ĐẠI HỌC QUỐC GIA TPHCM

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

KHOA KHOA HỌC MÁY TÍNH

PHÂN TÍCH VÀ THIẾT KẾ THUẬT TOÁN

Phân Tích Thuật Toán Không Đệ Quy

Nhóm 5 thực hiện

Môn học: CS112.P11.KHTN

Sinh viên thực hiện:

Nguyễn Thiên Bảo - 23520127

Trần Lê Minh Nhật - 23521098

Giáo viên hướng dẫn:

Nguyễn Thanh Sơn

ngày 26 tháng 10 năm 2024

Mục lục

1 Bài 1: Hệ thống tính toán chi phí đơn hàng

1.1 Mục tiêu

Phát triển hàm `TinhChiPhi` để tính tổng chi phí của đơn hàng, bao gồm các yếu tố:

- **Danh sách sản phẩm** với giá, số lượng, và phần trăm giảm giá.
- **Phí vận chuyển**, áp dụng nếu tổng giá trị đơn hàng nhỏ hơn 1 triệu.
- **Chiết khấu 10%** cho khách hàng thường xuyên.

1.2 Mã giả chi tiết

```
Function TinhChiPhi(Order order):
    Input:
        - order: an object containing information about products,
        customer type, and shipping fee.
    Steps:
        1. Initialize the total value before discount as
        ‘tongGiaTruocGiamGia = 0‘.
        2. Initialize the total value after discount as
        ‘tongGiaSauGiamGia = 0‘.
        3. For each product in ‘order.DanhSachSanPham‘:
            a. Calculate the product price before discount:
               - ‘giaTruocGiam = sanPham.Gia * sanPham.SoLuong‘
            b. Add this to the total value before discount:
               - ‘tongGiaTruocGiamGia += giaTruocGiam‘
            c. Calculate the product price after discount:
               - ‘giaSauGiam = giaTruocGiam * (1 - sanPham.PhanTramGiamGia / 100)‘
            d. Add this to the total value after discount:
```

```
        - ‘tongGiaSauGiamGia += giaSauGiam‘
4. Apply free shipping if applicable:
    - If ‘tongGiaTruocGiamGia >= 1 million‘:
        - Set ‘phiVanChuyen = 0‘
    - Else:
        - Set ‘phiVanChuyen = order.ShippingFee‘
5. Apply customer discount if applicable:
    - If ‘order.IsRegularCustomer == True‘:
        - Apply a 10% discount on ‘tongGiaSauGiamGia‘:
          - ‘tongGiaSauGiamGia *= 0.9‘
6. Calculate the final total cost:
    - ‘tongChiPhi = tongGiaSauGiamGia + phiVanChuyen‘
7. Return ‘tongChiPhi‘ as the final total cost.
```

1.3 Kiểm thử (Testing)

Dựa trên hướng dẫn kiểm thử từ tài liệu của bạn, dưới đây là các bước kiểm thử chi tiết với từng loại test:

1. **Unit Test**:
    - **Mục tiêu**: Kiểm tra các hàm tính toán riêng lẻ trong `TinhChiPhi`.
    - **Phần cần kiểm thử**:
        - `CalculateTotalPriceBeforeDiscount`: Tính tổng giá trị đơn hàng trước khi giảm giá.
        - `CalculateTotalPriceWithDiscount`:
          Tính tổng giá trị đơn hàng sau khi giảm giá sản phẩm.
        - `TinhChiPhi`: Kiểm tra logic tính chiết khấu và phí vận chuyển.
    - **Đặc điểm test cases**:

- **Input/Expected Output**:
  - Với `CalculateTotalPriceBeforeDiscount`, đầu vào là danh sách sản phẩm và đầu ra là tổng giá trị trước giảm giá.
  - Với `CalculateTotalPriceWithDiscount`, đầu vào bao gồm sản phẩm với các mức giảm giá và đầu ra là tổng giá trị sau khi giảm giá.
  - Với `TinhChiPhi`, đầu vào là trạng thái khách hàng và tổng giá trị đơn hàng, kết quả đầu ra là tổng chi phí cuối cùng.
- **Scenarios cụ thể**:
  - Kiểm tra các giá trị dương, âm, và 0 cho giá sản phẩm, số lượng, và phần trăm giảm giá.
  - Kiểm tra trạng thái khách hàng thường xuyên và không thường xuyên.
  - Kiểm tra tổng giá trị đơn hàng lớn hơn, nhỏ hơn, và bằng 1 triệu.

2. **White Box Test**:

- **Mục tiêu**: Đảm bảo kiểm tra đầy đủ các nhánh và điều kiện trong hàm `TinhChiPhi`.
- **Phần cần kiểm thử**:
  - Quyết định logic cho giảm giá, chiết khấu, và phí vận chuyển.
- **Coverage mục tiêu**:
  - **Decision Coverage**: Đảm bảo tất cả các nhánh (if/else) được kiểm tra.
  - **Condition Coverage**: Kiểm tra giá trị `True` và `False` của mỗi điều kiện.
- **Scenarios cụ thể**:
  - Đơn hàng có sản phẩm được giảm giá và không giảm giá.
  - Khách hàng thường xuyên và không thường xuyên với các mức tổng giá trị đơn hàng khác nhau (dưới, trên, và bằng 1 triệu).

3. **Black Box Test**:

- **Mục tiêu**: Kiểm tra hành vi tổng thể của `TinhChiPhi` đối với các loại đầu vào hợp lệ và không hợp lệ.
- **Phần cần kiểm thử**:
  - Hành vi tổng thể của hàm với các loại đầu vào khác nhau.
- **Đặc điểm test cases**:
  - **Equivalence Partitioning**:
    - Đầu vào hợp lệ: tổng giá trị đơn hàng, trạng thái khách hàng, giá sản phẩm, số lượng, và phần trăm giảm giá.
    - Đầu vào không hợp lệ: giá trị âm, chuỗi văn bản thay cho số, v.v.
  - **Boundary Value Analysis**:
    - Tổng giá trị đơn hàng ngay tại, dưới, và trên ngưỡng 1 triệu.
    - Giảm giá sản phẩm ở mức 0%, 100%, và vượt quá 100%.
- **Scenarios cụ thể**:
  - Đơn hàng trống.
  - Đơn hàng với nhiều sản phẩm có mức giảm giá khác nhau.
  - Thay đổi trạng thái khách hàng và quan sát ảnh hưởng đến tổng chi phí.

2 Bài 2: Tìm dãy con liên tiếp có tổng lớn nhất

2.1 Mục tiêu

Xây dựng hàm tìm dãy con liên tiếp có tổng lớn nhất với hai cách tiếp cận:

- **Code trâu** có độ phức tạp $O(n^2)$ hoặc $O(n^3)$.
- **Code tối ưu** sử dụng Kadane's Algorithm với độ phức tạp $O(n)$.

2.2 Cách tiếp cận chi tiết

1. **Code trâu** $O(n^2)$:

   - **Ý tưởng**: Xét tất cả các dãy con liên tiếp trong dãy và tìm tổng lớn nhất.
   - **Độ phức tạp**: Do xét tất cả các dãy con liên tiếp nên độ phức tạp là $O(n^2)$.

```
def max_subarray_sum_naive(arr):
    n = len(arr)
    max_sum = float('-inf')
    for i in range(n):
        current_sum = 0
        for j in range(i, n):
            current_sum += arr[j]
            max_sum = max(max_sum, current_sum)
    return max_sum
```

2. **Code tối ưu (Kadane's Algorithm)** $O(n)$:

   - **Ý tưởng**: Kadane's Algorithm duy trì `current_sum` cho đến khi dãy con có giá trị nhỏ hơn giá trị tiếp theo thì reset lại `current_sum`.
   - **Độ phức tạp**: Vì chỉ duyệt qua mảng một lần, độ phức tạp là $O(n)$.

```
def max_subarray_sum_kadane(arr):
    max_sum = arr[0]
    current_sum = arr[0]
    for i in range(1, len(arr)):
        current_sum = max(arr[i], current_sum + arr[i])
        max_sum = max(max_sum, current_sum)
    return max_sum
```

2.3 Trình sinh và so sánh test

- **Sinh test case**: Tạo dãy ngẫu nhiên hoặc các trường hợp đặc biệt.

```
import random


def generate_test_case(n):
    return [random.randint(-10000, 10000) for _ in range(n)]


test_case = generate_test_case(100)  # Sinh test có 100 phần tử
```

- **So sánh kết quả giữa code trâu và code tối ưu**: Kiểm tra kết quả của `max_subarray_sum_naive` và `max_subarray_sum_kadane` để đảm bảo tính nhất quán.

```
print("Kết quả code trâu:", max_subarray_sum_naive(test_case))
print("Kết quả code tối ưu:", max_subarray_sum_kadane(test_case))
```

2.4 Các trường hợp đặc biệt của testcase

1. **Dãy toàn số âm**: Kết quả sẽ là phần tử lớn nhất (ví dụ: $[-5, -2, -3, -8]$ trả về -2).
2. **Dãy chỉ có một phần tử**: Tổng là phần tử duy nhất (ví dụ: [3], [0], $[-1]$).
3. **Dãy có giá trị cực đại và cực tiểu xen kẽ**: Kiểm tra khả năng của thuật toán trong việc duy trì dãy con liên tiếp lớn nhất.