**STL**

**<iostream>, <iomanip>**

cout, cin, while (cin>> ...)

getline, while + getline, getline after cin

<iomanip>: fixed, setprecision, setw, setfill, hex / dec

<iomanip>: noskipws / skipws

cin / cout.tie (0); cin / cout.sync\_with\_stdio (false); endl vs '\ n'

<cstdio>: printf / scanf for bold inputs

**<vector>**

clean! = empty

begin, end, rbegin, rend

push\_back, pop\_back, emplace\_back, front, back

insert

size, resize, capacity, reserve, swap hack / shrink\_to\_fit

vector <bool>

**<string>**

string (10, ''), std :: string ("a") + "b"

length / size,

find, rfind, find \_ \* \_ of, string :: npos

<sstream>: stringstream ss ("str"), ss.str ()

**<cctype>**

isalpha, isalnum, isblank, isdigit, islower, isupper, isxdigit

tolower, toupper, use together with transform

**<deque>**

**<queue>:** priority\_queue

**<tuple>:** pair, make\_pair, .first / .second; tuple, make\_tuple, get <#> ();

**Lexicographic comparison**

**<map>, <set>**

map, key sorting

[key] = vs at, for (auto kv: mapa) {}

count, erase

set, insert

**<unordered\_set>, <unordered\_map>**

std :: hash <T>: : operator ()

**<algorithm>**

min, max, minmax, max\_element,

sort, predicate with tie, stable\_sort, is\_sorted

sort / iota + next\_permutation

unique / remove / remove\_if+ .erase

reverse

fill, copy, copy\_n, <iterator>: back\_inserter, istream\_iterator

most vexing parse

find, find\_if, count, count\_if

search

includes, set\_union ,set\_intersection, set\_difference, set\_symmetric\_difference

lower\_bound / upper\_bound

**<iterator>:** begin (cont), end (cont), size (cont)

**<numeric>:** accumulate, partial\_sum, iota

**<cmath>**

hypot, atan2, pi = atan (1) \* 4

round, floor, ceil

abs

**<complex>**

**<limits>:** numeric\_limits<int> :: max ()

**<functional>**

Compiler-specific: \_\_builtin\_popcount, \_\_builtin\_clz, \_\_builtin\_ctz, \_\_gcd, \_\_int128

**stringstreamss("5\na b c");**

int n;

string s;

ss >> n;

getline(ss, s);

getline(ss, s);

cout<< n <<" : "<< s <<endl;

**<iomanip>:** fixed, setprecision, setw, setfill, hex / dec

**A pair of setw and setfill allows you to output a string or number to a specified number of positions, possibly filling them with something non-whitespace.**

cout<<setw(8) <<"hello"<<endl; // " hello"

cout<<setfill('0') <<setw(3) <<7<<endl; // "007"

**Manipulators hex and dec allow you to display integers in hexadecimal or decimal notation.**

**int n;**

stringstream("2A") >> hex >> n;

cout<<dec<< n <<endl; // 42

cout<< hex <<42<<endl; // 2a

**<iomanip>:** noskipws / skipws

**cin / cout.tie (0); cin / cout.sync\_with\_stdio (false); endl vs '\ n'**

**I / O streams were designed to intensively expand and modify their state. Overhead caused by this can lead to TL. In such cases, view magic may help.**

cin.tie(0);

cin.sync\_with\_stdio(false);

<cstdio>: printf / scanf for bold inputs

**<vector>**

begin, end, rbegin, rend

for (auto it = v.begin(); it != v.end(); ++it) cout<< \*it <<endl;

The container can also be bypassed in the reverse order.

for (auto it = v.rbegin(); it != v.rend(); ++it) cout<< \*it <<endl;

push\_back, pop\_back, emplace\_back, front, back

vector<pair<int, string>> v;

v.swap(vector<int>(v));

cout<<v.size() <<" "<<v.capacity() <<endl; // 5 5

The .shrink\_to\_fit () method introduced in C ++ 11 has the same effect .

**<String>**

length / size, substr

Substring (start, length):

cout<<t.substr(18, 3) <<endl;

push\_back

Like vector, string has methods, push\_back ,pop\_back , front , back , insert .

to\_string, stoi

Numeric types can be converted to the string to\_string and vice versa with the family stoi / stoll / stoull / stof / stod .

double pi = 3.14159265;

string s = "Pi = " + to\_string(pi);

cout<< s <<endl; // Pi = 3.141593

s = "12345678901";

//cout<<stoi(s) <<endl; // Exception: out\_of\_range

cout<<stoll(s) <<endl; // 12345678901

find, rfind, find \_ \* \_ of, string :: npos

<sstream>: stringstream ss ("str"), ss.str ()

The input / output string stream is similar to cin / cout , but it does not work with stdin / stdout, but with a string.

stringstream ss;

ss <<2<<" "<<4<<" "<<8;

string s = ss.str();

cout<< s <<endl; // "2 4 8"

stringstreamss("1 2 3");

int n1, n2, n3;

ss >> n1 >> n2 >> n3;

cout<< n1 <<" "<< n2 <<" "<< n3 <<endl; // 1 2 3

**<cctype>**

isalpha, isalnum, isblank, isdigit, islower, isupper, isxdigit

In the JV, these character classes are most often useful, a list of 0-127:

isalpha - A-Za-z

isalnum - 0-9A-Za-z

isblank - \ t and space

isdigit - 0-9

islower - az

isupper - AZ

isxdigit - 0-9A-Fa-f

tolower, toupper, use with transform

transform(s.begin(), s.end(), s.begin(), toupper);

cout<< s <<endl; // WHERE IS STL?!

**sort / iota + next\_permutation**

vector<int>v(4);

iota(v.begin(), v.end(), 1);

do {

cout<<v[0] << v[1] << v[2] << v[3] <<endl;

} while (next\_permutation(v.begin(), v.end()));

**unique / remove / remove\_if+ .erase**

vector<int> v = {1, 1, 2, 3, 3, 3, 1, 1, 2, 2};

sort(v.begin(), v.end());

v.erase(unique(v.begin(), v.end()), v.end()); // 1 2 3

v.erase(remove\_if(v.begin(), v.end(), [](int x) {

return x % 2 == 0;

}), v.end()); // 1 3

reverse

Invert elements of a sequential container:

string s = "desserts";

reverse(s.begin(), s.end());

cout<< s <<endl; // stressed

**The search algorithm extends string :: find to arbitrary sequences.**

vector<int>v{3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5, 8, 9, 7, 9, 3, 2, 3, 8, 4, 6};

auto seq = {3, 5};

cout<< search(v.begin(), v.end(), seq.begin(), seq.end()) - v.begin() <<endl; // 9

**<numeric>: accumulate, partial\_sum, iota**

The sum of the segment specified by a pair of iterators. The type of the return value is determined by the initial value of the sum (sum vector <int64\_t> in 0LL ).

vector<int>v{1, 2, 3, 4};

cout<< accumulate(v.begin(), v.end(), 0) <<endl; // 10

**Prefix amounts:**

vector<int>v{1, 2, 3, 4, 5}, res(5);

partial\_sum(v.begin(), v.end(), res.begin()); // 1 3 6 10 15

Serial value generation (prefix ++ ):

vector<int>v(5);

iota(v.begin(), v.end(), -2); // -2 -1 0 1 2

**<complex>**

Complex numbers in SPs can reduce some 2D geometric calculations. For example, generating coordinate offsets on a square grid to 4 neighbors

complex<int>dir(1, 0);

for (inti = 0; i<4; ++i) {

cout<<dir<<" "<<dir.real() <<" + "<<dir.imag() <<"j"<<endl;

// (1,0) 1 + 0j; (0, 1) 0 + 1j; (-1, 0) -1 + 0j; (0, -1) 0 + -1j

dir \*= complex<int>(0, 1);

}

or area of a triangle.

stringstreamss("1 1\n1 4\n5 1");

double x, y;

ss >> x >> y; complex<double>a(x, y);

ss >> x >> y; complex<double>b(x, y);

ss >> x >> y; complex<double>c(x, y);

complex<double> A = b - a, B = c - a;

double S = abs((conj(A) \* B).imag()) / 2.0;

cout<< S <<endl; // 6

Do not forget that in wartime the value of the sine can reach four.

**<bitset>**

A known value bit value can be conveniently inferred using bitset .

cout<<bitset<10>(777) <<endl; // 1100001001

**BFS**

vector<int>adj[105];

int vis[105],dis[105];

int bfs(int s,int d){

vis[s] = 1;

dis[s]=0;

queue<int>Q;

Q.push(s);

while(!Q.empty()){

int u = Q.front();

cout<<u<<endl;

if(u==d) return dis[u];

Q.pop();

for(int i=0; i<adj[u].size(); i++){

if(!vis[adj[u][i]]){

vis[adj[u][i]] = 1;

dis[adj[u][i]]=dis[u]+1;

Q.push(adj[u][i]);

}

}

}

// cout<<endl;

}

**DFS**

void dfs(int s){

vis[s] = 1;

//dis[s]=0;

stack<int>Q;

Q.push(s);

while(!Q.empty()){

int u = Q.top();

cout<<u<<endl; //if(u==d) return dis[u];

Q.pop();

for(int i=0; i<adj[u].size(); i++){

if(!vis[adj[u][i]]){

vis[adj[u][i]] = 1;

//dis[adj[u][i]]=dis[u]+1;

Q.push(adj[u][i]);

}

}

}

}

**Divisor**

//inteligent prime factor

#include<bits/stdc++.h>

using namespace std;

int mark[105];

int main()

{

    int t,cas=0;

    cin>>t;

    while(t--){

        int x;

        cin>>x;

        int num=x,c=0,p=0;

        for(int j=x;j>1;j--){

            c=0;

            int n=j;

            if(n%2==0) p++;

            while(n%2==0){

                n=n/2;

                c++;

            }

            mark[2]+=c;

            for(int k=3;k<=sqrt(n);k++){

                    c=0;

                if(n%k==0){

                        p++;

                        while(n%k==0){

                            n=n/k;

                            c++;

                        }

                }

                mark[k]+=c;

            }

            if(n>1) {mark[n]+=1; p++;}

            //cout<<mark[3]<<" "<<j<<endl;

        }

       // cout<<p<<endl;

       cout<<"Case "<<++cas<<": "<<num<<" ="<<" "<<2<<" "<<"("<<mark[2]<<")";

        for(int m=3;m<105;m++){

            if(mark[m]!=0){

                cout<<" \*"<<" "<<m<<" "<<"("<<mark[m]<<")";

            }

        }

        cout<<endl;

        memset(mark,0,sizeof(mark));

    }

}

**Phi Algorithm**

#define MAXN 5000007

unsignedllphi[MAXN+1];

voidsieve()

{

    for(lli=2;i<=MAXN;i++)

        phi[i]=i;

    for(lli=2;i<=MAXN;i++)

    {

        if(phi[i]==i)

        {

            for(ll j=i; j<=MAXN; j+=i)

                phi[j]-=phi[j]/i;

        }

    }

}

intmain()

{

    ios\_base::sync\_with\_stdio(0);cin.tie(0);cout.tie(0);

    sieve();

    phi[0]=0;

    for(lli=1;i<MAXN;i++){

        phi[i]=phi[i-1]+(phi[i]\*phi[i]);

    }

    intt,m,n,cas=0;

    scanf("%d",&t);

    while(t--){

        scanf("%d %d",&n,&m);

        printf("Case %d: %llu**\n**",++cas,phi[m]-phi[n-1]);

    }

}

**Knapsack**

#include<bits/stdc++.h>

using namespace std;

long longdp[101][100001],n,w,value[9999],weight[9999];

long longans(long long pos, long long we){

   long longprofit,profit1;

   if(pos==n) return 0;

   if(dp[pos][we]!=-1) return dp[pos][we];

   if(we+weight[pos]<=w){

       profit=value[pos]+ans(pos+1,we+weight[pos]);

   }

   else profit=0;

   profit1=ans(pos+1,we);

   return dp[pos][we]=max(profit,profit1);

}

int main()

{

   memset(dp,-1,sizeof(dp));  cin>>n>>w;

   for(int i=0;i<n;i++){

       cin>>weight[i]>>value[i];

   }

   cout<<ans(0,0)<<endl;

}

**Big Mod**

long longbigmod( longlong 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;

}

[**Trailing zero in factorial**](http://alavolacodes.blogspot.com/2013/06/trailing-zero-in-factorial.html)

#include <stdio.h>

int main()

{

    int n,total,deno;

        scanf("%d",&n);

    total=0;

    deno=5;

    while(deno<=n)

    {

        total+=n/deno;

        deno\*=5;

    }

    printf("%d\n",total);

    return 0;

}

[**Sum of Divisors of a Number**](http://alavolacodes.blogspot.com/2013/06/sum-of-divisors-of-number.html)

#define i64 long long

i64 power(i64 N,i64 P)

{

    i64 sum=1,i;

    if(P==0)return 1;

    else

    {

        for(i=1;i<=P;i++)

            sum=sum\*N;

        return sum;

    }

}

i64 sumofdivisor(i64 n)

{

    i64 sum=1,i,count;

    i64 sq=(i64)sqrt(n);

    for(i=0;prime[i]<=sq;i++)

    {

        count=0;

        while(n%prime[i]==0)

        {

            count++;

            n/=prime[i];

        }

        sum\*=(i64)(power(prime[i],count+1)-1)/(prime[i]-1);

    }

    if(n>1)

        sum\*=(n+1);

    return sum;

}

**Number Of Divisors**

int number\_of\_divisor(int num)

{

 int j,count,div=1;

 for(j=0;prime[j]<=sqrt(num);j++) //prime array holds the prime numbers

 {

  count=0;

  while(num%prime[j]==0)

  {

   count++;

   num/=prime[j];

  }

  div\*=(count+1);

 }

 if(num>1)

  div<<=1;

 return div;

}

**Factorial of a Large Number**

#include<iostream>

using namespace std;

// Maximum number of digits in output

#define MAX 500

int multiply(int x, int res[], int res\_size);

// This function finds factorial of large numbers

// and prints them

void factorial(int n)

{

    int res[MAX];

    // Initialize result

    res[0] = 1;

    int res\_size = 1;

    // Apply simple factorial formula n! = 1 \* 2 \* 3 \* 4...\*n

    for (int x=2; x<=n; x++)

        res\_size = multiply(x, res, res\_size);

    cout<< "Factorial of given number is \n";

    for (int i=res\_size-1; i>=0; i--)

        cout<< res[i];

}

// This function multiplies x with the number

// represented by res[].

// res\_size is size of res[] or number of digits in the

// number represented by res[]. This function uses simple

// school mathematics for multiplication.

// This function may value of res\_size and returns the

// new value of res\_size

int multiply(int x, int res[], int res\_size)

{

    int carry = 0; // Initialize carry

    // One by one multiply n with individual digits of res[]

    for (int i=0; i<res\_size; i++)

    {

        int prod = res[i] \* x + carry;

        // Store last digit of 'prod' in res[]

        res[i] = prod % 10;

        // Put rest in carry

        carry = prod/10;

    }

    // Put carry in res and increase result size

    while (carry)

    {

        res[res\_size] = carry%10;

        carry = carry/10;

        res\_size++;

    }

    return res\_size;

}

// Driver program

int main()

{

    factorial(100);

    return 0;

}

**Binary Exponentiation**

llbinpow(ll base, ll power, ll mod){

    base=base%mod;

    ll res=1;

    while(power>0){

        if(power&1) res=(res\*base)%mod;

        base=(base\*base)%mod;

        power>>=1;

    }

    return res;

}

**Extended GCD**

llext\_gcd(lla,llb,ll&x,ll&y){

    if(a==0){

        x=0;

        y=1;

        return b;

    }

    ll x1,y1;

    ll d=ext\_gcd(b%a,a,x1,y1);

    x=y1-(b/a)\*x1;

    y=x1;

    return d;

}

**Inverse Mod**

llmodInv (lla,ll m){

    llx,y;

    ext\_gcd(a,m,x,y);

    x%=m;

    if(x<0) x+=m;

    return x;

}

**Finding frequency of a number**

//globally

unordered\_map< int, vector<int>> store;

int findFrequency(int arr[],int n,intleft,intright,int element)

{

    int a = lower\_bound(store[element].begin(),store[element].end(),left)- store[element].begin();

    int b = upper\_bound(store[element].begin(),store[element].end(),right)- store[element].begin();

    return b-a;

}

//inside main function

for(int i=0;i<n;++i)store[arr[i]].push\_back(i);

cout<<findFrequency(arr, n, b, p, v[0]) <<endl;

store.clear();

**Find the smallest window in a string containing all characters of another string**

const int no\_of\_chars = 256;

string findSubString(string str, string pat)

{

    int len1 = str.length();

    int len2 = pat.length();

    if (len1 < len2)  cout<< "No such window exists";  return "";

    int hash\_pat[no\_of\_chars] = {0};

    int hash\_str[no\_of\_chars] = {0};

    for (int i = 0; i< len2; i++)  hash\_pat[pat[i]]++;

    int start = 0, start\_index = -1, min\_len = INT\_MAX;

    int count = 0; // count of characters

    for (int j = 0; j < len1 ;j++)  {

       hash\_str[str[j]]++;

       if (hash\_pat[str[j]] != 0 &&

           hash\_str[str[j]] <= hash\_pat[str[j]] )

           count++;

       if (count == len2){.

           while ( hash\_str[str[start]] >hash\_pat[str[start]] || hash\_pat[str[start]] == 0){

               if (hash\_str[str[start]] >hash\_pat[str[start]])

                   hash\_str[str[start]]--;

               start++;

           }

           int len\_window = j - start + 1;

           if (min\_len>len\_window) {

               min\_len = len\_window; start\_index = start;

           }

       }

    }

    if (start\_index == -1)cout<< "No such window exists"; return "";

    }

return str.substr(start\_index, min\_len);

}

// Driver code

int main()

{

    string str = "this is a test string";

    string pat = "tist";

    cout<< "Smallest window is : \n"

       <<findSubString(str, pat);

    return 0;

}

**Smallest number with at least n trailing zeros in factorial**

#include<bits/stdc++.h>

using namespace std;

bool check(int p, int n){

    int temp = p, count = 0, f = 5;

    while (f <= temp){

       count += temp/f;

       f = f\*5;

    }

    return (count >= n);

}

int findNum(int n){

    if (n==1) return 5;.

    int low = 0;

    int high = 5\*n;

    while (low <high){

       int mid = (low + high) >> 1;

       if (check(mid, n)) high = mid;

       else  low = mid+1;

    }

    return low;

}

int main()

{

    int n = 6;

    cout<<findNum(n) <<endl;

    return 0;

}

**Prime Factorization using Sieve O(log n) for multiple queries**.

#include "bits/stdc++.h"

using namespace std;

#define MAXN 100001

int spf[MAXN];

void sieve(){

    spf[1] = 1;

    for (int i=2; i<MAXN; i++)  spf[i] = i;

    for (int i=4; i<MAXN; i+=2) spf[i] = 2;

    for (int i=3; i\*i<MAXN; i++){

       if (spf[i] == i)   {

           for (int j=i\*i; j<MAXN; j+=i)

               if (spf[j]==j)

                   spf[j] = i;

       }

    }

}

vector<int>getFactorization(int x){

    vector<int> ret;

    while (x != 1){

       ret.push\_back(spf[x]);

       x = x / spf[x];

    }

    return ret;

}

int main(int argc, char const \*argv[])

{

    sieve();

    int x = 12246;

    cout<< "prime factorization for " << x <<" : ";

    vector <int> p = getFactorization(x);

    for (int i=0; i<p.size(); i++)

       cout<< p[i] << " ";

    cout<<endl;

    return 0;

}

**Inverse Factorial / nCr of two number**

#include<bits/stdc++.h>

using namespace std;

const int maxn = 1022, mod = 1e9 + 7;

int fac[maxn];

int mod\_exp(int n, int p)

{

      if(p==-1) p=mod-2;

      int ret = 1;

      while(p){

        if(p&1) ret = (ret\*1LL\*n)%mod;

        p>>=1;

        n=(n\*1LL\*n)%mod;

    }

    return ret;

}

void init()

{

    fac[0] = 1;

    for(int i=1;i<maxn;i++) fac[i] = (fac[i-1]\*1LL\*i)%mod;

}

int nCr(int n, int r)

{

    return ((fac[n]\*1LL\*mod\_exp(fac[r], -1))%mod)\*mod\_exp(fac[n-r], -1)%mod;

}

int main()

{

    int n, m;

    init();

    cin>>n>>m;

    cout<<nCr(n+2\*m-1, n-1);

    return 0;

}

**count of divisor of a factorial (also ncr)**

//first we have to find the prime factorization count of that number

//we have to found out count after ncr calculation

//then add 1 with the count

//calculate if need more

#include<stdio.h>

#define ll long long

int prime[90], range=0;

int vis[500];

void inti(){

    int i,j;

    for(i=2; i\*i<=431; i++){

        if(vis[i]==0){

            for(j=i\*i;j<=431;j+=i){

                vis[j]=1;

            }

        }

    }

    for(i=2;i<=431;i++){

        if(vis[i]==0){

            //printf("%d\n", i);

            prime[range++]=i;

        }

    }

}

int cal(int n, int p){

    if(n<p) return 0;

    else return(n/p + cal(n/p, p));

}

int main(){

    //inti();

    int n,k,i;

    \_\_int64 result =1,pre;

    inti();

    //printf("%d\n", range);

    while(scanf("%d%d",&n,&k)==2){

        result=1;

        if(k==0||k==n){

            printf("1\n"); continue;

        }

        for(i=0;i<range&&prime[i]<=n;i++){

            pre=cal(n,prime[i]) - cal(n-k,prime[i]) - cal(k,prime[i]);

            result = result \* (pre+1);

        }

        printf("%I64d\n", result);

    }

    return 0;

}

**How many pair gcd greater than 1**

#include<bits/stdc++.h>

#define ll long long

#define maxx 100009

using namespace std;

ll phi[maxx];

ll mark[maxx];

llcu\_phi[maxx];

void phi\_sum(){

    cu\_phi[1]=1;

    cu\_phi[2]=1;

    for(int i=2;i<=maxx;i++) {cu\_phi[i]=phi[i]+cu\_phi[i-1];}

}

void sieve\_phi(){

    for(int i=1;i<=maxx;i++) phi[i]=i;

    phi[1]=1;

    mark[1]=1;

    for(int i=2;i<=maxx;i++){

        if(!mark[i]){

            for(int j=i;j<=maxx;j+=i){

                mark[j]=1;

                phi[j]=phi[j] / i\*(i-1);

            }

        }

    }

}

int main()

{

    sieve\_phi(); phi\_sum();

    int t,cas=0;   cin>>t;

    while(t--){

        ll x; cin>>x;

        cout<<"Case "<<++cas<<": "<<((x\*(x+1))/2)-cu\_phi[x]<<endl;

    }

}

**Divisor of a number using Prime Factorization (NOD)**

int main()

{

    prime\_number();

    int test\_case,case\_no=0;

    scanf("%d",&test\_case);

    while(test\_case--){

       long long number; cin>>number;

       int counting,total=1;

       for(int i=0;i<storage.size() and storage[i]<=sqrt(number);i++){

           counting=0;

           if(number<storage[i]) break;

           while(number%storage[i]==0){

               number=number/storage[i];

               counting++;

           }

           total=total\*(counting+1);

       }

       if(number>1) total=total\*2;

       printf("Case %d: %d\n",++case\_no,total-1);

    }

}

**Formula of Sum of Divisor of a Number (SOD)**

SOD = (((prime1)^(prime1\_power+1))-1)/(prime1-1) \* (((prime2)^(prime2\_power+1))-1)/(prime2-1) \* ……\* (((primeN)^(primeN\_power+1))-1)/(primeN-1)

N = last prime number that divides the number.

**Trie(Prefix cheaking)**

int res=1;

structtrie{

    int stop;

    trie\*next[4];

    intcnt[4];

**trie(){**

        stop =0;

        for(inti=0;i<4;i++){

            next[i]=**NULL**;

        }

**for(inti=0;i<4;i++)cnt[i]=0;**

    }

}\*root;

boolinsert(string s, trie\*cur){

**int l=s.length();** bool test =0;

    for(inti=0;i<l;i++){

*//        int now = s[i]-'A';*

*//        if(cur->next[now]==NULL){*

***//            cur->next[now]=new trie();***

*//        }*

*//        if(cur->stop){*

*//            test=1;*

*//            break;*

***//        }***

        int point;

        if(s[i]=='A') point=0;

        elseif(s[i]=='C') point=1;

        if(s[i]=='G') point=2;

**if(s[i]=='T') point=3;**

        cur->cnt[point]++;

        if(cur->next[point]==**NULL**){

            cur->next[point]=newtrie();

        }

**res=max(res, cur->cnt[point]\*(i+1));**

        cur=cur->next[point];

    }

    cur->stop=1;

    return test;

**}**

voiddel(trie\*cur){

    for(inti=0;i<4;i++){

        if(cur->next[i]) **del(cur->next[i]);**

    }

    for(inti=0;i<4;i++)if(cur->cnt[i]) cur->cnt[i]=0;

    delete(cur);

**}**

**template**<**typename** Container>

boolLexCompare(const Container& a, const Container& b){

    returnstd::lexicographical\_compare(a.begin(), a.end(),

                                        b.begin(), b.end());

**}**

**template**<**typename**ContainerIterator>

voidsort\_by\_lexicographical\_comapre(ContainerIterator beg,

                                     ContainerIterator end)

**{**

    std::sort(beg, end, LexCompare<**typename**ContainerIterator::value\_type>);

}

**intmain()**

{

    intt,cas=0;

    cin>>t;

    while(t--){

**res=1;**

        root =new(trie);

        string s;

        int x;

        cin>>x;

**getchar();**

        std::vector<string> v;

        bool test =0;

        for(inti=0;i<x;i++){

            cin>>s;

**v.push\_back(s);**

        }

*//sort(v.begin(),v.end(),);*

         sort\_by\_lexicographical\_comapre(v.begin(), v.end());

        for(inti=0;i<x;i++){

**test = insert(v[i],root);**

            if(test)break;

        }

        cout<<"Case "<<++cas<<": "<<res<<endl;

        del(root);

**}**

}

**Smallest subarray with k distinct numbers**

//n= size of array

//k= size of the length/unique element number/the size where you want to find most of the unique number

void minRange(long long arr[], long long n, long long k){

    ll l = 0, r = n;

    for (lli = 0; i< n; i++) {

        unordered\_set<ll> s;

        ll j;

        for (j = i; j < n; j++) {

            s.insert(arr[j]);

            if (s.size() == k) {

                if ((j - i) < (r - l)) {

                    r = j;

                    l = i;

                }

                break;

            }

        }

        if (j == n)

            break;

    }

        cout<< l << " " << r<<endl;

}

int main()

{

    int t,cas=0;

    cin>>t;

    while(t--){

        int x; cin>>x;

        long longarr[x];

        map<long long , long long> m;

        long long k=0;

        for(int i=0;i<x;i++){

            cin>>arr[i];

                if(m[arr[i]]==0){

                    k++;

                    m[arr[i]]=1;

              }

        }

        cout<<"Case "<<++cas<<": ";

        minRange(arr, x, k);

    }

    return 0;

}

///\*\*\*\*\*\* Very large number modular \*\*\*\*\*\*///

unsigned ll aModM(string s, unsigned ll mod){

unsigned ll number = 0;

for (unsigned ll i = 0; i < s.length(); i++) {

number = (number\*10 + (s[i] - '0'));

number %= mod; }

return number;

}

**///\*\*\*\*\*\* Inverse Modular \*\*\*\*///**

int modInverse(int a, int m){

a = a%m;

for (int x=1; x<m; x++)

if ((a\*x) % m == 1) return x;

}

**/// rhombus area, perimeter, diagonal**

area = (d1 \* d2) / 2;

perimeter = 2 \* sqrt(pow(d1, 2) + pow(d2, 2));///given 2 diagonals

**///Trapezoid**

area = (b1+b2)\*h///b are base

**///Cylinder**

area = 2\*pi\*r\*h

volume = pi\*r\*r\*h

**///Area of cycle inside a rhombus input 2 diagonals a b**

float circlearea(float a, float b) {

if (a < 0 || b < 0) return -1;

float A = (3.14 \* pow(a, 2) \* pow(b, 2))

/ (4 \* (pow(a, 2) + pow(b, 2)));

return A;

}

**///Area of cycle inside a rectangle**

float area(float r) {

if (r < 0) return -1;

float area = 3.14 \* pow(r / (2 \* sqrt(2)), 2);

return area;

}

**///area-of-decagon(10 side)-inscribed-within-the-circle/**

float area(float r) {

if (r < 0) return -1;

float area = (5 \* pow(r, 2) \* (3 - sqrt(5))

\* (sqrt(5) + (2 \* sqrt(5)))) / 4;

return area;

}

**///area-of-the-largest-triangle-inscribed-in-a-hexagon**

float trianglearea(float a) {

if (a < 0) return -1;

float area = (3 \* sqrt(3) \* pow(a, 2)) / 4;

return area;

}

**///largest-triangle-that-can-be-inscribed-in-an-ellipse/**

float trianglearea(float a, float b) {

if (a < 0 || b < 0) return -1;

float area = (3 \* sqrt(3) \* pow(a, 2)) / (4 \* b);

return area;

}

**///convert from a decimal to any base (up to base 20)**

char reVal(int num){

if (num >= 0 && num <= 9) return (char)(num + '0');

else return (char)(num - 10 + 'A');

}

void strev(char \*str){

int len = strlen(str); int i;

for (i = 0; i < len/2; i++) {

char temp = str[i];

str[i] = str[len-i-1];

str[len-i-1] = temp;

}

}

char\* fromDeci(int base, int inputNum){///call this function

int index = 0;char res[100000];

while (inputNum > 0) {

res[index++] = reVal(inputNum % base);

inputNum /= base;

}

res[index] = '\0';strev(res);

return res;

}

**///convert any number to decimal**

int val(char c){

if (c >= '0' && c <= '9')return (int)c - '0';

else return (int)c - 'A' + 10;

}

int toDeci(char \*str, int base){///call this method

int len = strlen(str);

int power = 1,num = 0,i;

for (i = len - 1; i >= 0; i--){

if (val(str[i]) >= base) {

printf("Invalid Number");return -1;}

num += val(str[i]) \* power;

power = power \* base;

}

return num;

}

**///segment sieve**

void segSieve(ll m, ll n){

bool isPrime[n-m+1];

for(int i = 0; i < n-m+1; ++i) isPrime[i] = true;

for(int i = 0; v[i] \* v[i] <= n; ++i){

int currentPrime = v[i];

ll b = (m / currentPrime) \* currentPrime;

if(b < m) b += currentPrime;

for(ll j = b; j <= n; j+=currentPrime)

isPrime[j-m] = false;

if(b == currentPrime) isPrime[b-m] = true;

}

for(int i = 0; i < n-m+1; ++i){

if (isPrime[i]){

ll x = i + m;

if(x == 1) continue;

cout << x <<endl;

}

}

}

**///Big Number Factorial**

**///very big factorial**

void BigFactorial(int n){

int a[200];

int i,j,temp,m,x;

a[0]=1;

m=1;

temp = 0;

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

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

x = a[j]\*i+temp;

a[j]=x%10;

temp = x/10;

}

while(temp>0) {

a[m]=temp%10;

temp = temp/10;

m++;

}

}

for(i=m-1;i>=0;i--)

cout << a[i];

cout << endl;

}

**///is a number power of 2**

bool isPowerOfTwo (int x) {

return x && (!(x&(x-1)));

}

///count no. of digit

int x = floor(log10(n)) + 1;

///divide x by 2^i

ll y = x >> i;

///multiple x by 2^i

ll y = x << i;

/// find remainder of x with 2^i

ll y = x & ((1 << i) - 1);

**///Maximum x that is power of 2 and divides n**

ll x = (n & -n);

**///set operation**

multiset <int, greater <int> > s;///store value descending order

multiset <int, greater <int> > :: iterator itr;

for (itr = s.begin(); itr != s.end(); ++itr){ cout<<\*itr;}

**/// assigning the elements from s to s1**

multiset <int> s1(s.begin(), s.end());//copy all the value of s in increasing order

int x = \*s.lower\_bound(value);

**///priority queue with pair**

priority\_queue<pair<int, int> > pq;

**///de-queue**

deque <int> dq;

dq.push\_back(10);

dq.push\_front(20);

dq.pop\_front();

dq.pop\_back();

///count 1 is there in the multimap

cout << "1 exists " << mp.count(1) << endl;

**///vector v[0 - 9] = {0,0,0,0,0,0......}**

vector<int> v(10, 0);

**///map iterator...**

map<int, int> gquiz2(gquiz1.begin(), gquiz1.end());

**///Strings Tricks//**

str1.erase(0, str1.find\_first\_not\_of('0')); ///F For Erasing Beginning 0

/// Deletes 4 characters from index number 1

str.erase(1, 4);

/// Deletes all characters except last 6 characters

str.erase(str.begin() + 0, str.end() - 6);

**///count-distinct-subsequences**

const int MAX\_CHAR = 256;

int countSub(string str){

vector<int> last(MAX\_CHAR, -1);

int n = str.length();

int dp[n+1];

dp[0] = 1;

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

dp[i] = 2\*dp[i-1];

if (last[str[i-1]] != -1)

dp[i] = dp[i] - dp[last[str[i-1]]];

last[str[i-1]] = (i-1);

}

return dp[n];

}

///pascal triangle

vector<ll>v;

void printPascal(int n){

for (int line = 1; line <= n; line++) {

ll C = 1;

for (int i = 1; i <= line; i++) {

v.pb(C);

C = C \* (line - i) / i;

}

}

}

**///count-numbers-in-range-1-to-n-which-are-divisible-by-x-but-not-by-y**

ll countNumbers(ll X, ll Y, ll N) {

ll divisibleByX = N / X;

ll divisibleByY = N / Y;

ll LCM = (X \* Y) / \_\_gcd(X, Y);

ll divisibleByLCM = N / LCM;

ll divisibleByXorY = divisibleByX + divisibleByY - divisibleByLCM;

ll divisibleByXnotY = divisibleByXorY - divisibleByY;

return divisibleByXnotY;

}

**///count-unique-subsequences-of-length-k/**

int solution(vector<int>& A, int k){

const int N = A.size();

if (N < k || N < 1 || k < 1)return 0;

if (N == k) return 1;

vector<int> v1(N, 0);

vector<int> v2(N, 0);

vector<int> v3(N, 0);

v2[N - 1] = 1;

v3[A[N - 1] - 1] = 1;

for (int i = N - 2; i >= 0; i--) {

v2[i] = v2[i + 1];

if (v3[A[i] - 1] == 0) { v2[i]++;

v3[A[i] - 1] = 1;

}

}

for (int j = 1; j < k; j++) {

fill(v3.begin(), v3.end(), 0);

v1[N - 1] = 0;

for (int i = N - 2; i >= 0; i--) {

v1[i] = v1[i + 1];

v1[i] = v1[i] + v2[i + 1];

v1[i] = v1[i] - v3[A[i] - 1];

v3[A[i] - 1] = v2[i + 1];

}

v2 = v1;

}

return v1[0];

}

void solve(int a[], int n, int k){

vector<int> v;

v.assign(a, a + n);

cout << solution(v, k);

}

int main(){

int a[] = { 1,1,1,1 ,2,3};

int n = sizeof(a) / sizeof(a[0]);

int k = 3;

solve(a, n, k);

return 0;

}

**///longest-common-increasing-subsequence from 2 arr and size**

int LCIS(int arr1[], int n, int arr2[], int m) {

int table[m], parent[m];

for (int j=0; j<m; j++) table[j] = 0;

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

int current = 0, last = -1;

for (int j=0; j<m; j++) {

if (arr1[i] == arr2[j]) {

if (current + 1 > table[j]) {

table[j] = current + 1;

parent[j] = last;

}

}

if (arr1[i] > arr2[j]) {

if (table[j] > current) {

current = table[j];

last = j;

}

}

}

}

int result = 0, index = -1;

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

if (table[i] > result) {

result = table[i];

index = i;

}

}

int lcis[result];

for (int i=0; index != -1; i++) {

lcis[i] = arr2[index];

index = parent[index];

} cout << "The LCIS is : ";

for (int i=result-1; i>=0; i--) printf ("%d ", lcis[i]);

return result;

}

int main() {

int arr1[] = {3, 4, 9, 1};

int arr2[] = {5, 3, 8, 9, 10, 2, 1};

int n = sizeof(arr1)/sizeof(arr1[0]);

int m = sizeof(arr2)/sizeof(arr2[0]);

cout << "\nLength of LCIS is "<< LCIS(arr1, n, arr2, m);

return (0);

}

**///FIND DAY 0 = SUNDAY, 1 = MONDAY, 2 = TUESDAY, 3 = WEDNESDAY, 4 = THURSDAY, 5 = FRIDAY, 6 = SATARDAY**

int dayofweek(int d, int m, int y){

static int t[] = { 0, 3, 2, 5, 0, 3, 5, 1, 4, 6, 2, 4 };

y -= m < 3;

return ( y + y/4 - y/100 + y/400 + t[m-1] + d) % 7;

}

int main(){

int day = dayofweek(11, 8, 2000);

printf ("%d", day);

return 0;

}

**///common prime factor in a and b**

#define MAXN 100001

bool prime[MAXN];

void SieveOfEratosthenes(){

memset(prime, true, sizeof(prime));

prime[0] = false;

prime[1] = false;

for (int p = 2; p \* p <= MAXN; p++) {

if (prime[p] == true) {

for (int i = p \* p; i <= MAXN; i += p)

prime[i] = false;

}

}

}

void common\_prime(int a, int b){

int gcd = \_\_gcd(a, b);

for (int i = 2; i <= (gcd); i++) {

if (prime[i] && gcd % i == 0) {

cout << i << " ";

}

}

}

**///way of choosing coin**

int count( int S[], int m, int n ){///arr ,len, sum

int i, j, x, y;

int table[n + 1][m];

for (i = 0; i < m; i++)

table[0][i] = 1;

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

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

x = (i-S[j] >= 0) ? table[i - S[j]][j] : 0;

y = (j >= 1) ? table[i][j - 1] : 0;

table[i][j] = x + y;

}

}

return table[n][m - 1];

}

**///KMP pattern search**

void computeLPSArray(char\* pat, int M, int\* lps);

void KMPSearch(char\* pat, char\* txt) {

int M = strlen(pat);

int N = strlen(txt);

int lps[M];

computeLPSArray(pat, M, lps);

int i = 0; // index for txt[]

int j = 0; // index for pat[]

while (i < N) {

if (pat[j] == txt[i]) {

j++; i++;

}

if (j == M) {

printf("Found pattern at index %d ", i - j);

j = lps[j - 1];

}

else if (i < N && pat[j] != txt[i]) {

if (j != 0) j = lps[j - 1];

else i = i + 1;

}

}

}

void computeLPSArray(char\* pat, int M, int\* lps) {

int len = 0;

lps[0] = 0;

int i = 1;

while (i < M) {

if (pat[i] == pat[len]) {

len++; lps[i] = len; i++;

}

else {

if (len != 0) len = lps[len - 1];

else lps[i] = 0; i++;

}

}

}

**///minimum cost**

#define R 3

#define C 3

int min(int x, int y, int z);

int minCost(int cost[R][C], int m, int n){

int i, j;

int tc[R][C];

tc[0][0] = cost[0][0];

for (i = 1; i <= m; i++)

tc[i][0] = tc[i-1][0] + cost[i][0];

for (j = 1; j <= n; j++)

tc[0][j] = tc[0][j-1] + cost[0][j];

for (i = 1; i <= m; i++)

for (j = 1; j <= n; j++)

tc[i][j] = min(tc[i-1][j-1],

tc[i-1][j],

2tc[i][j-1]) + cost[i][j];

return tc[m][n];

}

int min(int x, int y, int z){

if (x < y)

return (x < z)? x : z;

else

return (y < z)? y : z;

}

int main(){

int cost[R][C] = { {1, 2, 3},

{4, 8, 2},

{1, 5, 3} };

printf(" %d ", minCost(cost, 0, 2));

return 0;

}

**int main(){**

**ios\_base::sync\_with\_stdio(false);**

**cin.tie(NULL);**

**//freopen("testcases.txt","r",stdin);**

**return 0;**

**}**

**///Double value to string**

string cast(double x){

ostringstream str1;

str1 << x;

string geek = str1.str();

return geek;

}

**///\*String to Decimal**

atoi(str.c\_str());

**///\*Decimal to String**

int a = 10;

stringstream ss;

ss << a;

string str = ss.str();

**///\*Decimal to Binary**

string result = bitset<32>(number).to\_string();

cout << result.substr(result.find("1", 0));///print from 1 to last

**///\*Count set bits in an integer**

\_\_builtin\_popcount();

**/// Implementation for Range Query Update 1 value in a tree( Without Lazy )**

#define MAXN 100000

int ara[MAXN+7];

int tree[4\*MAXN+7];

void buildTree(int node,int l,int r) {

if (l==r) {tree[node]=ara[l];return;}

int mid=(l+r)/2;

buildTree(2\*node,l,mid);

buildTree(2\*node+1,mid+1,r);

tree[node]=tree[2\*node]+tree[2\*node+1];

}

int query(int node,int l,int r,int L,int R) {

if (L>r || R<l) return 0;

if (l>=L && r<=R) return tree[node];

int mid=(l+r)/2;

int ql=query(2\*node,l,mid,L,R);

int qr=query(2\*node+1,mid+1,r,L,R);

return ql+qr;

}

void update(int node,int l,int r,int idx,int val) {

if (idx>r || idx<l) return;

if (l==r) {tree[node]=val;return;}

int mid=(l+r)/2;

update(2\*node,l,mid,idx,val);

update(2\*node+1,mid+1,r,idx,val);

tree[node]=tree[2\*node]+tree[2\*node+1];

}

**///Lazy propagation update a range oldValue += newValue....**

#define mx 10000

#define ll long long

int arr[mx];

struct info{

ll prop, sum;

}tree[mx];

void update(int node, int b, int e, int i, int j, ll x){

if(i > e || j < b) return;

if(b >= i && e <= j){

tree[node].sum += ((e-b+1)\*x);

tree[node].prop += x;

return;

}

int left = node << 1;

int right = (node << 1) + 1;

int mid = (b+e) >> 1;

update(left,b,mid,i,j,x);

update(right,mid+1,e,i,j,x);

tree[node].sum = tree[node].sum + tree[node].sum + (e-b+1)\*tree[node].prop;

}

int query(int node, int b, int e, int i, int j, int c = 0){

if(i > e || j < b) return 0;

if(b >= i && e <= j) return (tree[node].sum + c \* (e - b + 1));

int left = node << 1;

int right = (node << 1) + 1;

int mid = (b+e) >> 1;

int p1 = query(left,b,mid,i,j,(c+tree[node].prop));

int p2 = query(right,mid+1,e,i,j,(c+tree[node].prop));

return p1+p2;

}

void init(int node, int b, int e){

if(b == e){tree[node].sum = arr[b]; return;}

int right = node\*2 + 1;

int left = node\*2;

int mid = (int)(b + e)/2;

init(left,b,mid);

init(right,mid+1,e);

tree[node].sum = tree[left].sum + tree[right].sum;

}

**///Fenwick Tree/Binary Indexed Tree/BIT/AddAndUpdate// return sum of 1 to index**

#define mx 10000

int ar[mx];

int tree[mx];

int read (int idx){

int sum = 0;

while(idx>0){

sum+=tree[idx];

idx-=(idx & - idx);

} return sum;

}

void update(int idx, int val, int n){

cout<<"update: ";

while(idx <= n){

cout<<idx<<" ";

tree[idx] += val;

idx += (idx & -idx);

}

}

int main()

{

int n; cin >> n;

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

cin >> ar[i];

update(i, ar[i], n);

}

///update(index, value, n);

return 0;

}

**///longest increasing subsequence**

int arr[100009];

int lis(int n){

int lis[n]; lis[0] = 1;

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

lis[i] = 1;

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

if ( arr[i] > arr[j] && lis[i] < lis[j] + 1)

lis[i] = lis[j] + 1;

}

return \*max\_element(lis, lis+n);

}

**/// edit-distance**

int min(int x, int y, int z){ // Utility function to find the minimum of three numbers

return min(min(x, y), z);

}

int editDistDP(string str1, string str2, int m, int n){///string1 string2 length1 length2

int dp[m+1][n+1]; // Create a table to store results of subproblems

for (int i=0; i<=m; i++) {// Fill d[][] in bottom up manner

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

if (i==0) dp[i][j] = j;

else if (j==0) dp[i][j] = i;

else if (str1[i-1] == str2[j-1]) dp[i][j] = dp[i-1][j-1];

else dp[i][j] = 1 + min(dp[i][j-1],dp[i-1][j],dp[i-1][j-1]);

}

} return dp[m][n];

}

**///count-number-of-ways-to-cover-a-distance**

int printCountDP(int dist){

int count[dist+1];

count[0] = 1, count[1] = 1, count[2] = 2;

for (int i=3; i<=dist; i++) count[i] = count[i-1] + count[i-2] + count[i-3];

return count[dist];

}

**/// sum of sub set / Returns true if there is a subset of set[] with sun equal to given sum**

bool isSubsetSum(int set[], int n, int sum){

bool subset[n+1][sum+1];

for (int i = 0; i <= n; i++) subset[i][0] = true;

for (int i = 1; i <= sum; i++) subset[0][i] = false;

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

for (int j = 1; j <= sum; j++) {

if(j<set[i-1]) subset[i][j] = subset[i-1][j];

if (j >= set[i-1])

subset[i][j] = subset[i-1][j] || subset[i - 1][j-set[i-1]];

}

}

/// uncomment this code to print table

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

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

printf ("%4d", subset[i][j]);

printf("\n");

}

return subset[n][sum];

}

/// Driver program to test above function

int main(){

int set[] = {3, 34, 4, 12, 4, 1 , 2};

int sum = 9;

int n = sizeof(set)/sizeof(set[0]);

if (isSubsetSum(set, n, sum) == true)

printf("Found a subset with given sum");

else

printf("No subset with given sum");

return 0;

}

**Dijkastra(Priority\_queue):**

void dijkastra(int start,int m)

{

priority\_queue<pair<int,int>, vector<pair<int,int>>, greater<pair<int,int>>>q;

q.push(make\_pair(0,start));

dis[start]=0;

while(!q.empty())

{

int current=q.top().second;

/\* if(current==m)

{

cout<<dis[m]<<endl; //for a perticular node

return;

}\*/

q.pop();

for(int i=0;i<adj[current].size();i++)

{

int u=adj[current][i];

if(dis[current]+edge[current][u]<dis[u])

{

dis[u]=dis[current]+edge[current][u];

q.push(make\_pair(dis[u],u));

}

}

}

}

**Prim’s Algorithm:**

vector<pii>v[10000];

int mark[100000];

int prim(int vtx)

{

int res=0;

priority\_queue<pair<int,int>, vector<pair<int,int>>, greater<pair<int,int>>>q;

q.push(make\_pair(0,vtx));

while(!q.empty())

{

pii x=q.top();q.pop();

if(!mark[x.second])

{

res+=x.first;

mark[x.second]=1;

for(auto u: v[x.second])

{

if(mark[u.second]==0)

{

q.push(make\_pair(u.first,u.second));

}

}

}

}

return res;

}

**Pattern Matching:**

bool check(string s, string sss)

{

for(int i=0,i<=s.size()-1;i++)

{

if(s[i]==sss[0])

{

if(sss.size()==1)

return true;

int k=i+1,j=1;

while(s[k]==sss[j])

{

if(j==sss.size()-1)

return true;

j++;k++;

}

}

}

return false;

}

**Euler path**

// A C++ program print Eulerian Trail in a given Eulerian or Semi-Eulerian Graph

#include <iostream>

#include <string.h>

#include <algorithm>

#include <list>

using namespace std;

// A class that represents an undirected graph

class Graph

{

int V; // No. of vertices

list<int> \*adj; // A dynamic array of adjacency lists

public:

// Constructor and destructor

Graph(int V) { this->V = V; adj = new list<int>[V];

} ~Graph() { delete [] adj; }

// functions to add and remove edge

void addEdge(int u, int v) { adj[u].push\_back(v); adj[v].push\_back(u); }

void rmvEdge(int u, int v);

// Methods to print Eulerian tour

void printEulerTour();

void printEulerUtil(int s);

// This function returns count of vertices reachable from v. It does DFS

int DFSCount(int v, bool visited[]);

// Utility function to check if edge u-v is a valid next edge in

// Eulerian trail or circuit

bool isValidNextEdge(int u, int v);

};

/\* The main function that print Eulerian Trail. It first finds an odd

degree vertex (if there is any) and then calls printEulerUtil()

to print the path \*/

void Graph::printEulerTour()

{

// Find a vertex with odd degree

int u = 0;

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

if (adj[i].size() & 1)

{ u = i; break; }

// Print tour starting from oddv

printEulerUtil(u);

cout << endl;

}

// Print Euler tour starting from vertex u

void Graph::printEulerUtil(int u)

{

// Recur for all the vertices adjacent to this vertex

list<int>::iterator i;

for (i = adj[u].begin(); i != adj[u].end(); ++i)

{

int v = \*i;

// If edge u-v is not removed and it's a a valid next edge

if (v != -1 && isValidNextEdge(u, v))

{

cout << u << "-" << v << " ";

rmvEdge(u, v);

printEulerUtil(v);

}

}

}

// The function to check if edge u-v can be considered as next edge in

// Euler Tout

bool Graph::isValidNextEdge(int u, int v)

{

// The edge u-v is valid in one of the following two cases:

// 1) If v is the only adjacent vertex of u

int count = 0; // To store count of adjacent vertices

list<int>::iterator i;

for (i = adj[u].begin(); i != adj[u].end(); ++i)

if (\*i != -1)

count++;

if (count == 1)

return true;

// 2) If there are multiple adjacents, then u-v is not a bridge

// Do following steps to check if u-v is a bridge

// 2.a) count of vertices reachable from u

bool visited[V];

memset(visited, false, V);

int count1 = DFSCount(u, visited);

// 2.b) Remove edge (u, v) and after removing the edge, count

// vertices reachable from u

rmvEdge(u, v);

memset(visited, false, V);

int count2 = DFSCount(u, visited);

// 2.c) Add the edge back to the graph

addEdge(u, v);

// 2.d) If count1 is greater, then edge (u, v) is a bridge

return (count1 > count2)? false: true;

}

// This function removes edge u-v from graph. It removes the edge by

// replacing adjcent vertex value with -1.

void Graph::rmvEdge(int u, int v)

{

// Find v in adjacency list of u and replace it with -1

list<int>::iterator iv = find(adj[u].begin(), adj[u].end(), v);

\*iv = -1;

// Find u in adjacency list of v and replace it with -1

list<int>::iterator iu = find(adj[v].begin(), adj[v].end(), u);

\*iu = -1;

}

// A DFS based function to count reachable vertices from v

int Graph::DFSCount(int v, bool visited[])

{

// Mark the current node as visited

visited[v] = true;

int count = 1;

// Recur for all vertices adjacent to this vertex

list<int>::iterator i;

for (i = adj[v].begin(); i != adj[v].end(); ++i)

if (\*i != -1 && !visited[\*i])

count += DFSCount(\*i, visited);

return count;

}

// Driver program to test above function

int main()

{

// Let us first create and test graphs shown in above figure

Graph g1(4);

g1.addEdge(0, 1);

g1.addEdge(0, 2);

g1.addEdge(1, 2);

g1.addEdge(2, 3);

g1.printEulerTour();

return 0;

}

**/\* Dynamic Programming C/C++ implementation of LCS problem \*/**

#include <bits/stdc++.h>

int max(int a, int b);

/\* Returns length of LCS for X[0..m-1], Y[0..n-1] \*/

int lcs(char\* X, char\* Y, int m, int n)

{

int L[m + 1][n + 1];

int i, j;

/\* Following steps build L[m+1][n+1] in bottom up fashion. Note

that L[i][j] contains length of LCS of X[0..i-1] and Y[0..j-1] \*/

for (i = 0; i <= m; i++) {

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

if (i == 0 || j == 0)

L[i][j] = 0;

else if (X[i - 1] == Y[j - 1])

L[i][j] = L[i - 1][j - 1] + 1;

else

L[i][j] = max(L[i - 1][j], L[i][j - 1]);

}

}

/\* L[m][n] contains length of LCS for X[0..n-1] and Y[0..m-1] \*/

return L[m][n];

}

/\* Utility function to get max of 2 integers \*/

int max(int a, int b)

{

return (a > b) ? a : b;

}

/\* Driver program to test above function \*/

int main()

{

char X[] = "AGGTAB";

char Y[] = "GXTXAYB";

int m = strlen(X);

int n = strlen(Y);

printf("Length of LCS is %d\n", lcs(X, Y, m, n));

return 0;

}

**//\*\*\*Hashing Implementation\*\*\*//**

ll base[maxx];

ll inversbase[maxx];

ll mod=293114467;

void basegenerate(ll bz)

{

base[0]=1;

inversbase[0]=1;

ll inv = InverseMod(bz,mod);

for(ll i=1; i<maxx; i++)

{

base[i]=(base[i-1]\*bz)%mod;

inversbase[i]=(inversbase[i-1]\*inv)%mod;

}

for(ll i=1; i<maxx; i++) inversbase[i]=(inversbase[i-1]+inversbase[i])%mod;

}

ll hashing(string s, ll (&encode)[maxx] )

{

ll l=s.size();

encode[0]=(s[0]\*base[0])%mod;

for(ll i=1; i<l; i++)

{

encode[i]=(encode[i-1]+((s[i]\*base[i])%mod))%mod;

}

}

ll hashValue(ll l, ll r, ll (&encode)[maxx])

{

return (encode[r]-encode[l-1])\*inversbase[l-1];

}

int main()

{

//freopen("testcases.txt","r",stdin);

ios\_base::sync\_with\_stdio(0); cin.tie(0); cout.tie(0);

basegenerate(1000);

string s;

while(cin>>s)

{

ll arr[maxx];

hashing(s, arr);

ll x,y; cin>>x>>y;

cout << hashValue(x, y, arr) << endl;

}

return 0;

}

**MATH**

**Rectangle:** Area= l\*w, Perimeter= 2( l+w )

**Parallelogram:**

Area=base\*height, Perimeter = 2\*(a+b)

**Triangle:**

Area= ½\*b\*h, Perimeter=a+b+c

**Trapezoid:**

Area= ½\*( (b1+b2)/2 )\*h,

Perimeter= a+b1+b2+c

**Circle:**

Circumference = πd=2πr, Area= π\*r\*r

**Rectangular solid:**

Volume= l\*w\*h,

Surface= 8\*(l\*l\*h\*h\*w\*w)

**Prism:**

Volume=b\*h

Surface=2\*b+P\*h

// b= area of base,p=perimeter of base

**Cylinder:**

Volume= π\*r\*r\*h

Surface = 2\*π\*r\*h + 2\*π\*r\*r

**Pyramid:**

Volume = 1/3\*b\*h

**Cone:**

Volume= 1/3 \*π\*r\*r\*h

Surface = π\*r\*r + π\*r\*s

=π\*r\*r + π\*r\*sqrt(r\*r + h\*h)

**Sphere:**

Volume= 4/3 π\*r\*r\*r

Surface=4\*π\*r\*r

**Sector of circle:**

R=radius,θ=angle in radian

Area=1/2\*θ\*r\*r

Area length, s=θ\*r

Area= π\*r\*r\*(θ/360®)

s=π\*r\*(θ/180®)

**Ellipse:**

a=semi major axis

b=semi minor axis

area = π\*a\*b

circumference = π\*(3\*(a+b)-sqrt((a+3\*b)\*(b+3\*a)))

**Equilateral Triangle:**

Height= (sqrt(3)/2) \*s

Area= (sqrt(3)/2) \*s\*s

**Annulus:**

r=inner radius

R=outer radius

Average radius, p= ½\*(r+R)

Width, w=R-r

Area = π(R\*R – r\*r) = 2\*π\*p\*w

Volume = 1/3\*π\*h\*(r\*r + r\*R+R\*R)

Degree and theta relation:

1º = π/180º

Radius = degree\*(π/180º)

**Cosine Law:**

(a/sinA)=(b/sinB)=(c/sinC)

© area= ½ab(sinC)=½bc(sinA)=½ca(sinB)

©when two angles and a side is given,

Find the unknown angle first,

B = 180º - A – C

Then apply,

(a/sinA)=(b/sinB)=(c/sinC)

**Polygon:**

Exterior Angle = 360º/n

Interior Angle = 180º - (360º/n)

= ((n-2)\*180º)/n

Area= (n\*side\*apothem)/2

Apothem = s/2\*(tan(180/n))

Perimeter = n\*side

Area = (perimeter\*apothem)/2

Radius(if given side) = s/2\*sin(180/n)

Radius(is given apothem) = apothem/cos(180/n)

Number of distinct diagonals

= n\*(n-3)/2

**Isosceles Triangle:**

Area = b/4 \*sqrt(4\*a\*a – b\*b)

©Circle radius (Inscribed a triangle)

r\*r = ((s-a)\*(s-b)\*(s-c))/s

©Circle radius (Circumscribed a triangle)

Area=abc/4\*R=sqrt(s\*(s-a)\*(s-b)\*(s-c))

R= abc/4\*Area(ABC)

**Series:**

No of terms = ( (least term – first term)/difference ) +1

Sum = n\*(n+1)/2

Average = n+1 / 2

Nth term = a\* rn-1

If r>0 sumn = a\*rn-1/r-1

If r<0 sumn = a\*i-rn/1-r

**Some Series:**

• 1+3+5+..+n = n\*n

• 2+4+6+…+n = n\*n +n

• 12 + 22 + …+ n2 = 1/6 \* n\*(n-1)\*(2n+1)

• 13 + 23 + 33 +…+n3 = ((n(n+1))/2)2

**Some Formula**

• Perimeter of a Square = P = 4a

Where a = Length of the sides of a Square

• Perimeter of a Rectangle = P = 2(l+b)

Where, l = Length ; b = Breadth

• Area of a Square = A = a2

Where a = Length of the sides of a Square

• Area of a Rectangle = A = l×b

Where, l = Length ; b = Breadth

• Area of a Triangle = A = ½×b×h

Where, b = base of the triangle ; h = height of the triangle

• Area of a Trapezoid = A = ½×(b1 + b2)×h

Where, b1 & b2 are the bases of the Trapezoid ; h = height of the Trapezoid

• Area of a Circle = A = π×r2

• Circumference of a Circle = A = 2πr

Where, r = Radius of the Circle

• Surface Area of a Cube = S = 6a2

Where, a = Length of the sides of a Cube

• Surface Area of a Cylinder=S=2πrh

• Volume of a Cylinder = V = πr2h

Where, r = Radius of the base of the Cylinder ; h = Height of the Cylinder

• Surface Area of a Cone = S = πr[r+√(h2+r2)]

• Volume of a Cone = V = ⅓×πr2h

Where, r = Radius of the base of the Cone, h = Height of the Cone

• Surface Area of a Sphere = S = 4πr2

• Volume of a Sphere = V = 4/3×πr3

Where, r = Radius of the Sphere

Area and Perimeter of a Hexagon

Perimeter = a + b1 + b2 + c

Area = (3√3/2 )r2

Area and Perimeter of an Octagon

• Perimeter = 8a

• Area = ( 2 + 2√2 )a2