ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

PHÂN TÍCH VÀ THIẾT KẾ THUẬT TOÁN
CS112.P11.KHTN

BÀI TẬP NHÓM 3

Sinh viên :
Nguyễn Văn Hồng Thái - 23521418
Hoàng Đức Dũng - 23520328

Giảng viên :
Nguyễn Thanh Sơn

Ngày 31 tháng 10 năm 2024

Mục lục

1 Bài toán tính tổng chi phí đơn hàng

1.1 Đề bài

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:

- **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.
- **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` bao 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.
- **Quy tắc**:
  - Có một hàm tính tổng giá thành khi áp dụng giảm giá và một hàm không áp dụng giảm giá.
  - Nếu giá trị đơn hàng trước khi áp dụng giảm giá các sản phẩm lớn hơn hoặc bằng 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.

1.2 Mã giả

```
Function TinhChiPhi(Order order):
    total_cost = 0

    // Tính tổng giá trị đơn hàng trước khi áp dụng giảm giá
    original_total = CalculateOriginalTotal(order.ProductList)

    // Tính tổng giá trị đơn hàng sau khi áp dụng giảm giá
    discounted_total = CalculateDiscountedTotal(order.ProductList)

    // Kiểm tra xem có cần miễn phí vận chuyển không
    if original_total >= 1000000:
        shipping_fee = 0
```

```
    else:
        shipping_fee = order.ShippingFee

    // Tính tổng chi phí sau khi áp dụng giảm giá và phí vận chuyển
    total_cost = discounted_total + shipping_fee

    // Nếu khách hàng là khách hàng thường xuyên, áp dụng chiết khấu 10%
    if order.IsRegularCustomer:
        total_cost = total_cost * 0.9

    return total_cost

// Hàm tính tổng giá trị không áp dụng giảm giá
Function CalculateOriginalTotal(ProductList):
    original_total = 0
    For each product in ProductList:
        original_total += product.price * product.quantity
    return original_total

// Hàm tính tổng giá trị áp dụng giảm giá
Function CalculateDiscountedTotal(ProductList):
    discounted_total = 0
    For each product in ProductList:
        discounted_price = product.price * (1 - product.discount)
        discounted_total += discounted_price * product.quantity
    return discounted_total
```

1.3 Phân tích kiểm thử

- **Unit Test**:
  - **Các hàm cần kiểm thử riêng lẻ**:
    - `CalculateOriginalTotal(ProductList)`: Kiểm thử hàm này với danh sách sản phẩm có nhiều trường hợp khác nhau, bao gồm giá trị và số lượng sản phẩm đa dạng, để đảm bảo tính đúng tổng giá trị trước khi áp dụng giảm giá.
    - `CalculateDiscountedTotal(ProductList)`: Kiểm thử với các sản phẩm có phần trăm giảm giá khác nhau, nhằm xác minh tính toán tổng giá trị sau khi áp dụng giảm giá.
    - `TinhChiPhi(Order order)`: Kiểm thử hàm chính với các trường hợp khác nhau về khách hàng, phí vận chuyển và tổng giá trị đơn hàng để đảm bảo kết quả cuối cùng chính xác.
- **White Box Test**:
  - **Đặc điểm các test case**:
    - Tạo các tình huống để kiểm tra tất cả các nhánh của điều kiện:

        - Đơn hàng lớn hơn hoặc bằng 1 triệu và nhỏ hơn 1 triệu để kiểm tra miễn phí vận chuyển.
        - Khách hàng thường xuyên và không thường xuyên để kiểm tra chiết khấu 10%.
    - Đảm bảo các đường dẫn qua các câu lệnh điều kiện `if` trong hàm `TinhChiPhi` đều được kiểm tra.

- **Black Box Test**:
  - **Đặc điểm các test case**:
    - Kiểm tra hàm `TinhChiPhi` với các giá trị đầu vào khác nhau cho `ProductList`, `IsRegularCustomer`, và `ShippingFee`.
    - Các trường hợp:
        - Không có sản phẩm trong đơn hàng.
        - Một sản phẩm không có giảm giá.
        - Nhiều sản phẩm với các mức giảm giá khác nhau.
        - Khách hàng thường xuyên và không thường xuyên.
        - Đơn hàng lớn hơn hoặc bằng 1 triệu và nhỏ hơn 1 triệu.

2 Dãy con có tổng lớn nhất

2.1 Đề bài

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 tổng lớn nhất, tức là tìm một cặp (l, r) với $1 \leq l \leq r \leq n$ sao cho $a_l + a_{l+1} + \cdots + a_r$ đạt giá trị lớn nhất.

- **Input**:
  - Dòng đầu chứa số nguyên dương n với $n \leq 10^5$.
  - Dòng thứ hai chứa n số $a_1, a_2, \ldots, a_n$ với $|a_i| \leq 10^4$.
- **Output**:
  - Ghi ra tổng lớn nhất tìm được.

2.2 Yêu cầu

1. Viết code "trâu" có độ phức tạp $O(n^2)$ hoặc $O(n^3)$.
2. Viết code tối ưu có độ phức tạp $O(n)$ hoặc $O(n \cdot \log(n))$.
3. Viết trình sinh test case và kiểm tra kết quả giữa “code trâu” và “code tối ưu”.
4. Trình bày các trường hợp đặc biệt của bài toán trong báo cáo.

2.3 Mã giả cho giải pháp trâu (Độ phức tạp $O(n^2)$ hoặc $O(n^3)$)

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

int maxSubArrayBruteForce(const vector<int>& arr) {
    int n = arr.size();
    int maxSum = arr[0];

    for (int i = 0; i < n; i++) {
        int currentSum = 0;
        for (int j = i; j < n; j++) {
            currentSum += arr[j];
            maxSum = max(maxSum, currentSum);
        }
    }

    return maxSum;
}

int main() {
    int n;
    cin >> n;
    vector<int> arr(n);

    for (int i = 0; i < n; i++) {
        cin >> arr[i];
    }

    cout << maxSubArrayBruteForce(arr) << endl;
    return 0;
}
```

2.4 Mã giả cho giải pháp tối ưu (Độ phức tạp $O(n)$ - Thuật toán Kadane)

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

int maxSubArrayOptimized(const vector<int>& arr) {
    int maxSum = arr[0];
    int currentSum = arr[0];
```

```
    for (int i = 1; i < arr.size(); i++) {
        currentSum = max(arr[i], currentSum + arr[i]);
        maxSum = max(maxSum, currentSum);
    }

    return maxSum;
}

int main() {
    int n;
    cin >> n;
    vector<int> arr(n);

    for (int i = 0; i < n; i++) {
        cin >> arr[i];
    }

    cout << maxSubArrayOptimized(arr) << endl;
    return 0;
}
```

2.5 Trình sinh test case và so sánh kết quả giữa hai giải pháp

```
#include <iostream>
#include <vector>
#include <cstdlib>
#include <ctime>
#include <algorithm>
using namespace std;

int maxSubArrayBruteForce(const vector<int>& arr);
int maxSubArrayOptimized(const vector<int>& arr);

vector<int> generateTestCase(int n) {
    vector<int> arr(n);
    for (int i = 0; i < n; i++) {
        arr[i] = rand() % 20001 - 10000; // sinh số ngẫu nhiên trong khoảng [-10000,
    }
    return arr;
}

int main() {
    srand(time(0));
    int numTests = 100; // số lượng test cần sinh

    for (int t = 0; t < numTests; t++) {
        int n = rand() % 100 + 1; // độ dài dãy ngẫu nhiên từ 1 đến 100
```

```
        vector<int> arr = generateTestCase(n);

        int bruteForceResult = maxSubArrayBruteForce(arr);
        int optimizedResult = maxSubArrayOptimized(arr);

        if (bruteForceResult != optimizedResult) {
            cout << "Test failed on test case " << t + 1 << endl;
            cout << "Brute Force Result: " << bruteForceResult << ", Optimized Result
            return 1;
        }
    }

    cout << "All tests passed!" << endl;
    return 0;
}

// Hàm tính tổng lớn nhất bằng phương pháp "trâu" O(n^2)
int maxSubArrayBruteForce(const vector<int>& arr) {
    int n = arr.size();
    int maxSum = arr[0];

    for (int i = 0; i < n; i++) {
        int currentSum = 0;
        for (int j = i; j < n; j++) {
            currentSum += arr[j];
            maxSum = max(maxSum, currentSum);
        }
    }

    return maxSum;
}

// Hàm tính tổng lớn nhất bằng phương pháp tối ưu O(n)
int maxSubArrayOptimized(const vector<int>& arr) {
    int maxSum = arr[0];
    int currentSum = arr[0];

    for (int i = 1; i < arr.size(); i++) {
        currentSum = max(arr[i], currentSum + arr[i]);
        maxSum = max(maxSum, currentSum);
    }

    return maxSum;
}
```

2.6 Các trường hợp đặc biệt cần xem xét

- **Trường hợp tất cả phần tử âm**: Nếu tất cả các phần tử đều âm, thì tổng lớn nhất sẽ là phần tử có giá trị lớn nhất trong mảng. Ví dụ: $[-3, -2, -5, -7]$, tổng lớn nhất là -2.
- **Trường hợp tất cả phần tử dương**: Nếu tất cả phần tử đều dương, thì tổng lớn nhất sẽ là tổng của toàn bộ dãy. Ví dụ: $[1, 2, 3, 4, 5]$, tổng lớn nhất là 15.
- **Trường hợp có cả số âm và số dương**: Dãy có cả số âm và số dương, và tổng lớn nhất nằm ở một dãy con liên tiếp không bao gồm các phần tử âm làm giảm giá trị tổng. Ví dụ: $[-2, 1, -3, 4, -1, 2, 1, -5, 4]$, tổng lớn nhất là 6 với dãy con $[4, -1, 2, 1]$.
- **Dãy có một phần tử duy nhất**: Nếu dãy chỉ có một phần tử, thì tổng lớn nhất là chính phần tử đó. Ví dụ: $[5]$, tổng lớn nhất là 5.
- **Dãy có giá trị 0**: Dãy có thể chứa các giá trị 0, không ảnh hưởng đến tổng nhưng cần đảm bảo không ảnh hưởng đến dãy con có tổng lớn nhất. Ví dụ: $[-1, 0, 2, -3, 5, -1]$, tổng lớn nhất là 5 với dãy con $[5]$.

Hết.