# 题目

给你一个文件，并且该文件只能通过给定的 read4 方法来读取，请实现一个方法使其能够读取 n 个字符。

read4 方法：

API read4 可以从文件中读取 4 个连续的字符，并且将它们写入缓存数组 buf 中。

返回值为实际读取的字符个数。

注意 read4() 自身拥有文件指针，很类似于 C 语言中的 FILE \*fp 。

read4 的定义：

参数类型: char[] buf4

返回类型: int

注意: buf4[] 是目标缓存区不是源缓存区，read4 的返回结果将会复制到 buf4[] 当中。

下列是一些使用 read4 的例子：

File file("abcde"); // 文件名为 "abcde"， 初始文件指针 (fp) 指向 'a'

char[] buf4 = new char[4]; // 创建一个缓存区使其能容纳足够的字符

read4(buf4); // read4 返回 4。现在 buf4 = "abcd"，fp 指向 'e'

read4(buf4); // read4 返回 1。现在 buf4 = "e"，fp 指向文件末尾

read4(buf4); // read4 返回 0。现在 buf = ""，fp 指向文件末尾

read 方法：

通过使用 read4 方法，实现 read 方法。该方法可以从文件中读取 n 个字符并将其存储到缓存数组 buf 中。您 不能 直接操作文件。

返回值为实际读取的字符。

read 的定义：

参数类型: char[] buf, int n

返回类型: int

注意: buf[] 是目标缓存区不是源缓存区，你需要将结果写入 buf[] 中。

示例 1：

输入：file = "abc", n = 4

输出：3

解释：当执行你的read方法后，buf 需要包含 "abc"。 文件一共 3 个字符，因此返回 3。 注意 "abc" 是文件的内容，不是 buf 的内容，buf 是你需要写入结果的目标缓存区。

示例2：

输入：file = "abcde", n = 5

输出：5

解释：当执行你的read方法后，buf需要包含"abcde"。文件共5个字符，因此返回5。

示例3:

输入：file = "abcdABCD1234", n = 12

输出：12

解释：当执行你的read方法后，buf需要包含"abcdABCD1234"。文件一共12个字符，因此返回12。

示例4:

输入：file = "leetcode", n = 5

输出：5

解释：当执行你的read方法后，buf需要包含"leetc"。文件中一共5个字符，因此返回5。

提示：

你不能直接操作该文件，文件只能通过read4获取而不能通过read。

read 函数只在每个测试用例调用一次。

你可以假定目标缓存数组 buf 保证有足够的空间存下 n个字符。

# 分析

## 方法一

这个题目我个人理解，是想考察设计一些通用API。 设计API时，要对可能的输入做一些假设。

比如：输入参数为(buf, n) 那么对于调用者，可能会用如下的方式：

··

char buf[1000];

f.read(buf, 1000);

··

也就是说，我们应该保证代码不会对buf中，下标超过n的地方进行读写。

还有一点就是，如果是现实中的读文件，会用到IO，消耗很大。所以对于文件的读，我们应保证次数最少。 也就是说，如果某次读取发现已经eof了（read4返回值小于4），应该不能再有读操作了。

除此之外，就是一些简单能看出来的逻辑： 如果文件大小<n，则返回全部文件，返回值为文件大小。否则返回前n个字符，返回值为n。

所以，虽然题目没说，但是作为api开发者，还是要做一些综合考虑的。 （个人的一个实现，权当抛砖引玉吧）

class Solution {

public:

/\*\*

\* @param buf Destination buffer

\* @param n Number of characters to read

\* @return The number of actual characters read

\*/

int read(char \*buf, int n) {

int needBytes = n;

int curIdx = 0;

char internalBuf[4];

bool fileEnd = false;

while (true) {

int bytes = read4(internalBuf);

fileEnd = bytes < 4;

int idx = 0;

while (needBytes > 0 && idx < bytes) {

buf[curIdx++] = internalBuf[idx++];

--needBytes;

}

if (needBytes == 0 || fileEnd) {

break;

}

}

return curIdx;

}

};

## 方法二

// Forward declaration of the read4 API.

int read4(char \*buf);

class Solution {

public:

/\*\*

\* @param buf Destination buffer

\* @param n Number of characters to read

\* @return The number of actual characters read

\*/

int read(char \*buf, int n) {

if (buf == nullptr || n <= 0)

return 0;

int total = 0, current = 0;

// 循环读文件，直到文件全部读完或者超过需要的字符

do {

current = read4(buf + total);

total += current;

} while (current == 4 && total < n);

// 如果有读入的字符超过需要的字符数，例如：“abcd”，但是只要读3个字符

// 那么需要清除多余的字符

if (total > n) {

buf[n] = 0;

total = n;

}

return total;

}

};