本章主要考察一些优化算法

Translate:USACO/fact4

Factorials 阶乘

描述

N 的阶乘写作 N!表示小于等于 N 的所有正整数的乘积。

阶乘会很快的变大，如 13!就必须用 32 位整数类型来存储，70！即使用浮点数也存不下了。

你的任务是找到阶乘最后面的非零位。举个例子:

```
5!=1*2*3*4*5=120 所以 5!的最后面的非零位是 2
7!=1*2*3*4*5*6*7=5040，所以最后面的非零位是 4
```

格式

PROGRAM NAME: fact4

INPUT FORMAT:

(file fact4.in)

共一行，一个整数不大于 4,220 的整数 N。

OUTPUT FORMAT:

(file fact4.out)

共一行，输出 N!最后面的非零位。

SAMPLE INPUT

```
7
```

SAMPLE OUTPUT

```
4
```

简单的把末尾的 0 去掉是不行的，因为我们不知道现在不是 0 的位会不会乘上下一个数之后就变成 0 了。

因为 10=2*5,所以每有一个 0 就有一对 2*5=10 出现，反之，如果这个数的质因数分解没有成对的 2，5，我们就可以简单的对 10 求模，而不用管前面的数字，因为它一定不会产生 0。

所以我们只要在处理阶乘的时候消掉所有成对的 2 和 5 就行了，容易理解，N!的质因数分解式里因子 2 远比 5 要多，所以只需要记录因子 2 的个数，有因子 5 就消掉，最后再把 2 乘回去就行了。

Translate:USACO/kimbits

Stringsobits01 串 Kim Schrijvers

描述

考虑排好序的 N(N<=31)位二进制数。

你会发现，这很有趣。因为他们是排列好的，而且包含所有长度为 N 且这个二进制数中 1 的位数的个数小于等于 L(L<=N)的数。

你的任务是输出第 i（1<=i<=长度为 N 的二进制数的个数）小的（注：题目这里表述不清，实际是，从最小的往大的数，数到第 i 个符合条件的，这个意思），长度为 N，且 1 的的位数的个数小于等于 L 的那个二进制数。

（例：100101 中，N=6，含有位数为 1 的个数为 3。）

格式

PROGRAM NAME: kimbits

INPUT FORMAT:

(file kimbits.in)

共一行，用空格分开的三个整数 N，L，i。

OUTPUT FORMAT:

(file kimbits.out)

共一行，输出满足条件的第 i 小的二进制数。.

SAMPLE INPUT

```
5 3 19
```

SAMPLE OUTPUT

```
10011
```

官方题解

假设我们知道如何计算含有给定个数的二进制 0 和 1 的集合大小。也就是说，假设我们有一个函数 sizeofset(n,m)返回含有至多 m 个 1 的 n 位二进制数的集合大小。

然后我们可以如下解决这个问题，我们在寻找至多 m 个 1 的 n 位二进制数的集合的第 i 个元素。这个集合有两部分，开始于 0 和 1 的。有 sizeofset(n-1, m)个数以 0 开始且至多有 m 个'1',有(n-1,m-1)个数以 1 开始并且至少有 m-1 个 1。

所以如果这个序号(i)比 sizeofset(n-1,m)小，题目所求的数出现在开始于 0 的那部分，不然的话，它以一个'1'开始。

由"printbits"实现的递归算法很合适

所剩的唯一困难是计算"sizeofset",我们可以用上面描述的特征使用动态规划解决这个问题。

sizeofset(n, m) = sizeofset(n-1, m) + sizeofset(n-1, m-1)

并且对于所有 m,sizeofset(0, m) = 1.我们使用 double 来存储大小，但这矫枉过正了，因为重写的问题只要求 31 位而不是 32 位。

动态规划

设长度为 j 的 01 串，1 的个数不大于 k 的个数为 f[j,k]!

方程：f[j,k]=f[j-1,k]+f[j-1,k-1]; //分别表示在当前位加上 0 和加上 1 时的两种状况

边界：f[j,0]=1,f[0,j]=1;f[j,k](k>j)=f[j,j]

这样我们得到了所有的 f[j,k](j,k∈Z, j,k>0),需要做的就是据此构造出所求字符串.

设所求串为 S,假设 S 的位中最高位的 1 在自右向左第 K+1 位，那么必然满足 F[K,L]< I,F[K+1,L] >=I,这样的 K 是唯一的。所以 S 的第一个 1 在从右至左第 K+1 位.因为有 F[K,L]个串第 K+1 位上为 0，所以所求的第 I 个数的后 K 位就应该是满足"位数为 K 且串中 1 不超过 L-1 个"这个条件的第 I-F[K,L]个数。

于是我们得到这样的算法:

for K:=n-1 downto 0 do {题目保证有解，所以 f[N,L]>I}

```
if F[K,L]<I then
begin
   s[N-K]:='1';
   dec(I,F[K,L]);{第 K+1 位是 1，所以 I 减去第 K+1 位是 0 的串的个数}
   dec(L);
end;
```

优化

其实从基本的组合原理出发，我们可以推出 F[j,k]的表达式:F[j,k]=C[j,0]+C[j,1]+C[j,2]+...+C[j,k].这里 C[i,j]表示从 i 个元素中取 j 的组合数因为 C[m,n]=C[m-1,n-1]+C[m-1,n],可得

F[j,k]=C[j-1,0]+C[j-1,0]+C[j-1,1]+...+C[j-1,K]+C[j-1,k-1]+C[j-1,K]

=2*(C[j-1,0]+C[j-1,1]+...+C[j-1,k-1])+C[j-1,K]

=2*F[j-1,k-1]+C[j-1,k]

$$F[j-1,k-1]=\frac{(F[j,k]-C[j-1,k])}{2}$$

这样我们就可以用一个递归程序求解该问题.

用,我们首先可以仅用 O(N)的时间就可以计算出 F(j,L)的值,并记录 C(j,L).若需添"1",则可由上式计算出 F(J-1,L-1)的值,并用 O(1)的时间将 C(J,L)改进为 C(J-1,L-1);若只需添"0",则可先计算出 C(J-1,L)的值,再将 F(J,L)改进为 F(J-1,L),也是 O(1)的时间. 因此我们只需要保留 C,F,N,M,K,I 等少量临时变量,空间复杂度为常数级,且可以在 O(N)的时间内计算出解,非常优化.

(还有 C[I,J]与 C[I-1,J],C[I,J-1]之间的 O(1)改进方法,参见程序 3,自己推也很容易)

Translate:USACO/spin

Spinning Wheels 纺车的轮子

1998 ACM NE Regionals

译 by Felicia Crazy

描述

一架纺车有五个纺轮，这五个不透明的轮子边缘上都有一些缺口。这些缺口必须被迅速而准确地排列好。每个轮子都有一个起始标记（在 0 度），这样所有的轮子都可以在统一的已知位置开始转动。轮子按照角度变大的方向旋转，所以从起始位置开始，在一定的时间内，它们依次转过 1 度，2 度等等（虽然这些轮子很可能不会同时转过这些角度）。

这是一个整数问题。轮子不会转过 1.5 度或 23.51234123 度这样的角度。例如，轮子可能在一秒钟内转过 20 到 25 度甚至 30 到 40 度（如果转得快的话）。

这个问题中的所有角度都限制在 0 <= 角度 <= 359 这个范围内。轮子转过 359 度后接下来就是 0 度。每个轮子都有一个确定的旋转速度，以秒作为单位。1 <= 速度 <= 180。

轮子上的缺口的起始角度和缺口大小（或长度）各由一个整数表示，都以度为单位。在一个轮子上，两个缺口之间至少有一度的间隔。

在起始位置，设时间为 0，所有的轮子的起始标记排列成一条直线。你的程序必须计算，最早出现每个的轮子上的缺口同其他轮子上的缺口对准（也就是一束光可以通过五个轮子上的五个缺口）情况的时间。这些缺口在任意一个角度对准。

格式

PROGRAM NAME: spin

INPUT FORMAT:

(file spin.in)

输入中的五行对应五个轮子。

第一个数字表示轮子的转动速度。下一个数字是缺口的数目 W。1 <= W <= 5。

接下来的 W 对数字表示每个缺口的起始角度和长度。

OUTPUT FORMAT: (file spin.out)

只有一行，包括一个整数，表示光能够通过这五个轮子的最早时间。如果无解，输出'none'（小写，不含引号）。

SAMPLE INPUT

```
30 1 0 120
50 1 150 90
60 1 60 90
70 1 180 180
90 1 180 60
```

SAMPLE OUTPUT

```
9
```

在 360 秒后，所有轮子都回到原处，所以只需模拟 0~359。

Translate:USACO/ratios

Feed Ratios 饲料调配

1998 ACM Finals, Dan Adkins

译 by Felicia Crazy

描述

农夫约翰从来只用调配得最好的饲料来喂他的奶牛。饲料用三种原料调配成：大麦，燕麦和小麦。他知道自己的饲料精确的配比，在市场上是买不到这样的饲料的。他只好购买其他三种混合饲料（同样都由三种麦子组成），然后将它们混合，来调配他的完美饲料。

给出三组整数，表示 大麦：燕麦：小麦 的比例，找出用这三种饲料调配 x：y：z 的饲料的方法。

例如，给出目标饲料 3：4：5 和三种饲料的比例：

```
1:2:3
3:7:1
2:1:2
```

你必须编程找出使这三种饲料用量最少的方案，要是不能用这三种饲料调配目标饲料，输出“NONE”。“用量最少”意味着三种饲料的用量（整数）的和必须最小。 对于上面的例子，你可以用 8 份饲料 1，1 份饲料 2，和 5 份饲料 3，来得到 7 份目标饲料：

```
8*(1:2:3) + 1*(3:7:1) + 5*(2:1:2) = (21:28:35) = 7*(3:4:5)
```

表示饲料比例的整数，都是小于 100 的非负整数。表示各种饲料的份数的整数，都小于 100。一种混合物的比例不会由其他混合物的比例直接相加得到。

格式

PROGRAM NAME: ratios

INPUT FORMAT:

(file ratios.in)

Line 1: 三个用空格分开的整数，表示目标饲料

Line 2..4: 每行包括三个用空格分开的整数，表示农夫约翰买进的饲料的比例

OUTPUT FORMAT:

(file ratios.out)

输出文件要包括一行，这一行要么有四个整数，要么是“NONE”。前三个整数表示三种饲料的份数，用这样的配比可以得到目标饲料。第四个整数表示混合三种饲料后得到的目标饲料的份数。

SAMPLE INPUT

```
3 4 5
1 2 3
3 7 1
2 1 2
```

SAMPLE OUTPUT

```
8 1 5 7
```

其实就是求解一个线性方程组，这里用向量的语言叙述了。

所求的实际上就是一个最简配比 x,y,z,k,使得 x(a1,b1,c1)+y(a2,b2,c2)+z(a3,b3,c3)=k(g1,g2,g3) 我们定义

$$\vec{p}=(a1,b1,c1),\vec{q}=(a2,b2,c2),\vec{r}=(a3,b3,c3),\vec{k}=(g1,g2,g3),$$

有 $x*\vec{p}+y*\vec{q}+z*\vec{r}=k*\vec{k}$ 我们用行列式求解这个问题，假设所得系数行列式分别为 D1,D2,D3,D,则 D=0(要么无解，要么多解，题目说没有任两个向量的线性组合是第三个，所以不成立)或是解不同号的情况则无解,要不然(D1,D2,D3,D)就是一组(x,y,z,k)，只需要用 gcd 约减下公约数就可以了。O(1);

Translate:USACO/msquare

Magic Squares 魔板

IOI'96

译 by Felicia Crazy

描述

在成功地发明了魔方之后，鲁比克先生发明了它的二维版本，称作魔板。这是一张有 8 个大小相同的格子的魔板：

```
1  2  3  4
8  7  6  5
```

我们知道魔板的每一个方格都有一种颜色。这 8 种颜色用前 8 个正整数来表示。可以用颜色的序列来表示一种魔板状态，规定从魔板的左上角开始，沿顺时针方向依次取出整数，构成一个颜色序列。对于上图的魔板状态，我们用序列（1，2，3，4，5，6，7，8）来表示。这是基本状态。

这里提供三种基本操作，分别用大写字母“A”，“B”，“C”来表示（可以通过这些操作改变魔板的状态）：

```
“A”：交换上下两行；
“B”：将最右边的一列插入最左边；
“C”：魔板中央四格作顺时针旋转。
```

下面是对基本状态进行操作的示范：

```
A:  8  7  6  5
    1  2  3  4
B:  4  1  2  3
    5  8  7  6
C:  1  7  2  4
```

```
8 6 3 5
```

对于每种可能的状态，这三种基本操作都可以使用。

你要编程计算用最少的基本操作完成基本状态到目标状态的转换，输出基本操作序列。

格式

PROGRAM NAME: msquare

INPUT FORMAT:

(file msquare.in)

只有一行，包括 8 个整数，用空格分开（这些整数在范围 1——8 之间），表示目标状态。

OUTPUT FORMAT:

(file msquare.out)

Line 1: 包括一个整数，表示最短操作序列的长度。

Line 2: 在字典序中最早出现的操作序列，用字符串表示，除最后一行外，每行输出 60 个字符。

SAMPLE INPUT

```
2 6 8 4 5 7 3 1
```

SAMPLE OUTPUT

```
7
BCABCCB
```

分析:

Just BFS+Hash

Translate:USACO/butter

Sweet Butter 香甜的黄油

描述

农夫 John 发现做出全威斯康辛州最甜的黄油的方法：糖。把糖放在一片牧场上，他知道 N（1<=N<=500）只奶牛会过来舔它，这样就能做出能卖好价钱的超甜黄油。当然，他将付出额外的费用在奶牛上。

农夫 John 很狡猾。像以前的巴甫洛夫，他知道他可以训练这些奶牛，让它们在听到铃声时去一个特定的牧场。他打算将糖放在那里然后下午发出铃声，以至他可以在晚上挤奶。

农夫 John 知道每只奶牛都在各自喜欢的牧场（一个牧场不一定只有一头牛）。给出各头牛在的牧场和牧场间的路线，找出使所有牛到达的路程和最短的牧场（他将把糖放在那）。

格式

PROGRAM NAME: butter

INPUT FORMAT:

(file butter.in)

第一行：三个数：奶牛数 N，牧场数 P（2<=P<=800），牧场间道路数 C(1<=C<=1450).

第二行到第 N+1 行: 1 到 N 头奶牛所在的牧场号.

第 N+2 行到第 N+C+1 行： 每行有三个数：相连的牧场 A、B，两牧场间距（1<=D<=255），当然,连接是双向的.

OUTPUT FORMAT:

(file butter.out)

一行 输出奶牛必须行走的最小的距离和.

SAMPLE INPUT

```
3 4 5
2
3
4
1 2 1
1 3 5
2 3 7
2 4 3
3 4 5
```

样例图形

```
         P2
P1 @--1--@ C1
    \    |\
     \   | \
      5  7  3
       \ |   \
        \|    \ C3
      C2 @--5--@
         P3    P4
```

SAMPLE OUTPUT

```
8
```

{说明：放在 4 号牧场最优.}

分析

此题的实际模型如下：给定一个有 P 个顶点的无向图，选择一个顶点，使得从该点到给定的 N 个顶点的权值之和最小。

Hint

一定要记得用邻接表存图

Bellman-Ford 算法

数据规模中边数小于 1450，使用 Bellman-Ford 求一次单源最短路的复杂度为 O(e*k)，k 为一个小常数，总复杂度也就是 O(n*e*k)。

d 数组储存最短路长，Bellman 算法的思想就是对每条边进行松弛，因为一共 n 个点，所以这种松弛操作最多执行 n 次后，d 数组的值就不会再改变。

还需要用一个布尔变量记录这次操作中是否产生了更小的最短路，如果没有，说明 d 数组中值已为最优，不需要再操作，直接退出。这样做或许时间上还是有点问题，便可以用 SPFA 优化。

SPFA 就是指每次松弛操作时记录哪些点更新了最短路值，将这些点加入数组，下次只对这些点关联的边进行松弛。进一步降低了复杂度。实现 SPFA 需要用邻接表储存边。

最后统计一次带权路径长，进行 n 次 Bellman-Ford 操作后得出结果。运行时间都在 0.4 秒以内。 --By Yangjhjh 07.08.11

Dijkstra 算法

求每对顶点之间的最短路径首先想到的便是 Floyd 算法:但 P 的范围是（P<=800）则时间复杂度为 O(800^3)，显然会超时。此时可以考虑使用 Dijkstra 算法求最短路径（Dijkstra 是 O(N^2)的算法），但直接使用 Dijkstra 仍然会超时 O(800*800^2)=O(800*^3)，此时可以用堆进行优化。

Dijkstra 算法每次都需要在 Dist[]数组中寻求一个最小值，如果图很大，则在此处花费的时间会很多，而堆则可以在 O(1)的时间内马上求得最小值，并且维护一个堆所需的时间都是在 log 级的。因此考虑把 Dist[]数组的值用堆来存储。

为了记录堆中的每一个元素是源点与哪个顶点的，可以增加了一个 point 变量，用来记录这条边所连接的另一个顶点。

用堆操作的 Dijkstra 的思想还是与一般的思想类似(在 OIBH 上找了一些信息)：

初始化 Dist[]=MAX，然后将源点 t 放入堆中(这就与 Dist[t]=0 类似),再从堆中找出一个最小值没有被确定的点 minp(就是 Dist[minp]=MAX)，将其确定下来(Dist[minp]=mins,mins 为从堆中取出来的那个最小值)，接着由这个最小值所连接的点求出从源点到这些点的路径长，若所连接的点没有求出最小值(Dist[i]=MAX)，则将这个点放入堆中(这就好比用 for 循环去修改每一个 Dist[]的值,不同的地方在于堆中存放的是源点到各个顶点的最小值，如果刚求出来的顶点在 Dist[]中已经被赋值了，说明求出来的肯定不是最小值，就像普通 Dijkstra 的 Mark[]一样，mark 过的点是不能再被计算的，所以不能放 Dist[]中有值的点。)这样 Dijkstra 的部分就完成了。

仔细观察了一下 N<=800，而道路数 C<=1450。800*800=640000 与 1450 的差距实在是大……可以改用存边的方式(同时保留邻接矩阵，这样可以很方便地知道两点之间的权值)，用一个二维的数组便可以存下边的信息。

补充：用 STL 中的 priority_queue 优化 dijkstra 编程复杂度更小！

SPFA 算法

对于枚举的每个牧场 i，用 SPFA 求出到每个点的最短路如下：dist[j]表示 i->j 的距离，初始值为 maxint，其中 dist[i]=0。维护一个队列，初始为 q[1]=i;由队首枚举其他的牧场 j 更新牧场 i 到 j 的最短距离并同时拓展队列。直到队列空为止。这样就求出了点 i 到所有点的最短距离。

SPFA 的优化：

如果枚举到的牧场 i，那么以表明 1--i-1 牧场为源点的最短路都已经求出过，也就是说 i 牧场到 1--i-1 中任一牧场最短路已知，用这个已知条件来初始化 dist[1]--dist[i-1]这样效率会有很大提升。**注意**：如果 min[j,k]表示 j 到 k 的最短路，在以 k 点为源点并进行上述优化时，应将 dist[k]:=min[j,k]+1，而不是 min[j,k]。否则 j 点不会进入队列，也就无法拓展经过 j 点的路径。

图的存储：

有人说用邻接表，但是我认为用链式前向星（参见 Malash 神牛 blog）（详见 http://malash.blog.163.com/blog/static/11934159720099180493847/ ）更方便而且省空间。可以在不增加时间复杂度的前提下把空间优化到 1000k 左右。