2023 CCF 非专业级软件能力认证

CSP-J/S 2023 第二轮认证

提高级

时间：2023 年 10 月 21 日 14:30 ～ 18:30

题目名称	密码锁	消消乐	结构体	种树
题目类型	传统型	传统型	传统型	传统型
目录	`lock`	`game`	`struct`	`tree`
可执行文件名	`lock`	`game`	`struct`	`tree`
输入文件名	`lock.in`	`game.in`	`struct.in`	`tree.in`
输出文件名	`lock.out`	`game.out`	`struct.out`	`tree.out`
每个测试点时限	1.0 秒	1.0 秒	1.0 秒	1.0 秒
内存限制	512 MiB	512 MiB	512 MiB	512 MiB
测试点数目	10	20	20	20
测试点是否等分	是	是	是	是

提交源程序文件名

对于 C++ 语言	`lock.cpp`	`game.cpp`	`struct.cpp`	`tree.cpp`

编译选项

对于 C++ 语言	`-O2 -std=c++14 -static`

注意事项（请仔细阅读）

1. 文件名（程序名和输入输出文件名）必须使用英文小写。
2. C/C++ 中函数 main() 的返回值类型必须是 int，程序正常结束时的返回值必须是 0。
3. 提交的程序代码文件的放置位置请参考各省的具体要求。
4. 因违反以上三点而出现的错误或问题，申诉时一律不予受理。
5. 若无特殊说明，结果的比较方式为全文比较（过滤行末空格及文末回车）。
6. 选手提交的程序源文件必须不大于 100KB。
7. 程序可使用的栈空间内存限制与题目的内存限制一致。
8. 全国统一评测时采用的机器配置为：Intel(R) Core(TM) i7-8700K CPU @3.70GHz，内存 32GB。上述时限以此配置为准。
9. 只提供 Linux 格式附加样例文件。
10. 评测在当前最新公布的 NOI Linux 下进行，各语言的编译器版本以此为准。

密码锁（lock）

【题目描述】

小 Y 有一把五个拨圈的密码锁。如图所示，每个拨圈上是从 0 到 9 的数字。每个拨圈都是从 0 到 9 的循环，即 9 拨动一个位置后可以变成 0 或 8，

图 1: 密码锁

因为校园里比较安全，小 Y 采用的锁车方式是：从正确密码开始，随机转动密码锁仅一次；每次都是以某个幅度仅转动一个拨圈或者同时转动两个相邻的拨圈。

当小 Y 选择同时转动两个相邻拨圈时，两个拨圈转动的幅度相同，即小 Y 可以将密码锁从 `0 0 1 1 5` 转成 `1 1 1 1 5`，但不会转成 `1 2 1 1 5`。

时间久了，小 Y 也担心这么锁车的安全性，所以小 Y 记下了自己锁车后密码锁的 n 个状态，注意这 n 个状态都不是正确密码。

为了检验这么锁车的安全性，小 Y 有多少种可能的正确密码，使得每个正确密码都能够按照他所采用的锁车方式产生锁车后密码锁的全部 n 个状态。

【输入格式】

从文件 ***lock.in*** 中读入数据。

输入的第一行包含一个正整数 n，表示锁车后密码锁的状态数。

接下来 n 行每行包含五个整数，表示一个密码锁的状态。

【输出格式】

输出到文件 ***lock.out*** 中。

输出一行包含一个整数，表示密码锁的这 n 个状态按照给定的锁车方式能对应多少种正确密码。

【样例 1 输入】

```
1
0 0 1 1 5
```

【样例 1 输出】

```
81
```

【样例 1 解释】

一共有 81 种可能的方案。
其中转动一个拨圈的方案有 45 种，转动两个拨圈的方案有 36 种。

【样例 2】

见选手目录下的 ***lock/lock2.in*** 与 ***lock/lock2.ans***。

【数据范围】

对于所有测试数据有：$1 \leq n \leq 8$。

测试点	$n \leq$	特殊性质
$1 \sim 3$	1	无
$4 \sim 5$	2	无
$6 \sim 8$	8	A
$9 \sim 10$	8	无

特殊性质 A：保证所有正确密码都可以通过仅转动一个拨圈得到测试数据给出的 n 个状态。

消消乐（game）

【题目描述】

小 L 现在在玩一个低配版本的消消乐，该版本的游戏是一维的，一次也只能消除两个相邻的元素。

现在，他有一个长度为 n 且仅由小写字母构成的字符串。我们称一个字符串是可消除的，当且仅当可以对这个字符串进行若干次操作，使之成为一个空字符串。

其中每次操作可以从字符串中删除两个相邻的相同字符，操作后剩余字符串会拼接在一起。

小 L 想知道，这个字符串的所有非空连续子串中，有多少个是可消除的。

【输入格式】

从文件 ***game.in*** 中读入数据。

输入的第一行包含一个正整数 n，表示字符串的长度。

输入的第二行包含一个长度为 n 且仅由小写字母构成的的字符串，表示题目中询问的字符串。

【输出格式】

输出到文件 ***game.out*** 中。

输出一行包含一个整数，表示题目询问的答案。

【样例 1 输入】

```
8
accabccb
```

【样例 1 输出】

```
5
```

【样例 1 解释】

一共有 5 个可消除的连续子串，分别是 cc、acca、cc、bccb、accabccb。

【样例 2】

见选手目录下的 ***game/game2.in*** 与 ***game/game2.ans***。

【样例 3】

见选手目录下的 ***game/game3.in*** 与 ***game/game3.ans***。

【样例 4】

见选手目录下的 ***game/game4.in*** 与 ***game/game4.ans***。

【数据范围】

对于所有测试数据有：$1 \leq n \leq 2 \times 10^6$，且询问的字符串仅由小写字母构成。

测试点	$n \leq$	特殊性质
$1 \sim 5$	10	无
$6 \sim 7$	800	无
$8 \sim 10$	8000	无
$11 \sim 12$	2×10^5	A
$13 \sim 14$	2×10^5	B
$15 \sim 17$	2×10^5	无
$18 \sim 20$	2×10^6	无

特殊性质 A：字符串中的每个字符独立等概率地从字符集中选择。
特殊性质 B：字符串仅由 `a` 和 `b` 构成。

结构体（struct）

【题目背景】

在 C++ 等高级语言中，除了 `int` 和 `float` 等基本类型外，通常还可以自定义结构体类型。在本题当中，你需要模拟一种类似 C++ 的高级语言的结构体定义方式，并计算出相应的内存占用等信息。

【题目描述】

在这种语言中，基本类型共有 4 种：`byte`，`short`，`int`，`long`，分别占据 $1, 2, 4, 8$ 字节的空间。

定义一个结构体**类型**时，需要给出**类型名**和**成员**，其中每个成员需要按顺序给出**类型**和**名称**。类型可以为基本类型，也可以为**先前定义过**的结构体类型。注意，定义结构体**类型**时不会定义具体元素，即不占用内存。

定义一个**元素**时，需要给出元素的**类型**和**名称**。元素将按照以下规则占据内存：

- 元素内的所有成员将按照**定义时给出的顺序**在内存中排布，对于类型为结构体的成员同理。
- 为了保证内存访问的效率，元素的地址占用需要满足**对齐规则**，即任何类型的**大小**和该类型元素在内存中的**起始地址**均应对齐到该类型对齐要求的**整数倍**。具体而言：
  - 对于基本类型：对齐要求等于其占据空间大小，如 `int` 类型需要对齐到 4 字节，其余同理。
  - 对于结构体类型：对齐要求等于其成员的对齐要求的**最大值**，如一个含有 `int` 和 `short` 的结构体类型需要对齐到 4 字节。

以下是一个例子（以 C++ 语言的格式书写）：

```
struct d {
    short a;
    int b;
    short c;
};
d e;
```

该代码定义了结构体类型 `d` 与元素 `e`。元素 `e` 包含三个成员 `e.a`，`e.b`，`e.c`，分别占据第 $0 \sim 1$，$4 \sim 7$，$8 \sim 9$ 字节的地址。由于类型 `d` 需要对齐到 4 字节，因此 `e` 占据了第 $0 \sim 11$ 字节的地址，大小为 12 字节。

你需要处理 n 次操作，每次操作为以下四种之一：

1. 定义一个结构体类型。具体而言，给定正整数 k 与字符串 $s, t_1, n_1, \ldots, t_k, n_k$，其中 k 表示该类型的成员数量，s 表示该类型的类型名，$t_1, t_2, \ldots, t_k$ 按顺序分别表示每个成员的类型，$n_1, n_2, \ldots, n_k$ 按顺序分别表示每个成员的名称。你需要输出该结构体类型的大小和对齐要求，用一个空格分隔。
2. 定义一个元素，具体而言，给定字符串 t，n 分别表示该元素的类型与名称。所有被定义的元素将按顺序，从内存地址为 0 开始依次排开，并需要满足地址对齐规则。你需要输出新定义的元素的起始地址。
3. 访问某个元素。具体而言，给定字符串 s，表示所访问的元素。与 C++ 等语言相同，采用 `.` 来访问结构体类型的成员。如 `a.b.c`，表示 `a` 是一个已定义的元素，它是一个结构体类型，有一个名称为 `b` 的成员，它也是一个结构体类型，有一个名称为 `c` 的成员。你需要输出如上被访问的**最内层**元素的起始地址。
4. 访问某个内存地址。具体而言，给定非负整数 $addr$，表示所访问的地址，你需要判断是否存在一个**基本类型**的元素占据了该地址。若是，则按操作 3 中的访问元素格式输出该元素；否则输出 `ERR`。

【输入格式】

从文件 ***struct.in*** 中读入数据。

第 1 行：一个正整数 n，表示操作的数量。

接下来若干行，依次描述每个操作，每行第一个正整数 op 表示操作类型：

- 若 $op = 1$，首先输入一个字符串 s 与一个正整数 k，表示类型名与成员数量，接下来 k 行每行输入两个字符串 t_i，n_i，依次表示每个成员的类型与名称。
- 若 $op = 2$，输入两个字符串 t，n，表示该元素的类型与名称。
- 若 $op = 3$，输入一个字符串 s，表示所访问的元素。
- 若 $op = 4$，输入一个非负整数 $addr$，表示所访问的地址。

【输出格式】

输出到文件 ***struct.out*** 中。

输出 n 行，依次表示每个操作的输出结果，输出要求如题目描述中所述。

【样例 1 输入】

```
5
1 a 2
short aa
int ab
1 b 2
a ba
```

```
long bb
2 b x
3 x.ba.ab
4 10
```

【样例 1 输出】

```
8 4
16 8
0
4
x.bb
```

【样例 1 解释】

结构体类型 a 中，int 类型的成员 aa 占据第 0 ～ 3 字节地址，short 类型的成员 ab 占据第 4 ～ 5 字节地址。又由于其对齐要求为 4 字节，可得其大小为 8 字节。由此可同理计算出结构体类型 b 的大小为 16 字节，对齐要求为 8 字节。

【样例 2】

见选手目录下的 ***struct/struct2.in*** 与 ***struct/struct2.ans***。

【样例 2 解释】

第二个操作 4 中，访问的内存地址恰好在为了地址对齐而留下的“洞”里，因此没有基本类型元素占据它。

【样例 3】

见选手目录下的 ***struct/struct3.in*** 与 ***struct/struct3.ans***。

【数据范围】

对于全部数据，满足 $1 \leq n \leq 100$，$1 \leq k \leq 100$，$0 \leq addr \leq 10^{18}$。

所有定义的结构体类型名、成员名称和定义的元素名称均由不超过 10 个字符的小写字母组成，且都不是 byte,short,int,long（即不与基本类型重名）。

所有定义的结构体类型名和元素名称互不相同，同一结构体内成员名称互不相同。但不同的结构体可能有相同的成员名称，某结构体内的成员名称也可能与定义的结构体或元素名称相同。

保证所有操作均符合题目所述的规范和要求，即结构体的定义不会包含不存在的类型、不会访问不存在的元素或成员等。

保证任意结构体大小及定义的元素占据的最高内存地址均不超过 10^{18}。

测试点编号	特殊性质
1	A、D
$2 \sim 3$	A
$4 \sim 5$	B、D
$6 \sim 8$	B
$9 \sim 10$	C、D
$11 \sim 13$	C
$14 \sim 16$	D
$17 \sim 20$	无

特殊性质 A：没有操作 1；

特殊性质 B：只有一个操作 1；

特殊性质 C：所有操作 1 中给出的成员类型均为基本类型；

特殊性质 D：基本类型只有 `long`。

【提示】

对于结构体类型的对齐要求和大小，形式化的定义方式如下：

- 设该结构体内有 k 个成员，其大小分别为 $s_1, ..., s_k$，对齐要求分别为 $a_1, ..., a_k$；
- 则该结构体的对齐要求为 $a = \max\{a_1, ..., a_k\}$；
- 再设这些成员排布时的**地址偏移量**分别为 $o_1, ..., o_k$，则：
  - $o_1 = 0$；
  - 对于 $i = 2, ..., k$，o_i 为满足 $o_{i-1} + s_{i-1} \le o_i$ 且 a_i 整除 o_i 的最小值；
  - 则该结构体的大小 s 为满足 $o_k + s_k \le s$ 且 a 整除 s 的最小值；

对于定义元素时的内存排布，形式化的定义方式如下：

- 设第 i 个被定义的元素大小为 s_i，对齐要求为 a_i，起始地址为 b_i；
- 则 $b_1 = 0$，对于 $2 \le i$，b_i 为满足 $b_{i-1} + s_{i-1} \le b_i$ 且 a_i 整除 b_i 的最小值。

种树（tree）

【题目描述】

你是一个森林养护员，有一天，你接到了一个任务：在一片森林内的地块上种树，并养护至树木长到指定的高度。

森林的地图有 n 片地块，其中 1 号地块连接森林的入口。共有 $n-1$ 条道路连接这些地块，使得每片地块都能通过道路互相到达。最开始，每片地块上都没有树木。

你的目标是：在每片地块上均种植一棵树木，并使得 i 号地块上的树的高度生长到不低于 a_i 米。

你每天可以选择一个未种树且**与某个已种树的地块直接邻接（即通过单条道路相连）**的地块，种一棵高度为 0 米的树。如果所有地块均已种过树，则你当天不进行任何操作。特别地，第 1 天你只能在 1 号空地种树。

对每个地块而言，从该地块被种下树的当天开始，该地块上的树每天都会生长一定的高度。由于气候和土壤条件不同，在第 x 天，i 号地块上的树会长高 $\max(b_i + x * c_i, 1)$ 米。注意这里的 x 是从整个任务的第一天，而非种下这棵树的第一天开始计算。

你想知道：最少需要多少天能够完成你的任务？

【输入格式】

从文件 ***tree.in*** 中读入数据。

输入的第一行包含一个正整数 n，表示森林的地块数量。

接下来 n 行：每行包含三个整数 a_i, b_i, c_i，分别描述一片地块，含义如题目描述中所述。

接下来 $n-1$ 行：每行包含两个正整数 u_i, v_i，表示一条连接地块 u_i 和 v_i 的道路。

【输出格式】

输出到文件 ***tree.out*** 中。

输出一行仅包含一个正整数，表示完成任务所需的最少天数。

【样例 1 输入】

```
4
12 1 1
2 4 -1
10 3 0
7 10 -2
1 2
```

```
1 3
3 4
```

【样例 1 输出】

```
5
```

【样例 1 解释】

第 1 天：在地块 1 种树，地块 1 的树木长高至 2 米。
第 2 天：在地块 3 种树，地块 1,3 的树木分别长高至 5,3 米。
第 3 天：在地块 4 种树，地块 1,3,4 的树木分别长高至 9,6,4 米。
第 4 天：在地块 2 种树，地块 1,2,3,4 的树木分别长高至 14,1,9,6 米。
第 5 天：地块 1,2,3,4 的树木分别长高至 20,2,12,7 米。

【样例 2】

见选手目录下的 ***tree/tree2.in*** 与 ***tree/tree2.ans***。

【样例 3】

见选手目录下的 ***tree/tree3.in*** 与 ***tree/tree3.ans***。

【样例 4】

见选手目录下的 ***tree/tree4.in*** 与 ***tree/tree4.ans***。

【数据范围】

对于所有测试数据有：$1 \le n \le 10^5$，$1 \le a_i \le 10^{18}$, $1 \le b_i \le 10^9$，$0 \le |c_i| \le 10^9$，$1 \le u_i, v_i \le n$。保证存在方案能在 10^9 天内完成任务

测试点编号	$n \leq$	特殊性质
1	20	A
$2 \sim 4$		无
$5 \sim 6$	500	A
$7 \sim 8$	10^5	
$9 \sim 10$		B
$11 \sim 13$		C
$14 \sim 16$		D
$17 \sim 20$		无

特殊性质 A：对于所有 $1 \leq i \leq n$，均有 $c_i = 0$；
特殊性质 B：对于所有 $1 \leq i < n$，均有 $u_i = i$，$v_i = i + 1$；
特殊性质 C：与任何地块直接相连的道路均不超过 2 条；
特殊性质 D：对于所有 $1 \leq i < n$，均有 $u_i = 1$。