Data Structures and Algorithms

Lecture 04 – Expression evaluation (Stacks for priority, Mathematical Induction and Catalan Number)

Pengju Ren
Institute of Artificial Intelligence and Robotics
Xi'an Jiaotong University

http://gr.xjtu.edu.cn/web/pengjuren

Problem Solving

Given an expression with arithmetic operations, how can we calculate the correct result?

Double-stack method (infix expressions)

```
def calculate_two_s(s: str) -> float:
    def compute(op, b, a):
        if op == '+': return a + b
        if op == '-': return a - b
        if op == '*': return a * b
        if op == '/': return a / b
        return 0

    pri = {'+':1, '-':1, '*':2, '/':2}
    num_s, op_s = [], []
    i, n = 0, len(s)

    while i < n:
        c = s[i]
        if c.isdigit(): # 读取完整多位数字
            j = i
            while j < n and s[j].isdigit():
                j += 1
            num_s.append(float(s[i:j]))
            i = j
        elif c == '(':
            op_s.append(c)
            #隔离重新开启一个计算子式
            i += 1
        elif c == ')':  # 计算括号内的所有运算
            while op_s and op_s[-1] != '(':
                op = op_s.pop()
                b = num_s.pop()
                a = num_s.pop()
                num_s.append(compute(op, b, a))
            op_s.pop()  # 弹出 '('
            i += 1
```

```
        elif c in '+-*/':
        # 当前运算符c优先级 <= 栈顶优先级时，先算栈顶
            while (op_s and op_s[-1] != '(' and
                   pri[c] <= pri.get(op_s[-1], 0)):
                op = op_s.pop()
                b = num_s.pop()
                a = num_s.pop()
                num_s.append(compute(op, b, a))
            op_s.append(c)
            i += 1
        else:
            i += 1  # 跳过空格等

    # 处理剩余的运算符
    while op_s:
        op = op_s.pop()
        b = num_s.pop()
        a = num_s.pop()
        num_s.append(compute(op, b, a))

    return num_s[0] if num_s else 0
```

Infix to postfix (Reverse Polish Notation) expressions

```
def infix2rpn (s: str) -> list:
    pri = {'+':1, '-':1, '*':2, '/':2}
    output = []
    stack = []
    i, n = 0, len(s)

    while i < n:
        c = s[i]
        if c.isdigit():
            j = i
            while j < n and s[j].isdigit():
                j += 1
            output.append(float(s[i:j]))
            i = j
        elif c == '(':
            stack.append(c)
        elif c == ')':
            while stack and stack[-1] != '(':
              output.append( stack.pop())
            stack.pop() #弹出 '('
        else:
          while stack and stack[-1] != '(' and pri.get(stack[-1], 0) >= pri[c]:
                output.append(stack.pop())
          stack.append(s)
        i += 1
    while stack:
        output.append(stack.pop())
    return output
```

Comprehensive Applications of stacks

- Function Call
- Expression Evaluation
- Recursive Invocation
- Browser Forward and Back
- Undo Operations in Text Editors
- Backtracking Algorithms
- Syntax Analysis in Compiler Design

Mathematical Induction (Ordinary Induction)

Definition and Principle

Mathematical Induction is a method used to prove a proposition P(n) related to natural numbers n.

It consists of two crucial steps:

- **Base Case**: Verify that the proposition $P(n_0)$ holds for the initial value $n = n_0$ (usually 0 or 1).
- **Inductive Step**: Assume that the proposition P(k) holds for $n = k$ (where $k \geq n_0$). This assumption is called the **inductive hypothesis**. Then, prove that the proposition P(k+1) also holds for $n = k+1$.

Once these two steps are completed, we can conclude that the proposition P(n) holds for all natural numbers $n \geq n_0$.

Strong Mathematical Induction (Complete Induction)

Why "Strong" Induction?

Sometimes, simply assuming P(k) is true is not enough to prove P(k+1). The validity of the proposition might depend on **all** previous cases, or a much earlier specific case (like P(k-1)). In such situations, we need to "strengthen" our assumption.

- **Base Case**: Verify that the proposition P(n_0) holds for n = n_0. (Sometimes multiple base values, like n_0, n_0+1, are needed).
- **Inductive Step**: Assume that the proposition P(m) holds for **all** m satisfying $n_0 \le m \le k$ (this is the **strong inductive hypothesis**). Then, prove that the proposition P(k+1) holds for n = k+1.

Diff Between Ordinary and Strong Induction

- **Ordinary Induction**: "Because the previous domino fell, the next one will fall."
- **Strong Induction**: "Because **all** the previous dominoes have fallen, the next one will definitely fall."

- **Note**: Logically, Ordinary Induction and Strong Induction are equivalent (within the domain of natural numbers). However, in practice, Strong Induction provides us with a more powerful tool for reasoning.

Summary

- **Stacks** can be used to adjust priority order.
- **Mathematical Induction** especially strong induction, can be used to analyze the generalization of algorithms from specific cases, which is very important for analyzing the **solvability** and **generalization** of algorithms.
- **Catalan numbers** are a core concept for analyzing algorithm complexity, designing enumeration algorithms, and understanding the behavior of data structures.

Homework 3

- **24-point game:** From a deck of playing cards, randomly draw 4 cards (J, Q, K represent 11, 12, and 13 respectively, and A represents 1). The player uses addition, subtraction, multiplication, division, and parentheses to combine these 4 numbers into an expression that equals 24. For example: given 4, 5, 6, 7, the expression (7-6+5) × 4 = 24 can be constructed.
- **Test data:**
  - Small-scale: Randomly generate 100 sets of 4 numbers from 1 to K.
  - Large-scale: Randomly generate 10000 sets of 4 numbers from 1 to K.
- **Verify the correctness of the solutions and calculate the success rate.**

Submission requirements:

1: Description of the implementation approach
2: Performance test results and analysis
3: Difficulties encountered and solutions

Homework naming: HW3-Class X-stu ID-name, e.g. HW3-Class01-12345678-张三

Submission email: chengyuma@stu.xjtu.edu.cn

Submission time: 3.22 24:00