==5)
                 dp[j][i]=5;
```

```
                    if(j==6)
                    dp[j][i]=4;
                    if(j==7)
                    dp[j][i]=3;
                    if(j==8)
                    dp[j][i]=5;
                    if(j==9)
                    dp[j][i]=3;
                }
                else
                {
                    if(j==0)
                    dp[j][i]=dp[0][i-1]+dp[8][i-1];
                    if(j==1)
                    dp[j][i]=dp[1][i-1]+dp[2][i-1]+dp[4][i-1];
                    if(j==2)
                    dp[j][i]=dp[2][i-1]+dp[1][i-1]+dp[3][i-1]+dp[5][i-1];
                    if(j==3)
                    dp[j][i]=dp[3][i-1]+dp[2][i-1]+dp[6][i-1];
                    if(j==4)
                    dp[j][i]=dp[4][i-1]+dp[1][i-1]+dp[5][i-1]+dp[7][i-1];
                    if(j==5)
                    dp[j][i]=dp[5][i-1]+dp[2][i-1]+dp[4][i-1]+dp[6][i-1]+dp[8][i-1];
                    if(j==6)
                    dp[j][i]=dp[6][i-1]+dp[3][i-1]+dp[5][i-1]+dp[9][i-1];
                    if(j==7)
                    dp[j][i]=dp[7][i-1]+dp[4][i-1]+dp[8][i-1];
                    if(j==8)
                    dp[j][i]=dp[8][i-1]+dp[0][i-1]+dp[5][i-1]+dp[7][i-1]+dp[9][i-1];
                    if(j==9)
                    dp[j][i]=dp[9][i-1]+dp[8][i-1]+dp[6][i-1];
                }

            }
        }
        if(N==1)
        return 10;

long long ans=0;
        for(long long i=0;i<=9;i++)
        {
            ans+=dp[i][N];
        }

        return ans;
```

```
        }
```

DP REACH A GIVEN SCORE USING 3 5 10

```
long long int count(long long int n)
{
   long long int dp[n+1];
   dp[0]=1;
   dp[1]=0;
   dp[2]=0;
   for( long long int i=3;i<=n;i++)
   dp[i]=dp[i-3];
   for( long long int i=5;i<=n;i++)
   dp[i]+=dp[i-5];
   for( long long int i=10;i<=n;i++)
   dp[i]+=dp[i-10];

   return dp[n];
}
```

DP ROD CUTTING

```
int cutRod(int price[], int n) {
      //code here
      int dp[n+1];
      dp[0]=0;
      for(int i=1;i<=n;i++)
      {
         int maxi=INT_MIN;
         for(int j=0;j<i;j++)
         {
            maxi=max(maxi,price[j]+dp[i-j-1]);
            dp[i]=maxi;
         }
      }
      return dp[n];
   }
```

DP TOTAL DECODING MESAGE

```
        int CountWays(string str){
                // Code here
                int len=str.length();
                int dp[len+1];
```

```
                dp[0]=1;
                dp[1]=1;
                int c1=0;
                int c2=0;
                if(str[0]=='0')
                return 0;
                for(int i=2;i<=len;i++)
                {
                   dp[i]=0;
                   c1=0;
                   c2=0;
                   if(str[i-1]>'0')
                   {
                      c1=dp[i-1]%1000000007;
                   }
                   if((str[i-2]=='1') || ((str[i-2]=='2')&& str[i-1]<'7'))
                   {
                      c2=dp[i-2]%1000000007;
                   }
                   dp[i]=((c1)%1000000007+(c2)%1000000007)%1000000007;
                }

                return dp[len]%1000000007;
            }
```

DP WILD CARD STRING MATCHING

```
int dp[1001][1001];
   int match(string wild, string patt , int i , int j){
      if(i < 0 && j < 0){
         return 1 ;
      }
      if(i< 0 || j<0){
         return 0 ;
      }
      if(dp[i][j] != -1){
         return dp[i][j] ;
      }
      if(wild[i] == patt[j]){
         return dp[i][j] = match(wild , patt , i-1,j-1);
      }
      if(wild[i] == '?'){
         return dp[i][j] = match(wild , patt , i-1,j-1) ;
      }
      if(wild[i] == '*'){
```

```
    return dp[i][j] = match(wild ,patt ,i-1,j) || match(wild,patt,i-1,j-1) ||match(wild,patt,i,j-1) ;
    }
    return 0 ; // CASE WHERE wild[i] != patt[j]

  }
  bool match(string wild, string pattern)
  {
    // code here
   int n = wild.size() , m = pattern.size() ;
    memset(dp,-1,sizeof(dp)) ;
    return match(wild , pattern , n-1 , m-1) ;
  }
```

FIND TWO NON REPETING NO

```
vector<int> singleNumber(vector<int> nums)
  {
    // Code here.
    vector<int> ans;
    int n=nums.size();
    int XOR=nums[0];
    for(int i=1;i<n;i++)
    {
      XOR = XOR^nums[i];
    }
    int rightbit= XOR & ~(XOR-1);
    int x,y;
    x=0;
    y=0;
    for(int i=0;i<n;i++)
    {
      if(nums[i]&rightbit)
      {
      x=x ^ nums[i];

      }
      else
      {
      y=y ^ nums[i];

      }
    }
    ans.push_back(x);
    ans.push_back(y);
```

```
    sort(ans.begin(),ans.end());
    return ans;
```

FIND DUPLICATE NO IN AN ARRAY

```
int findDuplicate(vector<int>& nums) {
    while (nums[0] != nums[nums[0]])
      swap(nums[0], nums[nums[0]]);
    return nums[0];
  }
```

GRAPH ALIEN DICT

```
string findOrder(string dict[], int N, int K) {
    //code here
    vector<vector<int>> adj(K);
    vector<int> deg(K);

    for (int i = 1; i < N; ++i)
        for (int j = 0; j < dict[i - 1].size() && j < dict[i].size(); ++j)
            if (dict[i - 1][j] != dict[i][j])
            {
                adj[dict[i - 1][j] - 'a'].push_back(dict[i][j] - 'a');
                ++ deg[dict[i][j] - 'a'];
                break;
            }

    string ret = "";
            que.push(i);

    while (que.size())
    {
        int u = que.front();
        que.pop();
        ret += u + 'a';

        for (int v : adj[u])
            if (-- deg[v] == 0)
                que.push(v);
  }

  return ret;
  }
```

GRAPH BIPARTITE GRAPH (BFS)

```
bool fun(int src,vector<int>adj[], int color[])
{
   queue<int>q;
   q.push(src);
   color[src]=1;
   while(!q.empty())
   {
      int node=q.front();
      q.pop();
      for(auto it:adj[node])
      {
         if(color[it]==-1)
         {
            color[it]=1-color[node];
            q.push(it);
         }
         else if(color[it]==color[node])
         return false;
      }
   }
   return true;
}
            bool isBipartite(int V, vector<int>adj[]){
               // Code here
               int color [V];
               memset(color,-1,sizeof color);
               for(int i=0;i<V;i++)
               {
                  if(color[i]==-1)
                  {
                     if(!fun(i,adj,color))return false;
                  }
               }
               return true;
            }

};
```

DFS

```
bool fun(int node,vector<int>adj[], int color[])
{
```

```
    if(color[node]==-1)
    {
        color[node]=1;
    }
    for(auto it: adj[node])
    {
        if(color[it]==-1)
        {
            color[it]=1-color[node];
            if(!fun(it,adj,color)) return false;
        }if(color[it]==color[node]) return false;
    }
    return true;
}
                bool isBipartite(int V, vector<int>adj[]){
                    // Code here
                    int color [V];
                    memset(color,-1,sizeof color);
                    for(int i=0;i<V;i++)
                    {
                        if(color[i]==-1)
                        {
                            if(!fun(i,adj,color))return false;
                        }
                    }
                    return true;
                }

};
```

GRAPH COURSE SCHEDULE

```
 vector<int> findOrder(int n, int m, vector<vector<int>> prerequisites)
    {
        vector<int>ans;
       vector<int>indeg(n,0);
       vector<int>g[n];
       queue<int>q;
       for(int i=0;i<prerequisites.size();i++){
           int u=prerequisites[i][0];
           int v=prerequisites[i][1];
           g[v].push_back(u);
           indeg[u]++;
       }
       for(int i=0;i<n;i++){
```

```
        if(indeg[i]==0){
            q.push(i);
        }
    }
    while(!q.empty()){
        int f=q.front();
        q.pop();
        ans.push_back(f);
        for(auto nbr:g[f]){
            indeg[nbr]--;
            if(indeg[nbr]==0){
                q.push(nbr);
            }
        }
    }
    if(ans.size()==n){
        return ans;
    }else{
        return {};
    }
  }
```

GRAPH GEEK IN A MAZE

```
int numberOfCells(int n, int m, int r, int c, int u, int d, vector<vector<char>> &mat)
        {
            // Your code goes here
            queue<vector<int>>q;
            q.push({r,c,0,0});
            if(mat[r][c]=='#') return 0;
            mat[r][c]=1;
            int count=1;
            while(!q.empty())
            {
                int x=q.front()[0];
                int y=q.front()[1];
                int up=q.front()[2];
                int down=q.front()[3];
                q.pop();
                if(y-1>=0 && mat[x][y-1]=='.')
                {
                    count++;
                    q.push({x,y-1,up,down});
                    mat[x][y-1]='1';
                }
```

```
                    if(y+1<m && mat[x][y+1]=='.')
                    {
                       count++;
                       q.push({x,y+1,up,down});
                       mat[x][y+1]='1';
                    }
                    if(up!=u && x-1>=0 && mat[x-1][y]=='.')
                    {
                       count++;
                       q.push({x-1,y,up+1,down});
                       mat[x-1][y]='1';
                    }
                    if(down!=d && x+1<n && mat[x+1][y]=='.')
                    {
                       count++;
                       q.push({x+1,y,up,down+1});
                       mat[x+1][y]='1';
                    }
                 }
                 return count;
        }
```

GRAPH LEVEL OF NODES

```
int nodeLevel(int V, vector<int> adj[], int X)
        {
           // code here
           queue<int>q;
           vector<int>vis(V,0);
           int level =1;
           if(X==0)
           return 0;
           q.push(0);
           vis[0]=1;
           while(!q.empty())
           {
              int n=q.size();
              while(n--)
              {
                 int temp=q.front();
                 q.pop();
                 for(auto it:adj[temp])
                 {
                    if(it==X)
```

```
                    return level;
                    if(!vis[it])
                    {
                        q.push(it);
                        vis[it]=1;
                    }
                }
            }
            level++;
        }
        return -1;
    }
```

GRAPH DETECT CYCLE IN DIRECTED GRAPH

```
bool checkcycle(int node,vector<int>adj[],int vis[], int dfsvis[])
  {
    vis[node]=1;
    dfsvis[node]=1;
    for(auto it:adj[node])
    {
      if(!vis[it])
      {
        if(checkcycle(it,adj,vis,dfsvis)) return true;
      }
      else if(dfsvis[it])
      return true;
    }
    dfsvis[node]=0;
    return false;
  }
        bool isPossible(int N, vector<pair<int, int>>& prerequisites) {
          // Code here
           vector<int>adj[N];
     for(auto it:prerequisites)
     adj[it.second].push_back(it.first);
     int vis[N], dfsvis[N];
     for(int i=0;i<N;i++)
     {
       vis[i]=0;
       dfsvis[i]=0;
     }
     for(int i=0;i<N;i++)
     {
       if(!vis[i])
```

```
        {
            if(checkcycle(i,adj,vis,dfsvis))
            return true;
        }
      }
      return false;

            }
```

GRAPH DIJKSTRA ALGO

```
vector <int> dijkstra(int V, vector<vector<int>> adj[], int S)
  {
     // Code here
     vector<int>dist(V, INT_MAX);
    queue<int>q;
    q.push(S);
    dist[S] = 0;
    while(!q.empty()){
       int node = q.front();
       q.pop();
       for(auto i : adj[node]){
          int it = i[0];
          int dis = i[1];
          if(dist[it] > dist[node] + dis){
             dist[it] = dist[node] + dis;
             q.push(it);
          }
       }
    }
    return dist;
  }
```

GRAPH DISTANCE OF NEAREST CELL HAVING 1

```
vector<vector<int>>nearest(vector<vector<int>>grid)
            {
                // Code here
                int n=grid.size();
                int m=grid[0].size();
                vector<vector<int>>ans(n,vector<int>(m,INT_MAX));
                queue<pair<int,int>>q;
                for(int i=0;i<n;i++)
                {
                   for(int j=0;j<m;j++)
                   {
```

```
                if(grid[i][j]==1)
                {
                    q.push({i,j});
                    ans[i][j]=0;
                }
                else
                ans[i][j]=INT_MAX;
            }
        }
        while(!q.empty())
        {
            int i=q.front().first;
            int j=q.front().second;

            if(i-1 >=0  && ans[i][j]+1<ans[i-1][j])
            {
                ans[i-1][j]=ans[i][j]+1;
                q.push({i-1,j});
            }
            if(i+1 <n  && ans[i][j]+1<ans[i+1][j])
            {
                ans[i+1][j]=ans[i][j]+1;
                q.push({i+1,j});
            }
            if(j-1 >=0  && ans[i][j]+1<ans[i][j-1])
            {
                ans[i][j-1]=ans[i][j]+1;
                q.push({i,j-1});
            }
            if(j+1 <m  && ans[i][j]+1<ans[i][j+1])
            {
                ans[i][j+1]=ans[i][j]+1;
                q.push({i,j+1});
            }
             q.pop();
        }
        return ans;
    }
```

GRAPH MIN UNIT PATH

```
void BFS(vector<int> adj[], int N, int src)
{
    int dist[N];
    for(int i = 0;i<N;i++) dist[i] = INT_MAX;
```

```
        queue<int>  q;


        dist[src] = 0;
        q.push(src);


        while(q.empty()==false)
        {
                int node = q.front();
                q.pop();


                for(auto it:adj[node]){
                   if(dist[node] + 1 < dist[it]){
                      dist[it]=dist[node]+1;
                      q.push(it);
                   }
                }
        }
        for(int i = 0;i<N;i++) cout << dist[i] << " ";


}
```

GRAPH MINIMUM SPANNING TREE(PRIM'S ALHO)

```
int spanningTree(int n, vector<vector<int>> adj[])
  {
     vector<bool> vis(n,false);
    int ans=0;
    priority_queue <pair<int,int>, vector<pair<int,int>>, greater<pair<int,int>> > pq;
    pq.push({0,0});
    while(!pq.empty()){
       auto p=pq.top();
       pq.pop();
       int cur=p.second;
       int cost=p.first;
       if(vis[cur])continue;
       vis[cur]=true;
       ans+=cost;
       for(auto p:adj[cur]){
          if(!vis[p[0]]){
             pq.push({p[1],p[0]});
          }
       }
    }
    return ans;
  }
```

GRAPH MOTHER VERTEX

```
vector<int>bfsofgraph(int V,int i, vector<int>adj1[])
{
   vector<int>bfs;
   vector<int>vis(V,0);

      if(!vis[i])
      {
         queue<int>q;
         q.push(i);
         vis[i]=1;
         while(!q.empty())
         {
            int node=q.front();
            q.pop();
            bfs.push_back(node);
            for (auto it:adj1[node])
            {
               if(!vis[it])
               {
                  q.push(it);
                  vis[it]=1;
               }
            }
         }
      }
   return bfs;
}


               int findMotherVertex(int V, vector<int>adj[])
               {
                  for(int i=0;i<V;i++)
                  {
                     vector<int>ans=bfsofgraph(V,i,adj);
                     if(ans.size()==V)
                     return i;
                  }
                  return -1;
               }


};
```

GRAPH FLOOD FILL ALGO

```
void dfs(int i, int j, vector<vector<int>>&vis, vector<vector<int>>&image,int newColor, int n, int m, int oldcolor)
{
   if(i<0 || j<0 || i>=n || j>=m)
   return;
   if(vis[i][j] || image[i][j]!=oldcolor)
   return;
   vis[i][j]=1;
   image[i][j]=newColor;
   dfs(i+1,j,vis,image,newColor,n,m,oldcolor);
   dfs(i-1,j,vis,image,newColor,n,m,oldcolor);
   dfs(i,j+1,vis,image,newColor,n,m,oldcolor);
   dfs(i,j-1,vis,image,newColor,n,m,oldcolor);
}
   vector<vector<int>> floodFill(vector<vector<int>>& image, int sr, int sc, int newColor) {
      // Code here
      int x=image.size();
      int y=image[0].size();
      vector<vector<int>>vis(x,vector<int>(y,0));
      int oldcolor=image[sr][sc];
      dfs(sr,sc,vis,image,newColor,x,y,oldcolor);
      return image;
   }
```

GRAPH FIND NO OF ISLAND

```
void dfs(int i, int j, int n, int m, int vis[501][501],int M[][COL])
{
   if(i<0 || j<0 || i>=n || j>=m)
   return;
   if(M[i][j]==0)
   return;
   if(!vis[i][j])
   {
      vis[i][j]=1;
      dfs(i+1,j,n,m,vis,M);
      dfs(i-1,j,n,m,vis,M);
      dfs(i,j+1,n,m,vis,M);
      dfs(i,j-1,n,m,vis,M);
      dfs(i+1,j+1,n,m,vis,M);
      dfs(i-1,j-1,n,m,vis,M);
      dfs(i+1,j-1,n,m,vis,M);
      dfs(i-1,j+1,n,m,vis,M);
   }
}
int countIslands(int M[][COL], int n, int m) {
```

```
    // your code goes here
    int vis[501][501];
    for(int i=0;i<n;i++)
    {
        for(int j=0;j<m;j++)
        {
            vis[i][j]=0;
        }
    }
    int c=0;
    for(int i=0;i<n;i++)
    {
        for(int j=0;j<m;j++)
        {
            if(!vis[i][j]  &&  M[i][j]==1)
            {
                dfs(i,j,n,m,vis,M);
                c++;
            }
        }
    }
    return c;
}
```

GRAPH NO OF PROVINCES

```
void dfs(vector<int>&visited,vector<vector<int>> adj, int i,int v)
    {
        visited[i]=1;
        for(int j=0;j<v;j++)
        {if(adj[i][j]==1 && !visited[j])
        dfs(visited,adj,j,v);
        }
    }
    int numProvinces(vector<vector<int>> adj, int V) {
        // code here
        int c=0;
        vector<int>visited(V,0);
        for(int i=0;i<V;i++)
        {
            if(!visited[i])
            {c++;
           dfs(visited,adj,i,V);
            }
        }
```

```
    return c;
  }
```

GRAPH NODES AT EEVEN DISTANCE

```
int countOfNodes(vector<int> graph[], int n)
  {
    // code here
    queue<int>q;
    q.push(1);
    vector<bool>vis(n+1,false);
    vis[1]=true;
    vector<int>dis(n+1,0);
    while(!q.empty())
    {
      int node=q.front();
      q.pop();
      for(auto x:graph[node])
      {
        if(!vis[x])
        {
          vis[x]=true;
          q.push(x);
          dis[x]=dis[node]+1;
        }
      }
    }
    int even=0, odd=0;
    for(int i=1;i<=n;i++)
    {
      if(dis[i]%2==0)
      {
        even++;
      }
      else
      {
        odd++;
      }
    }
    return (even*(even-1))/2 + (odd*(odd-1))/2;
  }
```

GRAPH PREREQUISITE TASK

```
bool checkcycle(int node,vector<int>adj[],int vis[], int dfsvis[])
  {
```

```
    vis[node]=1;
    dfsvis[node]=1;
    for(auto it:adj[node])
    {
      if(!vis[it])
      {
        if(checkcycle(it,adj,vis,dfsvis)) return true;
      }
      else if(dfsvis[it])
      return true;
    }
    dfsvis[node]=0;
    return false;
  }
            bool isPossible(int N, vector<pair<int, int>>& prerequisites) {
              // Code here
               vector<int>adj[N];
     for(auto it:prerequisites)
     adj[it.second].push_back(it.first);
     int vis[N], dfsvis[N];
     for(int i=0;i<N;i++)
     {
       vis[i]=0;
       dfsvis[i]=0;
     }
     for(int i=0;i<N;i++)
     {
       if(!vis[i])
       {
         if(checkcycle(i,adj,vis,dfsvis))
         return !true;
       }
     }
     return !false;

           }
```

GRAPH RAT IN A MAZE

```
vector<string>v;
  void dfs(int i,int j,string s,vector<vector<int>> &m,int n,vector<vector<int>>&vis)
  {
    if(i<0 || j<0 || i>=n||j>=n) return;
    if(m[i][j]==0||vis[i][j]==1) return;
```

```
    if(i==n-1 && j==n-1)
    {
    v.push_back(s);
    return;
    }
    vis[i][j]=1;
    dfs(i-1,j,s+'U',m,n,vis);
    dfs(i+1,j,s+'D',m,n,vis);
    dfs(i,j-1,s+'L',m,n,vis);
    dfs(i,j+1,s+'R',m,n,vis);
    vis[i][j]=0;
  }



  vector<string> findPath(vector<vector<int>> &m, int n) {
    // Your code goes here


    v.clear();
     vector<vector<int>>vis(n+1,vector<int>(n+1));
     if(m[0][0]==0 || m[n-1][n-1]==0)
     return v;
     string s="";
     dfs(0,0,s,m,n,vis);
     sort(v.begin(),v.end());
     return v;
  }
```

GRAPH REPLACE O WITH X IN A MATRIX

```
 void change(int x, int y, vector<vector<char>>& mat)
  {
    mat[x][y]='*';
    int dx[]={0,0,1,-1};
    int dy[]={1,-1,0,0};
    for(int i=0;i<4;i++)
    {
       int nx=x+dx[i];
       int ny=y+dy[i];
       if(nx>=0 && nx<mat.size() && ny>=0 && ny<mat[0].size() && mat[nx][ny]=='O')
       change(nx,ny,mat);
    }
  }
  vector<vector<char>> fill(int n, int m, vector<vector<char>> mat)
  {
    // code here
```

```
    for(int i=0;i<n;i++)
    {
       for(int j=0;j<m;j++)
       {
          if(i==0 || i==n-1 || j==0 || j==m-1)
          {
             if(mat[i][j]=='O')
             change(i,j,mat);
          }
       }
    }
    for(int i=0;i<n;i++)
    {
       for(int j=0;j<m;j++)
       {
          if(mat[i][j]=='O')
          mat[i][j]='X';
       }
    }
    for(int i=0;i<n;i++)
    {
       for(int j=0;j<m;j++)
       {
          if(mat[i][j]=='*')
          mat[i][j]='O';
       }
    }
    return mat;
  }
```

GRAPH ROTTEN ORANGES

```
int orangesRotting(vector<vector<int>>& grid) {
    // Code here
    if(grid.empty()) return 0;
    int m=grid.size(), n=grid[0].size(), days=0, tot=0, cnt=0;
    queue<pair<int,int>>rotten;
    for(int i=0;i<m;i++)
    {
       for(int j=0;j<n;j++)
       {
          if(grid[i][j]!=0) tot++;
          if(grid[i][j]==2) rotten.push({i,j});
       }
    }
```

```
        int dx[4]={0,0,1,-1};
        int dy[4]={1,-1,0,0};
        while(!rotten.empty())
        {
            int k=rotten.size();
            cnt+=k;
            while(k--){
            int x=rotten.front().first, y=rotten.front().second;
            rotten.pop();
            for(int i=0;i<4;i++)
            {
                int nx=x+dx[i], ny=y+dy[i];
                if(nx<0 || ny<0 || nx>=m || ny>=n || grid[nx][ny]!=1) continue;
                grid[nx][ny]=2;
                rotten.push({nx,ny});
            }
            }
            if(!rotten.empty()) days++;
        }
        return tot==cnt? days:-1;
    }
};
```

GRAPH SNAKE AND LADDER

```
int minThrow(int N, int arr[]){
        // code here
         unordered_map<int,int>snk;
        unordered_map<int,int>lad;
        for(int i=0;(i+1)<N*2;i+=2)
        {
            if(arr[i]<arr[i+1])
                lad[arr[i]]=arr[i+1];
            else
                snk[arr[i]]=arr[i+1];
        }
        int moves=0;
        queue<int>q;
        q.push(1);
        vector<int>vis(31,0);
        vis[1]=1;
        bool flag=false;
        while(!q.empty() && flag==false)
        {
            int n=q.size();
```

```
        while(n--)
        {
            int t=q.front();
            q.pop();
            for(int i=1;i<=6;i++)
            {
                if(t+i==30)
                {
                    flag=true;
                }
                else if(t+i<=30 and lad[t+i] and !vis[lad[t+i]])
                {
                    vis[lad[t+i]]=1;
                    if(lad[t+i]==30)
                    {
                        flag=true;
                    }
                    q.push(lad[t+i]);
                }
                else if(t+i<=30 and snk[t+i] and !vis[snk[t+i]])
                {
                    vis[snk[t+i]]=1;
                    if(snk[t+i]==30)
                    {
                        flag=true;
                    }
                    q.push(snk[t+i]);
                }
                else if(t+i<=30 and !vis[t+i])
                {
                    vis[t+i]=1;
                    q.push(t+i);
                }
            }
        }
        moves++;
    }
    if(flag==false)
    {
        return -1;
    }
    return moves;
}
```

GRAPH SPIDEY SENSE

```
void bfs(int i, int j, int m, int n,vector<vector<char> >& matrix, vector<vector<int>>& ans)
  {
    queue<pair<int,int>>q;
    q.push({i,j});
    int nx[4]={0,-1,0,1};
    int ny[4]={-1,0,1,0};
    ans[i][j]=0;
    while(!q.empty())
    {
      int I=q.front().first;
      int J=q.front().second;
      q.pop();
      for(int a=0;a<4;a++)
      {
        if(I+nx[a]>=0 && J+ny[a]>=0 && I+nx[a]<m && J+ny[a]<n)
        {
          int newi=I+nx[a];
          int newj=J+ny[a];
          if(matrix[newi][newj]=='O')
          {
            if(ans[newi][newj]>ans[I][J]+1)
            {
              ans[newi][newj]=ans[I][J]+1;
              q.push({newi,newj});
            }
          }
        }
      }
    }
  }

  vector<vector<int> > findDistance(vector<vector<char> >& matrix, int M, int N)
  {
    // Your code goes here
    vector<vector<int>>ans(M,vector<int>(N,INT_MAX));
    for(int i=0;i<M;i++)
    {
      for(int j=0;j<N;j++)
      {
        if(matrix[i][j]=='B')
        {
          bfs(i,j,M,N,matrix,ans);
        }
```

```
        }
    }
    for(int i=0;i<M;i++)
    {
        for(int j=0;j<N;j++)
        {
            if(ans[i][j]==INT_MAX)
            ans[i][j]=-1;
        }
    }
    return ans;

}
```

GRAPH STEPS BY NIGHT

```
int minStepToReachTarget(vector<int>&KnightPos,vector<int>&TargetPos,int N)
        {
            // Code here
            int x1= KnightPos[0];
            int y1= KnightPos[1];
            int x2= TargetPos[0];
            int y2= TargetPos[1];
            if(x1==x2  &&  y1==y2)
            return 0;
            int a[N][N];
            for(int i=0;i<N;i++)
            {
                for(int j=0;j<N;j++)
                {
                    a[i][j]=0;
                }
            }
            queue<pair<int,int>>q;
            q.push({x1-1,y1-1});
            while(!q.empty())
            {
                int i=q.front().first;
                int j=q.front().second;
                q.pop();
                if((i+1)>=0  &&  (i+1)<N &&  (j+2)>=0  &&  (j+2)<N  &&  a[i+1][j+2]==0)
                {
                    a[i+1][j+2]=a[i][j]+1;
                    q.push({i+1,j+2});
                }
```

```
            if((i+1)>=0 && (i+1)<N && (j-2)>=0 && (j-2)<N && a[i+1][j-2]==0)
            {
               a[i+1][j-2]=a[i][j]+1;
               q.push({i+1,j-2});
            }
            if((i-1)>=0 && (i-1)<N && (j+2)>=0 && (j+2)<N && a[i-1][j+2]==0)
            {
               a[i-1][j+2]=a[i][j]+1;
               q.push({i-1,j+2});
            }
            if((i-1)>=0 && (i-1)<N && (j-2)>=0 && (j-2)<N && a[i-1][j-2]==0)
            {
               a[i-1][j-2]=a[i][j]+1;
               q.push({i-1,j-2});
            }
            if((i+2)>=0 && (i+2)<N && (j+1)>=0 && (j+1)<N && a[i+2][j+1]==0)
            {
               a[i+2][j+1]=a[i][j]+1;
               q.push({i+2,j+1});
            }
            if((i+2)>=0 && (i+2)<N && (j-1)>=0 && (j-1)<N && a[i+2][j-1]==0)
            {
               a[i+2][j-1]=a[i][j]+1;
               q.push({i+2,j-1});
            }
            if((i-2)>=0 && (i-2)<N && (j+1)>=0 && (j+1)<N && a[i-2][j+1]==0)
            {
               a[i-2][j+1]=a[i][j]+1;
               q.push({i-2,j+1});
            }
            if((i-2)>=0 && (i-2)<N && (j-1)>=0 && (j-1)<N && a[i-2][j-1]==0)
            {
               a[i-2][j-1]=a[i][j]+1;
               q.push({i-2,j-1});
            }
          }
          return a[x2-1][y2-1];
        }
};
```

GRAPH TOPOLOGICAL SORT(DFS)

```
void fun(int node,vector<int>adj[],vector<int>& vis,stack<int>& st)
        {
          vis[node]=1;
```

```
        for(auto it:adj[node])
        {
            if(!vis[it])
            fun(it,adj,vis,st);
        }
        st.push(node);
    }


    vector<int> topoSort(int V, vector<int> adj[])
    {
        // code here
        stack<int>st;
        vector<int>vis(V,0);
        for(int i=0;i<V;i++)
        {
            if(vis[i]==0)
            {
                fun(i,adj,vis,st);
            }
        }
        vector<int>ans;
        while(!st.empty())
        {
            ans.push_back(st.top());
            st.pop();
        }
        return ans;
    }
```

BFS

```
vector<int> topoSort(int V, vector<int> adj[])
    {
        // code here
        queue<int>q;
        vector<int>indegree(V,0);
        for(int i=0;i<V;i++)
        {
            for(auto it:adj[i])
            indegree[it]++;
        }
        for(int i=0;i<V;i++)
        {
            if(indegree[i]==0)
            q.push(i);
        }
```

```
        vector<int> ans;
        while(!q.empty())
        {
          int node=q.front();
          q.pop();
          ans.push_back(node);
          for(auto it:adj[node])
          {
            indegree[it]--;
            if(indegree[it]==0)
            q.push(it);
          }
        }
        return ans;
      }
```

GRAPH FIND SHORTEST PATH BELLMANFORD ALGO

```
vector <int> bellman_ford(int n, vector<vector<int>> adj, int s) {
    // Code here
    vector<int>dist(n,100000000);
    dist[s]=0;

    for(int i=0;i<n;i++){
      for(auto it: adj){
        int u=it[0] , v=it[1], d=it[2];
        if(dist[u]!=INT_MAX && dist[u]+d < dist[v])
          dist[v] = dist[u]+d;
      }
    }

    return dist;



  }
```

GRAPH X TOTAL SHAPESS

```
void counter(int i, int j, vector<vector<char>> &grid, vector<vector<bool>> &visited)
 {
    if(i < 0 || j < 0 || i >= grid.size() || j >= grid[0].size() ||
       visited[i][j] == true || grid[i][j] == 'O')
       return ;

    visited[i][j] = true;
```

```
    counter(i-1,j,grid,visited);          // Up
    counter(i+1,j,grid,visited);          // Down
    counter(i,j-1,grid,visited);          // Left
    counter(i,j+1,grid,visited);          // Right
  }

  int xShape(vector<vector<char>>& grid)
  {
    // Code here
    int n = grid.size();
    int m = grid[0].size();
    vector<vector<bool>> visited(n,vector<bool> (m,false));
    int count = 0;

    for(int i=0; i<n; i++)
    {
      for(int j=0; j<m; j++)
      {
        if(grid[i][j] == 'X' && visited[i][j] == false)
        {
          counter(i,j,grid,visited);
          count++;
        }
      }
    }

    return count;
  }
```

HEIGHT OF A BINARY TREE

```
int height(struct node * root3)
{
  if(root3==NULL)
  return 0;
  else
  {
  tleft=height(root3->left);
  tright=height(root3->right);
  if(tleft>tright)
  return(tleft+1);
  else return(tright+1);
  }
```

```
}
```

IMP PRINT DUPLICATE OF ARRAY IN O(N)

```
vector<int> findDuplicates(vector<int>& arr) {


    vector<int>ans;
    for(auto i:arr)
    {
       i=abs(i);
       if(arr[i-1]>0)


          arr[i-1]*=-1;
       else
          ans.push_back(i);


    }
    return ans;
  }
```

INSERTION SORT

```
using namespace std;


int main()
{
   int i=0,j=0,n=0;
   int arr[10]={10,9,8,7,6,5,4,3,2,1};
   for(i=1;i<10;i++)
   {
      n=arr[i];
      j=i-1;
      while(j>=0&&arr[j]>n)
      {
         arr[j+1]=arr[j];
         j--;
      }
      arr[j+1]=n;
   }
   for(i=0;i<10;i++)
   cout<<arr[i]<<" ";
   return 0;
}
```

INTERSECTION POINT OF A LINKED LIST

```
int findlength(Node* head)
{
    int c=0;
    while(head)
    {
        c++;
        head=head->next;
    }
    return c;
}
int intersectPoint(Node* head1, Node* head2)
{
    // Your Code Here
    int len1=findlength(head1);
    int len2=findlength(head2);
    int diff=abs(len1-len2);
    // Node* hed1=head1;
    // Node* hed2=head2;
    if(len1>len2)
    {
        while(diff>0)
        {
            head1=head1->next;
            diff--;
        }
        while(head1->next!=head2->next)
        {
            head1=head1->next;
            head2=head2->next;
        }
        return head1->next->data;
    }
    else
    {
        while(diff>0)
        {
            head2=head2->next;
            diff--;
        }
        while(head1->next!=head2->next)
        {
            head1=head1->next;
```

```
        head2=head2->next;
      }
      return head1->next->data;
   }

}
```

GRAPH UNIT AREA OF LARGEST REGION OF 1

```
int n,m;
  int marea(int i,int j,vector<vector<int>>& grid)
  {
     if(i<0 || i>=n || j<0 || j>=m || !grid[i][j] )
     {
        return 0;
     }
     grid[i][j]=0;  // making traversed position as 0
     return 1+marea(i+1,j,grid)+marea(i+1,j+1,grid)+marea(i,j+1,grid)+

     marea(i-1,j+1,grid)+marea(i-1,j,grid)+marea(i-1,j-1,grid)+

     marea(i,j-1,grid)+marea(i+1,j-1,grid);  // for all 8 directions
  }
   int findMaxArea(vector<vector<int>>& grid) {
      // Code here
     int ans=0;
     n=grid.size();
     m=grid[0].size();
     for(int i=0;i<n;i++)
     {
        for(int j=0;j<m;j++)
        {
           if(grid[i][j]==1)
           {
              ans=max(ans,marea(i,j,grid));
           }
        }
     }
     return ans;
   }
```

ISOMORPHIC STRING

```
bool areIsomorphic(string str1, string str2)
   {
```

```
    // Your code here
    int n=str1.length(),m=str2.length();
   if(n!=m) return false;
   int i=1;
   int s1[256]={0},s2[256]={0};
   s1[str1[0]]++,s2[str2[0]]++;
    while(i<n){
      if(s1[str1[i]]==s2[str2[i]]){
          s1[str1[i]]++;
           s2[str2[i]]++;
            i++;
         }

      else{
         return false;
      }
    }
    return true;

  }
```

KADANE'S ALGORITHMS

```
long long maxSubarraySum(int arr[], int n){

    // Your code here
    int sum=0,currsum=INT_MIN;
    for(int i=0;i<n;i++)
    {
       sum=sum+arr[i];
       if(currsum<sum)
       currsum=sum;
       if(sum<0)
       sum=0;
    }
    return currsum;
  }
```

Kth ANCESTOR

```
node * kan(node * root2,int m,int d)
{
  if (root2==NULL){
```

```
    return NULL;
  }
  if ((root2->data==m) ||  (x=kan(root2->left,m,d)) ||  (x=kan(root2->right,m,d))){
  if(d>0)
  d--;
  else if (d==0){
  cout<<root2->data;
  return NULL;
  }
  return root2;
  }

}
```

LCA OF 3 STRING

```
int dp[n1+1][n2+1][n3+1];
  for(int i=0;i<=n1;i++)
  {
    for(int j=0;j<=n2;j++)
    {
      for(int k=0;k<=n3;k++)
      {
        if(i==0 || j==0 || k==0)
        {
          dp[i][j][k]=0;
        }
        else if(A[i-1]==B[j-1]  &&  A[i-1]==C[k-1])
        dp[i][j][k]=dp[i-1][j-1][k-1]+1;
        else
        {
          dp[i][j][k]=max(dp[i-1][j][k],max(dp[i][j-1][k],dp[i][j][k-1]));
        }
      }
    }
  }
  return dp[n1][n2][n3];
```

LCA OF A TREE

```
Node* LCA(Node *root, int n1, int n2)
{
  //Your code here
  if(root==NULL)
  return NULL;
  if(root->data==n1 || root->data==n2)
```

```
  return root;
  Node* leftlca=LCA(root->left,n1,n2);
  Node* rightlca=LCA(root->right,n1,n2);
  if(leftlca && rightlca)
  return root;
  if(leftlca!=NULL)
  return leftlca;
  return rightlca;
}
```

LEFT AND RIGHT VIEW USING QUEUE

```
 queue<Node*>pq;                    left view
  vector<int>v;
  if(root==NULL)
  return v;
  pq.push(root);
  while(!pq.empty())
  {
     int n=pq.size();
     for(int i=1;i<=n;i++)
     {
        Node* top=pq.front();
        pq.pop();
        if(i==1)
        v.push_back(top->data );
        if(top->left != NULL)
        pq.push(top->left);
        if(top->right != NULL)
        pq.push(top->right);

     }
  }
  return v;



 queue<Node*>pq;                    right view
  vector<int>v;
  if(root==NULL)
  return v;
  pq.push(root);
  while(!pq.empty())
  {
```

```
    int n=pq.size();
    for(int i=1;i<=n;i++)
    {
       Node* top=pq.front();
       pq.pop();
       if(i==n)
       v.push_back(top->data );
       if(top->left != NULL)
       pq.push(top->left);
       if(top->right != NULL)
       pq.push(top->right);

    }
  }
  return v;
```

LEFT VIEW

```
void leftview(struct node* root2)
{
   if (root2==NULL)
   return;
   pq.push(root2);
   while(pq.empty()==false)
   {
      int n=pq.size();
      for(int i=1;i<=n;i++)
      {
      struct node* curr=pq.front();
      pq.pop();
      if(i==1)
      cout<<curr->data;
      if(curr->left!=NULL)
      pq.push(curr->left);
      if(curr->right!=NULL)
      pq.push(curr->right);
      }
   }
}
```

recursive sol

```
void solve(Node* root,int l,vector<int> &v)
{
   if(!root) return;
   if(v.size()==l) v.push_back(root->data);
```

```
    solve(root->left,l+1,v);
    solve(root->right,l+1,v);
}
vector<int> leftView(Node *root)
{
   // Your code here
   vector<int>v;
   solve(root,0,v);
   return v;
}
```

LINKED LIST MERGE SORT

```
struct Node* SortedMerge(struct Node* a, struct Node* b)
{
    struct Node* h1=a;
    struct Node* h2=b;

   if (a==NULL)
   return b;
   if (b==NULL)
   return a;
   if(a->data<b->data)
   h1->next=SortedMerge(h1->next,h2);
   else
   h2->next=SortedMerge(h1,h2->next);
}
```

```
 Node sortedMerge(Node head1, Node head2) {

      // Maintain 2 extra pointers head and tail
      Node head = null,tail = null;
      Node ptr1 = head1,ptr2 = head2;

      // Sets the head of the merged linked list
      if(ptr1.data <= ptr2.data){
         head = ptr1;
         tail = ptr1;
         ptr1 = ptr1.next;
      }
      else{
         head = ptr2;
```

```
        tail = ptr2;
        ptr2 = ptr2.next;
      }

      // Loops till one of the List becomes empty
      while(ptr1 != null && ptr2 != null){

        if(ptr1.data <= ptr2.data){
          tail.next = ptr1;
          tail = ptr1;
          ptr1 = ptr1.next;
        }else{
          tail.next = ptr2;
          tail = ptr2;
          ptr2 = ptr2.next;
        }
      }

      if(ptr2 == null){
        tail.next = ptr1;
      }
      else{
        tail.next = ptr2;
      }
      return head;

  }
}
```

CONVERT SORTED LINKEDLIST INTO BST

```
TNode* bst(vector<int>v,int s,int e)
   {
      if(s>e)
      return NULL;
      int m=(s+e+1)/2;
      TNode* temp = new TNode(v[m]);
      temp->left=bst(v,s,m-1);
      temp->right=bst(v,m+1,e);
      return temp;
   }
   TNode* sortedListToBST(LNode *head) {
      //code here
      vector<int>v;
      int c=0;
```

```
        while(head)
        {
            v.push_back(head->data);
            head=head->next;
            c++;
        }
        int start=0;
        int end=c-1;
        return bst(v,start,end);
    }
```

LINKEDLIST MERGE TWO SORTED LINKEDLIST

```
struct Node* SortedMerge(struct Node* head1, struct Node* head2)
{
    if(head1 == NULL){
        return head2;
    }
    if(head2 == NULL){
        return head1;
    }
    struct Node *temp1 = head1;
    struct Node *temp2 = head2;
    struct Node *main = head1;
    if(temp1->data > temp2->data){
        main = temp2;
        temp2 = temp2->next;
    }else{
        temp1 = temp1->next;
    }
    struct Node *curr = main;
    while(temp1 && temp2){
        if(temp1->data > temp2->data){
            curr->next = temp2;
            temp2 = temp2->next;
        }else{
            curr->next = temp1;
            temp1 = temp1->next;
        }
        curr = curr->next;
    }
    if(temp1){
        curr->next = temp1;
    }else{
        curr->next = temp2;
```

```
  }
  return main;
}
```

LONGEST COMMON PREFIX KMP ALGO

```
int lps(string s) {
        // Your code goes here
        int len=s.length();
        int arr[len];
        arr[0]=0;

        for(int i=1;i<len;i++)
        {
           int j=arr[i-1];
           while(j>0 && s[i]!=s[j])
           j=arr[j-1];
           if(s[i]==s[j])

              j++;
              arr[i]=j;

        }
        return arr[len-1];
      }
```

LONGEST COMMON PREFIX IN AN ARRAY

```
string longestCommonPrefix (string arr[], int N)
  {
    // your code here
    string ans="";
    sort(arr,arr+N);
    int cnt=0;
    string first=arr[0];
    for(int i=0;i<first.size();++i){
       cnt=0;
       for(int j=1;j<N;++j){
          string next = arr[j];
          if(first[i]==next[i])cnt++;
       }
       if(cnt==N-1)ans+=first[i];
       else
       break;
    }
```

```
    if(ans=="")return "-1";
    else return ans;
  }
```

LONGEST CONSEQUTIVE SUBSCQUENCE

```
unordered_set<int> s;
    int j=0,ans=0;
    for(int i=0;i<N;i++)
    {
       s.insert(arr[i]);
    }
    for(int i=0;i<N;i++)
    {
       if(s.find(arr[i]-1)==s.end())
       {
           j=arr[i];
           while(s.find(j)!=s.end())
           j++;
           ans= max(ans,j-arr[i]);
       }
    }
    return ans;
```

IMP LONGEST INCREASING SUMSCQUENCE

```
 int longestSubsequence(int n, int a[])
  {
     // your code here
     int dp[n+1];
     for(int i=1;i<=n;i++)
     dp[i]=INT_MAX;
     dp[0]=INT_MIN;
     for(int i=0;i<n;i++)
     {
        int idx=upper_bound(dp,dp+n+1,a[i])-dp;
        if(a[i]>dp[idx-1] && a[i]<dp[idx])
        dp[idx]=a[i];
     }
     int ans=0;
     for(int i=n;i>=0;i--)
     {
        if(dp[i]!=INT_MAX)
        {
           ans=i;
           break;
```

```
        }


    }
    return ans;
  }
```

MAP ARRAY SUM DIVISIABILITY PROBLEM

```
bool canPair(vector<int> arr, int k) {
    int n=arr.size();
    if (n & 1)
     return false;

  unordered_map<int, int> freq;

  for (int i = 0; i < n; i++)
    freq[((arr[i] % k) + k) % k]++;

  for (int i = 0; i < n; i++) {

    int rem = ((arr[i] % k) + k) % k;

    if (2 * rem == k) {

      if (freq[rem] % 2 != 0)
        return false;
    }

    else if (rem == 0) {
      if (freq[rem] & 1)
        return false;
    }

    else if (freq[rem] != freq[k - rem])
      return false;
  }
  return true;
  }
```

MAP COUNT DIST ELEMENT IN EVERY WINDOW

```
vector <int> countDistinct (int arr[], int n, int k)
  {
    //code here.
    vector<int>ans;
    unordered_map<int,int>mp;
```

```
    int count=0;
    for(int i=0;i<k;i++)
    {
        if(mp[arr[i]]==0)
        {
            count++;
        }
        mp[arr[i]]++;
    }
    ans.push_back(count);
    for(int i=k;i<n;i++)
    {
        if(mp[arr[i-k]]==1)
        {
            count--;
        }
        mp[arr[i-k]]--;
        if(mp[arr[i]]==0)
        count++;
        mp[arr[i]]++;
        ans.push_back(count);
    }
    return ans;
}
```

MAP COUNT DIST PAIR WITH DIFF K

```
int TotalPairs(vector<int>nums, int k){
                // Code here
                unordered_map<int,int>mp;
  for(auto i:nums)
     mp[i]++;
 if(k==0){
    int cnt=0;
    for(auto i:mp)
        if(i.second>1) cnt++;
     return cnt;
 }else{
    int cnt=0;
    for(auto i:mp){
       if(mp.find(i.first+k)!=mp.end())
           cnt++;
    }
    return cnt;
            }
```

```
            }
```

MAP FIND AND REPLACE IN A STRING

```
string findAndReplace(string S ,int Q, int index[], string sources[], string targets[]) {
       // code here
       unordered_map<int, vector<string>> m;

      for(int i = 0; i < Q; i++)
      {
         if (S.substr(index[i], sources[i].size()) == sources[i])
         {
             m[index[i]].push_back(sources[i]);
             m[index[i]].push_back(targets[i]);
         }
      }

      string ans = "";

      for(int i = 0; i < S.size(); i++)
      {
         if (m.find(i) != m.end())
         {
            ans += m[i][1];
            i += m[i][0].size() - 1;
         }
         else
            ans += S[i];
      }

      return ans;

   }
```

MAP LARGEST SUBARRAY WITH EQUAL NO OF 0'S AND 1'S

```
int maxLen(int arr[], int N)
   {
       // Your code here
        int sum = 0, ans = 0;
      unordered_map<int, int> map;
      map[0] = -1;
      for(int i = 0; i < N; i++){
         if(arr[i] == 0) arr[i] = -1;
         sum += arr[i];
         if(map.count(sum)){
```

```
            ans = max(ans, i-map[sum]);
          }
          else map[sum] = i;
        }
        return ans;
    }
```

MAP LARGEST SUBARRAY WITH SUM 0

```
int maxLen(vector<int>&A, int n)
    {
        // Your code here
        int ans = 0, sum = 0;
        unordered_map<int, int> mp;
        mp[0] = -1;
        for (int i = 0; i < n; i++)
        {
            sum += A[i];
            if (mp.find(sum) != mp.end())
                ans = max(ans, i - mp[sum]);
            else
                mp[sum] = i;
        }
        return ans;
    }
```

MAP LONGEST SUBARRAY WITH SUM DIVISIBLE BY K

```
int longSubarrWthSumDivByK(int arr[], int n, int k)
                {
                    // Complete the function
                    unordered_map<int,int>m;
    m[0]=-1;
    int maxi=0;
    int sum=0;
    for(int i=0;i<n;i++){
        sum+=arr[i];
        int rem=sum%k;
        if(rem<0){
            rem+=k;
        }
        if(m.find(rem)!=m.end()){
            int len=i-m[rem];
            if(len>maxi){
                maxi=len;
            }
```

```
    }
    else if(m.find(rem)==m.end()){
        m[rem]=i;
    }
  }
  return maxi;


            }
```

MAP LONGEST SUBARRAY WITH K

```
int lenOfLongSubarr(int A[],  int N, int K)
  {
    // Complete the function
    int sum=0;
    int ans=0;
    unordered_map<int,int>mp;
    mp[0]=-1;
    for(int i=0;i<N;i++)
    {
        sum+=A[i];
        if(mp.find(sum-K)!=mp.end())
        {
            ans=max(ans,i-mp[sum-K]);
        }
        if(mp.find(sum)==mp.end())
        mp[sum]=i;
    }
    return ans;
  }
```

MAP NO OF PAIR DIVISIBLE BY K

```
long long countKdivPairs(int A[], int n, int K)
  {
    //code here
    long long ans = 0;
    unordered_map<int, int> mp;
    for (int i = 0; i < n; i++) {
        int j = A[i] % K;
        if (j == 0 && mp.find(0) != mp.end()) {
            ans += mp[0];
        }
        else if (mp.find(K - j) != mp.end()) {
            ans += mp[K - j];
        }
```

```
        mp[j]++;
      }
      return ans;
   }
```

MAXIMUM POINTS YO CAN OBTAIN FROM CARD

```
int maxScore(vector<int>& arr, int k) {
      vector<int>sum(arr.size(),0);
      int Sum=0;
      for(int i=0;i<arr.size();i++)
      {
         Sum+=arr[i];
         sum[i]=Sum;
      }
      if(arr.size()==k)
         return Sum;
      int score=0;
      int ans=0;
      for(int i=0;i<=k;i++)
      {
         int j=i+arr.size()-1-k;
         if(i==0)
            ans=sum[j];
         else
            ans=sum[j]-sum[i-1];
         score=max(score,Sum-ans);
      }
      return score;
   }
```

MAP REARRANGE CHARACTER SUCH THAT NO TWO ADJECENT CHARACTER ARE SAME

```
int main() {
            //code
            int t=0;
            cin>>t;
            while(t--)
            {
               map<char,int>mp;
               string s="";
               cin>>s;
               int len=s.length();
               int ans=0;
               for(int i=0;i<len;i++)
```

```
                {
                    mp[s[i]]++;
                }
                for(auto x:mp)
                {
                    ans=max(ans,x.second);
                }
                if(ans<=len-ans+1)
                cout<<1<<endl;
                else
                cout<<0<<endl;
            }
            return 0;
}
```

MAXIMUM SUM RECTANGLE

```
int kadene(int dp[],int C)
  {
     int sum=0,currsum=INT_MIN;
     for(int i=0;i<C;i++)
     {
        sum=sum+dp[i];
        if(currsum<sum)
        currsum=sum;
        if(sum<0)
        sum=0;
     }
     return currsum;
  }
 public:
  int maximumSumRectangle(int R, int C, vector<vector<int>> M) {
  // code here
    int ans=INT_MIN;
    int k=0;
    while(k<R)
    {
     int dp[C];
     for(int a=0;a<C;a++)
    {
       dp[a]=0;
    }
       for(int i=k;i<R;i++)
       {
          for(int j=0;j<C;j++)
```

```
            {
                dp[j]+=M[i][j];
            }
           ans=max(ans,kadene(dp,C));
        }
        k++;
    }
   return ans;
  }
```

MAXIMUM PRODUCT SUBARRAY

```
long long maxProduct(vector<int> arr, int n) {


                long long ma=arr[0];
                long long mi=arr[0];
                long long  ans=arr[0];
                for(int i=1;i<n;i++)
                {
                   if(arr[i]<0)
                   swap(ma,mi);
                   ma=max((long long)arr[i],(long long)arr[i]*ma);
                   mi=min((long long)arr[i],(long long)arr[i]*mi);
                   ans=max(ans,ma);
                }
                return ans;
               }
```

MAX HEAP IN O(1)

```
void MaxHeapify(int arr[], int i, int n)
{
   int l = 2*i + 1;
   int r = 2*i + 2;
   int largest = i;
   if (l < n && arr[l] > arr[i])
      largest = l;
   if (r < n && arr[r] > arr[largest])
      largest = r;
   if (largest != i)
   {
      swap(arr[i], arr[largest]);
      MaxHeapify(arr, largest, n);
   }
}
```

```
// This function basically builds max heap
void convertMaxHeap(int arr[], int n)
{
   // Start from bottommost and rightmost
   // internal mode and heapify all internal
   // modes in bottom up way
   for (int i = (n-2)/2; i >= 0; --i)
      MaxHeapify(arr, i, n);
}
```

MAX HEAP

```
#include <iostream>
using namespace std;
int main() {
		int arr[10]={7,2,6,3,8,9,10,4,5,1};
   for(int i=1;i<10;i++)
   {
     int  j=i;
      while(j>=1)
      {
      int n=j/2;
      if(arr[n]<arr[j])
      {
         int m=0;
         m=arr[j];
         arr[j]=arr[n];
         arr[n]=m;
      }
      j=n;
      }

   }
   for(int i=0;i<10;i++)
   cout<<arr[i]<<" ";
		return 0;
}
```

MAXIMUM PRODUCT SUBARRAY

```
int maxProduct(vector<int>& arr) {
      int n=arr.size();
      auto ma=arr[0];
		auto mi=arr[0];
		auto ans=ma;
```

```
            for(auto i=1;i<n;i++)
            {
               if(arr[i]<0)
               swap(ma,mi);
               ma=max(arr[i],arr[i]*ma);
               mi=min(arr[i],arr[i]*mi);
               ans=max(ans,ma);
            }
            return ans;
  }
```

MEARGE WITHOUT USING EXTRA SPACE

```
void merge(int arr1[], int arr2[], int n, int m) {
              // code here
               int i = n - 1, j = 0;
  while( i >= 0 && j < m){
    if(arr1[i] > arr2[j]){
      int x = arr1[i];
      arr1[i] = arr2[j];
      arr2[j] = x;
    }

    i--;
    j++;
  }
  sort(arr1, arr1 +n);
  sort(arr2, arr2 + m);
            }
```

MAXIMUM DIFF BETWEEN 0 AND 1

```
int maxSubstring(string str)
            {
               // Your code goes here
               int n=str.length();
               int current_sum = 0;
   int max_sum = 0;
   for (int i = 0; i < n; i++) {

      current_sum += (str[i] == '0' ? 1 : -1);

      if (current_sum < 0)
        current_sum = 0;

      max_sum = max(current_sum, max_sum);
```

```
    }
    return max_sum == 0 ? -1 : max_sum;
                }
```

```
                MIN HEAP
#include <iostream>
using namespace std;
int main() {
                int arr[10]={7,2,6,3,8,9,10,4,5,1};
    for(int i=1;i<10;i++)
    {
      int  j=i;
       while(j>=1)
       {
       int n=j/2;
       if(arr[n]>arr[j])
       {
          int m=0;
          m=arr[j];
          arr[j]=arr[n];
          arr[n]=m;
       }
       j=n;
       }

    }
    for(int i=0;i<10;i++)
    cout<<arr[i]<<" ";
                return 0;
}
```

MAXIMUM LENGTH CHAIN PROBLEM

```
bool comp(pair<int,int>a,pair<int,int>b){
return a.second<b.second;

}
int maxChainLen(struct val p[],int n)
{
//Your code
vector<pair<int,int>>v;
for(int i=0;i<n;i++)
{
    v.push_back({p[i].first,p[i].second});
```

```
}
sort(v.begin(),v.end(),comp);
int i=0;
int j=1;
int c=1;
int len=v.size();
while(j<len)
{
   if(v[i].second<v[j].first)
   {
      c++;
      i=j;
      j++;
   }
   else
   j++;
}
return c;
}
```

BINARY SEARCH MINIMUM NO IN SORTED ROTATED MATRIX

```
int minNumber(int arr[], int low, int high)
   {
      // Your code here
      int mid = 0;
     while(low<high)
     {
        if((high - low) == 1)
        {
           if(arr[high]<arr[low]) return arr[high];
           else return arr[low];
        }
        mid = (low+high)/2;
        if(arr[mid] > arr[high])
        {
           low = mid;
        }
        else if(arr[mid]<arr[high])
        {
           high = mid;
        }
     }

   }
```

MINIMUM BRACKET REVERSAL

```
int countRev (string s)
{
   // your code here
    if(s.size()&1) return -1;
  int count=0,ans=0;
  for(char ch:s){
     if(ch=='{'){
        count++;
     }
     else{
        count--;
     }
     if(count<0){
        ans++;
        count=1;
     }

  }
  return ans+count/2;

}
```

MINIMUM NO OF INSERTION AND DELETION LCS VARIATION

```
int minOperations(string str1, string str2)
                {
                   // Your code goes here
                   int x=str1.length();
                   int y=str2.length();
                   int arr[x+1][y+1];
      for(int i=0;i<=x;i++)
      {
         for(int j=0;j<=y;j++)
         {
            if(i==0 || j==0)
            arr[i][j]=0;
            else if(str1[i-1]==str2[j-1])
            arr[i][j]=1+arr[i-1][j-1];
            else
            arr[i][j]=max(arr[i-1][j],arr[i][j-1]);
         }
      }
```

```
        return x+y-2*arr[x][y];


                }
```

MINIMUM NO OF STEPS TO REACH END

```
 int minJumps(int arr[], int n){
        // Your code here
        if(n==1)
        return 0;
        if(arr[0]==0)
        return -1;
        int step=arr[0];
        int maxreach=arr[0];
        int jump=1;
        for(int i=1;i<n;i++)
        {
           if(i==n-1)
           return jump;
           maxreach=max(maxreach,i+arr[i]);
           step--;
           if(step==0)
           {
              jump++;
              if(i>=maxreach)
              return -1;
              step=maxreach-i;
           }
        }
        return -1;
     }
```

MINIMUM SWAPS TO SORT AN ARRAY

```
int minSwaps(vector<int>&nums)
                {
                    // Code here
                    vector<pair<int,int>>v(nums.size());
                    for(int i=0;i<nums.size();i++)
                    v[i]={nums[i],i};
                    sort(v.begin(),v.end());
                    int c=0;
                    for(int i=0;i<nums.size();i++)
                    {
                       if(v[i].second==i)
                       continue;
```

```
                else
                {
                    c++;
                    swap(v[i],v[v[i].second]);
                    i--;
                }
            }
            return c;
        }
```

MIRROR TREE

```
treenode* mirrorTree(node* root)
{
    // Base Case
    if (root == NULL)
        return root;
    node* t = root->left;
    root->left = root->right;
    root->right = t;

    if (root->left)
        mirrorTree(root->left);
    if (root->right)
        mirrorTree(root->right);

  return root;
```

MULTIPLY TWO LINKEDLIST

```
long long  multiplyTwoLists (Node* l1, Node* l2)
{
  //Your code here
     long long a = 0;
   long long b = 0;
   long long  mod = 1000000007;

   while (l1 != NULL){
       a = a*10;
       a = a%mod  + l1->data;
       l1 = l1->next;
   }
   while (l2 != NULL){
       b = b*10;
       b = b%mod  + l2->data;
       l2 = l2->next;
```

```
  }


  long long s = a%mod * b%mod;
  return s;
}
```

N MEETINGS IN A ROOM

```
int maxMeetings(int start[], int end[], int n)
  {
      // Your code here
      pair <int,int> a[n+1];
      for(int i=0;i<n;i++)
      {
         a[i].first=end[i];
         a[i].second=i;
      }
      sort(a,a+n);
      int time_limit=a[0].first;
      vector<int>m;
      int ans=1;
      for(int i=1;i<n;i++)
      {
         if(start[a[i].second]>time_limit)
         {
            ans++;
            time_limit=a[i].first;
         }
      }
      return ans;
  }
```

NEXT GREATER ELEMENT

```
vector<long long> nextLargerElement(vector<long long> arr, int n){
      // Your code here
      vector<long long>v;
     stack<long long>stk;
      for(int i=n-1;i>=0;i--){
         while(!stk.empty()){
            if(stk.top()>arr[i]){
               v.push_back(stk.top());
               stk.push(arr[i]);
               break;
            }
            else{
```

```
            stk.pop();
          }
        }
        if(stk.empty()){
          v.push_back(-1);
          stk.push(arr[i]);
        }

    }
    reverse(v.begin(),v.end());
    return v;
```

NO OF CUSTOMER WHO COULD NOT GET COMPUTER

```
int main()
{
   string s="ABCBCA";
   int com=1;
   int count=0;
   int count1=0;
   if (com==0)
   return s.length();
   int len=s.length();
   map<char,int>mp;
   for(int i=0;i<len;i++)
   {
      if((mp[s[i]]==0)&&(count<com))
      {
         mp[s[i]]++;
         count++;
      }
      else if(mp[s[i]]==1 && count<=com)
      {
         count--;
         mp[s[i]]=0;
      }
      else
      count1++;
   }
   cout<<count1/2<<endl;
   return 0;
}
```

NO OF PALINDROMIC SUBSTRING

```
long long int  countPS(string str)
  {
     //Your code here
     long long int len=str.length();
     if(len==1)
     return 1;
     if(len==2)
     {
        if(str[0]==str[1])
        return 3;
        return 2;
     }
     long long int dp[len+1][len+1];
     for(long long int i=0;i<=len;i++)
     for(long long int j=0;j<=len;j++)
     {
        dp[i][j]=1;
     }
     for(long long int i=0;i<=len-1;i++)
     for(long long int j=1;j<=len;j++)
     {
        if(str[i]==str[j])
        dp[i][j]=1;
        else
        dp[i][j]=0;
     }
     for(long long int i=0;i<=len-2;i++)
     for(long long int j=2;j<=len;j++)
     {
        if(str[i]==str[j]  &&  dp[i+1][j-1]==1)
        dp[i][j]=1;
        else
        dp[i][j]=0;
     }
     int ans=0;
      for(long long int i =0;i<=len;i++)
      for(long long int j=i;j<=len;j++)
      {
          if(dp[i][j]==1)
          ans=(ans+1)%1000000007;
      }

      return ans;
```

RECURSION NO OF PATHS

```
long long int numberOfPaths(int m, int n){
    // code here
    long long int dp[m][n];
    for(long long int i=0;i<m;i++)
    {
      for(long long int j=0;j<n;j++)
      {
        if((i==0) || (j==0))
        dp[i][j]=1;
        else
        dp[i][j]=0;
      }
    }
    for(long long int i=1;i<m;i++)
    {
      for(long long int j=1;j<n;j++)
      {
        dp[i][j]=((dp[i-1][j])%1000000007+(dp[i][j-1])%1000000007)%1000000007;
      }
    }
    return dp[m-1][n-1];
  }
```

LINKEDLIST PARTITION A LINKEDLIST AROUND A GIVEN VALUE

```
struct Node* partition(struct Node* head, int x) {
  // code here
  vector<int>v;
  struct Node* pt1=head;
  struct Node* pt2=head;
  struct Node* pt3=head;
  struct Node* pt4=head;
  if(head==NULL)
  return head;
  while(pt1!=NULL)
  {
    if(pt1->data<x)
    {
      v.push_back(pt1->data);
      pt1=pt1->next;
    }
    else
```

```
    pt1=pt1->next;
  }
   while(pt2!=NULL)
  {
    if(pt2->data==x)
    {
      v.push_back(pt2->data);
      pt2=pt2->next;
    }
    else
    pt2=pt2->next;
  }
   while(pt3!=NULL)
  {
    if(pt3->data>x)
    {
      v.push_back(pt3->data);
      pt3=pt3->next;
    }
    else
    pt3=pt3->next;
  }
  int i=0;
  while(pt4!=NULL)
  {
    pt4->data=v[i];
    pt4=pt4->next;
    i++;
  }
   return head;
}
```

PERMUTATION OF A STRING

```
vector <string> v;
        public:
          void permut(string s,int l,int r )
          {
            if(l>=r)
            {
              v.push_back(s);
              return;
            }
            for(int i=l;i<=r;i++)
            {
```

```
                swap(s[l],s[i]);
                permut(s,l+1,r);
                swap(s[l],s[i]);
            }
        }
                vector<string>find_permutation(string S)
                {
                   // Code here there
                  int n=S.size()-1;
                   int l=0;
                   int r=n;
                   permut(S,l,r);
                   sort(v.begin(),v.end());
                   return v;

                }
};
```

LONGEST EQUAL PREFIX

```
int findIndex(int arr[], int X, int Y, int N)
{
   // Your code goes here
    for(int i = 0; i<N; i++){
      if(arr[i] == X) arr[i] = 1;
      else if(arr[i] == Y) arr[i] = -1;
      else arr[i] = 0;
  }
  int sum = 0, index = -1;
  bool flag = false;
  for(int i = 0; i<N; i++){
      if(arr[i] == 1 || arr[i] == -1){
         sum += arr[i];
         flag = true;
      }
      if(flag && sum == 0) index = i;
  }
  return index;
}
```

PRIFIX SUM LONGEST SUM IN TWO BINARY STRING

```
int longestCommonSum(bool arr1[], bool arr2[], int n) {
        // code here
        unordered_map<int,int> m;
```

```
    int sum = 0;
    int result = 0;

    for(int i = 0; i < n; i++)
    {
        int diff = arr1[i]-arr2[i];
        sum += diff;

        if(sum == 0)
        {
            result = max(result,i+1);
        }

        if(m.find(sum) != m.end())
        {
            result = max(result, i - m[sum]);
        }

        else
        {
            m[sum] = i;
        }
    }

    return result;
  }
```

RECURSION PRINT ALL SUBSCQUENCE OF A STRING

```
using namespace std;
void fun(string st, string sub, int ind)
{
   if(ind >=5)
   {
      cout<<sub<<endl;
      return;
   }
   fun(st,sub,ind+1);
   fun(st,sub+st[ind],ind+1);
}
int main()
{
   string s="ABCDE";
   fun(s,"",0);
```

```
    return 0;
}
```

PRIORITY QUEUE NEARLY SORTED

```
vector <int> nearlySorted(int arr[], int num, int K){
        // Your code here
        priority_queue<int,vector<int>,greater<int>>pq;
        vector<int>ans;
        for(int i=0;i<num;i++)
        {
            if(i<K)
            pq.push(arr[i]);
            else
            {
                pq.push(arr[i]);
                ans.push_back(pq.top());
                pq.pop();
            }
        }
        while(!pq.empty())
        {
            ans.push_back(pq.top());
            pq.pop();
        }
        return ans;
    }
```

PRODUCT ARRAY PUZZLE

```
vector<long long int>ans;
        long long int prod=1;
        int c=0;
        for(int i=0;i<n;i++)
        {
            if(nums[i]==0)
            c++;
        }
        if(c>=2)
        {
            for(int i=0;i<n;i++)
            {
                ans.push_back(0);
            }
            return ans;
```

```
}
else if(c==0)
{
    for(int i=0;i<n;i++)
    {
        prod=prod*nums[i];
    }
    for(int i=0;i<n;i++)
    {
        long long int x=prod/nums[i];
        ans.push_back(x);
    }
    return ans;
}
else
{
    int y=0;
    long long int prod1=1;
    for(int i=0;i<n;i++)
    {
        if(nums[i]==0)
        {
            y=i;
        }
    }
    for(int i=0;i<n;i++)
    {
        if(i!=y)
        {
            prod1=prod1*nums[i];
        }
    }
    for(int i=0;i<n;i++)
    {
        if(i!=y)
        {
            ans.push_back(0);
        }
        else
        {
            ans.push_back(prod1);
        }
    }
    return ans;
}
```

```
  }
```

REARRANGE A LINKEDLIST

```
void rearrangeEvenOdd(Node *head)
  {
    // Your Code here


    Node* odd=head;
    Node* even=head->next;
    Node* evenhead=head->next;
    while(even && even->next)
    {
      odd->next=odd->next->next;
      even->next=even->next->next;
      odd=odd->next;
      even=even->next;
    }
    odd->next=evenhead;
  }
```

RECURSIVELY REMOVE CONSECUTIVE CHARACTERS

```
 string removeConsecutiveCharacter(string S)
  {
    // code here.
    int len=S.length();
    if(len==0  || len==1)
    {
      return S;
    }
    if(S[0]!=S[1])
    return S[0]+removeConsecutiveCharacter(S.substr(1));
    else
    return removeConsecutiveCharacter(S.substr(1));
  }
};
```

REMOVE DUP IN A SORTED LINKEDLIST

```
Node *removeDuplicates(Node *head)
{
 // your code goes here
 Node* ptr=head;
 while(ptr->next!=NULL)
 {
   if(ptr->data==ptr->next->data)
```

```
    ptr->next=ptr->next->next;
    else
    ptr=ptr->next;
 }
 return head;
}
```

REMOVE DUP IN UNSORTED LINKEDLIST

```
void removedup(struct node * head1)
{
   struct node * curr=NULL;
   curr=head1;
   unordered_set<int> seen;
   struct node * prev=NULL;
   while(curr!=NULL)
   {
      if(seen.find(curr->data)!=seen.end())
      {
         prev->next=curr->next;
         delete (curr);
      }
      else
      {
         seen.insert(curr->data);
         prev=curr;
      }
      curr=prev->next;
   }
}
```

REMOVE LOOP FROM A LINKEDLIST

```
void removeLoop(Node* head)
   {
      // code here
      // just remove the loop without losing any nodes
      Node* low=head;
      Node* high=head;
      while(low!=NULL and high!=NULL and high->next!=NULL)
      {
         low=low->next;
         high=high->next->next;
         if(low==high)
         break;
```

```
    }
    if(low==head)
    {
      while(high->next!=low)
      {
        high=high->next;
      }
      high->next=NULL;
    }
    else if(low==high)
    {
      low=head;
      while(low->next!=high->next)
      {
        low=low->next;
        high=high->next;
      }
      high->next=NULL;
    }
  }
```

REVERSE LINKEDLIST IN A GROUP OF SIZE K

```
class Solution
{
  public:
  struct node *reverse (struct node *head, int k)
  {
    // Complete this method
    if(head==NULL)
    return NULL;
    int c=0;
    node* curr=head;
    node* prev=NULL;
    node* nxt=NULL;
    while(curr!=NULL && c<k)
    {
      nxt=curr->next;
      curr->next=prev;
      prev=curr;
      curr=nxt;
      c++;
    }
    if(nxt!=NULL)
    {
```

```
        head->next=reverse(nxt,k);
      }
      return prev;
   }
};
```

REVERSE ALTERNATE NODE IN A LINKEDLIST

```
Node* reverseList(struct Node *head)
   {
      struct Node* curr = head,*prev = NULL,*nxt = NULL;
      while(curr){
         nxt = curr->next;
         curr->next = prev;
         prev = curr;
         curr = nxt;
      }
      return prev;
   }
   void rearrange(struct Node *head)
   {
      //add code here
     Node* odd=head;
     Node* even=head->next;
     Node* evenhead=head->next;
     Node* oddhead=head;
     while(even && even->next)
     {
        odd->next=odd->next->next;
        even->next=even->next->next;
        odd=odd->next;
        even=even->next;
     }
     odd->next=reverseList(evenhead);
   }
```

REVERSE LEVEL ORDER TRAVERSAL

```
void reverselevelorder(struct node* root2)
{
   if (root2==NULL)
   return;
   pq.push(root2);
   while(pq.empty()==false)
   {
      struct node* curr=pq.front();
```

```
        s.push(curr);
        cout<<curr->data;
        pq.pop();
        if(curr->left!=NULL)
        pq.push(curr->left);
        if(curr->right!=NULL)
        pq.push(curr->right);
    }
    cout<<endl;
    while(s.empty()==false)
    {
    struct node* n=s.top();
    cout<<n->data;
    s.pop();
    }
}
```

RIGHT VIEW

```
void leftview(struct node* root2)
{
    if (root2==NULL)
    return;
    pq.push(root2);
    while(pq.empty()==false)
    {
        int n=pq.size();
        for(int i=1;i<=n;i++)
        {
        struct node* curr=pq.front();
        pq.pop();
        if(i==n)
        cout<<curr->data;
        if(curr->left!=NULL)
        pq.push(curr->left);
        if(curr->right!=NULL)
        pq.push(curr->right);
        }
    }
}
```

recursive solution

```
class Solution
{
    public:
```

```
   void solve(Node* root,int l,vector<int> &v)
{
   if(!root) return;
   if(v.size()==l) v.push_back(root->data);
   solve(root->right,l+1,v);
   solve(root->left,l+1,v);
}
   //Function to return list containing elements of right view of binary tree.
   vector<int> rightView(Node *root)
   {
      // Your Code here
       vector<int>v;
  solve(root,0,v);
  return v;
   }
};
```

SLIDING WINDOW COUNT NO OF SUBARRAY WITH SUM < A

```
 long long int solve(vector<int>A,int N,long long int a){
    long long j=0;
    long long ans=0;
    long long sum=0;
    for(int i=0;i<N;i++){
       sum+=A[i];
       while(sum>a){
          sum-=A[j];
          j++;
       }
       ans+=i-j+1;

    }
    return ans;
 }
   long long countSubarray(int N,vector<int> A,long long L,long long R) {
      // code here
      return solve(A,N,R)-solve(A,N,L-1);
   }
```

SLIDING WINDOW MAX OF ALL SUBARR OF SIZE K

```
vector <int> max_of_subarrays(int *arr, int n, int k)
   {
      // your code here
      vector<int>ans;
```

```
    deque<int>dq;
    for(int i=0;i<n;i++)
    {
        if(!dq.empty() && dq.front()==i-k)
        dq.pop_front();
        while(!dq.empty()  &&  arr[dq.back()]<=arr[i])
        dq.pop_back();
        dq.push_back(i);
        if(i>=k-1)
        ans.push_back(arr[dq.front()]);
    }
    return ans;
  }
```

ROTATE A LINKEDLIST

```
Node* rotate(Node* head, int k)
  {
    // Your code here
    if(head==NULL)
    return head;
    else if (head->next==NULL)
    return head;
    int c=0;
    Node* curr2=head;
    while(curr2!=NULL)
    {
        c++;
        curr2=curr2->next;
    }
    //cout<<c;
    if(c==k)
    return head;
    Node* curr=head;
    Node* curr1=head;
    Node* temp;
    k-=1;
    while(curr->next!=NULL)
    {
        curr=curr->next;
    }
    while(k--)
    {
        curr1=curr1->next;
```

```
    }
    temp=curr1->next;
    curr1->next=NULL;
    curr->next=head;
    return temp;
  }
```

SIZE OF A TREE

```
int height(struct node* root2)
{
  if (root2==NULL)
  return 0;
  else return(height(root2->left) + 1 + height(root2->right));
}
```

SLIDING WINDOW EQUIVALENT SUB ARRAY

```
int countDistinctSubarray(int arr[], int n)
  {
    //code here.
    int i=0,j=0;
    int ans=0;
    unordered_map<int,int>mp;
    unordered_set<int>st;
    for(int i=0;i<n;i++)
      st.insert(arr[i]);
    int k=st.size();
    for(int i=0;i<n;i++){
      mp[arr[i]]++;
      while(mp.size()>=k&&j<=i){
        ans+=(n-i);
         mp[arr[j]]--;
        if(mp[arr[j]]==0)
          mp.erase(arr[j]);
        j++;}
       }
    return ans;
  }
```

SLIDING WINDOW FIRST NEGATIVE ELEMENT IN EVERY WINDOW

```
vector<long long> printFirstNegativeInteger(long long int A[],
                    long long int N, long long int K) {

    vector<long long>ans;
    queue<long long>pq;
```

```
    for(int i=0;i<K-1;i++)
    {
      if(A[i]<0)
      pq.push(A[i]);
    }
    for(int i=K-1;i<N;i++)
    {
      if(A[i]<0)
      pq.push(A[i]);
      if(!pq.empty())
      {
        ans.push_back(pq.front());
        if(A[i-K+1]==pq.front())
        {
          pq.pop();
        }
      }
      else
      ans.push_back(0);
    }
    return ans;
```

SLIDING WINDOW MINIMUM SUM OF SUBARRAY <=K

```
int findMaxSubarraySum(long long arr[], int n, long long k)
        {
          // Your code goes here
          long i = 0 , j = 0 , maxSum = 0 , sum = 0 ;

  while(j < n)
  {
    sum +=arr[j];

    if(sum <= k)
      maxSum = max(maxSum , sum);

     else if(sum > k)
     {
       while(sum > k)
       {
         sum -=arr[i];
         i++;
       }

       if(sum <= k)
```

```
            maxSum = max(maxSum , sum);
        }
        j++;
    }
    return maxSum;

                }
```

SLIDING WINDOW LENGTH OF LONGEST DISTINCT SUBSTRING

```
int longestUniqueSubsttr(string S){
        //code
        int len=S.length();
        unordered_map<char,int>mp;
        int i=0,j=1;
        mp[S[0]]=1;
        int ans=1;
        while(j<len)
        {
           if(mp[S[j]]==0)
           {
              mp[S[j]]++;
              ans=max(ans,j-i+1);
              j++;
           }
           else if(mp[S[j]]==1)
           {
              while(mp[S[j]]!=0)
              {
                 mp[S[i]]--;
                 i++;
              }
              mp[S[j]]++;
              j++;
           }

        }
        return ans;
    }
```

SLIDING WINDOW smallest window in a string containing all character of same string

```
int findSubString(string str)
    {
        // Your code goes here
```

```
    if(str.length()==1) return 1 ;
    set<char> s ;

    for(int i=0 ; i<str.length() ; i++)
      s.insert(str[i]) ;
    unordered_map<char,int> m ;
    int i=0, j=i+1, n=s.size(), c=0 ;
    c++ ;
    int mini=INT_MAX ;
    m[str[i]]++ ;
    while(i<=j and j<str.length())
    {
      if(c<n)
      {
        if(m[str[j]]==0) c++ ;
        m[str[j]]++ ;
        j++ ;
      }

      while(c==n)
        {
          mini=min(mini,j-i) ;
          if(m[str[i]]==1) c-- ;
          m[str[i]]-- ;
          i++ ;
        }

    }

    return mini==INT_MAX ? -1 : mini ;
  }
```

smallest window in a string containing all character of other string

```
string smallestWindow (string s, string p)
  {
    // Your code here
    unordered_map<int,int>m;
    for(int i=0;i<p.size();i++)  m[p[i]]++;
    int count=m.size();
    int i=0;
    int mn=s.size();
    string ans="-1";
    for(int j=0;j<s.size();j++){
```

```
        if(m.find(s[j])!=m.end()){
            m[s[j]]--;
            if(m[s[j]]==0) count--;
        }

        if(count==0)    {


         while(count==0)  {
             if(mn>j-i+1){
              ans=s.substr(i,j-i+1);
              mn=j-i+1;
             }
            if(m.find(s[i])!=m.end())  {
                m[s[i]]++;
                if(m[s[i]]==1) count++;
            }
            i++;
        }
        }
       }
        return ans;
    }
};
```

SLIDING WINDOW SMALLEST WINDOW CONTANING 0,1,2

```
int smallestSubstring(string S) {
        // Code here
        int z=0,o=0,t=0,i=0,j=2,n=S.length();
        int ans=INT_MAX;
        for(int i=0;i<3;i++)
        {
            if(S[i]=='0')
            z++;
            if(S[i]=='1')
            o++;
            if(S[i]=='2')
            t++;
        }
        while(j<n)
        {
            if(z>0  &&  o>0  &&  t>0)
            {
                ans=min(ans,j-i+1);
```

```
            i++;
            if(S[i-1]=='0')
            z--;
            if(S[i-1]=='1')
            o--;
            if(S[i-1]=='2')
            t--;
          }
          else
          {
            j++;
            if(S[j]=='0')
            z++;
            if(S[j]=='1')
            o++;
            if(S[j]=='2')
            t++;

          }
        }
        if(ans==INT_MAX)
        return -1;
        else
        return ans;
    }
```

SLIDING WINDOW SUBSTR OF LEN K WITH K-1 DISTINCT ELEMENT

```
int countOfSubstrings(string S, int K) {
        // code here
        int i,c=0,ans=0;
       unordered_map<char,int>m1;
       for(i=0;i<S.size();i++)
       {
          m1[S[i]]++;
          c++;
          if(c==K)
          {
             c=K-1;
             if(m1.size()==K-1)
             {
                ans++;
             }
          //   m1.erase(S[i-K+1]);  //
             m1[S[i-K+1]]--;
```

```
            if(m1[S[i-K+1]] == 0)
            m1.erase(S[i-K+1]);
        }
    }
    return ans;
  }
```

SPIRAL MATRIX

```
  int top=0,down=3,left=0,right=3;
  int dir=0;
  while(top<=down && left <= right)
  {
    if (dir==0)
    {
      for(int i=left;i<=right;i++)
      cout<<arr[top][i]<<" ";
      top++;
    }
    else if(dir==1)
    {
      for(int i=top;i<=down;i++)
      cout<<arr[i][right]<<" ";
      right--;
    }
    else if (dir==2)
    {
      for(int i=right;i>=left;i--)
      cout<<arr[down][i]<<" ";
      down--;
    }
    else
    {
      for(int i=down;i>=top;i--)
      cout<<arr[i][left]<<" ";
      left++;
    }
    dir=(dir+1)%4;
  }
  return 0;
}
```

SORT ALL 0 I 2 IN A LINKEDLIST

```
Node* segregate(Node *head) {
```

```
    // Add code here
    int zeroes=0;
    int ones=0;
    int twos=0;
    Node* temp=head;
    while(temp!=NULL){
        if(temp->data==0){
            zeroes++;
        }else if(temp->data==1){
            ones++;
        }else{
            twos++;
        }
        temp=temp->next;
    }
    temp=head;
    while(temp!=NULL){
        if(zeroes!=0){
            temp->data=0;
            zeroes--;
        }else if(ones!=0){
            temp->data=1;
            ones--;
        }else{
            temp->data=2;
            twos--;
        }
        temp=temp->next;
    }
    return head;
```

STRING PERMUTATION WITH SPACE

```
void fun(vector<string>&ans,string &S,int i, int n, string st)
  {

      if(i+1>=n)
      {
          st+=S[i];
          ans.push_back(st);
          return;
      }
```

```
        fun(ans,S,i+1,n,st+S[i]+" ");
        fun(ans,S,i+1,n,st+S[i]);
    }
    vector<string> permutation(string S){
        // Code Here
        vector<string>ans;
        string st="";
        int i=0;
        int n=S.size();
        fun(ans,S,i,n,st);
        return ans;
    }
```

SORT ALL 0,1,2

```
int main()
{
    int i=0,str=0,end=9,mid=0;
    int arr[10]={1,2,0,0,2,1,2,0,1,1};
    while(mid<=end)
    {
        if(arr[mid]==0)
        {
            swap(arr[str],arr[mid]);
            mid++;
            str++;
        }
        else if(arr[mid]==1)
        mid++;
        else
        {
            swap(arr[mid],arr[end]);
            end--;
        }
    }
}
```

TREE TOP VIEW

```
 vector<int> topView(Node *root)
    {
        //Your code here
        vector<int>ans;
        if(root==NULL) return ans;
        map<int,int>mp;
```

```
    queue<pair<Node*,int>>q;
    q.push({root,0});
    while(!q.empty())
    {
       auto it=q.front();
       q.pop();
       Node* node=it.first;
       int line=it.second;
       if(mp.find(line)==mp.end())
       mp[line]=node->data;
       if(node->left!=NULL)
       q.push({node->left,line-1});
       if(node->right!=NULL)
       q.push({node->right,line+1});
    }
    for(auto it:mp)
    {
       ans.push_back(it.second);
    }
    return ans;
  }
```

TRAPPING RAIN WATER PROBLEM

```
public:
  long long trappingWater(int arr[], int n){
    // code here
    int left[n];
    int right[n];
    left[0]=arr[0];
    right[n-1]=arr[n-1];
    for(long long i=1;i<n;i++)
    {
       left[i]=max(left[i-1],arr[i]);
    }
    for(long long i=n-2;i>=0;i--)
    {
       right[i]=max(right[i+1],arr[i]);
    }
    long long ans=0;
    for(long long i=0;i<n;i++)
    {
       ans+=min(left[i],right[i])-arr[i];
    }
```

```
        return ans;
    }
};
```

}TREE TRAVERSAL

```
void preorder(struct node* root1)
{
    if(root1)
    {
        cout<<root1->data;
        preorder(root1->left);
        preorder(root1->right);
    }
}
void inorder(struct node* root1)
{
    if(root1)
    {
        inorder(root1->left);
        cout<<root1->data;
        inorder(root1->right);
    }
}
void postorder(struct node* root1)
{
    if(root1)
    {
        postorder(root1->left);
        postorder(root1->right);
        cout<<root1->data;
    }
}
```

SUMTREE

```
int sumtree(node * root)
{
    if(root==NULL)
    return 0;
    int oldvalue=root->data;
    root->data=sumtree(root->left)+sumtree(root->right);
    return root->data+ oldvalue;
}
```

TWO SUM

```
bool hasArrayTwoCandidates(int arr[], int n, int x) {
        // code here
        unordered_map<int,int>mp;
        for(int i=0;i<n;i++)
        mp[arr[i]]++;
        for(int i=0;i<n;i++)
        {
           if(mp.find(x-arr[i])!=mp.end())
           {
              if(x-arr[i]==arr[i])
              {
                 if(mp[arr[i]]>=2)
                 return true;
              }
              else
              return true;
           }
        }
        return false;
    }
```

VALID PARENTHISES

```
int len=s.length();
   stack<int>st;
   if(s[0]=='}' || s[0]==')' || s[0]==']')
   return false;
   st.push(s[0]);
   for(int i=1;i<len;i++)
   {
      if(s[i]=='(' || s[i]=='{' || s[i]=='[')
      st.push(s[i]);
      else
      {
         if(st.empty())
         {
            return false;
         }
         else if((s[i]==')' && st.top()=='(') || (s[i]=='}' && st.top()=='{') || (s[i]==']' && st.top()=='['))
         {
            st.pop();
         }
         else
```

```
            return false;
        }
    }
    return (st.empty()==true);
```

WORD BREAK PROBLEM

```
unordered_map<string,int>dp;
    int solve(string s,vector<string>&b)
    {
        int len=s.length();
        if(len==0)
        return 1;
        if(dp[s]!=0)
        return dp[s];
        for(int i=1;i<=len;i++)
        {
            int f=0;
            string ss=s.substr(0,i);
            for(int j=0;j<b.size();j++)
            {
                if(ss.compare(b[j])==0)
                {
                    f=1;
                    break;
                }
            }
            if(f==1 && solve(s.substr(i,len-1),b)==1) return dp[s]=1;
        }
        return dp[s]=0;
    }
    int wordBreak(string A, vector<string> &B) {
        //code here
        int x=solve(A,B);
        if(x==1)
        return 1;
        else
        return 0;
    }
```