

Contents

1 Java	
2 Math	
2.1 Math Basic	2
2.2 Euler Function	2
2.3 Extended Euclidean	2
2.4 China Remain Theorm	2
2.5 Counting	2
2.6 Miller Rabin	2
2.7 Pollard rho	3
2.8 Linear Algebra	3
2.9 FFT	4
2.10Hash	4
2.11NIM	4
3 String	
3.1 KMP	4
3.2 LPS	4
3.3 AC Automation	5
3.4 Z	5
3.5 Suffix Array	5
3.6 BWT	6
4 Tree	
4.1 Tree Min Vertex Cover	6
4.2 Treap ordered	6
4.3 Treap unordered	7
4.4 Tree Heavy Light Decomposition	8
5 Graph	
5.1 Biconnected Components	8
5.2 Strong Connected Components	9
5.3 2 SAT	9
5.4 System of Difference Constraints	9
5.5 Bipartite: MaxMatch, MinVerCover, MaxIndSet	10
5.6 Bipartite: KM	10
5.7 Min Vertex Cover	11
5.8 Max Independent Set	11
5.9 Max Clique	11
5.10Kirchhoff's theorem	11
5.11Popular matching	11
5.12Blossom Algorithm	12
6 Flow	
6.1 Dinic Maxflow Mincut	12
7 Geometry	
7.1 Geometry basic	13
7.2 Minimal Enclouse Disk	13
7.3 2D Convex Hull	14
7.4 Closest Point	14
7.5 Min Max Triangle	14

```
set nocompatible
```

```
set enc=utf-8
set fenc=utf-8
```

```
set tabstop=4
set softtabstop=4
set shiftwidth=4
set backspace=2
set autoindent
set cindent
```

```
syntax on
set t_Co=256
set number
set showmatch
set hls
```

```
autocmd FileType cpp nnoemap <F9> :w <bar> :! g++ % -std=c++11 -
02 -Wall && ./a.out<CR>
```

1 Java

```
class Scan {
    BufferedReader buffer;
    StringTokenizer tok;
    Scan() {
        buffer = new BufferedReader(new InputStreamReader(System.in))
        ;
    }
    boolean hasNext() {
```

```
        while(tok == null || !tok.hasMoreElements())
            try {
                tok = new StringTokenizer(buffer.readLine());
            } catch (Exception ex) {
                return false;
            }
        return true;
    }
    String next() {
        if(hasNext())
            return tok.nextToken();
        return null;
    }
    String nextLine() {
        if(hasNext())
            return tok.nextToken("\n");
        return null;
    }
    int nextInt() {
        return Integer.parseInt(next());
    }
}

/* Compile: javac %
* Run: java [Class name] */
import java.util.*;
import java.lang.*;
import java.math.*;

class Main {
    public static void main (String[] args) {
        System.out.print(1);
        System.out.print(2);
        System.out.println("Hello World");
        System.out.printf("%.2f", 0.12345);

        Scanner sc = new Scanner(System.in);
        System.out.println(sc.nextLine()); //gets()
        System.out.println(sc.next()); //scanf("%s")
        System.out.println(sc.nextInt());
        System.out.println(sc.nextDouble());
        while(sc.hasNext()) { //EOF
            int a = sc.nextInt();
            System.out.println(a);
        }

        int[] a = {1,2,3};
        int[][] b = {{1,2},{3,4,5}};
        double[] c = new double[90];
        System.out.print(b[0][1]);
        System.out.print(b[1][2]);

        int[] d = {5,2,1,3,4};
        Integer[] e = {6,3,4,1,2};
        Arrays.sort(d);
        Arrays.sort(e, new MyCom());
        for(int i=0; i<d.length; i++) {
            System.out.print(d[i]);
        }
        for(int i=0; i<e.length; i++) {
            System.out.print(e[i]);
        }

        Set<String> s = new HashSet<String>(); //or TreeSet
        s.add("123");
        s.add("234");
        System.out.println(s);
        System.out.println(s.contains("123"));
        Map<String, Integer> m = new TreeMap<String, Integer>();
        m.put("haha", 123);
        m.put("hehe", 234);
        System.out.println(m);

        BigInteger b1 = new BigInteger("-1231237182379123712");
        BigInteger b2 = BigInteger.valueOf(234);

        System.out.println(b1.add(b2));
        System.out.println(b1.mod(b2));

        int z = Integer.parseInt("-123");
        System.out.println(z);

        System.out.println(Math.PI);
        System.out.println(Math.sin(1));
    }

    static class MyCom implements Comparator<Integer> {
        public int compare(Integer i1, Integer i2) {
            return i2 - i1;
        }
    }
}
```

```
    }
}
```

2 Math

2.1 Math Basic

```
vector<pii> primeFac(int n) {
    vector<pii> ret;
    for(int i=2; n>1; ++i){
        if( n%i != 0 ) continue;
        int e = 0;
        while( n%i == 0 ) ++e , n/=i;
        ret.push_back({i, e});
    }
    return ret;
}

long long fastPow(long long x, int n, long long m){
    long long ans = 1LL;
    while( n ){
        if( n&1 ) ans = ans * x % m;
        x = x*x % m;
        n >>= 1;
    }
    return ans;
}

long long modInv(long long x, long long p){
    return fastPow(x, p-2, p);
}

long long modInv_euler(long long x, long long m){
    // must be gcd(x,m)==1
    // phi is euler function: O(sqrt(x))
    return fastPow(x, phi(m)-1, m);
}

long long gt(long long a, long long b) {
    // smallest integer greater than a/b
    long long ret = a/b;
    if( ret>0 || a%b==0 ) ++ret;
    return ret;
}
}
```

2.2 Euler Function

```
int phi(int n){
    // euler function: in [0,n], # of coprime(i, n)
    vector<pii> fac = primeFac(n);
    int num = 1 , m = 1;
    for(auto &p : fac)
        num *= (p.first-1) , m *= p.first;
    return n/m * num;
}
}
```

2.3 Extended Euclidean

```
pll recur(long long n, long long m) {
    // solve one integer solution of
    // x*n + y*m = gcd(n,m)
    if( n%m == 0 )
        return {0LL, 1LL};
    pll res = recur(m, n%m);
    pll ret = {res.second, res.first - res.second * (n/m)};
    return ret;
}
}
```

2.4 China Remain Theorem

```
bool china_solvable(vector<pii> &rule) {
    for(int i=0; i<rule.size(); ++i)
        for(int j=1; j<rule.size(); ++j) {
            int gcd = __gcd(rule[i].second, rule[j].second);
            if( rule[i].first%gcd != rule[j].first%gcd )
                return false;
        }
    return true;
}

long long china(const vector<pii> &rule, int nlt=0){
    // solve x = ai (mod mi)
    // rule should solvable
    long long MM = 1LL;
    for(auto &r : rule)
        MM = lcm(MM, r.second);
    long long x = 0LL;
    for(auto &r : rule){
        long long ai = r.first;
        long long mi = r.second;
```

```
        long long Mi = MM / r.second;
        long long Mv = modInv_euler(Mi%mi, mi);
        long long tmp = ai*Mi%MM *Mv %MM;
        x = (x+tmp) % MM;
    }
    if( x>=nlt ) return x;
    long long n = ceil((nlt-x)*1.0/MM);
    return x + n*MM;
}
```

2.5 Counting

```
const int MaxNum = 1000004;
const int modNum = 1000000009;
long long fac [MaxNum];
long long facIv[MaxNum];
void initFac(){
    fac[0] = facIv[0] = 1LL;
    for(int i=1; i<MaxNum; ++i) {
        fac [i] = fac[i-1]*i % modNum;
        facIv[i] = modInv(fac[i], modNum);
    }
}

long long Cnm(int n, int m){
    if( m==0 || n==m ) return 1LL;
    return fac[n]*facIv[m] % modNum *facIv[n-m] % modNum;
}

long long nBlock_kColor(int n,int k){
    // n different blocks; k different colors
    // use inclusion-exclusion principle
    long long ans = fastPow(k, n, modNum);
    bool del = true;
    for(int i=k-1; i>0; --i, del=!del){
        long long now = Cnm(k, i)*fastPow(i, n, modNum) %modNum;
        if( del ) ans = (ans+modNum-now) % modNum;
        else ans = (ans+now) % modNum;
    }
    return ans;
}
}
```

2.6 Miller Rabin

```
#include <climits>
typedef unsigned long long int ull;

ull bases[20] = { 2ULL, 3ULL,5ULL,7ULL,11ULL,13ULL,17ULL,19ULL,23
    ULL,29ULL,31ULL,37ULL };

ull fake_mul(ull n, ull m, ull x);
ull fast_pow(ull n, ull p, ull x);

bool is_prime(ull n)
{
    if (n < 2ULL) return false;

    for (int tt = 0; tt < 12; tt++) {
        ull a;
        a = bases[tt] % n;

        if (a == 0 || a == 1 || a == n - 1) {
            continue;
        }

        int t = 0;
        ull u = n - 1ULL;
        while ((u & 1ULL) == 0ULL) u >>= 1, t++;

        ull x = fast_pow(a, u, n); // x = a ^ u % n;
        if (x == 1ULL || x == (n - 1)) continue;
        for (int i = 0; i < t - 1; i++)
        {
            if (ULLONG_MAX / x < x) {
                x = fake_mul(x, x, n);
            }
            else {
                x = x*x%n;
            }
            if (x == 1) return false;
            if (x == n - 1) break;
        }
        if (x == n - 1) continue;
        return false;
    }
    return true;
}

ull fake_mul(ull n, ull m, ull x)
{
}
```

```

ull re = 0ULL;
while (m != 0ULL) {
    if ((m & 1ULL) != 0ULL) {
        if (ULLONG_MAX - re < n) {
            ull temp = ULLONG_MAX%x;

            temp += (n - (ULLONG_MAX - re)) % x;
            re = temp%x;
        }
        else {
            re = (re + n) % x;
        }
    }

    if (ULLONG_MAX - n < n) {
        ull temp = ULLONG_MAX%x;

        temp += (n - (ULLONG_MAX - n)) % x;
        n = temp%x;
    }
    else {
        n = n + n%x;
    }

    m >>= 1;
}
return re;
}

ull fast_pow(ull n, ull p, ull x)
{
    ull re = 1ULL;

    while (p != 0ULL) {
        if ((p & 1ULL) != 0ULL) {
            if (ULLONG_MAX / re < n) {
                re = fake_mul(n, re, x);
            }
            else {
                re = (re*n) % x;
            }
        }

        if (ULLONG_MAX / n < n) {
            n = fake_mul(n, n, x);
        }
        else {
            n = (n*n) % x;
        }
        p >>= 1;
    }
    return re;
}

// Below is non-extreme version
ull fake_mul(ull n, ull m, ull x) {
    ull re = 0ULL;
    n %= x, m %= x;
    while (m) {
        if (m&1ULL)
            re = (re+n) % x;
        n = (n+n) % x;
        m >>= 1;
    }
    return re;
}

ull fast_pow(ull n, ull p, ull x) {
    ull re = 1ULL;
    while (p) {
        if (p&1ULL)
            re = fake_mul(re,n,x);
        n = fake_mul(n,n,x);
        p >>= 1;
    }
    return re;
}

bool is_prime(ull n) {
    static const int bNum = 12;
    static const ull bases[bNum] = {
        2ULL,3ULL,5ULL,7ULL,11ULL,13ULL,17ULL,19ULL,23ULL,29ULL,31ULL
        ,37ULL
    };
    if (n<=2ULL) return n==2ULL;
    if (!(n&1ULL)) return false;

    ull u = n-1;
    while (! (u&1ULL))
        u >>= 1;
    for (int i=0; i<bNum; i++) {
        if (bases[i]%n == 0) continue;

```

```

        ull t = u;
        ull a = fast_pow(bases[i], t, n);
        if (a==1 || a==n-1) continue;
        while (t!=n-1 && a!=1 && a!=n-1) {
            a = fake_mul(a,a,n);
            t <<= 1;
        }
        if (t==n-1 && a==1) continue;
        if (a!=n-1) return false;
    }
    return true;
}

```

2.7 Pollard rho

```

// need fack_mul, is_prime
ull gcd(ull a, ull b) {
    return (a%b==0)? b : gcd(b, a%b);
}

ull dif(ull a, ull b) {
    return a>b? a-b : b-a;
}

void pollard_rho(ull n, map<ull,int> &facs) {
    while( !(n&1ull) ) {
        // must extract factor 2
        int cnt = 0;
        while( !(n&1ull) )
            ++cnt, n>>=1;
        facs[2] = cnt;
    }
    if (n==1ull) return;
    if (is_prime(n)) {
        facs[n]++;
        return;
    }
    ull x = rand()%n;
    ull y = x;
    ull a = rand()%(n-1) + 1;
    ull g = 1ull;
    while( g==1ull ) {
        x = (fake_mul(x,x,n) + a) %n;
        y = (fake_mul(y,y,n) + a) %n;
        y = (fake_mul(y,y,n) + a) %n;
        if (x==y) {
            g = n;
            break;
        }
        g = gcd(dif(x,y), n);
    }
    if (g==n) // unluck try again
        pollard_rho(n, facs);
    else if (g>1ull) { // luck, found g
        pollard_rho(g, facs);
        pollard_rho(n/g, facs);
    }
}

```

2.8 Linear Algebra

```

#ifndef _MATRIX_H_
#define _MATRIX_H_

#include <iostream>
#include <vector>
using namespace std;

template <class T>
class Matrix{
public:
    int rSize, cSize;
    vector< vector<T> > mat;

    Matrix(int r = 0, int c = 0) :rSize(r), cSize(c), mat(rSize,
        vector<T>(cSize)){}
    vector<T>& operator[](int i) {
        return mat[i];
    }
    void print();
};

template <class T>
void Matrix<T>::print() {
    cout << "Matrix elements:" << endl;
    for (int i = 0; i < rSize; i++) {
        cout << "[";
        for (int j = 0; j < cSize; j++) {
            cout << "\t" << mat[i][j];
            if (j != cSize - 1)cout << ",";

```

```

    }
    cout << " ]" << endl;
}
}

#endif

#include "Matrix.h"

template <class T>
Matrix<T> matMul(Matrix<T> matA, Matrix<T> matB){
    Matrix<T> matRe(matA.rSize, matB.cSize);

    for (int i = 0; i < matRe.rSize; i++) {
        for (int j = 0; j < matRe.cSize; j++) {
            matRe[i][j] = 0;
            for (int k = 0; k < matA.cSize; k++) {
                matRe[i][j] += matA[i][k] * matB[k][j];
            }
        }
    }
    return matRe;
}

```

2.9 FFT

```

const double PI = acos(-1.0);
typedef complex<double> Complex;

void _fft(const Complex *in, Complex *out, const vector<Complex>
    &pls, int step, int size, double dir) {
    if(size < 1) return;
    if(size == 1) {
        out[0] = in[0];
        return;
    }
    _fft(in, out, pls, step * 2, size / 2, dir);
    _fft(in + step, out + size / 2, pls, step * 2, size / 2, dir);
    for(int i = 0; i < size / 2; ++i) {
        Complex even = out[i];
        Complex odd = out[i + size / 2];
        out[i] = even + pls[i * step] * odd;
        out[i + size / 2] = even + pls[(i + size / 2) * step] *
            odd;
    }
}

// x.size() must be 2^m
vector<Complex> fft(const vector<Complex> &x, bool isInv) {
    int n = x.size();
    double dir = isInv ? -1. : 1.;
    vector<Complex> pls(n);
    for(int i=0; i<n; ++i)
        pls[i] = polar(1.0, dir * 2 * PI * i / n);

    vector<Complex> out = x;
    _fft(&x[0], &out[0], pls, 1, n, dir);
    for(int i = 0; isInv && i < x.size(); i++)
        out[i] /= n;
    return out;
}

```

2.10 Hash

```

#include <iostream>
#include <cstdio>
#include <cstring>
#include <algorithm>
using namespace std;
const int hashRange = 131072;
int hashtable[4][256], shuffleArr[hashRange], ref[hashRange];
void buildHashTable() {
    memset(ref, -1, sizeof(ref));
    for(int i = 0; i < hashRange; i++)
        shuffleArr[i] = i;
    for(int i = 0; i < 4; i++) {
        random_shuffle(shuffleArr, shuffleArr + hashRange);
        for(int j = 0; j < 256; j++)
            hashtable[i][j] = shuffleArr[j];
    }
}

int myhash(int input) {
    int x[4];
    for(int i = 0; i < 4; i++)
        x[i] = hashtable[i][(input >> (i*8)) & (0xff)];
    int k = x[0] ^ x[1] ^ x[2] ^ x[3];
    if(ref[k] != input)

```

```

        for(int i = k; ; i = (i+1)%hashRange)
            if(ref[i] == -1 || ref[i] == input) {
                ref[i] = input;
                return i;
            }
    return k;
}

```

2.11 NIM

Requirement:

- Two player take turns
- In the same state, the action both players can take are the same
- Game must end in finite turn
- Lose if one player can not move in his/her turn

Solution:

- Using SG value to represent a game
- SG value of multiple game are the **xor** of all of them
- SG value = 0 iff first move lose. Vice versa.

Find SG value of each game:

- 0 when no move can be taken in the game
- Find the smallest non negative integer that is **not** belong to "All the SG value after one move"

3 String

3.1 KMP

```

class kmp{
private:
    int prefix[maxLen];
    char pat[maxLen];
public:
    void setPattern(const char *str){
        strcpy(pat, str);
        prefix[0] = -1;
        int i=1, j=0;
        for( ; str[i]!='\0' ; ++i, ++j){
            if( str[i]==str[j] )
                prefix[i] = prefix[j];
            else
                prefix[i] = j;
            while( j>=0 && str[j]!=str[i] )
                j = prefix[j];
        }
        prefix[i] = j;
    }
    int search(const char *str){
        // return index of str match pattern
        int i=0, j=0;
        for( ; str[i]!='\0' && pat[j]!='\0' ; ++i, ++j){
            while( j>=0 && pat[j]!=str[i] )
                j = prefix[j];
        }
        if( pat[j]=='\0' )
            return i-j;
        return -1;
    }
    int countMatched(const char *str){
        // return # of pattern in str
        int cnt = 0;
        int i=0, j=0;
        while( true ){
            if( pat[j]=='\0' ) ++cnt;
            if( str[i]=='\0' ) break;
            while( j>=0 && pat[j]!=str[i] )
                j = prefix[j];
            ++i, ++j;
        }
        return cnt;
    }
};

```

3.2 LPS

```

int lps(const char *str){
    // return len of longest palindrom substring
    static char emptyChar = '\0';
    static char tmp[maxLen*2];
    static int lprb[maxLen*2];
    // [i-lprb[i], i+lprb[i]] is the lps when mid is i
    for(int i=0, j=-1; true; ++i){
        if( str[i]=='\0' ){

```

```

    tmp[++j] = emptyChar;
    tmp[++j] = '\0';
    break;
}
tmp[++j] = emptyChar;
tmp[++j] = str[i];
}
lprb[0] = 0;
int rightBorder = 0, midId = 0;
for(int i=1; tmp[i]!='\0'; ++i){
    if( i>rightBorder ){
        rightBorder = i;
        midId = i;
        lprb[i] = 0;
    }
    int mirId = midId - (i-midId);
    if( i+lprb[mirId] > rightBorder )
        lprb[i] = rightBorder - i;
    else if( i+lprb[mirId] < rightBorder )
        lprb[i] = lprb[mirId];
    else{
        int j=lprb[mirId];
        while( tmp[i+j]!='\0' && i-j>=0 && tmp[i+j]==tmp[i-j])
            ++j;
        rightBorder = i+j-1;
        midId = i;
        lprb[i] = j-1;
    }
}
int ans = 1;
for(int i=0; tmp[i]!='\0'; ++i)
    if( lprb[i]>ans )
        ans = lprb[i];
return ans;
}

```

3.3 AC Automation

```

#include <queue>
#include <cstdio>
#include <cstring>
using namespace std;
struct AC_algorithm {
    struct node {
        static const int signNum = 52; //number of kind of character
        node *ch[signNum];
        node *suffix, *dict;
        int index;
        node() {
            memset(ch, 0, sizeof(ch));
            suffix = dict = 0;
            index = -1;
        }
    };

    static const int stringNum = 100010; //number of pattern
    node *root;
    int occur[stringNum]; //string i occur occur[i] times
    int reflect[stringNum]; //string i is the same as string
    reflect[i];

    AC_algorithm() {
        root = new node();
        memset(occur, 0, sizeof(occur));
        memset(reflect, -1, sizeof(reflect));
    }

    int decode(char c) { //decode char
        return c <= 'Z' ? (c - 'A') : (c - 'a' + 26);
    }

    void insert(char *s, int index) { //add string to trie
        node *p = root;
        for(; *s; s++) {
            int code = decode(*s);
            if(p->ch[code] == NULL)
                p->ch[code] = new node();
            p = p->ch[code];
        }
        if(p->index == -1)
            p->index = index;
        else
            reflect[index] = p->index;
    }

    void build() { //build machine
        queue<node*> q;
        q.push(root);
        while(!q.empty()) {

```

```

            node *p = q.front();
            for(int i = 0; i < node::signNum; i++)
                if(p->ch[i]) {
                    node *tmp = p->suffix;
                    while(tmp && !tmp->ch[i]) tmp = tmp->suffix;
                    if(tmp)
                        p->ch[i]->suffix = tmp->ch[i];
                    else
                        p->ch[i]->suffix = root;
                    tmp = p->ch[i]->suffix;
                    if(tmp->index != -1)
                        p->ch[i]->dict = tmp;
                    else
                        p->ch[i]->dict = tmp->dict;
                    q.push(p->ch[i]);
                }
            q.pop();
        }
    }

    void match(char *s) { //match patterns with Text
        node *p = root;
        for(; *s; s++) {
            int code = decode(*s);
            while(p && !p->ch[code]) p = p->suffix;
            if(p)
                p = p->ch[code];
            else
                p = root;
            node *tmp = p;
            while(tmp) {
                if(tmp->index != -1)
                    occur[tmp->index]++;
                tmp = tmp->dict;
            }
        }
    }

    ~AC_algorithm() {
        queue<node*> q;
        q.push(root);
        while(!q.empty()) {
            node *p = q.front();
            q.pop();
            for(int i = 0; i < node::signNum; i++)
                if(p->ch[i])
                    q.push(p->ch[i]);
            delete p;
        }
    }
};

```

3.4 Z

```

#include <cstring>
int z[length];
void z_function(char *str) {
    int len = strlen(str), L = 0, R = 1;
    z[0] = len;
    for(int i = 1; i < len; i++)
        if(R <= i || z[i-L] >= R-i) {
            int x = max(R, i);
            while(x < len && str[x] == str[x-i])
                x++;
            z[i] = x-i;
            L = i; R = x;
            //if(i < x) {L = i; R = x;}
        } else
            z[i] = z[i-L];
}

```

3.5 Suffix Array

```

vector<int> buildSuffixArray(const vector<int> &str, int
    endOfString=-1) {
    // sa: i -> start position of str
    // 0(n*lg n*lg n). probably faster than 0(n*lg n) version
    int len = str.size();
    vector<int> sa(len+1), rank(len+1);
    for(int i=0; i<len; ++i) rank[sa[i] = i] = str[i];
    rank[sa.back() = len] = endOfString;
    for(int ll=1, cnt=0; cnt!=len; ll<=1, cnt=rank[sa.back()]) {
        auto cmp = [&](const int l, const int r) {
            if( rank[l]!=rank[r] ) return rank[l] < rank[r];
            int lv = (l+ll < len) ? rank[l+ll] : 0;
            int rv = (r+ll < len) ? rank[r+ll] : 0;
            return lv < rv;
        };

```

```

    sort(sa.begin(), sa.end(), cmp);
    vector<int> tmp = rank;
    tmp[sa[0]] = 0;
    for(int i=1; i<sa.size(); ++i)
        tmp[sa[i]] = tmp[sa[i-1]] + cmp(sa[i-1], sa[i]);
    rank = tmp;
}
return sa;
}

vector<int> buildLcp(const vector<int> &str, const vector<int> &
sa) {
    // lcp: longest common prefix for sa[i-1] sa[i]
    int len = sa.size();
    vector<int> lcp(len, 0), idx(len);
    for(int i=0; i<len; ++i) idx[sa[i]] = i;
    for(int i=0, l=0; i<len; ++i) {
        if( idx[i] == 0 ) {
            l = 0;
            continue;
        }
        int j = sa[idx[i]-1];
        while( i+l<len && j+l<len && str[i+l]==str[j+l] )
            ++l;
        lcp[idx[i]] = l;
        l -= l>0;
    }
    return lcp;
}

pair<int,int> longestRepeatedSubstring(const vector<int> &sa,
const vector<int> &lcp, int k) {
    // longest repeated substring who occurs at least k times
    // return <longest length, start position>
    pair<int,int> ret = {-1, -1};
    if( k<=1 ) {
        for(int i=0; i<sa.size(); ++i)
            if( sa[i]==0 ) return {sa.size()-1, i};
    }
    if( lcp.size() < k ) return ret;
    deque<pair<int,int>> dq;
    auto maintain = [&](pair<int,int> v) {
        while( dq.size() && dq.front().second <= v.second-(k-1) )
            dq.pop_front();
        while( dq.size() && dq.back() >= v )
            dq.pop_back();
        dq.push_back(v);
    };
    for(int i=0; i<k-2; ++i) maintain({lcp[i], i});
    for(int i=k-2; i<lcp.size(); ++i) {
        maintain({lcp[i], i});
        ret = max(ret, dq.front());
    }
    return ret;
}

pair<int,int> lcs(const vector<int> &s1, const vector<int> &s2,
int e1, int e2) {
    // Longest Common "Substring" in O(n*lg n) n = s1+s2
    vector<int> arr(s1.size()+s2.size()+2);
    int idx = -1;
    for(int i=0; i<s1.size(); ++i) arr[++idx] = s1[i];
    arr[++idx] = e1;
    for(int i=0; i<s2.size(); ++i) arr[++idx] = s2[i];
    arr[++idx] = e2;
    vector<int> sa = buildSuffixArray(arr);
    vector<int> lcp = buildLcp(arr, sa);
    pair<int,int> ret = {-1, -1};
    for(int i=1; i<lcp.size(); ++i)
        if( (sa[i]<=s1.size()) ^ (sa[i-1]<=s1.size()) )
            ret = max(ret, {lcp[i], sa[i]});
    return ret;
}

```

3.6 BWT

```

vector<int> btwEncode(const vector<int> &src) {
    // O(n*lg n). probably faster than O(n*lg n) version
    int len = src.size();
    vector<int> sa(len), rank(len);
    for(int i=0; i<len; ++i) rank[sa[i]] = i;
    for(int ll=1, cnt=0; cnt!=len; ll<=1, cnt=rank[sa.back()]+1)
    {
        auto cmp = [&](const int l, const int r) {
            if( rank[l]!=rank[r] ) return rank[l] < rank[r];
            return rank[(l+ll)%len] < rank[(r+ll)%len];
        };
        sort(sa.begin(), sa.end(), cmp);
        vector<int> tmp = rank;

```

```

        tmp[sa[0]] = 0;
        for(int i=1; i<sa.size(); ++i)
            tmp[sa[i]] = tmp[sa[i-1]] + cmp(sa[i-1], sa[i]);
        rank = tmp;
    }
    vector<int> rst(len);
    for(int i=0; i<len; ++i) rst[i] = src[(sa[i]+len-1)%len];
    return rst;
}

vector<int> btwDecode(const vector<int> &rst) {
    int len = rst.size();
    vector<pair<int,int>> pre(len);
    for(int i=0; i<len; ++i) pre[i] = {rst[i], i};
    sort(pre.begin(), pre.end());
    vector<int> table(len);
    for(int i=0; i<len; ++i) table[pre[i].second] = i;
    vector<int> src(len);
    for(int i=rst.size()-1, idx=0; i>=0; --i, idx=table[idx])
        src[i] = rst[table[idx]];
    return src;
}

```

4 Tree

4.1 Tree Min Vertex Cover

```

class TreeMinVertexCover {
private:
    static const int maxNum = 100004;
    vector<int> G[maxNum];
    int in[maxNum];
public:
    bool pick[maxNum];
    int MVC; // min vertex cover
    void init() {
        for(int i=0; i<maxNum; ++i)
            G[i].clear();
        memset(in, 0, sizeof(in));
    }
    void addEdge(int u, int v) {
        G[u].emplace_back(v);
        G[v].emplace_back(u);
        ++in[u];
        ++in[v];
    }
    int treeMinVertexCover() {
        memset(pick, 0, sizeof(pick));
        MVC = 0;
        queue<int> myQ;
        for(int i=1; i<=maxNum; ++i)
            if( in[i]==1 ) myQ.push(i);
        while( myQ.size() ) {
            int nowAt = myQ.front();
            myQ.pop();
            if( in[nowAt]==0 ) continue;
            ++MVC;
            int id;
            for(int i=0; i<G[nowAt].size(); ++i)
                if( in[G[nowAt][i]] ) {
                    id = G[nowAt][i];
                    break;
                }
            for(int i=0; i<G[id].size(); ++i)
                if( in[G[id][i]] ) {
                    --in[G[id][i]];
                    --in[id];
                    if( in[G[id][i]]==1 )
                        myQ.push(G[id][i]);
                }
            }
        return MVC;
    }
};

```

4.2 Treap ordered

```

struct node {
    int v, p, sz;
    node *l, *r;
    node() { l = r = NULL; }
    node(int v_):v(v_),p(rand()),sz(1) { l = r = 0; }
    int size() {
        return this!=NULL ? this->sz : 0;
    }
    void maintain() {
        sz = l->size() + r->size() + 1;
    }
};

```

```

    }
} ;
void splite_v(node *t,node* &a,node* &b,int v) {
    if(!t)
        a = b = NULL ;
    else if(v >= t->v) {
        a = t ;
        splite_v(t->r,a->r,b,v) ;
        a->maintain() ;
    } else if(v < t->v) {
        b = t ;
        splite_v(t->l,a,b->l,v) ;
        b->maintain() ;
    }
}
node* merge(node *a,node *b) {
    if(a==NULL || b==NULL)
        return a!=NULL ? a : b ;
    if(a->p > b->p) {
        a->r = merge(a->r,b) ;
        a->maintain() ;
        return a ;
    } else if(a->p <= b->p) {
        b->l = merge(a,b->l) ;
        b->maintain() ;
        return b ;
    }
}
int kth(node *t,int k) {
    if(k<=t->l->size())
        return kth(t->l,k) ;
    else if(k>t->l->size()+1)
        return kth(t->r,k-t->l->size()-1) ;
    return t->v ;
}
void release(node *t) {
    if(t) {
        release(t->l) ;
        release(t->r) ;
        delete t ;
    }
}
}

```

4.3 Treap unordered

```

#include <iostream>
#include <cstdio>
#include <stdlib.h>
#include <cstring>
using namespace std;
const int oo = 1e9 ;
struct node {
    int v , p , sz ;
    int sum , presum , sufsum , maxsum , flag , set ;
    node *l , *r ;
    node(){}
    node(int v_) : p(rand()), sz(1), l(NULL), r(NULL) {
        v = sum = presum = sufsum = maxsum = v_ ;
        flag = 0 ;
        set = oo ;
    }
    int size() { return this ? sz : 0 ; }
    int Sum() { return this ? sum : 0 ; }
    int Presum() { return this ? (!flag ? presum : sufsum) : -oo ; }
    int Sufsum() { return this ? (!flag ? sufsum : presum) : -oo ; }
    int Maxsum() { return this ? maxsum : -oo ; }
    int max(int a,int b) { return a > b ? a : b ; }
    int max(int a,int b,int c) { return max(a,max(b,c)) ; }
    void makesame(int st) {
        if(this) {
            set = st ;
            sum = st*sz ;
            presum = sufsum = maxsum = (st <= 0 ? st : sum) ;
        }
    }
    void pushdown() {
        if(flag) {
            if(l) l->flag = !l->flag ;
            if(r) r->flag = !r->flag ;
            swap(l,r) ;
            swap(presum,sufsum) ;
            flag = 0 ;
        }
        if(set!=oo) {
            v = set ;
            l->makesame(set) ;
            r->makesame(set) ;
        }
    }
}

```

```

        set = oo ;
    }
}
void maintain() {
    presum = max(l->Presum(), l->Sum() + v, l->Sum() + v + r->Presum()) ;
    sufsum = max(r->Sufsum(), r->Sum() + v, r->Sum() + v + l->Sufsum()) ;
    int maxsum1 = max(l->Maxsum(), r->Maxsum(), v) ;
    int maxsum2 = max(l->Sufsum() + v, r->Presum() + v, l->Sufsum() + v + r->Presum()) ;
    maxsum = max(maxsum1, maxsum2) ;
    sum = l->Sum() + r->Sum() + v ;
    sz = 1 + l->size() + r->size() ;
}
}
void splite(node *t,node* &a,node* &b,int k) {
    if(t == NULL) {
        a = b = NULL ;
        return ;
    }
    t->pushdown() ;
    if(t->l->size()+1 <= k) {
        a = t ;
        splite(t->r, a->r, b, k-(t->l->size()+1)) ;
        a->maintain() ;
    } else {
        b = t ;
        splite(t->l,a,b->l,k) ;
        b->maintain() ;
    }
}
node* merge(node *a,node *b) {
    if(!a || !b)
        return a ? a : b ;
    if(a->p > b->p) {
        a->pushdown() ;
        a->r = merge(a->r,b) ;
        a->maintain() ;
        return a ;
    } else {
        b->pushdown() ;
        b->l = merge(a,b->l) ;
        b->maintain() ;
        return b ;
    }
}
void Delete(node *t) {
    if(!t) return ;
    Delete(t->l) ;
    Delete(t->r) ;
    delete t ;
}
void INSERT(node* &root) {
    int p , k , v ;
    node *t=0 , *L , *R ;
    scanf("%d%d",&p,&k) ;
    for(int i=0 ; i<k ; i++) {
        scanf("%d",&v) ;
        t = merge(t,new node(v)) ;
    }
    splite(root, L, R, p) ;
    root = merge(L, merge(t, R)) ;
}
void DELETE(node* &root) {
    int p , k ;
    scanf("%d%d",&p,&k) ;
    node *L , *M , *R ;
    splite(root, L, R, p-1) ;
    splite(R, M, R, k) ;
    Delete(M) ;
    root = merge(L, R) ;
}
void MAKE_SAME(node* &root) {
    int p , k , l ;
    scanf("%d%d%d",&p,&k,&l) ;
    node *L , *M , *R ;
    splite(root, L, R, p-1) ;
    splite(R, M, R, k) ;
    M->makesame(l) ;
    root = merge(L, merge(M, R)) ;
}
void REVERSE(node* &root) {
    int p , k ;
    scanf("%d%d",&p,&k) ;
    node *L , *M , *R ;
    splite(root, L, R, p-1) ;
    splite(R, M, R, k) ;
    M->flag = !M->flag ;
    root = merge(L, merge(M, R)) ;
}

```



```

}
int GET_SUM(node* &root) {
    int p, k, v;
    scanf("%d%d", &p, &k);
    node *L, *M, *R;
    splite(root, L, R, p-1);
    splite(R, M, R, k);
    v = M->Sum();
    root = merge(L, merge(M, R));
    return v;
}
int MAX_SUM(node* &root) {
    return root->Maxsum();
}
int main () {
    int n, m;
    srand(860514);
    while(scanf("%d%d", &n, &m)==2) {
        node *root=0;
        for(int i=0, v; i<n; i++) {
            scanf("%d", &v);
            root = merge(root, new node(v));
        }
        while(m--) {
            char s[10];
            scanf("%s", s);
            if(strcmp(s, "INSERT")==0) {
                INSERT(root);
                continue;
            }
            if(strcmp(s, "DELETE")==0) {
                DELETE(root);
                continue;
            }
            if(strcmp(s, "MAKE-SAME")==0) {
                MAKE_SAME(root);
                continue;
            }
            if(strcmp(s, "REVERSE")==0) {
                REVERSE(root);
                continue;
            }
            if(strcmp(s, "GET-SUM")==0) {
                printf("%d\n", GET_SUM(root));
                continue;
            }
            if(strcmp(s, "MAX-SUM")==0) {
                printf("%d\n", MAX_SUM(root));
                continue;
            }
        }
    }
}

```

4.4 Tree Heavy Light Decomposition

```

#include <iostream>
#include <cstdio>
#include <vector>
#include <string.h>
#define N 50000
using namespace std;

vector<int> G[N];
int ind[N], f[N], d[N], head[N], son[N], pn, n;

int dfs(int u, int parent) {
    int i, v, x, w = 0, sum = 1;
    f[u] = parent;
    son[u] = -1;
    for(i = 0; i < G[u].size(); i++) {
        v = G[u][i];
        if(v != parent) {
            d[v] = d[u] + 1;
            x = dfs(v, u);
            sum += x;
            if(x > w) {
                son[u] = v;
                w = x;
            }
        }
    }
    return sum;
}

void chain(int u) {
    int i, v;
    ind[u] = pn++;
    if(f[u] == -1 || son[f[u]] != u)
        head[u] = u;
}

```

```

else
    head[u] = head[f[u]];
if(son[u] != -1)
    chain(son[u]);
for(i=0; i<G[u].size(); i++) {
    v = G[u][i];
    if(v != f[u] && v != son[u])
        chain(v);
}

void modify(int u, int v, ...) {
    while(head[u] != head[v]) {
        if(d[head[u]] > d[head[v]]) {
            //do something in rang(ind[head[u]], ind[u]);
            u = f[head[u]];
        } else {
            //do something in rang(ind[head[v]], ind[v]);
            v = f[head[v]];
        }
    }
    if(d[u] > d[v])
        //do something in rang(ind[head[u]], ind[u]);
    else
        //do something in rang(ind[head[v]], ind[v]);
}

int getLca(int u, int v) {
    while(head[u] != head[v])
        if(d[head[u]] > d[head[v]])
            u = f[head[u]];
        else
            v = f[head[v]];
    if(d[u] > d[v])
        return v;
    else
        return u;
}

void build() {
    pn = 1;
    d[0] = 0; //set depth of root to 0
    dfs(0, -1);
    chain(0); //relabel node
}

```

5 Graph

5.1 Biconnected Components

```

#include <iostream>
#include <cstdio>
#include <cstring>
#include <stack>
#include <vector>
using namespace std;
//Biconnected Components
struct BCC {
    static const int maxNum = 1010;
    vector<int> G[maxNum], bccGroup[maxNum];
    int Node;
    int bcc_cnt;
    int timeStamp;
    int low[maxNum];
    int visit[maxNum];
    int bcc[maxNum];
    bool is_ap[maxNum];
    stack< pair<int, int> > S;
    BCC(int Node) {
        for(int i = 0; i < maxNum; i++) {
            G[i].clear();
            bccGroup[i].clear();
            low[i] = visit[i] = bcc[i] = -1;
            is_ap[i] = false;
        }
        this->Node = Node;
        bcc_cnt = 0;
    }
    void DFS(int u, int parent) {
        int children = 0;
        low[u] = visit[u] = timeStamp++;
        for(int i = 0; i < G[u].size(); i++) {
            int v = G[u][i];
            if(visit[v] == -1) {
                S.push(make_pair(u, v));
                children++;
                DFS(v, u);
                low[u] = min(low[u], low[v]);
                if(low[v] >= visit[u]) {
                    is_ap[u] = true;
                    pair<int, int> e;
                    do {

```



```

    e = S.top();
    if(bcc[e.first] != bcc_cnt) {
        bccGroup[bcc_cnt].push_back(e.first);
        bcc[e.first] = bcc_cnt;
    }
    if(bcc[e.second] != bcc_cnt) {
        bccGroup[bcc_cnt].push_back(e.second);
        bcc[e.second] = bcc_cnt;
    }
    S.pop();
    while(e.first!=u || e.second!=v);
    bcc_cnt++;
}
} else if(v != parent) {
    S.push(make_pair(u, v));
    low[u] = min(low[u], visit[v]);
}
}
if(u == parent) // u is root
    is_ap[u] = (children >= 2);
}
void articulation_vertex() {
    timeStamp = 0;
    for(int i = 0; i < Node; i++)
        if(low[i] == -1)
            DFS(i, i);
}
};

```

5.2 Strong Connected Components

```

class SCC {
private:
    static const int maxN = 10004;
    vector<int> G[maxN];
    vector<int> invG[maxN];
    vector<int> stk;
    bool visited[maxN];
    void dfs_1(int nowAt) {
        visited[nowAt] = true;
        for(auto v : G[nowAt])
            if(!visited[v])
                dfs_1(v);
        stk.emplace_back(nowAt);
    }
    void dfs_2(int nowAt, const int id) {
        sccID[nowAt] = id;
        for(auto v : invG[nowAt])
            if(sccID[v]==-1)
                dfs_2(v, id);
    }
public:
    int sccNum;
    int sccID[maxN];

    void init() {
        for(int i=0; i<maxN; ++i) {
            G[i].clear();
            invG[i].clear();
        }
    }
    void addEdge(int u, int v) {
        G[u].emplace_back(v);
        invG[v].emplace_back(u);
    }
    vector<vector<int>> findAllSCC(int base, int n) {
        memset(visited, 0, sizeof(visited));
        stk.clear();
        for(int i=base; i<=n; ++i)
            if(!visited[i])
                dfs_1(i);

        sccNum = 0;
        memset(sccID, -1, sizeof(sccID));
        for(int i=stk.size()-1; i>=0; --i)
            if(sccID[stk[i]]==-1) {
                dfs_2(stk[i], sccNum);
                ++sccNum;
            }

        // returned zero base scc dag
        vector<vector<int>> sccDAG(sccNum);
        vector<unordered_set<int>> have(sccNum);
        for(int u=base; u<=n; ++u) {
            int sccU = sccID[u];
            for(auto v : G[u]) {
                int sccV = sccID[v];
                if(sccU==sccV) continue;
                if(have[sccU].find(sccV) == have[sccU].end()) {

```

```

                    have[sccU].insert(sccV);
                    sccDAG[sccU].emplace_back(sccV);
                }
            }
        }
        return sccDAG;
    }
};

```

5.3 2 SAT

```

class TwoSAT {
private:
    int V;
    vector<bool> pick;
    vector<vector<int>> G;

    int id(int i, int T) { return (i<<1) + T; }
    int alter(int i) { return i^1; }
    bool dfsTry(int nowAt, vector<int> &stk) {
        if(pick[alter(nowAt)])
            return false;
        stk.emplace_back(nowAt);
        pick[nowAt] = true;
        for(auto v : G[nowAt]) {
            if(!pick[v] && !dfsTry(v, stk))
                return false;
        }
        return true;
    }
public:
    void init(int varNum) {
        V = varNum;
        pick = vector<bool>(V*2 + 4, false);
        G = vector<vector<int>>(V*2 + 4);
    }
    void addClause(bool TA, int A, bool TB, int B) {
        // Add clause (TA + TB)
        G[id(A, !TA)].emplace_back(id(B, TB));
        G[id(B, !TB)].emplace_back(id(A, TA));
    }
    void imply(bool TA, int A, bool TB, int B) {
        // TA -> TB
        addClause(!TA, A, TB, B);
    }
    void preset(bool TA, int A) {
        pick[id(A, TA)] = true;
    }
    bool solve() {
        vector<int> stk;
        for(int i=0; i<V; ++i) {
            if(pick[id(i, true)] && !dfsTry(id(i, true), stk))
                return false;
            if(pick[id(i, false)] && !dfsTry(id(i, false), stk))
                return false;
            stk.clear();
        }
        for(int i=0; i<V; ++i) {
            if(pick[id(i, 0)] || pick[id(i, 1)])
                continue;
            stk.clear();
            if(dfsTry(id(i, 0), stk))
                continue;
            for(auto v : stk)
                pick[v] = false;
            if(!dfsTry(id(i, 1), stk))
                return false;
        }
        return true;
    }
    bool T(int i) {
        // should solved first
        return pick[id(i, 1)];
    }
};

```

5.4 System of Difference Constraints

```

class System_of_DifConstrain {
private:
    static const int maxN = 504;
    static const int maxM = 3004;
    struct Edge {
        int s, t;
        long long cost;
    };
    Edge es[maxM];

```

```

int eSize;
public:
bool solvable;
long long x[maxN]; // one solution
void init() {
    eSize = -1;
}
void addConstrain(int xI, int xJ, long long c) {
    // add xi - xj <= c
    es[++eSize] = {xJ, xI, c};
}
bool solve(int n=maxN) {
    // n is max # of node of CC
    memset(x, 0, sizeof(x));
    for(int i=0; i<n; ++i)
        for(int j=0; j<=eSize; ++j)
            if( x[es[j].s] + es[j].cost < x[es[j].t] )
                x[es[j].t] = x[es[j].s] + es[j].cost;
    for(int j=0; j<=eSize; ++j)
        if( x[es[j].s] + es[j].cost < x[es[j].t] )
            return solvable = false;
    return solvable = true;
}
};

```

5.5 Bipartite: MaxMatch, MinVer-Cover, MaxIndSet

```

class Bipartite {
private:
    static const int MaxNum = 1004;
    vector<int> g[MaxNum];
    bool visited [MaxNum];

    bool bipart(int nowAt, int nowSide) {
        visited[nowAt] = true;
        side[nowAt] = nowSide;
        for(auto &id : g[nowAt])
            if( !visited[id] )
                bipart(id, !nowSide);
            else if( side[id]==nowSide )
                return false;
        return true;
    }
    bool maxMatch(int nowAt) {
        visited[nowAt] = true;
        for(auto &id : g[nowAt])
            if( cp[id]==-1
                || (!visited[cp[id]] && maxMatch(cp[id])) ) {
                cp[id] = nowAt;
                cp[nowAt] = id;
                return true;
            }
        return false;
    }
    void minVertexCover(int nowAt) {
        MVC[nowAt] = 1;
        for(auto &id : g[nowAt])
            if( !MVC[id] ) {
                MVC[id] = 1;
                minVertexCover(cp[id]);
            }
    }
    void maxIndependentSet(int nowAt) {
        MIS[nowAt] = 1;
        for(auto &id : g[nowAt])
            if( !MIS[cp[id]] )
                maxIndependentSet(cp[id]);
    }
public:
    int matchNum; // max match num
    int cp [MaxNum]; // id and cp[id] is couple
    bool side[MaxNum]; // left/right side
    bool MVC [MaxNum]; // min vertex cover
    bool MIS [MaxNum]; // max independent set
    void addEdge(int u, int v) {
        g[u].emplace_back(v);
        g[v].emplace_back(u);
    }
    void init() {
        for(int i=0; i<MaxNum; ++i)
            g[i].clear();
    }
    bool countAll() {
        // if graph is not bipartite return false

        // bipartite
        memset(side, 0, sizeof(side));

```

```

memset(visited, 0, sizeof(visited));
for(int i=0 ; i<MaxNum ; ++i)
    if( !visited[i] && !bipart(i, 0) )
        return false;

// maximum match
// O(VE), this code can be more optimized
// alternative: dinic O(V^0.5*E)
matchNum = 0;
memset(cp, -1, sizeof(cp));
for(int i=0 ; i<MaxNum ; ++i){
    if( cp[i]!=-1 ) continue;
    memset(visited, 0, sizeof(visited));
    if( maxMatch(i) )
        ++matchNum;
}

// min vertex cover
memset(MVC, 0, sizeof(MVC));
for(int i=0; i<MaxNum; ++i)
    if( side[i]==1 && cp[i]==-1 )
        minVertexCover(i);
for(int i=0; i<MaxNum; ++i)
    if( side[i]==1 )
        MVC[i] = !MVC[i];

// max independent set
memset(MIS, 0, sizeof(MIS));
for(int i=0; i<MaxNum; ++i)
    if( cp[i]==-1 )
        maxIndependentSet(i);
for(int i=0; i<MaxNum; ++i)
    if( side[i]==1 && cp[i]!=-1
        && !MIS[i] && !MIS[cp[i]] )
        MIS[i] = 1;

return true;
};

```

5.6 Bipartite: KM

```

#include <cstring>
#include <iostream>
using namespace std;
struct KM {
    static const int N = 105;
    int visx[N], visy[N];
    int nx, ny, matchx[N], matchy[N];
    int labelx[N], labely[N], G[N][N];
    bool labeled[N];

    int max_match_value;
    bool isPerfect;

    KM(int nx_, int ny_):nx(nx_),ny(ny_) {} ;

    bool DFS(int x) {
        visx[x] = true ;
        int y ;
        for(int y = 0; y < ny ; y++)
            if(hasEdge(x, y) && !visy[y] && labelx[x] + labely[y] == G[x][y] ) {
                visy[y] = true ;
                if(matchy[y]==-1 || DFS(matchy[y])) {
                    matchx[x] = y ;
                    matchy[y] = x ;
                    return true ;
                }
            }
        return false ;
    }

    bool max_match() { //Maximum Weight Perfect Bipartite Matching
        isPerfect = true;
        for(int y = 0; y < ny; y++)
            labely[y] = 0;
        memset(labeled, 0, sizeof(labeled));
        memset(matchx, -1, sizeof(matchx));
        memset(matchy, -1, sizeof(matchy));
        for(int x = 0; x < nx ; x++)
            for(int y = 0; y < ny ; y++)
                if(hasEdge(x, y))
                    if(!labeled[x]) {
                        labelx[x] = G[x][y];
                        labeled[x] = true;
                    } else

```

```

        labelx[x] = max(labelx[x], G[x][y]) ;
for(int i = 0; i < nx ; i++)
    while(true) {
        memset(visx, 0, sizeof(visx)) ;
        memset(visy, 0, sizeof(visy)) ;
        if(DFS(i)) break ;
        int d;
        bool flag = false;
        for(int x = 0; x < nx; x++) if(visx[x])
            for(int y = 0; y < ny; y++) if(!visy[y])
                if(hasEdge(x, y))
                    if(!flag) {
                        d = labelx[x] + labely[y] - G[x][y];
                        flag = true;
                    } else
                        d = min(d, labelx[x] + labely[y] - G[x][y]) ;
        if(!flag) {
            isPerfect = false;
            break;
        }
        for(int j = 0; j < nx; j++)
            if(visx[j]) labelx[j] -= d;
        for(int j = 0; j < ny; j++)
            if(visy[j]) labely[j] += d;
    }
int total = 0;
for(int x = 0; x < nx; x++) //must be perfect!!!
    if(matchx[x] != -1)
        total += G[x][matchx[x]];
max_match_value = total;
return isPerfect;
}
bool hasEdge(int u,int v) {
    //TODO
}
};
/*****
* change edge data type
* negative edge for min_match
* initialize G[][]
* call km.max_match()
* check km.isPerfect
* check km.max_match_value
*/

```

5.7 Min Vertex Cover

```

struct MinVertexCover {
private:
    static const int MaxNum = 54;
    vector<int> G[MaxNum];
    int in[MaxNum];

    int undo(vector<int> &record) {
        for(int i=0; i<record.size(); ++i)
            ++in[record[i]];
        record.clear();
    }

    int delNode(int u, vector<int> &record) {
        for(int i=0; i<G[u].size(); ++i)
            if( in[G[u][i]] ) {
                --in[G[u][i]];
                --in[u];
                record.push_back(G[u][i]);
                record.push_back(u);
            }
    }

    int cnt(int from, int *visited, bool type) {
        if( visited[from] ) return 0;
        if( type==1 ) visited[from] = 1;
        for(int i=0; i<G[from].size(); ++i)
            if( in[G[from][i]] && !visited[G[from][i]] )
                return type+cnt(G[from][i], visited, !type);
        return type;
    }

    int cnt(int *visited) {
        int ret = 0;
        for(int i=0; i<MaxNum; ++i)
            if( in[i]==1 && !visited[i] )
                ret += cnt(i, visited, 0);
        for(int i=0; i<MaxNum; ++i)
            if( in[i]==2 && !visited[i] )
                ret += cnt(i, visited, 0);
        return ret;
    }

public:
    int MVCPick[MaxNum];

```

```

int MVC; // min vertex cover
void init() {
    for(int i=0; i<MaxNum; ++i)
        G[i].clear();
    memset(in, 0, sizeof(in));
}
void addEdge(int u, int v) {
    G[u].push_back(v);
    G[v].push_back(u);
    ++in[u];
    ++in[v];
}
void minVertexCover(int nowMVC=0, const int *lastPick=NULL) {
    // 0(n^2 * 1.38^n)
    int nowPick[MaxNum] = {};
    if( nowMVC==0 ) {
        MVC = MaxNum;
        memset(MVCPick, 0, sizeof(MVCPick));
    }
    else memcpy(nowPick, lastPick, sizeof(nowPick));

    int maxid = 0;
    for(int i=0; i<MaxNum; ++i)
        if( in[i]>in[maxid] )
            maxid = i;
    if( in[maxid]<=2 ) {
        nowMVC += cnt(nowPick);
        if( nowMVC<MVC ) {
            MVC = nowMVC;
            memcpy(MVCPick, nowPick, sizeof(nowPick));
        }
        return;
    }

    vector<int> record;

    delNode(maxid, record);
    nowPick[maxid] = 1;
    minVertexCover(nowMVC+1, nowPick);
    nowPick[maxid] = 0;
    undo(record);

    int cnt = 0;
    for(int i=0; i<G[maxid].size(); ++i)
        if( in[G[maxid][i]] ) {
            ++cnt;
            delNode(G[maxid][i], record);
            nowPick[G[maxid][i]] = 1;
        }
    minVertexCover(nowMVC+cnt, nowPick);
    undo(record);
}
};

```

5.8 Max Independent Set

Pick those vertice **which** are not picked by **min** vertex cover.

5.9 Max Clique

Create a new graph $G'(V, E')$ in **which**
 $E' = \{(u, v) \mid \text{if } (u, v) \text{ not in } E\}$
 Max clique in $G \equiv \equiv$ Max independent **set** in G'

5.10 Kirchhoff's theorem

D = Degree matrix (diagonal entries are node's degree)
 A = Adjacency matrix (01 matrix)
 Q = Laplacian matrix = $D - A$
 Q^* = deleting **any** row and **any** column from Q
 Number of spanning tree = **det**(Q^*)

5.11 Popular matching

N applicant M position
 Each applicant have his own preference list (not required containing **all** position)
 Target: **find** a matching s.t. not existing another matching have **more** applicant prefer. This matching called popular matching
 A applicant u prefer another matching **which** imply that
 "This matching doesn't have u but the other does"
 or
 "The other matching give u the position that he more prefer"

Define **hot** position: the position is **any** applicant's first choice

Solve: Whether there is a popular matching

- (1) Each applicant build and edge with "his first choice" and "his first choice except all hot position" (if not exist, build a dummy position)
- (3) Return whether all applicant can have a position

Solve: Maximum popular matching

- (1) If popular matching not exist return No
- (2) Use the result above
- (3) Remove all dummy position
- (4) Find augmenting path for all unmatched applicant
- (5) For all unmatched hot position, rob an applicant (all it's applicant choice non hot position)
- (6) return the matching

5.12 Blossom Algorithm

```
#include <cstdio>
#include <queue>
#include <string>
using namespace std;

struct BlossomAlgorithm {
private:
    // number of vertex, zero base
    static const int N = 205;
    int match[N];
    int d[N];
    queue<int> q;
    deque<int> path[N];

    void label_one_side(int x, int y, int bi) {
        for (int i = bi+1; i < path[x].size(); i++) {
            int z = path[x][i];
            if (d[z] == 1) {
                path[z] = path[y];
                path[z].insert(path[z].end(), path[x].rbegin(), path[x].rend()-i);
                d[z] = 0;
                q.push(z);
            }
        }
    }

    bool BFS(int r) {
        for (int i = 0; i < n; ++i)
            path[i].clear();
        path[r].push_back(r);

        memset(d, -1, sizeof(d));
        d[r] = 0;

        while (!q.empty())
            q.pop();
        q.push(r);

        while (!q.empty()) {
            int x = q.front();
            q.pop();
            for (int y = 0; y < n; y++)
                if (map[x][y] && match[y] != y)
                    if (d[y] == -1)
                        if (match[y] == -1) {
                            for (int i = 0; i + 1 < path[x].size(); i += 2) {
                                match[path[x][i]] = path[x][i+1];
                                match[path[x][i+1]] = path[x][i];
                            }
                            match[x] = y; match[y] = x;
                            return true;
                        }
                    else {
                        int z = match[y];
                        path[z] = path[x];
                        path[z].push_back(y);
                        path[z].push_back(z);

                        d[y] = 1; d[z] = 0;
                        q.push(z);
                    }
                else if (d[y] == 0) {
                    int bi = 0;
                    while (bi < path[x].size() && bi < path[y].size() && path[x][bi] == path[y][bi])
                        bi++;
                    bi--;
                }
            }
        }
    }
};
```

```
label_one_side(x, y, bi);
label_one_side(y, x, bi);
}
return false;
}

public:
    int max_match;
    int n;
    bool map[N][N];

    void matchAll() {
        memset(match, -1, sizeof(match));

        max_match = 0;
        for (int i = 0; i < n; i++)
            if (match[i] == -1)
                if (BFS(i))
                    max_match++;
            else
                match[i] = i;
    }

    void init() {
        memset(map, false, sizeof(map));
    }

    void addedge(int u, int v) {
        map[u][v] = map[v][u] = true;
    }
} solver;

/**
 * Step:
 * solver.init();
 * solver.n = n;
 * loop:
 * solver.addedge(i, j);
 */
```

6 Flow

6.1 Dinic Maxflow Mincut

```
class Dinic {
private:
    typedef pair<int, int> pii;
    static const int maxN = 504;
    static const int inf = 1023456789;
    int cap[maxN][maxN];
    int pipe[maxN][maxN];
    vector<int> g[maxN];
    int level[maxN], visited[maxN];

    bool bfsLabeling(int s, int t) {
        memset(level, 0, sizeof(level));
        queue<int> myQ;
        myQ.push(s);
        level[s] = 1;
        while (!myQ.empty()) {
            int nowAt = myQ.front();
            myQ.pop();
            for (int i = 0; i < g[nowAt].size(); ++i)
                if (!level[g[nowAt][i]] && pipe[nowAt][g[nowAt][i]])
                    level[g[nowAt][i]] = level[nowAt] + 1;
            myQ.push(g[nowAt][i]);
        }
        return level[t];
    }

    int dfsFindRoute(int nowAt, int t, int maxC) {
        visited[nowAt] = true;
        if (nowAt == t)
            maxFlow += maxC;
            return maxC;
        for (int i = 0; i < g[nowAt].size(); ++i) {
            int next = g[nowAt][i];
            if (visited[next]) continue;
            if (level[next] != level[nowAt] + 1) continue;
            if (!pipe[nowAt][next]) continue;
            int nowOut = dfsFindRoute(next, t, min(maxC, pipe[nowAt][next]));
            if (nowOut == 0)
                continue;
            pipe[nowAt][next] -= nowOut;
            pipe[next][nowAt] += nowOut;
        }
    }
};
```

```

        return nowOut;
    }
    return 0;
}
void dfsFindMinCut(int nowAt) {
    ssize[nowAt] = 1;
    for(auto v : g[nowAt])
        if( !ssize[v] && pipe[nowAt][v] )
            dfsFindMinCut(v);
}

public:
    int maxFlow;
    bool ssize[maxN];
    vector<pii> minCut;

    void init() {
        memset(cap, 0, sizeof(cap));
        memset(pipe, 0, sizeof(pipe));
        memset(ssize, 0, sizeof(ssize));
        for(int i=0; i<maxN; ++i)
            g[i].clear();
        maxFlow = 0;
        minCut.clear();
    }

    void addEdge(int u, int v, int c) {
        if( u==v ) return;
        if( !cap[u][v] && !cap[v][u] ) {
            g[u].emplace_back(v);
            g[v].emplace_back(u);
        }
        cap[u][v] += c;
    }

    void coculAll(int s, int t) {
        memcpy(pipe, cap, sizeof(pipe));

        // max flow
        while( bfsLabeling(s,t) ) {
            memset(visited, 0, sizeof(visited));
            while( dfsFindRoute(s,t,infF) )
                ;
        }

        // min cut
        dfsFindMinCut(s);
        for(int u=0; u<maxN; ++u)
            if( ssize[u] )
                for(auto v : g[u])
                    if( !ssize[v] )
                        minCut.push_back({u, v});
    }
};

```

7 Geometry

7.1 Geometry basic

```

struct Point{
    double x,y;
    Point(double xi=0.0,double yi=0.0){
        x = xi , y = yi;
    }
    Point operator - (const Point &r)const{
        return Point(x-r.x , y-r.y);
    }
};

typedef Point Vector;

double angle(const Vector &v,const Vector &u){
    // return rad [0, pi] of two vector
    return acos( dot(v,u)/len(v)/len(u) );
}

Vector rotate(const Vector &v,double rad){
    return Vector(
        v.x*cos(rad) - v.y*sin(rad),
        v.x*sin(rad) + v.y*cos(rad)
    );
}

double pointSegLen(const Point &A,const Point &B,const Point &Q){
    if(A==B) return len(Q-A);
    if( dot(B-A , Q-A)<0 ) return len(Q-A);
    if( dot(B-A , Q-B)>0 ) return len(Q-B);
    return fabs( cross(B-A , Q-A) ) / len(B-A);
}

bool pointOnSeg(const Point &A,const Point &B,const Point &Q){
    return fabs( len(Q-B)+len(Q-A)-len(A-B) ) < 1e-9;
}

bool pointInPoly(const vector<Point> &poly, const Point &p) {

```

```

// Poly should be counter clockwise. 0(logN)
int r_bound = poly.size();
while( poly[0] == poly[r_bound-1] )
    r_bound--;
int l = 1, r = r_bound;
const Point target = p - poly[0];
while( l < r ) {
    int c = (l + r) / 2;
    if( fdif(cross(poly[c] - poly[0], target)) <= 0 )
        r = c;
    else
        l = c + 1;
}
if( l == 1 || l == r_bound ) return false;
double c1 = cross(poly[l] - poly[0], p - poly[0]);
double c2 = cross(poly[l-1] - poly[l], p - poly[l]);
double c3 = cross(poly[0] - poly[l-1], p - poly[l-1]);
return fdif(c1) == 0 || fdif(c2) == 0 || fdif(c3) == 0
    || (fdif(c1)>0 == fdif(c2)>0 && fdif(c2)>0 == fdif(c3)
        >0);
}

struct Line{
    Point P0;
    Vector v;
    Line(const Point &pi,const Vector &vi):p0(pi) , v(vi) {}
};

double pointLineLen(const Line &L,const Point &Q){
    return fabs( cross(L.v , Q-L.P0) ) / len(L.v);
}

Point projectToLine(const Line &L,const Point &Q){
    double t = dot(Q-L.P0 , L.v) / dot(L.v , L.v);
    return L.P0 + L.v * t;
}

Point innerCircle(point &p1, point &p2, point &p3){
    // p1,p2,p3 should not on same line
    double a1 = (-2*p1.x + 2*p2.x);
    double b1 = (-2*p1.y + 2*p2.y);
    double c1 = (p2.x*p2.x + p2.y*p2.y - p1.x*p1.x - p1.y*p1.y);
    double a2 = (-2*p1.x + 2*p3.x);
    double b2 = (-2*p1.y + 2*p3.y);
    double c2 = (p3.x*p3.x + p3.y*p3.y - p1.x*p1.x - p1.y*p1.y);
    double cx = (c1*b2-c2*b1) / (a1*b2-a2*b1);
    double cy = (a1*c2-a2*c1) / (a1*b2-a2*b1);
    return Point(cx, cy);
}

Point outerCircle(point &p1, point &p2, point &p3) {
    // p1,p2,p3 should not on same line
    double x1 = (p1.x+p2.x)/2.0;
    double y1 = (p1.y+p2.y)/2.0;
    double x2 = (p2.x+p3.x)/2.0;
    double y2 = (p2.y+p3.y)/2.0;
    double vx = p2.x-p1.x;
    double vy = p2.y-p1.y;
    double ux = p3.x-p2.x;
    double uy = p3.y-p2.y;
    double A = vx*x1 + vy*y1;
    double B = ux*x2 + uy*y2;
    double cx = (uy*A - vy*B) / (uy*vx - ux*vy);
    double cy = (ux*A - vx*B) / (ux*vy - uy*vx);
    return Point(cx, cy);
}

```

7.2 Minimal Enclose Disk

```

struct Circle {
    Point c;
    double R2; // square of radius
    Circle() {}
    Circle(const Point &p1, const Point &p2) {
        c.x = (p1.x+p2.x)/2.0;
        c.y = (p1.y+p2.y)/2.0;
        R2 = dot(p1-p2, p1-p2)/4.0;
    }
    Circle(const Point &p1, const Point &p2, const Point &p3) {
        // p1, p2, p3 should not on same line
        c = outerCircle(p1, p2, p3);
        double dx = p1.x - c.x;
        double dy = p1.y - c.y;
        R2 = dx*dx + dy*dy;
    }
    bool contain(const Point &p) const {
        double dx = c.x - p.x;
        double dy = c.y - p.y;
        return fdif(dx*dx + dy*dy - R2)<=0;
    }
};

```

```

Circle minEncloseDisk(vector<Point> &ps) {
    // Find minimal circular enclose all point
    // worst case  $O(n^3)$ , expected  $O(n)$ 
    Circle D;
    if( ps.size()==0 ) return D;
    if( ps.size()==1 ) {
        D.c = ps[0];
        D.R2 = 0.0;
        return D;
    }

    random_shuffle(ps.begin(), ps.end());
    D = Circle(ps[0], ps[1]);
    for(int i=2; i<ps.size(); ++i) {
        if( D.contain(ps[i]) )
            continue;
        D = Circle(ps[i], ps[0]);
        for(int j=1; j<i; ++j) {
            if( D.contain(ps[j]) )
                continue;
            D = Circle(ps[i], ps[j]);
            for(int k=0; k<j; ++k) {
                if( D.contain(ps[k]) )
                    continue;
                D = Circle(ps[i], ps[j], ps[k]);
            }
        }
    }
}

```

7.3 2D Convex Hull

```

bool turnLeft(const Vector &v1, const Vector &v2) {
    return fdif(cross(v1, v2)) > 0LL;
}

vector<Point> convexHull(vector<Point> &ps) {
    // return convex hull without redundant point
    sort(ps.begin(), ps.end());

    vector<Point> up;
    for(int i=0; i<ps.size(); ++i) {
        while( up.size()>1
            && !turnLeft(up.back()-up[up.size()-2],
                ps[i]-up.back()) )
            up.pop_back();
        up.emplace_back(ps[i]);
    }

    vector<Point> btn;
    for(int i=ps.size()-1; i>=0; --i) {
        while( btn.size()>1
            && !turnLeft(btn.back()-btn[btn.size()-2],
                ps[i]-btn.back()) )
            btn.pop_back();
        btn.emplace_back(ps[i]);
    }

    vector<Point> res(up);
    res.insert(res.end(), btn.begin()+1, btn.end());
    res.pop_back();
    return res;
}

```

7.4 Closest Point

```

#include <iostream>
#include <cstdio>
#include <algorithm>
#include <cstring>
#include <cmath>
#define N 10010
#define INFINITY__ 1e10
using namespace std;
int diff(double f) {
    if(fabs(f) < 1e-9)
        return 0;
    return f < 0 ? -1 : 1;
}

struct point {
    double x, y;
    bool operator < (const point &p) const {
        return diff(x - p.x) < 0;
    }
};

point pt[N], tmp[N];
double dis(point a, point b) {
    return sqrt((a.x - b.x)*(a.x - b.x) + (a.y - b.y)*(a.y - b.y));
}

```

```

}

double closestpoint(int L, int R) {
    if(R - L + 1 == 1) return INFINITY__;
    int M = (L+R)/2;
    double middle = pt[M].x;
    double Ldis = closestpoint(L, M);
    double Rdis = closestpoint(M+1, R);
    double radi = min(Ldis, Rdis);
    int cntpt = 0;
    merge(pt+L, pt+M+1, pt+M+1, pt+R+1, tmp+L, [](point a, point b)
        {
            return diff(a.y - b.y) < 0;
        });
    copy(tmp + L, tmp + R + 1, pt + L);
    for(int i = L; i <= R; i++)
        if(diff(fabs(pt[i].x - middle) - radi) < 0)
            tmp[cntpt++] = pt[i];
    for(int i = 0; i < cntpt; i++)
        for(int j = 1; i + j < cntpt && j < 8; j++)
            radi = min(radi, dis(tmp[i], tmp[i+j]));
    return radi;
}

```

7.5 Min Max Triangle

```

pair<double, double> findMinMaxTri(vector<Point> &ps) {
    static const double PI = acos(-1.0);
    struct Seg {
        double rad; // [0.5pi, 1.5pi]
        int s1, s2;
    };

    const int n = ps.size();
    sort(ps.begin(), ps.end(), [](const Point &l, const Point &r) {
        if( fdif(l.x - r.x) == 0 )
            return l.y > r.y;
        return l.x < r.x;
    });
    vector<int> id(n+4);
    for(int i=0; i<n; ++i)
        id[i] = i;

    // sort all pair of point
    vector<Seg> segs;
    for(int i=0; i<n; ++i)
        for(int j=i+1; j<n; ++j) {
            double m = atan2(ps[j].y-ps[i].y, ps[j].x-ps[i].x) + PI;
            segs.push_back({m, i, j});
        }
    sort(segs.begin(), segs.end(), [](const Seg &l, const Seg &r) {
        return fdif(l.rad - r.rad) < 0;
    });

    // find min max triangle
    pair<double, double> ret;
    ret.first = ret.second = fabs(cross(ps[0], ps[1], ps[2]));
    for(auto seg : segs) {
        swap(ps[id[seg.s1]], ps[id[seg.s2]]);
        swap(id[seg.s1], id[seg.s2]);

        const Point &p1 = ps[id[seg.s1]];
        const Point &p2 = ps[id[seg.s2]];
        int id1 = min(id[seg.s1], id[seg.s2]);
        int id2 = max(id[seg.s1], id[seg.s2]);

        // find min triangle
        if( id1-1 >= 0 ) {
            double a = fabs(cross(p1, p2, ps[id1-1]));
            if( a < ret.first )
                ret.first = a;
        }
        if( id2+1 < n ) {
            double a = fabs(cross(p1, p2, ps[id2+1]));
            if( a < ret.first )
                ret.first = a;
        }

        // find max triangle
        if( id1 != 0 ) {
            double a = fabs(cross(p1, p2, ps[0]));
            if( a > ret.second )
                ret.second = a;
        }
        if( id2 != n-1 ) {
            double a = fabs(cross(p1, p2, ps[n-1]));
            if( a > ret.second )
                ret.second = a;
        }
    }
}

```

```
ret.first /= 2.0;  
ret.second /= 2.0;  
return ret;  
}
```