ACM/ICPC 比赛资料

数据结构和算法

目录

知识来源：40%来自于书本，40%来自于做题，20%来自于网上其他资料

头文件(精简)

```
#include <cstdio>
#include <cstdlib>
#include <cmath>
#include <cstring>
#include <iostream>
#include <algorithm>
using namespace std;

typedef long long LL;
```

头文件(附加)

```
#include <algorithm>
#include <cassert>
#include <cstdio>
#include <cstdlib>
#include <cmath>
#include <cstring>
#include <cctype>
#include <climits>
#include <complex>
#include <deque>
#include <fstream>
#include <iostream>
#include <list>
#include <map>
#include <numeric>
#include <vector>
#include <sstream>
#include <string>
#include <set>
#include <stack>
#include <queue>
#include <utility>
#include <typeinfo>
#include <ctime>
using namespace std;
```

```
#define pb push_back
#define mp make_pair

#define TLE while(1)
#define RE throw(1)
#define WA exit(0)

#define gcd __gcd
#define testbit(i, j) (( (i) >> (j) ) & 1)
#define flipbit(i, j) ( (i) ^= (1LL << (j)) )
#define lowbit(x) ( (x) & -(x) )
#define ctz __builitin_ctzll
#define pairity __builtin_parityll

#define FOR(i, a, b) for (int i = (a); i <= (b); ++i)
#define rep(i, n) for (int i = 0; i < (n); ++i)
#define clr(a, v, n) memet(a, v, sizeof(a[0]) * ((n)+1) )

template <class T> inline bool checkmin(T &a, const T& b) { if (a > b) {a = b; return 1; } return 0; }
template <class T> inline bool checkmax(T &a, const T& b) { if (a < b) {a = b; return 1; } return 0; }

const double pi = acos(-1);
typedef complex<double> cpl;

const double eps = 1e-8;
int sign(double x) { return x < -eps ? -1 : x > eps; }

double ASIN(double x) { if ( fabs(x) > 1 ) x = sign(x); return asin(x); }
double ACOS(double x) { if ( fabs(x) > 1 ) x = sign(x); return acos(x); }
double FLOOR(double x) { return floor(x + eps); }
double CEIL(double x) { return ceil(x - eps); }
double ROUND(double x) { return x >= 0 ? FLOOR(x + 0.5) : CEIL(x – 0.5);    }
double SQRT(double x) { return x >= 0 ? sqrt(x) : 0; }
```

比赛经验

对于 codeblocks，可以将 Settings – Environment 中的 Terminal to launch console programs 改为 gnome-terminal -t $TITLE –x

1. **积累模块。**

2. **最后取得的成绩好不好，关键在于心理素质好不好。**

3. **想着比赛的时候可以打表。**

4. **初始化一定不要忘记。**

5. **提交时记得把所有的调试信息都关掉。**

6. **想着可以用二分法把问题转化为判定性问题。**

7. **对于几何问题，没想法就先动手画画图，别上来就用解析法。**

8. **数组一定要开的足够大,能用 LL 就别用 int. (这样可以规避很多错误)**

9. **对于 sqrt(x)的时候，为了避免 x 为-0 导致出错，一定要 sqrt(abs(x))而不是 sqrt(x + eps),如 hdu1007 用+eps 的方法就会 WA**

10. 每道题提交之前，一定要经过足够的测试。注意多测试不对称数据。**公式一定要化简后再带入**。

11. **一定要仔细读所有题目，不要只是扫一眼，要精读。**

12. 无论是否有人通过，所有题必须全读过，最好每道题都有两人以上读过，尽量杜绝讲题现象。要完全弄清题意，正确的判断出题目的难易，不要想当然.

13. 虽然讨论有助于出题，但是以往每赛区第一名基本都是各自为战，但是互相了解，觉得一道题适合其他人做就转手.

14. 保持头脑灵活，在正常方法不行时想想歪门邪道，比如换种不常见的特殊的数据结构，加预处理，限时搜索等。效率是第一位的，如果觉得 DP 麻烦就用记忆化搜索，总之考虑清楚后就要在最短时间出题。**(递归+记忆化虽然好写，但是若递归次数太多，还是会超时的，还是尽量递推吧)**

15. 竞赛中更需要比平时稳定，程序出来后要检查重点地方，尽量 1Y。对于 WA 的题，不要改一处就交，很可能还有错的地方，要稳，要懂得在压力下也要仔细。对 WA 的题测试时要完整，必须每个点都测到，但不一定特别复杂。要考虑到测试的各种边界情况，**比如矩阵可能为 1*1 或 1*n 或 m*1**。

16. 除非做出的人很多，否则最后考虑复杂几何题，精度造成的问题太多了。对 double 型操作要小心判断大小、绝对值等情况。为了确保精度，可以 floor（x+eps），**输出的时候要小心-0**.00000，比如 a=-0.0000001，printf(“%.5f”,a);

17. 纸上写程序要尽量完整，每道题上机时间（包括输入、测试和调试）不要超过一小时。程序出错如果一时无法排除就应该打印出来阅读而把机器让出来。

18. 尽可能想到题目可以用到的数学的东西。

19. 实在迫不得已才可换人做题。

20. 一般需要一些非主流方法压缩时间的，都是算法有问题。

21. 朱泽园:由于一个人的问题导致卡住，应该一个人解决，不应该让全队都陷入混乱中。

22. 注意 Floyd 算法一定要初始化 dist[i][i] = 0

23. Treedp 或者扔给 mm，或者和 tt 讨论，千万别自己一个人瞎想～～～

组队赛说明

1. 要有做题比较多的队员，对于各种题型都有所涉及，做题稳，一般对前两道简单题能够保证快速，并且 99%以上一次 AC。

2. 要有人专门应付数学与几何题，但复杂的几何题要放在最后做，对一些常用的函数要有模版准备。

3. 要有人能够对付麻烦的题，并保证一定的通过率，大多数的比赛都至少有一道这样的题，如 POJ 1913，TOJ 1092。

4. 要有人对 DP 非常之熟，单次、双次、相对等情况都不在话下。对经典 DP 手到擒来。

5. 要有人对稀奇古怪的算法都做过程序，涉猎广，对于数论、图论中的一些特殊结论都知道。如 TOJ 1584，ZOJ 1015，UVA 10733。

6. 要有人对复杂的通用算法做过程序，如网络流中的最小费用最大流等等一系列的流，求割点/割边，启发式搜索/搏弈等。

7. 模版要自己写，并且另两个人都认真读过，用以往题目进行多次的测试。模版要全，但要控制篇幅，因为很多赛区已开始限制页数。

8. 每次练习赛都要当作正式比赛来做，要确保所有的题都看过，赛后要把没做出来的题尽量补上。

9. 最好的情况就是对于各种题目三个队员都能做，但是又各有侧重。要保证出来一道题能够可做。

关于时空效率

65535K = 6.7E7，一般只能开 1000^2 矩阵，开不了 10000^2 矩阵

使用 stl 要在效率上做出一些牺牲，对于输入规模很大的题目，有时必须放弃 stl，这意味着我们不能存在“有了 stl 就可以不去管基本算法的实现”的想法；另外，熟练和恰当地使用 stl 必须经过一定时间的积累，准确地了解各种操作的时间复杂度，切忌对 stl 中不熟悉的部分滥用，因为这其中蕴涵着许多初学者不易发现的陷阱。stl 一般比自己写的要慢 3－5 倍。

CPU 关于 int 的除法的运算没有想象的那么慢，大概是加法(乘法)的不到二十(三十)倍

主定理(Master Theorem)

$$T(n) = aT(n/b) + f(n)$$

记$p = \log_b a$则$T(n)$的解分三种情况：

$$T(n) = \begin{cases} \Theta(n^p) & f(n) = O(n^{p-\varepsilon}) \\ \Theta(n^p \log^{k+1} n) & f(n) = \Theta(n^p \log^k n) \\ \Theta(f(n)) & f(n) = \Omega(n^{p+\varepsilon}) \end{cases}$$

非主流优化

IO 优化

```
char buf[30000 * 10 * 3], *ptr;

#define getuint(x) {\
    while (!isdigit(*ptr)) ptr++;\
    x = 0;\
    while (isdigit(*ptr)) x = x * 10 + *ptr++ - '0';\
}
```

然后在 main 函数中调用 **fread(ptr = buf, 1, sizeof buf, stdin)**即可
也可以考虑调用 **setvbuf(stdin, buf, _IOFBF, sizeof buf)**函数，直接设置一个很大的缓冲区。
（输出内容比较多时，也可以通过调用 **setvbuf(stdout, buf, _IOFBF, sizeof buf)**函数来加速）
此外在只使用 cin/cout 时，为了加速，可以设置成 **ios::sync_with_stdio(false)**
（Notice：可能 cout 用 sync_with_stdio 优化效果不明显，e.g. spoj lcmsum）

内嵌汇编，改 esp

注：在 hdu G++测试通过，C++编译错误

```
int main(){
    static const int stk_sz = 10000000;
    static int stack[stk_sz], esp_bak;
    __asm__ __volatile__ (
        "movl %%esp, %0\n\t" "movl %1, %%esp\n\t" : "=g"(esp_bak) : "g"(stack + stk_sz - 1)
    );
    solve();
    __asm__ __volatile__ (
        "movl %0, %%esp\n\t" : : "g"(esp_bak)
    ); //也可以不恢复 esp，直接 exit(0);
}
```

inline 优化

函数声明的时候添加：

```
inline void foo() __attribute__((always_inline, optimize(3)));
```

编译层面的优化

在程序中加入**#pragma GCC optimize ("O2")**来优化(要求 GCC 版本 >= 4.4)
在程序中加入#pragma GCC target("sse4.1, arch = core2")优化(要求 GCC 版本 >= 4.4)

编译原理

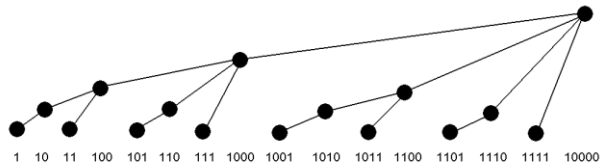
很多编译器会把 switch 语句实现为一个跳转表，而不是一系列的 if/else 条件。
特别注意：MinGW 的编译器有点特殊，输出 long long 是用"%I64d"的!
对于堆来讲，生长方向是向上的，也就是向着内存地址增加的方向；对于栈来讲，它的生长方向是向下的，是向着内存地址减小的方向增长
gcc 的动态数组(直接定义长度为变量的数组)最好不用，因为 gcc 会因此生成大量代码(七八十行汇编)
在定义多维数组时，我们不应该让任意一维的尺寸是 2 的方幂
一般情况下-O2 才会优化掉尾递归

acm 题型分布

搜索	动态规划	贪心	构造	图论	计算几何	纯数学问题	数据结构	其他
10%	15%	5%	5%	10%	5%	20%	5%	25%

数据结构

树状数组



C[i] = sum{ a[i-lowbit(i)+1..i] }
C[i] = lowbit(i) i=1..n 代表每个位置都赋值为 1
C[i][j] = sum{ a[i-lowbit(i)+1..i][j-lowbit(j)+1..j] }
C[i][j] = lowbit(i) * lowbit(j) i=1..m, j = 1..n 代表每个位置都赋值为 1
三维及以上类似

在树状数组上做一个 logn 的二分

struct Bit {

```
    static const int maxn = 100000 + 5;
    int c[maxn];
    void init(){
        memset(c, 0, sizeof c);
    }
    void inc(int i, int d) {
        for (; i < maxn; i += lowbit(i) ) c[i] += d;
    }
    LL query(int i) const {
        LL s = 0;
        for (; i; i -= lowbit(i) ) s += c[i];
        return s;
    }
    int lower_bound(int x) const {
        int ind = 0; LL s = 0;
        for (int i = 19; i >= 0; --i) { // 1 << (19+1) > maxn
            if ( ind + (1 << i) < maxn && s + c[ind + (1 << i)] < x ) {
                ind += (1 << i);
                s += c[ind];
            }
        }
        return ind + 1;
    }
    int upper_bound(int x) const {
        int ind = 0; LL s = 0;
        for (int i = 19; i >= 0; --i) { // 1 << (19+1) > maxn
            if ( ind + (1 << i) < maxn && s + c[ind + (1 << i)] <= x ) {
                ind += (1 << i);
                s += c[ind];
            }
        }
        return ind + 1;
    }
};
```

树状数组的应用

支持两类操作：

1.把一个区间内所有元素都加上一个值

2.查询某个区间内所有元素的和

方法：

update(int p, int d) //将区间 1..p 加上 d

操作：B[p] += d * p, C[1..p-1] += d;

query(int p) //查询区间 1..p 的和

操作：return sum{B[1..p]} + C[p] * p;

代码：

```
const int maxn = 100000 + 10;
LL B[maxn], C[maxn];

void update(LL A[], int i, LL d){ for (; i < maxn; i += lowbit(i)) A[i] += d;}
LL query(LL A[], int i){ LL ret = 0; for (; i >= 1; i -= lowbit(i)) ret += A[i]; return ret;}

void update(int p, LL d){
    p += 5; update(B, p, d * p); update(C, p-1, d * (p-1)); update(C, p-2, -d * (p-2));
}

LL query(int p){
    p += 5; return query(B, p) + (query(C, p) - query(C, p-1)) * p;
}

void update(int a, int b, int d){ update(b, d); update(a-1, d); }
LL query(int a, int b){ return query(b) - query(a-1);}
```

并查集(食物链)

对于并查集，虽然据说路径压缩直接把父节点指向祖父时间复杂度不变，但是没效果。

上次使用 2012.7

```
int find(int x){
    int r = x, totdist = 0;
    for (;r != fa[r]; r = fa[r])
        totdist += dist[r];
    for (int y, dx; x != r; totdist -= dx, x = y){
        y = fa[x]; dx = dist[x];
        fa[x] = r; dist[x] = totdist;
    }
    return r;
}
```

二叉堆

stl 提供以下函数

```
make_heap(a, a + n);
push_heap(a, a + n);
pop_heap(a, a + n);
__is_heap(a, a + n);
```

```
sort_heap(a, a + n);
```

有序表合并(有待重写)

简述：把 k 个有序表合并成一个有序表，元素共有 n 个。
方法：把每个表的最小元素放入二叉堆中，每次删除最小值并放入结果表中，然后加入此序列的下一个元素。时间复杂度 O(nlogk)
代码：

```
const int maxk = 10000;
const int maxn = 10000;

int src[maxn + 1], res[maxn + 1], end[maxk + 1];
int iter[maxk + 2] /*注意这里至少+2*/, iterend[maxk + 2];

namespace Heap{
    //Heap 下标从 1 开始，建立的是小顶堆
    void up(int n, int p){
        int q = p >> 1, a = iter[p], b = iterend[p];
        for (; q > 0 && src[a] < src[iter[q]]; p = q, q = p >> 1)
            iter[p] = iter[q], iterend[p] = iterend[q];
        iter[p] = a;
    }

    void down(int n, int p){
        int q = p << 1, a = iter[p], b = iterend[p];
        for (; q <= n; p = q, q = p << 1){
            if (q < n && src[iter[q+1]] < src[iter[q]]) ++q;
            if (src[iter[q]] >= src[a]) break;
            iter[p] = iter[q]; iterend[p] = iterend[q];
        }
        iter[p] = a; iterend[p] = b;
    }

    void pop(int &n){
        swap(iter[1], iter[n]); swap(iterend[1], iterend[n]); down(--n, 1);
    }

    void push(int &n, int a, int b){
        iter[++n] = a; iterend[n] = b; up(n, n);
    }

    void build(int n){
        for (int i = n >> 1; i >= 1; i--) down(n, i);
    }
```

```
};

int main(){
    int k, totn;
    while(scanf("%d", &k) != EOF){
        end[0] = 0;
        for (int i = 1; i <= k; i++){
            int n; scanf("%d", &n);
            iter[i] = iterend[i-1]; iterend[i] = iterend[i-1] + n;
            for (int j = iterend[i-1]; j < iterend[i]; j++)
                scanf("%d", &src[j]);
        }
        totn = iterend[k];
        Heap::build(k);
        for (int i = 0; i < totn; i++){
            res[i] = src[iter[1]];
            if (iter[1]+1 < iterend[1]) Heap::push(k, iter[1]+1, iterend[1]);
            Heap::pop(k);
        }
        for (int i = 0; i < totn; i++)
            printf("%d ", res[i]);
        putchar('\n');
    }
}
```

序列和的前 n 小元素(有待重写)

描述：给出两个长度为 n 的有序表 A 和 B，在 A 和 B 中各取一个元素，得到 n^2 个和，求这些和中最小的 n 个

解法：可以把这 n^2 个数看成 n 个长度为 n 的有序表：

$A_1 + B_1 \le A_1 + B_2 \le A_1 + B_3 \le \ldots\ldots \le A_1 + B_n$，

$A_2 + B_1 \le A_2 + B_2 \le A_2 + B_3 \le \ldots\ldots \le A_2 + B_n$，

.....

$A_n + B_1 \le A_n + B_2 \le A_n + B_3 \le \ldots\ldots \le A_n + B_n$，

套用刚才的算法，但只取 n 次最小元素，总的时间共 O(nlogn)

代码：

```
const int maxn = 10000;
int a[maxn + 1], b[maxn + 1], res[maxn + 1];

struct Sum{
    int ia, ib;
    friend bool operator < (const Sum& s1, const Sum& s2){
        return a[s1.ia] + b[s1.ib] < a[s2.ia] + b[s2.ib];
    }
```

```
    friend bool operator >= (const Sum& s1, const Sum& s2){
        return a[s1.ia] + b[s1.ib] >= a[s2.ia] + b[s2.ib];
    }
} sum[maxn + 2];

namespace Heap{ //Heap 下标从 1 开始,建立的是小顶堆
    typedef struct Sum T;
    void up(T H[], int n, int p){
        int q = p >> 1; T a = H[p];
        for (; q > 0 && a < H[q]; p = q, q = p >> 1)
            H[p] = H[q];
        H[p] = a;
    }

    void down(T H[], int n, int p){
        int q = p << 1;
        T a = H[p];
        for (; q <= n; p = q, q = p << 1){
            if (q < n && H[q+1] < H[q]) ++q;
            if (H[q] >= a) break;
            H[p] = H[q];
        }
        H[p] = a;
    }

    void pop(T H[], int &n){
        swap(H[1], H[n--]); down(H, n, 1);
    }

    void push(T H[], int &n, T a){
        H[++n] = a; up(H, n, n);
    }

    void build(T H[], int n){
        for (int i = n >> 1; i >= 1; i--)
            down(H, n, i);
    }
};

int main(){
    int n, nb;
    while(scanf("%d", &n) != EOF){
        for (int i = 1; i <= n; i++)
            scanf("%d", &a[i]);
```

```
            for (int i = 1; i <= n; i++)
                scanf("%d", &b[i]), sum[i].ia = 1, sum[i].ib = i;
            Heap::build(sum, n);
            nb = n;
            for (int i = 1; i <= nb; i++){
                res[i] = a[sum[1].ia] + b[sum[1].ib];
                sum[0].ia = sum[1].ia + 1; sum[0].ib = sum[1].ib;
                Heap::push(sum, n, sum[0]); Heap::pop(sum, n);
            }
            for (int i = 1; i <= nb; i++)
                printf("%d ", res[i]);
            putchar('\n');
        }
}
```

映射二叉堆

//已经通过 hdu1024，注意修改某个元素后要马上进行 up(down)操作，而不要几次修改后再分别对修改的位置进行 up(down)操作，代码参见 88 页 **struct Heap** 类

Huffman 编码生成

可以先按照频率把所有字符排成表 P，然后另外设置队列 Q。每次合并两个节点后放到队列 Q 中，由于后合并的频率一定比先合并的频率和大，因此 Q 内的元素有序。类似有序表的合并过程，每次只需检查 P 和 Q 的首元素即可找到频率最小的元素，时间复杂度为 O(n)，加上按照频率排序的过程，共 O(nlogn)。

也可以把表 P 和表 Q 合成一个，时间复杂度仍为 O(nlogn)，但时间效率略低。

左势堆(左偏树)

```
namespace LeftistTree {
    const int maxn = 10000;

    typedef struct LeftistHeapNode{
        int key, dist;
        LeftistHeapNode *left, *right, *father;
    } *LeftistHeap;

    LeftistHeap null = new LeftistHeapNode;

    void init(){
        null->dist = -1;
```

```
}

LeftistHeap newnode(int key, int dist = 0){
    LeftistHeap p = new LeftistHeapNode;
    p->left = p->right = p->father = null;
    p->key = key; p->dist = dist;
    return p;
}

LeftistHeap merge(LeftistHeap a, LeftistHeap b){ //时间复杂度 O(log(N))
    if (a == null) return b; else if (b == null) return a;
    if (a->key < b->key) swap(a, b); // >:小顶堆; <:大顶堆
    a->right = merge(a->right, b);
    a->right->father = a;

    if (a->left->dist < a->right->dist) swap(a->left, a->right);
    a->dist = a->right->dist + 1;
    return a;
}

void deletemin(LeftistHeap &p){ //时间复杂度 O(logN)
    p = merge(p->left, p->right);
}

void deletenode(LeftistHeap p){
    if (p == null) return;
    LeftistHeap f = p->father, q = merge(p->left, p->right);
    q->father = f;
    if (f->left == p) f->left = q; else f->right = q;
    for (;f != null; f = f->father){
        if ( f->left->dist < f->right->dist ) swap(f->left, f->right);
        if ( f->right->dist + 1 == f->dist ) return;
        f->dist = f->right->dist + 1;
    }
}

LeftistHeap Queue[maxn * 2], *Qbegin, *Qend;

LeftistHeap build(LeftistHeap Queue[], LeftistHeap *Qbegin, LeftistHeap *Qend){
    //时间复杂度 O(N)
    if (Qbegin == Qend) return 0; //0 elements
    for (;Qbegin + 1 != Qend; Qbegin += 2)
        *Qend++ = merge(*Qbegin, *(Qbegin+1));
    return *Qbegin;
```

```
    }
} //end namespace

/**表示一个集合，可以快速实现以下功能
 * 1. 取得这个集合的中位数(如果有两个中位数，取较小的那个)
 * 2. 合并两个集合
 */
struct MedianSet {
    int n;
    LeftistTree::LeftistHeap s; //大根堆

    MedianSet(int key){
        s = LeftistTree::newnode(key); n = 1;
    }

    int getMedian(){
        return s->key;
    }

    MedianSet &merge(MedianSet b){
        this->s = LeftistTree::merge(this->s, b.s);
        this->n += b.n;
        if (this->n % 2 && b.n % 2)    LeftistTree::deletemin(this->s);
        return *this;
    }
};

int main(){ //zju 3512
    LeftistTree::init();
    int n;
    while (cin >> n, n){
        static int a[50000 + 5];
        for (int i = 0; i < n; i++) scanf("%d", &a[i]);
        static char s[(50000 + 5) * sizeof(MedianSet)];
        MedianSet *stop, *sbase; sbase = stop = (MedianSet *)s;
        for (int i = 0; i < n; i++){
            *stop++ = MedianSet(a[i]);
            while (stop - sbase >= 2 && stop[-1].getMedian() < stop[-2].getMedian() ){
                stop[-2].merge(stop[-1]); --stop;
            }
        }
        long long res = 0;
        for (int i = 0, j = 0; i < n; i++){
            res += abs( sbase[0].getMedian() - a[i]);
```

```
            if (++j == sbase[0].n) j = 0, ++sbase;
        }
        cout << res << endl;
    }
}
```

Treap（准静态内存分配、有附加功能）

```
const int maxn = 100000;
typedef long long Tvalue;

struct TreapNode{ //维护的是一个小根堆
    Tvalue key;
    int size; //该节点和它的所有后代的个数
    int heapkey;
    TreapNode *left, *right;
};

class Treap{
public:
    typedef TreapNode *iterator;

    void init(){ end = root; root = NULL; }
    void insert(Tvalue newkey){ insert(root, newkey); }
    int countless(Tvalue key){ return countless(root, key); }
    int countgreater(Tvalue key){ return countgreater(root, key); }
    iterator find(Tvalue key){
        iterator p = root;
        for (;;){
            if (key == p->key) return p;
            if (key < p->key)
                if (p->left) p = p->left; else return NULL;
            else
                if (p->right) p = p->right; else return NULL;
        }
    }
    iterator nth_element(int n){
        if (n > root->size) return 0;
        iterator p = root;
        for (;;){
            if (p->left && n <= p->left->size) p = p->left;
            else {
                if (p->left) n -= p->left->size;
                n -= selfsize(p);
```

```
                    if (n <= 0) return p; else p = p->right;
                }
            }
        }

private:
    TreapNode it[maxn];
    iterator root, end;

    iterator newnode(Tvalue newkey){
        end->key = newkey; end->heapkey = rand();
        end->size = 1; end->left = end->right = 0;
        return end++;
    }

    void recalcsize(iterator p){
        if (p->left)    p->size += p->left->size;
        if (p->right) p->size += p->right->size;
    }

    int selfsize(iterator p) const{
        int count = p->size;
        if (p->left)    count -= p->left->size;
        if (p->right) count -= p->right->size;
        return count;
    }

    void rotateWithLeftChild(iterator& p){
        iterator q = p->left;
        p->size = selfsize(p); q->size = selfsize(q);
        p->left = q->right; q->right = p;
        recalcsize(p);//一定要先算 p 再算 q
        recalcsize(q);
        p = q;
    }

    void rotateWithRightChild(iterator& p){
        iterator q = p->right;
        p->size = selfsize(p); q->size = selfsize(q);
        p->right = q->left; q->left = p;
        recalcsize(p);
        recalcsize(q);
        p = q;
    }
```

```
    void insert(iterator p, Tvalue newkey){
        if (p == NULL) p = newnode(newkey);
        else{
            if (p->key == newkey) p->size++;
            else if(newkey < p->key){
                insert(p->left, newkey); p->size++;
                if (p->left->heapkey > p->heapkey) rotateWithLeftChild(p);
            }
            else{
                insert(p->right, newkey); p->size++;
                if (p->right->heapkey > p->heapkey) rotateWithRightChild(p);
            }
        }
    }

    int countless(iterator p, Tvalue key){
        if (p == NULL) return 0;
        int count = 0;
        if (p->key <= key){
            if (p->left) count += p->left->size;
            if (p->key < key){
                count += selfsize(p);
                count += countless(p->right, key);
            }
        }else
            count += countless(p->left, key);
        return count;
    }

    int countgreater(iterator p, Tvalue key){
        if (p == 0) return 0;
        int count = 0;
        if (p->key >= key){
            if (p->right) count += p->right->size;
            if (p->key > key){
                count += selfsize(p);
                count += countgreater(p->left, key);
            }
        }else
            count += countgreater(p->right, key);
        return count;
    }
}; /*end class Treap */
```

```
Treap tp;

int main(){ // hrboj 5086
	int n, a, b, casecnt;
	Tvalue s, total;
	scanf("%d", &casecnt);
	while (casecnt --){
		scanf("%d %d", &n, &a);
		s = total = 0;
		tp.init();
		tp.insert(s);

		for (int i=1; i<=n; i++){
			scanf("%d", &b);
			s += b;
			total += tp.countless(s-(long long)a*i);
			tp.insert(s-(long long)a*i);
		}
		cout << total << endl;
	}
}
```

Treap – 支持区间翻转 by HL

```
const int base = 259;
int power[120000];

struct node{
	node *l, *r, *f;
	int v[2];
	int rev, size;
	int self;
	int h;

	node(){
		l = r = f = 0;
		self = v[0] = v[1] = rev = 0;
		size = 1;
		h = (rand() << 15) + rand();
	}

	void push(){
		if(rev == 1){
```

```
            swap(l, r);
            swap(v[0], v[1]);
            rev = 0;

            if(l!=0) l->rev ^= 1;
            if(r!=0) r->rev ^= 1;
        }
    }

    void tidy(){
        push();
        if(l!=0) l->push();
        if(r!=0) r->push();

        int lsize = (l==0 ? 0 : l->size);
        int rsize = (r==0 ? 0 : r->size);
        size = lsize + rsize + 1;

        v[0] = (l==0 ? 0 : l->v[0])
            + power[lsize] * self
            + power[lsize+1] * (r==0 ? 0 : r->v[0]);

        v[1] = (r==0 ? 0 : r->v[1])
            + power[rsize] * self
            + power[rsize+1] * (l==0 ? 0 : l->v[1]);
    }
};

void tidy(node * root){
    while(root != 0){
        root->tidy();
        root = root->f;
    }
}

void split(node * root, int u, node *& l, node *& r, node* lf, node *rf){
    root->push();
    int lsize = root->l == 0 ? 0 : root->l->size;
    if(u < lsize){
        node * tmp = root->l;
        r = root; root->f = rf; root->l = 0;
        split(tmp, u, l, r->l, lf, root);
    }else if(u == lsize){
        l = root->l; if(root->l != 0) root->l->f = lf;
```

```
            r = root; root->f = rf; root->l = 0;
            tidy(l); tidy(r);
        } else if(u == lsize+1){
            r = root->r; if(root->r != 0) root->r->f = rf;
            l = root; root->f = lf; root->r = 0;
            tidy(l); tidy(r);
        }else{
            node * tmp = root->r;
            l = root; root->f = lf; root->r = 0;
            split(tmp, u-lsize-1, l->r, r, root, rf);
        }
}

node* merge(node *l, node *r){
        if(l == 0) return r;
        if(r == 0) return l;
        l->push(); r->push();
        if(l->h > r->h){
            l->r = merge(l->r, r);
            l->r->f = l;
            l->tidy();
            return l;
        } else {
            r->l = merge(l, r->l);
            r->l->f = r;
            r->tidy();
            return r;
        }
}

node * root;
char s[120000];
char command[20], m[5];
int n, left, right, pos, a, b, length;

void REVERSE(int left, int right){
        if(left > right) swap(left, right);
        node *r1 = 0, *r2 = 0, *r3 = 0, *r = 0;
        split(root, left - 1, r1, r, 0, 0); split(r, right - left + 1, r2, r3, 0, 0);
        r2->rev = true;
        r = merge(r2, r3); root = merge(r1, r);
}

void MODIFY(int pos, char m){
```

```
    node *r1 = 0, *r2 = 0, *r3 = 0, *r = 0;
    split(root, pos - 1, r1, r, 0, 0); split(r, 1, r2, r3, 0, 0);
    r2->rev = 0; r2->self = r2->v[0] = r2->v[1] = m;
    r = merge(r2, r3); root = merge(r1, r);
}

int query(int left, int right){
    if(left > right) swap(left, right);
    node *r1 = 0, *r2 = 0, *r3 = 0, *r = 0;
    split(root, left - 1, r1, r, 0, 0); split(r, right - left + 1, r2, r3, 0, 0);
    r2->push();
    int ans = r2->v[0];
    r = merge(r2, r3); root = merge(r1, r);
    return ans;
}

bool check(int a, int b, int l){
    int u = query(a, a+l-1);
    int v = query(b, b+l-1);
    return u == v;
}

int LCP(int a, int b){
    int lower = 0, upper = min(length - b + 1, length - a + 1);
    if(check(a, b, upper)) return upper;
    while(lower + 1 < upper){
        int mid = (lower + upper) / 2;
        if(check(a, b, mid)) lower = mid; else upper = mid;
    }
    return lower;
}

int cnt;
int T[240000];

void dfs(node * root){
    if(root==0) return;
    root->push();
    dfs(root->l);
    T[cnt++] = root->self; T[cnt++] = 128;
    dfs(root->r);
}

void kp(int str[], int n, int p[]) {
```

```
    int i;
    int mx = 0;
    int id;
    for(i=n; str[i]!=0; i++)
        str[i] = 0;
    for(i=1; i<n; i++) {
        if( mx > i )
            p[i] = min( p[2*id-i], p[id]+id-i );
        else
            p[i] = 1;
        for(; str[i+p[i]] == str[i-p[i]]; p[i]++)
            ;
        if( p[i] + i > mx )
            mx = p[i] + i, id = i;
    }
}

void getT(){
    cnt = 0; T[cnt++] = 129; T[cnt++] = 128;
    dfs(root);
    T[cnt++] = 130;
}

int p[240000];

int PAL(){
    memset(T, 0, sizeof(T));
    getT(); kp(T, cnt, p);
    int res = 0;
    for(int i=0; i<cnt; i++)
        if(res < p[i])
            res = p[i];
    return res-1;
}

void delete_tree(node * root){
    if(root->l != 0) delete_tree(root->l);
    if(root->r != 0) delete_tree(root->r);
    delete root;
}

int main(){
    power[0] = 1;
    for(int i=1; i<=110000; i++) power[i] = power[i-1] * base;
```

```
    while(scanf("%s", &s) != EOF){
        length = strlen(s);
        root = 0;
        for(int i=0; i<length; i++){
            node * newnode = new node();
            newnode->self = newnode->v[0] = newnode->v[1] = s[i];
            newnode->size = 1;
            root = merge(root, newnode);
        }
        scanf("%d", &n);
        for(int i=0; i<n; i++){
            scanf("%s", &command);
            switch(*command) {
            case 'R':
                scanf("%d%d", &left, &right);
                REVERSE(left, right);
                break;
            case 'M':
                scanf("%d%s", &pos, &m);
                MODIFY(pos, *m);
                break;
            case 'L':
                scanf("%d%d", &a, &b);
                printf("%d\n", LCP(a, b));
                break;
            case 'P':
                printf("%d\n", PAL());
                break;
            }
        }
        delete_tree(root);
    }
}
```

笛卡尔树

给出二叉树的 key 和 value，新建出二叉树/堆

```
struct Node {
    int /*TREE*/ key, /*HEAP*/value, ind;
    Node *father, *left, *right;
} node[maxn], *null;

struct lessKey {
```

```
    bool operator()(Node* x, Node* y) const{ return x->key < y->key;}
};

Node *build(Node node[], int n){
    static Node *pnode[maxn];
    Node *r, *last, *cur;
    for (int i = 0; i < n; i++) pnode[i] = node + i;
    sort(pnode, pnode + n, lessKey() );
    r = last = pnode[0];
    r->father = r->left = r->right = null;
    for (int i = 1; i < n; i++){
        cur = pnode[i];
        cur->father = cur->left = cur->right = null;
        if ( cur->value < r->value ){
            cur->left = r; r->father = cur;
            last = r = cur;
        } else {
            while (cur->value <= last->value) last = last->father;
            cur->left = last->right; last->right->father = cur;
            last->right = cur; cur->father = last;
            last = cur;
        }
    }
    return r;
}

int main(){ //poj2201
    null = new Node; null->ind = 0;
    for (int n; cin >> n; ){
        for (int i = 0; i < n; i++)
            scanf("%d%d", &node[i].key, &node[i].value),node[i].ind = i + 1;
        build(node, n);
        puts("YES");
        for (int i = 0; i < n; i++){
            Node *p = node + i;
            printf("%d %d %d\n", p->father->ind, p->left->ind, p->right->ind);
        }
    }
}
```

后缀数组(再看看吧)

```
//倍增算法略短，效率略差
//dc3 算法略长，效率略强
```

```
//已经通过 hdu3336
//sa[i]      排第 i 的是谁?
//rank[i]    i 排第几?

const int maxn = 200000 + 5;
const int maxm = UCHAR_MAX;

void bucket_sort(int *r, int *a, int *b, int n, int m = maxm){
    //对 a[]进行基数排序，结果存到 b[]中。其中 a[i]对应关键字 r[a[i]]
    static int bucket[maxn];
    static int val[maxn];
    if (r == 0){
        for (int i = 0; i < m; i++) bucket[i] = 0;
        for (int i = 0; i < n; i++) ++bucket[a[i]];
        for (int i = 1; i < m; i++) bucket[i] += bucket[i-1];
        for (int i = n-1; i >= 0; i--) b[--bucket[a[i]]] = i;
    }
    else {
        for (int i = 0; i < n; i++) val[i] = r[a[i]];
        for (int i = 0; i < m; i++) bucket[i] = 0;
        for (int i = 0; i < n; i++) ++bucket[val[i]];
        for (int i = 1; i < m; i++) bucket[i] += bucket[i-1];
        for (int i = n-1; i >= 0; i--) b[--bucket[val[i]]] = a[i];
    }
}

inline int equal(int *r, int a, int b, int len, int n){
    return r[a] == r[b] && r[a+len] == r[b+len];
}

void da(char *r, int *sa, int *rank, int n, int m = maxm){
    //已经通过 bnu4333
    //求字符串 r[]的后缀数组，结果放在 sa[0..n-1]中
    //待排序的字符放在 r 数组中，从 r[0]到 r[n-1], 长度为 n
    //要求：r 最大值严格小于 m。除 r[n-1]=0 外 r[i]>0
    //如果需要，可以将 r 的类型"直接"修改为 int
    static int rank2[maxn];
    int *x = rank, *y = rank2; //这里 x[]保存的值相当于 rank 值

    for (int i = 0; i < n; i++) x[i] = r[i];
    bucket_sort(0, x, sa, n, m);

    for (int j = 1, p = 1; p < n; j *= 2, m = p){     //这里变量 j 是当前字符串的长度
        //对第二关键字进行基数排序
```

```
        //y[]保存的是对第二关键字排序的 sa[]的值
        p = 0;
        for (int i = n-j; i < n; i++) y[p++] = i;
        for (int i = 0; i < n; i++)
            if (sa[i] >= j) y[p++] = sa[i] - j;

        //对第一关键字进行基数排序
        bucket_sort(x, y, sa, n, m);

        //计算 rank[]值(储存在 x[]中)
        swap(x, y);
        p = 1;
        x[sa[0]] = 0;
        for (int i = 1; i < n; i++)
            x[sa[i]] = equal(y, sa[i-1], sa[i], j, n) ? p-1 : p++;
    }
    if (x != rank) copy(x, x + n, rank);
}

/*
dc3 算法分 3 步
(1)先将后缀分成两部分，然后对第一部分的后缀排序
第一部分：后缀 k mod 3 != 0
第二部分：后缀 k mod 3 == 0
(2)利用(1)的结果，对第二部分的后缀排序
(3)将(1)(2)的结果合并，即完成对所有后缀的排序。
*/

int equal(int *r, int a, int b){
    return r[a] == r[b] && r[a+1] == r[b+1] && r[a+2] == r[b+2];
}

void dc3(int *r, int *sa, int n, int m = maxm){
    //时间复杂度 O(n)
    //r[]和 sa[]大小都要是 3n
    //r[]必须是 int[]类型，不能是 char[]void sort(int *r, int *a, int *b, int n, int m = maxm){
    //对 a[]进行基数排序，结果存到 b[]中。其中 a[i]对应关键字 r[a[i]]
    static int bucket[maxn], val[maxn];
    static int x1[maxn], x2[maxn]; //x1[], x2[]存的是 sa[]
    static int rank[maxn];
    int *rn = r + n;        //rn[]保存(1)中要递归处理的新字符串
    int *san = sa + n;      //san[]是新字符串的 sa[]
    int ta = 0;             //起始位置 mod 3=0 的后缀个数
    int tb = (n+1) / 3;     //起始位置 mod 3=1 的后缀个数
```

```
int tbc = 0;                //起始位置 mod 3=1 或 2 的后缀个数
int p;
//wv[]保存的值相当于 rank 值

//进行第(1)步
r[n] = r[n+1] = 0;
for (int i = 0; i < n; i++)
    if (i % 3 != 0) x1[tbc++] = i;
bucket_sort(r+2, x1, x2, tbc, m);
bucket_sort(r+1, x2, x1, tbc, m);
bucket_sort(r, x1, x2, tbc, m);
p = 1;
//F(x)计算出原字符串的 suffix 在新字符串中的起始位置
#define F(x) ((x) / 3 + ((x) % 3 == 1 ? 0 : tb))
rn[F(x2[0])] = 0;
for (int i = 1; i < tbc; i++)
    rn[F(x2[i])] = equal(r, x2[i-1],x2[i]) ? p-1 : p++;
#undef F
if (p < tbc) dc3(rn, san, tbc, p);
else for (int i = 0; i < tbc; i++) san[rn[i]] = i;

//进行第(2)步
for (int i = 0; i < tbc; i++)
    if (san[i] < tb) x2[ta++] = san[i] * 3;
if (n % 3 == 1) x2[ta++] = n-1;
//由于 r[n-1]<r[i], wb[ta-1]按照第一关键字排序一定会排到正确位置
bucket_sort(r, x2, x1, ta, m);
//G(x)是计算新字符串的 suffix 在原字符串的位置
#define G(x) ((x) < tb ? (x)*3+1 : ((x)-tb)*3+2)
for (int i = 0; i < tbc; i++)
    rank[x2[i] = G(san[i])] = i;
#undef G

//进行第(3)步，合并排序结果
#define less1(r,a,b) (r[a] < r[b] || r[a] == r[b] && rank[a+1] < rank[b+1])
#define less2(r,a,b) (r[a] < r[b] || r[a] == r[b] && less1(r,a+1,b+1))
#define less(k,r,a,b) ((k)==2 ? less2(r,a,b) : less1(r,a,b))
p = 0;
int i = 0, j = 0;
while (i < ta && j < tbc)
    sa[p++] = less(x2[j]%3,r,x1[i],x2[j]) ? x1[i++] : x2[j++];
while (i < ta) sa[p++] = x1[i++];
while (j < tbc) sa[p++] = x2[j++];
#undef less
```

```
	#undef less1
	#undef less2
}

void getRank(int *sa, int *rank, int n){
	for (int i = 0; i < n; i++)
		rank[sa[i]] = i;
}

int height[maxn];
//定义 height[i] = suffix(sa[i-1])和 suffix(sa[i])的最长公共前缀长度
void calheight(char *r, int *sa, int *rank, int n){
	//时间复杂度 O(n)
	//r[0..n-1], sa[1..n](sa[0]=n), rank[0..n-1](rank[n]=0)
	//-> height[rank[0..n-1]](height[rank[n]]=0)
	//要求 r[]和 sa[]是用上面方法求出的
	int k = 0, j;
	for (int i = 0; i < n; i++) { //from 0 to n-1
		if (k > 0) --k;
		j = sa[rank[i]-1];
		while (r[i+k] == r[j+k]) ++k;
		height[rank[i]] = k;
	}
//	for (int i = 0; i <= n; i++) cout << height[rank[i]] << " "; cout << endl;
}

char r[maxn];
int ir[maxn * 3];
int sa[maxn * 3], rank[maxn];

int main(){
	int t, n; scanf("%d", &t);
	while (t--){
		scanf("%d%s", &n, r);
		copy(r, r + n, ir);
		dc3(ir, sa, n+1); getRank(sa, rank, n+1);
		//da(r, sa, rank, n+1);
		calheight(r, sa, rank, n);
	}
}

//最长公共前缀（询问字符串两个后缀的最长公共前缀长度）
//O(nlogn + m)
```

```
//int RMQ[maxn];
int *RMQ = height - 1; //height[0..n-1] -> RMQ[1..n]
int mm[maxn];
int best[20][maxn];

#define POW2(n) (1<<(n))

void initRMQ(int n){
	int a, b;
	mm[0] = -1;
	for (int i = 1; i <= n; i++)
		mm[i] = (i & (i-1) == 0) ? mm[i-1]+1 : mm[i-1];
	for (int i = 1; i <= n; i++)
		best[0][i] = i;
	for (int i = 1; i <= mm[n]; i++){
		for (int j = 1; j <= n+1-POW2(i); j++){
			a = best[i-1][j];
			b = best[i-1][j+POW2(i-1)];
			best[i][j] = (RMQ[a] < RMQ[b] ? a : b);
		}
	}
}

int askRMQ(int a, int b){
	int t = mm[b-a+1];
	b -= POW2(t)-1;
	a = best[t][a]; b = best[t][b];
	return RMQ[a] < RMQ[b] ? a : b;
}

int lcp(int a, int b){
	//0 <= a, b < n
	//lcp -> height[rank[]]的最<小>值
	a = rank[a];
	b = rank[b];
	if (a > b) swap(a, b);
	return height[askRMQ(a+1, b)];
}

//----------------------
//单个字符串相关问题
//----------------------
//一般先求后缀数组 sa[]，再求 height[],并据此计算
```

//可重叠最长重复子串——求出 height 数组的最大值即可
//时间复杂度 O(n)
//证明：res = max{pre(suf(i), suf(j))} = max{min(h[i+1], ..., h[j])} = max(h)

//不可重叠最长重复子串 -- 二分答案
//将排序后的数组分成若干组，使得每组的 height 后缀之间的 height 值都>=k
//则对于每组只需判断 sa 的最大值和最小值只差是否小于 k
//时间复杂度 O(nlogn)

//可重叠的 k 次最长重复字串 —— 二分答案
//这次是是否有一组中后缀个数>=k
//时间复杂度 O(nlogn)

//不同子串的个数
//对于后缀 suffix(sa[i])的 n-sa[i]个前缀，其中有 height[i]个是与前面相同的
//所以最终答案 sigma{n-sa[i]-height[i]}

//连续重复字串
//已知原字符串是由某字符串重复 R 次得到的，求 max{R}
//重复串长度 k：若 k|len AND lcp(suffix(0), suffix(k)) == n-k 则 k 是一个解
//ans = max{k}

//重复次数最多的连续重复字串
//枚举字串长度 L(L <= Len / 2)
//看 suffix(i * k)和 suffix((i+1) * k))可以匹配到多远 K
//max{K div L + 1}即为所求
//时间复杂度 O(n/1 + n/2 + n/3 + ... + n/n) = O(nlogn)

//--------------------
//两个字符串相关问题
//--------------------
//先将所有字符串连接起来，然后求后缀数组和 height 数组，
//在利用 height 数组进行求解。之间可能二分答案

//最长公共子串
//将 A$B 连接
//max{height[rank[i]] | (sa[rank[i]-1]-|A|)(i-|A|)<0, i = 0..n-1}
//时间复杂度 O(|A| + |B|)

//在每个字符串中至少出现两次且不重叠的最长字串长度
//将 n 个字符串连接(中间加$)，求后缀数组，二分答案
//按照后缀分组,比较后缀起始位置的最值只差是否小于当前答案
//时间复杂度 O(nlogn)

```
int main(){
	char r[100] = "aabaaaab";
	int sa[100];
	int sa2[100];
	int n = strlen(r);
	dc3(r, sa, n+1, 'z');
	da(r, sa2, n+1);
	cout << n << endl;
	for (int i = 0; i <= n; i++) cout << sa[i] << " "; cout << endl;
	for (int i = 0; i <= n; i++) cout << sa2[i] << " "; cout << endl;
	calheight(r, sa, n);
}
```

字符串

散列

对于负载因子小于 0.20 的散列表，如果扩展他们，不会提供更好的性能。
对于负载因子大于 0.50 的散列表，再散列将不是一种切实可行的解决方案。
一般来讲，哈希表中存入的元素个数要在哈希表总长度的 60%-90%之间，因此表的长度要悬在 1.1n-1.7n 之间。
大多数专业开发人员都使用拉链法来解决散列表冲突。
确保每个散列表中的槽数是一个素数。

```
unsigned ELFHash(char *s){ //在 32 位系统与 PJW Hash 效果相同
	unsigned hash = 0, x = 0;
	while (*s){
		hash = (hash << 4) + *s++;
		x = hash & 0xF0000000;
		if (x != 0) (hash ^= x >> 24) &= ~x;
	}
	return hash;
}

unsigned BKDRHash(char *s){
	static const unsigned seed = 13131; // 31 131 1313 13131 131313...
	unsigned hash = 0;
	while (s) hash = hash * seed + *s++;
	return hash;
}

unsigned RSHash(char *s){
	static const unsigned b = 378551;
```

```
    unsigned a = 63689, hash = 0;
    while (*s){
        hash = hash * a + *s++; a *= b;
    }
    return hash;
}

unsigned JSHash(char *s){
    unsigned hash = 1315423911; //nearly a prime = 3 * 438474637
    while (*s) hash ^= (hash << 5) + *s++ + (hash >> 2);
    return hash;
}

unsigned DJBHash(char *s){
    unsigned hash = 5381;
    while (*s) hash += (hash << 5) + *s++;
    return hash;
}
```

KMP

已经通过 http://cs.scu.edu.cn/soj/problem.action?id=2307，并且这道题用 Horspool 会超时
mpNext[]的意义：x[i-next[i]..i-1] = x[0..next[i]-1]
kmpNext[]的意思：next'[i] = next[next[..[next[i]]]]　　(till next'[i] < 0 or x[next' [i]] == x[i])

```
void preMP(const char x[], int m, int next[]){ //poj3461 - 110ms
    int i, j;
    i = next[0] = -1; j = 0;
    while (j < m) {
        while (i > -1 && x[i] != x[j]) i = next[i];
        next [++j] = ++i;
    }
}

void preKMP(const char x[], int m, int next []){ //poj3461 – 94ms
    int i, j;
    i = next [0] = -1; j = 0;
    while (j < m){
        while (i > -1 && x[i] != x[j]) i = next [i];
        if (x[++i] == x[++j]) next[j] = next[i]; else next[j] = i;
    }
}

int kmp(const char x[], int m, const char y[], int n){ //x 是模式串，y 是主串
```

```
    int i, j, ret = 0;
    preKMP(x, m, next);
    i = j = 0;
    while (j < n){
        while (i > -1 && x[i] != y[j]) i = next[i];
        ++i; ++j;
        if (i >= m) {
            //OUTPUT(j - i);
            ret++;
            i = next[i];
        }
    }
    return ret;
}
```

exkmp

```
void prexkmp(const char x[], int m, int next[]){
    int j, k = 1, r = 0; next[0] = m;
    for (int i = 1; i < m; ++i){
        if ( i + next[i-k] < r ){
            next[i] = next[i-k];
        } else {
            for (j = max(r-i, 0); i + j < m && x[i+j] == x[j]; ++j );
            next[i] = j; k = i; r = i + j;
        }
    }
}

// next[i] : x[i..m-1]与 x[0..m-1]的最长公共前缀长度
//    ext[i] : y[i..n-1]与 x[0..m-1]的最长公共前缀长度
void exkmp(const char x[], int m, const char y[], int n, int next[], int ext[]){
    int k = 0, r = 0, j;
    prexkmp(x, m, next); next[0] = 0;
    for(int i = 0; i < n; ++i){
        if ( i + next[i-k] < r ){
            ext[i] = next[i-k];
        } else {
            for (j = max(r-i, 0); j < m && i+j < n && x[j] == y[i+j]; ++j );
            ext[i] = j; k = i; r = i + j;
        }
    }
}
```

字符串的最小表示

注：判断 A 和 B 旋转同构可以看 A 是否是 BB 的子串

1. 将字符串 s 加倍
2. 利用两个游标 i, j (不妨设 i < j)。初始化时 i 对应 s[0], j 对应 s[1]
3. k = 0 开始，检验 s[i+k]与 s[j+k]对应的字符是否相等
   a) s[i+k] > s[j+k]，则 i 滑动到 max(i+k+1, j+1)处
   b) s[i+k] < s[j+k]，则 j 滑动到 max(j+k+1, i+1)处
   c) s[i+k] = s[j+k]，则 k++
   注：若滑动后 i = j，则 j++；若滑动后 i > j，则 swap(i, j)
4. 终止条件
   a) 若 k = |s|，则返回 i
   b) 若 j >= |s|，则返回 i

代码

```
int minCircularDenote(char *s, int n){ //返回：字符串最小表示的首字母位置
    //已经通过 poj1509
    strncpy(s + n, s, n); //注意 s 会被 double
    int i = 0, j = 1;
    for (;;){
        int k = 0;
        while (k < n && s[i+k] == s[j+k]) k++;
        if (k == n) return i;
        if (s[i+k] > s[j+k]) i = max(i+k+1, j+1); else j = max(j+k+1, i+1);
        if (i == j) j++; else if (i > j) swap(i, j);
        if (j >= n) return i;
    }
}
```

AC 自动机/dfa

AC 自动机 – 统计每个模式串在目标串出现的次数

```
//已经通过某题、poj1204
const int maxc = 20000; //节点总数

struct node {
    node *fail, *next[26]; //这个数字可能需要改
    int iter; //指向原串的指针
    int cnt; //节点到达的次数
```

```
    void * operator new(size_t);
} *root, *null, pool[maxc], *pooltop;

typedef node *pnode;
pnode Queue[maxc], *Qbase, *Qtop; //注意队列长度一定要开到 maxc, 而不是 maxn

void *node::operator new(size_t){ memset(pooltop, 0, sizeof *pooltop); return pooltop++; }

void insert(char s[], int iter){
    node *p = root;
    for (int i; *s; p = p->next[i]){
        i = *s++ - 'a'; //请修改这里
        if (!p->next[i]) p->next[i] = new node;
    }
    p->iter = iter;
}

void build_ac(){
    pnode p, q;
    Qbase = Qtop = Queue; *Qtop++ = root;
    for (int i = 0; i < 26; i++) //这个数字可能需要改
        null->next[i] = root;
    root->fail = null;
    while (Qbase != Qtop){
        p = *Qbase++;
        for (int i = 0; i < 26; i++){ //这个数字可能需要改
            if (p->next[i]){
                q = p->fail;
                while (!q->next[i]) q = q->fail;
                p->next[i]->fail = q->next[i];
                *Qtop++ = p->next[i];
            } //else {
                //q = p->fail;
                //while (q != next[i]) q = q->fail;
                //p->next[i] = q->next[i];
            //}
        }
        //if (p->fail->iter) p->iter = p->fail->iter; //可能需要添加这句话~~~~
    } //end while
}

char x[155][100];
int cnt[155];
char s[10000000];
```

```
void query(char *s){
	pnode p = root, q;
	while (*s){
		int i = *s++ - 'a'; //请修改这里
		q = p;
		while (!p->next[i]) p = p->fail;
		p = p->next[i];
		for (; !q->next[i]; q = q->fail) q->next[i] = p; //路径压缩
		++p->cnt;
	}
	for (pnode *p = Qtop - 1; p >= Queue; p--){
		q = *p; q->fail->cnt += q->cnt;
		if (q->iter) cnt[q->iter] = q->cnt;
	}
}

int main(){
	int n;
	while (scanf("%d", &n), gets(s), n){
		pooltop = pool; null = new node; root = new node;
		for (int i = 1; i <= n; i++){
			cnt[i] = 0; gets(x[i]); insert(x[i], i);
		}
		build_ac();
		gets(s); query(s);
		int maxcnt = *max_element(cnt + 1, cnt + 1 + n);
		for (int i = 1; i <= n; i++)
			for (int j = i + 1; j <= n; j++)
				if (strcmp(x[i], x[j]) == 0)
					cnt[i] = cnt[j] = max(cnt[i], cnt[j]);
		printf("%d\n", maxcnt);
		for (int i = 1; i <= n; i++)
			if (maxcnt == cnt[i])
				puts(x[i]);
	}
}
```

dfa-dp

```
const int maxc = 200;
int hash[255];
```

```
void checkmax(int &a, int b){ if (a < b) a = b; }

struct node {
	node *fail, *next[4];
	int flag, ind, is_terminal, length;
	void *operator new(size_t);
} *root, *null, pool[maxc], *pooltop;

typedef node *pnode;
pnode Queue[maxc], *Qbase, *Qtop;

void *node::operator new(size_t){
	memset(pooltop, 0, sizeof *pooltop);
	pooltop->length = -1;
	return pooltop++;
}

void insert(char *s, int iter){
	node *p = root;
	int L = strlen(s);
	for (int i; *s; p = p->next[i]){
		i = hash[*s++];
		if (!p->next[i]) p->next[i] = new node;
	}
	p->flag |= 1 << iter;
	p->is_terminal = 1;
	checkmax(p->length, L );
}

void build_ac(){
	pnode p, q;
	Qbase = Qtop = Queue;
	*Qtop = root; root->ind = Qtop - Queue; Qtop++;
	for (int i = 0; i < 4; i++)
		null->next[i] = root;
	root->fail = null;
	while (Qbase != Qtop){
		p = *Qbase++;
		for (int i = 0; i < 4; i++){
			if (p->next[i]){
				q = p->fail;
				while (!q->next[i]) q = q->fail;
				p->next[i]->fail = q->next[i];
				*Qtop = p->next[i]; p->next[i]->ind = Qtop - Queue; Qtop++;
```

```
                checkmax( p->next[i]->length, p->next[i]->fail->length );
                p->next[i]->flag |= p->next[i]->fail->flag;
            } else {
                q = p->fail;
                while (!q->next[i]) q = q->fail;
                p->next[i] = q->next[i];
            }
        }
    }
}

int dp[2000][200][12];
const int p = 1000000009;

void add(int &a, int b){
    ((a += b) >= p) ? a -= p : a;
}

int main(){ //codeforce 86C
    hash['A'] = 0; hash['T'] = 1; hash['G'] = 2; hash['C'] = 3;

    int n, m;
    char s[100];
    while ( scanf("%d%d", &n, &m) == 2 ){
        pooltop = pool; null = new node; root = new node;
        for (int i = 0; i < m; i++){
            scanf("%s", s); insert(s, i);
        }
        build_ac();
        int cnt = Qtop - Queue;
        for (int i = 0; i <= n; i++)
            for (int j = 0; j < cnt; j++)
                for (int k = 0; k < 12; k++)
                    dp[i][j][k] = 0;
        #define Next(j, k) (Queue[j]->next[k]->ind)
        #define Flag(j) (Queue[j]->flag)
        #define IsTerminal(j) ( Queue[j]->is_terminal    )
        dp[0][0][0] = 1;

        for (int i = 0; i <= n; i++){
            for (int j = 0; j < cnt; j++){
                for (int L = 0; L < 10; L++){
                    if (dp[i][j][L]){
                        for (int k = 0; k < 4; k++){
```

```
                                int nj = Next(j, k);
                                if (L+1 <= Queue[nj]->length){
                                        add(dp[i+1][nj][0], dp[i][j][L]);
                                } else {
                                        add(dp[i+1][nj][L+1], dp[i][j][L]);
                                }
                        }
                }
        }
        int res = 0;
        for (int j = 0; j < cnt; j++){
                add(res, dp[n][j][0]);
        }
        cout << res << endl;
    }
}
```

AC 自动机 – java

已经通过 zju、ural censored!, (poj 数据有误)

```
class node {
    static int count = 0;
    public node fail, next[];
    public boolean isTerminal;
    public int index;
    public node() {
        fail = null;
        next = new node[Main.n];
        isTerminal = false;
        index = count++;
    }
}

public class Main {
    private static node root;
    private static node nil;
    private static String alphabet;
    public static int n;
    private static int sigma[][];
    private static boolean isTerminal[];

    public static void insert(String s){
```

```
        node p = root;
        for (int i = 0; i < s.length(); i++){
            int j = alphabet.indexOf( s.charAt(i) );
            if (p.next[j] == null) p.next[j] = new node();
            p = p.next[j];
        }
        p.isTerminal = true;
    }
    public static void build_ac(){
        Queue<node> Q = new LinkedList<node>();
        for (int i = 0; i < alphabet.length(); i++)
            nil.next[i] = root;
        root.fail = nil;
        Q.add(root);
        while ( ! Q.isEmpty() ){
            node p = Q.poll(), q;
            for (int i = 0; i < n; i++){
                if (p.next[i] != null){
                    q = p.fail;
                    while (q.next[i] == null) q = q.fail;
                    p.next[i].fail = q.next[i];
                    Q.add(p.next[i]);
                    if (p.next[i].fail.isTerminal) {
                        p.next[i].isTerminal = true;
                    }
                } else { // p.next[i] == null
                    q = p.fail;
                    while (q.next[i] == null) q = q.fail;
                    p.next[i] = q.next[i];
                }
                sigma[ p.index ][ i ] = p.next[i].index;
            }
            if (p.isTerminal) isTerminal[p.index] = true;
        }
    }
    public static void main(String[] args) {
        Scanner cin = new Scanner(new BufferedInputStream(System.in));
        while (cin.hasNext()){
            n = cin.nextInt();
            int m = cin.nextInt();
            int p = cin.nextInt();
            alphabet = cin.next();

            node.count = 0;
```

```
                nil = new node();
                root = new node();
                for (int i = 0; i < p; i++){
                    String s = cin.next();
                    insert(s);
                }
                sigma = new int[node.count][n];
                isTerminal = new boolean[node.count];
                build_ac();

                BigInteger dp[][] = new BigInteger[m+1][node.count];
                for (int i = 0; i <= m; i++)
                    for (int j = 1; j < node.count; j++)
                        dp[i][j] = BigInteger.ZERO;
                dp[0][1] = BigInteger.ONE;
                for (int i = 0; i < m; i++){
                    for (int j = 1; j < node.count; j++){
                        if (isTerminal[j]) continue;
                        if (dp[i][j] != BigInteger.ZERO){
                            for (int k = 0; k < n; k++){
                                if (isTerminal[ sigma[j][k] ]) continue;
                                dp[i+1][ sigma[j][k] ] =
                                    dp[i+1][ sigma[j][k] ].add(dp[i][j]);
                            }
                        }
                    }
                }
                BigInteger res = BigInteger.ZERO;
                for (int j = 1; j < node.count; j++){
                    if (isTerminal[j]) continue;
                    res = res.add( dp[m][j] );
                }
                System.out.println( res );
            }
        }
    }
```

AC 自动机数位 dp

```
//AC 自动机数位 dp( 1..n 的数中 BCD 码不含有特定的子串的数的个数, zju3494 )
//上次使用 2012.6.1,  基本确保正确
struct AekdyCoin {
    const static int maxc = 2048 + 5; //节点总数
```

```
    const static int alphabet = 2; // 0-1，请修改这里
public:
    void init(){
        c = 0; root = newNode();
    }
    void insert(const char *s, int n){
        node *p = root;
        for (int i = 0; i < n; i++){
            int j = s[i] - '0'; //请修改这里
            if ( !p->next[j] ) p->next[j] = newNode();
            p = p->next[j];    //别忘记添加
        }
        p->bad = 1;
    }
    void build(){
        Qbase = Qtop = Queue;
        for (int j = 0; j < alphabet; j++){
            if ( root->next[j] ){
                root->next[j]->fail = root;
                *Qtop++ = root->next[j];
            } else {
                root->next[j] = root;
            }
        }
        while ( Qbase != Qtop ){
            node *p = *Qbase++;
            p->bad |= p->fail->bad; //
            for (int j = 0; j < alphabet; j++){
                if (p->next[j]){
                    p->next[j]->fail = p->fail->next[j];
                    *Qtop++ = p->next[j];
                } else {
                    p->next[j] = p->fail->next[j];
                }
            }
        }
    }
public:
    struct node {
        node *fail, *next[alphabet]; int bad;
    } pool[maxc];
    int c;
private:
    node *root;
```

```
    node *Queue[maxc], **Qbase, **Qtop;
    node *newNode() {
        memset( &pool[c], 0, sizeof(node) );
        return &pool[c++];
    }
} ac;

const int maxn = 218 + 5;
const LL module = 1000000009;

int next[AekdyCoin::maxc][10]; //next[i][j]:表示处于状态 i，碰到数字 j 转移到的状态
LL dp[maxn][AekdyCoin::maxc]; //dp[i][j]:从状态 j 出发,长为 i 的路径的个数(可以有前导 0)

int getNext(int i, int x){
    AekdyCoin::node *p = &ac.pool[i];
    if ( p->bad ) return -1;
    for (int k = 3; k >= 0; k--){
        p = p->next[(x & (1 << k)) ? 1 : 0];
        if ( p->bad ) return -1;
    }
    return p - ac.pool;
}

char *dec(char *s){
    int n = strlen(s);
    for (int i = n - 1; i >= 0; i--){
        if (s[i] == '0'){
            s[i] = '9';
        } else {
            --s[i]; break;
        }
    }
    if (s[0] == '0') s++;
    return s;
}

//当前处于状态 p，后面的字符串为 s[0..n-1]的个数(可以有前导 0)
LL query(const char* s, int n, int p){
    if (p == -1) return 0;
    if (n == 0) return 1;
    LL ret = 0;
    for (int i = 0; i < *s - '0'; i++){
        int q = next[p][i];
        if (q != -1) ret += dp[n-1][q];
```

```
    }
    ret += query(s+1, n-1, next[p][*s-'0']);
    return ret % module;
}

//[1, s]的符合要求的数的个数(不能有前导 0)
LL query(const char *s){
    if ( !s[0] ) return 0;
    int n = strlen(s);
    LL ret = query(s + 1, n - 1, next[0][*s-'0']);
    for (int i = 1; i <= n; i++){ //长度为 i 的数
        for (int j = 1; j < (i == n ? *s-'0': 10); j++){
            int p = next[0][j];
            if (p != -1) ret += dp[i-1][p];
        }
    }
    return ret %= module;
}

void solve(){
    static char s[maxn];

    int n; cin >> n;
    ac.init();
    for (int i = 0; i < n; i++)
        scanf("%s", s), ac.insert(s, strlen(s));
    ac.build();

    for (int i = 0; i < ac.c; i++)
        for (int j = 0; j < 10; j++)
            next[i][j] = getNext(i, j);
    for (int j = 0; j < ac.c; j++)
        dp[0][j] = ac.pool[j].bad ? 0 : 1;
    for (int i = 1; i < maxn; i++){
        for (int j = 0; j < ac.c; j++){
            dp[i][j] = 0;
            if ( !ac.pool[j].bad ){
                for (int k = 0; k < 10; k++){
                    int jj = next[j][k];
                    if (jj != -1) dp[i][j] += dp[i-1][jj];
                }
                dp[i][j] %= module;
            }
        }
```

```
    }

    scanf("%s", s); LL a = query( dec(s) );
    scanf("%s", s); LL b = query(s);
    LL ans = (b + module - a) % module; //否则会出现负数
    cout << ans << endl;
}

int main(){
    int re; cin >> re;
    for (int ri = 1; ri <= re; ri++) solve();
}
```

字符串最小重复单元的重复次数 – kmp

```
int LongestRepeatedSubstring(char s[]){
    int m = strlen(s);
    preMP(s, m, next);
    return m % (m - next[m]) == 0 ? m / (m - next[m]) : 1;
}
```

Manacher 算法

```
// s[i]         为中心  -> p[i*2+2]
// s[i,i+1]为中心  -> p[i*2+3]
// p[j] : t[]以 j 为中心,s[]以(j-1)*0.5 为中心的最长回文串长度
//       -> s.substr( (j-1-p[j])/2, p[j] )
void Manacher(char s[maxn], int p[maxn*2+5]){
    static char t[maxn * 2 + 5];

    int n = 0;
    t[n++] = '^'; // of importance
    for (char *is = s; *is; is++ )
        t[n] = '#', t[n+1] = *is, n += 2;
    t[n++] = '#'; t[n] = 0;

    int c = 0, r = 0; p[0] = 0;
    for (int i = 1; i < n; i++){
        int j = 2 * c - i;
        p[i] = r > i ? min(r-i, p[j]) : 0;
        while ( t[ i+1+p[i] ] == t[ i-1-p[i] ] )
            p[i]++;
        if ( i+p[i] > r )
```

```
                c = i, r = i + p[i];
        }
}

struct node {
        int v; node *next;
} *enSetSlack[maxn], *deSetSlack[maxn], pool[maxn * 2], *pooltop;

void addedge(node *ge[], int a, int b){
        node *p = pooltop++; p->v = b; p->next = ge[a]; ge[a] = p;
}

void init(int n){
        for (int i = 0; i <= n; i++)
                enSetSlack[i] = 0, deSetSlack[i] = 0;
        pooltop = pool;
}

int main(){ //双倍回文, http://www.lydsy.com/JudgeOnline/problem.php?id=2342
        for (int n;cin >> n;){
                static char s[maxn]; static int p[maxn*2+5];
                scanf("%s", s); manacher(s, p);

                set<int> ss;
                init(n);
                for (int i = 0; i < n-1; i++){
                        if (p[i*2+3]){
                                addedge(enSetSlack, i+1, i);
                                addedge(deSetSlack, i+p[i*2+3]/2, i );
                        }
                }
                int maxradius = 0;
                for (int i = 0; i < n-1; i++){
                        for (node *it = enSetSlack[i]; it; it = it->next)
                                ss.insert( it->v );
                        int radius = p[i*2+3] / 4;
                        set<int>::iterator iter = ss.lower_bound( i - radius );
                        if ( iter != ss.end() && *iter < i )
                                maxradius = max(maxradius, i - *iter );
                        for (node *it = deSetSlack[i]; it; it = it->next)
                                ss.erase( it->v );
                }
                cout << maxradius * 4 <<endl;
        }
```

```
}
```

kdtree

```
const LL inf = 1000000000;
const int maxn = 50000 + 5;
const int maxk = 15;
const int maxdim = 5;

typedef pair<LL, int> Node;
#define dist first
#define ind second

LL sqr(LL x) { return x * x; }

struct Heap {
	Node v[maxk];
	int sz;
	void push(const Node& o) {
		v[sz++] = o; push_heap(v, v + sz);
	}
	void pop() {
		pop_heap(v, v + sz--);
	}
	Node top() const {
		return v[0];
	}
	void sort(){
		sort_heap(v, v + sz);
	}
	void init(int k){
		rep(i, k) v[i] = Node(inf, -1);
		sz = k;
	}
} hp;

int dim;

struct Space {
	struct Point {
		int v[maxdim];
		void read(){
			rep(i, dim) scanf("%d", &v[i]);
		}
```

```
    void write()const {
        rep(i, dim) printf("%d%c", v[i], " \n"[i == dim - 1]);
    }
    friend LL dist(const Point& a, const Point& b) {
        LL ans = 0;
        rep(i, dim) ans += sqr( a.v[i] - b.v[i] );
        return ans;
    }
};

struct LessPoint {
    bool operator()(const Point& a, const Point& b) const {
        return a.v[axis] < b.v[axis];
    }
    LessPoint(int axis) : axis(axis) {
    }
    int axis;
};

Point p[maxn];
int partition[maxn];
int n;
void read(int n){
    this->n = n;
    for (int i = 0; i < n; ++i) p[i].read();
}
void build(int left, int right, int axis) {
    if ( left > right ) return;
    int mid = (left + right) / 2;
    partition[mid] = axis;
    nth_element( p + left, p + mid, p + right + 1, LessPoint(axis) );
    axis = (axis + 1) % dim;
    build(left, mid - 1, axis); build(mid + 1, right, axis);
}

void query(int left, int right, const Point& q) {
    if ( left > right ) return;
    int mid = (left + right) / 2;
    LL d = dist(q, p[mid]);
    if ( d < hp.top().dist )
        hp.pop(), hp.push( Node(d, mid) );
    int left1 = left, right1 = mid - 1;
    int left2 = mid + 1, right2 = right;
    int axis = partition[mid];
```

```
        if ( p[mid].v[axis] < q.v[axis] )
            swap(left1, left2), swap(right1, right2);
        query(left1, right1, q);
        if ( sqr(p[mid].v[axis] - q.v[axis]) < hp.top().dist )
            query(left2, right2, q);
    }

    int *nearest(const Point &q, int k) {
        static int ans[maxk];
        hp.init(k); query(0, n-1, q); hp.sort();
        rep(i, k) ans[i] = hp.v[i].ind;
        return ans;
    }

    void solve(int n, int dim) {
        ::dim = dim;
        read(n); build(0, n-1, 0);
        int qcnt, k; Point q;
        cin >> qcnt;
        while (qcnt--) {
            q.read(); scanf("%d", &k);
            int *ans = nearest(q, k);
            printf("the closest %d points are:\n", k);
            rep(i, k) p[ans[i]].write();
        }
    }
} sp;

int main(){ for (int n, dim; cin >> n >> dim; solve(n, dim) ); }
```

线段树

矩形面积并

```
const int maxn = 100000 + 5; //上次使用 2012.9.23
typedef long long LL;

struct IntervalTree {
    #define pl (p << 1)
    #define pr (pl | 1)
    int time[maxn * 4], len[maxn * 4];
    int L, R; const int *v;
    void init(int v[], int L, int R) {
```

```
        this->L = L; this->R = R; this->v = v; build(1, L, R);
    }
    void update(int left, int right, int d) {
        update(1, left, right, L, R, d);
    }
    int query() {
        return time[1] ? v[R] - v[L] : len[1];
    }
private:
    void build(int p, int L, int R) {
        time[p] = len[p] = 0;
        if (L + 1 != R) {
            int M = (L + R) / 2; build(pl, L, M); build(pr, M, R);
        }
    }
    void update(int p, int left, int right, int L, int R, int d) {
        if ( left == L && right == R ) {
            time[p] += d;
        } else {
            int M = (L + R) / 2;
            push(p);
            if ( right <= M ) {
                update(pl, left, right, L, M, d);
            } else if ( left >= M ) {
                update(pr, left, right, M, R, d);
            } else {
                update(pl, left, M, L, M, d);
                update(pr, M, right, M, R, d);
            }
            pull(p);
            len[p] = (time[pl] ? v[M] - v[L] : len[pl]) + (time[pr] ? v[R] - v[M] : len[pr]);
        }
    }
    void push(int p) {
        time[pl] += time[p]; time[pr] += time[p]; time[p] = 0;
    }
    void pull(int p) {
        time[p] = min(time[pl], time[pr]); time[pl] -= time[p]; time[pr] -= time[p];
    }
    #undef pl
    #undef pr
};

int v[maxn * 2], vcnt;
```

```
struct Rect {
	char color;
	int x1, y1, x2, y2;
	void read(){
		static char tmp[5]; scanf("%s", tmp); color = tmp[0];
		scanf("%d%d%d%d", &x1, &y1, &x2, &y2);
		v[vcnt++] = y1; v[vcnt++] = y2;
	}
} rect[maxn];

struct Segment {
	int x, y1, y2;
	int flag;
	Segment(){
	}
	Segment(int x, int y1, int y2, int flag) : x(x), y1(y1), y2(y2), flag(flag) {
	}
	bool operator < (const Segment& o) const {
		return x < o.x;
	}
} seg[maxn];

LL gao(int n, const char *cc) {
	int segcnt = 0;
	for (int i = 0; i < n; ++i) {
		if ( !strchr(cc, rect[i].color) ) continue;
		seg[ segcnt++ ] = Segment(rect[i].x1, rect[i].y1, rect[i].y2, 1);
		seg[ segcnt++ ] = Segment(rect[i].x2, rect[i].y1, rect[i].y2, -1);
	}
	sort( seg, seg + segcnt );
	static IntervalTree tree;
	tree.init( v, 0, vcnt - 1 );
	int lastX = 0;
	LL ans = 0;
	for (int i = 0, j; i < segcnt; i = j) {
		for (j = i + 1; j < segcnt && seg[i].x == seg[j].x; ++j);
		LL len = tree.query();
		ans += (seg[i].x - lastX) * len; lastX = seg[i].x;
		for (int k = i; k < j; ++k) {
			int left = lower_bound(v, v + vcnt, seg[k].y1) - v;
			int right = lower_bound(v, v + vcnt, seg[k].y2) - v;
			tree.update(left, right , seg[k].flag);
		}
```

```
	}
	return ans;
}

void solve(int ri) {
	int n; cin >> n;
	vcnt = 0;
	for (int i = 0; i < n; ++i) {
		rect[i].read();
	}
	sort(v, v + vcnt); vcnt = unique(v, v + vcnt) - v;
	LL R = gao(n, "R"), G = gao(n, "G"), B = gao(n, "B");
	LL RorG = gao(n, "RG"), RorB = gao(n, "RB"), GorB = gao(n, "GB");
	LL RorGorB = gao(n, "RGB");
	LL RandG = R + G – RorG, RandB = R + B – RorB, GandB = G + B - GorB;
	LL RandGandB = RorGorB - R - G - B + RandG + RandB + GandB;

	LL ansR = R - RandG - RandB + RandGandB;
	LL ansG = G - RandG - GandB + RandGandB;
	LL ansB = B - RandB - GandB + RandGandB;
	LL ansRG = RandG - RandGandB;
	LL ansGB = GandB - RandGandB;
	LL ansRB = RandB - RandGandB;
	LL ansRGB = RandGandB;
	printf("Case %d:\n", ri);
	cout << ansR << endl;
	cout << ansG << endl;
	cout << ansB << endl;
	cout << ansRG << endl;
	cout << ansRB << endl;
	cout << ansGB << endl;
	cout << ansRGB << endl;
}

int main(){
//	freopen("input.txt", "r", stdin);
	int re; cin >> re; for (int ri = 1; ri <= re; ++ri) solve(ri);
}
```

在平面上每次删掉一个点，删掉点 p[i]后，会把曼哈顿距离 <=d[i]的点都递归的删掉，问每次回递归的删掉几个点？

进行坐标转化(x, y) -> (X, Y) = (x+y, x-y)，则变成把矩形[(X-d, Y-d) - (X+d,Y+d)]的点删掉，用线

段树套并查集即可(复杂度 nlognlogn)

```
const int oo = 2000000000;
static const int maxn = 100000 + 5;

struct Item {
	pair<int, int> c;
	int d, ind;
	void read(){
		scanf("%d%d%d", &c.x, &c.y, &d);
		c = make_pair( c.x + c.y, c.x - c.y );
	}
} item[maxn], *entry[maxn];

vector<Item *> dat[maxn << 1];
vector<int> xs, next[maxn << 1];
int n, vis[maxn], x_min, x_max, y_min, y_max, front, rear, que[maxn];

struct lessX { bool operator()(const Item *i, const Item* j) const { return i->c.x < j->c.x; } };
struct lessY { bool operator()(const Item *i, const Item* j) const { return i->c.y < j->c.y; } };
#define POS ( ((L) + (R)) | ((L) != (R)) )

void build(int L, int R) {
	int low = 0, high = n - 1, ans = high + 1;
	while ( low <= high ) {
		int mid = low + high >> 1;
		if ( entry[mid]->c.x >= xs[L] ) {
			ans = mid; high = mid - 1;
		} else {
			low = mid + 1;
		}
	}
	dat[POS].clear();
	for (int k = ans; k < n && entry[k]->c.x <= xs[R]; ++k ) dat[POS].push_back( entry[k] );
	sort( dat[POS].begin(), dat[POS].end(), lessY() );
	next[POS].resize( dat[POS].size() );
	for (int k = 0; k < next[POS].size(); ++k) next[POS][k] = k;
	if (L != R) {
		int M = L + R >> 1; build(L, M); build(M+1, R);
	}
}

int find(vector<int> &next, int i) {
	if ( i >= next.size() ) return next.size();
```

```
    return next[i] == i ? i : next[i] = find(next, next[i]);
}

void query(int L, int R) {
    if ( x_max < xs[L] || xs[R] < x_min    ) return;
    if ( x_min <= xs[L] && xs[R] <= x_max ) {
        int low = 0, high = dat[POS].size() - 1, ans = high + 1;
        while ( low <= high ) {
            int mid = low + high >> 1;
            if (    dat[POS][mid]->c.y >= y_min ) {
                ans = mid; high = mid - 1;
            } else {
                low = mid + 1;
            }
        }
        for (int k = find(next[POS], ans); k < next[POS].size() && dat[POS][k]->c.y <= y_max;
                                k = find(next[POS], k) ) {
            if ( !vis[ dat[POS][k]->ind ] ) {
                vis[ dat[POS][k]->ind ] = true; que[rear++] = dat[POS][k]->ind;
            }
            next[POS][k] = k + 1;
        }
    } else {
        int M = L + R >> 1; query(L, M); query(M+1, R);
    }
}

void solve(){
    xs.clear();
    for (int i = 0; i < n; ++i) {
        item[i].read(); item[i].ind = i; entry[i] = &item[i]; xs.push_back( item[i].c.x );
    }
    sort( entry, entry + n, lessX() );
    sort( xs.begin(), xs.end() ); xs.erase( unique(xs.begin(), xs.end()), xs.end() );
    int m = xs.size() - 1;
    build(0, m);
    int q; cin >> q;
    memset(vis, 0, sizeof vis);
    static int ri = 0; printf("Case #%d:\n", ++ri);
    while (q--) {
        int id; scanf("%d", &id); --id;
        if ( vis[id] ) {
            puts("0");
        } else {
```

```
            front = rear = 0; vis[id] = 1; que[rear++] = id;
            while ( front != rear ) {
                int u = que[front++];
                x_min = max( (LL)item[u].c.x - item[u].d, (LL)-oo );
                x_max = min( (LL)item[u].c.x + item[u].d, (LL)+oo );
                y_min = max( (LL)item[u].c.y - item[u].d, (LL)-oo );
                y_max = min( (LL)item[u].c.y + item[u].d, (LL)+oo );
                query(0, m);
            }
            printf("%d\n", rear);
        }
    }
};

int main(){ while (cin >> n && n) solve(); }
```

可持久化数据结构

函数式线段树

改段问段+询问历史

```
class FunctionalIntervalTree {
    const static int maxn = 100000 + 5;

    struct node {
        int L, R; LL sum, delta; node *pL, *pR;
        void init(int L, int R, LL sum, LL delta, node *pL, node *pR) {
            this->L = L; this->R = R; this->sum = sum; this->delta = delta;
            this->pL = pL; this->pR = pR;
        }
    } pool[maxn * 200], *pooltop, *history[maxn];
    int cur;

    node *newnode(int L, int R, LL sum, LL delta, node *pL, node *pR) {
        pooltop->init(L, R, sum, delta, pL, pR); return pooltop++;
    }

    node *build(int L, int R) {
        if (L == R) {
            int v; scanf("%d", &v); return newnode(L, R, v, 0, NULL, NULL);
```

```
    } else {
        int M = (L + R) / 2; node *pL = build(L, M), *pR = build(M+1, R);
        return newnode(L, R, pL->sum + pR->sum, 0, pL, pR );
    }
}

node *update(node *p, int L, int R, int d) {
    LL sum, delta = p->delta;
    node *pL = p->pL, *pR = p->pR;
    if (L == p->L && R == p->R) {
        sum = p->sum + (LL)d * (p->R - p->L + 1); delta += d;
    } else {
        int MM = (p->L + p->R) / 2;
        if ( R <= MM ) {
            pL = update(p->pL, L, R, d);
        } else if ( L >= MM + 1 ) {
            pR = update(p->pR, L, R, d);
        } else {
            pL = update(p->pL, L, MM, d);
            pR = update(p->pR, MM+1, R, d);
        }
        sum = pL->sum + pR->sum + delta * (p->R - p->L + 1);
    }
    return newnode(p->L, p->R, sum, delta, pL, pR);
}

LL query(node *p, int L, int R) {
    if (L == p->L && R == p->R) {
        return p->sum;
    } else {
        int MM = (p->L + p->R) / 2;
        LL ans = p->delta * (R - L + 1);
        if (R <= MM) {
            ans += query(p->pL, L, R);
        } else if (L >= MM + 1) {
            ans += query(p->pR, L, R);
        } else {
            ans += query(p->pL, L, MM);
            ans += query(p->pR, MM+1, R);
        }
        return ans;
    }
}
```

```
public:
    void init(int n){
        pooltop = pool; cur = 0; history[0] = build(1, n);
    }

    LL query(int L, int R, int t = -1) {
        return query(history[t != -1 ? t : cur], L, R);
    }

    void update(int L, int R, int d) {
        history[cur+1] = update(history[cur], L, R, d); ++cur;
    }

    void rollback(int t) {
        cur = t; pooltop = history[cur] + 1;
    }
} tree;

void solve(int n, int m) {
    tree.init(n);
    char cmd[5]; LL ans;
    for (int x, y, z; m--; ) {
        scanf("%s", cmd);
        switch (cmd[0]) {
            case 'Q':
                scanf("%d%d", &x, &y); ans = tree.query( x, y );
                printf("%I64d\n", ans); break;
            case 'H':
                scanf("%d%d%d", &x, &y, &z); ans = tree.query( x, y, z );
                printf("%I64d\n", ans); break;
            case 'C':
                scanf("%d%d%d", &x, &y, &z); tree.update( x, y, z); break;
            case 'B':
                scanf("%d", &z); tree.rollback(z); break;
        }
    }
}

int main(){ for (int n, m; cin >> n >> m; solve(n, m) ); }
```

区间第 k 小数 – 不可修改 – nlogn

上次使用 2012-8-28， 已经通过 poj2104

```
const int maxn = 100000 + 5;
const int inf = 1000000000 + 5;

class Interval_Kth {
    struct node {
        int L, R, sz;
        node *pL, *pR;
        void init(int L, int R, int sz, node *pL, node *pR) {
            this->L = L; this->R = R; this->sz = sz; this->pL = pL; this->pR = pR;
        }
    } pool[maxn * 100], *pooltop, *tree[maxn], *null;

    node *newnode(int L, int R, int sz, node *pL, node *pR) {
        pooltop->init(L, R, sz, pL, pR); return pooltop++;
    }
private:
    node *insert(node *p, int L, int R, int v) {
        if (L == R) {
            return newnode(L, R, p->sz + 1, null, null );
        } else {
            node *pL = p->pL, *pR = p->pR;
            int M = L + (R - L) / 2; //千万别用  (L + R) / 2
            if ( v <= M ) {
                pL = insert(pL, L, M, v);
            } else {
                pR = insert(pR, M+1, R, v);
            }
            return newnode(L, R, pL->sz + pR->sz, pL, pR);
        }
    }
    node *insert(node *p, int v) {
        return insert(p, -inf, inf, v);
    }
    int query(node *t1, node *t2, int k) {
        if ( t1->L == t1->R ) {
            return t1->L;
        } else {
            int d = k - (t1->pL->sz - t2->pL->sz);
            if ( d <= 0 ) {
                return query(t1->pL, t2->pL, k);
            } else {
                return query(t1->pR, t2->pR, d);
            }
        }
```

```
    }
public:
    void init(int a[], int n){
        pooltop = pool; tree[0] = null = newnode(-inf, inf, 0, pooltop, pooltop);
        for (int i = 1; i <= n; ++i) tree[i] = insert( tree[i-1], a[i] );
    }
    int kth(int L, int R, int k) { // 1 <= k <= R - L + 1
        return query(tree[R], tree[L-1], k);
    }
} task;

void solve(int n, int q) {
    static int a[maxn]; for (int i = 1; i <= n; ++i) scanf("%d", &a[i]);
    task.init(a, n);
    for (int L, R, k; q--; ) scanf("%d%d%d", &L, &R, &k), printf("%d\n", task.kth(L, R, k) );
}

int main() { for (int n, q; cin >> n >> q; solve(n, q) ); } // poj2104
```

区间第 k 小数 – 可修改 – nlognlogn

```
//已经通过 bzoj1901，而 zoj2112 超内存，上次使用 2012-8-28
const int inf = 1000000000 + 5;

class Interval_Kth {
    struct node {
        int L, R, sz; node *pL, *pR;
        void init(int L, int R, int sz, node *pL, node *pR) {
            this->L = L; this->R = R; this->sz = sz; this->pL = pL; this->pR = pR;
        }
    } pool[maxn * 120], *pooltop, *tree[maxn], *null;
    int a[maxn], n;

    node *newnode(int L, int R, int sz, node *pL, node *pR) {
        pooltop->init(L, R, sz, pL, pR); return pooltop++;
    }
private:
    node *update(node *p, int L, int R, int v, int flag /* 1 or -1*/ ) {
        if (L == R) {
            return newnode(L, R, p->sz + flag, null, null);
        } else {
            node *pL = p->pL, *pR = p->pR; int M = L + (R - L) / 2;
            if ( v <= M ) {
```

```
                pL = update(pL, L, M, v, flag);
            } else {
                pR = update(pR, M+1, R, v, flag);
            }
            return newnode(L, R, pL->sz + pR->sz, pL, pR);
        }
    }

    node *update(node *p, int v, int flag /* 1 or -1 */) {
        return update(p, -inf, inf, v, flag);
    }

    int query(node *p[], node *q[], int L, int R, int k) {
        if (L == R) {
            return L;
        } else {
            int sz = 0, M = L + (R - L) / 2;
            for (int i = 0; p[i]; ++i) sz += p[i]->pL->sz;
            for (int j = 0; q[j]; ++j) sz -= q[j]->pL->sz;
            if ( k - sz <= 0 ) {
                for (int i = 0; p[i]; ++i) p[i] = p[i]->pL;
                for (int j = 0; q[j]; ++j) q[j] = q[j]->pL;
                return query(p, q, L, M, k);
            } else {
                for (int i = 0; p[i]; ++i) p[i] = p[i]->pR;
                for (int j = 0; q[j]; ++j) q[j] = q[j]->pR;
                return query(p, q, M+1, R, k - sz);
            }
        }
    }

    void makeTree(node *p[], int i) {
        for (; i; i -= lowbit(i)) *p++ = tree[i]; *p = NULL;
    }
public:
    void init(int a[], int n) {
        this->n = n; pooltop = pool;
        null = newnode(-inf, inf, 0, pooltop, pooltop);
        for (int i = 0; i <= n; ++i) tree[i] = null;
        for (int i = 1; i <= n; ++i) {
            this->a[i] = a[i];
            if ( lowbit(i) == 1 ) {
                tree[i] = update( null, a[i], 1 );
            } else {
```

```
                int j = i - lowbit(i) / 2; tree[i] = tree[j];
                for (int k = j + 1; k <= i; ++k)    tree[i] = update( tree[i], a[k], 1 );
            }
            // for ( int j = i; j <= n; j += lowbit(j) ) tree[j] = update( tree[j], a[i], 1 );
        }
    }

    int kth(int L, int R, int k) {
        static node *p[35], *q[35];
        makeTree(p, R); makeTree(q, L-1);
        return query(p, q, -inf, inf, k);
    }

    void modify(int i, int v) {
        if ( a[i] == v ) continue;
        for (int j = i; j <= n; j += lowbit(j))
            tree[j] = update(tree[j], a[i], -1), tree[j] = update(tree[j], v, 1);
        a[i] = v;
    }
} task;

void solve(int n, int q){
    static int a[maxn]; char cmd[5];
    for (int i = 1; i <= n; ++i) scanf("%d", &a[i]);
    task.init(a, n);
    for (int i = 0, x, y, z; i < q; ++i) {
        scanf("%s", cmd);
        if ( cmd[0] == 'C' ) {
            scanf("%d%d", &x, &y); task.modify( x, y );
        } else {
            scanf("%d%d%d", &x, &y, &z ); printf("%d\n", task.kth( x, y, z ) );
        }
    }
}

int main(){ for (int n, q; cin >> n >> q; solve(n, q)); }
```

Middle(陈立杰)

在线询问区间 L->R 的最大中位数(第(L-R+1)/2 个)，其中 L = a..b, R = c..d, a < b < < d，上次使用 2012-8-29

```
const int maxn = 20000 + 5;
```

```
class Middle {
    #define Mid(L, R) ( (L) + ( (R) - (L) ) / 2 )
private:
    struct Item {
        int v, i;
        bool operator < (const Item& o) const { return v < o.v; }
        bool operator == (const Item& o) const { return v == o.v; }
    } item[maxn];
    int val[maxn], valcnt, n;
private:
    struct Info {
        int sum, minleft, minright;
        Info(int sum, int minleft, int minright) : sum(sum), minleft(minleft), minright(minright) {
        }
        Info(){
        }
        Info(int sum) : sum(sum) {
            minleft = minright = min(sum, 0);
        }
        Info operator + (const Info& o) const {
            return Info (
                sum + o.sum,
                min( minleft, sum + o.minleft ),
                min( o.minright, minright + o.sum )
            );
        }
    };
    struct node {
        int L, R; Info info;
        node *pL, *pR;
    } pool[maxn * 500], *pooltop, *tree[maxn];

    node *newnode(int L, int R, const Info& info, node *pL, node *pR) {
        pooltop->L = L; pooltop->R = R; pooltop->info = info;
        pooltop->pL = pL; pooltop->pR = pR; return pooltop++;
    }

    Info query(node *p, int L, int R) {
        if (p->L == L && p->R == R) {
            return p->info;
        } else {
            int MM = Mid(p->L, p->R);
            if ( R <= MM ) {
                return query(p->pL, L, R);
```

```
            } else if ( L >= MM + 1 ) {
                return query(p->pR, L, R);
            } else {
                return query(p->pL, L, MM) + query(p->pR, MM+1, R);
            }
        }
    }
private:
    node *build(int L, int R) {
        if (L == R) {
            return newnode(L, R, Info(-1), NULL, NULL );
        } else {
            int M = Mid(L, R); node *pL = build(L, M), *pR = build(M+1, R);
            return newnode(L, R, pL->info + pR->info, pL, pR );
        }
    }

    node *update(node *p, int i) {
        if ( p->L == p->R ) {
            return newnode(p->L, p->R, Info(1), NULL, NULL );
        } else {
            int M = Mid(p->L, p->R);
            node *pL = p->pL, *pR = p->pR;
            if (i <= M) {
                pL = update(pL, i);
            } else {
                pR = update(pR, i);
            }
            return newnode(p->L, p->R, pL->info + pR->info, pL, pR);
        }
    }

    int calc(node *p, int a, int b, int c, int d) {
        return query(p, a, b - 1).minright + query(p, b, c).sum + query(p, c + 1, d).minleft;
    }
public:
    void init(int a[], int n){
        pooltop = pool; this->n = n;

        for (int i = 0; i < n; ++i) item[i].v = a[i], item[i].i = i;
        sort(item, item + n);

        valcnt = 0;
        for (int i = 0, j; i < n; i = j, ++valcnt){
```

```
            for (j = i + 1; j < n && item[i] == item[j]; ++j)
                ;
            tree[valcnt] = valcnt ? tree[valcnt-1] : build(0, n - 1);
            for (int k = i; k < j; ++k)
                tree[valcnt] = update( tree[valcnt], item[k].i );
            val[valcnt] = item[i].v;
        }
    }
    int query(int a, int b, int c, int d) {
        int low = 0, high = valcnt - 1, ans;
        while (low <= high) {
            int mid = (low + high) / 2;
            node *p = tree[mid];
            if ( calc(tree[mid], a, b, c, d) > 0    ) {
                ans = mid; high = mid - 1;
            } else {
                low = mid + 1;
            }
        }
        return val[ans];
    }
} task;

void solve(int n) {
    static int a[maxn]; for (int i = 0; i < n; ++i) scanf("%d", &a[i]);
    task.init(a, n);
    int q; cin >> q;
    for (int ans = 0; q--; ) {
        int query[4];
        for (int k = 0; k < 4; ++k)
            scanf("%d", &query[k]), query[k] = (query[k] + ans) % n;
        sort(query, query + 4);
        ans = task.query(query[0], query[1], query[2], query[3]); printf("%d\n", ans);
    }
}

int main(){ for (int n; cin >> n; solve(n) ); }
```

Dyanmic len(set(a[L:R]))

对于一个序列，有两种操作

M x y // a[x] = y

Q x y // print len(set(a[x:y]))

上次使用 **2012.8.31**

```
class ChairTree {
private:
    struct node {
        int sz; node *pL, *pR;
    } *tree[maxn], pool[maxn * 500], *pooltop;
    int n;
private:
    node *newnode(node *pL, node *pR, int sz) {
        pooltop->pL = pL; pooltop->pR = pR; pooltop->sz = sz; return pooltop++;
    }
    node *build(int L, int R) {
        if (L == R) {
            return newnode(NULL, NULL, 0);
        } else {
            int M = (L + R) / 2;
            node *pL = build(L, M), *pR = build(M+1, R);
            return newnode(pL, pR, pL->sz + pR->sz);
        }
    }
    node *update(node *p, int L, int R, int i, int d) {
        if (L == R) {
            return newnode(NULL, NULL, p->sz + d);
        } else {
            int M = (L + R) / 2;
            node *pL = p->pL, *pR = p->pR;
            if ( i <= M ) {
                pL = update(pL, L, M, i, d);
            } else {
                pR = update(pR, M+1, R, i, d);
            }
            return newnode(pL, pR, pL->sz + pR->sz);
        }
    }
    int query(node *p, int LL, int RR, int L, int R) {
        if (LL == L && RR == R) {
            return p->sz;
        } else {
            int MM = (LL + RR) / 2;
            if (R <= MM) {
                return query(p->pL, LL, MM, L, R);
            } else if (L >= MM+1) {
                return query(p->pR, MM+1, RR, L, R);
            } else {
```

```
                return query(p->pL, LL, MM, L, MM) + query(p->pR, MM+1, RR, MM+1, R);
            }
        }
    }
public:
    void init(int a[], int n) {
        this->n = n; pooltop = pool;
        tree[0] = build(1, n);
        for (int i = 1; i <= n + 7; ++i)
            tree[i] = tree[i-1];
        for (int i = 1; i <= n; ++i)
            insert(i, a[i]);
    }
    int query(int L, int R) {
        if (L > R) return 0;
        int ans = 0;
        for (int k = L - 1 + 5; k > 0; k -= lowbit(k)) {
            ans += query(tree[k], 1, n, L, R);
        }
        return ans;
    }
    void erase(int i, int v) {
        for (int j = v + 5; j <= n + 7; j += lowbit(j)) {
            tree[j] = update(tree[j], 1, n, i, -1);
        }
    }
    void insert(int i, int v) {
        for (int j = v + 5; j <= n + 7; j += lowbit(j)) {
            tree[j] = update(tree[j], 1, n, i, 1);
        }
    }
} task;

void solve(int n, int q) {
    static int a[maxn];
    for (int i = 1; i <= n; ++i)
        scanf("%d", &a[i]);
    map< int, set<int> > mm;
    static int prev[maxn];
    for (int i = 1; i <= n; ++i) {
        set<int> &s = mm[ a[i] ];
        prev[i] = (s.size() ? *s.rbegin() : 0);
        s.insert(i);
    }
```

```
task.init(prev, n);

char cmd[5];
for (int i = 1; i <= q; ++i) {
	int x, y;
	scanf("%s%d%d", cmd, &x, &y);
	if ( cmd[0] == 'Q' ) {
		++x;
		int ans = task.query( x, y );
		printf("%d\n", ans);
	} else {
		++x;
		if (a[x] == y) continue;
		set<int> &s1 = mm[ a[x] ], &s2 = mm[ y ];
		set<int>::iterator iter;
		a[x] = y;

		//s1
		iter = s1.upper_bound(x);
		if ( iter != s1.end() ) {
			task.erase(*iter, prev[*iter]);
			prev[*iter] = prev[x];
			task.insert(*iter, prev[*iter]);
		}
		s1.erase(x);

		//s2
		iter = s2.upper_bound(x);
		if ( iter != s2.end() ) {
			task.erase(x, prev[x]);
			prev[x] = prev[*iter];
			task.insert(x, prev[x]);

			task.erase(*iter, prev[*iter]);
			prev[*iter] = x;
			task.insert(*iter, prev[*iter]);
		} else {
			task.erase(x, prev[x]);
			prev[x] = s2.empty() ? 0 : *--s2.end();
			task.insert(x, prev[x]);
		}
		s2.insert(x);
	}
}
```

```
}

int main() {for (int n, q; cin >> n >> q; solve(n, q)); }
```

[涉及标记下推]平面上的 n 个点，每次把第 L..R 号点进行操作(平移、旋转...)，问第 i 号点的位置

```
const int maxn = 50000 + 5;
const cpl one(1, 0), zero(0, 0);

const struct Info {
	cpl a, b;
	Info() {
	}
	Info(cpl a, cpl b) : a(a), b(b) {
	}
	Info operator * (const Info& o) const {
		return Info( a * o.a, b * o.a + o.b );
	}
	bool operator != (const Info& o) const {
		return a != o.a || b != o.b;
	}
} E(one, zero);

struct node *newnode(const Info &info, node *pL, node *pR );

struct node {
	Info info; node *pL, *pR;
	node *init(const Info& info, node *pL, node *pR) {
		this->info = info; this->pL = pL; this->pR = pR; return this;
	}
	node *update(const Info& o) const {
		return newnode( info * o, pL, pR );
	}
} pool[maxn * 20], *pooltop = 0, *poolbound = pool + sizeof(pool) / sizeof(pool[0]);

node *newnode(const Info &info, node * pL, node * pR ){
	return ( pooltop == poolbound? new node : pooltop++ )->init( info, pL, pR );
}

struct SegmentTree {
public:
```

```
    void init(const cpl *v, int L, int R) {
        pooltop = pool; this->v = v; this->L = L; this->R = R;
        entry[cur = 0] = build(L, R);
    }
    void move(int left, int right, double x, double y) {
        entry[cur+1] = update( entry[cur], left, right, L, R, Info( one, cpl(x, y) ) );
        ++cur;
    }
    void rotate(int left, int right, double rot) {
        entry[cur+1] = update( entry[cur], left, right, L, R, Info( polar(1.0, rot) , zero ) );
        ++cur;
    }
    void setzero(int left, int right) {
        entry[cur+1] = update( entry[cur], left, right, L, R, Info( zero, zero ) );
        ++cur;
    }
    void cancel(int i) {
        cur -= i;
    }
    void redo(int i) {
        cur += i;
    }
    cpl ask(int i) {
        return query( entry[cur], i, L, R).b;
    }
private:
    node *entry[maxn]; int cur;
    int L, R; const cpl *v;
    #define M (L + R >> 1)
    node *build(int L, int R){
        if (L == R) {
            return newnode( Info( zero, v[L] ), NULL, NULL );
        } else {
            return newnode( E, build(L, M), build(M+1, R) );
        }
    }

    node *update(node *p, int left, int right, int L, int R, const Info& info ) {
        node *pL = p->pL, *pR = p->pR;
        if ( left == L && right == R ) {
            return newnode( p->info * info, pL, pR );
        } else {
            if ( p->info != E ) {
                pL = pL->update( p->info );
```

```
                pR = pR->update( p->info );
            }
            if ( right <= M ) {
                pL = update(pL, left, right, L, M, info );
            } else if ( left >= M + 1 ) {
                pR = update(pR, left, right, M+1, R, info );
            } else {
                pL = update(pL, left, M, L, M, info);
                pR = update(pR, M+1, right, M+1, R, info);
            }
            return newnode( E, pL, pR );
        }
    }

    Info query(const node *p, int i, int L, int R) {
        if (L == R) {
            return p->info;
        } else {
            return ( i <= M ? query(p->pL, i, L, M) : query(p->pR, i, M+1, R) ) * p->info;
        }
    }
} tree;

void solve(int n){
    static cpl v[maxn];
    for (int i = 1; i <= n; ++i) {
        double x, y; scanf("%lf%lf", &x, &y); v[i].real() = x; v[i].imag() = y;
    }
    tree.init(v, 1, n);
    int q; cin >> q;
    char cmd[15]; int L, R; double a, b;
    for (q--) {
        scanf("%s", cmd);
        if ( cmd[0] == 'M' ) {
            scanf("%d%d%lf%lf", &L, &R, &a, &b); tree.move(L, R, a, b);
        } else if ( cmd[0] == 'P' ) {
            scanf("%d%d%lf", &L, &R, &a); tree.rotate(L, R, a);
        } else if ( cmd[0] == 'L' ) {
            scanf("%d%d", &L, &R); tree.setzero(L, R);
        }else if ( cmd[0] == 'C' ) {
            scanf("%d", &L); tree.cancel(L);
        } else if ( cmd[0] == 'R' ) {
            scanf("%d", &L); tree.redo(L);
        } else if ( cmd[0] == 'A' ){
```

```
            scanf("%d", &L); cpl c = tree.ask(L); printf("%f %f\n", c.real(), c.imag() );
        }
    }
}

int main(){ for (int n; cin >> n; solve(n)); }
```

treap

```
struct Treap {
    struct node {
        const int key, weight;
        const node *left, *right;
        node(int key, int weight, const node *left, const node *right)
            : key(key), weight(weight), left(left), right(right) {
        };
    };

    node *newnode(int key) {
        return new node( key, rand(), 0, 0 );
    }

    const node *insert(const node *a, int x) {
        return merge( merge( split_left(a, x), newnode(x) ), split_right(a, x) );
    }

    const node *remove(const node *a, int x) {
        return merge( split_left(a, x), split_right(a, x + 1) );
    }

    const node *merge(const node *a, const node *b) {
        return (!a || !b) ? (a ? a : b) :
            (a->weight < b->weight) ?
                new node(a->key, a->weight, a->left, merge(a->right, b)) :
                new node(b->key, b->weight, merge(a, b->left), b->right);
    }

    const node *split_left(const node *a, int key) {
        return !a ? 0 : a->key < key ?
            new node(a->key, a->weight, a->left, split_left(a->right, key) ) :
            split_left(a->right, key);
    }

    const node *split_right(const node *a, int key) {
```

```
        return !a ? 0 : a->key > key ?
            new node(a->key, a->weight, split_right(a->left, key), a->right ) :
            split_right(a->left, key);
    }
};
```

去掉表达式中冗余的括号(允许使用结合律)

```
struct StripParentheses {
    /*
    E -> F{+F}
    F -> T{*T}
    T -> (E) | terminal
    */
    struct node {
        char oper;
        node *left, *right;
    } nodes[MAX_SIZE], *free_node, *entry;
    char last, *ptr;

    void GetNextToken() {
        do last = *ptr++; while ((last == ' ') || (last == '\t'));
    }

    node *T() {
        if (last == '(') {
            GetNextToken();
            node *ret = E();
            GetNextToken(); /* preskocit ')' */
            return ret;
        }
        free_node->oper = last;
        GetNextToken(); /* preskocit terminator */
        return free_node++;
    }

    node *Fc(node *left) {
        if ( (last == '*') || (last == '/') ) {
            node *p = free_node++;
            p->oper = last; last = GetNextToken();
            p->left = left; p->right = T();
            return Fc(p);
        }
        return left;
```

```
    }

    node *F(void) { return Fc( T() ); }

    node *Ec(node *left) {
        if ((last == '+') || (last == '-')) {
            node *p = free_node++;
            p->oper = last; last = GetNextToken();
            p->left = left; p->right = F();
            return Ec(p);
        }
        return left;
    }

    node *E() { return Ec( F() ); }

    void print(node *node, int prior) {
        int lpri, rpri; // 0:null 1:*/ 2:+-*/
        switch(node->oper) {
            case '+': lpri = rpri = 2; break;
            case '-': lpri = 2; rpri = 1; break;
            case '*': lpri = rpri = 1; break;
            case '/': lpri = 1; rpri = 0; break;
            default: putchar(node->oper); return;
        }
        if (lpri > prior) putchar('(');
        print(node->left, lpri);
        putchar(node->oper);
        print(node->right, rpri);
        if (lpri > prior) putchar(')');
    }
public:
    void build(char s[]){
        ptr = s; GetNextToken();
        free_node = nodes;
        entry = E();
    }
    void print(){ print(entry, 4); puts(""); }
} solver;

int main() {
    int n; scanf("%d", &n);
    while (n--) {
        static char s[1000000]; scanf("%s", s);
```

```
            solver.build(s);solver.print();
        }
}
```

表达式求值 – java – by lqt

```
import java.util.*;
import javax.script.*;

public class Main {
    public static void main(String[] args) throws ScriptException{
        Scanner cin = new Scanner(System.in);
        ScriptEngine engine = new ScriptEngineManager().getEngineByName("JavaScript");
        int t = cin.nextInt(); cin.nextLine();
        Random rand = new Random();
        for (int i = 0; i < t; i++){
            String s01 = cin.nextLine(), s02 = cin.nextLine();
            boolean equal = true;
            for (int k = 0; equal && k < 10; k++){
                String s1 = s01, s2 = s02;
                for (int j = 0; j < 26; j++){
                    int x = rand.nextInt(100);
                    s1 = s1.replace(String.valueOf((char)(j+'a')), String.valueOf(x));
                    s2 = s2.replace(String.valueOf((char)(j+'a')), String.valueOf(x));
                }
                for (int j = 0; j < 26; j++){
                    int x = rand.nextInt(100);
                    s1 = s1.replace(String.valueOf((char)(j+'A')), String.valueOf(x));
                    s2 = s2.replace(String.valueOf((char)(j+'A')), String.valueOf(x));
                }
                s1 = engine.eval(s1).toString();
                s2 = engine.eval(s2).toString();
                if (!s1.equals(s2)) equal = false;
            }
            System.out.println(equal ? "YES" : "NO");
        }
    }
}
```

算法

贪心法

流水作业调度问题 – Jonhson 算法

有 n 个作业要在 A，B 组成的流水线上完成加工，每个作业都必须先花 ai 时间在 A 上加工，然后话 bi 时间在 B 上加工。确定 n 个作业的加工顺序，使得完工时间最早。

Jonhson 算法

设 P 为 a < b 的作业序列，Q 为 a >= b 的作业序列，将 P 中作业按照 a 递增(不减)排序，Q 中元素按照 b 递减(不增)排序，PQ 即为一个最优顺序。

注：三个和三个以上的流水线是 NP-hand 的。

不相交的区间选择问题

描述：数轴上有 n 个开区间(a_i, b_i)，选择尽量多个区间，使得这些区间两两没有公共点。

解法：区间图的最大独立集。按照右端点排序，从左到右能选就选

将 n 个正整数联接成一排，组成一个最大(最小)的多位整数

贪心，将所有整数看成字符串，然后排序即可。排序策略为 *a < b iff. a##b < b##a*

去掉区间包含[最好再找一人看过]

```
#define x first
#define y second

int eraseContainSmall(pair<int, int> p[], int n) { // 去掉小区间，保留大区间
	sort(p, p + n);
	int c = 1;
	for (int i = 1; i < n; ++i) {
		if ( p[i].second > p[c-1].second ) {
			if (p[i].first == p[c-1].first) p[c-1] = p[i];
			else p[c++] = p[i];
		}
	}
	return c;
}
```

```
int eraseContainBig(pair<int, int> p[], int n) { // 去掉大区间，保留小区间
	sort(p, p + n);
	int c = 0;
	for (int i = 0; i < n; ++i) {
		while (c && p[c-1].y >= p[i].y) --c;
		p[c++] = p[i];
	}
	return c;
}
```

动态规划

数位 dp

数位 dp 一定要写暴力 check 程序

cf 某题

```
int digit[21], ind[module + 5];
LL dp[21][module][48];

void init(){
	for (int i = 1, k = 0; i <= module; ++i) if ( module % i == 0 ) ind[i] = k++;
	memset(dp, -1, sizeof dp);
}

LL dfs(int pos, int preSum, int preLcm, bool tight) {
	if ( pos == -1 ) return preSum % preLcm == 0;
	LL &ref = dp[pos][preSum][ ind[preLcm] ];
	if ( !tight && ref != -1 ) return ref;
	LL ans = 0;
	int up = tight ? digit[pos] : 9;
	for (int i = 0; i <= up; ++i) {
		int curSum = (preSum * 10 + i) % module, curLcm = i == 0 ? preLcm : lcm(preLcm, i);
		//use i rather than digit[pos] !!!!!!
		ans += dfs(pos-1, curSum, curLcm, tight && i == up);
	}
	if ( !tight ) ref = ans;
	return ans;
}

LL calc(LL x) {
```

```
    int pos = 0; for(;x;x /= 10) digit[pos++] = x % 10;
    return dfs(pos-1, 0, 1, true);
}
```

SPOJ KPSUM

```
struct DigitDP {
    int digit[105]; LL dp[25][25][250];
    DigitDP(){
        memset(dp, -1, sizeof dp);
    }
    LL dfs(int pos, int len, int sum, bool tight, bool leading) {
        LL &ref = dp[pos][len][sum];
        if (!tight && !leading && ref != -1) return ref;
        LL ans = 0;
        int up = tight ? digit[pos] : 9;
        for (int i = leading ? 1 : 0; i <= up; ++i) {
            int nsum = sum + ( (len - pos) % 2 == 1 ? -1 : 1 ) * i;
            if (pos > 0) {
                ans += dfs(pos-1, len, nsum, tight && i == up, false);
            } else { //pos == 0
                if (len % 2 == 0) {
                    ans += nsum;
                } else {
                    ans += (i % 2 ? -1 : 1) * nsum;
                }
            }
        }
        if ( leading && pos > 0 ) ans += dfs(pos-1, len-1, sum, false, leading); //可能需改为 >=0
        if (!tight && !leading) ref = ans;
        return ans;
    }
    LL check(LL x) {
        vector<int> v; char buf[100];
        for (int i = 0; i <= x; ++i) {
            int len = sprintf(buf, "%d", i);
            for (int j = 0; j < len; ++j) v.push_back( buf[j] - '0' );
        }
        LL ans = 0;
        for (int i = 0; i < v.size(); ++i) ans += (i % 2 ? 1 : -1) * v[i];
        return ans;
    }
    LL calc(LL x) {
```

```
        if (x == 0) return 0;
        int pos = 0;
        for (LL i = x; i; i /= 10) digit[pos++] = i % 10;
        return dfs(pos-1, pos, 0, true, true);
    }
    void debug(){
        for (LL x; cin >> x; ) {
            LL a = check(x), b = calc(x);
            cout << a << endl << b << endl << (a == b ? "OK" : "ERROR!") << endl;
        }
    }
} solve;
```

最长公共子序列 LCS

设 A[1..i]和 B[1..j]的最长公共子序列是 C[1..k],长度为 k=f(i,j)，则：
(一)若 A[i] == B[j]，则：
(1)A[i] == C[k]，且 C[1..k-1]是 A[1..i-1]和 B[1..j-1]的最长公共子串(f(i-1, j-1) = k-1)。(反证)
(2) f(i, j) = f(i-1, j-1)+1
(二)若 A[i] != B[j]，有如下结论：
(1)若 A[i] != C[k]，则 f(i, j) = f(i-1, j)
(2)若 B[i] != C[k]，则 f(i, j) = f(i, j-1)
(3)以上两条(1)(2)的条件至少有一条会被满足,即 A[i] != C[k]和 B[j] != C[k]不可能都成立
(4)f(i, j) = max{f(i-1, j), f(i, j-1)}
(三)综上：设 f(i,j)是 A[1..i]和 B[1..j]的最长公共子序列，则：
A[i] == B[j] 时，f(i, j) = f(i-1, j-1)+1
A[i] != B[j] 时，f(i, j) = max{f(i-1, j), f(i, j-1)}
其中 f(0..i, 0) = f(0, 0..j) = 0

最长公共子串 $O(n^2)$ – dp, O(n) – suffixArray

状态函数：
dp[i][j]表示 A[1..i]的后缀与 B[1..j]的后缀的最长公共子串
状态转移方程：
if(A[i] == B[j]) dp[i][j] = dp[i-1][j-1] + 1;else dp[i][j] = 0;
目标函数：
ans = max{dp[1..La][1..Lb]};

最长上升子序列长度 LIS – O(nlogn)

//已经通过 cugb1009, cugb1032, cugb1049

```
int lis(int a[], int n, int d[]){ //d[i]: 以 a[i]结尾的最长上升子序列的长度
```

```
    static int g[maxn];
    fill(g, g + n, inf);
    for (int i = 0; i < n; i++){
        int j = lower_bound(g, g + n, a[i]) - g;
        d[i] = j+1; g[j] = a[i];
    }
    return *max_element(d, d + n);
}

int lis2(int a[], int n, int r[]){ //已经通过 hdu1160
    static int g[maxn], d[maxn], path[maxn], L[maxn];
    fill(g, g + n, inf); fill(path, path + n, -1); fill(L, L + n, -1);
    for (int i = 0; i < n; i++){
        int j = lower_bound(g, g + n, a[i]) - g;
        d[i] = j + 1; L[j+1] = i; g[j] = a[i]; path[i] = L[j];
    }
    int k = max_element(d, d + n) - d, cnt = 0;
    for (;k != -1; k = path[k]) r[cnt++]= k;
    reverse(r, r + cnt);
    return cnt;
}
```

对于最长下降(不升)子序列，只需将 inf 改成-inf, 并将比较函数改为 greater 即可
对于最长不降(不升)子序列，只需将 lower_bound 改为 upper_bound
结论：**上升**子序列的最小个数=最长**不升**子序列的长度

字典序最小的递增子序列

题意：给你一个序列，求长度为 L 的字典序最小的递增子序列
题解：倒着做一个最长递减子序列的 dp，求出 d[i]:i..n 的序列，以 i 开头的能到达的最长的递增子序列的长度;然后从左向右扫一下，能取就取就行了(这样保证了字典序最小)。

二维最长单调子序列 – O(nlognlogn)

dp[i] = max{dp[k], k < i, x[k] < x[i], y[k] < y[i]};

```
const int maxn = 100000 + 5;
const int inf = 2000000000;
struct Point { int x, y; } p[maxn];
bool operator < (const Point &a, const Point &b){ return (a.x != b.x) ? a.x < b.x : a.y > b.y; }
Point make_point(int x, int y){ Point p; p.x = x, p.y = y; return p;}

int lis2(Point p[], int n, int dp[]){
    static set<Point> S[maxn];
    typedef set<Point>::iterator siterator;
```

```
    int res = 0;
    for (int i = 0; i < n; i++){
        int L = 1, H = res, R = 1;
        while (L <= H){
            int M = (L + H) / 2, ok = 1;
            siterator iter = S[M].lower_bound(make_point(p[i].x - 1, -inf));
            --iter;
            if (p[i].y > iter->y) R = L = M + 1; else H = M - 1;
        }
        dp[i] = R;
        if (R > res){
            res = R;
            S[R].clear(); S[R].insert(make_point(-inf, inf)); S[R].insert(make_point(inf, -inf));
        }
        siterator iter;
        Point pt = make_point(p[i].x + 1, inf);
        while (iter = S[R].lower_bound(pt), p[i].y < iter->y) S[R].erase(iter);
        S[R].insert(p[i]);
    } /* end loop i*/
    return res;
}
```

最长公共上升子序列 – $O(n^2)$

```
int pathx[maxn][maxn], pathy[maxn][maxn];

int LCIS_insert(int L[], int Lx[], int Ly[], int a, int p, int i, int j){
    int x = p;
    while (L[x] < a) x++;
    L[x] = a; Lx[x] = i; Ly[x] = j;
    if (x != 1)
        pathx[i][j] = Lx[x-1], pathy[i][j] = Ly[x-1];
    return x;
}

int LCIS(int a[], int b[], int m, int n, int r[]){
    //数组下标从 1 开始，已经通过 hdu1423, zoj2432(需要将 L[]改成 long long 类型)
    static int L[maxn][maxn], Lx[maxn][maxn], Ly[maxn][maxn];
    //initialization
    if (m < n)
        swap(m, n), swap(a, b);
    for (int j = 1; j <= n; j++)
        for (int k = 1; k <= n; k++)
```

```
            L[j][k] = inf;
    for (int i = 1; i <= m; i++)
        for (int j = 1; j <= n; j++)
            pathx[i][j] = pathy[i][j] = -1;
    //main program
    for (int i = 1; i <= m; i++){
        int x = -1, p = 1;
        for (int j = 1; j <= n; j++)
            if (a[i] == b[j]) //the match case
                x = p = LCIS_insert(L[j], Lx[j], Ly[j], a[i], p, i, j);
            else            //the mismatch case
                if (x != -1 && L[j-1][x] < L[j][x])
                    L[j][x] = L[j-1][x], Lx[j][x] = Lx[j-1][x], Ly[j][x] = Ly[j-1][x];
                else
                    x = -1;
    }
    //recover a longest common increasing subsequencein reverse order
    int ret = lower_bound(L[n] + 1, L[n] + n + 1, inf) - (L[n] + 1);
    if (ret == 0) return 0;
    int x = Lx[n][ret], y = Ly[n][ret], tx, ty, cnt = 0;
    r[cnt++] = a[x];
    while (tx = pathx[x][y], ty = pathy[x][y], tx != -1 && ty != -1)
        x = tx, y = ty, r[cnt++] = a[x];
    reverse(r, r + cnt);
    return ret;
}
```

最长公共不降子序列 – $O(n^2)$

```
int pathx[maxn][maxn], pathy[maxn][maxn];

int LCIS_insert(int L[], int Lx[], int Ly[], int a, int p, int i, int j){
    int x = p;
    while (L[x] <= a) x++; //注意不降的时候有等号
    L[x] = a; Lx[x] = i; Ly[x] = j;
    if (x != 1)
        pathx[i][j] = Lx[x-1], pathy[i][j] = Ly[x-1];
    return x;
}

int LCIS(int a[], int b[], int m, int n, int r[]){
    //数组下标从 1 开始，已经通过 bianchengla2041
    static int L[maxn][maxn], Lx[maxn][maxn], Ly[maxn][maxn], lamda[maxn];
    //initialization
```

```
    if (m < n) { swap(m, n);swap(a, b);}
    for (int j = 1; j <= n; j++)
        for (int k = 1; k <= n; k++)
            L[j][k] = inf;
    for (int i = 1; i <= m; i++)
        for (int j = 1; j <= n; j++)
            pathx[i][j] = pathy[i][j] = -1;
    //main program
    for (int j = 1; j <= n; j++)
        lamda[j] = -1;
    for (int i = 1; i <= m; i++){
        int x = -1, p = 1;
        for (int j = 1; j <= n; j++){
            if (a[i] == b[j]){ //the match case
                if (lamda[j] == -1){
                    p = LCIS_insert(L[j], Lx[j], Ly[j], a[i], p, i, j);
                    lamda[j] = (x == -1 ? p : -1); x = p;
                } else { //lamda[j] != -1
                    lamda[j] = (x == -1 ? lamda[j] : -1); x = -1;
                }
            } else {    //the mismatch case
                if (x != -1 && L[j-1][x] < L[j][x]) {
                    L[j][x] = L[j-1][x]; Lx[j][x] = Lx[j-1][x]; Ly[j][x] = Ly[j-1][x];
                    lamda[j] = (lamda[j] == x ? -1 : lamda[j]);
                }
                else{
                    lamda[j] = (x == -1 ? lamda[j] : -1); x = -1;
                }
            }
        }
    }
    //recover a longest common increasing subsequence in reverse order
    int ret = lower_bound(L[n] + 1, L[n] + n + 1, inf) - (L[n] + 1);
    if (ret == 0) return 0;
    int x = Lx[n][ret], y = Ly[n][ret], tx, ty, cnt = 0;
    r[cnt++] = a[x];
    while (tx = pathx[x][y], ty = pathy[x][y], tx != -1 && ty != -1)
        x = tx, y = ty, r[cnt++] = a[x];
    reverse(r, r + cnt);
    return ret;
}
```

01 矩阵最大正方形空地 – $O(n^2)$

状态函数：
dp[i][j]表示右下角是(i,j)的最大正方形边长。
状态转移方程：
若(i,j)可行：dp[i][j] = min(dp[i-1][j], dp[i-1][j-1], dp[i][j-1]) + 1; 否则 dp[i][j] = 0
目标函数：
Ans = max{dp[1..m][1..n]};

01 矩形最大矩形空地 – $O(n^2)$

上次使用 2012.7.28

```
const int maxn = 1000;
int v[maxn][maxn];

int main(){
    int re; scanf("%d", &re);
    while (re--){
        int r, c; scanf("%d%d", &r, &c);
        for ( int i = 1; i <= r; i++ ){
            for (int j = 1; j <= c; j++)
                scanf("%1d", &v[i][j]);
            v[i][0] = v[i][c+1] = -1;
        }
        static int h[maxn];
        for (int j = 1; j <= c; j++)
            v[0][j] = -1, h[j] = 0;
        int res = 0;
        for ( int i = 1; i <= r; i++ ){
            for (int j = 1; j <= c; j++)
                h[j] = (v[i][j] == v[i-1][j] ? h[j] + 1 : 1);
            int left[maxn], right[maxn];
            for (int j = 1; j <= c; j++){
                left[j] = j;
                while (v[i][left[j]-1]==v[i][j] && h[left[j]-1]>=h[j] )
                    left[j] = left[ left[j]-1 ];
            }
            for (int j = c; j >= 1; j--){
                right[j] = j;
                while(v[i][right[j]+1]==v[i][j]&&h[right[j]+1]>=h[j])
                    right[j] = right[ right[j]+1 ];
            }
            for (int j = 1; j <= c; j++){
```

```
                    //TODO change here
                    checkmax(res, v[i][j] * (right[j] - left[j] + 1) * h[j] );
                }
            }
            cout << res << endl;
        }
}
```

最长子段和 O(n) – 略

长度 n 序列，最大 m 不相交子段和(nlogn)

```
typedef long long LL;
const LL inf = 1000000000000000LL;
const int maxn = 1000000 + 5;

LL absll(LL x) { return x >= 0 ? x : -x; }

template<class Cmp = less<int> > struct Heap {
    int hash[maxn], v[maxn], n; Cmp cmp;

    void init(){
        n = 0;
    }

    void up(int p){
        int a = v[p];
        for (int q = (p - 1) >> 1; q >= 0 && cmp(v[q], a); p = q, q = (p - 1) >> 1 ) {
            v[p] = v[q]; hash[ v[p] ] = p;
        }
        v[p] = a; hash[ v[p] ] = p;
    }

    void down(int p){
        int a = v[p];
        for (int q = p << 1 | 1; q < n; p = q, q = p << 1 | 1) {
            if ( q + 1 < n && cmp(v[q], v[q+1]) ) ++q;
            if ( cmp(v[q], a) ) break;
            v[p] = v[q]; hash[ v[p] ] = p;
        }
        v[p] = a; hash[ v[p] ] = p;
    }
```

```
    void push(int a){
        v[n++] = a; up(n-1);
    }

    int top() const {
        return v[0];
    }
/*
    void pop(){
        swap(v[0], v[--n]); down(0);
    }*/
};

LL a[maxn];

struct Cmp {
    bool operator()(int x, int y) const {
        return absll(a[x]) > abs(a[y]);
    }
};

Heap< Cmp > hp;
int v[maxn];

void solve(int m, int n) {
    int positive_cnt = 0, positive_interval_cnt = 0; LL r = 0;
    for (int i = 1; i <= n; ++i) {
        scanf("%d", &v[i]);
        if ( v[i] > 0 ) positive_cnt++, r += v[i];
    }
    if ( m >= positive_cnt ) {
        m -= positive_cnt; sort(v + 1, v + 1 + n);
        int i = n; while (v[i] > 0) --i;
        while (m--) r += v[i--]; //若最多 m 段，则删去本行
    } else {
        int k = 0; LL sum = 0;
        for (int i = 1; i <= n; ++i) {
            sum += v[i];
            if ( i == n || ((v[i] > 0) ^ (v[i+1] > 0)) ) {
                if (sum > 0) positive_interval_cnt++;
                if ( k > 0 || sum > 0 ) a[++k] = sum;
                sum = 0;
            }
        }
```

```
        if ( a[k] <= 0 ) --k;
        if ( m < positive_interval_cnt ) {
            static int prev[maxn], next[maxn];
            hp.init();
            for (int i = 1; i <= k; ++i)
                prev[i] = i - 1, next[i] = i + 1, hp.push(i);
            prev[1] = next[k] = -1;
            for (m = positive_interval_cnt - m; m--; ) {
                int v = hp.top(); r -= absll( a[v] );
                int pv = prev[v], nv = next[v];
                if ( pv != -1 && nv != -1 ) {
                    a[ pv ] += a[ v ] + a[ nv ];
                    hp.down( hp.hash[pv] ); hp.up( hp.hash[pv] );
                    a[ v ] = inf; hp.down( hp.hash[ v ] );
                    a[ nv ] = inf; hp.down( hp.hash[ nv ] );
                    int &nnv = next[nv];
                    next[pv] = nnv; if ( nnv != -1 ) prev[nnv] = pv;
                } else if ( pv == -1 ){
                    prev[ next[nv] ] = -1;
                    a[v] = inf; hp.down( hp.hash[v] );
                    a[nv] = inf; hp.down( hp.hash[nv] );
                } else if ( nv == -1 ) {
                    next[ prev[pv] ] = -1;
                    a[pv] = inf; hp.down( hp.hash[pv] );
                    a[v] = inf; hp.down( hp.hash[v] );
                }
            }
        }
    }
    cout << r << endl;
}

int main(){ for (int m, n; cin >> m >> n; solve(m, n) ); } //hdu1024
```

最大子阵和 – O(n^3)

枚举起始终止位置，压成一维序列转化为最大子序列和问题

最大子长方体 – O(n^5)

类似最大子阵和，枚举两维起止位置，压成一维序列。

石子合并 - $O(n^2)$

```
//实际效果很好，可以解决 n=50000 的石子合并，已通过 poj1738
int q[maxn], *qtop, res;

void combine(int *iter){
	int x = iter[0] + iter[-1], *p;
	res += x; qtop--;
	for (int *p = iter; p < qtop; p++) p[0] = p[1];
	for (p = iter-2; p[0] < x; p--) p[1] = p[0];
	p[1] = x;
	while (p > q && p[-1] <= x) {
		int d = qtop - p; combine(p); p = qtop - d;
	}
}

int stonecombine(int v[], int n){
	res = 0; qtop = q;
	*qtop++ = inf; *qtop++ = v[0];
	for (int i = 1; i < n; i++){
		while ( qtop[-2] <= v[i] ) combine(qtop-1);
		*qtop++ = v[i];
	}
	while (qtop > q+2) combine(qtop-1);
	return res;
}
```

二维方格取数

```
const int maxn = 300;
int cnt[maxn][maxn];
typedef pair<int, int> PII;
PII dp[400+5][200+5][200+5]; // <sum,diff>
const PII zero = make_pair(0, 0), neg_inf = make_pair(-1000000000, -1000000000);

void update(PII &dp, int v1, int v2){
	dp.first += v1 + v2;
	dp.second += v1 - v2;
}

int main(){
	int n, m, k;
	while ( scanf("%d%d%d", &m, &n, &k) == 3 ){
```

```
for (int i = 0; i < m; i++)
    for (int j = 0; j < n; j++)
        cnt[i][j] = 0;
while (k--){
    int a, b; scanf("%d%d", &a, &b); ++cnt[a-1][b-1];
}
if (m > n){
    for (int i = 0; i < m; i++)
        for (int j = 0; j < i; j++)
            swap(cnt[i][j], cnt[j][i]);
    swap(m, n);
}
dp[0][0][0] = zero;
for (int s = 1; s <= m-1; s++){
    for (int i1 = 0, j1; (j1 = s - i1), i1 <= s; i1++){
        for (int i2 = 0, j2; (j2 = s - i2), i2 <= s; i2++){
            PII &v = dp[s][i1][i2];
            v = neg_inf;
            checkmax(v, dp[s-1][i1][i2]);
            if (i1-1 >= 0) checkmax(v, dp[s-1][i1-1][i2]);
            if (i2-1 >= 0) checkmax(v, dp[s-1][i1][i2-1]);
            if (i1-1 >= 0 && i2-1 >= 0) checkmax(v, dp[s-1][i1-1][i2-1]);
            update(v, cnt[i1][j1], cnt[i2][j2]);
        }
        dp[s][i1][i1] = neg_inf;
    }
}
for (int i1 = 0, j1; i1 <= m-1 && (j1 = m-1 - i1) >= i1; i1++){
    if (i1 == j1){
        for (int i2 = 0, j2; (j2 = m-1 - i2) > j1; i2++)
            swap(dp[m-1][i1][i2], dp[m-1][j1][j2]);
    } else {
        for (int i2 = 0, j2; (j2 = m-1 - i2), i2 < m; i2++)
            swap(dp[m-1][i1][i2], dp[m-1][j1][j2]);
    }
}
for (int s = m; s < m-1+n-1; s++){
    for (int j1 = max(0, s-m+1), i1; (i1 = s - j1) >= 0 && j1 <= n-1; j1++){
        for (int j2 = max(0,s-m+1), i2; (i2 = s - j2) >= 0 && j2 <= n-1; j2++){
            PII &v = dp[s][j1][j2];
            v = neg_inf;
            checkmax(v, dp[s-1][j1][j2]);
            if (j1-1 >= 0) checkmax(v, dp[s-1][j1-1][j2]);
            if (j2-1 >= 0) checkmax(v, dp[s-1][j1][j2-1]);
```

```
                        if (j1-1 >= 0 && j2-1 >= 0) checkmax(v, dp[s-1][j1-1][j2-1]);
                        update(v, cnt[i1][j1], cnt[i2][j2]);
                    }
                    dp[s][j1][j1] = neg_inf;
                }
            }
            do {
                int s = m+n-2, i1, i2, j1, j2;
                i1 = i2 = j1 = j2 = n-1;
                PII &v = dp[s][j1][j2];
                v = zero;
                if (j1-1 >= 0) checkmax(v, dp[s-1][j1-1][j2]);
                if (j2-1 >= 0) checkmax(v, dp[s-1][j1][j2-1]);
            } while (0);
            PII r = dp[m+n-2][n-1][n-1];
            int sum = r.first;
            int a = sum - r.second >> 1;
            int b = sum + r.second >> 1;
            cout << sum << " " <<    a << " " << b << endl;
        }
}
```

多重背包-单调队列优化(TODO 有待整理)

```
/*  多重背包问题：给定 n 种物品和一个背包。第 i 种物品的价值是 Wi,其体积为 Vi，
数量是 Ki 件，背包的容量为 C。可以任意选择装入背包中的物品，求装入背包
中物品的最大总价值。*/

const int maxC = 10000; //背包的最大容量
int V[maxC], K[maxC], W[maxC];
long F[maxC]; //最大总重量 x 时的最大价值
int C, n;

/*  程序的基本思想是维护一个单调递减的队列，队列的元素为等价最大价值
由于队列的 push & pop 次数不会超过 C 次，所以可以使用数组模拟，总体上时间复杂度 O(n
* C)，空间复杂度 O(C) */

int deal(){
    int i, j, d, mj;
    long IB;
    static int A[maxC];
    static long B[maxC];//注意此数组可能出现负数
    int top, base;
    for (i = 1; i <= n; i++){
```

```
        for (d = 0; d < V[i]; d++){ //将体积按照除以第 i 种物品的价值(V[i])的余数分类
            base = top = 0; //初始化单调队列

            mj = (C-d)/V[i];
            for (j = 0; j < mj; j++){//j 表示第 i 种物品的等价物品数
                //将 j, F[j*V[i]+d] - j*W[i]插入单调队列
                IB = F[j*V[i]+d] - j*W[i];
                while (top != base && IB > B[top-1])
                    --top; //我们要维护的是单调递减的队列
                A[top] = j; B[top] = IB;

                if (A[base] < j-K[i]) ++base; //删除失效的队首元素
                F[j*V[i]+d] = B[base] + j*W[i];   //取队列头更新
            }
        }
    }
    return F[C];
}

int init(){ //如果必须要求把背包装满，只需将 F[1..C]初始化为 -inf
    fill(F, F + C, 0);
}
```

两个 01 串 S、T，对 S 的一次操作为将 S[i..j]整体赋值为 0 或 1，求达到 T 的最少操作次数

状态函数：dp'[i][j]: [0, i)都刷好了，且 i-1 被刷的次数为 j 次，使用的最少区间数
状态转移：用 dp[i][j]更新时(符号"←"表示 update)
1. T[i] == S[i]
dp[i+1][0] ← dp[i][j]
2. T[i] == T[i-1]
dp[i+1][j] ← dp[i][j] (j >= 1)
(其实应该写成 dp'[i+1][j], dp'[i+1][j-2], ... ← dp'[i][j])

3. T[i] != T[i-1]
dp[i+1][j+1] ← dp[i][j] + 1

dp[i+1][j-1] ← dp[i][j] (j >= 2)
(其实应该写成 dp'[i+1][j-1], dp'[i+1][j-3], ... ← dp'[i][j])

然后可以看出
dp'[i][1], dp'[i][3], dp'[i][5], ... 是(非严格)单调递增的
dp'[i][2], dp'[i][4], dp'[i][6], ...也是(非严格)单调递增的

所以这里的 dp'的含义，应该是 wy 教主实际使用的 dp 的 min，e.g. dp'[i][6] = min{dp[i][6], dp[i][8], dp[i][10], ...};

目标函数

res = min{dp'[n][0], dp'[n][1], dp'[n][2]}
= min{dp[n][0], dp[n][1], dp[n][2], dp[n][3], ..., dp[n][n]};

初始值：

dp'[0][i] = i
dp’[1..n][0..n] = inf

代码：

```
char S[maxn], T[maxn];
int dp[maxn][maxn];
inline void update(int &x, int y){ if (x > y) x = y;}

int main(){
    while (cin >> S >> T){
        int n = strlen(S);
        for (int i = 1; i <= n; i++)
            for (int j = 0; j <= n; j++)
                dp[i][j] = inf;
        for (int j = 0; j <= n; j++)
            dp[0][j] = j;
        for (int i = 0; i < n; i++){
            for (int j = 0; j <= n; j++){
                if (S[i] == T[i]) updata(dp[i+1][0], dp[i][j]);
                if (i >= 0 && T[i] == T[i-1])
                    if (j >= 1)
                        updata(dp[i+1][j], dp[i][j]);
                if (i == 0 || T[i] != T[i-1]){
                    updata(dp[i+1][j+1], dp[i][j] + 1);
                    if (j >= 2)
                        updata(dp[i+1][j-1], dp[i][j]);
                }
            }
        }
        int res = inf;
        for (int j = 0; j <= n; j++)
            updata(res, dp[n][j]);
        cout << res << endl;
    }
}
```

n 种硬币，面值 a[i]，数量 c[i]，要支付面值 dpcnt[j]的方法数

```
fill(dpcnt, dpcnt + m + 1, 0);
dpcnt[0] = 1;
for (int i = 0; i < n; i++){
	static int dpcnt2[maxc + 1];
	fill(dpcnt2, dpcnt2 + m + 1, 0);
	for (int j = 0; j < a[i]; j++){
		static int Q[maxc + 1], *Qbeg, *Qend;
		Qbeg = Qend = Q;
		int tot = 0;
		for (int k = j; k <= m; k += a[i]){
			if (Qend - Qbeg > c[i]) tot -= *Qbeg++;
			dpcnt2[k] += tot;
			tot += *Qend++ = dpcnt[k];
		}
	}
	for (int j = 0; j <= m; j++)
		dpcnt[j] += dpcnt2[j];
}
```

n 堆石子，加上最少的数目的石子，使得异或为 0

```
const int maxn = 10, inf = 1000000000; //上次使用 2012.9.21
int x[maxn], dp[25][1 << maxn], s[25], mask, n;

int calc(int pos, int stat) {
	if (pos == 22) {
		if ( stat == (1 << n) - 1 ) return inf; else return popcount(stat) % 2 ? 1 << pos : 0;
	}
	int &ans = dp[pos][stat];
	if ( ans != -1 ) return ans;
	ans = inf;
	int necessary = s[pos] & stat, optional = s[pos] ^ stat;
	for (int sub = optional; ; sub = (sub - 1) & optional) {
		int cost = popcount(sub) << pos;
		if ( parity(optional ^ sub) == 1 && popcount(optional) < n )
			cost += 1 << pos;
		if ( parity(optional ^ sub) == 0 || popcount(optional) < n )
			checkmin( ans, calc(pos + 1, sub | necessary) + cost );
		if (sub == 0) break;
	}
	return ans;
```

```
}

void solve() {
    mask = (1 << n) - 1;
    for (int i = 0; i < n; ++i) cin >> x[i];
    if ( n == 1 ) {
        puts( x[0] == 0 ? "0" : "impossible" ); return;
    }
    for (int i = 0; i < 25; ++i) {
        s[i] = 0;
        for (int j = 0; j < n; ++j) s[i] = s[i] << 1 | testbit(x[j], i);
    }
    memset(dp, -1, sizeof dp);
    cout << calc(0, 0) << endl;
}

int main() { while (cin >> n) solve(); } //hdu4317
```

动态规划优化

四边形不等式

在动态规划中，有一种常见的状态转移方程(如最小代价子母树)

$$\mathrm{m}(i,j)=\begin{cases}\min\limits_{i<k\le j}\{\mathrm{m}(i,k-1)+\mathrm{m}(k,j)+\mathrm{w}(i,j)\} & i<j\\ 0 & i=j\\ \infty & i>j\end{cases}$$

定义：对于$i\le i'<j\le j'$

函数 w 满足关于区间包含的单调性 *iff.* $\mathrm{w}(i',j)\le\mathrm{w}(i,j')$

函数 w 满足四边形不等式 *iff.* $\mathrm{w}(i,j)+\mathrm{w}(i',j')\le\mathrm{w}(i',j)+\mathrm{w}(i,j')$

定理：假如函数 w 满足上述条件，那么 m 也满足四边形不等式，即：

$$\mathrm{m}(i,j)+\mathrm{m}(i',j')\le\mathrm{m}(i',j)+\mathrm{m}(i,j')$$

定义：$\mathrm{s}(i,j)$为函数$\mathrm{m}(i,j)$对应的决策变量的**最大值**，即

$$\mathrm{s}(i,j)=\max_{i<k\le j}\{\mathrm{m}(i,j)=\mathrm{m}(i,k-1)+\mathrm{m}(k,j)+\mathrm{w}(i,j)\}$$

定理：若$\mathrm{m}(i,j)$满足四边形不等式，则$\mathrm{s}(i,j)$单调，即：

$$\mathrm{s}(i,j-1)\le\mathrm{s}(i,j)\le\mathrm{s}(i+1,j)$$

于是状态方程等价于

$$\mathrm{m}(i,j)=\begin{cases}\min\limits_{s(i,j-1)\le k\le s(i+1,j)}\{\mathrm{m}(i,k-1)+\mathrm{m}(k,j)+\mathrm{w}(i,j)\} & i<j\\ 0 & i=j\\ \infty & i>j\end{cases}$$

其中$s(i,i)=\mathrm{i}$

注：若$\mathrm{w}_k(i,j)=\left(\sum_{t=i}^{j}p(t)\right)-p(t)$，则$\mathrm{m}(i,j)$满足四边形不等式

四边形不等式的优化一般对于枚举分界点的动规都适用，但仍有例外，使用之前最好加以证明，据网上经验，求最小值时大多成立，而求最大值时并不满足
比赛时自己去证明来判断不现实。
推荐两种方法：
1、 看数据规模：N>1000 的，可以优化；反之不可
2、编写没优化和有优化的程序各一个。再弄个数据生成器，自己生成数据测试。连续 5 次以上两个程序答案相同，就...

石子合并 – O(n²)

```
const int maxn = 1000, inf = 1000000000;
int m[maxn][maxn], s[maxn][maxn], v[maxn], vs[maxn], n;
int main(){ //cugb1081
    while ( scanf("%d", &n) == 1){
        for (int i = 1; i <= n; i++) scanf("%d", &v[i]), v[i+n] = v[i];
        vs[0] = 0;
        for (int i = 1; i <= n + n; i++)
            vs[i] = vs[i-1] + v[i], s[i][i] = I, m[i][i] = 0;
        for (int d = 1; d < n; d++){
            for (int i = 1, j; (j = i + d) <= 2 * n; i++){
                int minval = inf;
                for (int k = s[i][j-1]; k <= s[i+1][j]; k++){
                    int thisval = m[i][k-1] + m[k][j];
                    if (thisval < minval) minval = thisval;
                    if (thisval == minval) s[i][j] = k;
                }
                m[i][j] = (minval += vs[j] - vs[i-1]);
            }
        } // end loop d
        int minval = inf;
        for (int i = 1; i <= n; i++) minval = min(minval, m[i][i+n-1]);
        cout << minval << endl;
    }
}
```

DLX(TODO 不熟)

精确覆盖

```
//已经通过 hust1017，zju3209

/* Procedure Algorithm_X(Dep)
    如果矩阵中所有的列均被删除，找到一组合法解，退出.
    任意选择一个未被删除的列 c,
    枚举一个未被删除的行 r，且 Matrix[r][c] = 1，将(r, c)加入 Ans.
        枚举所有的列 j, Matrix[r][j] = 1，将第 j 列删除.
            枚举所有的行 i, Matrix[i][j] = 1，将第 i 行删除.
        Algorithm_X(Dep + 1)
*/

struct Head;

struct Node {
    Node *L, *R, *U, *D;
    Head *H;
    int Line;
};

struct Head : Node{
    Head *L, *R;
    int size;
};

void remove(Head *h){
    //表示将 h 这一列的每个结点所在的行的所有结点全部删去,
    //且将 p 从第 0 行的链中删去
    h->R->L = h->L; h->L->R = h->R;
    for (Node *p = h->D; p != h; p = p->D){
        for (Node *q = p->R; q != p; q = q->R){
            q->D->U = q->U; q->U->D = q->D; --q->H->size;
        }
    }
}

void resume(Head *h){
    for (Node *p = h->U; p != h; p = p->U){
        for (Node *q = p->L; q != p; q = q->L){
            q->U->D = q; q->D->U = q; ++q->H->size;
```

```
            }
        }
        h->R->L = h->L->R = h;
}

const int maxm = 1005, maxn = 1005, maxe = maxm * maxn;
Head h[maxn], *hh;
Node e[maxe], *etop;
int Res[maxn], ResCnt;

void exact_cover(int dep = 0){
        if (dep >= ResCnt) return;
        if (hh->R == hh) {
            ResCnt = dep; return;
        }
        Head *hmin = hh->R;
        for (Head *h = hmin->R; h != hh; h = h->R)
            if (h->size < hmin->size) hmin = h;
        if (hmin->size == 0) return;
        remove(hmin);
        for (Node *p = hmin->D; p != hmin; p = p->D){
            for (Node *q = p->R; q != p; q = q->R) remove(q->H);
            Res[dep] = p->Line;
            exact_cover(dep+1);
            for (Node *q = p->L; q != p; q = q->L) resume(q->H);
        }
        resume(hmin);
}

void build(){
        int m, n, p;
        scanf("%d%d%d", &n, &m, &p);

        for (int i = 0; i < (n) * (m); i++){
            h[i].L = &h[i-1]; h[i].R = &h[i+1];
            h[i].U = h[i].D = &h[i]; h[i].size = 0;
        }
        hh = &h[(n) * (m)];
        hh->L = &h[(n) * (m) - 1]; h[(n) * (m) - 1].R = hh;
        hh->R = &h[0]; h[0].L = hh;
        etop = e;
        Node *dummy = etop++;

        for (int i = 1; i <= p; i++){
```

```
            int x1, y1, x2, y2;
            scanf("%d%d%d%d", &x1, &y1, &x2, &y2);

            dummy->L = dummy->R = dummy;
            for (int x = x1; x < x2; x++){
                for (int y = y1; y < y2; y++){
                    int c = x * (m) + y;
                    etop->D = &h[c];
                    etop->U = etop->D->U;
                    etop->D->U = etop->U->D = etop;
                    etop->L = dummy;
                    etop->R = etop->L->R;
                    etop->L->R = etop->R->L = etop;
                    etop->H = &h[c];
                    etop->H->size++;
                    etop->Line = i;
                    etop++;
                }
            }
            dummy->L->R = dummy->R;
            dummy->R->L = dummy->L;
        }
}

int main(){
    int t;
    scanf("%d", &t);
    while (t--){
        build();
        ResCnt = INT_MAX;
        exact_cover();
        if (ResCnt == INT_MAX) ResCnt = -1;
        cout << ResCnt << endl;
    }
}
```

一般覆盖

```
//已经通过 nuaa1507, tju3129(hdu2295)
struct Node {
    Node *L, *R, *U, *D, *H; int I;
};

void remove(Node *h){
```

```
    for (Node *p = h->D; p != h; p = p->D){
        p->L->R = p->R; p->R->L = p->L;
    }
}

void resume(Node *h){
    for (Node *p = h->U; p != h; p = p->U){
        p->L->R = p->R->L = p;
    }
}

const int maxm = 1005, maxn = 1005, maxe = maxm * maxn;
Node h[maxn], *hh;
Node e[maxe], *etop;

int hfunc(){
    static int hash[maxn];
    memset(hash, 0, sizeof hash);
    int ret = 0;
    for (Node *h = hh->R; h != hh; h = h->R){
        if (!hash[h - hh]){
            ret++;
            hash[h - hh] = true;
            for (Node *p = h->D; p != h; p = p->D){
                for (Node *q = p->R; q != p; q = q->R){
                    hash[q->H - hh] = true;
                }
            }
        }
    }
    return ret;
}

bool cover(int dep, int lim){
    if (dep + hfunc() > lim) return false;
    if (hh->R == hh) {
        return true;
    }
    Node *hmin = hh->R;
    for (Node *h = hmin->R; h != hh; h = h->R)
        if (h->I < hmin->I) hmin = h;
    for (Node *p = hmin->D; p != hmin; p = p->D){
        remove(p);
        for (Node *q = p->R; q != p; q = q->R) remove(q);
```

```
            Res[dep] = p->I;
            if (cover(dep+1, lim)) return true;
            for (Node *q = p->L; q != p; q = q->L) resume(q);
            resume(p);
        }
        return false;
}

int m, n, k;

complex<double> pR[maxm], pC[maxn];

bool build(double lim){
        for (int i = 0; i <= n; i++){
            h[i].L = &h[i-1]; h[i].R = &h[i+1];
            h[i].U = h[i].D = &h[i]; h[i].I = 0;
        }
        hh = &h[0];
        hh->L = &h[n]; h[n].R = hh;
        etop = e;
        Node *dummy = etop++;

        for (int i = 1; i <= m; i++){
            dummy->L = dummy->R = dummy;
            for (int j = 1; j <= n; j++){
                if (norm(pR[i] - pC[j]) > lim) continue;
                etop->D = &h[j];
                etop->U = etop->D->U;
                etop->D->U = etop->U->D = etop;
                etop->L = dummy;
                etop->R = etop->L->R;
                etop->L->R = etop->R->L = etop;
                etop->H = &h[j];
                etop->H->I++;
                etop->I = i;
                etop++;
            }
            dummy->L->R = dummy->R;
            dummy->R->L = dummy->L;
        }
        return true;
}

int main(){
```

```
    int casecnt;
    scanf("%d", &casecnt);
    while (casecnt--){
        scanf("%d%d%d", &n, &m, &k);
        for (int i = 1; i <= n; i++){
            scanf("%lf%lf", &pC[i].real(), &pC[i].imag());
        }
        for (int i = 1; i <= m; i++){
            scanf("%lf%lf", &pR[i].real(), &pR[i].imag());
        }
        double Low = 0, High = 1000 * 1000, Mid, Res;
        while (Low <= High + eps){
            Mid = (Low + High) / 2;
            build(Mid);
            if (cover(0, k)){
                Res = Mid;
                High = Mid - eps;
            } else {
                Low = Mid + eps;
            }
        }
        printf("%.6f\n", sqrt(Res));
    }
}
```

数独

```
const int maxdigit = 9;
const int maxn = maxdigit * maxdigit;
const int maxr = maxdigit * maxdigit * maxdigit;
const int maxc = maxdigit * maxdigit * 4;
int n = 9;

struct node {
    int r, c;
    node *U, *D, *L, *R;
} pool[maxr * maxc + maxr + maxc + 1], *pooltop, *row[maxr], *col[maxc], *head;

int cnt = 0, lim = 2;
int size[maxc];
int ans[maxn][maxn];

void init(int r, int c){
    pooltop = pool;
```

```
    head = pooltop++;
    head->r = r; head->c = c;
    head->L = head->R = head->U = head->D = head;
    for (int i = 0; i < c; i++){
        col[i] = pooltop++;
        col[i]->r = -1; col[i]->c = i;
        col[i]->L = head; col[i]->R = head->R;
        col[i]->U = col[i]->D = col[i]->L->R = col[i]->R->L = col[i];
        size[i] = 0;
    }
    for (int i = r - 1; i >= 0; i--){
        row[i] = pooltop++;
        row[i]->r = i; row[i]->c = -1;
        row[i]->U = head; row[i]->D = head->D;
        row[i]->L = row[i]->R = row[i]->D->U = row[i]->U->D = row[i];
    }
}

void insert(int r, int c){
    node *p = pooltop++;
    p->r = r; p->c = c;
    p->R = row[r]; p->L = row[r]->L;
    p->L->R = p->R->L = p;
    p->U = col[c]; p->D = col[c]->D;
    p->U->D = p->D->U = p;
    ++size[c];
}

void delLR(node *p){
    p->L->R = p->R; p->R->L = p->L;
}

void resumeLR(node *p){
    p->L->R = p->R->L = p;
}

void delUD(node *p){
    p->U->D = p->D; p->D->U = p->U;
}

void resumeUD(node *p){
    p->U->D = p->D->U = p;
}
```

```
void cover(int c){
	if (c == -1) return;
	delLR(col[c]);
	for (node *p = col[c]->D; p != col[c]; p = p->D)
		for (node *q = p->L; q != p; q = q->L)
			--size[q->c], delUD(q);
}

void resume(int c){
	if (c == -1) return;
	for (node *p = col[c]->U; p != col[c]; p = p->U)
		for (node *q = p->R; q != p; q = q->R)
			++size[q->c], resumeUD(q);
	resumeLR(col[c]);
}

bool DLX(int k){
	if (head->L == head)
		return (++cnt == lim);
	int msize = maxr * 2, c = -1;
	for (node *p = head->L; p != head; p = p->L)
		if (size[p->c] < msize)
			msize = size[c = p->c];
	if (msize == 0) return 0;
	cover(c);
	for (node *p = col[c]->D; p != col[c]; p = p->D){
		for (node *q = p->L; q != p; q = q->L)
			cover(q->c);
		int r = p->r;
		if (cnt == 0)
			ans[r/(n*n)][r/n%n] = r%n;
		if ( DLX(k+1) ) return 1;
		for (node *q = p->R; q != p; q = q->R)
			resume(q->c);
	}
	resume(c);
	return 0;
}

int mp[maxn][maxn];
int block[maxn][maxn], blockcnt;

int dir[4][2] = { -1, 0,	0, 1,	1, 0,	0, -1,};
```

```
void dfs(int i, int j){
	block[i][j] = blockcnt;
	for (int k = 0; k < 4; k++){
		if ( (mp[i][j] & (1 << k + 4)) == 0 ){
			int ii = i + dir[k][0], jj = j + dir[k][1];
			if (ii >= 0 && ii < n && jj >= 0 && jj < n && block[ii][jj] == -1)
				dfs(ii, jj);
		}
	}
}

void insert(int i, int j, int k){
	int r = (i * n + j) * n + k - 1;
	insert(r, k - 1 + i * n);
	insert(r, k - 1 + j * n + n * n);
	insert(r, k - 1 + block[i][j] * n + 2 * n * n);
	insert(r, j + i * n + 3 * n * n);
}

void Sodoku(){
	init(n*n*n,4*n*n);
	for (int i = 0; i < n; i++)
		for (int j = 0; j < n; j++)
			scanf("%d", &mp[i][j]), block[i][j] = -1;
	blockcnt = 0;
	for (int i = 0; i < n; i++)
		for (int j = 0; j < n; j++){
			if ( block[i][j] == -1 )
				dfs(i, j), blockcnt++;
	for (int i = 0; i < n; i++){
		for (int j = 0; j < n; j++){
			int k = mp[i][j] & 15;
			if (k)
				insert(i, j, k);
			else
				for (k = 1; k <= n; k++) insert(i, j, k);
		}
	}
	cnt = 0; lim = 2;
	DLX(0);
	static int icase = 0; printf("Case %d:\n", ++icase);
	if (cnt == 0){
		puts("No solution");
	} else if (cnt == 2){
```

```
            puts("Multiple Solutions");
        } else {
            for (int i = 0; i < n; i++){
                for (int j = 0; j < n; j++)
                    printf("%d", ans[i][j] + 1);
                puts("");
            }
        }
}

int main(){
    int _; scanf("%d", &_); while (_--) Sodoku();
}
```

点集的最小支配集

```
//在矩阵中选出一些行，把列覆盖掉
const int inf = 1000000000, maxn = 60;

struct node {
    int r, c;
    node *U, *D, *L, *R;
} pool[maxn * maxn * 2], *pooltop, *row[maxn], *col[maxn], *head;

int size[maxn];

void init(int r, int c){
    pooltop = pool;
    head = pooltop++;
    head->c = c; head->r = r;
    head->L = head->R = head->U = head->D = head;
    for (int i = 0; i < c; i++){
        col[i] = pooltop++; col[i]->c = i; col[i]->r = -1; size[i] = 0;
        col[i]->L = head; col[i]->R = head->R;
        col[i]->U = col[i]->D = col[i]->L->R = col[i]->R->L = col[i];
    }
    for (int i = r-1; i >= 0; i--){
        row[i] = pooltop++; row[i]->c = -1; row[i]->r = i;
        row[i]->U = head; row[i]->D = head->D;
        row[i]->L = row[i]->R = row[i]->D->U = row[i]->U->D = row[i];
    }
}

void insert(int r, int c){
```

```
    node *p = pooltop++;
    p->c = c; p->r = r; ++size[c];
    p->R = row[r]; p->L = row[r]->L; p->U = col[c]; p->D = col[c]->D;
    p->U->D = p->D->U = p->L->R = p->R->L = p;
}

int rescnt = inf;

void remove(node *p){
    for (node *q = p->D; q != p; q = q->D)
        q->L->R = q->R, q->R->L = q->L;
}

void resume(node *p){
    for (node *q = p->U; q != p; q = q->U)
        q->L->R = q->R->L = q; //注意是 q 不是 p
}

int hfunc(){
    static int hash[maxn], idx; ++idx;
    int ret = 0;
    for (node *p = head->R; p != head; p = p->R){
        if ( hash[p->c] != idx ){
            ret++; hash[p->c] = idx;
            for (node *q = p->D; q != p; q = q->D)
                for (node *r = q->R; r != q; r = r->R)
                    if (r->c != -1)
                        hash[r->c] = idx;
        }
    }
    return ret;
}

void DLX(int dep){
    if ( dep + hfunc() >= rescnt ) return;
    if (head->R == head) {
        rescnt = dep; return;
    }
    int msize = inf, c = -1;
    for (node *p = head->R; p != head; p = p->R)
        if ( size[p->c] < msize )
            msize = size[ c = p->c ];
    if (msize == 0) return;
    remove( col[c] );
```

```
    col[c]->L->R = col[c]->R; col[c]->R->L = col[c]->L;
    for (node *p = col[c]->D; p != col[c]; p = p->D){
        p->R->L = p->L->R = p;
        for (node *q = p->R; q != p; q = q->R){
            if (q->c == -1) continue;
            remove(q);
        }
//          res[dep] = p->r;
        DLX(dep+1);

        for (node *q = p->L; q != p; q = q->L){
            if (q->c == -1) continue;
            resume(q);
        }
        p->R->L = p->L; p->L->R = p->R;
    }
    col[c]->L->R = col[c]->R->L = col[c];
    resume( col[c] );
}

int mat[maxn][maxn];

int main(){ //hdu3498
    int n, m;
    while ( scanf("%d%d", &n, &m) == 2 ){
        for (int i = 0; i < n; i++){
            for (int j = 0; j < n; j++)
                mat[i][j] = 0;
            mat[i][i] = 1;
        }
        for (int i = 0; i < m; i++){
            int a, b; scanf("%d%d", &a, &b); --a; --b;
            mat[a][b] = mat[b][a] = 1;
        }
        init(n, n);
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                if ( mat[i][j] )
                    insert(i, j);
        rescnt = inf;
        DLX(0);
        cout << rescnt << endl;
    }
}
```

快速排序选择

stl 已经提供快速选择算法 nth_element(iv.begin(), iv.begin() + 5, iv.end(), greater<int>());

快速权选择

```
int partition(int A[], double W[], int left, int right, double &Ltot, double &Rtot){
	int x = A[left];
	double y = W[left];
	Ltot = Rtot = 0;
	while(left < right){
		while (left < right && A[right] > x)
			Rtot += W[right--];
		if (left < right){
			A[left] = A[right]; W[left] = W[right]; Ltot += W[left++];
		}
		while (left < right && A[left] < x)
			Ltot += W[left++];
		if (left < right){
			A[right] = A[left]; W[right] = W[left]; Rtot += W[right--];
		}
	}
	A[left] = x; W[left] = y;
	return left;
}

int quickselect(int A[], double W[], int left, int right, double k){
	double Ltot, Rtot, tot = 0;
	for (int i = left; i <= right; i++) tot += W[i];

	int mid = partition(A, W, left, right, Ltot, Rtot);
	while (k < Ltot || k > tot - Rtot){
		if (k < Ltot){
			right = mid - 1;
		} else{
			k -= tot - Rtot; left = mid + 1;
		}
		mid = partition(A, W, left, right, Ltot, Rtot);
	}
	return A[mid];
}
```

搜索

n 个棍子分成 k 组 poj1011

```
int n, len, a[100], next[100];

bool dfs(int curLen = len, int head = 0){
	if (curLen == len){ //装满了
		int first = next[0];
		if (first == n+1) return true; //ok,  装完了
		next[0] = next[first];
		if ( dfs(a[first], 0) ) return true;
		next[0] = first;
	} else {
		int prev = head, last = -1;
		for (int i = next[head]; i != n+1; i = next[i]){
			if (a[i] + curLen <= len && a[i] != last) {
				next[prev] = next[i];
				if ( dfs(curLen + a[i], prev) ) return true;
				next[prev] = i;
			}
			prev = i;last = a[i];
		}
	}
	return false;
}

int main(){
	while (cin >> n, n){
		int sum = 0;
		for (int i = 1;i <= n; ++i)
			cin >> a[i], next[i] = i+1, sum += a[i];
		next[0] = 1;
		sort(a + 1, a + 1 + n, greater<int>() );

		for (int i = a[1]; i <= sum; i++){
			if (sum % i == 0){
				len = i;
				if ( dfs() ) cout << i << endl; break;
			}
		}
	}
```

```
}
```

常用例程

逆序数-归并排序

```
long long Calc(int *first, int *last){ //左开右闭，已经通过 cugb1118
	static int buffer[maxn];
	if (last - first <= 1) return 0;
	int *mid = first + (last - first) / 2;
	long long res = Calc(first, mid) + Calc(mid, last);
	int *p = first, *q = mid, *r = buffer;
	while (p < mid && q < last){
		if (*p > *q){ //注意一定要是>，否则不保证排序的稳定性并且会将相等也记为逆序
			*r++ = *q++; res += mid - p;
		} else{ // *p <= *q
			*r++ = *p++;
		}
	}
	while (p < mid) *r++ = *p++;
	while (q < last) *r++ = *q++;
	memcpy(first, buffer, sizeof(buffer[0]) * (last - first));
	return res;
}

int main(){
	for(int n; cin >> n, n;){
		for (int i = 0; i < n; i++) scanf("%d", a + i);
		cout << Calc(a, a + n) << endl;
	}
}
```

日期

```
int isleap(int y){ return y % 400 == 0 || y % 4 == 0 && y % 100 != 0;}

int date2int(int y, int m, int d){
	static int maxday[] = {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
	int r = 0;
	for (int i = 1; i < m; i++)
		r += (i == 2 && isleap(y) ? 29 : maxday[i]);
```

```
    return r += d;
}

int date_diff(int y1, int m1, int d1, int y2, int m2, int d2){
    int r1 = date2int(y1, m1, d1);
    int r2 = date2int(y2, m2, d2);
    while (y1 > y2)
        r1 += 365 + (isleap(--y1));
    while (y2 > y1)
        r2 += 365 + (isleap(--y2));
    return r1 - r2;
}

int weekday(int y, int m, int d) {
    if ( m <= 2 ) m += 12, --y;
    return (d + (m+1) * 26 / 10 + y + y / 4 + y / 100 * 6 + y / 400 + 5) % 7 + 1;
}

void gao(){ for(int y, m, d; cin >> y >> m >> d; ) cout << ( (const char *[]) { "", "monday", "tuesday",
"wednesday", "thursday", "friday", "saturday", "sunday" } [ weekday(y, m, d) ] ) << endl; }
```

硬币翻转

一串 n 个硬币，只可以翻转其中连续的 k 个。问翻转多少次可以翻成全是正面？（如果达不到输出-1）

已经通过 hdu3275

```
#define maxn 110000
int main(){
    int n, k;
    char status[maxn];
    int list[maxn];

    while (cin >> n >> k && (n + k)){
        scanf("%s", status);
        bool ok = true;
        int h = 0, r = 0;
        if (k == 0){
            for (int i=0; i<n; i++)
                if (status[i] == '0') {
                    ok = false; break;
                }
        } else { //list[h..r]用来储存翻转的终点位置的集合
            for (int i=0; i<n; i++){
                while (h < r && list[h] < i) h++;
```

```
                    if (i + k <= n){
                        if ((r - h + status[i] - '0') % 2 == 0)
                            list[r++] = i + k - 1;
                    } else{
                        if ((r - h + status[i] - '0') % 2 == 0){
                            ok = false; break;
                        }
                    }
                }
            }
            if (!ok) r = -1;
            printf("%d\n", r);
        }
        return 0;
}
```

冒泡排序趟数

问：对 n 个数进行冒泡排序，数可能相同，给定待排序的序列，排序趟数。
解：将{pair<a[i], i>}进行排序，设其排序后的位置为 f(i) (0 <= f(i) < n)，则答案为 max{ f(i) - i }.
证：由于每一趟最多把不在正确位置(相对正确位置靠右)的数移动一个格，所以这些趟移动是必要的。
而只要某个数在正确的位置的右边，经过一趟排序，他一定会向左移动一格，因为他的左边一定有某个比他大的数沉到他的右边。
	int bubble(int a[], int n) {
	for(int i = 0; i < n; ++i) {
	bool modified = false;
	for (int j = 1; j < n-i; ++j) {
	if (a[j-1] < a[j]) {
	swap(a[j-1], a[j]);
	modified = true;
	}
	}
	if (!modified) return i + 1;
	}
	return n;
	}

要求给出一个 01 组成的串，弄成 n 的倍数

时间复杂度 O(n)，已经通过 http://info.zjfc.edu.cn/acm/problemDetail.aspx?pid=1414

```
const int maxn = 10000 + 5;

struct {
    char val; //last digit 0 or 1, -1 for illegal
    int prev;
} a[maxn];
```

```
int Queue[maxn];
int *const Stack = Queue;

int main()
{
	int n;
	while (scanf("%d", &n) != EOF){
		if (n == 0){
			printf("0\n"); continue;
		}
		//init
		for (int i = 0; i < n; i++)
			a[i].val = -1;
		a[1 % n].val = '1';
		a[1 % n].prev = -1;

		int *Base, *Top;
		Base = Top = Queue;
		*Top++ = 1;
		while (Base != Top && a[0].val == -1){
			int x = *Base++;
			int y = (x * 10) % n;
			if (a[y].val == -1){
				a[y].val = '0'; a[y].prev = x; *Top++ = y;
			}
			int z = y + 1;
			if (z >= n) z -= n;
			if (a[z].val == -1){
				a[z].val = '1'; a[z].prev = x; *Top++ = z;
			}
		}
		Base = Top = Stack;
		int x = 0;
		do {
			*Top++ = a[x].val; x = a[x].prev;
		} while (x != -1);

		while (Base != Top)
			putchar(*--Top);
		putchar('\n');
	}
	return 0;
}
```

康托展开 – 字典序全排列与序号的转化(下标从 0 开始)

```
int perm2num(int p[], int n){
	int ret = 0, k = 1;
	for (int i = n-2; i >= 0; k *= n - i, i--)
		for (int j = i+1; j < n; j++)
			if (p[i] > p[j])
				ret += k;
	return ret;
}

void num2perm(int x, int p[], int n){
	for (int i = n-1; i >= 0; i--){
		p[i] = x % (n-i); x /= (n-i);
	}
	for (int i = n-1; i >= 0; i--)
		for (int j = i-1; j >= 0; j--)
			if (p[j] <= p[i])
				++p[i]; //p[i]的意义是已知的，比 a[i]小的数的个数
}
```

有 N 个二维的点，(各点的与其他点的最大曼哈顿距离)之和

```
//时间复杂度 O(NlogN)
#define lowbit(x) ((x)&((x)^(x-1)))
const int maxn = 100000 + 5;

struct Point {
	int x, y;
	friend bool greaterdiff(const Point& a, const Point& b) {
		return a.x - a.y > b.x - b.y;
	}
} p[maxn];

Point *pPtr[maxn];

bool lesssum(Point *a, Point *b) {
	return a->x + a->y < b->x + b->y;
}

long long sumx[maxn], sumy[maxn], cnt[maxn];

void update(long long *a, int i, int x) {
```

```
    for (; i < maxn; i += lowbit(i))
        a[i] += x;
}

long long query(long long *a, int i) {
    long long s = 0;
    for (; i >= 1; i -= lowbit(i))
        s += a[i];
    return s;
}

int main() {
    int n;
    while (scanf("%d", &n) != EOF) {
        for (int i = 1; i <= n; i++) {
            scanf("%d%d", &p[i].x, &p[i].y);
            sumx[i] = sumy[i] = cnt[i] = 0;
            pPtr[i] = &p[i];
        }
        sort(p + 1, p + n + 1, greaterdiff);

        for (int i = 1; i <= n; i++) {
            update(sumx, i, p[i].x); update(sumy, i, p[i].y); update(cnt, i, 1);
        }

        sort(pPtr + 1, pPtr + n + 1, lesssum);

        long long ansx = 0, ansy = 0;
        for (int i = 1; i <= n; i++) {
            int m = pPtr[i] - p;
            ansx += query(sumx, m) - query(cnt, m) * pPtr[i]->x;
            ansy += (query(sumy, n) - query(sumy, m))
                    - (query(cnt, n) - query(cnt, m)) * pPtr[i]->y;
            update(sumx, m, -pPtr[i]->x); update(sumy, m, -pPtr[i]->y); update(cnt, m, -1);
        }
        cout << ansx + ansy << endl;
    }
}
```

K 维点的最大曼哈顿距离

给定 N 个 k 维的点$A(a_1, a_2, \dots, a_k)$，定义任意 A、B 两点之间曼哈顿距离

$$d_{AB} = |a_1 - b_1| + |a_2 - b_2| + \cdots + |a_k - b_k|$$

求 N 个点之间曼哈顿距离的最大值(对于最小值，以下关系不成立)。

1. k = 2 时

$$\max\{|x_1-x_2|+|y_1-y_2|\}=\max\begin{cases}x_1-x_2+y_1-y_2\\x_1-x_2-y_1+y_2\\-x_1+x_2+y_1-y_2\\-x_1+x_2-y_1+y_2\end{cases}$$

$$=\max\begin{cases}(x_1+y_1)-(x_2+y_2)\\(x_1-y_1)-(x_2-y_2)\\(-x_1+y_1)-(-x_2+y_2)\\(-x_1-y_1)-(-x_2-y_2)\end{cases}=\max\begin{cases}\max\{(x_1+y_1)-(x_2+y_2)\}\\\max\{(x_1-y_1)-(x_2-y_2)\}\\\max\{(-x_1+y_1)-(-x_2+y_2)\}\\\max\{(-x_1-y_1)-(-x_2-y_2)\}\end{cases}$$

$$=\max\begin{cases}\max\{x_1+y_1\}-\min\{x_2+y_2\}\\\max\{x_1-y_1\}-\min\{x_2-y_2\}\\\max\{-x_1+y_1\}-\min\{-x_2+y_2\}\\\max\{-x_1-y_1\}-\min\{-x_2-y_2\}\end{cases}=\max\begin{cases}\max\{x+y\}-\min\{x+y\}\\\max\{x-y\}-\min\{x-y\}\\\max\{-x+y\}-\min\{-x+y\}\\\max\{-x-y\}-\min\{-x-y\}\end{cases}$$

$$=\max\begin{cases}\max\{x+y\}-\min\{x+y\}\\\max\{x-y\}-\min\{x-y\}\end{cases}$$

2. k > 2 时

$$\max\{|a_1-a_2|+|b_1-b_2|+\cdots+|y_1-y_2|+|z_1-z_2|\}$$

$$=\max\begin{cases}\max\{a+b+\cdots+y+z\}-\min\{a+b+\cdots+y+z\}\\\max\{a+b+\cdots+y-z\}-\min\{a+b+\cdots+y-z\}\\\max\{a+b+\cdots-y+z\}-\min\{a+b+\cdots-y+z\}\\\max\{a+b+\cdots-y-z\}-\min\{a+b+\cdots-y-z\}\\\quad\cdots\cdots\quad(\text{共}2^{k-1}\text{项})\\\max\{a-b+\cdots-y+z\}-\min\{a-b+\cdots-y+z\}\\\max\{a-b+\cdots+y-z\}-\min\{a-b+\cdots+y-z\}\\\max\{a-b+\cdots-y+z\}-\min\{a-b+\cdots-y+z\}\\\max\{a-b+\cdots-y-z\}-\min\{a-b+\cdots-y-z\}\end{cases}$$

3. 时间复杂度$O(2^{k-1}N)$

N 数码问题可解性判断

```
/* n 数码问题可解性判断：

对于 m*n 的矩阵，两个格局可以互相转化当且仅当：
将空格用 0 代替以后
两格局中 0 的曼哈顿距离 + 逆序数之差 为偶数
(更加严密的方法是把 0 移到右下角后算逆序数)
*/

const int maxn = 1000 + 5;

int Calc(int *first, int *last){
    static int buffer[maxn * maxn];   //函数要求有一个足够大的 buffer 数组作为缓冲区
    if (last - first <= 1) return 0;

    int *mid = first + (last - first) / 2;
```

```
    int res = Calc(first, mid) + Calc(mid, last);
    int *p = first, *q = mid, *r = buffer;
    while (p < mid && q < last){
        if (*p > *q){ //注意一定要是>，否则不保证排序的稳定性并且会将相等也记为逆序
            *r++ = *q++; res += mid - p;
        } else{ // *p <= *q
            *r++ = *p++;
        }
    }
    while (p < mid) *r++ = *p++;
    while (q < last) *r++ = *q++;
    memcpy(first, buffer, sizeof(buffer[0]) * (last - first));
    return res;
}

int a[maxn * maxn];

int main(){
    int m, n;
    while (cin >> m >> n, m + n){
        int mn = m * n, zerox, zeroy;
        for (int i = 0; i < mn; i++){
            scanf("%d", &a[i]);
            if (a[i] == 0) { zerox = i / n; zeroy = i % n; }
        }
        int ans = Calc(a, a + mn);
        printf("%s\n", (ans + mn + m-1-zerox + n-1-zeroy) & 1 ? "YES" : "NO");
    }
}
```

神奇的位运算

High_bit

```
int highbit(int x){
    return x |= x >> 1, x |= x >> 2, x |= x >> 4, x |= x >> 8, x |= x >> 16, x ^= x >> 1;
}
int highbitind(int x){
    return 31 - __builtin_clz(x);
}
```

反转位的顺序

```
unsigned rev_bit(unsigned x){
    x = (x & 0x55555555) << 1 | (x >> 1 & 0x55555555);
    x = (x & 0x33333333) << 2 | (x >> 2 & 0x33333333);
    x = (x & 0x0f0f0f0f) << 4 | (x >> 4 & 0x0f0f0f0f);
    x = (x & 0x00ff00ff) << 8 | (x >> 8 & 0x00ff00ff);
    x = (x & 0x0000ffff) <<16 | (x >>16 & 0x0000ffff);
    return x;
}
```

将 x 上调(下调)为 align 的倍数

```
size_t roundup(size_t x, size_t align){ //将 x 上调为 align 的倍数，其中 align = 2 ^ k
    return (x + (align - 1)) & ~(align - 1);
}
size_t rounddown(size_t x, int k) { //将 x 下调为 2^k 的倍数
    return x & ((-1) << k);
}
//round up to the next highest pow of 2
int roundup(int v){
    --v;
    v |= v >> 1; v |= v >> 2; v |= v >> 4; v |= v >> 8; v |= v >> 16;
    return ++v;
}
```

遍历一个掩码的所有子集掩码

```
void iterateSubset(int mask){ //枚举 mask 的非平凡子集
    for (int sub = (mask - 1) & mask; sub > 0; sub = (sub - 1) & mask){
        int incsub = sub ^ mask; //  递增顺序的子集
        //blablabla
    }
}
```

注：

1. 若改为 int sub = mask 则 sub 含 mask，incsub 含 0
2. 若改为 sub != 0 则 sub 含 0，incsub 含 mask (注意：若 popcount(mask) = 1, 本条失效)
3. 若改为 for(int sub = mask, itertime = 1 << pop_count(mask); itertime--; sub = (sub-1) & mask);则 sub/incsub 既含 0 又含 mask

下一个包含同样数量的二进制 1 的掩码

```
unsigned snoob(unsigned mask){
    unsigned smallest, ripple, ones;    //  mask   = xxx0 1111 0000
    smallest = mask & -mask;           //smallest = 0000 0001 0000
    ripple = mask + smallest;          // ripple  = xxx1 0000 0000
    ones = mask ^ ripple;              //   ones   = 0001 1111 0000
    ones = (ones >> 2) / smallest;     //   ones   = 0000 0000 0111
    return ripple | ones;              //            xxx1 0000 0111
}
```

遍历{0, 1, …, n-1}的所有 k 元子集

```
void iterateSubset(int n, int k){
    int s = (1 << k) – 1;
    for (;!(s & (1 << n)); s = snoob(s)){
        blablabla;
    }
}
```

n 皇后问题

1	2	3	4	5	6	7	8	9	10
1	0	0	2	10	4	40	92	352	724
11	12	13	14	15	16	17	18		
2680	14200	73712	365596	2279184	14772512	95815104	666090624		

```
int upperlim; // (1 << n) – 1
int sum;

void nqueuecnt(int row = 0, int ld = 0, int rd = 0){
    if (row == upperlim)
        ++sum;
    else {
        int pos = upperlim & (~(row | ld | rd)), p, ret = 0;
        for (; pos; pos -= p){
            p = lowbit(pos); nqueuecnt(row | p, (ld | p) << 1, (rd | p) >> 1);
        }
    }
}
int main(){
    int n; cin >> n;
    upperlim = (1 << n) – 1; sum = 0;
```

```
    nqueuecnt(); cout << sum << endl;
}
```

SteinerTree

```
struct SteinerTree {
    static const int maxn = 200 + 5, maxm = 7;
    static const int inf = 1000000000;

    int a[maxn], n; //点权
    vector< pair<int, int> > ge[maxn];
    int critical[maxm], m;
    int dp[maxn][1 << maxm];
    pair<int, int> path[maxn][1 << maxm];
    int ans[maxn], anscnt;

    void init(int nn){
        n = nn; m = 0;
        for (int i = 0; i < n; ++i){
            ge[i].clear(); a[i] = 0;
        }
    }
    void addvertexcost(int i, int v){ //添加点权
        a[i] += v;
    }
    void addedge(int a, int b, int c){
        ge[a].pb( mp(b, c) ); ge[b].pb( mp(a, c) );
    }
    void addcritical(int a){
        critical[m++] = a;
    }

    void getans(int u, int s){ //如果还需要存边的信息，只需要让 path[][]存 < e, s >
        if (u < 0) return;
        ans[ anscnt++ ] = u;
        int v = path[u][s].first, t = path[u][s].second;
        getans(v, t);
        if ( t != s && t ) getans( v, s ^ t );
    }

    int solve(){ //复杂度 3 ^ m * |V| + 2 ^ m * SSSP(V, E)
        assert( m != 0 );
        int mask = (1 << m) - 1;
```

```
        REP(i, n) REP(j, mask + 1) dp[i][j] = inf;
        for (int i = 0; i < m; ++i){
            dp[ critical[i] ][1 << i] = a[ critical[i] ];
            path[ critical[i] ][1 << i] = mp(-1, -1);
        }
        for (int s = 1; s <= mask; ++s){
            for (int i = 0; i < n; ++i){
                for (int p = (s - 1) & s; p; p = (p - 1) & s){
                    int t = dp[i][p] + dp[i][p ^ s] - a[i];
                    if ( t < dp[i][s] ){
                        dp[i][s] = t, path[i][s] = mp(i, p);
                    }
                }
            }
            queue<int> q; static bool in[maxn];
            for (int i = 0; i < n; in[i++] = true ) q.push(i);
            while ( q.size() ){
                int u = q.front(); q.pop(); in[u] = false; //if (dp[u][s] == inf) continue;
                for ( int k = 0; k < ge[u].size(); ++k ){
                    int v = ge[u][k].first, c = ge[u][k].second;
                    int t = dp[u][s] + c + a[v];
                    if ( t < dp[v][s] ){
                        dp[v][s] = t; path[v][s] = mp(u, s);
                        if ( !in[v] ) {
                            in[v] = true; q.push(v);
                        }
                    }
                }
            }
        }
        int k = 0;
        for (int i = 0; i < n; ++i){
            if ( dp[i][mask] < dp[k][mask] ){
                k = i;
            }
        }
        anscnt = 0; getans(k, mask);
        return dp[k][mask];
    }
} st;

void solve(int r, int c, int k){
    static char ans[105][105];
    st.init(r * c);
```

```
    #define hash(i_, j_) ( (i_) * (c) + (j_) )
    REP(i, r) REP(j, c ) {
        ans[i][j] = '.';
        int v; scanf("%d", &v);
        st.addvertexcost( hash(i, j), v );
        if (i) st.addedge( hash(i, j), hash(i-1, j), 0 );
        if (j) st.addedge( hash(i, j), hash(i, j-1), 0 );
    }
    for (int x, y; k--; ){
        scanf("%d%d", &x, &y); st.addcritical( hash(x-1, y-1) );
    }
    cout << st.solve() << endl;
    for (int i = 0; i < st.anscnt; ++i){
        int x = st.ans[i] / c, y = st.ans[i] % c;
        ans[x][y] = 'X';
    }
    for (int i = 0; i < r; ++i){
        REP(j, c)putchar( ans[i][j] );
        puts("");
    }
}

int main(){
    for (int r, c, k; cin >> r >> c >> k; solve(r, c, k) );
}
```

有无穷的 kinds[1..kindcnt]的硬币，组成 n 元钱的方法数？

```
//时间复杂度 O(sum{kinds[1..kindcnt] ^ 3 * log(n)})
const int maxn = 100;
int kinds[] = {50, 25, 10, 5, 1};
const int kindcnt = sizeof(kinds) / sizeof(kinds[0]);
const int maxkinds = 50; // = *max_element(kinds, kinds + kindcnt);
int dp[kindcnt][maxkinds];
int sum[kindcnt+1];

struct mat {
    int m, n, data[maxn][maxn];
};

mat operator *(const mat& a, const mat &b){
    assert(a.n == b.m);
    mat r; r.m = a.m; r.n = b.n;
    memset(r.data, 0, sizeof r.data);
```

```
    for (int i = 0; i < r.m ; i++)
        for (int j = 0; j < r.n; j++)
            for (int k = 0; k < a.n; k++)
                r.data[i][j] += a.data[i][k] * b.data[k][j];
    return r;
}

mat makeA(){
    mat A; A.m = A.n = sum[kindcnt];
    memset(A.data, 0, sizeof A.data);
    for (int i = 0, j = 0; i < sum[kindcnt]; i++)
        if (i == sum[j])
            for (int k = ++j; k <= kindcnt; k++)
                A.data[i][sum[k] - 1] = 1;
        else
            A.data[i][i-1] = 1;
    return A;
}

mat makeP(){
    mat P; P.m = sum[kindcnt]; P.n = 1;
    int k = maxkinds - 1;
    for (int i = 0, j = 1; i < sum[kindcnt]; i++){
        P.data[i][0] = dp[kindcnt-j][k--];
        if (i == sum[j] - 1)
            j++, k = maxkinds - 1;
    }
    return P;
}

mat makeE(int n){
    mat E; E.m = E.n = n;
    memset(E.data, 0, sizeof E.data);
    for (int i = 0; i < n; i++) E.data[i][i] = 1;
    return E;
}

void makeInit(){
    sum[0] = 0;
    for (int i = 0; i < kindcnt; i++)
        sum[i+1] = sum[i] + kinds[i];
    for (int i = 0; i < kindcnt; i++)
        dp[i][0] = 1;
    for (int j = 1; j < maxkinds; j++)
```

```
        for (int i = 0; i < kindcnt; i++)
            dp[i][j] = (j - kinds[kindcnt-i-1] >= 0 ? dp[i][j - kinds[kindcnt-i-1]] : 0)
                    + (i - 1 >= 0 ? dp[i - 1][j] : 0);
}

mat pow(mat A, int n){
    assert(A.m == A.n);
    mat R = makeE(A.m);
    for (;n; n >>= 1, A = A * A)
        if (n & 1) R = R * A;
    return R;
}

int calc(int n){
    if (n < maxkinds) return dp[kindcnt-1][n];
    return (pow(makeA(), n - maxkinds + 1) * makeP()).data[0][0];
}
```

把坐标转化为 Excel 字母那种坐标

```
string toExcel(LL n){
    return n == 0 ? "" : toExcel(n / 26) + char('A' + --n % 26);
}
```

生成随机数

```
#include <tr1/random>
tr1::ranlux64_base_01 eng( time(0) );
tr1::normal_distribution<double> gen(0.0, 1.0); //正态分布
tr1::uniform_real<double> gen(3, 7);
tr1::uniform_int<int> gen(1, 52);
cout << gen(eng) << endl;
```

附录

一些常量

INT_MAX = 2147483647 > 2.1E9
UINT_MAX = 4294967295 > 4.2E9
LONG_LONG_MAX = 9223372036854775807 > 9.2E18
ULONG_LONG_MAX = 18446744073709551615 > 1.8E19

numeric_limits<#>:#	short	int	Long long	float	double	long double
Digit	15	31	63	24	53	64
Digit10	4	9	18	6	15	18

一些函数原型

```
double atan2(double y, double x);
typedef struct{ long int quot, rem;} div_t;
div_t div(int number, int denom);
double fmod(double x, double y);
double frexp(double value, int *eptr); //将浮点数分为底数和指数
double hypot(double x, double y); //计算三角形的斜边，操作失败返回无穷大.
double ldexp(double value, int exp); //计算 value * (2 ^ exp)
double log(double x); //自然对数
double log10(double x); //常用对数
double modf(double value, double *iptr); //分割浮点数为整数部分和小数部分
char *strchr(char *str, char c);
int strcspn(char *str1, char *str2); //str1 中，不含 str2 中字符的最长的前缀的长度
char *strncat(char *destin, char *source, size_t maxlen);
int strncmp(char *str1, char *str2, size_t maxlen);
char *strncpy(char *destin, char *source, int maxlen);
char *strstr(char *str1, char *str2); //查找 str2 在 str1 中首次出现的位置
double strtod(char *str, char **endptr);
long strtol(char *str, char **endptr, int radix);
long strtoul(char *str, char **endptr, int radix);
ForwardIterator unique(ForwardIterator first, ForwardIterator last, BinaryPredicate binary_pred);
OutputIterator set_union(InputIterator1 first1, InputIterator1 last1,
      InputIterator2 first2, InputIterator2 last2, OutputIterator result[, Compare comp] );
OutputIterator set_difference(InputIterator1 first1, InputIterator1 last1,
      InputIterator2 first2, InputIterator2 last2, OutputIterator result[, Compare comp] );
OutputIterator set_intersection(InputIterator1 first1, InputIterator1 last1,
      InputIterator2 first2, InputIterator2 last2, OutputIterator result[, Compare comp] );
OutputIterator set_symmetric_difference(InputIterator1 first1, InputIterator1 last1,
      InputIterator2 first2, InputIterator2 last2, OutputIterator result[, Compare comp] );
template <class InputIterator1, class InputIterator2>
   bool lexicographical_compare ( InputIterator1 first1, InputIterator1 last1,
                                  InputIterator2 first2, InputIterator2 last2 );
template <class T, class Container = vector<T>,
      class Compare = less<typename Container::value_type> > class priority_queue;
bool is_sorted(ForwardIterator first, ForwardIterator last[, StrictWeakOrdering comp]);
```

C++ Complex Library

```
class complex {
// Constructors and Conversion Operators
public:
    complex();
    complex(double real, double imag = 0.0);

//abs, arg, conj, imag, norm, polar, real
public:
    friend double abs(complex a);
    friend double arg(complex a);
    friend complex conj(complex a); //共轭
    friend double imag(complex a);
    friend double norm(complex a);
    friend complex polar(double r, double t);
    friend double real(complex a);

//exp, log, pow, sqrt
public:
    friend complex exp(complex a);
    friend complex log(complex a);
    friend complex pow(double a, complex b);
    friend complex pow(complex a, int b);
    friend complex pow(complex a, double b);
    friend complex pow(complex a, complex b);
    friend complex sqrt(complex a);

//sin[h], cos[h]
public:
    friend complex sin[h](complex a);
    friend complex cos[h](complex a);

//Complex Operators
public:
    friend complex operator +(complex a, complex b);
    friend complex operator -(complex a);
    friend complex operator -(complex a, complex b);
    friend complex operator *(complex a, complex b);
    friend complex operator /(complex a, complex b);
    friend complex operator /(complex a, double d);
    friend int operator ==(complex a, complex b);
    friend int operator !=(complex a, complex b);
```

```
    complex &operator +=(complex a);
    complex &operator -=(complex a);
    complex &operator *=(complex a);
    complex &operator /=(complex a);
    complex &operator /=(double d);

//Complex I/O Functions
public:
    ostream& operator <<(ostream& os, complex c);
    istream& operator >>(istream& is, complex& c);   //形式为(a, b)，如需要，请重载为 a b
};
```

bitset

```
template <size_t N>
class bitset{
    //bit reference
    class reference {
        friend class bitset<N>;
    public:
        ~reference();
        reference &operator= (bool);
        reference &operator= (const reference&);
        bool operator~() const;
        operator bool() const;
        reference &flip();
    };

    //constructors
    bitset();
    bitset(unsigned long);
    explicit bitset(const string&, size_t = 0, size_t = (size_t) (-1) );
    bitset<N> &operator =(const bitset<N> &);

    //bitwise operators and bitwise operator assignment
    bitset<N> &operator &= (const bitset<N> &);
    bitset<N> &operator |= (const bitset<N> &);
    bitset<N> &operator ^= (const bitset<N> &);
    bitset<N> &operator <<= (size_t);
    bitset<N> &operator >>= (size_t);

    //set, reset, flip
    bitset<N> &set();
    bitset<N> &set(size_t, int = 1);
```

```
    bitset<N> &reset();
    bitset<N> &reset(size_t);
    bitset<N> &operator ~() const;
    bitset<N> &flip();
    bitset<N> &flip(size_t);

    //element access
    reference operator[] (size_t);
    unsigned long to_ulong() const;
    string to_string() const;
    size_t count() const;
    size_t size() const; //return N
    bool operator== (const bitset<N> &) const;
    bool operator!= (const bitset<N> &) const;
    bool test (size_t) const;
    bool any() const;
    bool none() const;
    bitset<N> operator << (size_t) const;
    bitset<N> operator >> (size_t) const;
};

//Non-member operators
template <size_t N> bitset<N> operator & (const bitset<N> &, const bitset<N> &);
template <size_t N> bitset<N> operator | (const bitset<N> &, const bitset<N> &);
template <size_t N> bitset<N> operator ^ (const bitset<N> &, const bitset<N> &);
template <size_t N> istream &operator >> (istream &, bitset<N> &);
template <size_t N> ostream &operator << (ostream &, const bitset<N> &);

/* Bitset is missing two boolean queries: all() and some()
all: (~bitset).none()
some: (~bitset).any()
*/
```

gcc builtin 函数

```
int __builtin_ffs(unsigned int x);
return one plus index of least significant 1-bit of x, is x is zero, returns zero.
int __builtin_clz(unsigned int x);
int __bulitin_ctz(unsigned int x);
int __builtin_popcount(unsigned int x);
int __builtin_parity(unsigned int x); // popcount(x) % 2
double __builtin_powi(double, int);
```

hash_map 使用方法

Example1

```
#include <ext/hash_map>
using namespace std;
using namespace __gnu_cxx;

hash_map<int, int, hash<int> > mymap; //第三个参数为判断两个参数 1 类型的数据是否相等
typedef hash_map<int, int>::iterator iterator;
有时还需要重载 hash 仿函数和 equal_to 仿函数  e.g.
namespace __gnu_cxx {
	template <> struct hash<T> const{
		size_t operator()(const T& x){
		}
	};
}
```

Example2

```
#include <ext/hash_map>
#include <string>
#include <iostream>
#include <algorithm>
using namespace __gnu_cxx;
using namespace std;

struct str_hash {
	size_t operator() (const string& s) const {
		return __stl_hash_string( s.c_str() );
	}
};
struct str_equal {
	bool operator() (const string& s1, const string& s2) const {
		return s1 == s2;
	}
};
int main(){
	hash_map<string, int, str_hash, str_equal> mymap;
	mymap.insert( make_pair("a", 1) );
	mymap["b"] = 2;
	typeof(mymap.begin()) iter1 = mymap.begin(), iter2 = mymap.end(), iter;
```

```
    if ( (iter = mymap.find("a")) != mymap.end())
        cout << iter->first << " " << iter->second << endl; //a 1
    typeof(mymap) hmap;
    hmap.insert(iter1, iter2);
    cout << hmap.size() << endl; //2
}
```

运算符的优先级和结合律

Precedence	Operator	Associativity
1	() [] -> . :: ++ --(在后)	left to right
2	! ~ ++ --(在前) + -(正负号) * & (type) sizeof	**right to left**
3	->* .*	left to right
4	* / %	left to right
5	+ -	left to right
6	<< >>	left to right
7	< <= > >=	left to right
8	== !=	left to right
9	&	left to right
10	^	left to right
11	|	left to right
12	&&	left to right
13	||	left to right
14	? :	**right to left**
15	= += -= *= /= %= &= ^= |= <<= >>=	**right to left**
16	,	left to right

英语

below – 小于(不含等号)
evenly divisible - divisible with no remainder
subtree – A tree G' whose graph vertices and graph edges from subsets of the graph vertices and graph edges of a given tree G.
latitude – 纬度
longitude – 经度

java

头文件

```
import java.io.*;
import java.util.*;
import java.math.*;

class Task{
	void solve(int icase, InputReader in, PrintWriter out){
		BigDecimal a = new BigDecimal("123124161246591812865124.41224");
		a = a.round(new MathContext(30, RoundingMode.HALF_UP) );
		//保留k位有效数字
		out.println( a.toPlainString() );
	}
}

public class Main {
	public static void main(String []args){
		InputStream inputStream = System.in;
		OutputStream outputStream = System.out;
		InputReader in = new InputReader(inputStream);
		PrintWriter out = new PrintWriter(outputStream);
		Task solver = new Task();
		solver.solve(1, in, out);
		out.close();
	}
}

class InputReader {
	private BufferedReader reader;
	private StringTokenizer tokenizer;

	public InputReader(InputStream stream){
		reader = new BufferedReader(new InputStreamReader(stream));
		tokenizer = null;
	}

	public String next(){
		while (tokenizer == null || !tokenizer.hasMoreTokens()){
			try {
				tokenizer = new StringTokenizer(reader.readLine());
```

```
                } catch (IOException e){
                    throw new RuntimeException(e);
                }
            }
            return tokenizer.nextToken();
        }

        public int nextInt(){
            return Integer.parseInt(next());
        }

        public long nextLong(){
            return Long.parseLong(next());
        }
}

class OutputWriter {
        private final PrintWriter writer;

        public OutputWriter(OutputStream outputStream){
            writer = new PrintWriter(outputStream);
        }

        public OutputWriter(Writer writer){
            this.writer = new PrintWriter(writer);
        }

        public void print(Object...objects){
            for (int i = 0; i < objects.length; i++){
                if (i != 0)
                    writer.print(' ');
                writer.print(objects[i]);
            }
        }

        public void println(Object...objects){
            print(objects);
            writer.println();
        }

        public void close() {
            writer.close();
        }
}
```

```
class IOUtils {
    public static int []readIntArray(InputReader in, int size){
        int [] array = new int[size];
        for (int i = 0; i < size; i++)
            array[i] = in.nextInt();
        return array;
    }
}
```

分数

```
class Frac implements Comparable<Frac> {
    final static Frac ZERO = new Frac(BigInteger.ZERO, BigInteger.ONE);
    final static Frac ONE = new Frac(BigInteger.ONE, BigInteger.ONE);

    private BigInteger a, b;
    Frac(BigInteger a, BigInteger b){
        BigInteger g = a.gcd(b);
        a = a.divide(g); b = b.divide(g);
        if (b.signum() < 0){
            a = a.negate();  b = b.negate();
        }
        this.a = a; this.b = b;
    }
    Frac add(Frac f){
        return new Frac( this.a.multiply(f.b).add(this.b.multiply(f.a)),
                this.b.multiply(f.b) );
    }
    Frac sub(Frac f){
        return new Frac( this.a.multiply(f.b).subtract(this.b.multiply(f.a)),
                this.b.multiply(f.b) );
    }
    Frac mul(Frac f){
        return new Frac( this.a.multiply(f.a), this.b.multiply(f.b) );
    }
    Frac div(Frac f){
        return new Frac( this.a.multiply(f.b), this.b.multiply(f.a) );
    }
    Frac gcd(Frac f){
        return new Frac( this.a.gcd(f.a),
                this.b.divide(this.b.gcd(f.b)).multiply(f.b) );
    }
    int signum(){ // b.signum() is always positive
```

```
        return a.signum();
    }
    public int compareTo(Frac f){
        return this.a.multiply(f.b).compareTo(this.b.multiply(f.a));
    }
    Frac abs(){
        return new Frac(a.abs(), b);
    }
    public String toString(){
        String r = a.toString();
        if (!b.equals(BigInteger.ONE)){
            r += "/" + b.toString();
        }
        return r;
    }
}
```

大实数

```
class Double implements Comparable<Double> {
    static final int precision = 50;
    static final Double ZERO = new Double(BigDecimal.ZERO);
    static final Double ONE = new Double(BigDecimal.ONE);
    static final MathContext mc =
            new MathContext(precision, RoundingMode.HALF_UP);

    private BigDecimal val;
    private Double(BigDecimal val){
        this.val = val;
    }
    Double(double val){
        this.val = new BigDecimal(val, mc);
    }
    Double add(Double f){
        return new Double(val.add(f.val, mc));
    }
    Double sub(Double f){
        return new Double(val.subtract(f.val, mc));
    }
    Double mul(Double f){
        return new Double(val.multiply(f.val, mc));
    }
    Double div(Double f){
```

```
            return new Double(val.divide(f.val, mc));
        }
        int signum(){
            return val.signum();
        }
        Double abs(){
            return new Double(val.abs());
        }
        public String toString(){
            return val.toEngineeringString();
        }
        public String toString(int n){ // output the first n digits of this
            String s = this.val.unscaledValue().toString();
            return s.substring(0, Math.min(s.length(), n));
        }
        public int compareTo(Double o) {
            return this.val.compareTo(o.val);
        }
        public static void main(String args[]){
            // output the first n digits of Fibnacci numbers~~~~~
            int n = 500;
            Double fib[] = new Double[n+1];
            fib[0] = fib[1] = Double.ONE;
            for (int i = 2; i < n; i++){
                fib[i] = fib[i-1].add(fib[i-2]);
                System.out.println(fib[i].toString(10));
            }
        }
}

class Real implements Comparable<Real> {
        static final int scale = 20;
        static final BigDecimal eps = BigDecimal.ONE.divide(
                BigDecimal.TEN.pow(scale), scale, BigDecimal.ROUND_HALF_UP);
        static final Real ZERO = new Real(BigDecimal.ZERO);
        static final Real ONE = new Real(BigDecimal.ONE);

        private BigDecimal val;
        Real(double val){
            this.val = new BigDecimal(val).setScale(scale);
        }
        private Real(BigDecimal val){
            this.val = val;
        }
```

```
    Real add(Real f){
        return new Real(val.add(f.val));
    }
    Real sub(Real f){
        return new Real(val.subtract(f.val));
    }
    Real mul(Real f){
        return new Real(val.multiply(f.val)
                .setScale(scale, BigDecimal.ROUND_HALF_UP));
    }
    Real div(Real f){
        return new Real(val.divide(f.val, BigDecimal.ROUND_HALF_UP));
    }
    int signum(){
        return val.abs().compareTo(eps) <= 0 ? 0 : val.signum();
    }
    Real abs(){
        return new Real(val.abs());
    }
    public int compareTo(Real f){
        return this.val.compareTo(f.val);
    }
    public String toString(){
        return this.val.toPlainString();
    }
}
```

BigDecimal

上次使用 2012.8

```
BigDecimal t = new BigDecimal("123456789.987654321");
t.setScale(2, RoundingMode.HALF_UP) //小数点后 k 位
t.round( new MathContext(5, RoundingMode.HALF_UP)  ) //k 位有效数字
```

class 排序

```
class node implements Comparable {
    public int key;
    public int compareTo(Object obj) {
        if (obj instanceof node) {
            node b = (node) obj;
            if (this.key < b.key) return -1;
            if (this.key > b.key) return 1;
```

```
        }
        return 0;
    }
}

public class Main {
    static node a[];
    public static void main(String args[]) throws Exception {
        //blablabla
        Arrays.sort(a);
        //blablabla
    }
}
```

Java 相等

注：要覆盖 equals 函数，一定要同时覆盖 hashCode 函数

```
class Elem {
    @Override public boolean equals(Object rhs){
    }
    @Override public int hashCode() {
    }
}
```

任意进制数的转化

```
public static String toString(BigInteger n, int k){
    if (Math.abs(k) <= 1) return null;
    if (Math.abs(k) > 36) return null;
    if (k >= 2) return n.toString(k);
    StringBuffer ret = new StringBuffer();
    BigInteger negk = BigInteger.valueOf(-k);
    while (!n.equals(BigInteger.ZERO)){
        BigInteger qr[] = n.abs().divideAndRemainder(negk);
        if (n.signum() < 0){
            qr[1] = qr[1].negate();
            if (qr[1].signum() < 0){
                qr[1] = qr[1].add(negk);
                qr[0] = qr[0].add(BigInteger.ONE);
            }
        } else {
            qr[0] = qr[0].negate();
        }
```

```
            n = qr[0];
            ret.append("0123456789abcdefghijklmnopqrstuvwxyz".charAt(qr[1].intValue()));
        }
        return ret.reverse().toString();
}
```

Trick

transform(s.begin(), s.end(), s.begin(), ::tolower); 其中 s:string
//因为 std::tolower 在 locale 中重载为 inline _CharT tolower(_CharT __c, const locale& __loc)