1.30题解

——苏鲜乐

#A、函数调用时间

有一个单线程 CPU 正在运行一个含有 n 个函数的程序

每道函数都有一个位于 1 和 n 之间的唯一标识符

函数调用存储在一个调用栈上:

- 当一个函数调用开始时,它的标识符将会推入栈中
- 而当一个函数调用结束时,它的标识符将会从栈中弹出
- 标识符位于栈顶的函数是当前正在执行的函数

每当一个函数开始或者结束时,将会记录一条日志,包括函数标识符、是开始还是结束、以及相应的时间戳

给你一个由日志组成的列表 $logs$ 。$logs_i$ 表示第 i 条日志消息,该消息是一个 `op idx ts` 格式化的字符串,其中 op 是日志类型 `start` 或 `end` , idx 为函数的唯一标识, ts 表示当前时间戳

例如, `start 1 0` 意味着标识符为 1 的函数调用在时间戳 0 的起始开始执行。而 `end 1 2` 意味着标识符为 1 的函数调用在时间戳 2 的末尾结束执行

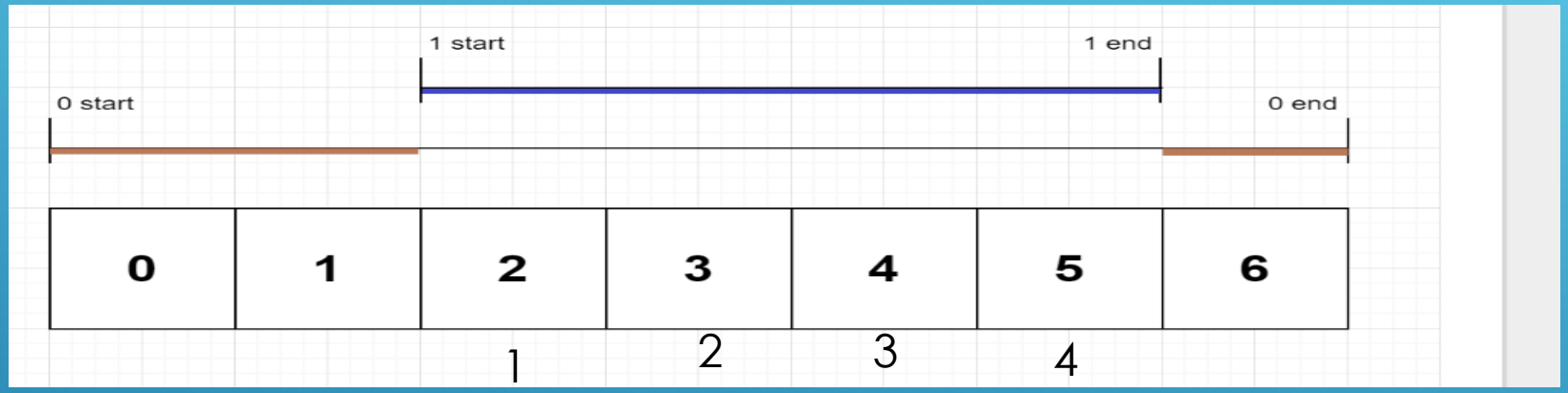
> *注意,函数可以调用多次,可能存在递归调用*

函数的**独占时间**定义是在这个函数在程序所有函数调用中执行时间的总和,调用其他函数花费的时间不算该函数的独占时间

例如,如果一个函数被调用两次,一次调用执行 2 单位时间,另一次调用执行 1 单位时间,那么该函数的 独占时间 为 $2 + 1 = 3$

现在给出日志信息请你求出所有函数的独占时间

- 注意这里是单线程：不能同时跑一个程序。用一个栈和一个结构体来维护，计时。比较简单
- 注意：时间=结束时间-开始时间+1
- 5-2+1=4




```
scanf("%s%lld%lld", op, &idx, &ts);
if (op[0] == 's')
{
  if (s.size())
    ans[s.top().idx] += ts - s.top().ts;
  s.push({idx, ts});
}
else
{
  auto cur = s.top();
  s.pop();
  ans[cur.idx] += ts - cur.ts + 1;
  if (s.size())
    s.top().ts = ts + 1;
}
```

核心代码

#D、逻辑表达式

题目描述

逻辑表达式是计算机科学中的重要概念和工具,包含逻辑值、逻辑运算、逻辑运算优先级等内容

在一个逻辑表达式中,元素的值只有两种可能： 0 (表示假)和 1 (表示真)

元素之间有多种可能的逻辑运算,本题中只需考虑如下两种：**与**(符号为 `&`)和**或**(符号为 `|`)

其运算规则如下:

$$0 \text{ and } 0 = 0 \text{ and } 1 = 1 \text{ and } 0 = 0$$
$$1 \text{ and } 1 = 1$$
$$0 \text{ or } 0 = 0$$
$$0 \text{ or } 1 = 1 \text{ or } 0 = 1 \text{ or } 1 = 1$$

在一个逻辑表达式中还可能有括号。规定在运算时,括号内的部分先运算;两种运算并列时, `&` 运算优先于 `|` 运算;同种运算并列时,从左向右运算

比如,表达式 `0|1&0` 的运算顺序等同于 `0|(1&0)`

表达式 `0&1&0|1` 的运算顺序等同于 `((0&1)&0)|1`

此外,在 C++ 等语言的有些编译器中,对逻辑表达式的计算会采用一种“**短路**的策略:

在形如 `a&b` 的逻辑表达式中,会先计算 `a` 部分的值,如果 $a = 0$,那么整个逻辑表达式的值就一定为 0 ,故无需再计算 `b` 部分的值

同理,在形如 `a|b` 的逻辑表达式中,会先计算 `a` 部分的值,如果 $a = 1$,那么整个逻辑表达式的值就一定为 1 ,无需再计算 `b` 部分的值

现在给你一个逻辑表达式,你需要计算出它的值,并且统计出在计算过程中,两种类型的**短路**各出现了多少次

需要注意的是,如果某处“短路”包含在更外层被**短路**的部分内则不被统计,如表达式 `1|(0&1)` 中,尽管 `0&1` 是一处“短路”,但由于外层的 `1|(0&1)` 本身就是一处**短路**,无需再计算 `0&1` 部分的值,因此不应当把这里的 `0&1` 计入一处**短路**

- 先将整个表达式转成后缀表达式，再维护一个栈,栈内存{a,b,c}三元组，分别表示当前值,与短路次数,或短路次数。
- 遍历后缀表达式，统计短路次数，遇到0/1，则压入栈

```
string res;
stack<char> s;
for (auto &&i : e)
{
  if (isdigit(i))
    res.push_back(i);
  else if (i == '(')
    s.push('(');
  else if (i == ')')
  {
    while (s.top() != '(')
      res.push_back(s.top()), s.pop();
    s.pop();
  }
  else
  {
    while (s.size() && fac[s.top()] >= fac[i])
      res.push_back(s.top()), s.pop();
    s.push(i);
  }
}
```

核心代码

谢谢观看！