

活动须知

1. 压缩包中包含除题面样例外，不便于排版的额外样例数据。
2. 请**严格按照题目要求用小写文件名** p1.cpp、p2.cpp、.....、p6.cpp 命名你的程序，并按照机房老师要求建立目录、提交源代码。提交代码大小限制为 100KB。
3. 程序直接使用 cin 或 scanf 读入数据，使用 cout 或 printf 输出，**不需要打开输入或输出文件**。
4. 所有测试点运行时间限制为 1s (以实际评测机为准)，内存限制为 1GB。

第一题 (p1.cpp, 10 分): 幸运数字

如果把一个数字十进制表示的**奇数位和偶数位分别相加得到的和相同**，小小就认为它是一个幸运的数字。例如：

- 12345 奇数位相加 $1 + 3 + 5 = 9$ 、偶数位相加 $2 + 4 = 6$ ，因此 12345 不是幸运数字；
- 2332 奇数位相加 $2 + 3 = 5$ 、偶数位相加 $3 + 2 = 5$ ，因此 2332 是幸运数字。

对于给定的 a 和 b ，小小希望你求出 $a, a + 1, a + 2, \dots, b$ 中幸运数字的数量。

输入格式

输入数据仅一行，包含空格分隔的两个整数 a 和 b 。

输出格式

输出一行一个整数，代表 $a, a + 1, a + 2, \dots, b$ 中幸运数字的数量。

样例输入 1

```
1 100
```

样例输出 1

```
9
```

样例输入 2

```
4096 65536
```

样例输出 2

```
3454
```

数据规模

- 对于 100% 的数据，满足 $1 \leq a \leq b \leq 1,000,000$ 。

第二题 (p2.cpp, 15 分): 精密计时

小小有一个非常精密的计时器，每秒可以计数 100 次 (两个连续的时钟计数之间恰好间隔百分之一秒)。例如，以下是从 13:01:02.37 到 13:01:03.01 的计时过程：

```
13:01:02.37 (13 时 1 分 2 秒.37)
13:01:02.38
13:01:02.39
...
13:01:02.98
13:01:02.99 (13 时 1 分 2 秒.99)
13:01:03.00 (13 时 1 分 3 秒.00)
13:01:03.01
```

小小记录了计时器上的两个时刻，你能帮助他计算这两个时刻之间经过了多少个“百分之一秒”吗？

输入格式

输入数据仅一行，包含空格分开的两个字符串，代表了两个计时器上显示的时刻。时刻中的时、分、秒、百分之一秒均使用两位十进制数字表示，不足两位时在十位添零。时、分、秒之间用冒号：分隔，秒和百分之一秒之间用小数点 . 分隔。

输出格式

输出一行一个整数，代表两个时刻之间经过的“百分之一秒”数。

样例输入 1

```
01:02:34.56 01:03:00.10
```

样例输出 1

```
2554
```

样例输入 2

```
00:00:00.00 23:59:59.99
```

样例输出 2

```
8639999
```

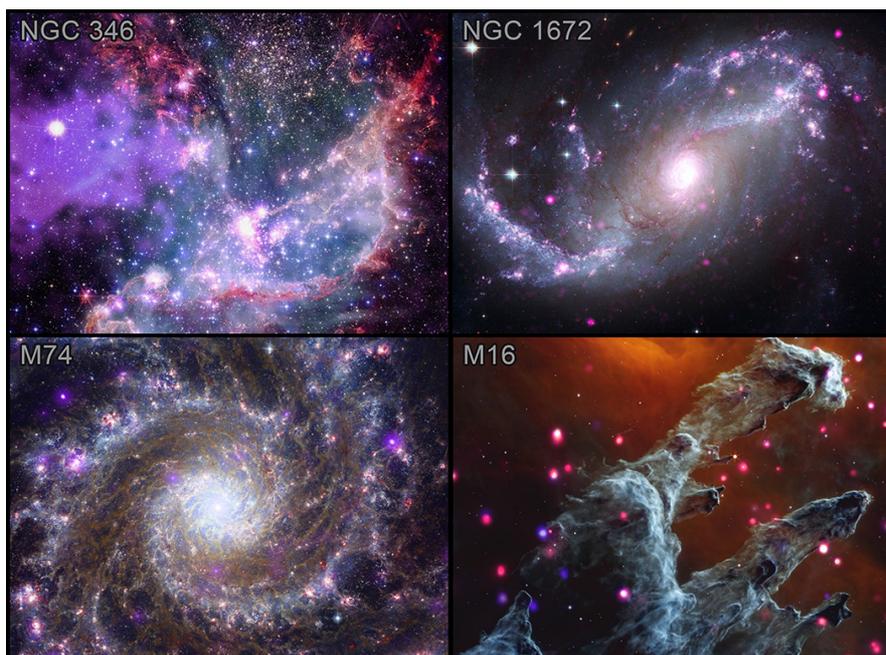
数据规模

- 对于 100% 的数据，两个时刻都来自 24 小时制的同一天 (00:00:00.00 到 23:59:59.99)，且保证后一个时刻晚于前一个时刻。

第三题 (p3.cpp, 15 分): 图像重建

JWST (詹姆斯·韦伯太空望远镜, James Webb Space Telescope) 是一台红外波段的大型太空望远镜，由美国国家航空航天局 (NASA)、欧洲航天局 (ESA) 和加拿大航天局 (CSA) 共同合作开发。它是哈勃太空望远镜的科学继任者，旨在解决一系列重要的天文学问题，包括宇宙的起源、星系的形成和演化、恒星和行星系统的形成，以及寻找宜居行星和生命迹象。

以下是 JWST 望远镜 5 月 23 日采集的深空图像：



太空望远镜在采集数据时，并不像我们日常生活中的手机或相机“一次拍摄成像”直接从传感器数据得到照片，而是由多次不同时段拍摄的图像拼接而成。在这个问题中，我们也来实现图像的拼接重建。

我们已经对一个区域拍摄了**两张**黑白图像。这两张图像面向同一区域拍摄，因此我们预期它们有相当一部分都是重叠的。你的任务就是将两张图像经过上下左右平移后尽可能“重叠”在一起，满足**重叠部分所有像素均完全相同，且重叠部分的面积尽可能大(即重叠部分的像素数量尽可能多)**。

输入格式

输入数据由两张图像的描述组成。两张图像的描述之间有一个空行。

对于每张图像，第一行包含两个整数 n 和 m ，代表了图像的尺寸(像素的行数和列数)。接下来 n 行，每行 m 个像素描述了拍摄的图像，其中 0 表示一个黑色像素，1 表示一个白色像素。每一行的像素由一个空格分隔。

输出格式

输出一行一个整数，即经过最优平移后，重叠部分的像素数量。

样例输入 1

```
3 3
0 0 0
0 1 1
0 1 1

2 4
1 1 0 0
1 1 0 1
```

样例输出 1

```
4
```

压缩包中包含更多的样例数据。

数据规模

- 对于 100% 的数据，满足 $1 \leq n, m \leq 50$ 。
-

第四题 (p4.cpp, 20 分): 程序分析

程序分析技术是一种用于理解和改进计算机程序的方法。它可以帮助我们找出程序中的错误、提高程序的性能、优化代码结构等。其中，静态分析技术在不运行程序的情况下对程序代码进行分析。它可以检查代码的语法、风格、潜在错误等。例如，静态分析可以帮助我们找出程序中未使用的变量、可能的数组越界等问题。

小小设计了一个自己的编程语言，并命名为 X 语言。你能为其设计一个静态分析器吗？

X 语言程序中只有两个整型变量 x 和 y ，且无需定义，可以直接使用。变量 x 的值从程序外输入（输入值可以是任何 C++ int 范围内的值）， y 的初始值是 0。一个 X 语言程序由若干行组成，每行**恰好包含一条命令**，是以下三种命令之一：

1. 条件分支：if (条件) {
2. 对 y 赋值：y = 数字；
3. 条件结束：}

其中，“条件”要么是“ $x >$ 数字”，要么是“ $x <$ 数字”。赋值语句和条件中的“数字”都是 1 到 1,000,000,000 之间的常数。if 和赋值的含义同 C++ 语言中的条件和赋值语句。

请你编写一个静态分析器，分析一个 X 语言程序执行结束时，所有可能的 y 的值。

输入格式

输入数据第一行为整数 n ，代表程序的行数。

接下来 n 行，每行一个命令，描述了一个合法的 X 语言程序：输入的程序保证括号配对，且符合问题描述中的约定。为了便于大家解析（例如用 cin 或 scanf 读入），输入程序中的 if 后、{ 前、=、<、> 左右都恰好有一个空格，行首可能有若干空格缩进。除此之外，输入不含多余的空格或空白字符。

输出格式

输出一行，从小到大不重复地输出程序结束时，变量 y 所有可能的值。数字之间由一个空格隔开。

样例输入 1

```
10
if (x > 1) {
  y = 2;
  if (x > 10) {
    y = 1;
    y = 4;
    if (x < 5) {
```

```
        y = 3;
    }
}
}
```

样例输出 1

```
0 2 4
```

压缩包中包含更多的样例数据。

数据规模

- 对于 100% 的数据，满足 $n \leq 1,000$ 。输入数据的每行都不超过 1,000 个字符。

第五题 (p5.cpp, 20 分): 文本压缩

数据压缩算法的目的是减少数据的大小，以便更快地传输和存储。我们会经常用到的 zip、rar 等压缩工具，就是利用数据压缩算法把多个文件或者文件夹压缩成一个更小的文件；我们的网页在传输时，通常也使用了 gzip 压缩。有些时候 (例如传输图像、视频时)，我们会允许在压缩过程中损失一些精度，以实现更好的压缩比。

在这个问题里，你需要自己设计一个**英文文本的无损压缩和解压缩算法**。你的程序需要同时实现压缩器和解压缩器两部分功能：

- 压缩器输入一个**仅由小写字母组成的字符串**，输出一个压缩后的字符串。压缩后的字符串允许使用大写字母、小写字母和数字，但不允许使用其他字符。
- 解压缩器输入一个压缩后的字符串，还原出小写字母的字符串。

注意，在这个问题中，**所有给压缩器的输入都来自人工智能 GPT-3.5-turbo 生成的英文文本保留字母 (并转换为小写) 后得到的**，也就是说，你可以假设除了偶尔的例外，字符串是由英文单词拼接而成的。这个性质是解决问题的关键——随机序列的压缩比“有规律”序列的压缩要困难得多。

输入格式

输入数据第一行为一个大写英文单词，代表压缩/解压缩的任务类型。“COMPRESS”代表压缩；“DECOMPRESS”代表解压缩。

第二行一个字符串，是压缩/解压缩任务对应的文本。对于压缩任务，是一个仅包含小写字母的字符串；对于解压缩任务，字符串总是来自你程序之前 COMPRESS 压缩的结果。

输出格式

输出一行一个字符串。对于压缩任务，输出一行为压缩后的文本；对于解压缩任务，输出一行为解压缩后的文本。

样例数据

见压缩包。样例仅给出了压缩任务所需的字符串 (由人工智能 GPT-3.5-turbo 生成)，不含样例输出；你可以在给出的样例上实验你的压缩算法。

数据规模

- 对于 100% 的数据，输入字符串的长度 n 满足 $30,000 \leq n \leq 40,000$ 。

评测说明与评分标准

本题的评测机会两次调用你的程序：

- 第一次调用你的程序执行压缩任务，例如输入

```
COMPRESS
aaaaaaaaabbbbbbbbbb
```

假设本次调用，你的程序输出了 `a8b10` 作为压缩后的文本。这个输出会被评测系统记住。

- 第二次，调用你的程序执行解压缩任务。针对刚才的例子，我们会为你的程序输入

```
DECOMPRESS
a8b10
```

如果你的程序解压缩输出 `aaaaaaaaabbbbbbbbbb` (即正确解压缩得到原字符串)，则被判定为“还原正确”。

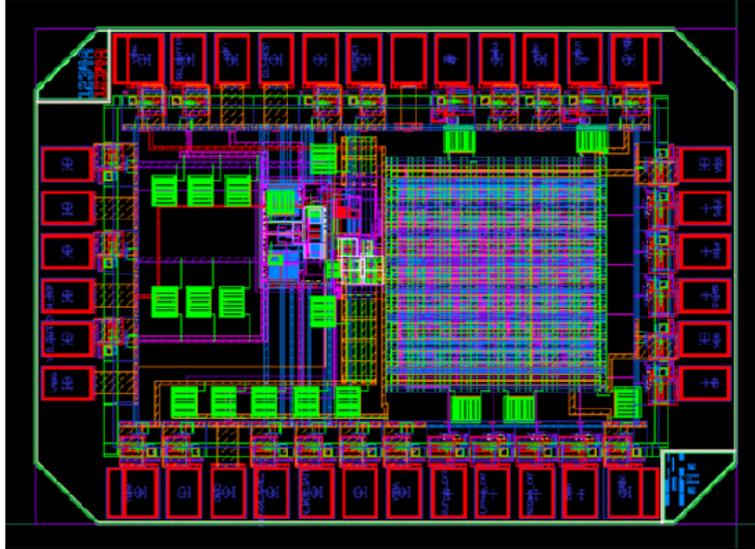
对于每个测试数据，压缩和解压缩分别有 1s 的运行时间限制。

对于每个测试数据，在解压缩能正确得到原字符串的前提下：

- 如果压缩率达到 75% (压缩字符串的长度小于等于输入长度的 $3/4$)，该测试点得满分。
 - 如果压缩率达到 80% (压缩字符串的长度小于等于输入长度的 $4/5$)，该测试点得一半分数。
-

第六题 (p6.cpp, 20 分): 电路布线

电路布局布线 (Circuit Layout and Routing) 是电子设计自动化 (EDA) 领域的一个重要概念，它涉及到在电路板或集成电路上安排和连接电子元件的过程。这个过程的目标是在满足电气性能、信号完整性、电磁兼容性等要求的同时，实现对空间、成本和生产工艺的优化。



小小现在需要解决一个简化的电路布线问题，在一个 $n \times m$ 的方格中进行电路布线。其中：

- 井号 # 标记的格子已经被占用，不能布线。
- 加号 + 标记的格子会连接到电路的其他部分，必须被布线。在给定的电路布线问题中，至少有一个格子必须被布线。
- 点号 . 标记的格子小小有权选择是否布线：布线即将该格标记为加号，不布线即保持为点号。

小小的任务是选择尽可能多的格子进行布线 (将“.”的格子标记为“+”)，满足：

1. 布线电路连通。即从任意一个已布线的格子，都能通过上、下、左、右移动到相邻已布线格子的方式，到达任意另一个布线的格子。
2. 布线不存在短路 (回路)，即不存在某个布线的格子能通过 > 2 步的上、下、左、右移动到相邻布线格子的方式回到自身，且经过的格子各不相同。

例如，以下是一个电路布线问题，已有三个格子被标记为必须布线 (加号)：

```
#....#  
....+#  
.+####  
.+...#
```

以下展示了一种合法和两种不合法的布线方案：

#+.+.#	#.+. .#	#++..#
++++#	..+++#	.++++#
.+####	.+####	.+####
.++++#	.+...#	.+...#
合法	不连通	有回路

输入格式

输入第一行是两个空格分隔的整数 n 和 m ，代表布线问题格子的行数和列数。

接下来 n 行，每行 m 个字符 (#, +, . 中的一个)，描述了具体的布线问题。

输入数据保证至少存在一种合法的布线方案。输入数据中至少有一个 +。

输出格式

输出 n 行，每行 m 个字符，代表最优的布线方案，其中被布线的格子尽可能多。如有多种可能的方案，输出任意一种即可。

样例输入 1

```
2 2
+.
..
```

样例输出 1

```
+.
++
```

样例输入 2

```
3 5
...+#
..###
.....+
```

样例输出 2

```
++++#
.+###
+++++
```

样例输入 3

```
5 6
..++..
.#..#.
.#..#.
.#..#.
.....
```

样例输出 3

```
++++++
+#.+#+
+#+.#+
+#++#+
+++.+++
```

数据规模

- 对于 40% 的数据，满足 $n \times m \leq 16$ 。
- 对于 100% 的数据，满足 $n, m \leq 6$ 。

评分标准

在你的布线方案合法 (连通且无回路) 的前提下：

- 如果你的方案是最优布线方案，即布线的格子最多，该测试点得满分。
- 否则，该测试点得一半分数。