ACM-ICPC 模板

—— E.S.Promise

2014/10/17

目录

1. 数据结构

1.1 Splay

```
#include <cstdio>
#include <cstring>
#include <iostream>
#include <vector>
#include <algorithm>

#define MAXN 400010

using namespace std;

struct Node{
    int key, sz, cnt, lazy;
    Node *chd[2], *pa;          //左右儿子和父亲
    Node(){lazy = 0;}
    Node(int x, int y, int z){
        key = x, sz = y, cnt = z;
        lazy = 0;
    }
    void push_up(){
        sz = chd[0]->sz + chd[1]->sz + cnt;
    }
    // 要用 lazy 的时候再写,下例为转置。
    void push_down()
    {
        if(lazy)
        {
            swap(chd[0], chd[1]);
            chd[0]->lazy ^= 1;
            chd[1]->lazy ^= 1;
            lazy = 0;
        }
    }
}nil(0, 0, 0), *NIL = &nil;

struct Splay{                //伸展树结构体类型
    vector<int> vec;
    Node *root;
    int ncnt;                //计算 key 值不同的结点数，注意已经去重了
    Node nod[MAXN];
    // 首先要初始化
    void init(){
        NIL->key = NIL->cnt = NIL->sz = NIL->lazy = 0;
        root = NIL;
        ncnt = 0;
    }
   //旋转操作，d 为 true 表示右旋
```

```
    void rotate(Node *x, bool d){
        Node *y = x->pa;
        y->push_down();
        x->push_down();
        y->chd[!d] = x->chd[d];
        if (x->chd[d] != NIL)
            x->chd[d]->pa = y;
        x->pa = y->pa;
        if (y->pa != NIL){
            if (y == y->pa->chd[d])
                y->pa->chd[d] = x;
            else
                y->pa->chd[!d] = x;
        }
        x->chd[d] = y;
        y->pa = x;
        y->push_up();
        x->push_up();
    }
    // 当 target 为 NIL 时，虽然结点 x 在 NIL 下，但是 root 不是 NIL
    void splay(Node *x, Node *target){        //将 x 伸展到 target 的儿子位置处
        Node *y;
        while (x->pa != target){
            y = x->pa;
            if (x == y->chd[0]){
                if (y->pa != target && y == y->pa->chd[0])
                    rotate(y, true);
                    rotate(x, true);
            }
            else{
                if (y->pa != target && y == y->pa->chd[1])
                    rotate(y, false);
                    rotate(x, false);
            }
        }
        if (target == NIL)
            root = x;
    }
    /******************以上一般不用修改*************/

    // 根据维护信息不同，相应修改. 插入一个值
    void insert(int key){
        if (root == NIL){
            ncnt = 0;
            root = &nod[++ncnt];            // 新结点都指向 NIL
            root->chd[0] = root->chd[1] = root->pa = NIL;
            root->key = key;
            root->sz = root->cnt = 1;
            return;
        }
        Node *x = root, *y;
        while (1)
        {
            x->sz++;
            if (key == x->key){
                x->cnt++;
                x->push_up();
                y = x;
                break;
            }
            else if (key < x->key){
                    if (x->chd[0] != NIL)
                        x = x->chd[0];
                    else{
                        x->chd[0] = &nod[++ncnt];
                        y = x->chd[0];
                        y->key = key;
                        y->sz = y->cnt = 1;
                        y->chd[0] = y->chd[1] = NIL;
                        y->pa = x;
                        break;
                    }
            }
            else{
                if (x->chd[1] != NIL)
                    x = x->chd[1];
                else{
                    x->chd[1] = &nod[++ncnt];
                    y = x->chd[1];
                    y->key = key;
                    y->sz = y->cnt = 1;
                    y->chd[0] = y->chd[1] = NIL;
                    y->pa = x;
                    break;
                }
            }
        }
        splay(y, NIL);
    }
    // 通过 键值 去寻找，以下还有通过第几个元素去寻找。
    Node* search(int key){                    //查找一个值，返回指针
        if (root == NIL)
            return NIL;
        Node *x = root, *y = NIL;
        while (1){
            if (key == x->key){
                y = x;
                break;
            }
            else if (key > x->key){
                if (x->chd[1] != NIL)
                x = x->chd[1];
                else
```

```
                break;
        }
        else{
            if (x->chd[0] != NIL)
                x = x->chd[0];
            else
                break;
        }
    }
    splay(x, NIL);
    return y;
}
//查找最小值，返回指针
Node* searchmin(Node *x){
    Node *y = x->pa;
    while (x->chd[0] != NIL){       //遍历到最左的儿子就是最小值
        x = x->chd[0];
    }
    splay(x, y);
    return x;
}
//删除一个值
void del(int key){
    if (root == NIL)
        return;
    Node *x = search(key), *y;
    if (x == NIL)
        return;
    if (x->cnt > 1){
        x->cnt--;
        x->push_up();
        return;
    }
    else if (x->chd[0] == NIL && x->chd[1] == NIL){
        init();
        return;
    }
    else if (x->chd[0] == NIL){
        root = x->chd[1];
        x->chd[1]->pa = NIL;
        return;
    }
    else if (x->chd[1] == NIL){
        root = x->chd[0];
        x->chd[0]->pa = NIL;
        return;
    }
    y = searchmin(x->chd[1]);
    y->pa = NIL;
    y->chd[0] = x->chd[0];
    x->chd[0]->pa = y;
    y->push_up();
```

```
        root = y;
    }
    int rank(int key){                  //求结点高度
        Node *x = search(key);
        if (x == NIL)
            return 0;
        return x->chd[0]->sz + 1        /* or x->cnt*/;
    }
    Node* findkth(int kth){             //查找第 k 小的值
        if (root == NIL || kth > root->sz)
            return NIL;
        Node *x = root;
        while (1){
            if (x->chd[0]->sz +1 <= kth && kth <= x->chd[0]->sz + x->cnt)
                break;
            else if (kth <= x->chd[0]->sz)
                x = x->chd[0];
            else{
                kth -= x->chd[0]->sz + x->cnt;
                x = x->chd[1];
            }
        }
        splay(x, NIL);
        return x;
    }
    //找第 x 个元素
    Node *getNth(Node *rt, int x)
    {
        if(rt == NIL) return NIL;
        rt->push_down();
        int l = rt->chd[0]->sz;
        if(x <= l)
            return getNth(rt->chd[0], x);
        else if(x == l+1)
            return rt;
        else
            return getNth(rt->chd[1], x-l-1);
    }
    //得到以 rt 为根的子树的序列
    void get_seq(Node *rt)
    {
        if(rt == NIL) return ;
        rt->push_down();
        get_seq(rt->chd[0]);
        vec.push_back(rt->key);
        get_seq(rt->chd[1]);
    }

}sp;

// HDU 3487 CUT 和 FLIP

```

```
int n, m;
int main(){
    while(~scanf("%d%d", &n, &m))
    {
        if(n == -1 && m == -1) break;
        sp.init();
        for(int i = 0; i <= n+1; i ++)
            sp.insert(i);
        char cmd[10];
        while(m --) {
            scanf("%s", cmd);
            if(!strcmp(cmd, "CUT"))
            {
                int l, r, z; scanf("%d%d%d", &l, &r, &z);
                l ++, r ++, z ++;
                Node *x, *y, *p;
                x =sp.getNth(sp.root, l-1), y = sp.getNth(sp.root, r+1);
                sp.splay(x, NIL);
                sp.splay(y, x);
                x = y->chd[0];
                y->chd[0] = NIL;
                y->push_up();
                y = sp.getNth(sp.root, z);
                sp.splay(y, NIL);
                p = sp.getNth(sp.root, z+1);
                sp.splay(p, y);
                p->chd[0] = x;
                x->pa = p;
                p->push_up();
            }
            else
            {
                int l, r; scanf("%d%d", &l, &r);
                l ++; r ++;
                Node *x, *y;
                x = sp.getNth(sp.root, l-1), y = sp.getNth(sp.root, r+1);
                sp.splay(x, NIL);
                sp.splay(y, x);
                x = y->chd[0];
                x->lazy ^= 1;
            }
        }
        sp.vec.clear();
        Node *x, *y;
        x = sp.getNth(sp.root, 1);
        y = sp.getNth(sp.root, n+2);
        sp.splay(x, NIL);
        sp.splay(y, x);
        sp.get_seq(y->chd[0]);
        printf("%d", sp.vec[0]);
        for(int i = 1; i < sp.vec.size(); i ++)
            printf(" %d", sp.vec[i]);
        printf("\n");
    }
    return 0;
}
```

1.2 Treap

```
#include <cstdio>
#include <cstdlib>
#include <iostream>
#include <cstring>

#define MAXN 20010
#define MAXM 60010
#define MAXQ 500010
using namespace std;

class TreapNode
{
public:
    int pri, val, sz;
    TreapNode *chd[2];
    TreapNode(int _val): val(_val)
    {
        chd[0] = chd[1] = NULL;
        pri = rand();
        sz = 1;
    }
    bool operator < (const TreapNode &para) const
    {
        return pri < para.pri;
    }

    int cmp(int x)
    {
        if(x == val)
            return -1;
        return x >= val;
    }
    void maintain()
    {
        sz = 1;
        if(chd[0]) sz += chd[0]->sz;
        if(chd[1]) sz += chd[1]->sz;
    }
```

```
};

void rotate(TreapNode *&rt, int d)
{
    TreapNode *k = rt->chd[d^1];
    rt->chd[d^1] = k->chd[d];
    k->chd[d] = rt;
    rt->maintain();   //先子树 maintain，此时 k 相对于 rt 已变成 root
    k->maintain();
    rt = k;
}

void insert(TreapNode *&rt, int x)
{
    if(rt == NULL)
        rt = new TreapNode(x);
    else
    {
        int d = (x >= rt->val);
        insert(rt->chd[d], x);
        if(rt->chd[d]->pri > rt->pri)
            rotate(rt, 1^d);
    }
    rt->maintain();
}

void remove(TreapNode *&rt, int x)
{
    int d = rt->cmp(x);
    if(d == -1)
    {
        TreapNode *u = rt;
        if(rt->chd[0] && rt->chd[1])
        {
            int d2 = (rt->chd[0]->pri > rt->chd[1]->pri);
            rotate(rt, d2);
            remove(rt->chd[d2], x);
        }
        else
        {
            if(rt->chd[0] == NULL)
                rt = rt->chd[1];
            else
                rt = rt->chd[0];
            delete u;
        }
    }
    else
        remove(rt->chd[d], x);
    if(rt != NULL)
        rt->maintain();
}

int kth(TreapNode *rt, int k)
{
    if(rt == NULL || k < 0 || k > rt->sz)
        return 0;
    int s = rt->chd[1] == NULL ? 0 : rt->chd[1]->sz;
    if(k == s + 1)
        return rt->val;
    else if(k <= s)
        return kth(rt->chd[1], k);
    else
        return kth(rt->chd[0], k-s-1);
}
```

1.3 矩形面积并

```
#include <cstdio>
#include <iostream>
#include <cstring>
#include <cmath>
#include <string>
#include <queue>
#include <map>
#include <vector>
#include <algorithm>
#include <set>

#define DEBUG 0
#define MP make_pair
#define lson id<<1, l, mid
#define rson id<<1|1, mid+1, r
#define A first
#define B second
#define INF 0x3fffffff
#define OUTSTARS    printf("****************************\n");
#define MAXN 1005

using namespace std;

typedef long long LL;
typedef pair<pair<double, double>, pair<double, int> > PDDDI;

using namespace std;

vector<PDDDI> seg;
set<double> DX;
vector<double> D;
int cnt[MAXN];
double S[MAXN];

void push_up(int id, int l, int r)
{
    if(cnt[id])
        S[id] = D[r] - D[l - 1];
    else if(l == r)
        S[id] = 0;
    else
        S[id] = S[id<<1] + S[id<<1|1];
}

void update(int id, int l, int r, int x, int y, int c)
{
    if(x <= l && y >= r) {
        cnt[id] += c;
        push_up(id, l, r);
        return ;
    }
    int mid = (l + r) >> 1;
    if(x <= mid) update(lson, x, y, c);
    if(y > mid) update(rson, x, y, c);
    push_up(id, l, r);
}

int main()
{
    int n, t = 1;
    double x1, y1, x2, y2;
    while(~scanf("%d", &n))
    {
        if(!n) break;
        DX.clear();
        seg.clear();
        memset(cnt, 0, sizeof(cnt));
        memset(S, 0, sizeof(S));
        for(int i = 0; i < n; i ++) {
            scanf("%lf%lf%lf%lf", &x1, &y1, &x2, &y2);
            DX.insert(x1);
            DX.insert(x2);
            seg.push_back(MP(MP(y1, x1), MP(x2, 1)));
            seg.push_back(MP(MP(y2, x1), MP(x2, -1)));
        }
        D = *(new vector<double>(DX.begin(), DX.end()));
        sort(seg.begin(), seg.end());
        double ans = 0, last;
        for(int i = 0; i < 2 * n; i ++) {
            if(i) ans += S[1] * (seg[i].A.A - last);
            int l = lower_bound(D.begin(), D.end(), seg[i].A.B) - D.begin();
            int r = lower_bound(D.begin(), D.end(), seg[i].B.A) - D.begin();
            update(1, 1, D.size(), l+1, r, seg[i].B.B);
            last = seg[i].A.A;
        }
        printf("Test case #%d\nTotal explored area: %.2lf\n\n", t++ , ans);
    }
    return 0;
}
```

1.4 Heap

```
#include <iostream>
using namespace std;
#define MAX_LONG 2147483647

class binary_heap
{
        protected:
        long cnt;
        long h[100000];
        binary_heap() {cnt=0;h[0]=0;}//其实 h[0]没用
        long pre(long n) {return n>>1;}
        long lch(long n) {return n<<1;}
        long rch(long n) {return (n<<1)+1;}

        public:
        long size(){return cnt;}
};
```

```
class maxheap:public binary_heap
{
        public:
        void ins(long n)
        {
                int p=++cnt;//元素个数+1，p 指向堆底
                while(p>1 && n>h[pre(p)])//若没到堆顶 且 待插入元素大于 p 的父亲
                        {
                          h[p]=h[pre(p)];//则把 p 的父亲移到他儿子的地方(就是 p 的地方)
                          p=pre(p);//p 指向它的父亲
                        }
                h[p]=n;//最后把待插入元素放入 p 的地方
        }
        long pop()
        {
             if(cnt==0) return -MAX_LONG;//返回-MAX_LONG 视为错误
             long p=1;
             long ans=h[1];
             long tmp=h[cnt--];//取出堆底元素然
             long to;
             while(p<=cnt)//若未到栈底
                   {
                     to=-MAX_LONG;//init
                     if(lch(p)<=cnt && tmp<h[lch(p)])
                   //若有左孩子且比取出的栈底元素大
                       to=lch(p);
                     if(rch(p)<=cnt && tmp<h[rch(p)] && h[to]<h[rch(p)])
                       to=rch(p);
                     if(to!=-MAX_LONG) {h[p]=h[to];p=to;}
                     else{h[p]=tmp;break;}
                   }

             return ans;
        }
};
class minheap:public binary_heap
{
        public:
        void ins(long n)
        {
                int p=++cnt;//元素个数+1，p 指向堆底
                while(p>1 && n<h[pre(p)])//若没到堆顶 且 待插入元素小于 p 的父亲
                        {
                          h[p]=h[pre(p)];//则把 p 的父亲移到他儿子的地方(就是 p 的地方)
                          p=pre(p);//p 指向它的父亲
                        }
                h[p]=n;//最后把待插入元素放入 p 的地方
        }
        long pop()
        {
             if(cnt==0) return -MAX_LONG;//返回-MAX_LONG 视为错误
             long p=1;
             long ans=h[1];
             long tmp=h[cnt--];//取出堆底元素然
             long to;
             while(p<=cnt)//若未到栈底
```

```
                   {
                     to=-MAX_LONG;//init
                     if(lch(p)<=cnt && tmp>h[lch(p)])//若有左孩子且比取出的栈底元素小
                        to=lch(p);
                     if(rch(p)<=cnt && tmp>h[rch(p)] && h[to]>h[rch(p)])//若有右孩子 且 比取出的栈底元素小 且 小于左孩子
                        to=rch(p);
                     if(to!=-MAX_LONG)  {h[p]=h[to];p=to;}//如果比左右孩子都小，就说明找对位置了，把取出的栈底放上去；否则交换
                     else{h[p]=tmp;break;}
                   }

             return ans;
        }
};

minheap h1;

int main()
{
    long tmp;
    while(cin >>tmp)
          h1.ins(tmp);
    cout <<endl<<"SIZE:"<<h1.size()<<endl<<endl;
    for(long i=h1.size();i>0;i--)
        cout <<h1.pop()<<endl;
    //system("pause");
}
```

1.5 RMQ

```
#include <cstdio>
#include <algorithm>
using namespace std;
const int MAXN = 50010;
int dp[MAXN][20];
int lg2[MAXN];          //求 log2

//初始化 RMQ, b 数组下标从 1 开始，从 0 开始简单修改
void initRMQ(int n, int b[])
{
    lg2[0] = -1;
    for(int i = 1; i <= n;i++)
    {
        lg2[i] = ((i&(i-1)) == 0)?lg2[i-1]+1:lg2[i-1];
        dp[i][0] = b[i];
    }
    for(int j = 1; j <= lg2[n];j++)
    for(int i = 1;i + (1<<j) -1 <= n;i++)
        dp[i][j] = max(dp[i][j-1],dp[i+(1<<(j-1))][j-1]);
}
//查询最大值
int rmq(int x,int y)
{
    int k = lg2[y-x+1];
    return max(dp[x][k], dp[y-(1<<k)+1][k]);
}

// 测试
```

```
int num[MAXN];
int main()
{
    for(int i = 1; i <= 10; i ++)
        num[i] = i;
    initRMQ(10, num);
    printf("%d\n", rmq(1, 1));
    return 0;
}
```

2 字符串

2.1 Kmp

```
#include <cstring>
#include <cstdio>
#include <iostream>
#include <cmath>

#define MAXN 1000005

using namespace std;

char str[MAXN], T[MAXN];
int next[MAXN];
int lenT, lenS;

void get_next(char *T,int lenT)
{
    next[0] = -1;           //next[i]=j,表示  str[0..j]=str[i-j..i]
    int j = next[0];
    for(int i = 1; i < lenT; i++)
    {
        while(j >= 0 && T[i] != T[j+1]) j = next[j];
        if(T[i] == T[j+1]) j ++;
        next[i] = j;
    }
}

/**
*     cnt  可用来计算出现次数
*     或者只返回第一次出现的下标
*/
int kmp(char *S, char *T)
{
    int j=next[0];
    int cnt = 0;
    for(int i = 0; i < lenS; i ++)
    {
        while(j >= 0 && S[i] != T[j+1]) j = next[j];
        if(S[i] == T[j+1]) j ++;
        if(j == lenT - 1)
        {
            return i - lenT + 1;
      //  如果要返回出现次数，或是记录多次出现的位置，不用 return
            j=next[j];
            cnt ++;
        }
    }
}
//     求循环节
int main()
{
    while(scanf("%s", T))
    {
        if(T[0] == '.') break;
        lenT = strlen(T);
        get_next(T, lenT);
        int times = sqrt(lenT);
        int ans = 1;
        int tp = lenT - 1 - next[lenT-1];
        if(lenT % tp == 0)
            ans = lenT / tp;

        printf("%d\n", ans);
    }
    return 0;
}
/**
当模式串为: abababcabcabb next[]={-1, -1, 0, 1, 2, 3, -1, 0, 1, -1, 0, 1, -1 }
*/
```

2.2 exKMP

2.3 后缀数组 （DA）

```
#include <iostream>
#include <cstdio>
#include <cstring>
#include <algorithm>

#define MAXN 1000005

using namespace std;

/**
* rank  下标从 0 开始，  值 0 - n-1  的排列
* sa     从 1 开始,因为最后一个字符(最小的)排在第 0 位
* high  从 1 开始,因为表示的是 sa[i-1]和 sa[i]
*/

int rank[MAXN], sa[MAXN], X[MAXN], Y[MAXN], high[MAXN], init[MAXN];
int buc[MAXN];

void calhigh(int n)
{
    int i , j , k = 0;
    for(i = 1; i <= n; i++) rank[sa[i]] = i;
    for(i = 0; i < n; high[rank[i++]] = k)
        for(k ? k-- : 0, j = sa[rank[i]-1]; init[i+k] == init[j+k]; k++)
            ;
}

bool cmp(int *r,int a,int b,int l)
```

```
{
    return (r[a] == r[b] && r[a+l] == r[b+l]);
}

void suffix(int n,int m = 128) {

    int i , l , p , *x = X , *y = Y;
    for(i = 0 ; i < m ; i ++) buc[i] = 0;
    for(i = 0 ; i < n ; i ++) buc[ x[i] = init[i]    ] ++;
    for(i = 1 ; i < m ; i ++) buc[i] += buc[i-1];
    for(i = n - 1; i >= 0 ; i --) sa[ --buc[ x[i] ]] = i;
    for(l = 1,p = 1 ; p < n ; m = p , l *= 2)
    {
        p = 0;
        for(i = n-l ; i < n ; i ++) y[p++] = i;
        for(i = 0 ; i < n ; i ++) if(sa[i] >= l) y[p++] = sa[i] - l;
        for(i = 0 ; i < m ; i ++) buc[i] = 0;
        for(i = 0 ; i < n ; i ++) buc[ x[y[i]] ] ++;
        for(i = 1 ; i < m ; i ++) buc[i] += buc[i-1];
        for(i = n - 1; i >= 0 ; i --) sa[ --buc[ x[y[i]] ] ] = y[i];
        for(swap(x,y) , x[sa[0]] = 0 , i = 1 , p = 1 ; i < n ; i ++)
            x[ sa[i] ] = cmp(y,sa[i-1],sa[i],l) ? p-1 : p++;
    }
    calhigh(n-1);               //后缀数组关键是求出 high,所以求 sa 的时候顺便把 rank 和 high 求出来
}

int dp[MAXN][20];
int lg2[MAXN];          //求  log2

//初始化 RMQ, b 数组下标从 1 开始，从 0 开始简单修改
void initRMQ(int n, int b[])
{
    lg2[0] = -1;
    for(int i = 1; i <= n; i++)
    {
        lg2[i] = ((i&(i-1)) == 0)?lg2[i-1]+1:lg2[i-1];
        dp[i][0] = b[i];
    }
    for(int j = 1; j <= lg2[n]; j++)
    for(int i = 1; i + (1<<j) -1 <= n; i++)
        dp[i][j] = min(dp[i][j-1], dp[i+(1<<(j-1))][j-1]);
}
//查询最大值
int rmq(int x,int y)
{
    int k = lg2[y-x+1];
    return min(dp[x][k], dp[y-(1<<k)+1][k]);
}

//  如果要求  lcp  先初始化  ST  表
//  询问直接用  rmq(int x, int y)
void cal_lcp(int n)
{
    initRMQ(n, high);
}


/*********************************************
**    n 为数组长度,下标 0 开始
**    将初始数据,保存在 init 里,并且保证每个数字都比 0 大
**    m = max{ init[i] } + 1
**    一般情况下大多是字符操作,所以 128 足够了
*******************************************/

int num[MAXN];
int n, k;
int cnt, idx[1000005];

/**
*     check:  分组 check 函数  注意：i <= n  即使这里  high[n]的值为 0，且无实际意义，也不能漏。
*     因为当  i == n  的时候即是检查最后一个分组。
*/

bool check(int x)
{
    int cnt = 1;
    for(int i = 1; i <= n; i ++)
    {
        if(high[i] < x)
        {
            if(cnt >= k) return true;
            cnt = 1;
        }
        else
            cnt ++;
    }
    return false;
}


void test()
{
    char str[10] = {"aabaaaab"};
    int len = strlen(str);

    for(int i = 0; i < len; i ++)
        init[i] = str[i];
    init[len] = 0;
    suffix(len+1);
    for(int i = 1; i <= len; i ++)
        printf("%d ", high[i]);
    printf("\n");
    cal_lcp(len);
    printf("lcp(1, 6)=%d\n", rmq(1, 6));
}

int main()
{

    test();

    while(~scanf("%d%d", &n, &k))
    {
        cnt = 1;
        for(int i = 0; i < n; i ++) {
            scanf("%d", num+i);
            init[i] = num[i];
        }
```

```
            sort(num, num+n);
            idx[num[0]] = cnt;
            for(int i = 1; i < n; i ++)
                idx[num[i]] = num[i] == num[i-1] ? cnt : ++ cnt;
            for(int i = 0; i < n; i ++)
                init[i] = idx[init[i]];
            init[n++]  =  0;                    // 最后一位补 0    n 表示 0 的下一位（用来分组） 实际只要用到 n-1
            suffix(n, cnt+1);                   // n  必须是  最后一位 0 的下一位
            int bi;
            int l = 1, r = n, ans = 1;
            while(l <= r)
            {
                bi = (l + r) >> 1;
                bool res = check(bi);
                if(res)
                {
                    l = bi + 1;
                    ans = max(ans, bi);
                }
                else
                    r = bi - 1;
            }
            printf("%d\n", ans);
        }

    return 0;
}

/**
*     sa[1-len]         = {3 4 5 0 6 1 7 2};
*     rank[0-len-1] = {4 6 8 1 2 3 5 7};
*     high[1-len]      = {0 3 2 3 1 2 0 1};
*
*     aaaab
*     aaab
*     aab
*     aabaaaab
*     ab
*     abaaaab
*     b
*     baaaab
*/
```

后缀数组（DC3）

```
#include <cstdlib>
#include <cstdio>
#include <iostream>
#include <cstring>
#include <string>
#include <cmath>
#include <algorithm>

#define F(x) ((x) / 3 + ((x) % 3 == 1 ? 0 : tb))
#define G(x) ((x) < tb ? (x) * 3 + 1 : ((x) - tb) * 3 + 2)
#define cmp1(r, a, b) (r[a] == r[b] && r[a+1] == r[b+1] && r[a+2] == r[b+2])
```

```
#define cmp3(r, a, b) (r[a] < r[b] || r[a] == r[b] && wv[a+1] < wv[b+1])
#define cmp2(k, r, a, b) (k == 2 ? (r[a] < r[b] || r[a] == r[b] && cmp3(r, a+1, b+1)):cmp3(r, a, b))

using namespace std;
typedef long long LL;
const int M = 20;
const int N = (1<<M);

/**
*   sa 数组从 sa[1]到 sa[n]，存储的是 0 到 n-1 的排列
*   rank 数组从 rank[0]到 rank[n-1]，存储的是 1 到 n 的排列
*   rank[i]记录的是以 i 为起点的后缀的排名
*   high[i]记录  lcp(i, i-1)
*/

class SA
{
public:
    int rank[N], sa[3*N], init[3*N], high[N], n;
    int buc[N], m, wv[N], i, j, k;
    int log[N], rmq[M][N];
    SA()
    {
        log[0] = -1;
        for(int i = 1; i < N; i ++)
            log[i] = (i & (i-1)) ? log[i-1] : log[i-1] + 1;
    }

    inline void sort(int *r, int *a, int *b, int n, int m)
    {
        for(i = 0; i < n; i ++) wv[i] = r[a[i]];
        for(i = 0; i < m; i ++) buc[i] = 0;
        for(i = 0; i < n; i ++) buc[wv[i]] ++ ;
        for(i = 1; i < m; i ++) buc[i] += buc[i-1];
        for(i = n - 1; i >= 0; i --) b[-- buc[wv[i]]] = a[i];
        return ;
    }

    inline void suffix_dc3(int *r, int *sa, int n, int m)
    {
        int *rn = r + n;
        int *san = sa + n, ta = 0, tb = (n + 1) / 3;
        int tbc = 0, p, *wa = rank, *wb = high;
        r[n] = r[n+1] = 0;
        for(int i = 0; i < n; i ++)
            if(i % 3 != 0) wa[tbc++] = i;
        sort(r+2, wa, wb, tbc, m);
        sort(r+1, wb, wa, tbc, m);
        sort(r, wa, wb, tbc, m);
        for(p = 1, rn[F(wb[0])] = 0, i = 1; i < tbc; i ++)
            rn[F(wb[i])] = cmp1(r, wb[i-1], wb[i]) ? p - 1 : p ++ ;
        if(p < tbc)
            suffix_dc3(rn, san, tbc, p);
        else
            for(i = 0; i < tbc; i ++)
                san[rn[i]] = i;
        for(i = 0; i < tbc; i ++)
            if(san[i] < tb)
                wb[ta++] = san[i] * 3;
        if(n % 3 == 1)
            wb[ta++] = n - 1;
        sort(r, wb, wa, ta, m);
        for(i = 0; i < tbc; i ++)
            wv[wb[i] = G(san[i])] = i;
        for(i = 0, j = 0, p = 0; i < ta && j < tbc; p ++)
            sa[p] = cmp2(wb[j] % 3, r, wa[i], wb[j]) ? wa[i++] : wb[j++];
        for( ; i < ta; sa[p++] = wa[i++])
            ;
        for( ; j < tbc; sa[p++] = wb[j++])
            ;
    }

    inline int exec(char *in)
    {
        for(int &p = n = m = 0; in[p]; p ++)
        {
            init[p] = in[p];
            m = max(m, init[p]+1);
        }
        init[n] = 0;
        suffix_dc3(init, sa, n+1, m);
        for(i = 1; i <= n; i ++)
            rank[sa[i]] = i;
        for(i = 0, k = 0; i < n; high[rank[i++]] = k)
            for(k ? k -- : 0, j = sa[rank[i]-1]; init[i+k] == init[j+k]; k ++)
                ;
        for(i = 1; i <= n; i ++)
            rmq[0][i] = high[i];
        for(int i = 1; i <= log[n]; i ++)
            for(int j = 1; j <= n-(1<<i)+1; j ++)
                rmq[i][j] = min(rmq[i-1][j], rmq[i-1][j+(1<<i>>1)]);
        return n;
    }

    /** lcp(rank[i],rank[j])  询问 i,j 后缀的最长公共前缀  */
    inline int lcp(int a, int b)
    {
        if(a == b) return n - sa[a];
        if(a > b)
            swap(a, b);
        int t = log[b-a];
        return min(rmq[t][a+1], rmq[t][b-(1<<t)+1]);
    }
} sa;

char str[N] = {"aabaaaab"};
int len = 8;
int main()
{
    sa.exec(str);
    freopen("out.txt", "w", stdout);

    for(int i = 1; i <= len; i ++)
        printf("%s\n", str+sa.sa[i]);

    for(int i = 1; i <= len; i ++)
        printf("%d ", sa.sa[i]);
    printf("\n");
    for(int i = 0; i < len; i ++)
```

```
        printf("%d ", sa.rank[i]);
    printf("\n");
    for(int i = 1; i <= len; i ++)
        printf("%d ", sa.high[i]);

    return 0;
}

/**
 *  sa[1-len]      = {3 4 5 0 6 1 7 2};
 *  rank[0-len-1] = {4 6 8 1 2 3 5 7};
 *  high[1-len]    = {0 3 2 3 1 2 0 1};
 *
 *  aaaab
 *  aaab
 *  aab
 *  aabaaaab
 *  ab
 *  abaaaab
 *  b
 *  baaaab
 */
```

2.4 AC 自动机（array）

```
#include <iostream>
#include <cstdio>
#include <cstring>
#include <queue>

#define MAXN 500005
#define CHILDREN 26
using namespace std;

class ACAutomaton
{
public:
    int sz;
    int val[MAXN];
    int fail[MAXN];
    int next[MAXN][CHILDREN];
    int ID[300];
    queue<int> que;

    /**  初始化  ID  映射  */
    ACAutomaton()
    {
        fail[0] = 0;
        sz = 1;
    }

    /**  初始化 AC 机，如 sz, val[], queue */
    void reset()
    {
        sz = 1;
        memset(next[0], 0, sizeof(next[0]));
    }

    void insert(char *str, int d)
    {
        int tp = 0;
        for(int i = 0; str[i]; i ++)
        {
            int idx = str[i] - 'a';       //取映射，如果有 ID[]，则 ID[str[i]]
            if(!next[tp][idx])
            {
                //printf("%c %d", str[i], sz);
                next[tp][idx] = sz;
                memset(next[sz], 0, sizeof(next[sz]));
                val[sz++] = 0;
            }
            tp = next[tp][idx];
        }
        val[tp] += d;                     // 考虑重复模式串
    }

    void build()
    {
        que.push(0);
        while(!que.empty())
        {
            int cur = que.front(); que.pop();
            for(int i = 0; i < CHILDREN; i ++)
            {
                int &tp = next[cur][i];
                if(tp)
                {
                    que.push(tp);
                    fail[tp] = cur ? next[fail[cur]][i] : 0;
                    // 判断 cur 是否为 root
                }
                else if(cur)
                    tp = next[fail[cur]][i];
            }
        }
    }

    /** HDU 2222 */
    int solve(char *str)
    {
        int ret = 0, tp = 0;
        for(int i = 0; str[i]; i ++)
        {
            int idx = str[i]-'a';          //OR ID[str[i]];
            while(tp && !next[tp][idx])
                tp = fail[tp];
            if(next[tp][idx])
                tp = next[tp][idx];
            for(int cur = tp; cur && val[cur] != -1; cur = fail[cur])
            {
                ret += val[cur];
                val[cur] = -1;
            }
        }
        return ret;
    }

}ac;
char str[55], main_str[1000005];
```

```
int main()
{
        int c; scanf("%d", &c);
        while(c --)
        {
                int n; scanf("%d", &n);
                ac.reset();
                while(n --)
                {
                        scanf("%s", str);
                        ac.insert(str, 1);
                }
                ac.build();
                scanf("%s", main_str);
                printf("%d\n", ac.solve(main_str));
        }
        return 0;
}
```

2.5 AC 自动机 （pointer）

```
#include <cstdio>
#include <cstring>
#include <iostream>

#define CHILDREN 26
#define MAXN 1000005

struct Node {
        int cnt;                                        //是否为单词最后一个节点
        Node *next[CHILDREN];
        Node *fail;
        Node() {
                fail = NULL;
                cnt = 0;
                memset(next, NULL, sizeof(next));
        }
};

class ACAutomaton
{
public:
        Node *root;
        Node *que[MAXN];                                //队列，bfs 构造失败指针
        int head, rear;
        void reset()
        {
                head = rear = 0;
                root = new Node();
        }

        void insert(char *str)
        {
                Node *tp = root;
                int idx;
                for(int i = 0; str[i]; i ++)
                {
                        int idx = str[i] - 'a';
                        if(tp->next[idx] == NULL)
                tp->next[idx] = new Node();
            tp = tp->next[idx];
        }
        tp->cnt ++;
    }

    void build()
    {
        root->fail = NULL;
        que[rear++] = root;
        while(head < rear)
        {
            Node *tp = que[head++];
            Node *p = NULL;
            for(int i = 0; i < 26; i ++)
            {
                if(tp->next[i])
                {
                    if(tp == root)
                        tp->next[i]->fail = root;
                    else
                    {
                        p = tp->fail;
                        while(p) {
                            if(p->next[i])
                            {
                                tp->next[i]->fail = p->next[i];
                                break;
                            }
                            p = p->fail;
                        }
                        if(!p)
                            tp->next[i]->fail = root;
                    }
                    que[rear ++] = tp->next[i];
                }
            }
        }
    }

    /** HDU 2222
    **    输入 n 个模式串，一个主串，AC 机跑一遍，找出现了多少个单词，一个单词最多只出现一次
    **/
    int solve(char *str)
    {
        int ret = 0, idx;
        Node *p = root;
        for(int i = 0; str[i]; i ++)
        {
            idx = str[i] - 'a';
            while(!p->next[idx] && p != root)
                p = p->fail;
            p = p->next[idx];
            p = !p ? root : p;
            Node *tp = p;
            while(tp != root && tp->cnt != -1)
            {
                ret += tp->cnt;
                tp->cnt = -1;
                tp = tp->fail;
            }
        }
        return ret;
    }

    /**  测试用.. */
    char for_print[30];
    void print(Node *cur, char *str, int idx)
    {
        for(int i = 0; i < CHILDREN; i ++)
        {
            if(cur->next[i])
            {
                str[idx] = (char)(i+'a');
                if(cur->next[i]->cnt == 1) {
                    str[idx+1] = '\0';
                    printf("%s\n", str);
                    cur->next[i]->cnt = 0;
                }
                print(cur->next[i], str, idx+1);
            }
        }
    }

};
char main_str[MAXN];
ACAutomaton ac;
int main()
{
    int t; scanf("%d", &t);
    while(t --)
    {
        int n;
        scanf("%d", &n);
        getchar();

        ac.reset();
        for(int i = 0; i < n; i ++)
        {
            char str[55];
            scanf("%s", str);
            ac.insert(str);
        }
        //ac.print(ac.root, ac.for_print, 0);
        ac.build();
        scanf("%s", main_str);
        printf("%d\n", ac.solve(main_str));
        ac.del(ac.root);
    }
    return 0;
}
```

2.6 字符串 Hash

```
unsigned int BKDRHash(char *str)
{
    unsigned int seed = 131; // 31 131 1313 13131 131313 etc..
    unsigned int hash = 0;

    while (*str)
    {
        hash = hash * seed + (*str++);
    }

    return (hash & 0x7FFFFFFF) % MOD;
}
```

3 图论

3.1 SPFA

```
#include <cstdio>
#include <iostream>
#include <cstring>
#include <cmath>
#include <string>
#include <queue>
#include <map>
#include <vector>
#include <algorithm>
#define DEBUG 0
#define INF 0x1fffffff
#define MAXS 105

typedef long long LL;
using namespace std;

struct Edge {
    int v, w;
    Edge() {}
    Edge(int vv, int ww) {v = vv; w = ww; }
};
vector<Edge> ver[MAXS];
int n, m;
int inq[MAXS], times[MAXS], dis[MAXS], gra[MAXS][MAXS];

bool creat_graph()
{
    scanf("%d%d", &n, &m);
    if(n == 0 && m == 0) return false;
    for(int i = 1; i <= n; i ++) ver[i].clear();
    for(int i = 1; i <= m; i ++)
    {
        int u, v, w;
        scanf("%d%d%d", &u, &v, &w);
        ver[u].push_back(Edge(v, w));
        ver[v].push_back(Edge(u, w));
    }
    return true;
}

bool creat_graph_juzhen()
{
    scanf("%d%d", &n, &m);
    if(n == 0 && m == 0) return false;
    for(int i = 1; i <= n; i ++)
```

```
            for(int j = 1; j <= n; j ++)
                gra[i][j] = INF;

        for(int i = 1; i <= m; i ++)
        {
            int u, v, w;
            scanf("%d%d%d", &u, &v, &w);
            gra[u][v] = gra[v][u] = w;
        }
        return true;
}


bool spfa(int s)
{
        for(int i = 0; i <= n; i ++)
        {
            dis[i] = INF;
            inq[i] = times[i] = 0;
        }
        queue<int> q;
        //while(!q.empty()) q.pop();
        q.push(s);
        dis[s] = 0;
        inq[s] = 1;
        while(!q.empty())
        {
            int cur = q.front(); q.pop();
            inq[cur] = 0;
            times[cur] ++;
            if(times[cur] > n) return false;      // negative?
            ///** Vector graph.
            for(int i = 0; i != ver[cur].size(); i ++)
            {
                int v = ver[cur][i].v, w = ver[cur][i].w;
                if( dis[v] > dis[cur] + w) {
                    dis[v] = dis[cur] + w;
                    if(!inq[v])
                    {
                        inq[v] = 1;
                        q.push(v);
                    }
                }
            }
            /** JuZhenal graph
            for(int i = 1; i <= n; i ++)
            {
                if(dis[i] > dis[cur] + gra[cur][i])
                    dis[i] = dis[cur] + gra[cur][i];
                if(!inq[i]) {
                    inq[i] = 1;
                    q.push(i);
                }
            }*/
        }
        return true;
}

int main()
{
    while(creat_graph())
    {
        spfa(1);
        printf("%d\n", dis[n]);
    }
    return 0;
}
```

3.2 Floyd(改进版：可求最小环权值)

```
#include <cstdio>
#include <algorithm>
#define DEBUG 0
#define INF 0x3f3f3f3f
#define MAXN 1000

typedef long long LL;
using namespace std;

int n, m;
int pre[MAXN][MAXN], dis[MAXN][MAXN], gra[MAXN][MAXN];

/**  返回值为最小环权值. */
int Floyd(int n) {
    int minCircle = INF;     /**  改进后的 Floyd 可求最小环。minCircle 用于记录最小环权值。  */

    for(int k = 0; k < n; k ++) {
        /**  改进部分  求最小环权值. */
        for(int i = 0; i < k; i ++)
            for(int j = 0; j < i;j ++)
                minCircle = min(minCircle, dis[i][j] + gra[i][k] + gra[k][j]);

        /**  通常部分。  */
        for(int i = 0; i < n; i ++) {
            for(int j = 0; j < i; j ++) {
                int tmp = dis[i][k] + dis[k][j];
                if(tmp < dis[i][j]) {
                    dis[i][j] = dis[j][i] = tmp;
                    pre[i][j] = pre[j][i] = k;
                }
            }
        }
    }
    return minCircle;
}

void init()
{
    for(int i = 0; i < n; i ++) {
        for(int j = 0; j < n; j ++) {
            dis[i][j] = INF;
            pre[i][j] = j;
        }
    }
}

int main()
{
    while(scanf("%d", &n))
    {
        init();
        for(int i = 0; i < n; i ++) {
            for(int j = 0; j < n; j ++) {
                scanf("%d", &dis[i][j]);
                if(dis[i][j] == 0) dis[i][j] = INF;
            }
        }
        int x, y;
        Floyd(n);
        scanf("%d%d", &x, &y);

        int swaped = 0;
        if(x < y) {
            swap(x, y);
            swaped = 1;
        }

        int t, path[100], cnt = 1;
        path[0] = x;
        t = pre[x][y];
        while(t != y) {
            path[cnt ++] = t;
            t = pre[t][y];
        } path[cnt ++] = y;
        if(dis[x][y] == INF) {
            printf("NO PATH\n");
        }
        else if(!swaped) {
            printf("distance: %d\n", dis[y][x]);
            printf("Path:\n");
            for(int i = 0; i < cnt; i ++)
                printf("%d ", path[i]);
        } else {
            printf("distance: %d\n", dis[y][x]);
            printf("Path:\n");
            for(int i = cnt - 1; i >= 0; i --)
                printf("%d ", path[i]);
        }
    }
    return 0;
}

/**
*     求当前最小环权值在更新 dis 之前是因为。。当前的 k 不应该被连入 dis[i][j]中。
*  也就是说当前迭代求 minCircle 是对之前已求 dis[i][j]包含 k-1 以前的点的最短路，
*  不能包含 k！ 如果包含 k（也就是说，如果求 minCircle 的代码在更新 dis 后面的话，
*  dis[i][j]就已经包含了 k），那么这次更新 minCircle 时
*  minCircle = min(minCircle, dis[i][j] + gra[i][k] + gra[k][j]); 就已经无效了
*  因为 gra[i][k]和 gra[k][j]很可能已经被加入到这个最短路了。这样的话就无法构成环了。
*/
```

3.3 Tarjan (无向图求割点)

```
#include <cstdio>
#include <iostream>
#include <cstring>
#include <cmath>
#include <string>
#include <queue>
#include <map>
#include <vector>
#include <algorithm>
#define DEBUG 0
#define INF 0x3fffffff
#define MAXS 1005

typedef long long LL;
using namespace std;

int dfn[MAXS], low[MAXS], cut[MAXS];
int n, m, root;     /** n 点，m 边，root 根. */
vector<int> ver[MAXS];
int DFN;


/**  邻接表(ver)存储的图的 tarjan：  （无向图求割点）  */
void tarjan_cut(int u, int fa)
{
        int son = 0;
        dfn[u] = low[u] = ++DFN;
        for(int i = 0; i != ver[u].size(); i ++)
        {
                int v = ver[u][i];
                if(!dfn[v]) {
                        tarjan_cut(v, u);
                        son ++;
                        low[u] = min(low[u], low[v]);
                        if((u == root && son > 1) || (u != root && dfn[u] <= low[v]))
                                cut[u] = 1;
                } else if(dfn[u] > dfn[v] && v != fa) {
                        low[u] = min(low[u], dfn[v]);
                }
        }
}


int bridge[MAXS][MAXS];
void tarjan_bridge(int u, int fa)
{
        int v;
        dfn[u] = low[u] = ++DFN;
        for(int i = 0; i != ver[u].size(); i ++)
        {
                v = ver[u][i];
                if(!dfn[v])
                {
                        tarjan_bridge(v, u);
                        low[u] = min(low[u], low[v]);
                        if(low[v] > dfn[u])
                        {
                                bridge[u][v] = bridge[v][u] = 1;
```

```
            }
        }
        else if(dfn[v] < dfn[u] && v != fa)
        {
            low[u] = min(low[u], dfn[v]);
        }
    }
}


/**  初始化. */
void init()
{
    DFN = 0;
    memset(dfn, 0, sizeof(dfn));
    memset(low, 0, sizeof(low));
    memset(bridge, 0, sizeof(bridge));
    for(int i = 1; i <= n; i ++)
    {
        ver[i].clear();
    }
}

void creat_graph()
{
    scanf("%d%d", &n, &m);
    for(int i = 1; i <= m; i ++)
    {
        int u, v;
        scanf("%d%d", &u, &v);
        ver[u].push_back(v);
        ver[v].push_back(u);
    }
}

void print_cut()
{
    printf("CUT:\n");
    for(int i = 1; i <= n ; i++)
    {
        if(cut[i])
            printf("%d ", i);
    }
    puts("");
}

void print_bridge()
{
    printf("BRIDGE:\n");
    for(int i = 1; i <= n; i ++)
    {
        for(int j = i + 1; j <= n; j ++)
        {
            if(bridge[i][j])
                printf("%d-%d, ", i, j);
        }
    }
    puts("");
}

```

```
int main()
{
	init();
	creat_graph();

	if(DEBUG)
	{
		for(int i = 1; i <= n; i ++)
		{
			for(vector<int>::size_type j = 0; j != ver[i].size(); j ++)
				printf("%d ", ver[i][j]);
			printf("\n");
		}
	}
	root = 1;
	//tarjan_cut(root, -1);
	print_cut();
	tarjan_bridge(root, -1);
	print_bridge();
	return 0;
}

/**
 *	当探寻当前节点的下一个节点的时候，先判断是否已 vis 过，如果没有则继续 tarjan
 *	如果 vis 过了。!!：还需判断是否不是其父节点!!  再更新 low[u] = min(low[u], dfn[v])
 */
```

3.4 BFS 判断二分图

```
#include <queue>
#include <cstring>
#include <iostream>
using namespace std;

const int N = 510;
int col[N],g[N][N];

/** 0 为白色，1 为黑色 */
bool bfs(int s, int n){
	queue<int> p;
	p.push(s);
	col[s] = 1;
	while(!p.empty()){
		int r = p.front();
		p.pop();
		for(int i = 1;i <= n;i++){
			if(g[r][i]&&col[i] == -1){
					p.push(i);
					col[i] = 1 - col[r];   /** 染成不同的颜色 */
			}
			if(g[r][i]&&col[r] == col[i])   /** 颜色有相同，则不是二分图 */
					return false;
		}
	}
	return true;
}

```

```
int main(){
	int n, m, a, b, i;
	memset(col, -1, sizeof(col));
	cin >> n >> m;
	for(i = 0;i < m;i++){
		cin >> a >> b;
		g[a][b] = g[b][a] = 1;
	}
	bool flag = false;
	for(i = 1;i <= n;i++)
		if(col[i] == -1&&!bfs(i, n)){//遍历各个连通分支
			flag = true;
			break;
		}
	if(flag)
		cout << "NO" <<endl;
	else
		cout << "YES" <<endl;
	return 0;
}
```

4 其他

4.1 大数类

```
#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <cmath>
#include <string>
#include <queue>
#include <map>
#include <vector>
#include <algorithm>
#include <cstdlib>
#include <ctime>

#define DEBUG 0
#define LSON(x) (x) << 1
#define RSON(x) (x) << 1 | 1
#define INF 0x1fffffff

#define MAXN 9999
#define MAXSIZE 10
#define DLEN 4

typedef long long LL;
using namespace std;


class BigNum
{
private:
	int a[500];	//可以控制大数的位数
	int len;		//大数长度
public:
        BigNum(){ len = 1;memset(a,0,sizeof(a)); }      //构造函数
        BigNum(const long long);              //将一个 int 类型的变量转化为大数
        BigNum(const char*);
  //将一个字符串类型的变量转化为大数
        BigNum(const BigNum &);
//拷贝构造函数
        BigNum &operator=(const BigNum &);
//重载赋值运算符，大数之间进行赋值运算

        friend istream& operator>>(istream&,    BigNum&);
//重载输入运算符
        friend ostream& operator<<(ostream&,    BigNum&);
//重载输出运算符

        BigNum operator+(const BigNum &) const;
//重载加法运算符，两个大数之间的相加运算
        BigNum operator-(const BigNum &) const;
//重载减法运算符，两个大数之间的相减运算
        BigNum operator*(const BigNum &) const;
//重载乘法运算符，两个大数之间的相乘运算
        BigNum operator/(const int      &) const;
//重载除法运算符，大数对一个整数进行相除运算

        BigNum operator^(const int    &) const;
//大数的 n 次方运算
        int        operator%(const int    &) const;
//大数对一个 int 类型的变量进行取模运算
        bool      operator>(const BigNum & T)const;
  //大数和另一个大数的大小比较
        bool      operator>(const int & t)const;
//大数和一个 int 类型的变量的大小比较

        void print();             //输出大数
};
BigNum::BigNum(const long long b)
//将一个 int 类型的变量转化为大数
{
        long long c,d = b;
        len = 0;
        memset(a,0,sizeof(a));
        while(d > MAXN)
        {
                c = d - (d / (MAXN + 1)) * (MAXN + 1);
                d = d / (MAXN + 1);
                a[len++] = c;
        }
        a[len++] = d;
}
BigNum::BigNum(const char*s)
//将一个字符串类型的变量转化为大数
{
        int t,k,index,l,i;
        memset(a,0,sizeof(a));
        l=strlen(s);
        len=l/DLEN;
        if(l%DLEN)
                len++;
        index=0;
        for(i=l-1;i>=0;i-=DLEN)
        {
            t=0;
            k=i-DLEN+1;
            if(k<0)
                k=0;
            for(int j=k;j<=i;j++)
                t=t*10+s[j]-'0';
            a[index++]=t;
        }
}
BigNum::BigNum(const BigNum & T) : len(T.len)    //拷贝构造函数
{
        int i;
        memset(a,0,sizeof(a));
        for(i = 0 ; i < len ; i++)
            a[i] = T.a[i];
}
BigNum & BigNum::operator=(const BigNum & n)
  //重载赋值运算符，大数之间进行赋值运算
{
        int i;
        len = n.len;
        memset(a,0,sizeof(a));
        for(i = 0 ; i < len ; i++)
            a[i] = n.a[i];
        return *this;
}
istream& operator>>(istream & in,    BigNum & b)
  //重载输入运算符
{
        char ch[MAXSIZE*4];
        int i = -1;
        in>>ch;
        int l=strlen(ch);
        int count=0,sum=0;
        for(i=l-1;i>=0;)
        {
            sum = 0;
            int t=1;
            for(int j=0;j<4&&i>=0;j++,i--,t*=10)
            {
                sum+=(ch[i]-'0')*t;
            }
            b.a[count]=sum;
            count++;
        }
        b.len =count++;
        return in;

}
ostream& operator<<(ostream& out,    BigNum& b)
//重载输出运算符
{
        int i;
        cout << b.a[b.len - 1];
        for(i = b.len - 2 ; i >= 0 ; i--)
        {
            cout.width(DLEN);
            cout.fill('0');
            cout << b.a[i];
        }
```

```
    return out;
}

BigNum BigNum::operator+(const BigNum & T) const
//两个大数之间的相加运算
{
    BigNum t(*this);
    int i,big;          //位数
    big = T.len > len ? T.len : len;
    for(i = 0 ; i < big ; i++)
    {
        t.a[i] +=T.a[i];
        if(t.a[i] > MAXN)
        {
            t.a[i + 1]++;
            t.a[i] -=MAXN+1;
        }
    }
    if(t.a[big] != 0)
        t.len = big + 1;
    else
        t.len = big;
    return t;
}
BigNum BigNum::operator-(const BigNum & T) const
//两个大数之间的相减运算
{
    int i,j,big;
    bool flag;
    BigNum t1,t2;
    if(*this>T)
    {
        t1=*this;
        t2=T;
        flag=0;
    }
    else
    {
        t1=T;
        t2=*this;
        flag=1;
    }
    big=t1.len;
    for(i = 0 ; i < big ; i++)
    {
        if(t1.a[i] < t2.a[i])
        {
            j = i + 1;
            while(t1.a[j] == 0)
                j++;
            t1.a[j--]--;
            while(j > i)
                t1.a[j--] += MAXN;
            t1.a[i] += MAXN + 1 - t2.a[i];
        }
        else
            t1.a[i] -= t2.a[i];
    }
    t1.len = big;
    while(t1.a[len - 1] == 0 && t1.len > 1)
    {
        t1.len--;
        big--;
    }
    if(flag)
        t1.a[big-1]=0-t1.a[big-1];
    return t1;
}

BigNum BigNum::operator*(const BigNum & T) const
//两个大数之间的相乘运算
{
    BigNum ret;
    int i,j,up;
    int temp,temp1;
    for(i = 0 ; i < len ; i++)
    {
        up = 0;
        for(j = 0 ; j < T.len ; j++)
        {
            temp = a[i] * T.a[j] + ret.a[i + j] + up;
            if(temp > MAXN)
            {
                temp1 = temp - temp / (MAXN + 1) * (MAXN + 1);
                up = temp / (MAXN + 1);
                ret.a[i + j] = temp1;
            }
            else
            {
                up = 0;
                ret.a[i + j] = temp;
            }
        }
        if(up != 0)
            ret.a[i + j] = up;
    }
    ret.len = i + j;
    while(ret.a[ret.len - 1] == 0 && ret.len > 1)
        ret.len--;
    return ret;
}
BigNum BigNum::operator/(const int & b) const
//大数对一个整数进行相除运算
{
    BigNum ret;
    int i,down = 0;
    for(i = len - 1 ; i >= 0 ; i--)
    {
        ret.a[i] = (a[i] + down * (MAXN + 1)) / b;
        down = a[i] + down * (MAXN + 1) - ret.a[i] * b;
    }
    ret.len = len;
    while(ret.a[ret.len - 1] == 0 && ret.len > 1)
        ret.len--;
    return ret;
}
int BigNum::operator %(const int & b) const
 //大数对一个 int 类型的变量进行取模运算
{
    int i,d=0;
    for (i = len-1; i>=0; i--)
    {
        d = ((d * (MAXN+1))% b + a[i])% b;
    }
    return d;
}
BigNum BigNum::operator^(const int & n) const
 //大数的 n 次方运算
{
    BigNum t,ret(1);
    int i;
    if(n<0)
        exit(-1);
    if(n==0)
        return 1;
    if(n==1)
        return *this;
    int m=n;
    while(m>1)
    {
        t=*this;
        for( i=1;i<<1<=m;i<<=1)
        {
            t=t*t;
        }
        m-=i;
        ret=ret*t;
        if(m==1)
            ret=ret*(*this);
    }
    return ret;
}
bool BigNum::operator>(const BigNum & T) const
//大数和另一个大数的大小比较
{
    int ln;
    if(len > T.len)
        return true;
    else if(len == T.len)
    {
        ln = len - 1;
        while(a[ln] == T.a[ln] && ln >= 0)
            ln--;
        if(ln >= 0 && a[ln] > T.a[ln])
            return true;
        else
            return false;
    }
    else
        return false;
}
bool BigNum::operator >(const int & t) const
//大数和一个 int 类型的变量的大小比较
{
    BigNum b(t);
    return *this>b;
}

void BigNum::print()      //输出大数
{
    int i;
    printf("%d", a[len-1]);
    //cout << a[len - 1];
    for(i = len - 2 ; i >= 0 ; i--)
    {
        printf("%00004d", a[i]);
//      cout.width(DLEN);
//      cout.fill('0');
//      cout << a[i];
    }
    //cout << endl;
    printf("\n");
}

int main(void)
{
//    freopen("out.txt", "r", stdin);
//    freopen("out1.txt", "w", stdout);
    int t; scanf("%d", &t);
    for(int c = 1; c <= t; c ++)
    {
        long long n, lim = 1000000000;
        scanf("%I64d", &n);
        if(n == 0)
        {
            printf("Case #%d: 1\n", c);
            continue;
        }
        if(n < lim)
        {
            printf("Case #%d: %I64d\n", c, 8 * n * n - 7*n +1);
            continue;
        }
        BigNum tp(n);
        BigNum ans, six(8), mins(n*7-1);
        ans = tp * tp;
        ans = ans * six;
        ans = ans - mins;
        printf("Case #%d: ", c);
        ans.print();
    }
    return 0;
}
```

4.2 输入挂

```

template <class T>
inline bool scan_d(T &ret) {
    char c; int sgn;
    if(c=getchar(),c==EOF) return 0; //EOF
    while(c!='-'&&(c<'0'||c>'9')) c=getchar();
    sgn=(c=='-')?-1:1;
    ret=(c=='-')?0:(c-'0');
    while(c=getchar(),c>='0'&&c<='9') ret=ret*10+(c-'0');
    ret*=sgn;
    return 1;
}

```