Team notebook

February 20, 2017

Contents

1 Auxiliary Code

1.1 Bit Magic

```
Set bit                                    A |= 1 << bit
Clear bit                                  A &= ~(1 << bit)
Test bit                                   (A & 1 << bit) != 0
Toggle bit                                 A^(1<<bit)
Set subtraction                      A&~B
Set negation ALL_BITS              ^A
Value with only last set bit  x & ~(x - 1)
Check if n is power of 2           !( n&(n-1)) except n=0
Count trailing zeros               __builtin_ctz(n)
Count leading zeros                __builtin_clz(n)
total set bits                           __builtin_popcount(n)
Position of rightmost set bit  log2(n&-n)
Turn off rightmost set bit         n&(n-1)
```

1.2 Bitset

```cpp
int main() {
    //bitset1.to_string()
    //bitset1.to_ulong()
    // M For bit size i.e. 32,16,8,4..
    // default constructor initializes with all bits 0
    bitset<M> bset1;
    // bset2 is initialized with bits of 20
    bitset<M> bset2(20);
    // bset3 is initialized with bits of specified binary string
    bitset<M> bset3(string("1100"));
    // count function returns number of set bits in bitset
    int numberof1 = set2.count();
    // size function returns total number of bits in bitset
    // so there difference will give us number of unset(0)
    // bits in bitset
    int numberof0 = set2.size() - numberof1;
    // test function return 1 if bit is set else returns 0
    for (int i = 0; i < set2.size(); i++)
        cout << set2.test(i) << " ";
    // any function returns true, if atleast 1 bit
    // is set
    if (!set8.any())
        cout << "set8 has no bit set.\n";
    if (!bset1.any())
        cout << "bset1 has no bit set.\n";
    // none function returns true, if none of the bit is set
    if (!bset1.none())
        cout << "bset1 has all bit set\n";
    // flip function flips all bits i.e. 1 <-> 0 and 0 <-> 1
    set2.flip(2);
    set2.flip();
    // Converting decimal number to binary by using bitset
    int num = 100;
    Decimal number: num
    Binary equivalent: bitset<8>(num);
    //Usage 01: Set and Reset a bit in the bitset
    bitset3.set();
    bitset2.set(position);
    bitset2.reset();
    bitset3.reset(position);
    bitset<4> bset1(9);    // bset1 contains 1001
    bitset<4> bset2(3);    // bset2 contains 0011
    // comparison operator
    cout << (bset1 == bset2) << endl; // false 0
    cout << (bset1 != bset2) << endl; // true 1
    // bitwise operation and assignment
    cout << (bset1 ^= bset2) << endl; // 1010
    cout << (bset1 &= bset2) << endl; // 0010
    cout << (bset1 |= bset2) << endl; // 0011
    // left and right shifting
    cout << (bset1 <<= 2) << endl;  // 1100
    cout << (bset1 >>= 1) << endl;  // 0110
    // not operator
    cout << (~bset2) << endl;       // 1100
    // bitwise operator
    cout << (bset1 & bset2) << endl; // 0010
    cout << (bset1 | bset2) << endl; // 0111
```

```
    cout << (bset1 ^ bset2) << endl; // 0101
}
```

1.3 Header File

```
#include <algorithm>
#include <bitset>
#include <cctype>
#include <cmath>
#include <complex>
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <ctime>
#include <deque>
#include <fstream>
#include <iostream>
#include <list>
#include <map>
#include <memory>
#include <queue>
#include <set>
#include <sstream>
#include <stack>
#include <string>
#include <utility>
#include <vector>
#include <iomanip>
#include <numeric>
#include <iterator>
#include <typeinfo>
#include <valarray>
#include <functional>
#include <cassert>
#include <climits>
```

1.4 Sample Code

```
#include <bits/stdc++.h>
using namespace std;
typedef long long           ll;
typedef long long unsigned   llu;
#define pi                  acos(-1.0)
#define gcd(a,b)            __gcd(a,b)
template <class T> inline T BMOD(T p,T e,T m) {
    T ret=1;
    while(e) {
        if(e&1) ret=(ret*p)%m;
        p=(p*p)%m;
        e>>=1;
    }
    return (T)ret;
}
template <class T> inline T MODINV(T a,T m) {
    return BMOD(a,m-2,m);
}
template <class T> inline T isPrime(T a) {
    for(T i=2; i<=sqrt(double(a)); i++) {
        if(a%i==0) {
            return 0;
        }
    }
    return 1;
}
template <class T> inline T lcm(T a, T b) {
    return (a/gcd(a,b))*b;
}
template <class T> inline T power(T a, T b) {
    return (b==0)?1:a*power(a,b-1);
}
template <class T> inline string toString(T t) {
    stringstream ss;
    ss<<t;
    return ss.str();
}
template <class T> inline long long toLong(T t) {
    stringstream ss;
    ss<<t;
    long long ret;
    ss>>ret;
    return ret;
}
cout << fixed << setprecision(20) << Ans << endl;
priority_queue<piii,vpiii, greater<piii> >pq; //for dijkstra
int main() {
    ios_base::sync_with_stdio(false);
```

```
    cin.tie(NULL);
    return 0;
}
```

2 Data Structures

2.1 2D BIT

```
/*
 *
     https://www.topcoder.com/community/data-science/data-science-tutorials/binary-indexed-trees/#2d
 * */
int BIT[MX+5][MX+5];
void update(int x,int y,int val){
       while(x<=MX){
              int y1=y;
              while(y1<=MX){
                     BIT[x][y1] += val;
                     y1 += (y1 & -y1);
              }
              x+=(x&-x);
       }
}
int query(int x,int y){
       int ret = 0, y1;
       while(x>0){
              y1= y;
              while(y1>0){
                     ret += BIT[x][y1];
                     y1 -= (y1 & -y1);
              }
              x -= (x & -x);
       }
       return ret;
}
```

2.2 BIT

```
int BIT[MX];
void update(int x,int value) {
    while(x<MX) {
        BIT[x] += value;
        x += (x &(-x));
    }
}
int query(int x) {
    int ret = 0;
    while(x) {
        ret += BIT[x];
        x -= (x &(-x));
    }
    return ret;
}
```

2.3 Hashing

```
#define Base1 10000019LL
#define Base2 10000079LL
#define Base3 10000103LL
#define MOD1 1000000007LL
#define MOD2 1000000009LL
#define MOD3 1000000021LL
PLL h1[MX];
LL b1[MX],b2[MX];
void Init() {
    b1[0] = b2[0] = 1;
    for (int i=1; i<MX; i++) {
        b1[i] = (b1[i-1] * Base1)%MOD1;
        b2[i] = (b2[i-1] * Base2)%MOD2;
    }
}
void HashValue(char *s, PLL *h) {
    h[0]= {0,0};
    for (int i=0; i<l1; i++) {
        int x = s[i]-'a'+1;
        h[i+1].F = (h[i].F * Base1 + x)%MOD1;
        h[i+1].S = (h[i].S * Base2 + x)%MOD2;
    }

}
PLL SubStr(PLL *h,int x,int y) {
    PLL ans(0,0);
    if (x <= y) {
```

```
        int len = y-x+1;
        ans.X = (h[y].F - (h[x-1].F*b1[len])%MOD1)%MOD1;
        ans.Y = (h[y].S - (h[x-1].S*b2[len])%MOD2)%MOD2;
        ans.X += MOD1;
        ans.X %= MOD1;
        ans.Y += MOD2;
        ans.Y %= MOD2;
    }
    return ans;
}
```

2.4 RMQ (Sparse Table)

```
int M[100200][25],a[100020],n,l,r;
void sparseTableMin(){
        for(int i=0;i<n;i++) M[i][0]=i;
        for(int j=1;1<<j<=n;j++){
                for(int i=0;i+(1<<(j-1))<n;i++){
                        if(a[M[i][j-1]]<=a[M[i+(1<<(j-1))][j-1]])
                            M[i][j]=M[i][j-1];
                        else M[i][j]= M[i+(1<<(j-1))][j-1];
                }
        }
}
int RMQMin(int x,int y){
        int  k = log2(y-x+1);
        //~ int k = 31 - __builtin_clz(y-x+1);
        int i=x,j=y;
        if(a[M[i][k]]<=a[M[j-(1<<k)+1][k]]) return M[i][k];
        return M[j-(1<<k)+1][k];
}
```

2.5 SegmentTree with Lazy

```
char s[200010];
int n,q,Ans,sv[200000*4+10],tr[4*200000+10];
void ini(int p,int l,int r){
    if(l==r){
        tr[p]=(s[l]=='>')?1:0;
        return ;
    }
    int md=(l+r)/2;
    ini(2*p,l,md);
    ini(2*p+1,md+1,r);
    tr[p]=tr[p*2]+tr[p*2+1];
}
void lazzy(int p,int l,int r){
    if(sv[p]){
        tr[p]=r-l+1-tr[p];
        sv[p]=0;
        if(l!=r){
            sv[2*p]=!sv[p*2];
            sv[2*p+1]=!sv[p*2+1];
        }
    }
}
void update(int p,int l,int r,int x,int y){
    if(l>=x && r<=y){
        sv[p]=!sv[p];
    }
    lazzy(p,l,r);
    if(l>y || r<x) return;
    if(l>=x && r<=y) return;
    int md=(l+r)>>1;
    update(2*p,l,md,x,y);
    update(2*p+1,md+1,r,x,y);
    tr[p]=tr[p*2]+tr[p*2+1];
}
int query(int p,int l,int r,int x,int y){
    lazzy(p,l,r);
    if(l>y || r<x) return 0;
    if(l>=x && r<=y){
        return tr[p];
    }
    int md=(l+r)>>1;
    return query(2*p,l,md,x,y)+query(2*p+1,md+1,r,x,y);
}
```

2.6 Trie

```
struct trie{
        int cnt;
        trie *nd[30];
        trie(){
```

```
        cnt = 0;
        for(int i=0;i<30;i++) nd[i]=NULL;
    }
    ~trie(){
        for(int i=0;i<30;i++) {
            if(nd[i]!=NULL){
                delete nd[i];
            }
        }
    }
};
int build(string s,trie *rt){
    rt->cnt=0;
    for (int i = 0; i < (int)s.size(); i += 1){
        int p=s[i]-'a';
        if(rt->nd[p]==NULL){
            rt->nd[p]=new trie();
            rt->nd[p]->cnt=0;
        }
        if(i==s.size()-1){
            rt->nd[p]->cnt++;
            return (rt->nd[p]->cnt);
        }
        rt=rt->nd[p];
    }
}
int main(){
    trie *rt=new trie();
    cin>>s;
    int ans=build(s,rt);
    delete rt;
}
```

3 Dynamic Programming

3.1 Edit Distance

```
int dp[2001][2001];
int EditDistance(string s,string ss) {
    int lf = s.size(),ls = ss.size();
    for(int i=0; i<=lf; i++) dp[i][0]=i;
    for(int i=0; i<=ls; i++) dp[0][i]=i;
    dp[0][0]=0;
    for(int i=1; i<=lf; i++) {
        for(int j=1; j<=ls; j++) {
            int t=(s[i-1]==ss[j-1])? 0:1;
            dp[i][j]=min(dp[i-1][j-1]+t,min(dp[i][j-1]+1,dp[i-1][j]+1));
        }
    }
    return dp[lf][ls];
}
```

3.2 Longest Common Subsequence with Print

```
int path[MX][MX],lcs[MX][MX];
string s,ss;
void printLCS(int x,int y) {
    if(!x || !y ) return ;
    if(path[x][y]=='C') {
        printLCS(x-1,y-1);
        cout<<(char)s[x-1];
    } else if(path[x][y]=='U') {
        printLCS(x-1,y);
    } else
        printLCS(x,y-1);
}
void LCSlength() {
    memset(lcs,0,sizeof lcs);
    for(int i=1; i<=(int)s.size(); i++) {
        for(int j=1; j<=(int)ss.size(); j++) {
            if(s[i-1]==ss[j-1]) {
                lcs[i][j]=1+lcs[i-1][j-1];
                path[i][j]='C';//corner
            } else if(lcs[i][j-1]<lcs[i-1][j]) {
                lcs[i][j]=lcs[i-1][j];
                path[i][j]='U';//upper
            } else {
                path[i][j]='L';//left
                lcs[i][j]=lcs[i][j-1];
            }
        }
    }
}
```

3.3 Longest Common Substring with Print

```
void findSubstrings ( const string &s, set< string> &p ) {
    int l = s.length( ) ;
    for ( int i = 0 ; i < l ; i++ ) {
        for ( int j = 1 ; j < l - i + 1 ; j++ ) {
            p.insert ( s.substr( i, j ) ) ;
        }
    }
}
string lcs ( const string & s1, const string & s2 ) {
    set< string> sub1, sub2 ;
    findSubstrings ( s1, sub1 ) ;
    findSubstrings ( s2, sub2 ) ;
    set< string> common ;
    set_intersection ( sub1.begin( ), sub1.end( ),
                       sub2.begin( ), sub2.end( ),
                       inserter ( common, common.begin( ) ) ) ;
    vector< string> commonSubs ( common.begin( ), common.end( ) ) ;
    sort ( commonSubs.begin( ), commonSubs.end( ),
    []( const string &s1,const string &s2 ) {
        return s1.length( ) > s2.length( ) ;
    } ) ;
    if(commonSubs.size()==0) return "";
    return *(commonSubs.begin( ) ) ;
}
```

3.4 Longest Common Substring

```
int LCS(const string& str1, const string& str2) {
    if(str1.empty() || str2.empty()) {
        return 0;
    }
    int *curr = new int [str2.size()];
    int *prev = new int [str2.size()];
    int *swap = nullptr;
    int maxSubstr = 0;
    for(int i = 0; i<str1.size(); ++i) {
        for(int j = 0; j<str2.size(); ++j) {
            if(str1[i] != str2[j]) {
                curr[j] = 0;
            } else {
                if(i == 0 || j == 0) {
                    curr[j] = 1;
                } else {
                    curr[j] = 1 + prev[j-1];
                }
                maxSubstr=max(maxSubstr,curr[j]);
            }
        }
        swap=curr;
        curr=prev;
        prev=swap;
    }
    delete [] curr;
    delete [] prev;
    return maxSubstr;
}
```

3.5 Longest Increasing Subsequence with Print

```
ll seq[MX],check[MX],prn[MX];
ll lisLength(int sz) {
    ll ln=1,mn=0;
    check[0]=seq[0];
    prn[0]=1;
    for(int k=1; k<sz; k++) {
        if(check[ln-1]<seq[k]) {
            check[ln++]=seq[k];
            prn[k]=ln;
        } else {
            mn=lower_bound(check,check+ln,seq[k])-check;
            check[mn]=seq[k];
            prn[k]=mn+1;
        }
    }
    return ln;
}
void printLIS(int sz) {
    int len=lisLength(sz);
    stack<int>mk;
    for(int k=sz-1; k>=0; k--) {
        if(prn[k]==len) {
            mk.push(seq[k]);
            len--;
        }
```

```
    }
    while(!mk.empty()) {
        cout<<mk.top()<<endl;
        mk.pop();
    }
}
```

4 Flow Matching

4.1 Biparted Machting

```
int n,seen[M],matchR[M];
vector<int>bpGraph[M];
bool bpm(int u){
    for(int v=0;v<(int)bpGraph[u].size();v++){
        int V=bpGraph[u][v];
        if(!seen[V]){
            seen[V]=true;
            if(matchR[V]==-1||bpm(matchR[V])){
                matchR[V]=u;
                return true;
            }
        }
    }
    return false;
}
int maxBPM(){
    memset(matchR,-1, sizeof matchR);
    int result=0;
    for(int i=0;i<n;i++){
        memset(seen,0,sizeof seen);
        if(bpm(i)){
            result++;
        }
    }
    return result;
}
```

4.2 Dinic (Maxflow)

```
const int MAXN = 20020;
const int INF = 1000000000;
struct edge {
        int a, b, cap, flow;
};
int N, src, snk, d[MAXN], ptr[MAXN], q[MAXN];
vector<edge> e;
vector<int> g[MAXN];
void add_edge (int a, int b, int cap) {
        edge e1 = { a, b, cap, 0 };
        edge e2 = { b, a, 0, 0 };
        g[a].push_back ((int) e.size());
        e.push_back (e1);
        g[b].push_back ((int) e.size());
        e.push_back (e2);
}
bool bfs() {
        int qh=0, qt=0;
        q[qt++] = src;
        memset (d, -1, N * sizeof d[0]);
        d[src] = 0;
        while (qh < qt && d[snk] == -1) {
                int v = q[qh++];
                for (size_t i=0; i<g[v].size(); ++i) {
                        int id = g[v][i],
                                to = e[id].b;
                        if (d[to] == -1 && e[id].flow < e[id].cap) {
                                q[qt++] = to;
                                d[to] = d[v] + 1;
                        }
                }
        }
        return d[snk] != -1;
}
int dfs (int v, int flow) {
        if (!flow) return 0;
        if (v == snk) return flow;
        for (; ptr[v]<(int)g[v].size(); ++ptr[v]) {
                int id = g[v][ptr[v]],
                        to = e[id].b;
                if (d[to] != d[v] + 1) continue;
                int pushed = dfs (to, min (flow, e[id].cap - e[id].flow));
                if (pushed) {
                        e[id].flow += pushed;
                        e[id^1].flow -= pushed;
```

```
                        return pushed;
                }
        }
        return 0;
}
int dinic() {
        int flow = 0;
        for (;;) {
                if (!bfs()) break;
                memset (ptr, 0, N * sizeof ptr[0]);
                while (int pushed = dfs (src, INF))
                        flow += pushed;
        }
        return flow;
}
```

5 Game Theory

5.1 Hackenbush (simple)

```
/*
 * g[] for adjacent list of Graph
* */
vector<int>g[505];
int hackenbush(int x,int pr=-1) {
    int ret=0;
    for(int i=0; i<(int)g[x].size(); i++) {
        int y = g[x][i];
        if(pr!=y) {
            ret ^= (1 + hackenbush(y,x));
        }
    }
    return ret;
}
```

5.2 Nim Game

Nimble: Given a line of squares labeled 0, 1, 2, ... Several coins are placed on some squares (it's possible to have more than one coin on a single square). Two players take turns. One move consists of taking any one coin and moving it to any square to the left of it. It's possible that the coin moved into a square already containing coins.

Solution:

This game is exactly a Nim Game where one coin in kth square is a pile of k stones. Moving a coin from kth square to its left is equivalent to removing stones from a pile of k stones in Nim Game.

Poker Nim:

This game is played as a standard Nim Game, but players have option to either substracting stones (like in standard Nim Game) or adding more stones in a pile but not exceeding the original number of stones in that pile. To ensure the game termination, each player is allowed to add stones at most k (a finite number of) times.

Solution:

If you already have a winning position in standard Nim Game, then when your opponent adds some stones to a pile, all you have to do is reverse your opponent's move by removing the exact number of stones he/she added to that pile, thus restoring your winning position. Hence, this kind of game can be viewed as standard Nim Game.

Actually the "not exceeding the original number of stones in that pile" part is not neccesary in this game, one can have the same winning strategy despite of the number of stones added by the opponent.

This type of Nim Game where player can add stones is called Bogus Nim.

turning Turtles:

Given a horizontal line of N coins with some coins showing heads and some tails. Each turn, a player have to flip one coin from head to tail, and in the same time (if he/she wants), flip one more coin to the left of it.

For example, consider this sequence of N = 10 coins:

1 2 3 4 5 6 7 8 9 10 T H T T H H T T T H

Possible moves from this position are:

Flip one coin from head to tail. Eg., coin in position 6 from head to tail. Flip one coin from head to tail and flip another coin (from tail to head) to the left of it. Eg., coin in position 6 from head to tail and flip coin in position 3 from tail to head. Flip one coin from head to tail and flip another coin (from head to tail) to the left of it. Eg., coin in position 6 from head to tail and flip coin in position 2 from head to tail.

Solution:

This game is equivalent to Nim Game with each coin showing head in kth position equals to a pile of k stones.

Flip one coin of position k from head to tail. This move is equivalent with removing all stones from a pile with k stones.

Flip one coin of position k from head to tail and flip another coin (from tail to

head) to the left of it in position t. This move is equivalent with removing some stones from a pile with k stones leaving t stones in that pile.

Flip one coin of position k from head to tail and flip another coin (from head to tail) to the left of it in position t. This move is equivalent with removing some stones from a pile with k stones leaving t stones in that pile. Note that having two piles with a same number of stones is the same as having none of both piles, since if you already have a winning position and your opponent make a move on one of those identic piles, you can always reverse his/her move by moving on the other pile (remember x x = 0).

Silver Dollar Game (without silver dollar):

This game is played on a line of squares labeled 0, 1, 2, ... with several coins are placed in some square such that no two coins are placed in a same square. One move consists of moving one coin to its left onto any empty square and not passing any other coin. The game ends when a player cannot make any legal moves, since all the coins are jammed at the left-end of the strip.

For example, given these 4 coins in position 2, 5, 7, 10:

1 2 3 4 5 6 7 8 9 10 11 0 1 0 0 1 0 1 0 0 0 1

Possible moves from this position is:

Move coin in position 11 to position 10, 9 or 8.

Move coin in position 7 to position 6.

Move coin in position 5 to position 4 or 3.

Move coin in position 2 to position 1.

Solution:

The solution to this problem is a bit tricky.

We can think spaces between coins as pile. For example, if we have coins in position 2, 5, 7, 11, then the gaps between coins are 1, 2, 1, 3. Then moving a coin from position 7 to 6, is like removing one stone in 3rd pile. But then, that same move also increase the size of 4th pile by one stone. In other word, any stones removed from a pile will increase the size of its immediate right pile by the same size, except for the right most pile. This decomposition is not good, because the subgames are not independent (moving one pile could increase another pile). We should think another way to make this work.

The problem in above decomposition was with adjacent piles, so we can try to skip every second piles. In fact, this is the solution to this problem. From right most pile, skip every second piles, and take the remaining piles as piles in standard Nim Game. For example, coins in position 2, 5, 7, 11 will have piles 1, 2, 1, 3 and if we remove all second piles from the right most, we'll get 2, 3.

In the example above, coins in position 2 and 5 correspond to one pile of 2 stones, while coins in position 7 and 11 correspond to one pile of 3 stones.

Moving coin in position 2 is equivalent with adding more stones to the respective pile (of 2 stones).

Moving coin in position 5 is equivalent with reducing stones from the respective pile (of 2 stones).

Moving coin in position 7 is equivalent with adding more stones to the respective pile (of 3 stones).

Moving coin in position 11 is equivalent with reducing stones from the respective pile (of 3 stones).

1 2 3 4 5 6 7 8 9 10 11 0 1 * * 1 0 1 * * * 1

Each * character represent the stone in each pile. So, this decomposition gives us a Bogus Nim (review the solution to Poker Nim), which can be viewed as a standard Nim Game.

But why from "right most pile"? If we remove every second stone from left most pile, we'll leave the right most coin without any corresponding pile, which is bad, because it's like allowing player to not make any move.

5.3 One Pile Grandy(normal)

```
/*
 * https://www.hackerrank.com/contests/w25/challenges/stone-division
 *
 * Solution: Just find Grandy Value among all cut
 * */
map<ll,int> sv;
ll a[15],m,n;
ll grandy(ll pile){
        if(pile<=1) return 0;
        if(sv.count(pile)) return sv[pile];
        set<int>st;
        for(int i=0;i<m;i++){
                if(pile%a[i]){}
                else{
                        if(a[i]%2){
                                st.insert(grandy(pile/a[i]));
                        }
                        else st.insert(0);
                }
        }
        int mn = 0;
        for(auto x: st){
                if(mn!=x) break;
                mn++;
        }
        return (sv[pile]=mn);
}
```

```
int main(){
	cin>>n>>m;
	for(int i=0;i<m;i++) cin>>a[i];
	ll Ans = grandy(n);
	if(Ans) cout<<"First"<<endl;
	else cout<<"Second"<<endl;
	return 0;
}
```

6 Geometry

6.1 Convex Hull (monotone chain)

```
// Implementation of Andrew's monotone chain 2D convex hull algorithm.
#include <algorithm>
#include <vector>
using namespace std;

typedef int coord_t;      // coordinate type
typedef long long coord2_t; // must be big enough to hold
    2*max(|coordinate|)^2

struct Point {
	coord_t x, y;

	bool operator <(const Point &p) const {
		return x < p.x || (x == p.x && y < p.y);
	}
};

// 2D cross product of OA and OB vectors, i.e. z-component of their 3D
    cross product.
// Returns a positive value, if OAB makes a counter-clockwise turn,
// negative for clockwise turn, and zero if the points are collinear.
coord2_t cross(const Point &O, const Point &A, const Point &B)
{
	return (A.x - O.x) * (coord2_t)(B.y - O.y) - (A.y - O.y) *
	    (coord2_t)(B.x - O.x);
}

// Returns a list of points on the convex hull in counter-clockwise order.
// Note: the last point in the returned list is the same as the first one.
vector<Point> convex_hull(vector<Point> P)
{
	int n = P.size(), k = 0;
	vector<Point> H(2*n);

	// Sort points lexicographically
	sort(P.begin(), P.end());

	// Build lower hull
	for (int i = 0; i < n; i++) {
		while (k >= 2 && cross(H[k-2], H[k-1], P[i]) <= 0) k--;
		H[k++] = P[i];
	}

	// Build upper hull
	for (int i = n-2, t = k+1; i >= 0; i--) {
		while (k >= t && cross(H[k-2], H[k-1], P[i]) <= 0) k--;
		H[k++] = P[i];
	}

	H.resize(k);
	return H;
}
```

6.2 Convex Hull (Graham Scan)

```
struct PT {
    int x,y;
}
// A globle point needed for sorting points with reference
// to the first point Used in compare function of qsort()
PT p0;
// A utility function to find next to top in a stack
PT nextToTop(stack<PT> &S) {
    PT p = S.top();
    S.pop();
    PT res = S.top();
    S.push(p);
    return res;
}
// A utility function to swap two points
int swap(PT &p1, PT &p2) {
    PT temp = p1;
```

```
    p1 = p2;
    p2 = temp;
}
// A utility function to return square of distance
// between p1 and p2
int distSq(PT p1, PT p2) {
    return (p1.x - p2.x)*(p1.x - p2.x) + (p1.y - p2.y)*(p1.y - p2.y);
}
// 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(PT p, PT q, PT 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
}
// A function used by library function qsort() to sort an array of
// points with respect to the first point
int compare(const void *vp1, const void *vp2) {
    Point *p1 = (Point *)vp1;
    Point *p2 = (Point *)vp2;
    // Find orientation
    int o = orientation(p0, *p1, *p2);
    if (o == 0)
        return (distSq(p0, *p2) >= distSq(p0, *p1))? -1 : 1;
    return (o == 2)? -1: 1;
}
// Prints convex hull of a set of n points.
void convexHull(PT points[], int n) {
    // Find the bottommost && leftmost point
    int ymin = points[0].y, min = 0;
    for (int i = 1; i < n; i++) {
        int y = points[i].y;
        if ((y < ymin) || (ymin == y && points[i].x < points[min].x))
            ymin = points[i].y, min = i;
    }
    // Place the bottom-most point at first position
    swap(points[0], points[min]);
    // Sort n-1 points with respect to the first point.
    // A point p1 comes before p2 in sorted ouput if p2
    // has larger polar angle (in counterclockwise
    // direction) than p1
    p0 = points[0];
    qsort(&points[1], n-1, sizeof(PT), compare);
    // If two or more points make same angle with p0,
    // Remove all but the one that is farthest from p0
    // Remember that, in above sorting, our criteria was
    // to keep the farthest point at the end when more than
    // one points have same angle.
    int m = 1; // Initialize size of modified array
    for (int i=1; i<n; i++) {
        // Keep removing i while angle of i and i+1 is same
        // with respect to p0
        while (i < n-1 && orientation(p0, points[i], points[i+1]) == 0)
            i++;
        points[m] = points[i];
        m++;  // Update size of modified array
    }
    // If modified array of points has less than 3 points,
    // convex hull is not possible
    if (m < 3) return;
    // Create an empty stack and push first three points
    // to it.
    stack<PT> S;
    S.push(points[0]);
    S.push(points[1]);
    S.push(points[2]);
    // Process remaining n-3 points
    for (int i = 3; i < m; i++) {
        // Keep removing top while the angle formed by
        // points next-to-top, top, and points[i] makes
        // a non-left turn
        while (orientation(nextToTop(S), S.top(), points[i]) != 2)
            S.pop();
        S.push(points[i]);
    }
    // Now stack has the output points, print contents of stack
    while (!S.empty()) {
        PT p = S.top();
        cout << "(" << p.x << ", " << p.y <<")" << endl;
        S.pop();
    }
}
```

6.3 Convex Hull(Monotone$_chain$)

```
// Compute the 2D convex hull of a set of points using the monotone chain
// algorithm. Eliminate redundant points from the hull if
    REMOVE_REDUNDANT is
// #defined.
//
// Running time: O(n log n)
//
// INPUT: a vector of input points, unordered.
// OUTPUT: a vector of points in the convex hull, counterclockwise,
    starting
// with bottommost/leftmost point
#include <cstdio>
#include <cassert>
#include <vector>
#include <algorithm>
#include <cmath>
using namespace std;
#define REMOVE_REDUNDANT
typedef double T;
const T EPS = 1e-7;
struct PT {
    T x, y;
    PT() {}
    PT(T x, T y) : x(x), y(y) {}
    bool operator<(const PT &rhs) const {
        return make_pair(y,x) < make_pair(rhs.y,rhs.x);
    }
    bool operator==(const PT &rhs) const {
        return make_pair(y,x) == make_pair(rhs.y,rhs.x);
    }
};
T cross(PT p, PT q) {
    return p.x*q.y-p.y*q.x;
}
T area2(PT a, PT b, PT c) {
    return cross(a,b) + cross(b,c) + cross(c,a);
}
#ifdef REMOVE_REDUNDANT
bool between(const PT &a, const PT &b, const PT &c) {
    return (fabs(area2(a,b,c)) < EPS && (a.x-b.x)*(c.x-b.x) <= 0 &&
        (a.y-b.y)*(c.y-b.y) <= 0);
}
#endif
void ConvexHull(vector<PT> &pts) {
    sort(pts.begin(), pts.end());
    pts.erase(unique(pts.begin(), pts.end()), pts.end());
    vector<PT> up, dn;
    for (int i = 0; i < pts.size(); i++) {
        while (up.size() > 1 && area2(up[up.size()-2], up.back(), pts[i])
            >= 0) up.pop_back();
        while (dn.size() > 1 && area2(dn[dn.size()-2], dn.back(), pts[i])
            <= 0) dn.pop_back();
        up.push_back(pts[i]);
        dn.push_back(pts[i]);
    }
    pts = dn;
    for (int i = (int) up.size() - 2; i >= 1; i--) pts.push_back(up[i]);
#ifdef REMOVE_REDUNDANT
    if (pts.size() <= 2) return;
    dn.clear();
    dn.push_back(pts[0]);
    dn.push_back(pts[1]);
    for (int i = 2; i < pts.size(); i++) {
        if (between(dn[dn.size()-2], dn[dn.size()-1], pts[i]))
            dn.pop_back();
        dn.push_back(pts[i]);
    }
    if (dn.size() >= 3 && between(dn.back(), dn[0], dn[1])) {
        dn[0] = dn.back();
        dn.pop_back();
    }
    pts = dn;
#endif
}
```

7 Graph Theory

7.1 Articulation Bridge

```
vector<pair<int,int> > Ans;
int n,ds[100010],lw[100010],pr[100010],vs[100100];
vector<int> adj[100010];
int cnt=0;
void dfs(int u){
    vs[u]=1;
    ds[u]=lw[u]=++cnt;
    for(int i=0;i<(int)adj[u].size();i++){
```

```
        int v=adj[u][i];
        if(!vs[v]){
            pr[v]=u;
            dfs(v);
            lw[u]=min(lw[u],lw[v]);
            if(lw[v]>ds[u]){
                Ans.push_back(make_pair(min(u,v),max(u,v)));
            }
        }
        else if(v!=pr[u]){
            lw[u]=min(lw[u],ds[v]);
        }
    }
}
// Multiple edges from a to b are not allowed.
// (they could be detected as a bridge).
// If you need to handle this, just count
// how many edges there are from a to b.
void bridge(){
    minus(pr);
    zero(vs);
    zero(lw);
    zero(ds);
    for(int i=0;i<n;i++){
        if(!vs[i]){
            cnt=0;
            dfs(i);
        }
    }
}
```

7.2 Articulation Point

```
/*
Ans[i] true for I is a Point.
*/
int n,ds[MX],lw[MX],pr[MX],vs[MX],Ans[MX];
vector<int> adj[MX];
int cnt=0;
void dfs(int u){
    vs[u]=1;
    ds[u]=lw[u]=++cnt;
    int cn=0;
    for(int i=0;i<(int)adj[u].size();i++){
        int v=adj[u][i];
        if(!vs[v]){
            pr[v]=u;
            cn++;
            dfs(v);
            lw[u]=min(lw[u],lw[v]);
            if(pr[u]==-1 && cn>1) Ans[u]=1; // for root check

            if(lw[v]>=ds[u] && pr[u]!=-1){
                Ans[u]=1;
            }
        }
        else if(v!=pr[u]){
            lw[u]=min(lw[u],ds[v]);
        }
    }
}
void ArticulationPoint(){
    minus(pr);
    zero(vs);
    zero(lw);
    zero(ds);
    zero(Ans);
    for(int i=1;i<=n;i++){
        if(!vs[i]){
            cnt=0;
            dfs(i);
        }
    }
}
```

7.3 Dijkstra Second Shortest Path

```
int n,m,ts,cs=0,a,b,c,ds[5][5005],vs[5][5005],Ans;
vpii g[5005];
int dijkstraSecond(int x){
    setinf(ds);
    zero(vs);
    priority_queue<piii,vpiii, greater<piii> >pq;
    ds[1][x]=0;
    pq.P(piii(1,pii(0,x)));
    while(!pq.empty()){
```

```
        piii p=pq.top();pq.pop();
        int u=p.S.S, t=p.F;
        if(vs[t][u]) continue;
        else vs[t][u]=1;
        //~ if(u==n and t==2) return ds[2][u];
        //~ if(p.S.F>ds[t][u]) continue;
        for(int i=0;i<(int)g[u].size();i++){
            pii V=g[u][i];
            int v=V.F;
            int tl=ds[t][u]+V.S;
            if(tl<ds[1][v]){
                ds[2][v]=ds[1][v];
                pq.P(piii(2,pii(ds[2][v],v)));
                ds[1][v]=tl;
                pq.P(piii(1,pii(ds[1][v],v)));
            }
            else if(tl>ds[1][v] && tl<ds[2][v]){
                ds[2][v]=tl;
                pq.P(piii(2,pii(ds[2][v],v)));
            }
        }
    }
    return ds[2][n];
}
```

7.4 Heavy-Light Decomposition

```
#define MX 30009
vi g[MX];
int
    pr[MX],lv[MX],sz[MX],sc[MX],hd[MX],tr[4*MX],ch[MX],pos[MX],Ans,ps,no,a[MX];
void DFS(int u,int p,int l){
        pr[u]=p;lv[u]=l;sz[u]=1;sc[u]=-1;
        int mx=-1;
        for(int i=0;i<(int)g[u].size();i++){
                int v=g[u][i];
                if(v!=p){
                        DFS(v,u,l+1);
                        sz[u]+=sz[v];
                        if(sz[v]>mx){
                                mx=sz[v];
                                sc[u]=v;
                        }
                }
        }
}
void HLD(int u){
        if(hd[no]==-1){
                hd[no]=u;
        }
        pos[u]=++ps;
        ch[u]=no;
        if(sc[u]!=-1){
                HLD(sc[u]);
                for(int i=0;i<(int)g[u].size();i++){
                        int v=g[u][i];
                        if(v!=sc[u] && v!=pr[u]){
                                no++;
                                HLD(v);
                        }
                }
        }
        else{
                for(int i=0;i<(int)g[u].size();i++){
                        int v=g[u][i];
                        if(v!=sc[u] && v!=pr[u]){
                                no++;
                                HLD(v);
                        }
                }
        }
}
void update(int p,int s,int e,int i,int v){
        if(s==i && e==i){
                tr[p]=v;
                return;
        }
        int m=(s+e)/2;
        int l=p*2;
        int r=l+1;
        if(i<=m) update(l,s,m,i,v);
        else update(r,m+1,e,i,v);
        tr[p]=tr[l]+tr[r];
}
int query(int p,int s,int e,int x,int y){
        if(x<=s && e<=y){
                return tr[p];
        }
```

```
	int m=(s+e)/2;
	int l=p*2;
	int r=l+1;
	if(y<=m) return query(l,s,m,x,y);
	else if(x>m) return query(r,m+1,e,x,y);
	return query(l,s,m,x,m)+query(r,m+1,e,m+1,y);
}

void LCA(int u,int v){
	int uc=ch[u];
	int vc=ch[v];
	if(uc==vc){
		if(lv[u]<lv[v]){
			Ans+=query(1,1,ps,pos[u],pos[v]);
		}
		else{
			Ans+=query(1,1,ps,pos[v],pos[u]);
		}
		return;
	}
	else{
		int uh=hd[uc];
		int vh=hd[vc];
		if(lv[uh]<lv[vh]){
			Ans+=query(1,1,ps,pos[vh],pos[v]);
			v=pr[vh];
		}
		else{
			Ans+=query(1,1,ps,pos[uh],pos[u]);
			u=pr[uh];
		}
		LCA(u,v);
	}
}
int main()
{
	int ts,cs=0,n,u,v,q,c;
	I(ts);
	while(ts--){
		I(n);
		for(int i=1;i<=n;i++) I(a[i]);
		for(int i=1;i<n;i++){
			I2(u,v);
			u++;v++;
			g[u].pb(v);
			g[v].pb(u);
		}
		DFS(1,-1,1);
		memset(hd,-1,sizeof hd);
		ps=0;
		no=0;
		HLD(1);
		memset(tr,0,sizeof tr);
		for(int i=1;i<=n;i++){
			update(1,1,ps,pos[i],a[i]);
		}
		I(q);
		printf("Case %d:\n",++cs);
		while(q--){
			I3(c,u,v);
			u++;
			v++;
			if(c==0){
				Ans=0;
				LCA(u,v);
				PI(Ans);
				nl;
			}
			else{
				update(1,1,ps,pos[u],v-1);
			}
		}
		for(int i=0;i<=n;i++) g[i].clear();
	}
	return 0;
}
```

7.5 Minimum Spanning Tree (Prim's)

```
#define md 1000000007ll
vector<p> g[100010];
bool vs[100010];
ll prim(ll x){
	priority_queue<p,vector<p>,greater<p> >pq;
	memset(vs,false,sizeof vs);
	ll ret=1;
	pq.push({1,x});
	while(!pq.empty()){
```

```
            ll u=pq.top().first;
            ll v=pq.top().second;
            pq.pop();
            if(vs[v]) continue;
            u%=md;
            ret*=u;
            ret%=md;
            vs[v]=true;
            for(int i=0;i<(int)g[v].size();i++){
                    u=g[v][i].second;
                    if(!vs[u]) pq.push(g[v][i]);
            }
        }
        return ret%md;
}
```

7.6 Minimum Spanning Tree (kruskal)

```
struct edge{
    int u,v,w;
    bool operator<(const edge &p) const{
        return w<p.w;
    }
};
int pr[MAXN];
vector<edge>e;
int FIND(int r){return (pr[r]==r)?r:FIND(pr[r]);}
int MST(int n){
        sort(e.begin(),e.end());
    for(int i=1;i<=n;i++) pr[i]=i;
    int cnt=0,s=0;
    for(int i=0;i<(int)e.size();i++){
        int u=FIND(e[i].u);
        int v=FIND(e[i].v);
        if(u!=v){
            pr[u]=v;
            cnt++;
            s+=e[i].w;
            if(cnt==n-1) break; //it can not be a loop
        }
    }
    return s;
}
```

7.7 Stable Marriage Problem

```
int main(){
    scanf("%d", &n);
    for( int i = 0; i < n; i++ ){
        for( int j = 0; j < n; j++ ){
            scanf("%d", &mPref[i][j]);
            mPref[i][j] -= (n+1);
        }
    }
    for( int i = 0; i < n; i++ ){
        for( int j = 0; j < n; j++ ){
            scanf("%d", &k);
            wPref[i][k-1] = n - j;
        }
    }
    memset( R, -1, sizeof(R) );
    memset( P, 0, sizeof(P) );
    for( int i = 0; i < n; i++ ){
        int man = i;
        while( man >= 0 ){
            int woman = mPref[man][ P[man]++ ];
            if( R[woman] == -1 ||
                wPref[woman][R[woman]]<wPref[woman][man]){
                L[man] = woman;
                swap( man, R[woman] );
            }
        }
    }
    for( int i = 0; i < n; i++ ) printf(" (%d %d)", i+1,L[i]+n+1);
}
```

7.8 Strongly Connected Component

```
bool vs[MX],flg[MX][MX];
int SCC[MX],dp[MX],Group=0;
vector<int> g[MX],rg[MX],stk;
void dfs(int u) {
    vs[u]=1;
    for(int i=0; i<(int)g[u].size(); i++) {
        if(!vs[ g[u][i] ]) dfs(g[u][i]);
    }
    stk.push_back(u);
```

```
}
void DFS(int u) {
    vs[u]=0;
    dp[Group]++;
    for(int i=0; i<(int)rg[u].size(); i++) {
        if(vs[ rg[u][i] ]) DFS(rg[u][i]);
    }
    SCC[u]=Group;
}
void FindSCC(int n) {
    for(int i=1; i<=n; i++) {
        if(!vs[i]) dfs(i);
    }
    for(int i=stk.size()-1; i>=0 ; i--) {
        if(vs[stk[i]]) {
            Group++;
            DFS(stk[i]);
        }
    }
}
void BuildDAG(int n) {
    for(int i=0; i<=n; i++) g[i].clear();
    for(int i=1; i<=n; i++) {
        for(int j=1; j<=n; j++) {
            if(SCC[i]==SCC[j]) continue;
            if(flg[i][j]) {
                g[SCC[i]].push_back(SCC[j]);
            }
        }
    }
}
```

7.9 Topological Sort (lexicographically)

```
vector<int>child[1000005];
int cnt[1000005];
queue<int>parent;
vector<int>Sort;

void topologicalSort(int n){
    int p;
    for(int i=1;i<=n;i++){
        if(!cnt[i]){
            parent.push(i);
        }
    }
    while(!parent.empty()){
        p=parent.front();
        for(int i=0;i<(int)child[p].size();i++){
            int cl=child[p][i];
            cnt[cl]--;
            if(!cnt[cl]){
                parent.push(cl);
            }
        }
        Sort.push_back(p);
        parent.pop();
    }
}
```

8 Math

8.1 Chinese Remainder Theorem (CRT)

```
ll modinv(ll n,ll m){
    ll x,y;
    ll gd = extendedEuclid(n, m, x, y);
    if(gd != 1) return 0;
    return (x+m)%m;
}
ll CRT(ll a[],ll m[],ll n){
    ll ret = 0, lc = 1;
    for(ll i = 0; i < n; i++) lc *= a[i];
    for(ll i = 0; i < n; i++){
        ll tmp = m[i] * (lc / a[i]);
        tmp %= lc;
        tmp = (tmp * modinv(lc/a[i],a[i]))%lc;
        ret += tmp;
        ret %= lc;
    }
    return (ret+lc)%lc;
}
```

8.2 Discrete Logarithm — Shank's Baby-step Giant-step Algorithm

```
/*
Discrete Logarithm:
The discrete logarithm is an integer x solving the equation
a^ xb (mod m)
where a and m are relatively prime. (Note: if they are not relatively
    prime, then the algorithm described below is incorret, though it can
    be modified so that it can work).
which has complexity O(mlog(m)).

Algorithm:
Consider the equation:
a^x    b (mod m)
where a and m are relatively prime.

Let x= npq , where n is some pre-selected constant (we will describe how
    to select n later). p is known as giant step, since increasing it by
    one increases x by n. Similarly, q is known as baby step.

Obviously, any value of x in the interval [0;m) can be represented in
    this form, where p[1;m/n]and  q [0;n].

Then, the equation becomes:
a^( npq )    b (mod m).

Using the fact that aa and mm are relatively prime, we obtain:
a^(np)    ba^q (mod m)

This new equation can be rewriten in a simplified form:
f1(p) = f2(q).

This problem can be solved using the method meet-in-the-middle as follows:
       We calculate f1 for all possible values of p. Sort these values.
       For each value of q, calculate f2, and look for the corresponding
           value of p using the sorted array of f1 using binary search.
*/
int Solve(int a,int b,int md){
       n=(int)sqrt(md+.0);
       int an=1;
       for(int i=0;i<n;++i){
              an=(an*a)%md;
       }
       //~ unordered_map<int,int>val;
       //~ map<int,int>val;
       for(int i=1,cr=an;i<=n;++i){
              if(!val.count(cr)){
                     val[cr]=i;
              }
              cr=(cr*an)%md;
       }
       for(int i=0,cr=b;i<=n;++i){
              if(val.count(cr)){
                     Ans=val[cr]*n-i;
                     if(Ans<md){
                            return Ans;
                     }
              }
              cr=(cr*a)%md;
       }
       return -1;
}
```

8.3 Divisor Sum

```
// Cumulative Sum of Divisors of 1 to n
int CSOD(int n) {
    int sq=sqrt(n);
    int Ans=0;
    for(int i=1; i<=sq; i++) {
        int j=n/i;
        Ans+=i*(j-i);
        Ans+=(i+j)*(j-i+1)/2;
    }
    return Ans;
}
// Sum of number of Divisor of 1 to n
int SNOD( int n ) {
    int res = 0;
    int u = sqrt(n);
    for ( int i = 1; i <= u; i++ ) {
        res += ( n / i ) - i; //Step 1
    }
    res *= 2; //Step 2
    res += u; //Step 3
    return res;
```

```
}
```

8.4 Euler Phi

```
#define MX (ll)1000000
ll vs[MX+1];
ll PHI[MX+1];
vl PRM;
void eulerPhi(){
    PHI[1]=1ll;
    for(ll i=2; i <= MX ; i++){
        if(vs[i]==0){
            vs[i]=i;
            PHI[i]=i-1;
            PRM.pb(i);
        }
        else{
            if(vs[i]==vs[i/vs[i]]){
                PHI[i]=PHI[i/vs[i]]*vs[i];
            }
            else{
                PHI[i]=PHI[i/vs[i]]*(vs[i]-1);
            }
        }
        for(int j=0; j<(int)PRM.size() && PRM[j]<=vs[i] && i*PRM[j]<=MX;
             j++){
            vs[i*PRM[j]]=PRM[j];
        }
    }
}
/*
You can get it from the formula for PHI:
PHI [n] = n * (1 - 1/p1) * (1 - 1/p2) * ... * (1 - 1/pn) where p1,p2.. pn
    are the prime factors of n.
Say we know the smallest prime factor of n, that is vs[n]. What is the
    formula for n/vs[n]?
Well there are two cases:
Case 1:
n/vs[n] is still divisible by vs[n], that is vs[n] is a factor of n with
    a power greater than or equal to 2, so n/vs[n] has exactly the same
    prime factors as n.
Then PHI[n/vs[n]] = (n/vs[n]) * (1 - 1/p1) * (1 - 1/p2) * ... * (1 -
    1/pn) It differs from the above formula by the first term. So just
    multiply phi[n/vs[n]] by vs[n] to get phi[n]. PHI[n] = PHI[n/vs[n]] *
    vs[n]
Case 2:
n/vs[n] is not divisible by vs[n], then the above formula for n/vs[n] is
    almost the same, it just doesn't have (1  1/vs[n]) as one of its
    terms. So we have to multiply phi[n/vs[n]] not only by vs[n] but also
    by (1  1/vs[n]).
Then PHI[n] = PHI[n/vs[n]] * vs[n] * (1 - 1/vs[n]] => PHI[n] =
    PHI[n/vs[n]] * (vs[n] - 1)
*/
```

8.5 Extended Euclid

```
ll extendedEuclid(ll a,ll b, ll &x,ll &y){
    if(a == 0){
        x = 0;
        y = 1;
        return b;
    }
    ll x1,y1;
    ll d = extendedEuclid(b%a , a, x1, y1);
    x = y1 - (ll)(b/a) * x1;
    y = x1;
    return d;
}
```

8.6 Modular Inverse 1 to N with Linear Time

```
vector<int> inverseArray(int n, int m) {
    vector<int> modInverse(n + 1,0);
    modInverse[1] = 1;
    for(int i = 2; i <= n; i++) {
        modInverse[i] = (-(m/i) * modInverse[m % i]) % m + m;
    }
    return modInverse;
}
```

8.7 Prime Sieve Linear

```
const int N = 10000000;
int lp[N+1];
vector<int> prm;
void sieveLinear(){
	for (int i=2; i<=N; ++i) {
		if (lp[i] == 0) {
			lp[i] = i;
			prm.push_back (i);
		}
		for (int j=0; j<(int)prm.size() && prm[j]<=lp[i] &&
		    i*prm[j]<=N; ++j)
			lp[i * prm[j]] = prm[j];
	}
}
```

8.8 nCr Precal

```
int nCr[MX];
void NCR() {
    for(int i=1; i<MX; i++) {
        nCr[i][0]=nCr[i][i]=1;
        for(int j=1; j<i; j++)
            nCr[i][j]=nCr[i-1][j]+nCr[i-1][j-1];
    }
}
```

8.9 segmented-sieve

```
/*
 * Firstly, generate all primes less than N
 */
int arr[SIZE];
///Returns number of primes between segment [a,b]
int segmentedSieve ( long long a, long long b ) {
    if ( a == 1 ) a++;
    int sqrtn = sqrt ( b );
    memset ( arr, 0, sizeof arr ); ///Make all index of arr 0.
    for ( int i = 0; i < prm.size() && prm[i] <= sqrtn; i++ ) {
        int p = prm[i];
        long long int j = p * p;
        ///If j is smaller than a, then shift it inside of segment [a,b]
        if ( j < a ) j = ( ( a + p - 1 ) / p ) * p;
        for ( ; j <= b; j += p ) {
            arr[j-a] = 1; ///mark them as not prime
        }
    }
    int res = 0;
    for ( long long i = a; i <= b; i++ ) {
        ///If it is not marked, then it is a prime
        if ( arr[i-a] == 0 ) res++;
    }
    return res;
}
```

9 Matrix

9.1 Gaussian Elimination

```
char ss[100];
struct frac{
	ll a,b;
	frac() { a=0; b=1; }
	frac(ll _a, ll _b){
		a=_a; b=_b;
		reduce();
	}
	frac operator + (const frac &p) const{
		ll ra= a*p.b + p.a*b;
		ll rb = b*p.b;
		return frac(ra,rb);
	}
	frac operator - (const frac &p) const{
		ll ra= a*p.b - p.a*b;
		ll rb = b*p.b;
		return frac(ra,rb);
	}
	frac operator * (const frac &p) const{
		ll ra= p.a*a;
		ll rb = b*p.b;
		return frac(ra,rb);
	}
	frac operator / (const frac &p) const{
```

```
        ll ra= a*p.b ;
        ll rb = b*p.a;
        return frac(ra,rb);
    }
    bool operator == (const frac &p) const{
        return a*p.b==p.a*b;
    }
    bool operator < (const frac &p) const{
        return a*p.b < p.a*b;
    }
    frac ABS(){
        return frac(abs(a),abs(b));
    }
    void print(){
        if (a==0) cout<<0;
        else if(b==1) cout<<a;
        else cout<<a<<"/"<<b;
    }
    void read(){
        scanf("%s",ss);
        bool flg=0;
        for(int i=0;ss[i] && !flg;i++){
            if(ss[i]=='/') flg=1;
        }
        if(flg){
            sscanf(ss,"%lld/%lld",&a,&b);
        }
        else{
            sscanf(ss,"%lld",&a);
            b=1;
        }
        reduce();
    }
    void reduce(){
        ll g=gcd(a,b);
        a/=g;
        b/=g;
        if(a==0) b=1;
        if(b<0) a=-a,b=-b;
    }
};
const frac ZERO = frac(0,1);
frac ar[55][55],res[55];
void fun(){
    for(int i=1,j=1;i<=n&&j<=m;){
        int idx=i;
        for(int k=i+1;k<=m;k++){
            if(!(ar[k][j]==ZERO)){
                idx=k;
                break;
            }
        }
        if(!(ar[idx][j]==ZERO)){
            for(int x=1;x<=n+1;x++){
                swap(ar[i][x],ar[idx][x]);
            }
            for(int x=1;x<=m;x++){
                if(x!=i){
                    for(int y=j+1;y<=n+1;y++)
                    {
                        ar[x][y]=ar[x][y]-ar[i][y]*(ar[x][j]/
                    }
                    ar[x][j]=ZERO;
                }
            }
            i++;
        }
        j++;
    }
    int rank=0;
    for(int i=1;i<=m;i++){
        bool zr=1;
        for(int j=1;j<=n;j++){
            if(!(ar[i][j]==ZERO)){
                zr=0;
                break;
            }
        }
        if(zr && !(ar[i][n+1]==ZERO)){
            rank=-1;
            break;
        }
    }
    if(rank>=0){
        for(int i=1,j=1;i<=m&&j<=n;){
            if(!(ar[i][j]==ZERO)){
                res[j]=ar[i][n+1]/ar[i][j];
                rank++;
                i++;
            }
```

```
                j++;
            }
        }
        if(rank==-1) puts("No Solution.");
        else if(rank<n){
            cout<<"Infinitely many solutions containing "<<n-rank<<"
                arbitrary constants."<<endl;
        }
        else{
            for(int i=1;i<=n;i++){
                cout<<"x["<<i<<"] = ";
                res[i].print();
                cout<<endl;
            }
        }
        return;
}
int main(){
        cin>>n>>m;
        for(int i=1;i<=m;i++){
            for(int j=1;j<=n+1;j++){
                ar[i][j].read();
            }
        }
        cout<<"Solution for Matrix System # "<<ts<<endl;
        fun();
        return 0;
}
```

9.2 Matrix Exponentiation

```
//http://zobayer.blogspot.com/2010/11/matrix-exponentiation.html
const ll MD=1e9;
const int mSize=2;
struct matrix{
    ll mat[mSize][mSize];
    void zero(){
        memset(mat,0,sizeof mat);
    }
    void identity(){
        for(int i=0; i<mSize; i++){
            for(int k=0; k<mSize; k++){
                mat[i][k]=(i==k);
            }
        }
    }
    friend matrix operator *(matrix a, matrix b){
        matrix result;
        result.zero();
        for(int i=0; i<mSize; i++){
            for(int k=0; k<mSize; k++){
                if(a.mat[i][k]){
                    for(int j=0; j<mSize; j++){
                        result.mat[i][j]+=a.mat[i][k]*b.mat[k][j];
                        result.mat[i][j]%=MD;
                    }
                }
            }
        }
        return result;
    }
    friend matrix operator^(matrix a,ll x){
        matrix result;
        result.identity();
        while(x){
            if(x&1) result=result*a;
            a=a*a;
            x/=2;
        }
        return result;
    }
};
int main(){
    matrix m,n,base,xy;
    base.mat[0][0]=1;
    base.mat[0][1]=1;
    base.mat[1][1]=0;
    base.mat[1][0]=1;
        n=(ba^MD);
        n*=xy
    return 0;
}
```

10 Misc

10.1 Monotonic Queue

```
struct monotonic_queue {
    int A[MX], B[MX],C[MX]; // normal , for min , for max
    int head_a,tail_a;
    int head_b,tail_b;
    int head_c,tail_c;
    void init() {
        head_a=tail_a=0;
        head_b=tail_b=0;
        head_c=tail_c=0;
    }
    void push(int v) {
        A[tail_a++] = v;
        // min-part
        while (head_b != tail_b && B[tail_b - 1] > v) tail_b--;
        B[tail_b++] = v;
        // max-part
        while (head_c != tail_c && C[tail_c - 1] < v) tail_c--;
        C[tail_c++] = v;
    }
    void pop() {

        if (head_a != tail_a) {
            if (A[head_a] == B[head_b]) ++head_b;// for min
            if (A[head_a] == C[head_c]) ++head_c;// for max
            ++head_a;// pop from main queue
        }
    }
    int get_min() {
        if (head_b == tail_b) return -1;
        return B[head_b];
    }
    int get_max() {
        if(head_c==tail_c) return -1;
        return C[head_c];
    }
    int Size() {
        return (tail_a-head_a);
    }
};
```

10.2 Sliding Window

```
int n,k,a[1000100];
deque< pair <int, int> >dq,qq;
void maxsld() {
    for(int i=0; i<n; i++) {
        while(!dq.empty()&& dq.back().first<=a[i]) {
            dq.pop_back();
        }
        dq.push_back(make_pair(a[i],i));
        while(!dq.empty()&&dq.front().second<(i-k+1)) {
            dq.pop_front();
        }
        if(i>=k-1) printf("%d ",dq.front().first);
    }
}
void minsld() {
    for(int i=0; i<n; i++) {
        while(!qq.empty()&&qq.back().first>=a[i]) {
            qq.pop_back();
        }
        qq.push_back(make_pair(a[i],i));
        while(!qq.empty()&&qq.front().second<(i-k+1)) {
            qq.pop_front();
        }
        if(i>=k-1) printf("%d ",qq.front().first);
    }
}
```

11 String

11.1 KMP

```
/*
 * http://www.iti.fh-flensburg.de/lang/algorithmen/pattern/kmpen.htm
 * */
int pre[2000101];
void prefixFunction(string s) {
    int j=0,ln=s.size();
    for(int i=1; i<ln; i++) {
        while(j && s[i]!=s[j]) j=pre[j-1];
        if(s[i]==s[j]) j++;
```

```
        pre[i]=j;
    }
}
void Failure(string s) {
    int j=0,ln=s.size(),i=1;
    pre[0]=-1;
    while(i<ln) {
        if(j>-1 && s[i]!=s[j]) j=pre[j];
        else {
            j++,i++;
            pre[i]=j;
        }
    }
}
int KMP(string s,string p) {
    int ret=0,j=0,ln=s.size();;
    prefixFunction(p);
    for(int i=0; i<ln; i++) {
        while(j && s[i]!=p[j]) j=pre[j-1];
        if(s[i]==p[j]) j++;
        if(j==(int)p.size()) ret++;
    }
    return ret;
}
```

11.2 Manacher

```
string preProcess(string s){
    int n = s.length();
    if (n == 0) return "^$";
    string ret = "^";
    for (int i = 0; i < n; i++)
        ret += "#" + s.substr(i, 1);
    ret += "#$";
    return ret;
}
int longestPalindrome(string s){
    string T = preProcess(s);
    int n = T.length();
    int *P = new int[n];
    int C = 0, R = 0;
    for (int i = 1; i < n-1; i++){
        int i_mirror = 2*C-i;
        P[i] = (R > i) ? min(R-i, P[i_mirror]) : 0;
        while (T[i + 1 + P[i]] == T[i - 1 - P[i]])
            P[i]++;
        if (i + P[i] > R){
            C = i;
            R = i + P[i];
        }
    }
    int maxLen = 0;
    for (int i = 1; i < n-1; i++){
        if (P[i] > maxLen){
            maxLen = P[i];
        }
    }
    delete[] P;
    return (maxLen);
}
```

11.3 Suffix Array

```
/*
 * lcp return the longest common prefix legnth
 * between two string.
 * */
#define F first
#define S second
const int N = (int)5*(1e4 + 7);
string s;
pair<pair<int,int>,int > Suffix[N],tmpSuffix[N];
int Rank[20][N],stp;
void SortSuffix(int n){
    for(int i=0; i<=n; i++){
        Suffix[i].F.F = (int)s[i];
        Suffix[i].F.S = 0;
        Suffix[i].S = i;
    }
    stp = 0;
    sort(Suffix,Suffix+n+1);
    for(int i = 0; i <= n; i++){
        if( !i ) Rank[i][Suffix[i].S] = i;
        else if ( Suffix[i].F != Suffix[i-1].F ){
            Rank[0][Suffix[i].S] = i;
        }
```

```
        else Rank[0][Suffix[i].S] = Rank[0][Suffix[i-1].S];
    }
    for(stp = 1; ; stp++){
        for(int i = 0; i <= n; i++) {
            int firstPart = Suffix[i].S;
            int secondPart = Suffix[i].S + (1 << (stp-1));
            tmpSuffix[i].F.F = Rank[stp-1][firstPart];
            if(secondPart <= n-1) tmpSuffix[i].F.S =
                Rank[stp-1][secondPart];
            else tmpSuffix[i].F.S = 0;
            tmpSuffix[i].S = -firstPart;
        }
        sort(tmpSuffix, tmpSuffix+n+1);
        for(int i = 0; i <= n; i++){
            Suffix[i] = tmpSuffix[i];
            Suffix[i].S *= -1;
            if( !i ) Rank[stp][Suffix[i].S] = i;
            else if(Suffix[i].F != Suffix[i-1].F) {
                Rank[stp][Suffix[i].S] = i;
            }
            else Rank[stp][Suffix[i].S] = Rank[stp][Suffix[i-1].S];
        }
        if( (1 << stp) >= n) break;
    }
}
int lcp(int x,int y,int n) {
    int ans = 0;
    if( x == y ) return n-x;
    for(int j = stp-1; j >= 0 && x < n; j--){
        if(Rank[j][x] == Rank[j][y]){
            x += (1 << j);
            y += (1 << j);
            ans += (1 << j);
        }
    }
    return ans;
}
int main(){
    cin>>s;
    int l = s.size();
    SortSuffix(l);
    for(int i = 0; i <= l; i++)
    {
        int a = (i>0)?lcp(Suffix[i].S, Suffix[i-1].S,l):0;
    }
    return 0;
}
```