Contents

1 Basic

1.1 Tips

1. 可能要先枚舉某些部分，在套演算法
2. Size很小，考慮狀態壓縮dp

1.2 vimrc

```
set nocompatible
set t_Co=256
set nu
set ai
set tabstop=4
set shiftwidth=4
set softtabstop=4
colorscheme torte
syntax on
filetype plugin indent on
```

1.3 default

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

using namespace std;

#define FI freopen("in.txt", "r", stdin)
#define FO freopen("out.txt", "w", stdout)
#define IOS ios_base::sync_with_stdio(0);cin.tie(0)
#define pb push_back
#define mp make_pair
#define ff first
#define ss second

typedef long long LL;

const int MOD = 1000000007;
const double PI = acos(-1.0);

int dx[] = {-1,0,1,0};
int dy[] = {0,1,0,-1};

int main(){
    IOS;
    return 0;
}
```

1.4 FastInput

```cpp
int readInt () {
  bool minus = false;
  int result = 0;
  char ch;
  ch = getchar();
  while (true) {
    if (ch == '-') break;
    if (ch >= '0' && ch <= '9') break;
    ch = getchar();
  }
  if (ch == '-') minus = true; else result = ch-'0';
  while (true) {
    ch = getchar();
    if (ch < '0' || ch > '9') break;
    result = result*10 + (ch - '0');
  }
  if (minus)
    return -result;
  else
    return result;
}
```

1.5 Int128

```cpp
#include <bits/stdc++.h>
using namespace std;

std::ostream& operator<<(std::ostream& dest, __int128_t
     value)
{
    std::ostream::sentry s(dest);
    if (s) {
        __uint128_t tmp = value < 0 ? -value : value;
        char buffer[128];
        char* d = std::end(buffer);
        do {
            --d;
            *d = "0123456789"[tmp % 10];
            tmp /= 10;
        } while (tmp != 0);
        if (value < 0) {
            --d;
            *d = '-';
        }
        int len = std::end(buffer) - d;
        if (dest.rdbuf()->sputn(d, len) != len) {
            dest.setstate(std::ios_base::badbit);
        }
    }
    return dest;
}

__int128 parse(string& s)
{
    __int128 ret = 0;
    for (int i = 0; i < s.length(); i++)
        if ('0' <= s[i] && s[i] <= '9')
            ret = 10 * ret + s[i] - '0';
    return ret;
}

int main()
{
    string s = "187821878218782187821878218782";
    __int128 x = parse(s);
    __int128 y = 1ULL << 63;
    __int128 z = 1ULL << 63;
    x *= 2;
    cout << x << endl;
    cout << y << endl;
    cout << y * z << endl;
}
```

2 Java

2.1 template

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

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 InputReader {
    public BufferedReader reader;
    public StringTokenizer tokenizer;

    public InputReader(InputStream stream) {
      reader = new BufferedReader(new InputStreamReader
          (stream), 32768);
      tokenizer = null;
    }

    public String next() {
      while (tokenizer == null || !tokenizer.
          hasMoreTokens()) {
        try {
          tokenizer = new StringTokenizer(reader.
              readLine());
        } catch (IOException e) {
          throw new RuntimeException(e);
        }
      }
      return tokenizer.nextToken();
    }

    public int nextInt() {
      return Integer.parseInt(next());
    }
    public double nextDouble(){
      return Double.parseDouble( next() );
    }

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

3 Data Structure

3.1 Disjoint Set

```
struct DS{
    vector<int> p;
    DS(int n){ for(int i=0;i<n;i++) p.push_back(i); }
        // zero-based
    int find(int x){ return x == p[x] ? x : p[x] = find
        (p[x]); } // Complexity : O(a(N))
    void join(int x,int y){ p[find(x)] = find(y); } //
        Complxity : O(a(N))
};
```

3.2 Segment Tree

```
/* HDU 1166 Partial Code */

struct ST{
    struct Node{
        int value,lazy;
        Node *lc,*rc;
        Node():lc(NULL),rc(NULL),lazy(0){}
        void pull(){ value = lc->value + rc->value; }
        void push(){
            if(!lazy) return;
            if(lc){ lc->lazy = lazy;lc->value += lazy;
                }
            if(rc){ rc->lazy = lazy;rc->value += lazy;
                }
            lazy = 0;
        }
    };

    vector<int> A;

    Node* build(int L,int R){
        Node *node = new Node();
        if(L == R){
            node->value = A[L];
            return node;
        }
        int mid = (L+R)>>1;
        node->lc = build(L,mid);
        node->rc = build(mid+1,R);
        node->pull();
        return node;
    }

    void modify(Node *node,int L,int R,int ql,int qr,
        int d){
        if(R < ql || qr < L) return;
        if(ql <= L && R <= qr){
            node->lazy += d;
            node->value += d;
            return;
        }
        node->push();
        int mid = (L+R)>>1;
        modify(node->lc,L,mid,ql,qr,d);
        modify(node->rc,mid+1,R,ql,qr,d);
        node->pull();
    }

    int query(Node* node,int L,int R,int ql,int qr){
        if(R < ql || qr < L) return 0;
        if(ql <= L && R <= qr) return node->value;
        node->push();
        int mid = (L+R)>>1;
        return query(node->lc,L,mid,ql,qr) + query(node
            ->rc,mid+1,R,ql,qr);
    }

    void delete_(Node* now){
        if(!now) return;
        delete_(now->lc);
        delete_(now->rc);
        delete now;
    }
};
```

3.3 Binary Index Tree

```
/*
can   single update, range query sum
can't insert, delete
*/

struct BIT{
    vector<int> val;

    inline int lsb(int x){ return x & -x; }

    int sum(int x){
        int s = 0;
        while(x > 0){
            s += val[x];
            x -= lower_bit(x);
        }
        return s;
    }

    void update(int x,int d){
        while(x <= A.size()){
            val[x] += d;
            x += lower_bit(x);
        }
    }

    int query(int a,int b){
        if(a > b) swap(a,b);
        return sum(b)-sum(a-1);
    }
};

struct BITRMQ{
    vector<int> val,LT,RT;

    BITRMQ(){}
    BITRMQ(int n,int v):val(n+1,v){
        LT.resize(n+1); RT.resize(n+1);
        for(int i=0;i<=n;i++) LT[i] = RT[i] = i;
    }

    inline int lsb(int x){ return x & -x; }

    int query(int a,int b){
        if(a >= SZ(LT) or b >= SZ(LT) or a < 1 or b <
            1) while(1);
        int s = LLONG_MAX,i = a,x = a,y = b;

        while(x + lsb(x) <= b){
            if(s > val[RT[x]]){ s = val[RT[x]]; i = RT[
                x]; }
            x += lsb(x);
        }
        if(s > val[x]){ s = val[x]; i = x; }
        while(y != x){
            if(s > val[LT[y]]){ s = val[LT[y]]; i = LT[
                y]; }
            y -= lsb(y);
        }

        return i;
    }

    void _magic(int now,int x,int d,bool LR){
        vector<int> &T = LR ? LT : RT;
        if(T[now] != x){ if(val[T[now]] > d) T[now] = x
            ; }
        else{
            int a = now,b = now;
            if(LR) a = now - lsb(now) + 1;
            else b = min(SZ(T)-1,now + lsb(now) - 1);
            if(a <= x-1){
                int s = query(a,x-1);
                if(val[s] < d) T[now] = s;
            }
```

```
            if(x+1 <= b){
                int s = query(x+1,b);
                if(val[s] < d) T[now] = s;
            }
        }
    }

    void update(int x,int d){
        val[x] = d;
        for(int now=x;now<=SZ(val);now+=lsb(now))
            _magic(now,x,d,true);
        for(int now=x;now>0;now-=lsb(now)) _magic(now,x
            ,d,false);
    }
};
```

3.4 zkw Segment Tree.cpp

```
const int NUM = 100;

int M,A[NUM],T[NUM*4];

//    1
//  2   3
// 4 5 6 7
// x 1 2 x
// one-based
void Build(int N){
    while(M=1;M<N+2;M<<=1);
    for(int i=1;i<=N;i++) T[M+i] = A[i];
    for(int i=M-1;i;i--) T[i] = T[i<<1]+T[i<<1|1];
}

// Single modify
void Modify(int x,int d){
    T[x+=M] = d;
    for(x>>1;x;x>>=1) T[x] = T[x<<1]+T[x<<1|1];
}
// Range query
int Query(int L,int R){
    L = L+M-1;R = R+M+1;
    int ans = 0;
    for(;L^R^1;L>>=1,R>>=1){
        if(~L&1) ans += T[L^1];
        if( R&1) ans += T[R^1];
    }
    return ans;
}
```

3.5 Treap

```
struct Treap {
    int key, pri , val ,sz ,  lazy;
    Treap *l, *r;
    Treap(int _key, int _val): key(_key) , val(_val),
        pri(rand()) , sz(1), lazy(0), l(NULL), r(NULL){
         }
};
inline int Size(Treap* t)
{
    return t?t->sz:0;
}
inline void Pull(Treap* t)
{
    t->sz = Size(t->l) + Size(t->r) + 1;
}
void Push(Treap* t)
{
    t->val += t->lazy;
    if (t->l)t->l->lazy += t->lazy;
    if (t->r)t->r->lazy += t->lazy;
    t->lazy = 0;
}
Treap* Merge(Treap* a, Treap* b)
{
    if (!a || !b) return a ? a : b;
    if (a->pri > b->pri) {
        a->r = Merge(a->r, b); Pull(a); return a;
    } else {
        b->l = Merge(a, b->r);Pull(b);return b;
    }
}
void Split(Treap* t, int k, Treap*& a, Treap*& b)
{
    if (!t) a = b = NULL;
    else {
      if (t->key <= k) {
            a = t; Split(t->r, k, a->r, b);Pull(a);
        } else {
            b = t; Split(t->l, k, a, b->l); Pull(b);
        }
    }
}
Treap* Del(Treap* t, int k) //delete all key=k
{
    if (t->key == k) {return Merge(t->l, t->r);
    } else if (k < t->key) { t->l = Del(t->l, k);
        return t;
    } else {  t->r = Del(t->r, k); return t;
    }
}
Treap* insert(Treap* t, int key,int val)
{
    Treap *tl, *tr;
    Split(t, key, tl, tr);
    Treap  tmp(key,val);
    Treap *ans = &tmp;
    Merge(ans,tl); Merge(ans,tr);
    return ans;
}
```

3.6 monotonic-queue

```
template <typename Item>
struct mqueue {
    deque<Item> data, aux;
    void push(Item& x)
    {
        data.push_back(x);
        while (!aux.empty() && aux.back() < x)
            aux.pop_back();
        aux.push_back(x);
    }
    void pop()
    {
        if (data.front() == aux.front())
            aux.pop_front();
        data.pop_front();
    }
    int size()
    {
        return data.size();
    }
    Item max()
    {
        return aux.front();
    }
};
```

4 Graph

4.1 BCC

```
int adj[9][9];
int visit[9], low[9], t = 0;
int stack[9], top = 0;
int contract[9];
void DFS(int i, int p)
{
    visit[i] = low[i] = ++t;
    stack[top++] = i;   // push i
    for (int j=0; j<9; ++j)
        if (adj[i][j]){
            if (!visit[j]) DFS(j, i);             //
                tree edge
```

```
            if (!(j == p && adj[i][j] == 1))
                            // tree edge + back edge
                low[i] = min(low[i], low[j]);
        }

    if (visit[i] == low[i])    // 形成BCC i點會是BCC裡
        面，最早拜訪的點。
    {
        int j;
        do {
            j = stack[--top];   // pop j
            contract[j] = i;
        } while (i != j);
    }
}

void tarjan()
{
    memset(visit, 0, sizeof(visit));
    t = 0;

    for (int i=0; i<9; ++i)
        if (!visit[i])
            DFS(i, i);
}
```

4.2 SCC

```
vector<int> e[10000];int visit[10000], low[10000];bool
    instack[10000];int belong[10000];stack<int> s;
int t;;int num; //number of SCC
void DFS(int u)
{
    visit[u] = low[u] = ++t; //進行標號
    s.push(u); instack[u] = true;
    for (int i = 0; i < e[u].size(); i++) {
        int v = e[u][i];
        if (!visit[v]) {
            DFS(v); low[u] = min(low[u], low[v]); // 找
                u 的最上層祖先
        }
        if (instack[v]) low[u] = min(low[u], visit[v]);
            //還在stack中 用 visit的值
    }
    if (visit[u] == low[u]){//SCC
        num++; int v = s.top();s.pop();
        instack[v] = false;belong[v] = num;
        while (v != u) {
            v = s.top();  s.pop();
            belong[v] = num; instack[v] = false;
        }
    }
}
int Tarjan(int n) //n:number of vertex 0-based
{
    t = 0, num = 0;
    memset(visit, 0, sizeof(visit));
    for (int i = 0; i < n; i++) e[i].clear();
    for (int i = 0; i < n; i++)
        if (!visit[i]) DFS(i);
    return num;
}
```

4.3 SPFA

```
struct Edge {
    int v,cost;
    Edge(int _v=0,int _cost=0):v(_v),cost(_cost){}
};
vector<Edge> E[MAXN]; //MAXN:num of point
bool visited[MAXN];int cnt[MAXN];int dist[MAXN];
bool SPFA(int start , int n)
{
    memset(visited,0,sizeof(visited));
    for(int i=1;i<n;i++) dist[i]=INT_MAX;
    visited[start]=true,dist[start]=0;
    queue<int> que;
    while(!que.empty()) que.pop();
    que.push(start); cnt[start]=1;
    while(!que.empty()){
        int u=que.front();
        que.pop();
        visited[u]=false;
        for(int i=0;i<E[u].size();i++){
            int v=E[u][i].v;
            if(dist[u]!=INT_MAX && dist[v]>dist[u]+E[u
                ][i].cost){
                dist[v]=dist[u]+E[u][i].cost;
                if(!visited[v]) {
                    visited[v]=true;
                    que.push(v);
                    if(++cnt[v]>n) return false; //有負
                        環
                }
            }
        }
    }
    return true; //正常
}
```

4.4 Dijkstra

```
int* Dijkstra(vector<VPII> E,int N,int S){
    bool *visit=new bool[N+1];for(int i=1;i<=N;i++)
        visit[i]=false;
    int *D=new int[N+1];for(int i=1;i<=N;i++)D[i]=INF;
    priority_queue<PII,VPII,greater<PII>> P;
    P.push(MP(0,S));D[S]=0;
    while(!P.empty()){
        int weight=P.top().ff,now=P.top().ss;P.pop();
        if(visit[now])continue;
        visit[now]=true;
        for(auto i:E[now]){
            int potential=D[now]+i.ff;
            if(!visit[i.ss] && potential < D[i.ss]){
                P.push(MP(D[i.ss]=potential,i.ss));
            }
        }
    }
    return D;
}
```

4.5 Floyd-Warshall

```
#include<bits/stdc++.h>
const int N = 500;
int dp[N][N];
void floyd_warshall()
{
    for (int k = 0; k < N; k++)
    for (int i = 0; i < N; i++)
    for (int j = 0; j < N; j++)
    if(dp[i][k]!=INT_MAX && dp[k][j]!=INT_MAX);
    dp[i][j] = min(dp[i][j], dp[i][k] + dp[k][j]);
}
```

4.6 Bipartite Match

```
#include <bits/stdc++.h>
using namespace std;
vector<int> g[10000];
bool check[10000];
int match[10000];
int num_left, num_right;
void init(int n)
{
    num_left = num_right = 0;
    for (int i = 0; i < n; i++)
        g[i].clear();
}
bool DFS(int u)
{
    for (int i = 0; i < g[u].size(); i++) {
        int v = g[u][i];
```

```
        if (!check[v]) {
            check[v] = true;
            if (match[v] == -1 || DFS(match[v]))
                /
                {
                    match[v] = u;
                    match[u] = v;
                    return true;
                }
        }
    }
    return false;
}
int Hungarian_DFS() //匈牙利算法
{
    int ans = 0;
    memset(match, -1, sizeof(match));
    for (int i = 0; i < num_left; i++) { //只要對二分圖
        的一邊即可
        memset(check, 0, sizeof(check));
        if (DFS(i))
            ans++;
    }
    return ans;
}
int Hungarian_BFS()
{
    int prev[10000];
    int ans = 0;
    memset(match, -1, sizeof(match));
    for (int i = 0; i < num_left; i++) {
        memset(check, 0, sizeof(check));
        if (match[i] == -1) {
            queue<int> q;
            q.push(i);
            prev[i] = -1;
            bool flag = false;
            while (!q.empty() && !flag) {
                int u = q.front();
                q.pop();
                for (int j = 0; j < g[u].size() && !
                    flag; j++) {
                    int v = g[u][j];
                    if (!check[v]) {
                        check[v] = true;
                        if (match[v] != -1) {
                            q.push(match[v]);
                            prev[match[v]] = u;
                        } else {
                            flag = true;
                            int d = u, e = v;
                            while (d != -1) {
                                int t = match[d];
                                match[d] = e;
                                match[e] = d;
                                d = prev[d];
                                e = t;
                            }
                        }
                    }
                }
            }
            if (match[i] != -1)
                ans++;
        }
    }
    return ans;
}
```

4.7 KM

```
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 110;
int n;
int g[MAXN][MAXN], lx[MAXN], ly[MAXN];
int match[MAXN], slack[MAXN];
bool vsx[MAXN], vsy[MAXN];

bool find(int x)
{
    if(vsx[x]) return false;
    vsx[x] = 1;
    for (int i = 0; i < n; i++) {
        if (vsy[i])
            continue;
        int t = lx[x] + ly[i] - g[x][i];
        if (!t) {
            vsy[i] = 1;
            if (match[i] == -1 || find(match[i])) {
                match[x] = i;
                return true;
            }
        } else
            slack[i] = min(slack[i], t);
    }
    return false;
}
int km(bool MIN = false) //二分圖最大匹配
{
    if (MIN)
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                g[i][j] = -g[i][j];
    fill(lx, lx + n, INT_MAX);
    fill(ly, ly + n, 0);
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            lx[i] = min(lx[i], g[i][j]);
    memset(match, -1, sizeof(match));
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++)
            fill(slack, slack + n, INT_MAX);
        while (true) {
            memset(vsx, 0, sizeof(vsx));
            memset(vsy, 0, sizeof(vsy));
            if (find(i))
                break;
            int d = INT_MAX;
            for (int j = 0; j < n; j++)
                if (!vsy[j])
                    d = min(d, slack[j]);
            for (int j = 0; j < n; j++) {
                if (vsx[j])
                    lx[j] -= d;
                if (vsy[j])
                    ly[j] += d;
                else
                    slack[j] -= d;
            }
        }
    }
    int sum = 0;
    for (int i = 0; i < n; i++)
        sum += g[match[i]][i];
    if (MIN) {
        sum = -sum;
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                g[i][j] = -g[i][j];
    }
    return sum;
}
```

4.8 General-Match

```
#include <bits/stdc++.h>
using namespace std;
struct DisjointSet {
    int N;
    vector<int> p;
    DisjointSet(int n)
        : N(n)
        , p(vector<int>(N))
    {
        init();
    }
    void init()
    {
        for (int i = 0; i < N; i++)
```

```
            p[i] = i;
    }
    int find(int x)
    {
        return p[x] == x ? x : p[x] = find(p[x]);
    }
    void U(int a, int b)
    {
        p[find(b)] = find(a);
    }
};
struct GMatch {
    int N;
    vector<vector<int> > vc;
    DisjointSet djs;
    vector<int> m, d, c1, c2, p, vis;
    queue<int> q;
    int ts;
    GMatch(int n)
        : N(n)
        , vc(vector<vector<int> >(N + 1))
        , djs(DisjointSet(N))
        , ts(0)
    {
    }
    void add(int a, int b)
    {
        vc[a].push_back(b);
        vc[b].push_back(a);
    }
    void path(int x, int r)
    {
        if (x == r)
            return;
        if (d[x] == 0) {
            int i = p[x], j = p[p[x]];
            path(j, r);
            m[i] = j, m[j] = i;

        } else if (d[x] == 1) {
            int i = c1[x], j = c2[x];
            path(i, m[x]);
            path(j, r);
            m[i] = j, m[j] = i;
        }
    }
    void blossom(int x, int y, int bi)
    {
        for (int i = djs.find(x); i != bi; i = djs.find
            (p[i])) {
            djs.U(bi, i);
            if (d[i] == 1)
                c1[i] = x, c2[i] = y, q.push(i);
        }
    }
    int lca(int x, int y, int r)
    {
        ts++;
        vis[r] = ts;
        for (int i = djs.find(x); i != r; i = djs.find(
            p[i]))
            vis[i] = ts;
        int b;
        for (b = djs.find(y); vis[b] != ts; b = djs.
            find(p[b]))
            ;
        return b;
    }
    bool Match(int x)
    {
        djs.init();
        d = vector<int>(N + 1, -1);
        d[x] = 0;
        q = queue<int>();
        q.push(x);
        while (!q.empty()) {
            int u = q.front();
            q.pop();
            for (int v : vc[u]) {
                if (m[v] != v && djs.find(u) != djs.
                    find(v)) {
                    if (d[v] == -1) {
                        if (m[v] == -1) {
                            path(u, x);
                            m[u] = v, m[v] = u;
                            return true;

                        } else {
                            p[v] = u, p[m[v]] = v;
                            d[v] = 1, d[m[v]] = 0;
                            q.push(m[v]);
                        }

                    } else {
                        if (d[djs.find(v)] == 0) {
                            int bi = lca(u, v, x);
                            blossom(u, v, bi);
                            blossom(v, u, bi);
                        }
                    }
                }
            }
        }
        return false;
    }
    int Solve()
    {
        m = c1 = c2 = d = p = vis = vector<int>(N + 1,
            -1);
        int ans = 0;
        for (int i = 0; i < N; i++) {
            if (m[i] == -1) {
                if (Match(i))
                    ans++;
                else
                    m[i] = i;
            }
        }
        return ans;
    }
};
```

4.9 Directed-MST

```
const int MAXN = 1010;
int pre[MAXN], min_dist[MAXN];
struct Edge {
    int from, to, cost;
    Edge() {}
    Edge(int _from, int _to, int _cost)
        : from(_from)
        , to(_to)
        , cost(_cost)
    {
    }
};
vector<Edge> E;
int solve(int n, int m, int root)
{
    int ans = 0;
    while (true) {
        fill(min_dist, min_dist + MAXN, INT_MAX);
        for (int i = 0; i < E.size(); i++) {
            int u = E[i].from, v = E[i].to, cost = E[i
                ].cost;
            if (cost < min_dist[v] && v != u) {
                min_dist[v] = cost;
                pre[v] = u;
            }
        }
        for (int i = 1; i <= n; i++)
            if (min_dist[i] == INT_MAX && i != root)
                return -1;
        int cnt_node = 1, id[MAXN], vis[MAXN];
        memset(id, -1, sizeof(id));
        memset(vis, 0, sizeof(vis));
        min_dist[root] = 0;
        for (int i = 1; i <= n; i++) {
            ans += min_dist[i];
            int v = i;
            while (vis[v] != i && id[v] == -1 && v !=
                root) {
```

```
                vis[v] = i;
                v = pre[v];
            }
            if (id[v] == -1 && v != root) {
                for (int u = v; u != v; u = pre[u])
                    id[u] = cnt_node;
                cnt_node++;
            }
        }
        if (cnt_node == 1)
            break;
        for (int i = 1; i <= n; i++)
            if (id[i] == -1)
                id[i] = cntnode++;
        for (int i = 0; i < E.size(); i++) {
            int v = E[i].to;
            E[i].from = id[E[i].from];
            E[i].to = id[E[i].to];
            if (E[i].from ! = E[i].to)
                E[i].cost -= min_dist[v];
        }
        n = cntnode - 1;
        root = id[root];
    }
    return ans;
}
```

4.10 LCA

```
/*
 多個點的LCA = >  DFS走訪順序中min 和 max 的LCA
 */
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 1000;
vector<int> tree[MAXN];
int depth[MAXN];
int father[MAXN][20];
void init()
{
    memset(depth, 0, sizeof(depth));
    memset(father, -1, sizeof(father));
}
void dfs(int u)
{
    for (int i = 0; i < tree[u].size(); i++) {
        int v = tree[u][i];
        if (!depth[v]) {
            depth[v] = depth[u] + 1;
            father[v][0] = u;
            dfs(v);
        }
    }
}
void build()
{
    for (int i = 1; (1 << i) < MAXN; i++) {
        for (int j = 0; j < MAXN; j++) {
            if (father[j][i - 1] != -1) {
                father[j][i] = father[father[j][i -
                    1]][i - 1];
            }
        }
    }
}
int lca(int u,int v)
{
    if(depth[u]<depth[v]) swap(u,v);
    for(int i =log2(MAXN-1);i>=0;i--)
    {
        if( father[u][i]!=-1 && depth[father[u][i]]>=
            depth[v]) u=father[u][i];
    }
    if(u==v) return v;
    for(int i=log2(MAXN-1);i>=0;i--)
    {
        if(father[u][i]!=father[v][i])
            u=father[u][i],v=father[v][i];
    }
    return father[u][0];
}
```

4.11 MST-kruskal

```
#include<bits/stdc++.h>
using namespace std;
const int N=20005;
struct Edge{
    int u,v,cost;
    Edge(int _u=0,int _v=0,int _cost=0){
        u=_u,v=_v,cost=_cost;
    }
    bool operator < (const Edge & a) const{
        return cost<a.cost;
    }
};
vector<Edge> E;
int ds[N];
void Init (int n){
    E.clear();
    for(int i=0;i<n;i++) ds[i]=i;
}
int Find(int x){
    return (ds[x]==x)?x:(ds[x]=Find(ds[x]));
}
int kruskal(int n) //point_number;
{
    sort(E.begin(),E.end());
    int ans=0;
    int num=1;
    for( auto &e : E ){
        int u=e.u,v=e.v;
        u=Find(u);v=Find(v);
        if(u!=v){
            ds[u]=v;
            num++;
            ans+=e.cost;
        }
        if(num==n) break;
    }
    return ans;

}
```

4.12 Manhattan-Mst

```
#include <bits/stdc++.h>
using namespace std;
const int N = 100100;
struct Point {
    int x, y, id;
    Point(int _x, int _y, int _id)
    : x(_x)
    , y(_y)
    , id(_id)
    {
    }
    bool operator<(const Point& p) const
    {
        return (x != p.x) ? x < p.x : y < p.y;
    }
};

struct BIT {
    int min_val, pos;
    void init()
    {
        min_val = INT_MAX;
        pos = -1;
    }
} bit[N];
struct Edge {
    int u, v, d;
    Edge() {}
    Edge(int _u, int _v, int _d)
    : u(_u)
    , v(_v)
    , d(_d)
```

```
    {
    }
    bool operator<(const Edge& e) const
    {
        return d < e.d;
    }
};
vector<Point> p;
vector<Edge> E;
int ds[N]; //disjoint set for kruskal

void add_edge(int u, int v, int d)
{
    E.push_back(Edge(u, v, d));
}
int find(int x)
{
    return ds[x] = (x == ds[x] ? x : find(ds[x]));
}
int dist(int i, int j)
{
    return abs(p[i].x - p[j].x) + abs(p[i].y - p[j].y);
}
inline int lowbit(int x)
{
    return x & (-x);
}
void update(int x, int val, int pos)
{
    for (int i = x; i >= 1; i -= lowbit(i)) {
        if (val < bit[i].min_val)
            bit[i].min_val = val, bit[i].pos = pos;
    }
}
int query(int x, int m)
{
    int min_val = INT_MAX, pos = -1;
    for (int i = x; i <= m; i += lowbit(i)) {
        if (bit[i].min_val < min_val) {
            min_val = bit[i].min_val, pos = bit[i].pos;
        }
    }
    return pos;
}
int Manhattan_MST(vector<Point>& P)
{
    int n = P.size();
    for (int dir = 0; dir < 4; dir++) {
        if (dir == 1 || dir == 3) {
            for (int i = 0; i < n; i++)
                swap(P[i].x, P[i].y);
        } else if (dir == 2) {
            for (int i = 0; i < n; i++)
                P[i].x = -P[i].x;
        }

        int T[N], hs[N];
        sort(P.begin(), P.end());
        for (int i = 0; i < n; i++) { //discretize
            T[i] = hs[i] = p[i].y - p[i].x;
        }
        sort(hs, hs + n);
        int m = unique(hs, hs + n) - hs;
        for (int i = 1; i <= m; i++)
            bit[i].init();
        for (int i = n - 1; i >= 0; i--) {
            int pos = lower_bound(hs, hs + m, T[i]) -
                hs + 1; //Bit is 1-based
            int w = query(pos, m);
            if (w != -1)
                add_edge(p[i].id, p[w].id, dist(i, w));
            update(pos, p[i].x + p[i].y, i);
        }
    }
    sort(E.begin(), E.end());
    int ans = 0;
    int p = 1;
    for (int i = 0; i < n; i++)
        ds[i] = i;
    for (int i = 0; i < (int)E.size() && p <= n; i++) {
        int fa = find(E[i].u);
        int fb = find(E[i].v);
        if (fa != fb) {
            p++;
            ds[fa] = fb;
            ans += E[i].d;
        }
    }
    return ans;
}
```

4.13 Flow-Dinic

```
#include <bits/stdc++.h>
#define pb push_back
#define SZ(x) (int)x.size()
using namespace std;
struct Dinic {
    struct Edge {
        int v, f, re;
        Edge(int v, int f, int re)
            : v(v)
            , f(f)
            , re(re)
        {
        }
    };
    vector<vector<Edge> > E;
    vector<int> level;
    int N, s, t;
    Dinic(int N, int s, int t)
        : N(N)
        , s(s)
        , t(t)
    {
        E.resize(N + 1);
    }
    void AddEdge(int u, int v, int c)
    {
        E[u].pb({ v, c, SZ(E[v]) });
        E[v].pb({ u, 0, SZ(E[u]) - 1 });
    }
    bool BFS()
    {
        level.clear();
        for (int i = 0; i <= N; i++)
            level.pb(-1);
        queue<int> Q;
        Q.push(s);
        level[s] = 0;
        while (!Q.empty()) {
            int now = Q.front();
            Q.pop();
            for (auto i : E[now]) {
                if (i.f > 0 && level[i.v] == -1) {
                    level[i.v] = level[now] + 1;
                    Q.push(i.v);
                }
            }
        }
        return level[t] != -1;
    }
    int DFS(int now, int nf)
    {
        if (now == t)
            return nf;
        int ans = 0;
        for (auto& i : E[now]) {
            if (i.f > 0 && level[i.v] == level[now] +
                1) {
                int tf = DFS(i.v, min(nf, i.f));
                ans += tf;
                nf -= tf;
                i.f -= tf;
                E[i.v][i.re].f += tf;
                if (nf == 0)
                    return ans;
            }
        }
        if (!ans)
            level[now] = -1;
        return ans;
```

```
    }
    int Flow()
    {
        int ans = 0;
        while (BFS())
            ans += DFS(s, INT_MAX);
        return ans;
    }
};
```

4.14 Flow-MinCost

```
#include <bits/stdc++.h>
using namespace std;
const int N = 1000;
struct Edge {
    int v; //連的點
    int cap; //容量
    int cost;
    int next; //下一條邊的
} e[N * N];
int id;
int p[N];
int pre[N];
int path[N];
int dist[N]; //dist;

void init() //初始化
{
    memset(e, 0, sizeof(e));
    memset(p, -1, sizeof(p));
    id = 0;
}
void add(int u, int v, int cap, int cost)
{
    e[id].v = v, e[id].cap = cap, e[id].cost = cost, e[
        id].next = p[u], p[u] = id++;
    e[id].v = u, e[id].cap = 0, e[id].cost = -cost, e[
        id].next = p[v], p[v] = id++;
}
bool SPFA(int s, int t)
{
    memset(pre, -1, sizeof(pre));
    fill(dist, dist + N, INT_MAX);
    bool vis[N]={};
    dist[s] = 0;
    queue<int> q;
    q.push(s);
    vis[s]=true;
    while (!q.empty()) {
        int u = q.front();
        q.pop();
        vis[u]=false;
        for (int i = p[u]; i != -1; i = e[i].next) {
            int v = e[i].v;
            if (e[i].cap > 0 && dist[u] + e[i].cost <
                dist[v]) {
                dist[v] = dist[u] + e[i].cost;
                pre[v] = u; //路徑
                path[v] = i; //邊的編號
                if(!vis[v])
                vis[v]=true,q.push(v);
            }
        }
    }
    if (pre[t] == -1)
        return false;
    return true;
}
int MinCostFlow(int s, int t)
{
    int cost = 0;
    int flow = 0;
    while (SPFA(s, t)) {
        int f = INT_MAX;
        for (int u = t; u != s; u = pre[u])
            f = min(f, e[path[u]].cap);
        flow+=f;
        cost+=dist[t]*f;
        for(int u=t;u!=s;u=pre[u]){
            e[path[u]].cap-=f;
            e[path[u]^1].cap+=f;
        }
    }
    return cost; //cost
}
```

4.15 HeavyLight-Decomposition

```
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 3000;
int size[MAXN], pre[MAXN], son[MAXN], dep[MAXN];
vector<int> E[MAXN];
struct Node{ // segmnet tree with lazy tag;
    Node * l ,* r ;
    int v,lazy;
}*root;
//1-based;
int dfs(int x, int fa)
{
    size[x] = 1;
    int max_v = INT_MIN;
    dep[x] = dep[fa] + 1;
    pre[x]=fa;
    for(auto &v:E[x]){
        size[x] += dfs(v, x);
        if (size[v] > max_v) {
            max_v = size[v];
            son[x] = v;
        }
    }
    return size[x];
}
int no,pos[MAXN],top[MAXN];
int repos(int x ,int fa,int tp){
    pos[x]=++no;
    top[x]=tp;
    if(son[x]) repos(son[x],x,tp);
    for(auto &v:E[x]){
        if(v!=son[x] && v!=fa) repos(v,x,v);
    }
}
// 1-based segment tree
void update_seg(Node* root,int l,int r ,int ql,int qr ,
    int v){
    ;
}
void query_seg(Node* root,int l,int r ,int ql,int qr){
    ;
}
void update(int x, int y,int v)
{
    while(top[x]!=top[y]){
        if(dep[x]<dep[y]) swap(x,y);
        update_seg(root,1,MAXN,pos[top[x]],pos[x],v);
        x=pre[top[x]];
    }
    update_seg(root,1,MAXN,min(pos[x],pos[y]),max(pos[x
        ],pos[y]),v);
}
//query from x to y
void query(int x ,int y){
    int ans;
    while(top[x]!=top[y]){
        if(dep[x]<dep[y]) swap(x,y);
        ans+=query_seg(root,1,MAXN,pos[top[x]],pos[x]);
        x=pre[top[x]];
    }
    ans+=query_seg(root,1,MAXN,min(pos[x],pos[y]),max(
        pos[x],pos[y]));

}
```

4.16 Maximal-Clique

```
/*
 * compute maximal cliques
```

```
 * |maximum clique|=
 * |comp(G)'s maximum independent set|
 * comp(G) Complete Graph's Edge - G's Edge
 */
#include <bits/stdc++.h>
using namespace std;
typedef unsigned long long ull;

ull adj[64];
vector<ull> cliques; // if ith bit is 1 then i is in
    that maximal clique

void BronKerbosch(ull R, ull P, ull X)
{
    if (P == 0 && X == 0) {
        cliques.push_back(R);
    }
    /*
     * Returns the number of trailing 0-bits in x,
         starting at the least significant bit position
        .
     * If x is 0, the result is undefined.
     */
    int p = __builtin_ctzll(P | X);
    ull Q = P & ~adj[p];
    while (Q) {
        int i = __builtin_ctzll(Q);
        BronKerbosch(R | (1ULL << i), P & adj[i], X &
            adj[i]);
        Q&=~(1ULL<<i) ; P&=~(1ULL<<i); X|=(1ULL<<i);
    }
}
```

4.17 2SAT

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

const int MAXN = 2010;
const int VN = MAXN * 2;
const int EN = 4000010;
#define False(a) a * 2
#define True(a) a * 2 + 1
class Graph {
public:
    void init()
    {
        size = 0;
        memset(head, -1, sizeof(head));
    }
    /*
    1）如果給出A和B的限制關係，A和B必須一起選，(A and B
        )||(!A and !B )==true 那麼選A必須選B，建邊<i,j>
        和<j,i>還有<i',j'>和<j',i'>
    2）如果給出A和B的限制關係，選A不能選B，那麼(A && !B
        )||(!A && B )==true,建邊<i,j'>和<j,i'>
    3）如果必須選A,那麼A==true,建邊<i',i>
    4）如果A一定不能選，那麼!A==true.建邊<i,i'>
  */
    void addEdge(int u, int v)
    {
        E[size].v = v;
        E[size].next = head[u];
        head[u] = size++;
    }

public:
    int size;
    int head[VN];
    struct Edge {
        int v, next;
    } E[EN];
} g;

class Two_Sat {
public:
    bool check(const Graph& g, const int n)
    {
        scc(g, n);
        for (int i = 0; i < n; ++i)
            if (belong[i * 2] == belong[i * 2 + 1])
                return false;
        return true;
    }

private:
    void tarjan(const Graph& g, const int u)
    {
        int v;
        DFN[u] = low[u] = ++idx;
        sta[top++] = u;
        inStack[u] = true;

        for (int e = g.head[u]; e != -1; e = g.E[e].
            next) {
            v = g.E[e].v;
            if (DFN[v] == -1) {
                tarjan(g, v);
                low[u] = min(low[u], low[v]);
            } else if (inStack[v]) {
                low[u] = min(low[u], DFN[v]);
            }
        }
        if (low[u] == DFN[u]) {
            ++bcnt;
            do {
                v = sta[--top];
                inStack[v] = false;
                belong[v] = bcnt;
            } while (u != v);
        }
    }
    void scc(const Graph& g, const int n)
    {
        top = idx = bcnt = 0;
        memset(DFN, -1, sizeof(DFN));
        memset(inStack, 0, sizeof(inStack));
        for (int i = 0; i < 2 * n; ++i) {
            if (DFN[i] == -1)
                tarjan(g, i);
        }
    }

private:
    int top, idx, bcnt;
    int sta[VN];
    int DFN[VN];
    int low[VN];
    int belong[VN];
    bool inStack[VN];
} sat;
```

5 String Theory

5.1 KMP

```
// Complexity : O(T+P)
void predo(string pattern,int dp[]){
    dp[0] = 0;
    for(int i=1;i<pattern.size();i++){
        dp[i] = dp[i-1];
        while(dp[i] > 0 && pattern[dp[i]] != pattern[i
            ]) dp[i] = dp[dp[i]-1];
        if(pattern[dp[i]] == pattern[i]) dp[i]++;
    }
}

void KMP(string text,string pattern){
    int dp[pattern.size()];predo(pattern,dp);
    for(int i=0,match=0;i<text.size();i++){
        while(match > 0 && pattern[match] != text[i])
            match = dp[match-1];
        if(pattern[match] == text[i]) match++;
        if(match == pattern.size()){
            // do something with i-pattern.size()+1
            match = dp[match-1];
        }
```

```
    }
}
```

5.2 Z

```
void ZAlgorithm(string word,string pattern){
    int Z[word.size()+pattern.size()];
    string S = pattern+word;
    Z[0] = 0;
    for(int i=1,best=0;i<S.size();i++){
      if(best+Z[best] <= i)Z[i] = 0;
      else Z[i] = min(Z[i-best],best+Z[best]-i);
      while(S[i+Z[i]] == S[Z[i]])Z[i]++;
      if(i+Z[i] > best+Z[best])best = i;
    }
    for(int i=pattern.size();i<S.size();i++){
        if(Z[i] >= pattern.size())cout << i-pattern.
            size() << " ";
    }
}
```

5.3 Trie

```
const int MAXCHAR = 10;
const char CHAR = '0';

struct Node{
    Node* child[MAXCHAR];
    int N;
    Node():N(0){for(int i=0;i<MAXCHAR;i++)child[i] =
        NULL;}
};
Node* root = new Node;
void word(string s){
    Node* now = root;
    for(int i=0;i<s.size();i++){
        int c = s[i] - CHAR;
        if(now->child[c] == NULL)now->child[c] = new
            Node;
        now = now->child[c];
    }
    now->N++;
}
void release(Node* now = root){
    for(int i=0;i<MAXCHAR;i++)if(now->child[i])release(
        now->child[i]);
    delete now;
}
```

5.4 AC automaton

```
const int MAXCHAR = 26;
const char CHAR = 'a';

struct Node{
    Node* child[MAXCHAR];
    Node* fail;
    int N;
    Node():N(-1),fail(NULL){for(int i=0;i<MAXCHAR;i++)
        child[i] = NULL;}
};

struct AC{
    Node* root;
    AC(){root = new Node;}
    void word(string s,int index){
        Node* now = root;
        for(int i=0;i<s.size();i++){
            int c = s[i] - CHAR;
            if(now->child[c] == NULL)now->child[c] =
                new Node;
            now = now->child[c];
        }
        if(now->N == -1)now->N = index;
    }
    void predo(){
        root->fail = NULL;
        Node* p;
        queue<Node*> Q;
        Q.push(root);
        while(!Q.empty()){
            Node* now = Q.front();Q.pop();
            for(int i=0;i<MAXCHAR;i++){
                if(!now->child[i])continue;
                Q.push(now->child[i]);
                p = now->fail;
                while(p != NULL && p->child[i] == NULL)
                    p = p->fail;
                if(p == NULL)now->child[i]->fail = root
                    ;
                else now->child[i]->fail = p->child[i];
            }
        }
    }
    void match(string text){
        Node* now = root;
        for(int i=0;i<text.size();i++){
            int c = text[i] - CHAR;
            while(now != root && now->child[c] == NULL)
                now = now->fail;
            if(now->child[c])now = now->child[c];
            if(now->N != -1)cout << "Got you" << endl;
        }
    }
    void release(Node* now = root){
        for(int i=0;i<MAXCHAR;i++)if(now->child[i])
            release(now->child[i]);
        delete now;
    }
};
```

5.5 Suffix Array

```
int SA[MAXNUM],H[MAXNUM];
void SuffixArray(string text){
    int N = text.size(),A = 128;
    int SA2[MAXNUM],rank[MAXNUM],rank2[MAXNUM],radix[
        MAXNUM];
    for(int i=0;i<A;i++)radix[i] = 0;
    for(int i=0;i<N;i++)radix[rank[i] = text[i]]++;
    for(int i=0;i<A;i++)radix[i] += radix[i-1];
    for(int i=N-1;i>=0;i--)SA[--radix[text[i]]] = i;

    for(int power=1;power<N;power<<=1){
        for(int i=0;i<A;i++)radix[i] = 0;
        for(int i=0;i<N;i++)radix[rank[i]]++;
        for(int i=0;i<A;i++)radix[i] += radix[i-1];

        int now = 0;
        for(int i=N-power;i<N;i++)SA2[now++] = i;
        for(int i=0;i<N;i++){
            if(SA[i]-power >= 0)SA2[now++] = SA[i]-
                power;
        }

        for(int i=N-1;i>=0;i--)SA[--radix[rank[SA2[i
            ]]]] = SA2[i];

        rank2[SA[0]] = now = 0;
        for (int i=1;i<N;i++){
            if (!(rank[SA[i-1]] == rank[SA[i]] && SA[i
                -1]+power < N && SA[i]+power < N &&
                rank[SA[i-1]+power] == rank[SA[i]+power
                ]))now++;
            rank2[SA[i]] = now;
        }
        swap(rank,rank2);
        if(now == N-1)break;
        A = now+1;
    }
    for(int i=0;i<N;i++)rank[SA[i]] = i;
    for(int i=0,k=0;i<N;i++,k?k--:0){
        if(rank[i] == 0){H[rank[i]] = 0;continue;}
        int j = SA[rank[i]-1];
        while(i+k < N && j+k < N && text[i+k] == text[j
            +k])k++;
```

```
        H[rank[i]] = k;
    }
}
```

6 Geometry

6.1 Point

```
#include <bits/stdc++.h>
using namespace std;
const double EPS = 1e-6;
int dcmp(double x)
{
    if (fabs(x) < EPS)
        return 0;
    else
        return x < 0 ? -1 : 1;
}
struct Point {
    double x, y;
    Point() { x = 0, y = 0; }
    Point(double _x, double _y)
    {
        x = _x;
        y = _y;
    }
    Point operator+(const Point& b)
    {
        return Point(x + b.x, y + b.y);
    }
    Point operator-(const Point& b) const
    {
        return Point(x - b.x, y - b.y);
    }
    Point operator*(double p)
    {
        return Point(x * p, y * p);
    }
    Point operator/(double p)
    {
        return Point(x / p, y / p);
    }
    double operator^(const Point& b) const
    {
        return x * b.y - y * b.x;
    }
    bool operator<(const Point& b)
    {
        return x < b.x || (x == b.x && y < b.y);
    }
    bool operator==(const Point& b)
    {
        return dcmp(x - b.x) == 0 && dcmp(y - b.y) ==
            0;
    }
};
typedef Point Vector;
double dot(Vector v1, Vector v2)
{
    return v1.x * v2.x + v1.y * v2.y;
}
double cross(Point& o, Point& a, Point& b) //OA X OB
{
    return (a.x - o.x) * (b.y - o.y) - (a.y - o.y) * (b
        .x - o.x);
}
double cross(Vector a, Vector b)
{
    return a.x * b.y - a.y * b.x;
}
double length(Vector v)
{
    return sqrt(v.x * v.x + v.y * v.y); //return sqrt(
        dot(v,v));
}
double length(Point a, Point b)
{
    return length(a - b);
}
double angle(const Vector& a, const Vector& b) { return
     acos(dot(a, b) / length(a) / length(b)); }
double Triarea(const Point& p1, const Point& p2, const
    Point& p3)
{
    return fabs(cross(p2 - p1, p3 - p1)) / 2;
}
Vector Rotate(const Vector& a, double rad)
{ //radian 0~2pi //counterclockwise{
    return Vector(a.x * cos(rad) - a.y * sin(rad), a.x
        * sin(rad) + a.y * cos(rad)); //旋轉矩陣
}
Vector Normal(const Vector& a)
{ //向量的單位法線

    double L = length(a);
    return Vector(-a.y / L, a.x / L);
}
struct Line {
    Point p1, p2;
};
typedef Line Segment;
Point GetLineIntersection(Point p, Vector v, Point q,
    Vector w) //點斜式交點 p+vt1 q+wt2
{
    Vector u = p - q;
    double t = cross(w, u) / cross(v, w); //t1
    return p + v * t; //p+vt1
}
Point GetLineProjection(Point p, Point a, Point b)
{
    Vector v = b - a;
    return a + v * (dot(v, p - a) / dot(v, v));
}
typedef Line Segment;
bool SegmentProperIntersection(Point a1, Point a2,
    Point b1, Point b2)
{
    int c1 = cross(b1 - a1, b2 - a1), c2 = cross(b1 -
        a2, b2 - a2);
    int c3 = cross(a1 - b1, a2 - b1), c4 = cross(a1 -
        b2, a2 - b2);
    return dcmp(c1) * dcmp(c2) < 0 && dcmp(c3) * dcmp(
        c4) < 0;
}
bool SegmentProperIntersection(Segment s1, Segment s2)
{
    return SegmentProperIntersection(s1.p1, s1.p2, s2.
        p1, s2.p2);
}
bool Onsegment(Point p, Point a1, Point a2)
{
    return dcmp(cross(p - a2, a1 - a2)) == 0 && dcmp(
        dot(a1 - p, a2 - p)) <= 0;
}
bool SegmentIntersection(Point a1, Point a2, Point b1,
    Point b2)
{
    if (cross(a2 - a1, b2 - b1) == 0)
        return false;
    if (Onsegment(a1, b1, b2) || Onsegment(a2, b1, b2)
        || Onsegment(b1, a1, a2) || Onsegment(b2, a1,
        a2))
        return true;
    if (SegmentProperIntersection(a1, a2, b1, b2))
        return true;
    return false;
}
bool SegmentIntersection(Line& l1, Line& l2)
{
    return SegmentIntersection(l1.p1, l1.p2, l2.p1, l2.
        p2);
}
double distance(Point& a, Point& b)
{
    return sqrt(length(b - a));
}
double distance(Point& p, Point& p1, Point& p2) //Line
    => p1,p2
{
    Vector v1 = p - p1, v2 = p2 - p1;
```

```
    return fabs(cross(v1, v2)) / length(v2); //面積/底=
        高(距離)
}
double distance(Point& p, Segment& s) //Point to
    Segment
{
    Vector v = s.p2 - s.p1;
    if (dcmp(length(v)) == 0)
        return length(p - s.p1); //線段退化成點
    Vector v1 = p - s.p1;
    Vector v2 = p - s.p2;
    if (dcmp(dot(v1, v)) < 0)
        return length(v1); // 點投影不在線上
    if (dcmp(dot(v2, v)) > 0)
        return length(v2); // 點投影不在線上
    return fabs(cross(v, v1)) / length(v);
}
double distance(Segment& s1, Segment& s2) //線段到線段
{
    if (SegmentIntersection(s1, s2))
        return 0;
    double d = 1e9;
    d = min(d, distance(s1.p1, s2)); //點到線段距離取最
        短
    d = min(d, distance(s1.p2, s2));
    d = min(d, distance(s2.p1, s1));
    d = min(d, distance(s2.p2, s1));
    return d;
}
double ldistance(Line& l1, Line& l2) //線段到線段距離
{
    Vector v1 = l1.p2 - l1.p1;
    Vector v2 = l2.p2 - l2.p1;
    if (cross(v1, v2) != 0)
        return 0;
    return distance(l1.p1, l2); //點到線段距離
}
void ConvexHull(vector<Point>& P, vector<Point>& res)
{
    sort(P.begin(), P.end());
    auto last = unique(P.begin(), P.end());
    P.erase(last, P.end());
    int cnt = P.size();
    res.resize(cnt);
    int m = 0;
    for (int i = 0; i < cnt; i++) {
        while (m > 1 && cross(res[m - 1] - res[m - 2],
            P[i] - res[m - 2]) <= 0)
            m--;
        res[m++] = P[i];
    }
    int k = m;
    for (int i = cnt - 2; i >= 0; i--) {
        while (m > k && cross(res[m - 1] - res[m - 2],
            P[i] - res[m - 2]) <= 0)
            m--;
        res[m++] = P[i];
    }
    if (cnt > 1)
        m--;
    res.resize(m);
}
double PolygonArea(Point* p, int n)
{
    double area = 0;
    for (int i = 0; i < n; ++i)
        area += cross(p[i], p[(i + 1) % n]);
    return fabs(area) / 2;
}
//半平面交
typedef vector<Point> Polygon;
Polygon halfplane_intersection(Polygon& p, Line& line)
{
    Polygon q;
    Point p1 = line.p1, p2 = line.p2;
    int n = p.size();
    for (int i = 0; i < n; i++) {
        double c = cross(p1, p2, p[i]);
        double d = cross(p1, p2, p[(i + 1) % n]);
        if (dcmp(c) >= 0)
            q.push_back(p[i]);
        if (dcmp(c * d) < 0)
            q.push_back(GetLineIntersection(p1, p2, p[i
                ], p[(i + 1) % n]));
    }
    return q;
}
```

6.2 rotating-caliper

```
#include "Point.cpp"

void rotating_caliper(vector<Point> P)
{
    sort(P.begin(), P.end());
    int l = 0, u = 0;
    Point L[10000], U[10000];
    int cnt = P.size();
    for (int i = 0; i < cnt; i++) {
        while (l >= 2 && cross(L[l - 2], L[l - 1], P[i
            ]) <= 0)
            l--;
        while (u >= 2 && cross(U[l - 2], U[l - 1], P[i
            ]) >= 0)
            u--;
        L[l++] = P[i];
        U[u++] = P[i];
    }
    if(u>=2) L[l]=U[u-2];
    for(int i=0,j=u-1;i<l && j>0;){
        //compute L[i] and U[j];
        if(cross(L[i+1]-L[i],U[j-1]-U[j])<0) i++;
        else j--;
    }
}
```

6.3 closet-pair

```
#include "Point.cpp"
bool cmpy(const Point& i, const Point& j) { return i.y
    < j.y; }
vector<Point> p;
double DnC(int L, int R, vector<Point>& p) // 區間
{
    if (L >= R)
        return 1e-9;
    if (L + 1 == R) {
        return length(p[L], p[R]);
    }
    int M = (L + R) >> 1;
    double d = min(DnC(L, M, p), DnC(M + 1, R, p));
    if (dcmp(d) == 0)
        return 0;
    int N = 0;
    Point t[10000];
    for (int i = M; i >= L && p[M].x - p[i].x < d; --i)
         {
        t[N++] = p[i];
    }
    for (int i = M + 1; i <= L && p[i].x - p[M].x < d;
        ++i) {
        t[N++] = p[i];
    }
    sort(t, t + N, cmpy);
    for (int i = 0; i < N; i++) {
        for (int j = 1; j <= 3; j++) {
            d = min(d, length(t[i], t[i + j]));
        }
    }
    return d;
}
double closet_pair(vector<Point>& p)
{
    sort(p.begin(),p.end());
    return DnC(0,p.size(),p);
}
```

6.4 Minimum-Cover-Circle

```
const double eps = 1e-7;

struct Point{
    double x,y;
    Point(){}
    Point(double x,double y):x(x),y(y){}
};

Point Circumcenter(Point a,Point b,Point c){
    double a1 = b.x - a.x, b1 = b.y - a.y, c1 = (a1*a1
        + b1*b1)/2;
    double a2 = c.x - a.x, b2 = c.y - a.y, c2 = (a2*a2
        + b2*b2)/2;
    double d = a1 * b2 - a2 * b1;
    return Point(a.x + (c1*b2 - c2*b1)/d,a.y + (a1*c2 -
         a2*c1)/d);
}

double Distance(Point A,Point B){
    return sqrt((A.x-B.x)*(A.x-B.x)+(A.y-B.y)*(A.y-B.y)
        );
}

// Expected Complexity : O(N)
pair<Point,double> MinimumCoverCircle(vector<Point> P){
  random_shuffle(P.begin(),P.end());
    Point center = P[0];
    double R = 0.0;
    for(int i=1;i<P.size();i++) if(Distance(center,P[i
        ])+eps > R){
    center = P[i];R = 0.0;
    for(int j=0;j<i;j++) if(Distance(center,P[j])+eps >
         R){
      center.x=(P[i].x+P[j].x)/2.0;
      center.y=(P[i].y+P[j].y)/2.0;
      R = Distance(center,P[j]);
      for(int k=0;k<j;k++) if(Distance(center,P[k])+eps
           > R){
        center = Circumcenter(P[i],P[j],P[k]);
        R = Distance(center,P[k]);
      }
    }
    }
  return make_pair(center,R);
}
```

6.5 Max-Triangle

```
#include "Point.cpp"

double max_triangle(vector<Point>& points){
    vector<Point> p = ConvexHull(points); // 最大三角形
        點一定在凸包上
    int n = p.size();
    p.push_back(p[0]);
    double ans = 0;
    for(int i=0; i<n; ++i)
    {
        int j = (i+1)%n;
        int k = (j+1)%n;
        //當Area(P[i], p[j], p[k+1]) <= Area(p[i], p[j
            ], p[k]) 時停止旋轉
        //即Cross(p[j]-p[i], p[k+1]-p[i]) - Cross(p[j]-
            p[i], p[k]-p[i]) <= 0
        //根據Cross(A,B) - Cross(A,C) = Cross(A,B-C)
        //化簡得Cross(p[j]-p[i], p[k+1] - p[k]) <= 0
        while(k!=i && Cross(p[j]-p[i], p[k+1]-p[k]) >
            0)
            k = (k+1) % n;
        if(k==i) continue;
        int kk = (k+1) % n;
        while(j!=kk && k!=i)
        {
            ans = max(ans, Cross(p[j]-p[i], p[k]-p[i]))
                ;
            while(k!=i && Cross(p[j]-p[i], p[k+1]-p[k])
                 > 0)
                k = (k+1) % n;
            j = (j+1) % n;
        }
    }
    return ans*0.5;
}
```

7 Sort

7.1 Heap Sort

```
void heap_sort(int* arr, int len)
{
    heapify(arr, len/2-1, len);
    max_heap(arr, len);
}
void heapify(int* ptr, int now, int last)
{
    if(now >= last/2 || now < 0)  return;
    sub_heapify(ptr, now, last);
    heapify(ptr, now-1, last);
}
void sub_heapify(int* ptr, int now, int last)
{
    if(now*2+2 < last && !(ptr[now] >= ptr[now*2+1] &&
        ptr[now] >= ptr[now*2+2])) {
        int max = (ptr[now*2+1] > ptr[now*2+2]) ? now
            *2+1 : now*2+2;
        swap(ptr, now, max, 1);
        if(max < last/2)  sub_heapify(ptr, max, last);
    }
    else if(now*2+1 < last && ptr[now] < ptr[now*2+1]){
        swap(ptr, now, now*2+1, 1);
        if(now*2+1 < last/2)sub_heapify(ptr, now*2+1,
            last);
    }
}
void max_heap(int* ptr, int len)
{
    if(len <= 1) return;
    swap(ptr, 0, len-1, 2);
    sub_heapify(ptr, 0, len-1);
    max_heap(ptr, len-1);
}
```

7.2 Merge Sort

```
void Merge(int* N,int L,int M){
    int tmp[L],p=0; int a,b;
    for(a=0,b=M;a<M && b<L;){
        if(N[a] < N[b]){ tmp[p++]=N[a]; a++; }
        else{ tmp[p++]=N[b]; b++;}
    }
    if(a == M)for(int i=b;i<L;i++)tmp[p++]=N[i];
    else for(int i=a;i<M;i++)tmp[p++]=N[i];
    for(int i=0;i<L;i++)N[i]=tmp[i];
}
void MergeSort(int* N,int L){
    int M=L/2;
    if(L == 1)return;
    MergeSort(N,M);
    MergeSort(N+M,L-M);
    Merge(N,L,M);
}
```

7.3 Radix Sort

```
int maxbit(int data[], int n) //輔助函数，求數据的最大
    位數
{
    int maxData = data[0];      ///< 最大數
    /// 先求出最大數，再求其位數，这样有原先依次每个數
        判斷其位數，稍微優化點。
    for (int i = 1; i < n; ++i) {
```

```
        if (maxData < data[i]) maxData = data[i];
    }
    int d = 1; int p = 10;
    while (maxData >= p){
        p *= 10;
        ++d;
    }
    return d;
    /*    int d = 1; //保存最大的位數
          int p = 10;
          for(int i = 0; i < n; ++i){
          while(data[i] >= p){
           p *= 10;
           ++d;
          }
          }
          return d;*/
}
void radixsort(int data[], int n) //基數排序
{
    int d = maxbit(data, n);
    int *tmp = new int[n];
    int *count = new int[10]; //計數器
    int i, j, k;
    int radix = 1;
    for(i = 1; i <= d; i++) { //進行d次排序
        for(j = 0; j < 10; j++) count[j] = 0; //每次分
            配前清空計數器
        for(j = 0; j < n; j++){
            k = (data[j] / radix) % 10; //統計每個桶中
                的記錄數
            count[k]++;
        }
        for(j = 1; j < 10; j++) count[j] = count[j - 1]
             + count[j]; //將tmp中的位置依次分配□每個
            桶
        for(j = n - 1; j >= 0; j--) { //將所有桶中記錄
            依次收集到tmp中
            k = (data[j] / radix) % 10;
            tmp[count[k] - 1] = data[j];
            count[k]--;
        }
        for(j = 0; j < n; j++) //將臨時數組的內容複製到
            data中
            data[j] = tmp[j];
        radix = radix * 10;
    }
    delete []tmp;
    delete []count;
}
```

7.4 Shell Sort

```
void shell_sort(int* ptr, int len)
{
    int gap = len / 2;
    while(gap){
        for(int i = gap; i < len; ++i, gap /= 2) {
            for(int j = i; j >= gap; j-=gap){
                if(ptr[j] > ptr[j-gap]) swap(ptr, j, j-
                    gap, gap);
                else break;
            }
        }
    }
}
```

8 Math

8.1 LIS

```
#include <bits/stdc++.h>
using namespace std;
template <typename E>
struct Node {
    E value;
    E* pointer;
};

template <class E>
struct node_ptr_less {
    bool operator()(E*& node1,
        E*& node2) const
    {
        return node1->value < node2->value;
    }
};
template <typename E>
std::vector<E> lis(const std::vector<E>& n)
{
    typedef E* NodePtr;

    std::vector<NodePtr> pileTops;
    // sort into piles
    for (typename std::vector<E>::const_iterator it = n
        .begin(); it != n.end(); it++) {
        NodePtr node(new Node<E>());
        node->value = *it;
        typename std::vector<NodePtr>::iterator j = std
            ::lower_bound(pileTops.begin(), pileTops.
            end(), node, node_ptr_less<E>());
        if (j != pileTops.begin())
            node->pointer = *(j - 1);
        if (j != pileTops.end())
            *j = node;
        else
            pileTops.push_back(node);
    }
    // extract LIS from piles
    std::vector<E> result;
    for (NodePtr node = pileTops.back(); node !=
        nullptr; node = node->pointer)
        result.push_back(node->value);
    std::reverse(result.begin(), result.end());
    return result;
}
int LIS(vector<int>& v)
{
    vector<int> ans;
    for (auto& i : v) {
        auto it = lower_bound(ans.begin(), ans.end(), i
            );
        if (ans.size() == 0 || i >= ans.back())
            ans.push_back(i);
        else {
            *it = i;
        }
    }
    return ans.size();
}
```

8.2 Extended Euclidean

```
int ExGCD(int A,int B,int& X,int& Y,int s0 = 1,int s1 =
     0,int t0 = 0,int t1 = 1){
    if(A%B == 0){
        X = s1;
        Y = t1;
        return B;
    }
    s0-=s1*(A/B);
    t0-=t1*(A/B);
    return ExGCD(B,A%B,X,Y,s1,s0,t1,t0);
}
```

8.3 Prime

```
// Complexity : O(NlogN)
void BuildPrime(bool prime[],int N){
    for(int i=2;i<N;i++) prime[i] = true;
    for(int i=2;i<N;i++){
        if(prime[i]) for(int j=i*i;j<N;j+=i)prime[j] =
            false;
    }
```

```
}

// Complexity : O(N)
void BuildPrime(vector<int> primelist,bool prime[],int
    N){
    for(int m=2;m<N;m++){
        if(prime[m] == true) primelist.push_back(m);
        for(auto i:primelist){
            if(m*i >= N) break;
            prime[m*i] = false;
            if(m%i == 0) break;
        }
    }
}

void ExBuildPrime(int first[],bool prime[],int N){
    for(int i=2;i<N;i++){
        prime[i] = true;
        first[i] = 1;
    }
    for(int i=2;i<N;i++){
        if(prime[i]) for(int j=i*i;j<N;j+=i){
            prime[j] = false;
            if(first[j] == 1) first[j] = i;
        }
    }
}
```

8.4 Factor Decomposition

```
vector<pair<int,int>> FactorDecomposition(int x){
    vector<pair<int,int>> ans;
    while(x > 1){
        int p,e = 0;
        if(prime[x] == true)p = x;else p = first[x];
        while(x%p == 0){x/=p;e++;}
        ans.push_back(make_pair(p,e));
    }
    return ans;
}
```

8.5 Module Inverse

```
int inverse(int A,int M,int X = 1,int Y = 0){
  if(A%M == 0){
    if(Y < 0)Y+=M;
    return Y;
  }
  X-=Y*(A/M);
  return inverse(M,A%M,Y,X);
}


inline int inverse(int A,int M){
    return ExPower(A,M-2,M);
}
```

8.6 Phi

```
int Phi(int x){
    vector<pair<int,int>> FD = FactorDecomposition(x);
    int ans = 1;
    for(auto i:FD){
        ans *= i.first-1;
        ans *= Power(i.first,i.second-1);
    }
    return ans;
}

void BuildPhi(int phi[],int N){
    for(int i=1;i<=N;i++) phi[i] = i;
    for(int i=1;i<=N;i++) for(x=i*2;x<=N;x+=i) phi[x]
        -= phi[i];
}

void BuildPhi(int phi[],int N){
    bool prime[N+1];for(int i=2;i<=N;i++) prime[i] =
        true;
    vector<int> primelist;
    phi[1] = 1;
    for(int m=2;m<=N;m++){
        if(prime[m] == true){
            phi[m] = m-1;
            primelist.push_back(m);
        }
        for(auto i:primelist){
            if(m*i > N) break;
            prime[m*i] = false;
            if(m%i == 0){
                int now = m,power = 1;
                while(now%i == 0){ now /= i;power *= i;
                    }
                phi[m*i] = phi[now]*power*(i-1);
                break;
            }
            else phi[m*i] = phi[m]*(i-1);
        }
    }
}
```

8.7 Miller Rabin

```
int ExMultiply(int a,int b,int n){
    a %= n;b %= n;
    int r = 0;
    while(b){
        if(b&1)r = ((a+r >= n)? a+r-n : a+r);
        a = ((a+a >= n)? a+a-n : a+a);
        b >>= 1;
    }
    return r;
}

int ExExPower(int a,int d,int n){
    if(d == 0)return 1;
    int k = ExExPower(a,d/2,n);
    if(d%2)return ExMultiply(ExMultiply(k,k,n),a,n);
    return ExMultiply(k,k,n);
}

bool MillerRabin(int n,int a){
    if(__gcd(n,a) == n)return true;
    if(__gcd(n,a) != 1)return false;
    // a^(d*2^r)
    int d = n-1, r = 0;
    while(d%2 == 0){ d /= 2;r++; }
    // a^d = ? (mod n)
    int remain = ExExPower(a,d,n);
    if(remain == 1 || remain == n-1)return true;
    while(r--){
        remain = ExMultiply(remain,remain,n);
        if(remain == n-1)return true;
    }
    return false;
}

bool IsPrime(int n){
    int a[7] =
        {2,325,9375,28178,450775,9780504,1795265022};
    for(int i=0;i<7;i++)if(!MillerRabin(n,a[i]))return
        false;
    return true;
}
```

8.8 Pollard-Rho

```
#include <bits/stdc++.h>
using namespace std;
//don't use __gcd for negative number
int gcd(int a, int b)
{
    if (a < 0)
        return gcd(-a, b);
    return b ? gcd(b, a % b) : a;
```

```
}
//super fast
int ExMultiply(int a, int b, int n)
{
    if (a == 0)
        return 0;
    return ((a & 1) * b % n + (ExMultiply(a >> 1, b, n)
         << 1) % n) % n;
}
int FastPow(int a, int b, int n)
{
    a %= n;
    int ans = 1;
    int d = a;
    while (b) {
        if (b & 1)
            ans = ExMultiply(ans, d, n);
        d = ExMultiply(d, d, n);
        b >>= 1;
    }
    return ans;
}

bool MillerRabin(int n, int a)
{
    if (n == a)
        return true;
    //even
    if (gcd(n, a) == n)
        return true;
    if (gcd(n, a) != 1)
        return false;
    // a^(d*2^r)
    int d = n - 1, r = 0;
    while (!(d & 1)) {
        d >>= 1;
        r++;
    }
    // a^d = ? (mod n)
    int remain = FastPow(a, d, n);
    if (remain == 1 || remain == n - 1)
        return true;
    while (r--) {
        remain = ExMultiply(remain, remain, n);
        if (remain == n - 1)
            return true;
    }
    return false;
}
bool IsPrime(int n)
{
    if (n == 2)
        return true;
    if (!(n & 1))
        return false;
    int a[7] = { 2, 325, 9375, 28178, 450775, 9780504,
        1795265022 };
    for (int i = 0; i < 7; i++) {
        if (!MillerRabin(n, a[i]))
            return false;
    }
    return true;
}
int PollardRho(int n, int c)
{
    int x = rand() % n, y = x, k = 2;
    for (int i = 2;; i++) {
        x = (ExMultiply(x, x, n) + c) % n;
        int d = gcd(x - y, n);
        if (d != 1 && d != n)
            return d;
        if (y == x)
            return n;
        if (i == k) {
            y = x;
            k <<= 1;
        }
    }
}
vector<int> Fac;
void fac(int n)
{
    if (IsPrime(n)) {
        Fac.push_back(n);
        return;
    }
    int p = n;
    while (p >= n)
        p = PollardRho(p, rand() % (n - 1) + 1);
    fac(p);
    fac(n / p);
}
```

8.9 Chinese-Remainder-Theorem

```
#include <bits/stdc++.h>
using namespace std;
int inverse(int A, int M)
{
    return A == 1 ? 1 : inverse(M % A, M) * (M - M / A)
         % M;
}

/*
 * chinese remainder theorem
 * check all m[i] are pairwise coprime
 * if x = a1(mod p) and x=a2(mod p) if a1!=a2
 * then no solution
 * return first positive answer
 * next answer is answer+M
 */
int CRT(vector<int> a, vector<int> m)
{
    if (a.size() != m.size())
        return -1;
    int M = 1;
    for (int i = 0; i < m.size(); i++) {
        M *= m[i];
    }
    int res = 0;
    for (int i = 0; i < a.size(); i++) {
        res = (res + a[i] * (M / m[i]) * inverse(M / m[
            i], m[i])) % M;
    }
    return (res+M)%M;
}
```

8.10 Lucas-Theorem

```
#include <bits/stdc++.h>
using namespace std;
const int p = 5; //prime<10^5
int fac[p + 1];
void build_fac(int p)
{
    fac[0] = 1;
    for (int i = 1; i <= p; i++)
        fac[i] = fac[i - 1] * i % p;
}
//mod inverse
int inv(int a, int p)
{
}
//called after build_fac
int Lucas(int n, int m, int p)
{
    if (m == 0)
        return 1;
    if (m > n)
        return 0;
    if (n < m)
        return fac[n] * inv(fac[m] * fac[n - m] % p, p)
             % p;
    else
        return Lucas(n / p, m / p, p) * Lucas(n % p, m
            % p, p) % p;
}
```

8.11 FFT

```
const int MAXN = 262144;

const double PI = acos(-1.0);
const complex<double> I(0, 1);
complex<double> omega[MAXN+1];

void pre_FFT(){
    for(int i=0; i<=MAXN; i++)omega[i] = exp(i * 2 * PI
         / MAXN * I);
}

void FFT(int n, complex<double> a[], bool inv=false){
    int basic = MAXN / n;
    int theta = basic;
    for(int m = n; m >= 2; m >>= 1) {
        int mh = m >> 1;
        for(int i = 0; i < mh; i++) {
            complex<double> w = omega[inv ? MAXN-(i*
                theta%MAXN) : i*theta%MAXN];
            for(int j = i; j < n; j += m) {
                int k = j + mh;
                complex<double> x = a[j] - a[k];
                a[j] += a[k];
                a[k] = w * x;
            }
        }
        theta = (theta * 2) % MAXN;
    }
    int i = 0;
    for(int j = 1; j < n - 1; j++) {
        for(int k = n >> 1; k > (i ^= k); k >>= 1);
        if(j < i) swap(a[i], a[j]);
    }
    if(inv)for(i = 0; i < n; i++)a[i] /= n;
}
```

8.12 Fraction

```
struct fraction_positive{
    int p,q;
    fraction_positive(){}
    fraction_positive(int p,int q):p(p),q(q){}
    void reduction(){
        int G = __gcd(p,q);
        p /= G;
        q /= G;
    }
    bool operator==(const fraction_positive& B) const {
        return (p == B.p && q == B.q);
    }
    bool operator!=(const fraction_positive& B) const {
        return (p != B.p || q != B.q);
    }
    bool operator>(const fraction_positive& B) const {
        return (p*B.q > B.p*q);
    }
    bool operator>=(const fraction_positive& B) const {
        return (p*B.q >= B.p*q);
    }
    bool operator<(const fraction_positive& B) const {
        return (p*B.q < B.p*q);
    }
    bool operator<=(const fraction_positive& B) const {
        return (p*B.q <= B.p*q);
    }
    fraction_positive operator+(const fraction_positive
        & B) const {
        fraction_positive F;
        F.p = p*B.q+B.p*q;
        F.q = q*B.q;
        F.reduction();
        return F;
    }
    fraction_positive operator-(const fraction_positive
        & B) const {
        fraction_positive F;
        F.p = p*B.q-B.p*q;
        F.q = q*B.q;
        F.reduction();
        return F;
    }
    fraction_positive operator*(const fraction_positive
        & B) const {
        fraction_positive F;
        F.p = p*B.p;
        F.q = q*B.q;
        F.reduction();
        return F;
    }
    fraction_positive operator/(const fraction_positive
        & B) const {
        fraction_positive F;
        F.p = p*B.q;
        F.q = q*B.p;
        F.reduction();
        return F;
    }
    fraction_positive operator*(int x) const {
        fraction_positive F = *this;
        F.p *= x;
        F.reduction();
        return F;
    }
    fraction_positive operator/(int x) const {
        fraction_positive F = *this;
        F.q *= x;
        F.reduction();
        return F;
    }
};

struct fraction{
    fraction_positive N;
    bool sign,broken;//0 positive 1 negative
    fraction():broken(false){}
    fraction(int p,int q,bool sign):sign(sign){
        if(q == 0){broken = true;cout << "===divide by
            zero===" << endl;}
        else{N.p = p;N.q = q;N.reduction();}
    }
    bool operator==(const fraction& B) const {
        return (N == B.N && sign == B.sign);
    }
    bool operator!=(const fraction& B) const {
        return (N != B.N || sign != B.sign);
    }
    bool operator>(const fraction& B) const {
        return (!sign && B.sign) || (!sign && N > B.N)
            || (sign && N < B.N);
    }
    bool operator>=(const fraction& B) const {
        return  (!sign && B.sign) || (!sign && N >= B.N
            ) || (sign && N <= B.N);
    }
    bool operator<(const fraction& B) const {
        return !(*this >= B);
    }
    bool operator<=(const fraction& B) const {
        return !(*this > B);
    }
    fraction operator+(const fraction& B) const {
        fraction F;
        if(broken || B.broken){F.broken = true;return F
            ;}
        if(sign^B.sign){
            const fraction_positive& big = (N > B.N ? N
                : B.N);
            const fraction_positive& small = (N <= B.N
                ? N : B.N);
            F.N = big - small;
            F.sign = (N > B.N ? sign : B.sign);
        }
        else{
            F.N = N+B.N;
            F.sign = sign;
        }
        return F;
    }
    fraction operator-(const fraction& B) const {
```

```
        fraction F = B;
        if(broken || B.broken){F.broken = true;return F
            ;}
        F.sign = !F.sign;
        return (*this+F);
    }
    fraction operator*(const fraction& B) const {
        fraction F;
        if(broken || B.broken){F.broken = true;return F
            ;}
        F.N = N*B.N;
        F.sign = sign^B.sign;
        return F;
    }
    fraction operator/(const fraction& B) const {
        fraction F;
        if(broken || B.broken || B.N.p == 0){F.broken =
             true;return F;}
        F.N = N/B.N;
        F.sign = sign^B.sign;
        return F;
    }
    fraction operator*(int x) const {
        fraction F = *this;
        if(broken){F.broken = true;return F;}
        F.N = F.N*abs(x);
        if(x < 0)F.sign = !F.sign;
        return F;
    }
    fraction operator/(int x) const {
        fraction F = *this;
        if(x == 0){F.broken = true;return F;}
        F.N = F.N/abs(x);
        if(x < 0)F.sign = !F.sign;
        return F;
    }
    friend istream& operator>>(istream& in,fraction& B)
        {
        int x;
        char c;
        B.sign = false;
        in >> x;if(x < 0){B.sign = true;x = -x;}
        B.N.p = x;
        in >> c >> x;if(x == 0){B.broken = true;return
            in;}
        B.N.q = x;
        B.N.reduction();
        return in;
    }
    friend ostream& operator<<(ostream& out,const
        fraction& B){
        if(B.broken){return out << "NaN";}
        if(B.sign)out << '-';
        return out << B.N.p << '/' << B.N.q;
    }
};
```

8.13 Matrix

```
#include <bits/stdc++.h>
using namespace std;
const double EPS = 1e-9;

template <typename T>
class Matrix {
public:
    Matrix()
        : wrong(false)
    {
    }
    Matrix(int _rows, int _cols)
        : wrong(false)
    {
        rows = _rows;
        cols = _cols;
        data.resize(_rows);
        for (int i = 0; i < _rows; i++)
            data[i].resize(_cols);
    }
    Matrix(T** _data, int _rows, int _cols)
        : wrong(false)
    {
        rows = _rows;
        cols = _cols;
        data.resize(_rows);
        for (int i = 0; i < _rows; i++)
            data[i].resize(_cols);
        for (int i = 0; i < rows; i++)
            for (int j = 0; j < cols; j++)
                data[i][j] = _data[i][j];
    }
    Matrix(const Matrix& N)
    {
        wrong = N.wrong;
        rows = N.rows;
        cols = N.cols;
        data.resize(rows);
        for (int i = 0; i < rows; i++)
            data[i].resize(cols);
        for (int i = 0; i < rows; i++)
            for (int j = 0; j < cols; j++)
                data[i][j] = N.data[i][j];
    }
    T& at(int a, int b)
    {
        return data[a][b];
    }
    Matrix operator+(const Matrix& N)
    {
        cout << (*this) << endl
             << N << endl;
        Matrix tmp = Matrix(*this);
        if (rows != N.rows || cols != N.cols)
            tmp.wrong = true;
        else
            for (int i = 0; i < rows; i++)
                for (int j = 0; j < cols; j++)
                    tmp.data[i][j] += N.data[i][j];
        return tmp;
    }
    Matrix operator-(const Matrix& N)
    {
        Matrix tmp = Matrix(*this);
        if (rows != N.rows || cols != N.cols)
            tmp.wrong = true;
        else
            for (int i = 0; i < rows; i++)
                for (int j = 0; j < cols; j++)
                    tmp.data[i][j] -= N.data[i][j];
        return tmp;
    }
    Matrix operator*(const Matrix& N)
    {
        Matrix tmp = Matrix(rows, N.cols);
        if (cols != N.rows)
            tmp.wrong = true;
        else
            for (int i = 0; i < tmp.rows; i++)
                for (int j = 0; j < tmp.cols; j++) {
                    tmp.data[i][j] = 0;
                    for (int k = 0; k < cols; k++)
                        tmp.data[i][j] += data[i][k] *
                            N.data[k][j];
                }
        return tmp;
    }
    Matrix operator*(int c)
    {
        Matrix tmp = Matrix(*this);
        for (int i = 0; i < rows; i++)
            for (int j = 0; j < cols; j++)
                tmp.data[i][j] *= c;
        return tmp;
    }
    Matrix operator=(const Matrix& N)
    {
        wrong = N.wrong;
        rows = N.rows;
        cols = N.cols;
        data = new T*[rows];
        for (int i = 0; i < rows; i++)
            data[i] = new T[cols];
```

```
        for (int i = 0; i < rows; i++)
            for (int j = 0; j < cols; j++)
                data[i][j] = N.data[i][j];
        return (*this);
    }
    Matrix transpose(void)
    {
        Matrix tmp = Matrix(*this);
        //int fuck = tmp.rows; tmp.rows = tmp.cols;tmp.
            cols = fuck;
        swap(tmp.rows, tmp.cols);
        tmp.data = new T*[tmp.rows];
        for (int i = 0; i < tmp.rows; i++)
            tmp.data[i] = new T[tmp.cols];
        for (int i = 0; i < rows; i++)
            for (int j = 0; j < cols; j++)
                tmp.data[j][i] = data[i][j];
        return tmp;
    }
    void Identity()
    { // rows==cols
        for (int i = 0; i < rows; i++) {
            at(i, i) = 1;
        }
    }
    Matrix pow(int rhs) const
    {
        if (rows != cols)
            return Matrix();
        Matrix res(rows, rows), p(*this);
        res.Identity();
        while (rhs) {
            if (rhs & 1)
                res = res * p;
            p = p * p;
            rhs >>= 1;
        }
        return res;
    }
    T det()
    {
        int ans = 1;
        for (int i = 0; i < rows; i++) {
            for (int j = i + 1; j < rows; j++) {
                int a = i, b = j;
                while (at(b, i)) {
                    int q = at(a, i) / at(b, i);
                    for (int k = 0; k < rows; k++) {
                        at(a, k) = at(a, k) - at(b, k)
                            * q;
                    }
                    swap(a, b);
                }
                if (a != i) {
                    swap(data[i], data[j]);
                    ans = -ans;
                }
            }
            if (fabs(at(i, i)) < EPS)
                return 0;
            else
                ans *= at(i, i);
        }
        return ans;
    }
    // r:non-free number l:l[i] is true if i-th
        variable is non-free
    Matrix GuassElimination(int& r, vector<bool>& l,
        int flag = 0)
    {
        l = vector<bool>(cols);
        r = 0;
        Matrix res(*this);
        for (int i = 0; i < res.cols - flag; i++) {
            for (int j = r; j < res.rows; j++) {
                if (fabs(res.at(j, i)) > EPS) {
                    swap(res.data[r], res.data[j]);
                    break;
                }
            }
            if (fabs(res.at(r, i)) < EPS) {
                continue;
            }
            for (int j = 0; j < res.rows; j++) {
                if (j != r && fabs(res.at(j, i)) > EPS)
                    {
                    double tmp = (double)res.at(j, i) /
                        (double)res.at(r, i);
                    for (int k = 0; k < res.cols; k++)
                        {
                        res.at(j, k) -= tmp * res.at(r,
                            k);
                    }
                }
            }
            r++;
            l[i] = true;
        }
        return res;
    }
    vector<double> Solve(vector<double> a)
    {
        if (rows != cols)
            return vector<double>();
        vector<double> res(rows);
        Matrix t(rows, cols + 1);
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++)
                t.at(i, j) = at(i, j);
            t.at(i, rows) = a[i];
        }
        int r = 0;
        vector<bool> l;
        t = t.GuassElimination(r, l, 1);
        if (r != rows)
            return vector<double>();
        for (int i = 0; i < cols; i++) {
            if (l[i])
                for (int j = 0; j < rows; j++) {
                    if (fabs(t.at(j, i)) > EPS)
                        res[i] = t.at(j, cols) / t.at(j
                            , i);
                }
        }
        return res;
    }
    Matrix Inverse()
    {
        if (rows != cols)
            return Matrix();
        Matrix t(rows, rows * 2);
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++)
                t.at(i, j) = at(i, j);
            t.at(i, i + rows) = 1;
        }
        int r = 0;
        vector<bool> l;
        t = t.GuassElimination(r, l, rows);
        if (r != rows)
            return Matrix();
        for (int i = 0; i < cols; i++) {
            if (l[i])
                for (int j = 0; j < rows; j++) {
                    if (fabs(t.at(j, i)) > EPS) {
                        for (int k = 0; k < cols; k++)
                            t.at(j, cols + k) /= t.at(j
                                , i);
                    }
                }
        }
        Matrix res(rows, cols);
        for (int i = 0; i < rows; i++)
            for (int j = 0; j < cols; j++)
                res.at(i, j) = t.at(i, j + cols);
        return res;
    }

    vector<vector<T> > data;
    int rows, cols;
    bool wrong;
};
```