LAB1 JumpIntoDSA report

Qianyi Fan 2020013250

March 8, 2024

1 黑盒测试

1.1 思考题 1

如果评测系统上时间限制为 1s，那么助教编写的指令所花费的时间最多为 0.5s。又因为评测系统每秒钟执行 10^8 条指令，当助教编写的程序执行 $10n^2$ 条指令时，有 $10n^2 = 0.5 \times 10^8$，解得 $n = 2236$。同理，当助教编写的程序执行 $20n \log_2 n$ 条指令时，有 $20n \log_2 n = 0.5 \times 10^8$，解得 $n = 145746$。

应该准备一部分 n 较大 m 较小的数据，测试与 n 相关的复杂度，控制其小于 $O(n)$ 级别；同时应该准备一部分 m 较大 n 较小的数据，测试与 m 相关的复杂度，控制其小于 $O(\log m)$ 级别。再准备一部分数据，测试 m 与 n 数据规模相当的情况，可设置为梯度的形式，满足一般的数据输入的规模。

1.2 思考题 2

当程序执行了不可忽略量级的具有相同用时的指令数目时，我们就可以假设执行一条指令的用时是确定的，这通常出现于循环体中。而上述所说的情况，如缓存读取，不同类型数据运算，区别是常量级的。

2 调试和测试

2.1 `Solution1.cpp` 和 `Solution2.cpp` 中的 bug

1. 在初始化原矩阵和 `rowsum` 矩阵时，应该修改 `maxn=2001`，这样才能保证在接下来对于 `matrix` 中忽略 0 行 0 列元素实现中，不会发生数组访问越界；

2. sum 变量的定义以及初始化为 0 应该在每一次查询的循环体里面进行，而不是在累加运算的循环体里进行；

3. sum 变量应该定义为 long long 型变量，因为在之后大规模的数据计算中 sum 的取值可能超过 int 数据类型的取值范围；

4. solution2.cpp 中，sum 的计算应该为 sum += rowsum[x+j][y+b-1] - rowsum[x+j][y-1]。

2.2 无限递归程序 gdb 调试时报错

报错信息为：Program received signal SIGSEGV, Segmentation fault.
0x00007ffff7bec660 in _IO_new_file_xsputn (n=6, data=0x7fffff7ff0a2, f=<optimized out>) at ./libio/fileops.c:1235
1235 ./libio/fileops.c: No such file or directory.

报错的原因是在递归调用过程中，函数调用的堆栈空间可能会耗尽，导致程序发生段错误（Segmentation fault）（报错的第一行）。堆栈空间是用来存储函数调用及其局部变量的内存区域，当递归层级过深时，堆栈空间可能会不足，从而导致段错误。后面两行的报错是 GDB 提供的堆栈跟踪信息的一部分，它显示了正在执行的程序在内存中的地址（0x00007ffff7bec660）、调用的函数（_IO_new_file_xsputn）、函数参数（n=6, data=0x7fffff7ff0a2, f=<optimized out>），以及函数所在的源文件和行号（./libio/fileops.c:1235）。

2.3 gdb 调试

在命令行中输入 g++ -g -o solution1 solution_1.cpp 来编译需要调试的程序。（需要使用 -g 编译选项）。在 gdb 调试中，首先用 gdb ./solution1 命令进入调试，并且设置断点（b <行数>）。开始执行程序（r）。不断跳到下一步（n），gdb 会给出当前行的代码。中途可以随时打印某个变量的值（p matrix[0][0]）。注意到逐步执行代码时，遇到 for 循环 gdb 会如实地根据条件重复返回到对应语句。调试成功后，退出程序（q）。

2.4 rand_input.cpp 中 srand(time(0)) 的作用

srand 函数和 rand 函数都是该文件包含的 C 语言标准库 <cstdlib> 中的函数，其中 srand 函数用于设置随机数种子，在该文件中通过当前时间来设置，便于后续 rand 函数生成伪随机数。

2.5 battle.cpp 中 system() 的作用

system() 函数接受字符串为输入，字符串即为系统命令。因此前四条 system 函数调用是在编译并链接该四个文件，使其成为可执行文件。后续两条是在运行 rand_input 可执行文件，并将输出（矩阵信息，测试信息）重定向到 rand.in 文件，后又将其输出至 check_input 可执行文件中检查生成的数据是否合法。后将 rand.in 文件输出至 solution_1 和 2 可执行文件中，并将两程序的输出重定向到 .out 文件，并通过 diff 指令观察输出。

2.6 gdb 调试过程中的文件重定向

可以先创建一个文本文件，储存我们执行程序时所要用到的所有输入参数，然后在 gdb 调试中，在执行程序一步用 '<' 符号进行文件重定向（r < input.txt）。这样就会将我们的文件直接输入到 gdb 调试程序中。

3 多文件项目

3.1 -Wall 编译选项出现警告

-Wall 是 g++ 编译器的一个选项，用于启用额外的警告信息。具体来说，它会开启一组额外的警告，用于检测代码中可能存在的潜在问题或不规范的代码风格。在此处特别地，警告说明了 ArrayN、ArrayM、N、M、Sum 和 Matrix 变量虽然在 solve.h 头文件里面定义了，但是之后没有使用到。可以通过删掉 .h 文件中对应变量的声明消除警告。

3.2 命令行参数 argv[0] 的含义

命令行参数列表中的第一个元素是程序的名称或路径，其本质是一个指针，它是程序启动时传递给 main 函数的参数之一。

3.3 关于文件的编译、运行和面向对象风格的类定义

编译时，输入 g++ -o main main.cpp solve.cpp 可对两个源文件直接进行编译并链接。当然也可以分开编译，再链接目标文件（第一步 g++ -c main.o main.cpp，第二步 g++ -c solve.o solve.cpp，第三步 g++ -o main main.o solve.o）。运行时在当前目录下输入 ./main <命令行参数> 运行。此处的命令行参数输入 1 或 2，分别代表使用 1 号和 2 号算法。（当命令行参数不为 1 或 2 时，程序中断报错）。

改写为面向对象风格时，类的定义为：

```
class Solver {
public:
    int* Matrix;
    int* Sum;
    int N, M, ArrayN, ArrayM;
    virtual void initialize(int *matrix, int *sum, int n, int m,
                            int arrayN, int arrayM) = 0;
    virtual int solve(Task task) = 0;
};

class Solver1 : public Solver {
public:
    virtual void initialize(int *matrix, int *sum, int n, int m,
                            int arrayN, int arrayM) override;
    virtual int solve(Task task) override;
};

class Solver2 : public Solver {
public:
    virtual void initialize(int *matrix, int *sum, int n, int m,
                            int arrayN, int arrayM) override;
    virtual int solve(Task task) override;
};
```

4 优化和比较

4.1 Solution3.cpp 优化的思路

进一步对原始矩阵进行预处理，生成 ULsum 矩阵，其中每一个位置的元素代表原矩阵原位置包含该元素的左上角的所有元素的和。Solution3 中，生成 ULsum 矩阵的时间复杂度为 $O(mn)$，而每次查询只需要 4 次查询 ULsum 矩阵中的元素的操作，因此每次查询时时间复杂度为 $O(1)$。查询次数为 q，则查询部分的时间复杂度为 $O(q)$。Solution3 总的时间复杂度为 $O(mn+q)$。同理，分析 solution2 的算法，时间复杂度应该为 $O(mn+mq)$。分析 solution1 的算法，时间复杂度应该为 $O(mnq)$。易知，solution3 算法的时间复杂度最低。

当 $n = 100$, $m = 1900$, $q = 100000$ 时，`solution1` 运行时间为 3.559 s，`solution2` 运行时间为 0.533s，`solution3` 运行时间为 0.386 s。

4.2 使得 Solution3 展现优势的测试数据

由上述时间复杂度的分析可知，使得 `solution3` 相比另外两个程序展现出明显优势的测试数据，需要满足查询次数 q 尽可能大；若与 `solution2` 相比也要展现出明显优势，还可以满足矩阵的列数 b 尽可能大。