历年蓝桥杯决赛试题及解析

【网络整理】

2012 年第三届蓝桥杯 C/C++程序设计本科 B 组决赛

1.星期几

2.数据压缩

3.拼音字母

4.DNA 比对

5.方块填数

星期几

1949 年的国庆节（10 月 1 日）是星期六。

今年（2012）的国庆节是星期一。
那么，从建国到现在，有几次国庆节正好是星期日呢？
只要答案，不限手段！
可以用 windows 日历，windows 计算器，Excel 公式，。。。。。
当然，也可以编程！
不要求写出具体是哪些年，只要一个数目！
千万不要提交源代码！

答案：9

数据压缩

某工业监控设备不断发回采样数据。每个数据是一个整数(0 到 1000 之间)。各个数据间用空白字符（空格，TAB 或回车换行）分隔。这些数据以文本形式被存储在文件中。
因为大多数时候，相邻的采样间隔数据是相同的，可以利用这个特征做数据的压缩存储。其方法是：对 n(n>1)个连续相同的数字只记录 n 和该数字本身；对 m(m>0)个连续不重复的数字，则记录 m*-1 和这些数字本身（之所以用负数，是为了与第一种情况区分，便于解压缩）。
例如：采样数字：
12 34 34 25 25 25 25 11 15 17 28 14 22 22 22 13

则根据上述规则变化后：
-1 12 2 34 4 25 -5 11 15 17 28 14 3 22 -1 13
下面的程序实现了这个功能。请仔细阅读分析代码，填写空白的部分。

[cpp] view plain copy

```
void pop(int s, int* buf, int c, FILE* fp)
{
	int i;
	if(s)
	{
		fprintf(fp, "%d %d ", c, *buf);
	}
	else
	{
		fprintf(fp, "%d ", -c);
		for(i=0; i<c; i++)
		{
			fprintf(fp, "%d ", buf[i]);
		}
	}
}

void dopack(FILE* r, FILE* w)
{
	int buf[BUF_N];

	int pos = 0;	// 下一个数字在 buf 中将要存放的位置
	int c = 0;	// 当前段已读入的整数个数
	int pst;
	int cst;

	while(fscanf(r, "%d", buf+pos)==1)
	{
		if(c==0)
		{
			c = pos = 1;
			continue;
		}

		if(c==1)
		{
			pst = buf[0] == buf[1];
			pos = pos + 1 - pst;
```

```
            c = 2;
            continue;
        }

        cst = buf[pos-1] == buf[pos];
        if(pst && !cst)
        {
            pop(pst, buf, c, w);
            buf[0] = buf[1];
            c = pos = 1;
            pst = cst;
        }
        else if(!pst && cst || pos == BUF_N-1)
        {
            pop(pst, buf, c-1, w);
            buf[0] = buf[pos-1];
            c = 2;

            if(!cst)
            {
                buf[1] = buf[pos];
                pos = 2;
            }
            else
            {
                pos = 1;
                pst = _____;   // 填空 1
            }
        }
        else
        {
            c++;
            if(!pst) pos++;
        }
    } // while

    if(c>0) _____;    // 填空 2
}

void main()
{
    FILE* rfp;
    FILE* wfp;
```

```
	if((rfp=fopen(RFILE, "r")) == NULL)
	{
		printf("can not open %s!\n", RFILE);
		exit(1);
	}

	if((wfp=fopen(WFILE, "w")) == NULL)
	{
		printf("can not open %s!\n", WFILE);
		fclose(rfp);
		exit(2);
	}

	dopack(rfp, wfp);

	fclose(wfp);
	fclose(rfp);
}
```

答案：cst

pop(pst, buf, c, w)

[cpp] view plain copy

```
void pop(int s, int* buf, int c, FILE* fp)
{
	int i;
	if(s)
	{
		fprintf(fp, "%d %d ", c, *buf);
	}
	else
	{
		fprintf(fp, "%d ", -c);
		for(i=0; i<c; i++)
		{
			fprintf(fp, "%d ", buf[i]);
		}
	}
}

void dopack(FILE* r, FILE* w)
{
	int buf[BUF_N];

	int pos = 0;	// 下一个数字在 buf 中将要存放的位置
	int c = 0;		// 当前段已读入的整数个数
```

```
int pst;
int cst;

while(fscanf(r, "%d", buf+pos)==1)
{
	if(c==0)
	{
		c = pos = 1;
		continue;
	}

	if(c==1)
	{
		pst = buf[0] == buf[1];
		pos = pos + 1 - pst;
		c = 2;
		continue;
	}

	cst = buf[pos-1] == buf[pos];
	if(pst && !cst)
	{
		pop(pst, buf, c, w);
		buf[0] = buf[1];
		c = pos = 1;
		pst = cst;
	}
	else if(!pst && cst || pos == BUF_N-1)
	{
		pop(pst, buf, c-1, w);
		buf[0] = buf[pos-1];
		c = 2;

		if(!cst)
		{
			buf[1] = buf[pos];
			pos = 2;
		}
		else
		{
			pos = 1;
			pst = cst;    // 填空 1
		}
	}
```

```
            else
            {
                c++;
                if(!pst) pos++;
            }
        } // while

        if(c>0) pop(pst, buf, c, w);     //  填空 2
}

void main()
{
        FILE* rfp;
        FILE* wfp;

        if((rfp=fopen(RFILE, "r")) == NULL)
        {
            printf("can not open %s!\n", RFILE);
            exit(1);
        }

        if((wfp=fopen(WFILE, "w")) == NULL)
        {
            printf("can not open %s!\n", WFILE);
            fclose(rfp);
            exit(2);
        }

        dopack(rfp, wfp);

        fclose(wfp);
        fclose(rfp);
}
```

拼音字母

在很多软件中，输入拼音的首写字母就可以快速定位到某个词条。比如，在铁路售票软件中，输入：“bj”就可以定位到“北京”。怎样在自己的软件中实现这个功能呢？问题的关键在于：对每个汉字必须能计算出它的拼音首字母。

GB2312 汉字编码方式中，一级汉字的 3755 个是按照拼音顺序排列的。我们可以利用这个特征，对常用汉字求拼音首字母。

GB2312 编码方案对每个汉字采用两个字节表示。第一个字节为区号，第二个字节为区中的偏移号。为了能与已有的 ASCII 编码兼容（中西文混排），区号和偏移编号都从 0xA1 开始。我们只要找到拼音 a,b,c,...x,y,z 每个字母所对应的 GB2312 编码的第一个汉字，就可以定位

所有一级汉字的拼音首字母了（不考虑多音字的情况）。下面这个表给出了前述信息。请你利用该表编写程序，求出常用汉字的拼音首字母。

a 啊 B0A1
b 芭 B0C5
c 擦 B2C1
d 搭 B4EE
e 蛾 B6EA
f 发 B7A2
g 噶 B8C1
h 哈 B9FE
j 击 BBF7
k 喀 BFA6
l 垃 C0AC
m 妈 C2E8
n 拿 C4C3
o 哦 C5B6
p 啪 C5BE
q 期 C6DA
r 然 C8BB
s 撒 C8F6
t 塌 CBFA
w 挖 CDDA
x 昔 CEF4
y 压 D1B9
z 匝 D4D1

【输入、输出格式要求】

用户先输入一个整数 n (n<100)，表示接下来将有 n 行文本。接着输入 n 行中文串（每个串不超过 50 个汉字）。

程序则输出 n 行，每行内容为用户输入的对应行的汉字的拼音首字母。

字母间不留空格，全部使用大写字母。

例如：

用户输入：

3
大家爱科学
北京天安门广场
软件大赛

则程序输出：

DJAKX
BJTAMGC
RJDS

【注意】

请仔细调试！您的程序只有能运行出正确结果的时候才有机会得分！

在评卷时使用的输入数据与试卷中给出的实例数据可能是不同的。

请把所有函数写在同一个文件中，调试好后，拷贝到【考生文件夹】下对应题号的“解答.txt”

中即可。
相关的工程文件不要拷入。
源代码中不能使用诸如绘图、Win32API、中断调用、硬件操作或与操作系统相关的 API。
允许使用 STL 类库，但不能使用 MFC 或 ATL 等非 ANSI c++标准的类库。
例如，不能使用 CString 类型（属于 MFC 类库），不能使用 randomize, random 函数（不属于 ANSI C++标准）

[cpp] view plain copy

```
#include<iostream>
#include<cstring>
using namespace std;
string a[2][23]= {{"A", "B", "C", "D", "E", "F", "G", "H", "J", "K", "L", "M", "N", "O", "P", "Q", "R", "S", "T", "W", "X", "Y", "Z"},
    {"啊","芭","擦","搭","蛾","发","噶","哈","击","喀","垃","妈","拿","哦","啪","期","然","撒","塌","挖","昔","压","匝"}
};
string search(string &temp)
{
    for(int i=1; i<24; i++)
    {
        if(temp>a[1][i-1])continue;
        else return a[0][i-2];
    }
}
string fun(string &str)
{
    int len = str.length();
    string result = "";
    for(int i=0; i<len; i+=2)
    {
        string temp = str.substr(i,2);
        result += search(temp);
    }
    return result;
}
int main()
{
    int n;
    string str;
    cin>>n;
    for(int i=0; i<n; i++)
    {
        cin>>str;
```

```
            cout<<fun(str)<<endl;
        }
        return 0;
}
```

DNA 比对

脱氧核糖核酸即常说的 DNA，是一类带有遗传信息的生物大分子。它由 4 种主要的脱氧核苷酸(dAMP、dGMP、dCMT 和 dTMP)通过磷酸二酯键连接而成。这 4 种核苷酸可以分别记为：A、G、C、T。
DNA 携带的遗传信息可以用形如：AGGTCGACTCCA.... 的串来表示。DNA 在转录复制的过程中可能会发生随机的偏差，这才最终造就了生物的多样性。
为了简化问题，我们假设，DNA 在复制的时候可能出现的偏差是（理论上，对每个碱基被复制时，都可能出现偏差）：
1. 漏掉某个脱氧核苷酸。例如把 AGGT 复制成为：AGT
2. 错码，例如把 AGGT 复制成了：AGCT
3. 重码，例如把 AGGT 复制成了：AAGGT
如果某 DNA 串 a，最少要经过 n 次出错，才能变为 DNA 串 b，则称这两个 DNA 串的距离为 n。
例如：AGGTCATATTCC 与 CGGTCATATTC 的距离为 2
你的任务是：编写程序，找到两个 DNA 串的距离。
【输入、输出格式要求】
用户先输入整数 n(n<100)，表示接下来有 2n 行数据。
接下来输入的 2n 行每 2 行表示一组要比对的 DNA。（每行数据长度<10000）
程序则输出 n 行，表示这 n 组 DNA 的距离。
例如：用户输入：
3
AGCTAAGGCCTT
AGCTAAGGCCT
AGCTAAGGCCTT
AGGCTAAGGCCTT
AGCTAAGGCCTT
AGCTTAAGGCTT
则程序应输出：
1
1
2
【注意】
请仔细调试！您的程序只有能运行出正确结果的时候才有机会得分！
在评卷时使用的输入数据与试卷中给出的实例数据可能是不同的。
请把所有函数写在同一个文件中，调试好后，拷贝到【考生文件夹】下对应题号的“解答.txt”中即可。
相关的工程文件不要拷入。
源代码中不能使用诸如绘图、Win32API、中断调用、硬件操作或与操作系统相关的 API。

允许使用 STL 类库，但不能使用 MFC 或 ATL 等非 ANSI c++标准的类库。
例如，不能使用 CString 类型（属于 MFC 类库），不能使用 randomize, random 函数（不属于 ANSI C++标准）

方块填数

“数独”是当下炙手可热的智力游戏。一般认为它的起源是“拉丁方块”，是大数学家欧拉于 1783 年发明的。
如图[1.jpg]所示：6x6 的小格被分为 6 个部分（图中用不同的颜色区分）,每个部分含有 6 个小格（以下也称为分组）。
开始的时候，某些小格中已经填写了字母（ABCDEF 之一）。需要在所有剩下的小格中补填字母。

全部填好后，必须满足如下约束：
1. 所填字母只允许是 A,B,C,D,E,F 中的某一个。
2. 每行的 6 个小格中，所填写的字母不能重复。
3. 每列的 6 个小格中，所填写的字母不能重复。
4. 每个分组（参见图中不同颜色表示）包含的 6 个小格中，所填写的字母不能重复。
为了表示上的方便，我们用下面的 6 阶方阵来表示图[1.jpg]对应的分组情况（组号为 0~5）：
000011
022013
221113
243333
244455
445555
用下面的数据表示其已有字母的填写情况：
02C
03B
05A
20D
35E
53F
很明显，第一列表示行号，第二列表示列号，第三列表示填写的字母。行号、列号都从 0 开始计算。
一种可行的填写方案（此题刚好答案唯一）为：
E F C B D A
A C E D F B
D A B E C F
F B D C A E
B D F A E C
C E A F B D
你的任务是：编写程序，对一般的拉丁方块问题求解，如果多解，要求找到所有解。
【输入、输出格式要求】

用户首先输入 6 行数据，表示拉丁方块的分组情况。
接着用户输入一个整数 n (n<36)，表示接下来的数据行数
接着输入 n 行数据，每行表示一个预先填写的字母。
程序则输出所有可能的解（各个解间的顺序不重要）。
每个解占用 7 行。
即，先输出一个整数，表示该解的序号（从 1 开始），接着输出一个 6x6 的字母方阵，表示该解。
解的字母之间用空格分开。
如果找不到任何满足条件的解，则输出“无解”
例如：用户输入：
000011
022013
221113
243333
244455
445555
6
02C
03B
05A
20D
35E
53F
则程序输出：
1
E F C B D A
A C E D F B
D A B E C F
F B D C A E
B D F A E C
C E A F B D
再如，用户输入：
001111
002113
022243
022443
544433
555553
7
04B
05A
13D
14C
24E

50C
51A
则程序输出：
1
D C E F B A
E F A D C B
A B F C E D
B E D A F C
F D C B A E
C A B E D F
2
D C E F B A
E F A D C B
A D F B E C
B E C A F D
F B D C A E
C A B E D F
3
D C F E B A
A E B D C F
F D A C E B
B F E A D C
E B C F A D
C A D B F E
4
D C F E B A
B E A D C F
A D C F E B
F B E A D C
E F B C A D
C A D B F E
5
D C F E B A
E F A D C B
A B C F E D
B E D A F C
F D B C A E
C A E B D F
6
D C F E B A
E F A D C B
A B D F E C
B E C A F D
F D B C A E

C A E B D F
7
D C F E B A
E F A D C B
A D B F E C
B E C A F D
F B D C A E
C A E B D F
8
D C F E B A
F E A D C B
A D B C E F
B F E A D C
E B C F A D
C A D B F E
9
D C F E B A
F E A D C B
A F C B E D
B D E A F C
E B D C A F
C A B F D E

2013 年第四届蓝桥杯 C/C++程序设计本科 B 组决赛

1.猜灯谜

2.连续奇数和

3.空白格式化

4.高僧斗法

5.格子刷油漆

6.农场阳光

猜灯谜

A 村的元宵节灯会上有一迷题：
请猜谜 * 请猜谜 = 请边赏灯边猜
小明想，一定是每个汉字代表一个数字，不同的汉字代表不同的数字。
请你用计算机按小明的思路算一下，然后提交“请猜谜”三个字所代表的整数即可。

思路：遍历。

答案：897

[cpp] view plain copy

```
#include <iostream>
using namespace std;
int main()
{
    for(int a=1; a<=9; a++)
        for(int b=0; b<=9; b++)
        {
            if(b==a) continue;
            for(int c=0; c<=9; c++)
            {
                if(c==a||c==b) continue;
                for(int d=0; d<=9; d++)
                {
                    if(d==a||d==b||d==c) continue;
                    for(int e=0; e<=9; e++)
                    {
                        if(e==a||e==b||e==c||e==d) continue;
                        for(int f=0; f<=9; f++)
                        {
                            if(f==a||f==b||f==c||f==d||f==e) continue;
                            int t=100*a+10*b+c;
                            int tt=100000*a+10000*d+1000*e+100*f+10*d+b;
                            if(t*t==tt)
                                cout<<a<<b<<c<<endl;
                        }
                    }
                }
            }
        }
    return 0;
}
```

连续奇数和

小明看到一本书上写着：任何数字的立方都可以表示为连续奇数的和。
比如：
2^3 = 8 = 3 + 5
3^3 = 27 = 7 + 9 + 11
4^3 = 64 = 1 + 3 + ... + 15
虽然他没有想出怎么证明，但他想通过计算机进行验证。
请你帮助小明写出 111 的立方之连续奇数和表示法的起始数字。如果有多个表示方案，选择起始数字小的方案。

思路：奇数的等差序列 an=2*n-1，Sn=n^2，只需要找到 111^3=m^2-(n-1)^2，n 为起始的奇数项，则 2*n-1 位起始数字。

答案：371

[cpp] view plain copy
```
#include <iostream>
using namespace std;
int main()
{
	for(int i=1;i<=2000;i++)
		for(int j=i;j<=2000;j++)
	{
		if(j*j-(i-1)*(i-1)==111*111*111)
			cout<<2*i-1<<"	"<<2*j-1<<endl;
	}
	return 0;
}
```

空白格式化

本次大赛采用了全自动机器测评系统。

如果你的答案与标准答案相差了一个空格，很可能无法得分，所以要加倍谨慎！
但也不必过于惊慌。因为在有些情况下，测评系统会把你的答案进行“空白格式化”。其具体做法是：去掉所有首尾空白；中间的多个空白替换为一个空格。所谓空白指的是：空格、制表符、回车符。
以下代码实现了这个功能。仔细阅读代码，填写缺失的部分。

[cpp] view plain copy
```
void f(char* from, char* to)
{
```

```
    char* p_from = from;
    char* p_to = to;

    while(*p_from==' ' || *p_from=='\t' || *p_from=='\n') p_from++;

    do
    {
        if(*p_from==' ' || *p_from=='\t' || *p_from=='\n')
        {
            do
            {
                p_from++;
            }
            while(*p_from==' ' || *p_from=='\t' || *p_from=='\n');
            if(____________________) *p_to++ = ' ';  //填空位置
        }
    }
    while(*p_to++ = *p_from++);
}
```

思路：当中间的多个空白替换为一个空格，并要判断是否判断到'\0'，所以就是填空位置表达的意思。

答案：*p_from

[cpp] view plain copy

```
void f(char* from, char* to)
{
    char* p_from = from;
    char* p_to = to;

    while(*p_from==' ' || *p_from=='\t' || *p_from=='\n') p_from++;

    do
    {
        if(*p_from==' ' || *p_from=='\t' || *p_from=='\n')
        {
            do
            {
                p_from++;
            }
            while(*p_from==' ' || *p_from=='\t' || *p_from=='\n');
            if(*p_from) *p_to++ = ' ';  //填空位置
        }
    }
```

```
    while(*p_to++ = *p_from++);
}
```

高僧斗法

古时丧葬活动中经常请高僧做法事。仪式结束后，有时会有“高僧斗法”的趣味节目，以舒缓压抑的气氛。

节目大略步骤为：先用粮食（一般是稻米）在地上“画”出若干级台阶（表示 N 级浮屠）。又有若干小和尚随机地“站”在某个台阶上。最高一级台阶必须站人，其它任意。(如图 1 所示)

两位参加游戏的法师分别指挥某个小和尚向上走任意多级的台阶，但会被站在高级台阶上的小和尚阻挡，不能越过。两个小和尚也不能站在同一台阶，也不能向低级台阶移动。
两法师轮流发出指令，最后所有小和尚必然会都挤在高段台阶，再也不能向上移动。轮到哪个法师指挥时无法继续移动，则游戏结束，该法师认输。
对于已知的台阶数和小和尚的分布位置，请你计算先发指令的法师该如何决策才能保证胜出。
输入数据为一行用空格分开的 N 个整数，表示小和尚的位置。台阶序号从 1 算起，所以最后一个小和尚的位置即是台阶的总数。(N<100，台阶总数<1000)
输出为一行用空格分开的两个整数: A B，表示把 A 位置的小和尚移动到 B 位置。若有多个解，输出 A 值较小的解，若无解则输出-1。

例如：
用户输入：
1 5 9
则程序输出：
1 4
再如：
用户输入：
1 5 8 10
则程序输出：
1 3
资源约定：
峰值内存消耗 < 64M
CPU 消耗 < 1000ms
请严格按要求输出，不要画蛇添足地打印类似：“请您输入...” 的多余内容。
所有代码放在同一个源文件中，调试通过后，拷贝提交该源码。
注意: main 函数需要返回 0
注意: 只使用 ANSI C/ANSI C++ 标准，不要调用依赖于编译环境或操作系统的特殊函数。
注意: 所有依赖的函数必须明确地在源文件中 #include <xxx>， 不能通过工程设置而省略

常用头文件。

参考博客：http://blog.csdn.net/mymilkbottles/article/details/51362703

[cpp] view plain copy

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

const int maxn=1005;
int a[maxn];
int b[maxn];
int c[maxn];
int d[105];
char str[maxn<<1];

bool solve(int n)
{
	memset(b,0,sizeof(b));
	int coun=0;
	for(int i=1; i<=n; ++i)
	{
		if(a[i])
			d[coun++]=i;
	}
	d[coun]=d[coun-1]+1;
	int ans=0;
	for(int i=1; i<=coun; i+=2)
	{
		ans^=(d[i]-d[i-1]-1);
	}
	return ans==0;
}

int main()
{
	gets(str);
	int len=strlen(str);
	int coun=0;
	for(int i=0; i<len;)
	{
		while(str[i]<'0'||str[i]>'9')
		{
			++i;
```

```
        }

        int t=0;
        for(; i<len; ++i)
        {
            if(str[i]>='0'&&str[i]<='9')
            {
                t=t*10+str[i]-'0';
            }
            else break;
        }
        a[coun++]=t;
    }
    int n=a[coun-1];
    a[coun]=a[coun-1]+1;
    int ans=0;
    for(int i=1; i<=coun; i+=2)
    {
        b[i]=a[i]-a[i-1]-1;
        ans^=b[i];
    }
    if(!ans)
    {
        printf("-1\n");
    }
    else
    {
        memset(c,0,sizeof(c));
        for(int i=0; i<coun; ++i)
        {
            c[a[i]]=1;
        }
        bool ans=true;
        for(int i=1; i<=n&&ans; ++i)
        {
            memcpy(a,c,sizeof(c));
            for(int j=i+1; j<=n&&ans; ++j)
            {
                if(!a[j])
                {
                    a[i]=0;
                    a[j]=1;
                    if(solve(n))
                    {
```

```
                                printf("%d %d\n",i,j);
                                ans=false;
                                break;
                            }
                            a[i]=1;
                            a[j]=0;
                        }
                        else break;
                    }
                }
            }
        return 0;
}
```

格子刷油漆

X 国的一段古城墙的顶端可以看成 2*N 个格子组成的矩形（如图 1 所示），现需要把这些格子刷上保护漆。
你可以从任意一个格子刷起，刷完一格，可以移动到和它相邻的格子（对角相邻也算数），但不能移动到较远的格子（因为油漆未干不能踩 !）
比如：a d b c e f 就是合格的刷漆顺序。
c e f d a b 是另一种合适的方案。
当已知 N 时，求总的方案数。当 N 较大时，结果会迅速增大，请把结果对 1000000007 (十亿零七) 取模。
输入数据为一个正整数（不大于 1000）
输出数据为一个正整数。

例如：

用户输入：
2
程序应该输出：
24
再例如：
用户输入：
3
程序应该输出：
96
再例如：
用户输入：
22
程序应该输出：
359635897

资源约定：
峰值内存消耗 < 64M
CPU 消耗 < 1000ms
请严格按要求输出，不要画蛇添足地打印类似：“请您输入...”的多余内容。
所有代码放在同一个源文件中，调试通过后，拷贝提交该源码。
注意: main 函数需要返回 0
注意: 只使用 ANSI C/ANSI C++ 标准，不要调用依赖于编译环境或操作系统的特殊函数。
注意: 所有依赖的函数必须明确地在源文件中 #include <xxx>， 不能通过工程设置而省略常用头文件。

思路：暴力，会超时，但是可以过一过小数据。采用 dp 解决。

参考博客：http://blog.csdn.net/u012629369/article/details/21172165

http://blog.csdn.net/y990041769/article/details/21243895

暴力代码：

[cpp] view plain copy

```
#include <iostream>
#include <string.h>
using namespace std;

int n,sum=0,vis[2][1005];
//上  下  左 右  左上 右上 左下 右下
int dir[8][2]= {-1,0,1,0,0,-1,0,1,-1,-1,-1,1,1,-1,1,1};

void dfs(int x,int y,int cou)
{
    if(cou==2*n)
    {
        sum=(sum+1)%MOD;
        return;
    }
    for(int i=0; i<8; i++)
    {
        int xx=x+dir[i][0];
        int yy=y+dir[i][1];
        if(xx>=0&&xx<2&&yy>=0&&yy<n&&!vis[xx][yy])
        {
            vis[xx][yy]=1;
```

```
                dfs(xx,yy,cou+1);
                vis[xx][yy]=0;
            }
        }
        return ;
}

int main()
{
        cin>>n;
        sum=0;
        for(int i=0; i<2; i++)
            for(int j=0; j<n; j++)
            {
                memset(vis,0,sizeof(vis));
                vis[i][j]=1;
                dfs(i,j,1);
            }
        cout<<sum%MOD<<endl;
        return 0;
}
```

dp：

[cpp] view plain copy

```
//格子刷油漆

#include <stdio.h>

#define MAX_NUM 1000000007

__int64 a[1001] = {0}, b[1001] = {0};   //a[]为从角落的一个格子开始刷的总可能数，  b[]为从角落的一个格子开始刷最终回到起始格子的正下方

int main()
{
        int n, i;

        __int64 sum;

        scanf("%d", &n);

        b[1] = 1;                                   //初始化 b[1]，显而易见 b[1] = 1

        for(i = 2; i <= n; i++)                     //因为要回到起点下方，  所以每列格子都有一来一
```

回两个， 除第一列， b[i] = 2*b[i-1]

```
    {
        b[i] = (b[i-1] * 2) % MAX_NUM;
    }

    a[1] = 1;                                   //初始化一下 a[1],a[2]，因为递推时要用到（至于a[1,2]怎样得到的，自己数吧）
    a[2] = 6;

    for(i = 3; i <= n; i++)                     //a[i]可以由三部分得来，1，i=1 时先走起点下方的格子然后的落点有两种（后面就和 a[i]一样了）。2，最终回到起点正下方（即 b[i]）。3，由 i-2 列辗转的来（如图）
    {
        a[i] = (2 * a[i-1] + b[i] + 4 * a[i-2]) % MAX_NUM;
    }

    sum = 4 * a[n];                             //sum 是四个角为起点的情况和中间为起点的情况之和

    for(i = 2; i <= n-1; i++)                   //中间为起点
    {
        sum = (sum + 2*2*a[n-i]*2*b[i-1] + 2*2*a[i-1]*2*b[n-i]) % MAX_NUM;   //中间为起点的情况可分为先向前和先向后两种， 因为必须经过正下方的格子所以必须有一半是 b[i-1]，一半是 a[i-1]得来
    }
    printf("%I64d\n", sum);
    return 0;
}
```

农场阳光

X 星球十分特殊，它的自转速度与公转速度相同，所以阳光总是以固定的角度照射。
最近，X 星球为发展星际旅游业，把空间位置出租给 Y 国游客来晒太阳。每个租位是漂浮在空中的圆盘形彩云（圆盘与地面平行）。当然，这会遮挡住部分阳光，被遮挡的土地植物无法生长。
本题的任务是计算某个农场宜于作物生长的土地面积有多大。
输入数据的第一行包含两个整数 a, b，表示某农场的长和宽分别是 a 和 b，此时，该农场的范围是由坐标(0, 0, 0), (a, 0, 0), (a, b, 0), (0, b, 0)围成的矩形区域。
第二行包含一个实数 g，表示阳光照射的角度。简单起见，我们假设阳光光线是垂直于农场的宽的，此时正好和农场的长的夹角是 g 度，此时，空间中的一点(x, y, z)在地面的投影点应该是(x + z * ctg(g 度), y, 0)，其中 ctg(g 度)表示 g 度对应的余切值。
第三行包含一个非负整数 n，表示空中租位个数。
接下来 n 行，描述每个租位。其中第 i 行包含 4 个整数 xi, yi, zi, ri，表示第 i 个租位彩云的圆心在(xi, yi, zi)位置，圆半径为 ri。

要求输出一个实数，四舍五入保留两位有效数字，表示农场里能长庄稼的土地的面积。

例如：
用户输入：
10 10
90.0
1
5 5 10 5
程序应该输出：
21.46
再例如：
用户输入：
8 8
90.0
1
4 4 10 5
程序应该输出：
1.81
样例 3：
用户输入：
20 10
45.0
2
5 0 5 5
8 6 14 6
程序输出：
130.15
资源约定：
峰值内存消耗 < 64M
CPU 消耗 < 1000ms
请严格按要求输出，不要画蛇添足地打印类似：“请您输入...”的多余内容。
所有代码放在同一个源文件中，调试通过后，拷贝提交该源码。
注意: main 函数需要返回 0
注意: 只使用 ANSI C/ANSI C++ 标准，不要调用依赖于编译环境或操作系统的特殊函数。
注意: 所有依赖的函数必须明确地在源文件中 #include <xxx>， 不能通过工程设置而省略常用头文件。

2014 年第五届蓝桥杯 C/C++程序设计本科 B 组决赛

1.年龄巧合

2.出栈次序

3.信号匹配

4.生物芯片

5.Log 大侠

6.殖民地

年龄巧合

小明和他的表弟一起去看电影，有人问他们的年龄。小明说：今年是我们的幸运年啊。我出生年份的四位数字加起来刚好是我的年龄。表弟的也是如此。已知今年是 2014 年，并且，小明说的年龄指的是周岁。
请推断并填写出小明的出生年份。

思路：遍历。

答案：1988

[cpp] view plain copy

```
#include <iostream>
using namespace std;
int main()
{
	for(int i=1;i<=9;i++)
	for(int j=0;j<=9;j++)
	for(int k=0;k<=9;k++)
	for(int l=0;l<=9;l++)
	{
		int t=i*1000+j*100+k*10+l;
		if(2014-t==(i+j+k+l))
			cout<<i<<j<<k<<l<<endl;
	}

	return 0;
}
```

出栈次序

X 星球特别讲究秩序，所有道路都是单行线。一个甲壳虫车队，共 16 辆车，按照编号先后发车，夹在其它车流中，缓缓前行。

路边有个死胡同，只能容一辆车通过，是临时的检查站，如图【p1.png】所示。
X 星球太死板，要求每辆路过的车必须进入检查站，也可能不检查就放行，也可能仔细检查。
如果车辆进入检查站和离开的次序可以任意交错。那么，该车队再次上路后，可能的次序有多少种？
为了方便起见，假设检查站可容纳任意数量的汽车。
显然，如果车队只有 1 辆车，可能次序 1 种；2 辆车可能次序 2 种；3 辆车可能次序 5 种。
现在足足有 16 辆车啊，亲！需要你计算出可能次序的数目。

题意：求 n 个元素的出栈情况有多少种。

思路：

方法一：

我们把 n 个元素的出栈个数的记为 f(n)，那么对于 1,2,3，我们很容易得出：

f(1)= 1　　//即 1

f(2)= 2　　//即 12、21

f(3)= 5　　//即 123、132、213、321、231

然后我们来考虑 f(4)，我们给 4 个元素编号为 a,b,c,d，那么考虑：元素 a 只可能出现在 1 号位置，2 号位置，3 号位置和 4 号位置(很容易理解，一共就 4 个位置，比如 abcd,元素 a 就在 1 号位置)。

分析：

1) 如果元素 a 在 1 号位置，那么只可能 a 进栈，马上出栈，此时还剩元素 b、c、d 等待操作，就是子问题 f(3)；

2) 如果元素 a 在 2 号位置，那么一定有一个元素比 a 先出栈，即有 f(1)种可能顺序（只能是 b），还剩 c、d，即 f(2)，　　根据乘法原理，一共的顺序个数为 f(1)* f(2)；

3) 如果元素 a 在 3 号位置，那么一定有两个元素比 1 先出栈，即有 f(2)种可能顺序（只能是 b、c），还剩 d，即 f(1)，

根据乘法原理，一共的顺序个数为 f(2) * f(1)；

4) 如果元素 a 在 4 号位置，那么一定是 a 先进栈，最后出栈，那么元素 b、c、d 的出栈顺序即是此小问题的解，即　　f(3)；

结合所有情况，即 f(4) = f(3) +f(2) * f(1) + f(1) * f(2) + f(3);

为了规整化，我们定义 f(0) = 1；于是 f(4)可以重新写为：

f(4) = f(0)*f(3) + f(1)*f(2) + f(2) * f(1)+ f(3)*f(0)

然后我们推广到 n，推广思路和 n=4 时完全一样，于是我们可以得到：

f(n) = f(0)*f(n-1) + f(1)*f(n-2) + … +f(n-1)*f(0)

即

方法二：Catalan 数：C(2n,n)/(n+1) （C(2n,n)表示 2n 里取 n）或者 C(2n,n)-C(2n,n-1)都可以解决。

参考博客：http://blog.csdn.net/zyearn/article/details/7758716

答案：35357670

[cpp] view plain copy

```
#include <iostream>
#include <string.h>
using namespace std;
int main()
{
    int f[20];
    memset(f,0,sizeof(f));
    f[0]=1;
    f[1]=1;
    f[2]=2;
    f[3]=5;
    for(int i=4; i<=16; i++)
    {
        for(int j=0; j<=i-1; j++)
            f[i]+=f[j]*f[i-1-j];
    }
    cout<<f[16]<<endl;
    return 0;
}
```

信号匹配

从 X 星球接收了一个数字信号序列。

现有一个已知的样板序列。需要在信号序列中查找它首次出现的位置。这类似于串的匹配操作。
如果信号序列较长，样板序列中重复数字较多，就应当注意比较的策略了。可以仿照串的 KMP 算法，进行无回溯的匹配。这种匹配方法的关键是构造 next 数组。
next[i] 表示第 i 项比较失配时，样板序列向右滑动，需要重新比较的项的序号。如果为-1，表示母序列可以进入失配位置的下一个位置进行新的比较。

下面的代码实现了这个功能，请仔细阅读源码，推断划线位置缺失的代码。

[cpp] view plain copy

```
// 生成 next 数组
int* make_next(int pa[], int pn)
{
    int* next = (int*)malloc(sizeof(int)*pn);
    next[0] = -1;
    int j = 0;
    int k = -1;
    while(j < pn-1){
        if(k==-1 || pa[j]==pa[k]){
            j++;
            k++;
            next[j] = k;
        }
        else
            k = next[k];
    }

    return next;
}

// da 中搜索 pa，  da 的长度为 an, pa 的长度为 pn
int find(int da[], int an, int pa[], int pn)
{
    int rst = -1;
    int* next = make_next(pa, pn);
    int i=0;   // da 中的指针
    int j=0;   // pa 中的指针
    int n = 0;
    while(i<an){
        n++;
        if(da[i]==pa[j] || j==-1){
            i++;
            j++;
```

```
            }
            else
                _____________________;   //填空位置

            if(j==pn) {
                rst = i-pn;
                break;
            }
        }

        free(next);

        return rst;
}

int main()
{
        int da[] = {1,2,1,2,1,1,2,1,2,1,1,2,1,1,2,1,1,2,1,2,1,1,2,1,2,1,1,2,1,1,2,1,2,1,1,2,1,2,3};
        int pa[] = {1,2,1,1,2,1,1,1,2};

        int n = find(da, sizeof(da)/sizeof(int), pa, sizeof(pa)/sizeof(int));
        printf("%d\n", n);

        return 0;
}
```

思路：kmp

答案：j=next[j]

[cpp] view plain copy

```
//  生成 next 数组
int* make_next(int pa[], int pn)
{
        int* next = (int*)malloc(sizeof(int)*pn);
        next[0] = -1;
        int j = 0;
        int k = -1;
        while(j < pn-1){
            if(k==-1 || pa[j]==pa[k]){
                j++;
                k++;
                next[j] = k;
            }
            else
```

```
            k = next[k];
        }

        return next;
}

// da 中搜索 pa,  da 的长度为 an, pa 的长度为 pn
int find(int da[], int an, int pa[], int pn)
{
        int rst = -1;
        int* next = make_next(pa, pn);
        int i=0;   // da 中的指针
        int j=0;   // pa 中的指针
        int n = 0;
        while(i<an){
                n++;
                if(da[i]==pa[j] || j==-1){
                        i++;
                        j++;
                }
                else
                        j=next[j];   //填空位置

                if(j==pn) {
                        rst = i-pn;
                        break;
                }
        }

        free(next);

        return rst;
}

int main()
{
        int da[] = {1,2,1,2,1,1,2,1,2,1,1,2,1,1,2,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,2,1,1,2,1,1,2,1,2,3};
        int pa[] = {1,2,1,1,2,1,1,1,2};

        int n = find(da, sizeof(da)/sizeof(int), pa, sizeof(pa)/sizeof(int));
        printf("%d\n", n);

        return 0;
}
```

生物芯片

X 博士正在研究一种生物芯片，其逻辑密集度、容量都远远高于普通的半导体芯片。
博士在芯片中设计了 n 个微型光源，每个光源操作一次就会改变其状态，即：点亮转为关闭，或关闭转为点亮。
这些光源的编号从 1 到 n，开始的时候所有光源都是关闭的。
博士计划在芯片上执行如下动作：
所有编号为 2 的倍数的光源操作一次，也就是把 2 4 6 8 ... 等序号光源打开
所有编号为 3 的倍数的光源操作一次，也就是对 3 6 9 ... 等序号光源操作，注意此时 6 号光源又关闭了。
所有编号为 4 的倍数的光源操作一次。
.....
直到编号为 n 的倍数的光源操作一次。
X 博士想知道：经过这些操作后，某个区间中的哪些光源是点亮的。
【输入格式】
3 个用空格分开的整数：N L R (L<R<N<10^15) N 表示光源数，L 表示区间的左边界，R 表示区间的右边界。
【输出格式】
输出 1 个整数，表示经过所有操作后，[L,R] 区间中有多少个光源是点亮的。
例如：
输入：
5 2 3
程序应该输出：
2
再例如：
输入：
10 3 6
程序应该输出：
3
资源约定：
峰值内存消耗 < 256M
CPU 消耗 < 1000ms
请严格按要求输出，不要画蛇添足地打印类似：“请您输入...”的多余内容。
所有代码放在同一个源文件中，调试通过后，拷贝提交该源码。
注意: main 函数需要返回 0
注意: 只使用 ANSI C/ANSI C++ 标准，不要调用依赖于编译环境或操作系统的特殊函数。
注意: 所有依赖的函数必须明确地在源文件中 #include <xxx>， 不能通过工程设置而省略常用头文件。
提交时，注意选择所期望的编译器类型。

思路：完全平方数的因子个数为奇数。

一个光源最后是打开的，当且仅当他被操作了奇数次，也即是说该数的因子数为奇数，即该数为完全平方数，但是这题是不算 1 的，故最终答案是，若该数不是完全平方数，则最后为打开状态。

[cpp] view plain copy

```
#include <iostream>
#include <cmath>
using namespace std;
int main()
{
	long long int n,l,r;
	cin>>n>>l>>r;
	long long int ll=(int)(sqrt(l-1))+1;
	long long int rr=(int)(sqrt(r));
	if(rr>=ll)
		cout<<r-l+1-(rr-ll+1);
	else
		cout<<r-l+1;
	return 0;
}
```

Log 大侠

atm 参加了速算训练班，经过刻苦修炼，对以 2 为底的对数算得飞快，人称 Log 大侠。
一天，Log 大侠的好友 drd 有一些整数序列需要变换，Log 大侠正好施展法力…
变换的规则是： 对其某个子序列的每个整数变为: [log_2 (x) + 1] 其中 [] 表示向下取整，就是对每个数字求以 2 为底的对数，然后取下整。
例如对序列 3 4 2 操作一次后，这个序列会变成 2 3 2。
drd 需要知道，每次这样操作后，序列的和是多少。

【输入格式】

第一行两个正整数 n m 。
第二行 n 个数，表示整数序列，都是正数。
接下来 m 行，每行两个数 L R 表示 atm 这次操作的是区间 [L, R]，数列序号从 1 开始。
【输出格式】
输出 m 行，依次表示 atm 每做完一个操作后，整个序列的和。
例如，输入：
3 3
5 6 4
1 2
2 3

1 3
程序应该输出：
10
8
6

【数据范围】

对于 30% 的数据， n, m <= 10^3
对于 100% 的数据， n, m <= 10^5
资源约定：
峰值内存消耗 < 256M
CPU 消耗 < 1000ms
请严格按要求输出，不要画蛇添足地打印类似：“请您输入...”的多余内容。
所有代码放在同一个源文件中，调试通过后，拷贝提交该源码。
注意: main 函数需要返回 0
注意: 只使用 ANSI C/ANSI C++ 标准，不要调用依赖于编译环境或操作系统的特殊函数。
注意: 所有依赖的函数必须明确地在源文件中 #include <xxx>， 不能通过工程设置而省略常用头文件。
提交时，注意选择所期望的编译器类型。

[cpp] view plain copy
```
#include <iostream>
using namespace std;
int a[100005];

int log(int n)
{
	int sum=1,ans=0;
	while(sum<n)
	{
		sum*=2;
		ans++;
	}
	if(sum==n)
		return ans+1;
	return ans;

}

int main()
{
	int n,m,l,r,sum=0;
	cin>>n>>m;
```

```
    for(int i=1; i<=n; i++)
    {
        cin>>a[i];
        sum+=a[i];
    }
    for(int i=1; i<=m; i++)
    {
        cin>>l>>r;
        for(int j=l; j<=r; j++)
        {
            sum-=a[j];
            a[j]=log(a[j]);
            sum+=a[j];
        }
        cout<<sum<<endl;
    }

    return 0;
}
```

殖民地

带着殖民扩张的野心，Pear 和他的星际舰队登上 X 星球的某平原。为了评估这块土地的潜在价值，Pear 把它划分成了 M*N 格，每个格子上用一个整数（可正可负）表示它的价值。Pear 要做的事很简单——选择一些格子，占领这些土地，通过建立围栏把它们和其它土地隔开。对于 M*N 的格子，一共有(M+1)*N+M*(N+1)条围栏，即每个格子都有上下左右四个围栏；不在边界上的围栏被相邻的两个格子公用。大概如下图【p1.png】所示。
图中，蓝色的一段是围栏，属于格子 1 和 2；红色的一段是围栏，属于格子 3 和 4。
每个格子有一个可正可负的收益，而建围栏的代价则一定是正的。
你需要选择一些格子，然后选择一些围栏把它们围起来，使得所有选择的格子和所有没被选的格子严格的被隔开。选择的格子可以不连通，也可以有“洞”，即一个连通块中间有一些格子没选。注意，若中间有“洞”，那么根据定义，“洞”和连通块也必须被隔开。

Pear 的目标很明确，花最小的代价，获得最大的收益。

【输入数据】
输入第一行两个正整数 M N，表示行数和列数。
接下来 M 行，每行 N 个整数，构成矩阵 A，A[i,j]表示第 i 行第 j 列格子的价值。
接下来 M+1 行，每行 N 个整数，构成矩阵 B，B[i,j]表示第 i 行第 j 列上方的围栏建立代价。特别的，B[M+1,j]表示第 M 行第 j 列下方的围栏建立代价。
接下来 M 行，每行 N+1 个整数，构成矩阵 C，C[i,j]表示第 i 行第 j 列左方的围栏建立代价。特别的，C[i,N+1]表示第 i 行第 N 列右方的围栏建立代价。

【输出数据】
一行。只有一个正整数，表示最大收益。
【输入样例 1】
3 3
65 -6 -11
15 65 32
-8 5 66
4 1 6
7 3 11
23 21 22
5 25 22
26 1 1 13
16 3 3 4
6 3 1 2
程序应当输出：
123
【输入样例 2】
6 6
72 2 -7 1 43 -12
74 74 -14 35 5 3
31 71 -12 70 38 66
40 -6 8 52 3 78
50 11 62 20 -6 61
76 55 67 28 -19 68
25 4 5 8 30 5
9 20 29 20 6 18
3 19 20 11 5 15
10 3 19 23 6 24
27 8 16 10 5 22
28 14 1 5 1 24
2 13 15 17 23 28
24 11 27 16 12 13 27
19 15 21 6 21 11 5
2 3 1 11 10 20 9
8 28 1 21 9 5 7
16 20 26 2 22 5 12
30 27 16 26 9 6 23
程序应当输出
870
【数据范围】
对于 20%的数据，M,N<=4
对于 50%的数据，M,N<=15
对于 100%的数据，M,N<=200
A、B、C 数组（所有的涉及到的格子、围栏输入数据）绝对值均不超过 1000。根据题意，

A 数组可正可负，B、C 数组均为正整数。
资源约定：
峰值内存消耗 < 256M
CPU 消耗 < 3000ms
请严格按要求输出，不要画蛇添足地打印类似：“请您输入...”的多余内容。
所有代码放在同一个源文件中，调试通过后，拷贝提交该源码。
注意: main 函数需要返回 0
注意: 只使用 ANSI C/ANSI C++ 标准，不要调用依赖于编译环境或操作系统的特殊函数。
注意: 所有依赖的函数必须明确地在源文件中 #include <xxx>， 不能通过工程设置而省略常用头文件。
提交时，注意选择所期望的编译器类型。

2015 年第六届蓝桥杯 C/C++程序设计本科 B 组决赛

1.积分之谜

2.完美正方形

3.关联账户

4.密文搜索

5.居民集会

6.模型染色

积分之迷

小明开了个网上商店，卖风铃。共有 3 个品牌：A，B，C。
为了促销，每件商品都会返固定的积分。
小明开业第一天收到了三笔订单：
第一笔：3 个 A + 7 个 B + 1 个 C，共返积分：315
第二笔：4 个 A + 10 个 B + 1 个 C，共返积分：420
第三笔：A + B + C，共返积分....
你能算出第三笔订单需要返积分多少吗？

答案：105

[cpp] view plain copy

```
#include <iostream>
#include <cmath>
using namespace std;
int main()
{
    for(int a=0; a<=110; a++)
        for(int b=0; b<=110; b++)
            for(int c=0; c<=110; c++)
            {
                if( (3*a+7*b+c==315)&&(4*a+10*b+c==420))
                    cout<<a+b+c<<endl;
            }
    return 0;
}
```

完美正方形

如果一些边长互不相同的正方形，可以恰好拼出一个更大的正方形，则称其为完美正方形。
历史上，人们花了很久才找到了若干完美正方形。比如：如下边长的 22 个正方形
2 3 4 6 7 8 12 13 14 15 16 17 18 21 22 23 24 26 27 28 50 60
如【图 1.png】那样组合，就是一种解法。此时，
紧贴上边沿的是：60 50
紧贴下边沿的是：26 28 17 21 18
22 阶完美正方形一共有 8 种。下面的组合是另一种：
2 5 9 11 16 17 19 21 22 24 26 30 31 33 35 36 41 46 47 50 52 61
如果告诉你该方案紧贴着上边沿的是从左到右依次为：47 46 61，
你能计算出紧贴着下边沿的是哪几个正方形吗？
请提交紧贴着下边沿的正方形的边长，从左到右，用空格分开。

答案：50 33 30 41

参考博客：http://blog.csdn.net/lonverce/article/details/51436195

[cpp] view plain copy

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

// 全局变量
namespace Global
{
const int MaxS = 46+47+61;  // 正方形的边长
```

```
int All[] = {2,5,9,11,16,17,19,21,22,
                    24,26,30,31,33,35,36,41,50,52
                  };// 备选正方形边长
int Length = sizeof(All)/sizeof(All[0]);// 备选正方形的数量
int Square[MaxS][MaxS]= {0};      // 表示结果的完美正方形
}
// 线段树结点
struct TNode
{
    int L, R;               // [L, R]
    int height;             // 段高度( -1 时表示该段不平滑,高度无意义 )
    int inc;                // 高度增量
    // 计算宽度
    inline int Width()const
    {
        return R-L+1;
    }
    // 计算中点
    inline int Mid()const
    {
        return (L+R)>>1;
    }
} NSet[1024];

inline int LSon( int i )
{
    return i<<1;
}
inline int RSon( int i )
{
    return LSon(i)+1;
}
inline int Parent( int i)
{
    return i>>1;
}

// 建立线段树
void BuildTree( int i, int L, int R )
{
    TNode* p = NSet+i;
    p->L = L;
    p->R = R;
    p->height = 0;
```

```
    p->inc = 0;

    if( L < R )
    {
        int m = p->Mid();
        BuildTree( LSon(i), L, m );
        BuildTree( RSon(i), m+1, R);
    }
}
// 增量的向下传递调整
// 注: NSet[i]必须为平滑区间
inline void Adjust( int i )
{
    if( NSet[i].inc != 0 )
    {
        NSet[i].height += NSet[i].inc;
        if( NSet[i].L != NSet[i].R )
        {
            NSet[LSon(i)].inc += NSet[i].inc;
            NSet[RSon(i)].inc += NSet[i].inc;
        }
        NSet[i].inc = 0;
    }
}

// 将区间[L,R]的高度统一增加(或减少)inc
// 注: 区间 [L, R] 必须平滑
void Add(int i, int L, int R, int inc )
{
    TNode* p = NSet+i;
    if( p->L == L && p->R == R )
    {
        // 操作后平滑性不变
        p->inc += inc;
        Adjust(i);
    }
    else
    {
        int m = p->Mid();

        if( p->height != -1 ) // 表示该段本是平滑的
        {
            Adjust(i);
            p->height = -1;     // 现在开始该段不再平滑
```

```
        }

        int ls = LSon(i), rs = RSon(i);
        if( R <= m )
            Add( ls, L, R, inc );
        else if( L > m )
            Add( rs, L, R, inc );
        else
        {
            Add( ls, L, m, inc );
            Add( rs, m+1, R, inc );
        }

        // 平滑性恢复检验
        if(NSet[ls].height != -1)
        {
            Adjust(ls);
            if(NSet[rs].height != -1)
            {
                Adjust(rs);
                if(NSet[ls].height == NSet[rs].height)
                    p->height = NSet[ls].height;
            }
        }
    } // end else
}
// 获取最低段区间
void GetLowestInterval(int i, TNode* prev, TNode* lowest )
{
    TNode* p = NSet+i;
    if( p->height == -1 )
    {
        GetLowestInterval( LSon(i), prev, lowest );
        GetLowestInterval( RSon(i), prev, lowest );
    }
    else
    {
        prev->R = p->R;
        // 检验等高区间连续性
        if( p->height != prev->height )
        {
            prev->L = p->L;
            prev->height = p->height;
        }
```

```
            // 更新最低区间
            if( prev->height <= lowest->height )
            {
                lowest->height = prev->height;
                lowest->L = prev->L;
                lowest->R = prev->R;
            }
        }
}
bool Dfs( int L, int R ,int H )
{
        if( L > R )
        {
            TNode p, m;
            m.height = 1000;
            m.L = m.R = 1000;
            p.height = 1000;
            p.L = p.R = -1;

            GetLowestInterval(1, &p, &m);
            L = m.L;
            R = m.R;
            H = m.height;
        }

        int Width = R-L+1, temp=0;
        if( Width == Global::MaxS && H != 0 ) return true;

        for( int i = Global::Length; i-- > 0 ;)
        {
            temp = Global::All[i];
            if( temp != 0 && temp <= Width )
            {
                Global::All[i] = 0;
                Add( 1, L, L+temp-1, temp);
                Global::Square[ H+temp-1 ][ L ] = temp;
                if( Dfs( L+temp, R, H ) == true ) return true;
                Add( 1, L, L+temp-1, -temp);
                Global::All[i] = temp;
            }
        }
        return false;
}
```

```
int main(int argc, char** argv)
{
	BuildTree( 1, 0, Global::MaxS-1 );

	Add( 1, 0, 46, 47);
	Add( 1, 47, 92, 46);
	Add( 1, 93, 153, 61);

	if( Dfs( 1, 0, 0 ) )
	{
		for( int i = 0,t=0; i < Global::MaxS; i+=t)
		{
			t = Global::Square[ Global::MaxS-1 ][i];
			cout << t << ' ';
		}
	}
	return 0;
}
```

关联账户

为增大反腐力度，某地警方专门支队，对若干银行账户展开调查。
如果两个账户间发生过转账，则认为有关联。如果 a,b 间有关联，b,c 间有关联，则认为 a,c 间也有关联。
对于调查范围内的 n 个账户（编号 0 到 n-1），警方已知道 m 条因转账引起的直接关联。
现在希望知道任意给定的两个账户，求出它们间是否有关联。有关联的输出 1，没有关联输出 0
小明给出了如下的解决方案：

[csharp] view plain copy

```
#include <stdio.h>
#define N 100

int connected(int* m, int p, int q)
{
	return m[p]==m[q]? 1 : 0;
}

void link(int* m, int p, int q)
{
	int i;
	if(connected(m,p,q)) return;
```

```
    int pID = m[p];
    int qID = m[q];
    for(i=0; i<N; i++) _____;  //填空位置
}

int main()
{
    int m[N];
    int i;
    for(i=0; i<N; i++) m[i] = i; //初始状态，每个节点自成一个连通域
    link(m,0,1); //添加两个账户间的转账关联
    link(m,1,2);
    link(m,3,4);
    link(m,5,6);
    link(m,6,7);
    link(m,8,9);
    link(m,3,7);

    printf("%d ", connected(m,4,7));
    printf("%d ", connected(m,4,5));
    printf("%d ", connected(m,7,9));
    printf("%d ", connected(m,9,2));
    return 0;
}
```

思路：并查集的连接。

答案：if(m[i]==qID) m[i]=pID

[cpp] view plain copy

```
#include <stdio.h>
#define N 100

int connected(int* m, int p, int q)
{
    return m[p]==m[q]? 1 : 0;
}

void link(int* m, int p, int q)
{
    int i;
    if(connected(m,p,q)) return;
    int pID = m[p];
    int qID = m[q];
```

```
    for(i=0; i<N; i++)  if(m[i]==qID) m[i]=pID;  //填空位置
}

int main()
{
    int m[N];
    int i;
    for(i=0; i<N; i++) m[i] = i; //初始状态，每个节点自成一个连通域
    link(m,0,1); //添加两个账户间的转账关联
    link(m,1,2);
    link(m,3,4);
    link(m,5,6);
    link(m,6,7);
    link(m,8,9);
    link(m,3,7);

    printf("%d ", connected(m,4,7));
    printf("%d ", connected(m,4,5));
    printf("%d ", connected(m,7,9));
    printf("%d ", connected(m,9,2));
    return 0;
}
```

密文搜索

福尔摩斯从 X 星收到一份资料，全部是小写字母组成。
他的助手提供了另一份资料：许多长度为 8 的密码列表。
福尔摩斯发现，这些密码是被打乱后隐藏在先前那份资料中的。
请你编写一个程序，从第一份资料中搜索可能隐藏密码的位置。要考虑密码的所有排列可能性。
数据格式：
输入第一行：一个字符串 s，全部由小写字母组成，长度小于 1024*1024
紧接着一行是一个整数 n,表示以下有 n 行密码，1<=n<=1000
紧接着是 n 行字符串，都是小写字母组成，长度都为 8
要求输出：
一个整数，表示每行密码的所有排列在 s 中匹配次数的总和。

例如：
用户输入：
aaaabbbbaabbcccc
2
aaaabbbb
abcabccc

则程序应该输出：
4
这是因为：第一个密码匹配了 3 次，第二个密码匹配了 1 次，一共 4 次。
资源约定：
峰值内存消耗 < 512M
CPU 消耗 < 3000ms
请严格按要求输出，不要画蛇添足地打印类似：“请您输入...”的多余内容。
所有代码放在同一个源文件中，调试通过后，拷贝提交该源码。
注意: main 函数需要返回 0
注意: 只使用 ANSI C/ANSI C++ 标准，不要调用依赖于编译环境或操作系统的特殊函数。
注意: 所有依赖的函数必须明确地在源文件中 #include <xxx>， 不能通过工程设置而省略常用头文件。
提交时，注意选择所期望的编译器类型。

思路：因为要求每行密码的所有排列在 s 中匹配次数的总和，所以可以统计每行密码中所包含的各个字母的个数，然后选取主串中的长度为 8 的区间，比较是否与密码中包含的各个字母的个数相等。在选取主串中长度为 8 的区间时，实际上只有只有主串长度-7 个长度为 8 的区间。

例如: 0 1 2 3 4 5 6 7 8 9 10 主串长度为 11

则长度为 8 的区间只有 0--7 1-- 8 2--9 3--10 这 4 个，即 11-4 个。

[cpp] view plain copy

```
#include <iostream>
#include <stdio.h>
#include <cstring>
using namespace std;

int main()
{
    int n;
    char str[1005],s[10];
    int  a[1005][26],b[26];

    memset(a,0,sizeof(a));

    scanf("%s", str);

    for(int i=0; i<=(strlen(str)-8); i++)
        for(int j=i; j<=i+7; j++)
```

```
            a[i][str[j]-'a']+=1;

    scanf("%d",&n);


    int sum=0;
    for(int k=0; k<n; k++)
    {
        int flag;
        memset(b,0,sizeof(b));

        scanf("%s", s);

        for(int i=0; i<strlen(s); i++)
            b[s[i]-'a']+=1;

        for(int i=0; i<=strlen(str)-8; i++)
        {
            flag=1;
            for(int j=0; j<26; j++)
            {
                if(a[i][j]!=b[j])
                {
                    flag=0;
                    break;
                }
            }
            if(flag==1)
                sum++;
        }
    }
    cout<<sum<<endl;
    return 0;
}
```

居民集会

蓝桥村的居民都生活在一条公路的边上，公路的长度为 L，每户家庭的位置都用这户家庭到公路的起点的距离来计算，第 i 户家庭距起点的距离为 di。
每年，蓝桥村都要举行一次集会。今年，由于村里的人口太多，村委会决定要在 4 个地方举行集会，其中 3 个位于公路中间，1 个位最公路的终点。
已知每户家庭都会向着远离公路起点的方向去参加集会，参加集会的路程开销为家庭内的人数 ti 与距离的乘积。

给定每户家庭的位置 di 和人数 ti，请为村委会寻找最好的集会举办地：p1, p2, p3, p4 (p1<=p2<=p3<=p4=L),使得村内所有人的路程开销和最小。
【输入格式】
输入的第一行包含两个整数 n, L，分别表示蓝桥村的家庭数和公路长度。
接下来 n 行，每行两个整数 di, ti，分别表示第 i 户家庭距离公路起点的距离和家庭中的人数。
【输出格式】
输出一行，包含一个整数，表示村内所有人路程的开销和。
【样例输入】
6 10
1 3
2 2
4 5
5 20
6 5
8 7
【样例输出】
18
【样例说明】
在距起点 2, 5, 8, 10 这 4 个地方集会，6 个家庭需要的走的距离分别为 1, 0, 1, 0, 2, 0，总的路程开销为 1*3+0*2+1*5+0*20+2*5+0*7=18。
【数据规模与约定】
对于 10%的评测数据，1<=n<=300。
对于 30%的评测数据，1<=n<=2000，1<=L<=10000，0<=di<=L，di<=di+1，0<=ti<=20。
对于 100%的评测数据，1<=n<=100000，1<=L<=1000000，0<=di<=L，di<=di+1，0<=ti<=1000000。
资源约定：
峰值内存消耗 < 512M
CPU 消耗 < 5000ms
请严格按要求输出，不要画蛇添足地打印类似：“请您输入...”的多余内容。
所有代码放在同一个源文件中，调试通过后，拷贝提交该源码。
注意: main 函数需要返回 0
注意: 只使用 ANSI C/ANSI C++ 标准，不要调用依赖于编译环境或操作系统的特殊函数。
注意: 所有依赖的函数必须明确地在源文件中 #include <xxx>， 不能通过工程设置而省略常用头文件。
提交时，注意选择所期望的编译器类型。

模型染色

在电影《超能陆战队》中，小宏可以使用他的微型机器人组合成各种各样的形状。
现在他用他的微型机器人拼成了一个大玩具给小朋友们玩。为了更加美观，他决定给玩具染色。
小宏的玩具由 n 个球型的端点和 m 段连接这些端点之间的边组成。下图给出了一个由 5 个

球型端点和 4 条边组成的玩具，看上去很像一个分子的球棍模型。
由于小宏的微型机器人很灵活，这些球型端点可以在空间中任意移动，同时连接相邻两个球型端点的边可以任意的伸缩，这样一个玩具可以变换出不同的形状。在变换的过程中，边不会增加，也不会减少。
小宏想给他的玩具染上不超过 k 种颜色，这样玩具看上去会不一样。如果通过变换可以使得玩具变成完全相同的颜色模式，则认为是本质相同的染色。现在小宏想知道，可能有多少种本质不同的染色。
【输入格式】
输入的第一行包含三个整数 n, m, k,
分别表示小宏的玩具上的端点数、边数和小宏可能使用的颜色数。端点从 1 到 n 编号。
接下来 m 行每行两个整数 a, b，表示第 a 个端点和第 b 个端点之间有一条边。输入保证不会出现两条相同的边。
【输出格式】
输出一行，表示本质不同的染色的方案数。由于方案数可能很多，请输入方案数除 10007 的余数。
【样例输入】
3 2 2
1 2
3 2
【样例输出】
6
【样例说明】
令(a, b, c)表示第一个端点染成 a，第二个端点染成 b，第三个端点染成 c，则下面 6 种本质不同的染色：(1, 1, 1), (1, 1, 2), (1, 2, 1), (1, 2, 2), (2, 1, 2), (2, 2, 2)。
而(2, 1, 1)与(1, 1, 2)是本质相同的，(2, 2, 1)与(2, 1, 2)是本质相同的。
【数据规模与约定】
对于 20%的评测数据，1<=n<=5， 1<=k<=2。
对于 50%的评测数据，1<=n<=10, 1<=k<=8。
对于 100%的评测数据，1<=n<=10, 1<=m<=45, 1<=k<=30。
资源约定：
峰值内存消耗 < 512M
CPU 消耗 < 5000ms
请严格按要求输出，不要画蛇添足地打印类似：“请您输入...”的多余内容。
所有代码放在同一个源文件中，调试通过后，拷贝提交该源码。
注意: main 函数需要返回 0
注意: 只使用 ANSI C/ANSI C++ 标准，不要调用依赖于编译环境或操作系统的特殊函数。
注意: 所有依赖的函数必须明确地在源文件中 #include <xxx>， 不能通过工程设置而省略常用头文件。
提交时，注意选择所期望的编译器类型。

2016 蓝桥杯决赛试题

1.一步之遥

从昏迷中醒来，小明发现自己被关在 X 星球的废矿车里。 矿车停在平直的废弃的轨道上。他的面前是两个按钮，分别写着“F”和“B”。

小明突然记起来，这两个按钮可以控制矿车在轨道上前进和后退。 按 F，会前进 97 米。按 B 会后退 127 米。
透过昏暗的灯光，小明看到自己前方 1 米远正好有个监控探头。 他必须设法使得矿车正好停在摄像头的下方，才有机会争取同伴的援助。
或许，通过多次操作 F 和 B 可以办到。

矿车上的动力已经不太足，黄色的警示灯在默默闪烁… 每次进行 F 或 B 操作都会消耗一定的能量。
小明飞快地计算，至少要多少次操作，才能把矿车准确地停在前方 1 米远的地方。

请填写为了达成目标，最少需要操作的次数。

注意，需要提交的是一个整数，不要填写任何无关内容（比如：解释说明等）
【答案】：97

【解析】：

本题有两种思路。

1、因 97 与 127 互质，其最大公约数恰好为 1，可用拓展欧几里得算法求得特解。

同时这组特解的绝对值之和就是答案，（特解 x0 与 y0 相对于其他解最靠近 0）

详细拓展欧几里得算法 ：点击打开链接

2、直接枚举找出很多满足 97*x+127*y=1 的 x，y。然后找到最小的 x，y 绝对值之和

【代码 1】

[cpp] view plain copy

```
#include<stdio.h>
#include<math.h>
int exgcd(int a,int b,int &x,int &y)
{
    if(b==0)
    {
```

```
            x=1;y=0;
            return a;
        }
        int d=exgcd(b,a%b,y,x);
        y=y-(a/b)*x;
        return d;
}
int main()
{
        int x,y;
        exgcd(127,97,x,y);
        printf("%d\n",abs(x)+abs(y));
}
```

【代码 2】

[cpp] view plain copy

```
#include<stdio.h>
#include<algorithm>
using namespace std;
int main()
{
        int ans[100];
        int top=0;
        for(int i=0;i<1000;i++)
                for(int j=0;j<1000;j++)
                        if(97*i-127*j==1)
                                ans[top++]=i+j;
        sort(ans,ans+top);
        printf("%d\n",ans[0]);
}
```

2.凑平方数

把 0~9 这 10 个数字，分成多个组，每个组恰好是一个平方数，这是能够办到的。 比如：0, 36, 5948721

再比如： 1098524736 1, 25, 6390784 0, 4, 289, 15376 等等…

注意，0 可以作为独立的数字，但不能作为多位数字的开始。 分组时，必须用完所有的数字，不能重复，不能遗漏。

如果不计较小组内数据的先后顺序，请问有多少种不同的分组方案？

注意：需要提交的是一个整数，不要填写多余内容。

【答案】：300

【解析】：

先打表，把所有的无重复数字的完全平方数计算出来（600 多个）

数据量不大，dfs 即可。这里我用了字符串 string 处理。（这样能避免第一个数字是 0 的时候造成的干扰）

【代码】：

[cpp] view plain copy

```
#include<stdio.h>
#include<math.h>
#include<string>
using namespace std;
typedef long long ll;
ll pow2[1000];
int top=0,ans;
int check(string num)//查重
{
    int vis[12]={0};//标记数字的有无
    int len=num.length();
    for(int i=0;i<len;i++)
    {
        int t=num[i]-'0';
        vis[t]++;
        if(vis[t]>1)return 0;//有重复
    }
    return 1;//无重复
}
string zhuan(ll num)//把长整型数据转化为字符串
{
    string s;
    if(num==0)s+="0";
    while(num)
    {
        char ch[]={num%10+'0','\0'};//临时字符串
        s.insert(0,ch);
        num/=10;
    }
    return s;
}
```

```
void init()//完全平方数打表
{
	for(ll i=0;i<=100000;i++)
	{
		ll j=i*i;
		if(check(zhuan(j)))
			pow2[top++]=j;
	}
}
void dfs(int start,string num)
{
	int len=num.length();
	if(len>10||check(num)==0)return;//已有重复，直接 return
	if(len==10&&check(num))
	{
		//printf("%s\n",&num[0]);
		ans++;return;
	}
	for(int i=start;i<top;i++)
		dfs(i+1,num+zhuan(pow2[i]));
}
int main()
{
	init();//完全平方数打表
	ans=0;
	dfs(0,"");
	printf("%d\n",ans);
	return 0;
}
```

3.棋子换位

有 n 个棋子 A，n 个棋子 B，在棋盘上排成一行。 它们中间隔着一个空位，用“.”表示，比如：

AAA.BBB

现在需要所有的 A 棋子和 B 棋子交换位置。 移动棋子的规则是：
1. A 棋子只能往右边移动，B 棋子只能往左边移动。
2. 每个棋子可以移动到相邻的空位。
3. 每个棋子可以跳过相异的一个棋子落入空位（A 跳过 B 或者 B 跳过 A）。

AAA.BBB 可以走法： 移动 A ==> AA.ABBB 移动 B ==> AAAB.BB

跳走的例子： AA.ABBB ==> AABA.BB

以下的程序完成了 AB 换位的功能，请仔细阅读分析源码，填写划线部分缺失的内容。

[html] view plain copy

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

void move(char* data, int from, int to)
{
    data[to] = data[from];
    data[from] = '.';
}

int valid(char* data, int k)
{
    if(k<0 || k>=strlen(data)) return 0;
    return 1;
}

void f(char* data)
{
    int i;
    int tag;
    int dd = 0; // 移动方向

    while(1){
        tag = 0;
        for(i=0; i<strlen(data); i++){
            if(data[i]=='.') continue;
            if(data[i]=='A') dd = 1;
            if(data[i]=='B') dd = -1;

            if(valid(data, i+dd) && valid(data,i+dd+dd)
            && data[i+dd]!=data[i] && data[i+dd+dd]=='.'){
            //如果能跳...
                move(data, i, i+dd+dd);
                printf("%s\n", data);
                tag = 1;
                break;
            }
        }
```

```
        if(tag) continue;

        for(i=0; i<strlen(data); i++){
            if(data[i]=='.') continue;
            if(data[i]=='A') dd = 1;
            if(data[i]=='B') dd = -1;

            if(valid(data, i+dd) && data[i+dd]=='.'){
            //  如果能移动...
                if( ____________________ ) continue;    //填空位置
                move(data, i, i+dd);
                printf("%s\n", data);
                tag = 1;
                break;
            }
        }

        if(tag==0) break;
    }
}

int main()
{
    char data[] = "AAA.BBB";
    f(data);
    return 0;
}
```

【答案】：valid(data, i+dd+dd) && valid(data,i-dd)&&data[i+dd+dd]==data[i-dd]
【解析】：

有规律，如果当前要移动的字母和点的位置同时去掉后，碰到一起的字母相同，就不能移动

比如运行结果的 2->3 步

【运行结果】：

```
AA.ABBB
AABA.BB
AABAB.B
AAB.BAB
A.BABAB
.ABABAB
BA.ABAB
BABA.AB
```

BABABA.
BABAB.A
BAB.BAA
B.BABAA
BB.ABAA
BBBA.AA
BBB.AAA

4.机器人塔

X 星球的机器人表演拉拉队有两种服装，A 和 B。 他们这次表演的是搭机器人塔。

类似：

A
B B
A B A
A A B B
B B B A B
A B A B B A
队内的组塔规则是：
A 只能站在 AA 或 BB 的肩上。 B 只能站在 AB 或 BA 的肩上。

你的任务是帮助拉拉队计算一下，在给定 A 与 B 的人数时，可以组成多少种花样的塔。

输入一行两个整数 M 和 N，空格分开（0<M,N<500）,分别表示 A、B 的人数，保证人数合理性。

要求输出一个整数，表示可以产生的花样种数。

例如：
用户输入：
1 2

程序应该输出：
3

再例如：

用户输入：
3 3

程序应该输出：
4
【解析】：此题没有搜到正解，只能用搜索了。

数据大了肯定超时。不过能过一部分数据也可以混点分，嘻嘻

根据 m，n 的值，最多有 44 层。

只要每一层的第一个确定下来，这一行就是一定的。

所有的情况有 2 的 44 次方，搜索必定超时

【代码】：

[cpp] view plain copy

```
#include<stdio.h>
#include<math.h>
int a[100][100];
const int A=1;
const int B=-1;
int ans;
int tall;
void dfs(int m,int n,int i)
{
    if(m<0||n<0||i>tall)return;
    if(m==0&&n==0)
    {
        ans++;return;
    }
    for(int k=B;k<=A;k=k+2)//假设为 A，B 两种情况
    {
        int mm=m,nn=n;
        a[i][1]=k;//假设行首
        if(a[i][1]==A) mm--;
        else   nn--;
        if(mm<0||nn<0)continue;
        for(int j=2;j<=i;j++)
        {
            if(a[i-1][j-1]==A)//头顶是 A，底下相同
```

```
                {
                    a[i][j]=a[i][j-1];
                    if(a[i][j]==A) mm--;
                    else    nn--;
                }
                else   //否则相异
                {
                    a[i][j]=-a[i][j-1];
                    if(a[i][j]==A) mm--;
                    else    nn--;
                }
                if(nn<0||mm<0)continue;
            }
            dfs(mm,nn,i+1);
        }
}
int main()
{
    int m,n;
    scanf("%d%d",&m,&n);
    ans=0;
    tall=(sqrt(1+8*(m+n))-1)/2;
    dfs(m,n,1);
    printf("%d\n",ans);
    return 0;
 }
```

5.广场舞

LQ 市的市民广场是一个多边形，广场上铺满了大理石的地板砖。

地板砖铺得方方正正，就像坐标轴纸一样。
以某四块砖相接的点为原点，地板砖的两条边为两个正方向，一块砖的边长为横纵坐标的单位长度，则所有横纵坐标都为整数的点都是四块砖的交点（如果在广场内）。

广场的砖单调无趣，却给跳广场舞的市民们提供了绝佳的参照物。每天傍晚，都会有大批市民前来跳舞。
舞者每次都会选一块完整的砖来跳舞，两个人不会选择同一块砖，如果一块砖在广场边上导致缺角或者边不完整，则没人会选这块砖。
（广场形状的例子参考【图 1.png】）

现在，告诉你广场的形状，请帮 LQ 市的市长计算一下，同一时刻最多有多少市民可以在广场跳舞。

【输入格式】 输入的第一行包含一个整数 n，表示广场是 n 边形的（因此有 n 个顶点）。接下来 n 行，每行两个整数，依次表示 n 边形每个顶点的坐标（也就是说广场边缘拐弯的地方都在砖的顶角上。数据保证广场是一个简单多边形。

【输出格式】 输出一个整数，表示最多有多少市民可以在广场跳舞。

【样例输入】 5 3 3 6 4 4 1 1 -1 0 4

【样例输出】 7

【样例说明】 广场如图 1.png 所示，一共有 7 块完整的地板砖，因此最多能有 7 位市民一起跳舞。

【数据规模与约定】 对于 30%的数据，n 不超过 100，横纵坐标的绝对值均不超过 100。
对于 50%的数据，n 不超过 1000，横纵坐标的绝对值均不超过 1000。
对于 100%的数据，n 不超过 1000，横纵坐标的绝对值均不超过 100000000（一亿）。

资源约定： 峰值内存消耗 < 256M CPU 消耗 < 1000ms

请严格按要求输出，不要画蛇添足地打印类似：“请您输入…”的多余内容。

所有代码放在同一个源文件中，调试通过后，拷贝提交该源码。

注意: main 函数需要返回 0 注意: 只使用 ANSI C/ANSI C++ 标准，不要调用依赖于编译环境或操作系统的特殊函数。 注意:
所有依赖的函数必须明确地在源文件中 #include ， 不能通过工程设置而省略常用头文件。

提交时，注意选择所期望的编译器类型。
这里写图片描述

6.生成树计数(本题图片部分丢失，没及时解题，望谅解)

给定一个 n*m 的格点图，包含 n 行 m 列共 n*m 个顶点，相邻的顶点之间有一条边。【图 1.png】给出了一个 3*4 的格点图的例子。

如果在图中删除部分顶点和其相邻的边，如上图删除第 2 行第 3 列和第 3 行第 1 列的顶点

后，如【图 2.png】所示。

图的生成树指包含图中的所有顶点和其中的一部分边，使得任意两个顶点之间都有由边构成的唯一路径。如果两个生成树包含有不同的边即被认为不同，则上图中共有 31 种不同的生成树，其中 a 边不选有 10 种，a 边选有 21 种。
给出格点图中保留的顶点的信息，请计算该图一共有多少种不同的生成树。

【输入格式】 输入的第一行包含两个整数 n, m，用空格分隔，表示格点图的行数和列数。接下来 n 行，每行 m 个字母（中间没有分隔字符），每个字母必然是大写 E 或大写 N，E 表示对应的顶点存在，N 表示对应的顶点不存在。保证存在至少一个顶点。

【输出格式】 输出一行，包含一个整数，表示生成树的个数。答案可能很大，你只需要计算答案除以 1000000007 的余数即可。

【样例输入】 3 4 EEEE EENE NEEE

【样例输出】 31

【数据规模与约定】 对于 10%的数据，1<=n<=2。 对于 30%的数据，1<=n<=3。 对于 40%的数据，1<=n<=4。
对于 50%的数据，1<=n<=5。另有 20%的数据，1<=n*m<=12。另有 10%的数据，1<=m<=15。
对于 100%的数据，1<=n<=6，1<=m<=100000。

资源约定： 峰值内存消耗 < 256M CPU 消耗 < 4500ms

请严格按要求输出，不要画蛇添足地打印类似：“请您输入…”的多余内容。

所有代码放在同一个源文件中，调试通过后，拷贝提交该源码。

注意: main 函数需要返回 0 注意: 只使用 ANSI C/ANSI C++ 标准，不要调用依赖于编译环境或操作系统的特殊函数。 注意:
所有依赖的函数必须明确地在源文件中 #include ， 不能通过工程设置而省略常用头文件。

提交时，注意选择所期望的编译器类型。

2017 第八届蓝桥杯决赛（大学 B 组）试题

1.36 进制

2.磁砖样式

3.希尔伯特曲线

4.发现环

5.对局匹配

6.观光旅游

标题：36 进制

对于 16 进制，我们使用字母 A-F 来表示 10 及以上的数字。
如法炮制，一直用到字母 Z，就可以表示 36 进制。
36 进制中，A 表示 10，Z 表示 35，AA 表示 370
你能算出 MANY 表示的数字用 10 进制表示是多少吗?
请提交一个整数，不要填写任何多余的内容（比如，说明文字）

答案：1040254

[cpp] view plain copy

```
#include<stdio.h>
int main()
{
	int a='Y'-'A'+10;
	int b='N'-'A'+10;
	int c=10;
	int d='M'-'A'+10;
	b=b*36;
	c=c*36*36;
	d=d*36*36*36;
	int res=a+b+c+d;
	printf("%d\n",res);
	return 0;
}
```

标题：磁砖样式

小明家的一面装饰墙原来是 3*10 的小方格。
现在手头有一批刚好能盖住 2 个小方格的长方形瓷砖。
瓷砖只有两种颜色：黄色和橙色。
小明想知道，对于这么简陋的原料，可以贴出多少种不同的花样来。
小明有个小小的强迫症：忍受不了任何 2*2 的小格子是同一种颜色。
（瓷砖不能切割，不能重叠，也不能只铺一部分。另外，只考虑组合图案，请忽略瓷砖的拼缝）
显然，对于 2*3 个小格子来说，口算都可以知道：一共 10 种贴法，如【p1.png 所示】

但对于 3*10 的格子呢？肯定是个不小的数目，请你利用计算机的威力算出该数字。
注意：你需要提交的是一个整数，不要填写任何多余的内容（比如：说明性文字）

答案：114434

[cpp] view plain copy

```
#include<stdio.h>
int a[5][12],res=0;
bool judge(int x,int y)
{
if(a[x][y]==a[x-1][y-1]&&a[x][y]==a[x-1][y]&&a[x][y]==a[x][y-1])
return false;
if(a[x][y]==a[x-1][y]&&a[x][y]==a[x-1][y+1]&&a[x][y]==a[x][y+1])
return false;
if(a[x][y]==a[x][y-1]&&a[x][y]==a[x+1][y-1]&&a[x][y]==a[x+1][y])
return false;
return true;

}
void dfs(int x,int y)
{
if(x==3&&y==10)
{
res++;
return;
}
if(y>10)
{
dfs(x+1,0);
return;
}
if(a[x][y]==-1)
{
if(a[x][y+1]==-1)
{
a[x][y]=1;
a[x][y+1]=1;
if(judge(x,y))
dfs(x,y+1);
a[x][y]=0;
a[x][y+1]=0;
```

```
a[x][y]=2;
a[x][y+1]=2;
if(judge(x,y))
dfs(x,y+1);
a[x][y]=-1;
a[x][y+1]=-1;
}
if(a[x+1][y]==-1)
{
a[x][y]=1;
a[x+1][y]=1;
if(judge(x,y))
dfs(x,y+1);
a[x][y]=-1;
a[x+1][y]=-1;

a[x][y]=2;
a[x+1][y]=2;
if(judge(x,y))
dfs(x,y+1);
a[x][y]=-1;
a[x+1][y]=-1;
}
}
else
{
dfs(x,y+1);
}
}


int main()
{
int i,j;
for(i=1;i<=3;i++)
for(j=1;j<=10;j++)
a[i][j]=-1;
dfs(1,1);
printf("%d\n",res);
return 0;
}
```

标题：希尔伯特曲线

希尔伯特曲线是以下一系列分形曲线 Hn 的极限。我们可以把 Hn 看作一条覆盖 2^n × 2^n 方格矩阵的曲线，曲线上一共有 2^n × 2^n 个顶点(包括左下角起点和右下角终点)，恰好覆盖每个方格一次。

[p1.png]

Hn(n > 1)可以通过如下方法构造：

1. 将 Hn-1 顺时针旋转 90 度放在左下角
2. 将 Hn-1 逆时针旋转 90 度放在右下角
3. 将 2 个 Hn-1 分别放在左上角和右上角
4. 用 3 条单位线段把 4 部分连接起来

对于 Hn 上每一个顶点 p ，我们定义 p 的坐标是它覆盖的小方格在矩阵中的坐标(左下角是(1, 1)，右上角是(2^n, 2^n)，从左到右是 X 轴正方向，从下到上是 Y 轴正方向)，

定义 p 的序号是它在曲线上从起点开始数第几个顶点(从 1 开始计数)。

以下程序对于给定的 n(n <= 30)和 p 点坐标(x, y)，输出 p 点的序号。请仔细阅读分析源码，填写划线部分缺失的内容。

```
#include <stdio.h>
long long f(int n, int x, int y) {
	if (n == 0) return 1;
	int m = 1 << (n - 1);
	if (x <= m && y <= m) {
		return f(n - 1, y, x);
	}
	if (x > m && y <= m) {
		return 3LL * m * m + f(n - 1, m-y+1 , m * 2 - x + 1); //   填空
	}
	if (x <= m && y > m) {
		return 1LL * m * m + f(n - 1, x, y - m);
	}
	if (x > m && y > m) {
		return 2LL * m * m + f(n - 1, x - m, y - m);
	}
}

int main() {
int n, x, y;
	scanf("%d %d %d", &n, &x, &y);
	printf("%lld", f(n, x, y));
	return 0;
}
```

注意：只填写划线处缺少的内容，不要填写已有的代码或符号，也不要填写任何解释说明文字等。

标题：发现环

小明的实验室有 N 台电脑，编号 1~N。原本这 N 台电脑之间有 N-1 条数据链接相连，恰好构成一个树形网络。在树形网络上，任意两台电脑之间有唯一的路径相连。

不过在最近一次维护网络时，管理员误操作使得某两台电脑之间增加了一条数据链接，于是网络中出现了环路。环路上的电脑由于两两之间不再是只有一条路径，使得这些电脑上的数据传输出现了 BUG。
为了恢复正常传输。小明需要找到所有在环路上的电脑，你能帮助他吗？
输入

第一行包含一个整数 N。
以下 N 行每行两个整数 a 和 b，表示 a 和 b 之间有一条数据链接相连。
对于 30%的数据，1 <= N <= 1000
对于 100%的数据, 1 <= N <= 100000， 1 <= a, b <= N
输入保证合法。
输出

按从小到大的顺序输出在环路上的电脑的编号，中间由一个空格分隔。
样例输入：
5
1 2
3 1
2 4
2 5
5 3
样例输出：
1 2 3 5
资源约定：
峰值内存消耗 < 256M
CPU 消耗 < 1000ms
请严格按要求输出，不要画蛇添足地打印类似：“请您输入...”的多余内容。
所有代码放在同一个源文件中，调试通过后，拷贝提交该源码。
注意: main 函数需要返回 0
注意: 只使用 ANSI C/ANSI C++ 标准，不要调用依赖于编译环境或操作系统的特殊函数。
注意: 所有依赖的函数必须明确地在源文件中 #include <xxx>， 不能通过工程设置而省略常用头文件。
提交时，注意选择所期望的编译器类型。

[cpp] view plain copy
```
#include<stdio.h>
#include<vector>
#include<algorithm>
using namespace std;
vector<int>v[100010];
int used[100010];
int pre[100010];
int ans[100010];
```

```
int ct=0;
bool over=false;
void dfs(int u,int f)
{
	int i;
	for(i=0;i<v[u].size();i++)
	{
		int to=v[u][i];
		if(to==f)
			continue;
		pre[to]=u;
		if(used[to]==1)
		{
			int a=to;
			do
			{
				ans[ct++]=a;
				a=pre[a];
			}while(a!=to);
			over=true;
			return;
		}
		used[to]=1;
		dfs(to,u);
		if(over)
			return;
	}
}
int main()
{
	int n,i,a,b;
	scanf("%d",&n);
	for(i=0;i<n;i++)
	{
		scanf("%d%d",&a,&b);
		v[a].push_back(b);
		v[b].push_back(a);
	}
	used[1]=1;
	dfs(1,0);
	sort(ans,ans+ct);
	for(i=0;i<ct-1;i++)
	{
		printf("%d ",ans[i]);
```

```
    }
    printf("%d\n",ans[i]);
    return 0;
}
```

标题：对局匹配

小明喜欢在一个围棋网站上找别人在线对弈。这个网站上所有注册用户都有一个积分，代表他的围棋水平。

小明发现网站的自动对局系统在匹配对手时，只会将积分差恰好是 K 的两名用户匹配在一起。如果两人分差小于或大于 K，系统都不会将他们匹配。

现在小明知道这个网站总共有 N 名用户，以及他们的积分分别是 A1, A2, ... AN。

小明想了解最多可能有多少名用户同时在线寻找对手，但是系统却一场对局都匹配不起来(任意两名用户积分差不等于 K)？

输入

第一行包含两个个整数 N 和 K。

第二行包含 N 个整数 A1, A2, ... AN。

对于 30%的数据，1 <= N <= 10

对于 100%的数据，1 <= N <= 100000, 0 <= Ai <= 100000, 0 <= K <= 100000

输出

一个整数，代表答案。

样例输入：

10 0

1 4 2 8 5 7 1 4 2 8

样例输出：

6

再比如，

样例输入：

10 1

2 1 1 1 1 4 4 3 4 4

样例输出：

8

资源约定：

峰值内存消耗 < 256M

CPU 消耗 < 1000ms

请严格按要求输出，不要画蛇添足地打印类似：“请您输入...”的多余内容。

所有代码放在同一个源文件中，调试通过后，拷贝提交该源码。

注意: main 函数需要返回 0

注意: 只使用 ANSI C/ANSI C++ 标准，不要调用依赖于编译环境或操作系统的特殊函数。

注意: 所有依赖的函数必须明确地在源文件中 #include <xxx>， 不能通过工程设置而省略常用头文件。

提交时，注意选择所期望的编译器类型。

这题对于 100%的数据来说我的代码肯定是超时了....

[cpp] view plain copy

```
#include<stdio.h>
#include<algorithm>
#define max(x,y)(x>y?x:y)
using namespace std;
int a[100010];
int dp[100010];
int main()
{
	int i,j,n,k,res=0;
	scanf("%d%d",&n,&k);
	for(i=0;i<n;i++)
		scanf("%d",&a[i]);
	sort(a,a+n);
	dp[0]=1;
	for(i=1;i<n;i++)
	{
		for(j=0;j<i;j++)
		{
			if(a[i]-a[j]==k)
				continue;
			dp[i]=max(dp[i],dp[j]+1);
		}
	}
	for(i=0;i<n;i++)
	{
		res=max(res,dp[i]);
	}
	printf("%d\n",res);
	return 0;
}
```

标题：观光铁路

跳蚤国正在大力发展旅游业，每个城市都被打造成了旅游景点。

许多跳蚤想去其他城市旅游，但是由于跳得比较慢，它们的愿望难以实现。这时，小 C 听说有一种叫做火车的交通工具，在铁路上跑得很快，便抓住了商机，创立了一家铁路公司，向跳蚤国王请示在每两个城市之间都修建铁路。

然而，由于小 C 不会扳道岔，火车到一个城市以后只能保证不原路返回，而会随机等概率地驶向与这个城市有铁路连接的另外一个城市。

跳蚤国王向广大居民征求意见，结果跳蚤们不太满意，因为这样修建铁路以后有可能只游览了 3 个城市（含出发的城市）以后就回来了，它们希望能多游览几个城市。于是跳蚤国王要求小 C 提供一个方案，使得每只跳蚤坐上火车后能多游览几个城市才回来。
小 C 提供了一种方案给跳蚤国王。跳蚤国王想知道这个方案中每个城市的居民旅游的期望时间（设火车经过每段铁路的时间都为 1），请你来帮跳蚤国王。
【输入格式】
输入的第一行包含两个正整数 n、m，其中 n 表示城市的数量，m 表示方案中的铁路条数。
接下来 m 行，每行包含两个正整数 u、v，表示方案中城市 u 和城市 v 之间有一条铁路。
保证方案中无重边无自环，每两个城市之间都能经过铁路直接或间接到达，且火车由任意一条铁路到任意一个城市以后一定有路可走。
【输出格式】
输出 n 行，第 i 行包含一个实数 ti，表示方案中城市 i 的居民旅游的期望时间。你应当输出足够多的小数位数，以保证输出的值和真实值之间的绝对或相对误差不超过 1e-9。
【样例输入】
4 5
1 2
2 3
3 4
4 1
1 3
【样例输出】
3.333333333333
5.000000000000
3.333333333333
5.000000000000
【样例输入】
10 15
1 2
1 9
1 5
2 3
2 7
3 4
3 10
4 5
4 8
5 6
6 7
6 10
7 8
8 9
9 10
【样例输出】
10.000000000000

10.000000000000
10.000000000000
10.000000000000
10.000000000000
10.000000000000
10.000000000000
10.000000000000
10.000000000000
10.000000000000
【数据规模与约定】
对于 10%的测试点，n <= 10；
对于 20%的测试点，n <= 12；
对于 50%的测试点，n <= 16；
对于 70%的测试点，n <= 19；
对于 100%的测试点，4 <= k <= n <= 21，1 <= u, v <= n。数据有梯度。
资源约定：
峰值内存消耗 < 256M
CPU 消耗 < 2000ms
请严格按要求输出，不要画蛇添足地打印类似：“请您输入...”的多余内容。
所有代码放在同一个源文件中，调试通过后，拷贝提交该源码。
注意: main 函数需要返回 0
注意: 只使用 ANSI C/ANSI C++ 标准，不要调用依赖于编译环境或操作系统的特殊函数。
注意: 所有依赖的函数必须明确地在源文件中 #include <xxx>， 不能通过工程设置而省略常用头文件。
提交时，注意选择所期望的编译器类型。