C8 - Solution

A - 报数

难度	考点
1	条件语句，循环语句

题目分析

模拟即可。

示例代码

```
#include <stdio.h>
int main()
{
    int n;
    scanf("%d",&n);
    for(int i=1;i<=99;i++)
    {
        if(!(i%n == 0 || i%10 == n|| i/10 == n))
            printf("%d ",i);
    }
    return 0;
}
```

B - EDGnb

难度	考点
1	字符串

题目分析

将字符串读入，然后依次对每一位判断是否出现了`EDGnb`五个字符即可。

示例代码

```
#include <stdio.h>
#include <string.h>
int main()
{
    int i, l;
    char s[777 + 5];
    scanf("%s", s + 1);
    l = strlen(s + 1);
```

```
    for (i = 1; i <= l; i++)
        if (i + 4 <= l && s[i] == 'E' && s[i + 1] == 'D' && s[i + 2] == 'G' && s[i + 3]
== 'n' && s[i + 4] == 'b')
        {
            printf("%d\n", i);
            return 0;
        }
    printf("-1\n");
    return 0;
}
```

C - Monica的雪花

难度	考点
3	函数，循环

题目分析

主要考察了对函数与简单循环判断的结合。

示例代码

```
#include<stdio.h>
#include<string.h>
#include<math.h>
#define MN (1000+5)
int L;
int s,cnt;
int judge(int x){
    int sqr=sqrt(x);
    for(int i=2;i<=sqr;++i)
        if(x%i==0)
            return 0;
    return 1;
}
int main(){
    scanf("%d",&L);
    for(int i=2;i+s<=L;++i)
        if(judge(i)){
            s+=i;
            cnt++;
        }
    printf("%d\n",cnt);
    return 0;
}
```

D - 前20长字符串

难度	考点
3	二维字符数组

题目分析

本题目的实现方法非常简单，可以20次遍历每次找到一个最长的，也可以构造一个前20长的序列来维护它是已读入序列的前20长的字符串

注意避免将字符数组开在主函数里，不然运行的空间不够编译器会闪退，直接提交也会爆REG

示例代码

```
#include <stdio.h>
#include <string.h>
char s[100005][1005];
int a[24], id[24];
void update(int len, int idc)
{
    for (int i = 1; i <= 20; i++)
    {
        if (len > a[i])
        {
            int min = a[i], mini = i;
            for (int j = i + 1; j <= 20; j++)
                if (a[j] <= min)
                    min = a[j], mini = j;
            for (int j = mini + 1; j <= 20; j++)
                a[j - 1] = a[j], id[j - 1] = id[j];
            a[20] = len, id[20] = idc;
            break;
        }
    }
}
int main()
{
    int ic;
    for (ic = 1; ic <= 20; ic++)
        gets(s[ic]), a[ic] = (int)strlen(s[ic]), id[ic] = ic;
    while (gets(s[++ic]) != NULL)
        update((int)strlen(s[ic]), ic);
    for (int i = 1; i <= 20; i++)
        printf("%s\n", s[id[i]]);
    return 0;
}
```

E - 垒石头

难度	考点
4	排序、贪心

题目分析

为表述方便，定义蓝x(x 唯一正整数)含义为有一蓝色石堆，石头数为 x 。类似定义红x。

可以发现，交换两个石堆对结果没有影响，不妨把红的石堆和蓝的石堆先单独分出来并且进行排序（也就是Hint里的"先分组在排序"）。

思路一：从 1 开始到 n 看看有没有石堆来填充对应的石头数。例如我们现在想要一个石头数为 3 的石堆，我们可以使用红2也可以使用蓝4来得到该石堆。此外，如果是从 1 开始枚举，注意需要先使用蓝色石堆来填充。

思路二：考虑"错落有致"的序列的等价状态。假设总共有 n 个石堆，其中 k 个是蓝色石堆 $(n-k)$ 个红色石堆。可以发现"错落有致"的序列有一等价状态为**前 k 个均为蓝色，后 $n-k$ 个均为红色**（例如蓝3和红2可以变换为蓝2和红3）。这样只须考虑蓝色能否填满前 k 个数，红色能否填满后 $n-k$ 个数即可。

示例代码

```c
// 本程序对应思路1。使用冒泡排序。
#include <stdio.h>
#include <stdlib.h>

char s[1005];
int b[1005], r[1005];
int a[1005];
void bubbleSort(int a[], int n)
{
    int i, j, hold, flag;
    for (i = 0; i < n - 1; i++)
    {
        flag = 0;
        for (j = 0; j < n - 1 - i; j++)
            if (a[j] > a[j + 1])
            {
                hold = a[j];
                a[j] = a[j + 1];
                a[j + 1] = hold;
                flag = 1;
            }
        if (0 == flag)
            break;
    }
}
int main()
{
    int t, n, lenb, lenr;
    scanf("%d", &t);
    while (t--)
    {
        lenb = lenr = 0;
        scanf("%d", &n);
        for (int i = 0; i < n; i++)
            scanf("%d", &a[i]);
        scanf("%s", s);
        for (int i = 0; i < n; i++)
        {
            if (s[i] == 'R')
```

```
                r[lenr++] = a[i];
            else if (s[i] == 'B')
                b[lenb++] = a[i];
        }
        bubbleSort(b, lenb);
        bubbleSort(r, lenr);
        int ok = 1;
        for (int i = 1, j = 0, k = 0; i <= n; i++)
        {
            if (b[j] >= i && j < lenb)
                j++;
            else if (r[k] <= i && k < lenr)
                k++;
            else
                ok = 0;
        }
        if (ok)
            puts("YES");
        else
            puts("NO");
    }
    return 0;
}
```

```
// 本程序对应思路2。使用快速排序（qsort）。
#include <stdio.h>
#include <stdlib.h>

char s[1005];
int b[1005], r[1005];
int a[1005];

int cmpl(const void *x, const void *y)
{
    return *(int *)x - *(int *)y;
}
int cmpg(const void *x, const void *y)
{
    return *(int *)y - *(int *)x;
}
int main()
{
    int t, n, lenb, lenr;
    scanf("%d", &t);
    while (t--)
    {
        lenb = lenr = 0;
        scanf("%d", &n);
        for (int i = 0; i < n; i++)
            scanf("%d", &a[i]);
        scanf("%s", s);
        for (int i = 0; i < n; i++)
        {
            if (s[i] == 'R')
                r[lenr++] = a[i];
```

```
            else if (s[i] == 'B')
                b[lenb++] = a[i];
        }
        qsort(b, lenb, sizeof(b[0]), cmpl);
        qsort(r, lenr, sizeof(r[0]), cmpg);
        int ok = 1;
        for (int i = 0; i < lenb; i++)
        {
            if (b[i] < i + 1)
                ok = 0;
        }
        for (int i = 0; i < lenr; i++)
        {
            if (r[i] > n - i)
                ok = 0;
        }
        if (ok)
            puts("YES");
        else
            puts("NO");
    }
    return 0;
}
```

F - 黄金分割与一维搜索

难度	考点
4	二分查找思想，模拟

题目分析

参考二分查找的思想，重点在于理解搜索范围的缩小机制，结合函数图像理解，$f(\lambda) > f(\mu)$ 时，说明应该把 λ 作为下一次搜索的左起点，而保留之前的右起点不变。类似的当 $f(\lambda) \le f(\mu)$ 时将 μ 作为下一次搜索的右起点（至于$f(\lambda) = f(\mu)$ 的情况，根据数分知识，因为 $\lambda \ne \mu$，由罗尔中值定理不难证明极值点 $x \in (\lambda, \mu)$）

为什么选取 $t = \frac{\sqrt{5}-1}{2}$ 作为系数呢，原因在于 λ, μ 的计算，理论上每一次缩小区间后，需要重新计算 λ, μ 的值，然而本题中：

- $f(\lambda) > f(\mu)$ 时，原先的 μ 可以作为新的 λ 值
- $f(\lambda) \le f(\mu)$，原先的 λ 可以作为新的 μ 值

这样一来重要在于每一次搜索只用计算一次函数值，本题的多项式函数最高幂次比较小所以无所谓，而当多项式最高幂次比较大时，每一次重新计算函数值开销的累积则不能忽略。

关于λ, μ 可以复用的证明，原先的λ, μ 和下一次搜索的 λ', μ'：对于搜索区间$[\alpha, \beta]$，即证明：
$\lambda = \alpha + (1-t)(\beta - \alpha) = \mu' = \alpha + t(\mu - \alpha)$，又有：$\mu = \alpha + t(\beta - \alpha) = \lambda' = \lambda + (1-t)(\beta - \lambda)$

随便拿一个方程出来消元 λ, μ，比如第二个，
$t(\beta - \alpha) = (1-t)(\beta - \alpha) + (1-t)(\beta - \lambda) = (1-t)(1+t)(\beta - \alpha)$，$\beta - \alpha$ 项可以约掉，得到关于 t 的二次方程：$t^2 + t - 1 = 0$，解方程即可得证。

顺便关于秦九韶算法，关于多项式 $f(x) = a_n x^n + \ldots + a_0 x^0$，朴素的想法是每次计算 x^i 反映在 C 语言就是调用 $pow(x, i)$ 会增大复杂度，最后复杂度是 $O(n + \ldots + 1 + 0) = O(\frac{n(n+1)}{2}) = O(n^2)$，而秦九韶算法则是采用这样一种方式：$f(x) = (((((a_n x) + a_{n-1})x) + a_{n-2})x \ldots)x + a_0$，用C语言实现即：`f = f * x + a[i]`，复杂度降为 $O(n)$

示例代码

```
#include <stdio.h>
const double t = 0.618;
const double eps = 1e-6;
int n, coef[15];
double f(double x)
{
    double ret = coef[n];
    for (int i = n - 1; i >= 0; --i)
        ret = ret * x + coef[i];
    return ret;
}

double fib_search(double l, double r)
{
    double lambda, mu, fl, fm;
    lambda = l + (1 - t) * (r - l);
    mu = l + t * (r - l);
    fl = f(lambda), fm = f(mu);
    while (fabs(r - l) > eps)
    {
        if (fl > fm)
        {
            l = lambda;
            lambda = mu;
            mu = l + t * (r - l);
            fl = fm;
            fm = f(mu);
        }
        else
        {
            r = mu;
            mu = lambda;
            lambda = l + (1 - t) * (r - l);
            fm = fl;
            fl = f(lambda);
        }
    }
    return (l + r) / 2;
}
int main()
{
    int L, R;
    double ans;
    scanf("%d", &n);
    for (int i = n; i >= 0; --i)
        scanf("%d", coef + i);
    scanf("%d%d", &L, &R);
```

```
    ans = fib_search((double)L, (double)R);
    printf("f(%.4lf)=%.4lf\n", ans, f(ans));
    return 0;
}
```

G - 跳格子

难度	考点
4	递推

题目分析

先考虑只能一次跳1格或2格的情况。

设$F[i]$表示跳到i号格的方法数。

在这种情况下，只能由$i-1$号和$i-2$号格跳到i号格。

那么“到i号格的方法数”即为“到$i-1$号格的方法数”+“到$i-2$号格的方法数”。

即

$$F[i]=F[i-1]+F[i-2]$$

那么考虑一次跳3格的情况呢?

设$G[i]$表示**跳到i号格且没有从$i-3$号格起跳**的方法数。不从$i-3$号格起跳，就只能从$i-1$和$i-2$号格起跳，所以$G[i]=F[i-1]+F[i-2]$。

显然有

$$F[i]=F[i-1]+F[i-2]+G[i-3]$$

即

$$F[i]=F[i-1]+F[i-2]+F[i-4]+F[i-5]$$

接下来只需要知道$F[1],F[2],F[3],F[4],F[5]$的值，就可以推出一切$F[i]$的值。

$F[1],F[2],F[3]$的值很显然，分别为$1,2,4$。

$F[4]=F[1]+F[2]+F[3]=7$

$F[5]=F[2]+F[3]+F[4]=13$

示例代码

```
#include<stdio.h>
int main()
{
    int ans[10005]={0,1,2,4,7,13};
    for(int i=6;i<=10000;i++){
        ans[i]=(ans[i-1]+ans[i-2]+ans[i-4]+ans[i-5])%10007;
    }
    //这道题会问很多很多很多次的N，所以直接把F[1]~F[10000]的结果都算出来，问的时候直接取结果就好，不必重复计算。
    int N;
```

```
    while(scanf("%d",&N)!=EOF){
        printf("%d\n",ans[N]);
    }
    return 0;
}
```

H - 简单易懂的题

难度	考点
4	字符串，高精度

题目分析

一个字符串表示的 n 进制数转十进制数的通常方法是：

```
int x = 0;
for (int i; s[i]; i ++)
    x = x * n + (s[i] >= '0' && s[i] <= '9' ? : s[i] - '0' : s[i] - 'A' + 10);
```

观察数据范围，本题应实现高精乘低精和高精加低精。

在进行高精度计算的时候，通常是将数字从低位到高位按顺序存到数组里，以便于计算。并且要注意维护数字的位数 len ，否则无法正确计算和输出。具体方法见代码和注释。

示例代码

```
#include <stdio.h>

int n, len = 1, a[2005];
char s[1005];

void muln()
{
    for (int i = 0; i < len; i ++) //先将每一位都乘n
        a[i] *= n;
    len += 3; //由于n最大为36，所以位数增加不会超过3
    for (int i = 0; i < len; i ++) //处理进位
        a[i+1] += a[i] / 10, a[i] %= 10;
    while (len > 1 && !a[len]) len --; //如果有前导零则减少位数，但位数至少为1
}

void add(int x)
{
    int c = x; //c表示进位
    for (int i = 0; c; i ++) //如果有进位就一直循环往后加，进位为0就退出循环
    {
        a[i] += c;
        c = a[i] / 10; //重新计算进位
        a[i] %= 10;
    }
    while (a[len]) len ++; //如果最高位的后面一位不为0，则位数增加
```

```
}

int main()
{
    scanf("%d %s", &n, s);
    for (int i = 0; s[i]; i ++)
    {
        int x = s[i] >= '0' && s[i] <= '9' ? s[i] - '0' : s[i] - 'A' + 10;
        muln();
        add(x);
    }
    while (len) //倒序输出
        printf("%d", a[-- len]);
    return 0;
}
```

I - 斐波那契的兔子

难度	考点
4	找规律，二分查找

题目分析

观察所给出的例图，我们会发现：每对兔子的父母的编号和这对兔子的编号之差总是一个斐波那契数，而且是不超过这对兔子编号的最大的那个斐波那契数。

于是为了从编号为a的兔子寻找到它的父母，只需要找到不超过a的最大的斐波那契数F，那么a的父母就是$a-F$，这一结论的证明如下：

令$f(i)$为第i个月的兔子总数（单位：对），那么显然有$f(0)=f(1)=1$。因为第 i个月时只有第$i-2$月就存在的兔子能生小兔子，所以：

$$f(i)=f(i-1)+f(i-2)$$

即：上一个月的兔子总数+新出生的（年龄大于2个月的每对生一对）兔子总数。

经观察，$\{f(n)\}$是一斐波那契数列。

按照编号规则，第i个月出生的第j对兔子的编号应当为：

$$x=f(i-1)+j$$

其父亲为j，显然有$j\leq f(i-2)$，那么$f(i-1)$就是不大于x的最大的斐波那契数，证毕。

我们把a和b一直向上递推，直到相遇，那么相遇的这个节点就是它们的"最近公共祖先"。

在数据范围内的斐波那契数大概有60个，如果顺序查找，有可能会TLE。因为斐波那契数列是有序的，所以我们可以二分查找。

因为本题数字比较大，所以记得开`long long`。

示例代码

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <ctype.h>
#include <stdlib.h>
#define LL long long
#define maxn 100+10
const double eps=1e-11;

LL fib[100];
void getFib(int);
void swap(LL*,LL*);
int lowerBound(LL);

int main() {
    getFib(90);
    LL m;
    scanf("%lld",&m);
    while(m--){
        LL a,b;
        scanf("%lld%lld",&a,&b);
        if(a<b) swap(&a,&b);

        while(a!=b){
            if(a<b) swap(&a,&b);
            a-=fib[lowerBound(a)];
        }
        printf("%lld\n",a);
    }
    return 0;
}

inline void getFib(int n){//递推计算斐波那契数表
    fib[1]=1LL;
    fib[2]=1LL;
    for(int i=3;i<=n;++i)
        fib[i]=fib[i-1]+fib[i-2];
}

inline void swap(LL *a,LL *b){//交换
    LL temp=*b;
    *b=*a;
    *a=temp;
}

inline int lowerBound(LL x){// 二分查找不大于x的最大的斐波那契数
    int l=1,r=90;
    while(l<r){
        int mid=(l+r)/2;
        if(r-l==1)
            return l;
        if(fib[mid]==x)
            return mid-1;
        if(fib[mid]>x)
            r=mid;
        if(fib[mid]<x)
```

```
            l=mid;
    }
    return l;
}
```

J - 排列之差

难度	考点
7	排列、数论、二分查找

题目分析

要想解决本题，首先要解决的问题是如何将给定的排列按字典序排好顺序。

将其看成一个 m 位 m 进制的数似乎是个不错的选择，这样就能建立全排列到整数的映射了。然而这个整数并不是该全排列的排名，因为排名是连续的，而该 m 进制数一定不是连续的。

我们考虑另外一种方法，称为康托展开。来看它的公式：

$X = a_n(n-1)! + a_{n-1}(n-2)! + \cdots + a_1 \cdot 0!$

据此公式，即可在 $O(m)$ 的时间求得任意全排列的排名。

我们将给定的 n 个全排列以排名为关键字排序，每次读入新的排列后求其排名，不妨设为 r，只需要在给定的 n 个全排列中二分查找排名大于 $r+k$ 的第一个全排列 P 和排名小于 $r-k$ 的最后一个全排列 Q，那么 $P+1 \sim Q-1$ 均为符合题意的全排列。

示例代码

```
#include<math.h>
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

#define int long long

int n, m, q,fac[15];
int hsh[100005], id[100005], b[100005];
char a[100005][15], s[100005][15], now[15];

void msort(int l, int r) {
    if (l == r) return;
    int mid = (l + r) >> 1;
    msort(l, mid), msort(mid + 1, r);
    int ql = l, qr = mid + 1, cnt = l;
    while (ql <= mid && qr <= r) {
        if (hsh[id[ql]] > hsh[id[qr]])
            b[cnt++] = id[qr], qr++;
        else b[cnt++] = id[ql++];
    }
    while (ql <= mid) b[cnt++] = id[ql++];
    while (qr <= r) b[cnt++] = id[qr++];
    for (int i = l; i <= r; i++)
```

```
        id[i] = b[i];
}

int cantor(int p, int n) {
    int x = 0;
    for (int i = 1; i <= n; ++i) {
        int smaller = 0;
        for (int j = i + 1; j <= n; ++j) {
            if (a[p][j] < a[p][i])
                smaller++;
        }
        x += fac[n - i] * smaller;
    }
    return x + 1;
}

int cantor2(int n) {
    int x = 0;
    for (int i = 1; i <= n; ++i) {
        int smaller = 0;
        for (int j = i + 1; j <= n; ++j) {
            if (now[j] < now[i])
                smaller++;
        }
        x += fac[n - i] * smaller;
    }
    return x + 1;
}

int find(int l, int r, int val) {
    while (l < r) {
        int mid = l + r >> 1;
        if (hsh[id[mid]] >= val) r = mid;
        else l = mid + 1;
    }
    return r;
}

signed main() {
    scanf("%lld%lld", &m, &n);
    fac[0] = 1;
    for (int i = 1; i <= m; i++) fac[i] = fac[i - 1] * i;
    for (int i = 1; i <= n; i++) scanf("%s", a[i] + 1), id[i] = i;
    for (int i = 1; i <= n; i++)
        hsh[i] = cantor(i, m);
    msort(1, n);
    id[n + 1] = n + 1, hsh[n + 1] = 2e18;
    for (int i = 1; i <= n; i++)
        memcpy(s[i], a[id[i]] + 1, sizeof s[i]);
    scanf("%lld", &q);
    while (q--) {
        int k;
        scanf("%s%lld", now + 1, &k);
        int p = cantor2(m);
        int l = find(1, n + 1, p - k), r = find(1, n + 1, p + k + 1);
        if (l == r) puts("No such permutation.");
```

```
        else for (int i = l; i < r; i++) printf("%s\n", s[i]);
    }
}
```