

算法竞赛模板

by ChaomengOrion

最后修改于[2025 年 3 月 28 日](#)

目录

1	前期准备	2
1.1	Cpp 文件一键编译测试	2
1.2	Cpp 模板	2
2	算法	3
2.1	快速幂	3
2.2	DP	3
2.2.1	LIS (最长上升子序列)	3
2.2.2	LCS (最长公共子序列)	4
2.2.3	01 背包	4
2.2.4	完全背包	5
2.2.5	多重背包	5
2.2.6	状压 dp	6
2.3	二维差分	7
2.4	并查集	8
2.5	图论	8
2.5.1	最短路 (Dijkstra)	8
2.5.2	最短路 (SPFA)	10
2.5.3	最小生成树	11
2.5.4	最近公共祖先 LCA	12
3	数据结构	13
3.1	树状数组	13
3.2	ST 表	16
3.3	线段树	17
4	计算几何	22
4.1	Point	22
5	数学相关	27
5.1	模运算	27
5.2	素数筛 & 欧拉函数	28
6	杂项	29
6.1	快速读入	29

1 前期准备

1.1 Cpp 文件一键编译测试

Windows - build.bat

```
1 @echo off
2 set exe=.\output\%~n1.exe
3 if not exist output mkdir output
4 g++ %1 -o %exe% -std=c++17 -Wall -Wextra -g3 -O2 -D_GLIBCXX_DEBUG && (
5     echo [build done to %exe%]
6     %exe%
7 )
```

1.2 Cpp 模板

注意题目是不是多组样例

```
1 #include <bits/stdc++.h>
2
3 #define LOG(...) std::cerr << "Line[" << __LINE__ << "]: " << __VA_ARGS__ << std::endl;
4 #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;
5
6 using i64 = long long;
7
8 void solve() {}
9
10 int main() {
11     std::cin.tie(nullptr)->sync_with_stdio(false);
12     int t; std::cin >> t; while (t--) solve();
13     return 0;
14 }
```

2 算法

2.1 快速幂

```
1 using i64 = long long;
2
3 i64 binpow(i64 a, i64 b) {
4     // a %= m;
5     i64 res = 1;
6     while (b > 0) {
7         if (b & 1) res = res * a; // % m;
8         a = a * a; // % m;
9         b >>= 1;
10     }
```

```
10     }
11     return res;
12 }
13
14 i64 f(i64 n, i64 m) {
15     // 求的是 m 个 n 相乘, 这里 n 是一个正整数
16     if (m == 0)
17         return 1;
18     else if (m == 1)
19         return n;
20     else if (m % 2 == 0)
21         return f(n * n, m / 2); // 偶数时的降幂
22     return f(n * n, m / 2) * n; // 奇数时的降幂
23 }
```

2.2 DP

2.2.1 LIS (最长上升子序列)

```
1  #include <bits/stdc++.h>
2
3  int main() {
4      std::vector<int> arr = {3, 1, 3, 4, 5, 3, 2};
5      int size = arr.size();
6      std::vector<int> low(size, INT_MAX);
7      int max_s = 0;
8      for (int i = 0; i < size; i++) {
9          int pos = std::lower_bound(low.begin(), low.end(), arr[i]) - low.begin();
10         low[pos] = arr[i];
11         max_s = std::max(max_s, pos + 1);
12     }
13     std::cout << max_s << std::endl;
14 }
```

2.2.2 LCS (最长公共子序列)

```
1  #include <bits/stdc++.h>
2
3  void solve() {
4      std::string S, T;
5      std::cin >> S >> T;
6      std::vector dp(S.size() + 1, std::vector<int>(T.size() + 1));
7      for (int i = 1; i <= S.size(); i++) {
8          for (int j = 1; j <= T.size(); j++) {
9              if (S[i - 1] == T[j - 1]) {
10                 dp[i][j] = dp[i - 1][j - 1] + 1;
11             } else {
```

```
12         dp[i][j] = std::max(dp[i - 1][j], dp[i][j - 1]);
13     }
14 }
15 }
16
17 std::string ans;
18 int i = S.size(), j = T.size();
19 while (i > 0 && j > 0) {
20     if (S[i - 1] == T[j - 1]) {
21         ans += S[i - 1];
22         i--; j--;
23     } else {
24         dp[i - 1][j] > dp[i][j - 1] ? i-- : j--;
25     }
26 }
27 std::reverse(ans.begin(), ans.end());
28 std::cout << ans << std::endl;
29 }
```

2.2.3 01 背包

```
1  #include <bits/stdc++.h>
2
3  void solve() {
4      int N, M;
5      std::cin >> N >> M;
6      std::vector<std::pair<int, int>> items(N + 1); // Wi Di
7      for (int i = 1; i <= N; i++) {
8          std::cin >> items[i].first >> items[i].second;
9      }
10     std::vector<int> dp(M + 1, 0), last(M + 1, 0);
11
12     for (int i = 1; i <= N; i++) {
13         for (int j = 0; j <= M; j++) {
14             if (j - items[i].first >= 0)
15                 dp[j] = std::max(last[j], last[j - items[i].first] + items[i].second);
16             else
17                 dp[j] = last[j];
18         }
19         std::swap(dp, last);
20         std::fill(dp.begin(), dp.end(), 0);
21     }
22
23     std::cout << last.back() << std::endl;
24 }
```

2.2.4 完全背包

```
1  #include <bits/stdc++.h>
2
3  using i64 = long long;
4
5  void solve()
6  {
7      int T, M;
8      std::cin >> T >> M; // T: max time cost, M: herbs count
9      std::vector<int> herbTimes(M), herbValues(M);
10     for (int i = 0; i < M; i++)
11         std::cin >> herbTimes[i] >> herbValues[i];
12     // dp[i][j] = max(dp[i-1][j], dp[i][j-w[i]]+v[i]) // j >= w[i]
13     std::vector dp(std::vector<i64>(T + 1, 0));
14     // dp[i][j] = max value of first i herbs with time j
15     for (int i = 1; i <= M; i++) {
16         for (int t = 0; t <= T; t++) {
17             if (t < herbTimes[i - 1])
18                 dp[t] = dp[t];
19             else
20                 dp[t] = std::max(dp[t], dp[t - herbTimes[i - 1]] + herbValues[i - 1]);
21         }
22     }
23     std::cout << dp.back() << std::endl;
24
25 }
```

2.2.5 多重背包

```
1  #include <bits/stdc++.h>
2
3  using i64 = long long;
4
5  void solve() {
6      int N, M;
7      std::cin >> N >> M;
8      std::vector<std::pair<int, int>> items; // weight, value
9
10     // 二进制拆分每种物品的数量
11     for (int i = 1; i <= N; i++) {
12         int V, W, m;
13         std::cin >> V >> W >> m;
14         int k = 1;
15         while(m > 0){
16             int cnt = std::min(k, m);
17             items.emplace_back(W * cnt, V * cnt);
18             m -= cnt;
19         }
20     }
21 }
```

```

19         k <= 1;
20     }
21 }
22
23 std::vector<int> dp(M + 1, 0);
24
25 // 0-1 背包标准遍历
26 for(auto &[weight, value] : items){
27     for(int j = M; j >= weight; j--){
28         dp[j] = std::max(dp[j], dp[j - weight] + value);
29     }
30 }
31
32 std::cout << dp[M] << std::endl;
33 }
34
35 int main(){
36     std::cin.tie(nullptr)->sync_with_stdio(false);
37     solve();
38     return 0;
39 }

```

2.2.6 状压 dp

K 国王

```

1 void solve() {
2     int N, K;
3     std::cin >> N >> K;
4     int limit = 1 << N;
5     std::vector dp(N + 1, std::vector(K + 1, std::vector<i64>(limit)));
6     dp[0][0][0] = 1;
7     for (int row = 0; row < N; row++) {
8         for (int k = 0; k <= K; k++) {
9             for (int st = 0; st < limit; st++) { // 枚举原状态
10                 if (dp[row][k][st] == 0) continue; // 跳过非法状态
11                 for (int nst = 0; nst < limit; nst++) { // 枚举新状态
12                     if ((nst << 1) & nst) continue; // 新的一行不能相邻
13                     if (nst & ((st << 1) | st | (st >> 1))) continue; // 与原来的一行也不
14                                     ↳ 能相邻
15                     int cnt = std::popcount<unsigned int>(nst); // 这个状态新增了多少个国王
16                     if (k + cnt > K) continue;
17                     dp[row + 1][k + cnt][nst] += dp[row][k][st]; // 下一行这个状态加上原来
18                                     ↳ 的方案数
19                 }
20             }
21         }
22     }
23 }

```

```

21     std::cout << std::accumulate(dp[N][K].begin(), dp[N][K].end(), 0LL) << '\n';
22 }

```

2.3 二维差分

```

1  void best()
2  {
3      int N, M; std::cin >> N >> M;
4      std::vector map(N + 1, std::vector<int>(N + 1, 0));
5      std::vector diff(N + 2, std::vector<int>(N + 2, 0));
6      for (int i = 1; i <= N; i++) for (int j = 1; j <= N; j++) std::cin >> map[i][j];
7
8      while (M--) {
9          int x1, x2, y1, y2;
10         std::cin >> x1 >> y1 >> x2 >> y2;
11         diff[x1][y1]++;
12         diff[x1][y2 + 1]--;
13         diff[x2 + 1][y1]--;
14         diff[x2 + 1][y2 + 1]++;
15     }
16     // 修改
17     // ###+@@@-
18     // ###@@@@#
19     // ###-###+
20     for (int i = 1; i <= N; i++)
21         for (int j = 1; j <= N; j++)
22             diff[i][j] += diff[i - 1][j];
23
24     for (int i = 1; i <= N; i++)
25         for (int j = 1; j <= N; j++)
26             diff[i][j] += diff[i][j - 1];
27
28     for (int i = 1; i <= N; i++) {
29         for (int j = 1; j <= N; j++) {
30             std::cout << diff[i][j] + map[i][j] << ' ';
31         }
32         std::cout << '\n';
33     }
34 }

```

2.4 并查集

```

1  #include <bits/stdc++.h>
2
3  struct DSU {
4      std::vector<int> fa;
5      std::vector<int> dis;

```

```

6
7     DSU(int size)
8     : fa(std::vector<int>(size + 1)),
9     dis(std::vector<int>(size + 1)) {
10         std::iota(fa.begin() + 1, fa.end(), 1);
11     }
12
13     int find(int x) {
14         if (fa[x] == x) return x;
15         int p = fa[x];
16         fa[x] = find(fa[p]);
17         dis[x] += dis[p];
18         return fa[x];
19     }
20
21     /* a.root -> b.root
22     void merge(int a, int b, int d) {
23         int ra = find(a), rb = find(b);
24         if (ra != rb) {
25             fa[ra] = rb;
26             dis[ra] = dis[b] - dis[a] + d; //? 四边形等式
27         }
28     }
29 };

```

2.5 图论

2.5.1 最短路 (Dijkstra)

```

1     #include <bits/stdc++.h>
2
3     struct node {
4         std::vector<std::pair<int, int>> to;
5         bool visit = false;
6         int dis = INT_MAX;
7     };
8
9     void solve()
10    {
11        int N, M, S; // 点 边 出发点
12        std::cin >> N >> M >> S;
13        std::vector<node> graph(N + 1);
14        for (int i = 0; i < M; i++) {
15            int u, v, len;
16            std::cin >> u >> v >> len;
17            graph[u].to.emplace_back(v, len);
18            //graph[v].to.emplace_back(u, len);

```



```

19     }
20
21     // <dis, id>
22     std::priority_queue<std::pair<int, int>, std::vector<std::pair<int, int>>,
    ↪ std::greater<std::pair<int, int>>> Q;
23     graph[S].dis = 0; Q.emplace(0, S); //! 记得
24     int cnt = 0;
25     while(!Q.empty() && cnt < N) {
26         auto [dis, x] = Q.top(); // 取出最近的
27         Q.pop();
28         if (graph[x].visit) continue; //? 可改成 cur.dis ≠ dis
29         graph[x].visit = true;
30         cnt++;
31         for (auto [son, cost] : graph[x].to) {
32             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
33             if (graph[son].dis > graph[x].dis + cost) { // 松弛边
34                 graph[son].dis = graph[x].dis + cost;
35                 Q.emplace(graph[x].dis + cost, son);
36             }
37         }
38     }
39
40     for (int i = 1; i <= N; i++) {
41         std::cout << graph[i].dis << ' ';
42     }
43 }
44
45 int main()
46 {
47     std::cin.tie(nullptr)->sync_with_stdio(false);
48     solve();
49     return 0;
50 }

```

2.5.2 最短路 (SPFA)

```

1  #include <bits/stdc++.h>
2
3  struct node {
4      std::vector<std::pair<int, int>> to;
5      int dis = INT_MAX;
6      bool inQueue = false; // 是否已经在 spfa 队列中
7      int cnt = 0; // 最后一次访问到该节点的 spfa 的轮数
8  };
9
10 void solve() {

```

```

11  int N, M;
12  std::cin >> N >> M;
13  std::vector<node> graph(N + 1);
14  for (int i = 0; i < M; i++) {
15      int u, v, c;
16      std::cin >> u >> v >> c;
17      if (c >= 0) {
18          graph[u].to.emplace_back(v, c);
19          graph[v].to.emplace_back(u, c);
20      } else {
21          graph[u].to.emplace_back(v, c);
22      }
23  }
24
25  /* SPFA
26  graph[1].cnt = 0, graph[1].dis = 0, graph[1].inQueue = true;
27  std::queue<int> Q;
28  Q.push(1);
29  while (!Q.empty()) {
30      int x = Q.front();
31      Q.pop();
32      graph[x].inQueue = false;
33      for (auto [son, cost] : graph[x].to) {
34          if (graph[x].dis + cost < graph[son].dis) { // 尝试松弛边，松弛成功就加入下一轮
35              graph[son].dis = graph[x].dis + cost;
36              graph[son].cnt = graph[x].cnt + 1;
37              if (graph[son].cnt >= N) {
38                  std::cout << "YES" << std::endl; // 存在负环
39                  return;
40              }
41              if (!graph[son].inQueue) {
42                  Q.emplace(son);
43                  graph[son].inQueue = true;
44              }
45          }
46      }
47  }
48
49  std::cout << "NO" << std::endl;
50  }

```

2.5.3 最小生成树

```

1  #include <bits/stdc++.h>
2
3  struct node {
4      std::vector<std::pair<int, int>> to;

```

```
5     bool visit = false;
6     int dis = INT_MAX;
7 };
8
9 void solve()
10 {
11     int N, M, S; // 点 边 出发点
12     std::cin >> N >> M >> S;
13     std::vector<node> graph(N + 1);
14     for (int i = 0; i < M; i++) {
15         int u, v, len;
16         std::cin >> u >> v >> len;
17         graph[u].to.emplace_back(v, len);
18         //graph[v].to.emplace_back(u, len);
19     }
20
21     // <dis, id>
22     std::priority_queue<std::pair<int, int>, std::vector<std::pair<int, int>>,
23         ↳ std::greater<std::pair<int, int>>> Q;
24     graph[S].dis = 0; Q.emplace(0, S); //! 记得
25     int cnt = 0;
26     while(!Q.empty() && cnt < N) {
27         auto [dis, x] = Q.top(); // 取出最近的
28         Q.pop();
29         if (graph[x].visit) continue; //? 可改成 cur.dis ≠ dis
30         graph[x].visit = true;
31         cnt++;
32         for (auto [son, cost] : graph[x].to) {
33             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
34             ↳ 不能删
35             if (graph[son].dis > graph[x].dis + cost) { // 松弛边
36                 graph[son].dis = graph[x].dis + cost;
37                 Q.emplace(graph[x].dis + cost, son);
38             }
39         }
40     }
41
42     for (int i = 1; i <= N; i++) {
43         std::cout << graph[i].dis << ' ';
44     }
45 }
46
47 int main()
48 {
49     std::cin.tie(nullptr)->sync_with_stdio(false);
50     solve();
51     return 0;
```

50 }

2.5.4 最近公共祖先 LCA

```

1  #include <bits/stdc++.h>
2
3  using i64 = long long;
4
5  struct node {
6      std::vector<int> to;
7      int depth = -1;
8  };
9
10 void solve() {
11     int N, Q, S;
12     std::cin >> N >> Q >> S;
13     std::vector<node> tree(N + 1);
14     for (int i = 0; i < N - 1; i++) {
15         int u, v;
16         std::cin >> u >> v;
17         tree[u].to.push_back(v);
18         tree[v].to.push_back(u);
19     }
20
21     std::array<std::vector<int>, 20> f;
22     std::fill(f.begin(), f.end(), std::vector<int>(N + 1, 0));
23
24     auto dfs_dep = [&](auto&& self, int x, int dep, int fa) -> void {
25         if (tree[x].depth != -1) return;
26         tree[x].depth = dep;
27         f[0][x] = fa;
28         for (int son : tree[x].to) {
29             self(self, son, dep + 1, x);
30         }
31     };
32
33     dfs_dep(dfs_dep, S, 1, 0);
34
35     for (int i = 1; i <= 19; i++) {
36         for (int j = 1; j <= N; j++) {
37             f[i][j] = f[i - 1][f[i - 1][j]];
38         }
39     }
40
41     auto get_lca = [&](int a, int b) -> int {
42         if (tree[a].depth > tree[b].depth) std::swap(a, b); // a <= b
43         int gap = tree[b].depth - tree[a].depth, cnt = 0;

```

```

44     while (gap) { //? 补齐到同一个高度
45         if (gap & 1) {
46             b = f[cnt][b];
47         }
48         cnt++;
49         gap >>= 1;
50     }
51     if (a == b) return a;
52     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
53         if (f[i][a] != f[i][b]) {
54             a = f[i][a];
55             b = f[i][b];
56         }
57     }
58     return f[0][a];
59 };
60
61 while (Q--) {
62     int a, b;
63     std::cin >> a >> b;
64     std::cout << get_lca(a, b) << std::endl;
65 }
66 }
67
68 int main() {
69     std::cin.tie(nullptr)->sync_with_stdio(false);
70     solve();
71     return 0;
72 }

```

3 数据结构

3.1 树状数组

```

1  #include <bits/stdc++.h>
2  using i64 = long long;
3
4  template<typename T>
5  class TreeArray
6  {
7  private:
8      int size;
9      std::vector<T> arr;
10
11 public:
12     TreeArray(int len) : size(len), arr(len + 1, 0) {}
13

```

```
14     TreeArray(std::vector<T>& source) : size(source.size()), arr(size + 1, 0)
15     {
16         for (int i = 1; i <= size; i++) {
17             add(i, source[i]);
18         }
19     }
20
21     inline static int lowbit(int x) { return x & -x; }
22
23     void add(int pos, T value)
24     {
25         while (pos <= size) {
26             arr[pos] += value;
27             pos += lowbit(pos);
28         }
29     }
30
31     i64 query(int pos)
32     {
33         i64 sum = 0;
34         while (pos >= 1) {
35             sum += arr[pos];
36             pos -= lowbit(pos);
37         }
38         return sum;
39     }
40
41     i64 query(int l, int r) { return query(r) - query(l - 1); }
42 };
43
44 const int MOD = 1e9+7;
45
46 template<typename T>
47 class TreeArrayMOD
48 {
49 private:
50     int size;
51     std::vector<T> arr;
52
53 public:
54     TreeArrayMOD(int len) : size(len), arr(len + 1, 0) { }
55
56     inline static int lowbit(int x) { return x & -x; }
57
58     void add(int pos, T value)
59     {
60         if (pos < 1) return;
```

```

61     value += MOD; value %= MOD;
62     while (pos <= size) {
63         arr[pos] += value;
64         arr[pos] %= MOD;
65         pos += lowbit(pos);
66     }
67 }
68
69 i64 query(int pos)
70 {
71     i64 sum = 0;
72     while (pos >= 1) {
73         sum += arr[pos];
74         sum %= MOD;
75         pos -= lowbit(pos);
76     }
77     return sum;
78 }
79
80 i64 query(int l, int r) { return (query(r) - query(l - 1) + MOD) % MOD; }
81
82 [[maybe_unused]] void debug() {
83     std::cerr << "BIT: [";
84     for (int i = 1; i <= size; i++) {
85         std::cerr << query(i, i) << ", ";
86     }
87     std::cerr << "]\n";
88 }
89 };

```

实现区间查询区间修改

```

1 void solve()
2 {
3     int N, M;
4     std::cin >> N >> M;
5     std::vector<int> diff(N + 1, 0);
6     std::vector<i64> diff2(N + 1, 0);
7     int last = 0;
8     for (int i = 1; i <= N; i++) {
9         int temp;
10        std::cin >> temp;
11        diff[i] = temp - last;
12        diff2[i] = 1LL * (i - 1) * diff[i];
13        last = temp;
14    }
15
16    BIT<int> tr1(diff);

```

```

17 BIT<i64> tr2(diff2);
18 // pre[i] = k*Σ(D[i]) - Σ((i-1)*D[i])
19 while (M--) {
20     int op;
21     std::cin >> op;
22     if (op == 1) {
23         int x, y, k;
24         std::cin >> x >> y >> k;
25         tr1.add(x, k);
26         tr1.add(y + 1, -k);
27         tr2.add(x, 1LL * (x - 1) * k);
28         tr2.add(y + 1, -1LL * y * k);
29     } else if (op == 2) {
30         int x, y;
31         std::cin >> x >> y;
32         i64 sum1 = 1LL * y * tr1.query(y) - tr2.query(y);
33         i64 sum2 = 1LL * (x - 1) * tr1.query(x - 1) - tr2.query(x - 1);
34         std::cout << sum1 - sum2 << std::endl;
35     }
36 }
37 }

```

3.2 ST 表

```

1  #include <bits/stdc++.h>
2
3  template<class T, class getFunc>
4  class ST
5  {
6  private:
7      std::vector<std::vector<T>> dp;
8      getFunc get = getFunc();
9
10 public:
11     ST(const std::vector<T>& inputs) {
12         size_t len = inputs.size();
13         int exp = log2(len);
14         // dp[s][k] 代表从 s 出发走 2^k 步内的最值
15         dp.resize(len, std::vector<T>(exp + 1, 0));
16         for (size_t s = 0; s < len; s++) {
17             dp[s][0] = inputs[s];
18         }
19
20         for (int k = 1; k <= exp; k++) {
21             for (size_t s = 0; s + (1 << k) <= len; s++) {
22                 dp[s][k] = get(dp[s][k - 1], dp[s + (1 << (k - 1))][k - 1]);
23             }
24         }
25     }
26 }

```



```

24     }
25 }
26
27 T query(size_t start, size_t end) {
28     int exp = log2(end - start + 1);
29     return get(dp[start][exp], dp[end - (1 << exp) + 1][exp]);
30 }
31 };
32
33 int main() {
34     struct GET {
35         int operator()(int a, int b) {
36             return std::min(a, b);
37         }
38     };
39     ST<int, GET> st({12, 3, 4, -21, 2, 8});
40     std::cout << st.query(0, 5) << std::endl;
41 }

```

3.3 线段树

```

1  #include <bits/stdc++.h>
2
3  using i64 = long long;
4
5  template<class InfoT>
6  struct SGT {
7      std::vector<InfoT> tree;
8      int size;
9
10     inline int lson(int p) { return p << 1; } /* 左子节点
11
12     inline int rson(int p) { return p << 1 | 1; } /* 右子节点
13
14     void pull_up(int p) { /* 用子节点 Info 更新父节点
15         tree[p] = tree[lson(p)] + tree[rson(p)];
16     }
17
18     template<class T>
19     SGT(std::vector<T>& source) {
20         size = source.size() - 1;
21         int bottom = 1;
22         while (bottom < size) bottom <= 1;
23         tree.resize(bottom << 1);
24         auto build = [&](auto&& self, int p, int curL, int curR) -> void {
25             if (curL == curR) {
26                 tree[p] = source[curL];

```

```

27         return;
28     }
29     int mid = (curL + curR) >> 1;
30     self(self, lson(p), curL, mid);
31     self(self, rson(p), mid + 1, curR);
32     pull_up(p);
33 };
34 build(build, 1, 1, size);
35 }
36
37 /* 区间查询
38 InfoT query(int L, int R) {
39     auto search = [&](auto&& self, int p, int curL, int curR) -> InfoT {
40         if (L <= curL && curR <= R) return tree[p];
41         InfoT res;
42         int mid = (curL + curR) >> 1; //? [curL, mid], [mid+1, curR] 中一定有至少一个
43         ↪ 与 [L, R] 有交集
44         bool flag = false;
45         if (L <= mid) res = self(self, lson(p), curL, mid), flag = true;
46         if (R >= mid + 1) res = flag ? res + self(self, rson(p), mid + 1, curR) :
47         ↪ self(self, rson(p), mid + 1, curR);
48         return res;
49     };
50     return search(search, 1, 1, size);
51 }
52
53 /* 单点修改
54 void modify(int pos, const InfoT& newVal) {
55     auto update = [&](auto&& self, int p, int curL, int curR) -> void {
56         if (curR == curL && curR == pos) {
57             tree[p] = newVal;
58             return;
59         }
60         int mid = (curL + curR) >> 1;
61         if (pos <= mid) self(self, lson(p), curL, mid);
62         else self(self, rson(p), mid + 1, curR);
63         pull_up(p);
64     };
65     update(update, 1, 1, size);
66 }
67
68 /* 单点修改 (使用 Delta)
69 template<class DeltaT>
70 void modify(int pos, const DeltaT& delta) {
71     auto update = [&](auto&& self, int p, int curL, int curR) -> void {
72         if (curR == curL && curR == pos) {
73             delta.applyTo(tree[p]);

```

```

72         return;
73     }
74     int mid = (curL + curR) >> 1;
75     if (pos <= mid) self(self, lson(p), curL, mid);
76     else self(self, rson(p), mid + 1, curR);
77     pull_up(p);
78 };
79 update(update, 1, 1, size);
80 }
81 };
82
83 template<class InfoT, class TagT>
84 class LSGT {
85 public:
86     struct Node {
87         InfoT info = InfoT();
88         TagT tag = TagT();
89     };
90     std::vector<Node> tree;
91     int size;
92
93     inline int lson(int p) { return p << 1; } /* 左子节点
94
95     inline int rson(int p) { return p << 1 | 1; } /* 右子节点
96
97     void addTag(int p, int curL, int curR, const TagT& delta) { /* 给代表 [l,r] 区间的
98         ↪ Info 打 Tag
99         tree[p].tag.applyTo(tree[p].info, curL, curR, delta);
100     }
101
102     void pull_up(int p) { /* 用子节点 Info 更新父节点
103         tree[p].info = tree[lson(p)].info + tree[rson(p)].info;
104     }
105
106     void push_down(int p, int curL, int curR) { /* 下传 Tag
107         if (!tree[p].tag.isVaild()) return;
108         int mid = (curL + curR) >> 1;
109         addTag(lson(p), curL, mid, tree[p].tag);
110         addTag(rson(p), mid + 1, curR, tree[p].tag);
111         tree[p].tag = TagT();
112     }
113
114     template<class T>
115     LSGT(std::vector<T>& source) {
116         size = source.size() - 1;
117         int bottom = 1;
118         while (bottom < size) bottom <= 1;

```

```

118     tree.resize(bottom << 1);
119     auto build = [&](auto&& self, int p, int curL, int curR) -> void {
120         if (curL == curR) {
121             tree[p].info = source[curL];
122             return;
123         }
124         int mid = (curL + curR) >> 1;
125         self(self, lson(p), curL, mid);
126         self(self, rson(p), mid + 1, curR);
127         pull_up(p);
128     };
129     build(build, 1, 1, size);
130 }
131
132 /* 区间查询
133 InfoT query(int L, int R) {
134     auto search = [&](auto&& self, int p, int curL, int curR) -> InfoT {
135         if (L <= curL && curR <= R) return tree[p].info;
136         push_down(p, curL, curR);
137         InfoT res;
138         int mid = (curL + curR) >> 1; //? [curL, mid], [mid+1, curR] 中一定有至少一个
            ↪ 与 [L, R] 有交集
139         bool flag = false;
140         if (L <= mid) res = self(self, lson(p), curL, mid), flag = true;
141         if (R >= mid + 1) res = flag ? res + self(self, rson(p), mid + 1, curR) :
            ↪ self(self, rson(p), mid + 1, curR);
142         return res;
143     };
144     return search(search, 1, 1, size);
145 }
146
147 void modify(int L, int R, const TagT& delta) {
148     auto update = [&](auto&& self, int p, int curL, int curR) -> void {
149         if (L <= curL && curR <= R) {
150             addTag(p, curL, curR, delta);
151             return;
152         }
153         push_down(p, curL, curR);
154         int mid = (curL + curR) >> 1;
155         if (L <= mid) self(self, lson(p), curL, mid);
156         if (R >= mid + 1) self(self, rson(p), mid + 1, curR);
157         pull_up(p);
158     };
159     update(update, 1, 1, size);
160 }
161 };
162

```

```

163 struct Info {
164     i64 val;
165     Info() = default; /* 无参初始化（重置时候用）
166     Info(i64 v) : val(v) {} /*? 从其他类型转换（Build 时候用）
167
168     friend Info operator+(const Info& a, const Info& b) { /* 合并操作，a 左 b 右
169         return Info(a.val + b.val);
170     }
171 };
172
173 //! Only for LSGT
174 struct Tag {
175     i64 add;
176     Tag() : add(0) {} /* 初始化（push_down 结束重置时候用）
177     Tag(i64 d) : add(d) {} /*? 从其他类型转换（Modify 时候用）
178     bool isValid() const { /* tag 是否生效
179         return add != 0;
180     }
181     void applyTo(Info& info, int l, int r, const Tag& delta) { /* 对代表 [l,r] 区间的
        ↪ info 打上 tag
182         add += delta.add;
183         info.val += delta.add * (r - l + 1);
184     }
185 };
186
187 //! Only for SGT
188 struct Delta {
189     i64 val;
190     Delta(i64 v) : val(v) {}
191     void applyTo(Info& info) const {
192         info.val += val;
193     }
194 };
195
196
197 int main() {
198     std::vector<i64> test = {0, 1, 2, 4, 5, 7};
199     LSGT<Info, Tag> A(test);
200     std::cout << A.query(1, 3).val << std::endl;
201     std::cout << A.query(2, 5).val << std::endl;
202     std::cout << A.query(1, 5).val << std::endl;
203     A.modify(1, 3, Tag(1));
204     std::cout << A.query(1, 3).val << std::endl;
205     std::cout << A.query(2, 5).val << std::endl;
206     std::cout << A.query(1, 5).val << std::endl;
207 }
208

```

```

209  /*struct SumInfo {
210      modint val;
211      SumInfo() : val(0) {} // 初始化（重置时候用）
212      SumInfo(modint v) : val(v) {} // 从其他类型转换（Build 时候用）
213      SumInfo mergeWith(const SumInfo& other) const { // 合并操作
214          return SumInfo(val + other.val);
215      }
216  };
217
218  struct AddMulTag {
219      modint add, mul;
220      AddMulTag() : add(0), mul(1) {} // 初始化（重置时候用）
221      bool isValid() const { // tag 是否生效
222          return add != modint(0) || mul != modint(1);
223      }
224      void applyTo(SumInfo& info, int l, int r, AddMulTag delta) { // 对代表 [l,r] 区间的
225          ↪ info 打上 tag
226          mul *= delta.mul;
227          add *= delta.mul;
228          add += delta.add;
229          info.val *= delta.mul;
230          info.val += delta.add * (r - l + 1);
231      }
232  };*/

```

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,
14     ↪ y + a.y); }
15     constexpr Vector2Base operator-(Vector2Base a) const { return Vector2Base(x - a.x,
16     ↪ y - a.y); }
17     constexpr Vector2Base operator-() const { return Vector2Base(-x, -y); }
18     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 {
31     if (x == a.x) return y > a.y;
32     return x > a.x;
33 }
34 constexpr bool operator<=(Vector2Base a) const {
35     if (x == a.x) return y <= a.y;
36     return x <= a.x;
37 }
38 constexpr bool operator>=(Vector2Base a) const {
39     if (x == a.x) return y >= a.y;
40     return x >= a.x;
41 }
42 constexpr bool operator==(Vector2Base a) const { return x == a.x && y == a.y; }
43 constexpr bool operator!=(Vector2Base a) const { return x != a.x || y != a.y; }
44
45 // * 输入输出
46 friend std::istream& operator>>(std::istream& in, Vector2Base& p) { return in >>
    ↪ p.x >> p.y; }
47 friend std::ostream& operator<<(std::ostream& ot, Vector2Base p) {
48     return ot << '(' << p.x << ", " << p.y << ')';
49 }
50 };
51
52 // * 点距平方
53 template<class T, class G> G dis2(Vector2Base<T, G> a, Vector2Base<T, G> b =
    ↪ Vector2Base<T, G>(0, 0)) {
54     Vector2Base<T, G> p = a - b;
55     return p * p;
56 }
57
58 using Vector2 = Vector2Base<int, i64>;
59
60 /* ===== */

```

```

61
62  /* 象限确认
63  int sgn(Vector2 p) {
64      if (p.x < 0 or p.x == 0 and p.y > 0) { //? 23 象限 + 正 y 轴 (左)
65          return 1;
66      } else {
67          return 0; //? 14 象限 + 负 y 轴 + 原点 (右)
68      }
69  }
70
71  /* 按极角排序, 4123 象限顺序
72  bool polarCmp(Vector2 p, Vector2 q) {
73      if (sgn(p) == sgn(q)) {
74          if (p % q == 0) {
75              return dis2(p) < dis2(q); // 重合返回短的
76          } else {
77              return p % q > 0; // true 代表 p 在 q 顺时针方向
78          }
79      } else {
80          return sgn(p) < sgn(q);
81      }
82  }
83
84  /* 计算 ∠AOB(弧度)
85  template<class Float = d128> Float getAngle(Vector2 a, Vector2 b, Vector2 o = Vector2(0,
↵ 0)) {
86      Vector2 A = a - o, B = b - o;
87      return std::atan2<Float>(A % B, A * B);
88  }
89
90  /* 把角度缩到范围 [0,π]
91  template<class Float = d128> Float normAngle(Float angle) {
92      angle = std::abs(angle);
93      return std::min(angle, 2 * std::numbers::pi_v<Float> - angle);
94  }
95
96
97  /* 点距
98  template<class Float = d128> Float dis(Vector2 a, Vector2 b = Vector2(0, 0)) {
99      return std::sqrt<Float>(dis2(a, b));
100  }
101
102  /* =====Line===== */
103  /* 点 A 是否在 * 直线 *BC 上
104  bool onLine(Vector2 a, Vector2 b, Vector2 c) {
105      b = b - a;
106      c = c - a;

```



```

107     return b % c == 0;
108 }
109
110 /* 点 A 是否在 * 线段 *BC 上
111 bool onSeg(Vector2 a, Vector2 b, Vector2 c) {
112     b = b - a;
113     c = c - a;
114     return b % c == 0 and b * c <= 0;
115 }
116
117 /* 点 A 到 * 直线 *BC 的距离
118 template<class Float = d128> Float disToLine(Vector2 a, Vector2 b, Vector2 c) {
119     Vector2 v1 = b - c, v2 = a - c;
120     return std::abs<Float>(v1 % v2) / std::sqrt<Float>(v1 * v1);
121 }
122
123 /* 点 A 到 * 线段 *BC 的距离
124 template<class Float = d128> Float disToSeg(Vector2 a, Vector2 b, Vector2 c) {
125     if ((a - b) * (c - b) <= 0 or (a - c) * (b - c) <= 0) {
126         return std::min<Float>(dis<Float>(a, b), dis<Float>(a, c));
127     }
128     return disToLine<Float>(a, b, c);
129 }
130
131 /* 点 A 在直线 BC 上的投影点
132 Vector2 foot(Vector2 a, Vector2 b, Vector2 c) {
133     Vector2 u = a - b, v = c - b;
134     return b + (u * v) / (v * v) * v;
135 }
136
137 /* 点 A 关于直线 BC 的对称点
138 Vector2 symmetry(Vector2 a, Vector2 b, Vector2 c) {
139     Vector2 ft = foot(a, b, c);
140     return 2 * ft - a;
141 }
142
143 /* 直线 AB 和直线 CD 的交点
144 Vector2 cross(Vector2 a, Vector2 b, Vector2 c, Vector2 d) {
145     Vector2 v = c - d;
146     auto sa = v % (a - d), sb = (b - d) % v;
147     return sa / (sa + sb) * (b - a) + a;
148 }
149
150 // 对四个不同的点判断四点共圆
151 // d 在 abc 外接圆外 return 1, 内 return -1
152 template<class Float = d128> int inCircle(Vector2 a, Vector2 b, Vector2 c, Vector2 d) {
153     const Float PI = acosl(-1);

```

```

154     Float ag1 = getAngle(a, b, c), ag2 = getAngle(d, c, b);
155     auto sgn = [](Float x) { return x < 0 ? -1 : (x > 0); };
156     if (sgn(ag1) == sgn(ag2)) {
157         return sgn(PI - std::abs(ag1 + ag2));
158     } else {
159         return sgn(std::abs(ag1) - std::abs(ag2));
160     }
161 }
162
163 /*=====*/
164
165 using Vextxs = std::vector<Vector2>;
166
167 /** Graham 法求凸包
168 Vextxs getConvexHull(Vextxs S) {
169     Vector2 base = *std::min_element(S.begin(), S.end()); // 以左下角为原点
170     for (int i = 0; i < (int)S.size(); i++) {
171         S[i] = S[i] - base;
172     }
173     std::sort(S.begin(), S.end(), polarCmp); // 按极角排序
174
175     /*
176     ? 不删共线点加上
177     int e = S.size() - 1;
178     while (e > 1) {
179         if (S[e] % S.back() == 0) e--;
180         else break;
181     }
182     std::reverse(S.begin() + e + 1, S.end());
183     */
184
185     std::deque<Vector2> Q;
186     Q.push_back(S[0]);
187     Q.push_back(S[1]);
188     for (int i = 2; i < (int)S.size(); i++) {
189         Vector2 B = Q.back(); Q.pop_back(); Vector2 A = Q.back(); // B 后 A 前
190         while ((S[i] - B) % (B - A) >= 0) { // ? 不删共线点就去掉等号
191             B = A; // 删 A
192             Q.pop_back();
193             A = Q.back();
194         }
195         Q.push_back(B);
196         Q.push_back(S[i]);
197     }
198     Vextxs ans(Q.size());
199     int pos = 0;
200     while (!Q.empty()) {

```

```
201         ans[pos++] = Q.front() + base;
202         Q.pop_front();
203     }
204     return ans;
205 }
206
207
208 //! test
209 int main() {
210     int N;
211     std::cin >> N;
212     Vertexs vs(N);
213     for (int i = 0; i < N; i++) {
214         std::cin >> vs[i];
215     }
216     Vertexs hull = getConvexHull(vs);
217     for (int i = 0; i < (int)hull.size(); i++) {
218         std::cout << hull[i] << std::endl;
219     }
220 }
```

5 数学相关

5.1 模运算

```
1  i64 add(i64 a, i64 b, i64 p) { // 加
2      return (a % p + b % p) % p;
3  }
4
5  i64 sub(i64 a, i64 b, i64 p) { // 减
6      return (a % p - b % p + p) % p;
7  }
8
9  i64 mul(i64 a, i64 b, i64 p) { // a > p 乘
10     a %= p; b %= p;
11     i64 ans = 0;
12     while (b > 0) {
13         if (b & 1) {
14             ans += a; ans %= p;
15         }
16         a <<= 1; a %= p;
17         b >>= 1;
18     }
19     return ans;
20 }
```

5.2 素数筛 & 欧拉函数

```
1  #include <bits/stdc++.h>
2
3  bool isPrime(int num)
4  {
5      if (num == 1) return 0;
6      if (num == 2 || num == 3) return 1;
7      if (num % 6 != 1 && num % 6 != 5) return 0;
8      int tmp = sqrt(num);
9      for (int i = 5; i <= tmp; i += 6)
10         if (num % i == 0 || num % (i + 2) == 0) return 0;
11     return 1;
12 }
13
14 /* 埃氏筛 O(NloglogN)
15 void E_sieve(int N) {
16     std::vector<bool> isP(N + 1, true);
17     for (int i = 2; i <= N; i++) {
18         if (!isP[i]) continue;
19         for (int j = i * i; j <= N; j += i) {
20             isP[j] = false;
21         }
22     }
23     isP[0] = isP[1] = false;
24 }
25
26 /* 线性欧拉筛 O(N)
27 void euler_sieve(int N) {
28     std::vector<int> prime(N + 1); //? 可小于 N, 约 lnN
29     std::vector<bool> vis(N + 1);
30     int cnt = 0;
31     for (int i = 2; i <= N; i++) {
32         if (!vis[i]) prime[cnt++] = i;
33         for (int j = 0; j < cnt; j++) {
34             if (i * prime[j] > N) break;
35             vis[i * prime[j]] = true;
36             if (i % prime[j] == 0) break;
37         }
38     }
39     for (int i = 0; i < static_cast<int>(prime.size()); i++) {
40         if (!prime[i]) {
41             prime.resize(i);
42             break;
43         }
44     }
45 }
```

```
47  /* 预处理求欧拉函数
48  void euler_pre(int N) {
49      std::vector<int> phi(N + 1);
50      std::iota(phi.begin() + 1, phi.end(), 1);
51      for (int i = 1; i <= N; i++) {
52          for (int j = 2 * i; j <= N; j += i) {
53              phi[j] -= phi[i];
54          }
55      }
56  }
57
58  int main() {
59      euler_pre(11);
60  }
```

6 杂项

6.1 快速读入

```
1  inline int read()
2  {
3      int x = 0, sgn = 1;
4      char ch = getchar();
5      while (ch < '0' || ch > '9') {
6          if (ch == '-') sgn = -1;
7          ch = getchar();
8      }
9      while (ch >= '0' && ch <= '9') {
10         x = x * 10 + ch - '0';
11         ch = getchar();
12     }
13     return x * sgn;
14 }
```