NJUPT XCPC Templates

Yunhai Bian

2020 年 12 月 10 日

目录

1 数据结构

1.1 树状数组

```
// 注意树状数组不能处理下标0开始
// 一维
int c[N];

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

int add(int x, int y) {
   for (int i = x; i <= n; i += lowbit(i)) c[i] += y;
}

int sum(int x) {
   int res = 0;
   for (int i = x; i; i -= lowbit(i)) res += c[i];
   return res;
}

// 二维
LL c[N][N];

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

void add(int x, int y, LL v) {
   for (int i = x; i <= n;i += lowbit(i))
      for (int j = y; j <= m;j += lowbit(j))
         c[i][j] += v;
}

LL query(int x, int y) {
   LL ans = 0;
   for (int i = x; i; i -= lowbit(i))
      for (int j = y; j; j -= lowbit(j))
         ans += c[i][j];
   return ans;
}
```

1.2 线段树

1.2.1 HDU1540

题意：有个点连成一条线，编号从左至右为，有三种操作： 摧毁一个点 查询某个点能到的所有点数（包括自己） 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 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。

```
#include <iostream>

using namespace std;

const int N = 50010;

int n, m;
struct Tree {
   int l, r;
   int lmax, rmax;
} tr[N << 2];
int stk[N], top;

void pushup(Tree &root, Tree &left, Tree &right) {
   root.lmax = left.lmax, root.rmax = right.rmax;
   if (left.r - left.l + 1 == left.lmax) root.lmax += right.lmax;
   if (right.r - right.l + 1 == right.rmax) root.rmax += left.rmax;
}

void pushup(int u) {
   pushup(tr[u], tr[u << 1], tr[u << 1 | 1]);
}

void build(int u, int l, int r) {
   if (l == r) {
      tr[u] = {l, r, 1, 1};
   } else {
      tr[u] = {l, r};
      int mid = l + r >> 1;
      build(u << 1, l, mid); build(u << 1 | 1, mid + 1, r);
      pushup(u);
   }
}

void modify(int u, int x, int y) {
   if (tr[u].l == x && tr[u].r == x) {
      tr[u].lmax = tr[u].rmax = y;
   } else {
      int mid = tr[u].l + tr[u].r >> 1;
      if (x <= mid) modify(u << 1, x, y);
      else modify(u << 1 | 1, x, y);
      pushup(u);
   }
}

int query(int u, int x) {
   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[02] 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.

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

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

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

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。

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

using namespace std;

typedef pair<int, int> PII;
typedef long long LL;

const int N = 100010, mod = 10007;

int n, m;
struct Tree {
   int l, r;
   LL same, mul, add, s[3];
 } tr[N << 2];

void build(int u, int l, int r) {
   tr[u] = {l, r, 0, 1, 0, 0, 0, 0};
   if (l == r) return;
   int mid = l + r >> 1;
   build(u << 1, l, mid), build(u << 1 | 1, mid + 1, r);
}

void pushup(int u) {
   for (int i = 0; i < 3; i++) {
      tr[u].s[i] = (tr[u << 1].s[i] + tr[u << 1 | 1].s[i]) % mod;
   }
}

void pushdown(int u) {
   auto &root = tr[u], &left = tr[u << 1], &right = tr[u << 1 | 1];
   if (root.same) {
      left.same = right.same = root.same;
      left.mul = right.mul = 1, left.add = right.add = 0;
      LL base = 1;
      for (int i = 0; i < 3; i++) {
         (base *= root.same) %= mod;
         left.s[i] = (left.r - left.l + 1) * base % mod;
         right.s[i] = (right.r - right.l + 1) * base % mod;
      }
      root.same = 0;
   }
   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.r -
              left.l + 1) * a * a * a) %= mod;
57        (right.s[2] += 3 * a * right.s[1] + 3 * a * a * right.s[0] + (right.r -
              right.l + 1) * a * a * a) %= mod;
58        (left.s[1] += 2 * a * left.s[0] + (left.r - left.l + 1) * a * a) %= mod
              ;
59        (right.s[1] += 2 * a * right.s[0] + (right.r - right.l + 1) * a * a) %=
              mod;
60        (left.s[0] += (left.r - left.l + 1) * a) %= mod, (right.s[0] += (right.
              r - right.l + 1) * a) %= mod;
61        root.add = 0;
62     }
63  }
64
65  // [l,r]乘k再加a
66  void modify_mul_add(int u, int l, int r, LL k, LL a) {
67     if (tr[u].l >= l && tr[u].r <= r) {
68        if (k != 1) {
69           (tr[u].mul *= k) %= mod;
70           (tr[u].add *= k) %= mod;
71           LL base = 1;
72           for (int i = 0; i < 3; i++) {
73              (base *= k) %= mod;
74              (tr[u].s[i] *= base) %= mod;
75           }
76        }
77        if (a) {
78           (tr[u].add += a) %= mod;
79           (tr[u].s[2] += 3 * a * tr[u].s[1] + 3 * a * a * tr[u].s[0] + (tr[u].
                 r - tr[u].l + 1) * a * a * a) %= mod;
80           (tr[u].s[1] += 2 * a * tr[u].s[0] + (tr[u].r - tr[u].l + 1) * a * a)
                  %= mod;
81           (tr[u].s[0] += (tr[u].r - tr[u].l + 1) * a) %= mod;
82        }
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%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： 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。 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。 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，代码中相关注释应该比较清楚。

```
// 1表示空闲
#include <bits/stdc++.h>

using namespace std;

typedef pair<int, int> PII;
typedef long long LL;

const int N = 100010;

int T, Case, n, m;
struct Tree {
    int l, r;
    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
} tr[N << 2];

void pushup(int u, int type) {
    auto &root = tr[u], &left = tr[u << 1], &right = tr[u << 1 | 1];
    root.lmax[type] = left.lmax[type];
    if (left.lmax[type] == left.r - left.l + 1) root.lmax[type] += right.lmax[type];
    root.rmax[type] = right.rmax[type];
    if (right.rmax[type] == right.r - right.l + 1) root.rmax[type] += left.rmax[type];
    root.tmax[type] = max(max(left.tmax[type], right.tmax[type]), left.rmax[type] + right.lmax[type]);
}

void build(int u, int l, int r) {
    tr[u] = {l, r, -1, -1, 1, 1, 1, 1, 1, 1};
    if (l == r) return;
    int mid = l + r >> 1;
    build(u << 1, l, mid), build(u << 1 | 1, mid + 1, r);
    pushup(u, 0), pushup(u, 1);
}

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]这一整
           段，其表示实际的区间为[ys[l],ys[r+1]]
18     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）的合并长
           度之和。
19 } tr[N << 2]; // 比如 y1, y2, y3 离散化后为k_y1,k_y2,k_y3。其区间[k_y1,k_y2],[
       k_y1,k_y2],[k_y2,k_y3]是线段树中的3个叶子结点。
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     if (tr[u].cnt) {
28         tr[u].len = ys[tr[u].r + 1] - ys[tr[u].l];
29     } else if (tr[u].l != tr[u].r) {
30         tr[u].len = tr[u << 1].len + tr[u << 1 | 1].len;
31     } else {
32         tr[u].len = 0;
33     }
34 }
35
36 void build(int u, int l, int r) {
37     tr[u] = {l, r, 0, 0};
38     if (l == r) return;
39     int mid = l + r >> 1;
40     build(u << 1, l, mid), build(u << 1 | 1, mid + 1, r);
41 }
42
43 void modify(int u, int l, int r, int k) {
44     if (tr[u].l >= l && tr[u].r <= r) {
45         tr[u].cnt += k;
46         pushup(u);
47     } else {
48         int mid = tr[u].l + tr[u].r >> 1;
49         if (l <= mid) modify(u << 1, l, r, k);
50         if (r > mid) modify(u << 1 | 1, l, r, k);
51         pushup(u);
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。只需要求改 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，len2对应
           cnt>1
18  } tr[N << 2];
19  vector<double> ys;
20
21  int find(double y) {
22     return lower_bound(ys.begin(), ys.end(), y) - ys.begin();
```

```
}

void pushup(int u) {
   // 更新len1
   if (tr[u].cnt > 0) {
      tr[u].len1 = ys[tr[u].r + 1] - ys[tr[u].l];
   } else if (tr[u].l == tr[u].r) {
      tr[u].len1 = 0;
   } else {
      tr[u].len1 = tr[u << 1].len1 + tr[u << 1 | 1].len1;
   }
   // 更新len2
   if (tr[u].cnt > 1) {
      tr[u].len2 = ys[tr[u].r + 1] - ys[tr[u].l];
   } else if (tr[u].l == tr[u].r) {
      tr[u].len2 = 0;
   } else {
      if (tr[u].cnt == 1) { // 被完全覆盖了1次
         // 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
         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
      } else { // cnt=0
         tr[u].len2 = tr[u << 1].len2 + tr[u << 1 | 1].len2;
      }
   }
}

void build(int u, int l, int r) {
   tr[u] = {l, r, 0, 0};
   if (l == r) return;
   int mid = l + r >> 1;
   build(u << 1, l, mid), build(u << 1 | 1, mid + 1, r);
}

void modify(int u, int l, int r, int k) {
   if (tr[u].l >= l && tr[u].r <= r) {
      tr[u].cnt += k;
      pushup(u);
   } else {
      int mid = tr[u].l + tr[u].r >> 1;
      if (l <= mid) modify(u << 1, l, r, k);
      if (r > mid) modify(u << 1 | 1, l, r, k);
      pushup(u);
   }
}

int main() {
   for (cin >> T; T--; ) {
      cin >> n;
      ys.clear();
      for (int i = 0; i < n; i++) {
         double x1, y1, x2, y2;
         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

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

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

using namespace std;

const int N = 100010, M = 200010, INF = 1e9;

int n, m, S, T;
int h[N], e[M], w[M], ne[M], idx;
int q[N];
int d[N], cur[N]; // d[i]表示点i的层次，cur[i]表示i的当前弧

void add(int a, int b, int c) {
   e[idx] = b, w[idx] = c, ne[idx] = h[a], h[a] = idx++;
   e[idx] = a, w[idx] = 0, ne[idx] = h[b], h[b] = idx++;
}

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）
   int hh = 0, tt = -1;
   memset(d, -1, sizeof d);
   q[++tt] = S, d[S] = 0, cur[S] = h[S];
   while (hh <= tt) {
      int t = q[hh++];
      for (int i = h[t]; ~i; i = ne[i]) {
         int j = e[i];
         if (d[j] == -1 && w[i]) {
            d[j] = d[t] + 1;
            cur[j] = h[j];
            q[++tt] = j;
            if (j == T) return true; // 找到了增广路，此时增广路的流量即f[T]
         }
      }
   }
   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
```

```
}

// 从起点到u，流量最大值为limit
int find(int u, int limit) {
   if (u == T) return limit;
   int flow = 0; // u->T的流量
   for (int i = cur[u]; ~i && flow < limit; i = ne[i]) {
      int j = e[i];
      cur[u] = i; // 更新当前弧
      if (d[j] == d[u] + 1 && w[i]) {
         int t = find(j, min(w[i], limit - flow));
         if (!t) d[j] = -1; // 删点
         w[i] -= t, w[i ^ 1] += t, flow += t;
      }
   }
   return flow;
}

int dinic() {
   int max_flow = 0;
   while (bfs()) while (int flow = find(S, INF)) max_flow += flow;
   return max_flow;
}

int main() {
   cin >> n >> m >> S >> T;
   memset(h, -1, sizeof h);
   while (m--) {
      int a, b, c;
      scanf("%d%d%d", &a, &b, &c);
      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
   }
   printf("%d\n", dinic());
   return 0;
}
```

2.8.2 EK

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

using namespace std;

const int N = 1010, M = 20010, INF = 1e9;

int n, m, S, T;
int h[N], e[M], w[M], ne[M], idx;
bool st[N];
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）
int pre[N]; // pre[i]表示i的前驱边的编号（即指向i的边）

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，此时增广路的流量即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 其他

3.1 高精度

```
struct hll {
   int num[4010], len, sign;
   hll() { len = 0, sign = 1; }
   hll(int x) { *this = x; }
   hll(long long x) { *this = x; }
   hll(char *ss) { *this = ss; }
   hll(string ss) { *this = ss; }
   hll& operator = (const int &x) {
      int val = x;
      if (val >= 0) sign = 1, len = 0;
      else if (val < 0) sign = -1, val = -val, len = 0;
      do {
         num[len++] = val % 10, val /= 10;
      } while (val);
      return *this;
   }
   hll& operator = (const long long &x) {
      long long val = x;
      if (val >= 0) sign = 1, len = 0;
      else if (val < 0) sign = -1, val = -val, len = 0;
      do {
         num[len++] = val % 10, val /= 10;
      } while (val);
      return *this;
   }
   hll& operator = (const string &ss) {
      len = ss.size();
      int start;
      if (ss[0] == '-') sign = -1, start = 1;
      else sign = 1, start = 0;
      for (int i = len - 1; i >= start; i--) num[len - i - 1] = ss[i] - '0';
      if (sign == -1) len--;
      return *this;
   }
   hll& operator = (const char *ss) {
      len = strlen(ss);
      int start;
      if (ss[0] == '-') sign = -1, start = 1;
      else sign = 1, start = 0;
      for (int i = len - 1; i >= start; i--) num[len - i - 1] = ss[i] - '0';
      if (sign == -1) len--;
      return *this;
   }
   hll& operator = (const hll &t) {
      len = t.len, sign = t.sign;
      for (int i = 0; i < len; i++) num[i] = t.num[i];
      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          if (i < a.len) carry += a.num[i];
75          if (i < b.len) carry += b.num[i];
76          ans.num[ans.len++] = carry % 10;
77          carry /= 10;
78       }
79       return ans;
80    }
81    hll abs_minus(const hll &a, const hll &b) { // ||a|-|b||, ans的符号为|a|-|b
          |的符号
82       hll ans, c, d;
83       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 };
```

3.2 莫队