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 15

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

Giảng viên :
Nguyễn Thanh Sơn

Ngày 19 tháng 12 năm 2024

Mục lục

Bài tập 1

1. Thuật toán Greedy

Thuật toán Greedy chọn đường đi dựa trên **khoảng cách Euclide nhỏ nhất đến Novgorod** tại mỗi bước.

Quy trình thực hiện:

1. Bắt đầu từ **London (2114)**.
2. Tại mỗi bước, chọn node có giá trị heuristic (khoảng cách Euclide đến Novgorod) nhỏ nhất.
3. Lặp lại cho đến khi đến **Novgorod**.

Lời giải chi tiết:

- **Bước 1:** Từ London (2114), các thành phố có thể đi:
  - Bergen (1467)
  - Amsterdam (1777)
  - Hamburg (1422)

  → Chọn **Hamburg** vì 1422 là nhỏ nhất.
- **Bước 2:** Từ Hamburg (1422), các thành phố có thể đi:
  - Falsterbo (1166)
  - Lubeck (1365)

  → Chọn **Falsterbo** vì 1166 là nhỏ nhất.
- **Bước 3:** Từ Falsterbo (1166), các thành phố có thể đi:
  - Copenhagen (1167)
  - Danzig (901)

  → Chọn **Danzig** vì 901 là nhỏ nhất.
- **Bước 4:** Từ Danzig (901), các thành phố có thể đi:
  - Tallinn (387)
  - Riga (459)

  → Chọn **Tallinn** vì 387 là nhỏ nhất.
- **Bước 5:** Từ Tallinn (387), các thành phố có thể đi:
  - Novgorod (0)

  → Chọn **Novgorod**.

Tuyến đường Greedy: **London → Hamburg → Falsterbo → Danzig → Tallinn → Novgorod**

Tổng chi phí đường đi (di chuyển thực tế):

$$801 + 324 + 498 + 590 + 474 = \mathbf{2687}$$

—

2. Thuật toán UCS (Uniform Cost Search)

Thuật toán UCS chọn đường đi dựa trên **chi phí thực tế nhỏ nhất từ điểm bắt đầu**.

Quy trình thực hiện:

1. Bắt đầu từ **London**.
2. Tại mỗi bước, mở rộng tất cả các node kề và thêm vào hàng đợi ưu tiên (priority queue) dựa trên chi phí thực tế.
3. Lặp lại cho đến khi đến **Novgorod**.

Lời giải chi tiết:

- **Bước 1:** Từ London (0), các thành phố có thể đi:
  - Bergen (1554)
  - Amsterdam (395)
  - Hamburg (801)

  → Chọn **Amsterdam** vì 395 là nhỏ nhất.
- **Bước 2:** Từ Amsterdam (395), các thành phố có thể đi:
  - Bergen (395 + 1463 = 1858)
  - Copenhagen (395 + 953 = 1348)
  - Hamburg (395 + 411 = 806)

  → Chọn **Hamburg** vì 806 là nhỏ nhất.
- **Bước 3:** Từ Hamburg (806), các thành phố có thể đi:
  - Falsterbo (806 + 324 = 1130)
  - Lubeck (806 + 64 = 870)

  → Chọn **Lubeck** vì 870 là nhỏ nhất.
- **Bước 4:** Từ Lubeck (870), các thành phố có thể đi:
  - Danzig (870 + 262 = 1132)

  - Visby (870 + 738 = 1608)

  → Chọn **Danzig** vì 1132 là nhỏ nhất.

- **Bước 5:** Từ Danzig (1132), các thành phố có thể đi:
  - Tallinn (1132 + 590 = 1722)
  - Riga (1132 + 606 = 1738)

  → Chọn **Tallinn** vì 1722 là nhỏ nhất.

- **Bước 6:** Từ Tallinn (1722), các thành phố có thể đi:
  - Novgorod (1722 + 474 = 2196)

  → Chọn **Novgorod**.

Tuyến đường UCS: London → Amsterdam → Hamburg → Lubeck → Danzig → Tallinn → Novgorod

Tổng chi phí đường đi (di chuyển thực tế):

$$395 + 411 + 64 + 262 + 590 + 474 = \mathbf{2196}$$

—

3. So sánh Greedy và UCS

- **Greedy:**
  - Tuyến đường: London → Hamburg → Falsterbo → Danzig → Tallinn → Novgorod
  - Tổng chi phí: **2687**
  - Không đảm bảo tối ưu do chỉ xét giá trị heuristic.
- **UCS:**
  - Tuyến đường: London → Amsterdam → Hamburg → Lubeck → Danzig → Tallinn → Novgorod
  - Tổng chi phí: **2196**
  - Đảm bảo tối ưu do xét toàn bộ chi phí thực tế.

—

Kết luận

- Đường đi của UCS là **tối ưu hơn** so với Greedy.
- Greedy không đảm bảo tối ưu vì chỉ dựa vào heuristic, trong khi UCS luôn đảm bảo tối ưu toàn cục.

Bài tập 2

Bài toán

Kaiser là một cảnh sát kỳ cựu. Năm 2200, anh ấy đã có cuộc sống hạnh phúc với những thành tựu và chiến công đáng kể mà anh ấy gặt hái được và có một gia đình với 2 con, 1 vợ vô cùng đầm ấm. Tuy nhiên, vào một ngày không nắng, vợ anh ấy là Kayra đã bị một tên tội phạm thời gian bắt cóc và giam giữ.

Vì là một người vô cùng yêu thương vợ nên Kaiser ngày đêm cố gắng lần theo dấu vết mà tên tội phạm để lại và đã xác định được vợ anh ấy đang bị giam ở đâu đó trong N thành phố, được đánh số từ 1 đến N, nối với nhau bởi M con đường một chiều. Tuy nhiên, vì tên bắt cóc vợ anh là một tên tội phạm thời gian khét tiếng nên một số con đường nối giữa các thành phố bị hắn đặt bẫy thời gian, dẫn đến độ dài của các con đường có thể là số âm.

Hãy giúp Kaiser xác định xem có chu trình âm trong thành phố hay không để anh ấy có thể tránh được nguy hiểm và tiếp tục đi giải cứu vợ.

Input

- Dòng đầu tiên chứa hai số nguyên N và M lần lượt là số thành phố và số con đường giữa các thành phố.
- M dòng tiếp theo: mỗi dòng gồm ba số nguyên a, b, c thể hiện rằng có một con đường một chiều nối từ thành phố a đến thành phố b với độ dài là c.

Output

- Nếu xuất hiện chu trình âm, in ra `YES` và in chu trình âm đó theo đúng thứ tự.
- Nếu không có chu trình âm, in ra `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
```

Ý tưởng thuật toán

Để phát hiện chu trình âm, sử dụng thuật toán Bellman-Ford với các bước sau:

[label=0.]Khởi tạo khoảng cách từ đỉnh nguồn (giả sử là đỉnh 1) đến tất cả các đỉnh khác là $+\infty$, trừ đỉnh nguồn với khoảng cách 0. Thực hiện cập nhật $N-1$ lần tất cả các cạnh để tìm khoảng cách ngắn nhất từ đỉnh nguồn đến các đỉnh khác:

$$\text{Nếu } \text{dist}[a] + c < \text{dist}[b], \text{ thì cập nhật } \text{dist}[b] = \text{dist}[a] + c.$$

Sau khi hoàn thành $N-1$ lần cập nhật, kiểm tra tất cả các cạnh để phát hiện chu trình âm:

$$\text{Nếu } \text{dist}[a] + c < \text{dist}[b], \text{ thì tồn tại chu trình âm.}$$

Sử dụng mảng `parent` để truy ngược lại chu trình âm.

Mã giả

```
function BellmanFord(N, M, edges):
    dist = [INF] * (N + 1)
    parent = [-1] * (N + 1)
    dist[1] = 0

    for i from 1 to N-1:
        for each edge (a, b, c) in edges:
            if dist[a] + c < dist[b]:
                dist[b] = dist[a] + c
                parent[b] = a

    for each edge (a, b, c) in edges:
        if dist[a] + c < dist[b]:
            x = b
            for i from 1 to N:
                x = parent[x]

            cycle = []
            start = x
            while True:
                cycle.append(x)
                x = parent[x]
                if x == start:
                    break
            cycle.reverse()
            return "YES", cycle

    return "NO"
```

Cài đặt bằng Python

```
from math import inf
```

```
def find_negative_cycle(n, edges):
    dist = [inf] * (n + 1)
    parent = [-1] * (n + 1)
    dist[1] = 0

    for _ in range(n - 1):
        for a, b, c in edges:
            if dist[a] != inf and dist[a] + c < dist[b]:
                dist[b] = dist[a] + c
                parent[b] = a

    for a, b, c in edges:
        if dist[a] != inf and dist[a] + c < dist[b]:
            cycle = []
            x = b
            for _ in range(n):
                x = parent[x]

            cycle_start = x
            while True:
                cycle.append(x)
                x = parent[x]
                if x == cycle_start and len(cycle) > 1:
                    break
            cycle.reverse()
            return "YES", cycle

    return "NO", None

if __name__ == "__main__":
    n, m = map(int, input().split())
    edges = []
    for _ in range(m):
        a, b, c = map(int, input().split())
        edges.append((a, b, c))

    result, cycle = find_negative_cycle(n, edges)
    if result == "YES":
        print(result)
        print(" ".join(map(str, cycle)))
    else:
        print(result)
```

Độ phức tạp

- **Thời gian:** $O(N \cdot M)$, với N là số đỉnh và M là số cạnh.

- **Không gian:** $O(N)$ để lưu mảng `dist` và `parent`.