算法竞赛模板

by ChaomengOrion

最后修改于2025 年 1 月 18 日

目录

1 前期准备

1.1 Cpp 文件一键编译测试

Windows - **build.bat**

```
@echo off
set exe=.\output\%~n1.exe
if not exist output mkdir output
g++ %1 -o %exe% -std=c++17 -Wall -Wextra -g3 -O2 -D_GLIBCXX_DEBUG && (
    echo [build done to %exe%]
    %exe%
)
```

1.2 Cpp 模板

注意题目是不是多组样例

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

using i64 = long long;

#define LOG(...) std::cerr << "Line[" << __LINE__ << "]: " << __VA_ARGS__ << std::endl;
#define LOGV(_vec, _size) std::cerr << #_vec << " = " << '['; for (int _i = 0; _i < (_size); _i++) { std::cerr << (_vec)[_i]; if (_i != (_size) - 1) std::cerr << ", "; } std::cerr << ']' << std::endl;

void solve() {}

int main() {
    std::cin.tie(nullptr)->sync_with_stdio(false);
    int t; std::cin >> t; while (t--) solve();
    return 0;
}
```

2 算法

2.1 快速幂

```
using i64 = long long;

i64 binpow(i64 a, i64 b) {
    // a %= m;
    i64 res = 1;
    while (b > 0) {
        if (b & 1) res = res * a; // % m;
        a = a * a; // % m;
        b >>= 1;
```

```
    }
    return res;
}

long long f(long long n, long long m) {
    // 求的是 m 个 n 相乘，这里 n 是一个正整数
    if (m == 0)
        return 1;
    else if (m == 1)
        return n;
    else if (m % 2 == 0)
        return f(n * n, m / 2); // 偶数时的降幂
    return f(n * n, m / 2) * n; // 奇数时的降幂
}
```

2.2 DP

2.2.1 LIS

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

int main() {
    std::vector<int> arr = {3, 1, 3, 4, 5, 3, 2};
    int size = arr.size();
    std::vector<int> low(size, INT_MAX);
    int max_s = 0;
    for (int i = 0; i < size; i++) {
        int pos = std::lower_bound(low.begin(), low.end(), arr[i]) - low.begin();
        low[pos] = arr[i];
        max_s = std::max(max_s, pos + 1);
    }
    std::cout << max_s << std::endl;
}
```

2.2.2 LIS

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

void solve() {
    std::string S, T;
    std::cin >> S >> T;
    std::vector dp(S.size() + 1, std::vector<int>(T.size() + 1));
    for (int i = 1; i <= S.size(); i++) {
        for (int j = 1; j <= T.size(); j++) {
            if (S[i - 1] == T[j - 1]) {
                dp[i][j] = dp[i - 1][j - 1] + 1;
            } else {
```

```
                dp[i][j] = std::max(dp[i - 1][j], dp[i][j - 1]);
            }
        }
    }

    std::string ans;
    int i = S.size(), j = T.size();
    while (i > 0 && j > 0) {
        if (S[i - 1] == T[j - 1]) {
            ans += S[i - 1];
            i--; j--;
        } else {
            dp[i - 1][j] > dp[i][j - 1] ? i-- : j--;
        }
    }
    std::reverse(ans.begin(), ans.end());
    std::cout << ans << std::endl;
}
```

2.2.3 01 背包

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

void solve() {
    int N, M;
    std::cin >> N >> M;
    std::vector<std::pair<int, int>> items(N + 1); // Wi Di
    for (int i = 1; i <= N; i++) {
        std::cin >> items[i].first >> items[i].second;
    }
    std::vector<int> dp(M + 1, 0), last(M + 1, 0);

    for (int i = 1; i <= N; i++) {
        for (int j = 0; j <= M; j++) {
            if (j - items[i].first >= 0)
                dp[j] = std::max(last[j], last[j - items[i].first] + items[i].second);
            else
                dp[j] = last[j];
        }
        std::swap(dp, last);
        std::fill(dp.begin(), dp.end(), 0);
    }

    std::cout << last.back() << std::endl;
}
```

2.2.4 完全背包

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

using i64 = long long;

#define TRACE(x) std::cout << "TRACE: " << #x << " = " << (x) << std::endl;
#define DEBUG(...) std::cerr << "DEBUG: " << __VA_ARGS__ << std::endl;
#define DEBUGV(vec, size) std::cerr << '['; for (int i = 0; i < size; i++) { std::cerr << vec[i]; if (i != size - 1) std::cerr << ", "; } std::cerr << ']' << std::endl;

void solve()
{
    int T, M;
    std::cin >> T >> M; // T: max time cost, M: herbs count
    std::vector<int> herbTimes(M), herbValues(M);
    for (int i = 0; i < M; i++)
        std::cin >> herbTimes[i] >> herbValues[i];
    // dp[i][j] = max(dp[i-1][j], dp[i][j-w[i]]+v[i]) // j >= w[i]
    std::vector dp(std::vector<i64>(T + 1, 0));
    // dp[i][j] = max value of first i herbs with time j
    for (int i = 1; i <= M; i++) {
        for (int t = 0; t <= T; t++) {
            if (t < herbTimes[i - 1])
                dp[t] = dp[t];
            else
                dp[t] = std::max(dp[t], dp[t - herbTimes[i - 1]] + herbValues[i - 1]);
        }
    }
    std::cout << dp.back() << std::endl;

}

int main()
{
    std::cin.tie(nullptr)->sync_with_stdio(false);
#ifdef CFMode
    int t; std::cin >> t; while (t--) solve();
#else
    solve();
#endif
    return 0;
}
```

2.2.5 多重背包

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

using i64 = long long;

void solve() {
    int N, M;
    std::cin >> N >> M;
    std::vector<std::pair<int, int>> items; // weight, value

    // 二进制拆分每种物品的数量
    for (int i = 1; i <= N; i++) {
        int V, W, m;
        std::cin >> V >> W >> m;
        int k = 1;
        while(m > 0){
            int cnt = std::min(k, m);
            items.emplace_back(W * cnt, V * cnt);
            m -= cnt;
            k <<= 1;
        }
    }

    std::vector<int> dp(M + 1, 0);

    // 0-1 背包标准遍历
    for(auto &[weight, value] : items){
        for(int j = M; j >= weight; j--){
            dp[j] = std::max(dp[j], dp[j - weight] + value);
        }
    }

    std::cout << dp[M] << std::endl;
}

int main(){
    std::cin.tie(nullptr)->sync_with_stdio(false);
    solve();
    return 0;
}
```

2.3 二维差分

```cpp
void best()
{
    int N, M; std::cin >> N >> M;
```

```
    std::vector map(N + 1, std::vector<int>(N + 1, 0));
    std::vector diff(N + 2, std::vector<int>(N + 2, 0));
    for (int i = 1; i <= N; i++) for (int j = 1; j <= N; j++) std::cin >> map[i][j];

    while (M--) {
        int x1, x2, y1, y2;
        std::cin >> x1 >> y1 >> x2 >> y2;
        diff[x1][y1]++;
        diff[x1][y2 + 1]--;
        diff[x2 + 1][y1]--;
        diff[x2 + 1][y2 + 1]++;
    }
//  修改
//  ###+@@@-
//  ###@@@@#
//  ###-###+
    for (int i = 1; i <= N; i++)
        for (int j = 1; j <= N; j++)
            diff[i][j] += diff[i - 1][j];

    for (int i = 1; i <= N; i++)
        for (int j = 1; j <= N; j++)
            diff[i][j] += diff[i][j - 1];

    for (int i = 1; i <= N; i++) {
        for (int j = 1; j <= N; j++) {
            std::cout << diff[i][j] + map[i][j] << ' ';
        }
        std::cout << '\n';
    }
}
```

2.4 并查集

```
void solve()
{
    int N, M;
    std::cin >> N >> M;
    std::vector<int> fa(N + 1);
    for (int i = 0; i <= N; i++) {
        fa[i] = i;
    }
    auto find = [&](auto&& find, int x) -> int {
        return x == fa[x] ? x : fa[x] = find(find, fa[x]);
    };

    while (M--) {
```

```
        int Z, X, Y;
        std::cin >> Z >> X >> Y;
        if (Z == 1) {
            fa[find(find, Y)] = fa[find(find, X)];
        } else if (Z == 2) {
            std::cout << (fa[find(find, Y)] == fa[find(find, X)] ? 'Y' : 'N') <<
            ↪ std::endl;
        }
    }
}
```

2.5 图论

2.5.1 最短路 (Dijkstra)

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

struct node {
    std::vector<std::pair<int, int>> to;
    bool visit = false;
    int dis = INT_MAX;
};

void solve()
{
    int N, M, S; // 点 边 出发点
    std::cin >> N >> M >> S;
    std::vector<node> graph(N + 1);
    for (int i = 0; i < M; i++) {
        int u, v, len;
        std::cin >> u >> v >> len;
        graph[u].to.emplace_back(v, len);
        //graph[v].to.emplace_back(u, len);
    }

    // <dis, id>
    std::priority_queue<std::pair<int, int>, std::vector<std::pair<int, int>>,
    ↪ std::greater<std::pair<int, int>>> Q;
    graph[S].dis = 0; Q.emplace(0, S); //! 记得
    int cnt = 0;
    while(!Q.empty() && cnt < N) {
        auto [dis, x] = Q.top(); // 取出最近的
        Q.pop();
        if (graph[x].visit) continue; //? 可改成 cur.dis != dis
        graph[x].visit = true;
        cnt++;
        for (auto [son, cost] : graph[x].to) {
```

```
            if (graph[son].visit) continue; //? 可省略，下一个 if 也会筛掉，不过最小生成树时候不能删
            if (graph[x].dis + cost < graph[son].dis) { // 松弛边
                graph[son].dis = graph[x].dis + cost;
                Q.emplace(graph[x].dis + cost, son);
            }
        }
    }

    for (int i = 1; i <= N; i++) {
        std::cout << graph[i].dis << ' ';
    }
}

int main()
{
    std::cin.tie(nullptr)->sync_with_stdio(false);
    solve();
    return 0;
}
```

2.5.2 最短路 (SPFA)

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

struct node {
    std::vector<std::pair<int, int>> to;
    int dis = INT_MAX;
    bool inQueue = false; // 是否已经在 spfa 队列中
    int cnt = 0; // 最后一次访问到该节点的 spfa 的轮数
};

void solve() {
    int N, M;
    std::cin >> N >> M;
    std::vector<node> graph(N + 1);
    for (int i = 0; i < M; i++) {
        int u, v, c;
        std::cin >> u >> v >> c;
        if (c >= 0) {
            graph[u].to.emplace_back(v, c);
            graph[v].to.emplace_back(u, c);
        } else {
            graph[u].to.emplace_back(v, c);
        }
    }

```

```
    //* SPFA
    graph[1].cnt = 0, graph[1].dis = 0, graph[1].inQueue = true;
    std::queue<int> Q;
    Q.push(1);
    while (!Q.empty()) {
        int x = Q.front();
        Q.pop();
        graph[x].inQueue = false;
        for (auto [son, cost] : graph[x].to) {
            if (graph[x].dis + cost < graph[son].dis) { // 尝试松弛边，松弛成功就加入下一轮
                graph[son].dis = graph[x].dis + cost;
                graph[son].cnt = graph[x].cnt + 1;
                if (graph[son].cnt >= N) {
                    std::cout << "YES" << std::endl; // 存在负环
                    return;
                }
                if (!graph[son].inQueue) {
                    Q.emplace(son);
                    graph[son].inQueue = true;
                }
            }
        }
    }

    std::cout << "NO" << std::endl;
}
```

2.5.3 最小生成树

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

struct node {
    std::vector<std::pair<int, int>> to;
    bool visit = false;
    int dis = INT_MAX;
};

void solve()
{
    int N, M, S; // 点 边 出发点
    std::cin >> N >> M >> S;
    std::vector<node> graph(N + 1);
    for (int i = 0; i < M; i++) {
        int u, v, len;
        std::cin >> u >> v >> len;
        graph[u].to.emplace_back(v, len);
        //graph[v].to.emplace_back(u, len);
```

```
    }

    // <dis, id>
    std::priority_queue<std::pair<int, int>, std::vector<std::pair<int, int>>,
    ↪ std::greater<std::pair<int, int>>> Q;
    graph[S].dis = 0; Q.emplace(0, S); //! 记得
    int cnt = 0;
    while(!Q.empty() && cnt < N) {
        auto [dis, x] = Q.top(); // 取出最近的
        Q.pop();
        if (graph[x].visit) continue; //? 可改成 cur.dis != dis
        graph[x].visit = true;
        cnt++;
        for (auto [son, cost] : graph[x].to) {
            if (graph[son].visit) continue; //? 可省略，下一个 if 也会筛掉，不过最小生成树时候
            ↪ 不能删
            if (graph[x].dis + cost < graph[son].dis) { // 松弛边
                graph[son].dis = graph[x].dis + cost;
                Q.emplace(graph[x].dis + cost, son);
            }
        }
    }

    for (int i = 1; i <= N; i++) {
        std::cout << graph[i].dis << ' ';
    }
}

int main()
{
    std::cin.tie(nullptr)->sync_with_stdio(false);
    solve();
    return 0;
}
```

2.5.4 最近公共祖先 LCA

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

using i64 = long long;

#define LOG(...) std::cerr << "Line[" << __LINE__ << "]: " << __VA_ARGS__ << std::endl;
#define LOGV(_vec, _size) std::cerr << #_vec << " = " << '['; for (int _i = 0; _i <
↪ (_size); _i++) { std::cerr << (_vec)[_i]; if (_i != (_size) - 1) std::cerr << ", ";
↪ } std::cerr << ']' << std::endl;

struct node {
```

```
    std::vector<int> to;
    int depth = -1;
};

void solve() {
    int N, Q, S;
    std::cin >> N >> Q >> S;
    std::vector<node> tree(N + 1);
    for (int i = 0; i < N - 1; i++) {
        int u, v;
        std::cin >> u >> v;
        tree[u].to.push_back(v);
        tree[v].to.push_back(u);
    }

    std::array<std::vector<int>, 20> f;
    std::fill(f.begin(), f.end(), std::vector<int>(N + 1, 0));

    auto dfs_dep = [&](auto&& self, int x, int dep, int fa) -> void {
        if (tree[x].depth != -1) return;
        tree[x].depth = dep;
        f[0][x] = fa;
        for (int son : tree[x].to) {
            self(self, son, dep + 1, x);
        }
    };

    dfs_dep(dfs_dep, S, 1, 0);

    for (int i = 1; i <= 19; i++) {
        for (int j = 1; j <= N; j++) {
            f[i][j] = f[i - 1][f[i - 1][j]];
        }
    }

    auto get_lca = [&](int a, int b) -> int {
        if (tree[a].depth > tree[b].depth) std::swap(a, b); // a <= b
        int gap = tree[b].depth - tree[a].depth, cnt = 0;
        while (gap) { //? 补齐到同一个高度
            if (gap & 1) {
                b = f[cnt][b];
            }
            cnt++;
            gap >>= 1;
        }
        if (a == b) return a;
        for (int i = 19; i >= 0; i--) { //? 在同一个高度基础上往上跳，跳到最近公共祖先的子节点
```

```
            if (f[i][a] != f[i][b]) {
                a = f[i][a];
                b = f[i][b];
            }
        }
        return f[0][a];
    };

    while (Q--) {
        int a, b;
        std::cin >> a >> b;
        std::cout << get_lca(a, b) << std::endl;
    }
}

int main() {
    std::cin.tie(nullptr)->sync_with_stdio(false);
    solve();
    return 0;
}
```

3 数据结构

3.1 树状数组

```
#include <bits/stdc++.h>
using i64 = long long;

template <class T, class Merge = std::plus<T>>
struct BIT {
    const Merge merge;
    std::vector<T> t;

    BIT(int n) : t(n + 1), merge(Merge()) {}

    // O(n) build BIT
    BIT(const std::vector<T>& a) : BIT(a.size()) {
        int n = a.size();
        for (int i = 1; i <= n; i++) {
            t[i] = merge(t[i], a[i - 1]);
            int j = i + (i & -i);
            if (j <= n)
                t[j] = merge(t[j], t[i]);
        }
    }

    void add(int i, const T &x) {
```

```
        for (i += 1; i < t.size(); i += i & -i)
            t[i] = merge(t[i], x);
    }

    T posQuery(int i)  {
        T res = T();
        for (i += 1; i; i -= i & -i)
            res = add(res, t[i]);
        return res;
    }

    T rangeQuery(int l, int r)  { // [l, r]
        return posQuery(r) - posQuery(l - 1);
    }
};

template<typename T>
class TreeArray
{
private:
    int size;
    std::vector<T> arr;

public:
    TreeArray(int len) : size(len), arr(len + 1, 0) { }

    TreeArray(std::vector<T>& source) : size(source.size()), arr(size + 1, 0)
    {
        for (int i = 1; i <= size; i++) {
            add(i, source[i]);
        }
    }

    inline static int lowbit(int x) { return x & -x; }

    void add(int pos, T value)
    {
        while (pos <= size) {
            arr[pos] += value;
            pos += lowbit(pos);
        }
    }

    i64 query(int pos)
    {
        i64 sum = 0;
        while (pos >= 1) {
```

```
            sum += arr[pos];
            pos -= lowbit(pos);
        }
        return sum;
    }

    i64 query(int l, int r) { return query(r) - query(l - 1); }
};
```

实现区间查询区间修改

```
void solve()
{
    int N, M;
    std::cin >> N >> M;
    std::vector<int> diff(N + 1, 0);
    std::vector<i64> diff2(N + 1, 0);
    int last = 0;
    for (int i = 1; i <= N; i++) {
        int temp;
        std::cin >> temp;
        diff[i] = temp - last;
        diff2[i] = 1LL * (i - 1) * diff[i];
        last = temp;
    }

    BIT<int> tr1(diff);
    BIT<i64> tr2(diff2);
    // pre[i] = k*Σ(D[i]) - Σ((i-1)*D[i])
    while (M--) {
        int op;
        std::cin >> op;
        if (op == 1) {
            int x, y, k;
            std::cin >> x >> y >> k;
            tr1.add(x, k);
            tr1.add(y + 1, -k);
            tr2.add(x, 1LL * (x - 1) * k);
            tr2.add(y + 1, -1LL * y * k);
        } else if (op == 2) {
            int x, y;
            std::cin >> x >> y;
            i64 sum1 = 1LL * y * tr1.query(y) - tr2.query(y);
            i64 sum2 = 1LL * (x - 1) * tr1.query(x - 1) - tr2.query(x - 1);
            std::cout << sum1 - sum2 << std::endl;
        }
    }
```

```
}
```

3.2 ST 表

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

template<class T, class getFunc>
class ST
{
private:
    std::vector<std::vector<T>> dp;
    getFunc get = getFunc();

public:
    ST(const std::vector<T>& inputs) {
        size_t len = inputs.size();
        int exp = log2(len);
        // dp[s][k] 代表从 s 出发走 2^k 步内的最值
        dp.resize(len, std::vector<T>(exp + 1, 0));
        for (size_t s = 0; s < len; s++) {
            dp[s][0] = inputs[s];
        }

        for (int k = 1; k <= exp; k++) {
            for (size_t s = 0; s + (1 << k) <= len; s++) {
                dp[s][k] = get(dp[s][k - 1], dp[s + (1 << (k - 1))][k - 1]);
            }
        }
    }

    T query(size_t start, size_t end) {
        int exp = log2(end - start + 1);
        return get(dp[start][exp], dp[end - (1 << exp) + 1][exp]);
    }
};

int main() {
    struct GET {
        int operator()(int a, int b) {
            return std::min(a, b);
        }
    };
    ST<int, GET> st({12, 3, 4, -21, 2, 8});
    std::cout << st.query(0, 5) << std::endl;
}
```

3.3 线段树

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

using i64 = long long;

#define LOG(...) std::cerr << "Line[" << __LINE__ << "]: " << __VA_ARGS__ << std::endl;
#define LOGV(_vec, _size) std::cerr << #_vec << " = " << '['; for (int _i = 0; _i < (_size); _i++) { std::cerr << (_vec)[_i]; if (_i != (_size) - 1) std::cerr << ", "; } std::cerr << ']' << std::endl;

template<class InfoT>
struct SGT {
    std::vector<InfoT> tree;
    int size;

    inline int lson(int p) { return p << 1; } //* 左子节点

    inline int rson(int p) { return p << 1 | 1; } //* 右子节点

    void pull_up(int p) { //* 用子节点 Info 更新父节点
        tree[p] = tree[lson(p)] + tree[rson(p)];
    }

    template<class T>
    SGT(std::vector<T>& source) {
        size = source.size() - 1;
        int bottom = 1;
        while (bottom < size) bottom <<= 1;
        tree.resize(bottom << 1);
        auto build = [&](auto&& self, int p, int curL, int curR) -> void {
            if (curL == curR) {
                tree[p] = source[curL];
                return;
            }
            int mid = (curL + curR) >> 1;
            self(self, lson(p), curL, mid);
            self(self, rson(p), mid + 1, curR);
            pull_up(p);
        };
        build(build, 1, 1, size);
    }

    //* 区间查询
    InfoT query(int L, int R) {
        auto search = [&](auto&& self, int p, int curL, int curR) -> InfoT {
            if (L <= curL && curR <= R) return tree[p];
            InfoT res;
```

```
            int mid = (curL + curR) >> 1; //? [curL, mid], [mid+1, curR] 中一定有至少一个与 [L, R] 有交集
            bool flag = false;
            if (L <= mid) res = self(self, lson(p), curL, mid), flag = true;
            if (R >= mid + 1) res = flag ? res + self(self, rson(p), mid + 1, curR) : self(self, rson(p), mid + 1, curR);
            return res;
        };
        return search(search, 1, 1, size);
    }

    //* 单点修改
    void modify(int pos, const InfoT& newVal) {
        auto update = [&](auto&& self, int p, int curL, int curR) -> void {
            if (curR == curL && curR == pos) {
                tree[p] = newVal;
                return;
            }
            int mid = (curL + curR) >> 1;
            if (pos <= mid) self(self, lson(p), curL, mid);
            else self(self, rson(p), mid + 1, curR);
            pull_up(p);
        };
        update(update, 1, 1, size);
    }

    //* 单点修改（使用 Delta）
    template<class DeltaT>
    void modify(int pos, const DeltaT& delta) {
        auto update = [&](auto&& self, int p, int curL, int curR) -> void {
            if (curR == curL && curR == pos) {
                delta.applyTo(tree[p]);
                return;
            }
            int mid = (curL + curR) >> 1;
            if (pos <= mid) self(self, lson(p), curL, mid);
            else self(self, rson(p), mid + 1, curR);
            pull_up(p);
        };
        update(update, 1, 1, size);
    }
};

template<class InfoT, class TagT>
class LSGT {
public:
    struct Node {
```

```
90          InfoT info = InfoT();
91          TagT tag = TagT();
92      };
93      std::vector<Node> tree;
94      int size;
95
96      inline int lson(int p) { return p << 1; } //* 左子节点
97
98      inline int rson(int p) { return p << 1 | 1; } //* 右子节点
99
100     void addTag(int p, int curL, int curR, const TagT& delta) { //* 给代表 [l,r] 区间的
        ↪ Info 打 Tag
101         tree[p].tag.applyTo(tree[p].info, curL, curR, delta);
102     }
103
104     void pull_up(int p) { //* 用子节点 Info 更新父节点
105         tree[p].info = tree[lson(p)].info + tree[rson(p)].info;
106     }
107
108     void push_down(int p, int curL, int curR) { //* 下传 Tag
109         if (!tree[p].tag.isVaild()) return;
110         int mid = (curL + curR) >> 1;
111         addTag(lson(p), curL, mid, tree[p].tag);
112         addTag(rson(p), mid + 1, curR, tree[p].tag);
113         tree[p].tag = TagT();
114     }
115
116     template<class T>
117     LSGT(std::vector<T>& source) {
118         size = source.size() - 1;
119         int bottom = 1;
120         while (bottom < size) bottom <<= 1;
121         tree.resize(bottom << 1);
122         auto build = [&](auto&& self, int p, int curL, int curR) -> void {
123             if (curL == curR) {
124                 tree[p].info = source[curL];
125                 return;
126             }
127             int mid = (curL + curR) >> 1;
128             self(self, lson(p), curL, mid);
129             self(self, rson(p), mid + 1, curR);
130             pull_up(p);
131         };
132         build(build, 1, 1, size);
133     }
134
135     //* 区间查询
```

```
136     InfoT query(int L, int R) {
137         auto search = [&](auto&& self, int p, int curL, int curR) -> InfoT {
138             if (L <= curL && curR <= R) return tree[p].info;
139             push_down(p, curL, curR);
140             InfoT res;
141             int mid = (curL + curR) >> 1; //? [curL, mid], [mid+1, curR] 中一定有至少一个
                ↪  与 [L, R] 有交集
142             bool flag = false;
143             if (L <= mid) res = self(self, lson(p), curL, mid), flag = true;
144             if (R >= mid + 1) res = flag ? res + self(self, rson(p), mid + 1, curR) :
                ↪  self(self, rson(p), mid + 1, curR);
145             return res;
146         };
147         return search(search, 1, 1, size);
148     }
149
150     void modify(int L, int R, const TagT& delta) {
151         auto update = [&](auto&& self, int p, int curL, int curR) -> void {
152             if (L <= curL && curR <= R) {
153                 addTag(p, curL, curR, delta);
154                 return;
155             }
156             push_down(p, curL, curR);
157             int mid = (curL + curR) >> 1;
158             if (L <= mid) self(self, lson(p), curL, mid);
159             if (R >= mid + 1) self(self, rson(p), mid + 1, curR);
160             pull_up(p);
161         };
162         update(update, 1, 1, size);
163     }
164 };
165
166 struct Info {
167     i64 val;
168     Info() = default; //* 无参初始化（重置时候用）
169     Info(i64 v) : val(v) {} //? 从其他类型转换（Build 时候用）
170
171     friend Info operator+(const Info& a, const Info& b) { //* 合并操作，a 左 b 右
172         return Info(a.val + b.val);
173     }
174 };
175
176 //! Only for LSGT
177 struct Tag {
178     i64 add;
179     Tag() : add(0) {} //* 初始化（push_down 结束重置时候用）
180     Tag(i64 d) : add(d) {} //? 从其他类型转换（Modify 时候用）
```

```
181     bool isVaild() const { //* tag 是否生效
182         return add != 0;
183     }
184     void applyTo(Info& info, int l, int r, const Tag& delta) { //* 对代表 [l,r] 区间的
        ↪ info 打上 tag
185         add += delta.add;
186         info.val += delta.add * (r - l + 1);
187     }
188 };
189 
190 //! Only for SGT
191 struct Delta {
192     i64 val;
193     Delta(i64 v) : val(v) {}
194     void applyTo(Info& info) const {
195         info.val += val;
196     }
197 };
198 
199 
200 int main() {
201     std::vector<i64> test = {0, 1, 2, 4, 5, 7};
202     LSGT<Info, Tag> A(test);
203     std::cout << A.query(1, 3).val << std::endl;
204     std::cout << A.query(2, 5).val << std::endl;
205     std::cout << A.query(1, 5).val << std::endl;
206     A.modify(1, 3, Tag(1));
207     std::cout << A.query(1, 3).val << std::endl;
208     std::cout << A.query(2, 5).val << std::endl;
209     std::cout << A.query(1, 5).val << std::endl;
210 }
211 
212 /*struct SumInfo {
213     modint val;
214     SumInfo() : val(0) {} // 初始化（重置时候用）
215     SumInfo(modint v) : val(v) {} // 从其他类型转换（Build 时候用）
216     SumInfo mergeWith(const SumInfo& other) const { // 合并操作
217         return SumInfo(val + other.val);
218     }
219 };
220 
221 struct AddMulTag {
222     modint add, mul;
223     AddMulTag() : add(0), mul(1) {} // 初始化（重置时候用）
224     bool isVaild() const { // tag 是否生效
225         return add != modint(0) || mul != modint(1);
226     }
```

```
227     void applyTo(SumInfo& info, int l, int r, AddMulTag delta) { // 对代表 [l,r] 区间的
        ↪ info 打上 tag
228         mul *= delta.mul;
229         add *= delta.mul;
230         add += delta.add;
231         info.val *= delta.mul;
232         info.val += delta.add * (r - l + 1);
233     }
234 };*/
```

4 计算几何

4.1 Point

```
1  #include <bits/stdc++.h>
2
3  using i64 = long long;
4  using d64 = double;
5  using d128 = long double;
6
7  template<class T, class G> struct Vector2Base {
8      T x, y;
9      constexpr Vector2Base() : Vector2Base(T(), T()) {}
10     constexpr Vector2Base(T x, T y) : x(x), y(y) {}
11
12     //* 运算
13     constexpr Vector2Base operator+(Vector2Base a) const { return Vector2Base(x + a.x,
       ↪ y + a.y); }
14     constexpr Vector2Base operator-(Vector2Base a) const { return Vector2Base(x - a.x,
       ↪ y - a.y); }
15     constexpr Vector2Base operator-() const { return Vector2Base(-x, -y); }
16     constexpr G operator*(Vector2Base a) const { return (G)x * a.x + (G)y * a.y; }
17     constexpr G operator%(Vector2Base a) const { return (G)x * a.y - (G)y * a.x; }
18     constexpr Vector2Base rotate() const { return Vector2Base(-y, x); } //* 逆 90 度
19     template<class F> constexpr Vector2Base rotate(F theta) const {
20         Vector2Base b(std::cos(theta), std::sin(theta));
21         return Vector2Base(x * b.x - y * b.y, x * b.y + y * b.x);
22     }
23     constexpr friend Vector2Base operator*(const T& a, Vector2Base b) { return
       ↪ Vector2Base(a * b.x, a * b.y); }
24
25     //* 比较
26     constexpr bool operator<(Vector2Base a) const {
27         if (x == a.x) return y < a.y;
28         return x < a.x;
29     }
30     constexpr bool operator>(Vector2Base a) const {
```

```
        if (x == a.x) return y > a.y;
        return x > a.x;
    }
    constexpr bool operator<=(Vector2Base a) const {
        if (x == a.x) return y <= a.y;
        return x <= a.x;
    }
    constexpr bool operator>=(Vector2Base a) const {
        if (x == a.x) return y >= a.y;
        return x >= a.x;
    }
    constexpr bool operator==(Vector2Base a) const { return x == a.x && y == a.y; }
    constexpr bool operator!=(Vector2Base a) const { return x != a.x || y != a.y; }

    //* 输入输出
    friend std::istream& operator>>(std::istream& in, Vector2Base& p) { return in >>
    ↪  p.x >> p.y; }
    friend std::ostream& operator<<(std::ostream& ot, Vector2Base p) {
        return ot << '(' << p.x << ", " << p.y << ')';
    }
};

//* 点距平方
template<class T, class G> G dis2(Vector2Base<T, G> a, Vector2Base<T, G> b =
↪  Vector2Base<T, G>(0, 0)) {
    Vector2Base<T, G> p = a - b;
    return p * p;
}

using Vector2 = Vector2Base<int, i64>;

/* ================= */

//* 象限确认
int sgn(Vector2 p) {
    if (p.x < 0 or p.x == 0 and p.y > 0) { //? 23 象限 + 正 y 轴（左）
        return 1;
    } else {
        return 0; //? 14 象限 + 负 y 轴 + 原点（右）
    }
}

//* 按极角排序，4123 象限顺序
bool polarCmp(Vector2 p, Vector2 q) {
    if (sgn(p) == sgn(q)) {
        if (p % q == 0) {
            return dis2(p) < dis2(q); // 重合返回短的
```

```
        } else {
            return p % q > 0; // true 代表 p 在 q 顺时针方向
        }
    } else {
        return sgn(p) < sgn(q);
    }
}

//* 计算 ∠AOB(弧度)
template<class Float = d128> Float getAngle(Vector2 a, Vector2 b, Vector2 o = Vector2(0, 0)) {
    Vector2 A = a - o, B = b - o;
    return std::atan2<Float>(A % B, A * B);
}

//* 把角度缩到范围 [0,π]
template<class Float = d128> Float normAngle(Float angle) {
    angle = std::abs(angle);
    return std::min(angle, 2 * std::numbers::pi_v<Float> - angle);
}


//* 点距
template<class Float = d128> Float dis(Vector2 a, Vector2 b = Vector2(0, 0)) {
    return std::sqrt<Float>(dis2(a, b));
}

/* ============Line============== */
//* 点 A 是否在 * 直线 *BC 上
bool onLine(Vector2 a, Vector2 b, Vector2 c) {
    b = b - a;
    c = c - a;
    return b % c == 0;
}

//* 点 A 是否在 * 线段 *BC 上
bool onSeg(Vector2 a, Vector2 b, Vector2 c) {
    b = b - a;
    c = c - a;
    return b % c == 0 and b * c <= 0;
}

//* 点 A 到 * 直线 *BC 的距离
template<class Float = d128> Float disToLine(Vector2 a, Vector2 b, Vector2 c) {
    Vector2 v1 = b - c, v2 = a - c;
    return std::abs<Float>(v1 % v2) / std::sqrt<Float>(v1 * v1);
}
```

```cpp

//* 点 A 到 * 线段 *BC 的距离
template<class Float = d128> Float disToSeg(Vector2 a, Vector2 b, Vector2 c) {
    if ((a - b) * (c - b) <= 0 or (a - c) * (b - c) <= 0) {
        return std::min<Float>(dis<Float>(a, b), dis<Float>(a, c));
    }
    return disToLine<Float>(a, b, c);
}

//* 点 A 在直线 BC 上的投影点
Vector2 foot(Vector2 a, Vector2 b, Vector2 c) {
    Vector2 u = a - b, v = c - b;
    return b + (u * v) / (v * v) * v;
}

//* 点 A 关于直线 BC 的对称点
Vector2 symmetry(Vector2 a, Vector2 b, Vector2 c) {
    Vector2 ft = foot(a, b, c);
    return 2 * ft - a;
}

//* 直线 AB 和直线 CD 的交点
Vector2 cross(Vector2 a, Vector2 b, Vector2 c, Vector2 d) {
    Vector2 v = c - d;
    auto sa = v % (a - d), sb = (b - d) % v;
    return sa / (sa + sb) * (b - a) + a;
}

// 对四个不同的点判断四点共圆
// d 在 abc 外接圆外 return 1, 内 return -1
template<class Float = d128> int inCircle(Vector2 a, Vector2 b, Vector2 c, Vector2 d) {
    const Float PI = acosl(-1);
    Float ag1 = getAngle(a, b, c), ag2 = getAngle(d, c, b);
    auto sgn = [](Float x) { return x < 0 ? -1 : (x > 0); };
    if (sgn(ag1) == sgn(ag2)) {
        return sgn(PI - std::abs(ag1 + ag2));
    } else {
        return sgn(std::abs(ag1) - std::abs(ag2));
    }
}

/*====================*/

using Vertexs = std::vector<Vector2>;

//* Graham 法求凸包
Vertexs getConvexHull(Vertexs S) {
```

```cpp
    Vector2 base = *std::min_element(S.begin(), S.end()); // 以左下角为原点
    for (int i = 0; i < (int)S.size(); i++) {
        S[i] = S[i] - base;
    }
    std::sort(S.begin(), S.end(), polarCmp); // 按极角排序

    /*
    ? 不删共线点加上
    int e = S.size() - 1;
    while (e > 1) {
        if (S[e] % S.back() == 0) e--;
        else break;
    }
    std::reverse(S.begin() + e + 1, S.end());
    */

    std::deque<Vector2> Q;
    Q.push_back(S[0]);
    Q.push_back(S[1]);
    for (int i = 2; i < (int)S.size(); i++) {
        Vector2 B = Q.back(); Q.pop_back(); Vector2 A = Q.back(); // B 后 A 前
        while ((S[i] - B) % (B - A) >= 0) { //? 不删共线点就去掉等号
            B = A; // 删 A
            Q.pop_back();
            A = Q.back();
        }
        Q.push_back(B);
        Q.push_back(S[i]);
    }
    Vertexs ans(Q.size());
    int pos = 0;
    while (!Q.empty()) {
        ans[pos++] = Q.front() + base;
        Q.pop_front();
    }
    return ans;
}


//! test
int main() {
    int N;
    std::cin >> N;
    Vertexs vs(N);
    for (int i = 0; i < N; i++) {
        std::cin >> vs[i];
    }
```

```
    Vertexs hull = getConvexHull(vs);
    for (int i = 0; i < (int)hull.size(); i++) {
        std::cout << hull[i] << std::endl;
    }
}
```

5 数学相关

5.1 模运算

```
i64 add(i64 a, i64 b, i64 p) { // 加
    return (a % p + b % p) % p;
}

i64 sub(i64 a, i64 b, i64 p) { // 减
    return (a % p - b % p) % p;
}

i64 mul(i64 a, i64 b, i64 p) { // a > p 乘
    a %= p;
    b %= p;
    i64 ans = 0;
    while (b > 0) {
        if (b & 1) {
            ans += a;
            ans %= p;
        }
        a <<= 1;
        a %= p;
        b >>= 1;
    }
    return ans;
}
```

5.2 素数筛 & 欧拉函数

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

bool isPrime(int num)
{
    if (num == 1) return 0;
    if (num == 2 || num == 3) return 1;
    if (num % 6 != 1 && num % 6 != 5) return 0;
    int tmp = sqrt(num);
    for (int i = 5; i <= tmp; i += 6)
        if (num % i == 0 || num % (i + 2) == 0) return 0;
    return 1;
```

```cpp
}

//* 埃氏筛 O(NloglogN)
void E_sieve(int N) {
    std::vector<bool> isP(N + 1, true);
    for (int i = 2; i <= N; i++) {
        if (!isP[i]) continue;
        for (int j = i * i; j <= N; j += i) {
            isP[j] = false;
        }
    }
    isP[0] = isP[1] = false;
}

//* 线性欧拉筛 O(N)
void euler_sieve(int N) {
    std::vector<int> prime(N + 1); //? 可小于 N, 约 lnN
    std::vector<bool> vis(N + 1);
    int cnt = 0;
    for (int i = 2; i <= N; i++) {
        if (!vis[i]) prime[cnt++] = i;
        for (int j = 0; j < cnt; j++) {
            if (i * prime[j] > N) break;
            vis[i * prime[j]] = true;
            if (i % prime[j] == 0) break;
        }
    }
    for (int i = 0; i < static_cast<int>(prime.size()); i++) {
        if (!prime[i]) {
            prime.resize(i);
            break;
        }
    }
}

//* 预处理求欧拉函数
void euler_pre(int N) {
    std::vector<int> phi(N + 1);
    std::iota(phi.begin() + 1, phi.end(), 1);
    for (int i = 1; i <= N; i++) {
        for (int j = 2 * i; j <= N; j += i) {
            phi[j] -= phi[i];
        }
    }
}

int main() {
```

```
    euler_pre(11);
}
```

6 杂项

6.1 快速读入

```
inline int read()
{
    int x = 0, sgn = 1;
    char ch = getchar();
    while (ch < '0' || ch > '9') {
        if (ch == '-') sgn = -1;
        ch = getchar();
    }
    while (ch >= '0' && ch <= '9') {
        x = x * 10 + ch - '0';
        ch = getchar();
    }
    return x * sgn;
}
```