precode@tanny411
aysha.kamal7@gmail.com

DEFINES

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

#include<limits.h>
#include <cstdio>
#include <sstream>
#include <cstdlib>
#include <cctype>
#include <cmath>
#include <algorithm>
#include <set>
#include <queue>
#include <stack>
#include <list>
#include <iostream>
#include <fstream>
#include <numeric>
#include <string>
#include <vector>
#include <cstring>
#include <map>
#include <iterator>

#define pb push_back
#define Pb pop_back

#define PI acos(-1.00)
#define pii pair<int,int>
#define ppi pair<pii,int>
#define INTMAX 1<<30
#define MOD 1000000007

#define ll long long
#define llu unsigned long long

#define fs first
#define sc second

#define eps .0000001
#define zero 0.0000000000000001
#define floatless(a,b) ( (a-b)<=eps )
#define floatequal(a,b) ( floatless(a,b) && floatless(b,a) )
#define incircle_r(a,b,c) 2*area(triangle)/perimeter(triangle)
#define circumcircle_R(a,b,c) a*b*c/sqrt(
(a+b+c)*(a+b-c)*(b+c-a)*(c+a-b) )
#define circumcircle_2R_sinelaw a/sinA
#define triarea(a,b,c) sqrt( (a+b+c)*(b+c-a)*(a-b+c)*(a+b-c) )/9
#define mediantoarea(a,b,c) (4/3)*triarea(a,b,c)

using namespace std;

int main()
{
   int t,cas=1;
   double R,r,theta,n;
   cin>>t;
   while(t--)
   {
      cin>>R>>n;
      printf("Case %d: ",cas++);
      theta=PI/n;
      theta=sin(theta);
      printf("%.10lf\n",R*theta/(1+theta));
   }
   return 0;
}
/*----------------------Graph Moves----------------*/
const int fx[]= {+1,-1,+0,+0};
const int fy[]= {+0,+0,+1,-1};
//const int fx[]={+0,+0,+1,-1,-1,+1,-1,+1};   // Kings Move
//const int fy[]={-1,+1,+0,+0,+1,+1,-1,-1};  // Kings Move
//const int fx[]={-2, -2, -1, -1,  1,  1,  2,  2};  // Knights Move
//const int fy[]={-1,  1, -2,  2, -2,  2, -1,  1}; // Knights Move
/*-----------------------------------------------*/

/*----------------------Bitmask------------------*/
//int Set(int N,int pos){return N=N | (1<<pos);}
//int reset(int N,int pos){return N= N & ~(1<<pos);}
//bool check(int N,int pos){return (bool)(N & (1<<pos));}
/*-----------------------------------------------*/
```

TRIE

```
#define mx 1000
///no. of nodes=number of strings*max string size
struct trietree
{
   int tr[mx][26];
   int leaf[mx];
   int nodenumber;
   int root=0;
   trietree()
   {
      memset(tr,-1,sizeof tr);
      memset(leaf,0,sizeof leaf);
      nodenumber=0;
   }
   void insert_string(string s)
   {
```

```
        int node=root;
        for(int i=0;i<s.length();i++)
        {
            if(tr[node][s[i]-'a']==-1)
            {
                tr[node][s[i]-'a']=++nodenumber;
            }
            node=tr[node][s[i]-'a'];
        }
        leaf[node]++;
    }
    int look(string s)
    {
        int node=root;
        for(int i=0;i<s.length();i++)
        {
            if(tr[node][s[i]-'a']==-1) return 0;
            node=tr[node][s[i]-'a'];
        }
        if(leaf[node]>0) return 1;
    }
};

int main()
{
    trietree lala;
    string keys[] = {"hi", "hello", "you", "ekta", "me"};

    for (int i = 0; i < 5; ++i)
                lala.insert_string (keys[i]);

    cout << lala.look("hipls") << endl;
    cout << lala.look("m") << endl;
    return 0;
}
```

TARJAN(SCC)

```
#define maxnode 1000
vector<int>gr[maxnode],disc(maxnode),low(maxnode),scc(maxnode
),stk(maxnode);
///all nodes in an scc will be marked similar value;
int t=0,sccno=0;
stack<int>s;

void dfs_scc(int p,int u)
{
    disc[u]=low[u]=++t;
    s.push(u);
    stk[u]=1;
    for(int i=0;i<gr[u].size();i++)
    {
        int v=gr[u][i];
        if(disc[v]==-1)
        {
            dfs_scc(u,v);
            low[u]=min(low[u],low[v]);
        }
        else if(stk[v]==1) low[u]=min(low[u],disc[v]);///if back-edge
    }
    if(low[u]==disc[u])
    {
        ++sccno;
        while(1)
        {
            int x=s.top();
            s.pop();
            scc[x]=sccno;
            stk[u]=2;///cross-edge now
            if(x==u) break;
        }
    }
}

void SCC(int n)
{
    /*
    Do necessary init and clear
    */
    for(int i=0;i<n+3;i++)
    {
        scc[i]=stk[i]=low[i]=0;
        disc[i]=-1;
    }

    sccno=0;
    for(int i=0;i<n;i++)
    {
        t=0;
        if(disc[i]==-1) dfs_scc(-1,i);
    }
}

int main()
{
    int n,m;
    cin>>n>>m;
    while(m--)
    {
```

```
        int x,y;
        cin>>x>>y;
        gr[x].pb(y);
    }
    SCC(n);
    for(int i=1;i<=sccno;i++)
    {
        for(int j=0;j<n;j++)
        {
            if(scc[j]==i) cout<<" "<<j;
        }
        cout<<endl;
    }
}
```

TARJAN(ARTICULATION POINT)

```
#define maxnode 10000+7
vector<int>gr[maxnode];
vector<int>arti(maxnode),low(maxnode),disc(maxnode,-1);
///true if is an articulation point
int t=0;

void dfs(int p,int u)
{
    disc[u]=low[u]=++t;
    int child=0;
    for(int i=0;i<gr[u].size();i++)
    {
        int v=gr[u][i];
        if(v==p) continue;
        if(disc[v]==-1)
        {
            dfs(u,v);
            low[u]=min(low[u],low[v]);
            if(p!=-1 && disc[u]<=low[v]) arti[u]=1;
            child++;
        }
        else
        {
            low[u]=min(low[u],disc[v]);
        }
    }
    if(p==-1 && child>1) arti[u]=1;
}

void ArticulationPoint(int n)
{
    /*
    Do necessary init and clear
    */
    for(int i=0;i<n+3;i++)
    {
        arti[i]=low[i]=0;
        disc[i]=-1;
    }

    for(int i=0;i<n;i++)
    {
        t=0;
        if(disc[i]==-1) dfs(-1,i);
    }
}

int main()
{
    int n,m;
    cin>>n>>m;
    /*
    Clear Graph
    */
    while(m--)
    {
        int x,y;
        cin>>x>>y;
        gr[x].pb(y);
        gr[y].pb(x);
    }

    ArticulationPoint(n);
    for(int i=0;i<n;i++) if(arti[i]) cout<<" "<<i;
    return 0;
}
```

TARJAN(ARTICULATION BRIDGE)

```
#define maxnode 1000
vector<int>gr[maxnode];
vector<int>disc(maxnode,-1),low(maxnode);
vector<pii>bridge;
int t=0;

void dfs(int p,int u)
{
    disc[u]=low[u]=++t;
    for(int i=0; i<gr[u].size(); i++)
    {
        int v=gr[u][i];
        if(v==p) continue;
        else if(disc[v]==-1)
```

```
        {
            dfs(u,v);
            low[u]=min(low[u],low[v]);
            if(disc[u]<low[v]) bridge.pb(pii(u,v));
        }
        else low[u]=min(low[u],disc[v]);
    }
}

void ArticulationBridge(int n)
{
    /*
    Do necessary init and clear
    */
    for(int i=0; i<n+3; i++)
    {
        low[i]=0;
        disc[i]=-1;
    }
    bridge.clear();

    for(int i=0; i<n; i++)
    {
        t=0;
        if(disc[i]==-1) dfs(-1,i);
    }
}

int main()
{
    int n,m;
    cin>>n>>m;
    /*
    Clear Graph
    */
    while(m--)
    {
        int x,y;
        cin>>x>>y;
        gr[x].pb(y);
        gr[y].pb(x);
    }
    ArticulationBridge(n);
    for(int i=0;i<bridge.size();i++)
    {
        cout<<bridge[i].fs<<" "<<bridge[i].sc<<endl;
    }
    return 0;
}
```

TABLE KNAPSACK

```
int main()
{
    int n;
    cin>>n;
    vector<int>cost(n+1),weight(n+1);
    for(int i=1;i<=n;i++) cin>>cost[i];
    for(int i=1;i<=n;i++) cin>>weight[i];
    int cap;
    cin>>cap;
    int tbl[n+1][cap+1];
    for(int i=0;i<=n;i++)
    {
        for(int j=0;j<=cap;j++)
        {
            if(i==0 || j==0) tbl[i][j]=0;
            else
            {
                if(j-weight[i]>-1)
tbl[i][j]=max(tbl[i-1][j],cost[i]+tbl[i-1][j-weight[i]]);
                else tbl[i][j]=tbl[i-1][j];
            }
        }
    }
    for(int i=0;i<=n;i++)
    {
        for(int j=0;j<=cap;j++)
        {
            cout<<" "<<tbl[i][j];
        }
        cout<<endl;
    }
    cout<<tbl[n][cap]<<endl;
    return 0;
}
```

PRINTING KNAPSACK SOLUTION

```
///code not runnable, only for reference
int dir[][]={{-1}};
int dp[][]={{-1}};
int func(int i,int w) //i নম্বর আইটেম নিয়ে চেষ্টা করা হচ্ছে,w ওজনের জিনিস নেয়া হয়েছে
{
    .....
    //BASE CASE
    .....
```

precode@tanny411
aysha.kamal7@gmail.com


```
    if(w+weight[i]<=CAP) //i নম্বর জিনিসটি নিবো
        profit1=cost[i]+func(i+1,w+weight[i])
    else
        profit1=0;
    profit2=func(i+1,w) // i নম্বর জিনিসটি নিবো না
    if(profit1>profit2){dir[i][w]=1; return dp[i][w]=profit1;}
    else {dir[i][w]=2; return dp[i][w]=profit2;}
}
```

SUM OF ALL SUBARRAY

```
arr[] = [1, 2, 3], n = 3
All subarrays :  [1], [1, 2], [1, 2, 3],
                 [2], [2, 3], [3]
here first element 'arr[0]' appears 3 times
     second element 'arr[1]' appears 4 times
     third element 'arr[2]' appears 3 times

Every element arr[i] appears in two types of subsets:
i)  In sybarrays beginning with arr[i]. There are
    (n-i) such subsets. For example [2] appears
    in [2] and [2, 3].
ii) In (n-i)*i subarrays where this element is not
    first element. For example [2] appears in
    [1, 2] and [1, 2, 3].

Total of above (i) and (ii) = (n-i) + (n-i)*i
                            = (n-i)(i+1)

For arr[] = {1, 2, 3}, sum of subarrays is:
  arr[0] * ( 0 + 1 ) * ( 3 - 0 ) +
  arr[1] * ( 1 + 1 ) * ( 3 - 1 ) +
  arr[2] * ( 2 + 1 ) * ( 3 - 2 )

= 1*3 + 2*4 + 3*3
= 20

///O(n)
```

SUFFIX ARRAY

```
///O( n(log n)^2 )

#define LEN 1000 + 5
#define LG log2(LEN) + 2

int suffara[LEN],P[LEN][(int)LG];
pair< pair<int,int> , int > L[LEN]; ///prev,now and index

bool suff_cmp(pair< pair<int,int> , int >a,pair< pair<int,int> , int >b){
    if(a.fs.fs!=b.fs.fs) return a.fs.fs<b.fs.fs;
    else return a.fs.sc<b.fs.sc;
}

void makeSuffAra(string s){
    int n=s.length();
    for(int i=0;i<n;i++){
        P[i][0]=s[i]-'a';
    }

    int step=1;
    for(int j=1;j<n;step++,j<<=1){///power(step-1)<n
        for(int i=0;i<n;i++){
            L[i].sc=i;
            L[i].fs.fs=P[i][step-1];
            L[i].fs.sc=(i+j<n)?P[i+j][step-1]:-1;
        }
        sort(L,L+n,suff_cmp);///perform counting or radix sort here
to reduce complexity
        P[L[0].sc][step]=0;
        for(int i=1;i<n;i++){
            P[L[i].sc][step]=(L[i].fs==L[i-1].fs)?P[L[i-1].sc][step]:i;
        }
    }
    step--;
    for(int i=0;i<n;i++){
        suffara[P[i][step]]=i;
    }
}

int main()
{
    string s;
    cin>>s;
    transform(s.begin(), s.end(), s.begin(), ::tolower);
    makeSuffAra(s);
    for(int i=0;i<s.length();i++){
        cout<<suffara[i]<<endl;
    }
    return 0;
}
```

STIRLING NUMBER OF 2nd KIND

```
// A Dynamic Programming based C++ program to count
// number of partitions of a set with n elements
// into k subsets
#include<iostream>
using namespace std;
```

```
// Returns count of different partitions of n
// elements in k subsets
int countP(int n, int k)
{
  // Table to store results of subproblems
  int dp[n+1][k+1];

  // Base cases
  for (int i=0; i<=n; i++)
    dp[i][0] = 0;
  for (int i=0; i<=k; i++)
    dp[0][k] = 0;

  // Fill rest of the entries in dp[][]
  // in bottom up manner
  for (int i=1; i<=n; i++)
    for (int j=1; j<=i; j++)
      if (j == 1 || i == j)
        dp[i][j] = 1;
      else
        dp[i][j] = j*dp[i-1][j] + dp[i-1][j-1];

  return dp[n][k];
}

// Driver program
int main()
{
   cout <<  countP(5, 2);
   return 0;
}
```

SIEVE PHI

```
#define MX 1000000
int phi[MX+2],flag[MX+2];
void sievephi()
{
    for(int i=2;i<=MX;i++)
    {
        phi[i]=i;
    }
    phi[1]=1;
    ///by definition
    flag[0]=flag[1]=1;
    for(int i=2;i<=MX;i++)
    {
        if(!flag[i])
        {
            for(int j=i;j<=MX;j+=i)
            {
                flag[j]=1;
                phi[j]=(phi[j]/i)*(i-1);
            }
        }
    }
}
```

SECOND SHORTEST PATH

```
///loj 1099
using namespace std;
vector<pair<int,int> >gr[5002];
int d[2][5002];
int flag[2][5002];

void dijkstra(int s)
{
    priority_queue<pair<int,pair<int,int> > ,vector<
pair<int,pair<int,int> > > ,greater< pair<int,pair<int,int> > > >q;
///dis,t=0(shortest),1(second shortest),u
    d[0][s]=0;
    q.push({0,{0,s}});
    while(!q.empty())
    {
        int t=q.top().sc.fs;
        int u=q.top().sc.sc;
        q.pop();
        if(flag[t][u]) continue;
        flag[t][u]=1;
        for(int i=0;i<gr[u].size();i++)
        {
            int v=gr[u][i].fs;
            int dd=gr[u][i].sc;
            if(d[0][v]>d[t][u]+dd)
            {
                d[1][v]=d[0][v];
                d[0][v]=d[t][u]+dd;
                q.push({d[1][v],{1,v}});
                q.push({d[0][v],{0,v}});
            }
            else if(d[0][v]<d[t][u]+dd && d[1][v]>d[t][u]+dd)///need to
give d[0][v]<d[t][u]+dd becasue the second shortest cant be equal
to shortest,if can be equal dont give this
            {
                d[1][v]=d[t][u]+dd;
                q.push({d[1][v],{1,v}});
            }
        }
    }
```

```
}

int main()
{
   int t,cas=1;
   cin>>t;
   while(t--)
   {
      int n,r;
      cin>>n>>r;
      for(int i=0;i<=n;i++)
      {
         gr[i].clear();
         d[0][i]=d[1][i]=1<<30;
         flag[0][i]=flag[1][i]=0;
      }
      while(r--)
      {
         int x,y,w;
         cin>>x>>y>>w;
         gr[x].pb({y,w});
         gr[y].pb({x,w});
      }
      dijkstra(1);
      cout<<"Case "<<cas++<<": "<<d[1][n]<<endl;
   }
   return 0;
}
```

SECOND SHORTEST PATH - 2

```
vector<pii>gr[5002];
int d1[5002],d2[5002];
void dijkstra(int s)
{
   d1[s]=0;
   priority_queue<pii,vector<pii>,greater<pii> >q;
   q.push(pii(0,s));
   while(!q.empty())
   {
      int u=q.top().sc;
      q.pop();
      for(int i=0;i<gr[u].size();i++)
      {
         int v=gr[u][i].fs;
         int d=gr[u][i].sc;
         if(d1[u]+d<d1[v])
         {
            int temp=d1[v];
            d1[v]=d1[u]+d;
            d2[v]=min(d2[v],min(temp,min(d2[u]+d,d1[u]+3*d)));
            q.push(pii(d1[v]+d2[v],v));
         }
         else if(d1[u]+d>d1[v] && d1[u]+d<d2[v]) d2[v]=d1[u]+d;
         else d2[v]=min(d2[v],min(d1[u]+3*d,d2[u]+d));
      }
   }
}
int main()
{
   int t,cas=1;
   cin>>t;
   while(t--)
   {
      int n,m;
      cin>>n>>m;
      for(int i=0;i<=n;i++)
      {
         gr[i].clear();
         d1[i]=d2[i]=30000000; ///not giving 1<<30 cuz we'll be
adding stuffs to it
      }
      while(m--)
      {
         int x,y,w;
         cin>>x>>y>>w;
         gr[x].pb(pii(y,w));
         gr[y].pb(pii(x,w));
      }
      dijkstra(1);
      cout<<"Case "<<cas++<<": "<<min(d2[n],d2[1]+d1[n])<<endl;
      ///d2[1]+d1[n] case is checked again because when working
with
      ///node 1, its d2[1] is not yet found and remains inf
   }
   return 0;
}
```

DIGIT DP PALINDROMES

```
ll dp[50][50];

ll rec(int len,int pos,int tight,int ok,string s)
{
   if(pos>=ceil(len/2.0))
   {
      if(tight==0 || (tight && ok)) return 1ll;
      else return 0ll;
   }
   if(!tight && dp[len][pos]!=-1) return dp[len][pos];
```

```
    ll ret=0;
    int lim=tight?(s[pos]-'0'):9;
    for(int i=0;i<=lim;i++)
    {
        if(i==0 && pos==0) continue;
        //if(i!=0 && i!=8 && i!=1) continue;
        int newtight=i<lim?0:tight;
        int newok=ok?(s[len-pos-1]-'0')>=i:(s[len-pos-1]-'0')>i;
        ret+=rec(len,pos+1,newtight,newok,s);
    }
    if(tight==0) dp[len][pos]=ret;
    return ret;
}

ll pal(string s)
{
    int l=s.length();
    ll ans=0;
    for(int i=1;i<=l;i++)
    {
        ans+=rec(i,0,i==l,1,s);
    }
    return ans;
}

int ispali(string s)
{
    int l=s.length();
    int m=l/2;
    for(int i=0;i<=m;i++)
    {
        if(s[i]!=s[l-i-1] || (s[i]!='1' && s[i]!='0' && s[i]!='8')) return 0ll;
    }
    return 1ll;
}

int main()
{
    memset(dp,-1,sizeof dp);

    int t;
    cin>>t;
    while(t--)
    {
        string a,b;
        cin>>a>>b;
        //cout<<"        "<<pal(b)<<" "<<pal(a)<<" "<<ispali(a)<<endl;
        cout<<pal(b)-pal(a)+ispali(a)<<endl;
    }
    return 0;
}
```

DIGIT DP SUM OF DIGITS

```
// Given two integers a and b. The task is to print
// sum of all the digits appearing in the
// integers between a and b
#include "bits/stdc++.h"
using namespace std;

// Memoization for the state results
long long dp[20][180][2];

// Stores the digits in x in a vector digit
long long getDigits(long long x, vector <int> &digit)
{
    while (x)
    {
            digit.push_back(x%10);
            x /= 10;
    }
}

// Return sum of digits from 1 to integer in
// digit vector
long long digitSum(int idx, int sum, int tight,
                                                vector <int> &digit)
{
    // base case
    if (idx == -1)
    return sum;

    // checking if already calculated this state
    if (dp[idx][sum][tight] != -1 and tight != 1)
            return dp[idx][sum][tight];
            //else if(tight==1) cout<<"idx="<<idx<<"
sum="<<sum<<"\n";

    long long ret = 0;

    // calculating range value
    int k = (tight)? digit[idx] : 9;

    for (int i = 0; i <= k ; i++)
    {
            // caclulating newTight value for next state
            int newTight = (digit[idx] == i)? tight : 0;

            // fetching answer from next state
            ret += digitSum(idx-1, sum+i, newTight, digit);
```

precode@tanny411
aysha.kamal7@gmail.com


```
    }

    if (!tight)
    dp[idx][sum][tight] = ret;

    return ret;
}

// Returns sum of digits in numbers from a to b.
int rangeDigitSum(int a, int b)
{
    // initializing dp with -1
    memset(dp, -1, sizeof(dp));

    // storing digits of a-1 in digit vector
    vector<int> digitA;
    getDigits(a-1, digitA);

    // Finding sum of digits from 1 to "a-1" which is passed
    // as digitA.
    long long ans1 = digitSum(digitA.size()-1, 0, 1, digitA);
    cout<<"shesh"<<endl;
    // Storing digits of b in digit vector
    vector<int> digitB;
    getDigits(b, digitB);

    // Finding sum of digits from 1 to "b" which is passed
    // as digitB.
    memset(dp, -1, sizeof(dp));
    long long ans2 = digitSum(digitB.size()-1, 0, 1, digitB);

    cout<<ans1<<endl;
    cout<<ans2<<endl;
    return (ans2 - ans1);
}

// driver function to call above function
int main()
{
    long long a = 123, b = 1024;
    cout << "digit sum for given range : "
                << rangeDigitSum(a, b) << endl;
    return 0;
}
```

DIGIT DP another way

///https://stackoverflow.com/questions/22394257/how-to-count-integers-between-large-a-and-b-with-a-certain-property/22394258#22394258

Indeed, there is an approach to this pattern which turns out to work quite often. It can also be used to enumerate all the X with the given property, provided that their number is reasonably small. You can even use it to aggregate some associative operator over all the X with the given property, for example to find their sum.

To understand the general idea, let us try to formulate the condition X ≤ Y in terms of the decimal representations of X and Y.

Say we have X = x1 x2 ... xn - 1 xn and Y = y1 y2 ... yn - 1 yn, where xi and yi are the decimal digits of X and Y. If the numbers have a different length, we can always add zero digits to the front of the shorter one.

Let us define leftmost_lo as the smallest i with xi < yi. We define leftmost_lo as n + 1 if there is no such i. Analogously, we define leftmost_hi as the smallest i with xi > yi, or n + 1 otherwise.

Now X ≤ Y is true if and exactly if leftmost_lo <= leftmost_hi. With that observation it becomes possible to apply a dynamic programming approach to the problem, that "sets" the digits of X one after another. I will demonstrate this with your example problems:

Compute the number f(Y) of integers X with the property X ≤ Y and X has the digit sum 60
Let n be the number of Y's digits and y[i] be the i-th decimal digit of Y according to the definition above. The following recursive algorithm solves the problem:

```
count(i, sum_so_far, leftmost_lo, leftmost_hi):
  if i == n + 1:
    # base case of the recursion, we have recursed beyond the last digit
    # now we check whether the number X we built is a valid solution
    if sum_so_far == 60 and leftmost_lo <= leftmost_hi:
      return 1
    else:
      return 0
  result = 0
  # we need to decide which digit to use for x[i]
  for d := 0 to 9
    leftmost_lo' = leftmost_lo
    leftmost_hi' = leftmost_hi
    if d < y[i] and i < leftmost_lo': leftmost_lo' = i
    if d > y[i] and i < leftmost_hi': leftmost_hi' = i
    result += count(i + 1, sum_so_far + d, leftmost_lo', leftmost_hi')
```

```
    return result
```

///Now we have f(Y) = count(1, 0, n + 1, n + 1) and we have solved the problem. We can add memoization to the function to make it fast. The runtime is O(n4) for this particular implementation. In fact we can cleverly optimize the idea to make it O(n). This is left as an exercise to the reader (Hint: You can compress the information stored in leftmost_lo and leftmost_hi into a single bit and you can prune if sum_so_far > 60). The solution can be found at the end of this post.

Compute the number f(Y) of integers X with the property X ≤ Y and X is palindromic
This one is slightly tougher. We need to be careful with leading zeroes: The mirror point of a palindromic number depends on how many leading zeroes we have, so we would need to keep track of the number of leading zeroes.

There is a trick to simplify it a bit though: If we can count the f(Y) with the additional restriction that all numbers X must have the same digit count as Y, then we can solve the original problem as well, by iterating over all possible digit counts and adding up the results.

So we can just assume that we don't have leading zeroes at all:

```
count(i, leftmost_lo, leftmost_hi):
   if i == ceil(n/2) + 1: # we stop after we have placed one half of the number
      if leftmost_lo <= leftmost_hi:
         return 1
      else:
         return 0
   result = 0
   start = (i == 1) ? 1 : 0    # no leading zero, remember?
   for d := start to 9
      leftmost_lo' = leftmost_lo
      leftmost_hi' = leftmost_hi
      # digit n - i + 1 is the mirrored place of index i, so we place both at
      # the same time here
      if d < y[i]     and i     < leftmost_lo': leftmost_lo' = i
      if d < y[n-i+1] and n-i+1 < leftmost_lo': leftmost_lo' = n-i+1
      if d > y[i]     and i     < leftmost_hi': leftmost_hi' = i
      if d > y[n-i+1] and n-i+1 < leftmost_hi': leftmost_hi' = n-i+1
      result += count(i + 1, leftmost_lo', leftmost_hi')
   return result
```

The result will again be f(Y) = count(1, n + 1, n + 1).

UPDATE: If we don't only want to count the numbers, but maybe enumerate them or compute some aggregate function from them which does not expose group structure, we need to enforce the lower bound on X as well during the recursion. This adds a few more parameters.

UPDATE 2: O(n) Solution for the "digit sum 60" example:

In this application we place the digits from left to right. Since we are only interested in whether leftmost_lo < leftmost_hi holds true, let us add a new parameter lo. lo is true iif leftmost_lo < i and false otherwise. If lo is true, we can use any digit for the position i. If it is false, we can only use the digits 0 to Y[i], since any larger digit would cause leftmost_hi = i < leftmost_lo and can thus not lead to a solution. Code:

```
def f(i, sum_so_far, lo):
   if i == n + 1: return sum_so_far == 60
   if sum_so_far > 60: return 0
   res = 0
   for d := 0 to (lo ? 9 : y[i]):
      res += f(i + 1, sum + d, lo || d < y[i])
   return res
```

Arguably, this way of looking at it is somewhat simpler, but also a bit less explicit than the leftmost_lo/leftmost_hi approach. It also doesn't work immediately for somewhat more complicated scenarios like the palindrome problem (although it can be used there as well).

NUMBER OF PALINDROMES FROM a TO b (another way digit dp)

```
ll dp[20][20][20];
vector<int>v;
int siz,hsiz;

ll pal(int ind,int leftmost_lo,int leftmost_hi)
{
   if(ind<=hsiz)
   {
      if(leftmost_lo>=leftmost_hi) return 1;
      else return 0;
   }
   if(dp[ind][leftmost_lo][leftmost_hi]!=-1) return dp[ind][leftmost_lo][leftmost_hi];
   ll ret=0;
   int start;
   if(ind==siz) start=1;
   else start=0;
```

```
    int n=siz-ind+1;
    for(int i=start; i<=9; i++)
    {
        int newleftmost_lo=leftmost_lo;
        int newleftmost_hi=leftmost_hi;
        if(i<v[ind] && ind>newleftmost_lo) newleftmost_lo=ind;
        if(i<v[n] && n>newleftmost_lo) newleftmost_lo=n;
        if(i>v[ind] && ind>newleftmost_hi) newleftmost_hi=ind;
        if(i>v[n] && n>newleftmost_hi) newleftmost_hi=n;
        ret+=pal(ind-1,newleftmost_lo,newleftmost_hi);
    }
    return dp[ind][leftmost_lo][leftmost_hi]=ret;
}
ll ara[20];
int main()
{
    ara[0]=1;
    ll add=9;
    for(int i=1; i<=17; i+=2)
    {
        ara[i]=ara[i-1]+add;
        ara[i+1]=ara[i]+add;
        add*=10;
    }
    int t,cas=1;
    cin>>t;
    while(t--)
    {
        ll a,b;
        cin>>a>>b;
        v.clear();
        if(a>b) swap(a,b);
        a--;
        ll ans1=0;
        if(a>=0)
        {
            v.pb(0);
            if(a==0) v.pb(0);
            else
            {
                while(a)
                {
                    v.pb(a%10);
                    a/=10;
                }
            }
            memset(dp,-1,sizeof dp);
            siz=v.size()-1;
            hsiz=siz/2;
            ans1=pal(siz,0,0);
            v.clear();
            ans1+=ara[siz-1];
        }
        v.pb(0);
        if(b==0) v.pb(0);
        else
        {
            while(b)
            {
                v.pb(b%10);
                b/=10;
            }
        }
        siz=v.size()-1;
        hsiz=siz/2;
        memset(dp,-1,sizeof dp);
        ll ans2=pal(siz,0,0);
        ans2+=ara[siz-1];
        cout<<"Case "<<cas++<<": "<<ans2-ans1<<"\n";
    }
    return 0;
}
```

PRIMS MST

```
template <class T>

struct Prims
{
    int n;
    vector< vector< pair<int,T> > > adj; ///the weight can be int/double so we use template
    ///when calling prims we put the type in <T>
    vector<T>dist;
    vector<int>parent;
    Prims (int n): n(n),adj(n){}
    void addedge(int x,int y,T w)
    {
        adj[x].pb( {y,w} );
        adj[y].pb( {x,w} );
    }
    int primsmst(int src)
    {
        priority_queue<pair<T,int>,vector<pair<T,int>> , greater<pair<T,int>> >q;
        dist = vector<T>(n,numeric_limits<T>::max()); ///T type er max value is in numeric limits max()
        parent = vector<int>(n,-1);
        vector<int>flag(n);
        dist[src]=0;
```

```
    parent[src]=src;
    q.push({0,src});
    T mst=0;
    int cnt=0;
    while(!q.empty())
    {
      int u=q.top().sc;
      if(flag[u]) {q.pop(); continue;}
      flag[u]=1;
      mst+=q.top().fs;
      cnt++;
      if(cnt==n-1) break;
      q.pop();
      for(int i=0;i<adj[u].size();i++)
      {
        int v=adj[u][i].fs;
        if(flag[v]) continue;
        T vd=adj[u][i].sc;
        if(vd<dist[v])
        {
          dist[v]=vd;
          q.push({dist[v],v});
          parent[v]=u;
        }
      }
    }
    return mst;
  }
};
```

N VARIABLE LINEAR DIOPHANTINE EQUATION POS'VE SOLN

```
// A Dynamic programming based C++ program to find number of
// non-negative solutions for a given linear equation
#include<bits/stdc++.h>
using namespace std;
// Returns counr of solutions for given rhs and coefficients
// coeff[0..n-1]
int countSol(int coeff[], int n, int rhs)
{
  // Create and initialize a table to store results of
  // subproblems
  int dp[rhs+1];
  memset(dp, 0, sizeof(dp));
  dp[0] = 1;

  // Fill table in bottom up manner
  for (int i=0; i<n; i++)
    for (int j=coeff[i]; j<=rhs; j++)
      dp[j] += dp[j-coeff[i]];

  for(int i=0;i<=rhs;i++)
  {
    cout<<" "<<dp[i];
  }
  cout<<endl;
  return dp[rhs];
}
// Driver program
int main()
{
  int coeff[]  = {2, 2, 5};
  int rhs = 4;
  int n = sizeof(coeff)/sizeof(coeff[0]);
  cout << countSol(coeff, n, rhs);
  return 0;
}
```

PSEUDO RANDOM GENERATOR

```
int main () {
  int i, n;
  time_t t;

  n = 5;

  /* Intializes random number generator */
  srand((unsigned) time(&t));

  /* Print 5 random numbers from 0 to 49 */
  for( i = 0 ; i < n ; i++ ) {
    printf("%d\n", rand() % 50);
  }

  return(0);
}
```

MAXIMUM NO. OF DIVISORS UPTO N

```
vector<int>prime;
void sieve(int n)
{
  prime.pb(2);
  vector<bool>flag(n+2);
  for(int i=3;i<=n;i+=2)
  {
    if(flag[i]==false) prime.pb(i);
    if(i<=sqrt(n) && flag[i]==false)
    {
```

```
            for(int j=i*i;j<=n;j+=(i+i)) flag[j]=true;
        }
    }
}
int main()
{
    ///maximum number of divisors from 1 to 100000
    int n=100000;
    sieve(n);
    int mx=0;
    for(int i=1;i<=n;i++)
    {
        int x=i;
        int ans=1;
        for(int j=0;prime[j]<=sqrt(x);j++)
        {
            if(x%prime[j]==0)
            {
                int cnt=1;
                while(x%prime[j]==0) cnt++,x/=prime[j];
                ans*=cnt;
            }
        }
        if(x>1) ans*=2;
        mx=max(mx,ans);
    }
    cout<<mx<<endl;
    return 0;
}
```

MINIMUM VERTEX COVER

```
/*
এটি একটি NP-hard প্রবলেম,অর্থাৎ এই প্রবলেমের কোনো পলিনমিয়াল টাইম সলিউশন নেই।
তবে গ্রাফটি যদি Tree হয় অর্থাত n-1 টা edge থাকে আর কোনো সাইকেল না থাকে তাহলে
ডাইনামিক প্রোগ্রামিং বা ম্যাক্স ফ্লো/বাইপারটাইট ম্যাচিং এর সাহায্যে প্রবলেমটি সলভ করা সম্ভব।
*/
vector<int>g[100005];
int dp[100005][2];
int minvert(int p,int u,int taken)
{
    if(dp[u][taken]!=-1) return dp[u][taken];
    int mx=0;
    for(int i=0;i<g[u].size();i++)
    {
        int v=g[u][i];
        if(v==p) continue;
        if(taken==0)
        {
            mx+=minvert(u,v,1);
        }
        else
        {
            mx+=min(minvert(u,v,1),minvert(u,v,0));
        }
    }
    return dp[u][taken]=mx+taken;
}
int main()
{
    int n,x,y;
    cin>>n;
    memset(dp,-1,sizeof(dp));
    for(int i=0;i<n-1;i++)
    {
        cin>>x>>y;
        g[x].pb(y);
        g[y].pb(x);
    }
    cout<<min(minvert(-1,1,0),minvert(-1,1,1))<<endl;
    return 0;
}
```

MOS

```
// Program to compute sum of ranges for different range
// queries
///O( (m+n)*sqrt(n) )
//It cannot work for problems where we have update operations also mixed with sum queries.
#include <bits/stdc++.h>
using namespace std;

// Variable to represent block size. This is made global
// so compare() of sort can use it.
int block;

// Structure to represent a query range
struct Query
{
    int L, R;
};

// Function used to sort all queries so that all queries
// of same block are arranged together and within a block,
// queries are sorted in increasing order of R values.
bool compare(Query x, Query y)
```

```
{
   // Different blocks, sort by block.
   if (x.L/block != y.L/block)
      return x.L/block < y.L/block;

   // Same block, sort by R value
   return x.R < y.R;
}

// Prints sum of all query ranges. m is number of queries
// n is size of array a[].
void queryResults(int a[], int n, Query q[], int m)
{
   // Find block size
   block = (int)sqrt(n);

   // Sort all queries so that queries of same blocks
   // are arranged together.
   sort(q, q + m, compare);

   // Initialize current L, current R and current sum
   int currL = 0, currR = 0;
   int currSum = 0;

   // Traverse through all queries
   for (int i=0; i<m; i++)
   {
      // L and R values of current range
      int L = q[i].L, R = q[i].R;

      // Remove extra elements of previous range. For
      // example if previous range is [0, 3] and current
      // range is [2, 5], then a[0] and a[1] are subtracted
      while (currL < L)
      {
         currSum -= a[currL];
         currL++;
      }

      // Add Elements of current Range
      while (currL > L)
      {
         currSum += a[currL-1];
         currL--;
      }
      while (currR <= R)
      {
         currSum += a[currR];
         currR++;
      }

      // Remove elements of previous range.  For example
      // when previous range is [0, 10] and current range
      // is [3, 8], then a[9] and a[10] are subtracted
      while (currR > R+1)
      {
         currSum -= a[currR-1];
         currR--;
      }

      // Print sum of current range
      cout << "Sum of [" << L << ", " << R
           << "] is "  << currSum << endl;
   }
}

// Driver program
int main()
{
   int a[] = {1, 1, 2, 1, 3, 4, 5, 2, 8};
   int n = sizeof(a)/sizeof(a[0]);
   Query q[] = {{0, 4}, {1, 3}, {2, 4}};
   int m = sizeof(q)/sizeof(q[0]);
   queryResults(a, n, q, m);
   return 0;
}
```

RMQ SPARSE TABLE

```
int n,lg;
vector<int>v(1000);
int table[1000][1000];///[n][lg];

void prn()
{
   for(int i=0;i<n;i++)
   {
      for(int j=0;j<=lg;j++)
      {
         cout<<table[i][j];
      }
      cout<<endl;
   }
}

void pre_process()
{
   ///O(nlogn)
   ///table stores index of minimum array element from i to i+2^j
-1
```

```
    for(int j=0; (1<<j)<=n ;j++ ) /// small to big ranges
    {
        for(int i=0; i+(1<<j)-1<n; i++)
        {
            if(j==0) table[i][j]=i;
            else
            {
                if(v[table[i][j-1]]<v[table[i+(1<<(j-1))][j-1]])
                {
                    table[i][j]=table[i][j-1];
                }
                else table[i][j]=table[i+(1<<(j-1))][j-1];
            }
        }
    }
    //prn();
}

int query(int l,int r)
{
    ///O(1)
    int Lg=log2(r-l+1);
    ///nearsest power of 2 to the range l to r;
    return min(v[table[l][Lg]],v[table[r-(1<<Lg)+1][Lg]]);
}

int main()
{
    cin>>n;
    lg=log2(n);
    for(int i=0;i<n;i++) cin>>v[i];
    pre_process();
    int q;
    cin>>q;
    while(q--)
    {
        int x,y;
        cin>>x>>y;
        cout<<query(x,y)<<endl;
    }
    return 0;
}
```

RMQ SQUARE ROOT DECOMPOSITION

```
#define MAXN 10000
#define SQRSIZE  100

int arr[MAXN];              // original array
int block[SQRSIZE];         // decomposed array
int blk_sz;                 // block size

// Time Complexity : O(1)
void update(int idx, int val)
{
    int blockNumber = idx / blk_sz;
    block[blockNumber] += val - arr[idx];
    arr[idx] = val;
}

// Time Complexity : O(sqrt(n))
int query(int l, int r)
{
    int sum = 0;
    while (l<r and l%blk_sz!=0 and l!=0)
    {
        // traversing first block in range
        sum += arr[l];
        l++;
    }
    while (l+blk_sz <= r)
    {
        // traversing completely overlapped blocks in range
        sum += block[l/blk_sz];
        l += blk_sz;
    }
    while (l<=r)
    {
        // traversing last block in range
        sum += arr[l];
        l++;
    }
    return sum;
    /**this way is little faster as has less number of division
operations
    int sum = 0;
    int len=blk_siz;
    int c_l = l / len,   c_r = r / len;
    if (c_l == c_r)
    for (int i=l; i<=r; ++i)
        sum += a[i];
    else {
    for (int i=l, end=(c_l+1)*len; i<end; ++i)
        sum += a[i];
    for (int i=c_l+1; i<=c_r-1; ++i)
        sum += b[i];
    for (int i=c_r*len; i<=r; ++i)
        sum += a[i];
    }
    */
```

```
}

// Fills values in input[]
void preprocess(int input[], int n)
{
   // initiating block pointer
   int blk_idx = -1;

   // calculating size of block
   blk_sz = sqrt(n);

   // building the decomposed array
   for (int i=0; i<n; i++)
   {
      arr[i] = input[i];
      if (i%blk_sz == 0)
      {
         // entering next block
         // incementing block pointer
         blk_idx++;
      }
      block[blk_idx] += arr[i];
   }
}

// Driver code
int main()
{
   // We have used separate array for input because
   // the purpose of this code is to explain SQRT
   // decomposition in competitive programming where
   // we have multiple inputs.
   int input[] = {1, 5, 2, 4, 6, 1, 3, 5, 7, 10};
   int n = sizeof(input)/sizeof(input[0]);

   preprocess(input, n);

   cout << "query(3,8) : " << query(3, 8) << endl;
   cout << "query(1,6) : " << query(1, 6) << endl;
   update(8, 0);
   cout << "query(8,8) : " << query(8, 8) << endl;
   /**
   --inorder to do range update in this, keep a separate list,
suppose c, just as block but initialized as 0;
   then when says add x to [i,j], to those blocks who completely fall
in it, just c[that_block]+=ans; and
   partial blocks in the beginning and end will have to be manually
updated;
   --when asks what is ara[i], put out ara[i]+c[i/blk_siz];
   **/
   return 0;
}
```

SUBSTRING SEARCH USING SUFFIX ARRAY

```
/**
The task is to find a string s inside some text t online - we know the
text t beforehand, but not the string s.
We can create the suffix array for the text t in O(|t|log|t|) time. Now
we can look for the substring s in the following way.
The occurrence of s must be a prefix of some suffix from t. Since
we sorted all the suffixes we can perform a binary search for s in p.
Comparing the current suffix and the substring s within the binary
search can be done in O(|s|) time,
therefore the complexity for finding the substring is O(|s|log|t|). Also
notice that if the substring occurs multiple times in t,
then all occurrences will be next to each other in p. Therefore the
number of occurrences can be found with a second binary search,
and all occurrences can be printed easily.
*/
#define LEN 1000 + 5
#define LG log2(LEN) + 2

int suffara[LEN],P[LEN][(int)LG];
pair< pair<int,int> , int > L[LEN]; ///prev,now and index

bool suff_cmp(pair< pair<int,int> , int >a,pair< pair<int,int> , int >b){
   if(a.fs.fs!=b.fs.fs) return a.fs.fs<b.fs.fs;
   else return a.fs.sc<b.fs.sc;
}

void makeSuffAra(string s){
   int n=s.length();
   for(int i=0;i<n;i++){
      P[i][0]=s[i]-'a';
   }

   int step=1;
   for(int j=1;j<n;step++,j<<=1){///power(step-1)<n
      for(int i=0;i<n;i++){
         L[i].sc=i;
         L[i].fs.fs=P[i][step-1];
         L[i].fs.sc=(i+j<n)?P[i+j][step-1]:-1;
      }
      sort(L,L+n,suff_cmp);
      P[L[0].sc][step]=0;
      for(int i=1;i<n;i++){
         P[L[i].sc][step]=(L[i].fs==L[i-1].fs)?P[L[i-1].sc][step]:i;
      }
```

```
    }
    step--;
    for(int i=0;i<n;i++){
        suffara[P[i][step]]=i;
    }
}

int main()
{
    int ns,np;
    string s;
    cin>>s;
    transform(s.begin(), s.end(), s.begin(), ::tolower);
    makeSuffAra(s);
    string p;
    cin>>p;
    ns=s.length();
    np=p.length();
    int ansini=-1,ansfin=-1;
    int lo=0,hi=ns-1;
    while(lo<hi){
        int m=(lo+hi)/2;
        int in=suffara[m];
        int f=-1;
        int i,j;
        for(i=in,j=0;i<ns&&j<np;i++,j++){
            if(s[i]<p[j]) {
                f=0;
                break;
            }
            else if(s[i]>p[j]){
                f=1;
                break;
            }
        }
        if(f==-1 && i==ns && j<np) f=0;
        if(f==-1) ansini=m;
        if(f==0)lo=m+1;
        else if(f==1||f==-1) hi=m;
    }
    if(ansini==-1){
        return 0;
    }
    lo=0;
    hi=ns-1;
    while(lo<hi){
        int m=ceil((lo+hi)/2.0);
        int in=suffara[m];
        int f=-1;
        int i,j;
        for(i=in,j=0;i<ns&&j<np;i++,j++){
            if(s[i]<p[j]) {
                f=0;
                break;
            }
            else if(s[i]>p[j]){
                f=1;
                break;
            }
        }
        if(f==-1 && i==ns && j<np) f=0;
        if(f==-1) ansfin=m;
        if(f==0 || f==-1)lo=m;
        else if(f==1) hi=m-1;
    }
    vector<int>v;
    for(int i=ansini;i<=ansfin;i++){
        v.pb(suffara[i]);
    }
    sort(v.begin(),v.end());
    ///sorted order index from where the pattern starts
    for(int i=0;i<v.size();i++){
        cout<<v[i]<<" ";
    }
    return 0;
}
```

LCP FROM SUFFIX ARRAY

```
#define LEN 10000
#define LG (int)(log2(LEN)) +2
int P[LEN][LG],suff[LEN];
pair<pii,int>L[LEN];

bool cmp(pair< pair<int,int> , int >a,pair< pair<int,int> , int >b){
    if(a.fs.fs!=b.fs.fs) return a.fs.fs<b.fs.fs;
    else return a.fs.sc<b.fs.sc;
}
void suffAra(string s,int n){
    for(int i=0;i<n;i++){
        P[i][0]=s[i]-'a';
    }
    int step=1;
    for(int j=1;j<n;j<<=1,step++){
        for(int i=0;i<n;i++){
            L[i].sc=i;
            L[i].fs.fs=P[i][step-1];
            L[i].fs.sc=(i+j<n)?P[i+j][step-1]:-1;
        }
        sort(L,L+n,cmp);
```

```
        P[L[0].sc][step]=0;
        for(int i=1;i<n;i++){
            P[L[i].sc][step]=(L[i].fs==L[i-1].fs)?P[L[i-1].sc][step]:i;
        }
    }
    step--;
    for(int i=0;i<n;i++){
        suff[P[i][step]]=i;
    }
}
int LCP(int a,int b,int n){///longest common prefix of a and b
index suffix of the string from suffix array, i.e index of suffix array
    ///n is the length of the original string
    a=suff[a];
    b=suff[b];
    int lg=ceil(log2(n))-1;
    int lcp=0;
    for(int i=lg;i>=0&&a<n&&b<n;i--){
        if(P[a][i]==P[b][i]){
            a+=(1<<i);
            b+=(1<<i);
            lcp+=(1<<i);
        }
    }
    return lcp;
}
int main()
{
    int q,t=1;
    cin>>q;
    while(q--){
        string s;
        cin>>s;
        int n=s.length();
        suffAra(s,n);
//        for(int i=0;i<n;i++){
//            cout<<suff[i]<<" ";
//        }
    }
    return 0;
}
```

KADANE(2 way)

```
///have to take care of all negative number separately
int _maxSubArraySum(int a[], int size)
{
  int max_so_far = 0, max_ending_here = 0;
  for (int i = 0; i < size; i++)
  {
    max_ending_here = max_ending_here + a[i];
    if (max_ending_here < 0)
      max_ending_here = 0;
    else if (max_so_far < max_ending_here)
      max_so_far = max_ending_here;
  }
  return max_so_far;
}
///also takes care of all negative numbers
int maxSubArraySum(int a[], int size)
{
  int max_so_far = a[0];
  int curr_max = a[0];

  for (int i = 1; i < size; i++)
  {
     curr_max = max(a[i], curr_max+a[i]);
     max_so_far = max(max_so_far, curr_max);
  }
  return max_so_far;
}
```

KRUSKAL MST

```
struct Disjoint_set
{
    int n;
    vector<int>parent,Rank;
    /*Disjoint_set(int _n)
    {
        n=_n;
        P=vector<int>(n+2);
        for(int i=0;i<=n;i++)
        {
            P[i]=i;
        }
    }*/
    ///constructor, two types
    Disjoint_set() {}
    Disjoint_set(int n):n(n+2),parent(n+2),Rank(n+2)
    {
        for(int i=0; i<=n; i++)
        {
            parent[i]=i;
        }
    }
    int findparent(int x)
    {
        if(parent[x]==x) return x;
```

```
        else return parent[x]=findparent(parent[x]);
    }
    int mergenodes(int x,int y)
    {
        ///if connected returns false
        int a=findparent(x);
        int b=findparent(y);
        if(a==b) return 0;
        else
        {
            if(Rank[a]>Rank[b]) parent[b]=a;
            else if(Rank[b]>Rank[a]) parent[a]=b;
            else
            {
                parent[a]=b;
                Rank[b]++;
            }
            return 1;
        }
    }
};

struct kruskal
{
    int n;
    kruskal(int n): n(n) {}
    vector< pair<int, pair<int,int> > > edges ;
    void addedge(int x,int y,int w)
    {
        edges.pb( pair<int, pair<int,int> > (w,pair<int,int>(x,y)) );
        //edges.pb ({w,{x,y}});
    }
    int Minst()
    {
        int cnt=0;
        Disjoint_set D(n);
        int mst=0;
        sort(edges.begin(),edges.end());
        for(int i=0; i<edges.size(); i++)
        {
            if(D.mergenodes(edges[i].sc.fs,edges[i].sc.sc)) {
mst+=edges[i].fs; cnt++;}
        }
        if(cnt!=n-1) return -1;
        return mst;
    }
    int Maxst()
    {
        Disjoint_set Dd(n);
        int mst=0,cnt=0;
        sort(edges.rbegin(),edges.rend());
        for(int i=0; i<edges.size(); i++)
        {
            if(Dd.mergenodes(edges[i].sc.fs,edges[i].sc.sc)) {
mst+=edges[i].fs; cnt++; }
        }
        if(cnt!=n-1) return -1;
        return mst;
    }
};

int main()
{
    int t,cas=1;
    cin>>t;
    while(t--)
    {
        int n,m;
        cin>>n>>m;
        kruskal K(n);
        while(m--)
        {
            int x,y,z;
            cin>>x>>y>>z;
            K.addedge(x,y,z);
        }
        int d=K.Minst();
        cout<<d<<endl;
    }
    return 0;
}
```

LCA (n,sqrt(h))

```
#define maxnode 10000
int n;//given no. of nodes;
vector<int>sec(maxnode+2),parent(maxnode+2,-1),lvl(maxnode+2,-1
),dist(maxnode+2);
vector<pii>gr[maxnode+2];
//sec=last node of previous section
//bfs level of each node
int nsec,start;//no. of sections


void lca_dfs(int node,int lv)
{
    if(lvl[node]<nsec) sec[node]=start;
    else if(lvl[node]%nsec==0) sec[node]=parent[node];
    else sec[node]=sec[parent[node]];
```

```
    for(int i=0; i<gr[node].size(); i++)
    {
        int v=gr[node][i].fs;
        if(lvl[v]!=-1) continue;
        lvl[v]=lv+1;
        parent[v]=node;
        dist[v]=dist[node]+gr[node][i].sc;
        lca_dfs(v,lvl[v]);
    }
}

int query(int x,int y)
{
    //take them up until their ancestor is common in th eupper section
    while(sec[x]!=sec[y])
    {
        if(lvl[x]>lvl[y]) x=sec[x];
        else y=sec[y];
    }
    //for the final section go manually
    while(x!=y)
    {
        if(lvl[x]>lvl[y]) x=parent[x];
        else y=parent[y];
    }
    //max complexity O(2*sqrt(h));
    //sqrt(h) sections should be traversed
    //and final section traversed manually is of length sqrt(h)
    return x;
}
void prn()
{
    for(int i=1; i<=n; i++) cout<<" "<<lvl[i];

    cout<<endl;
    for(int i=1; i<=n; i++) cout<<" "<<dist[i];
}

int main()
{
    cin>>n;
    nsec=sqrt(n);
    int x,y,w;
    int c=n-1;///its a tree
    while(c--)
    {
        cin>>x>>y>>w;
        gr[x].pb(pii(y,w));
        gr[y].pb(pii(x,w));
    }

    start=1;
    lvl[start]=dist[start]=0;
    parent[start]=start;
    lca_dfs(start,0);
    //prn();

    cin>>x>>y;
    cout<<query(x,y)<<endl;

    //distance from x to y
    cout<<dist[x]+dist[y]-2*dist[p]<<endl;
    return 0;
}
```

LCA TO RMQ

```
#define maxnode 10000+5
int n,N;
vector<int>lvl,euler,firstocc(maxnode,-1),gr[maxnode];
///lvl: lvl[i] is the lvl of the node in euler[i]
///euler: euler tour er nodes serially
///index of each nodes first occurences in euler;


void prn()
{
    cout<<"euler :";
    for(int i=0;i<euler.size();i++) cout<<" "<<euler[i];
    cout<<endl<<"firstocc :";
    for(int i=0;i<n;i++) cout<<" "<<firstocc[i];
    cout<<endl<<"lvl :";
    for(int i=0;i<lvl.size();i++) cout<<" "<<lvl[i];
    cout<<endl;
}

void dfs(int u,int lv)
{
    euler.pb(u);
    lvl.pb(lv);
    for(int i=0;i<gr[u].size();i++)
    {
        int v=gr[u][i];
        if(firstocc[v]==-1)
        {
            firstocc[v]=euler.size();
            dfs(v,lv+1);
            euler.pb(u);
            lvl.pb(lv);
```

```
      }
   }
}

///RMQ
/*
->E[1, 2*N-1] – the nodes visited in an Euler Tour of T;
E[i] is the label of i-th visited node in the tour
->L[1, 2*N-1] – the levels of the nodes visited in the Euler Tour;
L[i] is the level of node E[i]
->H[1, N] – H[i] is the index of the first occurrence of node i in E
(any occurrence would be good, so it's not bad if we consider the
first one)
-->Assume that H[u] < H[v] (otherwise you must swap u and v).
It's easy to see that the nodes between the first occurrence of u
and the first
occurrence of v are E[H[u]...H[v]]. Now, we must find the node
situated on the smallest level.
For this, we can use RMQ. So, LCAT(u, v) = E[RMQL(H[u], H[v])]
(remember that RMQ returns the index).
*/
int st[2*maxnode+2][(int)log2(2*maxnode)+2];
void prep()
{
   for(int j=0;(1<<j)<=N;j++)
   {
      for(int i=0;i+(1<<j)-1<N;i++)
      {
         if(j==0) st[i][j]=i;
         else if(lvl[st[i][j-1]]<lvl[st[i+(1<<(j-1))][j-1]]) st[i][j]=st[i][j-1];
         else st[i][j]=st[i+(1<<(j-1))][j-1];
      }
   }
}

int RMQ(int x,int y)
{
   int ans;
   int lg=log2(y-x+1);
   if(lvl[st[x][lg]]<lvl[st[y-(1<<lg)+1][lg]]) ans=st[x][lg];
   else ans=st[y-(1<<lg)+1][lg];
   return ans;
}

int main()
{
   cin>>n;
   while(1)
   {
      int x,y;
      cin>>x>>y;
      if(x==-1 && y==-1) break;
      gr[x].pb(y);
      gr[y].pb(x);
   }
   int start=1;
   firstocc[start]=0;
   dfs(start,0);
   N=euler.size();
   prep();
   int q;
   cin>>q;
   while(q--)
   {
      int x,y;
      cin>>x>>y;
      if(firstocc[x]>firstocc[y]) swap(x,y);
      int Q=RMQ(firstocc[x],firstocc[y]);
      cout<<euler[Q]<<endl;
   }
   //prn();
   return 0;
}
```

MAXFLOW

```
///Edmond-Karp
///loj 1177

vector<int>gr[1000],parent(1000);
ll cap[1000][1000];
int n;
void clr(int z)
{
   z+=2;
   for(int i=0; i<=2*z; i++) gr[i].clear();
   for(int i=0; i<=2*z; i++)
   {
      for(int j=0; j<=2*z; j++)
      {
         cap[i][j]=0;
      }
   }
}

ll bfs(int s,int d)
{
   vector<ll>capacity(2*n+3);
   queue<int>q;
   q.push(s);
```

```
    capacity[s]=1<<30;
    parent[s]=-2;///visited
    while(!q.empty())
    {
        int u=q.front();
        q.pop();
        for(int i=0;i<gr[u].size();i++)
        {
            int v=gr[u][i];
            if(cap[u][v]>0 && parent[v]==-1)
            {
                capacity[v]=min(capacity[u],cap[u][v]);
                parent[v]=u;
                if(v==d) return capacity[v];
                q.push(v);
            }
        }
    }
    return 0;
}

ll maxflow(int s,int d)
{
    ll flow=0;
    while(1)
    {
        for(int i=0;i<=2*n+3;i++) parent[i]=-1;
        int fl=bfs(s,d);
        if(fl==0) return flow;
        flow+=(ll)fl;
        int v=d;
        while(parent[v]!=-2)
        {
            int u=parent[v];
            cap[u][v]-=fl;
            cap[v][u]+=fl;
            v=u;
        }
    }
}

int main()
{
    ///nodes and edges both have weights
    int t,cas=1;
    cin>>t;
    while(t--)
    {
        int m;
        cin>>n>>m;
        clr(n);
        ///1 is source, n is destination
        ///the nodes in between have weights
        for(int i=2; i<n; i++)
        {
            int w;
            cin>>w;
            cap[i][i+n]=w;
            gr[i].pb(i+n);
            gr[i+n].pb(i);
        }

        cap[1][1+n]=1<<30;
        gr[1].pb(1+n);
        gr[1+n].pb(1);

        cap[n][n+n]=1<<30;
        gr[n].pb(n+n);
        gr[n+n].pb(n);

        for(int i=0;i<m;i++)
        {
            int x,y,w;
            cin>>x>>y>>w;

            cap[x+n][y]=w;
            cap[y+n][x]=w;

            gr[x+n].pb(y);
            gr[y].pb(x+n);
            gr[y+n].pb(x);
            gr[x].pb(y+n);
        }
        cout<<"Case "<<cas++<<": "<<maxflow(1,n+n)<<endl;
        /*for(int i=1;i<=2*n;i++)
        {
            cout<<i<<" :"<<endl;
            for(int j=0;j<gr[i].size();j++)
            {
                cout<<" "<<gr[i][j];
            }
            cout<<endl;
        }*/
    }
    return 0;
}
```

precode@tanny411
aysha.kamal7@gmail.com

LARGEST SUBSET FORMING CONVEX HULL

```
int dp[105][105][105];
pdd ara[105];
int n;
bool left(pdd a,pdd b)
{
   if(a.fs==b.fs) return a.sc<b.sc;
   else return a.fs<b.fs;
}
bool cw(pdd o,pdd a,pdd b)
{
   return (a.fs-o.fs)*(b.sc-o.sc)>(b.fs-o.fs)*(a.sc-o.sc);
}
int rec(int from,int s,int f)
{
   //cout<<from<<" "<<s<<" "<<f<<endl;
   if(dp[from][s][f]!=-1) return dp[from][s][f];
   int mx=0;
   for(int i=0;i<n;i++){
      if(left(ara[from],ara[i]) && cw(ara[from],ara[f],ara[i]) &&
cw(ara[s],ara[f],ara[i])) mx=max(mx,rec(from,f,i)+1);
   }
   return dp[from][s][f]=mx;
}
int main()
{
   memset(dp,-1,sizeof dp);
   cin>>n;
   for(int i=0;i<n;i++)
   {
      cin>>ara[i].fs>>ara[i].sc;
   }
   int mx=0;
   for(int i=0;i<n;i++)
   {
      for(int j=0;j<n;j++)
      {
         //cout<<i<<" "<<j<<" "<<left(ara[i],ara[j])<<endl;
         if(left(ara[i],ara[j])) mx=max(mx,rec(i,i,j));
      }
   }
   cout<<mx+2<<endl;
   return 0;
}
```

DISTINCT SUBSTR IN a WITHOUT HAVING b AS SUBSTR

```
/*
Loj 1428
Algorithm: KMP, Suffix Array
A nice tutorial on KMP:
http://www.iti.fh-flensburg.de/lang/algorithmen/pattern/kmpen.htm
Tutorial on Suffix Array:
http://www.stanford.edu/class/cs97si/suffix-array.pdf
First you need to know how to find the number of distinct substrings of a string using suffix
array. The idea has been described on the solution of problem 4 in the tutorial link I've given.
In code, you can do this in:
sum = 0;
for(i = 0; i < len; i++)
{
   k = (len - B[i]) - LCP[i];
   if(k > 0) sum += k;
}
(Skip the next section if you already know how to find the number of distinct substrings)
-----------------------------------------------------------------
B[i] deontes the starting index of the i-th suffix in sorted order. For example, for the string
abaab:
B[0] = 2 aab
B[1] = 3 ab
B[2] = 0 abaab
B[3] = 4 b
B[4] = 1 baab
so len - B[i] = length of the i-th suffix in sorted order
len - B[0] = 3 aab
len - B[1] = 2 ab
len - B[2] = 5 abaab
len - B[3] = 1 b
len - B[4] = 4 baab
LCP[i] deontes the longest common prefix of the i-th and (i-1)th suffix in sorted order.
LCP[0] = 0aab                distinct substrings: aab aa a, no common prefix
                        null     k = (len - B[i]) - LCP[i] = 3 - 0 = 3

LCP[1] = 1  ab               distinct substrings: ab, common prefix: a; so a not counted
```

precode@tanny411
aysha.kamal7@gmail.com
24

aab k = (len - B[i]) - LCP[i] = 2 - 1 = 1

LCP[2] = 2 abaab distinct substrings: abaab abaa aba, common prefix: ab; so ab, a not counted
ab k = (len - B[i]) - LCP[i] = 5 - 2 = 3

LCP[3] = 0 b distinct substrings: b, no common prefix
abaab k = (len - B[i]) - LCP[i] = 1 - 0 = 1

LCP[4] = 1 baab distinct substrings: baab baa ba, common prefix: b; so b not counted
b k = (len - B[i]) - LCP[i] = 4 - 1 = 3

So total distinct substrings = 3+1+3+1+3 = 11.
Our problem is harder, we have to find the distinct substrings of string A and remove those
substrings which contains B as a substring.

The first test case:
ababa
ba
Using KMP, we can find out in which positions in A, B is found as a substring.
ababa
-ba-- position 1
---ba position 3
q[i] denotes the nearest position on the right of A[i] where B is found as a substring
A[i] ababa
q[i] 11335
Now, look, whe we are considering the suffix ababa, previously we are considering 5 substrings
ababa, abab, aba, ab, a
and deleting the duplicates from these substrings using LCP. But now we know ababa, abab, aba
can't be the part of the ans, as they contains ba. So we have to delete them too. q[B[i]]+len2-1
denotes the position where we have to stop deletion. All the substrings before this position
contains B as a substring, so they need to be deleted. So the code becomes:

```
for(i = 0; i < len1; i++)
{
   k = min(len1, q[B[i]]+len2-1) - B[i] - LCP[i];
   if(k > 0) sum += k;
}
*/
int row, column, step;
int B[SIZE];
char str[SIZE];
char substr[SIZE];
int q[SIZE];

#define MAXLG 20
struct entry
{
   int first, second, p;
} L[SIZE];
int P[MAXLG][SIZE];
int LCP[SIZE];
int b[SIZE];

int cmp(const struct entry &a, const struct entry &b)
{
   return a.first == b.first ? (a.second < b.second ? 1 : 0) : (a.first < b.first ? 1 : 0);
}

int comp2(const int& a, const int& b)
{
   return P[step-1][a] < P[step-1][b];
}

void kmpPreprocess(char *substr, int *b)
{
   const int m = strlen(substr);
   int i = 0, j = -1;

   b[i] = -1;
   while(i < m)
   {
      while(j >= 0 && substr[i] != substr[j])
         j = b[j];

            i++, j++;
      b[i] = j;
   }
}

void kmp(char *str, char *substr)
{
   int i=0, j=0, count = 0;
   const int n = strlen(str);
   const int m = strlen(substr);
```

```
    kmpPreprocess(substr, b);
    while (i<n)
    {
        while (j>=0 && str[i]!=substr[j])
            j=b[j];

        i++, j++;
        if (j==m)
            q[i-j] = i-j;
    }
}

void suffixArray(char *A)
{
    int i, count;
    int len = strlen(A);

    for (i = 0; i < len; i ++)
        P[0][i] = A[i] - 'a';

    for (step = 1, count = 1; count>>1 < len; step++, count <<= 1)
    {
                for (i = 0; i < len; i++)
                {
                        L[i].first = P[step - 1][i];
                        L[i].second = i + count < len ? P[step - 1][i +
count] : -1;
                        L[i].p = i;
                }

                sort(L, L + len, cmp);

        for (i = 0; i < len; i ++)
            P[step][L[i].p] = i > 0 && L[i].first == L[i - 1].first &&
L[i].second == L[i - 1].second ?
                                        P[step][L[i - 1].p] : i;
    }

    for(i = 0; i < len; i++) B[i] = i;
    sort(B, B + len, comp2);

    for(i = 1; i < len; i++)
    {
        int k;
        int x = B[i], y = B[i - 1];

        LCP[i] = 0;
        for (k = step - 1; k >= 0 && x < len && y < len; k--)
            if (P[k][x] == P[k][y])
                x += 1 << k, y += 1 << k, LCP[i] += 1 << k;
    }
    /*
    for(i = 0; i < len; i++)
                cout << LCP[i] << ' ';
    cout << endl;*/
}

int main()
{
    //freopen("input.txt", "r", stdin);
    //freopen("out.txt", "w", stdout);

    int i, j, k, l, sum = 0;
    int tc, t, d, x, y, a, b, r, n, current, first;
    char ch;
    int mx;
    bool flag;
    unsigned long long ln;

    scanf("%d", &tc);

    for(t = 1; t <= tc; t++)
    {
                sum = 0;
                scanf("%s %s", str, substr);

                int len1 = strlen(str);
                int len2 = strlen(substr);

                if(len1 >= len2)
                {
                        suffixArray(str);
                        memset(q, -1, sizeof q);
                        kmp(str, substr);

                        l = len1;
                        for(i = len1 - 1; i >= 0; i--)
                        {
                                if(q[i] == -1) q[i] = l;
                                else l = q[i];
                        }
                        /*
                        for(i = 0; 1<<(i-1) <= len; i++)
                        {
                                for(j = 0; j < len; j++)
                                        printf("%2d ", P[i][j]);
                                cout << endl;
                        }
                        cout << endl;*/
```

```
                for(i = 0; i < len1; i++)
                {
                        k = min(len1, q[B[i]]+len2-1) - B[i] -
LCP[i];
                        if(k > 0) sum += k;
                }
        }

        printf("Case %d: %d\n", t, sum);

    }

    return 0;
}
```

LCS

```
int a,b;
string aa,bb;
int dp[1007][1007];

int lcs(int i,int j)
{
    ///worst-case comlexity O(2^n)
    if(i==a || j==b) return 0;
    if(dp[i][j]!=-1) return dp[i][j];
    if(aa[i]==bb[j]) return dp[i][j]=1+lcs(i+1,j+1);
    else return dp[i][j]=max(lcs(i+1,j),lcs(i,j+1));
}

void print(int i,int j)
{
    if(i==a || j==b) return;
    if(aa[i]==bb[j])
    {
        cout<<aa[i]<<" "<<i+1<<" "<<j+1<<endl;
        print(i+1,j+1);
    }
    else if(dp[i+1][j]>dp[i][j+1])
    {
        print(i+1,j);
    }
    else
    {
        print(i,j+1);
    }
}
string s;
void printAll(int i,int j)
{
    if(i==a || j==b)
    {
        cout<<s<<endl;
        return;
    }
    if(aa[i]==bb[j])
    {
        s+=aa[i];
        printAll(i+1,j+1);
        s.pop_back();
    }
    else if(dp[i+1][j]==dp[i][j+1])
    {
        printAll(i+1,j);
        printAll(i,j+1);
    }
    else if(dp[i+1][j]>dp[i][j+1])
    {
        printAll(i+1,j);
    }
    else
    {
        printAll(i,j+1);
    }
}
int table[1007][1007];
int tabledp(int n,int m)
{
    ///worst-case complexity O(n*m)
    for(int i=0; i<=n; i++)
    {
        for(int j=0; j<=m; j++)
        {
            if(i==0 || j==0) table[i][j]=0;
            else if(aa[i-1]==bb[j-1]) table[i][j]=1+table[i-1][j-1];
            else table[i][j]=max(table[i-1][j],table[i][j-1]);
        }
    }
    return table[n][m];
}

void printtable()
{
    int index=table[a][b];
    char lcsprn[index+1 ];
    lcsprn[index]=0;
    int i=a,j=b;
    while(i>0 && j>0)
    {
        if(aa[i-1]==bb[j-1])
```

```
      {
         lcsprn[index-1]=aa[i-1];
         index--;
         i--;
         j--;
      }
      else if(table[i-1][j]>table[i][j-1])///to print all:
      {
         ///if(table[i-1][j]==table[i][j-1])
         ///then we have to go both ways; how? I donno :3
         i--;
      }
      else
      {
         j--;
      }
   }
   puts(lcsprn);
}

int space_optimised_table_lcs()
{
   int n=a,m=b;
   int  optable[2][m+1];
   int in;
   memset(optable,0,sizeof optable);
   for(int i=0; i<=n; i++)
   {
      in=i&1;///odd hoile 1
      for(int j=0; j<=m; j++)
      {
         if(i==0 || j==0) optable[in][j]=0;
         else if(aa[i-1]==bb[j-1])
         {
            optable[in][j]=1+optable[1-in][j-1];
         }
         else
         {
            optable[in][j]=max(optable[1-in][j],optable[in][j-1]);

         }
      }
   }
   return optable[in][m];
}

int main()
{
   int n;
   int cas=1;
   cin>>n;
   while(n--)
   {
      cin>>a>>aa>>b>>bb;///aa is of length a
      ///recursion
      memset(dp,-1,sizeof(dp));
      cout<<"LCS = "<<lcs(0,0)<<endl;
      //print(0,0);
      //printAll(0,0);
      ///table dp
      /*Note that table[i][j] contains length of LCS of
      aa[0..i-1] and bb[0..j-1] */
      cout<<"LCS = "<<tabledp(a,b)<<endl;
      printtable();
      cout<<"space optimised =
"<<space_optimised_table_lcs()<<endl;
   }
   return 0;
}
```

LCP OF 3 STRINGS

```
pair<pii,int>L[LEN];
int P[LEN][LG],suff[LEN];

bool cmp(pair<pii,int>a,pair<pii,int>b){
   if(a.fs.fs==b.fs.fs) return a.fs.sc<b.fs.sc;
   else return a.fs.fs<b.fs.fs;
}
int lcp(int a,int b,int step){
   int ans=0;
   for(int i=step;i>=0;i--){
      if(P[a][i]==P[b][i]){
         ans+=(1<<i);
         a+=(1<<i);
         b+=(1<<i);
      }
   }
   return ans;
}
int main()
{
   ios_base::sync_with_stdio(false);cin.tie(0);cout.tie(0);
   int t,test=1;
   cin>>t;
   while(t--){
      string a,b,c;
      cin>>a>>b>>c;

      int n=a.length();
```

```
    int m=b.length();
    int p=c.length();
    a+='$'+b+'#'+c+'%';
    int len=a.length();

    //cout<<"aysha";
    for(int i=0;i<len;i++){
      P[i][0]=a[i]-'a';
    }
    int step=1;
    for(int j=1;j<len;j*=2,step++){
      for(int i=0;i<len;i++){
        L[i].sc=i;
        L[i].fs.fs=P[i][step-1];
        L[i].fs.sc=(i+j<len)?P[i+j][step-1]:-1000;
      }
      sort(L,L+len,cmp);
      for(int i=0;i<len;i++){
        P[L[i].sc][step]=(i>0 &&
L[i].fs==L[i-1].fs)?P[L[i-1].sc][step]:i;
      }
    }
    step--;
//      for(int i=0;i<len;i++){
//        suffara[P[i][step]]=i;
//      }
    //cout<<"kamal\n";
    int x=0,y=0,z=0;

    int ANS=0;
    //cout<<"tanny";
    for(int i=3,j=3;i<len;j++,i++){
      while(j<len && (x==0 || y==0 || z==0) ){
        //cout<<"aysha\n";
        int k=L[j].sc;
        if(k<n) x++;
        else if (k<(n+1+m)) y++;
        else z++;
        j++;
      }
      j--;
      if(x==0 || y==0 || z==0) break;
      //cout<<i<<" "<<j<<endl;
      ANS=max(ANS,lcp(L[i].sc,L[j].sc,step));
      int k=L[i].sc;
      if(k<n) x--;
      else if (k<(n+1+m)) y--;
      else z--;
    }
    //cout<<"aise";
    cout<<"Case "<<test++<<": "<<ANS<<endl;
  }
  return 0;
}
```

JOHNSONS ALGO

```
int n,m;
vector<pii>gr[1000];
int h[1000];
int bellman_ford(int s)
{
  for(int i=0; i<=n; i++) h[i]=0;
  int x=n-1;
  while(x--)
  {
    for(int i=0; i<=n; i++)
    {
      for(int j=0; j<gr[i].size(); j++)
      {
        int v=gr[i][j].fs;
        if(h[v]>h[i]+gr[i][j].sc) h[v]=h[i]+gr[i][j].sc;
      }
    }
  }
  for(int i=0; i<=n; i++)
  {
    for(int j=0; j<gr[i].size(); j++)
    {
      int v=gr[i][j].fs;
      if(h[v]>h[i]+gr[i][j].sc) return 0;
    }
  }
  return 1;
}
void reweight()
{
  for(int i=1; i<=n; i++)
  {
    for(int j=0; j<gr[i].size(); j++)
    {
      gr[i][j].sc+=h[i]-h[gr[i][j].fs];
    }
  }
}
int dist[1000][1000];
void dijkstra(int s)
{
  int vis[n+2];
  memset(vis,0,sizeof vis);
```

```
    for(int i=1; i<=n; i++) dist[s][i]=1<<30;
    dist[s][s]=0;
    priority_queue<pii,vector<pii>,greater<pii> >q;
    q.push(pii(0,s));
    while(!q.empty())
    {
        int u=q.top().sc;
        q.pop();
        if(vis[u]) continue;
        vis[u]=1;
        for(int i=0; i<gr[u].size(); i++)
        {
            int v=gr[u][i].fs;
            if(dist[s][v]>dist[s][u]+gr[u][i].sc)
            {
                dist[s][v]=dist[s][u]+gr[u][i].sc;
                q.push(pii(dist[s][v],v));
            }
        }
    }

}
int main()
{
    //freopen("in.txt","r",stdin);
    int t;
    cin>>t;
    int cas=1;
    while(t--)
    {
        if(cas>1) cout<<endl;
        cin>>n>>m;
        for(int i=0; i<=n; i++) gr[i].clear();
        while(m--)
        {
            int x,y,z;
            cin>>x>>y>>z;
            gr[x].pb(pii(y,z));
        }
        ///adding dummy node
        for(int i=1; i<=n; i++) gr[0].pb(pii(i,0));
        if(!bellman_ford(0)) cout<<"graph "<<cas++<<" no\n";
        else
        {
            cout<<"graph "<<cas++<<" yes\n\n";
            reweight();
            for(int i=1; i<=n; i++) dijkstra(i);
        for(int i=1;i<=n;i++)
        {
            cout<<h[i]<<" ";
        }
        cout<<h[0]<<endl<<endl;
        for(int i=1;i<=n;i++)
        {
            for(int j=1;j<=n;j++)
            {
                if(dist[i][j]!=1<<30) dist[i][j]+=h[j]-h[i];
            }
        }
        for(int i=1;i<=n;i++)
        {
            for(int j=1;j<=n;j++)
            {
                if(j>1) cout<<" ";
                if(dist[i][j]==1<<30) cout<<"#";
                else cout<<dist[i][j];
            }
            cout<<endl;
        }
        }

    }
    return 0;
}
```

LINKED LIST

```
#include<stdio.h>
#include<stdlib.h>
#include<stdbool.h>

struct test_struct
{
    int val;
    struct test_struct *next;
};

struct test_struct *head = NULL;
struct test_struct *curr = NULL;

struct test_struct* create_list(int val)
{
    printf("\n creating list with headnode as [%d]\n",val);
    struct test_struct *ptr = (struct test_struct*)malloc(sizeof(struct
test_struct));
    if(NULL == ptr)
    {
        printf("\n Node creation failed \n");
        return NULL;
    }
```

```
    ptr->val = val;
    ptr->next = NULL;

    head = curr = ptr;
    return ptr;
}

struct test_struct* add_to_list(int val, bool add_to_end)
{
    if(NULL == head)
    {
        return (create_list(val));
    }

    if(add_to_end)
        printf("\n Adding node to end of list with value [%d]\n",val);
    else
        printf("\n Adding node to beginning of list with value
[%d]\n",val);

    struct test_struct *ptr = (struct test_struct*)malloc(sizeof(struct
test_struct));
    if(NULL == ptr)
    {
        printf("\n Node creation failed \n");
        return NULL;
    }
    ptr->val = val;
    ptr->next = NULL;

    if(add_to_end)
    {
        curr->next = ptr;
        curr = ptr;
    }
    else
    {
        ptr->next = head;
        head = ptr;
    }
    return ptr;
}

struct test_struct* search_in_list(int val, struct test_struct
**prev)
{
    struct test_struct *ptr = head;
    struct test_struct *tmp = NULL;
    bool found = false;

    printf("\n Searching the list for value [%d] \n",val);

    while(ptr != NULL)
    {
        if(ptr->val == val)
        {
            found = true;
            break;
        }
        else
        {
            tmp = ptr;
            ptr = ptr->next;
        }
    }

    if(true == found)
    {
        if(prev)
            *prev = tmp;
        return ptr;
    }
    else
    {
        return NULL;
    }
}

int delete_from_list(int val)
{
    struct test_struct *prev = NULL;
    struct test_struct *del = NULL;

    printf("\n Deleting value [%d] from list\n",val);

    del = search_in_list(val,&prev);
    if(del == NULL)
    {
        return -1;
    }
    else
    {
        if(prev != NULL)
            prev->next = del->next;

        if(del == curr)
        {
            curr = prev;
        }
        else if(del == head)
```

```
        {
            head = del->next;
        }
    }

    free(del);
    del = NULL;

    return 0;
}

void print_list(void)
{
    struct test_struct *ptr = head;

    printf("\n -------Printing list Start------- \n");
    while(ptr != NULL)
    {
        printf("\n [%d] \n",ptr->val);
        ptr = ptr->next;
    }
    printf("\n -------Printing list End------- \n");

    return;
}

int main(void)
{
    int i = 0, ret = 0;
    struct test_struct *ptr = NULL;

    print_list();

    for(i = 5; i<10; i++)
        add_to_list(i,true);

    print_list();

    for(i = 4; i>0; i--)
        add_to_list(i,false);

    print_list();

    for(i = 1; i<10; i += 4)
    {
        ptr = search_in_list(i, NULL);
        if(NULL == ptr)
        {
            printf("\n Search [val = %d] failed, no such element
found\n",i);
        }
        else
        {
            printf("\n Search passed [val = %d]\n",ptr->val);
        }

        print_list();

        ret = delete_from_list(i);
        if(ret != 0)
        {
            printf("\n delete [val = %d] failed, no such element
found\n",i);
        }
        else
        {
            printf("\n delete [val = %d]  passed \n",i);
        }

        print_list();
    }

    return 0;
}
```

IS THERE A SUBSET DIVISIBLE BY m

If n > m there will always be a subset with sum divisible by m (which is easy to prove with pigeonhole principle). So we need to handle only cases of n <= m .

For n <= m we create a boolean DP table which will store the status of each value from 0 to m-1 which are possible subset sum (modulo m) which have been encountered so far.

Now we loop through each element of given array arr[], and we add (modulo m) j which have DP[j] = true and store all the such (j+arr[i])%m possible subset sum in a boolean array temp, and at the end of iteration over j, we update DP table with temp. Also we add arr[i] to DP ie.. DP[arr[i]%m] = true.

In the end if DP[0] is true then it means YES there exist a subset with sum which is divisible by m, else NO.

```
// C++ program to check if there is a subset
// with sum divisible by m.
#include <bits/stdc++.h>
using namespace std;

// Returns true if there is a subset
```

```
// of arr[] with sum divisible by m
bool modularSum(int arr[], int n, int m)
{
   if (n > m)
      return true;

   // This array will keep track of all
   // the possible sum (after modulo m)
   // which can be made using subsets of arr[]
   // initialising boolean array with all false
   bool DP[m];
   memset(DP, false, m);

   // we'll loop through all the elements of arr[]
   for (int i=0; i<n; i++)
   {
      // anytime we encounter a sum divisible
      // by m, we are done
      if (DP[0])
         return true;

      // To store all the new encountered sum (after
      // modulo). It is used to make sure that arr[i]
      // is added only to those entries for which DP[j]
      // was true before current iteration.
      bool temp[m];
      memset(temp,false,m);

      // For each element of arr[], we loop through
      // all elements of DP table from 1 to m and
      // we add current element i. e., arr[i] to
      // all those elements which are true in DP
      // table
      for (int j=0; j<m; j++)
      {
         // if an element is true in DP table
         if (DP[j] == true)
         {
            if (DP[(j+arr[i]) % m] == false)

               // We update it in temp and update
               // to DP once loop of j is over
               temp[(j+arr[i]) % m] = true;
         }
      }

      // Updating all the elements of temp
      // to DP table since iteration over
      // j is over
      for (int j=0; j<m; j++)
         if (temp[j])
            DP[j] = true;


      // Also since arr[i] is a single element
      // subset, arr[i]%m is one of the possible
      // sum
      DP[arr[i]%m] = true;
   }

   return DP[0];
}

// Driver code
int main()
{
   int arr[] = {1, 7};
   int n = sizeof(arr)/sizeof(arr[0]);
   int m = 5;

   modularSum(arr, n, m) ?  cout << "YESn" :
                            cout << "NOn";

   return 0;
}
```

HUGE MOD

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

ll phi(int n)
{
   ll ret=n;
   for(int i=2;i*i<=n;i++)
   {
      if(n%i==0)
      {
         while(n%i==0) n/=i;
         ret-=ret/i;
      }
   }
   if(n>1) ret-=ret/n;
   return ret;
}

ll bigmod(ll base,ll power, ll m)
{
   if(power==0) return 1ll;
```

```
    if(power==1) return base;
    ll x=bigmod(base,power/2,m);
    x=(x*x)%m;
    if(power&1) x=(x*base)%m;
    return x;
}

ll hugemod(string a,string b,int m)
{
    int la=a.length();
    int lb=b.length();
    ll A=0;
    for(int i=0;i<la;i++)
    {
        A=(A*10+a[i]-'0')%m;
    }
    ll x=1;
    for(int i=0;i<lb;i++)
    {
        x=bigmod(x,10,m);
        x=(x*bigmod(A,b[i]-'0',m))%m;
    }
    return x;
}

int main()
{
    string a,b;
    ll c;
    cin>>a>>b>>c;
    ll ans=hugemod(a,b,c);
    cout<<ans<<endl;
    return 0;
}
```

GAUSSIAN ELIMINATION

```
// C++ program to demostrate working of Guassian Elimination
// method
#include<bits/stdc++.h>
using namespace std;

#define N 3       // Number of unknowns

// function to reduce matrix to r.e.f.  Returns a value to
// indicate whether matrix is singular or not
int forwardElim(double mat[N][N+1]);

// function to calculate the values of the unknowns
void backSub(double mat[N][N+1]);

// function to get matrix content
void gaussianElimination(double mat[N][N+1])
{
    /* reduction into r.e.f. */
    int singular_flag = forwardElim(mat);

    /* if matrix is singular */
    if (singular_flag != -1)
    {
        printf("Singular Matrix.\n");

        /* if the RHS of equation corresponding to
           zero row  is 0, * system has infinitely
           many solutions, else inconsistent*/
        if (mat[singular_flag][N])
            printf("Inconsistent System.");
        else
            printf("May have infinitely many "
                   "solutions.");

        return;
    }

    /* get solution to system and print it using
       backward substitution */
    backSub(mat);
}

// function for elemntary operation of swapping two rows
void swap_row(double mat[N][N+1], int i, int j)
{
    //printf("Swapped rows %d and %d\n", i, j);

    for (int k=0; k<=N; k++)
    {
        double temp = mat[i][k];
        mat[i][k] = mat[j][k];
        mat[j][k] = temp;
    }
}

// function to print matrix content at any stage
void print(double mat[N][N+1])
{
    for (int i=0; i<N; i++, printf("\n"))
        for (int j=0; j<=N; j++)
            printf("%lf ", mat[i][j]);

    printf("\n");
```

```
}

// function to reduce matrix to r.e.f.
int forwardElim(double mat[N][N+1])
{
   for (int k=0; k<N; k++)
   {
      // Initialize maximum value and index for pivot
      int i_max = k;
      int v_max = mat[i_max][k];

      /* find greater amplitude for pivot if any */
      for (int i = k+1; i < N; i++)
         if (abs(mat[i][k]) > v_max)
            v_max = mat[i][k], i_max = i;

      /* if a prinicipal diagonal element  is zero,
       * it denotes that matrix is singular, and
       * will lead to a division-by-zero later. */
      if (!mat[k][i_max])
         return k; // Matrix is singular

      /* Swap the greatest value row with current row */
      if (i_max != k)
         swap_row(mat, k, i_max);


      for (int i=k+1; i<N; i++)
      {
         /* factor f to set current row kth elemnt to 0,
          * and subsequently remaining kth column to 0 */
         double f = mat[i][k]/mat[k][k];

         /* subtract fth multiple of corresponding kth
            row element*/
         for (int j=k+1; j<=N; j++)
            mat[i][j] -= mat[k][j]*f;

         /* filling lower triangular matrix with zeros*/
         mat[i][k] = 0;
      }

      //print(mat);       //for matrix state
   }
   //print(mat);          //for matrix state
   return -1;
}

// function to calculate the values of the unknowns
void backSub(double mat[N][N+1])
{
   double x[N];  // An array to store solution

   /* Start calculating from last equation up to the
      first */
   for (int i = N-1; i >= 0; i--)
   {
      /* start with the RHS of the equation */
      x[i] = mat[i][N];

      /* Initialize j to i+1 since matrix is upper
         triangular*/
      for (int j=i+1; j<N; j++)
      {
         /* subtract all the lhs values
          * except the coefficient of the variable
          * whose value is being calculated */
         x[i] -= mat[i][j]*x[j];
      }

      /* divide the RHS by the coefficient of the
         unknown being calculated */
      x[i] = x[i]/mat[i][i];
   }

   printf("\nSolution for the system:\n");
   for (int i=0; i<N; i++)
      printf("%lf\n", x[i]);
}

// Driver program
int main()
{
   /* input matrix */
   double mat[N][N+1] = {{3.0, 2.0,-4.0, 3.0},
                         {2.0, 3.0, 3.0, 15.0},
                         {5.0, -3, 1.0, 14.0}
                        };

   gaussianElimination(mat);

   return 0;
}
```

GAUSS JORDAN

```
// C++ Implementation for Gauss-Jordan
// Elimination Method
#include <bits/stdc++.h>
using namespace std;
```

```
#define M 10

// Function to print the matrix
void PrintMatrix(float a[][M], int n)
{
    for (int i = 0; i < n; i++) {
        for (int j = 0; j <= n; j++)
            cout << a[i][j] << " ";
        cout << endl;
    }
}

// function to reduce matrix to reduced
// row echelon form.
int PerformOperation(float a[][M], int n)
{
    int i, j, k = 0, c, flag = 0, m = 0;
    float pro = 0;

    // Performing elementary operations
    for (i = 0; i < n; i++)
    {
        if (a[i][i] == 0)
        {
            c = 1;
            while (a[i + c][i] == 0 && (i + c) < n)
                c++;
            if ((i + c) == n) {
                flag = 1;
                break;
            }
            for (j = i, k = 0; k <= n; k++)
                swap(a[j][k], a[j+c][k]);
        }

        for (j = 0; j < n; j++) {

            // Excluding all i == j
            if (i != j) {

                // Converting Matrix to reduced row
                // echelon form(diagonal matrix)
                float pro = a[j][i] / a[i][i];

                for (k = 0; k <= n; k++)
                    a[j][k] = a[j][k] - (a[i][k]) * pro;
            }
        }
    }
    return flag;
}

// Function to print the desired result
// if unique solutions exists, otherwise
// prints no solution or infinite solutions
// depending upon the input given.
void PrintResult(float a[][M], int n, int flag)
{
    cout << "Result is : ";

    if (flag == 2)
        cout << "Infinite Solutions Exists" << endl;
    else if (flag == 3)
        cout << "No Solution Exists" << endl;


    // Printing the solution by dividing constants by
    // their respective diagonal elements
    else {
        for (int i = 0; i < n; i++)
            cout << a[i][n] / a[i][i] << " ";
    }
}

// To check whether infinite solutions
// exists or no solution exists
int CheckConsistency(float a[][M], int n)
{
    int i, j;
    int f;

    // 2 for infinite solution
    // 3 for No solution
    for (i = 0; i < n; i++)
    {
        f = 0;
        for (j = 0; j < n; j++)
            f+=(a[i][j]==0);
        if (f==n && a[i][j]!=0)
            return 3;
    }
    return 2;
}

// Driver code
int main()
{
    float a[M][M] = {{ 0, 2, 1, 4 },
                     { 1, 1, 2, 6 },
```

```
                { 2, 1, 1, 7 }};

    // Order of Matrix(n)
    int n = 3, flag = 0;

    // Performing Matrix transformation
    flag = PerformOperation(a, n);

    if (flag == 1)
        flag = CheckConsistency(a, n);

    // Printing Final Matrix
    cout << "Final Augumented Matrix is : " << endl;
    PrintMatrix(a, n);
    cout << endl;

    // Printing Solutions(if exist)
    PrintResult(a, n, flag);

    return 0;
}
```

RANK OF MATRIX

```
// C++ program to find rank of a matrix
#include <bits/stdc++.h>
using namespace std;
#define R 3
#define C 3

/* function for exchanging two rows of
   a matrix */
void swap(int mat[R][C], int row1, int row2,
          int col)
{
    for (int i = 0; i < col; i++)
    {
        int temp = mat[row1][i];
        mat[row1][i] = mat[row2][i];
        mat[row2][i] = temp;
    }
}

// Function to display a matrix
void display(int mat[R][C], int row, int col);

/* function for finding rank of matrix */
int rankOfMatrix(int mat[R][C])
{
    int rank = C;

    for (int row = 0; row < rank; row++)
    {
        // Before we visit current row 'row', we make
        // sure that mat[row][0],....mat[row][row-1]
        // are 0.

        // Diagonal element is not zero
        if (mat[row][row])
        {
            for (int col = 0; col < R; col++)
            {
                if (col != row)
                {
                    // This makes all entries of current
                    // column as 0 except entry 'mat[row][row]'
                    double mult = (double)mat[col][row] /
                                  mat[row][row];
                    for (int i = 0; i < rank; i++)
                        mat[col][i] -= mult * mat[row][i];
                }
            }
        }

        // Diagonal element is already zero. Two cases
        // arise:
        // 1) If there is a row below it with non-zero
        //    entry, then swap this row with that row
        //    and process that row
        // 2) If all elements in current column below
        //    mat[r][row] are 0, then remvoe this column
        //    by swapping it with last column and
        //    reducing number of columns by 1.
        else
        {
            bool reduce = true;

            /* Find the non-zero element in current
               column  */
            for (int i = row + 1; i < R;  i++)
            {
                // Swap the row with non-zero element
                // with this row.
                if (mat[i][row])
                {
                    swap(mat, row, i, rank);
                    reduce = false;
                    break ;
                }
            }
```

```
        // If we did not find any row with non-zero
        // element in current columnm, then all
        // values in this column are 0.
        if (reduce)
        {
            // Reduce number of columns
            rank--;

            // Copy the last column here
            for (int i = 0; i < R; i ++)
                mat[i][row] = mat[i][rank];
        }

        // Process this row again
        row--;
    }

    // Uncomment these lines to see intermediate results
    // display(mat, R, C);
    // printf("\n");
  }
  return rank;
}

/* function for displaying the matrix */
void display(int mat[R][C], int row, int col)
{
  for (int i = 0; i < row; i++)
  {
    for (int j = 0; j < col; j++)
      printf("  %d", mat[i][j]);
    printf("\n");
  }
}

// Driver program to test above functions
int main()
{
 int mat[][3] = {{10,   20,   10},
                 {-20,  -30,   10},
                  {30,   50,   0}};
  printf("Rank of the matrix is : %d",
      rankOfMatrix(mat));
  return 0;
}
```

XOR GAUSS arb=rank

```
#include <stdio.h>
#include <algorithm>
using namespace std;
#define maxL (500>>5)+1
#define GET(x) (mark[x>>5]>>(x&31)&1)
#define SET(x) (mark[x>>5] |= 1<<(x&31))
int main() {
        // xor gauss
        int row = Pt, col = n, arb = 0;
        for(int r = 0, c = 0; r < row && c < col; c++) {
                int k = r;
                for(int j = r + 1; j < row; j++)
                        if(f[j][c]) k = j;
                for(int j = 0; j < col; j++)
                        swap(f[r][j], f[k][j]);

                if(f[r][c] == 0) {arb++;continue;}

                for(int j = 0; j < row; j++) {
                        if(r == j)  continue;
                        if(f[j][c]) {
                                for(int k = col; k >= c; k--)
                                        f[j][k] = f[j][k] ^
f[r][k];
                        }
                }
                r++;
        }
  return 0;
}
```

XOR MAXIMIZE

```
ll bits(ll n)
{
  ll cnt=0;
  while(n){
    cnt++;
    n/=2;
  }
  return cnt;
}
void gaussian(vector<ll> &v,int n)
{
  ll x=1ll<<62;
  for(int i=0;i<n && x;i++){
    if(!(x&v[i])){
      int f=0;
      for(int j=i+1;j<n;j++) if(x&v[j]) { swap(v[i],v[j]); f=1; break; }
      if(f==0){
        x>>=1;
```

```
            i--;
            continue;
          }
       }
       for(int j=0;j<n;j++) if(i!=j && x&v[j]) v[j]^=v[i];
       x>>=1;
       //cout<<i<<" --> "; for(int i=0;i<n;i++) cout<<" "<<v[i];
cout<<endl;
   }
}
int main()
{
   ///choosing a subset so that its xor is maximum
   ios_base::sync_with_stdio(false);
   int t,test=1;
   cin>>t;
   while(t--)
   {
      int n;
      cin>>n;
      vector<ll>v(n);
      for(int i=0;i<n;i++) cin>>v[i];
      //sort(v.rbegin(),v.rend());
      //dorkar nai, 63rd bit theke chalay dilei hobe
      gaussian(v,n);
      ll mx=0;
      for(int i=0;i<n;i++) mx^=v[i];
      cout<<"Case "<<test++<<": "<<mx<<endl;
   }
   return 0;
}
```

GAUSS JORDAN INVERSE FINDING

```
/*************** Gauss Jordan method for inverse matrix
*******************/
#include<bits/stdc++.h>


using namespace std;

int main()
{
   int i, j, k, n;
   float a[10][10] = { 0 }, d;
   cout << "No of equations ? ";
   cin >> n;
   cout << "Read all coefficients of matrix with b matrix too " <<
endl;
   for (i = 1; i <= n; i++)
      for (j = 1; j <= n; j++)
         cin >> a[i][j];

   for (i = 1; i <= n; i++)
      for (j = n+1; j <= 2 * n; j++)
         if (j == (i + n))
            a[i][j] = 1;

   /************** partial pivoting **************/
   for (i = n; i > 1; i--)
   {
      if (a[i - 1][1] < a[i][1])
         for (j = 1; j <= n * 2; j++)
         {
            d = a[i][j];
            a[i][j] = a[i - 1][j];
            a[i - 1][j] = d;
         }
   }
   cout << "pivoted output: " << endl;
   for (i = 1; i <= n; i++)
   {
      for (j = 1; j <= n * 2; j++)
         cout << a[i][j] << "     ";
      cout << endl;
   }
   /********** reducing to diagonal  matrix ***********/

   for (i = 1; i <= n; i++)
   {
      for (j = 1; j <= n * 2; j++)
         if (j != i)
         {
            d = a[j][i] / a[i][i];
            for (k = 1; k <= n * 2; k++)
               a[j][k] -= a[i][k] * d;
         }
   }
   /************** reducing to unit matrix *************/
   for (i = 1; i <= n; i++)
   {
      d = a[i][i];
      for (j = 1; j <= n * 2; j++)
         a[i][j] = a[i][j] / d;
   }

   cout << "your solutions: " << endl;
   for (i = 1; i <= n; i++)
   {
      for (j = n + 1; j <= n * 2; j++)
```

```
            cout << a[i][j] << "    ";
        cout << endl;
    }
    return 0;
}
```

FIND SUBARRAY WITH GIVEN SUM

```
// C++ program to print subarray with sum as given sum
#include<bits/stdc++.h>
using namespace std;

// Function to print subarray with sum as given sum
void subArraySum(int arr[], int n, int sum)
{
    // create an empty map
    unordered_map<int, int> map;

    // Maintains sum of elements so far
    int curr_sum = 0;

    for (int i = 0; i < n; i++)
    {
        // add current element to curr_sum
        curr_sum = curr_sum + arr[i];

        // if curr_sum is equal to target sum
        // we found a subarray starting from index 0
        // and ending at index i
        if (curr_sum == sum)
        {
            cout << "Sum found between indexes "
                 << 0 << " to " << i << endl;
            return;
        }

        // If curr_sum - sum already exists in map
        // we have found a subarray with target sum
        if (map.find(curr_sum - sum) != map.end())
        {
            cout << "Sum found between indexes "
                 << map[curr_sum - sum] + 1
                 << " to " << i << endl;
            return;
        }

        map[curr_sum] = i;
    }

    // If we reach here, then no subarray exists
    cout << "No subarray with given sum exists";
}

// Driver program to test above function
int main()
{
    int arr[] = {10, 2, -2, -20, 10};
    int n = sizeof(arr)/sizeof(arr[0]);
    int sum = -12;

    subArraySum(arr, n, sum);

    return 0;
}
```

EXTENDED EUCLID

```
pii extendedEuclid(ll a,ll b)
{
    if(b==0) return pii(1,0);
    else
    {
        pii d=extendedEuclid(b,a%b);
        return pii(d.sc,d.fs-d.sc*(a/b));
    }
}
ll modularInverse(ll a,ll n)
{
    pii ret=extendedEuclid(a,n);
    return ((ret.fs%n)+n)%n;
}
```

EDIT DISTANCE

```
#include<bits/stdc++.h>
using namespace std;
string aa,bb;
int dp[1000][1000];

int rec_edit(int i,int j)
{
    ///(i,j) means (0 to i),(0 to j) will be calculated and stored in (i,j)
    if(dp[i][j]!=-1) return dp[i][j];
    if(i==0) return j;
    if(j==0) return i;
    if(aa[i-1]==bb[j-1]) return dp[i][j]=rec_edit(i-1,j-1);
    return
dp[i][j]=1+min(rec_edit(i-1,j),min(rec_edit(i,j-1),rec_edit(i-1,j-1)));
}
void print_edit(int n,int m, int table[100][100])
{
```

```
  string a=aa;///for printing the string while editing
  ///all opertaions done on first string
  int i=n;
  int j=m;
  while(i>-1 && j>-1)
  {
     if(i==0 && j==0) break;
     else if(i==0)
     {
        cout<<j<<"I";///j characters inserted
        a.insert(0,bb.substr(0,j));
        i=j=-2;
     }
     else if(j==0)
     {
        cout<<i<<"D";///i characters deleted
        a.erase(0,i);
        i=j=-2;
     }
     else if(aa[i-1]==bb[j-1])
     {
        cout<<"M";///match found
        i--;
        j--;
     }
     else
     {
        if(table[i][j]==1+table[i-1][j-1])
        {
           cout<<"S";///character aa[i] substituted by charater bb[j]
           a[i-1]=bb[j-1];
           i--;
           j--;
        }
        else if(table[i][j]==1+table[i-1][j])
        {
           cout<<"D";/// character aa[i] deleted
           a.erase(a.begin()+i-1);
           i--;
        }
        else if(table[i][j]==1+table[i][j-1])
        {
           cout<<"I";///character bb[j] inserted at i
           a.insert(a.begin()+i,bb[j-1]);
           j--;
        }
     }
     cout<<"  string: "<<a<<endl;
     if(i==-2 || j==-2) break;
  }
}

int table_edit(int n,int m)
{
   //int table[n+2][m+2];
   int table[100][100];
   for(int i=0; i<=n; i++)
   {
      for(int j=0; j<=m; j++)
      {
         if(i==0) table[i][j]=j;
         else if(j==0) table[i][j]=i;
         else if(aa[i-1]==bb[j-1]) table[i][j]=table[i-1][j-1];
         else table[i][j]=1+min( table[i-1][j-1], min(
table[i-1][j],table[i][j-1] ) );
      }
   }
   /*for(int i=0;i<=n;i++)
   {
      for(int j=0;j<=m;j++)
      {
         cout<<table[i][j];
      }
      cout<<endl;
   }*/
   print_edit(n,m,table);
   return table[n][m];
}

int space_optimised(int n,int m)
{
   int table[2][m+1];
   int bi;
   for(int i=0; i<=n; i++)
   {
      bi=i&1;
      for(int j=0; j<=m; j++)
      {
         if(i==0) table[bi][j]=j;
         else if(j==0) table[bi][j]=i;
         else if(aa[i-1]==bb[j-1]) table[bi][j]=table[1-bi][j-1];
         else
table[bi][j]=1+min(table[1-bi][j],min(table[bi][j-1],table[1-bi][j-1]));
      }
   }
   return table[bi][m];
}


int main()
```

```
{
   cin>>aa>>bb;
   memset(dp,-1,sizeof dp);
   cout<<rec_edit(aa.length(),bb.length())<<endl;
   cout<<endl<<table_edit(aa.length(),bb.length())<<endl;
   cout<<space_optimised(aa.length(),bb.length())<<endl;
   return 0;
}
```

DSU ON TREE lomsat gelral

```
/*
You are given a rooted tree with root in vertex 1. Each vertex is coloured in some colour.
Let's call colour c dominating in the subtree of vertex v if there are no other colours that appear in the subtree of vertex v more times than colour c. So it's possible that two or more colours will be dominating in the subtree of some vertex.
The subtree of vertex v is the vertex v and all other vertices that contains vertex v in each path to the root.
For each vertex v find the sum of all dominating colours in the subtree of vertex v.
*/
#define nd 100005
using namespace std;
vector<int>gr[nd],col(nd),cnt(nd),siz(nd),st(nd),fn(nd),ver(nd);
vector<LL>ans(nd);
int t,n;
int mex=0;
LL ANS=0;
void dfssiz(int u,int p)
{
   t++;
   ver[t]=u;
   st[u]=t;
   siz[u]=1;
   for(int i=0; i<gr[u].size(); i++)
   {
      if(gr[u][i]!=p)
      {
         dfssiz(gr[u][i],u);
         siz[u]+=siz[gr[u][i]];
      }
   }
   fn[u]=t;
}
void dfs(int u,int p,int keep)
{
   int mx=-1,bigchild=-1;
   for(int i=0; i<gr[u].size(); i++)
   {
      int v=gr[u][i];
      if(v==p) continue;
      if(mx<siz[v])
      {
         mx=siz[v];
         bigchild=v;
      }
   }

   for(int i=0; i<gr[u].size(); i++)
   {
      int v=gr[u][i];
      if(v==p || v==bigchild) continue;
      dfs(v,u,0);
   }

   if(bigchild!=-1)dfs(bigchild,u,1);

   int temp=mex;
   cnt[col[u]]++;
   mex=max(mex,cnt[col[u]]);
   for(int i=0; i<gr[u].size(); i++)
   {
      int v=gr[u][i];
      if(v==p || v==bigchild) continue;
      for(int j=st[v]; j<=fn[v]; j++)
      {
         cnt[ col[ ver[j] ] ]++;
         mex=max(mex,cnt[ col[ ver[j] ] ]);
      }
   }
   if(mex!=temp) ANS=0;
   set<int>stk;
   for(int i=0; i<gr[u].size(); i++)
   {
      int v=gr[u][i];
      if(v==p || v==bigchild) continue;
      for(int j=st[v]; j<=fn[v]; j++)
      {
         if(cnt[ col[ ver[j] ] ]==mex) stk.insert(col[ ver[j] ]);
      }
   }
   if(cnt[col[u]]==mex) stk.insert(col[u]);
   for(auto i=stk.begin();i!=stk.end();i++)
   {
      ANS+=0ll+(*i);
   }
   ans[u]=ANS;
```

```
    if(keep==0) {mex=0;ANS=0;}
    if(keep==0)
        for(int i=st[u]; i<=fn[u]; i++)
        {
            cnt[ col[ ver[i] ] ]--;
        }
    return ;
}

int main()
{
    ios_base::sync_with_stdio(false);
    cin>>n;
    for(int i=1; i<=n; i++)
    {
        cin>>col[i];
    }
    for(int i=1; i<n; i++)
    {
        int x,y;
        cin>>x>>y;
        gr[x].pb(y);
        gr[y].pb(x);
    }
    t=0;
    dfssiz(1,-1);
    dfs(1,-1,0);
    for(int i=1; i<=n; i++)
    {
        if(i>1) cout<<" ";
        cout<<ans[i];
    }
    return 0;
}
```

DISCRETE LOG sqrt(m)log(b)

```
// C++ program to calculate discrete logarithm
#include<bits/stdc++.h>
using namespace std;

/* Iterative Function to calculate (x ^ y)%p in
  O(log y) */
int powmod(int x, int y, int p)
{
    int res = 1;  // Initialize result

    x = x % p;  // Update x if it is more than or
                // equal to p
    while (y > 0)
    {
        // If y is odd, multiply x with result
        if (y & 1)
            res = (res*x) % p;

        // y must be even now
        y = y>>1; // y = y/2
        x = (x*x) % p;
    }
    return res;
}

// Function to calculate k for given a, b, m
int discreteLogarithm(int a, int b, int m) {

    int n = (int) sqrt (m) + 1;

    unordered_map<int, int> value;

    // Store all values of a^(n*i) of LHS
    for (int i = n; i >= 1; --i)
        value[ powmod (a, i * n, m) ] = i;

    for (int j = 0; j < n; ++j)
    {
        // Calculate (a ^ j) * b and check
        // for collision
        int cur = (powmod (a, j, m) * b) % m;

        // If collision occurs i.e., LHS = RHS
        if (value[cur])
        {
            int ans = value[cur] * n - j;
            // Check whether ans lies below m or not
            if (ans < m)
                return ans;
        }
    }
    return -1;
}

// Driver code
int main()
{
    int a = 2, b = 3, m = 5;
    cout << discreteLogarithm(a, b, m) << endl;

    a = 3, b = 7, m = 11;
    cout << discreteLogarithm(a, b, m);
```

```
}
```

DISCRETE LOG sqrt(m)

```
// C++ program to calculate discrete logarithm
#include<bits/stdc++.h>
#define ll long long
using namespace std;

ll discreteLogarithm(ll a, ll b, ll m)
{
    if(b==1) return 0;
    ll n=(ll)sqrt (m+.0)+1;
    // Calculate a ^ n
    ll an=1;
    for (ll i=0;i<n;++i)
        an=(an*a)%m;

    unordered_map<ll, ll> value;

    // Store all values of a^(n*i) of LHS
    for (ll i=1, cur=an; i<=n; ++i)
    {
        if (!value.count(cur))
            value[cur]=i;
        cur=(cur*an)%m;
    }

    for (ll i=0, cur=b; i<=n; ++i)
    {
        // Calculate (a ^ j) * b and check
        // for collision
        if (value.count(cur))
        {
            ll ans = value[cur] * n - i;
            if (ans < m)
                return ans;
        }
        cur=(cur*a)%m;
    }
    return -1;
}

// Driver code
int main()
{
    ll a = 3, b = 9, m = 100000007;
    cout << discreteLogarithm(a, b, m) << endl;

    a = 2, b = 100, m = 100000007;
    cout << discreteLogarithm(a, b, m);
}
```

DIJKSTRA

```
template <class T>

struct Dijkstra
{
    int n;
    vector< vector< pair<int,T> > > adj; ///the weight can be
int/double so we use template
    ///when calling Dijkstra we put the type in <T>
    vector<T>dist;
    vector<int>parent;
    Dijkstra (int n): n(n),adj(n){}
    void addedge(int x,int y,T w)
    {
        adj[x].pb( {y,w} );
        adj[y].pb( {x,w} );
    }
    void shortestpath(int src)
    {
        priority_queue<pair<T,int>,vector<pair<T,int>> ,
greater<pair<T,int>> >q;
        dist = vector<T>(n,numeric_limits<T>::max()); ///T type er
max value is in numeric limits max()
        parent = vector<int>(n,-1);
        dist[src]=0;
        parent[src]=src;
        q.push({0,src});
        while(!q.empty())
        {
            int u=q.top().sc;
            q.pop();
            for(int i=0;i<adj[u].size();i++)
            {
                int v=adj[u][i].fs;
                T vd=adj[u][i].sc;
                if(dist[u]<dist[v]-vd)///to avoid overflow
                {
                    dist[v]=dist[u]+vd;
                    q.push({dist[v],v});
                    parent[v]=u;
                }
            }
        }
    }
};
```

```
using namespace std;

int main()
{
   int t,cas=1;
   cin>>t;
   while(t--)
   {
      int n,m,s,d;
      cin>>n>>m>>s>>d;
      Dijkstra<int> D(n);
      while(m--)
      {
         int x,y,w;
         cin>>x>>y>>w;
         D.addedge(x,y,w);
      }
      D.shortestpath(s);
      cout<<"Case #"<<cas++<<": ";
      if(D.dist[d]==numeric_limits<int>::max())
cout<<"unreachable"<<endl;
      else cout<<D.dist[d]<<endl;
   }

   return 0;
}
```

COUNTING SORT

```
// C Program for counting sort
#include <stdio.h>
#include <string.h>
#define RANGE 255

// The main function that sort the given string arr[] in
// alphabatical order
void countSort(char arr[])
{
   // The output character array that will have sorted arr
   char output[strlen(arr)];

   // Create a count array to store count of inidividul
   // characters and initialize count array as 0
   int cnt[RANGE + 1], i;
   memset(cnt, 0, sizeof(cnt));

   // Store count of each character
   for(i = 0; arr[i]; ++i)
      ++cnt[arr[i]];

   // Change cnt[i] so that cnt[i] now contains actual
   // position of this character in output array
   for (i = 1; i <= RANGE; ++i)///cummulative
      cnt[i] += cnt[i-1];

   // Build the output character array
   for (i = 0; arr[i]; ++i)
   {
      output[cnt[arr[i]]-1] = arr[i];
      --cnt[arr[i]];
   }

   // Copy the output array to arr, so that arr now
   // contains sorted characters
   for (i = 0; arr[i]; ++i)
      arr[i] = output[i];
}

// Driver program to test above function
int main()
{
   char arr[] = "geeksforgeeks";//"applepp";

   countSort(arr);

   printf("Sorted character array is %s\n", arr);
   return 0;
}
```

COUNTING BITS

```
int normal_count(int n)
{
   int cnt=0;
   while(n)
   {
      cnt+=n&1;
      n=n>>1;
   }
   return cnt;
}

int Brian_Kernighan(int n)
{
   int cnt=0;
   while(n)
   {
      n=n&(n-1);
      cnt++;
   }
```

```
    return cnt;
}
///http://www.geeksforgeeks.org/next-higher-number-with-same-number-of-set-bits/
/**
Going upwards:
Find the rightmost occurrence of "01" in the number and make it "10".
Justify all following 1-bits as far to the right as possible.

Going downwards:
Find the rightmost occurrence of "10" in the number and make it "01".
Left-justify all following 1-bits (i.e. don't do anything if the bit you just set is already followed by a 1).
*/
int next_greater(int n)
{
    int a=(~n)+1; //-n
    int t=a&n;
    int x=n+t;
    int y=x^n;
    y=y/t;
    y=y>>2;
    return x|y;
}
int next_small(int y)
{
    int t = y + 1;
    int u = t ^ y;
    int v = t & y;
    return v - (v & -v) / (u + 1);
}
int main()
{
    ///built_in
    //cout<< __builtin_popcount(7) << endl;
    /*
    __builtin_popcount = int
    __builtin_popcountl = long int
    __builtin_popcountll = long long
    */
    cout<<next_greater(156)<<endl;
    return 0;
}
```

ALL POSSIBLE INCREASING SUBSEQUENCE segment tree (nlogn)

```
ll tree[400000];
ll query (int node,int b, int e,int i,int j)
{
    if(i<=b and e<=j) return tree[node];
    else if(i>e || j<b) return 0;
    else
    {
        int m=(b+e)/2;
        int lc=node*2;
        int rc=lc+1;
        return query(lc,b,m,i,j)+query(rc,m+1,e,i,j);
    }
}
void update (int node,int b, int e,int val,int i)
{
    if(b==e && b==i) tree[node]=val;
    else if(i>=b and i<=e)
    {
        int m=(b+e)/2;
        int lc=node*2;
        int rc=lc+1;
        update(lc,b,m,val,i);
        update(rc,m+1,e,val,i);
        tree[node]=tree[lc]+tree[rc];
    }
}
bool cmp(pii a,pii b)
{
    if(a.fs==b.fs) return a.sc>b.sc; ///(for strictly increasing)
    else return a.fs<b.fs;
}
int main()
{
    int t;
    scanf("%d",&t);
    int cas=1;
    while(t--)
    {
        int n;
        scanf("%d",&n);
        memset(tree,0,sizeof tree);
        vector<pii>ara(n+1);
        for(int i=1;i<=n;i++)
        {
            scanf("%d",&ara[i].fs);
            ara[i].sc=i;
        }
        ll ans=0;
        ll a=0;
        sort(ara.begin(),ara.end(),cmp);
        for(int i=1;i<=n;i++)
```

```
    {
      a=query(1,1,n,1,ara[i].sc);
      update(1,1,n,(a+1)%md,ara[i].sc);
    }
    printf("Case %d: %lld\n",cas++,tree[1]%md);
  }
  return 0;
}
```

ALL POSSIBLE INCREASING SUBSEQUENCE dp (n^2)

```
///----------DP (n^2) to find number of all possible increasing subsequence-------///
int n;
int ara[100002];
ll dp[100002];
ll all_is(int pos)
{
  if(pos>=n) return 0;
  if(dp[pos]!=-1) return dp[pos];
  ll ans=1;
  for(int i=pos+1;i<n;i++)
  {
    if(ara[i]>ara[pos]) ans=(ans+all_is(i))%MOD;
  }
  return dp[pos]=ans;
}

int main()
{
  int t;
  int cas=1;
  scanf("%d",&t);
  while(t--)
  {
    scanf("%d",&n);
    for(int i=0;i<n;i++)
    {
      scanf("%d",&ara[i]);
      dp[i]=-1;
    }
    ///---------DP-----------///
    ll ans=0;
    for(int i=0;i<n;i++) ans=(ans+all_is(i))%MOD;
    printf("Case %d: %lld\n",cas++,ans);
    ///--------table (n^2) to find no. of all subs-----//
    ll table [100002];
    ///table[i] is no. of increasing sequences ending at i
    for(int i=0;i<n;i++) table[i]=1;
    for(int i=1;i<n;i++)
    {
      for(int j=0;j<i;j++)
      {
        if(ara[j]<ara[i]) table[i]+=table[j];
      }
    }
    ans=0;
    for(int i=0;i<n;i++) ans+=table[i];
    printf("Case %d: %lld\n",cas++,ans);
  }
  return 0;
}
```

ALL PALINDROMIC SUBSEQUENCE

```
char s[100];
ll dp[61][61];
ll rec(int i,int j)
{
  if(i==j) return 1;
  if(i>j) return 0;
  if(dp[i][j]!=-1) return dp[i][j];
  ll ret;
  if(s[i]==s[j]) ret=1+rec(i+1,j)+rec(i,j-1);
  else ret=rec(i+1,j)+rec(i,j-1)-rec(i+1,j-1);
  return dp[i][j]=ret;
}
int main()
{
  int t;
  int cas=1;
  scanf("%d",&t);
  getchar();
  while(t--)
  {
    memset(dp,-1,sizeof(dp));
    gets(s);
    printf("Case %d: %lld\n",cas++,rec(0,strlen(s)-1));
  }
  return 0;
}
```

BIT nlogn, logn

```
/**
  * To get parent
  * 1) 2's complement to get minus of index
  * 2) AND this with index
  * 3) Subtract that from index
  */
```

```
int getParent(int ind)
{
   return ind=ind - (ind & -ind);
}

/**
    * To get next
    * 1) 2's complement of get minus of index
    * 2) AND this with index
    * 3) Add it to index
    */
int getNext(int ind)
{
   return ind=ind + (ind & -ind);
}
#define maxn 1000+5

int n;
vector<int>ara(maxn),tree(maxn);

void update(int ind,int val)
{
   ind++;
   while(ind<n+1)
   {
      tree[ind]+=val;
      ind=getNext(ind);
   }
}

int getSum(int ind) ///cummulative sum from 0 to ind
{
   ind++;
   int sum=0;
   while(ind>0)
   {
      sum+=tree[ind];
      ind=getParent(ind);
   }
   return sum;
}

void buildTree()
{
   ///first tree is initialised to 0;
   ///do update operation for all nodes
   for(int i=0;i<n;i++)
   {
      update(i,ara[i]);
   }
}

int main()
{
   cin>>n;
   for(int i=0;i<n;i++)
   {
      cin>>ara[i];
   }
   buildTree();
   cout<<getSum(8)<<" "<<getSum(4)<<endl;
   update(3,7);
   cout<<getSum(8)<<" "<<getSum(4)<<endl;
   update(2,8);
   cout<<getSum(8)<<" "<<getSum(4)<<endl;
   return 0;
}
```

BELL NUMBER

```
///http://www.geeksforgeeks.org/bell-numbers-number-of-ways-to-partition-a-set/
// A C++ program to find n'th Bell number
#include<iostream>
using namespace std;

int bellNumber(int n)
{
  int bell[n+1][n+1];
  bell[0][0] = 1;
  for (int i=1; i<=n; i++)
  {
    // Explicitly fill for j = 0
    bell[i][0] = bell[i-1][i-1];

    // Fill for remaining values of j
    for (int j=1; j<=i; j++)
      bell[i][j] = bell[i-1][j-1] + bell[i][j-1];
  }
  return bell[n][0];
}

// Driver program
int main()
{
  for (int n=0; n<=5; n++)
    cout << "Bell Number " << n << " is "
         << bellNumber(n) << endl;
  return 0;
}
```

AHO CORASICK

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

///Max number of states = sum of length of patterns
#define MAXS 500
///Max characters
#define MAXC 26

///list of output strings in each node
///use int if u wanna keep indices of patterns in out
vector<string>out[MAXS];

///failure nodes
int f[MAXS];

int trie[MAXS][MAXC];

int buildTrie(vector<string>pat, int n)
{
   /** Building Trie */

   for(int i=0;i<MAXS;i++) out[i].clear();
   memset(trie,-1,sizeof trie);

   ///ROOT is 0
   int states=1;

   for(int i=0;i<n;i++)
   {
      string str=pat[i];
      int node=0;

      for(int j=0;j<str.length();j++)
      {
         int ch=str[j]-'a';

         if(trie[node][ch]==-1) trie[node][ch]=states++;

         node=trie[node][ch];
      }

      out[node].pb(str);
   }

   /** Failure Function Building */

   ///adding a track back to root, for those children of root that
dont exist
   for(int i=0;i<MAXC;i++) if(trie[0][i]==-1) trie[0][i]=0;

   memset(f, -1, sizeof f);

   queue<int>q;

   ///All children of root fail to root.
   for(int i=0;i<MAXC;i++) if(trie[0][i]!=0)
f[trie[0][i]]=0,q.push(trie[0][i]);

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


      for(int i=0;i<MAXC;i++)
      {
         if(trie[node][i]!=-1)
         {
            int failure=f[node];

            while(trie[failure][i]==-1)
            {
               failure=f[failure];
            }

            failure=trie[failure][i];
            f[trie[node][i]]=failure;

            for(int k=0;k<out[failure].size();k++)
            {
               out[trie[node][i]].pb(out[failure][k]);
            }

            q.push(trie[node][i]);
         }
      }
   }
   return states;
   ///no. of states in trie
}

void AhoCorasick(vector<string>pat, int n, string text)
{
   ///builds trie, failure and stores outputs
   buildTrie(pat,n);
```

```
    int curState=0;

    for(int i=0;i<text.size();i++)
    {
        int ch=text[i]-'a';

        while(trie[curState][ch]==-1)
        {
            curState=f[curState];
        }
        curState=trie[curState][ch];

        for(int j=0;j<out[curState].size();j++)
        {
            //if same word is in dictionary multiple times, will be
printed multiple times
            //also if word in text multiple times, also printed multiple
times
            //these are to be handled according to the problem
            cout<<out[curState][j]<< " appears in text from
"<<i-out[curState][j].size()+1<<"\n";
        }
    }
}


int main()
{
    int n;
    cin>>n;
    vector<string>pat(n);
    for(int i=0;i<n;i++) cin>>pat[i];
    string text;
    cin>>text;
    AhoCorasick(pat,n,text);
    return 0;
}
/*
4
he she hers his
ahishers
*/
/**
I stored in out all the strings that end at a specific node
I could also: use bitset to mark those positions (index of strings)
which are output at that position //bitset<MAXS>out[MAXS];
*/
```

AHO CORASICK linked list

```
const int NN = 1000005;
const int MM = 505;

int cases, caseno, n, res[MM];
char T[NN], A[MM][MM];

struct node {
    node *next[27];
    vector <int> out;
    node() {
        for( int i = 0; i < 27; i++ ) next[i] = NULL;
        out.clear();
    }
};
queue <node *> Q;

node* buildTrieWithFailure( char A[MM][MM], int n ) {
    node *root = new node;
    for( int i = 0; i < n; i++ ) {
        node *p = root;
        for( int j = 0; A[i][j]; j++ ) {
            int c = A[i][j] - 96;
            if( !p->next[c] ) p->next[c] = new node;
            p = p->next[c];
        }
        p->out.push_back(i);
    }
    for( int i = 1; i < 27; i++ ) {
        if( !root->next[i] ) root->next[i] = root;
        else {
            Q.push( root->next[i] );
            root->next[i]->next[0] = root;
        }
    }
    while( !Q.empty() ) {
        node *u = Q.front(); Q.pop();
        for( int i = 1; i < 27; i++ ) if( u->next[i] ) {
            node *v = u->next[i];
            node *w = u->next[0];
            while( !w->next[i] ) w = w->next[0];
            v->next[0] = w = w->next[i];
            for( int j = 0; j < w->out.size(); j++ ) v->out.push_back(
w->out[j] );
            Q.push(v);
        }
    }
    return root;
```

precode@tanny411
aysha.kamal7@gmail.com


```
}

void AhoCorasick( node *p, char *T ) {
   for( int i = 0; T[i]; i++ ) {
      int c = T[i] - 96;
      while( !p->next[c] ) p = p->next[0];
      p = p->next[c];
      for( int j = 0; j < p->out.size(); j++ ) res[ p->out[j] ]++;
   }
}

void deleteTrie( node *p ) {
   for( int i = 1; i < 27; i++ ) if( p->next[i] && p->next[i] != p )
deleteTrie( p->next[i] );
   delete p;
}

int main() {
   //freopen("a.in", "r", stdin);
   //freopen("a2.ans", "w", stdout);

   double cl = clock();

   scanf("%d", &cases);
   while( cases-- ) {
      scanf("%d %s", &n, T);
      assert( 1 <= n && n <= 500 );
      for( int i = 0; i < n; i++ ) {
         scanf("%s", A[i]);
         assert( strlen( A[i] ) <= 500 );
         res[i] = 0;
      }
      node *root = buildTrieWithFailure( A, n );
      AhoCorasick( root, T );

      printf("Case %d:\n", ++caseno);
      for( int i = 0; i < n; i++ ) printf("%d\n", res[i]);

      cerr << cases << endl;
      deleteTrie( root );
   }

   cl = clock() - cl;
   fprintf(stderr, "Total Execution Time = %lf seconds\n", cl /
CLOCKS_PER_SEC);

   return 0;
}
```

3 STRING LCS

```
///3 string lcs
///can be extended to n strings by n-dimensional array
int do_lcs(string a,string b,string c)
{
   int la=a.length();
   int lb=b.length();
   int lc=c.length();
   int ara[la+1][lb+1][lb+1];
   for(int i=0;i<=la;i++)
   {
      for(int j=0;j<=lb;j++)
      {
         for(int k=0;k<=lc;k++)
         {
            if(i==0 || j==0 || k==0) ara[i][j][k]=0;
            else if(a[i-1]==b[j-1]&& b[j-1]==c[k-1])
ara[i][j][k]=1+ara[i-1][j-1][k-1];
            else
ara[i][j][k]=max(ara[i-1][j][k],max(ara[i][j-1][k],ara[i][j][k-1]));
         }
      }
   }
   return ara[la][lb][lc];
}

int main()
{
   string a,b,c;
   cin>>a>>b>>c;
   cout<<do_lcs(a,b,c)<<endl;
   return 0;
}
```

2D KADANE

```
int kadane(vector<int>v, int* start, int* finish)///address of start
and finish
{
   int sum=0,maxsum=-1<<30,s;
   *finish=-1;

   for(int i=0;i<v.size();i++)
   {
      sum+=v[i];
      if(sum<0)
      {
         sum=0;
         s=i+1;
```

```
        }
        else if(sum>maxsum)
        {
            *start=s;
            *finish=i;
            maxsum=sum;
        }
    }
    if(*finish!=-1) return maxsum;

    for(int i=0;i<v.size();i++)
    {
        if(maxsum<v[i])
        {
            maxsum=v[i];
            *start=*finish=i;
        }
    }
    return maxsum;
}

int kadane_2D(vector< vector<int> >v , int* top,int* left,int* right,
int* bottom)
{
    int MX=-1<<30;
    ///select two columns by n^2
    ///for each row in the range of the selected column find
cummulative sum
    ///run kadane in that cummulative sum array to find the rows
of either sides
    int n=v.size();///row
    int m=v[0].size();///column
    ///start and end column
    for(int stcol=0;stcol<m;stcol++)
    {
        vector<int>temp(n);

        for(int encol=stcol;encol<m;encol++)
        {
            for(int i=0;i<n;i++) temp[i]+=v[i][encol];

            int start,finish;
            int mx=kadane(temp,&start,&finish);
            if(mx>MX)
            {
                MX=mx;
                *left=stcol;
                *right=encol;
                *top=start;
                *bottom=finish;
            }
        }
    }
    return MX;
}

int main()
{
/*int M[4][5] = {{1, 2, -1, -4, -20},
                {-8, -3, 4, 2, 1},
                {3, 8, 10, 1, 3},
                {-4, -1, 1, 7, -6}
               };*/
    //int M[4][5]={
-10,-20,-3,-4,-5,-10,-20,-38,-74,-85,-107,-20,-37,-48,-500,-100,-200
,-30,-40,-50};

    vector< vector<int> >v(4,vector<int>(5));
    for(int i=0;i<4;i++)
    {
        for(int j=0;j<5;j++)
        {
            v[i][j]=M[i][j];
        }
    }
    int t,l,r,b;
    cout<<kadane_2D(v,&t,&l,&r,&b)<<endl;
    cout<<"top "<<t<<endl;
    cout<<"left "<<l<<endl;
    cout<<"right "<<r<<endl;
    cout<<"bottom "<<b<<endl;
    return 0;
}
```

FINDING ONE NEG CYCLE in graph

```
void solve()
{
    vector<int> d (n, INF);
    d[v] = 0;
    vector<int> p (n - 1);
    int x;
    for (int i=0; i<n; ++i)
    {
        x = -1;
        for (int j=0; j<m; ++j)
            if (d[e[j].a] < INF)
                if (d[e[j].b] > d[e[j].a] + e[j].cost)
                {
                    d[e[j].b] = max (-INF, d[e[j].a] + e[j].cost);
```

```
                p[e[j].b] = e[j].a;
                x = e[j].b;
            }
    }

    if (x == -1)
        cout << "No negative cycle from " << v;
    else
    {
        int y = x;
        for (int i=0; i<n; ++i)
            y = p[y];

        vector<int> path;
        for (int cur=y; ; cur=p[cur])
        {
            path.push_back (cur);
            if (cur == y && path.size() > 1)
                break;
        }
        reverse (path.begin(), path.end());

        cout << "Negative cycle: ";
        for (size_t i=0; i<path.size(); ++i)
            cout << path[i] << ' ';
    }
}
/**
```

Everywhere above we considered that there is no negative cycle in the graph (precisely, we are interested in a negative cycle that is reachable
from the starting vertex v, and, for an unreachable cycles nothing in the above algorithm changes). In the presence of a negative cycle(s),
there are further complications associated with the fact that distances to all vertices in this cycle, as well as the distances to the vertices
reachable from this cycle is not defined — they should be equal to minus infinity (−∞).
It is easy to see that the Bellman-Ford algorithm can endlessly do the relaxation among all vertices of this cycle and the vertices reachable from it. Therefore, if you do not limit the number of phases to n−1, the algorithm will run indefinitely, constantly improving
the distance from these vertices.

Hence we obtain the criterion for presence of a cycle of negative weights reachable for source vertex v: after (n−1)th phase,
if we run algorithm for one more phase, and it performs at least one more relaxation, then the graph contains a negative weight cycle that is reachable from v; otherwise, such a cycle does not exist.

Moreover, if such a cycle is found, the Bellman-Ford algorithm can be modified so that it retrieves this cycle as a sequence
of vertices contained in it. For this, it is sufficient to remember the last vertex x for which there was a relaxation in nth phase.
This vertex will either lie in a negative weight cycle, or is reachable from it. To get the vertices that are guaranteed to lie in a negative cycle, starting from the vertex x, pass through to the predecessors n times. Hence we will get the vertex y, namely the vertex
in the cycle earliest reachable from source. We have to go from this vertex, through the predecessors, until we get back to the same vertex
y (and it will happen, because relaxation in a negative weight cycle occur in a circular manner).

Due to the presence of a negative cycle, for n iterations of the algorithm, the distances may go far in the negative
(apparently, to negative numbers of order − 2n). Hence in the code, we adopted additional measures against the integer overflow as follows:
d[e[j].b] = max (-INF, d[e[j].a] + e[j].cost);
The above implementation looks for a negative cycle reachable from some starting vertex v; however, the algorithm can be modified to
just looking for "any negative cycle in the graph". For this we need to put all the distance d[i] to zero and not infinity — as if we are looking for the shortest path from all vertices simultaneously; the validity of the detection of a negative cycle is not affected.

```
 */
```

LCA nlogn, logn

```
#define maxnode 10000+2
#define logmaxnode (int)log2(10000)+2
int n,start;
vector<int>gr[maxnode],lvl(maxnode,-1),parent(maxnode);

void _dfs(int node)
{
   for(int i=0; i<gr[node].size(); i++)
   {
      int v=gr[node][i];
      if(lvl[v]!=-1) continue;
      parent[v]=node;
      lvl[v]=lvl[node]+1;
      _dfs(v);
   }
```

```
}

int table[maxnode][logmaxnode];

void init_lca()
{
   memset(table,-1, sizeof table);

   for(int j=0; (1<<j)<n; j++)
   {
      for(int i=0; i<n; i++)
      {
         if(j==0) table[i][j]=parent[i];
         else if(table[i][j-1]!=-1) table[i][j]=table[table[i][j-1]][j-1];
      }
   }
}

int lca_query(int p,int q)
{
   if(lvl[q]>lvl[p]) swap(p,q);

   //int lg=log2(lvl[p]);
   int lg;
   for(lg=1; (1<<lg)<=lvl[p]; lg++);
   lg--;

   for(int i=lg; i>=0; i--)
   {
      if(lvl[p]-(1<<i)>=lvl[q])
      {
         p=table[p][i];
      }
   }

   if(p==q) return q;

   for(int i=lg; i>=0; i--)
   {
      if(table[p][i]!=-1 && table[p][i]!=table[q][i])
      {
         p=table[p][i];
         q=table[q][i];
      }
   }

   return parent[p];
}

int main()
{
   int m;
   cin>>n>>m;
   int x,y;
   while(m--)
   {
      cin>>x>>y;
      gr[x].pb(y);
      gr[y].pb(x);
   }
   start=0;
   lvl[start]=0;
   parent[start]=start;

   _dfs(start);
   init_lca();
   int q;
   cin>>q;
   while(q--)
   {

      cin>>x>>y;
      cout<<lca_query(x,y)<<endl;

   }
   return 0;
}
```

BICONNECTED COMPONENT

```
#define maxnode 1000+7
vector<int>gr[maxnode],disc(maxnode,-1),low(maxnode);
stack<pii>s;
int t=0;

void dfs_bcc(int p,int u)
{
   disc[u]=low[u]=++t;
   int child=0;

   for(int i=0;i<gr[u].size();i++)
   {
      int v=gr[u][i];
      if(v==p) continue;
      else if(disc[v]==-1)
      {
         s.push(pii(u,v));
         child++;
         dfs_bcc(u,v);
         low[u]=min(low[u],low[v]);
```

```
        if((p==-1 && child>1) || disc[u]<=low[v] )
        {
            ///u is a articulation point
            ///so all the edges we visited(tree/backedges) from here
are in a BCC
            cout<<"BCC : \n";
            while(1)
            {
                int x=s.top().fs;
                int y=s.top().sc;
                cout<<x<< " "<<y<<endl;
                s.pop();
                if(x==u && y==v) break;
            }
        }
    }
    else if(low[u]>disc[v]) ///edge in stack
    {
        s.push(pii(u,v));
        low[u]=min(low[u],disc[v]);
    }
  }
}

void BCC(int n)
{
  /*
  Do required clearing and init
  */
  for(int i=0;i<n;i++)
  {
    if(disc[i]==-1)
    {
      dfs_bcc(-1,i);
      if(!s.empty()) cout<<"BCC : "<<endl;
      while(!s.empty())
      {
        cout<<s.top().fs<<" "<<s.top().sc<<endl;
        s.pop();
      }
    }
  }
}

int main()
{
  int n,m;
  cin>>n>>m;
  while(m--)
  {
    int x,y;
    cin>>x>>y;
    gr[x].pb(y);
    gr[y].pb(x);
  }
  BCC(n);
  return 0;
}
```

IS PRIME

```
int is_prime(int n)
{
  int i,root;
  if(n<2) return 0;
  else if(n==2) return 1;
  else if(n%2==0) return 0;
  else{
    root=sqrt(n);
    for(int i=3;i<=root;i+=2){
      if(n%i==0) return 0;
    }
    return 1;
  }
}
```

GCD

```
int gcd_loop(int a,int b)
{
  if(a==0) return b;
  else if(b==0) return a;
  int t;
  for(;a!=0;){
    t=a;
    a=b%a;
    b=t;
  }
  return b;
}

int gcd_rec(int a,int b){ ///a>b
  if(b==0) return a;
  return gcd(b,a%b);
}
```

MOD OF BIG NUMBER

```
int main()
{
  string str="1792798238231230932";
```

```
    int n=str.length();
    int md=1000;
    LL ans=0;
    for(int i=0;i<n;i++){
        ans=(ans*10 + str[i]-'0') % md;
    }
    cout<<ans<<endl;
    return 0;
}
```

BIG MOD

```
long long bg(int base,int pow,int m)
{
    if(pow==0) return 1;
    long long x=bg(base,num/2,m);
    x=(x*x)%m;
    if(pow&1) x=(base*x)%m;
    return x;
}
///recursive
int bigmod ( long long a, int p, int m )
{
    if ( p == 0 )return 1; // If power is 0 ( a ^ 0 ), return 1

    if ( p & 1 ) // If power is odd
    {
        return ( ( a % m ) * ( bigmod ( a, p - 1, m ) ) ) % m;
    }
    else
    {
        long long tmp = bigmod ( a, p / 2, m );
        return ( tmp * tmp ) % m;
    }
}
///iterative
long long bigmod ( long long a, long long p, long long m )
{
    long long res = 1;
    long long x = a;

    while ( p )
    {
        if ( p & 1 ) //p is odd
        {
            res = ( res * x ) % m;
        }
        x = ( x * x ) % m;
        p = p >> 1;
    }
    return res;
}
```

PHI LOOP

```
ll phi(int n)
{
    ll ret=n;
    for(int i=2;i*i<=n;i++)
    {
        if(n%i==0)
        {
            while(n%i==0) n/=i;
            ret-=ret/i;
        }
    }
    if(n>1) ret-=ret/n;
    return ret;
}
```

EULER TRAIL OR CYCLE FINDING

```
struct edge{
    int v,flag;
    edge(){}
    edge(int a,int c){
        v=a;
        flag=c;
    }
};
vector<edge>gr[mxnode];
vector<int>path;

void dfs(int u){
    for(int i=0;i<gr[u].size();i++)
    {
        if(gr[u][i].flag==0){
            gr[u][i].flag=1;
            dfs(gr[u][i].v);
        }
    }
    path.pb(u);
}
int main()
{
    //ios_base::sync_with_stdio(false);
    //freopen("out.txt","w",stdout);
    int t;
    cin>>t;
    for(int test=1;test<=t;test++)
```

```
  {
    int m;
    cin>>m;
    for(int i=0;i<mxnode;i++) gr[i].clear();
    path.clear();
    vector<int>indegree(mxnode),outdegree(mxnode);
    int st=0;
    for(int i=0;i<m;i++){
      int u,v;
      cin>>u>>v;
      st=u;
      gr[u].push_back( edge(v,0) );
      outdegree[u]++;
      indegree[v]++;
    }

   int cnt1=0,cnt2=0,hobena=0,en=-1;
    for(int i=0;i<26 && hobena==0;i++){
      if(indegree[i]-outdegree[i]==1) cnt1++,en=i;
      else if(-indegree[i]+outdegree[i]==1) cnt2++,st=i;
      else if(indegree[i]!=outdegree[i]) hobena=1;
    }
    //cout<<st<<" "<<en<<endl;
    if( (cnt1==1 && cnt2==1 && hobena==0) || (cnt1==0 && cnt2==0
&& hobena==0) )
    {
      //cout<<"Dhuke\n";
      if((cnt1==1 && cnt2==1 && hobena==0))
gr[en].pb(edge(st,0)),m++;
      dfs(st);
      if(1){///fix this part
        cout<<"Case "<<test<<": Yes\n";
        int f=0,last=0;
        if((cnt1==1 && cnt2==1 && hobena==0))
        {
          last=1;
        }
        for(int i=path.size()-1;i>=last;i--){
          if(f) cout<<" ";
          f=1;
          cout<<path[i];
        }
        cout<<"\n";
      }
      else{
        cout<<"Case "<<test<<": No\n";
      }
    }
    else cout<<"Case "<<test<<": No\n";
  }
  return 0;
}
```

DINIC MAXFLOW

```
#define mxnode 1000

struct edge{
  int v,flow,cap,index;///index of reverse edge in gr[v]
  edge() {}
  edge(int a,int b,int c,int d)
  {
    v=a;flow=b;cap=c;index=d;
  }
};

vector<edge>gr[mxnode];
vector<int>level,start;

bool bfs(int s,int t,int n)
{
  level=vector<int>(n,-1);
  level[s]=0;
  queue<int>q;
  q.push(s);
  while(!q.empty())
  {
    int u=q.front();
    q.pop();
    for(int i=0;i<gr[u].size();i++)
    {
      edge e=gr[u][i];
      if(level[e.v]==-1 && e.flow<e.cap)
      {
        level[e.v]=level[u]+1;
        q.push(e.v);
      }
    }
  }
  return (level[t]!=-1);
}
int dfs(int u,int t,int f)
{
  if(u==t) return f;
  for(;start[u]<gr[u].size();start[u]++)
  {
    edge &e=gr[u][start[u]];
    if(level[e.v]==level[u]+1 && e.flow<e.cap)
    {
      int cur_flow=min(f,e.cap-e.flow);
```

```
            int temp_flow=dfs(e.v,t,cur_flow);
            if(temp_flow>0)
            {
               e.flow+=temp_flow;
               gr[e.v][e.index].flow -= temp_flow;
               return temp_flow;
            }
         }
      }
      return 0;
}

int Dinic(int s,int t,int n)
{

   if(s==t) return 0;
   int total=0;
   while(bfs(s,t,n)==true)
   {
      start=vector<int>(n);
      while(int flow=dfs(s,t,INT_MAX))
      {
         total+=flow;
      }
   }
   return total;
}

int main()
{
   int n,m;
   cin>>n>>m;
   while(m--)
   {
      int x,y,c;
      cin>>x>>y>>c;
      edge a(y,0,c,(int)gr[y].size());
      edge b(x,0,0,(int)gr[x].size());
      gr[x].pb(a);
      gr[y].pb(b);
   }
   int s,t;
   cin>>s>>t;
   cout<<Dinic(s,t,n)<<endl;
}
```

BPM

```
// A C++ program to find maximal Bipartite matching.
///has to be directed graph

#include <iostream>
#include <string.h>
using namespace std;

// M is number of applicants and N is number of jobs
#define M 6
#define N 6

// A DFS based recursive function that returns true if a
// matching for vertex u is possible
bool bpm(bool bpGraph[M][N], int u, bool seen[], int matchR[])
{
   // Try every job one by one
   for (int v = 0; v < N; v++)
   {
            // If applicant u is interested in job v and v is
            // not visited
            if (bpGraph[u][v] && !seen[v])
            {
                     seen[v] = true; // Mark v as visited

                     // If job 'v' is not assigned to an applicant OR
                     // previously assigned applicant for job v
(which is matchR[v])
                     // has an alternate job available.
                     // Since v is marked as visited in the above line,
matchR[v]
                     // in the following recursive call will not get job
'v' again
                     if (matchR[v] < 0 || bpm(bpGraph, matchR[v],
seen, matchR))
                     {
                              matchR[v] = u;
                              return true;
                     }
            }
   }
   return false;
}

// Returns maximum number of matching from M to N
int maxBPM(bool bpGraph[M][N])
{
   // An array to keep track of the applicants assigned to
   // jobs. The value of matchR[i] is the applicant number
   // assigned to job i, the value -1 indicates nobody is
   // assigned.
   int matchR[N];

   // Initially all jobs are available
```

```
    memset(matchR, -1, sizeof(matchR));

    int result = 0; // Count of jobs assigned to applicants
    for (int u = 0; u < M; u++)
    {
            // Mark all jobs as not seen for next applicant.
            bool seen[N];
            memset(seen, 0, sizeof(seen));

            // Find if the applicant 'u' can get a job
            if (bpm(bpGraph, u, seen, matchR))
                    result++;
    }
    return result;
}

// Driver program to test above functions
int main()
{
    // Let us create a bpGraph shown in the above example
    bool bpGraph[M][N] = { {0, 1, 1, 0, 0, 0},
                                                    {1, 0, 0, 1, 0, 0},
                                                    {0, 0, 1, 0, 0, 0},
                                                    {0, 0, 1, 1, 0, 0},
                                                    {0, 0, 0, 0, 0, 0},
                                                    {0, 0, 0, 0, 0, 1}
                                            };

    cout << "Maximum number of applicants that can get job is "
            << maxBPM(bpGraph);

    return 0;
}
```

NQUEEN

```
/* C/C++ program to solve N Queen Problem using
  backtracking */
#define N 4
#include<stdio.h>
#include<stdbool.h>

/* A utility function to print solution */
void printSolution(int board[N][N])
{
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
            printf(" %d ", board[i][j]);
        printf("\n");
    }
}

/* A utility function to check if a queen can
  be placed on board[row][col]. Note that this
  function is called when "col" queens are
  already placed in columns from 0 to col -1.
  So we need to check only left side for
  attacking queens */
bool isSafe(int board[N][N], int row, int col)
{
    int i, j;

    /* Check this row on left side */
    for (i = 0; i < col; i++)
        if (board[row][i])
            return false;

    /* Check upper diagonal on left side */
    for (i=row, j=col; i>=0 && j>=0; i--, j--)
        if (board[i][j])
            return false;

    /* Check lower diagonal on left side */
    for (i=row, j=col; j>=0 && i<N; i++, j--)
        if (board[i][j])
            return false;

    return true;
}

/* A recursive utility function to solve N
  Queen problem */
bool solveNQUtil(int board[N][N], int col)
{
    /* base case: If all queens are placed
      then return true */
    if (col >= N)
        return true;

    /* Consider this column and try placing
      this queen in all rows one by one */
    for (int i = 0; i < N; i++)
    {
        /* Check if the queen can be placed on
          board[i][col] */
        if ( isSafe(board, i, col) )
        {
            /* Place this queen in board[i][col] */
            board[i][col] = 1;
```

```
        /* recur to place rest of the queens */
        if ( solveNQUtil(board, col + 1) )
           return true;

        /* If placing queen in board[i][col]
          doesn't lead to a solution, then
          remove queen from board[i][col] */
        board[i][col] = 0; // BACKTRACK
      }
   }

    /* If the queen cannot be placed in any row in
       this colum col  then return false */
   return false;
}

/* This function solves the N Queen problem using
   Backtracking. It mainly uses solveNQUtil() to
   solve the problem. It returns false if queens
   cannot be placed, otherwise, return true and
   prints placement of queens in the form of 1s.
   Please note that there may be more than one
   solutions, this function prints one  of the
   feasible solutions.*/
bool solveNQ()
{
    int board[N][N] = { {0, 0, 0, 0},
       {0, 0, 0, 0},
       {0, 0, 0, 0},
       {0, 0, 0, 0}
    };

    if ( solveNQUtil(board, 0) == false )
    {
      printf("Solution does not exist");
      return false;
    }

    printSolution(board);
    return true;
}

// driver program to test above function
int main()
{
    solveNQ();
    return 0;
}
```

BITWISE NQUEEN

```
int n;
int solve(int row,int minDiag, int majDiag, int col){
   if(row==n) return 1;
   int res=0;
   int conflict=minDiag|majDiag|col;
   for(int queenPos=1;queenPos < 1<<n ;queenPos<<=1){
      if(!(conflict & queenPos))
      {
         int nextMinDiag=(minDiag|queenPos)>>1;
         int nextMajDiag=(majDiag|queenPos)<<1;
         int nextCol=col|queenPos;
         res+=solve(row+1,nextMinDiag,nextMajDiag,nextCol);
      }
   }
   return res;
}

int main()
{
   cin>>n;
   cout<<solve(0,0,0,0)<<endl;
   return 0;
}
```

LIS N^2

```
#define mx 1000
int n=7;
int value[]={-100000,5,0,9,2,7,3,4};
int dp[mx],dir[mx];
int longest(int u)
{
   if(dp[u]!=-1) return dp[u];
   int maxi=0;
   for(int v=u+1;v<=n;v++) //১ম শর্ত,v>u
   {
            if(value[v]>value[u]) //২য় শর্ত, value[v]>value[u]
            {
                     if(longest(v)>maxi) //সর্বোচ্চ মানটা নিবো
                     {
                              maxi=longest(v);
          dir[u]=v; ///konta te gele LIS hoy from u

                     }
            }
   }
   dp[u]=1+maxi; //১ যোগ হবে কারণ u নম্বর নোডটাও পাথের মধ্যে আছে
   return dp[u];
```

```
}
void solution(int start) ///printing the solution
{
   while(dir[start]!=-1)
   {
            printf("index %d value %d\n",start,value[start]);
            start=dir[start];
   }
}
int main()
{
   ///code wont run, just for reference
   memset(dp,-1,sizeof dp);
   memset(dir,-1,sizeof dir);
   int LIS=0,start;
   for(int i=1;i<=n;i++)
   {
            printf("longest path from: %d\n",longest(i));
            if(longest(i)>LIS)
            {
                     LIS=longest(i);
                     start=i;
            }
   }
   printf("LIS = %d Starting point %d\n",LIS,start);

   return 0;
}
```

LIS nlogn

```
int lis(vector<int>v)
{
   vector<int>temp;
   for(int i=0;i<v.size();i++)
   {
      int x=v[i];
      ///strictly increasing er khetre LOWER, naile UPPER bound
      int in=upper_bound(temp.begin(),temp.end(),x)-temp.begin();
      if(in==temp.size()) temp.pb(x);
      else temp[in]=min(x,temp[in]);
   }
   return temp.size();
}
```

BIG INTEGER

```
// header files
#include <cstdio>
#include <string>
#include <algorithm>
#include <iostream>
using namespace std;
struct Bigint
{
   // representations and structures
   string a; // to store the digits
   int sign; // sign = -1 for negative numbers, sign = 1 otherwise
   // constructors
   Bigint() {} // default constructor
   Bigint( string b )
   {
      (*this) = b;    // constructor for string
   }
   // some helpful methods
   int size()   // returns number of digits
   {
      return a.size();
   }
   Bigint inverseSign()   // changes the sign
   {
      sign *= -1;
      return (*this);
   }
   Bigint normalize( int newSign )   // removes leading 0, fixes sign
   {
      for( int i = a.size() - 1; i > 0 && a[i] == '0'; i-- )
         a.erase(a.begin() + i);
      sign = ( a.size() == 1 && a[0] == '0' ) ? 1 : newSign;
      return (*this);
   }
   // assignment operator
   void operator = ( string b )   // assigns a string to Bigint
   {
      a = b[0] == '-' ? b.substr(1) : b;
      reverse( a.begin(), a.end() );
      this->normalize( b[0] == '-' ? -1 : 1 );
   }
   // conditional operators
   bool operator < ( const Bigint &b ) const   // less than operator
   {
      if( sign != b.sign )
         return sign < b.sign;
      if( a.size() != b.a.size() )
         return sign == 1 ? a.size() < b.a.size() : a.size() > b.a.size();
      for( int i = a.size() - 1; i >= 0; i-- )
         if( a[i] != b.a[i] )
            return sign == 1 ? a[i] < b.a[i] : a[i] > b.a[i];
      return false;
   }
```

```
    bool operator == ( const Bigint &b ) const   // operator for
equality
    {
        return a == b.a && sign == b.sign;
    }
    // mathematical operators
    Bigint operator + ( Bigint b )   // addition operator overloading
    {
        if( sign != b.sign )
            return (*this) - b.inverseSign();
        Bigint c;
        for(int i = 0, carry = 0; i<a.size() || i<b.size() || carry; i++ )
        {
            carry+=(i<a.size() ? a[i]-48 : 0)+(i<b.a.size() ? b.a[i]-48 : 0);
            c.a += (carry % 10 + 48);
            carry /= 10;
        }
        return c.normalize(sign);
    }
    Bigint operator - ( Bigint b )   // subtraction operator
overloading
    {
        if( sign != b.sign )
            return (*this) + b.inverseSign();
        int s = sign;
        sign = b.sign = 1;
        if( (*this) < b )
            return ((b - (*this)).inverseSign()).normalize(-s);
        Bigint c;
        for( int i = 0, borrow = 0; i < a.size(); i++ )
        {
            borrow = a[i] - borrow - (i < b.size() ? b.a[i] : 48);
            c.a += borrow >= 0 ? borrow + 48 : borrow + 58;
            borrow = borrow >= 0 ? 0 : 1;
        }
        return c.normalize(s);
    }
    Bigint operator * ( Bigint b )   // multiplication operator
overloading
    {
        Bigint c("0");
        for( int i = 0, k = a[i] - 48; i < a.size(); i++, k = a[i] - 48 )
        {
            while(k--)
                c = c + b; // ith digit is k, so, we add k times
            b.a.insert(b.a.begin(), '0'); // multiplied by 10
        }
        return c.normalize(sign * b.sign);
    }
    Bigint operator / ( Bigint b )   // division operator overloading
    {
        if( b.size() == 1 && b.a[0] == '0' )
            b.a[0] /= ( b.a[0] - 48 );
        Bigint c("0"), d;
        for( int j = 0; j < a.size(); j++ )
            d.a += "0";
        int dSign = sign * b.sign;
        b.sign = 1;
        for( int i = a.size() - 1; i >= 0; i-- )
        {
            c.a.insert( c.a.begin(), '0');

            c = c + a.substr( i, 1 );

            while( !( c < b ) )
                c = c - b, d.a[i]++;

        }
        return d.normalize(dSign);

    }
    Bigint operator % ( Bigint b )   // modulo operator overloading
    {
        if( b.size() == 1 && b.a[0] == '0' )
            b.a[0] /= ( b.a[0] - 48 );
        Bigint c("0");
        b.sign = 1;
        for( int i = a.size() - 1; i >= 0; i-- )
        {
            c.a.insert( c.a.begin(), '0');
            c = c + a.substr( i, 1 );
            while( !( c < b ) )
                c = c - b;
        }
        return c.normalize(sign);
    }
// output method
    void print()
    {
        if( sign == -1 )
            putchar('-');
        for( int i = a.size() - 1; i >= 0; i-- )
            putchar(a[i]);
    }

};

int main()
{
```

```
    Bigint a, b, c; // declared some Bigint variables
    ////////////////////////
    // taking Bigint input //
    ////////////////////////
    string input; // string to take input
    cin >> input; // take the Big integer as string
    a = input; // assign the string to Bigint a
    cin >> input; // take the Big integer as string
    b = input; // assign the string to Bigint b
    ////////////////////////////////
    // Using mathematical operators //
    ////////////////////////////////
    c = a + b; // adding a and b
    c.print(); // printing the Bigint
    puts(""); // newline
    c = a - b; // subtracting b from a
    c.print(); // printing the Bigint
    puts(""); // newline
    c = a * b; // multiplying a and b
    c.print(); // printing the Bigint
    puts(""); // newline
    c = a / b; // dividing a by b
    c.print(); // printing the Bigint
    puts(""); // newline
    c = a % b; // a modulo b
    c.print(); // printing the Bigint
    puts(""); // newline
    ////////////////////////////////
    // Using conditional operators //
    ////////////////////////////////
    if( a == b )
        puts("equal"); // checking equality
    else
        puts("not equal");
    if( a < b )
        puts("a is smaller than b"); // checking less than operator
    return 0;
}
```

LOGN PRIME FACTORIZATION

```
// C++ program to find prime factorization of a
// number n in O(Log n) time with precomputation
// allowed.
#include "bits/stdc++.h"
using namespace std;

#define MAXN   100001

// stores smallest prime factor for every number
int spf[MAXN];

// Calculating SPF (Smallest Prime Factor) for every
// number till MAXN.
// Time Complexity : O(nloglogn)
void sieve()
{
    spf[1] = 1;
    for (int i=2; i<MAXN; i++)

        // marking smallest prime factor for every
        // number to be itself.
        spf[i] = i;

    // separately marking spf for every even
    // number as 2
    for (int i=4; i<MAXN; i+=2)
        spf[i] = 2;

    for (int i=3; i*i<MAXN; i++)
    {
        // checking if i is prime
        if (spf[i] == i)
        {
            // marking SPF for all numbers divisible by i
            for (int j=i*i; j<MAXN; j+=i)

                // marking spf[j] if it is not
                // previously marked
                if (spf[j]==j)
                    spf[j] = i;
        }
    }
}

// A O(log n) function returning primefactorization
// by dividing by smallest prime factor at every step
vector<int> getFactorization(int x)
{
    vector<int> ret;
    while (x != 1)
    {
        ret.push_back(spf[x]);
        x = x / spf[x];
    }
    return ret;
}

// driver program for above function
int main(int argc, char const *argv[])
```

```
{
   // precalculating Smallest Prime Factor
   sieve();
   int x = 12246;
   cout << "prime factorization for " << x << " : ";

   // calling getFactorization function
   vector <int> p = getFactorization(x);

   for (int i=0; i<p.size(); i++)
      cout << p[i] << " ";
   cout << endl;
   return 0;
}
```

BITMASK INCLUSION EXCLUSION

```
///number of numbers from 1 - n that are not divisible my any of the m given numbers
int main()
{
   ios_base::sync_with_stdio(false);
   int t,test=1;
   cin>>t;
   while(t--)
   {
      ll n,m;
      cin>>n>>m;
      vector<int>p(m);
      for(int i=0;i<m;i++)
      {
         cin>>p[i];
      }
      ll lim=(1<<m);
      //cout<<"lim = "<<lim<<endl;
      ll ans=0;
      for(int i=1;i<lim;i++)
      {
         ll lcm=0;
         int cnt=0,bit=0;
         int temp=i;
         while(temp)
         {
            if(temp&1){
               cnt++;
               if(lcm==0) lcm=p[bit];
               else lcm=(lcm*p[bit]/(__gcd(lcm,1ll*p[bit])));
            }
            bit++;
            temp/=2;
         }
         if(cnt&1) ans+=n/lcm;
         else ans-=n/lcm;
         //cout<<"sum = "<<n/lcm<<endl;
      }
      cout<<"Case "<<test++<<": "<<n-ans<<endl;
   }
   return 0;
}
```

DIOPHANTINE IN RANGE

```
pll extendedEuclid(ll a,ll b)
{
   if(b==0) return pll(1,0);
   else
   {
      pll d=extendedEuclid(b,a%b);
      return pll(d.sc,d.fs-d.sc*(a/b));
   }
}
ll diophantine(ll a,ll b,ll c, ll x1,ll x2,ll y1,ll y2)
{
   if(a<0)///dividing inequality with negative number reverses the signs(< & >)
   {
      a=-a;
      x1=-x1;
      x2=-x2;
      swap(x1,x2);
   }
   if(b<0)
   {
      b=-b;
      y1=-y1;
      y2=-y2;
      swap(y1,y2);
   }
   ll d=__gcd(a,b);
   if(d==0){
      if(c==0) return (x2-x1+1)*(y2-y1+1);
      return 0;
   }
   if(c%d!=0) return 0;
   if(b==0){
      if(c/a>=x1 && c/a<=x2) return y2-y1+1;
      return 0;
   }
   if(a==0){
      if(c/b>=y1 && c/b<=y2) return x2-x1+1;
```

```
    return 0;
  }
  a/=d;
  b/=d;
  c/=d;
  pll p=extendedEuclid(a,b);
  p.fs*=c;
  p.sc*=c;
  ///x and y high, low
  ///have to write floor explicitly
  ///else negative number floor not done properly
  ll xl,xh,yl,yh;
  xh=floor((p.fs-x1)/ (1.0*b) );
  xl=ceil((p.fs-x2)/ (1.0*b) );
  yh=floor((y2-p.sc)/ (1.0*a) );
  yl=ceil((y1-p.sc)/ (1.0*a) );
  xh=min(xh,yh);
  xl=max(xl,yl);
  return max((xh-xl+1),0ll);
}
int main()
{
  ios_base::sync_with_stdio(false);
  ll a,b,c,x1,y1,x2,y2; ///equation of the form ax+by+c=0;
  cin>>a>>b>>c>>x1>>x2>>y1>>y2;
  c=-c; ///turned into ax+by=c
  cout<<"Case "<<test++<<": "<<diophantine(a,b,c,x1,x2,y1,y2)<<endl;
  return 0;
}
```

NCR FOR BIG NUMBERS (lucas + CRT)

```
ll bg(ll base,ll power,ll mod)
{
  if(power==0) return 1ll;
  ll x=bg(base,power/2,mod);
  x=(x*x)%mod;
  if(power&1) x=(x*base)%mod;
  return x;
}
ll ncr(ll n,ll r,ll p)
{
  if(n<r) return 0ll;
  if(n==r) return 1ll;
  r=min(r,n-r);
  ll nu=1,de=1;
  for(ll i=n-r+1;i<=n;i++) nu=(nu*i)%p;
  for(ll i=1;i<=r;i++) de=(de*i)%p;
  return ((nu*bg(de,p-2,p))%p + p)%p; ///only for primes
}
ll lucas(ll n,ll r,ll p)///nCr % p //for p is prime or prime power
{
  if(r==0) return 1;
  return (ncr(n%p,r%p,p)*lucas(n/p,r/p,p))%p;
}
ll CRT(vector<pll>v,ll M)
{
  ll ret=0;
  for(int i=0;i<v.size();i++)
  {
    ll ai=v[i].fs;
    ll p=v[i].sc;
    ll Mi=M/p;
    ret+=ai*Mi*bg(Mi,p-2,p);
    ret%=M;
  }
  return ret;
}
int main()
{
  ios_base::sync_with_stdio(false);
  int t,test=1;
  cin>>t;
  while(t--)
  {
    vector<pll>v;
    ll n,r,m,M;
    cin>>n>>r>>m;
    M=m;
    for(ll i=2;i<=50;i++)
    {
      if(m%i==0)
      {
        m/=i; ///sqaure-free
        ll p=lucas(n,r,i);
        v.pb(pll(p,i));
      }
    }
    ll ans=CRT(v,M);
    cout<<ans<<endl;
  }
  return 0;
}
```

QTREE HLD

```
/*
You are given a tree (an acyclic undirected connected graph) with N nodes, and edges numbered 1, 2, 3...N-1.
```

precode@tanny411
aysha.kamal7@gmail.com

We will ask you to perfrom some instructions of the following form:

- CHANGE i ti : change the cost of the i-th edge to ti
- QUERY a b : ask for the maximum edge cost on the path from node a to node b

*/

```
#define node 10003
#define ln 14

vector<pair<pii,int> >gr[node]; ///v,w,indexOfEdge
int table[node][ln],level[node],siz[node],indexx[node];
int
chainNo[node],chainHead[node],chain,posBase[node],ptr,base[nod
e];
int tree[4*node];

void dfs(int u)
{
   siz[u]=1;
   for(int i=0; i<gr[u].size(); i++)
   {
      int v=gr[u][i].fs.fs;
      int w=gr[u][i].fs.sc;
      int in=gr[u][i].sc;
      if(level[v]==-1)
      {
         table[v][0]=u;
         level[v]=level[u]+1;
         indexx[in]=v;
         dfs(v);
         siz[u]+=siz[v];
      }
   }
}

int lca_query(int x,int y)
{
   if(level[x]<level[y]) swap(x,y);
   int lg;
   for(lg=1; (1<<lg)<=level[x]; lg++);
   lg--;
   for(int i=lg; i>=0; i--)
   {
      if(level[x]-(1<<i)>=level[y]) x=table[x][i];
   }
   if(x==y) return x;
   for(int i=lg; i>=0; i--)
   {
      if(table[x][i]!=table[y][i])
      {
         x=table[x][i];
         y=table[y][i];
      }
   }
   return table[x][0];
}

void hld(int u,int prev,int cost)
{
   if(chainHead[chain]==-1)
   {
      chainHead[chain]=u;
   }
   ptr++;
   chainNo[u]=chain;
   posBase[u]=ptr;
   base[ptr]=cost;
   int bc=-1,mx=-1,cs=-1;
   for(int i=0; i<gr[u].size(); i++)
   {
      int v=gr[u][i].fs.fs;
      //cout<<"aysha  "<<u<<" "<<v<<endl;
      if(v!=prev && siz[v]>mx) mx=siz[v],bc=v,cs=gr[u][i].fs.sc;
   }
   if(bc!=-1) hld(bc,u,cs);
   for(int i=0; i<gr[u].size(); i++)
   {
      int v=gr[u][i].fs.fs;
      if(v!=prev && v!=bc)
      {
         //cout<<"gese"<<u<<" "<<v<<" "<<bc<<endl;
         chain++;
         hld(v,u,gr[u][i].fs.sc);
      }
   }
   return;
}

void build(int nodes,int b,int e)
{
   if(b==e) tree[nodes]=base[b];
   else
   {
      int m=(b+e)/2;
      build(nodes*2,b,m);
      build(nodes*2+1,m+1,e);
      tree[nodes]=max(tree[nodes*2],tree[nodes*2+1]);
   }
}

int seg_query(int nodes,int b,int e,int i,int j)
```

```
{
   if(b>=i && j>=e) return tree[nodes];
   else if(i>e || j<b) return -1;
   int m=(b+e)/2;
   return max( seg_query(nodes*2,b,m,i,j),
seg_query(nodes*2+1,m+1,e,i,j) );
}

void update(int nodes,int b,int e,int i,int val)
{
   if(b==e && e==i) {tree[nodes]=val;return;}
   int m=(b+e)/2;
   if(i<=m) update(nodes*2,b,m,i,val);
   else update(nodes*2+1,m+1,e,i,val);
   tree[nodes]=max(tree[nodes*2],tree[nodes*2+1]);
   return;
}

int queryUp(int x,int v)///v=lca
{
   if(x==v) return 0;
   int mx=-1;
   while(1)
   {
      if(chainNo[x]==chainNo[v])
      {
         if(x==v) return mx;
         return max(mx,seg_query(1,1,ptr,posBase[v]+1,posBase[x]));
      }
      mx=max(mx,seg_query(1,1,ptr,posBase[ chainHead[ chainNo[x]
] ],posBase[x]));
      x=table[chainHead[ chainNo[x] ] ][0];
   }
   return mx;
}


int query(int x,int y)
{
   int v= lca_query(x,y);
   return max(queryUp(x,v),queryUp(y,v));
}


int main()
{
   int t;
   scanf("%d",&t);
   while(t--)
   {
      int n;
      scanf("%d",&n);

      for(int i=1; i<=n; i++)
      {
         gr[i].clear();
         chainHead[i]=-1;
         level[i]=-1;
         for(int j=0; j<ln; j++) table[i][j]=-1;
      }
      for(int i=1; i<n; i++)
      {
         int x,y,w;
         scanf("%d%d%d",&x,&y,&w);
         gr[x].pb({{y,w},i});
         gr[y].pb({{x,w},i});
      }

      /*for(int i=1;i<=n;i++)
      {
         cout<<i<<":";
         for(int j=0;j<gr[i].size();j++)
         {
            cout<<" "<<gr[i][j].fs.fs;
         }
         cout<<endl;
      }*/

      level[1]=0;
      dfs(1);

      for(int j=1; j<ln ; j++)
      {
         for(int i=1; i<=n; i++)
         {
            if(table[i][j-1]!=-1)
            {
               table[i][j]=table[table[i][j-1]][j-1];
            }
         }
      }

      ptr=-1;
      chain=1;
      hld(1,0,0);
      //ptr--;
      //cout<<"ptr = "<<ptr<<endl;
      //for(int i=0;i<6;i++) cout<<" "<<base[i]; cout<<endl;
      //for(int i=1;i<=n;i++) cout<<" "<<posBase[i]; cout<<endl;
      //for(int i=1;i<=n;i++) cout<<" "<<chainNo[i]; cout<<endl;
```

```
    //cout<<"no of chains "<<chain<<endl;
    build(1,1,ptr);

    char s[10];
    while(scanf("%s",s))
    {
      if(strcmp(s,"DONE")==0) break;
      if(strcmp(s,"QUERY")==0)
      {
        int x,y;
        scanf("%d%d",&x,&y);
        printf("%d\n",query(x,y));
      }
      else
      {
        int x,val;
        scanf("%d%d",&x,&val);
        x=posBase[ indexx[x] ];
        update(1,1,ptr,x,val);
      }
    }
  }
  return 0;
}
```

HLD SUM OF PATH, NODE VALUE UPDATE

```
vector<int>v(30002),gr[30002];
int level[30002],table[30002][17],tree[4*30000],sz[30002];
int n;
int
chainHead[30002],chain,ptr,chainNo[30002],base[30002],baseNo[
30002];

void dfs(int u)
{
  sz[u]=1;
  for(int i=0;i<gr[u].size();i++)
  {
    int v=gr[u][i];
    if(level[v]==-1)
    {
      level[v]=level[u]+1;
      table[v][0]=u;
      dfs(v);
      sz[u]+=sz[v];
    }
  }
}
void lca_pre()
{
  for(int j=1;j<ln;j++)
  {
    for(int i=0;i<n;i++){
      if(table[i][j-1]!=-1) table[i][j]=table[table[i][j-1]][j-1];
    }
  }
}
void hld(int u,int prev){
  if(chainHead[chain]==-1) chainHead[chain]=u;
  chainNo[u]=chain;
  base[ptr]=v[u];
  baseNo[u]=ptr++;
  int mx=-1,bc=-1;
  for(int i=0;i<gr[u].size();i++)
  {
    int v=gr[u][i];
    if(v!=prev && sz[v]>mx) mx=sz[v],bc=v;
  }
  if(bc!=-1) hld(bc,u);
  for(int i=0;i<gr[u].size();i++)
  {
    int v=gr[u][i];
    if(v==prev ||  v==bc) continue;
    chain++;
    hld(v,u);
  }
}
int lca_query(int x,int y)
{
  if(level[x]<level[y]) swap(x,y);
  int lg=1;
  for(;(1<<lg)<=level[x];lg++);
  lg--;
  for(int i=lg;i>=0;i--)
  {
    if(level[x]-(1<<i)>=level[y]) x=table[x][i];
  }
  if(x==y) return x;
  for(int i=lg;i>=0;i--)
  {
    if(table[x][i]!=table[y][i]) x=table[x][i],y=table[y][i];
  }
  return table[x][0];
}
int seg_query(int node,int b,int e,int i,int j)
{
  if(b>=i && j>=e) return tree[node];
  if(j<b || i>e) return 0;
```

```
    int m=(b+e)/2;
    return seg_query(node*2,b,m,i,j)+seg_query(node*2+1,m+1,e,i,j);
}
int hld_query(int x,int v)
{
    int sum=0;
    while(1){
        if(chainNo[x]==chainNo[v]){
            sum+=seg_query(1,0,ptr,baseNo[v],baseNo[x]);
            return sum;
        }

sum+=seg_query(1,0,ptr,baseNo[chainHead[chainNo[x]]],baseNo[x]
);
        x=table[chainHead[chainNo[x]]][0];
    }
}
void build(int node,int b,int e)
{
    if(b==e) {tree[node]=base[b]; return;}
    int m=(b+e)/2;
    build(node*2,b,m);
    build(node*2+1,m+1,e);
    tree[node]=tree[node*2]+tree[node*2+1];
}
int query(int x,int y)
{
    int v=lca_query(x,y);
    return hld_query(x,v)+hld_query(y,v)-base[baseNo[v]];
}
void update(int node,int b,int e,int i,int val)
{
    if(b==e && b==i) {tree[node]=val; return;}
    int m=(b+e)/2;
    if(i<=m) update(node*2,b,m,i,val);
    else update(node*2+1,m+1,e,i,val);
    tree[node]=tree[node*2]+tree[node*2+1];
    return;
}
int main()
{
    int t,test=1;
    scanf("%d",&t);
    while(t--)
    {
        scanf("%d",&n);

        for(int i=0;i<n;i++) {gr[i].clear(),level[i]=-1,chainHead[i]=-1;
for(int j=0;j<ln;j++) table[i][j]=-1;}
        for(int i=0;i<n;i++) cin>>v[i];
        for(int i=1;i<n;i++)
        {
            int x,y;
            scanf("%d%d",&x,&y);
            gr[x].pb(y);
            gr[y].pb(x);
        }
        level[0]=0;
        dfs(0);
        lca_pre();
        chain=1;
        ptr=0;
        hld(0,-1);
        ptr--;
        build(1,0,ptr);

        int q;
        scanf("%d",&q);
        printf("Case %d:\n",test++);
        while(q--)
        {
            int type,node,to;
            scanf("%d%d%d",&type,&node,&to);
            if(type){
                update(1,0,ptr,baseNo[node],to);
                base[ baseNo[node] ]=to;
            }
            else{
                printf("%d\n",query(node,to));
            }
        }
    }
    return 0;
}
```

MAXIMUM SUBARRAY SUM IN RANGE WITH UPDATE gss3

```
vector<ll>v;
struct data
{
    ll sum,mxsum,prefixsum,suffixsum;
} tree[4*50000];
data mergeit(data l,data r)
{
    data n;
    n.sum=l.sum+r.sum;
    n.prefixsum=max(l.prefixsum,r.prefixsum+l.sum);
    n.suffixsum=max(r.suffixsum,l.suffixsum+r.sum);
```

```
    n.mxsum=max(max(l.mxsum,r.mxsum),l.suffixsum+r.prefixsum);
    return n;
}
void build(int node,int b,int e)
{
    if(b==e)
    {

tree[node].sum=tree[node].mxsum=tree[node].prefixsum=tree[nod
e].suffixsum=v[e];
    }
    else
    {
        int m=(b+e)/2;
        build(node*2,b,m);
        build(node*2+1,m+1,e);
        tree[node]=mergeit(tree[node*2],tree[node*2+1]);
    }
}
data query(int node,int b,int e,int i,int j)
{
    if(i<=b && e<=j) return tree[node];
    else if(e<i || j<b)
    {
        data n;
        n.sum=n.mxsum=n.prefixsum=n.suffixsum=INT_MIN;
        return n;
    }
    else
    {
        int m=(b+e)/2;
        return mergeit(query(node*2,b,m,i,j),query(node*2+1,m+1,e,i,j));
    }
}
void update(int node,int b,int e,int i,int val)
{
    if(b==e && b==i)
    {

tree[node].sum=tree[node].mxsum=tree[node].prefixsum=tree[nod
e].suffixsum=val;
    }
    else if(i<b || e<i) return;
    else
    {
        int m=(b+e)/2;
        update(node*2,b,m,i,val);
        update(node*2+1,m+1,e,i,val);
        tree[node]=mergeit(tree[node*2],tree[node*2+1]);
    }
}
int main()
{
    int n,q;
    scanf("%d",&n);
    for(int i=0; i<n; i++)
    {
        int x;
        scanf("%d",&x);
        v.pb(x);
    }
    build(1,0,n-1);
    scanf("%d",&q);
    for(int i=0;i<q;i++)
    {
        int t;
        scanf("%d",&t);
        int x,y;
        scanf("%d%d",&x,&y);
        if(t)
        {
            printf("%lld\n",query(1,0,n-1,x-1,y-1).mxsum);
        }
        else update(1,0,n-1,x-1,(ll)y);
    }
    return 0;
}
```

MAXIMUM SUBARRAY SUM IN RANGE AVOIDING DUPLICATES, EMPTY ALLOWED gss2

```
vector<ll>v;
ll cur[nd],lazy[nd],best[nd],bestLazy[nd];
bool cmp(ppi a,ppi b)
{
    return a.fs.sc<b.fs.sc;
}
void propagate(int node)
{
    for(int child=2*node;child<=node*2+1;child++){

bestLazy[child]=max(bestLazy[child],lazy[child]+bestLazy[node]);
        lazy[child]+=lazy[node];
        best[child]=max(best[child],cur[child]+bestLazy[node]);
        cur[child]+=lazy[node];
    }
    lazy[node]=bestLazy[node]=0ll;
}
void update(int node,int b,int e,int i,int j,int val)
```

```
{
   if(j<b || i>e) return;
   if(i<=b && e<=j){
      lazy[node]+=val;
      cur[node]+=val;
      bestLazy[node]=max(bestLazy[node],lazy[node]);///for
children
      best[node]=max(best[node],cur[node]);///for self
      return;
   }
   ///self work is already done, so we can return without
propagating
   ///propagate
   if(b!=e){
      propagate(node);
   }
   int m=(b+e)/2;
   update(node*2,b,m,i,j,val);
   update(node*2+1,m+1,e,i,j,val);
   ///merge
   cur[node]=max(cur[node*2],cur[node*2+1]);
   //best[node]=max(best[node],cur[node]);
   //both correct, use whichever makes more sense
   best[node]=max(best[node*2],best[node*2+1]);
}
ll query(int node,int b,int e,int i,int j)
{
   if(i<=b && e<=j) return best[node];
   else if(j<b || i>e) return 0ll;
   else{
      propagate(node);
      int m=(b+e)/2;
      return max(query(node*2,b,m,i,j),query(node*2+1,m+1,e,i,j));
   }
}
int main()
{
   int n;
   scanf("%d",&n);
   v=vector<ll>(n);
   for(int i=0;i<n;i++) scanf("%lld",&v[i]);
   int nq;
   scanf("%d",&nq);
   vector<ppi>q(nq);
   ll ans[nq];
   for(int i=0;i<nq;i++) {scanf("%d%d",&q[i].fs.fs,&q[i].fs.sc); q[i].sc=i;}
   sort(q.begin(),q.end(),cmp);
   map<int,int>mp;///using array will be better O(1)
   for(int i=0,start=0;i<nq;i++)
   {
      int x=q[i].fs.fs;
      int y=q[i].fs.sc;
      x--;
      y--;
      while(start<=y){
         //cout<<start<<" ";
         update(1,0,n-1,mp[v[start]],start,v[start]);
         mp[v[start]]=start+1;
         start++;
      }
      ans[q[i].sc]=query(1,0,n-1,x,y);
   }
   for(int i=0;i<nq;i++) printf("%lld\n",ans[i]);
   return 0;
}
```

MAXIMUM SUM IN RANGE i,j WHERE RANGE FOR i AND j GIVEN gss5

```
vector<int>v;
struct data{
   int sum,maxsum,prefixsum,suffixsum;
   data(){}
   data(int a):sum(a),maxsum(a),prefixsum(a),suffixsum(a){}
   data(int a,int b,int c,int
d):sum(a),maxsum(b),prefixsum(c),suffixsum(d){}
};
data tree[4*200007];
data milao(data l,data r)
{
   data ret;
   ret.sum=l.sum+r.sum;
   ret.prefixsum=max(l.prefixsum,l.sum+r.prefixsum);
   ret.suffixsum=max(r.suffixsum,l.suffixsum+r.sum);
   ret.maxsum=max(l.maxsum,r.maxsum);
   ret.maxsum=max(ret.maxsum,l.suffixsum+r.prefixsum);
   return ret;
}
void build(int node,int b,int e)
{
   if(b==e) tree[node]=data(v[b]);
   else{
      int m=(b+e)/2;
      build(node*2,b,m);
      build(node*2+1,m+1,e);
      tree[node]=milao(tree[node*2],tree[node*2+1]);
   }
}
data query(int node,int b,int e,int i,int j)
{
```

```
    if(i>j || j<b || i>e) return data(0,-1<<30,-1<<30,-1<<30);
///important:sum=0, other=-inf, cuz others max is taken, sums
sum is taken
    if(i<=b && e<=j) return tree[node];
    int m=(b+e)/2;
    data l=query(node*2,b,m,i,j);
    data r=query(node*2+1,m+1,e,i,j);
    return milao(l,r);
}
int main()
{
    //ios_base::sync_with_stdio(false);
    int t,test=1;
    scanf("%d",&t);
    while(t--)
    {
        int n;
        scanf("%d",&n);
        v=vector<int>(n);
        for(int i=0;i<n;i++)
        {
            scanf("%d",&v[i]);
        }
        build(1,0,n-1);
        int m;
        scanf("%d\n",&m);
        while(m--)
        {
            int x1,x2,y1,y2,ans;
            scanf("%d%d%d%d",&x1,&y1,&x2,&y2);
            x1--;x2--;y1--;y2--;
            if(y1<x2){///no overlap

ans=query(1,0,n-1,x1,y1).suffixsum+query(1,0,n-1,y1+1,x2-1).sum+quer
y(1,0,n-1,x2,y2).prefixsum;
            }
            else {///overlap

ans=query(1,0,n-1,x1,y1).suffixsum+query(1,0,n-1,y1+1,y2).prefixsum;
                ans=max(ans,
query(1,0,n-1,x1,x2-1).suffixsum+query(1,0,n-1,x2,y2).prefixsum );
                ans=max(ans, query(1,0,n-1,x2,y1).maxsum );
            }
            printf("%d\n",ans);
        }
    }
    return 0;
}
```

CENTROID DECOMPOSITION

```
/**
Xenia the programmer has a tree consisting of n nodes. We will
consider the tree nodes indexed from 1 to n.
We will also consider the first node to be initially painted red, and
the other nodes — to be painted blue.

The distance between two tree nodes v and u is the number of
edges in the shortest path between v and u.

Xenia needs to learn how to quickly execute queries of two types:

1.paint a specified blue node in red;
2.calculate which red node is the closest to the given one and print
the shortest distance to the closest red node.
Your task is to write a program which will execute the described
queries.
*/

//Tanuj Khattar
#include<bits/stdc++.h>

using namespace std;

#define PB push_back
#define MP make_pair
#define F first
#define S second
#define SZ(a) (int)(a.size())
#define ALL(a) a.begin(),a.end()
#define SET(a,b) memset(a,b,sizeof(a))
#define LET(x,a) __typeof(a) x(a)
#define dout(n) printf("%d\n",n)
const int N = int(1e5)+10;
const int LOGN = 20;
const int INF = int(1e9);
set<int> g[N];
int par[N],sub[N],level[N],ans[N];
int DP[LOGN][N];
int n,m;
/*Using centroid Decomposition of a tree */

/*----------- Pre-Processing ------------*/
void dfs0(int u)
{
    for(auto it=g[u].begin();it!=g[u].end();it++)
                if(*it!=DP[0][u])
                {
```

```
                DP[0][*it]=u;
                level[*it]=level[u]+1;
                dfs0(*it);
        }
}
void preprocess()
{
   level[0]=0;
   DP[0][0]=0;///LCA table
   dfs0(0);
   for(int i=1;i<LOGN;i++)
        for(int j=0;j<n;j++)
                DP[i][j] = DP[i-1][DP[i-1][j]];
}
int lca(int a,int b)
{
   if(level[a]>level[b])swap(a,b);
   int d = level[b]-level[a];
   for(int i=0;i<LOGN;i++)
        if(d&(1<<i))
                b=DP[i][b];
   if(a==b)return a;
   for(int i=LOGN-1;i>=0;i--)
        if(DP[i][a]!=DP[i][b])
                a=DP[i][a],b=DP[i][b];
   return DP[0][a];
}
int dist(int u,int v)
{
   int x=u;
   int y=v;
   cout<<"lca  of "<<x<<" and "<<y<<" is "<<lca(x,y)<<endl;
   cout<<"dist "<<level[u] + level[v] - 2*level[lca(u,v)]<<endl;
   return level[u] + level[v] - 2*level[lca(u,v)];

   ///in this problem level is the distance
}
/*-----------------Decomposition
Part-------------------------*/
int nn;
void dfs1(int u,int p)
{
   sub[u]=1;
   nn++;
   for(auto it=g[u].begin();it!=g[u].end();it++)
        if(*it!=p)
        {
                dfs1(*it,u);
                sub[u]+=sub[*it];
        }
}
int dfs2(int u,int p)
{
   for(auto it=g[u].begin();it!=g[u].end();it++)
        if(*it!=p && sub[*it]>nn/2)
                return dfs2(*it,u);
   return u;
}
void decompose(int root,int p)
{
   nn=0;
   dfs1(root,root);
   int centroid = dfs2(root,root);
   if(p==-1)p=centroid;
   par[centroid]=p;///parent linking in centroid decomposed tree
   for(auto it=g[centroid].begin();it!=g[centroid].end();it++)
   {
        g[*it].erase(centroid);
        decompose(*it,centroid);
   }
   g[centroid].clear();
}
/*----------------- Handle the Queries -----------------*/

void update(int u)
{
   int x = u;
   while(1)
   {
        ans[x] = min(ans[x],dist(x,u));
        if(x==par[x])
                break;
        x = par[x];
   }
}
int query(int u)
{
   int x = u;
   int ret=INF;
   while(1)
   {
        ret = min(ret,dist(u,x) + ans[x]);
        if(x==par[x])
                break;
        x = par[x];
        cout<<"x is "<<x<<endl;
   }
   return ret;
}
int main()
```

```
{
   scanf("%d %d",&n,&m);
   for(int i=0;i<n-1;i++)
   {
            int u,v;
            scanf("%d %d",&u,&v);
            g[u-1].insert(v-1);
            g[v-1].insert(u-1);
   }
   preprocess();///made lca table
   decompose(0,-1);
   for(int i=0;i<n;i++)
            ans[i]=INF;
   update(0);//first node is initially painted red
   while(m--)
   {
            int t,v;
            scanf("%d %d",&t,&v);v--;
            if(t==1)
                     update(v);
            else
                     dout(query(v));
   }
   return 0;
}
```

CENTROID DECOMPOSITION mine

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

#define pb push_back
#define Pb pop_back

#define PI acos(-1.00)
#define pii pair<int,int>
#define ppi pair<pii,int>
#define inf int(1e9)
#define MOD 1000000007

#define LL long long
#define LLU unsigned long long

#define fs first
#define sc second
#define nd 100003
#define ln 20
using namespace std;

vector<int>gr[100003];
int n,table[nd][ln],level[nd],par[nd],sz[nd],Siz,ans[nd];///parent in centroid tree
bool is_centroid[nd];

void dfs1(int u,int p)
{
   for(int i=0;i<gr[u].size();i++)
   {
      int v=gr[u][i];
      if(v!=p){
         level[v]=level[u]+1;
         table[v][0]=u;
         dfs1(v,u);
      }
   }
}
void preprocess()
{
   memset(table,-1,sizeof table);
   level[0]=0;
   table[0][0]=0;
   dfs1(0,-1);
   ///lca precalc
   for(int j=1;(1<<j)<n;j++)
   {
      for(int i=0;i<n;i++)
      {
         if(table[i][j-1]!=-1) table[i][j]=table[table[i][j-1]][j-1];
      }
   }
}
int lca(int x,int y)
{
   if(level[x]<level[y]) swap(x,y);
   int lg=1;
   for(;(1<<lg)<=level[x];lg++);
   lg--;
   for(int i=lg;i>=0;i--)
   {
      if(level[x]-(1<<i)>=level[y]) x=table[x][i];
   }
   if(x==y) return x;
   for(int i=lg;i>=0;i--)
   {
      if(table[x][i]!=-1 && table[x][i]!=table[y][i]) {
         x=table[x][i];
         y=table[y][i];
      }
   }
   return table[x][0];
```

```
}
int dist(int x,int y)
{
   return level[x]+level[y]-2*level[lca(x,y)];
}
int dfs2(int u,int p){
   sz[u]=1;
   for(int i=0;i<gr[u].size();i++)
   {
      int v=gr[u][i];
      if(v!=p && !is_centroid[v]){
         sz[u]+=dfs2(v,u);
      }
   }
   return sz[u];
}
int dfs3(int u,int p)///centroid finder
{
   for(int i=0;i<gr[u].size();i++)
   {
      int v=gr[u][i];
      if(v!=p && !is_centroid[v] && sz[v]>Siz/2){
         return dfs3(v,u);
      }
   }
   return u;
}
void decompose(int u,int p)
{
   Siz=dfs2(u,u);///calculate size
   int centroid=dfs3(u,u);
   is_centroid[centroid]=true;
   par[centroid]=(p==-1?centroid:p);
   for(int i=0;i<gr[centroid].size();i++)
   {
      int v=gr[centroid][i];
      if(!is_centroid[v]) decompose(v,centroid);
   }
}
void update(int u)
{
   int x=u;
   while(1)
   {
      ans[x]=min(ans[x],dist(x,u));
      if(x==par[x]) break;
      x=par[x];
   }
}
int query(int u)
{
   int x=u;
   int ret=inf;
   while(1){
      ret=min(ret,ans[x]+dist(x,u));
      if(x==par[x]) break;
      x=par[x];
   }
   return ret;
}
int main()
{
   ios_base::sync_with_stdio(false);
   int m;
   cin>>n>>m;
   for(int i=0;i<n-1;i++)
   {
      int x,y;
      cin>>x>>y;
      x--;
      y--;
      gr[x].pb(y);
      gr[y].pb(x);
   }
   preprocess();
   decompose(0,-1);
   for(int i=0;i<n;i++) ans[i]=inf;
   update(0);
   while(m--){
      int type,v;
      cin>>type>>v;
      v--;
      if(type==1) update(v);
      else cout<<query(v)<<endl;
   }
   return 0;
}
```

QTREE5

/**
You are given a tree (an acyclic undirected connected graph) with N nodes.
The tree nodes are numbered from 1 to N. We define dist(a, b) as the number of edges on the path from node a to node b.

Each node has a color, white or black. All the nodes are black initially.

We will ask you to perform some instructions of the following form:

```
0 i : change the color of i-th node(from black to white, or from white to black).
1 v : ask for the minimum dist(u, v), node u must be white(u can be equal to v). Obviously, as long as node v is white, the result will always be 0.
**/
#define nd 100003
#define ln 20
using namespace std;

vector<int>gr[100003];
int n,table[nd][ln],level[nd],par[nd],sz[nd],Siz,col[nd];///parent in centroid tree
bool is_centroid[nd];
multiset<pii>ans[nd];

void dfs1(int u,int p)
{
   for(int i=0;i<gr[u].size();i++)
   {
      int v=gr[u][i];
      if(v!=p){
         level[v]=level[u]+1;
         table[v][0]=u;
         dfs1(v,u);
      }
   }
}
void preprocess()
{
   memset(table,-1,sizeof table);
   level[0]=0;
   table[0][0]=0;
   dfs1(0,-1);
   ///lca precalc
   for(int j=1;(1<<j)<n;j++)
   {
      for(int i=0;i<n;i++)
      {
         if(table[i][j-1]!=-1) table[i][j]=table[table[i][j-1]][j-1];
      }
   }
}
int lca(int x,int y)
{
   if(level[x]<level[y]) swap(x,y);
   int lg=1;
   for(;(1<<lg)<=level[x];lg++);
   lg--;
   for(int i=lg;i>=0;i--)
   {
      if(level[x]-(1<<i)>=level[y]) x=table[x][i];
   }
   if(x==y) return x;
   for(int i=lg;i>=0;i--)
   {
      if(table[x][i]!=-1 && table[x][i]!=table[y][i]) {
         x=table[x][i];
         y=table[y][i];
      }
   }
   return table[x][0];
}
int dist(int x,int y)
{
   return level[x]+level[y]-2*level[lca(x,y)];
}
int dfs2(int u,int p){
   sz[u]=1;
   for(int i=0;i<gr[u].size();i++)
   {
      int v=gr[u][i];
      if(v!=p && !is_centroid[v]){
         sz[u]+=dfs2(v,u);
      }
   }
   return sz[u];
}
int dfs3(int u,int p)///centroid finder
{
   for(int i=0;i<gr[u].size();i++)
   {
      int v=gr[u][i];
      if(v!=p && !is_centroid[v] && sz[v]>Siz/2){
         return dfs3(v,u);
      }
   }
   return u;
}
void decompose(int u,int p)
{
   Siz=dfs2(u,u);///calculate size
   int centroid=dfs3(u,u);
   is_centroid[centroid]=true;
   par[centroid]=(p==-1?centroid:p);
   for(int i=0;i<gr[centroid].size();i++)
   {
      int v=gr[centroid][i];
      if(!is_centroid[v]) decompose(v,centroid);
```

```
    }
}
void white_update(int u)
{
    col[u]=1;
    int x=u;
    while(1)
    {
        ans[x].insert(pii(dist(x,u),u));
        if(x==par[x]) break;
        x=par[x];
    }
}
void black_update(int u)
{
    col[u]=0;
    int x=u;
    while(1)
    {
        ans[x].erase(pii(dist(x,u),u));
        if(x==par[x]) break;
        x=par[x];
    }
}
int query(int u)
{
    int x=u;
    int ret=inf;
    while(1){
        if(!ans[x].empty())
            ret=min(ret,(*(ans[x].begin())).fs+dist(x,u));
        if(x==par[x]) break;
        x=par[x];
    }
    if(ret==inf) return -1;
    else return ret;
}
int main()
{
    ios_base::sync_with_stdio(false);
    int m;
    cin>>n;
    for(int i=0;i<n-1;i++)
    {
        int x,y;
        cin>>x>>y;
        x--;
        y--;
        gr[x].pb(y);
        gr[y].pb(x);
    }
    preprocess();
    decompose(0,-1);
    cin>>m;
    while(m--){
        int type,v;
        cin>>type>>v;
        v--;
        if(!type) {if(col[v]) black_update(v); else white_update(v);}
        else cout<<query(v)<<endl;
    }
    return 0;
}
```

GRAHAM'S SCAN CONVEX HULL

```
int n,fx=1<<30,fy=1<<30,in;///first points
vector<pii>p,Hull;

int cw_ccw(pii o,pii a,pii b)
{
    ///is OB cw or ccw wrt OA?
    ///OA x OB >0 ccw , <0 cw , =0 collinear
    int x=(a.fs-o.fs)*(b.sc-o.sc)-(b.fs-o.fs)*(a.sc-o.sc);
    return x;
}
ll dist(pii a,pii b)
{
    return (a.fs-b.fs)*(a.fs-b.fs)+(a.sc-b.sc)*(a.sc-b.sc);
}

bool comp (pii a, pii b)
{
    int x=cw_ccw(pii(fx,fy),a,b);
    if(x<0) return false;
    else if(x>0) return true;
    else if(dist(pii(fx,fy),a)>dist(pii(fx,fy),b)) return false;
    else return true;
}

void find_first_point()
{
    for(int i=0; i<n; i++)
    {
        if(p[i].sc<fy)
        {
            in=i;
            fy=p[i].sc;
            fx=p[i].fs;
        }
```

```
        else if(p[i].sc==fy && p[i].fs<fx)
        {
            in=i;
            fx=p[i].fs;
        }
    }
}

void MakeHull()
{
    find_first_point();
    p.erase(p.begin()+in);
    sort(p.begin(),p.end(),comp);
    p.insert(p.begin(),pii(fx,fy));
    for(int i=1; i<p.size()-1; i++)
    {
        if(cw_ccw(pii(fx,fy),p[i],p[i+1])==0)
        {
            p.erase(p.begin()+i);
            i--;
        }
    }
    if(p.size()<3) return;
    Hull.pb(p[0]);
    Hull.pb(p[1]);
    Hull.pb(p[2]);
    int hullsize=2;//last index
    for(int i=3; i<p.size(); i++)
    {
        while(cw_ccw(Hull[hullsize-1],Hull[hullsize],p[i])<=0)
        {
            Hull.pop_back();
            hullsize--;
        }
        Hull.pb(p[i]);
        hullsize++;
    }
}

void prntHull()
{
    cout<<"Vertices of Convex Hull : "<<endl;
    for(int i=0; i<Hull.size(); i++)
    {
        cout<<Hull[i].fs<<" "<<Hull[i].sc<<endl;
    }
}
```

MONOTONE CHAIN CONVEX HULL

```
int ccw(pii a,pii b,pii c)
{
    ///ab X ac
    return (b.fs-a.fs)*(c.sc-a.sc)-(b.sc-a.sc)*(c.fs-a.fs);
}

///probably works either way, first wrt to x coordinate or y coordinate(wiki)
bool cmp(pii a,pii b)
{
    if(a.sc==b.sc) return a.fs<b.fs;
    else return a.sc<b.sc;
}

vector<pii> monotone (vector<pii>p)
{
    int n=p.size();
    vector<pii>hull;
    if(n<3) return hull;
    sort(p.begin(),p.end(),cmp);
    hull.pb(p[0]);
    hull.pb(p[1]);
    int hullsize=2;
    ///lower hull
    for(int i=2;i<n;i++)
    {
        while(hullsize>=2 && ccw(hull[hullsize-2],hull[hullsize-1],p[i])<=0)
        {
            hull.pop_back();
            hullsize--;
        }
        hull.pb(p[i]);
        hullsize++;
    }
    int t=hullsize;
    ///upper hull
    for(int i=n-2;i>-1;i--)
    {
        ///at least 2 points in upper hull
        while(hullsize>t && ccw(hull[hullsize-2],hull[hullsize-1],p[i])<=0)
        {
            hull.pop_back();
            hullsize--;
        }
        hull.pb(p[i]);
```

```
        hullsize++;
    }
    hull.pop_back();
    hullsize--;
    return hull;
}

int main()
{
    int n;
    cin>>n;
    vector<pii>p(n);
    for(int i=0;i<n;i++)
    {
        cin>>p[i].fs>>p[i].sc;
    }
    vector<pii>hull;
    hull=monotone(p);
    cout<<"HUll :"<<endl;
    for(int i=0;i<hull.size();i++)
    {
        cout<<hull[i].fs<<" "<<hull[i].sc<<endl;
    }
    return 0;
}
```

DYNAMICALLY ADDING POINTS TO CONVEX HULL

```
// C++ program to add given a point p to a given
// convext hull. The program assumes that the
// point of given convext hull are in anti-clockwise
// order.
#include<bits/stdc++.h>
using namespace std;

// checks whether the point crosses the convex hull
// or not
int orientation(pair<int, int> a, pair<int, int> b,
                pair<int, int> c)
{
    int res = (b.second-a.second)*(c.first-b.first) -
              (c.second-b.second)*(b.first-a.first);

    if (res == 0)
        return 0;
    if (res > 0)
        return 1;
    return -1;
}

// Returns the square of distance between two input points
int sqDist(pair<int, int> p1, pair<int, int> p2)
{
    return (p1.first-p2.first)*(p1.first-p2.first) +
           (p1.second-p2.second)*(p1.second-p2.second);
}

// Checks whether the point is inside the convex hull or not
bool inside(vector<pair<int, int>> a, pair<int, int> p)
{
    // Initialize the centroid of the convex hull
    pair<int, int> mid = {0, 0};

    int n = a.size();

    // Multiplying with n to avoid floating point
    // arithmetic.
    p.first *= n;
    p.second *= n;
    for (int i=0; i<n; i++)
    {
        mid.first += a[i].first;
        mid.second += a[i].second;
        a[i].first *= n;
        a[i].second *= n;
    }

    // if the mid and the given point lies always
    // on the same side w.r.t every edge of the
    // convex hull, then the point lies inside
    // the convex hull
    for (int i=0, j; i<n; i++)
    {
        j = (i+1)%n;
        int x1 = a[i].first, x2 = a[j].first,
        int y1 = a[i].second, y2 = a[j].second;
        int a1 = y1-y2;
        int b1 = x2-x1;
        int c1 = x1*y2-y1*x2;
        int for_mid = a1*mid.first+b1*mid.second+c1;
        int for_p = a1*p.first+b1*p.second+c1;
        if (for_mid*for_p < 0)
            return false;
    }

    return true;
}

// Adds a point p to given convex hull a[]
```

```
void addPoint(vector<pair<int, int>> &a, pair<int, int> p)
{
   // If point is inside p
   if (inside(a, p))
      return;

   // point having minimum distance from the point p
   int ind = 0;
   int n = a.size();
   for (int i=1; i<n; i++)
      if (sqDist(p, a[i]) < sqDist(p, a[ind]))
         ind = i;

   // Find the upper tangent
   int up = ind;
   while (orientation(p, a[up], a[(up+1)%n])>=0)
      up = (up + 1) % n;

   // Find the lower tangent
   int low = ind;
   while (orientation(p, a[low], a[(n+low-1)%n])<=0)
      low = (n+low - 1) % n;

   // Initialize result
   vector<pair<int, int>>ret;

   // making the final hull by traversing points
   // from up to low of given convex hull.
   int curr = up;
   ret.push_back(a[curr]);
   while (curr != low)
   {
      curr = (curr+1)%n;
      ret.push_back(a[curr]);
   }

   // Modify the original vector
   ret.push_back(p);
   a.clear();
   for (int i=0; i<ret.size(); i++)
      a.push_back(ret[i]);
}

// Driver code
int main()
{
   // the set of points in the convex hull in anticlockwise direction
   vector<pair<int, int> > a;
   a.push_back({0, 0});
   a.push_back({3, -1});
   a.push_back({4, 5});
   a.push_back({-1, 4});
   int n = a.size();

   pair<int, int> p = {100, 100};
   addPoint(a, p);

   // Print the modified Convex Hull
   for (auto e : a)
      cout << "(" << e.first << ", "
           << e.second << ") ";

   return 0;
}
```

LINE SEGMENT INTERSECTION

```
// A C++ program to check if two given line segments intersect
#include <iostream>
using namespace std;

struct Point
{
   int x;
   int y;
};

// Given three colinear points p, q, r, the function checks if
// point q lies on line segment 'pr'
bool onSegment(Point p, Point q, Point r)
{
   if (q.x <= max(p.x, r.x) && q.x >= min(p.x, r.x) &&
       q.y <= max(p.y, r.y) && q.y >= min(p.y, r.y))
      return true;

   return false;
}

// To find orientation of ordered triplet (p, q, r).
// The function returns following values
// 0 --> p, q and r are colinear
// 1 --> Clockwise
// 2 --> Counterclockwise
int orientation(Point p, Point q, Point r)
{
   // See
https://www.geeksforgeeks.org/orientation-3-ordered-points/
   // for details of below formula.
   int val = (q.y - p.y) * (r.x - q.x) -
             (q.x - p.x) * (r.y - q.y);
```

```
    if (val == 0) return 0;  // colinear

    return (val > 0)? 1: 2; // clock or counterclock wise
}

// The main function that returns true if line segment 'p1q1'
// and 'p2q2' intersect.
bool doIntersect(Point p1, Point q1, Point p2, Point q2)
{
    // Find the four orientations needed for general and
    // special cases
    int o1 = orientation(p1, q1, p2);
    int o2 = orientation(p1, q1, q2);
    int o3 = orientation(p2, q2, p1);
    int o4 = orientation(p2, q2, q1);

    // General case
    if (o1 != o2 && o3 != o4)
        return true;

    // Special Cases
    // p1, q1 and p2 are colinear and p2 lies on segment p1q1
    if (o1 == 0 && onSegment(p1, p2, q1)) return true;

    // p1, q1 and q2 are colinear and q2 lies on segment p1q1
    if (o2 == 0 && onSegment(p1, q2, q1)) return true;

    // p2, q2 and p1 are colinear and p1 lies on segment p2q2
    if (o3 == 0 && onSegment(p2, p1, q2)) return true;

     // p2, q2 and q1 are colinear and q1 lies on segment p2q2
    if (o4 == 0 && onSegment(p2, q1, q2)) return true;

    return false; // Doesn't fall in any of the above cases
}

// Driver program to test above functions
int main()
{
    struct Point p1 = {1, 1}, q1 = {10, 1};
    struct Point p2 = {1, 2}, q2 = {10, 2};

    doIntersect(p1, q1, p2, q2)? cout << "Yes\n": cout << "No\n";

    p1 = {10, 0}, q1 = {0, 10};
    p2 = {0, 0}, q2 = {10, 10};
    doIntersect(p1, q1, p2, q2)? cout << "Yes\n": cout << "No\n";

    p1 = {-5, -5}, q1 = {0, 0};
    p2 = {1, 1}, q2 = {10, 10};
    doIntersect(p1, q1, p2, q2)? cout << "Yes\n": cout << "No\n";

    return 0;
}
```

ANY LINE SEGMENT INTERSECTION

STANFORD GEO LIB

```
// C++ routines for computational geometry.

#include <iostream>
#include <vector>
#include <cmath>
#include <cassert>

using namespace std;

double INF = 1e100;
double EPS = 1e-12;

struct PT {
  double x, y;
  PT() {}
  PT(double x, double y) : x(x), y(y) {}
  PT(const PT &p) : x(p.x), y(p.y)    {}
  PT operator + (const PT &p)  const { return PT(x+p.x, y+p.y); }
  PT operator - (const PT &p)  const { return PT(x-p.x, y-p.y); }
  PT operator * (double c)     const { return PT(x*c,   y*c  ); }
  PT operator / (double c)     const { return PT(x/c,   y/c  ); }
};

double dot(PT p, PT q)     { return p.x*q.x+p.y*q.y; }
double dist2(PT p, PT q)   { return dot(p-q,p-q); }
double cross(PT p, PT q)   { return p.x*q.y-p.y*q.x; }
ostream &operator<<(ostream &os, const PT &p) {
  os << "(" << p.x << "," << p.y << ")";
}

// rotate a point CCW or CW around the origin
PT RotateCCW90(PT p)   { return PT(-p.y,p.x); }
PT RotateCW90(PT p)    { return PT(p.y,-p.x); }
PT RotateCCW(PT p, double t) {
  return PT(p.x*cos(t)-p.y*sin(t), p.x*sin(t)+p.y*cos(t));
}

// project point c onto line through a and b
```

```
// assuming a != b
PT ProjectPointLine(PT a, PT b, PT c) {
  return a + (b-a)*dot(c-a, b-a)/dot(b-a, b-a);
}

// project point c onto line segment through a and b
PT ProjectPointSegment(PT a, PT b, PT c) {
  double r = dot(b-a,b-a);
  if (fabs(r) < EPS) return a;
  r = dot(c-a, b-a)/r;
  if (r < 0) return a;
  if (r > 1) return b;
  return a + (b-a)*r;
}

// compute distance from c to segment between a and b
double DistancePointSegment(PT a, PT b, PT c) {
  return sqrt(dist2(c, ProjectPointSegment(a, b, c)));
}

// compute distance between point (x,y,z) and plane ax+by+cz=d
double DistancePointPlane(double x, double y, double z,
                          double a, double b, double c, double d)
{
  return fabs(a*x+b*y+c*z-d)/sqrt(a*a+b*b+c*c);
}

// determine if lines from a to b and c to d are parallel or collinear
bool LinesParallel(PT a, PT b, PT c, PT d) {
  return fabs(cross(b-a, c-d)) < EPS;
}

bool LinesCollinear(PT a, PT b, PT c, PT d) {
  return LinesParallel(a, b, c, d)
      && fabs(cross(a-b, a-c)) < EPS
      && fabs(cross(c-d, c-a)) < EPS;
}

// determine if line segment from a to b intersects with
// line segment from c to d
bool SegmentsIntersect(PT a, PT b, PT c, PT d) {
  if (LinesCollinear(a, b, c, d)) {
    if (dist2(a, c) < EPS || dist2(a, d) < EPS ||
        dist2(b, c) < EPS || dist2(b, d) < EPS) return true;
    if (dot(c-a, c-b) > 0 && dot(d-a, d-b) > 0 && dot(c-b, d-b) > 0)
      return false;
    return true;
  }
  if (cross(d-a, b-a) * cross(c-a, b-a) > 0) return false;
  if (cross(a-c, d-c) * cross(b-c, d-c) > 0) return false;
  return true;
}

// compute intersection of line passing through a and b
// with line passing through c and d, assuming that unique
// intersection exists; for segment intersection, check if
// segments intersect first
PT ComputeLineIntersection(PT a, PT b, PT c, PT d) {
  b=b-a; d=c-d; c=c-a;
  assert(dot(b, b) > EPS && dot(d, d) > EPS);
  return a + b*cross(c, d)/cross(b, d);
}

// compute center of circle given three points
PT ComputeCircleCenter(PT a, PT b, PT c) {
  b=(a+b)/2;
  c=(a+c)/2;
  return ComputeLineIntersection(b, b+RotateCW90(a-b), c,
c+RotateCW90(a-c));
}

// determine if point is in a possibly non-convex polygon (by
William
// Randolph Franklin); returns 1 for strictly interior points, 0 for
// strictly exterior points, and 0 or 1 for the remaining points.
// Note that it is possible to convert this into an *exact* test using
// integer arithmetic by taking care of the division appropriately
// (making sure to deal with signs properly) and then by writing
exact
// tests for checking point on polygon boundary
bool PointInPolygon(const vector<PT> &p, PT q) {
  bool c = 0;
  for (int i = 0; i < p.size(); i++){
    int j = (i+1)%p.size();
    if ((p[i].y <= q.y && q.y < p[j].y ||
      p[j].y <= q.y && q.y < p[i].y) &&
      q.x < p[i].x + (p[j].x - p[i].x) * (q.y - p[i].y) / (p[j].y - p[i].y))
      c = !c;
  }
  return c;
}

// determine if point is on the boundary of a polygon
bool PointOnPolygon(const vector<PT> &p, PT q) {
  for (int i = 0; i < p.size(); i++)
    if (dist2(ProjectPointSegment(p[i], p[(i+1)%p.size()], q), q) < EPS)
      return true;
    return false;
}
```

```
// compute intersection of line through points a and b with
// circle centered at c with radius r > 0
vector<PT> CircleLineIntersection(PT a, PT b, PT c, double r) {
  vector<PT> ret;
  b = b-a;
  a = a-c;
  double A = dot(b, b);
  double B = dot(a, b);
  double C = dot(a, a) - r*r;
  double D = B*B - A*C;
  if (D < -EPS) return ret;
  ret.push_back(c+a+b*(-B+sqrt(D+EPS))/A);
  if (D > EPS)
    ret.push_back(c+a+b*(-B-sqrt(D))/A);
  return ret;
}

// compute intersection of circle centered at a with radius r
// with circle centered at b with radius R
vector<PT> CircleCircleIntersection(PT a, PT b, double r, double R)
{
  vector<PT> ret;
  double d = sqrt(dist2(a, b));
  if (d > r+R || d+min(r, R) < max(r, R)) return ret;
  double x = (d*d-R*R+r*r)/(2*d);
  double y = sqrt(r*r-x*x);
  PT v = (b-a)/d;
  ret.push_back(a+v*x + RotateCCW90(v)*y);
  if (y > 0)
    ret.push_back(a+v*x - RotateCCW90(v)*y);
  return ret;
}

// This code computes the area or centroid of a (possibly
nonconvex)
// polygon, assuming that the coordinates are listed in a clockwise
or
// counterclockwise fashion.  Note that the centroid is often
known as
// the "center of gravity" or "center of mass".
double ComputeSignedArea(const vector<PT> &p) {
  double area = 0;
  for(int i = 0; i < p.size(); i++) {
    int j = (i+1) % p.size();
    area += p[i].x*p[j].y - p[j].x*p[i].y;
  }
  return area / 2.0;
}

double ComputeArea(const vector<PT> &p) {
  return fabs(ComputeSignedArea(p));
}

PT ComputeCentroid(const vector<PT> &p) {
  PT c(0,0);
  double scale = 6.0 * ComputeSignedArea(p);
  for (int i = 0; i < p.size(); i++){
    int j = (i+1) % p.size();
    c = c + (p[i]+p[j])*(p[i].x*p[j].y - p[j].x*p[i].y);
  }
  return c / scale;
}

// tests whether or not a given polygon (in CW or CCW order) is
simple
bool IsSimple(const vector<PT> &p) {
  for (int i = 0; i < p.size(); i++) {
    for (int k = i+1; k < p.size(); k++) {
      int j = (i+1) % p.size();
      int l = (k+1) % p.size();
      if (i == l || j == k) continue;
      if (SegmentsIntersect(p[i], p[j], p[k], p[l]))
        return false;
    }
  }
  return true;
}

int main() {

  // expected: (-5,2)
  cerr << RotateCCW90(PT(2,5)) << endl;

  // expected: (5,-2)
  cerr << RotateCW90(PT(2,5)) << endl;

  // expected: (-5,2)
  cerr << RotateCCW(PT(2,5),M_PI/2) << endl;

  // expected: (5,2)
  cerr << ProjectPointLine(PT(-5,-2), PT(10,4), PT(3,7)) << endl;

  // expected: (5,2) (7.5,3) (2.5,1)
  cerr << ProjectPointSegment(PT(-5,-2), PT(10,4), PT(3,7)) << " "
       << ProjectPointSegment(PT(7.5,3), PT(10,4), PT(3,7)) << " "
       << ProjectPointSegment(PT(-5,-2), PT(2.5,1), PT(3,7)) << endl;

  // expected: 6.78903
  cerr << DistancePointPlane(4,-4,3,2,-2,5,-8) << endl;
```

precode@tanny411
aysha.kamal7@gmail.com


```
  // expected: 1 0 1
  cerr << LinesParallel(PT(1,1), PT(3,5), PT(2,1), PT(4,5)) << " "
     << LinesParallel(PT(1,1), PT(3,5), PT(2,0), PT(4,5)) << " "
     << LinesParallel(PT(1,1), PT(3,5), PT(5,9), PT(7,13)) << endl;

  // expected: 0 0 1
  cerr << LinesCollinear(PT(1,1), PT(3,5), PT(2,1), PT(4,5)) << " "
     << LinesCollinear(PT(1,1), PT(3,5), PT(2,0), PT(4,5)) << " "
     << LinesCollinear(PT(1,1), PT(3,5), PT(5,9), PT(7,13)) << endl;

  // expected: 1 1 1 0
  cerr << SegmentsIntersect(PT(0,0), PT(2,4), PT(3,1), PT(-1,3)) << " "
     << SegmentsIntersect(PT(0,0), PT(2,4), PT(4,3), PT(0,5)) << " "
     << SegmentsIntersect(PT(0,0), PT(2,4), PT(2,-1), PT(-2,1)) << " "
     << SegmentsIntersect(PT(0,0), PT(2,4), PT(5,5), PT(1,7)) << endl;

  // expected: (1,2)
  cerr << ComputeLineIntersection(PT(0,0), PT(2,4), PT(3,1), PT(-1,3))
<< endl;

  // expected: (1,1)
  cerr << ComputeCircleCenter(PT(-3,4), PT(6,1), PT(4,5)) << endl;

  vector<PT> v;
  v.push_back(PT(0,0));
  v.push_back(PT(5,0));
  v.push_back(PT(5,5));
  v.push_back(PT(0,5));

  // expected: 1 1 1 0 0
  cerr << PointInPolygon(v, PT(2,2)) << " "
     << PointInPolygon(v, PT(2,0)) << " "
     << PointInPolygon(v, PT(0,2)) << " "
     << PointInPolygon(v, PT(5,2)) << " "
     << PointInPolygon(v, PT(2,5)) << endl;

  // expected: 0 1 1 1 1
  cerr << PointOnPolygon(v, PT(2,2)) << " "
     << PointOnPolygon(v, PT(2,0)) << " "
     << PointOnPolygon(v, PT(0,2)) << " "
     << PointOnPolygon(v, PT(5,2)) << " "
     << PointOnPolygon(v, PT(2,5)) << endl;

  // expected: (1,6)
  //           (5,4) (4,5)
  //           blank line
  //           (4,5) (5,4)
  //           blank line
  //           (4,5) (5,4)
  vector<PT> u = CircleLineIntersection(PT(0,6), PT(2,6), PT(1,1), 5);
  for (int i = 0; i < u.size(); i++) cerr << u[i] << " "; cerr << endl;
  u = CircleLineIntersection(PT(0,9), PT(9,0), PT(1,1), 5);
  for (int i = 0; i < u.size(); i++) cerr << u[i] << " "; cerr << endl;
  u = CircleCircleIntersection(PT(1,1), PT(10,10), 5, 5);
  for (int i = 0; i < u.size(); i++) cerr << u[i] << " "; cerr << endl;
  u = CircleCircleIntersection(PT(1,1), PT(8,8), 5, 5);
  for (int i = 0; i < u.size(); i++) cerr << u[i] << " "; cerr << endl;
  u = CircleCircleIntersection(PT(1,1), PT(4.5,4.5), 10, sqrt(2.0)/2.0);
  for (int i = 0; i < u.size(); i++) cerr << u[i] << " "; cerr << endl;
  u = CircleCircleIntersection(PT(1,1), PT(4.5,4.5), 5, sqrt(2.0)/2.0);
  for (int i = 0; i < u.size(); i++) cerr << u[i] << " "; cerr << endl;

  // area should be 5.0
  // centroid should be (1.1666666, 1.166666)
  PT pa[] = { PT(0,0), PT(5,0), PT(1,1), PT(0,5) };
  vector<PT> p(pa, pa+4);
  PT c = ComputeCentroid(p);
  cerr << "Area: " << ComputeArea(p) << endl;
  cerr << "Centroid: " << c << endl;

  return 0;
}
```

GEO BASICS mine

```
struct point
{
   int x,y;///integer points
   ///double x,y; double points
   point() {}
   point (int _x,int _y)
   {
      x=_x;
      y=_y;
   }
   point operator+(const point& a)
   {
      return point(x+a.x,y+a.y);
   }
   point operator-(const point& a)
   {
      return point(x-a.x,y-a.y);
   }
   int operator*(const point& a)///dot product
   {
      return x*a.x+y*a.y;
   }
   int operator^(const point& a)///cross product
   {
      return x*a.y-y*a.x;
```

```
    }
};
struct line{
    int a,b,c;///ax+by=c;
    line() {}
    line(int x,int y,int z)
    {
        a=x;
        b=y;
        c=z;
    }
};
struct GEO
{
    double crossProduct(point o,point a,point b)
    {
        /// oa X ob = |oa||ob|sin(oa^ob)
        return (a.x-o.x)*(b.y-o.y)-(a.y-o.y)*(b.x-o.x);
    }
    double dotProduct(point o,point a,point b, int f=1)
    {
        /// oa.ob = |oa||ob|cos(oa^ob)
        if(f) return (a.x-o.x)*(b.x-o.x)+(a.y-o.y)*(b.y-o.y);
        ///if told to find ao.ob :
        ///ao=o-a; ob=b-o; (subtracting corresponding coordinates)
        ///then return sum of products
        else if(f==0) return (o.x-a.x)*(b.x-o.x)+(o.y-a.y)*(b.y-o.y);
    }
    double area_of_parallelogram(point o,point a,point b)
    {
        return abs(crossProduct(o,a,b));
    }
    double area_of_triangle(point o,point a,point b)
    {
        return area_of_parallelogram(o,a,b)*1.0/2;
        ///returned double
    }
    double dist(point a,point b)
    {
        return sqrt( (a.x-b.x)*(a.x-b.x) + (a.y-b.y)*(a.y-b.y) );
    }
    double perpDist(point d,point a,point b)
    {
        ///perpendicular distance from the point d, to the line
cointaing the points a and b
        return area_of_parallelogram(a,d,b)/dist(a,b);
        ///because 1/2*base*height=1/2*area of parallelogram
        ///so, height=area/base;
    }
    double nearestDistToSegement(point d,point a,point b)
    {
        ///nearest distance from point d to line segment ab
        if(dotProduct(b, a, d, 0)>0)
        {
            ///if ab.bd>0 that means d is beyond the point b. so the
            ///nearest point of the segment to the point d is b
            return dist( b,  d);
        }
        else if(dotProduct( a, b, d, 0)>0)
        {
            ///if ba.ad>0 d is beyond a
            return dist( d, a);
        }
        else
        {
            ///d's perpendicular lies on the segment, so thats the
shortest distance
            perpDist( d, a, b);
        }
    }
    int IntegerPointsOnPolygonBoundary(vector<point>&p)
    {
        int ret=0;
        int n=p.size();
        for(int i=0;i<n;i++)
        {
            int j=(i+1)%n;
            ret+=__gcd( abs(p[i].x-p[j].x) , abs(p[i].y-p[j].y) );
        }
        return ret;
    }
    double polyArea(vector<point>&p)///atleast 3 points
    {
        int n=p.size();
        double area=0.0;
        for(int i=0;i<n;i++)
        {
            int j=(i+1)%n;
            area+=crossProduct(point(0,0),p[i],p[j]);
        }
        return fabs(area)/2.0;
    }
    line lineFromPoints(point a,point b){///ax+by=c from two points
        int A=b.y-a.y;
        int B=a.x-b.x;
        return line(A,B,A*a.x+B*a.y);
    }
    point lineLineIntersection(line line1,line line2)
    {
        ///for two lines, NOT line segments
```

```
    double det = line1.a*line2.b - line2.a*line1.b,x,y;
    if(det == 0){
      //Lines are parallel
    }else{
      x = (line2.b*line1.c - line1.b*line2.c)/det;
      y = (line1.a*line2.c - line2.a*line1.c)/det;
    }
    return point(x,y);
    //Line segment intersection, do the above thing, and then check the point
    //by "isPointOnLineSegment" function
    // You must be careful about double precision issues though.
    //If your point is right on the edge of the segment, or if the segment is
    //horizontal or vertical, a simple comparison might be problematic.
    //In these cases, you can either do your comparisons with some tolerance,
    // or else use a fraction class.
  }
  bool isPointOnLineSegment(point startLine,point endLine, point x)
  {
    ///considering the point is already on the LINE
    return (x.x>=min(startLine.x,endLine.x) && x.x<=max(startLine.x,endLine.x)
      && x.y>=min(startLine.y,endLine.y) && x.y<=max(startLine.y,endLine.y));
  }
  line perpendicular(line X,point x)
  {
    //returns a line perpendicular to the given line through x
    int D=-X.b*x.x+X.a*x.y;
    return line(-X.b,X.a,D);
  }
  point midPoint(point A,point B)
  {
    return point((A.x+B.x)/2.0,(A.y+B.y)/2.0);
    //doesnt make sense to use it in 'int' (change accordingly)
  }
  pair<point,int> circleFromPoints(point X,point Y,point Z)
  {
    ///Points must be non-collinear
    //These points uniquely define a circle.
    //Returns Center and radius of the circle
    //We want to find the perpendicular bisectors of XY and YZ,
    //and then find the intersection of those two bisectors.
    //This gives us the center of the circle.
    line perpXY=perpendicular(lineFromPoints(X,Y),midPoint(X,Y));
    line perpYZ=perpendicular(lineFromPoints(Y,Z),midPoint(Y,Z));
    point center=lineLineIntersection(perpXY,perpYZ);
    int radius=dist(center,X);
    return pair<point,int>(center,radius);
  }
  point reflect(line l,point x)
  {
    //reflect x against a line l
    //1.xx' and l are perpendicular, and at equal distance
    line perp=perpendicular(l,x);
    point y=lineLineIntersection(l,perp);
    point xPrime=y-(x-y);
    return xPrime;
    //another way to find the reflected point is to rotate the original point 180 degrees about line l.
  }
  point Rotate(point a,double angle)
  {
    //angle must be in radians
    //counter clock-wise rotation about the origin
    ///if rotating about y, do x-y and after rotation do x+y
    double x=a.x*cos(angle)-a.y*sin(angle);
    double y=a.x*sin(angle)+a.y*cos(angle);
    return point(x,y);
  }
};
```

CLOSEST PAIR OF POINTS

```
///O(n*logn)
struct ClosestPairOfPoint
{
  ///points taken as double
  ///considered all distances ans the square of distances within double
  double inf=numeric_limits<double>::max();
  vector< pair<double,double> > pointsx,pointsy;
  ClosestPairOfPoint() {}
  void addpoint(double x,double y)
  {
    pointsx.push_back({x,y});;
  }
  void Clr()
  {
    pointsx.clear();
    pointsy.clear();
  }
  double dist(pair<double,double>a, pair<double,double>b)
  {
```

```
    double
d=(a.first-b.first)*(a.first-b.first)+(a.second-b.second)*(a.second-b.s
econd);
    return sqrt(d);
  }
  static bool compx(pair<double,double>a , pair<double,double>b)
  {
    return a.first<b.first;
  }
  static bool compy(pair<double,double>a , pair<double,double>b)
  {
    return a.second<b.second;
  }
  double bruteforce(int b,int e)
  {
    double mn=inf;
    for(int i=b;i<=e;i++)
    {
      for(int j=i+1;j<=e;j++)
      {
        mn=min(mn,dist(pointsx[i],pointsx[j]));
      }
    }
    return mn;
  }
  double stripClosest(vector<pair<double,double> >strip , double d)
  {
    for(int i=0;i<strip.size();i++)
    {
      for(int j=i+1;j<strip.size() &&
strip[j].second-strip[i].second<=d ; j++)
      {
        d=min(d,dist(strip[i],strip[j]));
      }
    }
    return d;
  }
  double ClosestPoint( int b,int e, vector< pair<double,double> >&
py )
  {
    if(e-b+1<=3) return bruteforce(b,e);
    int mid=(b+e)/2;
    int midx=pointsx[mid].first;
    vector< pair<double,double> >yl,yr;
    for(int i=0;i<py.size();i++)
    {
      if(py[i].first<=midx) yl.push_back(py[i]);
      else yr.push_back(py[i]);
    }
    double dl=ClosestPoint(b,mid,yl);
    double dr=ClosestPoint(mid+1,e,yr);
    double d=min(dl,dr);
    vector< pair<double,double> > strip;
    for(int i=0;i<py.size();i++)
    {
      if(abs(midx-py[i].first)<=d) strip.push_back(py[i]);
    }
    d=min(d,stripClosest(strip,d));
    return d;
  }
  double run()
  {
    pointsy=pointsx;
    sort(pointsx.begin(),pointsx.end(),compx);
    sort(pointsy.begin(),pointsy.end(),compy);
    int n=pointsx.size()-1;
    return ClosestPoint(0, n, pointsy );
  }
};
```

CLOSEST PAIR line sweep nlogn

```
#define px second
#define py first
typedef pair<long long, long long> pairll;
pairll pnts [MAX];
int compare(pairll a, pairll b)
{
  return a.px<b.px;
}
double closest_pair(pairll pnts[],int n)
{
  sort(pnts,pnts+n,compare);
  double best=INF;
  set<pairll> box;
  box.insert(pnts[0]);
  int left = 0;
  for (int i=1;i<n;++i)
  {
    while (left<i && pnts[i].px-pnts[left].px > best)
      box.erase(pnts[left++]);
    for(typeof(box.begin())
it=box.lower_bound(make_pair(pnts[i].py-best,
pnts[i].px-best));it!=box.end() && pnts[i].py+best>=it->py;it++)
      best = min(best, sqrt(pow(pnts[i].py - it->py,
2.0)+pow(pnts[i].px - it->px, 2.0)));
    box.insert(pnts[i]);
  }
  return best;
}
```

STRING FREQ WITH HASHING

```
int main()
{
    ios_base::sync_with_stdio(false);
    int t,cas=1;
    cin>>t;
    while(t--)
    {
        ///You have to find the number of times p(pattern) occurs as
a substring of t(text) all lower case.
        string t,p;
        cin>>t>>p;
        int m=p.length();
        ll ans=(ll)p[0];
        ll hh=(ll)t[0];
        ll baad=1;
        for(int i=0;i<m;i++)
        {
            baad*=29;
        }
        for(int i=1;i<m;i++)
        {
            ans=ans*29+(ll)p[i];
            hh=hh*29+(ll)t[i];
        }
        int cnt=0;
        if(hh==ans) cnt++;
        for(int i=m;i<t.length();i++)
        {
            hh=hh*29+(ll)t[i];
            hh-=baad*(ll)t[i-m];
            if(hh==ans) cnt++;
        }
        cout<<"Case "<<cas++<<": "<<cnt<<"\n";
    }
    return 0;
}
```

2D HASHING

```
///uva 11019
///number of occurances of one matrix in another one
const int maxn = 1000 + 10;
const int seed1 = 131;
const int seed2 = 13131;
const int MOD = 100013;
const int INF = 0x3f3f3f3f;
char s[maxn][maxn],str[maxn];
ull ha[maxn][maxn];
int T;
int n, m, x, y;
int solve(ull aim){
    ull bit = 1;
    for(int i = 1; i <= y - 1; i++)
        bit = bit * seed1;
    for(int i = 1; i <= n; i++){    //Preprocessing hash for each row
        ull sum = 0;
        for(int j = 1; j <= y - 1; j++){
            sum = sum * seed1 + s[i][j];
        }
        for(int j = y; j <= m; j++){
            sum = sum * seed1 + s[i][j];
            ha[i][j - y + 1] = sum;
            sum -= bit * s[i][j - y + 1];
        }
    }
    int ans = 0;
    bit = 1;
    for(int i = 1; i <= x - 1; i++)
        bit = bit * seed2;
    for(int j = 1; j <= m - y + 1; j++){
        ull sum = 0;
        for(int i = 1; i <= x - 1; i++){
            sum = sum * seed2 + ha[i][j];
        }
        for(int i = x; i <= n; i++){
            sum = sum * seed2 + ha[i][j];
            if(sum == aim) ans++;
            sum -= bit * ha[i - x + 1][j];
        }
    }
    return ans;

}
int main(){
    scanf("%d", &T);
    while(T--){
        scanf("%d%d", &n, &m);
        for(int i = 1; i <= n; i++)
            scanf("%s", s[i] + 1);
        scanf("%d%d", &x, &y);
        ull temp, aim = 0;  //First, x*y two dimensional hash.
        for(int i = 1; i <= x; i++){
            scanf("%s", str + 1);
            temp = 0;
            for(int j = 1; j <= y; j++){
                temp = temp * seed1 + str[j];
```

```
        }
        aim = aim * seed2 + temp;
      }
      printf("%d\n", solve(aim));
   }
   return 0;
}
```

KMP

```
#include<bits/stdc++.h>
using namespace std;
vector<int>lps;///longest 'proper' prefix,suffix
///The failure function: i,e where to go if characters fail to match.
void computeLPS(string pat,int m)
{
   lps[0]=0;
   int len=0;
   int i=1;
   while(i<m)
   {
      if(pat[len]==pat[i]) lps[i]=++len,i++;
      else if(len!=0) len=lps[len-1];
      else lps[i]=0,i++;
   }
}

vector<int> KMPsearch(string text,int n,string pat,int m)
{
   vector<int>in;///indices of matches
   int i=0,j=0;
   while(i<n)
   {
      if(text[i]==pat[j]) i++,j++;
      if(j==m) in.push_back(i-j),j=lps[j-1];
      else if(i<n && text[i]!=pat[j])
      {
         if(j>0) j=lps[j-1];
         else i++;
      }
   }
   return in;
}

int main()
{
   string text,pat;
   cin>>text>>pat;
   int n=text.length(),m=pat.length();
   lps=vector<int>(m);
   computeLPS(pat,m);
   vector<int> hu=KMPsearch(text,n,pat,m);
   for(int i=0;i<hu.size();i++)
   {
      cout<<hu[i]<<" ";
   }
   return 0;
}
```

RECTANGLE UNION

```
///codeforces.com/blog/entry/44484
#include<bits/stdc++.h>
#define P(x,y) make_pair(x,y)
using namespace std;
class Rectangle{
   public:
   int x1 , y1 , x2 , y2;
   static Rectangle empt;
   Rectangle(){
      x1=y1=x2=y2=0;
   }
   Rectangle(int X1 , int Y1 , int X2 , int Y2){
      x1 = X1; y1=Y1;
      x2 = X2; y2=Y2;
   }
   Rectangle intersect(Rectangle R){
      if(R.x1 >= x2 || R.x2 <= x1) return empt;
      if(R.y1 >= y2 || R.y2 <= y1) return empt;
      return Rectangle(max(x1 , R.x1) , max(y1 , R.y1) , min(x2 , R.x2) ,
min(y2 , R.y2));
   }
};
struct Event{
   int x , y1 , y2 , type;
   Event(){}
   Event(int x , int y1 , int y2 , int type):x(x) , y1(y1) , y2(y2) ,
type(type){}
};
bool operator < (const Event&A , const Event&B){
   //if(A.x != B.x)
   return A.x < B.x;
   //if(A.y1 != B.y1) return A.y1 < B.y1;
   //if(A.y2 != B.y2()) A.y2 < B.y2;
}
const int MX=(1<<17);
struct Node{
   int prob , sum , ans;
   Node(){}
```

```
    Node(int prob , int sum , int ans):prob(prob) , sum(sum) ,
ans(ans){}
};
Node tree[MX*4];
int interval[MX];
void build(int x , int a , int b){
    tree[x] = Node(0 , 0 , 0);
    if(a==b){
        tree[x].sum+=interval[a];
        return;
    }
    build(x*2 , a , (a+b)/2);
    build(x*2+1 , (a+b)/2+1 , b);
    tree[x].sum = tree[x*2].sum + tree[x*2+1].sum;
}
int ask(int x){
    if(tree[x].prob) return tree[x].sum;
    return tree[x].ans;
}
int st , en , V;
void update(int x , int a , int b){
    if(st>b || en<a) return;
    if(a>=st && b<=en){
        tree[x].prob+=V;
        return;
    }
    update(x*2 , a , (a+b)/2);
    update(x*2+1 , (a+b)/2+1 , b);
    tree[x].ans = ask(x*2) + ask(x*2+1);
}
Rectangle Rectangle::empt = Rectangle();
vector < Rectangle > Rect;
vector < int > sorted;
vector < Event > sweep;
void compressncalc(){
    sweep.clear();
    sorted.clear();
    for(auto R : Rect){
        sorted.push_back(R.y1);
        sorted.push_back(R.y2);
    }
    sort(sorted.begin() , sorted.end());
    sorted.erase(unique(sorted.begin() , sorted.end()) , sorted.end());
    int sz = sorted.size();
    for(int j=0;j<sorted.size() - 1;j++)
        interval[j+1] = sorted[j+1] - sorted[j];
    for(auto R : Rect){
        sweep.push_back(Event(R.x1 , R.y1 , R.y2 , 1));
        sweep.push_back(Event(R.x2 , R.y1 , R.y2 , -1));
    }
    sort(sweep.begin() , sweep.end());
    build(1,1,sz-1);
}
long long ans;
void Sweep(){
    ans=0;
    if(sorted.empty() || sweep.empty()) return;
    int last = 0 , sz_ = sorted.size();
    for(int j=0;j<sweep.size();j++){
        ans+= 1ll * (sweep[j].x - last) * ask(1);
        last = sweep[j].x;
        V = sweep[j].type;
        st = lower_bound(sorted.begin() , sorted.end() , sweep[j].y1) -
sorted.begin() + 1;
        en = lower_bound(sorted.begin() , sorted.end() , sweep[j].y2) -
sorted.begin();
        update(1 , 1 , sz_-1);
    }
}
int main(){
//    freopen("in.in","r",stdin);
    int n;
    scanf("%d",&n);
    for(int j=1;j<=n;j++){
        int a , b ,c  , d;
        scanf("%d %d %d %d",&a,&b,&c,&d);
        Rect.push_back(Rectangle(a , b , c , d));
    }
    compressncalc();
    Sweep();
    cout<<ans<<endl;
}
```

MAT EXPO

```
#include<bits/stdc++.h>
#define pb push_back

using namespace std;
///Did not Deal with doing mod; for that put mod in the multiply
function;all replace all int with long long
struct Matrix
{
    vector<vector<int> >mat;
    int row,col;

    Matrix() {}
    Matrix (const Matrix &a)
    {
        mat=a.mat;
```

```
        row=a.row;
        col=a.col;
    }
    Matrix(int row,int col):row(row),col(col)
    {
        mat.resize(row);
        for(int i=0; i<row; i++)
        {
            mat[i].resize(col);
        }
    }
    Matrix(vector<vector<int> >val):
mat(val),row(val.size()),col(val[0].size()) {}

    void takeIn(int i,int j,int val)
    {
        mat[i][j]=val;
    }

    Matrix operator*(const Matrix &a) const
    {
        int m=a.col;
        vector<vector<int> >res(row,vector<int>(m));
        for(int i=0; i<row; i++)
        {
            for(int k=0; k<m; k++)
            {
                for(int j=0; j<col; j++)
                {
                    res[i][k]+=mat[i][j]*a.mat[j][k];
                }
            }
        }
        return Matrix(res);
    }

    Matrix identity(int n)
    {
        Matrix a(n,n);
        for(int i=0; i<n; i++)
        {
            for(int j=0; j<n; j++)
            {
                if(i==j) a.mat[i][j]=1;
            }
        }
        return a;
    }

    Matrix expo(int power)
    {
        if(power==1) return mat;
        if(power==0) return identity(row);
        Matrix x=expo(power/2);
        x=x*x;
        if(power&1) x=x*mat;
        return x;
    }
    void printMat()
    {
        for(int i=0; i<row; i++)
        {
            for(int j=0; j<col; j++)
            {
                cout<<mat[i][j]<<" ";
            }
            cout<<endl;
        }
    }
    void WholeInput()
    {
        for(int i=0; i<row; i++)
        {
            for(int j=0; j<col; j++)
            {
                cin>>mat[i][j];
            }
        }
    }
};

int main()
{
    /*
    ///MULTIPLYING TWO MATRICES
    int n,m,k;
    cin>>n>>m>>k;
    Matrix a(n,m),b(m,k);
    a.WholeInput();
    b.WholeInput();
    Matrix res=a*b;
    res.printMat();
    ///END MULTIPLY
    */

    ///FINDING THE N-TH FIBONACCCI
    Matrix M(2,2);
    M.takeIn(0,0,1);
    M.takeIn(0,1,1);
    M.takeIn(1,0,1);
```

precode@tanny411
aysha.kamal7@gmail.com
91

```
    M.takeIn(1,1,0);
    Matrix F(2,1);
    F.takeIn(0,0,1);
    F.takeIn(1,0,1);
    int nth;
    while(cin>>nth)
    {
        Matrix r(M.expo(nth-1)*F);
        cout<<r.mat[0][0]<<endl;
    }
    return 0;
}
```

HUGE FIBONACCI mod m

```
#include <iostream>

long long get_pisano_period(long long m) {
    long long a = 0, b = 1, c = a + b;
    for (int i = 0; i < m * m; i++) {
        c = (a + b) % m;
        a = b;
        b = c;
        if (a == 0 && b == 1) return i + 1;
    }
}

long long get_fibonacci_huge(long long n, long long m) {
    long long remainder = n % get_pisano_period(m);

    long long first = 0;
    long long second = 1;

    long long res = remainder;

    for (int i = 1; i < remainder; i++) {
        res = (first + second) % m;
        first = second;
        second = res;
    }

    return res % m;
}

int main() {
//    for (int i = 2; i < 100; i++) {
//        std::cout << i << " : " << get_pisano_period(i) << std::endl;
//    }
//    return 0;
    long long n, m;
    std::cin >> n >> m;
    std::cout << get_fibonacci_huge(n, m) << '\n';
}
```

LENGTH OF UNION OF LINE SEGMENTS

```
// C++ program to implement Klee's algorithm
#include<bits/stdc++.h>
using namespace std;

// Returns sum of lengths covered by union of given
// segments
int segmentUnionLength(const vector <pair <int,int> > &seg)
{
    int n = seg.size();

    // Create a vector to store starting and ending
    // points
    vector <pair <int, bool> > points(n * 2);
    for (int i = 0; i < n; i++)
    {
        points[i*2]     = make_pair(seg[i].first, false);
        points[i*2 + 1] = make_pair(seg[i].second, true);
    }

    // Sorting all points by point value
    sort(points.begin(), points.end());

    int result = 0; // Initialize result

    // To keep track of counts of current open segments
    // (Starting point is processed, but ending point
    // is not)
    int Counter = 0;

    // Trvaerse through all points
    for (unsigned i=0; i<n*2; i++)
    {
        // If there are open points, then we add the
        // difference between previous and current point.
        // This is interesting as we don't check whether
        // current point is opening or closing,
        if (Counter)
            result += (points[i].first - points[i-1].first);

        // If this is an ending point, reduce, count of
        // open points.
        (points[i].second)? Counter-- : Counter++;
    }
    return result;
```

```
}

// Driver program for the above code
int main()
{
    vector< pair <int,int> > segments;
    segments.push_back(make_pair(3, 15));
    segments.push_back(make_pair(2, 5));
    segments.push_back(make_pair(4, 8));
    segments.push_back(make_pair(9, 12));
    cout << "Length of Union of All segments = ";
    cout << segmentUnionLength(segments) << endl;
    return 0;
}
```

IS POINT INSIDE POLYGON 1190

```
#include <bits/stdc++.h>
using namespace std;
#define INF 30000
struct Point {
    int x;
    int y;
};

bool onSegment(Point p, Point q, Point r) {
    if (q.x <= max(p.x, r.x) && q.x >= min(p.x, r.x) && q.y <= max(p.y,
r.y) && q.y >= min(p.y, r.y))
        return true;
    return false;
}
int orientation(Point p, Point q, Point r) {
    int val = (q.y - p.y) * (r.x - q.x) - (q.x - p.x) * (r.y - q.y);

    if (val == 0) return 0;  // colinear
    return (val > 0)? 1: 2; // clock or counterclock wise
}

int doIntersect(Point p1, Point q1, Point p2, Point q2) {
    int o1 = orientation(p1, q1, p2);
    int o2 = orientation(p1, q1, q2);
    int o3 = orientation(p2, q2, p1);
    int o4 = orientation(p2, q2, q1);

    // General case
    if (o1 != o2 && o3 != o4)
        return true;

    // Special Cases
    // p1, q1 and p2 are colinear and p2 lies on segment p1q1
    if (o1 == 0 && onSegment(p1, p2, q1)) return 2;
    // p1, q1 and p2 are colinear and q2 lies on segment p1q1
    if (o2 == 0 && onSegment(p1, q2, q1)) return true;
    // p2, q2 and p1 are colinear and p1 lies on segment p2q2
    if (o3 == 0 && onSegment(p2, p1, q2)) return true;
     // p2, q2 and q1 are colinear and q1 lies on segment p2q2
    if (o4 == 0 && onSegment(p2, q1, q2)) return true;
    return false; // Doesn't fall in any of the above cases
}

// Returns true if the point p lies inside the polygon[] with n
vertices
bool isInside(Point polygon[], int n, Point p) {
    if (n < 3)  return false;
    // Create a point for line segment from p to infinite
    Point extreme;
    extreme.x = p.x + INF; extreme.y = p.y+ INF+1;
    ///very imporatnt line
    ///extreme point was chosen in such a way that there is no
integer coordinate between
    ///extreme and point p, so the cases of vertex intersections
need not be handled separately.
    // Count intersections of the above line with sides of polygon
    int count = 0, i = 0;
    do {
        int next = (i+1)%n;
        if (int a = doIntersect(polygon[i], polygon[next], p, extreme)) {
            if (a == 2) return 1;
            if (orientation(polygon[i], p, polygon[next]) == 0)
                return onSegment(polygon[i], p, polygon[next]);
            count++;
        }
        i = next;
    } while (i != 0);

    // Return true if count is odd, false otherwise
    return count&1;  // Same as (count%2 == 1)
}
int main()
{
  int tc, cs = 0, n, q;
  //freopen ("in.txt", "r", stdin);
  cin >> tc;
  while (tc--) {
     cin >> n;
     Point polygon1[n];
     for (int i = 0; i < n; i++)
        cin >> polygon1[i].x >> polygon1[i].y;
     cin >> q;
     printf ("Case %d:\n", ++cs);
```

```
    while (q--) {
      Point pp;
      cin >> pp.x >> pp.y;
      isInside(polygon1, n, pp) ? cout << "Yes\n" : cout << "No\n";
    }
  }
  return 0;
}
```

MINIMUM COST POLYGON TRIANGULATION

```
//https://www.geeksforgeeks.org/minimum-cost-polygon-triangulation/
/**
// Recursive implementation for minimum cost convex polygon triangulation
//Similar to MCM
#include <iostream>
#include <cmath>
#define MAX 1000000.0
using namespace std;

// Structure of a point in 2D plane
struct Point
{
  int x, y;
};

// Utility function to find minimum of two double values
double min(double x, double y)
{
  return (x <= y)? x : y;
}

// A utility function to find distance between two points in a plane
double dist(Point p1, Point p2)
{
  return sqrt((p1.x - p2.x)*(p1.x - p2.x) +
              (p1.y - p2.y)*(p1.y - p2.y));
}

// A utility function to find cost of a triangle. The cost is considered
// as perimeter (sum of lengths of all edges) of the triangle
double cost(Point points[], int i, int j, int k)
{
  Point p1 = points[i], p2 = points[j], p3 = points[k];
  return dist(p1, p2) + dist(p2, p3) + dist(p3, p1);
}

// A recursive function to find minimum cost of polygon triangulation
// The polygon is represented by points[i..j].
double mTC(Point points[], int i, int j)
{
  // There must be at least three points between i and j
  // (including i and j)
  if (j < i+2)
    return 0;

  // Initialize result as infinite
  double res = MAX;

  // Find minimum triangulation by considering all
  for (int k=i+1; k<j; k++)
      res = min(res, (mTC(points, i, k) + mTC(points, k, j) +
                      cost(points, i, k, j)));
  return  res;
}

// Driver program to test above functions
int main()
{
   Point points[] = {{0, 0}, {1, 0}, {2, 1}, {1, 2}, {0, 2}};
   int n = sizeof(points)/sizeof(points[0]);
   cout << mTC(points, 0, n-1);
   return 0;
}
*/

///Approach with Memoization//O(n^3)
// A Dynamic Programming based program to find minimum cost of convex
// polygon triangulation
#include <iostream>
#include <cmath>
#define MAX 1000000.0
using namespace std;

// Structure of a point in 2D plane
struct Point
{
  int x, y;
};

// Utility function to find minimum of two double values
double min(double x, double y)
{
```

```
    return (x <= y)? x : y;
}

// A utility function to find distance between two points in a plane
double dist(Point p1, Point p2)
{
    return sqrt((p1.x - p2.x)*(p1.x - p2.x) +
                (p1.y - p2.y)*(p1.y - p2.y));
}

// A utility function to find cost of a triangle. The cost is
considered
// as perimeter (sum of lengths of all edges) of the triangle
double cost(Point points[], int i, int j, int k)
{
    Point p1 = points[i], p2 = points[j], p3 = points[k];
    return dist(p1, p2) + dist(p2, p3) + dist(p3, p1);
}

// A Dynamic programming based function to find minimum cost
for convex
// polygon triangulation.
double mTCDP(Point points[], int n)
{
   // There must be at least 3 points to form a triangle
   if (n < 3)
     return 0;

   // table to store results of subproblems.  table[i][j] stores cost of
   // triangulation of points from i to j.  The entry table[0][n-1]
stores
   // the final result.
   double table[n][n];

   // Fill table using above recursive formula. Note that the table
   // is filled in diagonal fashion i.e., from diagonal elements to
   // table[0][n-1] which is the result.
   for (int gap = 0; gap < n; gap++)
   {
      for (int i = 0, j = gap; j < n; i++, j++)
      {
         if (j < i+2)
            table[i][j] = 0.0;
         else
         {
            table[i][j] = MAX;
            for (int k = i+1; k < j; k++)
            {
              double val = table[i][k] + table[k][j] + cost(points,i,j,k);
              if (table[i][j] > val)
                table[i][j] = val;
            }
         }
      }
   }
   return  table[0][n-1];
}

// Driver program to test above functions
int main()
{
    Point points[] = {{0, 0}, {1, 0}, {2, 1}, {1, 2}, {0, 2}};
    int n = sizeof(points)/sizeof(points[0]);
    cout << mTCDP(points, n);
    return 0;
}
```

MAXIMUM POINTS IN A LINE

//https://www.geeksforgeeks.org/count-maximum-points-on-same-line/
/**
We can solve above problem by following approach – For each point p, calculate its slope with other points and use a map to record how many points
have same slope, by which we can find out how many points are on same line with p as their one point. For each point keep doing the same thing and
 update the maximum number of point count found so far.

Some things to note in implementation are:
1) if two point are (x1, y1) and (x2, y2) then their slope will be (y2 – y1) / (x2 – x1) which can be a double value and can cause precision problems.
To get rid of the precision problems, we treat slope as pair ((y2 – y1), (x2 – x1)) instead of ratio and reduce pair by their gcd before inserting into
 map. In below code points which are vertical or repeated are treated separately.

2) If we use unordered_map in c++ or HashMap in Java for storing the slope pair, then total time complexity of solution will be O(n^2) else O(n^2 * longn)
*/
/* C/C++ program to find maximum number of point
which lie on same line */
#include <bits/stdc++.h>
#include <boost/functional/hash.hpp>

using namespace std;

```
// method to find maximum colinear point
int maxPointOnSameLine(vector< pair<int, int> > points)
{
    int N = points.size();
    if (N < 2)
        return N;

    int maxPoint = 0;
    int curMax, overlapPoints, verticalPoints;

    // here since we are using unordered_map
    // which is based on hash function
    //But by default we don't have hash function for pairs
    //so we'll use hash function defined in Boost library
    unordered_map<pair<int, int>, int,boost::
            hash<pair<int, int> > > slopeMap;

    // looping for each point
    for (int i = 0; i < N; i++)
    {
        curMax = overlapPoints = verticalPoints = 0;

        // looping from i + 1 to ignore same pair again
        for (int j = i + 1; j < N; j++)
        {
            // If both point are equal then just
            // increase overlapPoint count
            if (points[i] == points[j])
                overlapPoints++;

            // If x co-ordinate is same, then both
            // point are vertical to each other
            else if (points[i].first == points[j].first)
                verticalPoints++;

            else
            {
                int yDif = points[j].second - points[i].second;
                int xDif = points[j].first - points[i].first;
                int g = __gcd(xDif, yDif);

                // reducing the difference by their gcd
                yDif /= g;
                xDif /= g;

                // increasing the frequency of current slope
                // in map
                slopeMap[make_pair(yDif, xDif)]++;
                curMax = max(curMax, slopeMap[make_pair(yDif, xDif)]);
            }

            curMax = max(curMax, verticalPoints);
        }

        // updating global maximum by current point's maximum
        maxPoint = max(maxPoint, curMax + overlapPoints + 1);

        // printf("maximum colinear point
        // which contains current point
        // are : %d\n", curMax + overlapPoints + 1);
        slopeMap.clear();
    }

    return maxPoint;
}

// Driver code
int main()
{
    const int N = 6;
    int arr[N][2] = {{-1, 1}, {0, 0}, {1, 1}, {2, 2},
                     {3, 3}, {3, 4}};

    vector< pair<int, int> > points;
    for (int i = 0; i < N; i++)
        points.push_back(make_pair(arr[i][0], arr[i][1]));

    cout << maxPointOnSameLine(points) << endl;

    return 0;
}
```

QUICKHULL divide and conquer

```
/**
```

The QuickHull algorithm is a Divide and Conquer algorithm similar to QuickSort. Let a[0...n-1] be the input array of points.
Following are the steps for finding the convex hull of these points.

1.Find the point with minimum x-coordinate lets say, min_x and similarly the point with maximum x-coordinate, max_x.
2.Make a line joining these two points, say L. This line will divide the the whole set into two parts.
Take both the parts one by one and proceed further.
3.For a part, find the point P with maximum distance from the line L. P forms a triangle with the points min_x, max_x.
It is clear that the points residing inside this triangle can never be the part of convex hull.

4.The above step divides the problem into two sub-problems (solved recursively).
Now the line joining the points P and min_x and the line joining the points P and max_x are new lines and the points
residing outside the triangle is the set of points. Repeat point no. 3 till there no point left with the line.
Add the end points of this point to the convex hull.

The time analysis is similar to Quick Sort.
On average, we get time complexity as O(n Log n), but in worst case, it can become O(n2)
*/

```
// C++ program to implement Quick Hull algorithm
// to find convex hull.
#include<bits/stdc++.h>
using namespace std;

// iPair is integer pairs
#define iPair pair<int, int>

// Stores the result (points of convex hull)
set<iPair> hull;

// Returns the side of point p with respect to line
// joining points p1 and p2.
int findSide(iPair p1, iPair p2, iPair p)
{
    int val = (p.second - p1.second) * (p2.first - p1.first) -
              (p2.second - p1.second) * (p.first - p1.first);

    if (val > 0)
        return 1;
    if (val < 0)
        return -1;
    return 0;
}

// Returns the square of distance between
// p1 and p2.
int dist(iPair p, iPair q)
{
    return (p.second - q.second) * (p.second - q.second) +
           (p.first - q.first) * (p.first - q.first);
}

// returns a value proportional to the distance
// between the point p and the line joining the
// points p1 and p2
int lineDist(iPair p1, iPair p2, iPair p)
{
    return abs ((p.second - p1.second) * (p2.first - p1.first) -
                (p2.second - p1.second) * (p.first - p1.first));
}

// End points of line L are p1 and p2.  side can have value
// 1 or -1 specifying each of the parts made by the line L
void quickHull(iPair a[], int n, iPair p1, iPair p2, int side)
{
    int ind = -1;
    int max_dist = 0;

    // finding the point with maximum distance
    // from L and also on the specified side of L.
    for (int i=0; i<n; i++)
    {
        int temp = lineDist(p1, p2, a[i]);
        if (findSide(p1, p2, a[i]) == side && temp > max_dist)
        {
            ind = i;
            max_dist = temp;
        }
    }

    // If no point is found, add the end points
    // of L to the convex hull.
    if (ind == -1)
    {
        hull.insert(p1);
        hull.insert(p2);
        return;
    }

    // Recur for the two parts divided by a[ind]
    quickHull(a, n, a[ind], p1, -findSide(a[ind], p1, p2));
    quickHull(a, n, a[ind], p2, -findSide(a[ind], p2, p1));
}

void printHull(iPair a[], int n)
{
    // a[i].second -> y-coordinate of the ith point
    if (n < 3)
    {
        cout << "Convex hull not possible\n";
        return;
    }

    // Finding the point with minimum and
    // maximum x-coordinate
    int min_x = 0, max_x = 0;
    for (int i=1; i<n; i++)
```

```
    {
        if (a[i].first < a[min_x].first)
            min_x = i;
        if (a[i].first > a[max_x].first)
            max_x = i;
    }

    // Recursively find convex hull points on
    // one side of line joining a[min_x] and
    // a[max_x]
    quickHull(a, n, a[min_x], a[max_x], 1);

    // Recursively find convex hull points on
    // other side of line joining a[min_x] and
    // a[max_x]
    quickHull(a, n, a[min_x], a[max_x], -1);

    cout << "The points in Convex Hull are:\n";
    while (!hull.empty())
    {
        cout << "(" <<( *hull.begin()).first << ", "
             << (*hull.begin()).second << ") ";
        hull.erase(hull.begin());
    }
}

// Driver code
int main()
{
    iPair a[] = {{0, 3}, {1, 1}, {2, 2}, {4, 4},
                 {0, 0}, {1, 2}, {3, 1}, {3, 3}};
    int n = sizeof(a)/sizeof(a[0]);
    printHull(a, n);
    return 0;
}
```

ANOTHER divide and conquer CONVEX HULL

```
/**
https://www.geeksforgeeks.org/convex-hull-using-divide-and-conquer-algorithm/
Given the set of points for which we have to find the convex hull.
Suppose we know the convex hull of the left half points and the right half points,
then the problem now is to merge these two convex hulls and determine the convex hull for the complete set.
This can be done by finding the upper and lower tangent to the right and left convex hulls.
https://www.geeksforgeeks.org/tangents-two-convex-polygons/
We have used the brute algorithm to find the convex hull for a small number of points and it has a time complexity of O(n^3).
But some people suggest the following, the convex hull for 3 or fewer points is the complete set of points. This is correct but the problem
comes when we try to merge a left convex hull of 2 points and right convex hull of 3 points,then the program gets trapped in an infinite loop
in some special cases.So, to get rid of this problem I directly found the convex hull for 5 or fewer points by O(n^3) algorithm,
which is somewhat greater but does not affect the overall complexity of the algorithm.

The merging of the left and the right convex hulls take O(n)time and as we are dividing the points into two equal parts,
so the time complexity of the above algorithm is O(n * log n).
*/

// A divide and conquer program to find convex
// hull of a given set of points.
#include<bits/stdc++.h>
using namespace std;

// stores the centre of polygon (It is made
// global because it is used in compare function)
pair<int, int> mid;

// determines the quadrant of a point
// (used in compare())
int quad(pair<int, int> p)
{
    if (p.first >= 0 && p.second >= 0)
        return 1;
    if (p.first <= 0 && p.second >= 0)
        return 2;
    if (p.first <= 0 && p.second <= 0)
        return 3;
    return 4;
}

// Checks whether the line is crossing the polygon
int orientation(pair<int, int> a, pair<int, int> b,
                pair<int, int> c)
{
    int res = (b.second-a.second)*(c.first-b.first) -
              (c.second-b.second)*(b.first-a.first);

    if (res == 0)
        return 0;
```

```
    if (res > 0)
        return 1;
    return -1;
}

// compare function for sorting
bool compare(pair<int, int> p1, pair<int, int> q1)
{
    pair<int, int> p = make_pair(p1.first - mid.first,
                            p1.second - mid.second);
    pair<int, int> q = make_pair(q1.first - mid.first,
                            q1.second - mid.second);

    int one = quad(p);
    int two = quad(q);

    if (one != two)
        return (one < two);
    return (p.second*q.first < q.second*p.first);
}

// Finds upper tangent of two polygons 'a' and 'b'
// represented as two vectors.
vector<pair<int, int>> merger(vector<pair<int, int> > a,
                              vector<pair<int, int> > b)
{
    // n1 -> number of points in polygon a
    // n2 -> number of points in polygon b
    int n1 = a.size(), n2 = b.size();

    int ia = 0, ib = 0;
    for (int i=1; i<n1; i++)
        if (a[i].first > a[ia].first)
            ia = i;

    // ib -> leftmost point of b
    for (int i=1; i<n2; i++)
        if (b[i].first < b[ib].first)
            ib=i;

    // finding the upper tangent
    int inda = ia, indb = ib;
    bool done = 0;
    while (!done)
    {
        done = 1;
        while (orientation(b[indb], a[inda], a[(inda+1)%n1]) >=0)
            inda = (inda + 1) % n1;

        while (orientation(a[inda], b[indb], b[(n2+indb-1)%n2]) <=0)
        {
            indb = (n2+indb-1)%n2;
            done = 0;
        }
    }

    int uppera = inda, upperb = indb;
    inda = ia, indb=ib;
    done = 0;
    int g = 0;
    while (!done)//finding the lower tangent
    {
        done = 1;
        while (orientation(a[inda], b[indb], b[(indb+1)%n2])>=0)
            indb=(indb+1)%n2;

        while (orientation(b[indb], a[inda], a[(n1+inda-1)%n1])<=0)
        {
            inda=(n1+inda-1)%n1;
            done=0;
        }
    }

    int lowera = inda, lowerb = indb;
    vector<pair<int, int>> ret;

    //ret contains the convex hull after merging the two convex
hulls
    //with the points sorted in anti-clockwise order
    int ind = uppera;
    ret.push_back(a[uppera]);
    while (ind != lowera)
    {
        ind = (ind+1)%n1;
        ret.push_back(a[ind]);
    }

    ind = lowerb;
    ret.push_back(b[lowerb]);
    while (ind != upperb)
    {
        ind = (ind+1)%n2;
        ret.push_back(b[ind]);
    }
    return ret;

}

// Brute force algorithm to find convex hull for a set
// of less than 6 points
```

precode@tanny411
aysha.kamal7@gmail.com


```
vector<pair<int, int>> bruteHull(vector<pair<int, int>> a)
{
    // Take any pair of points from the set and check
    // whether it is the edge of the convex hull or not.
    // if all the remaining points are on the same side
    // of the line then the line is the edge of convex
    // hull otherwise not
    set<pair<int, int> >s;

    for (int i=0; i<a.size(); i++)
    {
        for (int j=i+1; j<a.size(); j++)
        {
            int x1 = a[i].first, x2 = a[j].first;
            int y1 = a[i].second, y2 = a[j].second;

            int a1 = y1-y2;
            int b1 = x2-x1;
            int c1 = x1*y2-y1*x2;
            int pos = 0, neg = 0;
            for (int k=0; k<a.size(); k++)
            {
                if (a1*a[k].first+b1*a[k].second+c1 <= 0)
                    neg++;
                if (a1*a[k].first+b1*a[k].second+c1 >= 0)
                    pos++;
            }
            if (pos == a.size() || neg == a.size())
            {
                s.insert(a[i]);
                s.insert(a[j]);
            }
        }
    }

    vector<pair<int, int>>ret;
    for (auto e:s)
        ret.push_back(e);

    // Sorting the points in the anti-clockwise order
    mid = {0, 0};
    int n = ret.size();
    for (int i=0; i<n; i++)
    {
        mid.first += ret[i].first;
        mid.second += ret[i].second;
        ret[i].first *= n;
        ret[i].second *= n;
    }
    sort(ret.begin(), ret.end(), compare);

    for (int i=0; i<n; i++)
        ret[i] = make_pair(ret[i].first/n, ret[i].second/n);

    return ret;
}

// Returns the convex hull for the given set of points
vector<pair<int, int>> divide(vector<pair<int, int>> a)
{
    // If the number of points is less than 6 then the
    // function uses the brute algorithm to find the
    // convex hull
    if (a.size() <= 5)
        return bruteHull(a);

    // left contains the left half points
    // right contains the right half points
    vector<pair<int, int>>left, right;
    for (int i=0; i<a.size()/2; i++)
        left.push_back(a[i]);
    for (int i=a.size()/2; i<a.size(); i++)
        right.push_back(a[i]);

    // convex hull for the left and right sets
    vector<pair<int, int>>left_hull = divide(left);
    vector<pair<int, int>>right_hull = divide(right);

    // merging the convex hulls
    return merger(left_hull, right_hull);
}

// Driver code
int main()
{
    vector<pair<int, int> > a;
    a.push_back(make_pair(0, 0));
    a.push_back(make_pair(1, -4));
    a.push_back(make_pair(-1, -5));
    a.push_back(make_pair(-5, -3));
    a.push_back(make_pair(-3, -1));
    a.push_back(make_pair(-1, -3));
    a.push_back(make_pair(-2, -2));
    a.push_back(make_pair(-1, -1));
    a.push_back(make_pair(-2, -1));
    a.push_back(make_pair(-1, 1));

    int n = a.size();

    // sorting the set of points according
    // to the x-coordinate
```

precode@tanny411
aysha.kamal7@gmail.com


```
    sort(a.begin(), a.end());
    vector<pair<int, int> >ans = divide(a);

    cout << "convex hull:\n";
    for (auto e:ans)
        cout << e.first << " "
             << e.second << endl;

    return 0;
}
```

TANGENT OF TWO CONVEX HULLS

```
/**
For finding the upper tangent, we start by taking two points. The
rightmost point of a and leftmost point of b. The line joining them is
labelled as 1.
As this line passes through the polygon b (is not above polygon b)
so we take the anti-clockwise next point on b, the line is labelled 2.
Now the line
is above the polygon b, fine! But the line is crossing the polygon a,
so we move to the clockwise next point, labelled as 3 in the picture.
This again
crossing the polygon a so we move to line 4. This line is crossing b
so we move to line 5. Now this line is crossing neither of the points.
So this is
the upper tangent for the given polygons.
For finding the lower tangent we need to move inversely through
the polygons i.e. if the line is crossing the polygon b we move to
clockwise next and
to anti-clockwise next if the line is crossing the polygon a.

Only upper tangent shown in code, lower tangent can be found
similarly(both present in dovex hull divide and conquer another
way code)
*/
// C++ program to find upper tangent of two polygons.
#include<bits/stdc++.h>
using namespace std;

// stores the center of polygon (It is made
// global becuase it is used in compare function)
pair<int,int> mid;

// determines the quadrant of a point
// (used in compare())
int quad(pair<int,int> p)
{
    if (p.first >= 0 && p.second >= 0)
        return 1;
    if (p.first <= 0 && p.second >= 0)
        return 2;
    if (p.first <= 0 && p.second <= 0)
        return 3;
    return 4;
}

// Checks whether the line is crossing the polygon
int orientation(pair<int,int> a, pair<int,int> b,
                pair<int,int> c)
{
    int res = (b.second-a.second)*(c.first-b.first) -
              (c.second-b.second)*(b.first-a.first);

    if (res == 0)
        return 0;
    if (res > 0)
        return 1;
    return -1;
}

// compare function for sorting
bool compare(pair<int,int> p1, pair<int,int> q1)
{
    pair<int,int> p = make_pair(p1.first - mid.first,
                                p1.second - mid.second);
    pair<int,int> q = make_pair(q1.first - mid.first,
                                q1.second - mid.second);

    int one = quad(p);
    int two = quad(q);

    if (one != two)
        return (one < two);
    return (p.second*q.first < q.second*p.first);
}

// Finds upper tangent of two polygons 'a' and 'b'
// represented as two vectors.
void findUpperTangent(vector<pair<int,int> > a,
                      vector<pair<int,int> > b)
{
    // n1 -> number of points in polygon a
    // n2 -> number of points in polygon b
    int n1 = a.size(), n2 = b.size();

    // To find a point inside the convex polygon(centroid),
    // we sum up all the coordinates and then divide  by
    // n(number of points). But this would be a floating-point
    // value. So to get rid of this we multiply points
```

```
    // initially with n1 and then find the centre and
    // then divided it by n1 again.
    // Similarly we do divide and multiply for n2 (i.e.,
    // elements of b)

    // maxa and minb are used to check if polygon a
    // is left of b.
    int maxa = INT_MIN;
    for (int i=0; i<n1; i++)
    {
        maxa = max(maxa, a[i].first);
        mid.first  += a[i].first;
        mid.second += a[i].second;
        a[i].first *= n1;
        a[i].second *= n1;
    }

    // sorting the points in counter clockwise order
    // for polygon a
    sort(a.begin(), a.end(), compare);

    for (int i=0; i<n1; i++)
    {
        a[i].first /= n1;
        a[i].second /= n1;
    }

    mid = {0, 0};

    int minb = INT_MAX;
    for (int i=0; i<n2; i++)
    {
        mid.first += b[i].first;
        mid.second += b[i].second;
        minb = min(minb, b[i].first);
        b[i].first *= n2;
        b[i].second *= n2;
    }

    // sorting the points in counter clockwise
    // order for polygon b
    sort(b.begin(), b.end(), compare);

    for (int i=0; i<n2; i++)
    {
        b[i].first/=n2;
        b[i].second/=n2;
    }

    // If a is to the right of b, swap a and b
    // This makes sure a is left of b.
    if (minb < maxa)
    {
        a.swap(b);
        n1 = a.size();
        n2 = b.size();
    }

    // ia -> rightmost point of a
    int ia = 0, ib = 0;
    for (int i=1; i<n1; i++)
        if (a[i].first > a[ia].first)
            ia = i;

    // ib -> leftmost point of b
    for (int i=1; i<n2; i++)
        if (b[i].first < b[ib].first)
            ib=i;

    // finding the upper tangent
    int inda = ia, indb = ib;
    bool done = 0;
    while (!done)
    {
        done = 1;
        while (orientation(b[indb], a[inda], a[(inda+1)%n1]) > 0)
            inda = (inda + 1) % n1;

        while (orientation(a[inda], b[indb], b[(n2+indb-1)%n2]) < 0)
        {
            indb = (n2+indb-1)%n2;
            done = 0;
        }
    }

    cout << "upper tangent (" << a[inda].first << ","
        << a[inda].second << ") (" << b[indb].first
        << "," << b[indb].second << ")\n";
}

// Driver code
int main()
{
    vector<pair<int,int> > a;
    a.push_back({2, 2});
    a.push_back({3, 1});
    a.push_back({3, 3});
    a.push_back({5, 2});
    a.push_back({4, 0});
```

precode@tanny411
aysha.kamal7@gmail.com
102

```
    vector<pair<int,int> > b;
    b.push_back({0, 1});
    b.push_back({1, 0});
    b.push_back({0, -2});
    b.push_back({-1, 0});

    findUpperTangent(a, b);

    return 0;
}
```