ĐẠI HỌC QUỐC GIA TP.HCM
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

Khoa Khoa học máy tính

Môn học Phân Tích Và Thiết Kế Thuật Toán

Bài Tập Kiểm tra tính đúng đắn & hiệu năng của chương trình bằng bộ test

---

Cao Lê Công Thành
MSSV: 23521437
Đặng Quang Vinh
MSSV: 23521786

Mục lục

1 Bài 1: Cho bài toán như sau:

Giả sử bạn đang phát triển một hệ thống quản lý đơn hàng cho một cửa hàng trực tuyến. Hệ thống cần tính toán tổng chi phí cho một đơn hàng dựa trên các yếu tố sau:

1. **Thông tin đơn hàng:**
   - **Danh sách sản phẩm**: Mỗi sản phẩm có giá, số lượng, và có thể có giảm giá.
   - **Phí vận chuyển**: Tính thêm phí nếu tổng giá trị đơn hàng nhỏ hơn 1 triệu.
   - **Loại khách hàng**: Khách hàng thường xuyên có chiết khấu 10% trên tổng giá trị đơn hàng.
2. **Yêu cầu bài toán:**
   - Viết hàm "TinhChiPhi(Order order)" để tính tổng chi phí của đơn hàng sau khi đã áp dụng các quy tắc giảm giá, chiết khấu và vận chuyển.
   - **Order** gồm:
     - Danh sách các sản phẩm: Mỗi sản phẩm gồm giá gốc, số lượng và phần trăm giảm giá sản phẩm đó.
     - `IsRegularCustomer`: Khách hàng thường xuyên hay không.
     - `ShippingFee`: Phí vận chuyển.
3. **Quy tắc:**
   - Có 1 function tính tổng giá thành khi áp dụng giảm giá và 1 function không áp dụng giảm giá.
   - Nếu giá trị đơn hàng trước khi áp dụng giảm giá các sản phẩm $\geq$ 1 triệu, miễn phí vận chuyển.
   - Nếu khách hàng là khách hàng thường xuyên, chiết khấu 10% trên tổng giá trị đơn hàng sau khi áp dụng các giảm giá của sản phẩm.

Yêu cầu:

1. Viết mã giả cho bài toán trên.
2. Dựa vào mã giả của bạn, ta sẽ áp dụng Unit test, White box test, Black box test cho các phần nào của bài toán. Nếu áp dụng White box test/ Black box test thì ta cần sinh các test có đặc điểm như thế nào?

1.1 Mã giả cho bài toán:

Algorithm 1 Tính chi phí đơn hàng

```
1: function TinhChiPhi(Order order)
2:     totalOriginalPrice ← 0
3:     totalDiscountedPrice ← 0
4:     for each product in order.Products do
5:         productOriginalCost ← product.Price × product.Quantity
6:         productDiscountedCost ← productOriginalCost ×(1 − product.Discount/100)
7:         totalOriginalPrice ← totalOriginalPrice + productOriginalCost
8:         totalDiscountedPrice ← totalDiscountedPrice +
   productDiscountedCost
9:     end for
10:    if totalOriginalPrice < 1,000,000 then
11:        shippingCost ← order.ShippingFee
12:    else
13:        shippingCost ← 0
14:    end if
15:    if order.IsRegularCustomer then
16:        totalDiscountedPrice ← totalDiscountedPrice ×0.9
17:    end if
18:    totalCost ← totalDiscountedPrice + shippingCost
19:    return totalCost
20: end function
```

1.2 Kiểm thử

1. **Unit Test:**
   - Kiểm tra hàm `TinhChiPhi` với các đầu vào khác nhau để đảm bảo tính chính xác.
   - Ví dụ: Kiểm tra với một sản phẩm, nhiều sản phẩm, và các trường hợp giảm giá khác nhau.
2. **White Box Test:**
   - Kiểm tra logic bên trong của hàm, đặc biệt là các nhánh điều kiện trong `If` và vòng lặp `For`.
   - Ví dụ: Đưa vào các giá trị biên, như tổng giá trị đơn hàng bằng hoặc nhỏ hơn 1 triệu.
3. **Black Box Test:**
   - Kiểm tra đầu vào và đầu ra mà không xem xét chi tiết bên trong.
   - Ví dụ: Đưa vào các danh sách sản phẩm với thông tin cụ thể và kiểm tra xem tổng chi phí có đúng hay không.

1.3 Đặc điểm của các test case khi áp dụng White Box Test và Black Box Test

1. **White Box Test:**
   - **Đảm bảo bao quát mọi nhánh:** Mỗi nhánh điều kiện trong hàm phải được kiểm thử ít nhất một lần.
   - **Kiểm tra các tình huống biên:** Đưa vào các giá trị đặc biệt, như tổng giá trị đơn hàng bằng 1 triệu, số lượng sản phẩm bằng 0, và khách hàng không phải là khách hàng thường xuyên.
   - **Kiểm tra lỗi:** Đảm bảo rằng hàm xử lý chính xác khi gặp đầu vào không hợp lệ.

2. **Black Box Test:**

   - **Dựa vào đầu vào và đầu ra:** Không quan tâm đến cách thức hoạt động bên trong, chỉ kiểm tra kết quả.
   - **Đầu vào hợp lệ và không hợp lệ:** Thử nghiệm với các giá trị hợp lệ (giá sản phẩm, số lượng) và không hợp lệ (giá âm, số lượng âm).
   - **Kiểm tra tình huống thực tế:** Sử dụng các sản phẩm thực tế, nơi khách hàng thường xuyên, và các sản phẩm khác nhau với giảm giá khác nhau.

2 Bài 2

Cho dãy số nguyên $a_1, a_2, \ldots, a_n$, một dãy con liên tiếp của dãy a là $a_l + a_{l+1} + \cdots + a_r$. Hãy tìm dãy con liên tiếp có trọng số lớn nhất, hay tìm một cặp (l, r) mà $1 \leq l \leq r \leq n$ có $(a_l + a_{l+1} + \cdots + a_r)$ là lớn nhất.

Input:

- Dòng đầu chứa số nguyên dương $n (n \leq 10^5)$.
- Dòng thứ hai chứ n số $a_1, a_2, \ldots, a_n (|a_i| \leq 10^4)$.

Output:

- Ghi ra tổng lớn nhất tìm được.

2.1 Viết code trâu có độ phức tạp O(n^2) hoặc O(n^3)

```
def max_subarray_sum_brute_force(arr):
    n = len(arr)
    max_sum = float('-inf')  # Khoi tao tong lon nhat
    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

# Vi du su dung
n = int(input("Nhap so phan tu n: "))
arr = list(map(int, input("Nhap cac so nguyen cach nhau boi dau cach: ").split()))
result = max_subarray_sum_brute_force(arr)
print(f"Tong lon nhat cua day con lien tiep la: {result}")
```

2.2 Code full (O(n))

```python
def max_subarray_sum_kadane(arr):
    max_sum = float('-inf')
    current_sum = 0

    for num in arr:
        current_sum += num
        max_sum = max(max_sum, current_sum)
        if current_sum < 0:
            current_sum = 0  # Dat lai neu tong hien tai nho hon 0

    return max_sum

# Vi du su dung
n = int(input("Nhap so phan tu n: "))
arr = list(map(int, input("Nhap cac so nguyen cach nhau boi dau cach: ").split()))
result = max_subarray_sum_kadane(arr)
print(f"Tong lon nhat cua day con lien tiep la: {result}")
```

2.3 Viết trình sinh và so test giữa "code trâu" và "code full".

```python
import random
import time

def generate_test_case(n):
    return [random.randint(-10**4, 10**4) for _ in range(n)]

def test_cases(num_tests=10, n=1000):
    for _ in range(num_tests):
        arr = generate_test_case(n)

        # Kiem tra voi code trau
        start_time = time.time()
        result_brute = max_subarray_sum_brute_force(arr)
        end_time = time.time()
        brute_time = end_time - start_time

        # Kiem tra voi code full
        start_time = time.time()
```

```
        result_full = max_subarray_sum_kadane(arr)
        end_time = time.time()
        full_time = end_time - start_time

        assert result_brute == result_full, "Ket qua khong khop!"
        print(f"Test case: {arr}")
        print(f"Ket qua trau: {result_brute} (thoi gian: {brute_time:.6f}s)")
        print(f"Ket qua full: {result_full} (thoi gian: {full_time:.6f}s)")
        print()

# Goi ham kiem tra
test_cases()
```

2.4 Các trường hợp đặc biệt về testcase của bài toán

- Dãy chỉ có số âm: Đây là trường hợp mà tất cả các phần tử trong dãy đều âm. Trong trường hợp này, dãy con có tổng lớn nhất là phần tử âm lớn nhất, vì thêm bất kỳ phần tử nào khác sẽ làm giảm tổng.
  - Ví dụ: [-1, -2, -3] → Kết quả là -1.
- Dãy chỉ có số dương: Khi tất cả các phần tử đều dương, dãy con có tổng lớn nhất sẽ là toàn bộ dãy vì tổng của bất kỳ dãy con nào cũng luôn tăng khi thêm vào các phần tử dương.
  - Ví dụ: [1, 2, 3] → Kết quả là 6.
- Dãy có số 0: Trường hợp dãy có các phần tử bằng 0 là một testcase quan trọng. Khi có 0 trong dãy, tổng có thể không thay đổi khi thêm 0, và cần đảm bảo thuật toán xử lý đúng.
  - Ví dụ: [0, -1, 2, -3, 4] → Kết quả là 4.
- Dãy chỉ có 1 phần tử: Đây là trường hợp đơn giản nhất, nhưng cần kiểm tra để đảm bảo thuật toán không gặp lỗi với dãy có kích thước nhỏ nhất.
  - Ví dụ: [5] → Kết quả là 5.

- Dãy có chiều dài lớn: Khi dãy có số lượng phần tử lớn (ví dụ n=10^5), testcase này kiểm tra tính hiệu quả của thuật toán. Đối với những dãy lớn như vậy, các thuật toán có độ phức tạp cao như O(n^2) sẽ chạy chậm, nên cần sử dụng thuật toán tối ưu như O(n).
  - Ví dụ: Dãy gồm 100000 phần tử ngẫu nhiên.