E0-算法第0次练习赛-数据结构复习

想当年，也正是因为这场比赛，我认识到了自己和他人的巨大差距，以及能力上的垃圾...也不知道到了现在，我是否也能独当一面了呢?

A 二叉搜索树的和值

时间限制：1000ms 内存限制：65536kb

通过率：60/164 `(36.59%)` 正确率：60/558 `(10.75%)`

题目

知识点：二叉搜索树，不是模拟

二叉搜索树是一种特殊的二叉树（每个节点最多只有两个儿子的树）。树的每个节点上存有一个唯一的值，并且满足：这个节点的左子树内所有点的值都比这个节点的值小，且右子树内所有点的值都比这个节点的值要大。

我们定义一棵`二叉搜索树的和值`为当前树的所有节点的深度和（根节点的深度为0）。现在有 N 个数需要插入一棵空树中。给定插入序列，请在每个元素被插入之后，输出当前树的`和值`。

输入

第一行为一个整数 n 接下来一行是n个各不相同的数字，这些数字在[1, n]区间。

(0<n<3000000<n<300000)

输出

输出 n 行，第 i 行整数表示第 i个数插入树后，`当前树的和值`。

输入样例

```
8
3 5 1 6 8 7 2 4
```

输出样例

```
0
1
2
4
7
11
13
15
```

AC Code

```
#include <stdio.h>
#define N 300010
typedef long long ll;
int max(int x,int y) {return x > y ? x : y; }
int Nxt[N], Pre[N], n, a[N], dep[N];
void write(ll x) {
    if (x < 0)putchar_unlocked('-'), x = -x;
    if (x > 9)write(x / 10);
    putchar_unlocked(x % 10 + 48);
}
int read() {
    int k = 0, f = 1;
    char c = getchar_unlocked();
    while (c < '0' || c>'9') {
        if (c == '-')f = -1;
        c = getchar_unlocked();
    }
    while (c >= '0' && c <= '9') {
        k = (k << 1) + (k << 3) + c - 48;
        c = getchar_unlocked();
    }
    return k * f;
}
int main() {
    scanf("%d", &n);
    for(int i = 1;i <= n; ++i) {
        a[i] = read();
        Nxt[i] = i + 1;
        Pre[i] = i - 1;
    }
    Nxt[0] = 1; Pre[n + 1] = n;
    for(int i = n;i >= 1; --i) {
        Nxt[Pre[a[i]]] = Nxt[a[i]];
        Pre[Nxt[a[i]]] = Pre[a[i]];
    }
    ll ans = 0; dep[0] = dep[n + 1] = -1;
    for(int i = 1;i <= n; ++i) {
```

```
        dep[a[i]] = max(dep[Pre[a[i]]], dep[Nxt[a[i]]]) + 1;
        ans += dep[a[i]];
        write(ans),putchar_unlocked('\n');
    }
}
```

```
#include<cstdio>
#include<iostream>
#include<cmath>
#include<set>
using namespace std;
long long height[300002];
long long ans;

int main()
{
  set<int>s;
  int n, base;
  scanf("%d", &n);
  scanf("%d", &base);
  s.insert(base);
  height[base] = 0;
  int max_ = base;
  int min_ = base;
  printf("%lld\n", height[base]);
  for(int i = 1; i < n; i++)
  {
    int x;
    scanf("%d", &x);
    s.insert(x);
    max_ = max_ > x ? max_ : x;

    set<int>::iterator tmp, tmp1;
    tmp = s.lower_bound(x);
    if(*tmp == *s.begin()) height[x] = height[*(++tmp)] + 1;
    else if (*tmp == max_) height[x] = height[*(--tmp)] + 1;
    else
    {
      tmp1 = s.upper_bound(x);
      tmp--;
      height[x] = max(height[*tmp], height[*tmp1]) + 1;
    }
    ans += height[x];
    printf("%lld\n", ans);
  }
}
```

```
#include<cstdio>
```

```
#define maxn 300010
typedef long long ll;
#define max(a,b) (((a)>(b))?(a):(b))
//红黑树相关的宏定义函数,包括找兄弟,判断是左孩子右孩子,红黑节点
#define bro(x) (((x)->ftr->lc == (x)) ? ((x)->ftr->rc):((x)->ftr->lc))
#define islc(x) ((x)!=NULL && (x)->ftr->lc == (x))
#define isrc(x) ((x)!=NULL && (x)->ftr->rc == (x))
#define BLACK 0
#define RED 1
inline void write(ll x) {
    if (x < 0)putchar_unlocked('-'), x = -x;
    if (x > 9)write(x / 10);
    putchar_unlocked(x % 10 + 48);
}
inline int read() {
    int k = 0, f = 1;
    char c = getchar_unlocked();
    while (c < '0' || c>'9') {
        if (c == '-')f = -1;
        c = getchar_unlocked();
    }
    while (c >= '0' && c <= '9') {
        k = (k << 1) + (k << 3) + c - 48;
        c = getchar_unlocked();
    }
    return k * f;
}
template <typename T>
struct RedBlackTree {
    /**红黑树节点**/
    struct RBNode {
        T val;//节点值
        bool color;//true为红,false为黑
        RBNode* ftr;
        RBNode* lc, * rc;//父亲节点左孩子右孩子
        int _size;//排名专用:记录以此点为根的树的规模

        RBNode(T v = T(), bool RB = RED, RBNode* f = NULL,
            RBNode* l = NULL, RBNode* r = NULL, int s = 1)
            : val(v), color(RB), ftr(f), lc(l), rc(r), _size(s) {}

        //删除节点专用后继
        RBNode* single_succ() {
            RBNode* ret = rc;
            while (ret->lc) {
                --(ret->_size);
                ret = ret->lc;
            }
            return ret;
```

```
    }
    //直接前驱,从处注意可以是NULL
    RBNode* pred() {
        RBNode* ret = this;
        if (!lc) {
            while (ret->ftr && ret->ftr->lc == ret)
                ret = ret->ftr;
            ret = ret->ftr;
        }
        else {
            ret = ret->lc;
            while (ret->rc)
                ret = ret->rc;
        }
        return ret;
    }
    //直接后继
    RBNode* succ() {
        RBNode* ret = this;
        if (!rc) {
            while (ret->ftr && ret->ftr->rc == ret)
                ret = ret->ftr;
            ret = ret->ftr;
        }
        else {
            ret = ret->rc;
            while (ret->lc)
                ret = ret->lc;
        }
        return ret;
    }
    //维护域
    void maintain() {
        _size = 1;
        if (lc)_size += lc->_size;
        if (rc)_size += rc->_size;
    }
};

/**外部不可见部分**/

RBNode* _root;//根节点
RBNode* _hot;//查找专用命中_hot

void init(T v) { _root = new RBNode(v, BLACK, NULL, NULL, NULL, 1); }

//统一重平衡代码,考虑3个节点4个子树
//分类讨论排序不在此处做,传入接口时是排好序的
void connect34(RBNode* nroot, RBNode* nlc, RBNode* nrc,
```

```
        RBNode* ntree1, RBNode* ntree2, RBNode* ntree3, RBNode* ntree4) {
        nlc->lc = ntree1;
        if (ntree1)ntree1->ftr = nlc;
        nlc->rc = ntree2;
        if (ntree2)ntree2->ftr = nlc;
        nrc->lc = ntree3;
        if (ntree3)ntree3->ftr = nrc;
        nrc->rc = ntree4;
        if (ntree4)ntree4->ftr = nrc;
        nroot->lc = nlc, nlc->ftr = nroot;
        nroot->rc = nrc, nrc->ftr = nroot;
        nlc->maintain(), nrc->maintain();
        nroot->maintain();
    }

    //允许重复的查找,默认是找到同一个数的最后一个出现的位置
    RBNode* find(T v, const int op) {
        RBNode* ptn = _root;
        _hot = NULL;
        while (ptn) {
            _hot = ptn;
            ptn->_size += op;
            if (ptn->val > v)
                ptn = ptn->lc;
            else
                ptn = ptn->rc;
        }
        return ptn;
    }
    //不允许重复的查找,只要找到对应值就收手
    RBNode* rfind(T v, const int op) {
        RBNode* ptn = _root;
        _hot = NULL;
        while (ptn && ptn->val != v) {
            _hot = ptn;
            ptn->_size += op;
            if (ptn->val > v)
                ptn = ptn->lc;
            else
                ptn = ptn->rc;
        }
        return ptn;
    }

    //双红修正,采用迭代方式,迭代条件为RR-2上溢向上传2层
    //判断双红的时候,自己得先是红的,然后判断父亲节点
    void SolveDoubleRed(RBNode* nn) {
        while ((!(nn->ftr)) || nn->ftr->color == RED) {
            if (nn == _root) {
```

```
        //根节点强制重染色为黑色,并且增加黑高度
        _root->color = BLACK;
        return;
    }
    RBNode* p = nn->ftr;
    if (p->color == BLACK)return;//case 1:没有双红,直接返回
    RBNode* u = bro(p);
    RBNode* g = p->ftr;
    //case 2:RR-2
    if (u != NULL && u->color == RED) {
        g->color = RED;
        p->color = u->color = BLACK;
        nn = g;//下一次检查往上翻2层
    }
    //case 3:RR-1 直接返回
    //此时就是要先根据情况调整父子节点位置,排好3+4结构,重染色
    else {
        if (islc(p)) {
            if (islc(nn)) {//zig
                p->ftr = g->ftr;
                if (g == _root) _root = p;
                else if (g->ftr->lc == g) g->ftr->lc = p;
                else g->ftr->rc = p;
                connect34(p, nn, g, nn->lc, nn->rc, p->rc, u);
                p->color = BLACK;
                g->color = RED;
            }
            else {//zag-zig
                nn->ftr = g->ftr;
                if (g == _root) _root = nn;
                else if (g->ftr->lc == g) g->ftr->lc = nn;
                else g->ftr->rc = nn;
                connect34(nn, p, g, p->lc, nn->lc, nn->rc, u);
                nn->color = BLACK;
                g->color = RED;
            }
        }
        else {
            if (islc(nn)) {//zig-zag
                nn->ftr = g->ftr;
                if (g == _root) _root = nn;
                else if (g->ftr->lc == g) g->ftr->lc = nn;
                else g->ftr->rc = nn;
                connect34(nn, g, p, u, nn->lc, nn->rc, p->rc);
                nn->color = BLACK;
                g->color = RED;
            }
            else {//zag
                p->ftr = g->ftr;
```

```
                    if (g == _root) _root = p;
                    else if (g->ftr->lc == g) g->ftr->lc = p;
                    else g->ftr->rc = p;
                    connect34(p, g, nn, u, p->lc, nn->lc, nn->rc);
                    p->color = BLACK;
                    g->color = RED;
                }
            }
            return;
        }
    }
}

//双黑修正,采用迭代方式,迭代条件为BB-2B下溢向上传1层
//如果是根节点了,就可以直接停机了
void SolveDoubleBlack(RBNode* nn) {
    while (nn != _root) {
        RBNode* p = nn->ftr;
        RBNode* b = bro(nn);
        if (b->color == RED) { //Case 1:BB-3
            b->color = BLACK;
            p->color = RED;
            if (_root == p)_root = b;
            if (p->ftr) {
                if (p->ftr->lc == p)p->ftr->lc = b;
                else p->ftr->rc = b;
            }
            b->ftr = p->ftr;
            if (islc(nn))//zag
                connect34(b, p, b->rc, nn, b->lc, b->rc->lc, b->rc->rc);
            else//zig
                connect34(b, b->lc, p, b->lc->lc, b->lc->rc, b->rc, nn);
            b = bro(nn), p = nn->ftr;
            //转换问题并往后接着推,维护的核心节点nn不变
        }
        if (b->lc && b->lc->color == RED) { //Case 2-1:BB-1
            bool old_p_color = p->color;
            p->color = BLACK;
            if (p->lc == nn) {//zig-zag
                if (p->ftr) {
                    if (p->ftr->lc == p)p->ftr->lc = b->lc;
                    else p->ftr->rc = b->lc;
                }
                b->lc->ftr = p->ftr;
                if (_root == p)_root = b->lc;
                connect34(b->lc, p, b, nn, b->lc->lc, b->lc->rc, b->rc);
            }
            else {//zig
                b->lc->color = BLACK;
```

```
                    if (p->ftr) {
                        if (p->ftr->lc == p)p->ftr->lc = b;
                        else p->ftr->rc = b;
                    }
                    b->ftr = p->ftr;
                    if (_root == p)_root = b;
                    connect34(b, b->lc, p, b->lc->lc, b->lc->rc, b->rc, nn);
                }
                p->ftr->color = old_p_color;
                return;//BB-1直接返回
            }
            else if (b->rc && b->rc->color == RED) { //Case 2-2:BB-1
                bool old_p_color = p->color;
                p->color = BLACK;
                if (p->lc == nn) {//zag
                    b->rc->color = BLACK;
                    if (p->ftr) {
                        if (p->ftr->lc == p)p->ftr->lc = b;
                        else p->ftr->rc = b;
                    }
                    b->ftr = p->ftr;
                    if (_root == p)_root = b;
                    connect34(b, p, b->rc, nn, b->lc, b->rc->lc, b->rc->rc);
                }
                else {//zag-zig
                    if (p->ftr) {
                        if (p->ftr->lc == p)p->ftr->lc = b->rc;
                        else p->ftr->rc = b->rc;
                    }
                    b->rc->ftr = p->ftr;
                    if (_root == p)_root = b->rc;
                    connect34(b->rc, b, p, b->lc, b->rc->lc, b->rc->rc, nn);
                }
                p->ftr->color = old_p_color;
                return;//BB-1直接返回
            }
            if (p->color == RED) {//case 3:BB-2R
                p->color = BLACK;
                b->color = RED;
                return;//BB-2R直接返回
            }
            else {//case 4:BB-2B
                b->color = RED;
                nn = p;
                //BB-2B下溢上传,继续迭代
            }
        }
    }
```

```
    RBNode* findKth(int Rank, RBNode* ptn) {
        if (ptn->lc == NULL) {
            if (Rank == 1)return ptn;
            else return findKth(Rank - 1, ptn->rc);
        }
        else {
            if (ptn->lc->_size == Rank - 1)return ptn;
            else if (ptn->lc->_size >= Rank)return findKth(Rank, ptn->lc);
            else return findKth(Rank - (ptn->lc->_size) - 1, ptn->rc);
        }
    }

    int find_rank(T v, RBNode* ptn) {
        if (!ptn)return 1;
        else if (ptn->val >= v)return find_rank(v, ptn->lc);
        else return (1 + ((ptn->lc) ? (ptn->lc->_size) : 0) + find_rank(v, ptn-
>rc));
    }

    /**查看现象的debug接口**/

    void previs(RBNode* ptn) {
        printf("current node value is %d, color is %s, _size is %d\n", ptn-
>val, ptn->color ? "Red" : "Black", ptn->_size);
        printf("Lchild value is "); if (ptn->lc)printf("%d _size is %d\n", ptn-
>lc->val, ptn->lc->_size); else puts("NULL");
        printf("Rchild value is "); if (ptn->rc)printf("%d _size is %d\n", ptn-
>rc->val, ptn->lc->_size); else puts("NULL");
        if (ptn->lc)previs(ptn->lc);
        if (ptn->rc)previs(ptn->rc);
    }

    void invis(RBNode* ptn) {
        if (ptn->lc)invis(ptn->lc);
        printf("current node value is %d, color is %s, _size is %d\n", ptn-
>val, ptn->color ? "Red" : "Black", ptn->_size);
        printf("Lchild value is "); if (ptn->lc)printf("%d _size is %d\n", ptn-
>lc->val, ptn->lc->_size); else puts("NULL");
        printf("Rchild value is "); if (ptn->rc)printf("%d _size is %d\n", ptn-
>rc->val, ptn->lc->_size); else puts("NULL");
        if (ptn->rc)invis(ptn->rc);
    }

    void postvis(RBNode* ptn) {
        if (ptn->lc)postvis(ptn->lc);
        if (ptn->rc)postvis(ptn->rc);
        printf("current node value is %d, color is %s, _size is %d\n", ptn-
>val, ptn->color ? "Red" : "Black", ptn->_size);
```

```
        printf("Lchild value is "); if (ptn->lc)printf("%d _size is %d\n", ptn-
>lc->val, ptn->lc->_size); else puts("NULL");
        printf("Rchild value is "); if (ptn->rc)printf("%d _size is %d\n", ptn-
>rc->val, ptn->lc->_size); else puts("NULL");
    }

    void checkconnect(RBNode* ptn) {
        if (!ptn) return;
        if (ptn->lc && ptn->lc->ftr != ptn) {
            printf("Oops! %d has a lc %d, but it failed to point its ftr!\n",
ptn->val, ptn->lc->val);
        }
        if (ptn->rc && ptn->rc->ftr != ptn) {
            printf("Oops! %d has a rc %d, but it failed to point its ftr!\n",
ptn->val, ptn->rc->val);
        }
        int sss = ptn->_size;
        if (ptn->lc) sss -= ptn->lc->_size;
        if (ptn->rc) sss -= ptn->rc->_size;
        if (sss - 1) {
            printf("Fuck it! %d's size is %d, but the sum of its children's
size is %d!\n", ptn->val, ptn->_size, ptn->_size - sss);
        }
        checkconnect(ptn->lc);
        checkconnect(ptn->rc);
    }

    void correctlyconnected() {
        checkconnect(_root);
    }

    /**节点对应的迭代器**/
    struct iterator {
        //这个数据实际上也要private
        RBNode* _real__node;

        iterator& operator++() {
            _real__node = _real__node->succ();
            return *this;
        }
        iterator& operator--() {
            _real__node = _real__node->pred();
            return *this;
        }
        T operator*() {
            return _real__node->val;
        }

        iterator(RBNode* node_nn = NULL) :_real__node(node_nn) {}
```

```
        iterator(T const& val_vv) :_real__node(rfind(val_vv, 0)) {}
        iterator(iterator const& iter) :_real__node(iter._real__node) {}

    };

    /**对外端口部分**/

    RedBlackTree() :_root(NULL), _hot(NULL) {}

    //插入 返回对应迭代器
    iterator insert(T v) {
        RBNode* ptn = find(v, 1);
        if (_hot == NULL) {//仅有1个节点
            init(v);
            return iterator(_root);
        }
        ptn = new RBNode(v, RED, _hot, NULL, NULL, 1);
        if (_hot->val <= v)_hot->rc = ptn;
        else _hot->lc = ptn;
        SolveDoubleRed(ptn);
        return iterator(ptn);
    }
    //删除 返回成功或失败 本题中只可能是true
    bool remove(T v) {
        RBNode* ptn = rfind(v, -1);
        if (!ptn)return false;
        RBNode* node_suc;
        //单双分支情况处理
        while (ptn->lc || ptn->rc) {
            if (ptn->lc == NULL)node_suc = ptn->rc;
            else if (ptn->rc == NULL)node_suc = ptn->lc;
            else node_suc = ptn->single_succ();
            --(ptn->_size);
            ptn->val = node_suc->val;
            ptn = node_suc;
        }
        //当对应删去的节点为黑色,则需要双黑修正
        if (ptn->color == BLACK) {
            --(ptn->_size);
            SolveDoubleBlack(ptn);
        }
        if (ptn == _root) {
            _root = NULL;
            delete ptn;
            return true;
        }
        if (ptn->ftr->lc == ptn)ptn->ftr->lc = NULL;
        else ptn->ftr->rc = NULL;
        delete ptn;
```

```
        return true;
    }
    //查排名
    int get_rank(T v) {
        return find_rank(v, _root);
    }
    int size() {
        return _root->_size;
    }
    bool empty() {
        return !_root;
    }
    iterator Kth(int Rank) {
        return iterator(findKth(Rank, _root));
    }
    iterator lower_bound(T v) {
        RBNode* ptn = _root;
        while (ptn) {
            _hot = ptn;
            if (ptn->val < v)ptn = ptn->rc;
            else ptn = ptn->lc;
        }
        if (_hot->val < v)ptn = _hot;
        else ptn = _hot->pred();
        return iterator(ptn);
    }
    iterator upper_bound(T v) {
        RBNode* ptn = _root;
        while (ptn) {
            _hot = ptn;
            if (ptn->val > v)ptn = ptn->lc;
            else ptn = ptn->rc;
        }
        if (_hot->val > v)ptn = _hot;
        else ptn = _hot->succ();
        return iterator(ptn);
    }

};
RedBlackTree<int> Tree;
ll depth[maxn];
ll ans;
int n;
int x;
int main() {
    n = read();
    x = read();
    Tree.insert(x);
    puts("0");
```

```
    n--;
    while (n--) {
        x = read();
        Tree.insert(x);
        if (Tree.get_rank(x) == 1)
            depth[x] = 1 + depth[*Tree.upper_bound(x)];
        else if (Tree.get_rank(x) == Tree.size())
            depth[x] = 1 + depth[*Tree.lower_bound(x)];
        else
            depth[x] = 1 + max(depth[*Tree.upper_bound(x)],
depth[*Tree.lower_bound(x)]);
        ans += depth[x];
        write(ans), putchar_unlocked('\n');
    }

}
```

```
#include<cstdio>
#include<cstdlib>
#include<cctype>
#include<vector>
#include<map>
#include<algorithm>
#define lowsqrt(u) (1 << (M[u] >> 1))
#define low(u,x) (x % lowsqrt(u))
#define high(u,x) (x / lowsqrt(u))
#define index(u,x,y) (x*lowsqrt(u)+y)
//上下平方根相关函数
#define NIL -1
using namespace std;
typedef long long ll;
inline void write(ll x) {
    if (x < 0)putchar_unlocked('-'), x = -x;
    if (x > 9)write(x / 10);
    putchar_unlocked(x % 10 + 48);
}
inline int read() {
    int k = 0, f = 1;
    char c = getchar_unlocked();
    while (c < '0' || c>'9') {
        if (c == '-')f = -1;
        c = getchar_unlocked();
    }
    while (c >= '0' && c <= '9') {
        k = (k << 1) + (k << 3) + c - 48;
        c = getchar_unlocked();
    }
    return k * f;
```

```
}
map<int, int>M;
inline void buildM() {
    for (int i = 0; i < 32; ++i)M[1 << i] = i;
}
struct vanEmdeBoasTree {
    struct Node {
        int u, value_max, value_min;
        //全域大小,区域最大值最小值
        Node* summary;
        //veb(u的上平方根)结构
        vector<Node*>cluster;
        //u的上平方根veb(u的下平方根)结构
        inline void build(int bitSize) {
            if (bitSize <= 1) {//最底层节点
                value_max = value_min = NIL;
                u = 2;
                summary = NULL;
                return;
            }
            value_max = value_min = NIL;
            u = 1 << bitSize;
            int Nsize = (bitSize >> 1) + (bitSize & 1);
            //计算u的上平方根
            summary = new Node();
            summary->build(Nsize);
            cluster.resize(1 << Nsize);
            for (int i = 0; i < cluster.size(); ++i) {
                cluster[i] = new Node();
                cluster[i]->build(bitSize >> 1);
                //大小为u的下平方根
            }
        }
    } *root;
    int _size;
    inline void build(int size) {
        int bit = 0;
        for (size--; size; size >>= 1, ++bit);
        root = new Node();
        root->build(bit);
    }
    inline int _Min(Node* t) { return t->value_min; }
    inline int _Max(Node* t) { return t->value_max; }
    inline void _insert(Node* t, int x) {
        if (t->value_min == NIL) {
            t->value_min = t->value_max = x;
            return;
        }
        if (x < t->value_min) swap(x, t->value_min);
```

```
        int H = high(t->u, x), L = low(t->u, x);
        if (t->u > 2) {
            if (_Min(t->cluster[H]) == NIL) {
                _insert(t->summary, H);
                t->cluster[H]->value_min = L;
                t->cluster[H]->value_max = L;
            }
            else _insert(t->cluster[H], L);
        }
        if (x > t->value_max)t->value_max = x;
    }
    inline void _remove(Node* t, int x) {
        if (t->value_min == t->value_max)
            t->value_max = t->value_min = NIL;
        else {
            if (t->u == 2)
                t->value_min = t->value_max = !x;
            else {
                if (x == t->value_min) {
                    int first_cluster = _Min(t->summary);
                    x = index(t->u, first_cluster, _Min(t-
>cluster[first_cluster]));
                    t->value_min = x;
                }
                int H = high(t->u, x), L = low(t->u, x);
                _remove(t->cluster[H], L);
                if (_Min(t->cluster[H]) == -1) {
                    _remove(t->summary, H);
                    if (x == t->value_max) {
                        int summary_max = _Max(t->summary);
                        if (summary_max == NIL)
                            t->value_max = t->value_min;
                        else t->value_max = index(t->u, summary_max, _Max(t-
>cluster[summary_max]));
                    }
                }
                else if (x == t->value_max)
                    t->value_max = index(t->u, H, _Max(t->cluster[H]));
            }
        }
    }
    inline int _successor(Node* t, int x) {
        if (t->u == 2) {
            if (x == 0 && t->value_max == 1)return 1;
            else return NIL;
        }
        else if (t->value_min != NIL && x < t->value_min)return t->value_min;
        else {
            int H = high(t->u, x), L = low(t->u, x);
```

```
            int maxlow = _Max(t->cluster[H]);
            if (maxlow != NIL && L < maxlow) {
                int offset = _successor(t->cluster[H], L);
                return index(t->u, H, offset);
            }
            else {
                int succ_cluster = _successor(t->summary, H);
                if (succ_cluster == NIL)return NIL;
                else {
                    int offset = _Min(t->cluster[succ_cluster]);
                    return index(t->u, succ_cluster, offset);
                }
            }
        }
    }
    inline int _predecessor(Node* t, int x) {
        if (t->u == 2) {
            if (x == 1 && t->value_min == 0)return 0;
            else return NIL;
        }
        else if (t->value_max != NIL && x > t->value_max)return t->value_max;
        else {
            int H = high(t->u, x), L = low(t->u, x);
            int minhigh = _Min(t->cluster[H]);
            if (minhigh != NIL && L > minhigh) {
                int offset = _predecessor(t->cluster[H], L);
                return index(t->u, H, offset);
            }
            else {
                int pred_cluster = _predecessor(t->summary, H);
                if (pred_cluster == NIL) {
                    if (t->value_min != NIL && x > t->value_min)return t-
>value_min;
                    else return NIL;
                }
                else {
                    int offset = _Max(t->cluster[pred_cluster]);
                    return index(t->u, pred_cluster, offset);
                }
            }
        }
    }
    inline bool _exist(Node* t, int x) {
        if (x == t->value_min || x == t->value_max)return true;
        else if (t->u == 2)return false;
        else  return _exist(t->cluster[high(t->u, x)], low(t->u, x));
    }
    inline int Min() { return _Min(root); }
    inline int Max() { return _Max(root); }
```

```
    inline bool empty() { return _size == 0; }
    inline void insert(int x) { ++_size; _insert(root, x); }
    inline void remove(int x) { --_size; _remove(root, x); }
    inline int predecessor(int x) { return _predecessor(root, x); }
    inline int successor(int x) { return _successor(root, x); }
    inline bool exist(int x) { return _exist(root, x); }
};
vanEmdeBoasTree Tree;
int n, m;
int op, x;
inline void test() {
    n = read(), m = read();
    Tree.build(n);
    Tree._size = 0;
    while (m--) {
        op = read();
        if (op == 1) {
            x = read();
            if (!Tree.exist(x))Tree.insert(x);
        }
        else if (op == 2) {
            x = read();
            if (Tree.exist(x))Tree.remove(x);
        }
        else if (op == 3) {
            if (Tree.empty())puts("-1");
            else write(Tree.Min()), putchar('\n');
        }
        else if (op == 4) {
            if (Tree.empty())puts("-1");
            else write(Tree.Max()), putchar('\n');
        }
        else if (op == 5) {
            x = read();
            write(Tree.predecessor(x)), putchar('\n');
        }
        else if (op == 6) {
            x = read();
            write(Tree.successor(x)), putchar('\n');
        }
        else if (op == 7) {
            x = read();
            puts(Tree.exist(x) ? "1" : "-1");
        }
    }
}
ll ans;
ll depth[300010];
int main() {
```

```
    buildM();
    n = read();
    Tree.build(n+2);
    Tree._size = 0;
    Tree.insert(0), Tree.insert(n+1);
    depth[0] = depth[n+1] = -1;
    while(n--) {
        x = read();
        depth[x] = 1 + max(depth[Tree.predecessor(x)],
depth[Tree.successor(x)]);
        ans += depth[x];
        write(ans), putchar_unlocked('\n');
        Tree.insert(x);
    }
}
```

B Zexal的电影院

时间限制：1000ms 内存限制：65536kb

通过率：73/97 `(75.26%)` 正确率：73/514 `(14.20%)`

题目

知识点：优先队列，排序

ZexalZexal的电影院拿到了nn部电影的放映权，每部电影具有两个属性，LL（电影的时长），VV（电影给观众带来的愉悦值）。看KK部电影所带来的愉悦值为KK部电影的时长之和乘以KK部电影中最小的愉悦值。例如，喜欢看一套3部电影，其时长分别为 [5,7,4][5,7,4]和愉悦度分别为 [11,14,6][11,14,6]那么这一套电影所带来的愉悦值为(5+7+4）∗6=96(5+7+4）∗6=96现在电影院计划最多上映kk部不同的电影，那么电影院可以给观众带来的最大愉悦值是多少?

输入

第一行包含两个整数nn(1<n<1e51<n<1e5)和KK(1<k<1e31<k<1e3) 其中nn代表电影院所拥有放映权电影的总数，KK代表着可以上映的最大数量。

每个下一个nn行包含两个整数Li和ViLi和Vi（1<Li<1e6,1<Vi<1e61<Li<1e6,1<Vi<1e6)

输出

电影院可以带给观众的最大的愉悦值

输入样例

```
4 3
4 7
15 1
3 6
6 8
```

输出样例

```
78
```

样例解释

我们可以选择电影1,3,4，所以最大愉悦值是(4+3+6)∗6=78

AC Code

```
#include<cstdio>
#include<cstdlib>
#include<queue>
#include<algorithm>
#define maxn 100010
#define debug 0
using namespace std;

struct movie {
    int time;
    int happy;
}movies[maxn];

bool compare(movie a, movie b) {
    if (a.happy != b.happy)return a.happy > b.happy;
    else return a.time > b.time;
}
unsigned int n, k;
long long result, sum;
void printTab() {
    for (int i = 0; i < n; i++) {
        printf("%d : %d %d\n", i, movies[i].time, movies[i].happy);
    }
}

int main() {
    while (scanf("%d%d", &n, &k) != EOF) {
        for (int i = 0; i < n; ++i) {
            scanf("%d%d", &movies[i].time, &movies[i].happy);
        }
        sort(movies, movies + n, compare);
```

```
        if (debug)printTab();
        priority_queue<int, vector<int>, greater<int> > q;
        //这里改一下，不装电影，装int，更加简洁
        result = 0;
        sum = 0;

        for (int i = 0; i < n; ++i) {
            q.push(movies[i].time);
            sum += (long long)movies[i].time;
            if (q.size() > k) { sum -= q.top(); q.pop(); }
            //这里不保留当前的最优解的...因为更大的解并不一定在这个基础上更优
            result = max(result, movies[i].happy * sum);
        }
        printf("%lld\n", result);
    }
    return 0;
}
```

C 连续子序列的权值

时间限制：200ms 内存限制：65536kb

通过率：82/125 `(65.60%)` 正确率：82/553 `(14.83%)`

题目

知识点：单调栈

我们定义连续序列a[p],a[p+1]....a[q]a[p],a[p+1]....a[q]的权值为maxmax(a[p],a[p+1]....a[q])(a[p],a[p+1]....a[q])- minmin(a[p],a[p+1]....a[q])(a[p],a[p+1]....a[q])，给定一个由NN个整数组成的序列，请求出所有连续子序列的权值和。

输入

第11行：1个数NN，表示数组的长度。(1<=N<=50000)(1<=N<=50000)

第2−N+12−N+1行：每行1个数，表示数组中的元素(1<=A[p]<=50000)(1<=A[p]<=50000)

输出

输出所有连续子序列的权值和。

输入样例

```
5
1
2
3
4
5
```

输出样例

```
20
```

AC Code

```
#include<cstdio>
#define maxn 50010
#define printDivideMaxDebug 0
#define printMaxMin 0

int N;


/**********************************
0.I/O优化(其实读入可以不加负数的)
**********************************/

int read() {
  int k = 0;
  int f = 1;
  char c = getchar();
  while (c < '0' || c>'9') {
    if (c == '-')f = -1;
    c = getchar();
  }
  while (c >= '0' && c <= '9') {
    k = (k << 1) + (k << 3) + c - 48;
    c = getchar();
  }
  return k*f;
}
void write(long long x) {
  //if(x < 0)putchar('-'),x=-x;
  if (x > 9)write(x / 10);
  putchar(x % 10 + 48);
}

long long max_sum;
long long min_sum;
```

```
struct array_with_Index {
  int number;
  int index;
}a[maxn];

array_with_Index dpmax[maxn][21];
array_with_Index dpmin[maxn][21];
/************************************
1.RMQ打表
************************************/
void getMax(int i, int j) {
  if (dpmax[i][j - 1].number >= dpmax[i + (1 << (j - 1))][j - 1].number) {
    dpmax[i][j].number = dpmax[i][j - 1].number;
    dpmax[i][j].index = dpmax[i][j - 1].index;
  }
  else {
    dpmax[i][j].number = dpmax[i + (1 << (j - 1))][j - 1].number;
    dpmax[i][j].index = dpmax[i + (1 << (j - 1))][j - 1].index;
  }
}

void getMin(int i, int j) {
  if (dpmin[i][j - 1].number <= dpmin[i + (1 << (j - 1))][j - 1].number) {
    dpmin[i][j].number = dpmin[i][j - 1].number;
    dpmin[i][j].index = dpmin[i][j - 1].index;
  }
  else {
    dpmin[i][j].number = dpmin[i + (1 << (j - 1))][j - 1].number;
    dpmin[i][j].index = dpmin[i + (1 << (j - 1))][j - 1].index;
  }
}

void RMQ_Preorder(int n) {
  for (int i = 1; i <= n; ++i) {
    dpmax[i][0].number = dpmin[i][0].number = a[i].number;
    dpmax[i][0].index = dpmin[i][0].index = a[i].index;
  }

  for (int j = 1; j <= 20; ++j) {
    for (int i = 1; i <= n; ++i) {
      if (i + (1 << j) - 1 <= n) {
        getMax(i, j); getMin(i, j);
      }
    }
  }

}
```

```
int locate_Max(int lo, int hi, int k) {
  if (dpmax[lo][k].number >= dpmax[hi - (1 << k) + 1][k].number) {
    return  dpmax[lo][k].index;
  }
  else {
    return  dpmax[hi - (1 << k) + 1][k].index;
  }
}

int locate_Min(int lo, int hi, int k) {
  if (dpmin[lo][k].number <= dpmin[hi - (1 << k) + 1][k].number) {
    return  dpmin[lo][k].index;
  }
  else {
    return  dpmin[hi - (1 << k) + 1][k].index;
  }
}

//find index
int RMQ_find_Max(int lo, int hi) {
  int k = 0;
  while (((1 << (k + 1)) + lo) <= hi + 1)k++;
  int result = locate_Max(lo, hi, k);
  return result;
}

int RMQ_find_Min(int lo, int hi) {
  int k = 0;
  while (((1 << (k + 1)) + lo) <= hi + 1)k++;
  int result = locate_Min(lo, hi, k);
  return result;
}

void RMQ_divide_Max(int lo, int hi) {
  if (printDivideMaxDebug)printf("divide max get here : %d %d\n", lo, hi);
  if (lo > hi) {
    if (printDivideMaxDebug)printf("divide max end lo > hi : %d %d\n", lo, hi);
    return;
  }
  if (lo == hi) {
    max_sum += a[lo].number;
    if (printDivideMaxDebug)printf("divide max end lo == hi : %d %d\n", lo,
hi);
    return;
  }
  int max_index = RMQ_find_Max(lo, hi);
  int max = a[max_index].number;
```

```
  max_sum += (long long)max * (long long)((long long)max_index - (long long)lo
+ (long long)1) * (long long)((long long)hi - (long long)max_index + (long
long)1);
  RMQ_divide_Max(lo, max_index - 1);
  RMQ_divide_Max(max_index + 1, hi);
  if (printDivideMaxDebug)printf("divide max end normal : %d %d\n", lo, hi);
}

void RMQ_divide_Min(int lo, int hi) {
  if (lo > hi)return;
  if (lo == hi) {
    min_sum += a[lo].number;
    return;
  }
  int min_index = RMQ_find_Min(lo, hi);
  int min = a[min_index].number;
  min_sum += (long long)min * (long long)((long long)min_index - (long long)lo
+ (long long)1) * (long long)((long long)hi - (long long)min_index + (long
long)1);
  RMQ_divide_Min(lo, min_index - 1);
  RMQ_divide_Min(min_index + 1, hi);
}

int main() {
  N = read();
  for (int i = 1; i <= N; ++i) {
    a[i].number = read();
    a[i].index = i;
  }
  RMQ_Preorder(N);
  RMQ_divide_Max(1, N);
  RMQ_divide_Min(1, N);
  if (printMaxMin) { write(max_sum); putchar('\n'); }
  if (printMaxMin) { write(min_sum); putchar('\n'); }
  write(max_sum - min_sum);
  return 0;
}
```

```
#include <iostream>
#include <stack>
#define maxn 50010
using namespace std;
int n, k;
int a[maxn], l[maxn];
int main() {
  ios::sync_with_stdio(false);
  cin.tie(nullptr);
  cin >> n;
```

```
  for (int i = 0; i < n; ++i) {
    cin >> a[i];
  }
  long long result = 0;
  stack <int> DDStack;//目前没太看懂，貌似放的是下标?
  for (int i = 0; i < n; ++i) {
    while (!DDStack.empty() && a[DDStack.top()] < a[i]) {
      result += 1ll * (i - DDStack.top()) * (DDStack.top() - l[DDStack.top()])
* a[DDStack.top()];
      DDStack.pop();
    }
    if (DDStack.empty())l[i] = -1;
    else l[i] = DDStack.top();
    DDStack.push(i);
  }
  long long r = n;
  while (!DDStack.empty()) {
    result += 1ll * (r - DDStack.top()) * (DDStack.top() - l[DDStack.top()]) *
a[DDStack.top()];
    DDStack.pop();
  }

  for (int i = 0; i < n; ++i) {
    while (!DDStack.empty() && a[DDStack.top()] > a[i]) {
      result -= 1ll * (i - DDStack.top()) * (DDStack.top() - l[DDStack.top()])
* a[DDStack.top()];
      DDStack.pop();
    }
    if (DDStack.empty())l[i] = -1;
    else l[i] = DDStack.top();
    DDStack.push(i);
  }
  r = n;
  while (!DDStack.empty()) {
    result -= 1ll * (r - DDStack.top()) * (DDStack.top() - l[DDStack.top()]) *
a[DDStack.top()];
    DDStack.pop();
  }
  cout << result;
}
```

D Zexal的拯救世界

时间限制：1000ms 内存限制：65536kb

通过率：111/161 `(68.94%)` 正确率：111/608 `(18.26%)`

题目

知识点：并查集

在一条数轴上坐落着NN个国家，分别是1～N1～N。一开始所有的国家都处于黑暗状态。接着ZexalZexal使用MM次魔法，第ii次魔法将会为[Li,Ri][Li,Ri]这些国家带来光明。请输出每次魔法使用后仍然处于黑暗状态下的国家数量。

输入

输入一行为NN和MM。下面MM行每行两个数Li、RiLi、Ri (1<=Li<=Ri<=N<=200000,1<=M<=200000)
(1<=Li<=Ri<=N<=200000,1<=M<=200000)

输出

输出MM行，每次魔法使用后仍然处于黑暗状态下的国家数量。

输入样例

```
10 3
3 3
5 7
2 8
```

输出样例

```
9
6
3
```

Tips

线段树（×）

并查集（√）

AC Code

```
#include<cstdio>
#define maxn 200010

int a[maxn];

inline int getFather(int x) {
  return a[x] ? a[x] = getFather(a[x]) : x;//最顶层有的话，就返回最顶层，否则自己就是最顶层
}
int n, m;
int ans;
```

```
int l, r;
int main() {
  scanf("%d%d", &n, &m);
  ans = n;
  while (m--) {
    scanf("%d%d", &l, &r);
    l = getFather(l);
    while (l <= r) {
      //左端点往右边挪，减少答案值
      a[l] = l + 1;//l被拯救了
      ans--;
      l = getFather(l);
    }
    printf("%d\n", ans);
  }
  return 0;
}
```

```
#include<cstdio>
#include<cstring>
#define Lchild(x) ((x) << 1)
#define Rchild(x) (((x) << 1) + 1)
#define Max(a,b) (((a)>(b))?(a):(b))
#define Min(a,b) (((a)<(b))?(a):(b))
#define maxn 200010
inline void write(int x) {
    if (x < 0)putchar('-'), x = -x;
    if (x > 9)write(x / 10);
    putchar(x % 10 + 48);
}
inline int read() {
    int k = 0, f = 1;
    char c = getchar();
    while (c < '0' || c>'9') {
        if (c == '-')f = -1;
        c = getchar();
    }
    while (c >= '0' && c <= '9') {
        k = (k << 1) + (k << 3) + c - 48;
        c = getchar();
    }
    return k * f;
}
struct SegmentTree {
    struct Node {
        int value, tag_Set;
    }nodes[maxn << 2];
    SegmentTree() {
```

```
        memset(nodes, 0, sizeof(nodes));
    }
    inline void pushup(int root) {
        nodes[root].value = nodes[Lchild(root)].value +
nodes[Rchild(root)].value;
    }
    inline void build(int root, int l, int r) {
        nodes[root].tag_Set = 0;
        if (l == r)nodes[root].value = 0;
        else {
            int m = (l + r) >> 1;
            build(Lchild(root), l, m);
            build(Rchild(root), m + 1, r);
            pushup(root);
        }
    }
    inline void pushdown(int root, int l, int r) {
        int m = (l + r) >> 1;
        if (nodes[root].tag_Set == 1) {
            nodes[Lchild(root)].tag_Set = nodes[Rchild(root)].tag_Set =
nodes[root].tag_Set;
            nodes[Lchild(root)].value = (m - l + 1) * nodes[root].tag_Set;
            nodes[Rchild(root)].value = (r - m) * nodes[root].tag_Set;
        }
        if (nodes[root].tag_Set == -1) {
            nodes[Lchild(root)].tag_Set = nodes[Rchild(root)].tag_Set =
nodes[root].tag_Set;
            nodes[Lchild(root)].value = nodes[Rchild(root)].value = 0;
        }
        nodes[root].tag_Set = 0;
    }
    inline void updateSet(int root, int curl, int curr, int tarl, int tarr, int
k) {
        //k = 1 for save k = -1 for get into darkness
        if (tarr < curl || curr < tarl)return;
        if (tarl <= curl && curr <= tarr) {
            nodes[root].tag_Set = k;
            if (k == 1)nodes[root].value = curr - curl + 1;
            if (k == -1)nodes[root].value = 0;
            return;
        }
        pushdown(root, curl, curr);
        int m = (curl + curr) >> 1;
        if (tarl <= m) updateSet(Lchild(root), curl, m, tarl, tarr, k);
        if (tarr > m) updateSet(Rchild(root), m + 1, curr, tarl, tarr, k);
        pushup(root);
    }
    inline int query(int root, int curl, int curr, int tarl, int tarr) {
        if (tarr < curl || curr < tarl)return 0;
```

```
        if (tarl <= curl && curr <= tarr) {
            return nodes[root].value;
        }
        pushdown(root, curl, curr);
        int m = (curl + curr) >> 1;
        int ret = 0;
        if (tarl <= m) ret += query(Lchild(root), curl, m, tarl, tarr);
        if (tarr > m) ret += query(Rchild(root), m + 1, curr, tarl, tarr);
        return ret;
    }

};
SegmentTree tree;
int n, m;
int l, r;
int main() {
    n = read(), m = read();
    tree.build(1, 1, n);
    while (m--) {
        l = read(), r = read();
        tree.updateSet(1, 1, n, l, r, 1);
        write(n - tree.query(1, 1, n, 1, n));
        putchar('\n');
    }
}
```

E Zexal的二叉树（签到）

时间限制：1000ms 内存限制：65536kb

通过率：217/225 `(96.44%)` 正确率：217/654 `(33.18%)`

题目

知识点：树，数论，dp，递归（都可以做）

上学期我们学习了二叉树，也都知道3个结点的二叉树有5种， 现给你二叉树的结点个数nn，要你输出不同形态二叉树的种数。

输入

第一个数为一个整数n(n<=30)n(n<=30)

输出

对于每组数据，输出一行，不同形态二叉树的种数。

输入样例

```
3
```

输出样例

```
5
```

AC Code

```
#include<cstdio>
typedef long long ll;
int n;
int main() {
  while (scanf("%d", &n) != EOF) {
    if (n == 0)puts("0");
    else {
      ll a = 1;
      for (int i = 1; i <= n; ++i) {
        a *= (n + i), a /= i;
      }
      a /= (n + 1);
      printf("%lld\n", a);
    }
  }
}
```

F 多多岛

时间限制：1000ms 内存限制：65536kb

通过率：51/67 `(76.12%)` 正确率：51/207 `(24.64%)`

题目

知识点：树，古典概率

多多群岛是一个群岛，由n个岛屿构成，不同的岛屿之间由桥梁连接，一共有n-1个桥梁，任意两个岛屿一定联通。从一座岛屿跨过一座桥梁到另一个岛屿的时间是1。

多多群岛在只有一座桥与其他岛屿相连的岛屿上设有餐厅，就餐时间时，游客会选择距离他最近的餐厅就餐。

假设就餐时间时，一个游客在每座岛屿的概率相等，那么请问他到达餐厅花费时间的期望是多少。

输入

第一行一个正整数n表示岛屿的个数(2≤n<1052≤n<105)

接下来n-1行，每行两个整数x，y，表示第x座岛和第y座岛之间有一座桥梁 (1≤x,y≤n1≤x,y≤n)

输出

每组数据输出一行，保留4位小数

输入样例

```
2
1 2
```

输出样例

```
0.0000
```

AC Code

```
#include<iostream>
#include<algorithm>
#include<vector>
#include<queue>
#include<cstdlib>
#define maxn 100010
using namespace std;
typedef long long ll;
vector<int> g[maxn];
int d[maxn], n, m;
bool occur[maxn];
int a, b;
double ans;
int main() {
  ios::sync_with_stdio(false);
  cin.tie(0), cout.tie(0);
  cin >> n;
  for (int i = 0; i < n - 1; ++i) {
    cin >> a >> b;
    g[a].push_back(b), g[b].push_back(a);
  }
  queue<int> que;

  for (int i = 1; i <= n; ++i) {
    if (g[i].size() == 1) {
      //叶子节点情况
      que.push(i);
      occur[i] = true;
      d[i] = 0;
    }
```

```
  }
  while (!que.empty()) {
    int u = que.front(); que.pop();
    for (int i : g[u]) {
      if (!occur[i]) {
        d[i] = d[u] + 1;
        que.push(i);
        occur[i] = true;
      }
    }
  }

  for (int i = 1; i <= n; ++i)ans += d[i];
  printf("%.4lf", ans / n);
  return 0;
}
```

G 生日宴会

时间限制：1000ms 内存限制：65536kb

通过率：72/91 `(79.12%)` 正确率：72/329 `(21.88%)`

题目

知识点：拓扑排序，优先队列

贝克兰德的富商道恩.唐泰斯将要举办他的生日宴会，他将要邀请n个客人。现在他面临一个问题，安排客人的到场顺序。

在贝克兰德的社交礼仪中，一场宴会的客人总是一个接一个地到达，也就是说，没有两个客人可以在同一时间到达。到达顺序也有一定的限制，大佬应该在小弟全部到场后再到，丈夫应该在妻子之前入场等等。

满足礼仪的顺序有多种，但是因为不同的客人，道恩对他们的熟悉度程度不同，他想要在满足礼仪的情况下，使得他熟悉的人先到场。

现在，道恩对客人按照熟悉程度进行编号1-n，其中1号他最熟悉。然后客人之间有m个到场顺序限制。现在请你生成一个排序，使得在满足到场限制的条件下，使得1号尽可能早的入场，然后2号，3号.....以此类推

输入

第一行两个正整数n,m(1≤n≤105,0≤m≤2×1051≤n≤105,0≤m≤2×105)

接下来m行，每行两个整数x，y，表示x应该比y先到 (1≤x,y≤n1≤x,y≤n)

输出

一行，n个人的排序，保证有解

输入样例

```
3 1
3 1
```

输出样例

```
3 1 2
```

输入样例

```
5 6
2 1
5 2
4 1
5 4
3 1
5 3
```

输出样例

```
5 2 3 4 1
```

AC Code

```
#pragma GCC optimize(2)
#include<iostream>
#include<queue>
#include<stack>
#include<vector>
#include<cstdlib>
#include<cstring>
#define maxn 100010
#define print 0
using namespace std;
typedef vector<int>::iterator IT;
//不保证图连通也可，因为不连通的点也算作没有前驱的点
vector<int>edge[maxn];//邻接表
priority_queue<int> que;//拓扑队列
stack<int> topoList;
int inDegree[maxn];//入度表
int n, m;
int u, v;
inline bool topoSort() {
```

```
  while (!que.empty())que.pop();
  for (int i = 1; i <= n; ++i) {
    if (!inDegree[i])que.push(i);
  }
  int cnt = 0;//如果需要记录拓扑序列的话，换一个queue即可
  while (!que.empty()) {
    int newP = que.top();
    topoList.push(newP);
    if (print)cout << "debug : pop " << newP << " out queue" << endl;
    que.pop();
    ++cnt;
    for (int i = 0; i < edge[newP].size(); ++i) {
      inDegree[edge[newP][i]]--;
      if (inDegree[edge[newP][i]] == 0) {
        que.push(edge[newP][i]);
        if (print)cout << "debug : push " << edge[newP][i] << " into queue" <<
endl;
      }
    }
  }
  while (!topoList.empty()) cout << topoList.top() << " ", topoList.pop();
  cout << endl;
  return cnt == n;
}

int main() {
  ios::sync_with_stdio(false);
  cin.tie(0), cout.tie(0);
  while (cin >> n >> m) {
    while (!topoList.empty())topoList.pop();
    for (int i = 1; i <= n; ++i) {
      inDegree[i] = 0;
      edge[i].clear();
    }
    while (m--) {
      cin >> u >> v;
      ++inDegree[u];
      edge[v].push_back(u);
    }
    topoSort();
  }
}
```

H 魔法阵

时间限制：1000ms 内存限制：65536kb

通过率：42/50 `(84.00%)` 正确率：42/161 `(26.09%)`

题目

知识点：最短路

克莱恩在一场冒险中得到了得到了一个破损的魔法阵，这个魔法阵是一个有n个点m条边的有向有环图，任意两点之间最多只有一条边，每条边有一个能量值a（可能是负数，别问问就是magical），不存在负环。

克莱恩试图去修补这个魔法阵。已知，这个魔法阵缺少了3条边，且已经知道这3条边的起点和终点（有向）。对于每条边，克莱恩要赋予其一个能量值c，为了避免邪神出现，修补过程以及结束后也不能出现负环。

请问每次的最小花费是多少(保证有解，可以是负数)。

输入

第一行两个正整数n,m(1≤n≤300,n−1≤m≤5001≤n≤300,n−1≤m≤500)

接下来m行，每行三个整数x，y，z，表示x->y有一条权值为z的边 (0≤x,y<n，−1000≤z≤10000≤x,y<n，−1000≤z≤1000)

最后三行，每行两个整数u，v表示需要填补一条u->v的边

输出

三行，每行一个整数

输入样例

```
10 15
4 7 10
7 6 3
5 3 3
1 4 11
0 6 20
9 8 25
3 0 9
1 2 15
9 0 27
5 2 0
7 3 -5
1 7 21
5 0 1
9 3 16
1 8 4
4 1
0 3
6 9
```

输出样例

```
-11
-9
-45
```

AC Code

```
#include<algorithm>
#include<iostream>
#include<climits>
#define maxn 310
#define debug 0
using namespace std;
typedef long long ll;
ll floyd[maxn][maxn];
int n, m;
int x, y, z;
inline void init() {
  for (int i = 0; i < maxn; ++i) {
    for (int j = 0; j < maxn; ++j) {
      if (i == j)floyd[i][j] = 0;
      else floyd[i][j] = INT_MAX;
    }
  }
}
inline void buildDP() {
  for (int k = 0; k < n; ++k) {
    for (int i = 0; i < n; ++i) {
      for (int j = 0; j < n; ++j) {
        if(floyd[i][k]<INT_MAX&& floyd[k][j]<INT_MAX)
          floyd[i][j] = min(floyd[i][k] + floyd[k][j], floyd[i][j]);
      }
    }
  }
}
inline void printMap() {
  for (int i = 0; i < 10; ++i) {
    for (int j = 0; j < 10; ++j) {
      if (floyd[i][j] == INT_MAX) printf("--- ");
      else printf("%3lld ", floyd[i][j]);
    }
    printf("\n");
  }
}
int main() {
  ios::sync_with_stdio(false);
```

```
  cin.tie(0), cout.tie(0);
  init();
  cin >> n >> m;
  for (int i = 0; i < m; ++i) {
    cin >> x >> y >> z;
    floyd[x][y] = z;
  }
  buildDP();
  if (debug)printMap();
  cin >> x >> y;
  floyd[x][y] = -floyd[y][x];
  cout << floyd[x][y] << endl;
  buildDP();
  if (debug)printMap();
  cin >> x >> y;
  floyd[x][y] = -floyd[y][x];
  cout << floyd[x][y] << endl;
  buildDP();
  if (debug)printMap();
  cin >> x >> y;
  floyd[x][y] = -floyd[y][x];
  cout << floyd[x][y] << endl;
}
```

I 治安点

时间限制：300ms 内存限制：65536kb

通过率：33/45 `(73.33%)` 正确率：33/181 `(18.23%)`

题目

知识点：最短路，最短路的优化

贝克兰德有n个城镇，这些城镇之间有m条道路连接，每条道路有一个长度l。

zf在其中k个城镇设置了治安点。当一个城镇发生事件时，任意一个治安点都可以派人前往。但是为了节省资源，往往会选择距离最近的治安点。

那么请问，对于每一个城镇，最近的治安点距离为多少。

输入

第一行一个整数t表示数据组数(1≤t≤101≤t≤10)

每组数据第一行三个正整数n,m,k (1≤n≤103,n−1≤m≤n∗n/2,1≤k≤n1≤n≤103,n−1≤m≤n∗n/2,1≤k≤n)

第二行k个整数表示治安点所在的位置。

接下来m行，每行三个整数x，y，z，表示x,y之间有一条权值为z的边（无向边）（1≤x,y≤n，1≤z≤100001≤x,y≤n，1≤z≤10000)

输出

每组数据一行，n个数

输入样例

```
1
4 7 2
1 4
1 2 7
1 3 2
1 4 6
2 1 1
2 4 1
3 2 1
3 4 3
```

输出样例

```
0 1 2 0
```

AC Code

```
#pragma G++ optimize(2)
#include<iostream>
#include<algorithm>
#include<queue>
#include<cstring>
#include<climits>
#define maxn 1010
#define RADIX 10
using namespace std;
typedef long long ll;
int graph[maxn][maxn];
ll dis[maxn];
bool occur[maxn];
//SPFA 可处理负权，判断负环，略于堆优dij
inline int read() {
  int k = 0;
  char c = getchar();
  while (!isdigit(c))c = getchar();
  while (isdigit(c)) { k = (k << 1) + (k << 3) + c - 48; c = getchar(); }
  return k;
}
inline void write(int a) {
  if (a >= RADIX)write(a / RADIX);
```

```
  putchar(a % RADIX + 48);
}
int t;
int n, m, k;
int tmp;
int u, v, w;
queue<int>q;
inline void SPFA(int from, int size) {
  for (int i = 0; i <= n; ++i) dis[i] = INT_MAX;
  while (!q.empty())q.pop();
  memset(occur, 0, sizeof(occur));
  q.push(from);
  dis[from] = 0;
  occur[from] = true;
  while (!q.empty()) {
    int tmp = q.front();
    q.pop();
    occur[tmp] = false;
    for (int i = 0; i <= n; ++i) {
      if (dis[i] > dis[tmp] + graph[tmp][i]) {
        dis[i] = dis[tmp] + graph[tmp][i];
        //cout << "debug : " << i << " " << tmp << " " << dis[i] << endl;
        if (!occur[i]) {
          q.push(i); occur[i] = true;
        }
      }
    }
  }
}
int main() {
  t = read();
  while (t--) {
    n = read(), m = read(), k = read();
    for (int i = 0; i <= n; ++i)
      for (int j = 0; j <= n; ++j)
        if (i == j)graph[i][j] = 0;
        else graph[i][j] = INT_MAX;
    for (int i = 1; i <= k; ++i) {
      tmp = read();
      graph[0][tmp] = graph[tmp][0] = 0;
    }
    for (int i = 1; i <= m; ++i) {
      u = read(), v = read(), w = read();
      if (graph[u][v] > w)graph[u][v] = graph[v][u] = w;
    }
    SPFA(0, n);
    for (int i = 1; i <= n; ++i) { write(dis[i]); putchar(' '); }
    putchar('\n');
  }
```

```
}
```

J 不能和其他题目重名的最小生成树

时间限制：1000ms 内存限制：65536kb

通过率：54/66 `(81.82%)` 正确率：54/203 `(26.60%)`

题目

知识点：最小生成树的Kruskal 算法

已知一个完全图唯一的最小生成树（即知道这个树所有边的端点和权值）,其余的边权值未知，问这个完全图所有边权值和的最小值。

完全图是每对顶点之间都恰连有一条边的简单图。

输入

第一行一个整数t表示数据组数(1≤t≤101≤t≤10)

每组数据第一行一个正整数n，表示完全图的点数(2≤n≤1052≤n≤105)

接下来n-1行，每行三个整数x，y，z，表示x,y之间有一条权值为z的边（无向边）(1≤x,y≤n，1≤z≤100001≤x,y≤n，1≤z≤10000)

输出

每组数据一行一个整数

输入样例

```
2
3
1 2 2
1 3 3
4
1 2 3
2 3 4
3 4 5
```

输出样例

```
9
29
```

AC Code

```cpp
#pragma G++ optimize(2)
#include<iostream>
#include<algorithm>
#include<cstdlib>
#include<cstring>
#include<queue>
#include<climits>
#define RADIX 10
#define maxn 100010

using namespace std;
typedef long long ll;

int f[maxn];
int s[maxn];//以i为偏序关系顶端的集合的个数，用于逆向推导
int t;
int n;
inline void init() {
  for (int i = 1; i <= n; ++i) {
    f[i] = i;
    s[i] = 1;
  }
}
inline int getFather(int x) {
  return f[x] == x ? x : f[x] = getFather(f[x]);
}
inline int read() {
  int k = 0;
  char c = getchar();
  while (!isdigit(c))c = getchar();
  while (isdigit(c)) { k = (k << 1) + (k << 3) + c - 48; c = getchar(); }
  return k;
}
inline void write(ll a) {
  if (a >= RADIX)write(a / RADIX);
  putchar(a % RADIX + 48);
}
struct edge {
  int u, v;
  int w;
  bool operator < (const edge& a) {
    return w < a.w;
  }
}mstEdge[maxn];

ll res;
int p, q;
int main() {
```

```
  ios::sync_with_stdio(false);
  cin.tie(0), cout.tie(0);

  t = read();
  while(t--){
    n = read();
    res = 0;
    init();
    for (int i = 0; i < n - 1; ++i) {
      mstEdge[i].u = read();
      mstEdge[i].v = read();
      mstEdge[i].w = read();
      res += mstEdge[i].w;
    }
    sort(mstEdge, mstEdge + n - 1);
    for (int i = 0; i < n - 1; ++i) {
      p = getFather(mstEdge[i].u);
      q = getFather(mstEdge[i].v);
      if (p != q) {
        f[p] = f[q];
        res += (1ll * mstEdge[i].w + 1ll) * (1ll * s[p] * s[q] - 1ll);
        s[q] += s[p];
        s[p] = 0;
      }
    }
    write(res); putchar('\n');
  }
}
```

E2-算法第2次练习赛

A E2-01背包

时间限制：5000ms 内存限制：65536kb

通过率：202/204 `(99.02%)` 正确率：202/674 `(29.97%)`

题面

有N件物品和一个容量为V的背包。第i件物品的费用是c[i]，价值是w[i]。求解将哪些物品装入背包可使价值总和最大。

输入

多组输入数据

每组数据第一行两个数n，v，表示物品的数量和背包的容量。
(1≤n≤500,1≤v≤300001≤n≤500,1≤v≤30000)

接下来n行，每行两个整数，表示物品的费用和价值(1≤ci,wi≤5001≤ci,wi≤500)

输出

每组数据一行一个数

输入样例

```
3 6
2 1
3 2
2 3
```

输出样例

```
5
```

B E2-完全背包

时间限制：1000ms 内存限制：65536kb

通过率：195/198 `(98.48%)` 正确率：195/352 `(55.40%)`

题面

有N种物品和一个容量为V的背包，每种物品都有无限件可用。第i种物品的费用是c[i]，价值是w[i]。求解将哪些物品装入背包可使这些物品的费用总和不超过背包容量，且价值总和最大。

输入

多组输入数据

每组数据第一行两个数n，v，表示物品的数量和背包的容量。
(1≤n≤500,1≤v≤300001≤n≤500,1≤v≤30000)

接下来n行，每行两个整数，表示物品的费用和价值(1≤ci,wi≤5001≤ci,wi≤500)

输出

每组数据一行一个数

输入样例

```
3 6
2 1
3 2
2 3
```

输出样例

```
9
```

C E2-多重背包

时间限制：1000ms 内存限制：65536kb

通过率：187/193 `(96.89%)` 正确率：187/696 `(26.87%)`

题面

有N种物品和一个容量为V的背包。第i种物品最多有m[i]件可用，每件费用是c[i]，价值是w[i]。求解将哪些物品装入背包可使这些物品的费用总和不超过背包容量，且价值总和最大。

输入

多组输入数据

每组数据第一行两个数n，v，表示物品的数量和背包的容量。
(1≤n≤500,1≤v≤300001≤n≤500,1≤v≤30000)

接下来n行，每行三个整数，表示物品的费用,价值,数量(1≤ci,wi≤500,1≤mi≤2001≤ci,wi≤500,1≤mi≤200)

输出

每组数据一行一个数

输入样例

```
2 10
2 1 3
3 2 2
```

输出样例

```
6
```

D E2-组合背包

时间限制：1000ms 内存限制：65536kb

通过率：187/189 `(98.94%)` 正确率：187/296 `(63.18%)`

题面

组合背包：有的物品只可以取一次（01背包），有的物品可以取无限次（完全背包），有的物品可以取的次数有一个上限（多重背包）。

输入

多组输入数据

每组数据第一行两个数n，v，表示物品的数量和背包的容量。
(1≤n≤500,1≤v≤300001≤n≤500,1≤v≤30000)

接下来n行，每行三个整数，表示物品的费用,价值,数量
(1≤ci,wi≤500,1≤mi≤2001≤ci,wi≤500,1≤mi≤200)，如果m等于233 表示物品可以无限取。

输出

每组数据一行一个数

输入样例

```
3 10
2 2 233
2 3 1
3 4 3
```

输出样例

```
13
```

AC Code

下面这个需要改一下顺序...

```
#include<cstdio>
#include<cstring>
#include<deque>
#include<algorithm>
#define full 233
#define maxn 200
#define maxv 200000
using namespace std;
inline void write(int x) {
  if (x < 0)putchar('-'), x = -x;
  if (x > 9)write(x / 10);
  putchar(x % 10 + 48);
}
inline int read() {
  int k = 0, f = 1;
  char c = getchar();
```

```
  while (c < '0' || c>'9') {
    if (c == '-')f = -1;
    c = getchar();
  }
  while (c >= '0' && c <= '9') {
    k = (k << 1) + (k << 3) + c - 48;
    c = getchar();
  }
  return k * f;
}
template <typename T>
struct MonotoneQueue {
  deque<T>q, m;
  inline void push(const T& x) {
    q.push_back(x);
    while (!m.empty() && m.back() < x)m.pop_back();
    m.push_back(x);
  }
  inline void pop() {
    T x = q.front();
    q.pop_front();
    if (x == m.front())m.pop_front();
  }
  inline size_t size() {
    return q.size();
  }
  T top() {
    return m.front();
  }
};
int t;
int V, N;
int value, cost, cnt;
int f[maxv + 10];
inline void buildDP_01Pack(int cost, int value) {
  for (int v = V; v >= cost; --v) {
    f[v] = max(f[v], f[v - cost] + value);
  }
}
inline void buildDP_FullPack(int cost, int value) {
  for (int v = cost; v <= V; ++v) {
    f[v] = max(f[v], f[v - cost] + value);
  }
}
inline void buildDP_MultiPack(int cost, int value, int count) {
  count = min(count, V / cost);
  for (int r = 0; r < cost; ++r) {
    MonotoneQueue<int>q;
    int m = (V - r) / cost;
```

```
    for (int k = 0; k <= m; ++k) {
      if (q.size() == count + 1)q.pop();
      q.push(f[k * cost + r] - k * value);
      f[k * cost + r] = q.top() + k * value;
    }
  }
}
inline void buildDP_CombinedPack(int cost, int value, int count) {
  if (count == 1)buildDP_01Pack(cost, value);
  else if (count == full)buildDP_FullPack(cost, value);
  else buildDP_MultiPack(cost, value, count);
}
int main() {
  t = read();
  while (t--) {
    memset(f, 0, sizeof(f));
    V = read(), N = read();
    while (N--) {
      value = read(), cost = read(), cnt = read();
      buildDP_CombinedPack(cost, value, cnt);
    }
    write(f[V]), putchar('\n');
  }
}
```

E E2-股票I

时间限制：1000ms 内存限制：65536kb

通过率：193/197 `(97.97%)` 正确率：193/482 `(40.04%)`

题面

假设您有一个数组，第i个元素是第i天给定股票的价格。

如果只允许您最多完成一笔交易（即买入和卖出一股股票），请设计一种算法以找到最大的利润(卖出的价格-买入的价格)。

请注意，您不能在买股票之前卖出股票。

输入

多组输入数据

每组数据第一行一个数n。(1≤n≤1051≤n≤105)

接下来一行n个数表示股票的价格(1≤ai≤1091≤ai≤109)

输出

每组数据一行一个数

输入样例

```
5
1 2 3 4 5
```

输出样例

```
4
```

AC Code

```
#include<cstdio>
int main()
{
  int n;
  while(~scanf("%d", &n))
  {
    int min;
    int x;
    scanf("%d", &min);
    int ans = 0;
    for(int i = 1; i < n; i++)
    {
      scanf("%d", &x);
      ans = ans > x - min ? ans : x - min;
      min = min < x ? min : x;
    }
    printf("%d\n", ans);
  }
  return 0;
}
```

F E2-股票II

时间限制：1000ms 内存限制：65536kb

通过率：187/197 `(94.92%)` 正确率：187/556 `(33.63%)`

题面

假设您有一个数组，第i个元素是第i天给定股票的价格。

设计算法以找到最大的利润。您可以根据需要完成尽可能多的交易。

请注意，无法同时进行多项交易（即必须先出售股票才能再次购买）

输入

多组输入数据

每组数据第一行一个数n。(1≤n≤1051≤n≤105)

接下来一行n个数表示股票的价格(1≤ai≤1091≤ai≤109)

输出

每组数据一行一个数

输入样例

```
5
1 2 3 4 5
```

输出样例

```
4
```

AC Code

```
#include<cstdio>
int main()
{
  int n;
  while(~scanf("%d", &n))
  {
    int x, x0;
    scanf("%d", &x0);
    long long ans = 0;
    for(int i = 1; i < n; i++)
    {
      scanf("%d", &x);
      if(x > x0)
        ans += x - x0;
      x0 = x;
    }
    printf("%lld\n", ans);
  }
  return 0;
}
```

G E2-股票III

时间限制：1000ms 内存限制：65536kb

通过率：180/188 `(95.74%)` 正确率：180/485 `(37.11%)`

题面

假设您有一个数组，第i个元素是第i天给定股票的价格。

设计算法以找到最大的利润。您最多可以完成两次交易。

请注意，无法同时进行多项交易（即必须先出售股票才能再次购买）

输入

多组输入数据

每组数据第一行一个数n。(1≤n≤1051≤n≤105)

接下来一行n个数表示股票的价格(1≤ai≤1091≤ai≤109)

输出

每组数据一行一个数

输入样例

```
5
1 2 3 4 5
```

输出样例

```
4
```

AC Code

```
#include<cstdio>
#include<algorithm>
#include<cstring>
using namespace std;
//DP[i][k][0] = Max(DP[i - 1][k][0], DP[i - 1][k - 1][1] + a[i])
//DP[i][k][1] = Max(DP[i - 1][k][1], DP[i - 1][k - 1][0] - a[i])
long long dp[100001][3][2];
int money[100001];
int main()
{
  int n;
  while(~scanf("%d", &n))
  {
    for(int i = 0; i < n; i++)
```

```
    {
      scanf("%d", &money[i]);
    }
    dp[0][0][1] = -money[0];
    dp[0][1][1] = -money[0];
    for(int i = 1; i < n; i++)
    {
      dp[i][0][1] = max(dp[i - 1][0][0] - money[i], dp[i - 1][0][1]);
          dp[i][1][0] = max(dp[i - 1][0][1] + money[i], dp[i - 1][1][0]);
          dp[i][1][1] = max(dp[i - 1][1][0] - money[i], dp[i - 1][1][1]);
          dp[i][2][0] = max(dp[i - 1][1][1] + money[i], dp[i - 1][2][0]);
    }
    printf("%lld\n", dp[n-1][2][0]);
    memset(dp, 0, 6e5*sizeof(int));
  }
  return 0;
}
```

H E2-股票IV

时间限制：1000ms 内存限制：65536kb

通过率：170/186 `(91.40%)` 正确率：170/712 `(23.88%)`

题面

假设您有一个数组，第i个元素是第i天给定股票的价格。

设计算法以找到最大的利润。您最多可以完成k次交易。

请注意，无法同时进行多项交易（即必须先出售股票才能再次购买）

输入

多组输入数据

每组数据第一行两个数n,k，表示总天数和最多交易次数。(1≤n,k≤1031≤n,k≤103)

接下来一行n个数表示股票的价格(1≤ai≤1091≤ai≤109)

输出

每组数据一行一个数

输入样例

```
5 2
1 2 3 4 5
```

输出样例

```
4
```

AC Code

```
#include<cstdio>
#include<algorithm>
#include<cstring>
using namespace std;
//DP[i][k][0] = Max(DP[i - 1][k][0], DP[i - 1][k - 1][1] + a[i])
//DP[i][k][1] = Max(DP[i - 1][k][1], DP[i - 1][k - 1][0] - a[i])
long long dp[1001][1002][2];
int money[1001];
int main()
{
  int n, k;
  while(~scanf("%d%d", &n, &k))
  {
    for(int i = 0; i < n; i++)
    {
      scanf("%d", &money[i]);
    }
    for(int i = 0; i <= k; i++)
      dp[0][i][1] = -money[0];
    for(int i = 1; i < n; i++)
    {
      dp[i][0][1] = max(dp[i - 1][0][1], dp[i - 1][0][0] - money[i]);
          for (int j = 1; j <= k; j++)
      {
              dp[i][j][0] = max(dp[i - 1][j][0], dp[i - 1][j - 1][1] +
money[i]);
              dp[i][j][1] = max(dp[i - 1][j][1], dp[i - 1][j][0] - money[i]);
          }
    }
    printf("%lld\n", dp[n - 1][k][0]);
    memset(dp, 0, 2*1001*1002*sizeof(int));
  }
  return 0;
}
```