Contents

1 Basic

1.1 prefix sum

```cpp
// 1D
// build
for(int i = 1;i <= n;i++) s[i] = s[i - 1] + a[i];
// calculate a[l] + ... + a[r]
ans = s[r] - s[l - 1];


// 2D
// build
for(int i = 1;i <= m;i++)
    for(int j = 1;j <= n;j++)
        s[i][j] = s[i - 1][j] + s[i][j - 1] - s[i - 1][
            j - 1] + a[i][j];
// calculate a[r1][c1] + ... + a[r2][c2]
ans = s[r2][c2] - s[r2][c1 - 1] - s[r1 - 1][c2] + s[r1
    - 1][c1 - 1];
```

1.2 difference

```cpp
// 1D
// insert
void insert(int l, int r, int c){
    b[l] += c;
    b[r + 1] -= c;
}
// build
for(int i = 1;i <= n;i++) insert(i, i, a[i]);


// 2D
// insert
void insert(int r1, int c1, int r2, int c2, int c){
    b[r1][c1] += c;
    b[r1][c2 + 1] -= c;
    b[r2 + 1][c1] -= c;
    b[r2 + 1][c2 + 1] += c;
}
// build
for(int i = 1;i <= m;i++)
    for(int j = 1;j <= n;j++)
        insert(i, j, i, j, a[i][j]);
```

1.3 monotonic queue

```cpp
// max or min of sliding window(k size)
deque<int> q; // index

for(int i = 0;i < n;i++){
    if(!q.empty() && i - q.front() + 1 > k) q.pop_front
        ();
    while(!q.empty() && check(arr[i], q.back())) q.
        pop_back();

    q.push_back(i);

    if(i + 1 >= k) cout << q.front();
}
```

1.4 monotonic stack

```cpp
// find nearest min or max of ith element
stack<int> s;

for(int i = 0;i < n;i++){
    while(!s.empty() && check(a[i], s.top())) s.pop();
    if(s.empty()) cout << -1;
    else cout << s.top();

    s.push(nums2[i]);
}
```

1.5 kth element

```
int kth_element(int l, int r, int k){  // (l, r]
    if(r - l < 2) return a[l];
    int x = a[l + r >> 1], i = l - 1, j = r;
    while(i < j){
        do i++; while(a[i] < x);
        do j--; while(a[j] > x);
        if(i < j) swap(a[i], a[j]);
    }
    int sl = i - l;
    if(k <= sl) return kth_element(l, i, k);
    return kth_element(i, r, k - sl);
}
```

2 Graph

2.1 Traversal

```
void dfs(int x){
    v[x] = true;

    for(auto son : g[x])
        if(!v[son])
            dfs(son);
}

void bfs(int x){
    queue<int> q;
    v[x] = true;
    q.push(x);

    while(!q.empty()){
        int u = q.front();
        for(auto son : g[u]){
            if(!v[son]){
                q.push(son);
                v[son] = true;
            }

        }
        q.pop();
    }
}
```

2.2 Dijkstra

```
int dis[N];
vector<PII> g[N];
bool v[N];

int dijkstra(int st, int ed){
    memset(dis, 0x3f, sizeof dis);
    dis[st] = 0;
    priority_queue<PII, vector<PII>, greater<PII>> pq;
    pq.push({0, st});

    while(!pq.empty()){
        auto x = pq.top();
        pq.pop();
        int p = x.second, pd = x.first;
        if(v[p]) continue;

        v[p] = true;

        for(auto it : g[p]){
            int son = it.first, w = it.second;
            if(pd + w < dis[son]){
                dis[son] = pd + w;
                pq.push({dis[son], son});
            }
        }
    }

    if(dis[ed] == 0x3f3f3f3f) return -1;
    return dis[ed];
}
```

2.3 Bellman Ford

```
int dis[N];
vector<PII> g[N];

int bellman_ford(int st, int ed){
    memset(dis, 0x3f, sizeof dis);
    dis[st] = 0;

    for(int i = 0;i < k;i++){ // k times relaxation
        for(int p = 0;p < n;p++){
            for(auto it : g[p]){
                int son = it.first, d = it.second;
                if(dis[son] > dis[p] + d){
                    dis[son] = dis[p] + d;
                }
            }
        }
    }

    if(dis[ed] > INF / 2) return INF;
    return dis[ed];
}
```

2.4 SPFA

```
// spfa shortest path
int spfa(int st, int ed){
    memset(dis, 0x3f, sizeof dis);
    dis[st] = 0;
    queue<int> q;
    q.push(st);
    v[st] = true;

    while(!q.empty()){
        int p = q.front();
        q.pop();
        v[p] = false;

        for(auto it : g[p]){
            int son = it.first, w = it.second;
            if(dis[son] > dis[p] + w){
                dis[son] = dis[p] + w;
                if(!v[son]){
                    v[son] = true;
                    q.push(son);
                }
            }
        }
    }

    return dis[ed];
}

// check negetive loop
// if TLE try stack
bool spfa(){
    memset(dis, 0x3f, sizeof dis);
    dis[0] = 0;
    queue<int> q;
    q.push(0);
    v[0] = true;

    while(!q.empty()){
        int p = q.front();
        q.pop();
        v[p] = false;

        for(auto it : g[p]){
            int son = it.first, w = it. second;
            if(dis[son] > dis[p] + w){
                dis[son] = dis[p] + w;
                cnt[son] = cnt[p] + 1;

                if(cnt[son] >= n) return true;
                if(!v[son]){
                    v[son] = true;
                    q.push(son);
                }
```

```
            }
        }
    }

    return false;
}
```

2.5 Floyd

```
// init
for(int i = 0;i < n;i++){
    for(int j = 0;j < n;j++){
        if(i == j) dis[i][j] = 0;
        else dis[i][j] = INF;
    }
}

// dis[a][b] = distance of a to b
for(int k = 0;k < n;k++){
    for(int i = 0;i < n;i++){
        for(int j = 0;j < n;j++){
            dis[i][j] = min(dis[i][j], dis[i][k] + dis[
                k][j]);
        }
    }
}
```

2.6 Toposort

```
vector<int> ans;
int in[MXN];

bool toposort(){ // 0 base
    queue<int> q;
    for(int i = 0;i < n;i++){
        if(!in[i]){
            q.push(i);
            ans.push_back(i);
        }
    }

    while(!q.empty()){
        int p = q.front();
        for(auto son : g[p]){
            if(!--in[son]){
                q.push(son);
                ans.push_back(son);
            }
        }
        q.pop();
    }

    return ans.size() == n;
}
```

2.7 Kruskal

```
vector<pair<int, PII>> edges; // w a b

int find(int x){
    if(p[x] != x) p[x] = find(p[x]);
    return p[x];
}

int kruskal(){
    for(int i = 0;i < n;i++) p[i] = i;
    sort(edges.begin(), edges.end());

    int res = 0, cnt = 0;
    for(int i = 0;i < n;i++){
        int w = edges[i].F, a = edges[i].S.F, b = edges
            [i].S.S;
        a = find(a), b = find(b);

        if(a != b){
            p[a] = b;
            cnt++;
            res += w;
        }
    }

    if(cnt < n - 1) return res;
    return INF;
}
```

2.8 差分約束

```
/*
1. 求不等式組的可行解 Xi <= Xj + Ck
    [1] 把 Xi <= Xj + Ck 轉化成 Xj 到 Xi 長度為 Ck 的邊
    [2] 找一個超級源點，從他開始求最短路
    [3] 若有負環則無解，反之dis[i]是一組可行解(Xi)
2. 求最大最小可行解
    最大可行解: 最短路
    最小可行解: 最長路
*/
```

2.9 LCA

```
// 先把兩個點跳到同一層，同時往上跳直到跳到LCA的下一層
// 跳超過則fa[i, j] = 0, depth[0] = 0
#include <bits/stdc++.h>

using namespace std;

const int N = 40010, M = N * 2;

int n, m;
int depth[N], fa[N][16]; // 深度, f[i, j] = 第i個點的2^
    j祖先是誰
vector<int> g[N];

void add(int a, int b)
{
    g[a].push_back(b);
}

void bfs(int root)
{
    memset(depth, 0x3f, sizeof depth);
    queue<int> q;
    depth[0] = 0, depth[root] = 1;
    q.push(root);
    while (!q.empty())
    {
        int t = q.front();
        q.pop();
        for (auto son : g[t])
        {
            if (depth[son] > depth[t] + 1)
            {
                depth[son] = depth[t] + 1;
                q.push(son);
                fa[son][0] = t;
                for (int k = 1; k <= 15; k ++ )
                    fa[son][k] = fa[fa[son][k - 1]][k -
                        1];
            }
        }
    }
}

int lca(int a, int b)
{
    if (depth[a] < depth[b]) swap(a, b);
    for (int k = 15; k >= 0; k -- )
        if (depth[fa[a][k]] >= depth[b])
            a = fa[a][k];
    if (a == b) return a;
    for (int k = 15; k >= 0; k -- )
        if (fa[a][k] != fa[b][k])
        {
```

```
            a = fa[a][k];
            b = fa[b][k];
        }
    return fa[a][0];
}

int main()
{
    scanf("%d", &n);
    int root = 0;

    for (int i = 0; i < n; i ++ )
    {
        int a, b;
        scanf("%d%d", &a, &b);
        if (b == -1) root = a;  // a is root
        else add(a, b), add(b, a);
    }

    bfs(root);

    scanf("%d", &m);
    while (m -- )
    {
        int a, b;
        scanf("%d%d", &a, &b);
        int p = lca(a, b);
        if (p == a) puts("1");
        else if (p == b) puts("2");
        else puts("0");
    }

    return 0;
}
```

2.10 匈牙利算法

```
int match[N];
bool ch[N];

// 左半(1~n1) 右半(1~n2)

bool find(int x){
    for(auto it : g[x]){
        if(!ch[it]){
            ch[it] = true;
            if(match[it] == 0 || find(match[it])){
                match[it] = x;
                return true;
            }
        }
    }

    return false;
}

// in main
int ans = 0;
for(int i = 0;i < n1;i++){ // 某半邊的圖
    memset(ch, 0, sizeof(ch));
    if(find(i)) ans++;
}
cout << ans;
```

2.11 染色法

```
bool dfs(int u, int c){
    v[u] = c;
    for(auto it : g[u]){
        if(!v[it]){
            if(!dfs(it, 3 - c)) return false;
        }
        else if(v[it] == c) return false;
    }

    return true;
}

bool check(){
    bool flag = true;
    for(int i = 0;i < n;i++){
        if(!v[i]){
            if(!dfs(i, 1)){
                flag = false;
                break;
            }
        }
    }

    return flag;
}
```

2.12 BCC

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

using namespace std;

const int N = 60;
int timestamp, dfn[N], low[N];
vector<int> g[N];
set<int> ap;

void add(int a, int b){
    g[a].push_back(b);
    g[b].push_back(a);
}

void tarjan(int x){
    int child = 0;
    dfn[x] = low[x] = ++timestamp;

    for(auto u : g[x]){
        if(!dfn[u]){
            tarjan(u);
            child++;
            low[x] = min(low[x], low[u]);
            if(dfn[x] != 1 && low[u] >= dfn[x]) ap.
                insert(x);
        }
        else low[x] = min(low[x], dfn[u]);
    }

    if(dfn[x] == 1 && child > 1) ap.insert(x);
}

int main(){
    ios::sync_with_stdio(0);
    cin.tie(0);

    int a, b;
    while(cin >> a >> b) add(a, b);

    tarjan(a);

    if(ap.size()){
        cout << "false\n";
        int cnt = 0;
        for(auto it : ap){
            cout << it;
            if(++cnt < ap.size()) cout << " ";
        }
    }
    else cout << "true\n";

    return 0;
}
```

3 Data Structure

3.1 BIT

```
void lowbit(int x){
    return x & -x;
}
```

```
void add(int x, int c){
    for(int i = x;i <= n;i += lowbit(i))
        tr[i] += c;
}

// a[1] + ... + a[n]
int sum(int x){
    int res = 0;
    for(int i = x;i >= 0;i -= lowbit(i))
        res += tr[i];
    return res;
}
```

3.2 DSU

```
int p[MXN], sz[MXN], u;

int find(int x){
    if(p[x] != x) p[x] = find(p[x]);
    return p[x];
}

void Union(int a, int b){
    if(find(a) == find(b)) return;

    u--;
    sz[find(a)] += sz[find(b)];
    p[find(b)] = find(a);
    return;
}

// init
u = n;
for(int i = 0;i < n;i++){
    sz[i] = 1;
    p[i] = i;
}
```

3.3 Segment tree

```
// 查區間最大值
// 單點修改
struct Node{
    int l, r; // 左右端點
    int v; // [l. r]的最大值
}tr[N * 4];

void pushup(int u){
    tr[u].v = max(tr[u << 1].v, tr[u << 1 | 1].v);
}

void build(int u, int l, int r){
    tr[u] = {l, r}; // 存區間
    if(l == r) return; // 如果到葉節點=>停止
    int mid = l + r >> 1;
    build(u << 1, l, mid), build(u << 1 | 1, mid + 1, r
        ); // 遞迴建立左右子節點
}

void modify(int u, int x, int v){ // 把節點x改成v
    if(tr[u].l == x && tr[u].r == x) tr[u].v == v; //
        找到子節點
    else{
        int mid = tr[u].l + tr[u].r >> 1;
        if(x <= mid) modify(u << 1, x, v); // 在左邊
        else modify(u << 1 | 1, x, v); // 在右邊
        pushup(u);
    }
}

int query(int u, int l, int r){ // u通常是根節點，l和r
    是要查詢的區間
    if(tr[u].l >= l && tr[u].r <= r) return tr[u].v; //
        樹中節點完全被[l, r]包含
    int mid = tr[u].l + tr[u].r >> 1;
    int v = 0;
    if(l <= mid) v = query(u << 1, l, r); // 在左邊
    if(r > mid) v = max(v, query(u << 1 | 1, l, r)); //
        在右邊

    return v;
}

// 查最大連續子段和
// 單點修改

struct Node{
    int l, r;
    int sum, lmax, rmax, tmax; // 區間和/前綴最大和/後
        綴最大和/區間最大和
}tr[N * 4];

void pushup(Node &u, Node &l, Node &r){ // lr為子節點
    u.sum = l.sum + r.sum;
    u.lmax = max(l.lmax, l.sum + r.lmax);
    u.rmax = max(r.rmax, r.sum + l.rmax);
    u.tmax = max({l.tmax, r.tmax, l.rmax + r.lmax});
}

void pushup(int u){
    pushup(tr[u], tr[u << 1], tr[u << 1 | 1]);
}

void build(int u, int l, int r){ // u=當前節點, l=左子
    節點, r=右子節點
    if(l == r) tr[u] = {l, r, a[l], a[l], a[l], a[l]};
    else{
        tr[u] = {l, r};
        int mid = l + r >> 1;
        build(u << 1, l, mid), build(u << 1 | 1, mid +
            1, r);
        pushup(u);
    }
}

void modify(int u, int x, int v) // 把第x個數改成v
{
    if (tr[u].l == x && tr[u].r == x) tr[u] = {x, x, v,
        v, v, v};
    else
    {
        int mid = tr[u].l + tr[u].r >> 1;
        if (x <= mid) modify(u << 1, x, v);
        else modify(u << 1 | 1, x, v);
        pushup(u);
    }
}

Node query(int u, int l, int r) // 查詢[l, r]的資訊
{
    if (tr[u].l >= l && tr[u].r <= r) return tr[u];
    else
    {
        int mid = tr[u].l + tr[u].r >> 1;
        if (r <= mid) return query(u << 1, l, r);
        else if (l > mid) return query(u << 1 | 1, l, r
            );
        else
        {
            auto left = query(u << 1, l, r);
            auto right = query(u << 1 | 1, l, r);
            Node res;
            pushup(res, left, right);
            return res;
        }
    }
}

// 區間最大公因數
// w[l, r]都加上d
// 求 w[l, r]最大公因數
// 用線段樹維護差分數組
struct Node
{
    int l, r;
```

```
    LL sum, d; // 區間和/GCD
}tr[N * 4];

LL gcd(LL a, LL b)
{
    return b ? gcd(b, a % b) : a;
}

void pushup(Node &u, Node &l, Node &r)
{
    u.sum = l.sum + r.sum;
    u.d = gcd(l.d, r.d);
}

void pushup(int u)
{
    pushup(tr[u], tr[u << 1], tr[u << 1 | 1]);
}

void build(int u, int l, int r)
{
    if (l == r)
    {
        LL b = w[r] - w[r - 1];
        tr[u] = {l, r, b, b};
    }
    else
    {
        tr[u].l = l, tr[u].r = r;
        int mid = l + r >> 1;
        build(u << 1, l, mid), build(u << 1 | 1, mid +
            1, r);
        pushup(u);
    }
}

void modify(int u, int x, LL v)
{
    if (tr[u].l == x && tr[u].r == x)
    {
        LL b = tr[u].sum + v;
        tr[u] = {x, x, b, b};
    }
    else
    {
        int mid = tr[u].l + tr[u].r >> 1;
        if (x <= mid) modify(u << 1, x, v);
        else modify(u << 1 | 1, x, v);
        pushup(u);
    }
}

Node query(int u, int l, int r)
{
    if (tr[u].l >= l && tr[u].r <= r) return tr[u];
    else
    {
        int mid = tr[u].l + tr[u].r >> 1;
        if (r <= mid) return query(u << 1, l, r);
        else if (l > mid) return query(u << 1 | 1, l, r
            );
        else
        {
            auto left = query(u << 1, l, r);
            auto right = query(u << 1 | 1, l, r);
            Node res;
            pushup(res, left, right);
            return res;
        }
    }
}

void ask(int l, int r){ // 查[l, r]的gcd
    auto left = query(1, 1, l);
    Node right({0, 0, 0, 0});
    if (l + 1 <= r) right = query(1, l + 1, r);
    printf("%lld\n", abs(gcd(left.sum, right.d)));
}

void modify_diff(int l, int r, int d){ // [l, r] + d
    modify(1, l, d);
    if (r + 1 <= n) modify(1, r + 1, -d);
}

// w[l, r] + d
// 查詢w[l, r]之和
struct Node
{
    int l, r;
    LL sum, add; // 區間和 / 懶標
}tr[N * 4];

void pushup(int u)
{
    tr[u].sum = tr[u << 1].sum + tr[u << 1 | 1].sum;
}

void pushdown(int u)
{
    auto &root = tr[u], &left = tr[u << 1], &right = tr
        [u << 1 | 1];
    if (root.add)
    {
        left.add += root.add, left.sum += (LL)(left.r -
            left.l + 1) * root.add;
        right.add += root.add, right.sum += (LL)(right.
            r - right.l + 1) * root.add;
        root.add = 0;
    }
}

void build(int u, int l, int r)
{
    if (l == r) tr[u] = {l, r, w[r], 0};
    else
    {
        tr[u] = {l, r};
        int mid = l + r >> 1;
        build(u << 1, l, mid), build(u << 1 | 1, mid +
            1, r);
        pushup(u);
    }
}

void modify(int u, int l, int r, int d)
{
    if (tr[u].l >= l && tr[u].r <= r)
    {
        tr[u].sum += (LL)(tr[u].r - tr[u].l + 1) * d;
        tr[u].add += d;
    }
    else    // 一定要分裂
    {
        pushdown(u);
        int mid = tr[u].l + tr[u].r >> 1;
        if (l <= mid) modify(u << 1, l, r, d);
        if (r > mid) modify(u << 1 | 1, l, r, d);
        pushup(u);
    }
}

LL query(int u, int l, int r)
{
    if (tr[u].l >= l && tr[u].r <= r) return tr[u].sum;

    pushdown(u);
    int mid = tr[u].l + tr[u].r >> 1;
    LL sum = 0;
    if (l <= mid) sum = query(u << 1, l, r);
    if (r > mid) sum += query(u << 1 | 1, l, r);
    return sum;
}

// 掃描線+懶標線段樹
struct Segment
{
    double x, y1, y2;
    int k;
    bool operator< (const Segment &t)const
    {
        return x < t.x;
    }
```

```
}seg[N * 2];
struct Node
{
    int l, r;
    int cnt; // [l, r]被覆蓋幾次
    double len; // [l, r] 中 cnt > 0 的區間長度多長
}tr[N * 8];

vector<double> ys;

int find(double y)
{
    return lower_bound(ys.begin(), ys.end(), y) - ys.
        begin();
}

void pushup(int u)
{
    if (tr[u].cnt) tr[u].len = ys[tr[u].r + 1] - ys[tr[
        u].l];
    else if (tr[u].l != tr[u].r)
    {
        tr[u].len = tr[u << 1].len + tr[u << 1 | 1].len
            ;
    }
    else tr[u].len = 0;
}

void build(int u, int l, int r)
{
    tr[u] = {l, r, 0, 0};
    if (l != r)
    {
        int mid = l + r >> 1;
        build(u << 1, l, mid), build(u << 1 | 1, mid +
            1, r);
    }
}

void modify(int u, int l, int r, int k)
{
    if (tr[u].l >= l && tr[u].r <= r)
    {
        tr[u].cnt += k;
        pushup(u);
    }
    else
    {
        int mid = tr[u].l + tr[u].r >> 1;
        if (l <= mid) modify(u << 1, l, r, k);
        if (r > mid) modify(u << 1 | 1, l, r, k);
        pushup(u);
    }
}

int solve(int n)
{
    ys.clear();
    for (int i = 0, j = 0; i < n; i ++ )
    {
        double x1, y1, x2, y2;
        scanf("%lf%lf%lf%lf", &x1, &y1, &x2, &y2);
        seg[j ++ ] = {x1, y1, y2, 1};
        seg[j ++ ] = {x2, y1, y2, -1};
        ys.push_back(y1), ys.push_back(y2);
    }

    sort(ys.begin(), ys.end());
    ys.erase(unique(ys.begin(), ys.end()), ys.end());

    build(1, 0, ys.size() - 2);

    sort(seg, seg + n * 2);

    double res = 0;
    for (int i = 0; i < n * 2; i ++ )
    {
        if (i > 0) res += tr[1].len * (seg[i].x - seg[i
            - 1].x);
        modify(1, find(seg[i].y1), find(seg[i].y2) - 1,
            seg[i].k);
    }

    return res;
}

// w[l, r] * c
// w[l, r] + c
// 求 w[l, r] 區間和 % P的值
struct Node
{
    int l, r;
    int sum, add, mul; // 區間和 / 懶標(加乘)
}tr[N * 4];

void pushup(int u)
{
    tr[u].sum = (tr[u << 1].sum + tr[u << 1 | 1].sum) %
        p;
}

void eval(Node &t, int add, int mul)
{ // 對節點t修改
    t.sum = ((LL)t.sum * mul + (LL)(t.r - t.l + 1) *
        add) % p;
    t.mul = (LL)t.mul * mul % p;
    t.add = ((LL)t.add * mul + add) % p;
}

void pushdown(int u)
{
    eval(tr[u << 1], tr[u].add, tr[u].mul);
    eval(tr[u << 1 | 1], tr[u].add, tr[u].mul);
    tr[u].add = 0, tr[u].mul = 1;
}

void build(int u, int l, int r)
{
    if (l == r) tr[u] = {l, r, w[r], 0, 1};
    else
    {
        tr[u] = {l, r, 0, 0, 1};
        int mid = l + r >> 1;
        build(u << 1, l, mid), build(u << 1 | 1, mid +
            1, r);
        pushup(u);
    }
}

void modify(int u, int l, int r, int add, int mul)
{
    if (tr[u].l >= l && tr[u].r <= r) eval(tr[u], add,
        mul);
    else
    {
        pushdown(u);
        int mid = tr[u].l + tr[u].r >> 1;
        if (l <= mid) modify(u << 1, l, r, add, mul);
        if (r > mid) modify(u << 1 | 1, l, r, add, mul)
            ;
        pushup(u);
    }
}

int query(int u, int l, int r)
{
    if (tr[u].l >= l && tr[u].r <= r) return tr[u].sum;

    pushdown(u);
    int mid = tr[u].l + tr[u].r >> 1;
    int sum = 0;
    if (l <= mid) sum = query(u << 1, l, r);
    if (r > mid) sum = (sum + query(u << 1 | 1, l, r))
        % p;
    return sum;
}

void modify_mul(int l, int r, int c){
    modify(1, l, r, 0, c);
}

void modify_add(int l, int r, int c){
```

```
    modify(1, l, r, c, 1);
}
```

4 String

4.1 KMP

```
// |a| < |b| (1 base)

int f[N];
// find f[]
for(int i = 2, j = 0;i <= a.size();i++){
    while(j && a[i] != a[j + 1]) j = f[j];
    if(a[i] == a[j + 1]) j++;
    f[i] = j;
}
// find a in b
for(int i = 1, j = 0;i <= b.size();i++){
    while(j && a[j + 1] != b[i]) j = f[j];
    if(a[j + 1] == b[i]) j++;
    if(j == a.size()){
        j = f[j];
        // print
    }
}
```

4.2 Rabin Karp

```
// string hash
// according to experience P = 131 || 13331 & mod 2^64
    is best
// 1 base
unsigned long long h[MXN], p[MNX];
// init
p[0] = 1;
for(int i = 1;i <= n;i++){
    h[i] = h[i - 1] * P + str[i];
    p[i] = p[i - 1] * P;
}

unsigned long long int find(int l, int r){
    return h[r] - h[l] * h[r - l + 1];
}
```

4.3 Trie

```
const int MXN = operation * s.length();
int son[MXN][26], cnt[MXN], idx;

void insert(string s){
    int p = 0;
    for(int i = 0;i < s.length();i++){
        int u = s[i] - 'a';
        if(!son[p][u]) son[p][u] = ++idx;
        p = son[p][u];
    }
    cnt[p]++;
}

int find(string s){
    int p = 0;
    for(int i = 0;i < s.length();i++){
        int u = s[i] - 'a';
        if(!son[p][u]) return 0;
        p = son[p][u];
    }
    return cnt[p];
}

bool find_prefix(string s){
    int p = 0;
    for(int i = 0;i < s.length();i++){
        int u = s[i] - 'a';
        if(!son[p][u]) return false;
        p = son[p][u];
    }
    return true;
}
```

5 Math

5.1 extgcd

```
int extgcd(int a, int b, int &x, int &y){
    if(b == 0){
        x = 1, y = 0;
        return a;
    }
    int t = extgcd(b, a % b, y, x);
    y = y - a / b * x;
    return t;
}
```

5.2 線性篩

```
vector<int> prime;
bool ch[N];

void get_prime(){
    for(int i = 2;i <= n;i++){
        if(!ch[i]) prime.push_back(i);
        for(int j = 0;prime[j] <= n / i;j++){
            ch[prime[j] * i] = true;
            if(i % prime[j] == 0) break;
        }
    }
}
```

5.3 euler

```
// one
int euler(int x){
    int res = x;
    for(int i = 2;i <= x / i;i++){
        if(x % i == 0){
            res = res - res / i;
            while(x % i == 0) x /= i;
        }
    }
    if(x > 1) res = res - res / x;

    return res;
}

// 1 ~ n
euler[1] = 1;
for(int i = 2;i <= n;i++){
    if(!ch[i]){
        prime.push_back(i);
        euler[i] = i - 1;
    }

    for(int j = 0;prime[j] <= n / i;i++){
        int t = prime[j] * i;
        ch[t] = true;
        if(i % prime[j] == 0){
            euler[t] = euler[i] * prime[j];
            break;
        }
        euler[t] = euler[i] * (prime[j] - 1);
    }
}
```

5.4 guass

```
// a是增廣矩陣
double a[N][N], eps = 1e-6;

int guass(){
    int r, c;
```

```
    for(r = 0, c = 0;c < n;c++){
        // 找最大的開頭;
        int t = r;
        for(int i = r;i < n;i++){
            if(abs(a[i][c]) > abs(a[t][c]))
                t = i;
        }

        if(abs(a[t][c]) < eps) continue;

        // 換到上面
        for(int i = c;i <= n;i++) swap(a[t][i], a[r][i
            ]);
        // 弄出leading one(每個數除開頭)
        for(int i = n;i >= c;i--) a[r][i] /= a[r][c];
        // 去掉底下的0
        for(int i = r + 1; i < n;i++){
            if(abs(a[i][c]) > eps){
                for(int j = n;j >= c;j--){
                    a[i][j] -= a[r][j] * a[i][c];
                }
            }
        }
        r++;
    }

    if(r < n){
        for(int i = r;i < n;i++){
            if(abs(a[i][n]) > eps)
                return 2;// 無解
        }
        return 1; // 無限多組解
    }
    for(int i = n - 1;i >= 0;i--){
        for(int j = i + 1;j < n;j++)
            a[i][n] -= a[i][j] * a[j][n];
    }
    return 0;// 有唯一解
}

void solve(int n){
    for(int i = 0;i < n;i++){
        for(int j = 0;j <= n;j++){
            cin >> a[i][j];
        }
    }

    int t = guass();

    if(t == 0){
        for(int i = 0;i < n;i++){
            if(abs(a[i][n]) < eps) cout << 0.00;
            else cout << fixed << setprecision(2) << a[
                i][n];
            cout << "\n";
        }
    }
    else if(t == 1) cout << "Infinite group solutions";
    else cout << "No solution";
}
```

5.5 快速冪

```
ll qpow(int a, int k, int p){
    ll res = 1;
    while(k){
        if(a & 1) res = res * a % p;
        a = (ll)a * a % p;
        k >>= 1;
    }

    return res;
}
```

5.6 模逆元

```
// a * x 同餘 1 (mod p)

// by qpow
ll solve(int a, int p){
    if(a % p == 0) cout << "NO";
    else cout << qpow(a, p - 2, p);
}

// by extgcd
ll solve(int a, int p){
    if(gcd(a, p) != 1) cout << "No";
    else{
        ll ans, k;
        extgcd(a, p, ans, k);
        ans = (ans % p + p) % p;
        cout << ans << "\n";
    }
}
```

5.7 簡單博弈論

```
// 先手必敗:a_1 ^ a_2 ^...^ a_k = 0
// 先手必勝:a_1 ^ a_2 ^...^ a_k != 0

const int n, m // n = 石頭堆數   m = s[n]數量
int s[n], f[m]; // s[n] = 可一次拿走的石頭數   f[m] =
    記憶化搜索sg

memset(f, -1, sizeof f);
int sg(int x){
    if(f[x] != -1) return f[x];

    unorded_set<int> S;
    for(int i = 0;i < m;i++){
        if(x > s[i]) S.insert(sg[x - s[i]]);
    }

    // mex 找出集合中沒出現最小的正整數
    for(int i = 0;;i++)
        if(!S.count(i))
            return f[x] = i;
}
```

5.8 CRT

$$\begin{cases} x \equiv a_1 (mod\ m_1) \\ x \equiv a_2 (mod\ m_2) \\ \quad \dots \\ x \equiv a_k (mod\ m_k) \end{cases}$$

$x = a_1 M_1 M_1^{-1} + a_2 M_2 M_2^{-1} + .. + a_k M_k M_k^{-1}$

$M = m_1 m_2 ... m_k$

$M_i = \frac{M}{m_i}$

5.9 卡特蘭數

$C_{2n}^n - C_{2n}^{n-1} = \frac{(2n)!}{n!n!} - \frac{(2n)!}{(n+1)!(n-1)!} = \frac{(2n)!(n+1)-(2n)!n}{(n+1)!n!} = \frac{(2n)!}{(n+1)!n!} = \frac{1}{n+1} \times \frac{(2n)!}{n!n!} = \frac{C_{2n}^n}{n+1}$

6 Geometry

6.1 Basic

```
struct Point { double x, y; };         // 点
using Vec = Point;                     // 向量
struct Line { Point P; Vec v; };       // 直线（点向式）
struct Seg { Point A, B; };            // 线段（存两个端
    点）
struct Circle { Point O; double r; }; // 圆（存圆心和半
    径）

const Point O = {0, 0};                          // 原点
const Line Ox = {O, {1, 0}}, Oy = {O, {0, 1}}; // 坐标
    轴
const double PI = acos(-1), EPS = 1e-9;
```

```
bool eq(double a, double b) { return abs(a - b) < EPS;
    } // ==
bool gt(double a, double b) { return a - b > EPS; }
        // >
bool lt(double a, double b) { return a - b < -EPS; }
        // <
bool ge(double a, double b) { return a - b > -EPS; }
        // >=
bool le(double a, double b) { return a - b < EPS; }
        // <=

Vec r90a(Vec v) { return {-v.y, v.x}; }
                                // 逆时针旋转90度的向量
Vec r90c(Vec v) { return {v.y, -v.x}; }
                                // 顺时针旋转90度的向量
Vec operator+(Vec u, Vec v) { return {u.x + v.x, u.y +
    v.y}; }   // 向量加向量
Vec operator-(Vec u, Vec v) { return {u.x - v.x, u.y -
    v.y}; }   // 向量减向量
Vec operator*(double k, Vec v) { return {k * v.x, k * v
    .y}; }    // 数乘
double operator*(Vec u, Vec v) { return u.x * v.x + u.y
     * v.y; } // 点乘
double operator^(Vec u, Vec v) { return u.x * v.y - u.y
     * v.x; } // 叉乘
double len(Vec v) { return sqrt(v.x * v.x + v.y * v.y);
     }        // 向量长度
double slope(Vec v) { return v.y / v.x; }
                              // 斜率 // NOTE 不要用isinf
    判断斜率不存在，用后面的paral_y


// 两向量的夹角余弦
// DEPENDS len, V*V
double cos_t(Vec u, Vec v) { return u * v / len(u) /
    len(v); }

// 归一化向量（与原向量方向相同的单位向量）
// DEPENDS len
Vec norm(Vec v) { return {v.x / len(v), v.y / len(v)};
    }

// 与原向量平行且横坐标大于等于0的单位向量
// DEPENDS d*V, len
Vec pnorm(Vec v) { return (v.x < 0 ? -1 : 1) / len(v) *
     v; }

// 线段的方向向量
// DEPENDS V-V
// NOTE 直线的方向向量直接访问属性v
Vec dvec(Seg l) { return l.B - l.A; }



// 两点式直线
// DEPENDS V-V
Line line(Point A, Point B) { return {A, B - A}; }

// 斜截式直线
Line line(double k, double b) { return {{0, b}, {1, k
    }}; }

// 点斜式直线
Line line(Point P, double k) { return {P, {1, k}}; }

// 线段所在直线
// DEPENDS V-V
Line line(Seg l) { return {l.A, l.B - l.A}; }

// 给定直线的横坐标求纵坐标
// NOTE 请确保直线不与y轴平行
double at_x(Line l, double x) { return l.P.y + (x - l.P
    .x) * l.v.y / l.v.x; }

// 给定直线的纵坐标求横坐标
// NOTE 请确保直线不与x轴平行
double at_y(Line l, double y) { return l.P.x - (y + l.P
    .y) * l.v.x / l.v.y; }
```

```
// 点到直线的垂足
// DEPENDS V-V, V*V, d*V
Point pedal(Point P, Line l) { return l.P - (l.P - P) *
     l.v / (l.v * l.v) * l.v; }

// 过某点作直线的垂线
// DEPENDS r90c
Line perp(Line l, Point P) { return {P, r90c(l.v)}; }

// 角平分线
// DEPENDS V+V, len, norm
Line bisec(Point P, Vec u, Vec v) { return {P, norm(u)
    + norm(v)}; }


// 线段的方向向量
// DEPENDS V-V
// NOTE 直线的方向向量直接访问属性v
Vec dvec(Seg l) { return l.B - l.A; }

// 线段中点
Point midp(Seg l) { return {(l.A.x + l.B.x) / 2, (l.A.y
     + l.B.y) / 2}; }

// 线段中垂线
// DEPENDS r90c, V-V, midp
Line perp(Seg l) { return {midp(l), r90c(l.B - l.A)}; }


// 向量是否互相垂直
// DEPENDS eq, V*V
bool verti(Vec u, Vec v) { return eq(u * v, 0); }

// 向量是否互相平行
// DEPENDS eq, V^V
bool paral(Vec u, Vec v) { return eq(u ^ v, 0); }

// 向量是否与x轴平行
// DEPENDS eq
bool paral_x(Vec v) { return eq(v.y, 0); }

// 向量是否与y轴平行
// DEPENDS eq
bool paral_y(Vec v) { return eq(v.x, 0); }

// 点是否在直线上
// DEPENDS eq
bool on(Point P, Line l) { return eq((P.x - l.P.x) * l.
    v.y, (P.y - l.P.y) * l.v.x); }

// 点是否在线段上
// DEPENDS eq, len, V-V
bool on(Point P, Seg l) { return eq(len(P - l.A) + len(
    P - l.B), len(l.A - l.B)); }

// 两个点是否重合
// DEPENDS eq
bool operator==(Point A, Point B) { return eq(A.x, B.x)
     && eq(A.y, B.y); }

// 两条直线是否重合
// DEPENDS eq, on(L)
bool operator==(Line a, Line b) { return on(a.P, b) &&
    on(a.P + a.v, b); }

// 两条线段是否重合
// DEPENDS eq, P==P
bool operator==(Seg a, Seg b) { return (a.A == b.A && a
    .B == b.B) || (a.A == b.B && a.B == b.A); }

// 以横坐标为第一关键词、纵坐标为第二关键词比较两个点
// DEPENDS eq, lt
bool operator<(Point A, Point B) { return lt(A.x, B.x)
    || (eq(A.x, B.x) && lt(A.y, B.y)); }

// 直线与圆是否相切
// DEPENDS eq, V^V, len
bool tangency(Line l, Circle C) { return eq(abs((C.O ^
    l.v) - (l.P ^ l.v)), C.r * len(l.v)); }
```

```cpp
// 圆与圆是否相切
// DEPENDS eq, V-V, len
bool tangency(Circle C1, Circle C2) { return eq(len(C1.O - C2.O), C1.r + C2.r); }


// 两点间的距离
// DEPENDS len, V-V
double dis(Point A, Point B) { return len(A - B); }

// 点到直线的距离
// DEPENDS V^V, len
double dis(Point P, Line l) { return abs((P ^ l.v) - (l.P ^ l.v)) / len(l.v); }

// 平行直线间的距离
// DEPENDS d*V, V^V, len, pnorm
// NOTE 请确保两直线是平行的
double dis(Line a, Line b) { return abs((a.P ^ pnorm(a.v)) - (b.P ^ pnorm(b.v))); }


// 平移
// DEPENDS V+V
Line operator+(Line l, Vec v) { return {l.P + v, l.v}; }
Seg operator+(Seg l, Vec v) { return {l.A + v, l.B + v}; }

// 旋转
// DEPENDS V+V, V-V
Point rotate(Point P, double rad) { return {cos(rad) * P.x - sin(rad) * P.y, sin(rad) * P.x + cos(rad) * P.y}; }
Point rotate(Point P, double rad, Point C) { return C + rotate(P - C, rad); }                    // DEPENDS ^1
Line rotate(Line l, double rad, Point C = O) { return {rotate(l.P, rad, C), rotate(l.v, rad)}; }  // DEPENDS ^1, ^2
Seg rotate(Seg l, double rad, Point C = O) { return {rotate(l.A, rad, C), rotate(l.B, rad, C)}; } // DEPENDS ^1, ^2


// 对称
// 关于点对称
Point reflect(Point A, Point P) { return {P.x * 2 - A.x, P.y * 2 - A.y}; }
Line reflect(Line l, Point P) { return {reflect(l.P, P), l.v}; }           // DEPENDS ^1
Seg reflect(Seg l, Point P) { return {reflect(l.A, P), reflect(l.B, P)}; } // DEPENDS ^1
// 关于直线对称
// DEPENDS V-V, V*V, d*V, pedal
// NOTE 向量和点在这里的表现不同，求向量关于某直线的对称向量需要用reflect_v
Point reflect(Point A, Line ax) { return reflect(A, pedal(A, ax)); }             // DEPENDS ^1
Vec reflect_v(Vec v, Line ax) { return reflect(v, ax) - reflect(O, ax); }        // DEPENDS ^1, ^4
Line reflect(Line l, Line ax) { return {reflect(l.P, ax), reflect_v(l.v, ax)}; } // DEPENDS ^1, ^4, ^5
Seg reflect(Seg l, Line ax) { return {reflect(l.A, ax), reflect(l.B, ax)}; }     // DEPENDS ^1, ^4


// 直线与直线交点
// DEPENDS eq, d*V, V*V, V+V, V^V
vector<Point> inter(Line a, Line b)
{
    double c = a.v ^ b.v;
    if (eq(c, 0)) return {};
    Vec v = 1 / c * Vec{a.P ^ (a.P + a.v), b.P ^ (b.P + b.v)};
    return {{v * Vec{-b.v.x, a.v.x}, v * Vec{-b.v.y, a.v.y}}};
}

// 直线与圆交点
// DEPENDS eq, gt, V+V, V-V, V*V, d*V, len, pedal
vector<Point> inter(Line l, Circle C)
{
    Point P = pedal(C.O, l);
    double h = len(P - C.O);
    if (gt(h, C.r)) return {};
    if (eq(h, C.r)) return {P};
    double d = sqrt(C.r * C.r - h * h);
    Vec vec = d / len(l.v) * l.v;
    return {P + vec, P - vec};
}

// 圆与圆的交点
// DEPENDS eq, gt, V+V, V-V, d*V, len, r90c
vector<Point> inter(Circle C1, Circle C2)
{
    Vec v1 = C2.O - C1.O, v2 = r90c(v1);
    double d = len(v1);
    if (gt(d, C1.r + C2.r) || gt(abs(C1.r - C2.r), d))
        return {};
    if (eq(d, C1.r + C2.r) || eq(d, abs(C1.r - C2.r)))
        return {C1.O + C1.r / d * v1};
    double a = ((C1.r * C1.r - C2.r * C2.r) / d + d) / 2;
    double h = sqrt(C1.r * C1.r - a * a);
    Vec av = a / len(v1) * v1, hv = h / len(v2) * v2;
    return {C1.O + av + hv, C1.O + av - hv};
}


// 三角形的重心
Point barycenter(Point A, Point B, Point C)
{
    return {(A.x + B.x + C.x) / 3, (A.y + B.y + C.y) / 3};
}

// 三角形的外心
// DEPENDS r90c, V*V, d*V, V-V, V+V
// NOTE 给定圆上三点求圆，要先判断是否三点共线
Point circumcenter(Point A, Point B, Point C)
{
    double a = A * A, b = B * B, c = C * C;
    double d = 2 * (A.x * (B.y - C.y) + B.x * (C.y - A.y) + C.x * (A.y - B.y));
    return 1 / d * r90c(a * (B - C) + b * (C - A) + c * (A - B));
}

// 三角形的内心
// DEPENDS len, d*V, V-V, V+V
Point incenter(Point A, Point B, Point C)
{
    double a = len(B - C), b = len(A - C), c = len(A - B);
    double d = a + b + c;
    return 1 / d * (a * A + b * B + c * C);
}

// 三角形的垂心
// DEPENDS V*V, d*V, V-V, V^V, r90c
Point orthocenter(Point A, Point B, Point C)
{
    double n = B * (A - C), m = A * (B - C);
    double d = (B - C) ^ (A - C);
    return 1 / d * r90c(n * (C - B) - m * (C - A));
}
```

7 Other

7.1 backpack

```cpp
// 多重背包
const int N = 250000, M = 2010;
int v[N], w[N], n, V; // 體積/價值/n種物品/背包容量
int dp[M];

int solve(){
    int cnt = 0;
    for(int i = 0, a, b, c;i < n;i++){
```

```
        cin >> a >> b >> c;

        int k = 1;
        while(k <= c){
            cnt++;
            v[cnt] = a * k;
            w[cnt] = b * k;

            c -= k;
            k <<= 1;
        }
        if(c){
            cnt++;
            v[cnt] = a * c;
            w[cnt] = b * c;
        }
    }

    n = cnt;

    for(int i = 1;i <= n;i++)
        for(int j = V;j >= v[i];j--)
            dp[j] = max(dp[j], dp[j - v[i]] + w[i]);

    return dp[V];
}

// 區間DP
for(區間長度: len)
    for(區間的剖分點: i = 1 ~ n - len + 1)
        j = i + len - 1
        for(k = i ~ j)
            dp[] = ...
```

7.2 cmp

```
// a = b 時 必為false
// 5 4 3 2 1  sort(a, a + n, greater<int>());
bool cmp(int lhs, int rhs){
    return lhs > rhs;
}

bool cmp(struct lhs, struct rhs){
    if(s1.age==s2.age)return s1.name <s2.name;
    else  return s1.age < s2.age;
}
```

7.3 Python

```
# import
import math
from math import *
import math as M
from math import sqrt

# input
n = int( input() )
a = [ int(x) for x in input().split() ]

# EOF
while True:
    try:
        solve()
    except:
        break

# output
print( x, sep=' ')
print( ''.join( str(x)+' ' for x in a ) )
print( '{:5d}'.format(x) )

# sort
a.sort()
sorted(a)

# list
a = [ x for x in range(n) ]
a.append(x)

# Basic operator
a, b = 10, 20
a/b # 0.5
a//b # 0
a%b # 10
a**b # 10^20

# if, else if, else
if a==0:
    print('zero')
elif a>0:
    print('postive')
else:
    print('negative')

# loop
while a==b and b==c:
for i in LIST:

# stack   # C++
stack = [3,4,5]
stack.append(6) # push()
stack.pop() # pop()
stack[-1] # top()
len(stack) # size() O(1)

# queue      # C++
from collections import deque
queue = deque([3,4,5])
queue.append(6) # push()
queue.popleft() # pop()
queue[0] # front()
len(queue) # size() O(1)

# random
from random import *
randrange(L,R,step) # [L,R) L+k*step
randint(L,R) # int from [L,R]
choice(list) # pick 1 item from list
choices(list,k) # pick k item
shuffle(list)
Uniform(L,R) # float from [L,R]


# Decimal
from fractions import Fraction
from decimal import Decimal, getcontext
getcontext().prec = 250 # set precision
itwo = Decimal(0.5)
two = Decimal(2)
N = 200
def angle(cosT):
"""given cos(theta) in decimal return theta"""
for i in range(N):
    cosT = ((cosT + 1) / two) ** itwo
sinT = (1- cosT * cosT) ** itwo
return sinT * (2 ** N)
pi = angle(Decimal(-1))

# log
math.log(10, 2)  # log_2 10
```