

# NJUPT XCPC Templates

Yunhai Bian

2020 年 12 月 10 日

## 目录

<b>1</b>	<b>数据结构</b>	<b>2</b>	<b>2.3</b>	<b>次小生成树</b>	<b>29</b>
1.1	树状数组	2	2.4	有向图的强连通分量	29
1.2	线段树	3	2.5	无向图的双连通分量	29
1.2.1	HDU1540	3	2.6	最近公共祖先	29
1.2.2	HDU4578	5	2.7	2-SAT	29
1.2.3	HDU4553	9	2.8	网络流	29
1.2.4	HDU1542	12	2.8.1	Dinic	29
1.2.5	HDU1255	14	2.8.2	EK	31
1.3	ST 表	17	2.9	二分图	32
1.4	Splay	17	<b>3</b>	<b>字符串</b>	<b>32</b>
1.5	Treap	17	3.1	Manacher	32
1.6	树链剖分	21	3.2	KMP	33
1.6.1	点权	21	3.3	AC 自动机	34
1.6.2	边权	25	3.4	后缀数组	34
<b>2</b>	<b>图论</b>	<b>29</b>	<b>4</b>	<b>其他</b>	<b>34</b>
2.1	最短路	29	4.1	高精度	34
2.2	最小生成树	29	4.2	莫队	39

# 1 数据结构

## 1.1 树状数组

```
1 // 注意树状数组不能处理下标0开始
2 // 一维
3 int c[N];
4
5 inline int lowbit(int x) {
6     return x & -x;
7 }
8
9 int add(int x, int y) {
10     for (int i = x; i <= n; i += lowbit(i)) c[i] += y;
11 }
12
13 int sum(int x) {
14     int res = 0;
15     for (int i = x; i; i -= lowbit(i)) res += c[i];
16     return res;
17 }
18
19 // 二维
20 LL c[N][N];
21
22 inline int lowbit(int x) {
23     return x & -x;
24 }
25
26 void add(int x, int y, LL v) {
27     for (int i = x; i <= n; i += lowbit(i))
28         for (int j = y; j <= m; j += lowbit(j))
29             c[i][j] += v;
30 }
31
32 LL query(int x, int y) {
33     LL ans = 0;
34     for (int i = x; i; i -= lowbit(i))
35         for (int j = y; j; j -= lowbit(j))
36             ans += c[i][j];
37     return ans;
38 }
```

## 1.2 线段树

### 1.2.1 HDU1540

题意：有  $n$  个点连成一条线，编号从左至右为  $1 \sim n$ ，有三种操作：1. 摧毁一个点 2. 查询某个点能到的所有点数（包括自己）3. 重建上一次被摧毁的点。

分析：用一个栈  $stk$  来存放被摧毁的点，摧毁点  $x$  就  $stk[++top] = x$ ，重建上一个点就只需要取出栈顶  $x = stk[top--]$ 。线段树每个节点维护区间左侧连续最大长度（点数） $lmax$  以及右侧最大连续长度  $rmax$ 。摧毁一个点就是在线段树中找到该点并将其  $lmax=rmax=0$ ，重建就是  $lmax=rmax=1$ ，然后  $pushup$  上去。难点在于 2 号查询操作，如果点  $x$  在当前结点的左孩子，分两种情况来看，如果点  $x$  被左孩子的右侧最大连续区间包含了，那么  $x$  能到达的所有点数就是左孩子的  $rmax$  + 右孩子的  $lmax$ ，否则递归直接递归左孩子即可。剩余情况类似。

```
1  #include <iostream>
2
3  using namespace std;
4
5  const int N = 50010;
6
7  int n, m;
8  struct Tree {
9      int l, r;
10     int lmax, rmax;
11 } tr[N << 2];
12 int stk[N], top;
13
14 void pushup(Tree &root, Tree &left, Tree &right) {
15     root.lmax = left.lmax, root.rmax = right.rmax;
16     if (left.r - left.l + 1 == left.lmax) root.lmax += right.lmax;
17     if (right.r - right.l + 1 == right.rmax) root.rmax += left.rmax;
18 }
19
20 void pushup(int u) {
21     pushup(tr[u], tr[u << 1], tr[u << 1 | 1]);
22 }
23
24 void build(int u, int l, int r) {
25     if (l == r) {
26         tr[u] = {l, r, 1, 1};
27     } else {
```

```
28     tr[u] = {l, r};
29     int mid = l + r >> 1;
30     build(u << 1, l, mid); build(u << 1 | 1, mid + 1, r);
31     pushup(u);
32 }
33 }
34
35 void modify(int u, int x, int y) {
36     if (tr[u].l == x && tr[u].r == x) {
37         tr[u].lmax = tr[u].rmax = y;
38     } else {
39         int mid = tr[u].l + tr[u].r >> 1;
40         if (x <= mid) modify(u << 1, x, y);
41         else modify(u << 1 | 1, x, y);
42         pushup(u);
43     }
44 }
45
46 int query(int u, int x) {
47     if (tr[u].l == x && tr[u].r == x) {
48         return tr[u].lmax;
49     } else {
50         int mid = tr[u].l + tr[u].r >> 1;
51         if (x <= mid) {
52             if (tr[u << 1].r - tr[u << 1].rmax + 1 <= x) {
53                 return tr[u << 1].rmax + tr[u << 1 | 1].lmax;
54             } else {
55                 return query(u << 1, x);
56             }
57         } else {
58             if (tr[u << 1 | 1].l + tr[u << 1 | 1].lmax - 1 >= x) {
59                 return tr[u << 1 | 1].lmax + tr[u << 1].rmax;
60             } else {
61                 return query(u << 1 | 1, x);
62             }
63         }
64     }
65 }
66
67 int main() {
68     while (scanf("%d%d", &n, &m) != EOF) {
69         build(1, 1, n);
70         while (m--) {
```

```

71     char op[2]; int x;
72     scanf("%s", op);
73     if (*op == 'D') {
74         scanf("%d", &x);
75         modify(1, x, 0);
76         stk[++top] = x;
77     } else if (*op == 'R') {
78         int x = stk[top--];
79         modify(1, x, 1);
80     } else {
81         scanf("%d", &x);
82         printf("%d\n", query(1, x));
83     }
84 }
85 }
86 return 0;
87 }

```

### 1.2.2 HDU4578

题意：线段树区间加，区间乘，区间置数，区间和，平方和，立方和。

分析：需要维护，置数标记 same，乘法标记 mul，加法标记 add，区间和标记 s[0~2] 分别表示和，平方和，立方和。

首先确定前三个标记维护优先级，same > mul > add，然后就是三个和的维护需要推导一下。

1. 区间置数，三个和很好维护不说了
2. 区间乘  $k$ ，三个和分别乘以  $k, k^2, k^3$
3. 区间加  $a$ ，初始有  $s[0] = \sum x, s[1] = \sum x^2, s[2] = \sum x^3$ ，区间长度为  $len$

$$\sum (x + a) = \sum x + \sum a = s[0] + len * a \quad (1)$$

$$\sum (x + a)^2 = \sum x^2 + 2a \sum x + \sum a^2 = s[1] + 2a * s[0] + len * a^2 \quad (2)$$

$$\sum (x + a)^3 = \sum x^3 + 3a \sum x^2 + 3a^2 \sum x + \sum a^3 = s[2] + 3a * s[1] + 3a^2 * s[0] + len * a^3 \quad (3)$$

注意维护和的时应该倒序维护（立方和，平方和，和），防止要用的值被先更新了。

```

1 #include <bits/stdc++.h>
2
3 using namespace std;

```

```
4
5 typedef pair<int, int> PII;
6 typedef long long LL;
7
8 const int N = 100010, mod = 10007;
9
10 int n, m;
11 struct Tree {
12     int l, r;
13     LL same, mul, add, s[3];
14 } tr[N << 2];
15
16 void build(int u, int l, int r) {
17     tr[u] = {l, r, 0, 1, 0, 0, 0, 0};
18     if (l == r) return;
19     int mid = l + r >> 1;
20     build(u << 1, l, mid), build(u << 1 | 1, mid + 1, r);
21 }
22
23 void pushup(int u) {
24     for (int i = 0; i < 3; i++) {
25         tr[u].s[i] = (tr[u << 1].s[i] + tr[u << 1 | 1].s[i]) % mod;
26     }
27 }
28
29 void pushdown(int u) {
30     auto &root = tr[u], &left = tr[u << 1], &right = tr[u << 1 | 1];
31     if (root.same) {
32         left.same = right.same = root.same;
33         left.mul = right.mul = 1, left.add = right.add = 0;
34         LL base = 1;
35         for (int i = 0; i < 3; i++) {
36             (base *= root.same) %= mod;
37             left.s[i] = (left.r - left.l + 1) * base % mod;
38             right.s[i] = (right.r - right.l + 1) * base % mod;
39         }
40         root.same = 0;
41     }
42     if (root.mul != 1) {
43         LL k = root.mul;
44         (left.mul *= k) %= mod, (right.mul *= k) %= mod;
45         (left.add *= k) %= mod, (right.add *= k) %= mod;
46         LL base = 1;
```

```

47     for (int i = 0; i < 3; i++) {
48         (base *= k) %= mod;
49         (left.s[i] *= base) %= mod, (right.s[i] *= base) %= mod;
50     }
51     root.mul = 1;
52 }
53 if (root.add) {
54     LL a = root.add;
55     (left.add += a) %= mod, (right.add += a) %= mod;
56     (left.s[2] += 3 * a * left.s[1] + 3 * a * a * left.s[0] + (left.
57         r - left.l + 1) * a * a * a) %= mod;
58     (right.s[2] += 3 * a * right.s[1] + 3 * a * a * right.s[0] + (
59         right.r - right.l + 1) * a * a * a) %= mod;
60     (left.s[1] += 2 * a * left.s[0] + (left.r - left.l + 1) * a * a)
61         %= mod;
62     (right.s[1] += 2 * a * right.s[0] + (right.r - right.l + 1) * a
63         * a) %= mod;
64     (left.s[0] += (left.r - left.l + 1) * a) %= mod, (right.s[0] +=
65         (right.r - right.l + 1) * a) %= mod;
66     root.add = 0;
67 }
68 }
69
70 // [l,r]乘k再加a
71 void modify_mul_add(int u, int l, int r, LL k, LL a) {
72     if (tr[u].l >= l && tr[u].r <= r) {
73         if (k != 1) {
74             (tr[u].mul *= k) %= mod;
75             (tr[u].add *= k) %= mod;
76             LL base = 1;
77             for (int i = 0; i < 3; i++) {
78                 (base *= k) %= mod;
79                 (tr[u].s[i] *= base) %= mod;
80             }
81             if (a) {
82                 (tr[u].add += a) %= mod;
83                 (tr[u].s[2] += 3 * a * tr[u].s[1] + 3 * a * a * tr[u].s[0] +
84                     (tr[u].r - tr[u].l + 1) * a * a * a) %= mod;
85                 (tr[u].s[1] += 2 * a * tr[u].s[0] + (tr[u].r - tr[u].l + 1) *
86                     a * a) %= mod;
87                 (tr[u].s[0] += (tr[u].r - tr[u].l + 1) * a) %= mod;
88             }
89         }
90     }

```

```

83     } else {
84         pushdown(u);
85         int mid = tr[u].l + tr[u].r >> 1;
86         if (l <= mid) modify_mul_add(u << 1, l, r, k, a);
87         if (r > mid) modify_mul_add(u << 1 | 1, l, r, k, a);
88         pushup(u);
89     }
90 }
91
92 void modify_assign(int u, int l, int r, int c) {
93     if (tr[u].l >= l && tr[u].r <= r) {
94         tr[u].same = c, tr[u].mul = 1, tr[u].add = 0;
95         LL base = 1;
96         for (int i = 0; i < 3; i++) {
97             (base *= tr[u].same) %= mod;
98             tr[u].s[i] = (tr[u].r - tr[u].l + 1) * base % mod;
99         }
100     } else {
101         pushdown(u);
102         int mid = tr[u].l + tr[u].r >> 1;
103         if (l <= mid) modify_assign(u << 1, l, r, c);
104         if (r > mid) modify_assign(u << 1 | 1, l, r, c);
105         pushup(u);
106     }
107 }
108
109 LL query(int u, int l, int r, int type) {
110     if (tr[u].l >= l && tr[u].r <= r) {
111         return tr[u].s[type];
112     } else {
113         pushdown(u);
114         LL res = 0;
115         int mid = tr[u].l + tr[u].r >> 1;
116         if (l <= mid) (res += query(u << 1, l, r, type)) %= mod;
117         if (r > mid) (res += query(u << 1 | 1, l, r, type)) %= mod;
118         return res;
119     }
120 }
121
122 int main() {
123     while (cin >> n >> m && n && m) {
124         build(1, 1, n);
125         while (m--) {

```



```

126     int type, x, y, c;
127     scanf("%d%d%d", &type, &x, &y, &c);
128     if (type == 1) {
129         modify_mul_add(1, x, y, 1ll, c);
130     } else if (type == 2) {
131         modify_mul_add(1, x, y, c, 0ll);
132     } else if (type == 3) {
133         modify_assign(1, x, y, c);
134     } else {
135         printf("%lld\n", query(1, x, y, c - 1));
136     }
137 }
138 }
139 return 0;
140 }

```

### 1.2.3 HDU4553

题意：有一个长度为  $n$  的时间轴，有两种操作：1. DS QT 表示屌丝申请第一段长度为 QT 的空闲时间，能申请到就输出起始时间。2. NS QT 表示女神申请第一段长度为 QT 的空闲时间，如果能申请到输出起始时间，如果找不到，可以无视屌丝已经申请的时间，再找到一个第一个连续空闲时间大于等于 QT 的起始位置。3. STUDY!! L R 表示清空这段时间的所有申请用于学习，由于三分钟热度，之后再有人申请到 STUDY 的时间还是会分配出去。

分析：线段树维护两个时间轴的信息，分别表示屌丝时间轴的分配情况，还有女神时间轴的分配情况。详见代码，下标 0 表示屌丝，下标 1 表示女神。same 为区间相同的值的标记，lmax, rmax, tmax 分别表示区间左侧最长连续空闲时间，右侧最长连续空闲时间，区间内最长连续空闲时间，用 1 表示空闲。然后根据题目要求操作即可。

```

1  // 1表示空闲
2  #include <bits/stdc++.h>
3
4  using namespace std;
5
6  typedef pair<int, int> PII;
7  typedef long long LL;
8
9  const int N = 100010;
10
11 int T, Case, n, m;

```

```

12 struct Tree {
13     int l, r;
14     int same[2], lmax[2], rmax[2], tmax[2]; // 下标0维护分配给屌丝的时间,
        下标1维护分配给女神的时间
15 } tr[N << 2];
16
17 void pushup(int u, int type) {
18     auto &root = tr[u], &left = tr[u << 1], &right = tr[u << 1 | 1];
19     root.lmax[type] = left.lmax[type];
20     if (left.lmax[type] == left.r - left.l + 1) root.lmax[type] +=
        right.lmax[type];
21     root.rmax[type] = right.rmax[type];
22     if (right.rmax[type] == right.r - right.l + 1) root.rmax[type] +=
        left.rmax[type];
23     root.tmax[type] = max(max(left.tmax[type], right.tmax[type]), left.
        rmax[type] + right.lmax[type]);
24 }
25
26 void build(int u, int l, int r) {
27     tr[u] = {l, r, -1, -1, 1, 1, 1, 1, 1, 1};
28     if (l == r) return;
29     int mid = l + r >> 1;
30     build(u << 1, l, mid), build(u << 1 | 1, mid + 1, r);
31     pushup(u, 0), pushup(u, 1);
32 }
33
34 void pushdown(int u, int type) {
35     auto &root = tr[u], &left = tr[u << 1], &right = tr[u << 1 | 1];
36     if (root.same[type] != -1) {
37         left.same[type] = right.same[type] = root.same[type];
38         left.lmax[type] = left.rmax[type] = left.tmax[type] = root.same[
            type] * (left.r - left.l + 1);
39         right.lmax[type] = right.rmax[type] = right.tmax[type] = root.
            same[type] * (right.r - right.l + 1);
40         root.same[type] = -1;
41     }
42 }
43
44 void modify(int u, int l, int r, int x, int type) {
45     if (tr[u].l >= l && tr[u].r <= r) {
46         tr[u].same[type] = x;
47         tr[u].lmax[type] = tr[u].rmax[type] = tr[u].tmax[type] = x * (tr
            [u].r - tr[u].l + 1);

```

```

48     } else {
49         pushdown(u, type);
50         int mid = tr[u].l + tr[u].r >> 1;
51         if (l <= mid) modify(u << 1, l, r, x, type);
52         if (r > mid) modify(u << 1 | 1, l, r, x, type);
53         pushup(u, type);
54     }
55 }
56
57 // 找到第一段长度为x的连续空闲区间的左端点
58 int query(int u, int x, int type) {
59     if (tr[u].tmax[type] < x) return -1; // 不存在
60     pushdown(u, type);
61     if (tr[u << 1].tmax[type] >= x) return query(u << 1, x, type);
62     if (tr[u << 1].rmax[type] + tr[u << 1 | 1].lmax[type] >= x) return
        tr[u << 1].r - tr[u << 1].rmax[type] + 1;
63     return query(u << 1 | 1, x, type);
64 }
65
66 int main() {
67     for (cin >> T; T--; ) {
68         printf("Case %d:\n", ++Case);
69         scanf("%d%d", &n, &m);
70         build(1, 1, n);
71         while (m--) {
72             char op[10];
73             int x, y;
74             scanf("%s", op);
75             if (*op == 'N') {
76                 scanf("%d", &x);
77                 int st = query(1, x, 0); // 先在屌丝时间轴查询是否存在长度为x的
                    连续空闲(1)区间
78                 if (st != -1) { // 在屌丝时间轴查到了,同时修改两个时间轴的区间[st
                    ,st+x-1]置为忙碌状态
79                     modify(1, st, st + x - 1, 0, 0);
80                     modify(1, st, st + x - 1, 0, 1);
81                     printf("%d,don't put my gezi\n", st);
82                 } else { // 屌丝时间轴中没有这样的区间
83                     st = query(1, x, 1); // 在女神时间轴中查
84                     if (st != -1) { // 在女神时间轴查到,就同时修改两个时间轴的区间
                        [st,st+x-1]置为忙碌状态
85                         modify(1, st, st + x - 1, 0, 0);
86                         modify(1, st, st + x - 1, 0, 1);

```

```
87         printf("%d,don't put my gezi\n", st);
88     } else { // 没有空闲时间
89         puts("wait for me");
90     }
91 }
92 } else if (*op == 'D') {
93     scanf("%d", &x);
94     int st = query(1, x, 0);
95     if (st != -1) { // 在屌丝时间轴查到了，区间修改为忙碌状态
96         modify(1, st, st + x - 1, 0, 0);
97         printf("%d,let's fly\n", st);
98     } else { // 没有空闲时间
99         puts("fly with yourself");
100     }
101 } else { // 由于是三分钟热度，应该是将区间置为空闲状态
102     scanf("%d%d", &x, &y);
103     modify(1, x, y, 1, 0);
104     modify(1, x, y, 1, 1);
105     puts("I am the hope of chinese chengxuyuan!!");
106 }
107 }
108 }
109 return 0;
110 }
```

#### 1.2.4 HDU1542

题意：线段树扫描线求矩形面积并。

分析：注意线段树的每一个叶子结点表示的不是单个点，而是一个区间，其中的标记含义如注释。

```
1  #include <bits/stdc++.h>
2
3  using namespace std;
4
5  const int N = 200010;
6
7  int n, Case;
8  struct Segment {
9      double x, y1, y2;
10     int k;
11     bool operator < (const Segment &W) const {
```

```

12     return x < W.x;
13 }
14 } seg[N];
15 // 线段树的每一个叶子结点（假设下标为i），表示一个区间[y_i, y_{i+1}]
16 struct Node {
17     int l, r, cnt; // cnt表示[l,r]区间被完全覆盖的次数，cnt>0就表示要算上[l,r]
18     // 这一整段，其表示实际的区间为[ys[l],ys[r+1]]
19     double len; // len表示当前线段树的区间[l,r]内，cnt>0（即被覆盖的实际区间）
20     // 的合并长度之和。
21 } tr[N << 2]; // 比如 y1, y2, y3 离散化后为k_y1,k_y2,k_y3。其区间[k_y1,
22     // k_y2],[k_y1,k_y2],[k_y2,k_y3]是线段树中的3个叶子结点。
23 vector<double> ys;
24
25 int find(double y) {
26     return lower_bound(ys.begin(), ys.end(), y) - ys.begin();
27 }
28
29 void pushup(int u) {
30     if (tr[u].cnt) {
31         tr[u].len = ys[tr[u].r + 1] - ys[tr[u].l];
32     } else if (tr[u].l != tr[u].r) {
33         tr[u].len = tr[u << 1].len + tr[u << 1 | 1].len;
34     } else {
35         tr[u].len = 0;
36     }
37 }
38
39 void build(int u, int l, int r) {
40     tr[u] = {l, r, 0, 0};
41     if (l == r) return;
42     int mid = l + r >> 1;
43     build(u << 1, l, mid), build(u << 1 | 1, mid + 1, r);
44 }
45
46 void modify(int u, int l, int r, int k) {
47     if (tr[u].l >= l && tr[u].r <= r) {
48         tr[u].cnt += k;
49         pushup(u);
50     } else {
51         int mid = tr[u].l + tr[u].r >> 1;
52         if (l <= mid) modify(u << 1, l, r, k);
53         if (r > mid) modify(u << 1 | 1, l, r, k);
54         pushup(u);
55     }
56 }

```

```

52     }
53 }
54
55 int main() {
56     while (cin >> n, n) {
57         ys.clear();
58         for (int i = 0; i < n; i++) {
59             double x1, y1, x2, y2;
60             scanf("%lf%lf%lf%lf", &x1, &y1, &x2, &y2);
61             seg[i * 2] = {x1, y1, y2, 1};
62             seg[i * 2 + 1] = {x2, y1, y2, -1};
63             ys.push_back(y1), ys.push_back(y2);
64         }
65         sort(ys.begin(), ys.end());
66         ys.erase(unique(ys.begin(), ys.end()), ys.end());
67         build(1, 0, ys.size() - 2); // 共ys.size()个y, 那么相邻之间就有ys.
            size()-1个区间, 就有ys.size()-1个线段树的叶子节点。
68         sort(seg, seg + n * 2);
69         double res = 0;
70         for (int i = 0; i < n * 2; i++){
71             if (i) res += tr[1].len * (seg[i].x - seg[i - 1].x);
72             int l = find(seg[i].y1), r = find(seg[i].y2) - 1;
73             // 右端点注意要减去1, 假设实际区间为[L,R], 那么对应线段树中的区间就是[L
                ,R-1]
74             modify(1, l, r, seg[i].k);
75         }
76         printf("Test case #%d\n", ++Case);
77         printf("Total explored area: %.2lf\n\n", res);
78     }
79     return 0;
80 }

```

### 1.2.5 HDU1255

题意：线段树扫描线求至少被覆盖 2 次的矩形面积并。

分析：与上一个题目类似，这里需要分别维护 len1, len2，其中 len1 含义与上一个题的 len 一样，len2 表示线段树区间内被覆盖至少 2 次的实际区间的合并的长度。只要求改 pushup 函数，更新 len2 时候需要分情况讨论，如果区间被完全覆盖了至少 2 次，len2 就是区间长度；否则，如果当前是叶子结点，那么此时最多会被完全覆盖 1 次，对 len2 没有贡献；否则，如果不是叶子结点并且恰好被覆盖 1 次，那么想要求该区间内至少被覆

盖 2 次的长度，就需要计算当前结点的左右子结点中被覆盖至少 1 次的长度，如果不是叶子结点并且没有被完全覆盖过，直接用子结点的 len2 之和来更新当前结点的 len2 即可。有点绕，但是并不难理解。

```

1  #include <bits/stdc++.h>
2
3  using namespace std;
4
5  const int N = 2010;
6
7  int T, n;
8  struct Segment {
9      double x, y1, y2;
10     int k;
11     bool operator < (const Segment &W) const {
12         return x < W.x;
13     }
14 } seg[N];
15 struct Node {
16     int l, r, cnt; // cnt表示[l,r]区间被完全覆盖的次数
17     double len1, len2; // len1表示线段树区间[l,r]中cnt>0的区间合并后的长度,
18                        // len2对应cnt>1
19 } tr[N << 2];
20 vector<double> ys;
21
22 int find(double y) {
23     return lower_bound(ys.begin(), ys.end(), y) - ys.begin();
24 }
25
26 void pushup(int u) {
27     // 更新len1
28     if (tr[u].cnt > 0) {
29         tr[u].len1 = ys[tr[u].r + 1] - ys[tr[u].l];
30     } else if (tr[u].l == tr[u].r) {
31         tr[u].len1 = 0;
32     } else {
33         tr[u].len1 = tr[u << 1].len1 + tr[u << 1 | 1].len1;
34     }
35     // 更新len2
36     if (tr[u].cnt > 1) {
37         tr[u].len2 = ys[tr[u].r + 1] - ys[tr[u].l];
38     } else if (tr[u].l == tr[u].r) {
39         tr[u].len2 = 0;

```

```
39     } else {
40         if (tr[u].cnt == 1) { // 被完全覆盖了1次
41             // 如果子区间有恰好被覆盖至少1次的, 那么合在一起就是至少覆盖2次的面积了
42             tr[u].len2 = tr[u << 1].len1 + tr[u << 1 | 1].len1; // 加上子
                区间至少覆盖1次的面积
43         } else { // cnt=0
44             tr[u].len2 = tr[u << 1].len2 + tr[u << 1 | 1].len2;
45         }
46     }
47 }
48
49 void build(int u, int l, int r) {
50     tr[u] = {l, r, 0, 0};
51     if (l == r) return;
52     int mid = l + r >> 1;
53     build(u << 1, l, mid), build(u << 1 | 1, mid + 1, r);
54 }
55
56 void modify(int u, int l, int r, int k) {
57     if (tr[u].l >= l && tr[u].r <= r) {
58         tr[u].cnt += k;
59         pushup(u);
60     } else {
61         int mid = tr[u].l + tr[u].r >> 1;
62         if (l <= mid) modify(u << 1, l, r, k);
63         if (r > mid) modify(u << 1 | 1, l, r, k);
64         pushup(u);
65     }
66 }
67
68 int main() {
69     for (cin >> T; T--; ) {
70         cin >> n;
71         ys.clear();
72         for (int i = 0; i < n; i++) {
73             double x1, y1, x2, y2;
74             scanf("%lf%lf%lf%lf", &x1, &y1, &x2, &y2);
75             seg[i * 2] = {x1, y1, y2, 1};
76             seg[i * 2 + 1] = {x2, y1, y2, -1};
77             ys.push_back(y1), ys.push_back(y2);
78         }
79         sort(ys.begin(), ys.end());
80         ys.erase(unique(ys.begin(), ys.end()), ys.end());
```



```
81     build(1, 0, ys.size() - 2);
82     sort(seg, seg + n * 2);
83     double res = 0;
84     for (int i = 0; i < n * 2; i++){
85         if (i) res += tr[1].len2 * (seg[i].x - seg[i - 1].x);
86         int l = find(seg[i].y1), r = find(seg[i].y2) - 1;
87         modify(1, l, r, seg[i].k);
88     }
89     printf("%.2lf\n", res);
90 }
91 return 0;
92 }
```

### 1.3 ST 表

```
1 // ST表可维护区间最值/区间gcd
2 int n, a[N];
3 int f[N][M]; // f[i][j]表示区间[i, i+2^j-1]区间的最大值
4 int Log2[N];
5
6 void ST_pre() {
7     Log2[2] = 1;
8     for (int i = 3; i < N; i++) Log2[i] = Log2[i >> 1] + 1;
9     for (int i = 1; i <= n; i++) f[i][0] = a[i];
10    for (int j = 1; j < M; j++) {
11        for (int i = 1; i + (1 << j) - 1 <= n; i++)
12            f[i][j] = max(f[i][j-1], f[i + (1 << j - 1)][j - 1]);
13    }
14 }
15
16 int query(int l, int r) {
17     int k = Log2[r - l + 1];
18     return max(f[l][k], f[r - (1 << k) + 1][k]);
19 }
```

### 1.4 Splay

### 1.5 Treap

```
1  #include <bits/stdc++.h>
2
3  using namespace std;
4
5  const int N = 100010, INF = 1e8;
6
7  int n;
8  struct Node {
9      int l, r;
10     int key, val;
11     int cnt, sz;
12 } tr[N];
13 int root, idx;
14
15 int new_node(int key) {
16     tr[++idx].key = key;
17     tr[idx].val = rand();
18     tr[idx].cnt = tr[idx].sz = 1;
19     return idx;
20 }
21
22 void pushup(int p) {
23     tr[p].sz = tr[tr[p].l].sz + tr[tr[p].r].sz + tr[p].cnt;
24 }
25
26 void build() {
27     new_node(-INF), new_node(INF);
28     root = 1, tr[1].r = 2;
29     pushup(root);
30 }
31
32 void RR(int &p) {
33     int q = tr[p].l;
34     tr[p].l = tr[q].r, tr[q].r = p, p = q;
35     pushup(tr[p].r), pushup(p);
36 }
37
38 void LR(int &p) {
39     int q = tr[p].r;
40     tr[p].r = tr[q].l, tr[q].l = p, p = q;
41     pushup(tr[p].l), pushup(p);
42 }
```

```
43
44 void insert(int &p, int key) {
45     if (!p) p = new_node(key);
46     else if (tr[p].key == key) tr[p].cnt++;
47     else if (tr[p].key > key) {
48         insert(tr[p].l, key);
49         if (tr[tr[p].l].val > tr[p].val) RR(p);
50     } else {
51         insert(tr[p].r, key);
52         if (tr[tr[p].r].val > tr[p].val) LR(p);
53     }
54     pushup(p);
55 }
56
57 void remove(int &p, int key) {
58     if (!p) return;
59     if (tr[p].key == key) {
60         if (tr[p].cnt > 1) tr[p].cnt--;
61         else if (tr[p].l || tr[p].r) {
62             if (!tr[p].r || tr[tr[p].l].val > tr[tr[p].r].val) {
63                 RR(p);
64                 remove(tr[p].r, key);
65             } else {
66                 LR(p);
67                 remove(tr[p].l, key);
68             }
69         } else {
70             p = 0;
71         }
72     } else if (tr[p].key > key) {
73         remove(tr[p].l, key);
74     } else {
75         remove(tr[p].r, key);
76     }
77     pushup(p);
78 }
79
80 int get_rank_by_key(int p, int key) {
81     if (!p) return 0; // never occur in this problem
82     if (tr[p].key == key) return tr[tr[p].l].sz + 1;
83     if (tr[p].key > key) return get_rank_by_key(tr[p].l, key);
84     return tr[tr[p].l].sz + tr[p].cnt + get_rank_by_key(tr[p].r, key);
85 }
```

```
86
87 int get_key_by_rank(int p, int rank) {
88     if (!p) return INF; // never occur in this problem
89     if (tr[tr[p].l].sz >= rank) return get_key_by_rank(tr[p].l, rank);
90     if (tr[tr[p].l].sz + tr[p].cnt >= rank) return tr[p].key;
91     return get_key_by_rank(tr[p].r, rank - tr[tr[p].l].sz - tr[p].cnt);
92 }
93
94 // find max that smaller than key
95 int get_prev(int p, int key) {
96     if (!p) return -INF;
97     if (tr[p].key >= key) return get_prev(tr[p].l, key);
98     return max(tr[p].key, get_prev(tr[p].r, key));
99 }
100
101 // find min that bigger than key
102 int get_next(int p, int key) {
103     if (!p) return INF;
104     if (tr[p].key <= key) return get_next(tr[p].r, key);
105     return min(tr[p].key, get_next(tr[p].l, key));
106 }
107
108 int main() {
109     build();
110     cin >> n;
111     while (n--) {
112         int op, x;
113         cin >> op >> x;
114         if (op == 1) insert(root, x);
115         else if (op == 2) remove(root, x);
116         else if (op == 3) printf("%d\n", get_rank_by_key(root, x) - 1);
117             // 注意之前插入了-INF的哨兵
118         else if (op == 4) printf("%d\n", get_key_by_rank(root, x + 1));
119         else if (op == 5) printf("%d\n", get_prev(root, x));
120         else printf("%d\n", get_next(root, x));
121     }
122     return 0;
123 }
```

## 1.6 树链剖分

### 1.6.1 点权

题意：给定一棵树，树中包含  $n$  个结点，有点权。初始时，1 号结点为树的根结点。现在要对该树进行  $m$  次操作，操作分为以下 4 种类型：

1.  $u\ v\ k$ ，修改路径上节点权值，将节点  $u$  和节点  $v$  之间路径上的所有节点（包括这两个节点）的权值增加  $k$ 。
2.  $u\ k$ ，修改子树上节点权值，将以节点  $u$  为根的子树上的所有节点的权值增加  $k$ 。
3.  $u\ v$ ，询问路径，询问节点  $u$  和节点  $v$  之间路径上的所有节点（包括这两个节点）的权值和。
4.  $u$ ，询问子树，询问以节点  $u$  为根的子树上的所有节点的权值和。

分析：树剖后以 DFS 序建线段树，子树  $u$  对应线段树上的区间  $[dfn[u], dfn[u]+sz[u]-1]$ ， $u\ v$  之间路径的区间构成详见代码。

```

1  #include <bits/stdc++.h>
2
3  using namespace std;
4
5  typedef long long LL;
6
7  const int N = 100010, M = N << 1;
8
9  int n, m;
10 int a[N], na[N]; // na表示新编号
11 int h[N], e[M], ne[M], idx;
12 struct Tree {
13     int l, r;
14     LL sum, add;
15 } tr[N << 2];
16 int dfn[N], ts; // dfn表示dfs序（优先遍历重儿子）
17 int dep[N], sz[N], top[N], fa[N], son[N];
18 // dep[i]表示i的深度（根节点的深度为1），sz[i]表示以i为根的子树的大小
19 // top[i]表示i所在重链的顶点，fa[i]表示i的父结点，son[i]表示子树i的重儿子
20
21 void add(int a, int b) {
22     e[idx] = b, ne[idx] = h[a], h[a] = idx++;
23 }
24
25 void dfs1(int u, int father, int depth) {
26     dep[u] = depth, fa[u] = father, sz[u] = 1;

```

```

27     for (int i = h[u]; ~i; i = ne[i]) {
28         int j = e[i];
29         if (j == father) continue;
30         dfs1(j, u, depth + 1);
31         sz[u] += sz[j];
32         if (sz[son[u]] < sz[j]) son[u] = j;
33     }
34 }
35
36 // 点u所属的重链的顶点为t
37 void dfs2(int u, int t) {
38     dfn[u] = ++ts, na[ts] = a[u], top[u] = t;
39     if (!son[u]) return; // u为叶结点
40     dfs2(son[u], t); // 重儿子
41     // 处理轻儿子
42     for (int i = h[u]; ~i; i = ne[i]) {
43         int j = e[i];
44         if (j == fa[u] || j == son[u]) continue;
45         dfs2(j, j); // 轻儿子所处重链的顶点就是自己
46     }
47 }
48
49 void pushup(int u) {
50     tr[u].sum = tr[u << 1].sum + tr[u << 1 | 1].sum;
51 }
52
53 void build(int u, int l, int r) {
54     if (l == r) {
55         tr[u] = {l, r, na[r], 0};
56     } else {
57         tr[u] = {l, r};
58         int mid = l + r >> 1;
59         build(u << 1, l, mid), build(u << 1 | 1, mid + 1, r);
60         pushup(u);
61     }
62 }
63
64 void pushdown(int u) {
65     auto &root = tr[u], &left = tr[u << 1], &right = tr[u << 1 | 1];
66     if (root.add) {
67         left.add += root.add, left.sum += root.add * (left.r - left.l +
68             1);
69         right.add += root.add, right.sum += root.add * (right.r - right.

```

```
        l + 1);
69     root.add = 0;
70 }
71 }
72
73 // 将[l,r]区间加上k
74 void update(int u, int l, int r, int k) {
75     if (tr[u].l >= l && tr[u].r <= r) {
76         tr[u].add += k;
77         tr[u].sum += k * (tr[u].r - tr[u].l + 1);
78         return;
79     }
80     pushdown(u);
81     int mid = tr[u].l + tr[u].r >> 1;
82     if (l <= mid) update(u << 1, l, r, k);
83     if (r > mid) update(u << 1 | 1, l, r, k);
84     pushup(u);
85 }
86
87 // 求[l,r]区间和
88 LL query(int u, int l, int r) {
89     if (tr[u].l >= l && tr[u].r <= r) {
90         return tr[u].sum;
91     }
92     pushdown(u); // 下传add标记
93     LL res = 0;
94     int mid = tr[u].l + tr[u].r >> 1;
95     if (l <= mid) res += query(u << 1, l, r);
96     if (r > mid) res += query(u << 1 | 1, l, r);
97     return res;
98 }
99
100 // 将树上u->v的路径全部加上k
101 void update_path(int u, int v, int k) {
102     while (top[u] != top[v]) { // u,v不在同一个重链上
103         if (dep[top[u]] < dep[top[v]]) swap(u, v);
104         // 加上u所在的重链和
105         // 其区间为[dfn[top[u]], dfn[u]]
106         update(1, dfn[top[u]], dfn[u], k);
107         u = fa[top[u]];
108     }
109     if (dep[u] < dep[v]) swap(u, v);
110     // 加上u-v之间路径的和
```

```
111     update(1, dfn[v], dfn[u], k);
112 }
113
114 // 求树上u-v之间的路径和
115 LL query_path(int u, int v) {
116     LL res = 0;
117     while (top[u] != top[v]) { // u,v不在同一个重链上
118         if (dep[top[u]] < dep[top[v]]) swap(u, v);
119         // 加上u所在的重链和
120         // 其区间为[dfn[top[u]], dfn[u]]
121         res += query(1, dfn[top[u]], dfn[u]);
122         u = fa[top[u]];
123     }
124     if (dep[u] < dep[v]) swap(u, v);
125     // 加上u-v之间路径的和
126     res += query(1, dfn[v], dfn[u]);
127     return res;
128 }
129
130 // 将树上以u为根的子树全部加上k
131 void update_tree(int u, int k) {
132     // 对应区间 [dfn[u], dfn[u]+sz[u]-1]
133     update(1, dfn[u], dfn[u] + sz[u] - 1, k);
134 }
135
136 // 求树上以u为根的子树的和
137 LL query_tree(int u) {
138     // 对应区间 [dfn[u], dfn[u]+sz[u]-1]
139     return query(1, dfn[u], dfn[u] + sz[u] - 1);
140 }
141
142 int main() {
143     memset(h, -1, sizeof h);
144     scanf("%d", &n);
145     for (int i = 1; i <= n; i++) scanf("%d", &a[i]);
146     for (int i = 0; i < n - 1; i++) {
147         int a, b;
148         scanf("%d%d", &a, &b);
149         add(a, b), add(b, a);
150     }
151     dfs1(1, -1, 1); // 预处理dep,fa,sz
152     dfs2(1, 1); // 求dfs序dfn,top
153     build(1, 1, n); // 建立线段树
```



```

154     for (scanf("%d", &m); m--; ) {
155         int type, u, v, k;
156         scanf("%d", &type);
157         if (type == 1) {
158             scanf("%d%d%d", &u, &v, &k);
159             update_path(u, v, k);
160         } else if (type == 2) {
161             scanf("%d%d", &u, &k);
162             update_tree(u, k);
163         } else if (type == 3) {
164             scanf("%d%d", &u, &v);
165             printf("%lld\n", query_path(u, v));
166         } else {
167             scanf("%d", &u);
168             printf("%lld\n", query_tree(u));
169         }
170     }
171     return 0;
172 }

```

### 1.6.2 边权

题意：有  $n$  个点的树，有边权。两种操作：1. 0 a b，表示更新第  $a$  条边权为  $b$ ，2. 1 a b 表示询问  $a$  到  $b$  的路径上边权之和。

分析：树链剖分后转化为序列问题用线段树维护，由于线段树没法维护边权，因此在原来树中需要将边权下放到点权，由于每个结点（除根结点）只有 1 个父亲，因此可以将父亲  $\rightarrow$  儿子的边权下放到儿子的点权上。求树上两点  $u, v$  距离时，需要注意不能把  $LCA(u, v)$  的点权计算进去。

```

1  /* FZU2082 */
2  #include <iostream>
3  #include <algorithm>
4  #include <cstdio>
5  #include <cstring>
6
7  using namespace std;
8
9  typedef pair<int, int> PII;
10 typedef long long LL;
11
12 const int N = 50010, M = N << 1;

```

```
13
14 int n, m;
15 int h[N], e[M], w[M], ne[M], idx;
16 struct Tree {
17     int l, r;
18     LL sum;
19 } tr[N << 2];
20 int dfn[N], ts; // dfn表示dfs序 (优先遍历重儿子)
21 int dep[N], sz[N], top[N], fa[N], son[N];
22 // dep[i]表示i的深度 (根节点的深度为1), sz[i]表示以i为根的子树的大小
23 // top[i]表示i所在重链的顶点, fa[i]表示i的父结点, son[i]表示子树i的重儿子
24
25 void add(int a, int b, int c) {
26     e[idx] = b, w[idx] = c, ne[idx] = h[a], h[a] = idx++;
27 }
28
29 void dfs1(int u, int father, int depth) {
30     dep[u] = depth, fa[u] = father, sz[u] = 1;
31     for (int i = h[u]; ~i; i = ne[i]) {
32         int j = e[i];
33         if (j == father) continue;
34         dfs1(j, u, depth + 1);
35         sz[u] += sz[j];
36         if (sz[son[u]] < sz[j]) son[u] = j;
37     }
38 }
39
40 // 点u所属的重链的顶点为t
41 void dfs2(int u, int t) {
42     dfn[u] = ++ts, top[u] = t;
43     if (!son[u]) return; // u为叶结点
44     dfs2(son[u], t); // 处理重儿子
45     // 处理轻儿子
46     for (int i = h[u]; ~i; i = ne[i]) {
47         int j = e[i];
48         if (j == fa[u] || j == son[u]) continue;
49         dfs2(j, j); // 轻儿子所处重链的顶点就是自己
50     }
51 }
52
53 void pushup(int u) {
54     tr[u].sum = tr[u << 1].sum + tr[u << 1 | 1].sum;
55 }
```

```
56
57 void build(int u, int l, int r) {
58     if (l == r) {
59         tr[u].l = l, tr[u].r = r, tr[u].sum = 0;
60     } else {
61         tr[u].l = l, tr[u].r = r;
62         int mid = l + r >> 1;
63         build(u << 1, l, mid), build(u << 1 | 1, mid + 1, r);
64     }
65 }
66
67 // 将x位置修改成y
68 void update(int u, int x, int y) {
69     if (tr[u].l == x && tr[u].r == x) {
70         tr[u].sum = y;
71     } else {
72         int mid = tr[u].l + tr[u].r >> 1;
73         if (x <= mid) update(u << 1, x, y);
74         else update(u << 1 | 1, x, y);
75         pushup(u);
76     }
77 }
78
79 // 求线段树中[l,r]区间和
80 LL query(int u, int l, int r) {
81     if (tr[u].l >= l && tr[u].r <= r) {
82         return tr[u].sum;
83     } else {
84         LL res = 0;
85         int mid = tr[u].l + tr[u].r >> 1;
86         if (l <= mid) res += query(u << 1, l, r);
87         if (r > mid) res += query(u << 1 | 1, l, r);
88         return res;
89     }
90 }
91
92 // 求树上u-v之间的路径和
93 LL query_path(int u, int v) {
94     LL res = 0;
95     while (top[u] != top[v]) { // u,v不在同一个重链上
96         if (dep[top[u]] < dep[top[v]]) swap(u, v);
97         // 加上u所在的重链和
98         // 其区间为[dfn[top[u]], dfn[u]]
```

```
99     res += query(1, dfn[top[u]], dfn[u]);
100     u = fa[top[u]];
101 }
102 if (u == v) return res;
103 if (dep[u] < dep[v]) swap(u, v); // 保证u在下面v在上面
104 // 加上u-v之间路径的和, v当前在LCA位置, 这个点的点权不能算上。
105 res += query(1, dfn[son[v]], dfn[u]);
106 return res;
107 }
108
109 int main() {
110     while (cin >> n >> m) {
111         for (int i = 1; i <= n; i++) {
112             h[i] = -1, dfn[i] = 0, son[i] = 0;
113         }
114         idx = ts = 0;
115         for (int i = 0; i < n - 1; i++) {
116             int a, b, c;
117             scanf("%d%d%d", &a, &b, &c);
118             add(a, b, c), add(b, a, c);
119         }
120         dfs1(1, -1, 1); // 预处理dep, fa, sz
121         dfs2(1, 1); // 求dfs序dfn, top
122         build(1, 1, n);
123         for (int i = 0; i < idx; i += 2) {
124             int u = e[i], v = e[i ^ 1];
125             if (dep[u] > dep[v]) swap(u, v);
126             update(1, dfn[v], w[i]);
127         }
128         while (m--) {
129             int type, x, y;
130             cin >> type >> x >> y;
131             if (type == 0) { // 更新第x条路的过路费为y
132                 // 第1条 idx=0,1
133                 // 第2条 idx=2,3
134                 // 第x条 idx=2x-2,2x-1
135                 int v = e[2 * x - 2], u = e[2 * x - 1];
136                 if (dep[u] > dep[v]) swap(u, v);
137                 // u是父亲, v是儿子, 令点v权值更新为y
138                 update(1, dfn[v], y);
139             } else {
140                 printf("%lld\n", query_path(x, y));
141             }
142         }
143     }
144 }
```

```
142     }
143 }
144     return 0;
145 }
```

## 2 图论

### 2.1 最短路

### 2.2 最小生成树

### 2.3 次小生成树

### 2.4 有向图的强连通分量

### 2.5 无向图的双连通分量

### 2.6 最近公共祖先

### 2.7 2-SAT

### 2.8 网络流

#### 2.8.1 Dinic

test

```
1  #include <bits/stdc++.h>
2
3  using namespace std;
4
5  const int N = 100010, M = 200010, INF = 1e9;
6
7  int n, m, S, T;
8  int h[N], e[M], w[M], ne[M], idx;
9  int q[N];
10 int d[N], cur[N]; // d[i]表示点i的层次, cur[i]表示i的当前弧
11
12 void add(int a, int b, int c) {
13     e[idx] = b, w[idx] = c, ne[idx] = h[a], h[a] = idx++;
14     e[idx] = a, w[idx] = 0, ne[idx] = h[b], h[b] = idx++;
15 }
16
17 bool bfs() { // 判断残留网络是否存在增广路 (即从S到T存在边全大于0的路径)
```

```

18     int hh = 0, tt = -1;
19     memset(d, -1, sizeof d);
20     q[++tt] = S, d[S] = 0, cur[S] = h[S];
21     while (hh <= tt) {
22         int t = q[hh++];
23         for (int i = h[t]; ~i; i = ne[i]) {
24             int j = e[i];
25             if (d[j] == -1 && w[i]) {
26                 d[j] = d[t] + 1;
27                 cur[j] = h[j];
28                 q[++tt] = j;
29                 if (j == T) return true; // 找到了增广路, 此时增广路的流量即f[T]
30             }
31         }
32     }
33     return false; // 残留网络不存在增广路, 那么此时原图的可行流流量就是最大流
34 }
35
36 // 从起点到u, 流量最大值为limit
37 int find(int u, int limit) {
38     if (u == T) return limit;
39     int flow = 0; // u->T的流量
40     for (int i = cur[u]; ~i && flow < limit; i = ne[i]) {
41         int j = e[i];
42         cur[u] = i; // 更新当前弧
43         if (d[j] == d[u] + 1 && w[i]) {
44             int t = find(j, min(w[i], limit - flow));
45             if (!t) d[j] = -1; // 删点
46             w[i] -= t, w[i ^ 1] += t, flow += t;
47         }
48     }
49     return flow;
50 }
51
52 int dinic() {
53     int max_flow = 0;
54     while (bfs()) while (int flow = find(S, INF)) max_flow += flow;
55     return max_flow;
56 }
57
58 int main() {
59     cin >> n >> m >> S >> T;

```

```

60     memset(h, -1, sizeof h);
61     while (m--) {
62         int a, b, c;
63         scanf("%d%d%d", &a, &b, &c);
64         add(a, b, c); // 初始流量为0, 对于原图的残留网络, 正向边容量为c-0=c, 反
            向边容量为0+0=0
65     }
66     printf("%d\n", dinic());
67     return 0;
68 }

```

### 2.8.2 EK

```

1  #include <bits/stdc++.h>
2
3  using namespace std;
4
5  const int N = 1010, M = 20010, INF = 1e9;
6
7  int n, m, S, T;
8  int h[N], e[M], w[M], ne[M], idx;
9  bool st[N];
10 int q[N], f[N]; // f[i]表示以i结尾的增广路径的流量 (即路径上容量的最小值)
11 int pre[N]; // pre[i]表示i的前驱边的编号 (即指向i的边)
12
13 void add(int a, int b, int c) {
14     e[idx] = b, w[idx] = c, ne[idx] = h[a], h[a] = idx++;
15     e[idx] = a, w[idx] = 0, ne[idx] = h[b], h[b] = idx++;
16 }
17
18 bool bfs() { // 判断残留网络是否存在增广路 (即从S到T存在边全大于0的路径)
19     int hh = 0, tt = -1;
20     memset(st, false, sizeof st);
21     q[++tt] = S, st[S] = true, f[S] = INF;
22     while (hh <= tt) {
23         int t = q[hh++];
24         for (int i = h[t]; ~i; i = ne[i]) {
25             int j = e[i];
26             if (!st[j] && w[i]) {
27                 q[++tt] = j;
28                 st[j] = true;
29                 pre[j] = i; // 记录j的前驱边
30                 f[j] = min(f[t], w[i]);

```

```

31         if (j == T) return true; // j是终点，即找到了增广路，此时增广路
           流量即f[T]
32     }
33 }
34 }
35 return false; // 残留网络不存在增广路，那么此时原图的可行流流量就是最大流
36 }
37
38 int EK() {
39     int flow = 0;
40     while (bfs()) { // 如果残留网络存在增广路f', 就将他的流量加到原网络的流量上
41         flow += f[T];
42         for (int i = T; i != S; i = e[pre[i] ^ 1]) { // 更新残留网络
43             w[pre[i]] -= f[T], w[pre[i] ^ 1] += f[T];
44         }
45     }
46     return flow; // 最大流
47 }
48
49 int main() {
50     cin >> n >> m >> S >> T;
51     memset(h, -1, sizeof h);
52     while (m--) {
53         int a, b, c;
54         scanf("%d%d%d", &a, &b, &c);
55         add(a, b, c); // 初始流量为0，对于原图的残留网络，正向边容量为c-0=c，反
           向边容量为0+0=0
56     }
57     printf("%d\n", EK());
58     return 0;
59 }

```

## 2.9 二分图

# 3 字符串

## 3.1 Manacher

```

1  /* O(n) 求字符串s的最大回文长度 */
2  #include <iostream>
3  #include <cstring>
4  #include <cstdio>

```



```
5 #include <algorithm>
6
7 using namespace std;
8
9 const int N = 2000010;
10
11 int n, m, Case;
12 char s[N], str[N]; // s为原串, str为插入分隔符后的串
13 int p[N]; // p[i]为str中以下标i为中心的最大回文半径
14 // p[i]-1为s中以i为回文中心的最大回文长度
15
16 void manacher() {
17     int rt = 0, mid = 0;
18     int res = 0;
19     for (int i = 1; i <= m; i++) {
20         p[i] = i < rt ? min(p[2 * mid - i], rt - i) : 1;
21         while (str[i + p[i]] == str[i - p[i]]) p[i]++;
22         if (i + p[i] > rt) {
23             rt = i + p[i];
24             mid = i;
25         }
26         res = max(res, p[i] - 1);
27     }
28     printf("Case %d: %d\n", ++Case, res);
29 }
30
31 int main() {
32     str[0] = '!', str[1] = '#'; /* str[0]为哨兵 */
33     while (scanf("%s", s), s[0] != 'E') {
34         n = strlen(s);
35         for (int i = 0; i < n; i++) {
36             str[i * 2 + 2] = s[i];
37             str[i * 2 + 3] = '#';
38         }
39         m = n * 2 + 1;
40         str[m + 1] = '@'; /* 哨兵 */
41         manacher();
42     }
43     return 0;
44 }
```

## 3.2 KMP

```
1  /* 求出模板串P在模式串S中所有出现的位置的起始下标 */
2  #include <iostream>
3  #include <algorithm>
4
5  using namespace std;
6
7  const int N = 10010, M = 100010;
8
9  int n, m; // n,m分别为p,s的长度
10 char p[N], s[M];
11 int ne[N]; // ne[i]表示以i结尾的真后缀能够匹配前缀的最大长度
12
13 int main() {
14     cin >> n >> p + 1 >> m >> s + 1;
15     for (int i = 2, j = 0; i <= n; i++) {
16         while (j && p[i] != p[j + 1]) j = ne[j];
17         if (p[i] == p[j + 1]) j++;
18         ne[i] = j;
19     }
20     for (int i = 1, j = 0; i <= m; i++) {
21         while (j && (s[i] != p[j + 1])) j = ne[j];
22         if (s[i] == p[j + 1]) j++;
23         if (j == n) { // 匹配完成
24             printf("%d ", i - 1 - n + 1);
25             j = ne[j];
26         }
27     }
28     return 0;
29 }
```

### 3.3 AC 自动机

### 3.4 后缀数组

## 4 其他

### 4.1 高精度

```
1 struct hll {
2     int num[4010], len, sign;
3     hll() { len = 0, sign = 1; }
4     hll(int x) { *this = x; }
```

```
5  hll(long long x) { *this = x; }
6  hll(char *ss) { *this = ss; }
7  hll(string ss) { *this = ss; }
8  hll& operator = (const int &x) {
9      int val = x;
10     if (val >= 0) sign = 1, len = 0;
11     else if (val < 0) sign = -1, val = -val, len = 0;
12     do {
13         num[len++] = val % 10, val /= 10;
14     } while (val);
15     return *this;
16 }
17 hll& operator = (const long long &x) {
18     long long val = x;
19     if (val >= 0) sign = 1, len = 0;
20     else if (val < 0) sign = -1, val = -val, len = 0;
21     do {
22         num[len++] = val % 10, val /= 10;
23     } while (val);
24     return *this;
25 }
26 hll& operator = (const string &ss) {
27     len = ss.size();
28     int start;
29     if (ss[0] == '-') sign = -1, start = 1;
30     else sign = 1, start = 0;
31     for (int i = len - 1; i >= start; i--) num[len - i - 1] = ss[i]
        - '0';
32     if (sign == -1) len--;
33     return *this;
34 }
35 hll& operator = (const char *ss) {
36     len = strlen(ss);
37     int start;
38     if (ss[0] == '-') sign = -1, start = 1;
39     else sign = 1, start = 0;
40     for (int i = len - 1; i >= start; i--) num[len - i - 1] = ss[i]
        - '0';
41     if (sign == -1) len--;
42     return *this;
43 }
44 hll& operator = (const hll &t) {
45     len = t.len, sign = t.sign;
```

```
46     for (int i = 0; i < len; i++) num[i] = t.num[i];
47     return *this;
48 }
49 int abs_cmp(const hll &a, const hll &b) const { // |a|>|b|时返回1,
    相等返回0, 小于返回-1
50     if (a.len > b.len) return 1;
51     else if (a.len < b.len) return -1;
52     else {
53         for (int i = a.len - 1; i >= 0; i--) {
54             if (a.num[i] < b.num[i]) return -1;
55             if (a.num[i] > b.num[i]) return 1;
56         }
57         return 0;
58     }
59 }
60 int cmp(const hll &t) const { // *this与t比较, 小于返回-1, 等于返回0, 大
    于返回1
61     if (sign != t.sign) {
62         if (sign == 1) return 1;
63         else return -1;
64     } else {
65         if (abs_cmp(*this, t) == 1) return sign;
66         else if (abs_cmp(*this, t) == 0) return 0;
67         else return -sign;
68     }
69 }
70 hll abs_plus(const hll &a, const hll &b) { // |a|+|b|, ans的符号与a
    和b原来的符号相同
71     hll ans;
72     ans.sign = a.sign;
73     for (int i = 0, carry = 0; i < a.len || i < b.len || carry; i++)
74     {
75         if (i < a.len) carry += a.num[i];
76         if (i < b.len) carry += b.num[i];
77         ans.num[ans.len++] = carry % 10;
78         carry /= 10;
79     }
80     return ans;
81 }
82 hll abs_minus(const hll &a, const hll &b) { // ||a|-|b||, ans的符号
    为|a|-|b|的符号
83     hll ans, c, d;
84     if (abs_cmp(a, b) >= 0) ans.sign = 1, c = a, d = b;
```

```
84     else ans.sign = -1, c = b, d = a;
85     for (int i = 0, borrow = 0; i < c.len; i++) {
86         borrow = c.num[i] - borrow;
87         if (i < d.len) borrow -= d.num[i];
88         ans.num[ans.len++] = (borrow + 10) % 10;
89         if (borrow >= 0) borrow = 0;
90         else borrow = 1;
91     }
92     while (ans.len > 1 && ans.num[ans.len - 1] == 0) ans.len--; //
        去除前导0
93     return ans;
94 }
95 bool operator == (const hll &t) const { return cmp(t) == 0; }
96 bool operator != (const hll &t) const { return !(cmp(t) == 0); }
97 bool operator < (const hll &t) const { return cmp(t) == -1; }
98 bool operator > (const hll &t) const { return cmp(t) == 1; }
99 bool operator <= (const hll &t) const { return !(cmp(t) == 1); }
100 bool operator >= (const hll &t) const { return !(cmp(t) == -1); }
101 hll operator + (const hll &t) {
102     hll ans;
103     if (sign == t.sign) { // 同号 直接相加, 符号不变
104         ans = abs_plus(*this, t);
105     } else { // 异号
106         if (sign == 1) { // 前正 + 后负 == 前绝对值 - 后绝对值
107             ans = abs_minus(*this, t);
108         } else { // 前负 + 后正 == 后绝对值 - 前绝对值
109             ans = abs_minus(t, *this);
110         }
111     }
112     return ans;
113 }
114 hll operator - (const hll &t) {
115     hll ans;
116     if (sign == t.sign) { // 同号
117         ans = abs_minus(*this, t);
118         if (sign == 1) { // 前正 - 后正
119             ; // 不用做了
120         } else { // 前负 - 后负
121             ans.sign *= -1;
122         }
123     } else { // 异号
124         if (sign == 1) { // 前正 - 后负 == 前绝对值 + 后绝对值
125             ans = abs_plus(*this, t);
```

```
126         } else { // 前负 - 后正 == -(前绝对值 + 前绝对值)
127             ans = abs_plus(t, *this);
128             ans.sign = -1;
129         }
130     }
131     return ans;
132 }
133 hll operator * (const hll &t) { // 高精度*高精度
134     hll ans;
135     memset(ans.num, 0, len + t.len << 2);
136     ans.sign = sign * t.sign;
137     ans.len = len + t.len - 1; // a位数乘以b位数, 得到的结果是a+b-1位数,
        或a+b位数
138     for (int i = 0; i < len; i++) {
139         for (int j = 0; j < t.len; j++)
140             ans.num[i + j] += num[i] * t.num[j];
141     }
142     for (int i = 0; i < ans.len - 1; i++) {
143         if (ans.num[i] >= 10) {
144             ans.num[i + 1] += ans.num[i] / 10;
145             ans.num[i] %= 10;
146         }
147     }
148     // 看最高位是否需要进位, 如果有进位, 答案最终是a+b位数, 否则是a+b-1位数
149     if (ans.num[ans.len - 1] >= 10) {
150         ans.num[ans.len] = ans.num[ans.len - 1] / 10;
151         ans.num[ans.len - 1] %= 10;
152         ans.len++;
153     }
154     while (ans.len > 1 && ans.num[ans.len - 1] == 0) ans.len--; //
        去除前导0
155     return ans;
156 }
157 hll operator += (const hll &t) {
158     return *this + t;
159 }
160 hll operator *= (const hll &t) {
161     return *this * t;
162 }
163 void print() {
164     if (sign == -1) putchar('-');
165     for (int i = len - 1; i >= 0; i--) putchar(num[i] + '0');
166 }
```

167  $\};$

## 4.2 莫队