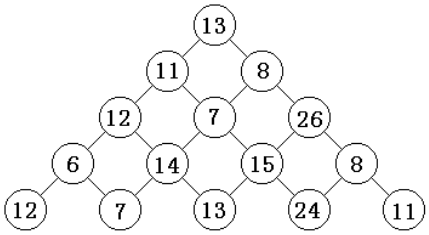
## 数字金字塔(tower.cpp)

总时间限制: 1s　　内存限制: 64MB

**【问题描述】**

观察下面的数字金字塔。写一个程序查找从最高点到底部任意处结束的路径，使路径经过数字的和最大。每一步可以从当前点走到左下方的点也可以到达右下方的点。




在上面的样例中,从 13 到 8 到 26 到 15 到 24 的路径产生了最大的和 86。

**【输入格式】**

第一个行包含 R(1<= R<=1000)，表示行的数目。

后面每行为这个数字金字塔特定行包含的整数。

所有的被供应的整数是非负的且不大于 100。

**【输出格式】**

单独的一行，包含那个可能得到的最大的和。

**【输入样例】**（tower.in）

```
5                          //数塔层数
13
11    8
12    7     26
6    14     15     8
12    7     13    24     11
```

**【输出样例】**（tower.out）

```
86
```

---

## 求最长不下降序列 (lic.cpp）

总时间限制: 1s　　内存限制: 64MB

**【问题描述】**

设有由 n 个不相同的整数组成的数列，记为:b(1)、b(2)、……、b(n)且 b(i)<>b(j)　(i<>j)，若存在 i1<i2<i3< … <ie 且有 b(i1)<b(i2)< … <b(ie)则称为长度为 e 的不下降序列。程序要求，求出最长的不下降序列。

例如 13，7，9，16，38，24，37，18，44，19，21，22，63，15。例中 13，16，18，19，21，22，63 就是一个长度为 7 的不下降序列，同时也有 7 ，9，16，18，19，21，22，63 长度为 8 的不下降序列。

**【输入格式】**

第一行：整数 n，表示整数序列的个数。

第二行：n 个用空格分隔的整数。

**【输出格式】**

第一行：max=t，t 表示最长不下降序列整数序列的个数。

**【输入样例】**（lic.in）

```
14
13   7   9   16   38   24   37   18   44   19   21   22   63   15
```

**【输出样例】**（lic.out）

```
max=8
```

## 拦截导弹(missile.cpp)

总时间限制：1s　　内存限制：64MB

**【问题描述】**

某国为了防御敌国的导弹袭击，发展出一种导弹拦截系统。但是这种导弹拦截系统有一个缺陷：虽然它的第一发炮弹能够到达任意的高度，但是以后每一发炮弹都不能高于前一发的高度。某天，雷达捕捉到敌国的导弹来袭。由于该系统还在试用阶段，所以只有一套系统，因此有可能不能拦截所有的导弹。

输入导弹依次飞来的高度（雷达给出的高度数据是不大于 30000 的正整数，导弹数不超过 1000），计算这套系统最多能拦截多少导弹，如果要拦截所有导弹最少要配备多少套这种导弹拦截系统。

**【输入格式】**

若干整数，用空格隔开。

**【输出格式】**

第一行，最多能拦截的导弹数。

第一行，要拦截所有导弹最少要配备的系统数。

**【输入样例】**（missile.in）

389　207　155　300　299　170　158　65

**【输出样例】**（missile.out）

6

2

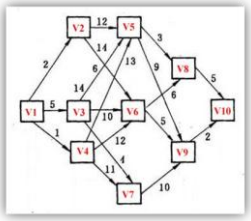
---

## 最短路径（short.cpp）

总时间限制：1s　　内存限制：64MB

**【问题描述】**

现在你有一张 n 个城市的交通网络图，试用动态规划的最优化原理求出 1 号城市到 n 号城市的最省费用。例如下图表示 10 个城市之间的交通路网，线段上的数字表示费用，单向通行由 V1->V10。V1->V10 的最省费用为 19。




**【输入格式】**

第一行：整数 n（n<=20），表示城市数目。接着是 n 行 n 列：表示每个城市之间的交通费用，为 0 表示没有道路。

**【输出格式】**

第一行：minlong=t，t 表示 1 号城市到 n 号城市的最省费用。第二行：最省费用所经过的城市编号。

**【输入样例】**（short.in）　　测试数据保证只有唯一的一条最省路径。

10

0　2　5　1　0　0　0　0　0　0

0　0　0　0　12　14　0　0　0　0

0　0　0　0　6　10　4　0　0　0

0　0　0　0　13　12　11　0　0　0

0　0　0　0　0　0　0　3　9　0

0　0　0　0　0　0　0　6　5　0

0　0　0　0　0　0　0　0　10　0

0　0　0　0　0　0　0　0　0　5

0　0　0　0　0　0　0　0　0　2

0　0　0　0　0　0　0　0　0　0

**【输出样例】**（short.out）

minlong=19

3　5　8　10

# 挖地雷(mine.cpp)

总时间限制: 1s　　内存限制: 64MB

**【问题描述】**

在一个地图上有 N 个地窖（N<=200）,每个地窖中埋有一定数量的地雷。同时，给出地窖之间的连接路径，并规定路径都是单向的,也不存在可以从一个地窖出发经过若干地窖后又回到原来地窖的路径。某人可以从任一处开始挖地雷，然后沿着指出的连接往下挖（仅能选择一条路径），当无连接时挖地雷工作结束。设计一个挖地雷的方案，使他能挖到最多的地雷。

**【输入格式】**

```
N                          //地窖的个数
W1，W2，……WN               //每个地窖中的地雷数
X1，Y1                     //表示从 X1 可到 Y1， X1<Y1
X2，Y2
……
0，0                       //表示输入结束
```

**【输出格式】**

```
K1-K2-…-Kv                 //挖地雷的顺序
MAX                        //最多挖出的地雷数
```

**【输入样例】**（mine.in）

```
6
5  10  20  5  4  5
1  2
1  4
2  4
3  4
4  5
4  6
5  6
0  0
```

**【输出样例】**（mine.out）

```
3-4-5-6
34
```

## 友好城市（ship.cpp）

总时间限制：1s　　内存限制：64MB

**【问题描述】**

Palmia 国有一条横贯东西的大河，河有笔直的南北两岸，岸上各有位置各不相同的 N 个城市。北岸的每个城市有且仅有一个友好城市在南岸，而且不同城市的友好城市不相同。

每对友好城市都向政府申请在河上开辟一条直线航道连接两个城市，但是由于河上雾太大，政府决定避免任意两条航道交叉，以避免事故。编程帮助政府做出一些批准和拒绝申请的决定，使得在保证任意两条航线不相交的情况下，被批准的申请尽量多。

**【输入格式】**

第 1 行，一个整数 N(1<=N<=5000)，表示城市数。

第 2 行到第 n+1 行，每行两个整数，中间用 1 个空格隔开，分别表示南岸和北岸的一对友好城市的坐标。(0<=xi<=10000)。

**【输出格式】**

仅一行，输出一个整数，表示政府所能批准的最多申请数。

**【输入样例】**（ship.in）

```
7
22  4
2   6
10  3
15  12
9   8
17  17
4   2
```

**【输出样例】**（ship.out）

```
4
```

---

## 合唱队形（chorus.cpp）

总时间限制：1s　　内存限制：64MB

**【问题描述】**

N 位同学站成一排，音乐老师要请其中的(N-K)位同学出列，使得剩下的 K 位同学排成合唱队形。

合唱队形是指这样的一种队形：设 K 位同学从左到右依次编号为 1, 2, …, K，他们的身高分别为 T1, T2, …, TK，则他们的身高满足 T1 < T2 < … < Ti , Ti > Ti+1 > … > TK (1≤i≤K)。

你的任务是，已知所有 N 位同学的身高，计算最少需要几位同学出列，可以使得剩下的同学排成合唱队形。

**【输入格式】**

输入文件 chorus.in 的第一行是一个整数 N（2 ≤ N ≤ 100），表示同学的总数。第二行有 N 个整数，用空格分隔，第 i 个整数 Ti（130 ≤ Ti ≤ 230）是第 i 位同学的身高（厘米）。

**【输出格式】**

输出文件 chorus.out 包括一行，这一行只包含一个整数，就是最少需要几位同学出列。

**【输入样例】**（chorus.in）

```
8
186  186  150  200  160  130  197  220
```

**【输出样例】**（chorus.out）

```
4
```

**【数据规模】**

对于 50%的数据，保证有 n ≤ 20；对于全部的数据，保证有 n≤100。

# 最长公共子序列（lcs.cpp）

总时间限制：1s　　内存限制：64MB

**【问题描述】**

一个给定序列的子序列是在该序列中删去若干元素后得到的序列。确切地说，若给定序列 $X=<x_1,x_2,\cdots,x_m>$，则另一序列 $Z=<z_1，z_2，\cdots，z_k>$是 X 的子序列是指存在一个严格递增的下标序列$<i_1,i_2,\cdots,i_k>$,使得对于所有 j=1,2,…,k 有：

$Xi_j=Z_j$

例如，序列 Z=<B,C,D,B>是序列 X=<A,B,C,B,D,A,B>的子序列,相应的递增下标序列为<2,3,5,7>。给定两个序列 X 和 Y，当另一序列 Z 既是 X 的子序列又是 Y 的子序列时，称 Z 是序列 X 和 Y 的公共子序列。例如，若 X=<A,B,C,B,D,A,B>和 Y=<B,D,C,A,B,A>，则序列<B,C,A>是 X 和 Y 的一个公共子序列,序列 <B,C,B,A>也是 X 和 Y 的一个公共子序列。而且，后者是 X 和 Y 的一个最长公共子序列．因为 X 和 Y 没有长度大于 4 的公共子序列。

给定两个序列 $X=<x_1，x_2，\cdots，x_m>$和 $Y=<y_1,y_2\cdots．y_n>$．要求找出 X 和 Y 的一个最长公共子序列。

**【输入格式】**

输入文件共有两行。每行为一个由大写字母构成的长度不超过 200 的字符串，表示序列 X 和 Y。

**【输出格式】**

输出文件第一行为一个非负整数。表示所求得的最长公共子序列的长度。若不存在公共子序列．则输出文件仅有一行输出一个整数 0。否则在输出文件的第二行输出所求得的最长公共子序列(也用一个大写字母组成的字符串表示)。若符合条件的最长公共子序列不止一个，只需输出其中任意一个。

**【输入样例】**（lcs.in）

ABCBDAB

BDCABA

**【输出样例】**（lcs.out）

4

**【提示】**:

最长公共子串（Longest Common Substring）和最长公共子序列（Longest Common Subsequence，LCS）的区别为：子串是串的一个连续的部分，子序列则是从不改变序列的顺序，而从序列中去掉任意的元素而获得新的序列；也就是说，子串中字符的位置必须是连续的，子序列则可以不必连续。字符串长度小于等于 1000。

# 机器分配（assigned.cpp）

总时间限制：1s　　内存限制：64MB

**【问题描述】**

总公司拥有高效设备 M 台，准备分给下属的 N 个分公司。各分公司若获得这些设备，可以为国家提供一定的盈利。问：如何分配这 M 台设备才能使国家得到的盈利最大？求出最大盈利值。其中 M≤15，N≤10。分配原则：每个公司有权获得任意数目的设备，但总台数不超过设备数 M。

输入数据文件格式为：第一行有两个数，第一个数是分公司数 N，第二个数是设备台数 M。

接下来是一个 N*M 的矩阵，表明了第 I 个公司分配 J 台机器的盈利。

**【输入格式】**

第一行有两个数，第一个数是分公司数 N，第二个数是设备台数 M。

接下来是一个 N*M 的矩阵，表明了第 I 个公司分配 J 台机器的盈利。

**【输出格式】**

第一行输出最大盈利值。

接下来 N 行，每行 2 个数，即分公司编号和该分公司获得设备台数。

**【输入样例】**（assigned.in）

```
3 3                                  //3 个分公司分 3 台机器
30  40  50
20  30  50
20  25  30
```

**【输出样例】**（assigned.out）

```
70                                   //最大盈利值为 70
1 1                                  //第一分公司分 1 台
2 1                                  //第二分公司分 1 台
3 1                                  //第三分公司分 1 台
```

**【数据规模】**

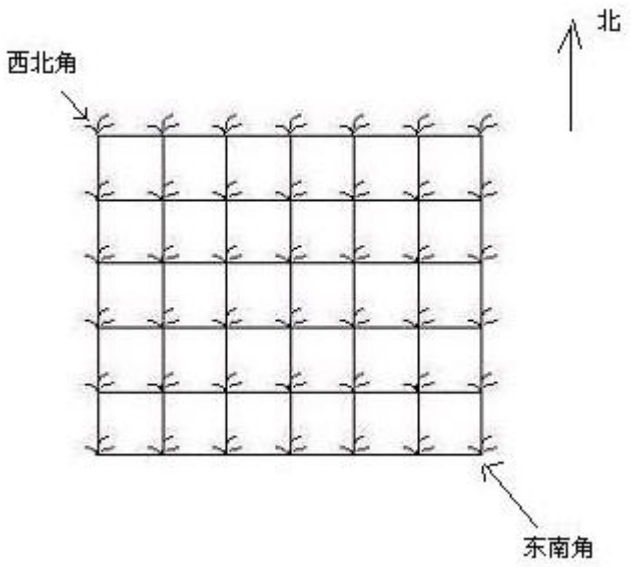
对于 50%的数据，保证有 n ≤ 20；对于全部的数据，保证有 n≤100。

# 摘花生（peanut.cpp）

总时间限制：1s　　内存限制：64MB

**【问题描述】**

Hello Kitty 想摘点花生送给她喜欢的米老鼠。她来到一片有网格状道路的矩形花生地(如下图)，从西北角进去，东南角出来。地里每个道路的交叉点上都有种着一株花生苗，上面有若干颗花生，经过一株花生苗就能摘走该它上面所有的花生。Hello Kitty 只能向东或向南走，不能向西或向北走。问 Hello Kitty 最多能够摘到多少颗花生。




**【输入格式】**

第一行是一个整数 T，代表一共有多少组数据。1<=T <= 100

接下来是 T 组数据。

每组数据的第一行是两个整数，分别代表花生苗的行数 R 和列数 C ( 1<= R,C <=100)

每组数据的接下来 R 行数据，从北向南依次描述每行花生苗的情况。每行数据有 C 个整数，按从西向东的顺序描述了该行每株花生苗上的花生数目 M ( 0<= M <= 1000)。

**【输出格式】**

对每组输入数据，输出一行，内容为 Hello Kitty 能摘到得最多的花生颗数。

**【输入样例】**（peanut.in）

```
2
2 2
1 1
3 4
2 3
2 3 4
1 6 5
```

**【输出样例】**（peanut.out）

```
8
16
```

## 怪盗基德的滑翔翼（kidd.cpp）

总时间限制：1s　　内存限制：64MB

**【问题描述】**

怪盗基德是一个充满传奇色彩的怪盗，专门以珠宝为目标的超级盗窃犯。而他最为突出的地方，就是他每次都能逃脱中村警部的重重围堵，而这也很大程度上是多亏了他随身携带的便于操作的滑翔翼。

有一天，怪盗基德像往常一样偷走了一颗珍贵的钻石，不料却被柯南小朋友识破了伪装，而他的滑翔翼的动力装置也被柯南踢出的足球破坏了。不得已，怪盗基德只能操作受损的滑翔翼逃脱。

假设城市中一共有 N 幢建筑排成一条线，每幢建筑的高度各不相同。初始时，怪盗基德可以在任何一幢建筑的顶端。他可以选择一个方向逃跑，但是不能中途改变方向（因为中森警部会在后面追击）。因为滑翔翼动力装置受损，他只能往下滑行（即：只能从较高的建筑滑翔到较低的建筑）。他希望尽可能多地经过不同建筑的顶部，这样可以减缓下降时的冲击力，减少受伤的可能性。请问，他最多可以经过多少幢不同建筑的顶部（包含初始时的建筑）？

**【输入格式】**

输入数据第一行是一个整数 K（K < 100），代表有 K 组测试数据。

每组测试数据包含两行：第一行是一个整数 N(N < 100)，代表有 N 幢建筑。第二行包含 N 个不同的整数，每一个对应一幢建筑的高度 h（0 < h < 10000），按照建筑的排列顺序给出。

**【输出格式】**

对于每一组测试数据，输出一行，包含一个整数，代表怪盗基德最多可以经过的建筑数量。

**【输入样例】**（kidd.in）

```
3
8
300  207  155  299  298  170  158  65
8
65  158  170  298  299  155  207  300
10
2  1  3  4  5  6  7  8  9  10
```

**【输出样例】**（kidd.out）

```
6
6
9
```

---

## 登山（mountain.cpp）

总时间限制：1s　　内存限制：64MB

**【问题描述】**

五一到了，PKU-ACM 队组织大家去登山观光，队员们发现山上一个有 N 个景点，并且决定按照顺序来浏览这些景点，即每次所浏览景点的编号都要大于前一个浏览景点的编号。同时队员们还有另一个登山习惯，就是不连续浏览海拔相同的两个景点，并且一旦开始下山，就不再向上走了。队员们希望在满足上面条件的同时，尽可能多的浏览景点，你能帮他们找出最多可能浏览的景点数么？

**【输入格式】**

第一行：N (2 <= N <= 1000) 景点数。

第二行：N 个整数，每个景点的海拔。

**【输出格式】**

最多能浏览的景点数。

**【输入样例】**（mountain.in）

```
8
186  186  150  200  160  130  197  220
```

**【输出样例】**（mountain.out）

```
4
```

# 最低通行费（tolls.cpp）

总时间限制：1s　　内存限制：64MB

**【问题描述】**

一个商人穿过一个 N*N 的正方形的网格，去参加一个非常重要的商务活动。他要从网格的左上角进，右下角出。每穿越中间 1 个小方格，都要花费 1 个单位时间。商人必须在(2N-1)个单位时间穿越出去。而在经过中间的每个小方格时，都需要缴纳一定的费用。

这个商人期望在规定时间内用最少费用穿越出去。请问至少需要多少费用？

注意：不能对角穿越各个小方格（即，只能向上下左右四个方向移动且不能离开网格）。

**【输入格式】**

第一行是一个整数，表示正方形的宽度 N (1 <= N < 100)；

后面 N 行，每行 N 个不大于 100 的整数，为网格上每个小方格的费用。

**【输出格式】**

输出至少需要的费用。

**【输入样例】**（tolls.in）

```
5
1   4   6   8  10
2   5   7  15  17
6   8   9  18  20
10 11  12  19  21
20 23  25  29  33
```

**【输出样例】**（tolls.out）

```
109
```

**【提示】**

样例中，最小值为 109=1+2+5+7+9+12+19+21+33。