CCF CSP 计算机软件能力认证

CCF CSP

第 35 次认证

时间：2024 年 9 月 22 日 13:30 ~ 17:30

题目名称	密码	字符串变换	补丁应用	通讯延迟	木板切割
题目类型	传统型	传统型	传统型	传统型	传统型
输入	标准输入	标准输入	标准输入	标准输入	标准输入
输出	标准输出	标准输出	标准输出	标准输出	标准输出
每个测试点时限	1.0 秒	1.0 秒	1.0 秒	1.5 秒	1.0 秒
内存限制	512 MiB	512 MiB	512 MiB	512 MiB	512 MiB
子任务数目	10	20	7	20	20
测试点是否等分	是	是	否	是	是

密码（key）

【题目背景】

西西艾弗网对用户密码有一套安全级别评定标准。

【题目描述】

在西西艾弗网上，用户的密码是一个由大写字母（A-Z）、小写字母（a-z）、数字（0-9）和特殊字符（* 和 #）共 64 种字符组成的字符串。

根据复杂程度不同，密码安全度被分为高、中、低三档。

- 高：由上述 64 种字符组成，长度大于等于 6 个字符，包含字母、数字和特殊字符，同一个字符出现不超过 2 次；
- 中：由上述 64 种字符组成，长度大于等于 6 个字符，包含字母、数字和特殊字符，且未达到高安全度要求；
- 低：由上述 64 种字符组成，长度大于等于 6 个字符，且未达到中安全度要求；

小 P 为自己准备了 n 个候选密码，试编写程序帮小 P 自动判别每个密码的安全级别。保证这 n 个密码都至少满足低安全度要求，当安全度为高、中、低时分别输出 2、1、0 即可。

【输入格式】

从标准输入读入数据。

输入共 $n+1$ 行。

第一行包含一个正整数 n，表示待判别的密码个数；

接下来 n 行，每行一个字符串，表示一个安全度至少为低的候选密码。

【输出格式】

输出到标准输出。

输出共 n 行，每行输出一个整数 2、1 或 0，表示对应密码的安全度。

【样例输入】

```
4
csp#ccsp
csp#ccsp2024
Csp#ccsp2024
CSP#2024
```

【样例输出】

```
0
1
2
2
```

【样例解释】

第一个密码不含数字，安全度为低；

第二个密码中小写字母 c 出现 3 次，安全度为中；

和第二个密码相比，第三个密码把一个小写字母 c 变为了大写，满足了高安全度要求；

第四个密码同样满足高安全度要求。

【子任务】

全部的测试数据满足 $n \leq 100$，且输入的每个字符串均不超过 20 个字符。

字符串变换（str）

【题目描述】

本题涉及字符包括大小写字母（`A-Z` 和 `a-z`）、数字（`0-9`）和空格共 63 种。在这个字符集合上，小 P 定义了一个字符替换函数 $f(ch)$，表示将字符 ch 替换为 $f(ch)$。例如 $f(a)=b$ 表示将 `a` 替换为 `b`，$f(b)=0$ 表示将 `b` 替换为 `0`。进而可以将其扩展为字符串变换函数 $F(s)$，表示对字符串 `s` 进行变换，将 `s` 中每个字符 ch 都替换为 $f(ch)$。

字符替换函数 f 可表示为 n 个字符对 (ch_1, ch_2)，即 $f(ch_1)=ch_2$。

- n 个字符对中，ch_1 两两不同，即不会出现同时定义了 $f(a)=b$ 和 $f(a)=0$ 的情况；
- 未定义的 $f(ch)$，可视为 $f(ch)=ch$，即字符 ch 保持不变；
- 函数 f 为单射，即当 $ch1 \neq ch2$ 时有 $f(ch1) \neq f(ch2)$，例如不会同时有 $f(b)=0$ 和 $f(0)=0$（`b` 和 `0` 都被替换为 `0`）。

现给定初始字符串 `s`，试处理 m 个查询：每个查询包含一个正整数 k，询问对初始字符串 `s` 变换 k 次后的结果 $F^k(s)$。

【输入格式】

从标准输入读入数据。

输入共 $n+4$ 行。

输入的第一行包含一个字符串，形如 `#s#`，即用两个井号字符 `#` 将初始字符串 `s` 囊括其中。

输入的第二行包含一个正整数 n，表示组成函数 f 的字符对数；接下来 n 行每行输入一个形如 `#xy#` 的字符串，表示 $f(x)=y$。

输入的第 $n+3$ 行包含一个正整数 m，表示查询的个数；下一行包含空格分隔的 m 个正整数 $k_1, k_2, \cdots, k_m$，表示 m 个查询。

【输出格式】

输出到标准输出。

输出共 m 行，依次输出 m 个查询的结果；输出时每行同样是一个形如 `#s#` 的字符串，即用两个井号把变换后的字符串 `s` 括起。

【样例输入】

```
#Hello World#
6
#HH#
```

```
#e #
# r#
#re#
#oa#
#ao#
3
1 2 3
```

【样例输出】

```
#H llarWaeld#
#HrlloeWo ld#
#Hella Warld#
```

【子任务】

前 60% 的测试数据保证初始字符串 s 仅包含小写字母，且输入的 n 个字符对也皆为小写字母（即保证小写字母仅会被替换为小写字母）；

前 80% 的测试数据保证查询数量 $m \leq 10$、变换次数 $k \leq 100$；

全部测试数据保证 $0 < n \leq 63$、$0 < m \leq 10^3$、$0 < k \leq 10^9$ 且初始字符串 s 包含不超过 100 个字符。

【提示】

由于读入的字符串中包含空格字符，推荐使用按行读取的方式，避免读入时跳过空格（如 cin 直接读入字符串）。

C 语言：可以使用 fgets() 函数读入整行，fgets(s, count, stdin) 会从标准连续输入读入至多 count - 1 个字符，并存入字符数组 s，直到遇到换行符或文件末尾为止。在前一种情况下，s 结尾处会存有读入的换行符。也可使用 getchar() 函数逐个字符读入，如 char ch = getchar();，这种方法需手动判断是否到达行末，即返回值是否为 \n。

C++：可以使用 std::getline() 函数读入整行：std::getline(std::cin, s) 会从标准输入读入字符，并存入字符串 std::string s 中，直到换行符或文件末尾为止。在前一种情况下，换行符会从标准输入中读出，但不会存入字符串 s 中。

Python：使用 input() 函数即可读入整行。

补丁应用（patch）

【题目背景】

西西艾弗岛运营公司的信息技术部门，需要协作开展程序开发和代码审查工作。他们的工作流程是这样的：首先，开发者将代码复制一份，并修改代码副本，从而得到期望的代码。然后，开发者使用 diff 工具比较修改前后的代码的区别，并将其输出用邮件发送给代码审查者。审查者收到邮件后，可以直接观察开发者作出的代码变更，并提出意见。反复进行后，当代码审查者对变更满意时，会使用 patch 程序，将开发者者提出的修改应用到原代码上，从而得到最终的代码。

现在，他们已经可以实现 diff 程序，但是需要你帮助他们实现这个 patch 程序。

diff 的输出称作补丁。补丁由一个或多个块组成。每个块包含若干行文本，表示对文件的一处修改。其中第一行以 @@ 开头和结尾，形如：

```
@@ -NN,MM +nn,mm @@
```

其中 NN、MM、nn、mm 表示一个 1 至 9 之间的字符和零个或多个 0 至 9 之间的字符组成的字符串，表示一个正整数。每块的第一行表示原文件和新文件的行号范围。其中，NN 表示这处修改在原文件的第 NN 行开始（原文件的行号从 1 开始编号）；MM 表示这处修改涉及原文件的 MM 行；nn 表示这处修改，在修改后从新文件的第 nn 行开始；mm 表示这处修改在修改后，在新文件中有 mm 行。

随后会有若干行文本，表示修改的内容。如果一行文本以 - 开头，表示这行文本在原文件中被删除；如果一行文本以 + 开头，表示这行文本在新文件中被添加；如果一行文本以空格开头，表示这行文本在原文件和新文件中都存在，未发生变化。因此，一个块中所有以 - 开头的行和以空格开头的行的总数，应该等于 MM；一个块中所有以 + 开头的行和以空格开头的行的总数，应该等于 mm。一处修改的描述中，可以适当包含若干不变的行，以便确定修改的上下文。

例如，下面是一个块的内容：

```
@@ -1,4 +1,5 @@
-a
+b
 1
+c
 2
 3
```

表示该处修改自原文件的第 1 行开始，共 4 行；修改后的文本从新文件的第 1 行开始，共 5 行。此处修改前，原文件的内容应该为：

```
a
```

```
1
2
3
```

修改后，新文件的内容应该为：

```
b
1
c
2
3
```

【题目描述】

但是，patch 程序在处理 diff 的输出时，对格式的要求可以较为宽松。例如，它可以允许块内有注释，也可以允许块的行号与实际原文件的行号不匹配。这是因为，在实际应用中，diff 生成后，源文件可能经历了其它的变更，导致行号出现了挪动。patch 程序的具体工作过程是：

1. 读取全部输入，将 # 开头的行视为注释，并移除；
2. 寻找 @ 开头的行，并将该行至下一个 @ 开头的行（或文本结尾）之间的内容视为一个块，如果没有找到 @ 开头的行，则认为补丁损坏；
3. 从前到后依次对每个块：
    1. 解析第一行，检查其格式是否正确，如果不正确，则认为补丁损坏；
    2. 解析出 NN、MM、nn、mm，其中忽略 nn；
    3. 如果这个块不是第一个块，检查 NN 是否不小于前一个块的 NN 与 MM 之和，如果不是，则认为补丁损坏；
    4. 解析其余行，如果这些行中存在不是以 -、+、空格开头的行，则认为补丁损坏；
    5. 将块中所有以 - 开头的行和以空格开头的行提取出来，作为原文件的内容片段；
    6. 检查原文件的内容片段的行数是否与 MM 一致，如果不一致，则认为补丁损坏；
    7. 将块中所有以 + 开头的行和以空格开头的行提取出来，作为新文件的内容片段；
    8. 检查新文件的内容片段的行数是否与 mm 一致，如果不一致，则认为补丁损坏；
4. 如果所有块都通过了检查，则对于每个块：

1. 检查是否存在绝对值小于 MM 的整数 δ，使得自原文件的第 NN $+ \delta$ 行开始的 MM 行，与块的原文件内容片段完全匹配。如果不是第一个块，还需满足 NN $+ \delta$ 不小于前一个块的 NN 与 MM 之和，即满足每块对应的原文区域没有重叠。如果不存在这样的 δ，则认为补丁损坏；
2. 如果存在多个这样的 δ，则取绝对值最小的那个，如果仍然存在多个，则取最小的那个；
3. 将原文件的第 NN $+ \delta$ 行开始的 MM 行替换为块的新文件内容片段；
4. 将该块和此后的所有块的 NN 加上 δ。

【输入格式】

从标准输入读入数据。
输入的第一行包含一个正整数 n，表示原文件的总行数。
接下来的 n 行文本，表示原文件的内容。
接下来的若干行，表示待应用的补丁，其中补丁可能是损坏的。

【输出格式】

输出到标准输出。
输出应用补丁后的文件内容；如果补丁损坏，输出 `Patch is damaged.`。

【样例 1 输入】

```
7
bbb
a
1
2
3
4
5
dummy
@@ -1,4 +1,5 @@
-a
+b
 1
+c
 2
 3
```

```
@@ -6,2 +6,2 @@
-4
+6
 5
```

【样例 1 输出】

```
bbb
b
1
c
2
3
6
5
```

【样例 1 解释】

输入原始文本共有 7 行，之后开始解析补丁。遇到的第一个以 @ 开头的行是 @@ -1,4 +1,5 @@，表示第一个块内容的开始，因此忽略此前的 dummy 行，即输入的第 8 行。第一个块即为题目描述中的块，该块的原始内容与输入的原始内容的第 2 行到第 5 行相同，因此对应将这些行替换，且将该块及其后的所有块的行号加上 1。所以此时输出 bbb（未包括于第一个块）、b、1、c、2、3。第二个块的头部标识该块起始于原始文件的第 6 行，加上 1 为第 7 行，该块的原始内容共两行，分别为 4、5。这一内容与原始文件的第 6 行、第 7 行相同，因此对应将这两行替换，且将该块及其后的所有块的行号减去 1。所以此时接下来输出 6、5。

【样例 2 输入】

```
7
bbb
a
1
2
3
4
5
```

```
dummy
```

【样例 2 输出】

```
Patch is damaged.
```

【样例 2 解释】

输入原始文本共有 7 行，之后开始解析补丁。补丁中不含有有效的块，因此为非法补丁。

【样例 3 输入】

```
8
bbb
a
1
2
3
4
5
6
dummy
@@ -1,4 +1,5 @@
-a
+b
 1
+c
 2
 3
@@ -6,2 +6,2 @@
-4
+6
 5
 6
```

【样例 3 输出】

```
Patch is damaged.
```

【样例 3 解释】

输入原始文本共有 8 行，之后开始解析补丁。补丁的第二个块中，其第一行表示其原始内容有 2 行，但随后却给出了 3 行的原始内容，因此为非法补丁。

【样例 4 输入】

```
8
bbb
a
1
2
3
4
5
6
dummy
@@ -1,4 +1,5 @@
-a
+b
 1
+c
 2
 3
@@ -6,2 +6,2 @@
-4
+6
 5
# 6
```

【样例 4 输出】

```
bbb
```

```
b
1
c
2
3
6
5
6
```

【样例 4 解释】

与样例 3 相比，多出来的一行前增加了 # 号被忽略，因此为合法补丁，其应用过程与样例 1 类似。

【子任务】

对于 30% 的数据，补丁仅包含一个块，且不含有注释。

对于 60% 的数据，有 $n \leq 60$，且输入中行的长度不超过 120，补丁是合法补丁，补丁块相对于原始内容无偏移。

对于 100% 的数据，有 $n \leq 2000$，且输入中行的长度不超过 830，且输入的总长度不超过 500KiB，输入数据中仅包含 ASCII 码在 32 至 126 之间的字符和换行符（ASCII 码为 10），且补丁块不超过 25 个。

【评分方式】

本题共包括十个测试点：

- 前六个测试点每通过一个可得 10 分；
- 后四个测试点同时通过可得 40 分，否则不得分。换言之，对所有测试点皆输出补丁损坏不会得分。

【提示】

本题目的输入数据中，每一行，包括最后一行在内，都以换行符（ASCII 码为 10，即 16 进制的 `0x0a`，亦为 escape 序列 `\n`）结束。

通讯延迟（delay）

【题目描述】

给定二维平面上 n 个节点，以及 m 个通讯基站。第 i 个基站可以覆盖以坐标 (x_i, y_i) 为中心、$2r_i$ 为边长的正方形区域，并使正方形区域内（包含边界）所有节点以 t_i 单位时间的延迟进行相互通讯。

求节点 1 到 n 的最短通讯延迟。

【输入格式】

从标准输入读入数据。

第一行包含空格分隔的两个正整数 n、m；

接下来 n 行，每行两个整数 x_i, y_i，代表第 i 个节点的坐标；

接下来 m 行，每行四个整数 x_j, y_j, r_j, t_j，代表第 j 个通讯基站的坐标，通讯半径与通讯延迟。

【输出格式】

输出到标准输出。

输出一行，即节点 1 到 n 的最短通讯延迟；如果无法通讯，则输出 `Nan`。

【样例输入】

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

【样例输出】

```
6
```

【样例解释】

1 号通讯基站延迟为 5，覆盖节点 1、2；
2 号通讯基站延迟为 6，覆盖节点 2、4、5；
3 号通讯基站延迟为 1，覆盖节点 1、3；
4 号通讯基站延迟为 3，覆盖节点 2、3、4；
5 号通讯基站延迟为 2，覆盖节点 4、5。
最短延迟方案为：

1. 节点 1 通过 3 号基站传讯至节点 3，延迟 1；
2. 节点 3 通过 4 号基站传讯至节点 4，延迟 3；
3. 节点 4 通过 5 号基站传讯至节点 5，延迟 2；

总计延迟为 6。

【子任务】

30% 的测试数据满足 $n, m \leq 100$；
对于额外 30% 的测试数据，每个通讯基站至多覆盖 20 个节点；
全部的测试数据满足 $n, m \leq 5000$ 且 $0 \leq x_i, y_i, r_i \leq 10^9$、$1 \leq t_i \leq 10^5$。

木板切割（cut）

【题目描述】

你有一块长度为 n 的木板和 m 种颜色，木板被平均分成 n 段，分别编号为 $1, 2, \cdots, n$。第 i 段被染为颜色 c_i。这块木板为 1 号木板。

你要进行 k 次切割操作，第 i（$1 \le i \le k$）次切割操作有三个参数 x_i, l_i, r_i：

- 表示将 x_i 号木板中编号在 $[l_i, r_i]$ 之间的所有段切割下来，作为第 $i+1$ 号木板；
- 原先木板切割剩下的部分重新连接成一块木板，木板编号仍为 x_i；
- 每一段的编号不受切割操作影响；
- 特别的，木板长度可以为 0，即切下的木板不包含任一段。

你想要知道，每次切割操作切下的木板：

1. 包含多少种不同的颜色？
2. 包含多少个颜色段？

一个颜色段定义为：一块木板上极长的连续若干段，满足这些段具有相同的颜色。若切下的木板长度为 0，则不同颜色数和颜色段数都视为 0。

【输入格式】

从标准输入读入数据。

输入共 $k+2$ 行。

第一行包含三个正整数 n, m, k。

第二行包含 n 个正整数 $c_1, c_2, \cdots, c_n$，表示木板上每一段的颜色。

接下来共有 k 行，每行三个整数 x_i, l_i, r_i，表示一次切割操作。

【输出格式】

输出到标准输出。

共 k 行，第 i 行两个正整数分别表示第 i 次切下的木板中不同颜色数和颜色段数。

【样例输入】

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

【样例输出】

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

【样例解释】

初始 1 号木板包含段 $(1,2,3,4,5,6)$，颜色序列为 $(1,2,2,3,1,2)$。

第一次切割操作，切下 1 号木板上段落编号在 $[3,4]$ 的部分，作为 2 号木板：

- 2 号木板包含段 $(3,4)$，其颜色序列为 $(2,3)$；
- 1 号木板剩余段 $(1,2,5,6)$，其颜色序列为 $(1,2,1,2)$。

第二次切割操作，切下 1 号木板上段落编号在 $[5,5]$ 的部分，作为 3 号木板：

- 3 号木板包含段 (5)，其颜色序列为 (1)；
- 1 号木板剩余段 $(1,2,6)$，其颜色序列为 $(1,2,2)$。

第三次切割操作，切下 1 号木板上段落编号在 $[4,5]$ 的部分，作为 4 号木板：

- 因为 1 号木板上已经不含段 4 和 5，新切下的 4 号木板为空；

第四次切割操作，切下 1 号木板上段落编号在 $[1,6]$ 的部分，作为 5 号木板：

- 5 号木板包含段 $(1,2,6)$，其颜色序列为 $(1,2,2)$；
- 1 号木板剩余部分为空。

第五次切割操作，切下 2 号木板上段落编号在 $[4,4]$ 的部分，作为 6 号木板：

- 6 号木板包含段 (4)，其颜色序列为 (3)；
- 2 号木板剩余段 (3)，其颜色序列为 (2)。

【子任务】

测试点编号	n,m,k	特殊性质
$1 \sim 5$	≤ 2000	无
$6 \sim 7$	$\leq 10^5$	A
$8 \sim 10$	$\leq 10^5$	B
$11 \sim 13$	$\leq 10^5$	C
$14 \sim 20$	$\leq 10^5$	无

- 特殊性质 A：$r_i - l_i \leq 1000$；
- 特殊性质 B：$c_i = i$；
- 特殊性质 C：对于所有的 $1 \leq i < j \leq n$ 满足 $c_i \leq c_j$。

全部的数据满足 $1 \leq n, m, k \leq 10^5$、$1 \leq c_i \leq m$、$1 \leq l_i \leq r_i \leq n$ 和 $1 \leq x_i \leq i$。