Standard Code Library

Xavier_Cai

2020 年 10 月 17 日

目录

1 数据结构

1.1 LCT

1.1.1 lct

```
class LCT { public:
   int val[MAXN], sum[MAXN];
   int st_top, st[MAXN]; // stack操作
   int fa[MAXN], ch[MAXN][2];
   bool rev[MAXN];

   inline bool isroot(int x) { // 判断x是否为一个splay的根
      return ch[fa[x]][0] != x && ch[fa[x]][1] != x;
   }
   inline void push_up(int x) {
      int l = ch[x][0], r = ch[x][1];
      sum[x] = sum[l] ^ sum[r] ^ val[x]; // 记录链上异或值
   }
   inline void push_down(int x) {
      int l = ch[x][0], r = ch[x][1];
      if (rev[x]) {
         if (l) swap(ch[l][0], ch[l][1]), rev[l] ^= 1;
         if (r) swap(ch[r][0], ch[r][1]), rev[r] ^= 1;
         rev[x] = 0;
      }
   }
   inline void rotate(int x) { // x向上旋转
      int y = fa[x], z = fa[y], l, r;
      if (ch[y][0] == x) l = 0;
      else l = 1;
      r = l ^ 1;
      if (!isroot(y)) {
         if (ch[z][0] == y) ch[z][0] = x;
         else ch[z][1] = x;
      }
      fa[x] = z, fa[y] = x;
      fa[ch[x][r]] = y;
      ch[y][l] = ch[x][r];
      ch[x][r] = y;
      push_up(y), push_up(x);
   }
   inline void splay(int x) { // 使得x成为当前splay中的根
      st_top = 0;
      st[++st_top] = x;
      for (int i = x; !isroot(i); i = fa[i]) st[++st_top] = fa[i];
      for (int i = st_top; i; i--) push_down(st[i]);
      while (!isroot(x)) {
         int y = fa[x], z = fa[y];
         if (!isroot(y)) {
            if (ch[y][0] == x ^ ch[z][0] == y) rotate(x);
            else rotate(y);
         }
         rotate(x);
      }
   }
   inline void access(int x) { //把x到根节点的路径搞成一个splay
      for (int i = 0; x; i = x, x = fa[x])
         splay(x), ch[x][1] = i, push_up(x);
   }
   inline void makeroot(int x) { // 使得p成为原树的根
      access(x);
      splay(x);
      swap(ch[x][0], ch[x][1]), rev[x] ^= 1;
   }
   inline int find(int x) { // 找到x在原树的根
      access(x);
      splay(x);
      while (ch[x][0]) x = ch[x][0];
      splay(x); // 非常重要！一定注意！
      return x;
   }

   void split(int x, int y) { // 拉出x-y的路径搞成一个splay
```

```
        makeroot(x);
        access(y);
        splay(y); // y为根, call: tree.sum[y]
    }
    void link(int x, int y) { // 连接x,y
        makeroot(x);
        if (find(y) == x) return;
        fa[x] = y;
        return;
    }
    void cut(int x, int y) { // 断开x,y
        makeroot(x);
        if (find(y) != x || fa[y] != x || ch[y][0]) return; // 两条不连通
        ch[x][1] = fa[y] = 0;
        push_up(x);
        return ;
    }
    void change(int x, int v) { // 修改某一点的值
        splay(x);
        val[x] = v;
        push_up(x);
    }
    bool isconnect(int x, int y) { // 判断两点是否连通
        makeroot(x);
        if (find(y) != x) return 0; // 两条不连通
        else return 1;
    }
} tree;
```

1.1.2 树上路径染色

```
// 颜色段的定义是极长的连续相同颜色被认为是一段。例如112221由三段组成：11、222、1。
class LCT { public:
    int st_top, st[MAXN];
    int fa[MAXN], ch[MAXN][2];
    bool rev[MAXN];
    int col[MAXN], colL[MAXN], colR[MAXN], sum[MAXN];
    int lazy[MAXN];

    inline void push_up(int x) {
        int l = ch[x][0], r = ch[x][1];
        colL[x] = l ? colL[l] : col[x];
        colR[x] = r ? colR[r] : col[x];
        if (l && r) sum[x] = sum[l] + sum[r] + 1 - (colR[l] == col[x])
            - (colL[r] == col[x]);
        if (l && !r) sum[x] = sum[l] + 1 - (colR[l] == col[x]);
        if (!l && r) sum[x] = sum[r] + 1 - (colL[r] == col[x]);
        if (!l && !r) sum[x] = 1;
    }

    inline void push_down(int x) {
        int l = ch[x][0], r = ch[x][1];
        if (rev[x]) {
            if (l) swap(ch[l][0], ch[l][1]), swap(colL[l], colR[l]), rev
                [l] ^= 1;
            if (r) swap(ch[r][0], ch[r][1]), swap(colL[r], colR[r]), rev
                [r] ^= 1;
            rev[x] = 0;
        }
        if (lazy[x]) {
            if (l) colL[l] = colR[l] = col[l] = lazy[x], sum[l] = 1,
                lazy[l] = lazy[x];
            if (r) colL[r] = colR[r] = col[r] = lazy[x], sum[r] = 1,
                lazy[r] = lazy[x];
            lazy[x] = 0;
        }
    }

    inline void makeroot(int x) { // 使得p成为原树的根
        access(x);
        splay(x);
        swap(ch[x][0], ch[x][1]), swap(colL[x], colR[x]), rev[x] ^= 1;
    }
```

```
} tree;

int main() {
    int n, q; cin >> n >> q;
    for (int i = 1; i <= n; i++) {
        cin >> tree.col[i];
        tree.colL[i] = tree.colR[i] = tree.col[i];
        tree.sum[i] = 1;
    }
    for (int i = 2; i <= n; i++) {
        int u, v; cin >> u >> v;
        tree.link(u, v);
    }
    while (q--) {
        char opt; cin >> opt;
        if (opt == 'C') { // 将节点u到节点v的路径上的所有点（包括u和v）都染成颜色c
            int u, v, c; cin >> u >> v >> c;
            tree.split(u, v);
            tree.colL[v] = tree.colR[v] = tree.col[v] = c, tree.sum[v] = 1, tree.lazy[v] = c;
        } else { // 询问节点u到节点v的路径上的颜色段数量
            int u, v; cin >> u >> v;
            tree.split(u, v);
            printf("%d\n", tree.sum[v]);
        }
    }
}
```

1.1.3 离线统计区间本质不同字串个数

时间复杂度：$O(n \log^2 n + m \log n)$.

```
/*
    input output
    aababc
    3
    1 2 2
    2 4 5
    3 6 9
*/
namespace SAM_SEG_LCT {
    SEG seg, SAM sam; // 后缀自动机长度为两倍
    class LCT { public:
        int val[MAXN], lazy[MAXN];
        int st_top, st[MAXN]; // stack操作
        int fa[MAXN], ch[MAXN][2];

        inline bool isroot(int x) { // 判断x是否为一个splay的根
            return ch[fa[x]][0] != x && ch[fa[x]][1] != x;
        }
        inline void push_up(int x) { }
        inline void push_down(int x) {
            int l = ch[x][0], r = ch[x][1];
            if (lazy[x]) {
                if (l) val[l] = lazy[x], lazy[l] = lazy[x];
                if (r) val[r] = lazy[x], lazy[r] = lazy[x];
                lazy[x] = 0;
            }
        }
        inline void rotate(int x) { // x向上旋转
            int y = fa[x], z = fa[y], l, r;
            if (ch[y][0] == x) l = 0;
            else l = 1;
            r = l ^ 1;
            if (!isroot(y)) {
                if (ch[z][0] == y) ch[z][0] = x;
                else ch[z][1] = x;
            }
            fa[x] = z, fa[y] = x;
            fa[ch[x][r]] = y;
            ch[y][l] = ch[x][r];
            ch[x][r] = y;
            push_up(y), push_up(x);
```

```
    }
    inline void splay(int x) { // 使得x成为当前splay中的根
        st_top = 0;
        st[++st_top] = x;
        for (int i = x; !isroot(i); i = fa[i]) st[++st_top] = fa[i];
        for (int i = st_top; i; i--) push_down(st[i]);
        while (!isroot(x)) {
            int y = fa[x], z = fa[y];
            if (!isroot(y)) {
                if (ch[y][0] == x ^ ch[z][0] == y) rotate(x);
                else rotate(y);
            }
            rotate(x);
        }
    }

    void access(int x, int p) { //把x到根节点的路径搞成一个splay，主要是
        这里修改
        int y = 0;
        while (x) {
            splay(x);
            if (int k = val[x]) seg.change(1, k - sam.maxlen[x] + 1,
                k - sam.maxlen[fa[x]], -1);
            ch[x][1] = y, y = x, x = fa[x];
        }
        val[y] = p, lazy[y] = p;
        seg.change(1, 1, p, 1); // 线段树区间修改chang(rt, L, R, val);
    }

    void build() {
        st_top = 0;
        fa[1] = ch[1][0] = ch[1][1] = val[1] = lazy[1] = 0;
        fa[0] = ch[0][0] = ch[0][1] = val[0] = lazy[0] = 0;
        for (int i = 2; i <= sam.rt; i++) {
            val[i] = lazy[i] = 0;
            ch[i][0] = ch[i][1] = 0;
            fa[i] = sam.link[i];
        }
    }
} lct;
}

using namespace SAM_SEG_LCT;
struct Query {
    int l, r, id;
    bool operator<(const Query &tb) const { return r < tb.r;}
} query[MAXN];

char str[MAXN]; int endpos[MAXN]; ll res[MAXN];
int main() {
    scanf("%s", str + 1); int len = strlen(str + 1);
    sam.init(); int sam_last = 1;
    for (int i = 1; i <= len; i++) sam_last = endpos[i] = sam.insert(
        str[i] - 'a', sam_last);
    lct.build(), seg.build(1, 1, len);

    int q; scanf("%d", &q);
    for (int i = 1; i <= q; i++) scanf("%d%d", &query[i].l, &query[i].r
        ), query[i].id = i;
    sort(query + 1, query + 1 + q);

    int pos = 1;
    for (int i = 1; i <= q; i++) {
        while (pos <= query[i].r) lct.access(endpos[pos], pos), pos++;
        res[query[i].id] = seg.query(1, query[i].l, query[i].r);
    }
    for (int i = 1; i <= q; i++) printf("%lld\n", res[i]); // 输出答案
}
```

1.1.4 在线查询边的区间内连通块个数

时间复杂度：$O(m \log n + q \log m)$.

```
struct Edge {
    int u, v;
```

```
} e[MAXM];

class LCT { public:
   int val[MAXN+MAXM], minn_id[MAXN+MAXM];
   int stk_top, stk[MAXN+MAXM];
   int fa[MAXN+MAXM], ch[MAXN+MAXM][2];
   bool rev[MAXN+MAXM];

   inline bool isroot(int x) {
      return ch[fa[x]][0] != x && ch[fa[x]][1] != x;
   }
   inline void push_up(int x) {
      int l = ch[x][0], r = ch[x][1];
      minn_id[x] = x;
      if (val[minn_id[l]] < val[minn_id[x]]) minn_id[x] = minn_id[l];
      if (val[minn_id[r]] < val[minn_id[x]]) minn_id[x] = minn_id[r];
   }
   inline void split(int x, int y) {
      makeroot(x);
      access(y);
      splay(y);
   }

   int query(int x, int y) {
      split(x, y);
      return minn_id[y];
   }
} lct;

class HJT { public:
   int ch[MAXM * 70][2], sum[MAXM * 70];
   int tot;
   inline void push_up(int rt) {
      sum[rt] = sum[ch[rt][0]] + sum[ch[rt][1]];
   }
   int query(int lrt, int rrt, int L, int R, int be, int en) {
      if (L <= be && en <= R) return sum[rrt] - sum[lrt];
      int mid = (be + en) >> 1;
      int ans = 0;
      if (L <= mid) ans += query(ch[lrt][0], ch[rrt][0], L, R, be,
          mid);
      if (R > mid) ans += query(ch[lrt][1], ch[rrt][1], L, R, mid+1,
          en);
      return ans;
   }
} tree;

int del[MAXM], root[MAXM];
int main() {
   int n, m, q, type;
   scanf("%d%d%d%d", &n, &m, &q, &type); // type标识在线参数

   lct.val[0] = inf; // init
   for (int i = 1; i <= n; i++) lct.minn_id[i] = i, lct.val[i] = inf;
   for (int i = 1; i <= m; i++) {
      scanf("%d%d", &e[i].u, &e[i].v);
   }

   // pre begin
   int tot = n;
   for (int i = 1; i <= m; i++) {
      int u = e[i].u, v = e[i].v;
      if (u == v) {
         del[i] = i; continue;
      }
      if (lct.find(u) == lct.find(v)) {
         int tmp = lct.query(u, v), x = lct.val[tmp];
         del[i] = x;
         lct.cut(e[x].u, tmp), lct.cut(e[x].v, tmp);
      }
      tot++;
      lct.minn_id[tot] = tot, lct.val[tot] = i;
      lct.link(u, tot), lct.link(v, tot);
   }
```

```
    root[0] = 0;
    for (int i = 1; i <= m; i++) {
        del[i]++; // [0, m] -> [1, m+1]
        root[i] = tree.update(root[i - 1], del[i], 1, 1, m + 1);
    }
    // pre end
    int lastans = 0;
    while (q--) {
        int l, r;
        scanf("%d%d", &l, &r);
        lastans = n - tree.query(root[l-1], root[r], 1, l, 1, m+1);
        printf("%d\n", lastans);
    }
}
```

1.2 有旋 Treap & 无旋 Treap

1.2.1 带跳 fa 的 Treap（[ZJOI2006] 书架）

```
class FHQ { public:
    int ch[MAXN][2];
    int val[MAXN], dat[MAXN], siz[MAXN], pos[MAXN], fa[MAXN];
    int tot, root;
    void init() {
        root = 1, tot = 0, val[0] = siz[0] = 0;
        fa[1] = 0;
    }
    int New(int v) {
        val[++tot] = v, dat[tot] = rand(), siz[tot] = 1, fa[tot] = 0;
        ch[tot][0] = ch[tot][1] = 0;
        pos[v] = tot; // 值所在的位置，用于跳fa
        return tot;
    }
    inline void push_up(int rt) {
        siz[rt] = siz[ch[rt][0]] + siz[ch[rt][1]] + 1;
        if (ch[rt][0]) fa[ch[rt][0]] = rt;
        if (ch[rt][1]) fa[ch[rt][1]] = rt;
    }
    int build(int l, int r) {
        if (l > r) return 0;
        int mid = (l + r) >> 1;
        int newnode = New(b[mid]);
        ch[newnode][0] = build(l, mid - 1);
        ch[newnode][1] = build(mid + 1, r);
        push_up(newnode);
        return newnode;
    }
    void split_id(int rt, int k, int &x, int &y) {
        if (!rt) x = y = 0;
        else {
            if (k <= siz[ch[rt][0]]) {
                y = rt;
                split_id(ch[rt][0], k, x, ch[rt][0]);
            } else {
                x = rt;
                split_id(ch[rt][1], k - siz[ch[rt][0]] - 1, ch[rt][1], y)
                    ;
            }
            push_up(rt);
        }
    }
    int merge(int x, int y) {
        if (!x || !y) return x + y;
        if (dat[x] < dat[y]) {
            ch[x][1] = merge(ch[x][1], y);
            push_up(x);
            return x;
        } else {
            ch[y][0] = merge(x, ch[y][0]);
            push_up(y);
            return y;
        }
    }
    int get_pos(int v) {
```

```
      int rt = pos[v];
      int ans = 1 + siz[ch[rt][0]];
      while (fa[rt] && rt!= root) {
         int f = fa[rt];
         if (ch[f][1] == rt) ans += 1 + siz[ch[f][0]];
         rt = f;
      }
      return ans;
   }
} tree;
```

1.2.2 并查集 + 启发式合并（HDU3726）

```
class Treap { public:
   int ch[MAXN][2], dat[MAXN], siz[MAXN], val[MAXN], cnt[MAXN];
   int tot;
   int pool[MAXN], pool_cnt;
   void init() { tot = 0, pool_cnt = 0; }
   inline int Newid() {
      return pool_cnt ? pool[pool_cnt--] : ++tot;
   }
   inline void Delid(int &rt) {
      if (!rt) return;
      pool[++pool_cnt] = rt;
      dat[rt] = siz[rt] = val[rt] = cnt[rt] = 0;
      ch[rt][0] = ch[rt][1] = val[rt] = 0;
      rt = 0;
   }
   inline int Newnode(int v, int _cnt = 1) {
      int nid = Newid();
      val[nid] = v, dat[nid] = rand(), siz[nid] = _cnt, cnt[nid] =
          _cnt;
      ch[nid][0] = ch[nid][1] = 0;
      return nid;
   }
   inline void push_up(int rt) {
      siz[rt] = siz[ch[rt][0]] + siz[ch[rt][1]] + cnt[rt];
   }
   inline void Rotate(int &rt, int d) {
      int temp = ch[rt][d ^ 1];
      ch[rt][d ^ 1] = ch[temp][d];
      ch[temp][d] = rt;
      rt = temp;
      push_up(ch[rt][d]), push_up(rt);
   }

   void insert(int &rt, int v, int _cnt = 1) {
      if (!rt) {
         rt = Newnode(v, _cnt);
         return;
      }
      if (v == val[rt]) cnt[rt] += _cnt;
      else {
         int d = v < val[rt] ? 0 : 1;
         insert(ch[rt][d], v, _cnt);
         if (dat[rt] < dat[ch[rt][d]]) Rotate(rt, d ^ 1);
      }
      push_up(rt);
   }
   void remove(int &rt, int v) {
      if (!rt) return;
      if (v == val[rt]) {
         if (cnt[rt] > 1) {
            cnt[rt]--, push_up(rt);
            return;
         }
         if (ch[rt][0] || ch[rt][1]) {
            if (!ch[rt][1] || dat[ch[rt][0]] > dat[ch[rt][1]]) {
               Rotate(rt, 1), remove(ch[rt][1], v);
            } else {
               Rotate(rt, 0), remove(ch[rt][0], v);
            }
            push_up(rt);
         } else Delid(rt);
```

```
            return;
        }
        v < val[rt] ? remove(ch[rt][0], v) : remove(ch[rt][1], v);
        push_up(rt);
    }

    int Kth(int rt, int k) { // call:tree.Kth(root[find(x)], k); 与x相连的第k大值
        if (!rt) return 0;
        if (k <= siz[ch[rt][1]]) return Kth(ch[rt][1], k);
        else if (k <= siz[ch[rt][1]] + cnt[rt]) return val[rt];
        else return Kth(ch[rt][0], k - siz[ch[rt][1]] - cnt[rt]);
    }
    void merge(int &x, int &y) {
        if (ch[x][0]) merge(ch[x][0], y);
        if (ch[x][1]) merge(ch[x][1], y);
        if (cnt[x] > 0) insert(y, val[x], cnt[x]);
        Delid(x);
    }
} tree;

int F[MAXN];
int root[MAXN];

int find(int x) {
    if (F[x] == x) return x;
    else return F[x] = find(F[x]);
}
void join(int u, int v) { // 两个点相连
    int fu = find(u), fv = find(v);
    if (fu != fv) {
        if (tree.siz[fu] < tree.siz[fv]) F[fu] = fv, tree.merge(root[fu], root[fv]);
        else F[fv] = fu, tree.merge(root[fv], root[fu]);
    }
}

void change(int x, int v, int oldv) { // 将点x的值从oldv -> v
    int fx = find(x);
    tree.remove(root[fx], oldv);
    tree.insert(root[fx], v);
}
```

1.3 李超线段树

1.3.1 多条线段定点最值（[HEOI2013]Segment）

要求在平面直角坐标系下维护两个操作（强制在线）：

1 x0 y0 x1 y1 在平面上加入一条线段。

0 x 给定一个数，询问与直线 $y = x$ 相交的线段中，交点纵坐标最大的线段的编号（若有多条线段与查询直线的交点纵坐标都是最大的，则输出编号最小的线段）。特别地，若不存在线段与给定直线相交，输出 0。

```
/*
 input output
 6
 1 8 5 10 8
 1 6 7 2 6
 0 2 2
 0 11 0
 1 4 7 6 7
 0 5 3
*/
class LC { public:
    struct LINE {
        double k, b;
    } p[MAXN];
    int sum[MAXN << 2];
    int cnt = 0;
    void init() { cnt = 0; }
    void add_seg(int x0, int y0, int x1, int y1) {
        cnt++;
        if (x0 == x1) p[cnt].k = 0, p[cnt].b = max(y0, y1);
        else p[cnt].k = 1.0 * (y1 - y0) / (x1 - x0), p[cnt].b = y0 - p[cnt].k * x0;
```

```
    }
    void update(int rt, int L, int R, int be, int en, int u) {
        int mid = (be + en) >> 1;
        if (L <= be && en <= R) {
            int v = sum[rt];
            double ansu = p[u].b + p[u].k * mid, ansv = p[v].b + p[v].k
                * mid;
            if (be == en) {
                if (ansu > ansv) sum[rt] = u;
                return;
            }
            if (p[v].k < p[u].k) {
                if (ansu > ansv) {
                    sum[rt] = u;
                    update(rt << 1, L, R, be, mid, v);
                } else update(rt << 1 | 1, L, R, mid + 1, en, u);
            } else if (p[v].k > p[u].k) {
                if (ansu > ansv) {
                    sum[rt] = u;
                    update(rt << 1 | 1, L, R, mid + 1, en, v);
                } else update(rt << 1, L, R, be, mid, u);
            } else {
                if (p[u].b > p[v].b) sum[rt] = u;
            }
            return;
        }
        if (L <= mid) update(rt << 1, L, R, be, mid, u);
        if (R > mid) update(rt << 1 | 1, L, R, mid + 1, en, u);
    }
    typedef pair<double, int> pdi;
    pdi pmax(pdi x, pdi y) {
        if (x.first < y.first) return y;
        else if (x.first > y.first) return x;
        else return x.second < y.second ? x : y;
    }
    pdi query(int rt, int d, int be, int en) {
        if (be == en) {
            int v = sum[rt];
            double ans = (p[v].b + p[v].k * d);
            return {ans, sum[rt]};
        }
        int mid = (be + en) >> 1;
        int v = sum[rt];
        double res = (p[v].b + p[v].k * d);
        pdi ans = {res, sum[rt]};
        if (d <= mid) return pmax(ans, query(rt << 1, d, be, mid));
        else return pmax(ans, query(rt << 1 | 1, d, mid + 1, en));
    }
} tree;
int main() {
    int n; scanf("%d", &n);
    while (n--) {
        int opt; scanf("%d", &opt);
        if (opt == 1) {
            int x0, y0, x1, y1; scanf("%d%d%d%d", &x0, &y0, &x1, &y1);
            if (x0 > x1) swap(x0, x1), swap(y0, y1); // notice x0, x1
            tree.add_seg(x0, y0, x1, y1);
            tree.update(1, x0, x1, 1, MOD1, tree.cnt);
        } else {
            int x; scanf("%d", &x);
            printf("%d\n", tree.query(1, x, 1, MOD1).second);
        }
    }
}
```

1.4 YNOI 系列

1.4.1 带修查询能否连续重排为值域连续的序列（线段树 + 散列异或）（洛谷 P3792）

1 x y 修改 x 位置的值为 y

2 l r 查询区间 $[l,r]$ 是否可以重排为值域上连续的一段

```
struct Query {
    int opt, x, y;
```

```
} q[MAXN];
ull rnd[MAXN<<2], pre_rnd[MAXN<<2]; // 空间注意!
int a[MAXN];

class SEG { public:
   struct node {
      int l, r, minn;
      ull sum;
   } T[MAXN << 2];

   inline void push_up(int rt) {
      T[rt].minn = min(T[rt << 1].minn, T[rt << 1 | 1].minn);
      T[rt].sum = T[rt << 1].sum ^ T[rt << 1 | 1].sum;
   }

   void build(int rt, int l, int r) {
      T[rt].l = l, T[rt].r = r;
      if (l == r) {
         T[rt].minn = a[l];
         T[rt].sum = rnd[T[rt].minn];
         return;
      }
      int mid = (l + r) >> 1;
      build(rt << 1, l, mid), build(rt << 1 | 1, mid + 1, r);
      push_up(rt);
   }

   void update(int rt, int pos, int val) {
      if (T[rt].l == T[rt].r) {
         T[rt].minn = val;
         T[rt].sum = rnd[T[rt].minn];
         return;
      }
      int mid = (T[rt].l + T[rt].r) >> 1;
      if (pos <= mid) update(rt << 1, pos, val);
      else update(rt << 1 | 1, pos, val);
      push_up(rt);
   }

   ull query_sum(int rt, int L, int R) {
      if (L <= T[rt].l && T[rt].r <= R) return T[rt].sum;
      int mid = (T[rt].l + T[rt].r) >> 1;
      ull ans = 0;
      if (L <= mid) ans ^= query_sum(rt << 1, L, R);
      if (R > mid) ans ^= query_sum(rt << 1 | 1, L, R);
      return ans;
   }

   int query_min(int rt, int L, int R) {
      if (L <= T[rt].l && T[rt].r <= R) return T[rt].minn;
      int mid = (T[rt].l + T[rt].r) >> 1;
      int ans = inf;
      if (L <= mid) ans = min(ans, query_min(rt << 1, L, R));
      if (R > mid) ans = min(ans, query_min(rt << 1 | 1, L, R));
      return ans;
   }
} tree;

int main() {
   srand(19260817);
   int n, m; scanf("%d%d", &n, &m);
   for (int i = 1; i <= n; i++) {
      scanf("%d", &a[i]);
      Discrete::insert(a[i]), Discrete::insert(a[i] + 1);
   }

   for (int i = 1; i <= m; i++) {
      scanf("%d%d%d", &q[i].opt, &q[i].x, &q[i].y);
      if (q[i].opt == 1) Discrete::insert(q[i].y), Discrete::insert(q
          [i].y + 1);
   }

   Discrete::init();
   for (int i = 1; i <= n; i++) {
```

```
        a[i] = val2id(a[i]);
    }

    pre_rnd[0] = 0;
    for (int i = 1; i < (MAXN<<2); i++) rnd[i] = Newrnd(), pre_rnd[i]
        = pre_rnd[i - 1] ^ rnd[i];

    tree.build(1, 1, n);
    for (int i = 1; i <= m; i++) {
        if (q[i].opt == 1) {
            tree.update(1, q[i].x, val2id(q[i].y));
        } else {
            int l = tree.query_min(1, q[i].x, q[i].y);
            int r = l + (q[i].y - q[i].x);
            ull tmp1 = pre_rnd[r] ^ pre_rnd[l-1];
            ull tmp2 = tree.query_sum(1, q[i].x, q[i].y);
            if (tmp1 == tmp2) printf("damushen\n"); // 能够重排为值域上连续的
                一段
            else printf("yuanxing\n"); // 不能够重排为值域上连续的一段
        }
    }
}
```

2 字符串

2.1 后缀连接字典序最小

```
/*
    input ouput
    3 1 3 2
    arc
    2
    zz 1 2 / 2 1
    5
    abaab 3 1 4 2 5
*/
using RMQ::get_LCP; using SA::rk;
int num[MAXN];
int N; char str[MAXN];
bool cmp(int x, int y) {
    if (x < y) {
        int lcp = get_LCP(rk[x], rk[y]);
        if (lcp < N - y + 1) { // part 1
            if (str[x + lcp] < str[y + lcp]) return 1;
            else return 0;
        }
        lcp = get_LCP(rk[x + (N - y + 1)], rk[x]);
        if (str[x + (N - y + 1) + lcp] < str[x + lcp]) return 1;
        else return 0;
    } else {
        int lcp = get_LCP(rk[x], rk[y]);
        if (lcp < N - x+ 1) { // part 1
            if (str[x + lcp] < str[y + lcp]) return 1;
            else return 0;
        }
        lcp = get_LCP(rk[y + (N - x + 1)], rk[y]);
        if (str[y + lcp] < str[y+ (N - x + 1) + lcp]) return 1;
        else return 0;
    }
}
int main() {
    scanf("%d", &N); scanf("%s", str + 1);
    SA::run(str, N); SA::get_height(str);
    RMQ::init(N);
    for (int i = 1; i <= N; i++) num[i] = i;
    sort(num+1, num+1+N, cmp);
    for (int i = 1; i <= N; i++) {
        printf("%d\n", num[i]);
    }
}
```

3 动态规划

3.1 #2 字符串 T 在字符串 S 子序列出现的次数

```
/*
  input:S = "babgbag", T = "bag"
  output:5
*/
ll dp[10][MAXN];
ll fun(const string &S, int slen, const string &T, int tlen) {
    for (int i = 0; i <= tlen + 1; i++) dp[i][0] = 0;
    for (int i = 0; i <= slen + 1; i++) dp[0][i] = 1;
    for (int i = 1; i <= tlen + 1; i++) {
        for (int j = 1; j <= slen + 1; j++) {
            if (T[i - 1] == S[j - 1]) dp[i][j] = (dp[i - 1][j - 1] + dp
                [i][j - 1]) % mod;
            else dp[i][j] = dp[i][j - 1];
        }
    }
    return dp[tlen][slen];
}
```

3.2 #3 N 种长度为 1 元素填充 L

```
/*
  input N = 3, L = 3, K = 1
  output 6 [1, 2, 3], [1, 3, 2], [2, 1, 3], [2, 3, 1], [3, 1, 2], [3, 2,
      1].
  input N = 2, L = 3, K = 0
  output 6 [1, 1, 2], [1, 2, 1], [2, 1, 1], [2, 2, 1], [2, 1, 2], [1, 2,
      2]
  input N = 2, L = 3, K = 1
  output [1, 2, 1], [2, 1, 2]
*/
vector<vector<long long>> dp;
// 空间复杂度O（NL）
int numMusicPlaylists(int N, int L, int K) {
    dp.resize(L + 1);
    for (int i = 0; i <= L; i++) dp[i].resize(N + 1);
    dp[0][0] = 1;
    for (int i = 1; i <= L; i++) {
        for (int j = 1; j <= N; j++) {
            dp[i][j] = (dp[i][j] + dp[i - 1][j - 1] * (long long) (N -
                j + 1)) % mod;
            dp[i][j] = (dp[i][j] + dp[i-1][j] * max(j - K, 0)) % mod;
        }
    }
    return dp[L][N];
}
// 空间复杂度O（L）
int numMusicPlaylists(int N, int L, int K) {
    dp.resize(2);
    for (int i = 0; i < 2; i++) dp[i].resize(N + 1);
    for (int i = 0; i < 2; i++) {
        for (int j = 0; j <= N; j++) dp[i][j] = 0;
    }
    dp[0][0] = 1;
    int flag = 0;
    for (int i = 1; i <= L; i++) {
        dp[!flag][0] = 0;
        for (int j = 1; j <= N; j++) {
            dp[!flag][j] = 0;
            dp[!flag][j] = (dp[!flag][j] + dp[flag][j - 1] * (long long)
                 (N - j + 1)) % mod;
            dp[!flag][j] = (dp[!flag][j] + dp[flag][j] * max(j - K, 0))
                % mod;
        }
        flag = !flag;
    }
    return dp[flag][N];
}
```

3.3 #4 分割数组

```cpp
/*
  nums = [7,2,5,10,8], m = 2
  res = 18
*/
vector<int> dp, sum;
int splitArray(vector<int> &nums, int m) {
    int n = (int) nums.size();
    dp.resize(n), sum.resize(n);
    sum[0] = nums[0];
    for (int i = 1; i < n; i++) sum[i] = sum[i - 1] + nums[i];
    for (int i = 0; i < n; i++) dp[i] = sum[i];
    for (int i = 2; i <= m; i++) {
        int pos = n - 1;
        for (int j = n - 1; j >= 1; j--) {
            while (pos >= 0 && dp[pos] >= sum[j] - sum[pos]) {
                pos--;
            }
            if (pos >= 0) dp[j] = min(dp[j], max(dp[pos], sum[j] - sum[
                pos]));
            if (pos < n - 1) dp[j] = min(dp[j], max(dp[pos + 1], sum[j]
                - sum[pos + 1]));
        }
    }
    return dp[n - 1];
}
```

3.4 #5 划分为 K 个相等的子集

```cpp
/*
  nums = [4, 3, 2, 3, 5, 2, 1], k = 4
  return true
*/
bool canPartitionKSubsets(vector<int> &nums, int k) { // 调用函数
    int sum = 0;
    for (int i = 0; i < nums.size(); ++i) {
        sum += nums[i];
    }
    if (sum % k != 0) return false; //不能被整除,返回
    int target = sum / k;
    sort(nums.begin(), nums.end(), greater<int>());//从大到小排序,在计算
        dfs中的sum时,从最大的元素开始累加可以减少递归的次数
    if (target < nums[0]) return false;//不加这一句会超时
    vector<int> mark(nums.size(), 0);//标记数组,标记该元素已经被使用过,减少
        重复次数
    return dfs(nums, mark, 0, k, target);
}

bool dfs(vector<int> &nums, vector<int> &mark, int sum, int k, int
    target) {
    if (k == 0) return true;//找到了k个子集
    if (sum == target) return dfs(nums, mark, 0, k - 1, target);//找到
        了一个和为target的子集,sum置0,k减去一
    for (int i = 0; i < nums.size(); ++i) {//依次循环计算sum
        if (mark[i]) continue;//如果该元素已经被其他子集占有,则直接跳过
        if (sum > target) return false;//sum比target大,则直接返回false
        //将该元素标记为使用过
        mark[i] = 1;
        if (dfs(nums, mark, sum + nums[i], k, target)) return true;//一
            直递归知道找到k个子集时,返会true
        mark[i] = 0;//回溯后,该元素不符合要求,将该元素的使用标志置为0
    }
    return false;//遍历完整个数组都不能找到符合要求的子集
}
```

4 杂项

4.1 散列处理异或碰撞

常用于处理判断出现偶数次（HDU6291），重排后为值域上的连续一段（洛谷 P3792、YNOI）。

```cpp
ull Newrnd() {
```

```
    return ((ull) rand() << 45) | ((ull) rand() << 30) | (rand() <<
        15) | rand();
}
int main() {
    srand(114514); // random
}
```

4.2 fread

```
char buf[100000], *p1 = buf, *p2 = buf;
inline char nc() {
    return p1 == p2 && (p2 = (p1 = buf) + fread(buf, 1, 100000, stdin)
        , p1 == p2) ? EOF : *p1++;
}
inline bool read(int &x) {
    char c = nc(); x = 0;
    if (c == EOF) return false;
    for (; !isdigit(c); c = nc());
    for (; isdigit(c); x = x * 10 + c - '0', c = nc());
    return true;
}
```

5 习题整理

5.1 图上加边最多最少连通块（线段树二分贪心）（ZOJ4100）

时间复杂度：$O(q \log^2 n)$.

```
/*
    input ouput
    5 5
    1 1 2
    2 1 3 3
    1 1 3
    2 1 2 3
    2 3 1 2
*/
ll com[MAXN];
int n;
class SEG { public:
    struct node {
        int l, r, cnt, sum;
        ll edge;
    } T[MAXN << 2];
    inline void push_up(int rt) {
        T[rt].cnt = T[rt << 1].cnt + T[rt << 1 | 1].cnt;
        T[rt].edge = T[rt << 1].edge + T[rt << 1 | 1].edge;
        T[rt].sum = T[rt << 1].sum + T[rt << 1 | 1].sum;
    }

    void build(int rt, int l, int r) {
        T[rt].l = l, T[rt].r = r;
        if (l == r) {
            if (l == 1) {
                T[rt].edge = com[l] * n, T[rt].cnt = n, T[rt].sum = n;
            } else {
                T[rt].edge = 0, T[rt].cnt = 0, T[rt].sum = 0;
            }
            return;
        }
        int mid = (l + r) >> 1;
        build(rt << 1, l, mid), build(rt << 1 | 1, mid + 1, r);
        push_up(rt);
    }
    void update(int rt, int pos, int v) {
        if (T[rt].l == T[rt].r) {
            T[rt].cnt += v;
            T[rt].sum += v * T[rt].l;
            T[rt].edge += com[T[rt].l] * v;
            return;
        }
        int mid = (T[rt].l + T[rt].r) >> 1;
        if (pos <= mid) update(rt << 1, pos, v);
```

```
        else update(rt << 1 | 1, pos, v);
        push_up(rt);
    }
    int query(int rt, ll k, int vs, ll vk) { // 二分查找
        if (T[rt].l == T[rt].r) {
            int L = 0, R = T[rt].cnt;
            while (L < R) {
                int mid = (L + R) >> 1;
                if (com[mid * T[rt].l + vs] - mid * com[T[rt].l] - vk < k
                    ) L = mid + 1;
                else R = mid;
            }
            return L;
        }
        if (com[vs + T[rt << 1 | 1].sum] - T[rt << 1 | 1].edge >= k)
            return query(rt << 1 | 1, k, vs, vk);
        else return T[rt << 1 | 1].cnt + query(rt << 1, k, vs + T[rt <<
            1 | 1].sum, vk + T[rt << 1 | 1].edge);
    }
} tree;
int cnt[MAXN]; ll edge[MAXN];
int fa[MAXN];
int find(int x) {
    if (fa[x] == x) return x;
    else return fa[x] = find(fa[x]);
}
int main() {
    int T; scanf("%d", &T);
    for (int i = 1; i < MAXN; i++) com[i] = (ll) i * (i - 1) / 2;
    while (T--) {
        scanf("%d", &n);
        for (int i = 1; i <= n; i++) fa[i] = i;
        for (int i = 1; i <= n; i++) cnt[i] = 1, edge[i] = 0;
        tree.build(1, 1, n);
        int q; scanf("%d", &q);
        int blocks = n;
        ll free = 0;
        while (q--) {
            int opt; scanf("%d", &opt);
            if (opt == 1) {
                int x, y; scanf("%d%d", &x, &y);
                int fx = find(x), fy = find(y);
                if (fx == fy) {
                    edge[fx]++; free--;
                } else {
                    blocks--;
                    free -= com[cnt[fx]] - edge[fx];
                    free -= com[cnt[fy]] - edge[fy];
                    tree.update(1, cnt[fx], -1);
                    tree.update(1, cnt[fy], -1);
                    fa[fx] = fy;
                    cnt[fy] += cnt[fx];
                    edge[fy] += edge[fx] + 1;
                    free += com[cnt[fy]] - edge[fy];
                    tree.update(1, cnt[fy], 1);
                }
            } else {
                ll k;
                scanf("%lld", &k);
                printf("%lld ", max(1ll, (ll) blocks - k)); // minn
                if (free >= k) printf("%d\n", blocks); // maxx
                else {
                    k -= free;
                    int tmp = tree.query(1, k, 0, 0);
                    printf("%d\n", blocks - tmp + 1); // maxx
                }
            }
        }
    }
}
```

5.2 错排后字典序最小（ZOJ4102）

时间复杂度：$O(n \log n)$.

```cpp
/*
   input output
   4 1 3 2 1 2 4 3
   1 1 2 3 2 3 1 1
   1 1 1 Impossible
*/
int a[MAXN], ban[MAXN], todo[MAXN];
int res[MAXN]; // 字典序最小的存在res中
int main() {
   priority_queue<pii > pq;
   vector<int> vec;
   int n; scanf("%d", &n);
   for (int i = 1; i <= n; i++) scanf("%d", &a[i]);
   for (int i = 1; i <= n; i++) ban[i] = todo[i] = 0;

   int flag = 1;
   for (int i = 1; i <= n; i++) ban[a[i]]++, todo[a[i]]++;
   for (int i = 1; i <= n; i++) {
      if (ban[i]) {
         vec.push_back(i);
         pq.push(mp(ban[i] + todo[i], i));
         if (ban[i] + todo[i] > n) {
            flag = 0; break;
         }
      }
   }
   if (!flag) { printf("Impossible\n"); continue; }

   int pos = 0;
   for (int i = 1; i <= n; i++) {
      pii u = pq.top(); pq.pop();
      while (u.first != ban[u.second] + todo[u.second]) u = pq.top(),
          pq.pop();
      if (u.first == n - i + 1 && u.second != a[i]) {
         res[i] = u.second;
      } else {
         pq.push(u);
         for (int j = pos; j < SZ(vec); j++) {
            if (vec[j] != a[i] && todo[vec[j]] > 0) {
               res[i] = vec[j]; break;
            }
         }
      }
      todo[res[i]]--; ban[a[i]]--;
      pq.push(mp(todo[res[i]] + ban[res[i]], res[i]));
      pq.push(mp(todo[a[i]] + ban[a[i]], a[i]));
      if (todo[vec[pos]] == 0) pos++;
   }
}
```

5.3 若干个区间选数字使相与之和最小（ZOJ4135）

```cpp
/*
   input output
   3 6
   [0,8],[2,6],[3,9]
   3 1
   [0,7],[0,3],[4,5]
*/
int L[MAXN], R[MAXN];
int main() {
   int n; scanf("%d", &n);
   for (int i = 1; i <= n; i++) scanf("%d%d", &L[i], &R[i]);
   int res = 0;
   for (int i = 30; i >= 0; i--) {
      int tmp = (1 << i);
      int flag = 1;
      for (int j = 1; j <= n; j++) {
         if (tmp > R[j]) {
            flag = 0; break;
         }
      }
      if (flag) { // can provide 1
         res += tmp;
```

```
            for (int j = 1; j <= n; j++) {
                L[j] = max(L[j], tmp) - tmp;
                R[j] = R[j] - tmp;
            }
        } else {
            for (int j = 1; j <= n; j++) {
                if (L[j] >= tmp && R[j] >= tmp) {
                    L[j] -= tmp, R[j] -= tmp;
                } else if (L[j] < tmp && R[j] >= tmp) {
                    R[j] = tmp - 1;
                }
            }
        }
    }
    printf("%d\n", res);
}
```

5.4 2019 徐州 L

给一颗字符串树，1 为根，求从哪个结点向上 L 长度的字符串共有多少种本质不同的字符串

```
/*
    input output
    6 3
    ABABBA
    1 1 3 3 4
    2 2 3
    2 1 3
    6 4 1
*/
class SAM { public:
    struct Edge {
        int to, nex;
    } e[MAXN];
    int head[MAXN], tol;

    void addEdge(int u, int v) {
        e[tol].to = v, e[tol].nex = head[u], head[u] = tol, tol++;
    }
    void dfs(int u) {
        for (int i = head[u]; ~i; i = e[i].nex) {
            int v = e[i].to;
            dfs(v);
            val[u] += val[v];
        }
    }

    int fa[MAXN][32];
    void build() {
        for (int i = 1; i <= rt; i++) head[i] = -1;
        for (int i = 2; i <= rt; i++) addEdge(link[i], i);
        dfs(1);
        for (int i = 1; i <= rt; i++) fa[i][0] = link[i];
        for (int i = 1; i < 32; i++) {
            for (int j = 1; j <= rt; j++) {
                fa[j][i] = fa[fa[j][i - 1]][i - 1];
            }
        }
    }

    int query(int X, int L) {
        for (int i = 31; ~i; i--) { if (maxlen[fa[X][i]] >= L) X = fa[X
            ][i]; }
        return val[X];
    }

    void debug() {
        for (int i = 1; i <= rt; i++) printf("link[%d] = %d\n", i, link[i
            ]);
        for (int i = 1; i <= rt; i++) printf("val[%d] = %d\n", i, val[i])
            ;
        for (int i = 1; i <= rt; i++) printf("maxlen[%d] = %d\n", i,
            maxlen[i]);
    }
} sa;
```

```

struct Edge {
   int to, nex;
} e[MAXNODE];
int head[MAXNODE], tol;

void addEdge(int u, int v) {
   e[tol].to = v, e[tol].nex = head[u], head[u] = tol, tol++;
}

char str[MAXNODE]; int pos[MAXN];

struct node {
   int v, last;
   node(int _v = 0, int _last = 0) : v(_v), last(_last) {}
};

void bfs() {
   queue<node> q;
   q.push(node(1, 1));
   int last = 1;
   while (!q.empty()) {
      node u = q.front(); q.pop();
      int nls = sa.insert(str[u.v]-'A', u.last);
      pos[u.v] = nls;
      for (int i = head[u.v]; ~i; i = e[i].nex) {
         int to = e[i].to;
         q.push(node(to, nls));
      }
   }
// pos[u] = last = sa.insert(str[u] - 'A', last);
// for (int i = head[u]; ~i; i = e[i].nex) {
// int v = e[i].to;
// dfs(v, last);
// }
}

```

```
void init(int n) { for (int i = 1; i <= n; i++) head[i] = -1; }
int main() {
   int n, q; scanf("%d%d", &n, &q);
   init(n);
   scanf("%s", str + 1);
   for (int i = 2; i <= n; i++) {
      int x; scanf("%d", &x);
      addEdge(x, i);
   }
   bfs();
// dfs(1, 1); // make sam
   sa.build(); // get fail tree
   while (q--) {
      int X, L; scanf("%d%d", &X, &L);
      printf("%d\n", sa.query(pos[X], L));
   }
}
```

6 Java & Python

eclipse 下 ALT+/，自动补全代码。

6.1 Java

```
import java.io.*;
import java.util.*;
import java.math.*;
public class Main {//大数加法
   static Scanner cin = new Scanner(System.in);
   static PrintWriter cout = new PrintWriter(System.out);
   public static void main(String[] args) throws IOException {
      BigInteger a=cin.nextBigInteger(),b...;
      cout.println(a.multiply(b));
      cout.flush();
   }
}
```

```java
public class Main {//排序
   static BufferedReader reader = new BufferedReader(new
       InputStreamReader(System.in), 1 << 16);
   static BufferedWriter writer = new BufferedWriter(new
       OutputStreamWriter(System.out), 1 << 16);
   public static void main(String[] args) throws IOException {
      int n = Integer.parseInt(reader.readLine());
      int[] array = new int[n];
      for(int i = 0; i < n; i++) array[i] = Integer.parseInt(reader.
          readLine());
      Arrays.sort(array);
      for(int i = 0; i < n; i++)
         writer.write(array[i] + "\r\n");
      writer.flush();
   }
}
public class Main {//大数开方
   static BufferedReader reader = new BufferedReader(new
       InputStreamReader(System.in), 1 << 16);
   static BufferedWriter writer = new BufferedWriter(new
       OutputStreamWriter(System.out), 1 << 16);
   public static void main(String[] args) throws Exception {
      writer.write(BigIntSqrt(reader.readLine()) + "\r\n");
      writer.flush();
   }
   public static String BigIntSqrt(String nStr) {
      BigDecimal n = new BigDecimal(nStr);
      BigDecimal ans = new BigDecimal(nStr.substring(0, nStr.length()
          /2+1));
      BigDecimal tmp = BigDecimal.ONE;
      BigDecimal two = new BigDecimal("2");
      int length = 2;
      while (true) {
         tmp = ans.add(n.divide(ans, length, RoundingMode.HALF_DOWN))
             ;
         tmp = tmp.divide(two, length, RoundingMode.HALF_DOWN);
         if (tmp.subtract(ans).abs().compareTo(BigDecimal.ONE) == -1)
             break;
         ans = tmp;
      }
      String str = ans.toString();
      return str.substring(0, str.length() - length - 1);
   }
}
```

```java
public class Main {
   public static BigInteger value0=BigInteger.valueOf(0);
   public static BigInteger value1=BigInteger.valueOf(1);
   public static int MAXN=100005;
   public static void main(String args[]) {
      Scanner cin=new Scanner(System.in);
      int n,m;
      n=cin.nextInt();
      m=cin.nextInt();
      int a[]=new int[MAXN];
      int r[]=new int[MAXN];
      for(int i=1;i<=n;i++) {
         a[i]=cin.nextInt();
         r[i]=cin.nextInt();
      }
      BigInteger ans=excrt(a, r, n);
      System.out.println(ans);
   }
   public static BigInteger[] exgcd(BigInteger a,BigInteger b){
      BigInteger ans;
      BigInteger[] result=new BigInteger[3];
      if(b.equals(value0)){
         result[0]=a;
         result[1]=value1;
         result[2]=value0;
         return result;
      }
      BigInteger [] temp=exgcd(b,a.mod(b));
      ans = temp[0];
```

```
        result[0]=ans;
        result[1]=temp[2];
        result[2]=temp[1].subtract(a.divide(b).multiply(temp[2]));
        return result;
    }
    public static BigInteger excrt(int a[],int r[],int n){
        BigInteger M=BigInteger.valueOf(a[1]),R=BigInteger.valueOf(r
            [1]);
        BigInteger tmp[]=new BigInteger[3];
        for(int i=2;i<=n;i++){
            tmp=exgcd(M,BigInteger.valueOf(a[i]));
            if(!R.subtract(BigInteger.valueOf(r[i])).mod(tmp[0]).equals(
                value0))return BigInteger.valueOf(-1);
            tmp[1]=(R.subtract(BigInteger.valueOf(r[i]))).divide(tmp[0])
                .multiply(tmp[1]).mod(BigInteger.valueOf(a[i]));
            R=R.subtract(M.multiply(tmp[1]));
            M=M.divide(tmp[0]).multiply(BigInteger.valueOf(a[i]));
            R=R.mod(M);
        }
        R=R.mod(M);
        R=R.add(M);
        R=R.mod(M);
        return R;
    }
}
```

6.2 Python

```
#print怎么输出后不换行？
print(待输出,end = '')
#python是允许这样赋值的
a,b,c = 1,2,3
print(a,b,c)
#python玩acm读取输入应该这么干
a,b,c = input().strip().split()#其实strip()可有可无
print(a,b,c)
#strip('可选字符，默认为空格')的用处：去掉字符串首位连续的某字符
#split('可选指定分隔符',可选分割次数)的用法：通过分隔符将字符串切片处理
#注意，python的格式控制是这样的。
print(a+b+c,a*b*c,"{:.2f}".format((a+b+c)/3))
#也是这样的
print("{}\n{}\n{:.6f}".format(100,'A',3.14))
print("{:02d}:{:02d}:{:02d}".format(timeA,timeB,timeC))
#d代表输出int，2代表输出宽度，0代表剩余位用0来填充。
01:08:31
#如果你想在字符串中表示\ 请用"\\" 转义

# input: 2(2(2+2(0))+2)+2(2(2+2(0)))+2(2(2)+2(0))+2+2(0)
# output: 1315
a = input()
a = a.replace("2(", "pow(2,")
a = int(eval(a))
print(a)
```