



## 算法竞赛模板

组织编写:

22 网络-1 肖建华

内容编写:

22 网络-1 肖建华

最后提交日期:

Monday 19<sup>th</sup> May, 2025

计算机学院  
2024 年春季学期

### 摘要

该模板由闽南师范大学 2022 级网络一班肖建华主编，同时欢迎同校师生共同维护，其目的为了方便分享以及打印，大部分模板引用于jiangly 算法模板收集。

如有需要，可自取，

望周知！

# 目录

<b>1</b>	<b>数据结构专题模板</b>	<b>1</b>
1.1	普通并查集 . . . . .	1
1.2	带权并查集 . . . . .	2
1.3	可撤销并查集 . . . . .	3
1.4	树状数组 . . . . .	5
1.5	二维树状数组 . . . . .	6
1.6	线段树 . . . . .	7
1.6.1	无 lazy . . . . .	7
1.6.2	有 lazy . . . . .	10
1.7	ST 表 . . . . .	14
<b>2</b>	<b>图论专题模板</b>	<b>15</b>
2.1	链式前向星建图 . . . . .	15
2.2	Dijkstra . . . . .	16
2.3	Spfa . . . . .	17
2.4	Floyd . . . . .	18
2.5	Kruskal . . . . .	19
2.6	二分图最大匹配 . . . . .	20
2.6.1	匈牙利算法 . . . . .	20
2.6.2	HK 算法 . . . . .	21
2.7	二分图判定 . . . . .	23
2.8	Kruskal 重构树 . . . . .	25
2.9	SCC 点-双连通分量 . . . . .	27
2.10	EBCC 边-双连通分量 . . . . .	29
2.11	MaxFlow . . . . .	31
2.12	Minflow . . . . .	34
2.13	可行流/最大流 . . . . .	37
2.14	TwoSat . . . . .	39
<b>3</b>	<b>树论专题模板</b>	<b>40</b>
3.1	BlockCutTree . . . . .	40
3.2	树的重心 . . . . .	42
3.3	树的直径 . . . . .	43
3.4	笛卡尔树 . . . . .	45
3.5	HLD . . . . .	46
3.6	树上差分 . . . . .	50
3.6.1	点差分 . . . . .	50

3.6.2	边差分 . . . . .	51
3.7	Splay . . . . .	51
<b>4</b>	<b>数学专题模板</b>	<b>57</b>
4.1	Exgcd . . . . .	57
4.2	Frac . . . . .	57
4.3	ModInt . . . . .	59
4.4	Sieve . . . . .	67
4.5	Comb(结合 ModInt) . . . . .	68
4.6	BigInt . . . . .	70
4.7	Miller-RabinAndPollard-Rho . . . . .	76
<b>5</b>	<b>字符串专题模板</b>	<b>78</b>
5.1	StringHash . . . . .	78
5.2	AC 自动机 . . . . .	80
5.3	马拉车 . . . . .	82
5.4	Z 函数 . . . . .	83
5.5	后缀数组 . . . . .	83
5.5.1	使用方法示例 . . . . .	85
5.6	回文自动机 . . . . .	86
5.7	KMP . . . . .	88
<b>6</b>	<b>算法杂项专题模板</b>	<b>91</b>
6.1	FastIO . . . . .	91
6.2	defs . . . . .	95
6.3	Int128 . . . . .	96
6.4	二分搜索 . . . . .	99
6.5	自定义哈希 . . . . .	100
<b>7</b>	<b>平面几何专题模板</b>	<b>101</b>
7.1	平面几何 . . . . .	101
7.2	线段在多边形内 . . . . .	110
7.3	钝角直角三角形计数问题 . . . . .	111
7.4	向量夹角 . . . . .	112
7.5	凸包上旋转卡尺算法的其他应用 . . . . .	112
7.6	最大最小三角形 . . . . .	114
7.7	动态凸包 . . . . .	115
7.8	闵可夫斯基和 . . . . .	117
7.9	半平面交 . . . . .	118

<b>8 常用 STL</b>	<b>119</b>
8.1 算法库 . . . . .	119
8.1.1 搜索操作 . . . . .	119
8.1.2 搜索操作 . . . . .	120
8.1.3 交换操作 . . . . .	121
8.1.4 生成操作 . . . . .	121
8.1.5 移除操作 . . . . .	121
8.1.6 顺序变更操作 . . . . .	121
8.1.7 划分操作 . . . . .	121
8.1.8 排序操作 . . . . .	121
8.1.9 二分搜索操作 . . . . .	121
8.1.10 集合操作（在已排序范围上） . . . . .	122
8.1.11 最小/最大操作 . . . . .	122
8.1.12 排列操作 . . . . .	122
8.1.13 数值运算 . . . . .	122

# 数据结构专题模板

## 1.1 普通并查集

```
1 struct DSU {
2     std::vector<int> f, siz;
3
4     DSU() {}
5     DSU(int n){
6         init(n);
7     }
8
9     void init(int n){
10         f.resize(n);
11         std::iota(f.begin(), f.end(), 0);
12         siz.assign(n, 1);
13     }
14
15     int find(int x){
16         while(x != f[x]){
17             x = f[x] = f[f[x]];
18         }
19         return x;
20     }
21
22     bool same(int x, int y){
23         return find(x) == find(y);
24     }
25
26     bool merge(int x, int y){
27         x = find(x);
28         y = find(y);
29         if(x == y){
30             return false;
31         }
32         siz[x] += siz[y];
33         f[y] = x;
34         return true;
35     }
36 }
```

```
35     }
36
37     int size(int x){
38         return siz[find(x)];
39     }
40
41     int operator[](const int x) {
42         return find(x);
43     }
44 };
```

## 1.2 带权并查集

```
1 struct WDSU {
2     std::vector<int> dist, f;
3
4     WDSU() {}
5
6     WDSU(int n) {
7         init(n);
8     }
9
10    void init(int n) {
11        f.resize(n);
12        std::iota(f.begin(), f.end(), 0);
13        dist.assign(n, 0);
14    }
15
16    int find(int x) {
17        while(x != f[x]) {
18            int tmp = f[x];
19            f[x] = find(tmp);
20            dist[x] += dist[tmp];
21        }
22        return f[x];
23    }
24
25    bool merge(int l, int r, int v) {
26        int lf = find(l), rf = find(r);
```

```
27         if(lf != rf) {
28             f[lf] = rf;
29             dist[lf] = v + dist[r] - dist[l];
30         }
31     }
32
33     int query(int l, int r) {
34         if(find(l) != find(r)) {
35             return -1;
36         }
37         return dist[l] - dist[r];
38     }
39     int operator[](const int x) {
40         return find(x);
41     }
42 };
```

### 1.3 可撤销并查集

```
1 struct UndoDSU {
2     std::vector<int> f, siz, rank;
3     std::stack<std::pair<int, int>> stk;
4
5     DSU() {}
6     DSU(int n){
7         init(n);
8     }
9
10    void init(int n){
11        f.resize(n);
12        std::iota(f.begin(), f.end(), 0);
13        siz.assign(n, 1);
14        rank.assign(n, 0);
15    }
16
17    int find(int x){
18        while(x != f[x]){
19            x = f[x];
20        }
```



```
21         return x;
22     }
23
24     bool same(int x,int y){
25         return find(x) == find(y);
26     }
27
28     bool merge(int x,int y){
29         x = find(x);
30         y = find(y);
31         if(x == y){
32             stk.push({-1, -1});
33             return false;
34         }
35         if(rank[x] > rank[y]) {
36             std::swap(x, y);
37         }
38
39         f[x] = y;
40         siz[y] += siz[x];
41         stk.push({x, 0});
42         if(rank[x] == rank[y]) {
43             rank[y]++;
44             stk.top().second = 1;
45         }
46
47     }
48
49     void rollback() {
50         auto v = stk.top();
51         stk.pop();
52         if(v.first == -1) {
53             return;
54         }
55         int x = v.first, y = f[x];
56         rank[y] -= v.second;
57         siz[y] -= siz[x];
58         f[x] = x;
59     }
```

```
60
61     int size(int x){
62         return siz[find(x)];
63     }
64     int operator[](const int x) {
65         return find(x);
66     }
67 };
```

## 1.4 树状数组

```
1  template <typename T>
2  struct Fenwick {
3      int n;
4      std::vector<T> a;
5
6      Fenwick(int n_ = 0) {
7          init(n_);
8      }
9
10     void init(int n_) {
11         n = n_;
12         a.assign(n, T{});
13     }
14
15     void add(int x, const T &v) {
16         for (int i = x + 1; i <= n; i += i & -i) {
17             a[i - 1] = a[i - 1] + v;
18         }
19     }
20
21     T sum(int x) {
22         T ans{};
23         for (int i = x; i > 0; i -= i & -i) {
24             ans = ans + a[i - 1];
25         }
26         return ans;
27     }
28 }
```

```

29     T rangeSum(int l, int r) {
30         return sum(r) - sum(l);
31     }
32
33     // first p, query(0, p) >= k
34     int select(const T &k) {
35         int x = 0;
36         T cur{};
37         for (int i = 1 << std::lg(n); i; i /= 2) {
38             if (x + i <= n && cur + a[x + i - 1] <= k) {
39                 x += i;
40                 cur = cur + a[x - 1];
41             }
42         }
43         return x;
44     }
45 };

```

## 1.5 二维树状数组

```

1 struct BIT_2D {
2     using T = long long;
3     int n, m;
4     vector<vector<T>> sum[4];
5     BIT_2D(int _n, int _m):n(_n), m(_m) {
6         for (int i=0; i<4; ++i) sum[i].assign(n+1, vector<T>(m
          +1, 0));
7     }
8     void add(int x, int y, T val) {
9         for (int i=x; i<=n; i+=i&-i) {
10             for (int j=y; j<=m; j+=j&-j) {
11                 sum[0][i][j]+=val;
12                 sum[1][i][j]+=val*x;
13                 sum[2][i][j]+=val*y;
14                 sum[3][i][j]+=val*x*y;
15             }
16         }
17     }
18     void range_add(int x1, int y1, int x2, int y2, T x) {

```

```

19         add(x1, y1, x);
20         add(x1, y2+1, -x);
21         add(x2+1, y1, -x);
22         add(x2+1, y2+1, x);
23     }
24     T ask(int x, int y) {
25         T res[4] = {};
26         for (int i=x; i>0; i-=i&-i)
27             for (int j=y; j>0; j-=j&-j)
28                 for (int k=0; k<4; ++k)
29                     res[k] += sum[k][i][j];
30         return (x+1)*(y+1)*res[0] - (y+1)*res[1] - (x+1)*res[2] + res
            [3];
31     }
32     T range_ask(int x1, int y1, int x2, int y2) {
33         return ask(x2, y2) - ask(x1-1, y2) - ask(x2, y1-1) + ask(x1
            -1, y1-1);
34     }
35 };

```

## 1.6 线段树

### 1.6.1 无 lazy

```

1  template<class Info>
2  struct SegmentTree {
3      int n;
4      std::vector<Info> info;
5      SegmentTree() : n(0) {}
6      SegmentTree(int n_, Info v_ = Info()) {
7          init(n_, v_);
8      }
9      template<class T>
10     SegmentTree(std::vector<T> init_) {
11         init(init_);
12     }
13     void init(int n_, Info v_ = Info()) {
14         init(std::vector(n_, v_));
15     }

```

```
16     template<class T>
17     void init(std::vector<T> init_) {
18         n = init_.size();
19         info.assign(4 << std::__lg(n), Info());
20         std::function<void(int, int, int)> build = [&](int p,
21             int l, int r) {
22             if (r - l == 1) {
23                 info[p] = init_[l];
24                 return;
25             }
26             int m = (l + r) / 2;
27             build(2 * p, l, m);
28             build(2 * p + 1, m, r);
29             pull(p);
30         };
31         build(1, 0, n);
32     }
33     void pull(int p) {
34         info[p] = info[2 * p] + info[2 * p + 1];
35     }
36     void modify(int p, int l, int r, int x, const Info &v) {
37         if (r - l == 1) {
38             info[p] = v;
39             return;
40         }
41         int m = (l + r) / 2;
42         if (x < m) {
43             modify(2 * p, l, m, x, v);
44         } else {
45             modify(2 * p + 1, m, r, x, v);
46         }
47         pull(p);
48     }
49     void modify(int p, const Info &v) {
50         modify(1, 0, n, p, v);
51     }
52     Info rangeQuery(int p, int l, int r, int x, int y) {
53         if (l >= y || r <= x) {
54             return Info();
55         }
56         int m = (l + r) / 2;
57         Info left = rangeQuery(2 * p, l, m, x, y);
58         Info right = rangeQuery(2 * p + 1, m, r, x, y);
59         return left + right;
60     }
61 }
```

```
54     }
55     if (l >= x && r <= y) {
56         return info[p];
57     }
58     int m = (l + r) / 2;
59     return rangeQuery(2 * p, l, m, x, y) \
60         + \
61         rangeQuery(2 * p + 1, m, r, x, y);
62 }
63 Info rangeQuery(int l, int r) {
64     return rangeQuery(1, 0, n, l, r);
65 }
66 template<class F>
67 int findFirst(int p, int l, int r, int x, int y, F &&pred)
68 {
69     if (l >= y || r <= x) {
70         return -1;
71     }
72     if (l >= x && r <= y && !pred(info[p])) {
73         return -1;
74     }
75     if (r - l == 1) {
76         return l;
77     }
78     int m = (l + r) / 2;
79     int res = findFirst(2 * p, l, m, x, y, pred);
80     if (res == -1) {
81         res = findFirst(2 * p + 1, m, r, x, y, pred);
82     }
83     return res;
84 }
85 template<class F>
86 int findFirst(int l, int r, F &&pred) {
87     return findFirst(1, 0, n, l, r, pred);
88 }
89 template<class F>
90 int findLast(int p, int l, int r, int x, int y, F &&pred) {
91     if (l >= y || r <= x) {
```

```
92     }
93     if (l >= x && r <= y && !pred(info[p])) {
94         return -1;
95     }
96     if (r - l == 1) {
97         return l;
98     }
99     int m = (l + r) / 2;
100    int res = findLast(2 * p + 1, m, r, x, y, pred);
101    if (res == -1) {
102        res = findLast(2 * p, l, m, x, y, pred);
103    }
104    return res;
105 }
106 template<class F>
107 int findLast(int l, int r, F &&pred) {
108     return findLast(1, 0, n, l, r, pred);
109 }
110 };
```

### 1.6.2 有 lazy

```
1  template<class Info, class Tag>
2  struct LazySegmentTree {
3      int n;
4      std::vector<Info> info;
5      std::vector<Tag> tag;
6      LazySegmentTree() : n(0) {}
7      LazySegmentTree(int n_, Info v_ = Info()) {
8          init(n_, v_);
9      }
10     template<class T>
11     LazySegmentTree(std::vector<T> init_) {
12         init(init_);
13     }
14     void init(int n_, Info v_ = Info()) {
15         init(std::vector(n_, v_));
16     }
17     template<class T>
```

```
18     void init(std::vector<T> init_) {
19         n = init_.size();
20         info.assign(4 << std::__lg(n), Info());
21         tag.assign(4 << std::__lg(n), Tag());
22         std::function<void(int, int, int)> build = [&](int p,
23             int l, int r) {
24             if (r - l == 1) {
25                 info[p] = init_[l];
26                 return;
27             }
28             int m = (l + r) / 2;
29             build(2 * p, l, m);
30             build(2 * p + 1, m, r);
31             pull(p);
32         };
33         build(1, 0, n);
34     }
35     void pull(int p) {
36         info[p] = info[2 * p] + info[2 * p + 1];
37     }
38     void apply(int p, const Tag &v) {
39         info[p].apply(v);
40         tag[p].apply(v);
41     }
42     void push(int p) {
43         apply(2 * p, tag[p]);
44         apply(2 * p + 1, tag[p]);
45         tag[p] = Tag();
46     }
47     void modify(int p, int l, int r, int x, const Info &v) {
48         if (r - l == 1) {
49             info[p] = v;
50             return;
51         }
52         int m = (l + r) / 2;
53         push(p);
54         if (x < m) {
55             modify(2 * p, l, m, x, v);
```



```
56         modify(2 * p + 1, m, r, x, v);
57     }
58     pull(p);
59 }
60 void modify(int p, const Info &v) {
61     modify(1, 0, n, p, v);
62 }
63 Info rangeQuery(int p, int l, int r, int x, int y) {
64     if (l >= y || r <= x) {
65         return Info();
66     }
67     if (l >= x && r <= y) {
68         return info[p];
69     }
70     int m = (l + r) / 2;
71     push(p);
72     return rangeQuery(2 * p, l, m, x, y) \
73         + \
74         rangeQuery(2 * p + 1, m, r, x, y);
75 }
76 Info rangeQuery(int l, int r) {
77     return rangeQuery(1, 0, n, l, r);
78 }
79 void rangeApply(int p, int l, int r, int x, int y, const
80     Tag &v) {
81     if (l >= y || r <= x) {
82         return;
83     }
84     if (l >= x && r <= y) {
85         apply(p, v);
86         return;
87     }
88     int m = (l + r) / 2;
89     push(p);
90     rangeApply(2 * p, l, m, x, y, v);
91     rangeApply(2 * p + 1, m, r, x, y, v);
92     pull(p);
93 }
94 void rangeApply(int l, int r, const Tag &v) {
```

```
94         return rangeApply(1, 0, n, l, r, v);
95     }
96     template<class F>
97     int findFirst(int p, int l, int r, int x, int y, F pred) {
98         if (l >= y || r <= x || !pred(info[p])) {
99             return -1;
100         }
101         if (r - l == 1) {
102             return l;
103         }
104         int m = (l + r) / 2;
105         push(p);
106         int res = findFirst(2 * p, l, m, x, y, pred);
107         if (res == -1) {
108             res = findFirst(2 * p + 1, m, r, x, y, pred);
109         }
110         return res;
111     }
112     template<class F>
113     int findFirst(int l, int r, F pred) {
114         return findFirst(1, 0, n, l, r, pred);
115     }
116     template<class F>
117     int findLast(int p, int l, int r, int x, int y, F pred) {
118         if (l >= y || r <= x || !pred(info[p])) {
119             return -1;
120         }
121         if (r - l == 1) {
122             return l;
123         }
124         int m = (l + r) / 2;
125         push(p);
126         int res = findLast(2 * p + 1, m, r, x, y, pred);
127         if (res == -1) {
128             res = findLast(2 * p, l, m, x, y, pred);
129         }
130         return res;
131     }
132     template<class F>
```

```
133     int findLast(int l, int r, F pred) {
134         return findLast(1, 0, n, l, r, pred);
135     }
136 };
```

## 1.7 ST 表

```
1 // vector<int> a(n + 1);
2 template<typename T>
3 class SparseTable {
4 public:
5     SparseTable() = default;
6
7     explicit SparseTable(const std::vector<T>& data)
8     {
9         Initialize(data);
10    }
11
12    void Initialize(const std::vector<T>& data) {
13        this->n = data.size() - 1;
14
15        log_table.resize(n + 1);
16        log_table[0] = -1;
17        for(int i = 1; i <= n; i++) {
18            log_table[i] = log_table[i >> 1] + 1;
19        }
20
21        st_table.resize(n + 1, std::vector<int>(21));
22
23        for(int i = 1; i <= n; i++) {
24            st_table[i][0] = data[i];
25        }
26
27        for(int p = 1; p <= log_table[n]; p++) {
28            for(int i = 1; i + (1 << p) - 1 <= n; i++) {
29                st_table[i][p] = op(st_table[i][p - 1],
30                                    st_table[i + (1 << (p - 1))][p - 1]);
31            }
32        }
```

```
33
34     }
35
36     T Query(size_t left, size_t right) {
37         const int k = log_table[right - left + 1];
38         return op(st_table[left][k], st_table[right - (1 << k)
39                 + 1][k]);
40     }
41 private:
42     int n;
43     std::vector<int> log_table;
44     std::vector<std::vector<T>> st_table;
45
46     T op(const T& lv, const T& rv) const {
47         return std::max(lv, rv);
48     }
49 };
```

## 图论专题模板

### 2.1 链式前向星建图

```
1  const int N = 2e6 + 10;
2  struct Enode {
3      int next, to, w;
4  }edges[N];
5  int head[N], cnt = 0;
6
7  void init() {
8      fill(head, head + N, -1);
9      cnt = 0;
10 }
11
12 void addEdge(int u, int v, int w = 1) {
13     edges[cnt].to = v;
14     edges[cnt].w = w;
15     edges[cnt].next = head[u];
```

```
16     head[u] = cnt++;
17 }
```

## 2.2 Dijkstra

```
1  struct Dijkstra {    // index-base-0
2      using i64 = long long;
3      int n;
4      std::vector<std::vector<std::pair<int, i64>>> adj;
5      std::vector<i64> dis;
6      std::vector<bool> vis;
7      Dijkstra(int n) {
8          init(n);
9      }
10
11     void init(int n) {
12         this->n = n;
13         adj.assign(n, {});
14         vis.assign(n, false);
15         dis.assign(n, 1e18);
16     }
17
18     void add_edge(int u, int v, i64 d) {
19         adj[u].push_back({v, d});
20         adj[v].push_back({u, d});
21     }
22
23
24     struct node {
25         i64 dis;
26         int pos;
27         bool operator <(const node &x) const {
28             return x.dis < dis;
29         }
30     };
31
32     std::priority_queue<node> q;
33     void work(int r) {
34         dis[r] = 0;
```

```
35         q.push((node){0, r});
36
37         while(!q.empty()) {
38             node tmp = q.top();
39             q.pop();
40             int u = tmp.pos;
41             i64 d = tmp.dis;
42
43             if (vis[u]) {
44                 continue;
45             }
46
47             vis[u] = true;
48             for(auto [v, w] : adj[u]) {
49                 if(dis[v] > dis[u] + w) {
50                     dis[v] = dis[u] + w;
51                     if(!vis[v]) {
52                         q.push((node){dis[v], v});
53                     }
54                 }
55             }
56         }
57     }
58 };
```

## 2.3 Spfa

```
1  template<class T = long long>
2  struct Spfa {    // index-base-0
3      static constexpr T inf = std::numeric_limits<T>::max() / 2;
4
5      int n;
6      std::vector<std::vector<std::pair<int, T>>> g;
7      std::vector<T> dis;
8
9      Spfa() {}
10
11     Spfa(int n) : n(n), g(n) {}
12 }
```

```

13     void add_Edge(int u, int v, T w) {
14         g[u].emplace_back({v, w});
15     }
16
17     bool work(int root = 0) {
18         std::vector<bool> vis(n, false);
19         std::vector<int> cnt(n, 0);
20         dis.assign(n, inf);
21         dis[root] = 0;
22         // 最长路
23         // dis.assign(n, -inf);
24         std::queue<int> q;
25         q.push(root);
26         vis[root] = true;
27         while(q.size()) {
28             auto u = q.front();
29             q.pop();
30             vis[u] = false;
31             for(auto [v, w] : g[u]) {
32                 if(dis[v] > dis[u] + w) { // 最长路 ->  $v < u + w$ 
33                     dis[v] = dis[u] + w;
34                     if((cnt[v] = cnt[u] + 1) >= n) {
35                         return false;
36                     }
37                     if(!vis[v]) {
38                         q.push(v);
39                         vis[v] = true;
40                     }
41                 }
42             }
43         }
44         return true;
45     }
46 };

```

## 2.4 Floyd

```

1  const int inf = 1e9;

```

```

2 void floyd(std::vector<std::vector<int>>& dis) {
3     int n = dis.size();
4     for(int k=0; k<n; ++k)
5         for(int i=0; i<n; ++i)
6             for(int j=0; j<n; ++j) {
7                 if(i!=j && i!=k && j!=k
8                     && dis[i][k] < inf && dis[k][j] < inf // 边权有
                        负数必加
9                     && dis[i][j] > dis[i][k] + dis[k][j])
10                    dis[i][j] = dis[i][k]+dis[k][j];
11            }
12 }
13 // 同时记录方案数
14 void floyd(std::vector<std::vector<int>>& dis, std::vector<std
    ::vector<Z>>& f) {
15     int n = dis.size();
16     for(int k=0; k<n; ++k)
17         for(int i=0; i<n; ++i)
18             for(int j=0; j<n; ++j)
19                 if(i!=j && i!=k && j!=k) {
20                     if(dis[i][j] > dis[i][k] + dis[k][j]) {
21                         dis[i][j] = dis[i][k] + dis[k][j];
22                         f[i][i] = f[i][k] * f[k][j];
23                     } else if(dis[i][j] == dis[i][k] + dis[k][j]
24                         ) {
25                         f[i][j] += f[i][k] * f[k][j];
26                     }
27 }

```

## 2.5 Kruskal

```

1 template<class T>
2 struct KruskalMst {      // index-base-0
3     int n;
4     std::vector<int> f;
5     std::vector<std::tuple<T, int, int>> e;
6     KruskalMst(int n) {
7         this->n = n;

```



```
8         f.resize(n);
9         std::iota(f.begin(), f.end(), 0);
10    }
11    void addEdge(int u, int v, T w) {
12        e.emplace_back(w, u, v);
13    }
14    int find(int u) {
15        return f[u] == u ? u : f[u] = find(f[u]);
16    }
17    T work() {
18        std::sort(e.begin(), e.end());
19        T ans = 0;
20        int cnt = 0;
21        for(auto [w, u, v] : e) {
22            u = find(u);
23            v = find(v);
24            if(u == v) {
25                continue;
26            }
27            f[u] = v;
28            ans += w;
29            if(++cnt == n - 1) {
30                break;
31            }
32        }
33        // assert(cnt==n-1);
34        return ans;
35    }
36 };
```

## 2.6 二分图最大匹配

### 2.6.1 匈牙利算法

```
1 struct BipartiteGraph {      // index-base-0
2     int n, m;
3     std::vector<std::vector<int>>> g;
4     std::vector<int> vis, link;
5     BipartiteGraph(int n, int m)
```

```

6      :n(n),m(m),g(n),vis(m),link(m) {}
7
8      void addEdge(int u, int v) { // left->right
9          g[u].push_back(v);
10     }
11     bool find(int u) { // 左边u -> 右边v
12         for(int v: g[u]) {
13             if(vis[v]) {
14                 continue;
15             }
16             vis[v] = 1;
17             if(link[v] == -1 || find(link[v])) {
18                 link[v] = u;
19                 return true;
20             }
21         }
22         return false;
23     }
24     int maxMatching() {
25         int cnt = 0;
26         std::fill_n(link.begin(), m, -1);
27         for(int i = 0; i < n; ++i) {
28             if(!find(i)) {
29                 continue;
30             }
31             std::fill_n(vis.begin(), m, 0);
32             ++cnt;
33         }
34         return cnt;
35     }
36 };

```

### 2.6.2 HK 算法

```

1 struct HopcroftKarp { // index-base-0
2     static constexpr int INF = 0x3f3f3f3f;
3     int n, m, dis;
4     std::vector<std::vector<int>> e;
5     std::vector<int> matchX, matchY, dx, dy;

```

```
6      std::vector<bool> used;
7      HopcroftKarp(int n, int m)
8          :n(n), m(m), e(n), matchX(n), matchY(m), dx(n), dy(m), used
9              (m) {}
10     void addEdge(int u, int v) {
11         e[u].push_back(v);
12     }
13     bool searchP() {
14         std::fill(dx.begin(), dx.end(), -1);
15         std::fill(dy.begin(), dy.end(), -1);
16         dis = INF;
17         std::queue<int> q;
18         for(int i = 0; i < n; ++i) {
19             if(matchX[i] == -1) {
20                 q.push(i), dx[i] = 0;
21             }
22         }
23         while(!q.empty()) {
24             int u = q.front();
25             q.pop();
26             if(dx[u] > dis) {
27                 break;
28             }
29             for(int v : e[u])
30                 if(dy[v] == -1) {
31                     dy[v] = dx[u] + 1;
32                     if(matchY[v] == -1) {
33                         dis = dy[v];
34                     }
35                     else {
36                         dx[matchY[v]] = dy[v] + 1;
37                         q.push(matchY[v]);
38                     }
39                 }
40             return dis != INF;
41         }
42     bool dfs(int u) {
43         for(int v : e[u]) {
```

```

44         if(used[v] || dy[v] != dx[u] + 1) {
45             continue;
46         }
47         used[v] = true;
48         if(matchY[v] != -1 && dy[v] == dis) {
49             continue;
50         }
51         if(matchY[v] == -1 || dfs(matchY[v])) {
52             matchY[v] = u;
53             matchX[u] = v;
54             return true;
55         }
56     }
57     return false;
58 }
59 int maxMatching() {
60     int res = 0;
61     std::fill(matchX.begin(), matchX.end(), -1);
62     std::fill(matchY.begin(), matchY.end(), -1);
63     while(searchP()) {
64         std::fill(used.begin(), used.end(), false);
65         for(int i = 0; i < n; ++i) {
66             if(matchX[i] == -1 && dfs(i)) {
67                 ++res;
68             }
69         }
70     }
71     return res;
72 }
73 };

```

## 2.7 二分图判定

```

1 struct JudgeBG {    // index-base-0
2     int n;
3     std::vector<int> bel;    // bel:1 or 2
4     std::vector<std::vector<int>> g;
5     JudgeBG(int n): n(n),g(n),bel(n) {}
6     void addEdge(int u, int v) {

```

```
7         g[u].push_back(v);
8         g[v].push_back(u);
9     }
10    bool dfs(int u, int color = 1) {
11        if(bel[u]) {
12            return bel[u]==color;
13        }
14        bel[u] = color;
15        for(int v : g[u])
16            if(!dfs(v, 3-color)) {
17                return false;
18            }
19        return true;
20    }
21    bool paint() { // 二分图染色
22        for(int i = 0; i < n; ++i) {
23            if(bel[i]) {
24                continue;
25            }
26            if(!dfs(i)) {
27                return false;
28            }
29        }
30        return true;
31    }
32    // std::vector<int> to; //[0,ln), [0,rn)
33    // // use it to divide the graph if paint()==true
34    // BipartiteGraph reLabel() { // left->right
35    //     int ln = 0, rn = 0;
36    //     to.resize(n);
37    //     for(int i = 0; i < n; ++i) {
38    //         if(bel[i] == 1) {
39    //             to[i] = ln++;
40    //         }
41    //         else {
42    //             to[i] = rn++;
43    //         }
44    //     }
45    //     BipartiteGraph bg(ln, rn);
```

```

46      //      for(int i = 0; i < n; ++i) {
47      //          for(int j : g[i]) {
48      //              if(bel[i] == 1) {
49      //                  bg.addEdge(to[i], to[j]);
50      //              }
51      //          }
52      //      }
53      //      return bg;
54      //  }
55  };

```

## 2.8 Kruskal 重构树

```

1  namespace krt {      // index-1
2      constexpr int MAXN = 2e5 + 10, MAXH = 21;
3      constexpr bool ASC = true;  // 边权从小到大排序
4      int father[MAXN], head[MAXN], next[MAXN], to[MAXN];
5      int nodekey[MAXN], dep[MAXN], stjump[MAXN][MAXH];
6      int /*dfn[MAXN], */seg[MAXN], in[MAXN], out[MAXN];
7      int cntu, cntg, cntd;
8      int find(int x) {
9          return x == father[x] ? x : father[x] = find(father[x])
10         ;
11     }
12     void addEdge(int u, int v) {
13         next[++cntg] = head[u];
14         to[cntg] = v;
15         head[u] = cntg;
16     }
17
18     int lca(int x, int y) {
19         if(dep[x] < dep[y]) {
20             std::swap(x, y);
21         }
22         for(int p = MAXH - 1; p >= 0; p--) {
23             if(dep[stjump[x][p]] >= dep[y]) {
24                 x = stjump[x][p];
25             }

```

```
26     }
27     if(x == y) {
28         return x;
29     }
30     for(int p = MAXH - 1; p >= 0; p--) {
31         if(stjump[x][p] != stjump[y][p]) {
32             x = stjump[x][p];
33             y = stjump[y][p];
34         }
35     }
36     return stjump[x][0];
37 }
38
39 void build(int n, std::vector<std::tuple<int, int, int>> e)
40 {
41     for(int i = 1; i <= n; i++) {
42         father[i] = i;
43     }
44     if constexpr (ASC) {
45         std::sort(e.begin(), e.end());
46     } else {
47         std::sort(e.begin(), e.end(), std::greater());
48     }
49     cntu = n;
50     for(auto& [w, u, v] : e) {
51         int fx = find(u);
52         int fy = find(v);
53         if(fx != fy) {
54             father[fx] = father[fy] = ++cntu;
55             father[cntu] = cntu;
56             nodekey[cntu] = w;
57             addEdge(cntu, fx);
58             addEdge(cntu, fy);
59         }
60     }
61     std::function<void(int, int)> dfs1 = [&](int u, int fa)
62     -> void {
63         dep[u] = dep[fa] + 1;
64         in[u] = ++cntd;
```

```

63         // dfn[u] = ++cntd;
64         seg[cntd] = u;
65         stjump[u][0] = fa;
66         for(int i = 1; i < MAXH; i++) {
67             stjump[u][i] = stjump[stjump[u][i - 1]][i - 1];
68         }
69         for(int e = head[u]; e > 0; e = next[e]) {
70             dfs1(to[e], u);
71         }
72         out[u] = cntd;
73     };
74     for(int i = 1; i <= cntu; i++) {
75         if(i == father[i]) {
76             dfs1(i, 0);
77         }
78     }
79 }
80 // 在 lim 的限制下最高能跳到哪一个节点
81 int jumpUp(int u, int lim) {
82     for(int i = MAXH - 1; i >= 0; --i) {
83         if constexpr (ASC) {
84             if(stjump[i][u] and nodekey[stjump[i][u]] <= lim)
85                 u = stjump[i][u];
86         } else {
87             if(stjump[i][u] and nodekey[stjump[i][u]] >= lim)
88                 u = stjump[i][u];
89         }
90     }
91     return u;
92 }
93 }
94 }
95 }

```

## 2.9 SCC 点-双连通分量

```
1 // index-base-0
```



```
2 struct SCC {
3     int n;
4     std::vector<std::vector<int>> adj;
5     std::vector<int> stk;
6     std::vector<int> dfn, low, bel;
7     int cur, cnt;
8
9     SCC() {}
10    SCC(int n) {
11        init(n);
12    }
13
14    void init(int n) {
15        this->n = n;
16        adj.assign(n, {});
17        dfn.assign(n, -1);
18        low.resize(n);
19        bel.assign(n, -1);
20        stk.clear();
21        cur = cnt = 0;
22    }
23
24    void addEdge(int u, int v) {
25        adj[u].push_back(v);
26    }
27
28    void dfs(int x) {
29        dfn[x] = low[x] = cur++;
30        stk.push_back(x);
31
32        for (auto y : adj[x]) {
33            if (dfn[y] == -1) {
34                dfs(y);
35                low[x] = std::min(low[x], low[y]);
36            } else if (bel[y] == -1) {
37                low[x] = std::min(low[x], dfn[y]);
38            }
39        }
40    }
```

```
41         if (dfn[x] == low[x]) {
42             int y;
43             do {
44                 y = stk.back();
45                 bel[y] = cnt;
46                 stk.pop_back();
47             } while (y != x);
48             cnt++;
49         }
50     }
51
52     std::vector<int> work() {
53         for (int i = 0; i < n; i++) {
54             if (dfn[i] == -1) {
55                 dfs(i);
56             }
57         }
58         return bel;
59     }
60 };
```

## 2.10 EBCC 边-双连通分量

```
1  std::set<std::pair<int, int>>> E;
2
3  struct EBCC {
4      int n;
5      std::vector<std::vector<int>>> adj;
6      std::vector<int> stk;
7      std::vector<int> dfn, low, bel;
8      int cur, cnt;
9
10     EBCC() {}
11     EBCC(int n) {
12         init(n);
13     }
14
15     void init(int n) {
16         this->n = n;
```

```
17         adj.assign(n, {});
18         dfn.assign(n, -1);
19         low.resize(n);
20         bel.assign(n, -1);
21         stk.clear();
22         cur = cnt = 0;
23     }
24
25     void addEdge(int u, int v) {
26         adj[u].push_back(v);
27         adj[v].push_back(u);
28     }
29
30     void dfs(int x, int p) {
31         dfn[x] = low[x] = cur++;
32         stk.push_back(x);
33
34         for (auto y : adj[x]) {
35             if (y == p) {
36                 continue;
37             }
38             if (dfn[y] == -1) {
39                 E.emplace(x, y);
40                 dfs(y, x);
41                 low[x] = std::min(low[x], low[y]);
42             } else if (bel[y] == -1 && dfn[y] < dfn[x]) {
43                 E.emplace(x, y);
44                 low[x] = std::min(low[x], dfn[y]);
45             }
46         }
47
48         if (dfn[x] == low[x]) {
49             int y;
50             do {
51                 y = stk.back();
52                 bel[y] = cnt;
53                 stk.pop_back();
54             } while (y != x);
55             cnt++;
```

```
56     }
57 }
58
59 std::vector<int> work() {
60     dfs(0, -1);
61     return bel;
62 }
63
64 struct Graph {
65     int n;
66     std::vector<std::pair<int, int>> edges;
67     std::vector<int> siz;
68     std::vector<int> cnte;
69 };
70 Graph compress() {
71     Graph g;
72     g.n = cnt;
73     g.siz.resize(cnt);
74     g.cnte.resize(cnt);
75     for (int i = 0; i < n; i++) {
76         g.siz[bel[i]]++;
77         for (auto j : adj[i]) {
78             if (bel[i] < bel[j]) {
79                 g.edges.emplace_back(bel[i], bel[j]);
80             } else if (i < j) {
81                 g.cnte[bel[i]]++;
82             }
83         }
84     }
85     return g;
86 }
87 };
```

## 2.11 MaxFlow

```
1 template<class T>
2 struct MaxFlow {
3     struct _Edge {
4         int to;
```

```
5         T cap;
6         _Edge(int to, T cap) : to(to), cap(cap) {}
7     };
8     int n;
9     std::vector<_Edge> e;
10    std::vector<std::vector<int>> g;
11    std::vector<int> cur, h;
12    MaxFlow() {}
13    MaxFlow(int n) {
14        init(n);
15    }
16    void init(int n) {
17        this->n = n;
18        e.clear();
19        g.assign(n, {});
20        cur.resize(n);
21        h.resize(n);
22    }
23    bool bfs(int s, int t) {
24        h.assign(n, -1);
25        std::queue<int> que;
26        h[s] = 0;
27        que.push(s);
28        while (!que.empty()) {
29            const int u = que.front();
30            que.pop();
31            for (int i : g[u]) {
32                auto [v, c] = e[i];
33                if (c > 0 && h[v] == -1) {
34                    h[v] = h[u] + 1;
35                    if (v == t) {
36                        return true;
37                    }
38                    que.push(v);
39                }
40            }
41        }
42        return false;
43    }
```

```
44     T dfs(int u, int t, T f) {
45         if (u == t) {
46             return f;
47         }
48         auto r = f;
49         for (int &i = cur[u]; i < int(g[u].size()); ++i) {
50             const int j = g[u][i];
51             auto [v, c] = e[j];
52             if (c > 0 && h[v] == h[u] + 1) {
53                 auto a = dfs(v, t, std::min(r, c));
54                 e[j].cap -= a;
55                 e[j ^ 1].cap += a;
56                 r -= a;
57                 if (r == 0) {
58                     return f;
59                 }
60             }
61         }
62         return f - r;
63     }
64     void addEdge(int u, int v, T c) {
65         g[u].push_back(e.size());
66         e.emplace_back(v, c);
67         g[v].push_back(e.size());
68         e.emplace_back(u, 0);
69     }
70     T flow(int s, int t) {
71         T ans = 0;
72         while (bfs(s, t)) {
73             cur.assign(n, 0);
74             ans += dfs(s, t, std::numeric_limits<T>::max());
75         }
76         return ans;
77     }
78     std::vector<bool> minCut() {
79         std::vector<bool> c(n);
80         for (int i = 0; i < n; i++) {
81             c[i] = (h[i] != -1);
82         }
```

```
83         return c;
84     }
85
86
87     struct Edge {
88         int from;
89         int to;
90         T cap;
91         T flow;
92     };
93     std::vector<Edge> edges() {
94         std::vector<Edge> a;
95         for (int i = 0; i < e.size(); i += 2) {
96             Edge x;
97             x.from = e[i + 1].to;
98             x.to = e[i].to;
99             x.cap = e[i].cap + e[i + 1].cap;
100             x.flow = e[i + 1].cap;
101             a.push_back(x);
102         }
103         return a;
104     }
105 };
```

## 2.12 Minflow

```
1  template<class T>
2  struct MinCostFlow {
3      struct _Edge {
4          int to;
5          T cap;
6          T cost;
7          _Edge(int to_, T cap_, T cost_) : to(to_), cap(cap_),
8              cost(cost_) {}
9      };
10     int n;
11     std::vector<_Edge> e;
12     std::vector<std::vector<int>>> g;
13     std::vector<T> h, dis;
```

```
13     std::vector<int> pre;
14     bool dijkstra(int s, int t) {
15         dis.assign(n, std::numeric_limits<T>::max());
16         pre.assign(n, -1);
17         std::priority_queue<std::pair<T, int>, std::vector<std
            ::pair<T, int>>, std::greater<std::pair<T, int>>>
            que;
18         dis[s] = 0;
19         que.emplace(0, s);
20         while (!que.empty()) {
21             T d = que.top().first;
22             int u = que.top().second;
23             que.pop();
24             if (dis[u] != d) {
25                 continue;
26             }
27             for (int i : g[u]) {
28                 int v = e[i].to;
29                 T cap = e[i].cap;
30                 T cost = e[i].cost;
31                 if (cap > 0 && dis[v] > d + h[u] - h[v] + cost)
32                     {
33                         dis[v] = d + h[u] - h[v] + cost;
34                         pre[v] = i;
35                         que.emplace(dis[v], v);
36                     }
37             }
38             return dis[t] != std::numeric_limits<T>::max();
39         }
40         MinCostFlow() {}
41         MinCostFlow(int n_) {
42             init(n_);
43         }
44         void init(int n_) {
45             n = n_;
46             e.clear();
47             g.assign(n, {});
48         }
```



```
49     void addEdge(int u, int v, T cap, T cost) {
50         g[u].push_back(e.size());
51         e.emplace_back(v, cap, cost);
52         g[v].push_back(e.size());
53         e.emplace_back(u, 0, -cost);
54     }
55     std::pair<T, T> flow(int s, int t) {
56         T flow = 0;
57         T cost = 0;
58         h.assign(n, 0);
59         while (dijkstra(s, t)) {
60             for (int i = 0; i < n; ++i) {
61                 h[i] += dis[i];
62             }
63             T aug = std::numeric_limits<int>::max();
64             for (int i = t; i != s; i = e[pre[i] ^ 1].to) {
65                 aug = std::min(aug, e[pre[i]].cap);
66             }
67             for (int i = t; i != s; i = e[pre[i] ^ 1].to) {
68                 e[pre[i]].cap -= aug;
69                 e[pre[i] ^ 1].cap += aug;
70             }
71             flow += aug;
72             cost += aug * h[t];
73         }
74         return std::make_pair(flow, cost);
75     }
76     struct Edge {
77         int from;
78         int to;
79         T cap;
80         T cost;
81         T flow;
82     };
83     std::vector<Edge> edges() {
84         std::vector<Edge> a;
85         for (int i = 0; i < e.size(); i += 2) {
86             Edge x;
87             x.from = e[i + 1].to;
```

```
88         x.to = e[i].to;
89         x.cap = e[i].cap + e[i + 1].cap;
90         x.cost = e[i].cost;
91         x.flow = e[i + 1].cap;
92         a.push_back(x);
93     }
94     return a;
95 }
96 };
```

## 2.13 可行流/最大流

```
1  struct MCFGraph {
2      struct Edge {
3          int v, c, f;
4          Edge(int v, int c, int f) : v(v), c(c), f(f) {}
5      };
6      const int n;
7      std::vector<Edge> e;
8      std::vector<std::vector<int>>> g;
9      std::vector<i64> h, dis;
10     std::vector<int> pre;
11     bool dijkstra(int s, int t) {
12         dis.assign(n, std::numeric_limits<i64>::max());
13         pre.assign(n, -1);
14         std::priority_queue<std::pair<i64, int>, std::vector<
15             std::pair<i64, int>>, std::greater<std::pair<i64,
16                 int>>> que;
17         dis[s] = 0;
18         que.emplace(0, s);
19         while (!que.empty()) {
20             i64 d = que.top().first;
21             int u = que.top().second;
22             que.pop();
23             if (dis[u] < d) continue;
24             for (int i : g[u]) {
25                 int v = e[i].v;
```

```
26         if (c > 0 && dis[v] > d + h[u] - h[v] + f) {
27             dis[v] = d + h[u] - h[v] + f;
28             pre[v] = i;
29             que.emplace(dis[v], v);
30         }
31     }
32 }
33 return dis[t] != std::numeric_limits<i64>::max();
34 }
35 MCFGGraph(int n) : n(n), g(n) {}
36 //可行流
37 void addEdge(int u, int v, int c, int f) {
38     if (f < 0) {
39         g[u].push_back(e.size());
40         e.emplace_back(v, 0, f);
41         g[v].push_back(e.size());
42         e.emplace_back(u, c, -f);
43     } else {
44         g[u].push_back(e.size());
45         e.emplace_back(v, c, f);
46         g[v].push_back(e.size());
47         e.emplace_back(u, 0, -f);
48     }
49 }
50 //最大流
51 /*void addEdge(int u, int v, int c, int f) {
52     g[u].push_back(e.size());
53     e.emplace_back(v, c, f);
54     g[v].push_back(e.size());
55     e.emplace_back(u, 0, -f);
56 }*/
57 std::pair<int, i64> flow(int s, int t) {
58     int flow = 0;
59     i64 cost = 0;
60     h.assign(n, 0);
61     while (dijkstra(s, t)) {
62         for (int i = 0; i < n; ++i) h[i] += dis[i];
63         int aug = std::numeric_limits<int>::max();
```

```

64         for (int i = t; i != s; i = e[pre[i] ^ 1].v) aug =
            std::min(aug, e[pre[i]].c);
65         for (int i = t; i != s; i = e[pre[i] ^ 1].v) {
66             e[pre[i]].c -= aug;
67             e[pre[i] ^ 1].c += aug;
68         }
69         flow += aug;
70         cost += i64(aug) * h[t];
71     }
72     return std::make_pair(flow, cost);
73 }
74 };

```

## 2.14 TwoSat

```

1  struct TwoSat {
2      int n;
3      std::vector<std::vector<int>> e;
4      std::vector<bool> ans;
5      TwoSat(int n) : n(n), e(2 * n), ans(n) {}
6      void addClause(int u, bool f, int v, bool g) {
7          e[2 * u + !f].push_back(2 * v + g);
8          e[2 * v + !g].push_back(2 * u + f);
9      }
10     bool satisfiable() {
11         std::vector<int> id(2 * n, -1), dfn(2 * n, -1), low(2 *
            n, -1);
12         std::vector<int> stk;
13         int now = 0, cnt = 0;
14         std::function<void(int)> tarjan = [&](int u) {
15             stk.push_back(u);
16             dfn[u] = low[u] = now++;
17             for (auto v : e[u]) {
18                 if (dfn[v] == -1) {
19                     tarjan(v);
20                     low[u] = std::min(low[u], low[v]);
21                 } else if (id[v] == -1) {
22                     low[u] = std::min(low[u], dfn[v]);
23                 }

```

```

24         }
25         if (dfn[u] == low[u]) {
26             int v;
27             do {
28                 v = stk.back();
29                 stk.pop_back();
30                 id[v] = cnt;
31             } while (v != u);
32             ++cnt;
33         }
34     };
35     for (int i = 0; i < 2 * n; ++i) if (dfn[i] == -1)
36         tarjan(i);
37     for (int i = 0; i < n; ++i) {
38         if (id[2 * i] == id[2 * i + 1]) return false;
39         ans[i] = id[2 * i] > id[2 * i + 1];
40     }
41     return true;
42 }
43 std::vector<bool> answer() { return ans; }
44 };

```

## 树论专题模板

### 3.1 BlockCutTree

```

1 struct BlockCutTree {
2     int n;
3     std::vector<std::vector<int>>> adj;
4     std::vector<int> dfn, low, stk;
5     int cnt, cur;
6     std::vector<std::pair<int, int>>> edges;
7
8     BlockCutTree() {}
9     BlockCutTree(int n) {
10         init(n);
11     }
12

```

```
13     void init(int n) {
14         this->n = n;
15         adj.assign(n, {});
16         dfn.assign(n, -1);
17         low.resize(n);
18         stk.clear();
19         cnt = cur = 0;
20         edges.clear();
21     }
22
23     void addEdge(int u, int v) {
24         adj[u].push_back(v);
25         adj[v].push_back(u);
26     }
27
28     void dfs(int x) {
29         stk.push_back(x);
30         dfn[x] = low[x] = cur++;
31
32         for (auto y : adj[x]) {
33             if (dfn[y] == -1) {
34                 dfs(y);
35                 low[x] = std::min(low[x], low[y]);
36                 if (low[y] == dfn[x]) {
37                     int v;
38                     do {
39                         v = stk.back();
40                         stk.pop_back();
41                         edges.emplace_back(n + cnt, v);
42                     } while (v != y);
43                     edges.emplace_back(x, n + cnt);
44                     cnt++;
45                 }
46             } else {
47                 low[x] = std::min(low[x], dfn[y]);
48             }
49         }
50     }
51
```

```
52     std::pair<int, std::vector<std::pair<int, int>>> work() {
53         for (int i = 0; i < n; i++) {
54             if (dfn[i] == -1) {
55                 stk.clear();
56                 dfs(i);
57             }
58         }
59         return {cnt, edges};
60     }
61 };
```

### 3.2 树的重心

```
1  struct CodeOfTree {
2      int n, cur;
3      std::vector<std::vector<int>> adj;
4      std::vector<int> siz, dep, p, in, ord;
5      CodeOfTree() = default;
6
7      CodeOfTree(int _n) {
8          init(_n);
9      }
10
11     void init(int _n) {
12         this->n = _n;
13         cur = 0;
14         adj.assign(n, {});
15         siz.resize(n);
16         dep.resize(n);
17         p.resize(n);
18         in.resize(n);
19         ord.resize(n);
20     }
21
22     void addEdge(int u, int v) {
23         adj[u].push_back(v);
24         adj[v].push_back(u);
25     }
26 }
```

```
27     void dfs(int u) {
28         siz[u] = 1;
29         in[u] = cur++;
30         ord[in[u]] = u;
31         for(auto v : adj[u]) {
32             if(v == p[u]) {
33                 continue;
34             }
35             p[v] = u;
36             dep[v] = dep[u] + 1;
37             dfs(v);
38             siz[u] += siz[v];
39         }
40     }
41
42     int find(int u) {
43         for(auto v : adj[u]) {
44             if(v == p[u] || 2 * siz[v] <= n) {
45                 continue;
46             }
47             return find(v);
48         }
49         return u;
50     }
51
52     void work() {
53         p[0] = -1;
54         dfs(0);
55         int rt = find(0);
56         dep[rt] = 0;
57         p[rt] = -1;
58         cur = 0;
59         dfs(rt);
60     }
61 };
```

### 3.3 树的直径

```
1 struct Diameter {
```



```
2     int n, start, end;
3     std::vector<std::vector<int>>> adj;
4     std::vector<int> pa;
5
6     Diameter() = default;
7     Diameter(int _n) {
8         init(_n);
9     }
10
11    void init(int _n) {
12        this->n = _n;
13        start = end = -1;
14        adj.assign(n, {});
15        pa.resize(n);
16        std::iota(pa.begin(), pa.end(), 0);
17    }
18
19    void addEdge(int u, int v) {
20        adj[u].push_back(v);
21        adj[v].push_back(u);
22    }
23
24    std::pair<int, int> dfs(int u, int fa, std::vector<bool>&
        vis) {
25        std::pair<int, int> ans{1, u};
26        pa[u] = fa;
27        vis[u] = true;
28        for(auto v : adj[u]) {
29            if(v == fa || vis[v]) {
30                continue;
31            }
32            auto pi = dfs(v, u, vis);
33            pi.first += 1;
34            ans = max(ans, pi);
35        }
36        return ans;
37    }
38
39    std::pair<int, int> work() {
```

```
40         std::vector<bool> vis(n);
41         auto [d1, ss] = dfs(0, -1, vis);
42         auto [d2, ee] = dfs(ss, -1, vis);
43         start = ss, end = ee;
44         return std::make_pair(start, end);
45     }
46
47     /* default = end -> start */
48     std::vector<int> getPath() {
49         std::vector<int> v;
50         int cur = end;
51         while(cur != -1) {
52             v.push_back(cur);
53             cur = pa[cur];
54         }
55         // std::reverse(v.begin(), v.end());
56         return v;
57     }
58 };
```

### 3.4 笛卡尔树

```
1  template<class T>
2  struct DescartesTree {
3
4      std::vector<int> lch, rch, stk;
5
6      DescartesTree() {}
7
8      DescartesTree(const std::vector<T>& v) {
9          work(v);
10     }
11
12     void work(const std::vector<T>& v) {
13         int n = v.size();
14         lch.resize(n, n);
15         rch.resize(n, n);
16         for(int i = 0; i < n; i++) {
17             while(!stk.empty() && v[i] < v[stk.back()]) {
```

```
18         rch[stk.back()] = lch[i];
19         lch[i] = stk.back();
20         stk.pop_back();
21     }
22     stk.push_back(i);
23 }
24 while(stk.size() > 1) {
25     int x = stk.back();
26     stk.pop_back();
27     rch[stk.back()] = x;
28 }
29 }
30 };
```

### 3.5 HLD

```
1 struct HLD {
2     int n;
3     std::vector<int> siz, top, dep, parent, in, out, seq, son;
4     std::vector<std::vector<int>> adj;
5     int cur;
6
7     HLD() {}
8     HLD(int n) {
9         init(n);
10    }
11    void init(int n) {
12        this->n = n;
13        siz.resize(n);
14        top.resize(n);
15        dep.resize(n);
16        parent.resize(n);
17        in.resize(n);
18        out.resize(n);
19        seq.resize(n);
20        son.resize(n);
21        cur = 0;
22        adj.assign(n, {});
23    }
```

```
24     void addEdge(int u, int v) {
25         adj[u].push_back(v);
26         adj[v].push_back(u);
27     }
28     void work(int root = 0) {
29         top[root] = root;
30         dep[root] = 0;
31         parent[root] = -1;
32         dfs1(root);
33         dfs2(root);
34     }
35     void dfs1(int u) {
36         if (parent[u] != -1) {
37             adj[u].erase(std::find(adj[u].begin(), adj[u].end()
38                                     , parent[u]));
39
40             siz[u] = 1;
41             for (auto &v : adj[u]) {
42                 parent[v] = u;
43                 dep[v] = dep[u] + 1;
44                 dfs1(v);
45                 siz[u] += siz[v];
46                 if (siz[v] > siz[adj[u][0]]) {
47                     son[u] = v;
48                     std::swap(v, adj[u][0]);
49                 }
50             }
51         }
52         void dfs2(int u) {
53             in[u] = cur++;
54             seq[in[u]] = u;
55             for (auto v : adj[u]) {
56                 top[v] = v == adj[u][0] ? top[u] : v;
57                 dfs2(v);
58             }
59             out[u] = cur;
60         }
61         int lca(int u, int v) {
```

```
62         while (top[u] != top[v]) {
63             if (dep[top[u]] > dep[top[v]]) {
64                 u = parent[top[u]];
65             } else {
66                 v = parent[top[v]];
67             }
68         }
69         return dep[u] < dep[v] ? u : v;
70     }
71
72     int dist(int u, int v) {
73         return dep[u] + dep[v] - 2 * dep[lca(u, v)];
74     }
75
76     int jump(int u, int k) {
77         if (dep[u] < k) {
78             return -1;
79         }
80
81         int d = dep[u] - k;
82
83         while (dep[top[u]] > d) {
84             u = parent[top[u]];
85         }
86
87         return seq[in[u] - dep[u] + d];
88     }
89
90     /*
91      * 判断u是否是v的祖先
92      */
93     bool isAncestor(int u, int v) {
94         return in[u] <= in[v] && in[v] < out[u];
95     }
96
97     int rootedParent(int u, int v) {
98         std::swap(u, v);
99         if (u == v) {
100             return u;
```

```
101     }
102     if (!isAncestor(u, v)) {
103         return parent[u];
104     }
105     auto it = std::upper_bound(adj[u].begin(), adj[u].end()
106         , v, [&](int x, int y) {
107         return in[x] < in[y];
108     }) - 1;
109     return *it;
110 }
111
112 /*
113  * 返回在以  $v$  为根时, 节点  $u$  的子树大小。
114 */
115 int rootedSize(int u, int v) {
116     if (u == v) {
117         return n;
118     }
119     if (!isAncestor(v, u)) {
120         return siz[v];
121     }
122     return n - siz[rootedParent(u, v)];
123 }
124
125 int rootedLca(int a, int b, int c) {
126     return lca(a, b) ^ lca(b, c) ^ lca(c, a);
127 }
128
129 std::vector<std::pair<int, int>> get_path(int u, int v) {
130     std::vector<std::pair<int, int>> v1, v2;
131     while(top[u] != top[v]) {
132         if(dep[top[u]] > dep[top[v]]) {
133             v1.push_back({dfn[u], dfn[top[u]]});
134             u = parent[top[u]];
135         } else {
136             v2.push_back({dfn[top[v]], dfn[v]});
137             v = parent[top[v]];
138         }
```

```
139     }
140     v1.reserve(v1.size() + v2.size() + 1);
141     v1.push_back({dfn[u], dfn[v]});
142     reverse(v2.begin(), v2.end());
143     for(auto v : v2) {
144         v1.push_back(v);
145     }
146     return v1;
147 }
148 };
```

### 3.6 树上差分

注意！由于这部分内容大部分与树上 *LCA* 高度重合，因此在此作简要说明！

树上 *LCA* 问题可以使用 *HLD* 或者倍增表实现!!!

树上点的编号从 1 开始，并且我们认为 1 的父亲节点是编号 0!!!

#### 3.6.1 点差分

```
1 // modify
2 int u, v;
3 cin >> u >> v;
4 lca = lca(u, v);
5 lcafather = stjump[lca][0];
6 num[u]++;
7 num[v]++;
8 num[lca]--;
9 num[lcafather]--;
10
11 // calc
12 auto dfs = [&](this auto&&self, int u, int fa) -> void {
13     for(auto v : adj[u]) {
14         if(v != fa) {
15             self(v, u);
16             num[u] += num[v];
17         }
18     }
19 };
20 dfs(dfs, 1, 0);
```

### 3.6.2 边差分

```
1 // modify
2 int u, v;
3 cin >> u >> v;
4 lca = lca(u, v);
5 lcafather = stjump[lca][0];
6 num[u]++;
7 num[v]++;
8 num[lca] -= 2;
9
10 // calc
11 auto dfs = [&](this auto&&self, int u, int fa) -> void {
12     for(auto v : adj[u]) {
13         if(v != fa) {
14             self(v, u);
15             num[u] += num[v];
16         }
17     }
18 };
19 dfs(dfs, 1, 0);
```

## 3.7 Splay

```
1 class Splay {
2 public:
3     Splay() {}
4
5     Splay(int n) {
6         init(n);
7     }
8
9     void init(int n) {
10         cnt = head = 0;
11         Tree.assign(n + 5, {});
12     }
13
14     // add a num to the Tree
15     void add(int num) {
```



```
16         add(head, num);
17     }
18
19     // find the rank`s node in the Tree`s inorder
20     int find(int rank) {
21         return find(head, rank);
22     }
23
24     // query the num`s rank in the Tree
25     int rank(int num) {
26         return rank(head, num);
27     }
28
29     // return x-th`s value after sorting
30     int index(int x) {
31         int i = find(x);
32         splay(i, 0);
33         return Tree[i].key;
34     }
35
36     // return the pre num`s value
37     int pre(int num) {
38         return pre(head, num);
39     }
40
41     // return the post num`s value
42     int post(int num) {
43         return post(head, num);
44     }
45
46     // remove a num from the Tree
47     void remove(int num) {
48         int kth = rank(num);
49         if(kth != rank(num + 1)) {
50             int i = find(kth);
51             splay(i, 0);
52             if(Tree[i].ls == 0) {
53                 head = Tree[i].rs;
54             } else if(Tree[i].rs == 0) {
```

```
55         head = Tree[i].ls;
56     } else {
57         int j = find(kth + 1);
58         splay(j, i);
59         Tree[j].ls = Tree[i].ls;
60         Tree[Tree[j].ls].father = j;
61         up(j);
62         head = j;
63     }
64     Tree[head].father = 0;
65 }
66 }
67
68 private:
69
70     // Summary all the info about node[i]'s son
71     void up(int i) {
72         Tree[i].size = Tree[Tree[i].ls].size + Tree[Tree[i].rs
73             ].size + 1;
74     }
75
76     // check i is the right child of its father
77     int lr(int i) {
78         return Tree[Tree[i].father].rs == i ? 1 : 0;
79     }
80
81     void rotate(int i) {
82         int f = Tree[i].father, g = Tree[f].father;
83         int soni = lr(i), sonf = lr(f);
84         if(soni == 1) {
85             Tree[f].rs = Tree[i].ls;
86             if(Tree[f].rs != 0) {
87                 Tree[Tree[f].rs].father = f;
88             }
89             Tree[i].ls = f;
90         } else {
91             Tree[f].ls = Tree[i].rs;
92             if(Tree[f].ls != 0) {
93                 Tree[Tree[f].ls].father = f;
```

```
93         }
94         Tree[i].rs = f;
95     }
96     if(g != 0) {
97         if(sonf == 0) {
98             Tree[g].ls = i;
99         } else {
100             Tree[g].rs = i;
101         }
102     }
103     Tree[i].father = g;
104     Tree[f].father = i;
105     up(f);
106     up(i);
107 }
108
109 // make node[i] is a child of node[goal]
110 void splay(int i, int goal) {
111     int f = Tree[i].father, g = Tree[f].father;
112     while(f != goal) {
113         if(g != goal) {
114             if(lr(i) == lr(f)) {
115                 rotate(f);
116             } else {
117                 rotate(i);
118             }
119         }
120         rotate(i);
121         f = Tree[i].father;
122         g = Tree[f].father;
123     }
124     if(goal == 0) {
125         head = i;
126     }
127 }
128
129 void add(int i, int num) {
130     Tree[++cnt].key = num;
131     Tree[cnt].size = 1;
```

```
132         if(head == 0) {
133             head = cnt;
134         } else {
135             int f = 0, son = 0;
136             while(i != 0) {
137                 f = i;
138                 if(Tree[i].key <= num) {
139                     son = 1;
140                     i = Tree[i].rs;
141                 } else {
142                     son = 0;
143                     i = Tree[i].ls;
144                 }
145             }
146             if(son == 0) {
147                 Tree[f].ls = cnt;
148             } else {
149                 Tree[f].rs = cnt;
150             }
151             Tree[cnt].father = f;
152             splay(cnt, 0);
153         }
154     }
155
156     int find(int i, int rank) {
157         while(i != 0) {
158             if(Tree[Tree[i].ls].size + 1 == rank) {
159                 return i;
160             } else if(Tree[Tree[i].ls].size >= rank) {
161                 i = Tree[i].ls;
162             } else {
163                 rank -= Tree[Tree[i].ls].size + 1;
164                 i = Tree[i].rs;
165             }
166         }
167         return 0;
168     }
169
170     int rank(int i, int num) {
```

```
171         int ans = 0, f = 0;
172         while(i != 0) {
173             f = i;
174             if(Tree[i].key >= num) {
175                 i = Tree[i].ls;
176             } else {
177                 ans += Tree[Tree[i].ls].size + 1;
178                 i = Tree[i].rs;
179             }
180         }
181         splay(f, 0);
182         return ans + 1;
183     }
184
185     int pre(int i, int num) {
186         int last = head;
187         int ans = std::numeric_limits<int>::min();
188         while(i != 0) {
189             last = i;
190             if(Tree[i].key < num) {
191                 ans = std::max(ans, Tree[i].key);
192                 i = Tree[i].rs;
193             } else {
194                 i = Tree[i].ls;
195             }
196         }
197         splay(last, 0);
198         return ans;
199     }
200
201     int post(int i, int num) {
202         int last = head;
203         int ans = std::numeric_limits<int>::max();
204         while(i != 0) {
205             last = i;
206             if(Tree[i].key > num) {
207                 ans = std::min(ans, Tree[i].key);
208                 i = Tree[i].ls;
209             } else {
```

```

210             i = Tree[i].rs;
211         }
212     }
213     splay(last, 0);
214     return ans;
215 }
216
217 struct Node {
218     int ls, rs, size, father, key;
219 };
220
221 int cnt, head;
222 std::vector<Node> Tree;
223 };

```

## 数学专题模板

### 4.1 Exgcd

```

1 int exgcd(int a, int b, int &x, int &y) {
2     if (b == 0) {
3         x = 1, y = 0;
4         return a;
5     }
6     int d = exgcd(b, a % b, y, x);
7     y -= (a / b * x);
8     return d;
9 }

```

### 4.2 Frac

```

1 template<class T>
2 struct Frac {
3     T num;
4     T den;
5     Frac(T num_, T den_) : num(num_), den(den_) {
6         if (den < 0) {
7             den = -den;

```

```
8         num = -num;
9     }
10 }
11 Frac() : Frac(0, 1) {}
12 Frac(T num_) : Frac(num_, 1) {}
13 explicit operator double() const {
14     return 1. * num / den;
15 }
16 Frac &operator+=(const Frac &rhs) {
17     num = num * rhs.den + rhs.num * den;
18     den *= rhs.den;
19     return *this;
20 }
21 Frac &operator-=(const Frac &rhs) {
22     num = num * rhs.den - rhs.num * den;
23     den *= rhs.den;
24     return *this;
25 }
26 Frac &operator*=(const Frac &rhs) {
27     num *= rhs.num;
28     den *= rhs.den;
29     return *this;
30 }
31 Frac &operator/=(const Frac &rhs) {
32     num *= rhs.den;
33     den *= rhs.num;
34     if (den < 0) {
35         num = -num;
36         den = -den;
37     }
38     return *this;
39 }
40 friend Frac operator+(Frac lhs, const Frac &rhs) {
41     return lhs += rhs;
42 }
43 friend Frac operator-(Frac lhs, const Frac &rhs) {
44     return lhs -= rhs;
45 }
46 friend Frac operator*(Frac lhs, const Frac &rhs) {
```

```

47         return lhs *= rhs;
48     }
49     friend Frac operator/(Frac lhs, const Frac &rhs) {
50         return lhs /= rhs;
51     }
52     friend Frac operator-(const Frac &a) {
53         return Frac(-a.num, a.den);
54     }
55     friend bool operator==(const Frac &lhs, const Frac &rhs) {
56         return lhs.num * rhs.den == rhs.num * lhs.den;
57     }
58     friend bool operator!=(const Frac &lhs, const Frac &rhs) {
59         return lhs.num * rhs.den != rhs.num * lhs.den;
60     }
61     friend bool operator<(const Frac &lhs, const Frac &rhs) {
62         return lhs.num * rhs.den < rhs.num * lhs.den;
63     }
64     friend bool operator>(const Frac &lhs, const Frac &rhs) {
65         return lhs.num * rhs.den > rhs.num * lhs.den;
66     }
67     friend bool operator<=(const Frac &lhs, const Frac &rhs) {
68         return lhs.num * rhs.den <= rhs.num * lhs.den;
69     }
70     friend bool operator>=(const Frac &lhs, const Frac &rhs) {
71         return lhs.num * rhs.den >= rhs.num * lhs.den;
72     }
73     friend std::ostream &operator<<(std::ostream &os, Frac x) {
74         T g = std::gcd(x.num, x.den);
75         if (x.den == g) {
76             return os << x.num / g;
77         } else {
78             return os << x.num / g << "/" << x.den / g;
79         }
80     }
81 };

```

### 4.3 ModInt

```

1  template<class T>

```



```
2 constexpr T power(T a, u64 b, T res = 1) {
3     for (; b != 0; b /= 2, a *= a) {
4         if (b & 1) {
5             res *= a;
6         }
7     }
8     return res;
9 }
10
11 template<u32 P>
12 constexpr u32 mulMod(u32 a, u32 b) {
13     return u64(a) * b % P;
14 }
15
16 template<u64 P>
17 constexpr u64 mulMod(u64 a, u64 b) {
18     u64 res = a * b - u64(1.L * a * b / P - 0.5L) * P;
19     res %= P;
20     return res;
21 }
22
23 constexpr i64 safeMod(i64 x, i64 m) {
24     x %= m;
25     if (x < 0) {
26         x += m;
27     }
28     return x;
29 }
30
31 constexpr std::pair<i64, i64> invGcd(i64 a, i64 b) {
32     a = safeMod(a, b);
33     if (a == 0) {
34         return {b, 0};
35     }
36
37     i64 s = b, t = a;
38     i64 m0 = 0, m1 = 1;
39
40     while (t) {
```

```
41         i64 u = s / t;
42         s -= t * u;
43         m0 -= m1 * u;
44
45         std::swap(s, t);
46         std::swap(m0, m1);
47     }
48
49     if (m0 < 0) {
50         m0 += b / s;
51     }
52
53     return {s, m0};
54 }
55
56 template<std::unsigned_integral U, U P>
57 struct ModIntBase {
58 public:
59     constexpr ModIntBase() : x(0) {}
60     template<std::unsigned_integral T>
61     constexpr ModIntBase(T x_) : x(x_ % mod()) {}
62     template<std::signed_integral T>
63     constexpr ModIntBase(T x_) {
64         using S = std::make_signed_t<U>;
65         S v = x_ % S(mod());
66         if (v < 0) {
67             v += mod();
68         }
69         x = v;
70     }
71
72     constexpr static U mod() {
73         return P;
74     }
75
76     constexpr U val() const {
77         return x;
78     }
79 }
```

```

80     constexpr ModIntBase operator-() const {
81         ModIntBase res;
82         res.x = (x == 0 ? 0 : mod() - x);
83         return res;
84     }
85
86     constexpr ModIntBase inv() const {
87         return power(*this, mod() - 2);
88     }
89
90     constexpr ModIntBase &operator*=(const ModIntBase &rhs) & {
91         x = mulMod<mod()>(x, rhs.val());
92         return *this;
93     }
94     constexpr ModIntBase &operator+=(const ModIntBase &rhs) & {
95         x += rhs.val();
96         if (x >= mod()) {
97             x -= mod();
98         }
99         return *this;
100    }
101    constexpr ModIntBase &operator-=(const ModIntBase &rhs) & {
102        x -= rhs.val();
103        if (x >= mod()) {
104            x += mod();
105        }
106        return *this;
107    }
108    constexpr ModIntBase &operator/=(const ModIntBase &rhs) & {
109        return *this *= rhs.inv();
110    }
111
112    friend constexpr ModIntBase operator*(ModIntBase lhs, const
        ModIntBase &rhs) {
113        lhs *= rhs;
114        return lhs;
115    }
116    friend constexpr ModIntBase operator+(ModIntBase lhs, const
        ModIntBase &rhs) {

```

```
117         lhs += rhs;
118         return lhs;
119     }
120     friend constexpr ModIntBase operator-(ModIntBase lhs, const
        ModIntBase &rhs) {
121         lhs -= rhs;
122         return lhs;
123     }
124     friend constexpr ModIntBase operator/(ModIntBase lhs, const
        ModIntBase &rhs) {
125         lhs /= rhs;
126         return lhs;
127     }
128
129     friend constexpr std::istream &operator>>(std::istream &is,
        ModIntBase &a) {
130         i64 i;
131         is >> i;
132         a = i;
133         return is;
134     }
135     friend constexpr std::ostream &operator<<(std::ostream &os,
        const ModIntBase &a) {
136         return os << a.val();
137     }
138
139     friend constexpr std::strong_ordering operator<=>(
        ModIntBase lhs, ModIntBase rhs) {
140         return lhs.val() <=> rhs.val();
141     }
142
143 private:
144     U x;
145 };
146
147 template<u32 P>
148 using ModInt = ModIntBase<u32, P>;
149 template<u64 P>
150 using ModInt64 = ModIntBase<u64, P>;
```

```
151
152 struct Barrett {
153 public:
154     Barrett(u32 m_) : m(m_), im((u64)(-1) / m_ + 1) {}
155
156     constexpr u32 mod() const {
157         return m;
158     }
159
160     constexpr u32 mul(u32 a, u32 b) const {
161         u64 z = a;
162         z *= b;
163
164         u64 x = u64((u128(z) * im) >> 64);
165
166         u32 v = u32(z - x * m);
167         if (m <= v) {
168             v += m;
169         }
170         return v;
171     }
172
173 private:
174     u32 m;
175     u64 im;
176 };
177
178 template<u32 Id>
179 struct DynModInt {
180 public:
181     constexpr DynModInt() : x(0) {}
182     template<std::unsigned_integral T>
183     constexpr DynModInt(T x_) : x(x_ % mod()) {}
184     template<std::signed_integral T>
185     constexpr DynModInt(T x_) {
186         int v = x_ % int(mod());
187         if (v < 0) {
188             v += mod();
189         }
```

```
190         x = v;
191     }
192
193     constexpr static void setMod(u32 m) {
194         bt = m;
195     }
196
197     static u32 mod() {
198         return bt.mod();
199     }
200
201     constexpr u32 val() const {
202         return x;
203     }
204
205     constexpr DynModInt operator-() const {
206         DynModInt res;
207         res.x = (x == 0 ? 0 : mod() - x);
208         return res;
209     }
210
211     constexpr DynModInt inv() const {
212         auto v = invGcd(x, mod());
213         assert(v.first == 1);
214         return v.second;
215     }
216
217     constexpr DynModInt &operator*=(const DynModInt &rhs) & {
218         x = bt.mul(x, rhs.val());
219         return *this;
220     }
221     constexpr DynModInt &operator+=(const DynModInt &rhs) & {
222         x += rhs.val();
223         if (x >= mod()) {
224             x -= mod();
225         }
226         return *this;
227     }
228     constexpr DynModInt &operator-=(const DynModInt &rhs) & {
```

```
229         x -= rhs.val();
230         if (x >= mod()) {
231             x += mod();
232         }
233         return *this;
234     }
235     constexpr DynModInt &operator/=(const DynModInt &rhs) & {
236         return *this *= rhs.inv();
237     }
238
239     friend constexpr DynModInt operator*(DynModInt lhs, const
        DynModInt &rhs) {
240         lhs *= rhs;
241         return lhs;
242     }
243     friend constexpr DynModInt operator+(DynModInt lhs, const
        DynModInt &rhs) {
244         lhs += rhs;
245         return lhs;
246     }
247     friend constexpr DynModInt operator-(DynModInt lhs, const
        DynModInt &rhs) {
248         lhs -= rhs;
249         return lhs;
250     }
251     friend constexpr DynModInt operator/(DynModInt lhs, const
        DynModInt &rhs) {
252         lhs /= rhs;
253         return lhs;
254     }
255
256     friend constexpr std::istream &operator>>(std::istream &is,
        DynModInt &a) {
257         i64 i;
258         is >> i;
259         a = i;
260         return is;
261     }
```

```

262     friend constexpr std::ostream &operator<<(std::ostream &os,
          const DynModInt &a) {
263         return os << a.val();
264     }
265
266     friend constexpr std::strong_ordering operator<=>(DynModInt
          lhs, DynModInt rhs) {
267         return lhs.val() <=> rhs.val();
268     }
269
270 private:
271     u32 x;
272     static Barrett bt;
273 };
274
275 constexpr int MOD = 1'000'000'007;
276 // constexpr int MOD = 998'244'353;
277
278 template<u32 Id>
279 Barrett DynModInt<Id>::bt = MOD;
280
281 using Z = ModInt<MOD>;

```

## 4.4 Sieve

```

1 // 输入:  $n$  - 筛的范围上限
2 // 输出:
3 //   primes      - 质数集合
4 //   min_prime   - 每个数的最小质因子
5 //   phi         - 欧拉函数值
6 //   mu          - 莫比乌斯函数值
7 //   d           - 约数个数
8 //   cnt_min_p   - 最小质因子的次数
9 std::vector<int> primes, min_prime, phi, mu, d, cnt_min_p;
10 std::vector<bool> is_prime;
11 void euler_sieve(int n) {
12     min_prime.resize(n + 1, 0);
13     phi.resize(n + 1, 0);
14     mu.resize(n + 1, 0);

```



```
15     d.resize(n + 1, 0);
16     cnt_min_p.resize(n + 1, 0);
17     is_prime.resize(n + 1, true);
18     phi[1] = mu[1] = d[1] = 1;
19     for(int i = 2; i <= n; i++) {
20         if(is_prime[i]) {
21             primes.push_back(i);
22             min_prime[i] = i;
23             phi[i] = i - 1;
24             mu[i] = -1;
25             d[i] = 2;
26             cnt_min_p[i] = 1;
27         }
28         for(auto p : primes) {
29             int num = i * p;
30             if(num > n) {
31                 break;
32             }
33             is_prime[num] = false;
34             min_prime[num] = p;
35             if(i % p == 0) {
36                 phi[num] = phi[i] * p;
37                 mu[num] = 0;
38                 cnt_min_p[num] = cnt_min_p[i] + 1;
39                 d[num] = d[i] / (cnt_min_p[i] + 1) * (cnt_min_p
40                     [i] + 1);
41                 break;
42             } else {
43                 phi[num] = phi[i] * (p - 1);
44                 mu[num] = -mu[i];
45                 cnt_min_p[num] = 1;
46                 d[num] = d[i] * 2;
47             }
48         }
49     }
```

## 4.5 Comb(结合 ModInt)

```
1 struct Comb {
2     int n;
3     std::vector<Z> _fac;
4     std::vector<Z> _invfac;
5     std::vector<Z> _inv;
6
7     Comb() : n{0}, _fac{1}, _invfac{1}, _inv{0} {}
8     Comb(int n) : Comb() {
9         init(n);
10    }
11
12    void init(int m) {
13        if (m <= n) return;
14        _fac.resize(m + 1);
15        _invfac.resize(m + 1);
16        _inv.resize(m + 1);
17
18        for (int i = n + 1; i <= m; i++) {
19            _fac[i] = _fac[i - 1] * i;
20        }
21        _invfac[m] = _fac[m].inv();
22        for (int i = m; i > n; i--) {
23            _invfac[i - 1] = _invfac[i] * i;
24            _inv[i] = _invfac[i] * _fac[i - 1];
25        }
26        n = m;
27    }
28
29    Z fac(int m) {
30        if (m > n) init(2 * m);
31        return _fac[m];
32    }
33    Z invfac(int m) {
34        if (m > n) init(2 * m);
35        return _invfac[m];
36    }
37    Z inv(int m) {
38        if (m > n) init(2 * m);
39        return _inv[m];
```

```

40     }
41     Z C(int n, int m) {
42         if (n < m || m < 0) return 0;
43         return fac(n) * invfac(m) * invfac(n - m);
44     }
45     Z A(int n, int m) {
46         if(n < m || m < 0) return 0;
47         return fac(n) * invfac(n - m);
48     }
49 } comb;

```

## 4.6 BigInt

```

1  constexpr int BASE = 1000000000;
2  constexpr int MAX_LENGTH = 1000;
3  constexpr int NUM_DIGIT = 8;
4  i64 mul_mod (i64 x, i64 y, i64 n){
5      i64 T = std::floor(std::sqrt(n) + 0.5);
6      i64 t = T * T - n;
7      i64 a = x / T; i64 b = x % T;
8      i64 c = y / T; i64 d = y % T;
9      i64 e = a * c / T; i64 f = a * c % T;
10     i64 v = ((a * d + b * c) % n + e * t) % n;
11     i64 g = v / T; i64 h = v % T;
12     i64 ans = (((f + g) * t % n + b * d) % n + h * T) % n;
13     while (ans < 0) {
14         ans += n;
15     }
16     return ans;
17 }
18 struct BigInt {
19
20     BigInt(const char* str = "0") {
21         (*this) = str;
22     }
23
24     BigInt operator=(const char* str) {
25         int j = std::strlen(str) - 1;
26         len = j / NUM_DIGIT + 1;

```

```
27         for(int i = 0; i <= len; i++) {
28             s[i] = 0;
29         }
30         for(int i = 0; i <= j; i++) {
31             int k = (j - i) / NUM_DIGIT + 1;
32             s[k] = s[k] * 10 + (str[i] - '0');
33         }
34         return *this;
35     }
36
37     void print() {
38         printf("%d", s[len]);
39         for(int i = len - 1; i >= 1; i--) {
40             printf("%0*d", NUM_DIGIT, s[i]);
41         }
42     }
43
44     int len;
45     int s[MAX_LENGTH];
46 };
47
48 // > return > 0
49 // < return < 0
50 // = return = 0
51 int compare(const BigInt &lhs, const BigInt &rhs) {
52     if(lhs.len > rhs.len) {
53         return 1;
54     }
55     if(lhs.len < rhs.len) {
56         return -1;
57     }
58     int cur = lhs.len;
59     while((cur > 1) && (lhs.s[cur] == rhs.s[cur])) {
60         cur--;
61     }
62     return lhs.s[cur] - rhs.s[cur];
63 }
64
65 bool operator<(const BigInt &lhs, const BigInt &rhs) {
```

```
66     return compare(lhs, rhs) < 0;
67 }
68
69 bool operator<=(const BigInt &lhs, const BigInt &rhs) {
70     return compare(lhs, rhs) <= 0;
71 }
72
73 bool operator>(const BigInt &lhs, const BigInt &rhs) {
74     return compare(lhs, rhs) > 0;
75 }
76
77 bool operator>=(const BigInt &lhs, const BigInt &rhs) {
78     return compare(lhs, rhs) >= 0;
79 }
80
81 bool operator==(const BigInt &lhs, const BigInt &rhs) {
82     return compare(lhs, rhs) == 0;
83 }
84
85 bool operator!=(const BigInt &lhs, const BigInt &rhs) {
86     return compare(lhs, rhs) != 0;
87 }
88
89
90 BigInt operator+(const BigInt &lhs, const BigInt &rhs) {
91     BigInt ret;
92     int i;
93     for(i = 1; i <= lhs.len || i <= rhs.len || ret.s[i]; i++) {
94         if(i <= lhs.len) {
95             ret.s[i] += lhs.s[i];
96         }
97         if(i <= rhs.len) {
98             ret.s[i] += rhs.s[i];
99         }
100         ret.s[i + 1] = ret.s[i] / BASE;
101         ret.s[i] %= BASE;
102     }
103     ret.len = i - 1;
104     if(ret.len == 0) {
```

```
105         ret.len = 1;
106     }
107     return ret;
108 }
109
110 // lhs > rhs
111 BigInt operator-(const BigInt &lhs, const BigInt &rhs) {
112     BigInt ret;
113     for(int i = 1, j = 0; i <= lhs.len; i++) {
114         ret.s[i] = lhs.s[i] - j;
115         if(i <= rhs.len) {
116             ret.s[i] -= rhs.s[i];
117         }
118         if(ret.s[i] < 0) {
119             j = 1;
120             ret.s[i] += BASE;
121         } else {
122             j = 0;
123         }
124     }
125     ret.len = lhs.len;
126     while(ret.len > 1 && !ret.s[ret.len]) {
127         ret.len--;
128     }
129     return ret;
130 }
131
132 BigInt operator*(const BigInt &lhs, const BigInt &rhs) {
133     BigInt ret;
134     i64 g = 0;
135     ret.len = lhs.len + rhs.len;
136     ret.s[0] = 0;
137     for(int i = 1; i <= ret.len; i++) {
138         ret.s[i] = 0;
139     }
140     for(int k = 1; k <= ret.len; k++) {
141         i64 tmp = g;
142         int j = k + 1 - rhs.len;
143         if(j < 1) {
```

```
144         j = 1;
145     }
146     for(; j <= k && j <= lhs.len; j++) {
147         tmp += (i64)lhs.s[j] * (i64)rhs.s[k + 1 - j];
148     }
149     g = tmp / BASE;
150     ret.s[k] = tmp % BASE;
151 }
152 while(ret.len > 1 && !ret.s[ret.len]) {
153     ret.len--;
154 }
155 return ret;
156 }
157 BigInt operator/(const BigInt &lhs, int num) {
158     i64 g = 0;
159     BigInt ret;
160     ret.len = lhs.len;
161     for(int i = lhs.len; i > 0; i--) {
162         i64 tmp = g * BASE + lhs.s[i];
163         ret.s[i] = tmp / num;
164         g = tmp % num;
165     }
166     while(ret.len > 1 && !ret.s[ret.len]) {
167         ret.len--;
168     }
169     return ret;
170 }
171 BigInt operator/(const BigInt &lhs, const BigInt &rhs) {
172     BigInt l = "0", r = lhs;
173     while(l < r) {
174         BigInt m = l + (r - l + "1") / 2;
175         if(m * rhs <= lhs) {
176             l = m;
177         } else {
178             r = m - "1";
179         }
180     }
181     return l;
182 }
```

```
183
184 i64 BigMod(const BigInt &a, i64 m) {
185     i64 d = 0;
186     for(int i = a.len; i > 0; i--) {
187         d = mul_mod(d, BASE, m);
188         d = (d + a.s[i]) % m;
189     }
190     return d;
191 }
192 BigInt sqrt(const BigInt &a) {
193     BigInt x, y = a;
194     do {
195         x = y;
196         y = (x + a / x) / 2;
197     }while(y < x);
198
199     return x;
200 }
201 BigInt gcd(BigInt a, BigInt b) {
202     BigInt c = "1";
203     while(true) {
204         if(a == b) {
205             return a * c;
206         } else if(a.s[1] % 2 == 0 && b.s[1] % 2 == 0) {
207             a = a / 2;
208             b = b / 2;
209             c = c * "2";
210         } else if(a.s[1] % 2 == 0) {
211             a = a / 2;
212         } else if(b.s[1] % 2 == 0) {
213             b = b / 2;
214         } else if(b < a) {
215             a = a - b;
216         } else {
217             b = b - a;
218         }
219     }
220 }
```



## 4.7 Miller-RabinAndPollard-Rho

```
1 i64 mul(i64 a, i64 b, i64 m) {
2     return static_cast<__int128>(a) * b % m;
3 }
4 i64 power(i64 a, i64 b, i64 m) {
5     i64 res = 1 % m;
6     for (; b; b >>= 1, a = mul(a, a, m))
7         if (b & 1)
8             res = mul(res, a, m);
9     return res;
10 }
11 bool isprime(i64 n) {
12     if (n < 2)
13         return false;
14     static constexpr int A[] = {2, 3, 5, 7, 11, 13, 17, 19,
15         23};
16     int s = __builtin_ctzll(n - 1);
17     i64 d = (n - 1) >> s;
18     for (auto a : A) {
19         if (a == n)
20             return true;
21         i64 x = power(a, d, n);
22         if (x == 1 || x == n - 1)
23             continue;
24         bool ok = false;
25         for (int i = 0; i < s - 1; ++i) {
26             x = mul(x, x, n);
27             if (x == n - 1) {
28                 ok = true;
29                 break;
30             }
31         }
32         if (!ok)
33             return false;
34     }
35     return true;
36 }
37 std::vector<i64> factorize(i64 n) {
```

```
37     std::vector<i64> p;
38     std::function<void(i64)> f = [&](i64 n) {
39         if (n <= 10000) {
40             for (int i = 2; i * i <= n; ++i)
41                 for (; n % i == 0; n /= i)
42                     p.push_back(i);
43             if (n > 1)
44                 p.push_back(n);
45             return;
46         }
47         if (isprime(n)) {
48             p.push_back(n);
49             return;
50         }
51         auto g = [&](i64 x) {
52             return (mul(x, x, n) + 1) % n;
53         };
54         i64 x0 = 2;
55         while (true) {
56             i64 x = x0;
57             i64 y = x0;
58             i64 d = 1;
59             i64 power = 1, lam = 0;
60             i64 v = 1;
61             while (d == 1) {
62                 y = g(y);
63                 ++lam;
64                 v = mul(v, std::abs(x - y), n);
65                 if (lam % 127 == 0) {
66                     d = std::gcd(v, n);
67                     v = 1;
68                 }
69                 if (power == lam) {
70                     x = y;
71                     power *= 2;
72                     lam = 0;
73                     d = std::gcd(v, n);
74                     v = 1;
75                 }
76             }
77         }
```

```
76         }
77         if (d != n) {
78             f(d);
79             f(n / d);
80             return;
81         }
82         ++x0;
83     }
84 };
85 f(n);
86 std::sort(p.begin(), p.end());
87 return p;
88 }
```

## 字符串专题模板

### 5.1 StringHash

```
1  template<const long long N>
2  struct StringHash {
3      using i64 = long long;
4      using PII = std::pair<i64, i64>;
5      const i64 mod1 = 1e9 + 97, mod2 = 998244853, p1 = 131, p2 =
        233;
6      std::array<i64, N> a1, a2;
7      std::array<i64, N> Phs1, Phs2;
8      std::array<i64, N> Shs1, Shs2;
9      StringHash() {
10         init(N - 1);
11     }
12     StringHash(const std::string& S) {
13         init(N - 1);
14         work(S);
15     }
16     void work(const std::string& s) {
17         i64 n = s.size();
18         assert(n + 1 <= N);
19         for (int i = 0; i < n; ++i) {
```

```
20         i64 t = n - i - 1;
21         Phs1[i + 1] = ((i64)Phs1[i] * p1 + s[i]) % mod1;
22         Phs2[i + 1] = ((i64)Phs2[i] * p2 + s[i]) % mod2;
23         Shs1[t + 1] = ((i64)Shs1[t + 2] * p1 + s[t]) % mod1
24         ;
25         Shs2[t + 1] = ((i64)Shs2[t + 2] * p2 + s[t]) % mod2
26         ;
27     }
28 }
29 PII PreHash(i64 l, i64 r) {
30     assert(l <= r);
31     i64 P1 = (Phs1[r] - (i64)Phs1[l - 1] * a1[r - l + 1] %
32         mod1 + mod1) % mod1;
33     i64 P2 = (Phs2[r] - (i64)Phs2[l - 1] * a2[r - l + 1] %
34         mod2 + mod2) % mod2;
35     return PII(P1, P2);
36 };
37 PII SufHash(i64 l, i64 r) {
38     assert(l <= r);
39     i64 S1 = (Shs1[l] - (i64)Shs1[r + 1] * a1[r - l + 1] %
40         mod1 + mod1) % mod1;
41     i64 S2 = (Shs2[l] - (i64)Shs2[r + 1] * a2[r - l + 1] %
42         mod2 + mod2) % mod2;
43     return PII(S1, S2);
44 };
45 bool isPlalindrome(i64 l, i64 r) {
46     auto [P1, P2] = PreHash(l, r);
47     auto [S1, S2] = SufHash(l, r);
48     return P1 == S1 && P2 == S2;
49 };
50 void init(i64 n) {
51     a1[0] = a2[0] = 1;
52     for (int i = 0; i < n; ++i) {
53         a1[i + 1] = (i64)a1[i] * p1 % mod1;
54         a2[i + 1] = (i64)a2[i] * p2 % mod2;
55     }
56 }
```

```
53 StringHash<N> h;
```

## 5.2 AC 自动机

```
1  template<u64 MAXN, u64 MAXS>
2  struct AhoCorasick {
3      static constexpr int ALPHABEL = 26;
4
5      int cnt;
6      std::vector<int> end;
7      std::vector<std::array<int, ALPHABEL>> tree;
8      std::vector<int> fail;
9      std::vector<bool> alert;
10     std::vector<int> times;
11
12
13     AhoCorasick() {
14         end.assign(MAXN + 5, -1);
15         tree.assign(MAXS + 5, {});
16         fail.assign(MAXS + 5, 0);
17         alert.assign(MAXS + 5, false);
18         times.assign(MAXS + 5, 0);
19         cnt = 0;
20     }
21
22     void add(int i, const std::string &s) {
23         int u = 0;
24         for(int j = 0, c; j < s.size(); j++) {
25             c = s[j] - 'a';
26             if(tree[u][c] == 0) {
27                 tree[u][c] = ++cnt;
28             }
29             u = tree[u][c];
30         }
31         end[i] = u;
32         alert[u] = true;
33     }
34
35     void setfail() {
```

```
36         std::queue<int> q;
37         for(int i = 0; i < ALPHABEL; i++) {
38             if(tree[0][i] > 0) {
39                 q.push(tree[0][i]);
40             }
41         }
42         while(q.size()) {
43             int u = q.front();
44             q.pop();
45
46             for(int i = 0; i < ALPHABEL; i++) {
47                 if(tree[u][i] == 0) {
48                     tree[u][i] = tree[fail[u]][i];
49                 } else {
50                     fail[tree[u][i]] = tree[fail[u]][i];
51                     q.push(tree[u][i]);
52                 }
53             }
54
55             if(alert[fail[u]]) {
56                 alert[u] = true;
57             }
58         }
59     }
60
61     template<bool Counter = true>
62     void work() {
63         setfail();
64         if constexpr (Counter) {
65             std::string s;
66             std::cin >> s;
67
68             for(int i = 0, u = 0; i < s.size(); i++) {
69                 u = tree[u][s[i] - 'a'];
70                 times[u]++;
71             }
72
73             std::vector<std::vector<int>> g(MAXS + 5);
74
```

```
75         auto add_Edge = [&](int u, int v) -> void {
76             g[u].push_back(v);
77         };
78
79         auto dfs = [&] (auto &&self, int u) -> void {
80             for(auto v : g[u]) {
81                 self(self, v);
82                 times[u] += times[v];
83             }
84         };
85
86         for(int i = 1; i <= cnt; i++) {
87
88             add_Edge(fail[i], i);
89         }
90
91         dfs(dfs, 0);
92     }
93 }
94
95 int get_index_i_times(int i) {
96     return times[end[i]];
97 }
98
99 };
```

### 5.3 马拉车

```
1 // the real length in s from p is p[i] - 1
2 std::vector<int> manacher(std::string s) {
3     std::string t = "#";
4     for (auto c : s) {
5         t += c;
6         t += '#';
7     }
8     int n = t.size();
9     std::vector<int> p(n);
10    for(int i = 0, r = 0, c = 0, len; i < n; i++) {
11        len = r > i ? std::min(p[2 * c - i], r - i) : 1;
```

```
12         while(i + len < n && i - len >= 0 && t[i + len] == t[i  
13             - len]) {  
14             len++;  
15         }  
16         if(i + len > r) {  
17             r = i + len;  
18             c = i;  
19         }  
20         p[i] = len;  
21     }  
22     return p;  
23 }
```

## 5.4 Z 函数

```
1 std::vector<int> zFunction(std::string s) {  
2     int n = s.size();  
3     std::vector<int> z(n + 1);  
4     z[0] = n;  
5     for(int i = 1, c = 1, r = 1, len; i < n; i++) {  
6         len = r > i ? std::min(r - i, z[i - c]) : 0;  
7         while(i + len < n and s[i + len] == s[i]) {  
8             len++;  
9         }  
10        if(i + len > r) {  
11            r = i + len;  
12            c = i;  
13        }  
14        z[i] = len;  
15    }  
16    return z;  
17 }
```

## 5.5 后缀数组

```
1 struct SuffixArray {  
2     int n;  
3     std::vector<int> sa, rk, lc;
```



```
4     SuffixArray(const std::string &s) {
5         n = s.length();
6         sa.resize(n);
7         lc.resize(n - 1);
8         rk.resize(n);
9         std::iota(sa.begin(), sa.end(), 0);
10        std::sort(sa.begin(), sa.end(), [&](int a, int b) {
11            return s[a] < s[b];});
12        rk[sa[0]] = 0;
13        for (int i = 1; i < n; ++i)
14            rk[sa[i]] = rk[sa[i - 1]] + (s[sa[i]] != s[sa[i -
15                1]]);
16        int k = 1;
17        std::vector<int> tmp, cnt(n);
18        tmp.reserve(n);
19        while (rk[sa[n - 1]] < n - 1) {
20            tmp.clear();
21            for (int i = 0; i < k; ++i)
22                tmp.push_back(n - k + i);
23            for (auto i : sa)
24                if (i >= k)
25                    tmp.push_back(i - k);
26            std::fill(cnt.begin(), cnt.end(), 0);
27            for (int i = 0; i < n; ++i)
28                ++cnt[rk[i]];
29            for (int i = 1; i < n; ++i)
30                cnt[i] += cnt[i - 1];
31            for (int i = n - 1; i >= 0; --i)
32                sa[--cnt[rk[tmp[i]]]] = tmp[i];
33            std::swap(rk, tmp);
34            rk[sa[0]] = 0;
35            for (int i = 1; i < n; ++i)
36                rk[sa[i]] = rk[sa[i - 1]] + (tmp[sa[i - 1]] <
37                    tmp[sa[i]] || sa[i - 1] + k == n || tmp[sa[i]
38                        - 1] + k < tmp[sa[i] + k]);
39            k *= 2;
40        }
41        for (int i = 0, j = 0; i < n; ++i) {
42            if (rk[i] == 0) {
```

```
39         j = 0;
40     } else {
41         for (j -= j > 0; i + j < n && sa[rk[i] - 1] + j
              < n && s[i + j] == s[sa[rk[i] - 1] + j]; )
42             ++j;
43         lc[rk[i] - 1] = j;
44     }
45 }
46 }
47 };
```

### 5.5.1 使用方法示例

```
1  // deepseek
2  #include <iostream>
3  #include <string>
4  #include <vector>
5
6  struct SAM {
7      // ... (用户提供的模板代码)
8  };
9
10 int main() {
11     SAM sam;
12     int last = 1;  // 初始状态
13     std::string s = "abba";
14     for (char c : s) {
15         last = sam.extend(last, c, 'a');
16     }
17
18     // 统计不同子串数量
19     int count = 0;
20     for (int i = 2; i < sam.size(); ++i) {
21         count += sam.len(i) - sam.len(sam.link(i));
22     }
23     std::cout << "不同子串数量: " << count << std::endl;  // 输出 10
24
25     // 检查子串是否存在
```

```
26     auto is_substring = [&](const std::string& t) {
27         int p = 1;
28         for (char c : t) {
29             p = sam.next(p, c, 'a');
30             if (p == 0) return false;
31         }
32         return true;
33     };
34     std::cout << "子串 'ab' 是否存在: " << is_substring("ab")
35         << std::endl; // 输出 1
36
37     // 查找最长重复子串
38     int max_len = 0;
39     for (int i = 2; i < sam.size(); ++i) {
40         max_len = std::max(max_len, sam.len(i));
41     }
42     std::cout << "最长重复子串长度: " << max_len << std::endl;
43     // 输出 4
44
45     return 0;
46 }
```

## 5.6 回文自动机

```
1 struct PAM {
2     static constexpr int ALPHABET_SIZE = 28;
3     struct Node {
4         int len;
5         int link;
6         int cnt;
7         std::array<int, ALPHABET_SIZE> next;
8         Node() : len{}, link{}, cnt{}, next{} {}
9     };
10    std::vector<Node> t;
11    int suff;
12    std::string s;
13    PAM() {
14        init();
15    }
```

```
16     void init() {
17         t.assign(2, Node());
18         t[0].len = -1;
19         suff = 1;
20         s.clear();
21     }
22     int newNode() {
23         t.emplace_back();
24         return t.size() - 1;
25     }
26
27     bool add(char c, char offset = 'a') {
28         int pos = s.size();
29         s += c;
30         int let = c - offset;
31         int cur = suff, curlen = 0;
32
33         while (true) {
34             curlen = t[cur].len;
35             if (pos - 1 - curlen >= 0 && s[pos - 1 - curlen] ==
36                 s[pos])
37                 break;
38             cur = t[cur].link;
39         }
40         if (t[cur].next[let]) {
41             suff = t[cur].next[let];
42             return false;
43         }
44         int num = newNode();
45         suff = num;
46         t[num].len = t[cur].len + 2;
47         t[cur].next[let] = num;
48
49         if (t[num].len == 1) {
50             t[num].link = 1;
51             t[num].cnt = 1;
52             return true;
53         }
```

```
54         while (true) {
55             cur = t[cur].link;
56             curlen = t[cur].len;
57             if (pos - 1 - curlen >= 0 && s[pos - 1 - curlen] ==
58                 s[pos]) {
59                 t[num].link = t[cur].next[let];
60                 break;
61             }
62             t[num].cnt = 1 + t[t[num].link].cnt;
63             return true;
64         }
65     };
66     PAM pam;
```

## 5.7 KMP

```
1  template <typename T>
2  struct KMP {
3      using value_type = typename T::value_type;
4      using size_type = uint32_t;
5      T m_seq;
6      std::vector<size_type> m_pi;
7      void init() {
8          m_seq.push_back({});
9          m_pi.push_back(0);
10     }
11     KMP() {
12         clear();
13     }
14     template <typename InitMapping>
15     KMP(size_type length, InitMapping mapping) {
16         resize(length, mapping);
17     }
18     template <typename Iterator>
19     KMP(Iterator first, Iterator last) {
20         reset(first, last);
21     }
22     KMP(const T &seq) : KMP(seq.begin(), seq.end()) {}
```

```
23     template <typename InitMapping>
24     void resize(size_type length, InitMapping mapping) {
25         reserve(length);
26         for (size_type i = 0; i != length; i++) {
27             push_back(mapping(i));
28         }
29     }
30     template <typename Iterator>
31     void reset(Iterator first, Iterator last) {
32         resize(last - first, [&](size_type i) { return *(first
33             + i); });
34     }
35     void reserve(size_type length) {
36         clear();
37         m_seq.reserve(length);
38         m_pi.reserve(length);
39     }
40     void clear() {
41         m_seq.clear();
42         m_pi.clear();
43         init();
44     }
45     void push_back(const value_type &elem) {
46         m_seq.push_back(elem);
47         if (size() > 1) {
48             size_type pi = jump(m_pi.back(), elem);
49             m_pi.push_back(pi + (m_seq[pi + 1] == elem));
50         } else
51             m_pi.push_back(0);
52     }
53     void pop_back() {
54         m_seq.pop_back();
55         m_pi.pop_back();
56     }
57     size_type size() const { return m_seq.size() - 1; }
58     size_type jump(size_type last_pi, const value_type &elem)
59         const {
60         size_type len = last_pi;
```

```
59         while (len && (len == size() || m_seq[len + 1] != elem)
60             ) {
61             len = m_pi[len];
62         }
63         return len;
64     }
65     // Check if it is included in a certain sequence
66     // if not exist, return -1
67     template <typename Iterator>
68     size_type contained_by(Iterator first, Iterator last) const
69     {
70         if (!size()) return 0;
71         size_type len = 0;
72         for (auto it = first; it != last; ++it) {
73             const value_type &elem = *it;
74             while (len && m_seq[len + 1] != elem) {
75                 len = m_pi[len];
76             }
77             if (m_seq[len + 1] == elem) {
78                 len++;
79             }
80             if (len == size()) {
81                 return (it - first) - len + 1;
82             }
83         }
84         return -1;
85     }
86     // Call callback for all borders at a certain location
87     /*
88     kmp.do_for_each_border(j, [&](int pi) {
89         if (pi != j) cout << "/";
90         cout << p.substr(0, pi);
91     });
92     */
93     template <typename Callback>
94     void do_for_each_border(size_type init_border, Callback &&
95         call) {
96         size_type pi = init_border;
97         while (pi) {
```

```

95         call(pi);
96         pi = query_Pi(pi - 1);
97     }
98 }
99 // If there is a prefix string and a suffix string that are
    the same
100 // then return greater than 0 O(1)
101 size_type query_Pi(size_type i) const {
102     return m_pi[i + 1];
103 }
104 };
105 using KMP_string = KMP<std::string>;
106 template <typename ValueType>
107 using KMP_vector = KMP<std::vector<ValueType>>;

```

## 算法杂项专题模板

### 6.1 FastIO

```

1 namespace io_lib {
2 #ifdef FREAD
3 #define MAXBUFFERSIZE 1000000
4 inline char fgetc() {
5     static char buf[MAXBUFFERSIZE + 5], *p1 = buf, *p2 = buf;
6     return p1 == p2 && (p2 = (p1 = buf) + fread(buf, 1,
7         MAXBUFFERSIZE, stdin), p1 == p2) ? EOF : *p1++;
8 }
9 #undef MAXBUFFERSIZE
10 #define getchar fgetc
11 #define gc getchar
12 struct IReader {
13     template <typename T, typename std::enable_if<std::
14         is_integral<T>::value, int>::type = 0>
15     inline const IReader& operator>>(T& a) const {
16         a = 0;
17         bool flg = false;
18         char ch = gc();

```



```
18     while (ch < '0' || ch > '9') {
19         if (ch == '-') flg ^= 1;
20         ch = gc();
21     }
22     while (ch >= '0' && ch <= '9') {
23         a = (a << 3) + (a << 1) + (ch ^ '0');
24         ch = gc();
25     }
26     if (flg) a = -a;
27     return *this;
28 }
29 inline const IReader& operator>>(std::string& a) const {
30     a.clear();
31     char ch = gc();
32     while (isspace(ch) && ch != EOF) ch = gc();
33     while (!isspace(ch) && ch != EOF) a += ch, ch = gc();
34     return *this;
35 }
36 inline const IReader& operator>>(char* a) const {
37     char ch = gc();
38     while (isspace(ch) && ch != EOF) ch = gc();
39     while (!isspace(ch) && ch != EOF) *(a++) = ch, ch = gc();
40     *a = '\0';
41     return *this;
42 }
43 inline const IReader& operator>>(char& a) const {
44     a = gc();
45     while (isspace(a)) a = gc();
46     return *this;
47 }
48 template <typename T, typename std::enable_if<std::
49     is_floating_point<T>::value, int>::type = 0>
49 inline const IReader& operator>>(T& a) const {
50     a = 0;
51     bool flg = false;
52     char ch = gc();
53     while ((ch < '0' || ch > '9') && ch != '.') {
54         if (ch == '-') flg ^= 1;
55         ch = gc();
```

```
56     }
57     while (ch >= '0' && ch <= '9') {
58         a = a * 10 + (ch ^ '0');
59         ch = gc();
60     }
61     if (ch == '.') {
62         ch = gc();
63         T p = 0.1;
64         while (ch >= '0' && ch <= '9') {
65             a += p * (ch ^ '0');
66             ch = gc();
67             p *= 0.1;
68         }
69     }
70     if (flg) a = -a;
71     return *this;
72 }
73 template <typename T1, typename T2>
74 inline const IOReader& operator>>(std::pair<T1, T2>& p) const
75 {
76     return operator>>(p.first), operator>>(p.second), *this;
77 }
78 template <typename T, const unsigned long long N>
79 inline const IOReader& operator>>(std::array<T, N>& p) const
80 {
81     for (unsigned long long i = 0; i < N; i++)
82         operator>>(p[i]);
83     return *this;
84 }
85 template <typename... Ts>
86 inline const IOReader& operator>>(std::tuple<Ts...>& p) const
87 ;
88 #undef importRealReader
89 };
90 const IOReader io;
91 #undef gc
92 template <typename T>
93 void read(T& val) { io >> val; }
94 template <typename T>
```

```

92 void read(int l, int r, T& A) { for (int i = l; i <= r; i++) io
    >> A[i]; }
93 template <typename T>
94 void write(const T& A, int l, int r, const char* sp, const char
    * end = "") { for (int i = l; i <= r; i++) printf(sp, A[i]);
    printf("%s", end); }
95 template <typename T>
96 void write(const auto& A, const T* sp, const char* end = "") {
    for (auto e : A) printf(sp, e); printf("%s", end); }
97 template <typename T = int>
98 T read() { T res; io >> res; return res; }
99 template <typename T, int N>
100 std::array<T, N> read() { return read<std::array<T, N>>(); }
101 template <typename Tuple, typename Func, size_t... N>
102 void func_call_tuple(Tuple& t, Func&& func, std::index_sequence
    <N...>) { static_cast<void>(std::initializer_list<int>{(func
    (std::get<N>(t)), 0)...}); }
103 template <typename... Args, typename Func>
104 void travel_tuple(std::tuple<Args...>& t, Func&& func) {
    func_call_tuple(t, std::forward<Func>(func), std::
    make_index_sequence<sizeof...(Args)>{}); }
105 template <typename... Ts>
106 std::tuple<Ts...> reads() {
107     std::tuple<Ts...> res;
108     travel_tuple(res, [&](auto&& val) { io >> val; });
109     return res;
110 }
111 template <typename... Ts>
112 inline const IORReader& IORReader::operator>>(std::tuple<Ts...>&
    p) const { return p = reads<Ts...>(), *this; }
113 template <typename T = int>
114 std::vector<T> getv(int n, int start = 0) {
115     std::vector<T> res(start + n);
116     for (int i = start; i < start + n; i++) io >> res[i];
117     return res;
118 }
119 template <typename T, typename T1, typename... Ts>
120 std::vector<std::tuple<T, T1, Ts...>> getv(int n, int start =
    0) {

```

```

121     std::vector<std::tuple<T, T1, Ts...>> res(start + n);
122     for (int i = start; i < start + n; i++) io >> res[i];
123     return res;
124 }} // namespace io_lib
125 using namespace io_lib;
126
127 #define cin io

```

## 6.2 defs

```

1 namespace defs {
2 #define YES cout << "YES" << endl;
3 #define NO cout << "NO" << endl;
4 #define Yes cout << "Yes" << endl;
5 #define No cout << "No" << endl;
6 #define all(x) (x).begin(), (x).end()
7 #define rall(x) (x).rbegin(), (x).rend()
8 #define rep(i, j, k) for(int i = (j); i <= k; ++i)
9 #define per(i, j, k) for(int i = (j); i >= k; --i)
10 #define multiCase() \
11     int totCases; std::cin >> totCases; \
12     for(int currCase = 1; currCase <= totCases; currCase++)
13 using i32 = int;
14 using u32 = unsigned int;
15 using i64 = long long;
16 using u64 = unsigned long long;
17 using i128 = __int128;
18 using u128 = __uint128_t;
19 using f32 = float;
20 using f64 = double;
21 using TII = tuple<int, int, int>;
22 const i64 mod = 1'000'000'007 /* 998'244'353 */;
23 template <typename T> void sort(T& v) { sort(all(v)); }
24 template <typename T> T sorted(T v) { return sort(v), v; }
25 template <typename T> void rsort(T& v) { sort(rall(v)); }
26 template <typename T, typename T2> void sort(T& v, T2 compare)
27     { sort(all(v), compare); }
27 template <typename T, typename T2> T sorted(T v, T2 compare) {
28     return sort(v, compare), v; }

```

```

28 template <typename T> void reverse(T& v) { reverse(all(v)); }
29 template <typename T> T reversed(T v) { return reverse(v), v; }
30 template <typename T> void unique(vector<T>& v) { v.erase(
    unique(all(v)), v.end()); }
31 template <typename T> vector<T> uniqued(vector<T> v) { return
    unique(v), v; }
32 template <typename T> T min(const vector<T> &v) { return *
    min_element(all(v)); }
33 template <typename T> T max(const vector<T> &v) { return *
    max_element(all(v)); }
34 template <typename T> T acc(const vector<T> &v) { return
    accumulate(v.begin(), v.end(), T(0LL)); }
35 template <typename T> istream& operator>>(istream& is, std::
    vector<T>& v) { for(auto& x : v) { is >> x; } return is; }
36
37 }
38 using namespace defs;

```

### 6.3 Int128

```

1  #if defined(__GNUC__) || defined(__clang__)
2  using i128 = __int128;
3  using u128 = unsigned __int128;
4  #else
5  #error "int128 is only supported on GCC and Clang compilers"
6  #endif
7
8  namespace std {
9  template <>
10 class numeric_limits<i128> {
11 public:
12     static constexpr bool is_specialized = true;
13     static constexpr i128 min() { return static_cast<u128>(1)
        << 127; }
14     static constexpr i128 max() { return ~(static_cast<u128>(1)
        << 127); };
15 };
16 } // namespace std
17

```

```
18 std::ostream& operator<<(std::ostream& os, i128 n) {
19     if (n == 0) return os << '0';
20
21     const bool is_negative = n < 0;
22     u128 abs_n = is_negative ? -static_cast<u128>(n) :
        static_cast<u128>(n);
23
24     char buffer[40] = {0};
25     char* ptr = buffer + sizeof(buffer) - 1;
26
27     while (abs_n > 0) {
28         *--ptr = '0' + abs_n % 10;
29         abs_n /= 10;
30     }
31
32     if (is_negative) *--ptr = '-';
33     return os << ptr;
34 }
35
36
37 std::istream& operator>>(std::istream& is, i128& n) {
38     std::string s;
39     is >> s;
40
41     try {
42         n = toi128(s);
43     } catch (const std::exception& e) {
44         is.setstate(std::ios::failbit);
45         throw;
46     }
47     return is;
48 }
49
50 i128 toi128(const std::string& s) {
51     if (s.empty()) throw std::invalid_argument("Empty input
        string");
52
53     size_t pos = 0;
54     const bool negative = (s[0] == '-');
```

```
55     if (negative || s[0] == '+') pos++;
56     if (pos >= s.size()) throw std::invalid_argument("Invalid
        number format");
57
58     constexpr i128 max_prev = std::numeric_limits<i128>::max()
        / 10;
59     constexpr i128 max_digit = std::numeric_limits<i128>::max()
        % 10;
60
61     i128 result = 0;
62     for (; pos < s.size(); ++pos) {
63         if (!std::isdigit(s[pos])) {
64             throw std::invalid_argument("Non-digit character in
                input");
65         }
66
67         const int digit = s[pos] - '0';
68         if (result > max_prev || (result == max_prev && digit >
                max_digit + negative)) {
69             throw std::overflow_error("i128 overflow");
70         }
71
72         result = result * 10 + digit;
73     }
74     return negative ? -result : result;
75 }
76
77 i128 sqrti128(i128 n) {
78     if (n < 0) throw std::domain_error("Square root of negative
        number");
79     if (n == 0) return 0;
80
81     i128 low = 0;
82     i128 high = (static_cast<u128>(1) << 63) - 1;
83
84     while (low < high) {
85         const i128 mid = (low + high + 1) / 2;
86
87         bool overflow = false;
```

```
88         i128 square;
89         if (mid > std::numeric_limits<i128>::max() / mid) {
90             overflow = true;
91         } else {
92             square = mid * mid;
93         }
94
95         if (overflow || square > n) {
96             high = mid - 1;
97         } else {
98             low = mid;
99         }
100     }
101     return low;
102 }
103
104 i128 gcd(i128 a, i128 b) noexcept {
105     if (a == 0) return b;
106     if (b == 0) return a;
107
108     int shift = 0;
109     while (((a | b) & 1) == 0) {
110         a >>= 1;
111         b >>= 1;
112         ++shift;
113     }
114
115     while (a != b) {
116         while ((a & 1) == 0) a >>= 1;
117         while ((b & 1) == 0) b >>= 1;
118         if (a > b) std::swap(a, b);
119         b -= a;
120     }
121     return a << shift;
122 }
```

## 6.4 二分搜索

```
1 // return the first ans in [lo, hi], such as check(md) = true
```



```
2 // if no such ans, return hi + 1
3 template<class T, class Func>
4 T binary_min_left(T lo, T hi, Func check) {
5     T ans = hi + 1;
6     while(lo <= hi) {
7         T md = lo + (hi - lo) >> 1;
8         if(check(md)) {
9             ans = md;
10            hi = md - 1;
11        } else {
12            lo = md + 1;
13        }
14    }
15    return ans;
16 }
17
18
19 // return the last ans in [lo, hi], such as check(md) = true
20 // if no such ans, return lo - 1
21 template<class T, class Func>
22 T binary_max_right(T lo, T hi, Func check) {
23     T ans = lo - 1;
24     while(lo <= hi) {
25         T md = lo + (hi - lo) >> 1;
26         if(check(md)) {
27             ans = md;
28             lo = md + 1;
29         } else {
30             hi = md - 1;
31         }
32     }
33     return ans;
34 }
```

## 6.5 自定义哈希

```
1 struct custom_hash_64 {
2     static uint64_t splitmix64(uint64_t x) {
3         x += 0x9e3779b97f4a7c15;
```

```

4         x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
5         x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
6         return x ^ (x >> 31);
7     }
8
9     size_t operator()(uint64_t x) const {
10         static const uint64_t FIXED_RANDOM =
11             std::chrono::steady_clock::now().time_since_epoch()
12                 .count();
13         return splitmix64(x ^ FIXED_RANDOM);
14     }
15 };
16 struct custom_hash_32 {
17     uint64_t operator()(uint32_t x) const {
18         static const uint32_t RANDOM =
19             std::chrono::steady_clock::now().time_since_epoch()
20                 .count();
21         return (x ^ RANDOM) * 0x9e3779b1;
22     }
23 };
24 // unordered_map<int, int, custom_hash_xxx> cnt;

```

## 平面几何专题模板

### 7.1 平面几何

```

1 template<class T>
2 int sgn(const T& v) {
3     static constexpr T eps = 1e-8;
4     return v > eps ? 1 : v < -eps ? -1 : 0;
5 }
6 template<class T>
7 struct Point { // Point or Vector
8     T x, y;
9     Point() : x(0), y(0) {}
10    Point(T x, T y) : x(x), y(y) {}

```

```
11     template<class U>
12     explicit operator Point<U>() const {
13         return Point<U>(U(x), U(y));
14     }
15     Point operator+(const Point& o) const {
16         return Point(x + o.x, y + o.y);
17     }
18     Point operator-(const Point& o) const {
19         return Point(x - o.x, y - o.y);
20     }
21     Point operator-() const {
22         return Point(-x, -y);
23     }
24     Point operator*(const T& v) const {
25         return Point(x * v, y * v);
26     }
27     friend Point operator*(const T& v, const Point<T>& o) {
28         return Point(o.x * v, o.y * v);
29     }
30     Point operator/(const T& v) const {
31         return Point(x / v, y / v);
32     }
33     Point operator+=(const Point& o) {
34         x += o.x, y += o.y;
35         return *this;
36     }
37     Point operator-=(const Point& o) {
38         x -= o.x, y -= o.y;
39         return *this;
40     }
41     Point operator*=(const T& v) {
42         x *= v, y *= v;
43         return *this;
44     }
45     Point operator/=(const T& v) {
46         x /= v, y /= v;
47         return *this;
48     }
49     bool operator==(const Point& o) const {
```

```

50         return sgn(x - o.x) == 0 and sgn(y - o.y) == 0;
51     }
52     bool operator!=(const Point& o) const {
53         return sgn(x - o.x) != 0 or sgn(y - o.y) != 0;
54     }
55     bool operator<(const Point& o) const {
56         return sgn(x - o.x) < 0 or sgn(x - o.x) == 0 and sgn(y
            - o.y) < 0;
57     }
58     bool operator>(const Point& o) const {
59         return sgn(x - o.x) > 0 or sgn(x - o.x) == 0 and sgn(y
            - o.y) > 0;
60     }
61     static bool argcmp(const Point& a, const Point& b) {
62         static auto get = [&](const Point& o) {
63             if(sgn(o.x) == 0 and sgn(o.y) == 0) return 0;
64             if(sgn(o.y) > 0 or sgn(o.y) == 0 and sgn(o.x) < 0)
                return 1;
65             return -1;
66         };
67         int ta = get(a), tb = get(b);
68         if(ta != tb) return ta < tb;
69         return a.toLeft(b) == 1; // 不关注极径
70         // int tole = a.toLeft(b);
71         // if(tole != 0) return tole == 1;
72         // return sgn(a.square()-b.square()) < 0; // 极角相同按
            极径排
73     }
74     T dot(const Point& o) const {
75         return x * o.x + y * o.y;
76     }
77     T cross(const Point& o) const {
78         return x * o.y - y * o.x;
79     }
80     int toLeft(const Point& o) const {
81         return sgn(cross(o));
82     }
83     T square() const {
84         return x * x + y * y;

```

```

85     }
86     T interSquare(const Point& o) const {
87         return (*this - o).square();
88     }
89     friend istream& operator>>(istream& in, Point& o) {
90         return in >> o.x >> o.y;
91     }
92     friend ostream& operator<<(ostream& out, Point const& o) {
93         return out << "(" << o.x << "," << o.y << ")";
94     }
95     // 涉及浮点数
96     double length() const {
97         return sqrtl(square());
98     }
99     double distance(const Point& o) const {
100         return (*this - o).length();
101     }
102     // 逆时针旋转 rad
103     template<class U>
104     Point<U> rotate(U cosr, U sinr) const {
105         return Point(x * cosr - y * sinr, x * sinr + y * cosr);
106     }
107     // 两向量夹角范围是  $[0, \pi]$ 
108     double ang(const Point& o) const {
109         return acosl(max(-1.01, min(1.01, dot(o) / (length() *
110             o.length()))));
111     }
112 };
113 template<class T>
114 struct Line { // Line or Segment
115     Point<T> a, b; // 方向为 a->b
116     Line() {}
117     Line(const Point<T>& a, const Point<T>& b) : a(a), b(b) {}
118     template<class U>
119     Line(const Point<U>& a, const Point<U>& b) : a(a), b(b) {}
120     Point<T> vec() const {
121         return b - a;
122     }

```

```

122      // Line
      -----

123      bool parallel(const Line& l) const {
124          return sgn((b - a).cross(l.b - l.a)) == 0;
125      }
126      int toLeft(const Point<T>& o) const {
127          return (b - a).toLeft(o - a);
128      }
129      // 涉及浮点数
130      // 直线交点
131      Point<double> lineIntersection(const Line& l) const {
132          return Point<double>(a) + Point<double>(b - a) *
133              (1. * (l.b - l.a).cross(a - l.a) / (l.b - l.a).
134                  cross(a - b));
135      }
136      // 点到直线的距离
137      double distanceLP(const Point<T>& o) const {
138          return abs((a - b).cross(a - o)) / (a - b).length();
139      }
140      // 点在直线上的投影
141      Point<T> projection(const Point<T>& o) const {
142          return a + (b - a) * (1. * (b - a).dot(o - a) / (b - a)
143              .square());
144      }
145      // Segment
146      -----

147      // -1 点在线段端点 / 0 点不在线段上 / 1 点严格在线段上
148      int contain(const Point<T>& o) const {
149          if(o == a or o == b) return -1;
150          return (o - a).toLeft(o - b) == 0 and sgn((o - a).dot(o
151              - b)) < 0;
152      }
153      // 判断线段直线是否相交
154      // 0 线段和直线不相交 / 1 线段和直线严格相交 / 2 仅在某一线
155          段端点处相交 / 3 直线包含线段
156      int interWithLine(const Line& l) const {
157          int num = !l.toLeft(a) + !l.toLeft(b);

```

```

153         if(num) return num + 1;
154         return l.toLeft(a) != l.toLeft(b);
155     }
156     // 判断两线段是否相交
157     // 0 两线段不相交 / 1 两线段严格相交 / 2 仅在某一线段端点处
        相交 / 3 两线段有重叠
158     int interWithSegment(Line s) const {
159         if((a < b) != (s.a < s.b))
160             swap(s.a, s.b);
161         int num = (contain(s.a) != 0) + (contain(s.b) != 0)
162             + (s.contain(a) != 0) + (s.contain(b) != 0);
163         if(parallel(s)) {
164             if(!num) return 0;
165             if(b == s.a or a == s.b) return 2; // -.-
166             return 3;
167         }
168         if(num) return 2;
169         return toLeft(s.a) * toLeft(s.b) == -1 and s.toLeft(a)
            * s.toLeft(b) == -1;
170     }
171     // 点到线段的距离
172     double distanceSP(const Point<T>& o) const {
173         if(sgn((o - a).dot(b - a)) < 0) return o.distance(a);
174         if(sgn((o - b).dot(a - b)) < 0) return o.distance(b);
175         return abs((a - b).cross(a - o)) / (a - b).length();
176     }
177     // 两线段间距离
178     double distanceSS(const Line& s) const {
179         if(interWithSegment(s)) return 0;
180         return min({distanceSP(s.a), distanceSP(s.b),
181             s.distanceSP(a), s.distanceSP(b)});
182     }
183 };
184 template<class T>
185 struct Polygon {
186     int n;
187     vector<Point<T>> p;
188     // p 以逆时针顺序存储 2 遍
189     Polygon(vector<Point<T>> const& p_) : n(p_.size()), p(p_) {

```

```

190         p.insert(p.end(), p_.begin(), p_.end());
191     }
192     // 返回 回转数 = 逆时针转头圈数 - 顺时针转头圈数
193     // 1e9 在 多边形上 / 0 不在多边形内 / !=0 在 多边形内
194     int contain(const Point<T>& o) const {
195         int cnt = 0;
196         for(int i = 0; i < n; ++i) {
197             Point<T> const& u = p[i], v = p[i + 1];
198             Line<T> const l(u, v);
199             if(l.contain(o)) return 1e9;
200             cnt += l.toLeft(o) > 0 and sgn(u.y - o.y) < 0 and
                sgn(v.y - o.y) >= 0;
201             cnt -= l.toLeft(o) < 0 and sgn(u.y - o.y) >= 0 and
                sgn(v.y - o.y) < 0;
202         }
203         return cnt;
204     }
205     // 多边形面积的两倍，可用于判断点的存储顺序是顺时针或逆时针
        (逆正顺负)
206     T area() const {
207         T sum = 0;
208         for(int i = 0; i < n; ++i)
209             sum += p[i].cross(p[i + 1]);
210         return sum;
211     }
212     // 多边形的周长
213     double perimeter() const {
214         double sum = 0;
215         for(int i = 0; i < n; ++i)
216             sum += p[i].distance(p[i + 1]);
217         return sum;
218     }
219 };
220
221 template<class T>
222 struct Convex : Polygon<T> {
223     using Polygon<T>::n;
224     using Polygon<T>::p;

```



```

225     Convex(vector<Point<T>> const& p, bool keepRaw) : Polygon<T
        >(p) {}
226     Convex(vector<Point<T>> const& p_) : Polygon<T>(andrew(p_))
        {}
227     // 对点集 p 求凸包
228     static auto andrew(vector<Point<T>> p) {
229         sort(p.begin(), p.end());
230         p.erase(unique(p.begin(), p.end()), p.end());
231         if(p.size() <= 1) return p;
232         vector<Point<T>> st;
233         for(auto& e : p) {
234             while(st.size() > 1 and
235                 (st.back() - st.end()[-2]).ToLeft(e - st.back())
236                 <= 0)
237                 st.pop_back();
238             st.push_back(e);
239         }
240         int sz = st.size();
241         for(int i = (int)p.size() - 2; i >= 0; --i) {
242             while(st.size() > sz and
243                 (st.back() - st.end()[-2]).ToLeft(p[i] - st.
244                 back()) <= 0)
245                 st.pop_back();
246             st.push_back(p[i]);
247         }
248         st.pop_back();
249         return st;
250     }
251     // O(logn)判断点是否在凸多边形内
252     // -1 在边界上 / 0 在外部 / 1 严格在内部
253     int contain(const Point<T>& o) const {
254         if(n == 1) return p[0] == o ? -1 : 0;
255         int fTo = (p[1] - p[0]).ToLeft(o - p[0]);
256         int bTo = (p.back() - p[0]).ToLeft(o - p[0]);
257         if(fTo == -1 or bTo == 1) return 0;
258         if(fTo == 0) return sgn((o - p[0]).dot(o - p[1])) <= 0
            ? -1 : 0;
259         if(bTo == 0) return sgn((o - p[0]).dot(o - p.back()))
            <= 0 ? -1 : 0;

```

```

258
259     int i = partition_point(p.begin() + 2, p.begin() + n,
260                             [&](Point<T> const& v) {
261                                 return (v - p[0]).ToLeft(o - p[0]) >= 0;
262                             }) - p.begin();
263     Line<T> const l(p[i - 1], p[i]);
264     return l.contain(o) ? -1 : l.ToLeft(o) == 1;
265 }
266 //  $O(\log n)$  二分找到  $f$  方向上的切点  $i$ , 满足  $p[i]-p[i-1], f(p[i]), p[i+1]-p[i]$  逆时旋转
267 template<class Func>
268 int extreme(Func const& f) {
269     assert(n > 2);
270     Point<ll> const divVec = f(p[0]);
271     bool const flag = (p[0] - p[n - 1]).ToLeft(divVec) < 0;
272     return partition_point(p.begin(), p.begin() + n, [&](
273         Point<T> const& a) {
274         if(divVec.ToLeft(a - p[0]) > 0) return flag;
275         return (*(a + 1) - a).ToLeft(f(a)) > 0;
276     }) - p.begin();
277 }
278 //  $O(\log n)$  二分找到  $v$  方向 和  $-v$  方向上的切点, 返回值切点下
279 // 标 in  $[0, n-1]$ 
280 array<int, 2> tangentByLine(Point<T> const& v) {
281     int i = extreme([&](...) {return v; });
282     int j = extreme([&](...) {return -v; });
283     return {i, j};
284 }
285 //  $O(\log n)$  过点  $o$  向凸包做两条切线 (先左后右), 返回值切点
286 // 下标 in  $[0, n-1]$ 
287 // 需要保证  $o$  在凸包外面
288 array<int, 2> tangentByPoint(Point<T> const& o) {
289     int i = extreme([&](Point<T> const& a) {return o - a;
290         });
291     int j = extreme([&](Point<T> const& a) {return a - o;
292         });
293     return {i, j};
294 }
295 };

```

## 7.2 线段在多边形内

```

1 // struct Polygon {
2 // O(|p|) 判断线段在多边形内
3 // 可以用整型判断 / 如果用浮点型，可能要把精度调松一点
4 bool contain(const Line<T>& s) const {
5     if(!contain(s.a) or !contain(s.b))
6         return false;
7     if(s.a == s.b)
8         return true;
9     vector<int> t(p.size());
10    for(int i=0; i<p.size(); ++i) {
11        auto& u = p[i];
12        auto& v = p[nxt(i)];
13        Line<T> uv(u,v);
14        t[i] = s.interWithSegment(uv);
15        if(t[i] == 0) continue; // not intersect
16        if(t[i] == 1) return false; // strickly intersect
17        if(t[i] == 2 and uv.contain(s.a)==1 and uv.toLeft(s.b)
18            ==-1)
19            return false;
20        if(t[i] == 3) { // overlap
21            if(s.contain(v)==1 and uv.toLeft(p[nxt(nxt(i))])
22                ==1)
23                return false;
24            if(s.contain(u)==1 and uv.toLeft(p[pre(i)])==1)
25                return false;
26        }
27    }
28    for(int i=0; i<p.size(); ++i) {
29        if(!(t[i]==2 and t[nxt(i)]==2)) continue;
30        auto& v = p[nxt(i)];
31        // intersect at v
32        if(s.contain(v) and s.b != v) {
33            auto& u = p[i];
34            auto& w = p[nxt(nxt(i))];
35            if((v-u).toLeft(w-u)==1) {
36                if(s.toLeft(u)==1 and s.toLeft(w)==-1);
37            }
38            else return false;
39        }
40    }
41    return true;
42 }
```

```
36         }else {
37             if(s.toLeft(u)==-1 and s.toLeft(w)==1)
38                 return false;
39         }
40     }
41 }
42 return true;
43 }
```

### 7.3 钝角直角三角形计数问题

```
1  for(int k=0; k<n; ++k) {
2      vector<Point<ll>> b;
3      b.reserve(n-1);
4      for(int i=0; i<n; ++i) {
5          if(a[i] == a[k]) continue;
6          b.emplace_back(a[i]-a[k]);
7      }
8      sort(ALL(b), Point<ll>::argcmp);
9      int sz = b.size();
10     if(!sz) continue;
11     b.insert(b.end(), b.begin(), b.end());
12     for(int i=0, l=0, r=0; i<sz; ++i) {
13         // 0
14         auto eq0 = [&](int j) {
15             return b[i].cross(b[j])==0 and b[i].dot(b[j])>0;
16         };
17         // [0, 89]
18         auto le89 = [&](int j) {
19             return b[i].cross(b[j])>=0 and b[i].dot(b[j])>0;
20         };
21         // [0, 180)
22         auto le179 = [&](int j) {
23             int t = b[i].toLeft(b[j]);
24             return t>0 or t==0 and b[i].dot(b[j])==0;
25         };
26         // [l, r) -> [90, 180)
27         l = max(l, i);
28         while(l<i+sz and le89(l)) l++;
```

```

29         r = max(r, l);
30         while(r<i+sz and le179(r) and !eq0(r)) r++;
31         ans += r-l;
32     }
33 }

```

## 7.4 向量夹角

```

1 double alpha = atan2(v.cross(w), v.dot(w)); // v, w 不是零向量

```

## 7.5 凸包上旋转卡尺算法的其他应用

```

1 // struct Convex {
2 // 旋转卡尺求直径的平方
3 T rotatingCalipers() const {
4     if(p.size()==1) return 0;
5     if(p.size()==2) return p[0].interSquare(p[1]);
6     T ans = 0;
7     for(int i=0, j=1; i<p.size(); ++i) {
8         Point<T> v = p[nxt(i)]-p[i];
9         while(v.cross(p[nxt(j)]-p[i]) > v.cross(p[j]-p[i]))
10             j=nxt(j);
11         ans = max({ans, p[i].interSquare(p[j]),
12                     p[nxt(i)].interSquare(p[j])});
13         if(v.cross(p[nxt(j)]-p[i]) == v.cross(p[j]-p[i]))
14             ans = max({ans, p[i].interSquare(p[nxt(j)]),
15                         p[nxt(i)].interSquare(p[nxt(j)])});
16     }
17     return ans;
18 }
19 // 结论：覆盖凸包的最小面积/周长矩形，一定有一条边和凸包某条边
    重叠
20 // 旋转卡尺求最小面积矩形
21 double minErea() const {
22     if(p.size()<=2) return 0;
23     double ans = numeric_limits<double>::max();
24     for(int i=0, j=1, k=1, l=1; i<p.size(); ++i) {
25         Point<T> v = p[nxt(i)]-p[i];

```

```

26         while(v.cross(p[nxt(j)]-p[i]) > v.cross(p[j]-p[i])) j =
           nxt(j);
27         while(v.dot(p[nxt(k)]-p[i]) > v.dot(p[k]-p[i])) k = nxt
           (k);
28         if(!i) l = j;
29         while(v.dot(p[nxt(l)]-p[i]) < v.dot(p[l]-p[i])) l = nxt
           (l);
30         ans = min(ans, 1. * v.cross(p[j]-p[i]) / v.square()
31                 * (v.dot(p[k]-p[i]) - v.dot(p[l]-p[i]))
32                 );
33     }
34     return ans;
35 }
36 // 旋转卡尺求最小宽度
37 double minWidth() const {
38     if(p.size()<=2) return 0;
39     double ans = numeric_limits<double>::max();
40     for(int i=0, j=1; i<p.size(); ++i) {
41         Point<T> v = p[nxt(i)]-p[i];
42         while(v.cross(p[nxt(j)]-p[i]) > v.cross(p[j]-p[i])) j =
           nxt(j);
43         ans = min(ans, v.cross(p[j]-p[i])/v.length());
44     }
45     return ans;
46 }
47 // 计算两个相离的凸包之间的最短距离，注意调用两次取min
48 double distance(const Convex<double>& B) {
49     double ans = numeric_limits<double>::max();
50     for(int i=0, j=0; i<p.size(); ++i) {
51         Point<double> v(p[nxt(i)]-p[i]);
52         Line<double> s(p[i], p[nxt(i)]);
53         if(i == 0) {
54             double mx = numeric_limits<double>::min();
55             for(int k=0; k<B.p.size(); ++k) {
56                 double cro = v.cross(B.p[k]-p[i]);
57                 if(sgn(cro-mx)>0) {
58                     mx = cro;
59                     j = k;

```

```

60         }
61         ans = min(ans, s.distanceSP(!j ? B.p.back(): B.p[j
           -1]));
62     }
63     ans = min(ans, s.distanceSP(B.p[j]));
64     while(sgn(v.cross(B.p[B.nxt(j)]-p[i])-v.cross(B.p[j]-p[
           i])) >= 0) {
65         j = B.nxt(j);
66         ans = min(ans, s.distanceSP(B.p[j]));
67     }
68 }
69 return ans;
70 }

```

## 7.6 最大最小三角形

```

1 // 求出的是最大/最小三角形面积的两倍
2 template<class T>
3 pair<T,T> minMaxTriangle(const vector<Point<T>>& a) {
4     int n = a.size();
5     T mn = numeric_limits<T>::max();
6     T mx = 0;
7     using Node = tuple<int,int,Point<T>>;
8     vector<Node> all;
9     all.reserve(n*n);
10    vector<int> id(n), pos(n);
11    for(int i=0; i<n; ++i) {
12        id[i] = i;
13        for(int j=0; j<n; ++j) {
14            if(i==j) continue;
15            if(a[i]==a[j]) mn=0;
16            else all.emplace_back(i,j,a[j]-a[i]);
17        }
18    }
19    sort(all.begin(), all.end(), [&](const Node& x,const Node&
        y) {
20        return Point<T>::argcmp(get<2>(x), get<2>(y));
21    });
22    sort(id.begin(), id.end(), [&](const int& i,const int& j) {

```

```

23         return a[i].y < a[j].y or a[i].y==a[j].y and a[i].x>a[j]
           ].x;
24     });
25     for(int i=0; i<n; ++i)
26         pos[id[i]] = i;
27     for(auto [i,j,v]:all) {
28         // 如果没有三点共线 assert(pos[i] = pos[j] + 1);
29         if(pos[i] > pos[j]) {
30             swap(id[pos[i]], id[pos[j]]);
31             swap(pos[i], pos[j]);
32         }
33         int t = max(pos[i],pos[j])+1;
34         if(t<n) {
35             mn = min(mn, (a[id[t]]-a[i]).cross(v));
36             mx = max(mx, (a[id.back()]-a[i]).cross(v));
37         }
38     }
39     return {mn, mx};
40 }

```

## 7.7 动态凸包

```

1  template<class T>
2  struct DynamicConvex {
3      /// @note operator< 使用极角序, 并考虑极径
4      Point<T> o;
5      set<Point<T>> s; // 坐标扩大三倍, 使得三角形中心为整数
6      using Iter = decltype(s.begin());
7      auto nxt(Iter it) const {
8          return next(it) == s.end() ? s.begin() : next(it);
9      }
10     auto pre(Iter it) const {
11         return it == s.begin() ? --s.end() : prev(it);
12     }
13     bool contain(Point<T> const& a) const {
14         if(s.size() == 0) return 0;
15         if(s.size() == 1) return *s.begin() == a * 3;
16         if(s.size() == 2) return Line<T>(*s.begin(), *s.rbegin
           ()).contain(a);

```



```

17         auto it = s.lower_bound(a * 3 - o);
18         if(it == s.end()) it = s.begin();
19         return (*it - *pre(it)).toLeft(a * 3 - o - *pre(it)) >=
                0;
20     }
21     void add(Point<T> a) {
22         if(s.size() <= 1) {
23             s.insert(a * 3);
24             return;
25         }
26         if(s.size() == 2) {
27             auto u = *s.begin(), v = *s.rbegin();
28             if((u - v).toLeft(a * 3 - v) == 0) return;
29             o = (u + v + a * 3) / 3;
30             s = {u - o, v - o, a * 3 - o};
31             for(auto it = s.begin(); it != s.end(); ++it)
32                 addEdge(it, nxt(it));
33             return;
34         }
35         if(contain(a)) return;
36         a = a * 3 - o;
37         auto it = s.insert(a).first, np = nxt(it), pp = pre(it)
38             ;
39         delEdge(pp, np);
40         while(s.size() > 3 and ((*np - a).toLeft(*nxt(np) - *np
41             )) != 1) {
42             delEdge(np, nxt(np));
43             s.erase(np);
44             np = nxt(it);
45         }
46         while(s.size() > 3 and ((*pp - *pre(pp)).toLeft(a - *pp
47             )) != 1) {
48             delEdge(pre(pp), pp);
49             s.erase(pp);
50             pp = pre(it);
51         }
52         addEdge(pre(it), it);
53         addEdge(it, nxt(it));
54     }

```

```

51
52     double D; // 周长
53     map<Point<T>, Iter> edge;
54     void addEdge(Iter it, Iter nit) { // s.size() >= 3 时维护信息
55         D += it->distance(*nit);
56         edge[*nit - *it] = it;
57     }
58     void delEdge(Iter it, Iter nit) {
59         D -= it->distance(*nit);
60         edge.erase(*nit - *it);
61     }
62     /// @note 调用前注意直线  $ax+by=c$  的坐标扩大三倍  $ax+by=3c$ 
63     array<Point<T>, 2> extremeByLine(Point<T> const& v) const {
64         assert(s.size() > 2);
65         auto get = [&](Point<T> const& v) {
66             auto it = edge.lower_bound(v);
67             if(it == edge.end()) it = edge.begin();
68             return *(it->second) + this->o;
69         };
70         return {get(v), get(-v)};
71     }
72 };

```

## 7.8 闵可夫斯基和

```

1 //  $A+B = \{a+b \mid a \in A, b \in B\}$ , 复杂度  $O(n)$ 
2 template<class T>
3 Convex<T> MinkowskiSum(Convex<T> const& A, Convex<T> const& B)
4 {
5     auto cmp = [&](Point<T> const& a, Point<T> const& b) {
6         return a.y > b.y or a.y == b.y and a.x < b.x;
7     };
8     int a = min_element(A.p.begin(), A.p.begin() + A.n, cmp) -
9         A.p.begin();
10    int b = min_element(B.p.begin(), B.p.begin() + B.n, cmp) -
11        B.p.begin();
12    Point<T> s(A.p[a] + B.p[b]);
13    vector<Point<T>> ps(1, s);
14    auto popC = [&](Point<T> const& e, Point<T> const& f) {

```

```

12         return (e - f).toLeft(s - e) == 0 and sgn((e - f).dot(s
           - e)) >= 0;
13     };
14     auto f = [&](int owner, int i) {
15         return !owner ? A.p[a + i + 1] - A.p[a + i] : B.p[b + i
           + 1] - B.p[b + i];
16     };
17     for(int i = 0, j = 0; i < A.n or j < B.n; ) {
18         if(j >= B.n or i < A.n and Point<T>::argcmp(f(0, i), f
           (1, j))) s += f(0, i++);
19         else s += f(1, j++);
20         while(ps.size() > 1 and popC(ps.back(), ps.end()[-2]))
           ps.pop_back();
21         ps.emplace_back(s);
22     }
23     ps.pop_back();
24     return Convex<T>(ps, true);
25 };

```

## 7.9 半平面交

```

1  template<class T>
2  vector<Line<T>> hp(vector<Line<T>> vs, T inf = T(1e9)) {
3      vs.emplace_back(Point<T>(inf, -inf), Point<T>(inf, inf));
4      vs.emplace_back(Point<T>(inf, inf), Point<T>(-inf, inf));
5      vs.emplace_back(Point<T>(-inf, inf), Point<T>(-inf, -inf));
6      vs.emplace_back(Point<T>(-inf, -inf), Point<T>(inf, -inf));
7      auto sameDir = [&](Line<T> const& a, Line<T> const& b) {
8          return a.parallel(b) and sgn(a.vec().dot(b.vec())) >=
           0;
9      };
10     sort(vs.begin(), vs.end(), [&](Line<T> const& a, Line<T>
           const& b) {
11         if(sameDir(a, b)) return a.toLeft(b.a) == -1;
12         return Point<T>::argcmp(a.vec(), b.vec());
13     });
14     auto canPop = [&](Line<T> const& a, Line<T> const& b, Line<
           T> const& c) {
15         if constexpr(!is_same_v<T, double>) {

```

```

16         __int128_t x = (c.b - c.a).cross(b.a - c.a), y = (c
           .b - c.a).cross(b.a - b.b);
17         using P = Point<__int128_t>;
18         return P(a.vec()).ToLeft(P(b.a) * y + P(b.vec()) *
           x - P(a.a) * y) == -sgn(y);
19     }
20     return Point<double>(a.vec()).ToLeft(b.lineIntersection
           (c) - Point<double>(a.a)) < 0;
21 };
22 deque<Line<T>> q;
23 for(auto& v : vs) {
24     if(q.size() and sameDir(q.back(), v)) continue;
25     while(q.size() > 1 and canPop(v, q.back(), q[q.size() -
           2])) q.pop_back();
26     while(q.size() > 1 and canPop(v, q[0], q[1])) q.
           pop_front();
27     if(q.size() and q.back().vec().ToLeft(v.vec()) <= 0)
           return {};
28     q.push_back(v);
29 }
30 while(q.size() > 1 and canPop(q[0], q.back(), q[q.size() -
           2])) q.pop_back();
31 while(q.size() > 1 and canPop(q.back(), q[0], q[1])) q.
           pop_front();
32 return vector<Line<T>>(q.begin(), q.end());
33 }

```

## 常用 STL

### 8.1 算法库

««««< HEAD

#### 8.1.1 搜索操作

1. *bool all\_of(first, last, compare)*, 判断区间的数是否全符合 *compare*;
2. *bool any\_of(first, last, compare)*, 判断区间的数是否存在一个 *compare*;
3. *bool none\_of(first, last, compare)*, 判断区间的数是否都不符合 *compare*;

表 1: 关键值说明表

值	含义	备注
区间	<code>[first, last)</code>	左闭右开
前缀 <i>val</i>	基本数据类型	与同行中的 <i>val_x</i> 同类型
前缀 <i>T</i>	任意数据类型	模板类
排序	排序操作相关	默认按照 <code>std::less()</code> 排序
<i>compare</i>	自定义比较类型	需重载操作符 <code>()</code> 或使用 lambda 表达式
<i>iterator</i>	迭代器类型	迭代器
<i>dest</i>	容器	目标容器
<i>bid</i>	<code>std::back_inserter(dest)</code>	一个定义
<i>size_t</i>	无符号整型	

4. `iterator find(first, last, val_a)`, 返回区间中第一个等于 *val\_a* 的位置, 否则返回 *last*;
5. `iterator find_if(first, last, compare)`, 返回区间中第一个满足 *compare* 的位置, 否则返回 *last*;
6. `iterator find_if_not(first, last, compare)`, 返回区间中第一个不满足 *compare* 的位置, 否则返回 *last*;
7. `size_t count(first, last, val_a)`, 返回区间中等于 *val\_a* 的个数;
8. `size_t count_if(first, last, compare)`, 返回区间中满足 *compare* 的个数;

=====

### 8.1.2 搜索操作

1. `bool all_of(first, last, compare)`, 判断区间的数是否全符合 *compare*;
2. `bool any_of(first, last, compare)`, 判断区间的数是否存在一个 *compare*;
3. `bool none_of(first, last, compare)`, 判断区间的数是否都不符合 *compare*;
4. `iterator find(first, last, val_a)`, 返回区间中第一个等于 *val\_a* 的位置, 否则返回 *last*;
5. `iterator find_if(first, last, compare)`, 返回区间中第一个满足 *compare* 的位置, 否则返回 *last*;
6. `iterator find_if_not(first, last, compare)`, 返回区间中第一个不满足 *compare* 的位置, 否则返回 *last*;

7. `size_t count(first, last, val_a)`, 返回区间中等于 `val_a` 的个数;
8. `size_t count_if(first, last, compare)`, 返回区间中满足 `compare` 的个数;

»»»> 7666a2125f38494322892b8ec7d7687da1437b6b

### 8.1.3 交换操作

1. `void swap(T_a, T_b)`, 交换 `T_a` 和 `T_b` 之间的数据;

### 8.1.4 生成操作

1. `void fill(first, last, val_a)`, 填充区间中的值为 `val_a`;

### 8.1.5 移除操作

1. `iterator unique(first, last)`, 对区间中重复的元素去重, 返回重复元素的第一个位置;

### 8.1.6 顺序变更操作

1. `void reverse(first, last)`, 翻转区间中的元素;

### 8.1.7 划分操作

1. `iterator partition(first, last, compare)`, 对区间中的数根据 `compare` 进行分割, 并返回不满足 `compare` 位置的起始位置的指针;

### 8.1.8 排序操作

1. `void sort(first, last)`, 对区间的数以非降序排序;
2. `void sort(first, last, compare)`, 对区间范围的数据根据 `compare` 规则进行排序;
3. `bool is_sorted(first, last)`, 判断区间的数是否以非降序排序;
4. `bool is_sorted(first, last, compare)`, 判断区间的数是否按照 `compare` 规则排序;

### 8.1.9 二分搜索操作

1. `iterator lower_bound(first, last, val_a)`, 返回第一个大于等于 `val_a` 元素的指针, 否则返回 `last`;
2. `iterator upper_bound(first, last, val_a)`, 返回第一个大于 `val_a` 元素的指针, 否则返回 `last`;

### 8.1.10 集合操作（在已排序范围上）

1. *iterator set\_union(first1, last1, first2, last2, bid)*, 将范围 1 和范围 2 中的元素取并集放在 *dest* 容器中;
2. *iterator set\_intersection(first1, last1, first2, last2, bid)*, 将范围 1 和范围 2 中的元素取交集放在 *dest* 容器中;
3. *iterator set\_difference(first1, last1, first2, last2, bid)*, 将范围 1 和范围 2 中的元素取差集放在 *dest* 容器中;

### 8.1.11 最小/最大操作

1. *val max(val\_a, val\_b)*, 返回 *val\_a, val\_b* 间的最大;
2. *val min(val\_a, val\_b)*, 返回 *val\_a, val\_b* 间的最小;
3. *iterator min\_element(first, last)*, 返回区间中的最小值的指针;
4. *iterator max\_element(first, last)*, 返回区间中的最大值的指针;

### 8.1.12 排列操作

1. *bool next\_permutation(first, last)*, 对区间中的元素进行全排列, 通常结合 *do\_while()*;

### 8.1.13 数值运算

1. *void iota(first, last, val\_a)*, 对区间的数以初始值为 *val\_a* 按顺序递增填充;
2. *val accumulate(first, last, val\_a)*, 对区间中的数以 *val\_a* 为初始值进行求和并返回;