<https://www.wxz.name/2016/07/27/acmac/>

<http://www.cnblogs.com/zjutlitao/p/4337775.html>

<http://www.cnblogs.com/zichi/p/5173917.html>

<http://m.blog.csdn.net/article/details?id=47448599>

# Chessboard

**Time Limit: 2000/1000 MS (Java/Others)    Memory Limit: 32768/32768 K (Java/Others)  
Total Submission(s): 1055    Accepted Submission(s): 447**

**Problem Description**

Consider the problem of tiling an n×n chessboard by polyomino pieces that are k×1 in size; Every one of the k pieces of each polyomino tile must align exactly with one of the chessboard squares. Your task is to figure out the maximum number of chessboard squares tiled.

**Input**

There are multiple test cases in the input file.  
First line contain the number of cases T (T≤10000).   
In the next T lines contain T cases , Each case has two integers n and k. (1≤n,k≤100)

**Output**

Print the maximum number of chessboard squares tiled.

**Sample Input**

2

6 3

5 3

**Sample Output**

36

24

**用 k × 1 的小矩形覆盖一个 n × n 的正方形棋盘，问正方形棋盘最多能被覆盖多少。**

**规律就是：如果n<k，肯定不行。**

**定义mod=n%k;**

**如果(mod<=k/2),结果为：n\*n-mod\*mod；**

**否则结果为：n\*n-(k-mod)\*(k-mod)；**

#include<stdio.h>

int main()

{

int t,n,k;

scanf("%d",&t);

while(t--)

{

scanf("%d%d",&n,&k);

if(n<k){printf("0\n");continue;}

int mod=n%k;

if(mod<=k/2)printf("%d\n",n\*n-mod\*mod);

else printf("%d\n",n\*n-(k-mod)\*(k-mod));

}

return 0;

}

**费马小定理**（Fermat Theory）是数论中的一个重要定理，其内容为： 假如p是[质数](http://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=67850&ss_c=ssc.citiao.link" \t "http://blog.csdn.net/baidu_23955875/article/details/_blank)，且Gcd(a,p)=1，那么 a(p-1) ≡1（mod p）。即：假如a是整数，p是质数，且a,p互质(即两者只有一个[公约数](http://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=7531927&ss_c=ssc.citiao.link" \t "http://blog.csdn.net/baidu_23955875/article/details/_blank)1)，那么a的(p-1)[次方](http://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=73782511&ss_c=ssc.citiao.link" \t "http://blog.csdn.net/baidu_23955875/article/details/_blank)除以p的[余数](http://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=795886&ss_c=ssc.citiao.link" \t "http://blog.csdn.net/baidu_23955875/article/details/_blank)恒等于1。该定理是1636年皮埃尔·德·费马发现的。

**所以我们要做的就是把这一题与费马小定理联系起来！**

**f(x)=5\*x^13+13\*x^5+k\*a\*x转化成f(x)=(5\*x^12+13\*x^4+k\*a)\*x**

**1.当x=65的倍数时就行了；**

**2.当x=5的倍数时则需要（5\*x^12+k\*a）是13的倍数，由费马小定理可知，因为x是5的倍数所以（x^(13-1)%13==1）,所以（5\*x^12%13==5），所以只需要做到k\*a%13==8即可！**

**3.同理可得当x=13的倍数时，只需要做到（13\*x^4+k\*a)是5的倍数由费马小定理可知，因为x是5的倍数所以（x^(5-1)%5==1）,所以（13\*x^(5-1)%5==3），所以只需要做到k\*a%5==2即可**

**4.当x不是上面的特殊数时，则需要f(x)=(5\*x^12+13\*x^4+k\*a)\*x被65整除，也就是需要同时满足上面那两个条件！**

# Ignatius's puzzle

**Time Limit: 2000/1000 MS (Java/Others)    Memory Limit: 65536/32768 K (Java/Others)Total Problem Description**

Ignatius is poor at math,he falls across a puzzle problem,so he has no choice but to appeal to Eddy. this problem describes that:f(x)=5\*x^13+13\*x^5+k\*a\*x,input a nonegative integer k(k<10000),to find the minimal nonegative integer a,make the arbitrary integer x ,65|f(x)if  
no exists that a,then print "no".

**Input**

The input contains several test cases. Each test case consists of a nonegative integer k, More details in the Sample Input.

**Output**

The output contains a string "no",if you can't find a,or you should output a line contains the a.More details in the Sample Output.

**Sample Input**

11

100

9999

**Sample Output**

22

no

43

#include <stdio.h>

int main()

{

int k,flag;

while (scanf("%d",&k)!=EOF)

{

flag=0;

for (int i=1;i<=65;i++)

{

if (i\*k%13==8&&i\*k%5==2)

{

flag=i;

break;

}

}

if (flag)

{

printf("%d\n",flag);

}

else

{

printf("no\n");

}

}

}

大数相加模板

#include<iostream>

#include<string>

using namespace std;

int main()

{

int n,i;

scanf("%d",&n);

for(i=1;i<=n;i++)

{

string sum(string s1,string s2);

string s1,s2;

cin>>s1>>s2;

printf("Case %d:\n",i);

cout<<s1<<" + "<<s2<<" = "<<sum(s1,s2);

printf("\n\n");

}

return 0;

}

string sum(string s1,string s2)

{

if(s1.length()<s2.length())

{

string temp=s1;

s1=s2;

s2=temp;

}

for(int i=s1.length()-1,j=s2.length ()-1;i>=0;i--,j--)

{

s1[i]=char(s1[i]+(j>=0?s2[j]-'0':0));

if(s1[i]-'0'>=10)

{

s1[i]=char((s1[i]-'0')%10+'0');

if(i)

s1[i-1]++;

else

s1='1'+s1;

}

}

return s1;

}

大数阶乘

代码如下：

#include<iostream>

#include<cmath>

using namespace std;

int main()

{

void factorial(int n) ;

int n;

cin>>n;

factorial(n);

return 0;

}

void factorial(int n)

{

long a[10000];

int i,j,l,c,m=0,w;

a[0]=1;

for(i=1;i<=n;i++)

{

c=0;

for(j=0;j<=m;j++)

{

a[j]=a[j]\*i+c;

c=a[j]/10000;

a[j]=a[j]%10000;

}

if(c>0)

{

m++;

a[m]=c;

}

}

cout<<a[m];

for(i=m-1;i>=0;i--)

cout<<a[i];

cout<<endl;

}

**[Jerry Lee’s Stones(动态规划（最长不重复子串）+Hash)](http://blog.csdn.net/baidu_23955875/article/details/47206805)  jlnu oj**

#include<stdio.h>

#include<string.h>

**int** str[10000000];

**int** max(**int** a,**int** b)

{

**if**(a<b)

**return** b;

**else**

**return** a;

}

**int** solve(**int** \*str,**int** n)

{

**int** last=0,glo=0,loc=0;

**int** Hash[10000000]={0};

**for**(**int** i = 0; i <n; i ++)

    {

**int** k = str[i]-0;

         Hash[k] ++;

**if**(Hash[k] == 2)

         {

            glo = max(glo, loc);

**for**(; last < i && str[last] != str[i]; last++)

                Hash[str[last]-0]--;

            loc = i - last;

            Hash[k]--;

            last++;

         }

**else**

         loc ++;

    }

**return** max(glo, loc);

}

**int** main()

{

**int** t,n,i,a;

**scanf**("%d",&t);

**while**(t--)

    {

**scanf**("%d",&n);

**for**(i=0;i<n;i++)

**scanf**("%d",&str[i]);

**printf**("%d\n",solve(str,n));

    }

}

**康托展开：**

X=an\*(n-1)!+an-1\*(n-2)!+...+ai\*(i-1)!+...+a2\*1!+a1\*0!

ai为整数，并且0<=ai<i(1<=i<=n)

应用实例：

{1,2,3,4,...,n}的排列总共有n!种，将它们从小到大排序，怎样知道其中一种排列是有序序列中的第几个？

如 {1,2,3} 按从小到大排列一共6个：123 132 213 231 312 321。想知道321是{1,2,3}中第几个大的数。

这样考虑：第一位是3，小于3的数有1、2 。所以有2\*2!个。再看小于第二位，小于2的数只有一个就是1 ，所以有1\*1!=1 所以小于32

的{1,2,3}排列数有2\*2!+1\*1!=5个。所以321是第6个大的数。2\*2!+1\*1!是康托展开。

再举个例子：1324是{1,2,3,4}排列数中第几个大的数：第一位是1小于1的数没有，是0个，0\*3!，第二位是3小于3的数有1和2，但1已经在第一位了，所以只有一个数2，1\*2! 。第三位是2小于2的数是1，但1在第一位，所以有0个数，0\*1!，所以比1324小的排列有0\*3!+1\*2!+0\*1!=2个，1324是第三个大数。

int fac[] = {1,1,2,6,24,120,720,5040,40320}; //i的阶乘为fac[i]

/\* 康托展开.

{1...n}的全排列由小到大有序，s[]为第几个数 \*/

int KT(int n, int s[])

{

int i, j, t, sum;

sum = 0;

for (i=0; i<n; i++)

{

t = 0;

for (j=i+1; j<n; j++)

if (s[j] < s[i])

t++;

sum += t\*fac[n-i-1];

}

return sum+1;

}

康托展开的逆运算：

{1,2,3,4,5}的全排列已经从小到大排序，要找出第16个数：

1. 首先用16-1得到15

2. 用15去除4! 得到0余15

3. 用15去除3! 得到2余3

4. 用3去除2! 得到1余1

5. 用1去除1! 得到1余0

有0个数比它小的数是1

所以第一位是1

有2个数比它小的数是3，但1已经在之前出现过了所以是4

有1个数比它小的数是2，但1已经在之前出现过了所以是3

有1个数比它小的数是2，但1,3,4都出现过了所以是5

最后一个数只能是2

所以这个数是14352

**/\* 康托展开的逆运算.**

**{1...n}的全排列，中的第k个数为s[] \*/**

**void invKT(int n, int k, int s[])**

**{**

**int i, j, t, vst[8]={0};**

**k--;**

**for (i=0; i<n; i++)**

**{**

**t = k/fac[n-i-1];**

**for (j=1; j<=n; j++)**

**if (!vst[j])**

**{**

**if (t == 0) break;**

**t--;**

**}**

**s[i] = j;**

**vst[j] = 1;**

**k %= fac[n-i-1];**

**}**

**}**

**Power of Cryptography**

**2 16**

**3 27**

**7 4357186184021382204544**

**求b的1/a次幂是多少：**

**#include<cstdio>**

**#include<cmath>**

**#include<iostream>**

**using namespace std;**

**int main()**

**{**

**double a,b;**

**while(scanf("%lf%lf",&a,&b)!=EOF)**

**{**

**printf("%.0lf\n",pow(b,1.0/a));**

**// cout<<pow(b,1.0/a)<<endl;**

**}**

**}**

**凯莱定理：**

**定理的另一种表述**

**过n个有标志顶点的[树](http://baike.baidu.com/view/143352.htm" \t "http://blog.csdn.net/baidu_23955875/article/details/_blank)的数目等于n^(n-2)。**

**[3]****定理的理解**

此定理说明用n-1条边将n个一致的顶点连接起来的**[连通图](http://baike.baidu.com/view/3148644.htm" \t "http://blog.csdn.net/baidu_23955875/article/details/_blank)**的个数为**n^(n-2)**，也可以这样[理解](http://baike.baidu.com/view/58766.htm" \t "http://blog.csdn.net/baidu_23955875/article/details/_blank)，将n个城市连接起来的树状公路网络有**n^(n-2)**种[方案](http://baike.baidu.com/view/556358.htm" \t "http://blog.csdn.net/baidu_23955875/article/details/_blank)。所谓树状，指的是用n-1条边将n个顶点构成一个连通图。当然，建造一个树状的公路网络将n个城市连接起来，应求其中长度最短、造价最省的一种，或效益最大的一种。Cayley定理只是说明可能方案的数目。

星际之门（一）

时间限制：3000 ms  |  内存限制：65535 KB

难度：3

描述

公元3000年，子虚帝国统领着N个星系，原先它们是靠近光束飞船来进行旅行的，近来，X博士发明了星际之门，它利用虫洞技术，一条虫洞可以连通任意的两个星系，使人们不必再待待便可立刻到达目的地。

帝国皇帝认为这种发明很给力，决定用星际之门把自己统治的各个星系连结在一起。

可以证明，修建N-1条虫洞就可以把这N个星系连结起来。

现在，问题来了，皇帝想知道有多少种修建方案可以把这N个星系用N-1条虫洞连结起来？

输入

第一行输入一个整数T,表示测试数据的组数(T<=100)  
每组测试数据只有一行，该行只有一个整数N，表示有N个星系。(2<=N<=1000000)

输出

对于每组测试数据输出一个整数，表示满足题意的修建的方案的个数。输出结果可能很大，请输出修建方案数对10003取余之后的结果。

样例输入

2

3

4

样例输出

3

16

**#include<cstdio>**

**#include<iostream>**

**#include<cmath>**

**#define ll long long**

**using namespace std;**

**ll pow\_mod(ll a,ll n,ll m)**

**{**

**ll ans = 1;**

**a = a % m;**

**while(n>0)**

**{**

**if(n%2 ==1)**

**ans=(ans\*a)%m;**

**n=n/2;**

**a=(a\*a) % m;**

**}**

**return ans;**

**}**

**int main()**

**{**

**long long n;**

**int t;**

**cin>>t;**

**while(t--)**

**{**

**cin>>n;**

**cout<<pow\_mod(n,n-2,10003)<<endl;**

**}**

**}**

# [Just a Numble（求小数的第N位数）](http://blog.csdn.net/baidu_23955875/article/details/42236681)

# Just a Numble

**Time Limit: 3000/1000 MS (Java/Others)    Memory Limit: 32768/32768 K (Java/Others)  
Total Submission(s): 2410    Accepted Submission(s): 1160**

**Problem Description**

Now give you two integers n m, you just tell me the m-th number after radix point in 1/n,for example n=4,the first numble after point is 2,the second is 5,and all 0 followed

**Input**

Each line of input will contain a pair of integers for n and m(1<=n<=10^7,1<=m<=10^5)

**Output**

For each line of input, your program should print a numble on a line,according to the above rules

**Sample Input**

4 2

5 7

123 123

**Sample Output**

5

0

8

**#include<stdio.h>**

**int main()**

**{**

**int n,m,i,j,sum;**

**int k; while(scanf("%d%d",&n,&m)!=EOF)**

**{**

**sum=1;**

**for(i=0;i<m;i++)**

**{**

**sum=sum\*10;//这个题的意思是让每一位都X10，比如求得第一位**

**k=sum/n;//k即为第一位小数**

**sum=sum%n;//求得余数，还得让余数继续除以n也很好理解！**

**}**

**printf("%d\n",k%10);**

**}**

**}**

**[异或详解](http://blog.csdn.net/baidu_23955875/article/details/42124685)**

**异或门符号'^',在编程中的使用，有的时候能够让你的程序更加精炼简捷，  
尤其在C++的acm程序应用中，可以避免许多麻烦！！！它的几个简单作用  
讲解如下：  
它的简单规则是相同为0，不同为1，例如  
int a=3=011（2进制，在计算机中的存储形式）；  
int b=6=110（2进制，在计算机中的存储形式）；  
int c=a^b=101=5;  
切忌，在异或门的计算中都是应用的2进制！！  
任何数和0异或都等于它本身;两个相同的数异或后的结果是0；  
1）实现两个数的交换  
 　a=a^b; 　　  
   b=a^b;//b=a^b^b=a^0=a; 　　  
   a=a^b;//a=a^a^b=0^b=b;  
应用例子如下：  
#include<iostream>  
using namespace std;  
int main()  
{  
 int a=3;  
 int b=4;  
 a=a^b;  
 b=a^b;  
 a=a^b;  
 cout<<a<<' '<<b<<endl;  
 return 0;  
}  
2在一排数中找到独一无二的一个数  
例子如下：  
他的思路是;  
只要有相同的数那么他们异或后的结果就是0，最后就剩下0和那个独一无二的数相异或，就等于  
那个独一无二的数了！！哈哈！！例子，如下：  
#include<iostream>  
using namespace std;  
int main()  
{  
    int n,i,x,y;  
    while(scanf("%d",&n)!=EOF&&n)  
    {  
           x=0;  
           while(n--)  
           {  
               cin>>y;  
                x^=y;      
           }          
           printf("%d\n",x);              
    }  
    return 0;  
}  
随便输入几个数，找到其中没有和他相同的那个数！！哈！  
3）判断两个数是否相等  
if(a^b==0)  
printf("a=b");  
else   
printf("a!=b");**

**auto函数：**

#include <stdlib.h>

#include <stdio.h>

int main(void)

{

    int n;

    char \*str = "12345.67";

    n = atoi(str);

    printf("int=%d\n",n);

    return 0;

}

输出：

int = 12345

|  |
| --- |
| //vs2013里调用printf函数请使用预处理命令#define \_CRT\_SECURE\_NO\_WARNINGS  #include <stdlib.h>  #include <stdio.h>    int main()  {      char a[] = "-100";      char b[] = "123";      int c;      c = atoi(a) + atoi(b);      printf("c=%d\n", c);      return 0;  } |

执行结果：

c = 23

**[RMQ (Range Minimum/Maximum Query)算法](http://blog.csdn.net/baidu_23955875/article/details/44901357)**

本文章已收录于：

RMQ（Range Minimum/Maximum Query），即区间最值查询，是指这样一个问题：对于长度为n的数列A，回答若干询问RMQ（A,i,j）(i,j<=n)，返回数列A中下标在i，j之间的最小/大值。这两个问题是在实际应用中经常遇到的问题，下面介绍一下解决这两种问题的比较高效的算法。当然，该问题也可以用线段树（也叫区间树）解决，算法复杂度为：O(N)~O(logN)，这里我们暂不介绍。

**2.RMQ算法**

对于该问题，最容易想到的解决方案是遍历，复杂度是O(n)。但当数据量非常大且查询很频繁时，该算法无法在有效的时间内查询出正解。

本节介绍了一种比较高效的在线算法（ST算法）解决这个问题。所谓在线算法，是指用户每输入一个查询便马上处理一个查询。该算法一般用较长的时间做预处理，待信息充足以后便可以用较少的时间回答每个查询。ST（Sparse Table）算法是一个非常有名的在线处理RMQ问题的算法，它可以在O(nlogn)时间内进行预处理，然后在O(1)时间内回答每个查询。

**（一）首先是预处理，用动态规划（DP）解决。**

设A[i]是要求区间最值的数列，F[i, j]表示从第i个数起连续2^j个数中的最大值。（DP的状态）

例如：

A数列为：3 2 4 5 6 8 1 2 9 7

F[1，0]表示第1个数起，长度为2^0=1的最大值，其实就是3这个数。同理 F[1,1] = max(3,2) = 3, F[1，2]=max(3,2,4,5) = 5，F[1，3] = max(3,2,4,5,6,8,1,2) = 8;

并且我们可以容易的看出F[i,0]就等于A[i]。（DP的初始值）

这样，DP的状态、初值都已经有了，剩下的就是状态转移方程。

我们把F[i，j]平均分成两段（因为f[i，j]一定是偶数个数字），从 i 到i + 2 ^ (j - 1) - 1为一段，i + 2 ^ (j - 1)到i + 2 ^ j - 1为一段(长度都为2 ^ (j - 1))。用上例说明，当i=1，j=3时就是3,2,4,5 和 6,8,1,2这两段。F[i，j]就是这两段各自最大值中的最大值。于是我们得到了状态转移方程F[i, j]=max（F[i，j-1], F[i + 2^(j-1)，j-1]）。

代码如下：

**[cpp]** [view plain](http://blog.csdn.net/niushuai666/article/details/6624672" \t "_blank" \o "view plain)[copy](http://blog.csdn.net/niushuai666/article/details/6624672" \t "_blank" \o "copy)

1. **void** RMQ(**int** num) //预处理->O(nlogn)
2. {
3. **for**(**int** j = 1; j < 20; ++j)
4. **for**(**int** i = 1; i <= num; ++i)
5. **if**(i + (1 << j) - 1 <= num)
6. {
7. maxsum[i][j] = max(maxsum[i][j - 1], maxsum[i + (1 << (j - 1))][j - 1]);
8. minsum[i][j] = min(minsum[i][j - 1], minsum[i + (1 << (j - 1))][j - 1]);
9. }
10. }

[http://static.blog.csdn.net/images/save_snippets.png](javascript:;)

这里我们需要注意的是循环的顺序，我们发现外层是j，内层所i，这是为什么呢？可以是i在外，j在内吗？

答案是不可以。因为我们需要理解这个状态转移方程的意义。

状态转移方程的含义是：先更新所有长度为F[i,0]即1个元素，然后通过2个1个元素的最值，获得所有长度为F[i,1]即2个元素的最值，然后再通过2个2个元素的最值，获得所有长度为F[i,2]即4个元素的最值，以此类推更新所有长度的最值。

而如果是i在外，j在内的话，我们更新的顺序就是F[1,0],F[1,1],F[1,2],F[1,3],表示更新从1开始1个元素，2个元素，4个元素，8个元素（A[0],A[1],....A[7]）的最值，这里F[1,3] = max(max(A[0],A[1],A[2],A[3]),max(A[4],A[5],A[6],A[7]))的值，但是我们根本没有计算max(A[0],A[1],A[2],A[3])和max(A[4],A[5],A[6],A[7])，所以这样的方法肯定是错误的。

为了避免这样的错误，一定要好好理解这个状态转移方程所代表的含义。

**（二）然后是查询。**

假如我们需要查询的区间为(i,j)，那么我们需要找到覆盖这个闭区间(左边界取i，右边界取j)的最小幂（可以重复，比如查询5，6，7，8，9，我们可以查询5678和6789）。

因为这个区间的长度为j - i + 1,所以我们可以取k=log2( j - i + 1)，则有：RMQ(A, i, j)=max{F[i , k], F[ j - 2 ^ k + 1, k]}。

举例说明，要求区间[2，8]的最大值，k = log2（8 - 2 + 1）= 2，即求max(F[2, 2]，F[8 - 2 ^ 2 + 1, 2]) = max(F[2, 2]，F[5, 2])；

在这里我们也需要注意一个地方，就是<<运算符和+-运算符的优先级。

比如这个表达式：5 - 1 << 2是多少？

答案是：4 \* 2 \* 2 = 16。所以我们要写成5 - (1 << 2)才是5-1 \* 2 \* 2 = 1。

## 1735: 小明的账目

Time Limit: 10 Sec  Memory Limit: 1280 MB  
Submit: 46  Solved: 6  
[[Submit](http://10.5.54.252/oj/submitpage.php?id=1735)][[Status](http://10.5.54.252/oj/problemstatus.php?id=1735)][[Web Board](http://10.5.54.252/oj/bbs.php?pid=1735)]

## Description

小明是一个聪明能干的人。他工作的10年中，为了让账目清楚，每天都要记k次账。小明的上司将每次的账目按1，2，3…编号，然后不定时的问小明问题，问题是这样的：在a到b号账中最少的一笔是多少？上司总是一次问多个问题。请你帮小明来回答。

## Input

输入中第一行有两个数m,n表示有m(m< =100000)笔账,n表示有n个问题，n< =100000。第二行为m个数,分别是账目的钱数后面n行分别是n个问题,每行有2个数字说明开始结束的账目编号。

## Output

输出为每个问题的答案。具体查看样例。

## Sample Input

10 3

1 2 3 4 5 6 7 8 9 10

2 7

3 9

1 10

## Sample Output

2 3 1

## HINT

**#include<stdio.h>**

**#include<string.h>**

**#include<string>**

**#include<stack>**

**#include<queue>**

**#include<math.h>**

**#include<limits.h>**

**#include<iostream>**

**#include<algorithm>**

**using namespace std;**

**const int N=100002;**

**int minsum[20][N];**

**void RMQ(int num)//RMQ算法**

**{**

**for(int i=1;i!=20;++i)//动态规划思想预处理**

**for(int j=1;j<=num;++j)**

**if(j+(1<<i)-1<=num)**

**minsum[i][j]=min(minsum[i-1][j],minsum[i-1][j+(1<<i>>1)]);**

**}**

**int main()**

**{**

**int m,n,i,l,r;**

**scanf("%d%d",&m,&n);**

**for(i=1;i<=m;i++)**

**scanf("%d",&minsum[0][i]);**

**RMQ(m);**

**for(i=0;i<n;i++)**

**{**

**scanf("%d%d",&l,&r);**

**int k=(int)(log(r-l+1.0)/log(2.0));//找到被2的n次方覆盖的区间**

**int minres=min(minsum[k][l],minsum[k][r-(1<<k)+1]);**

**if(i==n-1)**

**printf("%d\n",minres);**

**else**

**printf("%d ",minres);**

**}**

**return 0;**

**}**

士兵杀敌（三）   
时间限制：2000 ms | 内存限制：65535 KB   
难度：5   
描述   
南将军统率着N个士兵，士兵分别编号为1~N,南将军经常爱拿某一段编号内杀敌数最高的人与杀敌数最低的人进行比较，计算出两个人的杀敌数差值，用这种方法一方面能鼓舞杀敌数高的人，另一方面也算是批评杀敌数低的人，起到了很好的效果。

所以，南将军经常问军师小工第i号士兵到第j号士兵中，杀敌数最高的人与杀敌数最低的人之间军功差值是多少。

现在，请你写一个程序，帮小工回答南将军每次的询问吧。

注意，南将军可能询问很多次。

输入   
只有一组测试数据   
第一行是两个整数N,Q，其中N表示士兵的总数。Q表示南将军询问的次数。(1<N<=100000,1<Q<=1000000)   
随后的一行有N个整数Vi(0<=Vi<100000000)，分别表示每个人的杀敌数。   
再之后的Q行，每行有两个正正数m,n，表示南将军询问的是第m号士兵到第n号士兵。   
输出   
对于每次询问，输出第m号士兵到第n号士兵之间所有士兵杀敌数的最大值与最小值的差。   
样例输入   
5 2   
1 2 6 9 3   
1 2   
2 4   
样例输出   
1   
7

RMQ算法的水题，当然也可以用线段树，等会贴出来RMQ算法的总结，我要先去上（shui）课（jiao）了。

代码贴上，解析晚上写

#include<stdio.h>

#include<string.h>

#include<string>

#include<stack>

#include<queue>

#include<math.h>

#include<limits.h>

#include<iostream>

#include<algorithm>

using namespace std;

const int N=100010;

int maxsum[20][N],minsum[20][N];

void RMQ(int num)//RMQ算法

{

for(int i=1;i!=20;++i)//动态规划思想预处理

for(int j=1;j<=num;++j)

if(j+(1<<i)-1<=num)

{//位运算用多了会不会很难懂

maxsum[i][j]=max(maxsum[i-1][j],maxsum[i-1][j+(1<<i>>1)]);

minsum[i][j]=min(minsum[i-1][j],minsum[i-1][j+(1<<i>>1)]);

}

}

int main()

{

int num,query\_num;//士兵数量，询问次数

int m,n;//询问区间

scanf("%d%d",&num,&query\_num);

for(int i=1;i<=num;++i) //输入信息处理

{

scanf("%d",&maxsum[0][i]);

minsum[0][i]=maxsum[0][i];

}

RMQ(num);

while(query\_num--)//直接查询，此时时间复杂度为O(1)!!!

{

scanf("%d%d",&m,&n);

int k=(int)(log(n-m+1.0)/log(2.0));//找到被2的n次方覆盖的区间

int maxres=max(maxsum[k][m],maxsum[k][n-(1<<k)+1]);

int minres=min(minsum[k][m],minsum[k][n-(1<<k)+1]);

printf("%d\n", maxres-minres);

}

return 0;

}

**线段树**

**主要应用**

(1)：区间最值查询问题（见模板1）同 **RMQ**

(2)：连续区间修改或者单节点更新的动态查询问题 （见模板2）

(3)：多维空间的动态查询 （见模板3）

**Problem Description（模板2**连续区间修改或者单节点更新的动态查询问题**）**

C国的死对头A国这段时间正在进行军事演习，所以C国间谍头子Derek和他手下Tidy又开始忙乎了。A国在海岸线沿直线布置了N个工兵营 地,Derek和Tidy的任务就是要监视这些工兵营地的活动情况。由于采取了某种先进的监测手段，所以每个工兵营地的人数C国都掌握的一清二楚,每个工 兵营地的人数都有可能发生变动，可能增加或减少若干人手,但这些都逃不过C国的监视。  
中央情报局要研究敌人究竟演习什么战术,所以Tidy要随时向Derek汇报某一段连续的工兵营地一共有多少人,例如Derek问:“Tidy,马上汇报 第3个营地到第10个营地共有多少人!”Tidy就要马上开始计算这一段的总人数并汇报。但敌兵营地的人数经常变动，而Derek每次询问的段都不一样， 所以Tidy不得不每次都一个一个营地的去数，很快就精疲力尽了，Derek对Tidy的计算速度越来越不满:"你个死肥仔，算得这么慢，我炒你鱿 鱼!”Tidy想：“你自己来算算看，这可真是一项累人的工作!我恨不得你炒我鱿鱼呢!”无奈之下，Tidy只好打电话向计算机专家 Windbreaker求救,Windbreaker说：“死肥仔，叫你平时做多点acm题和看多点算法书，现在尝到苦果了吧!”Tidy说："我知错 了。。。"但Windbreaker已经挂掉电话了。Tidy很苦恼，这么算他真的会崩溃的，聪明的读者，你能写个程序帮他完成这项工作吗？不过如果你的 程序效率不够高的话，Tidy还是会受到Derek的责骂的.

**Input**

第一行一个整数T，表示有T组数据。  
每组数据第一行一个正整数N（N<=50000）,表示敌人有N个工兵营地，接下来有N个正整数,第i个正整数ai代表第i个工兵营地里开始时有ai个人（1<=ai<=50）。  
接下来每行有一条命令，命令有4种形式：  
(1) Add i j,i和j为正整数,表示第i个营地增加j个人（j不超过30）  
(2)Sub i j ,i和j为正整数,表示第i个营地减少j个人（j不超过30）;  
(3)Query i j ,i和j为正整数,i<=j，表示询问第i到第j个营地的总人数;  
(4)End 表示结束，这条命令在每组数据最后出现;  
每组数据最多有40000条命令

**Output**

对第i组数据,首先输出“Case i:”和回车,  
对于每个Query询问，输出一个整数并回车,表示询问的段中的总人数,这个数保持在int以内。

**Sample Input**

1

10

1 2 3 4 5 6 7 8 9 10

Query 1 3

Add 3 6

Query 2 7

Sub 10 2

Add 6 3

Query 3 10

End

**Sample Output**

Case 1:

6

33

59

思路：这道题目考查的是对区间的一个动态操作，所以需要用到数据结构中的线段树，本题的难点便在于线段树的构建。

#include"iostream"

#include"cstdlib"

using namespace std;

#define maxn 50010//指定所要构造的线段树节点的最大值

struct Node

{

int sum;

int l, r;

}node[maxn\*4];

int num[maxn];

int ans;

void build\_tree(int rt, int ll, int rr)

{

if (ll == rr)

{

node[rt].l = ll;

node[rt].r = rr;

node[rt].sum = num[ll];

return;

}

node[rt].l = ll;

node[rt].r = rr;

int mid = (ll + rr) / 2;

build\_tree(rt \* 2, ll, mid);

build\_tree(rt \* 2 + 1, mid + 1, rr);

node[rt].sum = node[rt \* 2].sum + node[rt \* 2 + 1].sum;//sum值更新到父亲节点

}

void update(int rt, int pos, int value)

{

if (node[rt].l == pos&&node[rt].r == pos)

{

node[rt].sum += value;

return;

}

int mid = (node[rt].l + node[rt].r) / 2;

if (pos <= mid)

{

update(rt \* 2, pos, value);

}

else

{

update(rt \* 2 + 1, pos, value);

}

node[rt].sum = node[rt \* 2].sum + node[rt \* 2 + 1].sum;

}

void query(int rt, int l, int r)

{

int mid = (node[rt].l + node[rt].r) / 2;

if (node[rt].l == l&&node[rt].r == r)

{

ans += node[rt].sum;

}

else if (l > mid){

query(rt \* 2 + 1, l, r);

}

else if (mid >= r){

query(rt \* 2, l, r);

}

else{

query(rt \* 2, l, mid);

query(rt \* 2 + 1, mid + 1, r);

}

}

int main()

{

int t,n;

int u = 0;

cin >> t;

char str[12];

while (t--)

{

cin >> n;

for (int i = 1; i <= n; i++)

{

cin >> num[i];

}

build\_tree(1, 1, n);

cout << "Case " << ++u <<":"<< endl;

while (cin>>str)

{

int x, y;

cin >> x >> y;

if (str[0] == 'E'){

break;

}

if (str[0] == 'Q')

{

ans = 0;

query(1, x, y);

cout << ans << endl;

}

else if (str[0] == 'A')

{

update(1, x, y);

}

else if (str[0] == 'S')

{

update(1,x,-y);

}

}

}

return 0;

}

**字典树**

#include <iostream>

using namespace std;

#include <stdlib.h>

#include <string.h>

#include <stdio.h>

typedef struct Trie{

int v;

Trie \*next[26];

}Trie;

Trie root;

void createTrie(char \*str)

{

int len = strlen(str);

Trie \*p = &root, \*q;

for(int i=0; i<len; ++i)

{

int id = str[i]-'a';

if(p->next[id] == NULL)

{

q = (Trie \*)malloc(sizeof(root));

q->v = 1;

for(int j=0; j<26; ++j)

q->next[j] = NULL;

p->next[id] = q;

p = p->next[id];

}

else

{

p->next[id]->v++;

p = p->next[id];

}

}

}

int findTrie(char \*str)

{

int len = strlen(str);

Trie \*p = &root;

for(int i=0; i<len; ++i)

{

int id = str[i]-'a';

p = p->next[id];

if(p == NULL)

return 0;

}

return p->v;

}

int main()

{

//freopen("input.txt", "r", stdin);

char str[15];

int i;

for(i=0; i<26; ++i)

root.next[i] = NULL;

while(gets(str) && str[0]!='\0')

createTrie(str);

memset(str, 0, sizeof(str));

while(scanf("%s", str) != EOF)

{

int ans = findTrie(str);

printf("%d\n", ans);

}

return 0;

}

**模板：**

**#define MAX 26//字符集大小**

**typedef struct TrieNode**

**{**

**int nCount;//记录该字符出现次数**

**struct TrieNode\* next[MAX];**

**}TrieNode;**

**TrieNode Memory[1000000];**

**int allocp=0;**

**/\*初始化\*/**

**void InitTrieRoot(TrieNode\* \*pRoot)**

**{**

**\*pRoot=NULL;**

**}**

**/\*创建新结点\*/**

**TrieNode\* CreateTrieNode()**

**{**

**int i;**

**TrieNode \*p;**

**p=&Memory[allocp++];**

**p->nCount=1;**

**for(i=0;i<MAX;i++)**

**{**

**p->next[i]=NULL;**

**}**

**return p;**

**}**

**/\*插入\*/**

**void InsertTrie(TrieNode\* \*pRoot,char \*s)**

**{**

**inti,k;**

**TrieNode\*p;**

**if(!(p=\*pRoot))**

**{**

**p=\*pRoot=CreateTrieNode();**

**}**

**i=0;**

**while(s[i])**

**{**

**k=s[i++]-'a';//确定branch**

**if(!p->next[k])**

**p->next[k]=CreateTrieNode();**

**p->next[k]->nCount++;**

**p=p->next[k];**

**}**

**}**

**//查找**

**int SearchTrie(TrieNode\* \*pRoot,char \*s)**

**{**

**TrieNode \*p;**

**int i,k;**

**if(!(p=\*pRoot))**

**{**

**return0;**

**}**

**i=0;**

**while(s[i])**

**{**

**k=s[i++]-'a';**

**if(p->next[k]==NULL) return 0;**

**p=p->next[k];**

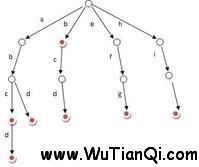
**}**

**return p->nCount;**

**}**

## [字典树（讲解+模版）](http://www.cnblogs.com/tanky_woo/archive/2010/09/24/1833717.html)

又称**单词查找树**，[Trie树](http://www.wutianqi.com/?p=1359)，是一种树形结构，是一种哈希树的变种。典型应用是用于统计，排序和保存大量的字符串（但不仅限于字符串），所以经常被搜索引擎系统用于文本词频统计。它的优点是：利用字符串的公共前缀来节约存储空间，最大限度地减少无谓的字符串比较，查询效率比哈希表高。



字典树与字典很相似,当你要查一个单词是不是在字典树中,首先看单词的第一个字母是不是在字典的第一层,如果不在,说明字典树里没有该单词,如果在 就在该字母的孩子节点里找是不是有单词的第二个字母,没有说明没有该单词,有的话用同样的方法继续查找.字典树不仅可以用来储存字母,也可以储存数字等其 它数据。

**Trie的数据结构定义：**

[IMG_257](http://www.cnblogs.com/tanky_woo/archive/2010/09/24/javascript:void(0);)

#define MAX 26  
typedef struct Trie     
{     
    Trie \*next[MAX];     
    int v;   //根据需要变化  
};     
   
Trie \*root;

[IMG_258](http://www.cnblogs.com/tanky_woo/archive/2010/09/24/javascript:void(0);)

next是表示每层有多少种类的数，如果只是小写字母，则26即可，若改为大小写字母，则是52，若再加上数字，则是62了，这里根据题意来确定。  
v可以表示一个字典树到此有多少相同前缀的数目，这里根据需要应当学会自由变化。

**Trie的查找（最主要的操作）：**  
(1) 每次从根结点开始一次搜索；  
(2) 取得要查找关键词的第一个字母，并根据该字母选择对应的子树并转到该子树继续进行检索； 　　(3) 在相应的子树上，取得要查找关键词的第二个字母,并进一步选择对应的子树进行检索。 　　  
(4) 迭代过程…… 　　  
(5) 在某个结点处，关键词的所有字母已被取出，则读取附在该结点上的信息，即完成查找。

这里给出生成字典树和查找的**模版**：  
生成字典树：

[IMG_259](http://www.cnblogs.com/tanky_woo/archive/2010/09/24/javascript:void(0);)

void createTrie(char \*str)  
{  
    int len = strlen(str);  
    Trie \*p = root, \*q;  
    for(int i=0; i<len; ++i)  
    {  
        int id = str[i]-'0';  
        if(p->next[id] == NULL)  
        {  
            q = (Trie \*)malloc(sizeof(Trie));  
            q->v = 1;    //初始v==1  
            for(int j=0; j<MAX; ++j)  
                q->next[j] = NULL;  
            p->next[id] = q;  
            p = p->next[id];  
        }  
        else  
        {  
            p->next[id]->v++;  
            p = p->next[id];  
        }  
    }  
    p->v = -1;   //若为结尾，则将v改成-1表示  
}

[IMG_260](http://www.cnblogs.com/tanky_woo/archive/2010/09/24/javascript:void(0);)

接下来是查找的过程了：

[IMG_261](http://www.cnblogs.com/tanky_woo/archive/2010/09/24/javascript:void(0);)

int findTrie(char \*str)  
{  
    int len = strlen(str);  
    Trie \*p = root;  
    for(int i=0; i<len; ++i)  
    {  
        int id = str[i]-'0';  
        p = p->next[id];  
        if(p == NULL)   //若为空集，表示不存以此为前缀的串  
            return 0;  
        if(p->v == -1)   //字符集中已有串是此串的前缀  
            return -1;  
    }  
    return -1;   //此串是字符集中某串的前缀  
}

[IMG_262](http://www.cnblogs.com/tanky_woo/archive/2010/09/24/javascript:void(0);)

对于上述动态字典树，有时会超内存，比如 [HDOJ 1671 Phone List](http://acm.hdu.edu.cn/showproblem.php?pid=1671)，这是就要记得释放空间了：

[IMG_263](http://www.cnblogs.com/tanky_woo/archive/2010/09/24/javascript:void(0);)

int dealTrie(Trie\* T)  
{  
    int i;  
    if(T==NULL)  
        return 0;  
    for(i=0;i<MAX;i++)  
    {  
        if(T->next[i]!=NULL)  
            deal(T->next[i]);  
    }  
    free(T);  
    return 0;  
}

## [IMG_264](http://www.cnblogs.com/tanky_woo/archive/2010/09/24/javascript:void(0);)1645: IMMEDIATE DECODABILITY

时间限制: 1 Sec  内存限制: 64 MB  
提交: 1  解决: 1  
[[提交](http://cst2.jlnu.edu.cn/oj/submitpage.php?id=1645)][[状态](http://cst2.jlnu.edu.cn/oj/problemstatus.php?id=1645)][[讨论版](http://cst2.jlnu.edu.cn/oj/bbs.php?pid=1645)]

## 题目描述

An encoding of a set of symbols is said to be immediately decodable if no code for one symbol is the prefix of a code for another symbol. We will assume for this problem that all codes are in binary, that no two codes within a set of codes are the same, that each code has at least one bit and no more than ten bits, and that each set has at least two codes and no more than eight. Examples: Assume an alphabet that has symbols {A, B, C, D} The following code is immediately decodable: A:01 B:10 C:0010 D:0000 but this one is not: A:01 B:10 C:010 D:0000 (Note that A is a prefix of C)

## 输入

Write a program that accepts as input a series of groups of records from standard input. Each record in a group contains a collection of zeroes and ones representing a binary code for a different symbol. Each group is followed by a single separator record containing a single 9; the separator records are not part of the group. Each group is independent of other groups; the codes in one group are not related to codes in any other group (that is, each group is to be processed independently).

## 输出

For each group, your program should determine whether the codes in that group are immediately decodable, and should print a single output line giving the group number and stating whether the group is, or is not, immediately decodable.

## 样例输入

01

10

0010

0000

9

01

10

010

0000

9

## 样例输出

Set 1 is immediately decodable

Set 2 is not immediately decodable

## 提示

## 来源

#include<cstdio>

#include<cstring>

#include<cstdlib>

#include<iostream>

using namespace std;

char word[15][20];

struct Tree

{

int num;//判断此节点是否为空的（就是一棵树的结束点）

Tree \*next[2];//这个是这棵树的子节点

Tree()

{

num=0;

for(int i=0;i<2;i++)

{

next[i]=NULL;

}

}//这是一棵树

}\*root;//建立一个树根

void insert(Tree \*p,char \*s)

{

int i=0;

while(s[i])//当这个字符串的某一个元素不为空的时候

{

int x=s[i]-'0';//看这个元素在哪里？

if(p->next[x]==NULL)//然后看这棵树有没有这个元素如果没有

{

p->next[x]=new Tree();//就在这棵树的此节点重新建立一棵子树

}

p=p->next[x];//p指向他的子树，这一点你就可以知道了所有字符串的元素原来他们都是主仆关系没有相同的等级关系！

p->num++;

i++;//继续遍历

}

}//插入字符

int find(Tree \*p,char \*s)

{

int i=0,ans=0;//寻找树的元素，ans是结果！

while(s[i])//当他不为空的时候就是字符串还有的时候

{

int n=s[i]-'0';//找到他在这个树的节点

if(p->next[n])//如果节点不为空

{

p=p->next[n];//p就指向该节点

ans=p->num;//结果应该为1

i++;

}

else

{

return 0;

}

}

return ans;

}

int deltree(Tree\* T)

{

int i;

if(T==NULL)

return 0;

for(i=0;i<2;i++)

{

if(T->next[i]!=NULL)

deltree(T->next[i]);

}

free(T);

return 0;

}

int main()

{

int times=1,i=0,j,success=1;

char s[20];

root=new Tree();

while(cin>>s)

{

if(strcmp(s,"9")==0)

{

for(j=0;j<i;j++)

{

// cout<<find(root,word[j])<<endl;

if(find(root,word[j])>1)

{

success=0;

break;

}

}

if(success)

{

printf("Set %d is immediately decodable\n",times++);

}

else

{

printf("Set %d is not immediately decodable\n",times++);

}

deltree(root);

root=new Tree();

success=1;

i=0;

}

else

{

strcpy(word[i],s);

insert(root,word[i]);

i++;

}

}

}

**最长公共子串（LCS）**

#include<iostream>

#include<cstring>

#include<vector>

using namespace std;

//str1为横向，str2这纵向

const string LCS(const string& str1,const string& str2){

int xlen=str1.size(); //横向长度

vector<int> tmp(xlen); //保存矩阵的上一行

vector<int> arr(tmp); //当前行

int ylen=str2.size(); //纵向长度

int maxele=0; //矩阵元素中的最大值

int pos=0; //矩阵元素最大值出现在第几列

for(int i=0;i<ylen;i++){

string s=str2.substr(i,1);

arr.assign(xlen,0); //数组清0

for(int j=0;j<xlen;j++){

if(str1.compare(j,1,s)==0){

if(j==0)

arr[j]=1;

else

arr[j]=tmp[j-1]+1;

if(arr[j]>maxele){

maxele=arr[j];

pos=j;

}

}

}

tmp.assign(arr.begin(),arr.end());

}

string res=str1.substr(pos-maxele+1,maxele);

return res;

}

int main(){

string str1("21232523311324");

string str2("312123223445");

string lcs=LCS(str1,str2);

cout<<lcs<<endl;

return 0;

# }

# [【模板】各种欧几里得](http://blog.csdn.net/libin56842/article/details/9364231)

int gcd(int n,int m)//n>m

{

//最大公约数

int r;

while(m)

{

r = n%m;

n = m;

m = r;

}

return n;

}

int kgcd(int a,int b)

{

if(!a) return b;

if(!b) return a;

if(!(a&1) && !(b&1))

return kgcd(a>>1,b>>1)<<1;

else if(!(b&1)) return kgcd(a,b>>1);

else if(!(a&1)) return kgcd(a>>1,b);

else return kgcd(abs(a-b),min(a,b));

}

//扩展欧几里得

//求方程ax+by+c = 0有多少整数解

int extgcd(int a,int b,int &x,int &y)

{

if(!b)

{

x=1;

y=0;

return a;

}

int d = extgcd(b,a%b,x,y);

int t = x;

x=y;

y=t-a/b\*y;

return d;

}

**数据结构**

**1.优先队列**

书：189

**AC自动机**

**拓扑排序**

# 确定比赛名次

**Time Limit: 2000/1000 MS (Java/Others)    Memory Limit: 65536/32768 K (Java/Others)  
Total Submission(s): 22765    Accepted Submission(s): 9231**

Problem Description

有 N个比赛队（1<=N<=500），编号依次为1，2，3，。。。。，N进行比赛，比赛结束后，裁判委员会要将所有参赛队伍从前往后依次排 名，但现在裁判委员会不能直接获得每个队的比赛成绩，只知道每场比赛的结果，即P1赢P2，用P1，P2表示，排名时P1在P2之前。现在请你编程序确定 排名。

Input

输入有若干组，每组中的第一行为二个数N（1<=N<=500），M；其中N表示队伍的个数，M表示接着有M行的输入数据。接下来的M行数据中，每行也有两个整数P1，P2表示即P1队赢了P2队。

Output

给出一个符合要求的排名。输出时队伍号之间有空格，最后一名后面没有空格。  
  
其他说明：符合条件的排名可能不是唯一的，此时要求输出时编号小的队伍在前；输入数据保证是正确的，即输入数据确保一定能有一个符合要求的排名。

Sample Input

4 3

1 2

2 3

4 3

Sample Output

1 2 4 3

#include <iostream>

using namespace std;

int map[502][502], indegree[502], m, n, pur[502];

**void topsort()**

**{**

**int i, j, k=1;**

**for(i=1; i<=n; i++)**

**{**

**for(j=1; j<=n; j++)**

**{**

**if(indegree[j]==0)**

**{**

**indegree[j]--;**

**pur[k++] = j;**

**for(int x=1; x<=n; x++)**

**if(map[j][x])**

**indegree[x]--;**

**break;**

**}**

**if(j>n)**

**{**

**cout<<"存在环"<<endl;**

**return ;**

**}**

**}**

**}**

**}**

void main()

{

int i, j;

while(cin>>n>>m)

{

memset(map,0,sizeof(map));

memset(indegree,0,sizeof(indegree));

int a, b;

for(i=1; i<=m; i++)

{

cin>>a>>b;

if(!map[a][b])

{

map[a][b] = 1;

indegree[b]++;

}

}

topsort();

for(i=1; i<=n; i++)

if(i!=n)

cout<<pur[i]<<" ";

else

cout<<pur[i]<<endl;

}

}