Contents

1 Basic

1.1 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
```

1.2 default

```
#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;
}
```

2 Data Structure

2.1 Disjoint Set

```
struct DisjointSet{
    int p[N];
    void init(int n){for(int i=1;i<=n;i++)p[i] = i;}
    int Find(int x){return x == p[x] ? x : p[x] = Find(
        p[x]); }
    void Union(int x,int y){p[Find(x)] = Find(y);}
};
```

2.2 Segment Tree

```
struct Node{
    int value;
    Node *lc,*rc;
    Node(){value = 0;lc = rc = NULL;}
    void pull(){
        value = lc->value+rc->value;
    }
};
int v[N];
Node* build(int L,int R){
    Node *node = new Node();
    if(L == R){
        node->value = v[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 i,int d){
    if(L == R){
        node->value += d;
        return;
    }
    int mid = (L+R)>>1;
    if(i<=mid)modify(node->lc,L,mid,i,d);
    else modify(node->rc,mid+1,R,i,d);
    node->pull();
}
int query(Node* node,int L,int R,int ql,int qr){
    if(ql > R || qr < L)return 0;
    if(ql <= L && R <= qr)return node->value;
    int mid = (L+R)>>1;
    return query(node->lc,L,mid,ql,qr)+query(node->rc,
        mid+1,R,ql,qr);
}
```

2.3 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 Graph

3.1 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□
         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);
}
```

3.2 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[i].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;
}
```

3.3 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
}
```

3.4 Prim

```
const int MAXN=110;bool vis[MAXN];int lowc[MAXN];
int Prim(int cost[][MAXN],int n) //0-based
{
    int ans=0;memset(vis,0,sizeof(vis));vis[0]=false;
    for(int i=1;i<n;i++)lowc[i]=cost[0][i];
    for(int i=1; i<n;i++){
        int minc=INT_MAX;
        int p=-1;
        for(int j=0;j<n;j++){
            if(!vis[j] && minc>lowc[j]){
                minc=lowc[j];
                p=j;
            }
        }
        if(minc==INT_MAX) return -1; //failed
        ans+=minc;
        vis[p]=true;
        for(int j=0;j<n;j++)
            if(!vis[j] && lowc[j]>cost[p][j])
                lowc[j]=cost[p][j];
    }
    return ans;
}
```

3.5 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;
}
```

3.6 Bipartite Match

```
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;
}
```

3.7 LCA

```
#include <bits/stdc++.h>
using namespace std;
//Tarjan
const int MAXN = 1000; //max_number of nodes
int n;
int root;
int indeg[MAXN]; //入度
vector<int> tree[MAXN];
void inputTree()
{
    cin >> n;
    for (int i = 0; i < n; i++)
        tree[i].clear(), indeg[i] = 0;
    for (int i = 0; i < n - 1; i++) {
        int x, y;
        cin >> x >> y;
        tree[x].push_back(y);
        indeg[y]++;
    }
    for (int i = 0; i < n; i++)
        if (indeg[i] == 0) {
            root = i;
            break;
        }
}
vector<int> query[MAXN];
void inputQuery()
{
    for (int i = 0; i < n; i++) {
        query[i].clear();
    }
    int m; //number of query
    cin >> m;
    while (m--) {
        int u, v;
        cin >> u >> v;
        query[u].push_back(v);
        query[v].push_back(u);
    }
}
int father[MAXN], depth[MAXN];
void makeSet()
{
    for (int i = 0; i < n; i++) {
        father[i] = i, depth[i] = 0;
    }
}
int findSet(int x)
{
    return (x == father[x]) ? x : father[x] = findSet(
        father[x]);
}
void unionSet(int x, int y)
{
    x = findSet(x), y = findSet(y);
    if (x == y)
        return;
    if (depth[x] > depth[y]) {
        father[y] = x;
        depth[x] += depth[y];
        depth[y] = 0;
    } else {
        father[x] = y;
        depth[y] += depth[x];
        depth[x] = 0;
    }
}
int ancestor[MAXN];
int lca[MAXN][MAXN];
bool visited[MAXN];
void Tarjan(int u)
{
    for(int i=0;i<tree[u].size();i++)
    {
        int v=tree[u][i];
        Tarjan(v);
        unionSet(u,v);
        ancestor[findSet(u)]=u;
    }
    visited[u]=true;
    for(int i=0;i<query[u].size();i++)
    {
        int v=query[u][i];
        if(visited[v])
        {
            lca[u][v]=lca[u][v]=ancestor[findSet(v)];
        }
    }
}
```

3.8 LCA2

```
#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];
}
```

3.9 manhattan-mst

```
#include <bits/stdc++.h>
const int N = 10000;
struct Point {
    int x, y, 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 T[N], hs[N];
int m, mt;
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 k)
{
    mt = 0;
    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;
        }
    }
    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;)
    m=unique(hs,hs+n)-hs;
    for(int i=1;i<=m;i++) bit[i].init();
    for(int i=n-1;i>=0;i--){
        pos=lower_bound(hs,hs+m,T[i])-hs+1; //Bit is 1-based
        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<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 Stringology

4.1 KMP

```
int* predo(string pattern){
    int* dp = new int[pattern.size()];
    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]++;
    }
    return dp;
}

void KMP(string text,string pattern){
    int* dp = predo(pattern);
    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()){
            cout << i-pattern.size()+1 << endl;
            match = dp[match-1];
        }
    }
    delete [] dp;
}
```

4.2 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() << " ";
    }
}
```

4.3 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;
}
```

4.4 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;
    }
};
```

4.5 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;
    }
}
```

5 Geometry

5.1 Point

```
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);
    }
    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 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 0~2
    pi //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🄵
    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 Onsegment(Point p, Point a1, Point a2) //點在🄵上
{
    //平行                      //端點在兩側
    return dcmp(cross(a1 - p, a2 - p)) == 0 && dcmp(dot
        (a1 - p, a2 - p)) < 0;
}
bool SegmentProperIntersection(Point a1, Point a2,
    Point b1, Point b2)
{
    // 規範相交 :交點不能是🄵段的交點
    double c1 = cross(a2 - a1, b1 - a1), c2 = cross(a2
        - a1, b2 - a1);
    double c3 = cross(b2 - b1, a1 - b1), c4 = cross(b2
        - b1, a2 - b1);
    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 SegmentInterSection(Point a1, Point a2, Point b1,
    Point b2) //非規範相交
{
    //端點相交
    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; //規範相交
    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); //🄵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🄵上
    if (dcmp(dot(v2, v)) > 0)
        return length(v2); // 點投影不在🄵上
    return fabs(cross(v, v1)) / length(v);
}
double distance(Segment& s1, Segment& s2) //🄵段到🄵段
{
    if (SegmentInterSection(s1, s2))
        return 0;
    double d = 1e9;
    d = min(d, distance(s1.p1, s2)); //點到🄵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) //🄵段到🄵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🄵段距離
}
int ConvexHull(vector<Point>& P, Point* res)
{ //凸包Andrew's Monotone Chain
    sort(P.begin(), P.end()); //先x 後 y
    auto last = unique(P.begin(), P.end()); //非重🄵的
        點數量
    P.erase(last, P.end());
    int cnt = P.size();
    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) //頭尾 1個點不用--
        m--;
    return m; //凸包點數
}
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 Sort

6.1 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);
}
```

6.2 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);
}
```

6.3 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
            配前清空计數器
        for(j = 0; j < n; j++){
            k = (data[j] / radix) % 10; //統计每個桶中
                的記🄵數
            count[k]++;
        }
        for(j = 1; j < 10; j++) count[j] = count[j - 1] + count[j]; //將tmp中的位置依次分配给每個桶
        for(j = n - 1; j >= 0; j--) { //將所有桶中記🄵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++) //將臨時數组的内容🄵🄵到data中
            data[j] = tmp[j];
        radix = radix * 10;
    }
    delete []tmp;
    delete []count;
}
```

6.4 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;
            }
        }
    }
}
```

7 Math

7.1 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);
}
```

7.2 Prime

```
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;}
  }
}

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;
    }}
  }
}
```

7.3 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;
}
```

7.4 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);
}

int inverse(int A,int M){
    return A == 1?1:inverse(M%A)*(M-M/A)%M;
}

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

7.5 Miller Rabin

```
bool MillerRabin(int a,int n){
    if(a == n)return true;
    if(__gcd(a,n) != 1)return false;
    int s = 0,d = n-1;
    int power_of_a;
    while(d%2 == 0){d/=2;s++;}
    power_of_a=ExPower(a,d,n);
    if(power_of_a == 1)return true;
    for(int i=0;i<s;i++){
        if(power_of_a == n-1)return true;
        power_of_a=power_of_a*power_of_a%n;
    }
    return false;
}

bool PrimeMillerRabin(int n){
    int a_set[3]={2,7,61};
    //LL a_set[7]={2,325,9375,28178,450775,9780504,1795265022};

    if(n == 2 || n == 3)return true;
    if(n <= 1 || n%2 == 0 || n%3 == 0)return false;
    for(int i=0;i<3;i++){
        if(!MillerRabin(a_set[i],n))return false;
    }
    return true;
}
```

7.6 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;
    }
};
```

7.7 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 = new T*[rows];
        for (int i = 0; i < rows; i++)
            data[i] = new T[cols];
    }
    Matrix(T** _data, int _rows, int _cols)
        : wrong(false)
    {
        rows = _rows;
        cols = _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] = _data[i][j];
    }
    Matrix(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];
    }
    ~Matrix()
    {
        delete data;
    }
    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);
        delete tmp.data;
        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;
    }
    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;
    }
    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;
    }


    T** data;
    int rows, cols;
    bool wrong;
};

template <typename T>
ostream& operator<<(ostream& o, const Matrix<T>& N)
{
    if (N.wrong) {
        o << "Error: Wrong Matrix Dimension" << endl;
        return o;
    }
    for (int i = 0; i < N.rows; i++)
        for (int j = 0; j < N.cols; j++) {
            if (j == 0) {
                if (i == 0)
                    o << '[';
                else
                    o << ' ';
            }
            o << N.data[i][j];
            if (j == N.cols - 1) {
                if (i == N.rows - 1)
                    o << ']';
                else
                    o << ';' << endl;
            } else
                o << ' ';
        }
    return o;
}

template <typename T>
T det(Matrix<T> N)
{
    if (N.cols == 2)
        return N.data[0][0] * N.data[1][1] - N.data
            [0][1] * N.data[1][0];
    T sum = 0;
    for (int i = 0; i < N.cols; i++) {
        Matrix<T> tmp(N.cols - 1, N.cols - 1);
        for (int j = 0; j < N.cols - 1; j++)
            for (int k = 0; k < N.cols - 1; k++) {
                int r = j + 1, c;
                if (k < i)
                    c = k;
                else
                    c = k + 1;
                tmp.data[j][k] = N.data[r][c];
            }
        int Ans;
        if (i % 2)
            Ans = -1;
        else
            Ans = 1;
        sum += Ans * N.data[0][i] * det(tmp);
    }
    return sum;
}
```

7.8 BigInt

```
const int MAX_DIGIT = 1000;
const int POSTIONAL = 4;
const int POSTIONAL_NOTATION = 10000;

struct PositiveBigInt{
    int N[MAX_DIGIT],L;
    PositiveBigInt():L(0){}
    PositiveBigInt(string S){
        set_value(S);
    }
    PositiveBigInt(int* N,int L){
        set_value(N,L);
    }
    void set_value(string S){
        L=(S.size()-1)/POSTIONAL+1;
        for(int i=0;i<L;i++)N[i]=0;
        for(int i=0;i*POSTIONAL<S.size();i++){
            int pow10=1;
            for(int j=i*POSTIONAL;j<S.size() && j<i*
                POSTIONAL+POSTIONAL;j++){
                N[i]+=(S[S.size()-1-j]-48)*pow10;
                pow10*=10;
            }
        }
    }
    void set_value(int* N,int L){
        this->L=L;
        for(int i=0;i<L;i++)this->N[i]=N[i];
    }
    bool equal_zero() const {
        if(L == 1 && N[0] == 0)return true;
        return false;
    }
    void kill_zero(){
        while(L > 1 && N[L-1] == 0)L--;
    }
    int magic(int *Num,int Length,const PositiveBigInt&
         A) const {
        PositiveBigInt B(Num,Length);
        B.kill_zero();
        int Ans=0;
        while(B >= A){
            B=B-A;
            Ans++;
        }
        for(int i=0;i<Length;i++){
            if(i < B.L)Num[i]=B.N[i];
            else Num[i]=0;
        }
        return Ans;
    }
    PositiveBigInt operator+(const PositiveBigInt& A)
        const {
        const PositiveBigInt &X=(*this > A ? *this : A)
            ;
        const PositiveBigInt &Y=(*this <= A ? *this : A
            );
        PositiveBigInt tmp=X;
        for(int i=0;i<Y.L;i++)tmp.N[i]+=Y.N[i];
        tmp.N[tmp.L]=0;
        for(int i=0;i<tmp.L;i++){
            tmp.N[i+1]+=tmp.N[i]/POSTIONAL_NOTATION;
            tmp.N[i]%=POSTIONAL_NOTATION;
        }
        if(tmp.N[tmp.L] > 0)tmp.L++;
        return tmp;
    }
    PositiveBigInt operator-(const PositiveBigInt& A)
        const {
        const PositiveBigInt &X=(*this > A ? *this : A)
            ;
        const PositiveBigInt &Y=(*this <= A ? *this : A
            );
        PositiveBigInt tmp=X;
        for(int i=0;i<Y.L;i++)tmp.N[i]-=Y.N[i];
        for(int i=0;i<tmp.L;i++){
            if(tmp.N[i] < 0){
                tmp.N[i+1]--;
                tmp.N[i]+=POSTIONAL_NOTATION;
            }
        }
        tmp.kill_zero();
        return tmp;
    }
    PositiveBigInt operator*(const PositiveBigInt& A)
        const {
        PositiveBigInt tmp;
        tmp.L=L+A.L;
        for(int i=0;i<tmp.L;i++)tmp.N[i]=0;
        for(int i=0;i<L;i++){
            for(int j=0;j<A.L;j++)tmp.N[i+j]+=N[i]*A.N[
                j];
            for(int j=0;j<tmp.L;j++){
                tmp.N[j+1]+=tmp.N[j]/POSTIONAL_NOTATION
                    ;
                tmp.N[j]%=POSTIONAL_NOTATION;
            }
        }
        tmp.kill_zero();
        return tmp;
    }
    PositiveBigInt operator/(const PositiveBigInt& A)
        const {
        if(*this < A)return PositiveBigInt("0");
        PositiveBigInt Div,Mod=*this;
        Div.L=L;
        for(int i=L-1;i>=0;i--)Div.N[i]=magic(Mod.N+i,
            Mod.L-i,A);
        Div.kill_zero();
        return Div;
    }
    PositiveBigInt operator%(const PositiveBigInt& A)
        const {
        if(*this < A)return *this;
        PositiveBigInt Mod=*this;
        for(int i=L-1;i>=0;i--)magic(Mod.N+i,Mod.L-i,A)
            ;
        Mod.kill_zero();
        return Mod;
    }
    bool operator>(const PositiveBigInt& A) const {
        if(L > A.L)return true;
        else if(L < A.L)return false;
        for(int i=L-1;i>=0;i--){
            if(N[i] > A.N[i])return true;
            else if(N[i] < A.N[i])return false;
        }
        return false;
    }
    bool operator>=(const PositiveBigInt& A) const {
        if(L > A.L)return true;
        else if(L < A.L)return false;
        for(int i=L-1;i>=0;i--){
            if(N[i] > A.N[i])return true;
            else if(N[i] < A.N[i])return false;
        }
        return true;
    }
    bool operator<(const PositiveBigInt& A) const {
        return !(*this >= A);
    }
    bool operator<=(const PositiveBigInt& A) const {
        return !(*this > A);
    }
    bool operator==(const PositiveBigInt& A) const {
        if(L != A.L)return false;
        for(int i=0;i<L;i++){
            if(N[i] != A.N[i])return false;
        }
        return true;
    }
    bool operator!=(const PositiveBigInt& A) const {
        return !(*this == A);
    }
    PositiveBigInt operator=(const PositiveBigInt& A){
        L=A.L;
        for(int i=0;i<L;i++)N[i]=A.N[i];
        return (*this);
    }
};

ostream& operator<<(ostream& o,const PositiveBigInt& A)
    {
    o << A.N[A.L-1];
    for(int i=A.L-2;i>=0;i--){
        for(int c=1,tmp=A.N[i];c < POSTIONAL && tmp*10
            < POSTIONAL_NOTATION;c++,tmp*=10)o << "0";
        o << A.N[i];
    }
    return o;
}

struct BigInt{
    PositiveBigInt N;
    bool sign;
    BigInt(){}
    BigInt(string S){
        set_value(S);
    }
    void set_value(string S){
        if(S[0] == '-'){
            sign=false;
            N.set_value(S.substr(1,S.size()-1));
        }
        else{
            sign=true;
            N.set_value(S);
        }
    }
    BigInt operator+(const BigInt& A) const {
        BigInt tmp;
        if(sign^A.sign){
            tmp.N=N-A.N;
            if(N > A.N)tmp.sign=sign;
            else if(N < A.N)tmp.sign=A.sign;
            else tmp.sign=true;
        }
        else{
            tmp.N=N+A.N;
            tmp.sign=sign;
        }
        return tmp;
    }
    BigInt operator-(const BigInt& A) const {
        BigInt tmp=A;
        tmp.sign=!tmp.sign;
        return (*this + tmp);
    }
    BigInt operator*(const BigInt& A) const {
        BigInt tmp;
        if(N.equal_zero() || A.N.equal_zero())tmp.sign=
            true;
        else tmp.sign=!(sign^A.sign);
        tmp.N=N*A.N;
        return tmp;
    }
    BigInt operator/(const BigInt& A) const {
        if(A.N.equal_zero()){printf("divided by 0 error
            !!\n");return BigInt("0");}
        BigInt tmp;
        if(N.equal_zero())tmp.sign=true;
        else tmp.sign=!(sign^A.sign);
        tmp.N=N/A.N;
        return tmp;
    }
    BigInt operator%(const BigInt& A) const {
        if(A.N.equal_zero()){printf("divided by 0 error
            !!\n");return BigInt("0");}
        BigInt tmp;
        tmp.sign=true;
        tmp.N=N%A.N;
        return tmp;
    }
    bool operator>(const BigInt& A) const {
        if(sign == true && A.sign == true)return (N > A.N);
        if(sign == true && A.sign == false)return true;
        if(sign == false && A.sign == true)return false;
        if(sign == false && A.sign == false)return (N < A.N);
    }
    bool operator>=(const BigInt& A) const {
        if(sign == true && A.sign == true)return (N >= A.N);
        if(sign == true && A.sign == false)return true;
        if(sign == false && A.sign == true)return false;
        if(sign == false && A.sign == false)return (N <= A.N);
    }
    bool operator<(const BigInt& A) const {
        return !(*this >= A);
    }
    bool operator<=(const BigInt& A) const {
        return !(*this > A);
    }
    bool operator==(const BigInt& A) const {
        if(sign != A.sign)return false;
        return (N == A.N);
    }
    bool operator!=(const BigInt& A) const {
        return !(*this == A);
    }
    BigInt operator=(const BigInt &A){
        N=A.N;
        sign=A.sign;
        return (*this);
    }
    BigInt operator+=(const BigInt &A){
        return (*this)=(*this)+A;
    }
    BigInt operator-=(const BigInt &A){
        return (*this)=(*this)-A;
    }
    BigInt operator*=(const BigInt &A){
        return (*this)=(*this)*A;
    }
    BigInt operator/=(const BigInt &A){
        return (*this)=(*this)/A;
    }
    BigInt operator%=(const BigInt &A){
        return (*this)=(*this)%A;
    }
    BigInt operator++(){
        (*this)=(*this)+BigInt("1");
        return (*this);
    }
    BigInt operator++(int useless){
        BigInt tmp=(*this);
        (*this)=(*this)+BigInt("1");
        return tmp;
    }
    BigInt operator--(){
        (*this)=(*this)-BigInt("1");
        return (*this);
    }
    BigInt operator--(int useless){
        BigInt tmp=(*this);
        (*this)=(*this)-BigInt("1");
        return tmp;
    }
};

ostream& operator<<(ostream& o,const BigInt& A){
    if(!A.sign)o << '-';
    return o << A.N;
}
```