第 1 章　算法概述

算法分析题 1

1-1　函数的渐近表达式。

求下列函数的渐近表达式：

$3n^2+10n$；$n^2/10+2^n$；$21+1/n$；$\log n^3$；$10\log 3^n$。

分析与解答：

$3n^2+10n=O(n^2)$；　　$\log n^3=O(\log n)$；

$n^2/10+2^n=O(2^n)$；　　$10\log 3^n=O(n)$。

$21+1/n=O(1)$；

1-2　$O(1)$和 $O(2)$的区别。

分析与解答：根据符号 O 的定义易知 $O(1)=O(2)$。用 $O(1)$或 $O(2)$表示同一个函数时，差别仅在于其中的常数因子。

1-3　按渐近阶排列表达式。

按照渐近阶从低到高的顺序排列以下表达式：$4n^2$，$\log n$，3^n，$20n$，2，$n^{2/3}$。又 $n!$ 应该排在哪一位？

分析与解答：按渐近阶从低到高，函数排列顺序如下：2，$\log n$，$n^{2/3}$，$20n$，$4n^2$，3^n，$n!$。

1-4　算法效率。

（1）假设某算法在输入规模为 n 时的计算时间为 $T(n)=3\times 2^n$。在某台计算机上实现并完成该算法的时间为 t 秒。现有另一台计算机，其运行速度为第一台的 64 倍，那么在这台新机器上用同一算法在 t 秒内能解输入规模为多大的问题？

（2）若上述算法的计算时间改进为 $T(n)=n^2$，其余条件不变，则在新机器上用 t 秒时间能解输入规模为多大的问题？

（3）若上述算法的计算时间进一步改进为 $T(n)=8$，其余条件不变，那么在新机器上用 t 秒时间能解输入规模为多大的问题？

分析与解答：

（1）设新机器用同一算法在 t 秒内能解输入规模为 $n1$ 的问题。因此有：$t=3\times 2^n=3\times 2^{n1}/64$，解得 $n1=n+6$。

（2）$n1^2=64n^2 \Rightarrow n1=8n$。

（3）由于 $T(n)$=常数，因此算法可解任意规模的问题。

1-5　硬件效率。

硬件厂商 XYZ 公司宣称他们最新研制的微处理器运行速度为其竞争对手 ABC 公司同类产品的 100 倍。对于计算复杂性分别为 n、n^2、n^3 和 $n!$ 的各算法，若用 ABC 公司的计算机在 1 小时内能解输入规模为 n 的问题，那么用 XYZ 公司的计算机在 1 小时内分别能解输入

规模为多大的问题？

分析与解答：

$n'=100n$　　$n'^3=100n^3 \Rightarrow n'=\sqrt[3]{100n}=4.64n$

$n'^2=100n^2 \Rightarrow n'=10n$　　$n'!=100n! \Rightarrow n'<n+\log 100=n+6.64$

1-6　函数渐近阶。

对于下列各组函数 $f(n)$和 $g(n)$，确定 $f(n)=O(g(n))$或 $f(n)=\Omega(g(n))$或 $f(n)=\theta(g(n))$，并简述理由。

（1）$f(n)=\log n^2$；　$g(n)=\log n+5$　　（5）$f(n)=10$；　$g(n)=\log 10$

（2）$f(n)=\log n^2$；　$g(n)=\sqrt{n}$　　（6）$f(n)=\log^2 n$；　$g(n)=\log n$

（3）$f(n)=n$；　$g(n)=\log^2 n$　　（7）$f(n)=2^n$；　$g(n)=100n^2$

（4）$f(n)=n\log n+n$；　$g(n)=\log n$　　（8）$f(n)=2^n$；　$g(n)=3^n$

分析与解答：

（1）$\log n^2=\theta(\log n+5)$　　（5）$10=\theta(\log 10)$

（2）$\log n^2=O(\sqrt{n})$　　（6）$\log^2 n=\Omega(\log n)$

（3）$n=\Omega(\log^2 n)$　　（7）$2^n=\Omega(100n^2)$

（4）$n\log n+n=\Omega(\log n)$　　（8）$2^n=O(3^n)$

1-7　$n!$的阶。

证明：$n!=o(n^n)$。

分析与解答：Stirling's approximation：

$$n!=\sqrt{2\pi n}\left(\frac{n}{\mathrm{e}}\right)^n\left[1+\theta\left(\frac{1}{n}\right)\right]$$

$$\lim_{n\to\infty} n!/n^n=\frac{\sqrt{2\pi n}\left[1+\theta\left(\frac{1}{n}\right)\right]}{\mathrm{e}^n}=0$$

1-8　$3n+1$ 问题。

下面的算法片段用于确定 n 的初始值。试分析该算法片段所需计算时间的上界和下界。

```
while(n>1)
    if(odd(n))
        n = 3*n+1;
    else
        n = n/2;
```

分析与解答：该算法表述的是著名的 $3n+1$ 问题。在最坏情况下，该算法的计算时间下界显然为 $\Omega(\log n)$。

算法的计算时间上界至今未知。算法是否在有限时间内结束，至今还是一个悬而未决的问题。日本学者米田信夫曾对 10^{13} 内的所有自然数验证上述算法均在有限步结束。人们猜测，对所有自然数，上述算法均在有限步结束，但无法给出理论证明，因此也无法分析上述算法的计算时间上界。这个猜测就成为著名的 $3n+1$ 猜想，也称为 Collatz 猜想。

1-9　平均情况下的计算时间复杂性。

证明：如果一个算法在平均情况下的计算时间复杂性为 $\theta(f(n))$，则该算法在最坏情况下所需的计算时间为 $\Omega(f(n))$。

分析与解答：

$$T_{\text{avg}}(N)=\sum_{I\in D_N}P(I)T(N,I)\leqslant\sum_{I\in D_N}P(I)\max_{I'\in D_N}T(N,I')$$
$$=T(N,I^*)\sum_{I\in D_N}P(I)=T(N,I^*)=T_{\max}(N)$$

因此，$T_{\max}(N)=\Omega(T_{\text{avg}}(N))=\Omega(\theta(f(n)))=\Omega(f(n))$。

算法实现题 1

1-1 统计数字问题。

问题描述：一本书的页码从自然数 1 开始顺序编码直到自然数 n。书的页码按照通常的习惯编排，每个页码不含多余的前导数字 0。例如，第 6 页用数字 6 表示而不是 06 或 006 等。数字计数问题要求对给定书的总页码 n，计算书的全部页码分别用到多少次数字 0、1、2、…、9。

算法设计：给定表示书的总页码的十进制整数 n（$1\leqslant n\leqslant 10^9$），计算书的全部页码中分别用到多少次数字 0、1、2、…、9。

数据输入：输入数据由文件名为 input.txt 的文本文件提供。每个文件只有 1 行，给出表示书的总页码的整数 n。

结果输出：将计算结果输出到文件 output.txt。输出文件共 10 行，在第 k（k=1, 2, …, 10）行输出页码中用到数字 k–1 的次数。

输入文件示例	输出文件示例
input.txt	output.txt
11	1
	4
	1
	1
	1
	1
	1
	1
	1
	1

分析与解答：考察由 0, 1, 2, …, 9 组成的所有 n 位数。从 n 个 0 到 n 个 9 共有 10^n 个 n 位数。在这 10^n 个 n 位数中，0, 1, 2, …, 9 每个数字使用次数相同，设为 $f(n)$。$f(n)$满足如下递归式：

$$f(n)=\begin{cases}10f(n-1)+10^{n-1} & n>1\\ 1 & n=1\end{cases}$$

由此可知，$f(n)=n10^{n-1}$。

据此，可从高位向低位进行统计，再减去多余的 0 的个数即可。

1-2　字典序问题。

问题描述：在数据加密和数据压缩中常需要对特殊的字符串进行编码。给定的字母表 A 由 26 个小写英文字母组成，即 A={a, b, …, z}。该字母表产生的升序字符串是指字符串中字母从左到右出现的次序与字母在字母表中出现的次序相同，且每个字符最多出现 1 次。例如，a、b、ab、bc、xyz 等字符串都是升序字符串。现在对字母表 A 产生的所有长度不超过 6 的升序字符串按照字典序排列并编码如下。

1	2	…	26	27	28	…
a	b	…	z	ab	ac	…

对于任意长度不超过 6 的升序字符串，迅速计算出它在上述字典中的编码。

算法设计：对于给定的长度不超过 6 的升序字符串，计算它在上述字典中的编码。

数据输入：输入数据由文件名为 input.txt 的文本文件提供。文件的第 1 行是一个正整数 k，表示接下来有 k 行。在接下来的 k 行中，每行给出一个字符串。

结果输出：将计算结果输出到文件 output.txt。文件有 k 行，每行对应一个字符串的编码。

输入文件示例	输出文件示例
input.txt	output.txt
2	1
a	2
b	

分析与解答：考察一般情况下长度不超过 k 的升序字符串。

设以第 i 个字符打头的长度不超过 k 的升序字符串个数为 $f(i, k)$，长度不超过 k 的升序字符串总个数为 $g(k)$，则 $g(k)=\sum_{i=1}^{26} f(i,k)$。易知

$$f(i,1)=1 \qquad g(1)=\sum_{i=1}^{26} f(i,1)=26$$

$$f(i,2)=\sum_{j=i+1}^{26} f(j,1)=26-i \qquad g(2)=\sum_{i=1}^{26} f(i,2)=\sum_{i=1}^{26}(26-i)=325$$

一般情况下有

$$f(i,k)=\sum_{j=i+1}^{26} f(j,k-1) \qquad g(k)=\sum_{i=1}^{26} f(i,k)=\sum_{i=1}^{26}\sum_{j=i+1}^{26} f(j,k-1)$$

据此可计算出每个升序字符串的编码。

1-3　最多约数问题。

问题描述：正整数 x 的约数是能整除 x 的正整数。正整数 x 的约数个数记为 div(x)。例如，1、2、5、10 都是正整数 10 的约数，且 div(10)=4。设 a 和 b 是 2 个正整数，$a \leqslant b$，找出 a 和 b 之间约数个数最多的数 x。

算法设计：对于给定的 2 个正整数 $a \leqslant b$，计算 a 和 b 之间约数个数最多的数。

数据输入：输入数据由文件名为 input.txt 的文本文件提供。文件的第 1 行有 2 个正整数 a 和 b。

结果输出：若找到的 a 和 b 之间约数个数最多的数是 x，则将 div(x)输出到文件 output.txt。

输入文件示例	输出文件示例
input.txt	output.txt

分析与解答：设正整数 x 的质因子分解为

$$x = p_1^{N_1} \ p_2^{N_2} \cdots p_k^{N_k}$$

则

$$\mathrm{div}(x)=(N_1+1)(N_2+1)\cdots(N_k+1)$$

搜索区间[a, b]中数的质因子分解。

primes()函数用于产生质数：

```
void primes(){
   bool get[MAXP+1];
   for(int i=2; i <= MAXP; i++)
      get[i]=true;
   for(i=2; i <= MAXP; i++)
      if(get[i]) {
         int j=i+i;
         while(j <= MAXP) {
            get[j]=false;
            j += i;
         }
      }
   for(int ii=2, j=0; ii <= MAXP; ii++)
      if(get[ii])
         prim[++j]=ii;
}
```

search()函数用于搜索最多约数：

```
void search(int from,int tot,int num,int low,int up) {
   if(num >= 1)
      if((tot > max) || ((tot == max) && (num < numb))) {
         max=tot;
         numb=num;
      }
      if((low == up) && (low > num))
         search(from, tot*2, num*low, 1, 1);
      for(int i=from; i <= PCOUNT; i++) {
         if(prim[i] > up)
            return;
         else {
            int j = prim[i], x = low1, y = up, n = num, t = tot, m = 1;
            while(true) {
               m++;
               t += tot;
               x /= j;
               y /= j;
               if(x == y)
                  break;
               n *= j;
               search(i+1, t, n, x+1, y);
```

```
        }
      }
      m = 1 << m;
      if(tot < max/m)
         return;
   }
}
```

实现算法的主函数如下。

```
int main() {
   primes();
   cin >> l >> u;
   if((l == 1) && (u == 1)) {
      max = 1;
      numb = 1;
   }
   else {
      max = 2;
      numb = l;
      search(1, 1, 1, l, u);
   }
   cout << max << endl;
   return 0;
}
```

1-4 金币阵列问题。

问题描述：有 $m \times n$（$m \leqslant 100$，$n \leqslant 100$）枚金币在桌面上排成一个 m 行 n 列的金币阵列。每枚金币或正面朝上或背面朝上。用数字表示金币状态，0 表示金币正面朝上，1 表示金币背面朝上。

金币阵列游戏的规则是：① 每次可将任一行金币翻过来放在原来的位置上；② 每次可任选 2 列，交换这 2 列金币的位置。

算法设计：给定金币阵列的初始状态和目标状态，计算按金币游戏规则，将金币阵列从初始状态变换到目标状态所需的最少变换次数。

数据输入：由文件 input.txt 给出输入数据。文件中有多组数据。文件的第 1 行有 1 个正整数 k，表示有 k 组数据。每组数据的第 1 行有 2 个正整数 m 和 n。以下 m 行是金币阵列的初始状态，每行有 n 个数字表示该行金币的状态，0 表示正面朝上，1 表示背面朝上。接着的 m 行是金币阵列的目标状态。

结果输出：将计算出的最少变换次数按照输入数据的次序输出到文件 output.txt。相应数据无解时，输出−1。

输入文件示例	输出文件示例
input.txt	output.txt
2	2
4 3	−1
1 0 1	
0 0 0	
1 1 0	

1 0 1
1 0 1
1 1 1
0 1 1
1 0 1
4 3
1 0 1
0 0 0
1 0 0
1 1 1
1 1 0
1 1 1
0 1 1
1 0 1

分析与解答：枚举初始状态每列变换为目标状态第 1 列的情况。算法描述如下。

```
int  k, n, m, count, best;
int  b0[Size+1][Size+1], b1[Size+1][Size+1], b[Size+1][Size+1];
bool  found;
int main() {
   cin >> k;
   for(int i=1; i <= k; i++) {
      cin >> n >> m;
      for(int x=1; x <= n; x++) {
         for(int y=1; y <= m; y++)
            cin >> b0[x][y];
         for(x=1; x <= n; x++)
            for(int y=1; y <= m; y++)
               cin >> b1[x][y];
         acpy(b, b1);
         best = m+n+1;
         for(int j=1; j <= m; j++) {
            acpy(b1, b);
            count = 0;
            trans2(1, j);
            for(int p=1; p <= n; p++)
               if(b0[p][1] != b1[p][1])
                  trans1(p);
            for(p=1; p <= m; p++) {
               found = false;
               for(int q=p; q <= m; q++) {
                  if(same(p, q)) {
                     trans2(p, q);
                     found = true;
                     break;
                  }
                  if(!found)
                     break;
               }
```

```
            }
            if(found && count < best)
               best = count;
         }
         if(best < m+n+1)
            cout << best << endl;
         else
            cout << 1 << endl;
      }
   }
   return 0;
}
```

其中，trans1()函数模拟行翻转变换，trans2()函数模拟列交换变换。

```
void trans1(int x) {
   for(int i=1; i <= m; i++)
      b1[x][i] = b1[x][i]^1;
   count++;
}
void trans2(int x, int y) {
   for(int i=1; i <= n; i++)
      Swap(b1[i][x], b1[i][y]);
   if(x != y)
      count++;
}
bool same(int x, int y) {
   for(int i=1; i <= n; i++)
      if(b0[i][x] != b1[i][y])
         return false;
   return true;
}
void acpy(int a[Size+1][Size+1], int b[Size+1][Size+1]) {
   for(int i=1; i <= n; i++)
      for(int j=1; j <= m; j++)
         a[i][j] = b[i][j];
}
```

1-5 最大间隙问题。

问题描述：最大间隙问题：给定 n 个实数 x_1、x_2、…、x_n，求这 n 个数在实轴上相邻两个数之间的最大差值。假设对任何实数的下取整函数耗时 $O(1)$，设计解最大间隙问题的线性时间算法。

算法设计：对于给定的 n 个实数 x_1、x_2、…、x_n，计算它们的最大间隙。

数据输入：输入数据由文件名为 input.txt 的文本文件提供。文件的第 1 行有 1 个正整数 n。接下来的 1 行中有 n 个实数 x_1、x_2、…、x_n。

结果输出：将找到的最大间隙输出到文件 output.txt。

输入文件示例	输出文件示例
input.txt	output.txt
5	3.2

2.3 3.1 7.5 1.5 6.3

分析与解答：用鸽舍原理设计最大间隙问题的线性时间算法如下。

```
double maxgap(int n, double *x) {
   double  minx = x[mini(n, x)], maxx = x[maxi(n, x)];
   // 用 n-2 个等间距点分割区间[minx, maxx]，产生 n-1 个桶
   // 每个桶的 i 中用 high[i]和 low[i]分别存储分配给桶 i 的数中的最大数和最小数
   int  *count = new int[n+1];
   double  *low = new double[n+1];
   double  *high = new double[n+1];
   // 桶初始化
   for(int i=1; i <= n1; i++) {
      count[i] = 0;
      low[i] = maxx;
      high[i] = minx;
   }
   // 将 n 个数置于 n-1 个桶中
   for(i=1; i <= n; i++) {
      int  bucket = int ((n-1)*(x[i]-minx) / (maxx-minx))+1;
      count[bucket]++;
      if(x[i] < low[bucket])
         low[bucket] = x[i];
      if(x[i] > high[bucket])
         high[bucket] = x[i];
   }
   // 此时，除了 maxx 和 minx 的 n-2 个数被置于 n-1 个桶中。由鸽舍原理即知，至少有一个桶是空的
   // 这意味着最大间隙不会出现在同一桶中的两个数之间对每个桶做一次线性扫描即可找出最大间隙
   double  tmp = 0, left = high[1];
   for(i=2; i <= n1; i++) {
      if(count[i]) {
         double  thisgap = low[i]-left;
         if(thisgap > tmp)
            tmp = thisgap;
         left = high[i];
      }
   }
   return tmp;
}
```

其中，mini()函数和 maxi()函数分别计算数组中最小元素和最大元素的下标。

```
template<class T>
int mini(int n, T* x) {
   T  tmp = x[1];
   for(int i=1, k=1; i <= n; i++) {
      if(x[i] < tmp) {
         tmp = x[i];
         k = i;
      }
   }
```

```
   return k;
}
template<class T>
int maxi(int n, T* x) {
   T  tmp = x[1];
   for(int i=1, k=1; i <= n; i++) {
      if(x[i] > tmp) {
         tmp = x[i];
         k = i;
      }
   }
   return k;
}
```

由于下取整函数耗时 $O(1)$，故循环体内的运算耗时 $O(1)$。因此，整个算法耗时 $O(n)$。即算法 maxgap 是求最大间隙问题的线性时间算法。注意到在代数判定树计算模型下，$\Omega(n\log n)$是最大间隙问题的一个计算时间下界。这意味着在代数判定树的计算模型下，最大间隙问题是不可能有线性时间算法的。在此题中假设下取整函数耗时 $O(1)$，实际上这可以看作是在代数判定树模型中，将下取整运算作为基本运算增加到原有的基本运算集中，从而使代数判定树计算模型的计算能力得到增强。因而可以在线性时间内解最大间隙问题。