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

BÀI TẬP NHÓM 3
BỞI NHÓM 12

Giáo viên hướng dẫn: ***Nguyễn Thanh Sơn***

Nhóm 12:

- ***Hoàng Minh Thái*** **- 23521414**
- ***Nguyễn Trọng Tất Thành*** **- 23521455**

Mục lục

GIẢI BÀI TẬP BỞI NHÓM 12

Bài toán 1

Kaiser, một cảnh sát kỳ cựu, đang phải đối mặt với một thử thách khắc nghiệt để cứu vợ mình, Kayra, khỏi tay tên tội phạm thời gian. Để tìm ra nơi vợ anh ấy bị giam giữ, Kaiser phải đi qua một số thành phố được nối với nhau bằng các con đường một chiều. Tuy nhiên, một số con đường có trọng số âm (do tên tội phạm tạo ra bẫy thời gian).

Bài toán yêu cầu xác định xem có chu trình âm nào trong đồ thị hay không, nếu có thì in ra chu trình đó.

Ý tưởng và phương pháp thiết kế thuật toán

Thuật toán Bellman-Ford sẽ được sử dụng để phát hiện chu trình âm trong đồ thị có trọng số âm. Các bước thực hiện bao gồm:

- Bắt đầu từ một đỉnh nguồn và thực hiện $N-1$ lần lặp qua tất cả các cạnh của đồ thị để cập nhật khoảng cách tối thiểu.
- Sau $N-1$ lần lặp, nếu có sự thay đổi trong lần lặp thứ N, thì chắc chắn đồ thị chứa chu trình âm.
- Lần theo các đỉnh để tìm ra chu trình âm.

Độ phức tạp: Thuật toán Bellman-Ford có độ phức tạp thời gian là $O(N \times M)$, với N là số đỉnh và M là số cạnh trong đồ thị.

Mã giả của thuật toán

```
1. Khởi tạo khoảng cách của tất cả các đỉnh là vô cùng lớn, ngoại trừ đỉnh nguồn là 0
2. Lặp M lần, mỗi lần:
   a. Duyệt tất cả các cạnh a -> b với trọng số c.
   b. Nếu khoảng cách[a] + c < khoảng cách[b], cập nhật khoảng cách[b].
3. Sau M lần lặp, kiểm tra xem có thay đổi gì không. Nếu có, ta phát hiện chu trình â
4. Nếu có chu trình âm, lần theo các đỉnh để tìm chu trình và in ra chu trình.
5. Nếu không có chu trình âm, in ra "NO".
```

Cài đặt thuật toán

Python code:

```
def find_negative_cycle(N, M, edges):
    # Khởi tạo khoảng cách từ nguồn tới tất cả các đỉnh là vô cùng lớn
    INF = float('inf')
    distance = [INF] * (N + 1)
    parent = [-1] * (N + 1)
    distance[1] = 0  # Giả sử đỉnh nguồn là 1

    # Bellman-Ford: Lặp qua tất cả các cạnh M lần
    for i in range(N - 1):
        for u, v, w in edges:
            if distance[u] != INF and distance[u] + w < distance[v]:
                distance[v] = distance[u] + w
                parent[v] = u

    # Kiểm tra chu trình âm: Lặp qua một lần nữa để tìm sự thay đổi
    for u, v, w in edges:
        if distance[u] != INF and distance[u] + w < distance[v]:
            # Phát hiện chu trình âm, tìm chu trình
            cycle = []
            visited = [False] * (N + 1)
            x = v
            # Lấy chu trình bằng cách lần theo các đỉnh trong chu trình
            for _ in range(N):
                x = parent[x]
            cycle_start = x
            cycle.append(cycle_start)
            current = parent[cycle_start]
            while current != cycle_start:
                cycle.append(current)
                current = parent[current]
            cycle.append(cycle_start)
            cycle.reverse()

            # In ra chu trình âm
            print("YES")
            print(" ".join(map(str, cycle)))
            return

    # Nếu không có chu trình âm
    print("NO")

# Đọc đầu vào
N, M = map(int, input().split())
edges = []
```

```
for _ in range(M):
    a, b, c = map(int, input().split())
    edges.append((a, b, c))

# Gọi hàm kiểm tra chu trình âm
find_negative_cycle(N, M, edges)
```

Giải thích mã

1. **Khởi tạo:** Mảng ‘distance‘ lưu khoảng cách từ đỉnh nguồn đến các đỉnh còn lại. Mảng ‘parent‘ lưu đỉnh cha của mỗi đỉnh để có thể truy vết chu trình nếu phát hiện. 2. **Thuật toán Bellman-Ford:** Lặp qua tất cả các cạnh M lần. Nếu tìm thấy một khoảng cách nhỏ hơn giá trị hiện tại, cập nhật khoảng cách và lưu lại đỉnh cha. 3. **Kiểm tra chu trình âm:** Sau khi lặp $N-1$ lần, nếu có thay đổi trong lần lặp thứ N, ta bắt đầu lần theo các đỉnh cha để tìm chu trình âm và in ra. 4. **Đầu ra:** Nếu phát hiện chu trình âm, in "YES"và chu trình. Nếu không có chu trình âm, in "NO".

Ví dụ

Input:

$$4\ 5$$
$$1\ 2\ 1$$
$$2\ 4\ 1$$
$$3\ 1\ 1$$
$$4\ 1\ -3$$
$$4\ 3\ -2$$

Output:

$$YES$$
$$1\ 2\ 4\ 1$$

Bài toán 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 tổng $a_l + a_{l+1} + \cdots + a_r$ là lớn nhất.

Yêu cầu:

- Viết code trâu có độ phức tạp $O(n^2)$ hoặc $O(n^3)$.
- Viết code full có độ phức tạp $O(n)$ hoặc $O(n \cdot \log n)$.
- Viết trình sinh và so test giữa "code trâu"và "code full".
- Trình bày các trường hợp đặc biệt về testcase của bài toán vào báo cáo.

Mã giả

Thuật toán brute force (code trâu)

Thuật toán brute force sẽ thử tất cả các dãy con liên tiếp và tính tổng của chúng, từ đó tìm ra dãy con có tổng lớn nhất.

```
Function FindMaxSumNaive(arr, n):
    maxSum = -infinity
    For l = 1 to n:
        For r = l to n:
            sum = 0
            For i = l to r:
                sum += arr[i]
            maxSum = max(maxSum, sum)
    Return maxSum
```

Mô tả:

- Với mỗi cặp chỉ số (l, r), ta tính tổng của dãy con từ chỉ số l đến r bằng cách cộng các phần tử trong dãy con.
- Lặp qua tất cả các dãy con có thể có từ dãy a.
- Tính tổng và tìm ra giá trị lớn nhất.
- Độ phức tạp thời gian của thuật toán này là $O(n^3)$.

Thuật toán tối ưu (code full)

Thuật toán tối ưu sử dụng phương pháp Kadane's Algorithm, cho phép tính tổng dãy con liên tiếp có tổng lớn nhất trong thời gian $O(n)$.

```
Function FindMaxSumOptimized(arr, n):
    maxSoFar = arr[0]
    maxEndingHere = arr[0]

    For i = 1 to n-1:
        maxEndingHere = max(arr[i], maxEndingHere + arr[i])
        maxSoFar = max(maxSoFar, maxEndingHere)

    Return maxSoFar
```

Mô tả:

- Khởi tạo hai biến: `maxEndingHere` là tổng của dãy con kết thúc tại vị trí hiện tại và `maxSoFar` là tổng lớn nhất tìm được.
- Với mỗi phần tử $arr[i]$, ta quyết định xem có nên tiếp tục dãy con hiện tại (tính `maxEndingHere` + $arr[i]$) hay bắt đầu một dãy con mới từ $arr[i]$.
- Cập nhật `maxSoFar` để giữ tổng dãy con lớn nhất cho đến thời điểm hiện tại.
- Độ phức tạp thời gian của thuật toán này là $O(n)$.

Trình sinh và so sánh giữa "code trâu"và "code full

Trình sinh test

Để kiểm tra độ chính xác giữa thuật toán "code trâu"và "code full", ta cần tạo các dãy số ngẫu nhiên với chiều dài n và các giá trị a_i nằm trong phạm vi từ -10^4 đến 10^4. Sau đó so sánh kết quả trả về từ hai thuật toán.

```
Function GenerateRandomTest(n):
    arr = []
    For i = 1 to n:
        arr[i] = RandomIntegerInRange(-10000, 10000)
    Return arr
```

So sánh kết quả giữa "code trâu"và "code full

```
Function CompareResults(arr, n):
    naiveResult = FindMaxSumNaive(arr, n)
    optimizedResult = FindMaxSumOptimized(arr, n)

    If naiveResult == optimizedResult:
        Print "Test passed"
    Else:
        Print "Test failed"
```

Trường hợp đặc biệt

Trong quá trình kiểm thử, có một số trường hợp đặc biệt cần lưu ý:

- **Dãy số có tất cả phần tử âm:** Trường hợp này sẽ kiểm tra khả năng của thuật toán trong việc xử lý các dãy con có tổng âm lớn nhất.
- **Dãy số có một phần tử duy nhất:** Trường hợp này sẽ kiểm tra xem thuật toán có xử lý đúng khi dãy số chỉ có một phần tử.
- **Dãy số có tổng dãy con lớn nhất là 0:** Trường hợp này cần kiểm tra khả năng của thuật toán trong việc phát hiện tổng lớn nhất là 0.
- **Dãy số có tổng dãy con lớn nhất là tổng toàn bộ dãy:** Trường hợp này sẽ kiểm tra khả năng của thuật toán khi toàn bộ dãy là dãy con lớn nhất.