# 题目

设计并实现一个迭代压缩字符串的数据结构。给定的压缩字符串的形式是，每个字母后面紧跟一个正整数，表示该字母在原始未压缩字符串中出现的次数。

设计一个数据结构，它支持如下两种操作： next 和 hasNext。

next() - 如果原始字符串中仍有未压缩字符，则返回下一个字符，否则返回空格。

hasNext() - 如果原始字符串中存在未压缩的的字母，则返回true，否则返回false。

示例 1：

输入：

["StringIterator", "next", "next", "next", "next", "next", "next", "hasNext", "next", "hasNext"]

[["L1e2t1C1o1d1e1"], [], [], [], [], [], [], [], [], []]

输出：

[null, "L", "e", "e", "t", "C", "o", true, "d", true]

解释：

StringIterator stringIterator = new StringIterator("L1e2t1C1o1d1e1");

stringIterator.next(); // 返回 "L"

stringIterator.next(); // 返回 "e"

stringIterator.next(); // 返回 "e"

stringIterator.next(); // 返回 "t"

stringIterator.next(); // 返回 "C"

stringIterator.next(); // 返回 "o"

stringIterator.hasNext(); // 返回 True

stringIterator.next(); // 返回 "d"

stringIterator.hasNext(); // 返回 True

提示：

1 <= compressedString.length <= 1000

compressedString 由小写字母、大写字母和数字组成。

在 compressedString 中，单个字符的重复次数在 [1,10^9] 范围内。

next 和 hasNext 的操作数最多为 100 。

# 分析

要解决这个问题，我们需要设计一个迭代器，用于解压压缩字符串（格式为“字符+重复次数”），支持 next()（返回下一个未压缩字符）和 hasNext()（判断是否还有未压缩字符）两种操作。核心思路是提前解析压缩字符串的结构，将“字符-剩余次数”的对应关系存储起来，再通过维护当前字符和剩余次数，高效响应迭代操作。

思路：

1、解析压缩字符串：压缩字符串的结构是“字符+1个或多个数字”的循环（如“L1e2”对应“L”重复1次、“e”重复2次）。需要遍历压缩字符串，分离出每个字符及其对应的重复次数，存储到一个列表中（避免每次迭代时重复解析）。

2、维护迭代状态：用两个变量跟踪当前迭代位置：

- currentIdx：指向“字符-次数”列表中当前正在处理的字符索引；

- remaining：当前字符剩余的未返回次数（初始为列表中当前字符的总次数，每调用一次next()减1）。

3、响应迭代操作：

- hasNext()：判断是否存在未返回的字符（即是否还有未处理的字符，或当前字符仍有剩余次数）；

- next()：若有未返回字符，返回当前字符并减少剩余次数；若当前字符已用完，切换到下一个字符并更新剩余次数；若无字符可返回，返回空格。

代码：

class StringIterator {

private:

// 存储解析后的“字符-总次数”对

vector<pair<char, long long>> charCounts;

// currentIdx：当前处理的charCounts的索引；remaining：当前字符剩余的未返回次数

int currentIdx;

long long remaining;

public:

// 构造函数：解析压缩字符串，初始化charCounts

StringIterator(string compressedString) {

int n = compressedString.size();

int i = 0;

while (i < n) {

// 步骤1：提取当前字符（压缩字符串中，非数字的字符即为字母）

char c = compressedString[i++];

// 步骤2：提取当前字符的重复次数（可能是多位数，甚至超过int范围，用long long存储）

long long count = 0;

while (i < n && isdigit(compressedString[i])) {

count = count \* 10 + (compressedString[i] - '0');

i++;

}

// 步骤3：将“字符-次数”对加入列表

charCounts.emplace\_back(c, count);

}

// 初始化迭代状态：从第一个字符开始，剩余次数为第一个字符的总次数

currentIdx = 0;

remaining = (charCounts.empty() ? 0 : charCounts[0].second);

}

// 返回下一个未压缩字符，无字符则返回空格

char next() {

// 若没有未返回的字符，返回空格

if (!hasNext()) {

return ' ';

}

// 步骤1：获取当前字符

char currChar = charCounts[currentIdx].first;

// 步骤2：减少当前字符的剩余次数

remaining--;

// 步骤3：若当前字符已用完，切换到下一个字符

if (remaining == 0) {

currentIdx++;

// 若下一个字符存在，更新remaining为下一个字符的总次数

if (currentIdx < charCounts.size()) {

remaining = charCounts[currentIdx].second;

}

}

return currChar;

}

// 判断是否还有未压缩的字符

bool hasNext() {

// 条件：当前索引未超出charCounts范围，且当前字符仍有剩余次数

return currentIdx < charCounts.size() && remaining > 0;

}

};

/\*\*

\* Your StringIterator object will be instantiated and called as such:

\* StringIterator\* obj = new StringIterator(compressedString);

\* char param\_1 = obj->next();

\* bool param\_2 = obj->hasNext();

\*/

代码解释：

1、构造函数解析压缩字符串：

- 遍历压缩字符串，先提取非数字的字符作为当前字母；

- 接着提取后续所有连续数字，转换为long long类型的次数（避免次数超过`int`范围，题目中次数最大为10^9，long long可覆盖）；

- 将“字母-次数”对存入`charCounts`列表，完成初始化。

2、hasNext()判断逻辑：

- 核心条件是“当前处理的字符索引未超出列表范围”且“当前字符仍有剩余次数”，两者同时满足则存在未压缩字符。

3、next()返回逻辑：

- 先通过hasNext()判断是否有字符可返回，无则返回空格；

- 有则返回当前字符，并将剩余次数减1；

- 若当前字符剩余次数为0，切换到下一个字符，并更新剩余次数为下一个字符的总次数（若下一个字符存在）。

关键细节

- 次数存储类型：题目中重复次数最大为10^9，超过int（通常为32位，最大约2^31-1=21亿）的上限，因此用long long存储次数，避免溢出。

- 提前解析：构造函数中一次性解析完压缩字符串，后续next()和hasNext()无需重复解析，提升效率（操作数最多100次，性能无压力）。

- 边界处理：当压缩字符串为空（虽题目提示长度≥1，但代码兼容）或所有字符已返回时，next()返回空格，hasNext()返回false。