

恶魔妹妹的算法模板。第几版已经知道了。懒得维护版本号和更新内容了。反正想起来了传一份最新版。

最后，欢迎加群菜菜园子：730602769。

2024年7月25日。

## 编译

```
1  #include<bits/stdc++.h>
2  using namespace std;
3  /*=====*/
4  #define endl "\n"
5  /*=====*/
6  typedef long long lnt;
7  /*=====*/
8  void solve(void)
9  {
10
11 }
12 /*=====*/
13 int main()
14 {
15     #ifndef ONLINE_JUDGE
16         freopen("IN.txt", "r+", stdin);
17     #endif
18     ios::sync_with_stdio(false);
19     cin.tie(NULL), cout.tie(NULL);
20     int T = 1; //cin >> T;
21     while (T--) solve();
22     return 0;
23 }
```

## 表达式

### 获取优先级

```
1  int GetPriority(char c)
2  {
3      if (c == '*') return 0;
4      if (c == '+') return -1;
5      return -2;
6  }
```

### 中缀转后缀

```
1  string PostfixExpression(string str)
2  {
3      string res;
4      stack<char> stk;
```

```

5     for (auto c : str)
6     {
7         if (c == '_')
8         {
9             res.push_back(c);
10        }
11        if (c == '(' || c == ')')
12        {
13            if (c == '(')
14            {
15                stk.push('(');
16            }
17            if (c == ')')
18            {
19                while (!stk.empty() && stk.top() != '(')
20                {
21                    res.push_back(stk.top()); stk.pop();
22                }
23                stk.pop();
24            }
25        }
26        if (c == '+' || c == '*')
27        {
28            while (!stk.empty() && GetPriority(stk.top()) >= GetPriority(c))
29            {
30                res.push_back(stk.top()); stk.pop();
31            }
32            stk.push(c);
33        }
34    }
35    while (!stk.empty())
36    {
37        res.push_back(stk.top()); stk.pop();
38    }
39    return res;
40 }

```

## 建立表达式树

```

1  struct Node
2  {
3      int val;
4      char tag;
5      Node* lch, * rch;
6      Node(int _val = 0, char _tag = ' ', Node* _lch = NULL, Node* _rch =
7      NULL)
8      {
9          val = _val, tag = _tag;
10         lch = _lch, rch = _rch;
11     };
12     Node* Build(string str)
13     {
14         stack<Node*>stk;

```

```

15     for (auto c : str)
16     {
17         if (c == '0' || c == '1')
18         {
19             stk.push(new Node(c - '0', ' ', NULL, NULL));
20         }
21         else
22         {
23             Node* rch = stk.top(); stk.pop();
24             Node* lch = stk.top(); stk.pop();
25             stk.push(new Node(((c == '&') ? (lch->val & rch->val) : (lch-
>val | rch->val)), c, lch, rch));
26         }
27     }
28     return stk.top();
29 }

```

## 读写

### \_int128

```

1  ostream& operator<<(ostream& output, Int integer)
2  {
3      if (integer < 0)
4      {
5          output << "-"; integer *= -1;
6      }
7      string str;
8      do
9      {
10         str.push_back('0' + char(integer % 10)); integer /= 10;
11     } while (integer != 0);
12     reverse(str.begin(), str.end());
13     output << str;
14     return output;
15 }
16 istream& operator>>(istream& input, Int& integer)
17 {
18     string str; input >> str; integer = 0;
19     for (int i = 0; i < str.size(); ++i)
20     {
21         integer = integer * 10 + str[i] - '0';
22     }
23     return input;
24 }

```

## 快读快写

```

1  namespace FastIO
2  {
3      template<int SIZE>
4      class C_IOBUF
5      {

```

```

6     private:
7         char p[SIZE], * p1, * p2, q[SIZE], * q1, * q2;
8     public:
9         C_IOBUF(void) { p1 = p2 = p, q1 = q, q2 = q + SIZE; }
10        char GetChar(void)
11        {
12            return p1 == p2 && (p2 = (p1 = p) + fread(p, 1, SIZE, stdin)),
p1 == p2 ? EOF : *p1++;
13        }
14        void PutChar(char c)
15        {
16            if (q1 == q2) q2 = (q1 = q) + fwrite(q, 1, SIZE, stdout); *q1++
= c;
17        }
18        ~C_IOBUF(void) { fwrite(q, 1, q1 - q, stdout); }
19    };
20    C_IOBUF<1 << 20>BUF;
21    class C_IStream
22    {
23    private:
24        bool IsChar(const char& c)
25        {
26            return c != ' ' && c != '\n';
27        }
28        bool IsDigit(const char& c)
29        {
30            return '0' <= c && c <= '9';
31        }
32    public:
33        C_IStream& operator >>(int& temp)
34        {
35            temp = 0; char c = BUF.GetChar(); bool flag = false;
36            while (IsDigit(c) != true) { if (c == '-')flag = true; c =
BUF.GetChar(); }
37            while (IsDigit(c) == true) { temp = temp * 10 + c - '0', c =
BUF.GetChar(); }
38            temp = flag ? -temp : temp; return *this;
39        }
40        C_IStream& operator >>(Int& temp)
41        {
42            temp = 0; char c = BUF.GetChar(); bool flag = false;
43            while (IsDigit(c) != true) { if (c == '-')flag = true; c =
BUF.GetChar(); }
44            while (IsDigit(c) == true) { temp = temp * 10 + c - '0', c =
BUF.GetChar(); }
45            temp = flag ? -temp : temp; return *this;
46        }
47        C_IStream& operator >>(char& temp)
48        {
49            temp = ' '; char c = BUF.GetChar();
50            while (IsChar(c) != true)c = BUF.GetChar();
51            temp = c; return *this;
52        }
53        C_IStream& operator >>(string& temp)
54        {

```

```

55         temp.clear(); char c = BUF.GetChar();
56         while (IsChar(c) != true)c = BUF.GetChar();
57         while (IsChar(c) == true)temp += c, c = BUF.GetChar();
58         return *this;
59     }
60 }cin;
61 class C_OStream
62 {
63 public:
64     C_OStream& operator <<(int temp)
65     {
66         int Top = 0; static int Stack[64];
67         if (temp < 0) { BUF.PutChar('-'); temp = -temp; }
68         do { Stack[Top++] = temp % 10; temp /= 10; } while (temp);
69         while (Top) { BUF.PutChar(Stack[--Top] + '0'); } return *this;
70     }
71     C_OStream& operator <<(Int temp)
72     {
73         int Top = 0; static int Stack[64];
74         if (temp < 0) { BUF.PutChar('-'); temp = -temp; }
75         do { Stack[Top++] = temp % 10; temp /= 10; } while (temp);
76         while (Top) { BUF.PutChar(Stack[--Top] + '0'); } return *this;
77     }
78     C_OStream& operator <<(char temp)
79     {
80         BUF.PutChar(temp); return *this;
81     }
82     C_OStream& operator <<(string temp)
83     {
84         for (auto c : temp)BUF.PutChar(c); return *this;
85     }
86     C_OStream& operator <<(const char temp[])
87     {
88         int p = 0; while (temp[p] != '\0')BUF.PutChar(temp[p++]); return
89 *this;
90     }
91 }cout;
92 }

```

## 枚举

## 排列

## 全排列

```

1 void AllPermutation(int l, int r)
2 {
3     vector<int>p;
4     for (int i = l; i <= r; ++i)
5     {
6         p.push_back(i);
7     }
8     do
9     {
10         //do something
11     } while (next_permutation(p.begin(), p.end()));
12 }

```

## 子集

### 枚举全部子集

```

1 void AllSubset(int s)
2 {
3     for (int i = s; i; i = (i - 1) & s)
4     {
5         //do something
6     }
7 }

```

### 枚举k大小子集

```

1 void GospersHack(int k, int n)
2 {
3     int cur = (1 << k) - 1, limit = (1 << n);
4     while (cur < limit)
5     {
6         // do something
7         int lb = cur & -cur, r = cur + lb;
8         cur = (((r ^ cur) >> 2) / lb) | r;
9     }
10 }

```

## 树论

### LCT

```

1 class C_LinkCutTree
2 {
3 public:
4     int rev[N], fa[N], ch[N][2];
5     /*=====*/
6     bool which(int x)
7     {
8         return ch[fa[x]][1] == x;
9     }

```

```

10  bool IsRoot(int x)
11  {
12      return ch[fa[x]][which(x)] != x;
13  }
14  /*=====*/
15  void PushUp(int x)
16  {
17      /*PushUp*/
18  }
19  void PushAll(int x)
20  {
21      if (!IsRoot(x))
22      {
23          PushAll(fa[x]);
24      }
25      PushDown(x);
26  }
27  void PushDown(int x)
28  {
29      if (rev[x])
30      {
31          swap(ch[x][0], ch[x][1]);
32          rev[ch[x][0]] ^= 1;
33          rev[ch[x][1]] ^= 1;
34          rev[x] = 0;
35      }
36      /*PushDown*/
37  }
38  /*=====*/
39  void Rotate(int x)
40  {
41      int y = fa[x], z = fa[y], w = which(x);
42      if (!IsRoot(y)) ch[z][which(y)] = x; fa[x] = z;
43      ch[y][w] = ch[x][w ^ 1];
44      if (ch[y][w]) fa[ch[y][w]] = y;
45      ch[x][w ^ 1] = y; fa[y] = x;
46      PushUp(y); PushUp(x);
47  }
48  void Splay(int x)
49  {
50      PushAll(x);
51      for (; !IsRoot(x); Rotate(x))
52      {
53          if (!IsRoot(fa[x]))
54          {
55              Rotate(which(x) == which(fa[x]) ? fa[x] : x);
56          }
57      }
58  }
59  /*=====*/
60  void Access(int x)
61  {
62      for (int p = 0; x; p = x, x = fa[x])
63      {
64          Splay(x), ch[x][1] = p, PushUp(x);

```

```

65     }
66 }
67 /*=====*/
68 int FindRoot(int x)
69 {
70     Access(x); Splay(x);
71     while (ch[x][0]) x = ch[x][0];
72     Splay(x); return x;
73 }
74 void MakeRoot(int x)
75 {
76     Access(x); Splay(x); rev[x] ^= 1;
77 }
78 /*=====*/
79 void Cut(int u, int v)
80 {
81     Split(u, v);
82     if (fa[u] == v && !ch[u][1])
83     {
84         ch[v][0] = fa[u] = 0;
85     }
86 }
87 void Link(int u, int v)
88 {
89     MakeRoot(u); fa[u] = v;
90 }
91 /*=====*/
92 void Split(int u, int v)
93 {
94     MakeRoot(u); Access(v); Splay(v);
95 }
96 /*=====*/
97 int LCA(int u, int v)
98 {
99     Access(u); int ans = 0;
100    for (int child = 0; v; child = v, v = fa[v])
101    {
102        Splay(v); ch[v][1] = child; ans = v;
103    }
104    return ans;
105 }
106 };

```

## LCA

### 树剖法

```

1  class C_LCA
2  {
3  private:
4      struct Node
5      {
6          int pre, dep, siz, son, top;
7          Node(void)

```



```

8      {
9          pre = -1, dep = +0, siz = +1, son = -1, top = -1;
10     }
11 };
12 /*=====*/
13 int root;
14 vector<int>* G;
15 vector<Node>node;
16 /*=====*/
17 void DFS1(int pre, int cur)
18 {
19     node[cur].pre = pre;
20     node[cur].dep = node[pre].dep + 1;
21     for (auto nxt : G[cur])
22     {
23         if (nxt != pre)
24         {
25             DFS1(cur, nxt);
26             node[cur].siz += node[nxt].siz;
27             if (node[cur].son == -1)
28             {
29                 node[cur].son = nxt;
30             }
31             else if (node[nxt].siz > node[node[cur].son].siz)
32             {
33                 node[cur].son = nxt;
34             }
35         }
36     }
37 }
38 void DFS2(int cur, int top)
39 {
40     node[cur].top = top;
41     if (node[cur].son != -1)
42     {
43         DFS2(node[cur].son, top);
44         for (auto nxt : G[cur])
45         {
46             if (nxt == node[cur].pre)continue;
47             if (nxt == node[cur].son)continue;
48             DFS2(nxt, nxt);
49         }
50     }
51 }
52 public:
53 void Init(int n, vector<int>G[], int root = 1)
54 {
55     this->G = G;
56     this->root = root;
57     node.assign(n + 1, Node());
58     DFS1(root, root); DFS2(root, root);
59 }
60 int operator()(int u, int v)
61 {
62     while (node[u].top != node[v].top)

```

```

63     {
64         int topu = node[u].top;
65         int topv = node[v].top;
66         if (node[topu].dep > node[topv].dep)
67         {
68             u = node[topu].pre;
69         }
70         else
71         {
72             v = node[topv].pre;
73         }
74     }
75     return node[u].dep > node[v].dep ? v : u;
76 }
77 };

```

## ST表法

```

1  class C_LCA
2  {
3  private:
4      vector<int>* G;
5      vector<int>dfn;
6      vector<vector<int>>>table;
7      /*=====*/
8      void DFS(int pre, int cur)
9      {
10         table[0][dfn[cur] = ++dfn[0]] = pre;
11         for (auto nxt : G[cur])if (nxt != pre)DFS(cur, nxt);
12     }
13     int Get(int x, int y)
14     {
15         return dfn[x] < dfn[y] ? x : y;
16     }
17 public:
18     void Init(int n, vector<int>G[], int root = 1)
19     {
20         this->G = G; dfn.assign(n + 1, 0);
21         table.assign(__lg(n) + 1, vector<int>(n + 1, 0));
22         /*=====*/
23         DFS(0, root);
24         for (int d = 1; (1 << d) <= n; ++d)
25         {
26             for (int i = 1; i + (1 << d) - 1 <= n; ++i)
27             {
28                 table[d][i] = Get(table[d - 1][i], table[d - 1][i + (1 << (d
29 - 1))]);
30             }
31         }
32     int operator()(int u, int v)
33     {
34         if (u == v)return u;
35         if ((u = dfn[u]) > (v = dfn[v]))swap(u, v);

```

```

36         int d = __lg(v - u++); return Get(table[d][u], table[d][v - (1 << d)
+ 1]);
37     }
38 };

```

## 树哈希

```

1  class C_TreeHash
2  {
3  private:
4      vector<int>* G;
5      map<vector<int>, int>mp;
6      /*=====*/
7      int DFS(int pre, int cur)
8      {
9          vector<int>vec;
10         for (auto nxt : G[cur])
11         {
12             if (nxt != pre)
13             {
14                 vec.push_back(DFS(cur, nxt));
15             }
16         }
17         sort(vec.begin(), vec.end());
18         if (mp.find(vec) == mp.end())
19         {
20             mp[vec] = mp.size();
21         }
22         return mp[vec];
23     }
24 public:
25     int operator()(vector<int>G[], int root)
26     {
27         this->G = G;
28         return DFS(root, root);
29     }
30 };

```

## 树分治

### 点分治

```

1  class C_TCD
2  {
3  private:
4      vector<int>* G;
5      /*=====*/
6      vector<int>siz;
7      vector<bool>rooted;
8      /*=====*/
9      int centroid, ans_part;
10     /*=====*/
11     void DFS(int pre, int cur, int all)

```

```

12     {
13         siz[cur] = 1; int max_part = 0;
14         for (auto nxt : G[cur])
15         {
16             if (nxt == pre) continue;
17             if (rooted[nxt]) continue;
18             DFS(cur, nxt, all);
19             siz[cur] += siz[nxt];
20             max_part = max(max_part, siz[nxt]);
21         }
22         max_part = max(max_part, all - siz[cur]);
23         if (max_part < ans_part)
24         {
25             ans_part = max_part, centroid = cur;
26         }
27     }
28     int Centroid(int cur, int all)
29     {
30         centroid = -1; ans_part = all;
31         DFS(cur, cur, all); return centroid;
32     }
33     /*=====*/
34     void CalcSon(int pre, int cur)
35     {
36         /*
37          添加cur到右树tree中
38         */
39         for (auto nxt : G[cur])
40         {
41             if (nxt == pre) continue;
42             if (rooted[nxt]) continue;
43             CalcSon(cur, nxt);
44         }
45     }
46     void CalcRoot(int root)
47     {
48         /*
49          初始化左库lib
50         */
51         for (auto son : G[root])
52         {
53             if (!rooted[son])
54             {
55                 /*
56                  初始化右树tree
57                 */
58                 CalcSon(son, son);
59                 /*
60                  遍历右树tree匹配左库lib
61                 */
62                 /*
63                  添加右树tree到左库lib
64                 */
65             }
66         }

```

```

67     }
68     /*=====*/
69     void DividTree(int root)
70     {
71         rooted[root] = true; CalcRoot(root);
72         for (auto son : G[root])
73         {
74             if (!rooted[son])
75             {
76                 DividTree(Centroid(son, siz[son]));
77             }
78         }
79     }
80 public:
81     void operator()(int n, vector<int>G[])
82     {
83         this->G = G;
84         siz.assign(n + 1, 0);
85         rooted.assign(n + 1, false);
86         DividTree(Centroid(1, n));
87     }
88 };

```

## K级祖先

```

1  class C_KthAncestor
2  {
3  private:
4      int n, root; vector<int>* G;
5      /*=====*/
6      struct Node
7      {
8          int pre, dep, siz, son;
9          int top, dfn, idx;
10         Node(void)
11         {
12             pre = -1; dep = +0;
13             siz = +1; son = -1;
14             dfn = +0, idx = +0;
15             top = -1;
16         }
17     };
18     /*=====*/
19     vector<Node>node; int cnt;
20     /*=====*/
21     void DFS1(int pre, int cur)
22     {
23         node[cur].pre = pre;
24         node[cur].dep = node[pre].dep + 1;
25         for (auto nxt : G[cur])
26         {
27             if (nxt != pre)
28             {
29                 DFS1(cur, nxt); node[cur].siz += node[nxt].siz;

```

```

30         if (node[cur].son == -1)
31         {
32             node[cur].son = nxt;
33         }
34         else if (node[nxt].siz > node[node[cur].son].siz)
35         {
36             node[cur].son = nxt;
37         }
38     }
39 }
40 }
41 void DFS2(int cur, int top)
42 {
43     node[cur].top = top;
44     node[cur].dfn = ++cnt;
45     node[cnt].idx = cur;
46     if (node[cur].son != -1)
47     {
48         DFS2(node[cur].son, top);
49         for (auto nxt : G[cur])
50         {
51             if (nxt == node[cur].pre) continue;
52             if (nxt == node[cur].son) continue;
53             DFS2(nxt, top);
54         }
55     }
56 }
57 public:
58 void Init(int n, vector<int>G[], int root = 1)
59 {
60     node.assign(n + 1, Node());
61     this->n = n, this->root = root, this->G = G;
62     cnt = 0; DFS1(root, root); DFS2(root, root);
63 }
64 int operator()(int x, int k)
65 {
66     int topx = node[x].top;
67     while (k > 0)
68     {
69         topx = node[x].top;
70         if (node[x].dep - node[topx].dep < k)
71         {
72             k -= node[x].dep - node[topx].dep + 1;
73             x = node[topx].pre;
74         }
75         else
76         {
77             x = node[node[x].dfn - k].idx; k = 0;
78         }
79     }
80     return x;
81 }
82 };

```

## 树链剖分

## 重链剖分

```
1  class C_HLD
2  {
3  public:
4      int n, root; vector<int>* G;
5      /*=====*/
6      struct Node
7      {
8          int pre, dep, siz, son;
9          int top, dfn, idx;
10         Node(void)
11         {
12             pre = -1; dep = +0;
13             siz = +1; son = -1;
14             dfn = +0, idx = +0;
15             top = -1;
16         }
17     };
18     private:
19         vector<Node>node; int cnt;
20         /*=====*/
21         void DFS1(int pre, int cur)
22         {
23             node[cur].pre = pre;
24             node[cur].dep = node[pre].dep + 1;
25             for (auto nxt : G[cur])
26             {
27                 if (nxt != pre)
28                 {
29                     DFS1(cur, nxt); node[cur].siz += node[nxt].siz;
30                     if (node[cur].son == -1)
31                     {
32                         node[cur].son = nxt;
33                     }
34                     else if (node[nxt].siz > node[node[cur].son].siz)
35                     {
36                         node[cur].son = nxt;
37                     }
38                 }
39             }
40         }
41         void DFS2(int cur, int top)
42         {
43             node[cur].top = top;
44             node[cur].dfn = ++cnt;
45             node[cnt].idx = cur;
46             if (node[cur].son != -1)
47             {
48                 DFS2(node[cur].son, top);
49                 for (auto nxt : G[cur])
50                 {
51                     if (nxt == node[cur].pre)continue;
52                     if (nxt == node[cur].son)continue;
53                     DFS2(nxt, nxt);
```

```

54         }
55     }
56 }
57 public:
58     void Init(int n, vector<int>G[], int root = 1)
59     {
60         node.assign(n + 1, Node());
61         this->n = n, this->root = root, this->G = G;
62         cnt = 0; DFS1(root, root); DFS2(root, root);
63     }
64     Node& operator[](int idx) { return node[idx]; }
65 };

```

## 树的重心

```

1  class C_TreeCentroid
2  {
3  private:
4      vector<int>siz;
5      vector<int>* G;
6      int centroid, ans_part;
7      /*=====*/
8      void DFS(int pre, int cur, int all)
9      {
10         siz[cur] = 1; int max_part = 0;
11         for (auto nxt : G[cur])
12         {
13             if (nxt != pre)
14             {
15                 DFS(cur, nxt, all);
16                 siz[cur] += siz[nxt];
17                 max_part = max(max_part, siz[nxt]);
18             }
19         }
20         max_part = max(max_part, all - siz[cur]);
21         if (max_part < ans_part)
22         {
23             ans_part = max_part, centroid = cur;
24         }
25     }
26 public:
27     int operator()(int n, vector<int>G[])
28     {
29         this->G = G; siz.assign(n + 1, 0);
30         centroid = -1; ans_part = n;
31         DFS(1, 1, n); return centroid;
32     }
33 };

```

## 树上路径求交



假设当前要求路径  $(a, b)$  和  $(c, d)$  的交。

设  $d_x$  表示  $x$  的深度。

先求出  $p[4] = lca(a, c), lca(a, d), lca(b, c), lca(b, d)$ 。

将  $p$  数组按深度排序，取出深度较大的两个，记为  $p_0, p_1$ 。

若存在交，则  $(p_0, p_1)$  即所求。

现在只需要判断路径是否有交。

若  $p_0 \neq p_1$ ，则一定有交。

否则若  $d_{p_0} = \max(d_{lca(a,b)}, d_{lca(c,d)})$ ，也有交。

否则路径不相交。

## 树上启发式合并

对于以  $u$  为根的子树

- ①. 先统计它轻子树(轻儿子为根的子树)的答案，统计完后删除信息
- ②. 再统计它重子树(重儿子为根的子树)的答案，统计完后保留信息
- ③. 然后再将重子树的信息合并到  $u$  上
- ④. 再去遍历  $u$  的轻子树，然后把轻子树的信息合并到  $u$  上
- ⑤. 判断  $u$  的信息是否需要传递给它的父节点 ( $u$  是否是它父节点的重儿子)

```
1 void DFS(int root, int cur)
2 {
3     cnt[node[cur].val]++;
4     if (cnt[node[cur].val] > maxcnt)
5     {
6         ans[root] = node[cur].val;
7         maxcnt = cnt[node[cur].val];
8     }
9     else if (cnt[node[cur].val] == maxcnt)
10    {
11        ans[root] += node[cur].val;
12    }
13    for (auto nxt : G[cur])
14    {
15        if (nxt == node[cur].pre)continue;
16        if (nxt == node[root].son)continue;
17        DFS(root, nxt);
18    }
19 }
20 void DSU(int cur, bool keep)
21 {
22     for (auto nxt : G[cur])
23     {
24         if (nxt == node[cur].pre)continue;
25         if (nxt == node[cur].son)continue;
26         DSU(nxt, false);
27     }
28     if (node[cur].son != -1)DSU(node[cur].son, true);
29     if (node[cur].son != -1)
30     {
31         ans[cur] = ans[node[cur].son];
32     }
```

```

33     DFS(cur, cur);
34     if (keep == false)
35     {
36         maxcnt = 0;
37         for (int i = node[cur].dfn; i < node[cur].dfn + node[cur].siz; ++i)
38         {
39             cnt[node[node[i].idx].val]--;
40         }
41     }
42 }

```

## 数据结构

### 莫队

```

1  int n, m;
2  /*=====*/
3  int s;
4  struct Query
5  {
6      int l, r, idx;
7      Query(int _l = 0, int _r = 0, int _idx = 0)
8      {
9          l = _l, r = _r, idx = _idx;
10     }
11     friend bool operator<(const Query& a, const Query& b)
12     {
13         return (a.l / s == b.l / s) ? (((a.l / s) & 1) ? (a.r > b.r) : (a.r
14         < b.r)) : (a.l < b.l);
15     }
16 }query[M];
17 /*=====*/
18 int ans[M];
19 /*=====*/
20 void Add(int pos)
21 {
22 }
23 void Del(int pos)
24 {
25 }
26 }
27 /*=====*/
28 void Solve(void)
29 {
30     cin >> n >> m;
31     s = n / sqrt(m) + 1;
32     for (int i = 1; i <= m; ++i)
33     {
34         int l, r; cin >> l >> r;
35         query[i] = Query(l, r, i);
36     }
37     sort(query + 1, query + 1 + m);
38     int l = 1, r = 0;

```

```

39     for (int i = 1; i <= m; ++i)
40     {
41         while (query[i].l < l)Add(--l);
42         while (r < query[i].r)Add(++r);
43         while (l < query[i].l)Del(l++);
44         while (query[i].r < r)Del(r--);
45         //获得ans[query[i].idx];
46     }
47 }

```

## 猫树

```

1  #include<iostream>
2  using namespace std;
3
4  const int N = 1 << 20;
5  const int LEVEL = 20;
6
7  int a[N];
8  int lg[N];
9  int pos[N];
10 int MaoA[LEVEL][N]; //最大子段和
11 int MaoB[LEVEL][N]; //最大连续和
12
13 inline int ls(int p) { return p << 1; }
14 inline int rs(int p) { return (p << 1) | 1; }
15
16 void Build(int p, int l, int r, int level)
17 {
18     if (l == r) { pos[l] = p; return; }
19     int mid = (l + r) >> 1;
20     int tempA, tempB;
21     //the left
22     MaoA[level][mid] = MaoB[level][mid] = a[mid];
23     tempA = max(a[mid], 0), tempB = a[mid];
24     for (int i = mid - 1; i >= l; --i)
25     {
26         tempA += a[i]; MaoA[level][i] = max(MaoA[level][i + 1], tempA);
27         tempA = max(tempA, 0);
28         tempB += a[i]; MaoB[level][i] = max(MaoB[level][i + 1], tempB);
29     }
30     //the right
31     MaoA[level][mid + 1] = MaoB[level][mid + 1] = a[mid + 1];
32     tempA = max(a[mid + 1], 0), tempB = a[mid + 1];
33     for (int i = mid + 2; i <= r; ++i)
34     {
35         tempA += a[i]; MaoA[level][i] = max(MaoA[level][i - 1], tempA);
36         tempA = max(tempA, 0);
37         tempB += a[i]; MaoB[level][i] = max(MaoB[level][i - 1], tempB);
38     }
39     //
40     Build(ls(p), l, mid, level + 1); Build(rs(p), mid + 1, r, level + 1);
41 }
42 int Ask(int l, int r)

```

```

41 {
42     if (l == r) return a[l];
43     int level = (lg[pos[l]] - lg[pos[l] ^ pos[r]]);
44     return max(max(MaoA[level][l], MaoA[level][r]), MaoB[level][l] +
MaoB[level][r]);
45 }
46
47 int main()
48 {
49     int n; cin >> n;
50     int len = 1; while (len < n) len <<= 1;
51     for (int i = 2; i < N; ++i) lg[i] = lg[i >> 1] + 1;
52     for (int i = 1; i <= n; ++i) cin >> a[i];
53     Build(1, 1, len, 1);
54     int m; cin >> m;
55     for (int i = 1; i <= m; ++i)
56     {
57         int l, r; cin >> l >> r;
58         cout << Ask(l, r) << endl;
59     }
60     return 0;
61 }

```

## ST表

```

1  template<class Type>
2  class C_ST
3  {
4  public:
5      Type operator()(int l, int r)
6      {
7          int d = __lg(r - l + 1); return op(table[d][l], table[d][r - (1 <<
d) + 1]);
8      }
9      void Init(int n, Type arr[], Type(*op)(Type, Type))
10     {
11         this->op = op;
12         /*=====*/
13         table.assign(__lg(n) + 1, vector<Type>(n + 1, Type()));
14         for (int i = 1; i <= n; ++i) table[0][i] = arr[i];
15         /*=====*/
16         for (int d = 1; (1 << d) <= n; ++d)
17         {
18             for (int i = 1; i + (1 << d) - 1 <= n; ++i)
19             {
20                 table[d][i] = op(table[d - 1][i], table[d - 1][i + (1 << (d
- 1))]);
21             }
22         }
23     }
24 private:
25     Type(*op)(Type, Type);
26     vector<vector<Type>> table;
27 };

```

# 扫描线

## 矩形面积并

```
1 namespace ScanLine
2 {
3     const int N = 1e5 + 10;
4     /*=====*/
5     struct Rectangle
6     {
7         double x1, y1;
8         double x2, y2;
9     };
10    Rectangle rectangle[N];
11    /*=====*/
12    vector<double>pos;
13    /*=====*/
14    struct Line
15    {
16        int val;
17        int l, r;
18        double h;
19        Line(int _l = 0, int _r = 0, double _h = 0, int _val = 0)
20        {
21            l = _l, r = _r, h = _h, val = _val;
22        }
23        friend bool operator<(const Line& a, const Line& b)
24        {
25            if (a.h != b.h)
26            {
27                return a.h < b.h;
28            }
29            else
30            {
31                return a.val > b.val;
32            }
33        }
34    };
35    vector<Line>line;
36    /*=====*/
37    struct Tree
38    {
39        int l, r;
40        int cnt; double len;
41    };
42    Tree tree[N << 3];
43    int ls(int p)
44    {
45        return p << 1;
46    }
47    int rs(int p)
48    {
49        return p << 1 | 1;
50    }
```

```

51 void PushUp(int p)
52 {
53     if (tree[p].cnt >= 1)
54     {
55         tree[p].len = pos[tree[p].r] - pos[tree[p].l - 1];
56     }
57     else
58     {
59         if (tree[p].l != tree[p].r)
60         {
61             tree[p].len = tree[ls(p)].len + tree[rs(p)].len;
62         }
63         else
64         {
65             tree[p].len = 0;
66         }
67     }
68 }
69 void Build(int p, int l, int r)
70 {
71     tree[p].l = l, tree[p].r = r;
72     tree[p].cnt = 0; tree[p].len = 0;
73     if (tree[p].l != tree[p].r)
74     {
75         int mid = (tree[p].l + tree[p].r) >> 1;
76         Build(ls(p), l, mid + 0);
77         Build(rs(p), mid + 1, r);
78     }
79 }
80 void Change(int p, int l, int r, int d)
81 {
82     if (l <= tree[p].l && tree[p].r <= r)
83     {
84         tree[p].cnt += d; PushUp(p);
85     }
86     else
87     {
88         int mid = (tree[p].l + tree[p].r) >> 1;
89         if (l <= mid) Change(ls(p), l, r, d);
90         if (mid < r) Change(rs(p), l, r, d);
91         PushUp(p);
92     }
93 }
94 /*=====*/
95 double Init(void)
96 {
97     int n; cin >> n;
98     pos.clear(); line.clear();
99     for (int i = 1; i <= n; ++i)
100     {
101         double x1, y1; cin >> x1 >> y1; //左上
102         double x2, y2; cin >> x2 >> y2; //右下
103         pos.push_back(x1); pos.push_back(x2);
104         rectangle[i].x1 = x1; rectangle[i].y1 = y1;
105         rectangle[i].x2 = x2; rectangle[i].y2 = y2;

```

```

106     }
107     sort(pos.begin(), pos.end());
108     pos.erase(unique(pos.begin(), pos.end()), pos.end());
109     for (int i = 1; i <= n; ++i)
110     {
111         int l = lower_bound(pos.begin(), pos.end(), rectangle[i].x1) -
pos.begin();
112         int r = lower_bound(pos.begin(), pos.end(), rectangle[i].x2) -
pos.begin();
113         line.push_back(Line(l, r, rectangle[i].y1, +1));
114         line.push_back(Line(l, r, rectangle[i].y2, -1));
115     }
116     sort(line.begin(), line.end());
117     Build(1, 1, pos.size() - 1);
118     bool flag = true;
119     double ans = 0.0;
120     double last = 0.0;
121     auto it = line.begin();
122     while (it != line.end())
123     {
124         double high = it->h;
125         if (flag) last = high, flag = false;
126         ans += (high - last) * tree[1].len;
127         while (it != line.end() && it->h == high)
128         {
129             Change(1, it->l + 1, it->r, it->val); it++;
130         }
131         last = high;
132     }
133     return ans;
134 }
135 }

```

## 矩形面积交

```

1  namespace ScanLine
2  {
3      const int N = 1e5 + 10;
4      /*=====*/
5      struct Rectangle
6      {
7          double x1, y1;
8          double x2, y2;
9      };
10     Rectangle rectangle[N];
11     /*=====*/
12     vector<double>pos;
13     /*=====*/
14     struct Line
15     {
16         int val;
17         int l, r;
18         double h;
19         Line(int _l = 0, int _r = 0, double _h = 0, int _val = 0)
20         {

```

```

21         l = _l, r = _r, h = _h, val = _val;
22     }
23     friend bool operator<(const Line& a, const Line& b)
24     {
25         if (a.h != b.h)
26         {
27             return a.h < b.h;
28         }
29         else
30         {
31             return a.val > b.val;
32         }
33     }
34 };
35 vector<Line>line;
36 /*=====*/
37 struct Tree
38 {
39     int cnt;
40     int l, r;
41     double len1;
42     double len2;
43 };
44 Tree tree[N << 3];
45 int ls(int p)
46 {
47     return p << 1;
48 }
49 int rs(int p)
50 {
51     return p << 1 | 1;
52 }
53 void PushUp(int p)
54 {
55     if (tree[p].cnt >= 1)
56     {
57         tree[p].len1 = pos[tree[p].r] - pos[tree[p].l - 1];
58     }
59     else
60     {
61         if (tree[p].l != tree[p].r)
62         {
63             tree[p].len1 = tree[ls(p)].len1 + tree[rs(p)].len1;
64         }
65         else
66         {
67             tree[p].len1 = 0;
68         }
69     }
70     if (tree[p].cnt >= 2)
71     {
72         tree[p].len2 = pos[tree[p].r] - pos[tree[p].l - 1];
73     }
74     else
75     {

```



```

76         if (tree[p].l != tree[p].r)
77         {
78             if (tree[p].cnt == 1)
79             {
80                 tree[p].len2 = tree[ls(p)].len1 + tree[rs(p)].len1;
81             }
82             else
83             {
84                 tree[p].len2 = tree[ls(p)].len2 + tree[rs(p)].len2;
85             }
86         }
87         else
88         {
89             tree[p].len2 = 0;
90         }
91     }
92 }
93 void Build(int p, int l, int r)
94 {
95     tree[p].cnt = 0;
96     tree[p].l = l, tree[p].r = r;
97     tree[p].len1 = 0; tree[p].len2 = 0;
98     if (tree[p].l != tree[p].r)
99     {
100         int mid = (tree[p].l + tree[p].r) >> 1;
101         Build(ls(p), l, mid + 0);
102         Build(rs(p), mid + 1, r);
103     }
104 }
105 void Change(int p, int l, int r, int d)
106 {
107     if (l <= tree[p].l && tree[p].r <= r)
108     {
109         tree[p].cnt += d; PushUp(p);
110     }
111     else
112     {
113         int mid = (tree[p].l + tree[p].r) >> 1;
114         if (l <= mid) Change(ls(p), l, r, d);
115         if (mid < r) Change(rs(p), l, r, d);
116         PushUp(p);
117     }
118 }
119 /*=====*/
120 double Init(void)
121 {
122     int n; cin >> n;
123     pos.clear(); line.clear();
124     for (int i = 1; i <= n; ++i)
125     {
126         double x1, y1; cin >> x1 >> y1; //左上
127         double x2, y2; cin >> x2 >> y2; //右下
128         pos.push_back(x1); pos.push_back(x2);
129         rectangle[i].x1 = x1; rectangle[i].y1 = y1;
130         rectangle[i].x2 = x2; rectangle[i].y2 = y2;

```

```

131     }
132     sort(pos.begin(), pos.end());
133     pos.erase(unique(pos.begin(), pos.end()), pos.end());
134     for (int i = 1; i <= n; ++i)
135     {
136         int l = lower_bound(pos.begin(), pos.end(), rectangle[i].x1) -
pos.begin();
137         int r = lower_bound(pos.begin(), pos.end(), rectangle[i].x2) -
pos.begin();
138         line.push_back(Line(l, r, rectangle[i].y1, +1));
139         line.push_back(Line(l, r, rectangle[i].y2, -1));
140     }
141     sort(line.begin(), line.end());
142     Build(1, 1, pos.size() - 1);
143     bool flag = true;
144     double ans = 0.0;
145     double last = 0.0;
146     auto it = line.begin();
147     while (it != line.end())
148     {
149         double high = it->h;
150         if (flag) last = high, flag = false;
151         ans += (high - last) * tree[1].len2;
152         while (it != line.end() && it->h == high)
153         {
154             Change(1, it->l + 1, it->r, it->val); it++;
155         }
156         last = high;
157     }
158     return ans;
159 }
160 }

```

## 矩形周长并

```

1 namespace ScanLine
2 {
3     const int N = 1e5 + 10;
4     /*=====*/
5     struct Rectangle
6     {
7         double x1, y1;
8         double x2, y2;
9     };
10    Rectangle rectangle[N];
11    /*=====*/
12    vector<double>pos;
13    /*=====*/
14    struct Line
15    {
16        int val;
17        int l, r;
18        double h;
19        Line(int _l = 0, int _r = 0, double _h = 0, int _val = 0)
20        {

```

```

21         l = _l, r = _r, h = _h, val = _val;
22     }
23     friend bool operator<(const Line& a, const Line& b)
24     {
25         if (a.h != b.h)
26         {
27             return a.h < b.h;
28         }
29         else
30         {
31             return a.val > b.val;
32         }
33     }
34 };
35 vector<Line> line;
36 typedef vector<Line>::iterator iter;
37 /*=====*/
38 struct Tree
39 {
40     int l, r;
41     int cnt; double len;
42 };
43 Tree tree[N << 3];
44 int ls(int p)
45 {
46     return p << 1;
47 }
48 int rs(int p)
49 {
50     return p << 1 | 1;
51 }
52 void PushUp(int p)
53 {
54     if (tree[p].cnt >= 1)
55     {
56         tree[p].len = pos[tree[p].r] - pos[tree[p].l - 1];
57     }
58     else
59     {
60         if (tree[p].l != tree[p].r)
61         {
62             tree[p].len = tree[ls(p)].len + tree[rs(p)].len;
63         }
64         else
65         {
66             tree[p].len = 0;
67         }
68     }
69 }
70 void Build(int p, int l, int r)
71 {
72     tree[p].l = l, tree[p].r = r;
73     tree[p].cnt = 0; tree[p].len = 0;
74     if (tree[p].l != tree[p].r)
75     {

```

```

76         int mid = (tree[p].l + tree[p].r) >> 1;
77         Build(ls(p), l, mid + 0);
78         Build(rs(p), mid + 1, r);
79     }
80 }
81 void Change(int p, int l, int r, int d)
82 {
83     if (l <= tree[p].l && tree[p].r <= r)
84     {
85         tree[p].cnt += d; PushUp(p);
86     }
87     else
88     {
89         int mid = (tree[p].l + tree[p].r) >> 1;
90         if (l <= mid) Change(ls(p), l, r, d);
91         if (mid < r) Change(rs(p), l, r, d);
92         PushUp(p);
93     }
94 }
95 /*=====*/
96 double Init(void)
97 {
98     int n; cin >> n; double ans = 0;
99     for (int i = 1; i <= n; ++i)
100     {
101         double x1, y1; cin >> x1 >> y1; //左上
102         double x2, y2; cin >> x2 >> y2; //右下
103         rectangle[i].x1 = x1; rectangle[i].y1 = y1;
104         rectangle[i].x2 = x2; rectangle[i].y2 = y2;
105     }
106     /*=====*/
107     pos.clear(); line.clear();
108     for (int i = 1; i <= n; ++i)
109     {
110         pos.push_back(rectangle[i].x1);
111         pos.push_back(rectangle[i].x2);
112     }
113     sort(pos.begin(), pos.end());
114     pos.erase(unique(pos.begin(), pos.end()), pos.end());
115     for (int i = 1; i <= n; ++i)
116     {
117         int l = lower_bound(pos.begin(), pos.end(), rectangle[i].x1) -
pos.begin();
118         int r = lower_bound(pos.begin(), pos.end(), rectangle[i].x2) -
pos.begin();
119         line.push_back(Line(l, r, rectangle[i].y1, +1));
120         line.push_back(Line(l, r, rectangle[i].y2, -1));
121     }
122     sort(line.begin(), line.end());
123     Build(1, 1, pos.size() - 1);
124     double last1 = 0;
125     for (iter it = line.begin(); it != line.end(); ++it)
126     {
127         Change(1, it->l + 1, it->r, it->val);
128         ans += abs(tree[1].len - last1); last1 = tree[1].len;

```

```

129     }
130     /*=====*/
131     pos.clear(); line.clear();
132     for (int i = 1; i <= n; ++i)
133     {
134         pos.push_back(rectangle[i].y1);
135         pos.push_back(rectangle[i].y2);
136     }
137     sort(pos.begin(), pos.end());
138     pos.erase(unique(pos.begin(), pos.end()), pos.end());
139     for (int i = 1; i <= n; ++i)
140     {
141         int l = lower_bound(pos.begin(), pos.end(), rectangle[i].y1) -
pos.begin();
142         int r = lower_bound(pos.begin(), pos.end(), rectangle[i].y2) -
pos.begin();
143         line.push_back(Line(l, r, rectangle[i].x1, +1));
144         line.push_back(Line(l, r, rectangle[i].x2, -1));
145     }
146     sort(line.begin(), line.end());
147     Build(1, 1, pos.size() - 1);
148     double last2 = 0;
149     for (iter it = line.begin(); it != line.end(); ++it)
150     {
151         Change(1, it->l + 1, it->r, it->val);
152         ans += abs(tree[1].len - last2); last2 = tree[1].len;
153     }
154     /*=====*/
155     return ans;
156 }
157 }

```

## 可删堆

```

1  template<class Type, class Comp = greater<Type>>
2  class C_Heap
3  {
4  private:
5      priority_queue<Type, vector<Type>, Comp>heap1;
6      priority_queue<Type, vector<Type>, Comp>heap2;
7  public:
8      Type Top(void)
9      {
10         while (!heap2.Empty() && heap1.Top() == heap2.Top())
11         {
12             heap1.Pop(); heap2.Pop();
13         }
14         return heap1.Top();
15     }
16     void Pop(void)
17     {
18         while (!heap2.Empty() && heap1.Top() == heap2.Top())
19         {
20             heap1.Pop(); heap2.Pop();

```

```

21     }
22     heap1.Pop();
23 }
24 int Size(void)
25 {
26     return heap1.Size() - heap2.Size();
27 }
28 void Clear(void)
29 {
30     while (!heap1.Empty())heap1.Pop();
31     while (!heap2.Empty())heap2.Pop();
32 }
33 bool Empty(void)
34 {
35     return heap1.Size() == heap2.Size();
36 }
37 void Erase(Type val)
38 {
39     heap2.push(val);
40 }
41 void Insert(Type val)
42 {
43     heap1.push(val);
44 }
45 };

```

## 并查集

```

1  class C_DSU
2  {
3  private:
4      vector<int>pre, siz;
5      /*=====*/
6      int Find(int cur)
7      {
8          return cur == pre[cur] ? cur : pre[cur] = Find(pre[cur]);
9      }
10 public:
11     void Init(int n)
12     {
13         pre.assign(n + 1, 0);siz.assign(n + 1, 1);
14         for (int i = 0; i <= n; ++i)pre[i] = i;
15     }
16     int operator[](int cur)
17     {
18         return Find(cur);
19     }
20     void operator()(int u, int v)
21     {
22         u = Find(u), v = Find(v);
23         if (siz[u] < siz[v])
24         {
25             pre[u] = v, siz[v] += siz[u];
26         }

```

```

27         else
28         {
29             pre[v] = u, siz[u] += siz[v];
30         }
31     }
32 };

```

## 主席树

```

1  template<class Valu>
2  class C_ChairmanTree
3  {
4  private:
5      struct Tree
6      {
7          int cnt;
8          int ls, rs;
9          Tree(void)
10         {
11             cnt = 0;
12             ls = rs = -1;
13         }
14     };
15     /*=====*/
16     vector<int>root;
17     vector<Tree>tree;
18     /*=====*/
19     int Creat(void)
20     {
21         tree.push_back(Tree());
22         return tree.size() - 1;
23     }
24     void Build(int& cur, int tree1, int treer)
25     {
26         cur = Creat();
27         if (tree1 == treer)return;
28         int mid = (tree1 + treer) >> 1;
29         Build(tree[cur].ls, tree1, mid + 0);
30         Build(tree[cur].rs, mid + 1, treer);
31     }
32     void Insert(int pre, int& cur, int x, int tree1, int treer)
33     {
34         cur = Creat();
35         tree[cur] = tree[pre]; tree[cur].cnt++;
36         if (tree1 == treer)return;
37         int mid = (tree1 + treer) >> 1;
38         if (x <= mid)Insert(tree[pre].ls, tree[cur].ls, x, tree1, mid + 0);
39         if (mid < x) Insert(tree[pre].rs, tree[cur].rs, x, mid + 1, treer);
40     }
41     int Ask(int pre, int cur, int k, int tree1, int treer)
42     {
43         if (tree1 == treer)return tree1;
44         int mid = (tree1 + treer) >> 1;
45         int cnt = tree[tree[cur].ls].cnt - tree[tree[pre].ls].cnt;

```

```

46         if (cnt < k) return Ask(tree[pre].rs, tree[cur].rs, k - cnt, mid + 1,
treeer);
47         else return Ask(tree[pre].ls, tree[cur].ls, k, tree1, mid + 0);
48     }
49     /*=====*/
50     vector<valu> lib;
51 public:
52     void Init(int n, valu arr[])
53     {
54         for (int i = 1; i <= n; ++i)
55         {
56             lib.push_back(arr[i]);
57         }
58         sort(lib.begin(), lib.end());
59         lib.erase(unique(lib.begin(), lib.end()), lib.end());
60         /*=====*/
61         tree.reserve(2 * lib.size() + n * (ceil(log2(lib.size())) + 1));
62         /*=====*/
63         root.assign(n + 1, -1);
64         Build(root[0], 1, lib.size());
65         for (int i = 1; i <= n; ++i)
66         {
67             int x = lower_bound(lib.begin(), lib.end(), arr[i]) -
lib.begin() + 1;
68             Insert(root[i - 1], root[i], x, 1, lib.size());
69         }
70     }
71     valu operator()(int l, int r, int k)
72     {
73         return lib[Ask(root[l - 1], root[r], k, 1, lib.size()) - 1];
74     }
75 };

```

## 平衡树

### 带旋·Treap·权值树

```

1  template<class Valu>
2  class C_Treap
3  {
4  private:
5      struct Tree
6      {
7          int siz;
8          valu valu;
9          int priority;
10         Tree* lch, * rch;
11         Tree(void)
12         {
13             siz = 0;
14             valu = valu();
15             lch = rch = NULL;
16             priority = rand();
17         }

```



```

18     };
19     /*=====*/
20     Tree* null, * root;
21     /*=====*/
22     Tree* Creat(Valu valu)
23     {
24         Tree* node = new Tree;
25         node->valu = valu;
26         node->lch = null;
27         node->rch = null;
28         node->siz = 1;
29         return node;
30     }
31     /*=====*/
32     void PushUp(Tree* cur)
33     {
34         cur->siz = cur->lch->siz + cur->rch->siz + 1;
35     }
36     /*=====*/
37     void LRotate(Tree*& cur)
38     {
39         Tree* son = cur->rch;
40         cur->rch = son->lch; son->lch = cur; cur = son;
41         PushUp(cur->lch); PushUp(cur);
42     }
43     void RRotate(Tree*& cur)
44     {
45         Tree* son = cur->lch;
46         cur->lch = son->rch; son->rch = cur; cur = son;
47         PushUp(cur->rch); PushUp(cur);
48     }
49     /*=====*/
50     void Insert(Tree*& cur, Valu valu)
51     {
52         if (cur == null)
53         {
54             cur = Creat(valu);
55         }
56         else
57         {
58             if (valu < cur->valu)
59             {
60                 Insert(cur->lch, valu);
61                 if (cur->priority < cur->lch->priority)
62                 {
63                     RRotate(cur);
64                 }
65             }
66             else
67             {
68                 Insert(cur->rch, valu);
69                 if (cur->priority < cur->rch->priority)
70                 {
71                     LRotate(cur);
72                 }

```

```

73     }
74     PushUp(cur);
75 }
76 }
77 void Delete(Tree*& cur, valu valu)
78 {
79     if (cur == null) return;
80     if (valu == cur->valu)
81     {
82         if (cur->lch != null && cur->rch != null)
83         {
84             if (cur->lch->priority < cur->rch->priority)
85             {
86                 LRotate(cur); Delete(cur->lch, valu); PushUp(cur);
87             }
88             else
89             {
90                 RRotate(cur); Delete(cur->rch, valu); PushUp(cur);
91             }
92         }
93         else if (cur->lch != null)
94         {
95             RRotate(cur); Delete(cur->rch, valu); PushUp(cur);
96         }
97         else if (cur->rch != null)
98         {
99             LRotate(cur); Delete(cur->lch, valu); PushUp(cur);
100        }
101        else
102        {
103            delete cur; cur = null;
104        }
105    }
106    else
107    {
108        if (valu < cur->valu)
109        {
110            Delete(cur->lch, valu);
111        }
112        else
113        {
114            Delete(cur->rch, valu);
115        }
116        PushUp(cur);
117    }
118 }
119 /*=====*/
120 valu GetValueByRank(int rank)
121 {
122     Tree* cur = root;
123     while (cur != null)
124     {
125         if (cur->lch->siz + 1 == rank)
126         {
127             return cur->valu;

```

```

128         }
129         else
130         {
131             if (cur->lch->siz < rank)
132             {
133                 rank -= cur->lch->siz + 1;
134                 cur = cur->rch;
135             }
136             else
137             {
138                 cur = cur->lch;
139             }
140         }
141     }
142     return valu();
143 }
144 int GetRankByValu(valu valu)
145 {
146     int res = 1;
147     Tree* cur = root;
148     while (cur != null)
149     {
150         if (cur->valu < valu)
151         {
152             res += cur->lch->siz + 1;
153             cur = cur->rch;
154         }
155         else
156         {
157             cur = cur->lch;
158         }
159     }
160     return res;
161 }
162 /*=====*/
163 void clear(Tree* cur)
164 {
165     if (cur != null)
166     {
167         clear(cur->lch);
168         clear(cur->rch);
169         delete cur;
170     }
171 }
172 public:
173 C_Treap(void)
174 {
175     root = null = new Tree;
176 }
177 ~C_Treap(void)
178 {
179     clear(root); delete null;
180 }
181 /*=====*/
182 int Size(void)

```

```

183     {
184         return root->siz;
185     }
186     void Clear(void)
187     {
188         Clear(root); root = null;
189     }
190     bool Empty(void)
191     {
192         return root->siz == 0;
193     }
194     void Erase(Valu valu)
195     {
196         Delete(root, valu);
197     }
198     void Insert(Valu valu)
199     {
200         Insert(root, valu);
201     }
202     int operator()(Valu valu)
203     {
204         return GetRankByValu(valu);
205     }
206     Valu operator[](int rank)
207     {
208         return GetValuByRank(rank);
209     }
210 };

```

## 无旋·Treap·序列树

```

1  template<class Valu>
2  class C_Treap
3  {
4  public:
5      struct Tree
6      {
7          int siz = 0;
8          Valu valu = Valu();
9          int priority = rand();
10         Tree* lch = NULL, * rch = NULL;
11     };
12     /*=====*/
13     Tree* null, * root;
14     /*=====*/
15     Tree* Creat(Valu valu)
16     {
17         Tree* node = new Tree;
18         node->valu = valu;
19         node->lch = null;
20         node->rch = null;
21         node->siz = 1;
22         return node;
23     }
24     /*=====*/

```

```

25 void PushUp(Tree* cur)
26 {
27     cur->siz = cur->lch->siz + cur->rch->siz + 1;
28 }
29 void PushDown(Tree* cur)
30 {
31     /*
32     * 预留
33     */
34 }
35 /*=====*/
36 Tree* Build(int l, int r, Valu arr[])
37 {
38     if (l > r) return null;
39     int mid = (l + r) >> 1;
40     Tree* cur = Creat(arr[mid]);
41     cur->lch = Build(l, mid - 1, arr);
42     cur->rch = Build(mid + 1, r, arr);
43     PushUp(cur); return cur;
44 }
45 Tree* Build(int l, int r, vector<Valu>& arr)
46 {
47     if (l > r) return null;
48     int mid = (l + r) >> 1;
49     Tree* cur = Creat(arr[mid]);
50     cur->lch = Build(l, mid - 1, arr);
51     cur->rch = Build(mid + 1, r, arr);
52     PushUp(cur); return cur;
53 }
54 /*=====*/
55 void Lower_Split(Tree* cur, int index, Tree*& ltree, Tree*&
rtree)//index留在rtree
56 {
57     if (cur == null)
58     {
59         ltree = rtree = null; return;
60     }
61     PushDown(cur);
62     if (cur->lch->siz + 1 < index)
63     {
64         Lower_Split(cur->rch, index - cur->lch->siz - 1, ltree, rtree);
65         cur->rch = ltree; PushUp(cur); ltree = cur;
66     }
67     else
68     {
69         Lower_Split(cur->lch, index, ltree, rtree);
70         cur->lch = rtree; PushUp(cur); rtree = cur;
71     }
72 }
73 void Upper_Split(Tree* cur, int index, Tree*& ltree, Tree*&
rtree)//index留在ltree
74 {
75     if (cur == null)
76     {
77         ltree = rtree = null; return;

```

```

78     }
79     PushDown(cur);
80     if (cur->lch->siz < index)
81     {
82         Upper_Split(cur->rch, index - cur->lch->siz - 1, ltree, rtree);
83         cur->rch = ltree; PushUp(cur); ltree = cur;
84     }
85     else
86     {
87         Upper_Split(cur->lch, index, ltree, rtree);
88         cur->lch = rtree; PushUp(cur); rtree = cur;
89     }
90 }
91 /*=====*/
92 Tree* Merge(Tree* ltree, Tree* rtree)
93 {
94     if (ltree == null) return rtree;
95     if (rtree == null) return ltree;
96     PushDown(ltree); PushDown(rtree);
97     if (ltree->priority < rtree->priority)
98     {
99         rtree->lch = Merge(ltree, rtree->lch);
100        PushUp(rtree); return rtree;
101    }
102    else
103    {
104        ltree->rch = Merge(ltree->rch, rtree);
105        PushUp(ltree); return ltree;
106    }
107 }
108 /*=====*/
109 void Split(int l, int r, Tree*& ltree, Tree*& mtree, Tree*& rtree)
110 {
111     Tree* o1, * o2, * o3, * o4;
112     Upper_Split(root, r, o1, o2);
113     Lower_Split(o1, l, o3, o4);
114     ltree = o3, mtree = o4, rtree = o2;
115 }
116 void Merge(Tree* ltree, Tree* mtree, Tree* rtree)
117 {
118     root = Merge(Merge(ltree, mtree), rtree);
119 }
120 /*=====*/
121 valu operator[](int rank)
122 {
123     Tree* cur = root;
124     while (cur != null)
125     {
126         PushDown(cur);
127         if (cur->lch->siz + 1 == rank)
128         {
129             return cur->valu;
130         }
131         else
132         {

```

```

133         if (cur->lch->siz < rank)
134         {
135             rank -= cur->lch->siz + 1;
136             cur = cur->rch;
137         }
138         else
139         {
140             cur = cur->lch;
141         }
142     }
143 }
144 return val;
145 }
146 /*=====*/
147 C_Treap(void)
148 {
149     root = null = new Tree;
150 }
151 void Clear(Tree* cur)
152 {
153     if (cur != null)
154     {
155         Clear(cur->lch);
156         Clear(cur->rch);
157         delete cur;
158     }
159 }
160 ~C_Treap(void)
161 {
162     Clear(root); delete null;
163 }
164 };

```

## 双旋·Splay·权值树

```

1  template<class Type>
2  class Splay
3  {
4  public:
5      ~Splay(void)
6      {
7          Clear(root);
8          delete null;
9      }
10     int size(void)
11     {
12         return count;
13     }
14     void clear(void)
15     {
16         Clear(root);
17         root = null;
18     }
19     bool empty(void)
20     {

```

```

21         return count == 0;
22     }
23     Type pre(Type val)
24     {
25         root = splay(FindPre(root, val));
26         return root->val;
27     }
28     Type nxt(Type val)
29     {
30         root = splay(FindNxt(root, val));
31         return root->val;
32     }
33     void erase(Type val)
34     {
35         count--; root = Delete(FindByValu(root, val));
36     }
37     void insert(Type val)
38     {
39         count++; root = splay(Insert(root, val));
40     }
41     int operator()(Type val)
42     {
43         root = splay(FindByValu(root, val));
44         return root->lch->siz + 1;
45     }
46     Type operator[](int rank)
47     {
48         root = splay(FindByRank(root, rank));
49         return root->val;
50     }
51     Type lower_bound(Type val)
52     {
53         root = splay(FindLower(root, val));
54         return root->val;
55     }
56     Type upper_bound(Type val)
57     {
58         root = splay(FindUpper(root, val));
59         return root->val;
60     }
61 private:
62     struct Node
63     {
64         int siz = 0;
65         Type val = Type();
66         Node* fa = NULL;
67         Node* lch = NULL;
68         Node* rch = NULL;
69     };
70     /*=====*/
71     typedef bool CHILD;
72     const CHILD LCH = true;
73     const CHILD RCH = false;
74     /*=====*/
75     int count = 0;

```



```

76 Node* null = new Node;
77 Node* root = null;
78 /*=====*/
79 CHILD Child(Node* cur)
80 {
81     Node* pre = cur->fa;
82     if (pre->lch == cur)
83     {
84         return LCH;
85     }
86     else
87     {
88         return RCH;
89     }
90 }
91
92 void PushUp(Node* cur)
93 {
94     cur->siz = cur->lch->siz + cur->rch->siz + 1;
95 }
96
97 void Del(Node* cur, Node* pre, CHILD WCH)
98 {
99     cur->fa = null;
100     if (WCH == LCH)pre->lch = null;
101     if (WCH == RCH)pre->rch = null;
102 }
103 void Add(Node* cur, Node* pre, CHILD WCH)
104 {
105     cur->fa = pre;
106     if (WCH == LCH)pre->lch = cur;
107     if (WCH == RCH)pre->rch = cur;
108 }
109
110 void LRotate(Node* cur)
111 {
112     Node* pre = cur->fa, * nxt = cur->lch, * anc = pre->fa;
113     CHILD WCH = Child(pre);
114     Del(nxt, cur, LCH); Del(cur, pre, RCH); Del(pre, anc, WCH);
115     Add(nxt, pre, RCH); Add(pre, cur, LCH); Add(cur, anc, WCH);
116     PushUp(pre); PushUp(cur);
117 }
118 void RRotate(Node* cur)
119 {
120     Node* pre = cur->fa, * nxt = cur->rch, * anc = pre->fa;
121     CHILD WCH = Child(pre);
122     Del(nxt, cur, RCH); Del(cur, pre, LCH); Del(pre, anc, WCH);
123     Add(nxt, pre, LCH); Add(pre, cur, RCH); Add(cur, anc, WCH);
124     PushUp(pre); PushUp(cur);
125 }
126
127 void Rotate(Node* cur)
128 {
129     if (Child(cur) == LCH)
130     {

```

```

131         RRotate(cur);
132     }
133     else
134     {
135         LRotate(cur);
136     }
137 }
138
139 void Clear(Node* cur)
140 {
141     if (cur != null)
142     {
143         Clear(cur->lch);
144         Clear(cur->rch);
145         delete cur;
146     }
147 }
148
149 Node* Creat(Type val)
150 {
151     Node* cur = new Node;
152     cur->lch = null;
153     cur->rch = null;
154     cur->fa = null;
155     cur->val = val;
156     cur->siz = 1;
157     return cur;
158 }
159
160 Node* splay(Node* cur)
161 {
162     while (true)
163     {
164         Node* pre = cur->fa;
165         if (cur->fa == null) break;
166         if (pre->fa == null) break;
167         CHILD CHpre = Child(pre);
168         CHILD CHcur = Child(cur);
169         if (CHpre == CHcur)
170         {
171             Rotate(pre); Rotate(cur); continue;
172         }
173         if (CHpre != CHcur)
174         {
175             Rotate(cur); Rotate(cur); continue;
176         }
177     }
178     if (cur->fa != null) Rotate(cur); return cur;
179 }
180
181 Node* Insert(Node* cur, Type val)
182 {
183     CHILD WCH = LCH; Node* pre = null;
184     while (cur != null)
185     {

```

```

186         if (val < cur->val)
187         {
188             pre = cur; cur = cur->lch; WCH = LCH;
189         }
190         else
191         {
192             pre = cur; cur = cur->rch; WCH = RCH;
193         }
194     }
195     cur = Creat(val); Add(cur, pre, WCH); return cur;
196 }
197
198 Node* Delete(Node* cur)
199 {
200     splay(cur);
201     Node* lch = cur->lch;
202     Node* rch = cur->rch;
203     delete cur; return Merge(lch, rch);
204 }
205
206 Node* Merge(Node* ltree, Node* rtree)
207 {
208     if (ltree == null)
209     {
210         rtree->fa = null; return rtree;
211     }
212     if (rtree == null)
213     {
214         ltree->fa = null; return ltree;
215     }
216     ltree->fa = null; rtree->fa = null;
217     if (ltree->siz < rtree->siz)
218     {
219         Node* cur = FindMax(ltree); splay(cur);
220         Add(rtree, cur, RCH); PushUp(cur); return cur;
221     }
222     else
223     {
224         Node* cur = FindMin(rtree); splay(cur);
225         Add(ltree, cur, LCH); PushUp(cur); return cur;
226     }
227 }
228
229 Node* FindByValu(Node* cur, Type val)
230 {
231     Node* res = null;
232     while (cur != null)
233     {
234         if (val == cur->val)
235         {
236             res = cur; cur = cur->lch;
237         }
238         else
239         {
240             if (val < cur->val)

```

```

241         {
242             cur = cur->lch;
243         }
244         else
245         {
246             cur = cur->rch;
247         }
248     }
249 }
250 return res;
251 }
252 Node* FindByRank(Node* cur, int rank)
253 {
254     while (cur != null)
255     {
256         if (cur->lch->siz + 1 == rank)
257         {
258             return cur;
259         }
260         else
261         {
262             if (cur->lch->siz < rank)
263             {
264                 rank -= cur->lch->siz + 1;
265                 cur = cur->rch;
266             }
267             else
268             {
269                 cur = cur->lch;
270             }
271         }
272     }
273     return null;
274 }
275
276 Node* FindLower(Node* cur, Type val)
277 {
278     Node* res = null;
279     while (cur != null)
280     {
281         if (cur->val < val)
282         {
283             cur = cur->rch;
284         }
285         else
286         {
287             res = cur;
288             cur = cur->lch;
289         }
290     }
291     return res;
292 }
293 Node* FindUpper(Node* cur, Type val)
294 {
295     Node* res = null;

```

```

296     while (cur != null)
297     {
298         if (val < cur->val)
299         {
300             res = cur;
301             cur = cur->lch;
302         }
303         else
304         {
305             cur = cur->rch;
306         }
307     }
308     return res;
309 }
310
311 Node* FindMin(Node* cur)
312 {
313     while (cur->lch != null)
314     {
315         cur = cur->lch;
316     }
317     return cur;
318 }
319 Node* FindMax(Node* cur)
320 {
321     while (cur->rch != null)
322     {
323         cur = cur->rch;
324     }
325     return cur;
326 }
327
328 Node* FindPre(Node* cur, Type val)
329 {
330     Node* res = null;
331     while (cur != null)
332     {
333         if (cur->val < val)
334         {
335             res = cur;
336             cur = cur->rch;
337         }
338         else
339         {
340             cur = cur->lch;
341         }
342     }
343     return res;
344 }
345 Node* FindNxt(Node* cur, Type val)
346 {
347     Node* res = null;
348     while (cur != null)
349     {
350         if (val < cur->val)

```

```

351         {
352             res = cur;
353             cur = cur->lch;
354         }
355         else
356         {
357             cur = cur->rch;
358         }
359     }
360     return res;
361 }
362 };

```

## 珂朵莉树

```

1  namespace Chtholly
2  {
3      struct Tree
4      {
5          int l, r;
6          mutable int val;
7          Tree(int _l = 0, int _r = 0, int _val = int())
8          {
9              l = _l, r = _r, val = _val;
10         }
11         friend bool operator<(const Tree& a, const Tree& b)
12         {
13             return a.l < b.l;
14         }
15         int len(void) const { return r - l + 1; }
16     };
17     /*=====*/
18     int n; set<Tree> tree;
19     /*=====*/
20     set<Tree>::iterator Find(int pos)
21     {
22         return --tree.upper_bound(Tree(pos));
23     }
24     set<Tree>::iterator Split(int pos) //lower
25     {
26         if (pos > n) return tree.end();
27         auto it = Find(pos); if (it->l == pos) return it;
28         int l = it->l, r = it->r; auto val = it->val; tree.erase(it);
29         tree.insert(Tree(l, pos - 1, val)); return tree.insert(Tree(pos, r,
30         val)).first;
31     }
32     /*=====*/
33     void GetRange(int l, int r, auto& begin, auto& end)
34     {
35         end = Split(r + 1), begin = Split(l);
36     }
37     void EraseRange(int l, int r)
38     {

```

```

38     set<Tree>::iterator begin, end; GetRange(l, r, begin, end);
tree.erase(begin, end);
39 }
40 void InsertRange(int l, int r, int val)
41 {
42     EraseRange(l, r); auto it = tree.insert(Tree(l, r, val)).first;
43     {auto __it = it; __it--; if (it != tree.begin() && __it->val ==
val)l = __it->l; }
44     {auto __it = it; __it++; if (__it != tree.end() && __it->val ==
val)r = __it->r; }
45     EraseRange(l, r); tree.insert(Tree(l, r, val));
46 }
47 /*=====*/
48 void Build(int _n)
49 {
50     n = _n; tree.insert(Tree(1, n, 1));
51 }
52 }

```

## 树状数组

### 权值树状数组

```

1  class C_FenwickTree
2  {
3  private:
4      int n, m; vector<int>tree;
5      /*=====*/
6      int lowbit(int x) { return x & (-x); }
7  public:
8      int Size(void)
9      {
10         return tree[0];
11     }
12     void Init(int n)
13     {
14         this->n = n; m = 0;
15         tree.assign(n + 1, 0);
16         while ((1 << (m + 1)) <= n)m++;
17     }
18     bool Empty(void)
19     {
20         return tree[0] == 0;
21     }
22     void Erase(int x)
23     {
24         tree[0]--;
25         while (x <= n)
26         {
27             tree[x] -= 1; x += lowbit(x);
28         }
29     }
30     void Insert(int x)
31     {

```

```

32     tree[0]++;
33     while (x <= n)
34     {
35         tree[x] += 1; x += lowbit(x);
36     }
37 }
38 int operator()(int valu)
39 {
40     valu--;
41     int rank = 0;
42     while (valu)
43     {
44         rank += tree[valu];
45         valu -= lowbit(valu);
46     }
47     return rank + 1;
48 }
49 int operator[](int rank)
50 {
51     int sum = 0, valu = 0;
52     for (int i = m; i >= 0; --i)
53     {
54         int temp = valu + (1 << i);
55         if (temp <= n && sum + tree[temp] < rank)
56         {
57             sum += tree[temp]; valu = temp;
58         }
59     }
60     return valu + 1;
61 }
62 };

```

## 一维树状数组

```

1  template<class Type>
2  class FenwickTree
3  {
4  public:
5      Type ask(int pos)
6      {
7          Type res = Type();
8          while (pos)
9          {
10             res += tree[pos];
11             pos -= lowbit(pos);
12         }
13         return res;
14     }
15     void init(int n)
16     {
17         this->n = n;
18         tree = new Type[n + 1];
19         for (int i = 0; i <= n; ++i)
20         {
21             tree[i] = Type();

```



```

22     }
23 }
24 ~FenwickTree(void)
25 {
26     delete[] tree;
27 }
28 void add(int pos, Type d)
29 {
30     while (pos <= n)
31     {
32         tree[pos] += d;
33         pos += lowbit(pos);
34     }
35 }
36 Type ask(int l, int r)
37 {
38     Type res = Type(); l--;
39     while (r > l) res += tree[r], r -= lowbit(r);
40     while (l > r) res -= tree[l], l -= lowbit(l);
41     return res;
42 }
43 private:
44     int n = 0;
45     Type* tree = NULL;
46     /*=====*/
47     int lowbit(int x) { return x & (-x); }
48 };

```

## 二维树状数组

```

1  template<class Type>
2  class FenwickTree
3  {
4  public:
5      ~FenwickTree(void)
6      {
7          for (int i = 0; i <= n; ++i)
8          {
9              delete[] tree[i];
10             }
11             delete[] tree;
12         }
13         Type ask(int x, int y)
14         {
15             Type res = Type();
16             while (x)
17             {
18                 int tempy = y;
19                 while (tempy)
20                 {
21                     res += tree[x][tempy];
22                     tempy -= lowbit(tempy);
23                 }
24                 x -= lowbit(x);
25             }

```

```

26         return res;
27     }
28     void init(int n, int m)
29     {
30         this->n = n;
31         this->m = m;
32         tree = new Type * [n + 1];
33         for (int i = 0; i <= n; ++i)
34         {
35             tree[i] = new Type[m + 1];
36             for (int j = 0; j <= m; ++j)
37             {
38                 tree[i][j] = Type();
39             }
40         }
41     }
42     void add(int x, int y, Type d)
43     {
44         while (x <= n)
45         {
46             int tempy = y;
47             while (tempy <= m)
48             {
49                 tree[x][tempy] += d;
50                 tempy += lowbit(tempy);
51             }
52             x += lowbit(x);
53         }
54     }
55     Type ask(int x1, int y1, int x2, int y2)
56     {
57         return ask(x2, y2) - ask(x1 - 1, y2) - ask(x2, y1 - 1) + ask(x1 - 1,
58         y1 - 1);
59     }
60 private:
61     int n = 0, m = 0;
62     Type** tree = NULL;
63     /*=====*/
64     int lowbit(int x) { return x & (-x); }
65 };

```

## 区间mex

```

1  class Range_MEX
2  {
3  public:
4      Range_MEX(int n, int arr[], int l, int r)
5      {
6          root = new int[n + 1];
7          tree = new Tree[4200000 + 10];
8
9          for (int i = 0; i <= n; ++i) root[i] = -1;
10
11         BuildZero(root[0], l, r);

```

```

12         for (int i = 1; i <= n; ++i)
13         {
14             BuildChain(root[i - 1], root[i], i, arr[i]);
15         }
16     }
17     int operator()(int l, int r)
18     {
19         return Ask(root[r], l);
20     }
21     ~Range_MEX(void)
22     {
23         delete[] tree;
24         delete[] root;
25     }
26 private:
27     struct Tree
28     {
29         int idx;
30         int l, r;
31         int ls, rs;
32         Tree(void)
33         {
34             idx = 0;
35             l = 0, r = 0;
36             ls = -1, rs = -1;
37         }
38     };
39     /*=====*/
40     int * root;
41     Tree* tree; int cnt = -1;
42     /*=====*/
43     void PushUp(int cur)
44     {
45         tree[cur].idx = min(tree[tree[cur].ls].idx, tree[tree[cur].rs].idx);
46     }
47     /*=====*/
48     void BuildZero(int& cur, int l, int r)
49     {
50         if (cur == -1) cur = ++cnt;
51         /*=====*/
52         tree[cur].l = l, tree[cur].r = r;
53         if (l != r)
54         {
55             int mid = (l + r) >> 1;
56             BuildZero(tree[cur].ls, l, mid + 0);
57             BuildZero(tree[cur].rs, mid + 1, r);
58         }
59     }
60     void BuildChain(int& pre, int& cur, int idx, int val)
61     {
62         if (cur == -1) cur = ++cnt;
63         /*=====*/
64         tree[cur].l = tree[pre].l, tree[cur].r = tree[pre].r;
65         tree[cur].ls = tree[pre].ls, tree[cur].rs = tree[pre].rs;
66         if (tree[cur].l == tree[cur].r)

```

```

67     {
68         tree[cur].idx = idx;
69     }
70     else
71     {
72         int mid = (tree[cur].l + tree[cur].r) >> 1;
73         if (val <= mid) BuildChain(tree[pre].ls, tree[cur].ls = -1, idx,
val);
74         if (mid < val) BuildChain(tree[pre].rs, tree[cur].rs = -1, idx,
val);
75         PushUp(cur);
76     }
77 }
78 /*=====*/
79 int Ask(int& cur, int l)
80 {
81     if (tree[cur].l == tree[cur].r) return tree[cur].l;
82     if (tree[tree[cur].ls].idx < l) return Ask(tree[cur].ls, l);
83     if (tree[tree[cur].rs].idx < l) return Ask(tree[cur].rs, l);
84     return tree[cur].r + 1;
85 }
86 };

```

## Hash\_MAP

```

1  const int Base = 19260817;
2  class Hash_Map
3  {
4  public:
5      Hash_Map()
6      {
7          memset(head, -1, sizeof(head));
8          nxt.reserve(1e7);
9          key.reserve(1e7);
10         val.reserve(1e7);
11     }
12     int& operator[](int k)
13     {
14         int h = hash(k);
15         for (int i = head[h]; ~i; i = nxt[i])
16         {
17             if (key[i] == k)
18             {
19                 return val[i];
20             }
21         }
22         nxt.push_back(head[h]);
23         key.push_back(k);
24         val.push_back(0);
25         head[h] = nxt.size() - 1;
26         return val.back();
27     }
28     int has_key(int k)
29     {

```

```

30         int h = hash(k);
31         for (int i = head[h]; ~i; i = nxt[i])
32         {
33             if (key[i] == k)
34             {
35                 return true;
36             }
37         }
38         return false;
39     }
40 private:
41     int head[Base];
42     vector<int>nxt;
43     vector<int>key;
44     vector<int>val;
45     int hash(int k)
46     {
47         return k % Base;
48     }
49 };

```

## 线段树合并

```

1  /*
2  * 与动态开点权值线段树搭配使用
3  */
4  Tree* Merge(Tree*& a, Tree*& b, int tree1, int treer)
5  {
6      Tree* cur = null;
7      if (a == null)
8      {
9          cur = b; b = null; return cur;
10     }
11     if (b == null)
12     {
13         cur = a; a = null; return cur;
14     }
15     cur = Creat();
16     if (tree1 == treer)
17     {
18         cur->siz = a->siz + b->siz;
19     }
20     else
21     {
22         int mid = (tree1 + treer) >> 1;
23         cur->ls = Merge(a->ls, b->ls, tree1, mid + 0);
24         cur->rs = Merge(a->rs, b->rs, mid + 1, treer);
25         cur->siz = cur->ls->siz + cur->rs->siz;
26     }
27     delete a; a = null; delete b; b = null; return cur;
28 }

```

## 李超线段树

```

1  class C_LiChaoTree
2  {
3  private:
4      struct Function
5      {
6          int k, b;
7          int operator()(int x)
8          {
9              return k * x + b;
10         }
11         Function(int _k = 0, int _b = +INF)
12         {
13             k = _k, b = _b;
14         }
15     };
16     /*=====*/
17     struct Tree
18     {
19         Function f;
20         Tree* ls, * rs;
21         Tree(void)
22         {
23             ls = rs = NULL;
24         }
25     };
26     /*=====*/
27     Tree* root; int tree1, treer;
28     /*=====*/
29     void PushDown(Tree*& cur, Function f, int tree1, int treer)
30     {
31         if (cur == NULL) cur = new Tree;
32         int mid = (tree1 + treer) >> 1;
33         if (tree1 != treer)
34         {
35             if (cur->f.k < f.k)
36             {
37                 if (f(mid) < cur->f(mid))
38                 {
39                     PushDown(cur->rs, cur->f, tree1, mid + 0);
40                 }
41                 else
42                 {
43                     PushDown(cur->ls, f, mid + 1, treer);
44                 }
45             }
46             else
47             {
48                 if (f(mid) < cur->f(mid))
49                 {
50                     PushDown(cur->ls, cur->f, tree1, mid + 0);
51                 }
52                 else
53                 {
54                     PushDown(cur->rs, f, mid + 1, treer);
55                 }

```

```

56         }
57     }
58     if (f(mid) < cur->f(mid)) cur->f = f;
59 }
60 void Add(Tree*& cur, int l, int r, Function f, int tree1, int treer)
61 {
62     if (cur == NULL) cur = new Tree;
63     if (l <= tree1 && treer <= r)
64     {
65         PushDown(cur, f, tree1, treer);
66     }
67     else
68     {
69         int mid = (tree1 + treer) >> 1;
70         if (l <= mid) Add(cur->ls, l, r, f, tree1, mid + 0);
71         if (mid < r) Add(cur->rs, l, r, f, mid + 1, treer);
72     }
73 }
74 int Ask(Tree* cur, int x, int tree1, int treer)
75 {
76     int res = +INF;
77     if (cur != NULL)
78     {
79         res = cur->f(x);
80         if (tree1 != treer)
81         {
82             int mid = (tree1 + treer) >> 1;
83             if (x <= mid) res = min(res, Ask(cur->ls, x, tree1, mid +
0));
84             if (mid < x) res = min(res, Ask(cur->rs, x, mid + 1,
treer));
85         }
86     }
87     return res;
88 }
89 public:
90     C_LichaoTree(void)
91     {
92         root = NULL, tree1 = treer = 0;
93     }
94     ~C_LichaoTree(void)
95     {
96         if (root != NULL)
97         {
98             queue<Tree*> q; q.push(root);
99             while (!q.empty())
100             {
101                 if (q.front()->ls != NULL) q.push(q.front()->ls);
102                 if (q.front()->rs != NULL) q.push(q.front()->rs);
103                 delete q.front(); q.pop();
104             }
105         }
106     }
107     /*=====*/
108     void Init(int tree1, int treer)

```

```

109     {
110         this->tree1 = tree1;
111         this->treer = treer;
112     }
113     void operator()(int l, int r, Function f)
114     {
115         Add(root, l, r, f, tree1, treer);
116     }
117     int operator()(int x)
118     {
119         return Ask(root, x, tree1, treer);
120     }
121 };

```

## 可持久化01Trie

```

1  int root[N], tot;
2  int siz[35 * N];
3  int trie[35 * N][2];
4  /*=====*/
5  void Insert(int pre, int cur, int num)
6  {
7      for (int i = 32; i >= 0; --i)
8      {
9          siz[cur] = siz[pre] + 1;
10         int bit = (num & (1ll << i)) ? 1 : 0;
11         trie[cur][bit ^ 1] = trie[pre][bit ^ 1], trie[cur][bit] = ++tot;
12         pre = trie[pre][bit]; cur = trie[cur][bit];
13     }
14     siz[cur] = siz[pre] + 1;
15 }
16 int Query(int cur, int num, int k)
17 {
18     int ans = 0;
19     for (int i = 32; i >= 0; --i)
20     {
21         int bit = (num & (1ll << i)) ? 1 : 0;
22         if (siz[trie[cur][bit ^ 1]] >= k)
23         {
24             ans += 1ll << i;
25             cur = trie[cur][bit ^ 1];
26         }
27         else
28         {
29             k -= siz[trie[cur][bit ^ 1]];
30             cur = trie[cur][bit];
31         }
32     }
33     return ans;
34 }

```

## Treap维护珂朵莉树



```

1 namespace Treap
2 {
3     struct Node
4     {
5         Range range;
6         int priority;
7         int lch, rch;
8     }node[N * 4];
9     /*=====*/
10    int null = -1;
11    int root = -1;
12    /*=====*/
13    int Creat(Range range)
14    {
15        static int cnt = 0; ++cnt;
16        node[cnt].lch = null;
17        node[cnt].rch = null;
18        node[cnt].priority = rand();
19        node[cnt].range = range;
20        return cnt;
21    }
22    /*=====*/
23    void PushUp(int cur)
24    {
25        node[cur].range.L = node[cur].range.l;
26        node[cur].range.R = node[cur].range.r;
27        node[cur].range.Sum = node[cur].range.sum();
28        if (node[cur].lch != null)
29        {
30            node[cur].range.L = node[node[cur].lch].range.L;
31            node[cur].range.Sum += node[node[cur].lch].range.Sum;
32        }
33        if (node[cur].rch != null)
34        {
35            node[cur].range.R = node[node[cur].rch].range.R;
36            node[cur].range.Sum += node[node[cur].rch].range.Sum;
37        }
38    }
39    void PushDown(int cur)
40    {
41        if (node[cur].range.lazy != 0)
42        {
43            if (node[cur].lch != null)
44            {
45                node[node[cur].lch].range.Maintain(node[cur].range.lazy);
46            }
47            if (node[cur].rch != null)
48            {
49                node[node[cur].rch].range.Maintain(node[cur].range.lazy);
50            }
51            node[cur].range.lazy = 0;
52        }
53    }
54    /*=====*/
55    int Merge(int ltree, int rtree)

```

```

56     {
57         if (ltree == null) return rtree;
58         if (rtree == null) return ltree;
59         PushDown(ltree); PushDown(rtree);
60         if (node[ltree].priority < node[rtree].priority)
61         {
62             node[rtree].lch = Merge(ltree, node[rtree].lch);
63             /*=====*/PushUp(rtree); return rtree;
64         }
65         else
66         {
67             node[ltree].rch = Merge(node[ltree].rch, rtree);
68             /*=====*/PushUp(ltree); return ltree;
69         }
70     }
71     /*=====*/
72     void Lower_Split(int cur, int index, int& ltree, int& rtree)//index留在
rtree
73     {
74         if (cur == null)
75         {
76             ltree = rtree = null; return;
77         }
78         PushDown(cur);
79         if (node[cur].range.r < index)
80         {
81             Lower_Split(node[cur].rch, index, ltree, rtree);
82             node[cur].rch = ltree; PushUp(cur); ltree = cur;
83         }
84         else
85         {
86             Lower_Split(node[cur].lch, index, ltree, rtree);
87             node[cur].lch = rtree; PushUp(cur); rtree = cur;
88         }
89     }
90     void Upper_Split(int cur, int index, int& ltree, int& rtree)//index留在
ltree
91     {
92         if (cur == null)
93         {
94             ltree = rtree = null; return;
95         }
96         PushDown(cur);
97         if (node[cur].range.l > index)
98         {
99             Upper_Split(node[cur].lch, index, ltree, rtree);
100             node[cur].lch = rtree; PushUp(cur); rtree = cur;
101         }
102         else
103         {
104
105             Upper_Split(node[cur].rch, index, ltree, rtree);
106             node[cur].rch = ltree; PushUp(cur); ltree = cur;
107         }
108     }

```

```

109  /*=====*/
110  void SplitL(int root, int index, int& ltree, int& rtree)
111  {
112      int _temp, _ltree, _mtree, _rtree;
113      Upper_Split(root, index, _temp, _rtree);
114      Lower_Split(_temp, index, _ltree, _mtree);
115      int l = node[_mtree].range.l;
116      int r = node[_mtree].range.r;
117      int val = node[_mtree].range.val;
118      if (l != index)
119      {
120          ltree = Merge(_ltree, Creat(Range(l, index - 1, val)));
121          rtree = Merge(Creat(Range(index + 0, r, val)), _mtree);
122      }
123      else
124      {
125          ltree = _ltree;
126          rtree = Merge(_mtree, _rtree);
127      }
128  }
129  void SplitR(int root, int index, int& ltree, int& rtree)
130  {
131      int _temp, _ltree, _mtree, _rtree;
132      Upper_Split(root, index, _temp, _rtree);
133      Lower_Split(_temp, index, _ltree, _mtree);
134      int l = node[_mtree].range.l;
135      int r = node[_mtree].range.r;
136      int val = node[_mtree].range.val;
137      if (r != index)
138      {
139          ltree = Merge(_ltree, Creat(Range(l, index + 0, val)));
140          rtree = Merge(Creat(Range(index + 1, r, val)), _mtree);
141      }
142      else
143      {
144          ltree = Merge(_ltree, _mtree);
145          rtree = _rtree;
146      }
147  }
148  }

```

## 动态开点权值线段树

```

1  class C_SegmentTree
2  {
3  private:
4      struct Tree
5      {
6          int siz;
7          Tree* ls, * rs;
8          Tree(void)
9          {
10             siz = 0; ls = rs = NULL;
11         }

```

```

12     };
13     /*=====*/
14     Tree* null;
15     /*=====*/
16     Tree* root; int tree1, treer;
17     /*=====*/
18     Tree* Creat(int siz = 0)
19     {
20         Tree* cur = new Tree;
21         cur->siz = siz; cur->ls = cur->rs = null;
22         return cur;
23     }
24     /*=====*/
25     void Change(Tree*& cur, int valu, int delta, int tree1, int treer)
26     {
27         if (cur == null)cur = Creat();
28         cur->siz += delta;
29         if (tree1 != treer)
30         {
31             int mid = (tree1 + treer) >> 1;
32             if (valu <= mid)Change(cur->ls, valu, delta, tree1, mid + 0);
33             if (mid < valu) Change(cur->rs, valu, delta, mid + 1, treer);
34         }
35     }
36     /*=====*/
37     int GetValueByRank(Tree* cur, int rank, int tree1, int treer)
38     {
39         if (tree1 == treer)
40         {
41             return tree1;
42         }
43         else
44         {
45             int mid = (tree1 + treer) >> 1;
46             if (cur->ls->siz >= rank)
47             {
48                 return GetValueByRank(cur->ls, rank, tree1, mid + 0);
49             }
50             else
51             {
52                 return GetValueByRank(cur->rs, rank - cur->ls->siz, mid + 1,
treeer);
53             }
54         }
55     }
56     int GetRankByValu(Tree* cur, int valu, int tree1, int treer)
57     {
58         if (cur == null)
59         {
60             return 0;
61         }
62         else
63         {
64             if (treer < valu)
65             {

```

```

66         return cur->siz;
67     }
68     else
69     {
70         int res = 0;
71         int mid = (tree1 + treer) >> 1;
72         res += GetRankByValu(cur->ls, valu, tree1, mid + 0);
73         if (mid + 1 < valu) res += GetRankByValu(cur->rs, valu, mid
+ 1, treer);
74         return res;
75     }
76 }
77 }
78 public:
79     C_SegmentTree(void)
80     {
81         root = null = new Tree; tree1 = treer = 0;
82     }
83     ~C_SegmentTree(void)
84     {
85         if (root != null)
86         {
87             queue<Tree*>q; q.push(root);
88             while (!q.empty())
89             {
90                 if (q.front()->ls != null)q.push(q.front()->ls);
91                 if (q.front()->rs != null)q.push(q.front()->rs);
92                 delete q.front(); q.pop();
93             }
94         }
95         delete null;
96     }
97     /*=====*/
98     int Size(void)
99     {
100         return root->siz;
101     }
102     bool Empty(void)
103     {
104         return root->siz == 0;
105     }
106     void Init(int tree1, int treer)
107     {
108         this->tree1 = tree1; this->treer = treer;
109     }
110     void Erase(int valu)
111     {
112         Change(root, valu, -1, tree1, treer);
113     }
114     void Insert(int valu)
115     {
116         Change(root, valu, +1, tree1, treer);
117     }
118     int operator[](int rank)
119     {

```

```

120         return GetValueByRank(root, rank, tree1, treer);
121     }
122     int operator()(int valu)
123     {
124         return GetRankByValu(root, valu, tree1, treer) + 1;
125     }
126 };

```

## 数据类型

### 大数类

```

1  struct Bigint
2  {
3      int sign; string digits;
4      /*=====*/
5      Bigint(void) {}
6      Bigint(string b) { (*this) = b; }
7      Bigint(int b) { (*this) = to_string(b); }
8      /*=====*/
9      int size(void)
10     {
11         return digits.size();
12     }
13     Bigint inverseSign(void)
14     {
15         sign *= -1; return (*this);
16     }
17     Bigint normalize(int newSign)
18     {
19         for (int i = digits.size() - 1; i > 0 && digits[i] == '0'; i--)
20         {
21             digits.erase(digits.begin() + i);
22         }
23         sign = (digits.size() == 1 && digits[0] == '0') ? 1 : newSign;
24         return (*this);
25     }
26     /*=====*/
27     void operator = (string b)
28     {
29         digits = b[0] == '-' ? b.substr(1) : b;
30         reverse(digits.begin(), digits.end());
31         this->normalize(b[0] == '-' ? -1 : 1);
32     }
33     /*=====*/
34     bool operator < (const Bigint& b) const
35     {
36         if (sign != b.sign) return sign < b.sign;
37         if (digits.size() != b.digits.size())
38             return sign == 1 ? digits.size() < b.digits.size() :
39             digits.size() > b.digits.size();
40         for (int i = digits.size() - 1; i >= 0; i--) if (digits[i] !=
41             b.digits[i])

```

```

39         return sign == 1 ? digits[i] < b.digits[i] : digits[i] >
b.digits[i];
40         return false;
41     }
42     bool operator == (const Bigint& b) const
43     {
44         return digits == b.digits && sign == b.sign;
45     }
46     /*=====*/
47     Bigint operator + (Bigint b)
48     {
49         if (sign != b.sign) return (*this) - b.inverseSign();
50         Bigint c;
51         for (int i = 0, carry = 0; i < digits.size() || i < b.size() ||
carry; i++) {
52             carry += (i < digits.size() ? digits[i] - 48 : 0) + (i <
b.digits.size() ? b.digits[i] - 48 : 0);
53             c.digits += (carry % 10 + 48);
54             carry /= 10;
55         }
56         return c.normalize(sign);
57     }
58     Bigint operator - (Bigint b)
59     {
60         if (sign != b.sign) return (*this) + b.inverseSign();
61         int s = sign; sign = b.sign = 1;
62         if ((*this) < b) return ((b - (*this)).inverseSign()).normalize(-
s);
63         Bigint c;
64         for (int i = 0, borrow = 0; i < digits.size(); i++) {
65             borrow = digits[i] - borrow - (i < b.size() ? b.digits[i] :
48);
66             c.digits += borrow >= 0 ? borrow + 48 : borrow + 58;
67             borrow = borrow >= 0 ? 0 : 1;
68         }
69         return c.normalize(s);
70     }
71     Bigint operator * (Bigint b)
72     {
73         Bigint c("0");
74         for (int i = 0, k = digits[i] - 48; i < digits.size(); i++, k =
digits[i] - 48) {
75             while (k--) c = c + b;
76             b.digits.insert(b.digits.begin(), '0');
77         }
78         return c.normalize(sign * b.sign);
79     }
80     Bigint operator / (Bigint b)
81     {
82         if (b.size() == 1 && b.digits[0] == '0') b.digits[0] /=
(b.digits[0] - 48);
83         Bigint c("0"), d;
84         for (int j = 0; j < digits.size(); j++) d.digits += "0";
85         int dsign = sign * b.sign; b.sign = 1;
86         for (int i = digits.size() - 1; i >= 0; i--) {

```

```

87         c.digits.insert(c.digits.begin(), '0');
88         c = c + digits.substr(i, 1);
89         while (!(c < b)) c = c - b, d.digits[i]++;
90     }
91     return d.normalize(dSign);
92 }
93 Bigint operator % (Bigint b)
94 {
95     if (b.size() == 1 && b.digits[0] == '0') b.digits[0] /=
(b.digits[0] - 48);
96     Bigint c("0");
97     b.sign = 1;
98     for (int i = digits.size() - 1; i >= 0; i--) {
99         c.digits.insert(c.digits.begin(), '0');
100        c = c + digits.substr(i, 1);
101        while (!(c < b)) c = c - b;
102    }
103    return c.normalize(sign);
104 }
105 /*=====*/
106 friend ostream& operator<<(ostream& output, Bigint integer)
107 {
108     if (integer.sign == -1) output << "-";
109     for (int i = integer.digits.size() - 1; i >= 0; i--)
110     {
111         output << integer.digits[i];
112     }
113     return output;
114 }
115 friend istream& operator>>(istream& input, Bigint& integer)
116 {
117     string str; input >> str; integer = str; return input;
118 }
119 };

```

## 分数类

```

1  class Fraction
2  {
3  public:
4      int up, dw;
5  private:
6      int GCD(int a, int b)
7      {
8          return b == 0 ? a : GCD(b, a % b);
9      }
10     void Simplest(void)
11     {
12         int divisor = GCD(up, dw);
13         if (divisor != 0)
14         {
15             up /= divisor, dw /= divisor;
16             if (dw < 0) dw *= -1, up *= -1;
17         }

```



```

18     }
19 public:
20     Fraction(const Fraction& temp)
21     {
22         up = temp.up, dw = temp.dw;
23     }
24     Fraction(int _up = 0, int _dw = 1)
25     {
26         up = _up, dw = _dw; Simplest();
27     }
28     /*=====*/
29     double Val(void)
30     {
31         return double(up) / double(dw);
32     }
33     /*=====*/
34     void operator+=(const Fraction& x)
35     {
36         up = up * x.dw + x.up * dw; dw = dw * x.dw; Simplest();
37     }
38     void operator-=(const Fraction& x)
39     {
40         up = up * x.dw - x.up * dw; dw = dw * x.dw; Simplest();
41     }
42     void operator*=(const Fraction& x)
43     {
44         up = up * x.up; dw = dw * x.dw; Simplest();
45     }
46     void operator/=(const Fraction& x)
47     {
48         up = up * x.dw; dw = dw * x.up; Simplest();
49     }
50     /*=====*/
51     friend Fraction operator+(const Fraction& a, const Fraction& b)
52     {
53         return Fraction(a.up * b.dw + b.up * a.dw, a.dw * b.dw);
54     }
55     friend Fraction operator-(const Fraction& a, const Fraction& b)
56     {
57         return Fraction(a.up * b.dw - b.up * a.dw, a.dw * b.dw);
58     }
59     friend Fraction operator*(const Fraction& a, const Fraction& b)
60     {
61         return Fraction(a.up * b.up, a.dw * b.dw);
62     }
63     friend Fraction operator/(const Fraction& a, const Fraction& b)
64     {
65         return Fraction(a.up * b.dw, a.dw * b.up);
66     }
67     /*=====*/
68     friend bool operator<(const Fraction& a, const Fraction& b)
69     {
70         return (a.up * b.dw) < (b.up * a.dw);
71     }
72     friend bool operator>(const Fraction& a, const Fraction& b)

```

```

73     {
74         return (a.up * b.dw) > (b.up * a.dw);
75     }
76     friend bool operator==(const Fraction& a, const Fraction& b)
77     {
78         return (a.up == b.up) && (a.dw == b.dw);
79     }
80     friend bool operator!=(const Fraction& a, const Fraction& b)
81     {
82         return !(a == b);
83     }
84     friend bool operator<=(const Fraction& a, const Fraction& b)
85     {
86         return !(a > b);
87     }
88     friend bool operator>=(const Fraction& a, const Fraction& b)
89     {
90         return !(a < b);
91     }
92 };

```

## 模数类

```

1  template<int MOD = 998244353>
2  class Modulo
3  {
4  private:
5      static int Pow(int a, int b)
6      {
7          int res = 1;
8          while (b)
9          {
10             if (b & 1)
11             {
12                 res = (1ll * res * a) % MOD;
13             }
14             b >>= 1, a = (1ll * a * a) % MOD;
15         }
16         return res;
17     }
18     static int Inv(int x)
19     {
20         return Pow(x, MOD - 2);
21     }
22 public:
23     int num;
24     /*=====*/
25     Modulo(int temp = 0)
26     {
27         num = temp % MOD;
28     }
29     Modulo(const Modulo& temp)
30     {
31         num = temp.num;

```

```

32     }
33     /*=====*/
34     friend Modulo operator+(const Modulo& a, const Modulo& b)
35     {
36         return Modulo((a.num + b.num) >= MOD ? a.num + b.num - MOD : a.num
+ b.num);
37     }
38     friend Modulo operator-(const Modulo& a, const Modulo& b)
39     {
40         return Modulo((a.num - b.num + MOD) >= MOD ? a.num - b.num : a.num
- b.num + MOD);
41     }
42     friend Modulo operator*(const Modulo& a, const Modulo& b)
43     {
44         return Modulo(a.num * b.num % MOD);
45     }
46     friend Modulo operator/(const Modulo& a, const Modulo& b)
47     {
48         return Modulo(a.num * Inv(b.num) % MOD);
49     }
50     /*=====*/
51     friend bool operator< (const Modulo& a, const Modulo& b)
52     {
53         return a.num < b.num;
54     }
55     friend bool operator==(const Modulo& a, const Modulo& b)
56     {
57         return a.num == b.num;
58     }
59     friend bool operator> (const Modulo& a, const Modulo& b)
60     {
61         return a.num > b.num;
62     }
63     friend bool operator<=(const Modulo& a, const Modulo& b)
64     {
65         return a.num <= b.num;
66     }
67     friend bool operator!=(const Modulo& a, const Modulo& b)
68     {
69         return a.num != b.num;
70     }
71     friend bool operator>=(const Modulo& a, const Modulo& b)
72     {
73         return a.num >= b.num;
74     }
75     /*=====*/
76     void operator+=(const Modulo& x)
77     {
78         num = num + x.num; if (num >= MOD) num -= MOD;
79     }
80     void operator-=(const Modulo& x)
81     {
82         num = num - x.num + MOD; if (num >= MOD) num -= MOD;
83     }
84     void operator*=(const Modulo& x)

```

```

85     {
86         num = num * x.num % MOD;
87     }
88     void operator/=(const Modulo& x)
89     {
90         num = num * Inv(x.num) % MOD;
91     }
92     /*=====*/
93     friend ostream& operator<<(ostream& output, Modulo integer)
94     {
95         output << integer.num; return output;
96     }
97     friend istream& operator>>(istream& input, Modulo& integer)
98     {
99         int temp; input >> temp; integer = (temp % MOD + MOD) % MOD; return
input;
100     }
101 };

```

# 数学

## 逆元

### 离线法

```

1  class C_INV
2  {
3  private:
4      int Pow(int a, int b, int P)
5      {
6          int res = 1;
7          while (b)
8          {
9              if (b & 1)
10             {
11                 res = (1ll * res * a) % P;
12             }
13             b >>= 1, a = (1ll * a * a) % P;
14         }
15         return res;
16     }
17 public:
18     vector<int> operator()(vector<int>& arr, int P)
19     {
20         vector<int>sum = arr;
21         for (int i = 1; i < sum.size(); ++i)
22         {
23             sum[i] = 1ll * sum[i - 1] * arr[i] % P;
24         }
25         vector<int>inv = sum;
26         inv.back() = Pow(inv.back(), P - 2, P);
27         for (int i = inv.size() - 2; i >= 0; --i)
28         {
29             inv[i] = 1ll * inv[i + 1] * arr[i + 1] % P;

```

```

30     }
31     for (int i = 1; i < inv.size(); ++i)
32     {
33         inv[i] = 111 * sum[i - 1] * inv[i] % P;
34     }
35     return inv;
36 }
37 };

```

## 快速幂法

```

1  class C_INV
2  {
3  private:
4      int P;
5      /*=====*/
6      int Pow(int a, int b, int P)
7      {
8          int res = 1;
9          while (b)
10         {
11             if (b & 1)
12             {
13                 res = (111 * res * a) % P;
14             }
15             b >>= 1, a = (111 * a * a) % P;
16         }
17         return res;
18     }
19 public:
20     void Init(int P)
21     {
22         this->P = P;
23     }
24     int operator[](int x)
25     {
26         return Pow(x % P, P - 2, P);
27     }
28 };

```

## Farey序列

```

1  class C_INV
2  {
3  private:
4      struct Frac
5      {
6          int up, dw;
7          Frac(int _up = 0, int _dw = 0)
8          {
9              up = _up, dw = _dw;
10         }
11     };
12     /*=====*/

```

```

13     int P, B1, B2;
14     vector<Frac>s;
15     vector<int>inv;
16     vector<int>pre, suf;
17     /*=====*/
18     int Calc(int x, Frac frac)
19     {
20         int pos = 111 * x * frac.dw;
21         if (abs(pos - 111 * P * frac.up) > P / B1) return -1;
22         if ((pos %= P) <= P / B1) return 111 * frac.dw * inv[pos] % P;
23         return P - 111 * frac.dw * inv[P - pos] % P;
24     }
25 public:
26     void Init(int P)
27     {
28         this->P = P;
29
30         B1 = pow(P, 1.0 / 3); B2 = B1 * B1;
31
32         s.assign(B2 + 1, Frac());
33         inv.assign(P / B1 + 1, 0);
34         pre.assign(B2 + 1, 0); suf.assign(B2 + 1, 0);
35
36         inv[1] = 1;
37         for (int i = 2; i <= P / B1; ++i)
38         {
39             inv[i] = 111 * (P - P / i) * inv[P % i] % P;
40         }
41
42         s[0] = Frac(0, 1); s[B2] = Frac(1, 1);
43         pre[B2] = suf[B2] = B2;
44         for (int i = 2; i <= B1; ++i)
45         {
46             for (int j = 1; j < i; ++j)
47             {
48                 int pos = 111 * j * B2 / i;
49                 if (pre[pos]) continue;
50                 pre[pos] = suf[pos] = pos;
51                 s[pos] = Frac(j, i);
52             }
53         }
54
55         for (int i = 1; i <= B2; ++i) if (!pre[i]) pre[i] = pre[i - 1];
56         for (int i = B2; i >= 0; --i) if (!suf[i]) suf[i] = suf[i + 1];
57     }
58     int operator[](int x)
59     {
60         if (x <= P / B1) return inv[x];
61         int pos = 111 * x * B2 / P, res = Calc(x, s[pre[pos]]);
62         if (res == -1) res = Calc(x, s[suf[pos]]);
63         return res;
64     }
65 };

```

## 线性递推法

```
1  class C_INV
2  {
3  private:
4      vector<int>inv;
5  public:
6      void Init(int n, int P)
7      {
8          inv.assign(n + 1, 0); inv[1] = 1;
9          for (int i = 2; i <= n; ++i)
10         {
11             inv[i] = 1ll * (P - P / i) * inv[P % i] % P;
12         }
13     }
14     int operator[](int x)
15     {
16         return inv[x];
17     }
18 };
```

## 扩展欧几里得法

```
1  class C_INV
2  {
3  private:
4      int P;
5      /*=====*/
6      void exgcd(int a, int b, int& x, int& y)
7      {
8          if (b == 0)
9          {
10             x = 1, y = 0;
11         }
12         else
13         {
14             exgcd(b, a % b, y, x);
15             y -= a / b * x;
16         }
17     }
18  public:
19      void Init(int P)
20      {
21          this->P = P;
22      }
23      int operator[](int x)
24      {
25          int a, b;
26          exgcd(x, P, a, b);
27          return (a % P + P) % P;
28      }
29  };
```

# 质数

## 判定

### 六倍试除法

```
1  class C_Prime
2  {
3  public:
4      bool operator()(int x)
5      {
6          if (x <= 1) return false;
7          if (x == 2 || x == 3 || x == 5) return true;
8          if (x % 2 == 0 || x % 3 == 0) return false;
9          for (int i = 5; i * i <= x; i += 6)
10         {
11             if (x % i == 0 || x % (i + 2) == 0)
12             {
13                 return false;
14             }
15         }
16         return true;
17     }
18 };
```

### Miller-Rabin

```
1  class C_Prime
2  {
3  private:
4      int Mul(int a, int b, int p)
5      {
6          return (__int128)a * b % p;
7      }
8      int Pow(int a, int b, int p)
9      {
10         int res = 1;
11         while (b)
12         {
13             if (b % 2 == 1)
14             {
15                 res = Mul(res, a, p);
16             }
17             b /= 2, a = Mul(a, a, p);
18         }
19         return res;
20     }
21 public:
22     bool operator()(int x)
23     {
24         if (x < 3 || x % 2 == 0) return x == 2;
25         int a = x - 1, b = 0;
26         while (a % 2 == 0) a /= 2, ++b;
27         //int lib[] = { 2, 7, 61 };
```



```

28     int lib[] = { 2,325,9375,28178,450775,9780504,1795265022 };
29     for (int r : lib)
30     {
31         int v = Pow(r, a, x);
32         if (v == 1 || v == x - 1 || v == 0) continue;
33         for (int j = 1; j <= b; j++)
34         {
35             v = Mul(v, v, x);
36             if (v == x - 1 && j != b) { v = 1; break; }
37             if (v == 1) return false;
38         }
39         if (v != 1) return false;
40     }
41     return true;
42 }
43 };

```

## 筛法

### 欧拉筛

```

1  class C_Prime
2  {
3  private:
4      vector<int>prime;
5      vector<bool>isprime;
6  public:
7      void Init(int n)
8      {
9          isprime.assign(n + 1, true);
10         isprime[0] = isprime[1] = false;
11         for (int i = 2; i <= n; ++i)
12         {
13             if (isprime[i])prime.push_back(i);
14             for (int j = 0; j < prime.size(); ++j)
15             {
16                 if (i * prime[j] > n)break;
17                 isprime[i * prime[j]] = false;
18                 if (i % prime[j] == 0)break;
19             }
20         }
21     }
22     int Size(void)
23     {
24         return prime.size();
25     }
26     bool operator()(int x)
27     {
28         return isprime[x];
29     }
30     int operator[](int x)
31     {
32         return prime[x - 1];
33     }
34 };

```

## 快速幂

```
1  int Pow(int a, int b, const int P)
2  {
3      int res = 1;
4      while (b)
5      {
6          if (b & 1)
7          {
8              res = (res * a) % P;
9          }
10         b >>= 1, a = (a * a) % P;
11     }
12     return res;
13 }
```

## 龟速乘

```
1  int Mul(int a, int b, int p)
2  {
3      int res = 0;
4      while (b)
5      {
6          if (b & 1)
7          {
8              res = (res + a) % p;
9          }
10         b >>= 1, a = (a * 2) % p;
11     }
12     return res;
13 }
14 int Mul(int a, int b, int p)
15 {
16     return (a * b - (int)(a / (long double)p * b + 1e-3) * p + p) % p;
17 }
18 int Mul(int a, int b, int p)
19 {
20     return (__int128)a * b % p;
21 }
```

## 逆序对

```
1  template<class Type>
2  class _CountInversions
3  {
4  public:
5      int operator()(int n, Type arr[])
6      {
7          res = 0;
8          a = new Type[n + 1];
9          temp = new Type[n + 1];
10         for (int i = 1; i <= n; ++i)
```

```

11     {
12         a[i] = arr[i];
13     }
14     MergeSort(l, n);
15     delete[] a; delete[] temp;
16     return res;
17 }
18 private:
19     Type* a = NULL;
20     lnt res = 0;
21     Type* temp = NULL;
22     /*=====*/
23     void MergeSort(int l, int r)
24     {
25         if (l < r)
26         {
27             int i = l;
28             int mid = (l + r) >> 1;
29             int p = l, q = mid + 1;
30             MergeSort(l, mid + 0);
31             MergeSort(mid + 1, r);
32             while (p <= mid && q <= r)
33             {
34                 if (a[p] <= a[q])
35                 {
36                     temp[i++] = a[p++];
37                 }
38                 else
39                 {
40                     temp[i++] = a[q++];
41                     res += (lnt)(mid - p + 1);
42                 }
43             }
44             while (p <= mid) temp[i++] = a[p++];
45             while (q <= r) temp[i++] = a[q++];
46             for (i = l; i <= r; i++) a[i] = temp[i];
47         }
48     }
49 };

```

## 多项式

### 快速傅里叶变换

```

1     typedef complex<double> Comp;
2
3     const ll SIZE = 4000000 + 10;
4     const double PI = acos(-1.0);
5
6     Comp temp[SIZE];
7     void FFT(Comp* p, ll len, ll inv) //inv==1 FFT; inv=-1 IFFT
8     {
9         if (len == 1) return;
10        const int E = 0, O = len / 2;

```

```

11     for (ll i = 0; i < len; ++i) temp[i] = p[i];
12     for (ll i = 0; i < len; ++i)
13     {
14         if ((i & 1) == 1) p[i / 2 + 0] = temp[i];
15         if ((i & 1) == 0) p[i / 2 + E] = temp[i];
16     }
17     Comp* pe = p + E; FFT(pe, len / 2, inv);
18     Comp* po = p + 0; FFT(po, len / 2, inv);
19     Comp omega(1, 0);
20     const double Angle = 2 * PI / len;
21     const Comp step(cos(Angle), sin(inv * Angle));
22     for (ll k = 0; k < len / 2; ++k, omega *= step)
23     {
24         temp[k + E] = pe[k] + omega * po[k];
25         temp[k + 0] = pe[k] - omega * po[k];
26     }
27     for (ll i = 0; i < len; ++i) p[i] = temp[i];
28 }

```

## 离散对数

```

1  ll BSGS(ll a, ll b, ll m)
2  {
3      static unordered_map<ll, ll> hs;
4      hs.clear();
5      ll cur = 1, t = sqrt(m) + 1;
6      for (int B = 1; B <= t; ++B)
7      {
8          (cur *= a) %= m;
9          hs[b * cur % m] = B; // 哈希表中存B的值
10     }
11     ll now = cur; // 此时cur = a^t
12     for (int A = 1; A <= t; ++A)
13     {
14         auto it = hs.find(now);
15         if (it != hs.end())
16             return A * t - it->second;
17         (now *= cur) %= m;
18     }
19     return -1; // 没有找到, 无解
20 }

```

## 基数排序

```

1  template <int B = 8, class Type>
2  void RadixSort(vector<Type>& a)
3  {
4      const int mask = (1 << B) - 1, n = a.size();
5      vector<Type> b(n); vector<int> cnt(1 << B);
6      Type maxV = *max_element(a.begin(), a.end());
7      for (int i = 0; maxV; i += B, maxV >>= B)
8      {
9          fill(cnt.begin(), cnt.end(), 0);

```

```

10         for (int j = 0; j < n; j++)cnt[a[j] >> i & mask] += 1;
11         for (int j = 1; j < (1 << B); j++)cnt[j] += cnt[j - 1];
12         for (int j = n - 1; j >= 0; j--)b[--cnt[a[j] >> i & mask]] = a[j];
13         swap(a, b);
14     }
15 }

```

## 欧拉函数

### 试除法

```

1  class PHI
2  {
3  public:
4      int operator()(int x)
5      {
6          return GetPhi(x);
7      }
8  private:
9      int GetPhi(int x)
10     {
11         int res = x;
12         for (int i = 2; i * i <= x; ++i)
13         {
14             if (x % i == 0)
15             {
16                 res = res / i * (i - 1);
17                 while (x % i == 0) x /= i;
18             }
19         }
20         if (x > 1) res = res / x * (x - 1);
21         return res;
22     }
23 };

```

### 欧拉筛法

```

1  class PHI
2  {
3  public:
4      ~PHI(void)
5      {
6          delete[] phi;
7      }
8      void init(int n)
9      {
10         GetPhi(n);
11     }
12     int operator()(int x)
13     {
14         return phi[x];
15     }
16 private:

```

```

17     int* phi = NULL;
18     void GetPhi(int n)
19     {
20         phi = new int[n + 1];
21         bool* vis = new bool[n + 1];
22         int* table = new int[n + 1];
23         for (int i = 0; i <= n; ++i)
24         {
25             vis[i] = true;
26         }
27         int cnt = 0; phi[1] = 1;
28         for (int i = 2; i <= n; ++i)
29         {
30             if (vis[i])
31             {
32                 phi[i] = i - 1;
33                 table[++cnt] = i;
34             }
35             for (int j = 1; j <= cnt; ++j)
36             {
37                 if (i * table[j] > n)break;
38                 vis[i * table[j]] = false;
39                 if (i % table[j] == 0)
40                 {
41                     phi[i * table[j]] = phi[i] * table[j]; break;
42                 }
43                 else
44                 {
45                     phi[i * table[j]] = phi[i] * (table[j] - 1);
46                 }
47             }
48         }
49         delete[] vis; delete[] table;
50     }
51 };

```

## 欧拉降幂

```

1     class EX_Euler
2     {
3     public:
4         int operator()(int a, string s, int p)
5         {
6             int b = 0;
7             bool flag = false;
8             int phi = GetPhi(p);
9             for (auto c : s)
10            {
11                b = (b * 10 + c - '0');
12                if (b >= phi)flag = true, b %= phi;
13            }
14            if (flag)b += phi; return Pow(a % p, b, p);
15        }
16    private:

```

```

17     int GetPhi(int x)
18     {
19         int res = x;
20         for (int i = 2; i * i <= x; ++i)
21         {
22             if (x % i == 0)
23             {
24                 res = res / i * (i - 1);
25                 while (x % i == 0) x /= i;
26             }
27         }
28         if (x > 1) res = res / x * (x - 1);
29         return res;
30     }
31     int Pow(int a, int b, int p)
32     {
33         int res = 1;
34         while (b != 0)
35         {
36             if (b % 2 == 1)
37             {
38                 res = (res * a) % p;
39             }
40             b /= 2, a = (a * a) % p;
41         }
42         return res;
43     }
44 };

```

## 欧几里得

### 最大公因数

```

1  int gcd(int a, int b)
2  {
3      return b == 0 ? a : gcd(b, a % b);
4  }

```

### 最小公倍数

```

1  int lcm(int a, int b)
2  {
3      return a / gcd(a, b) * b;
4  }

```

## 扩展欧几里得

```

1 void exgcd(int a, int b, int& x, int& y)
2 {
3     if (b == 0)
4     {
5         x = 1, y = 0;
6     }
7     else
8     {
9         exgcd(b, a % b, y, x);
10        y -= a / b * x;
11    }
12 }

```

## 分解质因数

### Pollard-Rho

```

1 class C_PrimeFactorization
2 {
3 private:
4     lnt Mul(lnt a, lnt b, lnt p)
5     {
6         return (__int128)a * b % p;
7     }
8     lnt Pow(lnt a, lnt b, lnt p)
9     {
10        lnt res = 1;
11        while (b)
12        {
13            if (b % 2 == 1)
14            {
15                res = Mul(res, a, p);
16            }
17            b /= 2, a = Mul(a, a, p);
18        }
19        return res;
20    }
21    bool MillerRabin(lnt x)
22    {
23        if (x < 3 || x % 2 == 0) return x == 2;
24        lnt a = x - 1, b = 0;
25        while (a % 2 == 0) a /= 2, ++b;
26        //lnt lib[] = { 2,7,61 };
27        lnt lib[] = { 2,325,9375,28178,450775,9780504,1795265022 };
28        for (lnt r : lib)
29        {
30            lnt v = Pow(r, a, x);
31            if (v == 1 || v == x - 1 || v == 0) continue;
32            for (lnt j = 1; j <= b; j++)
33            {
34                v = Mul(v, v, x);
35                if (v == x - 1 && j != b) { v = 1; break; }
36                if (v == 1) return false;
37            }

```



```

38         if (v != 1) return false;
39     }
40     return true;
41 }
42 /*=====*/
43 Int GCD(Int a, Int b)
44 {
45     return b == 0 ? a : GCD(b, a % b);
46 }
47 Int GetFactor(Int x)
48 {
49     if (x == 4) return 2;
50     if (MillerRabin(x)) return x;
51     mt19937 rng(time(NULL));
52     while (1)
53     {
54         Int c = rng() % (x - 1) + 1;
55         auto f = [=](Int x) { return ((__int128)x * x + c) % x; };
56         Int t = 0, r = 0, p = 1, q;
57         do
58         {
59             for (int i = 0; i < 128; ++i)
60             {
61                 t = f(t), r = f(f(r));
62                 if (t == r || (q = (__int128)p * abs(t - r) % x) ==
0) break;
63                 p = q;
64             }
65             Int d = GCD(p, x); if (d > 1) return d;
66         } while (t != r);
67     }
68 }
69 void GetAllFactor(Int x, vector<Int>& lib)
70 {
71     Int fac = GetFactor(x);
72     if (fac == x) lib.push_back(fac);
73     else GetAllFactor(fac, lib), GetAllFactor(x / fac, lib);
74 }
75 public:
76 void operator()(Int x, vector<Int>& num, vector<Int>& cnt)
77 {
78     num.clear(), cnt.clear();
79     GetAllFactor(x, num);
80     sort(num.begin(), num.end());
81     for (int i = 0; i < num.size(); ++i)
82     {
83         if (i == 0 || num[i] != num[i - 1])
84         {
85             cnt.push_back(0);
86         }
87         cnt.back()++;
88     }
89     num.erase(unique(num.begin(), num.end()), num.end());
90 }
91 };

```

## 预处理质数表法

```
1  class C_PrimeFactorization
2  {
3  private:
4      vector<int>prime;
5  public:
6      void Init(int n)
7      {
8          vector<bool>isprime;
9          isprime.assign(n + 1, true);
10         isprime[0] = isprime[1] = false;
11         for (int i = 2; i <= n; ++i)
12         {
13             if (isprime[i])prime.push_back(i);
14             for (int j = 0; j < prime.size(); ++j)
15             {
16                 if (i * prime[j] > n)break;
17                 isprime[i * prime[j]] = false;
18                 if (i % prime[j] == 0)break;
19             }
20         }
21     }
22     void operator()(int x, vector<int>& num, vector<int>& cnt)
23     {
24         num.clear(), cnt.clear();
25         for (auto p : prime)
26         {
27             if (p * p > x)break;
28             if (x % p == 0)
29             {
30                 num.push_back(p), cnt.push_back(0);
31                 while (x % p == 0)cnt.back()++, x /= p;
32             }
33         }
34         if (x != 1)
35         {
36             num.push_back(x), cnt.push_back(1);
37         }
38     }
39 };
```

## 预处理最小质因子法

```
1  class C_PrimeFactorization
2  {
3  private:
4      vector<int>mpf;
5  public:
6      void Init(int n)
7      {
8          vector<int>prime;
9          vector<bool>isprime;
10         mpf.assign(n + 1, 0);
```

```

11     isprime.assign(n + 1, true);
12     isprime[0] = isprime[1] = false;
13     for (int i = 2; i <= n; ++i)
14     {
15         if (isprime[i]) prime.push_back(i), mpf[i] = i;
16         for (int j = 0; j < prime.size(); ++j)
17         {
18             if (i * prime[j] > n) break;
19             isprime[i * prime[j]] = false;
20             mpf[i * prime[j]] = prime[j];
21             if (i % prime[j] == 0) break;
22         }
23     }
24 }
25 void operator()(int x, vector<int>& num, vector<int>& cnt)
26 {
27     num.clear(), cnt.clear();
28     while (x > 1)
29     {
30         if (num.empty() || num.back() != mpf[x])
31         {
32             num.push_back(mpf[x]); cnt.push_back(1);
33         }
34         else
35         {
36             cnt.back()++;
37         }
38         x /= mpf[x];
39     }
40 }
41 };

```

## 获得全部因数

```

1 void GetDivisor(int x, vector<int>& divisor)
2 {
3     vector<int> num, cnt;
4     PFF(x, num, cnt);
5     divisor.push_back(1);
6     for (int i = 0; i < num.size(); ++i)
7     {
8         int val = 1;
9         int lim = divisor.size();
10        for (int j = 1; j <= cnt[i]; ++j)
11        {
12            val *= num[i];
13            for (int k = 0; k < lim; ++k)
14            {
15                divisor.push_back(divisor[k] * val);
16            }
17        }
18    }
19 }

```

# 随机化

## 随机数生成器

```
1 | mt19937 Rand(random_device{}());
```

## 模拟退火

```
1 | #include<bits/stdc++.h>
2 | using namespace std;
3 | /*=====*/
4 | #define endl "\n"
5 | /*=====*/
6 | typedef long long lnt;
7 | /*=====*/
8 | const int N = 1e3 + 10;
9 | /*=====*/
10 | double begintime = 0;
11 | bool TLE(void)
12 | {
13 |     if ((clock() - begintime) / CLOCKS_PER_SEC > 0.9)
14 |     {
15 |         return true;
16 |     }
17 |     return false;
18 | }
19 | /*=====*/
20 | int n;
21 | /*=====*/
22 | struct Node
23 | {
24 |     double w, x, y;
25 |     Node(double _w = 0, double _x = 0, double _y = 0)
26 |     {
27 |         w = _w, x = _x, y = _y;
28 |     }
29 | };
30 | Node node[N];
31 | /*=====*/
32 | double ansx, ansy, ansSigma = 1e18;
33 | /*=====*/
34 | double GetSigma(double x, double y)
35 | {
36 |     double res = 0;
37 |     for (int i = 1; i <= n; ++i)
38 |     {
39 |         double dx = node[i].x - x;
40 |         double dy = node[i].y - y;
41 |         res += sqrt(dx * dx + dy * dy) * node[i].w * 100;
42 |     }
43 |     if (res < ansSigma)
44 |     {
45 |         ansx = x, ansy = y, ansSigma = res;
```

```

46     }
47     return res;
48 }
49 /*=====*/
50 double Rand() { return (double)rand() / RAND_MAX; }
51 /*=====*/
52 void SA(void)
53 {
54     double curx = 0, cury = 0, curSigma = 0;
55     for (int i = 1; i <= n; ++i)
56     {
57         curx += node[i].x, cury += node[i].y;
58     }
59     curx /= n, cury /= n; curSigma = GetSigma(curx, cury);
60
61     double t = 1e4;
62     while (t > 5e-4)
63     {
64         double nextx = curx + t * (Rand() * 2.0 - 1.0);
65         double nexty = cury + t * (Rand() * 2.0 - 1.0);
66         double nextSigma = GetSigma(nextx, nexty);
67         double delta = nextSigma - curSigma;
68         if (exp(-delta / t) > Rand())
69         {
70             curx = nextx, cury = nexty, curSigma = nextSigma;
71         }
72         t *= 0.9996;
73     }
74     for (int i = 1; i <= 5000; ++i)
75     {
76         double nextx = ansx + t * (Rand() * 2 - 1);
77         double nexty = ansy + t * (Rand() * 2 - 1);
78         double nextSigma = GetSigma(nextx, nexty);
79     }
80 }
81 /*=====*/
82 void solve(void)
83 {
84     cin >> n;
85     for (int i = 1; i <= n; ++i)
86     {
87         double w, x, y;
88         cin >> x >> y >> w;
89         node[i] = Node(w, x, y);
90     }
91     while (!TLE())SA();
92     printf("%.3f %.3f\n", ansx, ansy);
93 }
94 /*=====*/
95 int main()
96 {
97     #ifndef ONLINE_JUDGE
98         freopen("IN.txt", "r+", stdin);
99     #endif
100     srand(time(0)); begintime = clock();

```

```

101     ios::sync_with_stdio(false);
102     cin.tie(NULL), cout.tie(NULL);
103     int T = 1; //cin >> T;
104     while (T--) Solve();
105     return 0;
106 }

```

## 图论

### 存储

```

1  class C_Graph
2  {
3  public:
4      struct Edge
5      {
6          int u, v, w;
7          Edge(int _u = 0, int _v = 0, int _w = 0)
8          {
9              u = _u, v = _v, w = _w;
10         }
11         friend bool operator<(const Edge& a, const Edge& b)
12         {
13             return a.w < b.w;
14         }
15         int node(int x) const
16         {
17             return x == u ? v : u;
18         }
19     };
20     /*=====*/
21     int n;
22     vector<Edge> edge;
23     vector<vector<int>> G;
24     /*=====*/
25     void Init(int n)
26     {
27         this->n = n; G.assign(n + 1, vector<int>());
28     }
29     void AddEdge(int u, int v, int w)
30     {
31         edge.push_back(Edge(u, v, w));
32         G[u].push_back(edge.size() - 1);
33     }
34 };

```

## 2-SAT

```

1  namespace _TwoSAT
2  {
3      using namespace _SCC;
4      void Init(void)
5      {

```

```

6         for (int i = 1; i <= n; ++i)
7         {
8             int u = idx[i][1], v = idx[i][0];
9             if (belong[u] == belong[v])
10            {
11                cout << "IMPOSSIBLE" << endl; return;
12            }
13        }
14        cout << "POSSIBLE" << endl;
15        for (int i = 1; i <= n; ++i)
16        {
17            int u = idx[i][1], v = idx[i][0];
18            cout << ((belong[u] < belong[v]) ? 1 : 0) << " ";
19        }
20        cout << endl;
21    }
22 }

```

## 欧拉图

## 无向图

```

1 namespace _Euler
2 {
3     /*
4     默认连通图, -1不存在, 0存在欧拉路径, 1存在欧拉回路
5     */
6     const int N = 1e5 + 10;
7     /*=====*/
8     int degree[N];
9     /*=====*/
10    int Init(void)
11    {
12        for (int i = 1; i <= n; ++i)
13        {
14            degree[i] = G[i].size();
15        }
16        int cnt = 0;
17        for (int i = 1; i <= n; ++i)
18        {
19            if (degree[i] % 2 == 1) cnt++;
20        }
21        return cnt == 0 ? 1 : (cnt == 2 ? 0 : -1);
22    }
23 }

```

## 有向图

```

1 namespace _Euler
2 {
3     /*
4     默认连通图, -1不存在, 0存在欧拉路径, 1存在欧拉回路
5     */

```

```

6      const int N = 1e5 + 10;
7      /*=====*/
8      int degree[N];
9      /*=====*/
10     int Init(void)
11     {
12         for (int i = 1; i <= n; ++i)
13         {
14             degree[i] = 0;
15         }
16         for (int i = 1; i <= m; ++i)
17         {
18             int u = edge[i].u;
19             int v = edge[i].v;
20             degree[u]++, degree[v]--;
21         }
22         int cnt1 = 0, cnt2 = 0, cnt3 = 0;
23         for (int i = 1; i <= n; ++i)
24         {
25             if (degree[i] == -1) cnt1++;
26             if (degree[i] == +0) cnt2++;
27             if (degree[i] == +1) cnt3++;
28         }
29         if (cnt1 == 1 && cnt3 == 1 && cnt2 + 2 == n)
30         {
31             return +0;
32         }
33         else if (cnt2 == n)
34         {
35             return +1;
36         }
37         else
38         {
39             return -1;
40         }
41     }
42 }

```

## 最大团

```

1  namespace _MaxClique
2  {
3      const int N = 5e1 + 10;
4      /*=====*/
5      int n; int G[N][N];
6      int dp[N], stk[N][N], res;
7      /*=====*/
8      bool DFS(int ns, int dep)
9      {
10         if (ns == 0)
11         {
12             if (dep > res)
13             {
14                 res = dep; return true;

```



```

15     }
16     return false;
17 }
18 for (int i = 0; i < ns; ++i)
19 {
20     int u = stk[dep][i], cnt = 0;
21     if (dep + dp[u] <= res) return false;
22     if (dep + ns - i <= res) return false;
23     for (int j = i + 1; j < ns; ++j)
24     {
25         int v = stk[dep][j];
26         if (G[u][v]) stk[dep + 1][cnt++] = v;
27     }
28     if (DFS(cnt, dep + 1)) return true;
29 }
30 return false;
31 }
32 /*=====*/
33 int Init(void)
34 {
35     cin >> n; res = 0;
36     memset(dp, 0, sizeof(dp));
37     for (int i = 1; i <= n; ++i)
38     {
39         for (int j = 1; j <= n; ++j)
40         {
41             cin >> G[i][j];
42         }
43     }
44     for (int i = n; i >= 1; --i)
45     {
46         int ns = 0;
47         for (int j = i + 1; j <= n; ++j)
48         {
49             if (G[i][j]) stk[1][ns++] = j;
50         }
51         DFS(ns, 1); dp[i] = res;
52     }
53     return res;
54 }
55 }

```

## 最短路

### SPFA

```

1 namespace _SPFA
2 {
3     const int N = 1e5 + 10;
4     /*=====*/
5     int dis[N]; bool vis[N];
6     /*=====*/
7     void Init(int s)
8     {

```

```

9      memset(dis, 0x3F, sizeof(dis));
10     memset(vis, false, sizeof(vis));
11     queue<int>q; dis[s] = 0; q.push(s);
12     while (!q.empty())
13     {
14         int cur = q.front(); q.pop(); vis[cur] = false;
15         for (int i = 0; i < G[cur].size(); ++i)
16         {
17             int val = edge[G[cur][i]].w;
18             int nxt = edge[G[cur][i]].node(cur);
19             if (dis[nxt] > dis[cur] + val)
20             {
21                 dis[nxt] = dis[cur] + val;
22                 if (!vis[nxt])
23                 {
24                     q.push(nxt); vis[nxt] = true;
25                 }
26             }
27         }
28     }
29 }
30 }

```

## Floyd

```

1  namespace _Floyd
2  {
3      const int N = 2e2 + 10;
4      /*=====*/
5      int dp[N][N];
6      /*=====*/
7      void Init(void)
8      {
9          for (int k = 1; k <= n; ++k)
10         {
11             for (int i = 1; i <= n; ++i)
12             {
13                 for (int j = 1; j <= n; ++j)
14                 {
15                     dp[i][j] = min(dp[i][j], dp[i][k] + dp[k][j]);
16                 }
17             }
18         }
19     }
20 }

```

## Dijkstra

```

1  namespace _Dijkstra
2  {
3      const int N = 1e5 + 10;
4      /*=====*/
5      struct Unit
6      {

```

```

7     int v, w;
8     Unit(int _v = 0, int _w = 0)
9     {
10         v = _v, w = _w;
11     }
12     friend bool operator<(const Unit& a, const Unit& b)
13     {
14         return a.w > b.w;
15     }
16 };
17 /*=====*/
18 int dis[N]; bool vis[N];
19 /*=====*/
20 void Init(int s)
21 {
22     memset(dis, 0x3F, sizeof(dis));
23     memset(vis, false, sizeof(vis));
24     priority_queue<Unit>q;
25     q.push(Unit(s, dis[s] = 0));
26     while (!q.empty())
27     {
28         int cur = q.top().v; q.pop();
29         if (vis[cur])continue; vis[cur] = true;
30         for (int i = 0; i < G[cur].size(); ++i)
31         {
32             int val = edge[G[cur][i]].w;
33             int nxt = edge[G[cur][i]].node(cur);
34             if (dis[nxt] > dis[cur] + val)
35             {
36                 dis[nxt] = dis[cur] + val;
37                 q.push(Unit(nxt, dis[nxt]));
38             }
39         }
40     }
41 }
42 }

```

## SPFA-SLF

```

1 namespace _SPFA
2 {
3     const int N = 1e5 + 10;
4     /*=====*/
5     int dis[N]; bool vis[N];
6     /*=====*/
7     void Init(int s)
8     {
9         memset(dis, 0x3F, sizeof(dis));
10        memset(vis, false, sizeof(vis));
11        deque<int>q; dis[s] = 0; q.push_back(s);
12        while (!q.empty())
13        {
14            int cur = q.front();
15            q.pop_front(); vis[cur] = false;
16            if (!q.empty() && dis[q.front()] > dis[q.back()])

```

```

17         {
18             swap(q.front(), q.back());
19         }
20         for (int i = 0; i < G[cur].size(); ++i)
21         {
22             int val = edge[G[cur][i]].w;
23             int nxt = edge[G[cur][i]].node(cur);
24             if (dis[nxt] > dis[cur] + val)
25             {
26                 dis[nxt] = dis[cur] + val;
27                 if (!vis[nxt])
28                 {
29                     vis[nxt] = true;
30                     if (!q.empty() && dis[nxt] < dis[q.front()])
31                     {
32                         q.push_front(nxt);
33                     }
34                     else
35                     {
36                         q.push_back(nxt);
37                     }
38                 }
39             }
40         }
41     }
42 }
43 }

```

## 环相关

### 判环

```

1  namespace _Loop
2  {
3      const int N = 1e5 + 10;
4      /*=====*/
5      int indegree[N];
6      /*=====*/
7      bool Init(void)
8      {
9          queue<int>q; int cnt = 0;
10         memset(indegree, 0, sizeof(indegree));
11         for (int i = 1; i <= m; ++i)
12         {
13             indegree[edge[i].v]++;
14         }
15         for (int i = 1; i <= n; ++i)
16         {
17             if (indegree[i] == 0)q.push(i);
18         }
19         while (!q.empty())
20         {
21             int cur = q.front(); q.pop(); cnt++;
22             for (int i = 0; i < G[cur].size(); ++i)

```

```

23         {
24             int nxt = edge[G[cur][i]].node(cur);
25             if (--indegree[nxt] == 0)q.push(nxt);
26         }
27     }
28     return cnt == n;
29 }
30 }

```

## 判负环

```

1  namespace _Loop
2  {
3      const int N = 1e5 + 10;
4      /*=====*/
5      int dis[N], cnt[N]; bool vis[N];
6      /*=====*/
7      bool Init(void)
8      {
9          queue<int>q;
10         memset(cnt, 0, sizeof(cnt));
11         memset(dis, 0, sizeof(dis));
12         for (int i = 1; i <= n; ++i)
13         {
14             q.push(i); vis[i] = true;
15         }
16         while (!q.empty())
17         {
18             int cur = q.front();
19             q.pop(); vis[cur] = false;
20             for (int i = 0; i < G[cur].size(); ++i)
21             {
22                 int val = edge[G[cur][i]].w;
23                 int nxt = edge[G[cur][i]].node(cur);
24                 if (dis[nxt] > dis[cur] + val)
25                 {
26                     cnt[nxt] = cnt[cur] + 1;
27                     dis[nxt] = dis[cur] + val;
28                     if (cnt[nxt] > n)
29                     {
30                         return true;
31                     }
32                     if (!vis[nxt])
33                     {
34                         q.push(nxt); vis[nxt] = true;
35                     }
36                 }
37             }
38         }
39         return false;
40     }
41 }

```

## 求最小环

```
1 namespace _Loop
2 {
3     int Init(void)
4     {
5         int res = 0x3F3F3F3F;
6         for (int i = 1; i <= m; ++i)
7         {
8             _Dijkstra::Init(edge[i].v, i);
9             res = min(res, dis[edge[i].u] + edge[i].w);
10        }
11        return res;
12    }
13 }
```

## 网络流

### 最大流

#### ISAP

```
1 namespace _ISAP
2 {
3     const int N = 1e5 + 10;
4     const int M = 1e5 + 10;
5     /*=====*/
6     const int INF = 0x7FFFFFFF;
7     /*=====*/
8     struct Edge
9     {
10         int u, v, c, f;
11         Edge(int _u = 0, int _v = 0, int _c = 0, int _f = 0)
12         {
13             u = _u, v = _v, c = _c, f = _f;
14         }
15     };
16     /*=====*/
17     int pre[N]; // 路径前驱
18     int cur[N]; // 当前弧优化
19     int n, m, s, t; // 点, 边, 源, 汇
20     vector<int> G[N]; // 邻接表
21     int d[N], vis[N], num[N]; // 图分层
22     Edge edge[2 * M]; int cnt; // 边
23     /*=====*/
24     void AddEdge(int u, int v, int c)
25     {
26         edge[cnt++] = Edge(u, v, c, 0);
27         edge[cnt++] = Edge(v, u, 0, 0);
28         G[u].push_back(cnt - 2);
29         G[v].push_back(cnt - 1);
30     }
31     /*=====*/
32     void BFS(void)
```

```

33     {
34         for (int i = 0; i <= n; ++i)
35         {
36             d[i] = vis[i] = num[i] = 0;
37         }
38         queue<int>q; q.push(t);
39         d[t] = 0; vis[t] = 1;
40         while (!q.empty())
41         {
42             int x = q.front(); q.pop();
43             for (int i = 0; i < G[x].size(); ++i)
44             {
45                 Edge& e = edge[G[x][i]];
46                 if (!vis[e.u] && e.c > e.f)
47                 {
48                     vis[e.u] = 1; d[e.u] = d[x] + 1; q.push(e.u);
49                 }
50             }
51         }
52         for (int i = 0; i < n; ++i) num[d[i]]++;
53     }
54     int Augument(void)
55     {
56         int x, k = INF;
57         x = t; while (x != s)
58         {
59             Edge& e = edge[pre[x]];
60             k = min(k, e.c - e.f);
61             x = edge[pre[x]].u;
62         }
63         x = t; while (x != s)
64         {
65             edge[pre[x]].f += k;
66             edge[pre[x] ^ 1].f -= k;
67             x = edge[pre[x]].u;
68         }
69         return k;
70     }
71     int MaxFlow(void)
72     {
73         for (int i = 1; i <= n; ++i)
74         {
75             pre[i] = cur[i] = 0;
76         }
77
78         BFS(); int x = s, flow = 0;
79
80         while (d[s] < n)
81         {
82             if (x == t)
83             {
84                 flow += Augument(); x = s;
85             }
86             int flag = 0;
87             for (int& i = cur[x]; i < G[x].size(); ++i)

```

```

88         {
89             Edge& e = edge[G[x][i]];
90             if (e.c > e.f && d[x] == d[e.v] + 1)
91             {
92                 flag = 1; pre[e.v] = G[x][i]; x = e.v; break;
93             }
94         }
95         if (!flag)
96         {
97             int l = n - 1;
98             for (int i = 0; i < G[x].size(); ++i)
99             {
100                 Edge& e = edge[G[x][i]];
101                 if (e.c > e.f) l = min(l, d[e.v]);
102             }
103             if (--num[d[x]] == 0) break;
104             num[d[x] = l + 1]++; cur[x] = 0;
105             if (x != s) x = edge[pre[x]].u;
106         }
107     }
108     return flow;
109 }
110 /*=====*/
111 int Init(void)
112 {
113     cnt = 0;
114     cin >> n >> m >> s >> t;
115     for (int i = 1; i <= n; ++i)
116     {
117         G[i].clear();
118     }
119     for (int i = 1; i <= m; ++i)
120     {
121         int u, v, c;
122         cin >> u >> v >> c;
123         AddEdge(u, v, c);
124     }
125     return MaxFlow();
126 }
127 };

```

## HLPP

```

1  namespace _HLPP
2  {
3      const int N = 1e5 + 10;
4      const int M = 1e5 + 10;
5      /*=====*/
6      const int INF = 0x7FFFFFFF;
7      /*=====*/
8      struct Edge
9      {
10         int next, v, c;
11         Edge(int _next = 0, int _v = 0, int _c = 0)
12         {

```



```

13         next = _next, v = _v, c = _c;
14     }
15 };
16 /*=====*/
17 int n, m, s, t;
18 int d[N], num[N];
19 stack<int> lib[N];
20 int ex[N], level = 0;
21 Edge edge[2 * M]; int head[N], cnt;
22 /*=====*/
23 void AddEdge(int u, int v, int c)
24 {
25     edge[cnt] = Edge(head[u], v, c), head[u] = cnt++;
26     edge[cnt] = Edge(head[v], u, 0), head[v] = cnt++;
27 }
28 /*=====*/
29 int Push(int u)
30 {
31     bool init = u == s;
32     for (int i = head[u]; i != -1; i = edge[i].next)
33     {
34         const int& v = edge[i].v, & c = edge[i].c;
35         if (!c || init == false && d[u] != d[v] + 1) continue;
36         int k = init ? c : min(c, ex[u]);
37         if (v != s && v != t && !ex[v]) lib[d[v]].push(v), level =
max(level, d[v]);
38         ex[u] -= k, ex[v] += k, edge[i].c -= k, edge[i ^ 1].c += k;
39         if (!ex[u]) return 0;
40     }
41     return 1;
42 }
43 void Relabel(int x)
44 {
45     d[x] = INF;
46     for (int i = head[x]; i != -1; i = edge[i].next)
47     {
48         if (edge[i].c) d[x] = min(d[x], d[edge[i].v]);
49     }
50     if (++d[x] < n)
51     {
52         lib[d[x]].push(x); level = max(level, d[x]); ++num[d[x]];
53     }
54 }
55 bool BFS(void)
56 {
57     for (int i = 1; i <= n; ++i)
58     {
59         d[i] = INF; num[i] = 0;
60     }
61     queue<int> q; q.push(t), d[t] = 0;
62     while (!q.empty())
63     {
64         int u = q.front(); q.pop(); num[d[u]]++;
65         for (int i = head[u]; i != -1; i = edge[i].next)
66         {

```

```

67         const int& v = edge[i].v;
68         if (edge[i ^ 1].c && d[v] > d[u] + 1) d[v] = d[u] + 1,
q.push(v);
69     }
70 }
71 return d[s] != INF;
72 }
73 int Select(void)
74 {
75     while (lib[level].size() == 0 && level > -1) level--;
76     return level == -1 ? 0 : lib[level].top();
77 }
78 int MaxFlow(void)
79 {
80     if (!BFS()) return 0;
81     d[s] = n; Push(s); int x;
82     while (x = Select())
83     {
84         lib[level].pop();
85         if (Push(x))
86         {
87             if (!--num[d[x]])
88             {
89                 for (int i = 1; i <= n; ++i)
90                 {
91                     if (i != s && i != t && d[i] > d[x] && d[i] < n +
1)
92                     {
93                         d[i] = n + 1;
94                     }
95                 }
96             }
97             Relabel(x);
98         }
99     }
100     return ex[t];
101 }
102 /*=====*/
103 int Init(void)
104 {
105     cnt = 0;
106     cin >> n >> m >> s >> t;
107     memset(head, -1, sizeof(head));
108     for (int i = 1; i <= m; ++i)
109     {
110         int u, v, c;
111         cin >> u >> v >> c;
112         AddEdge(u, v, c);
113     }
114     return MaxFlow();
115 }
116 }

```

```

1  /*
2  * 使用方法
3  * 1.创建对象
4  * 2.调用AddVertex()创建点
5  * 3.调用Init()初始化大小
6  * 4.调用AddEdge()创建边
7  * 5.调用MaxFlow()获取最大流
8  */
9  class C_Dinic
10 {
11 public:
12     static const int INF = 0x7FFFFFFF;
13     /*=====*/
14     struct Edge
15     {
16         int u, v, c, f;
17         Edge(int _u = 0, int _v = 0, int _c = 0, int _f = 0)
18         {
19             u = _u, v = _v, c = _c, f = _f;
20         }
21     };
22     /*=====*/
23     int n, s, t;
24     /*=====*/
25     vector<Edge>edge;
26     vector<vector<int>>>G;
27     /*=====*/
28     C_Dinic(void)
29     {
30         n = 2, s = 1, t = 2;
31     }
32 private:
33     vector<int>cur;//当前弧优化
34     vector<int>d, vis;//图分层
35     /*=====*/
36     bool BFS(void)
37     {
38         fill(d.begin(), d.end(), 0);
39         fill(vis.begin(), vis.end(), 0);
40         d[s] = 0; vis[s] = 1;
41         queue<int>q; q.push(s);
42         while (!q.empty())
43         {
44             int x = q.front(); q.pop();
45             for (int i = 0; i < G[x].size(); ++i)
46             {
47                 Edge& e = edge[G[x][i]];
48                 if (!vis[e.v] && e.c > e.f)
49                 {
50                     vis[e.v] = 1; d[e.v] = d[x] + 1; q.push(e.v);
51                 }
52             }
53         }

```

```

54         return vis[t];
55     }
56     int DFS(int x, int k)
57     {
58         int flow = 0, f;
59         if (x == t || k == 0) return k;
60         for (int& i = cur[x]; i < G[x].size(); ++i)
61         {
62             Edge& e = edge[G[x][i]];
63             if (d[x] + 1 == d[e.v] && (f = DFS(e.v, min(k, e.c - e.f))) >
0)
64             {
65                 e.f += f; edge[G[x][i] ^ 1].f -= f;
66                 flow += f; k -= f; if (k == 0) break;
67             }
68         }
69         return flow;
70     }
71 public:
72     int S(void) { return s; }
73     int T(void) { return t; }
74     /*=====*/
75     int AddVertex(void)
76     {
77         return ++n;
78     }
79     int AddEdge(int u, int v, int c)
80     {
81         edge.push_back(Edge(u, v, c, 0));
82         edge.push_back(Edge(v, u, 0, 0));
83         G[u].push_back(edge.size() - 2);
84         G[v].push_back(edge.size() - 1);
85         return edge.size() - 2;
86     }
87     /*=====*/
88     void Init(void)
89     {
90         d.assign(n + 1, int());
91         vis.assign(n + 1, int());
92         cur.assign(n + 1, int());
93         G.assign(n + 1, vector<int>());
94     }
95     /*=====*/
96     int MaxFlow(void)
97     {
98         int flow = 0;
99         while (BFS())
100         {
101             flow += DFS(s, INF);
102             fill(cur.begin(), cur.end(), 0);
103         }
104         return flow;
105     }
106 };

```

## Dinic最后反悔

```
1  class C_MaxFlow
2  {
3  public:
4      static const int INF = 0x7FFFFFFF;
5      /*=====*/
6      struct Edge
7      {
8          int u, v, c, f;
9          Edge(int _u = 0, int _v = 0, int _c = 0, int _f = 0)
10         {
11             u = _u, v = _v, c = _c, f = _f;
12         }
13     };
14     /*=====*/
15     int n = 2, s = 1, t = 2;
16     /*=====*/
17     vector<Edge>edge;
18     vector<vector<int>>G1;
19     vector<vector<int>>G2;
20 private:
21     vector<int>cur; //当前弧优化
22     vector<int>d, vis; //图分层
23     /*=====*/
24     bool BFS(void)
25     {
26         fill(d.begin(), d.end(), 0);
27         fill(vis.begin(), vis.end(), 0);
28         d[s] = 0; vis[s] = 1;
29         queue<int>q; q.push(s);
30         while (!q.empty())
31         {
32             int x = q.front(); q.pop();
33             for (int i = 0; i < G1[x].size(); ++i)
34             {
35                 Edge& e = edge[G1[x][i]];
36                 if (!vis[e.v] && e.c > e.f)
37                 {
38                     vis[e.v] = 1; d[e.v] = d[x] + 1; q.push(e.v);
39                 }
40             }
41         }
42         return vis[t];
43     }
44     int DFS(int x, int k)
45     {
46         int flow = 0, f;
47         if (x == t || k == 0) return k;
48         for (int& i = cur[x]; i < G1[x].size(); ++i)
49         {
50             Edge& e = edge[G1[x][i]];
51             if (d[x] + 1 == d[e.v] && (f = DFS(e.v, min(k, e.c - e.f))) >
0)
52                 {
```

```

53         e.f += f; edge[G1[x][i] ^ 1].f -= f;
54         flow += f; k -= f; if (k == 0) break;
55     }
56 }
57 return flow;
58 }
59 public:
60     int S(void) { return s; }
61     int T(void) { return t; }
62     /*=====*/
63     int AddVertex(void)
64     {
65         return ++n;
66     }
67     int AddEdge(int u, int v, int c)
68     {
69         edge.push_back(Edge(u, v, c, 0));
70         edge.push_back(Edge(v, u, 0, 0));
71         G1[u].push_back(edge.size() - 2);
72         G2[v].push_back(edge.size() - 1);
73         return edge.size() - 2;
74     }
75     /*=====*/
76     void Init(void)
77     {
78         d.assign(n + 1, int());
79         vis.assign(n + 1, int());
80         cur.assign(n + 1, int());
81         G1.assign(n + 1, vector<int>());
82         G2.assign(n + 1, vector<int>());
83     }
84     /*=====*/
85     int MaxFlow(void)
86     {
87         int flow = 0;
88         while (BFS())
89         {
90             flow += DFS(s, INF);
91             fill(cur.begin(), cur.end(), 0);
92         }
93         for (int i = 1; i <= n; ++i)
94         {
95             for (auto idx : G2[i])
96             {
97                 G1[i].push_back(idx);
98             }
99         }
100         while (BFS())
101         {
102             flow += DFS(s, INF);
103             fill(cur.begin(), cur.end(), 0);
104         }
105         return flow;
106     }
107 };

```

## Dinic-Scaling

```
1 namespace _Dinic
2 {
3     const int N = 1e5 + 10;
4     const int M = 1e5 + 10;
5     /*=====*/
6     const int INF = 0x7FFFFFFF;
7     /*=====*/
8     struct Edge
9     {
10         int u, v, c, f;
11         Edge(int _u = 0, int _v = 0, int _c = 0, int _f = 0)
12         {
13             u = _u, v = _v, c = _c, f = _f;
14         }
15         friend bool operator<(const Edge& a, const Edge& b)
16         {
17             return a.c > b.c;
18         }
19     };
20     /*=====*/
21     int d[N]; //图分层
22     int cur[N]; //当前弧优化
23     Edge _edge[M]; //即将加入流网络的边
24     int n, m, s, t; //点, 边, 源, 汇
25     vector<int> G[N]; //邻接表
26     Edge edge[2 * M]; int cnt; //边
27     /*=====*/
28     void AddEdge(int u, int v, int c)
29     {
30         edge[cnt++] = Edge(u, v, c, 0);
31         edge[cnt++] = Edge(v, u, 0, 0);
32         G[u].push_back(cnt - 2);
33     }
34     /*=====*/
35     bool BFS(void)
36     {
37         for (int i = 0; i <= n; ++i)
38         {
39             d[i] = INF;
40         }
41         queue<int> q; q.push(s); d[s] = 0;
42         while (!q.empty())
43         {
44             int x = q.front(); q.pop();
45             for (int i = 0; i < G[x].size(); ++i)
46             {
47                 Edge& e = edge[G[x][i]];
48                 if (d[e.v] >= INF && e.c > e.f)
49                 {
50                     d[e.v] = d[x] + 1; q.push(e.v);
51                 }
52             }
53         }
```

```

54         return d[t] < INF;
55     }
56     int DFS(int x, int k)
57     {
58         int flow = 0, f;
59         if (x == t || k == 0) return k;
60         for (int& i = cur[x]; i < G[x].size(); ++i)
61         {
62             Edge& e = edge[G[x][i]];
63             if (d[x] + 1 == d[e.v] && (f = DFS(e.v, min(k, e.c - e.f))) >
0)
64             {
65                 e.f += f; edge[G[x][i] ^ 1].f -= f;
66                 flow += f; k -= f; if (k == 0) break;
67             }
68         }
69         return flow;
70     }
71     int Dinic(void)
72     {
73         int flow = 0;
74         while (BFS())
75         {
76             flow += DFS(s, INF);
77             for (int i = 1; i <= n; ++i)
78             {
79                 cur[i] = 0;
80             }
81         }
82         return flow;
83     }
84     int MaxFlow(void)
85     {
86         int flow = 0;
87         sort(_edge, _edge + m);
88         for (int type : {0, 1})
89         {
90             for (int p = 1 << 30, i = 0; p; p /= 2)
91             {
92                 while (i < m && _edge[i].c >= p)
93                 {
94                     if (type == 0) AddEdge(_edge[i].u, _edge[i].v,
_edge[i].c);
95                     if (type == 1) G[_edge[i].v].push_back(i * 2 + 1); i++;
96                 }
97                 flow += Dinic();
98             }
99         }
100         return flow;
101     }
102     /*=====*/
103     int Init(void)
104     {
105         cnt = 0;
106         cin >> n >> m >> s >> t;

```



```

107     for (int i = 1; i <= n; ++i)
108     {
109         G[i].clear();
110     }
111     for (int i = 0; i < m; ++i)
112     {
113         int u, v, c;
114         cin >> u >> v >> c;
115         _edge[i] = Edge(u, v, c);
116     }
117     return MaxFlow();
118 }
119 }

```

## 费用流

### EK

```

1  namespace _EK
2  {
3      const int N = 1e5 + 10;
4      const int M = 1e5 + 10;
5      /*=====*/
6      const int INF = 0x3F3F3F3F;
7      /*=====*/
8      struct Edge
9      {
10         int next, v, c, w;
11         Edge(int _next = 0, int _v = 0, int _c = 0, int _w = 0)
12         {
13             next = _next, v = _v, c = _c, w = _w;
14         }
15     };
16     /*=====*/
17     int n, m, s, t;
18     int maxflow, mincost;
19     Edge edge[2 * M]; int head[N], cnt;
20     int dis[N], pre[N], incf[N]; bool vis[N];
21     /*=====*/
22     void AddEdge(int u, int v, int c, int w)
23     {
24         edge[cnt] = Edge(head[u], v, c, +w); head[u] = cnt++;
25         edge[cnt] = Edge(head[v], u, 0, -w); head[v] = cnt++;
26     }
27     /*=====*/
28     bool SPFA(void)
29     {
30         memset(dis, 0x3F, sizeof(dis));
31         queue<int> q; q.push(s);
32         dis[s] = 0, incf[s] = INF, incf[t] = 0;
33         while (!q.empty())
34         {
35             int u = q.front(); q.pop(); vis[u] = false;
36             for (int i = head[u]; i != -1; i = edge[i].next)
37             {

```

```

38         int v = edge[i].v, c = edge[i].c, w = edge[i].w;
39         if (!c || dis[v] <= dis[u] + w) continue;
40         dis[v] = dis[u] + w, incf[v] = min(c, incf[u]), pre[v] = i;
41         if (!vis[v])q.push(v), vis[v] = true;
42     }
43 }
44 return incf[t];
45 }
46 int MinCost(void)
47 {
48     while (SPFA())
49     {
50         maxflow += incf[t];
51         for (int u = t; u != s; u = edge[pre[u] ^ 1].v)
52         {
53             edge[pre[u]].c -= incf[t];
54             edge[pre[u] ^ 1].c += incf[t];
55             mincost += incf[t] * edge[pre[u]].w;
56         }
57     }
58     return mincost;
59 }
60 /*=====*/
61 int Init(void)
62 {
63     cin >> n >> m >> s >> t;
64     mincost = maxflow = cnt = 0;
65     memset(head, -1, sizeof(head));
66     for (int i = 1; i <= m; ++i)
67     {
68         int u, v, c, w;
69         cin >> u >> v >> c >> w;
70         AddEdge(u, v, c, w);
71     }
72     return MinCost();
73 }
74 }

```

## ZKW费用流

```

1  /*
2  * 使用方法
3  * 1.创建对象
4  * 2.调用AddVertex()创建点
5  * 3.调用Init()初始化大小
6  * 4.调用AddEdge()创建边
7  * 5.调用MinCost()获取最小费用
8  */
9  class Min_Cost
10 {
11 public:
12     static const int INF = 0x7FFFFFFF;
13     /*=====*/
14     struct Edge
15     {

```

```

16     int u, v, c, w;
17     Edge(int _u = 0, int _v = 0, int _c = 0, int _w = 0)
18     {
19         u = _u, v = _v, c = _c, w = _w;
20     }
21 };
22 /*=====*/
23 int n = 2, s = 1, t = 2;
24 /*=====*/
25 vector<Edge>edge;
26 vector<vector<int>>G;
27 int mincost, maxflow;
28 private:
29     vector<int>dis;
30     vector<bool>vis;
31     /*=====*/
32     bool SPFA(void)
33     {
34         fill(vis.begin(), vis.end(), false);
35         fill(dis.begin(), dis.end(), INF);
36         vis[t] = true, dis[t] = 0;
37         deque<int> q; q.push_back(t);
38         while (!q.empty())
39         {
40             int x = q.front(); q.pop_front(), vis[x] = false;
41             if (!q.empty() && dis[q.front()] > dis[q.back()])
42             {
43                 swap(q.front(), q.back());
44             }
45             for (int i = 0; i < G[x].size(); ++i)
46             {
47                 Edge& e1 = edge[G[x][i] ^ 0];
48                 Edge& e2 = edge[G[x][i] ^ 1];
49                 if (e2.c != 0 && dis[e1.v] > dis[x] - e1.w)
50                 {
51                     dis[e1.v] = dis[x] - e1.w;
52                     if (!vis[e1.v])
53                     {
54                         vis[e1.v] = true;
55                         if (!q.empty() && dis[e1.v] < dis[q.front()])
56                         {
57                             q.push_front(e1.v);
58                         }
59                         else
60                         {
61                             q.push_back(e1.v);
62                         }
63                     }
64                 }
65             }
66         }
67         return dis[s] < INF;
68     }
69     int DFS(int x, int k)
70     {

```

```

71     vis[x] = true; int flow = 0, f;
72     if (x == t || k == 0) return k;
73     for (int i = 0; i < G[x].size(); ++i)
74     {
75         Edge& e1 = edge[G[x][i] ^ 0];
76         Edge& e2 = edge[G[x][i] ^ 1];
77         if (vis[e1.v] || e1.c == 0) continue;
78         if (dis[x] - e1.w == dis[e1.v] && (f = DFS(e1.v, min(k, e1.c)))
> 0)
79         {
80             e1.c -= f, e2.c += f; flow += f, k -= f;
81             mincost += f * e1.w; if (k == 0) break;
82         }
83     }
84     return flow;
85 }
86 public:
87     /*=====*/
88     int S(void) { return s; }
89     int T(void) { return t; }
90     /*=====*/
91     int AddVertex(void)
92     {
93         return ++n;
94     }
95     int AddEdge(int u, int v, int c, int w)
96     {
97         edge.push_back(Edge(u, v, c, +w));
98         edge.push_back(Edge(v, u, 0, -w));
99         G[u].push_back(edge.size() - 2);
100        G[v].push_back(edge.size() - 1);
101        return edge.size() - 2;
102    }
103    /*=====*/
104    void Init(void)
105    {
106        dis.assign(n + 1, int());
107        vis.assign(n + 1, int());
108        G.assign(n + 1, vector<int>());
109    }
110    /*=====*/
111    int MinCost(void)
112    {
113        maxflow = mincost = 0;
114        while (SPFA())
115        {
116            vis[t] = true;
117            while (vis[t])
118            {
119                fill(vis.begin(), vis.end(), false);
120                maxflow += DFS(s, INF);
121            }
122        }
123        return mincost;
124    }

```

## 支配树

```

1 namespace Lengauer_Tarjan
2 {
3     struct Edge
4     {
5         int v, x;
6         Edge(int _v = 0, int _x = 0)
7         {
8             v = _v, x = _x;
9         }
10    };
11    /*=====*/
12    int n, m;
13    Edge edge[M * 3]; int head[3][N], tot;
14    int idx[N], dfn[N], dfc;
15    int fa[N], fth[N], mn[N], idm[N], sdm[N];
16    /*=====*/
17    void Add(int x, int u, int v)
18    {
19        edge[head[x][u] = ++tot] = Edge(v, head[x][u]);
20    }
21    void Add(int u, int v)
22    {
23        Add(0, u, v); Add(1, v, u);
24    }
25    void DFS(int u)
26    {
27        idx[dfn[u] = ++dfc] = u;
28        for (int i = head[0][u]; i; i = edge[i].x)
29        {
30            int v = edge[i].v;
31            if (!dfn[v])
32            {
33                DFS(v), fth[v] = u;
34            }
35        }
36    }
37    int Find(int x)
38    {
39        if (fa[x] == x)
40        {
41            return x;
42        }
43        int tmp = fa[x];
44        fa[x] = Find(fa[x]);
45        if (dfn[sdm[mn[tmp]]] < dfn[sdm[mn[x]]])
46        {
47            mn[x] = mn[tmp];
48        }
49        return fa[x];
50    }

```

```

51 void Tarjan(int st)
52 {
53     DFS(st);
54     for (int i = 1; i <= n; ++i)
55     {
56         fa[i] = sdm[i] = mn[i] = i;
57     }
58     for (int i = dfc; i >= 2; --i)
59     {
60         int u = idx[i], res = INF;
61         for (int j = head[1][u]; j; j = edge[j].x)
62         {
63             int v = edge[j].v; Find(v);
64             if (dfn[v] < dfn[u])
65             {
66                 res = min(res, dfn[v]);
67             }
68             else
69             {
70                 res = min(res, dfn[sdm[mn[v]]]);
71             }
72         }
73         sdm[u] = idx[res];
74         fa[u] = fth[u];
75         Add(2, sdm[u], u);
76         u = fth[u];
77         for (int j = head[2][u]; j; j = edge[j].x)
78         {
79             int v = edge[j].v; Find(v);
80             if (sdm[mn[v]] == u)
81             {
82                 idm[v] = u;
83             }
84             else
85             {
86                 idm[v] = mn[v];
87             }
88         }
89         head[2][u] = 0;
90     }
91     for (int i = 2; i <= dfc; ++i)
92     {
93         int u = idx[i];
94         if (idm[u] != sdm[u])
95         {
96             idm[u] = idm[idm[u]];
97         }
98     }
99 }
100 /*=====*/
101 void Init(int s)
102 {
103     Tarjan(s);
104     tot = dfc = 0;
105     for (int i = 1; i <= n; ++i)

```

```

106         {
107             dfn[i] = head[0][i] = head[1][i] = head[2][i] = 0;
108         }
109     }
110     //树上连边idm[i] -> i;
111 }

```

## 拓扑排序

```

1 void TopSort(C_Graph& G)
2 {
3     vector<int> indegree(G.n + 1, 0); queue<int> q;
4     for (const auto& edge : G.edge) indegree[edge.v]++;
5     for (int i = 1; i <= G.n; ++i) if (indegree[i] == 0) q.push(i);
6     while (!q.empty())
7     {
8         int cur = q.front(); q.pop(); cout << cur << " ";
9         for (int i = 0; i < G[cur].size(); ++i)
10            {
11                int nxt = G[G[cur][i]].node(cur);
12                if (--indegree[nxt] == 0) q.push(nxt);
13            }
14    }
15 }

```

## 差分约束

```

1 namespace _SDC
2 {
3     /*
4     存在负环时无解
5     记得建立一个超级源点
6     A-B<=W的不等式，由B->A，边权为W
7     跑最短路时为最大差值，跑最长路时为最小差值
8     */
9     const int N = 1e5 + 10;
10    /*=====*/
11    const int INF = 0x3f3f3f3f;
12    /*=====*/
13    int dis[N]; int cnt[N]; bool vis[N];
14    /*=====*/
15    bool Init(void)
16    {
17        G[0].clear(); cnt[0] = 0;
18        for (int i = 1; i <= n; ++i)
19            {
20                G[0].push_back(++m);
21                edge[m] = Edge(0, i, 0);
22                dis[i] = INF, vis[i] = false, cnt[i] = 0;
23            }
24        queue<int> q; dis[0] = 0; q.push(0);
25        while (!q.empty())
26            {

```

```

27         int cur = q.front(); q.pop(); vis[cur] = false;
28         for (int i = 0; i < G[cur].size(); ++i)
29         {
30             int val = edge[G[cur][i]].w;
31             int nxt = edge[G[cur][i]].node(cur);
32             if (dis[nxt] > dis[cur] + val)
33             {
34                 cnt[nxt] = cnt[cur] + 1;
35                 dis[nxt] = dis[cur] + val;
36                 if (cnt[nxt] > n) return false;
37                 if (!vis[nxt])
38                 {
39                     q.push(nxt); vis[nxt] = true;
40                 }
41             }
42         }
43     }
44     return true;
45 }
46 }

```

## 图的连通性

### 双连通分量

#### 边双连通分量

```

1  namespace _E_DCC
2  {
3      const int N = 1e5 + 10;
4      /*=====*/
5      int belong[N], cnt;
6      int dfn[N], low[N], num;
7      /*=====*/
8      void Tarjan(int cur, int in_edge)
9      {
10         dfn[cur] = low[cur] = ++num;
11         for (int i = 0; i < G[cur].size(); ++i)
12         {
13             int nxt = edge[G[cur][i]].node(cur);
14             if (!dfn[nxt])
15             {
16                 Tarjan(nxt, G[cur][i]);
17                 low[cur] = min(low[cur], low[nxt]);
18                 if (low[nxt] > dfn[cur])
19                 {
20                     edge[G[cur][i]].bridge = true;
21                 }
22             }
23             else if (i != in_edge)
24             {
25                 low[cur] = min(low[cur], dfn[nxt]);
26             }
27         }

```



```

28     }
29     void DFS(int cur)
30     {
31         belong[cur] = cnt;
32         for (int i = 0; i < G[cur].size(); ++i)
33         {
34             int nxt = edge[G[cur][i]].node(cur);
35             if (edge[G[cur][i]].bridge) continue;
36             if (belong[nxt]) continue; DFS(nxt);
37         }
38     }
39     /*=====*/
40     void Init(void)
41     {
42         for (int i = 1; i <= n; ++i)
43         {
44             if (!dfn[i]) Tarjan(i, 0);
45         }
46         for (int i = 1; i <= n; ++i)
47         {
48             if (!belong[i]) cnt++, DFS(i);
49         }
50     }
51 }

```

## 点双连通分量

```

1  namespace _V_DCC
2  {
3      const int N = 1e5 + 10;
4      /*=====*/
5      vector<int> dcc[N];
6      bool cut[N]; int cnt;
7      stack<int> lib; int root;
8      int dfn[N], low[N], num;
9      /*=====*/
10     void Tarjan(int cur)
11     {
12         int flag = 0; lib.push(cur);
13         dfn[cur] = low[cur] = ++num;
14         if (cur == root && G[cur].size() == 0)
15         {
16             dcc[++cnt].push_back(cur); return;
17         }
18         for (int i = 0; i < G[cur].size(); ++i)
19         {
20             int nxt = edge[G[cur][i]].node(cur);
21             if (!dfn[nxt])
22             {
23                 Tarjan(nxt);
24                 low[cur] = min(low[cur], low[nxt]);
25                 if (low[nxt] >= dfn[cur])
26                 {
27                     flag++; cnt++; int top;
28                     if (cur != root || flag > 1)

```

```

29         {
30             cut[cur] = true;
31         }
32         do
33         {
34             top = lib.top(); lib.pop();
35             dcc[cnt].push_back(top);
36         } while (top != nxt);
37         dcc[cnt].push_back(cur);
38     }
39 }
40 else
41 {
42     low[cur] = min(low[cur], dfn[nxt]);
43 }
44 }
45 }
46 /*=====*/
47 void Init(void)
48 {
49     for (int i = 1; i <= n; ++i)
50     {
51         if (!dfn[i]) Tarjan(root = i);
52     }
53 }
54 }

```

## 获取点双内部的边

```

1 void Tarjan(int cur, int e)
2 {
3     dfn[cur] = low[cur] = ++num;
4     if (cur != root || G[cur].size() != 0)
5     {
6         for (int i = 0; i < G[cur].size(); ++i)
7         {
8             int nxt = edge[G[cur][i]].node(cur);
9             if (!dfn[nxt])
10            {
11                lib.push(G[cur][i]); Tarjan(nxt, G[cur][i]);
12                low[cur] = min(low[cur], low[nxt]);
13                if (low[nxt] >= dfn[cur])
14                {
15                    cnt++; int top;
16                    do
17                    {
18                        top = lib.top(); lib.pop();
19                        dcc[cnt].push_back(edge[top].w);
20                    } while (top != G[cur][i]);
21                }
22            }
23            else
24            {
25                if (dfn[nxt] < dfn[cur] && G[cur][i] != e)
26                {

```

```

27         lib.push(G[cur][i]);
28     }
29     low[cur] = min(low[cur], dfn[nxt]);
30 }
31 }
32 }
33 }

```

## 强连通分量

```

1  namespace _SCC
2  {
3      const int N = 1e5 + 10;
4      /*=====*/
5      int belong[N];
6      int dfn[N], low[N], num;
7      stack<int>lib; int ins[N];
8      vector<int>scc[N]; int cnt;
9      /*=====*/
10     void Tarjan(int cur)
11     {
12         lib.push(cur); ins[cur] = 1;
13         dfn[cur] = low[cur] = ++num;
14         for (int i = 0; i < G[cur].size(); ++i)
15         {
16             int nxt = edge[G[cur][i]].node(cur);
17             if (!dfn[nxt])
18             {
19                 Tarjan(nxt);
20                 low[cur] = min(low[cur], low[nxt]);
21             }
22             else if (ins[nxt])
23             {
24                 low[cur] = min(low[cur], dfn[nxt]);
25             }
26         }
27         if (dfn[cur] == low[cur])
28         {
29             cnt++; int top;
30             do
31             {
32                 top = lib.top(); lib.pop(); ins[top] = 0;
33                 belong[top] = cnt; scc[cnt].push_back(top);
34             } while (top != cur);
35         }
36     }
37     /*=====*/
38     void Init(void)
39     {
40         num = cnt = 0;
41         for (int i = 1; i <= n; ++i)
42         {
43             dfn[i] = low[i] = 0;
44         }
45         for (int i = 1; i <= n; ++i)

```

```

46     {
47         if (!dfn[i])Tarjan(i);
48     }
49 }
50 }

```

## 最小树形图

## 有向有环图

1 | 挖坑：朱刘算法

## 有向无环图

```

1  namespace _DMST
2  {
3      const int N = 1e5 + 10;
4      /*=====*/
5      const int INF = 0x7FFFFFFF;
6      /*=====*/
7      int val[N], sum;
8      /*=====*/
9      void Init(void)
10     {
11         sum = 0;
12         for (int i = 1; i <= n; ++i)
13         {
14             val[i] = INF;
15         }
16         for (int i = 1; i <= m; ++i)
17         {
18             int u = edge[i].u;
19             int v = edge[i].v;
20             int w = edge[i].w;
21             val[v] = min(val[v], w);
22         }
23         for (int i = 1; i <= n; ++i)
24         {
25             if (val[i] != INF)
26             {
27                 sum += val[i];
28             }
29         }
30     }
31 }

```

## 三元环计数

```

1  const int N = 1e5 + 10;
2  const int M = 2e5 + 10;
3  /*=====*/
4  int n, m;

```

```

5 struct Edge
6 {
7     int u, v;
8     Edge(int _u = 0, int _v = 0)
9     {
10         u = _u, v = _v;
11     }
12 };
13 Edge edge[M];
14 int degree[N];
15 vector<int>Out[N];
16 /*=====*/
17 int tag[N];
18 /*=====*/
19 void solve(void)
20 {
21     cin >> n >> m;
22     for (int i = 1; i <= m; ++i)
23     {
24         int u, v;
25         cin >> u >> v;
26         edge[i] = Edge(u, v);
27         degree[u]++, degree[v]++;
28     }
29     for (int i = 1; i <= m; ++i)
30     {
31         int u = edge[i].u, v = edge[i].v;
32         if (degree[u] == degree[v] && u > v) swap(u, v);
33         if (degree[u] != degree[v] && degree[u] > degree[v]) swap(u, v);
34         out[u].push_back(v);
35     }
36     int ans = 0;
37     for (int u = 1; u <= n; ++u)
38     {
39         for (auto v : Out[u])
40         {
41             tag[v] = u;
42         }
43         for (auto v : Out[u])
44         {
45             for (auto w : Out[v])
46             {
47                 if (tag[w] == u)
48                 {
49                     ans++;
50                 }
51             }
52         }
53     }
54     cout << ans << endl;
55 }

```

## 字符串

# Trie

## 01Trie

```
1  template<int BIT = 31>
2  class C_Trie
3  {
4  private:
5      int root, cnt;
6      vector<int>siz;
7      vector<vector<int>>trie;
8  public:
9      void Init(int sigma_n)
10     {
11         sigma_n *= BIT;
12         cnt = -1; root = ++cnt;
13         siz.assign(sigma_n + 1, 0);
14         trie.assign(sigma_n + 1, vector<int>(2, -1));
15     }
16     void Insert(int val)
17     {
18         int cur = root; siz[cur]++;
19         for (int i = BIT - 1; i >= 0; --i)
20         {
21             int bit = (val >> i) & 1;
22             if (trie[cur][bit] == -1)
23             {
24                 trie[cur][bit] = ++cnt;
25             }
26             siz[cur = trie[cur][bit]]++;
27         }
28     }
29     int Query(int val)
30     {
31         int cur = root;
32         for (int i = BIT - 1; i >= 0; --i)
33         {
34             int bit = (val >> i) & 1;
35             if (trie[cur][bit] == -1)
36             {
37                 return 0;
38             }
39             cur = trie[cur][bit];
40         }
41         return siz[cur];
42     }
43     int MaxXOR(int val)
44     {
45         int res = 0, cur = root;
46         for (int i = BIT - 1; i >= 0; --i)
47         {
48             int bit = ((val >> i) & 1) ^ 1;
49             if (trie[cur][bit] == -1) bit ^= 1;
50             res += 1 << i; cur = trie[cur][bit];
```

```

51     }
52     return res;
53 }
54 };

```

## 字符Trie

```

1  template<int ASCII = 128>
2  class C_Trie
3  {
4  private:
5      int root, cnt;
6      vector<int> siz;
7      vector<vector<int>> trie;
8  public:
9      void Init(int sigma_s)
10     {
11         cnt = -1; root = ++cnt;
12         siz.assign(sigma_s + 1, 0);
13         trie.assign(sigma_s + 1, vector<int>(ASCII, -1));
14     }
15     void Insert(const string& str)
16     {
17         int cur = root; siz[cur]++;
18         for (int i = 0; i < str.size(); ++i)
19         {
20             if (trie[cur][str[i]] == -1)
21             {
22                 trie[cur][str[i]] = ++cnt;
23             }
24             siz[cur = trie[cur][str[i]]]++;
25         }
26     }
27     int Query(const string& str)
28     {
29         int cur = root;
30         for (int i = 0; i < str.size(); ++i)
31         {
32             if (trie[cur][str[i]] == -1)
33             {
34                 return 0;
35             }
36             cur = trie[cur][str[i]];
37         }
38         return siz[cur];
39     }
40 };

```

## Hash

## 单Hash

```
1 class C_Hash
2 {
3 private:
4     static const lnt MOD = 998244353;
5     /*=====*/
6     vector<lnt>powbase, invbase, sumhash;
7 public:
8     void Init(const string& str, lnt base = 233)
9     {
10         powbase.assign(str.size(), 0);
11         invbase.assign(str.size(), 0);
12         sumhash.assign(str.size(), 0);
13         /*=====*/
14         for (int i = 0; i < str.size(); ++i)
15         {
16             if (i == 0)powbase[i] = 1;
17             else powbase[i] = powbase[i - 1] * base % MOD;
18         }
19         base = Pow(base % MOD, MOD - 2, MOD);
20         for (int i = 0; i < str.size(); ++i)
21         {
22             if (i == 0)invbase[i] = 1;
23             else invbase[i] = invbase[i - 1] * base % MOD;
24         }
25         /*=====*/
26         for (int i = 0; i < str.size(); ++i)
27         {
28             if (i == 0)sumhash[i] = str[i] * powbase[i] % MOD;
29             else sumhash[i] = (sumhash[i - 1] + str[i] * powbase[i]) % MOD;
30         }
31     }
32     lnt operator()(int l, int r)
33     {
34         return (sumhash[r] - (l > 0 ? sumhash[l - 1] : 0) + MOD) *
35         invbase[l] % MOD;
36     }
37 };
```

## 双Hash

```
1 class C_DoubleHash
2 {
3 private:
4     C_Hash Hash1, Hash2;
5 public:
6     void Init(const string& str, lnt base1 = 233, lnt base2 = 19260817)
7     {
8         Hash1.Init(str, base1), Hash2.Init(str, base2);
9     }
10    pair<lnt, lnt> operator()(int l, int r)
11    {
12        return { Hash1(l, r), Hash2(l, r) };
```



```
13     }
14 };
```

## Split

```
1  class C_Split
2  {
3  public:
4      vector<string> operator()(const string &str, char c)
5      {
6          string temp;
7          vector<string> res;
8          istringstream iss(str);
9          while (getline(iss, temp, c))
10         {
11             if (temp != "")
12             {
13                 res.push_back(temp);
14             }
15         }
16         return res;
17     }
18 };
```

## Z函数

```
1  vector<int> Z_Function(const string& str)
2  {
3      int n = str.size() - 1;
4      vector<int> z(str.size());
5      int l = 1, r = 1; z[1] = n;
6      for (int i = 2; i <= n; ++i)
7      {
8          z[i] = (i <= r ? min(z[i - l + 1], r - i + 1) : 0);
9          while (i + z[i] <= n && str[l + z[i]] == str[i + z[i]]) z[i]++;
10         if (i + z[i] - 1 > r) r = i + z[i] - 1, l = i;
11     }
12     return z;
13 }
```