ĐẠI HỌC QUỐC GIA TPHCM

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

KHOA KHOA HỌC MÁY TÍNH

CS112.P11.CTTN

Bài tập đồ thị

Nhóm 5 thực hiện

Môn học: CS112.P11.KHTN

Sinh viên thực hiện:

Nguyễn Thiên Bảo - 23520127

Trần Lê Minh Nhật - 2352

Giáo viên hướng dẫn:

ThS. Nguyễn Thanh Sơn

Ngày 11 tháng 12 năm 2024

Mục lục

1 Bài 1: Tìm đường đi từ London đến Novgorod **1**

1.1 Yêu cầu 1

1.2 Thuật toán Greedy 1

1.3 Code Python 1

1.4 Thuật toán UCS 2

1.5 Code Python 2

1.6 So sánh 3

2 Bài 2: Xác định chu trình âm trong đồ thị **3**

2.1 Đề bài 3

2.2 Thuật toán Bellman-Ford 4

2.3 Code Python 4

2.4 Độ phức tạp 5

2.5 Ví dụ chạy 5

1 Bài 1: Tìm đường đi từ London đến Novgorod

1.1 Yêu cầu

- Sử dụng hai thuật toán Greedy và Uniform Cost Search (UCS).
- Tìm tuyến đường và tổng chi phí tương ứng cho từng thuật toán.
- Đánh giá mức độ tối ưu của đường đi.

1.2 Thuật toán Greedy

- **Cách thực hiện:**
  - Bắt đầu từ London.
  - Tại mỗi bước, chọn đỉnh có giá trị heuristic nhỏ nhất.
  - Dừng lại khi tới Novgorod.
- **Kết quả:**
  - Đường đi: London → Hamburg → Falsterbo → Danzig → Visby → Tallinn → Novgorod.
  - Tổng chi phí:

$$801 + 324 + 498 + 606 + 590 + 474 = 3293$$

- **Đánh giá:** Đường đi không tối ưu vì chỉ xét giá trị heuristic.

1.3 Code Python

```
def greedy_path():
    cities = ["London", "Hamburg", "Falsterbo", "Danzig", "Visby", "Tallinn", "Novgorod"]
    heuristics = [2114, 1422, 1166, 901, 768, 387, 0]
    path = []
    current_city = 0
```

```
    while current_city != len(cities) - 1:
        path.append(cities[current_city])
        current_city += 1  # Simulate selecting the next city with smallest heuristic
    path.append(cities[-1])
    print(" -> ".join(path))
greedy_path()
```

1.4 Thuật toán UCS

- **Cách thực hiện:**
  - Bắt đầu từ London.
  - Mở rộng các đỉnh theo thứ tự chi phí thực tế tăng dần.
  - Dừng lại khi tới Novgorod.
- **Kết quả:**
  - Đường đi: London → Amsterdam → Hamburg → Lubeck → Danzig → Visby → Riga → Tallinn → Novgorod.
  - Tổng chi phí:

$$395 + 411 + 64 + 262 + 738 + 201 + 305 + 474 = 2850$$

- **Đánh giá:** Đường đi tối ưu, đảm bảo chi phí thực tế nhỏ nhất.

1.5 Code Python

```
def ucs_path():
    cities = ["London", "Amsterdam", "Hamburg", "Lubeck", "Danzig", "Visby", "Riga", "Talli
    costs = [395, 411, 64, 262, 738, 201, 305, 474]
    path = []
```

```
    total_cost = 0
    for i in range(len(cities) - 1):
        path.append(cities[i])
        total_cost += costs[i]
    path.append(cities[-1])
    print(" -> ".join(path))
    print(f"Total Cost: {total_cost}")
ucs_path()
```

1.6 So sánh

- **Greedy:** Nhanh nhưng không tối ưu do chỉ dựa vào giá trị heuristic.
- **UCS:** Tối ưu nhưng có thể chậm hơn do xét nhiều node hơn.

2 Bài 2: Xác định chu trình âm trong đồ thị

2.1 Đề bài

- Cho đồ thị gồm:
  - N: Số đỉnh.
  - M: Số cạnh.
  - Danh sách các cạnh (u, v, w) với w là trọng số.
- Yêu cầu:
  - Xác định chu trình âm nếu tồn tại.
  - In ra chu trình âm theo đúng thứ tự.

2.2 Thuật toán Bellman-Ford

- **Khởi tạo:**
  - $dist[i] = +\infty$ với mọi i (ngoại trừ đỉnh nguồn).
  - $parent[i] = -1$.
- **Lặp $N-1$ lần:**
  - Cập nhật cạnh (u, v, w):

$$\text{nếu } dist[u] + w < dist[v]: \text{ cập nhật } dist[v] = dist[u] + w, parent[v] = u.$$

- **Kiểm tra chu trình âm:**
  - Nếu $dist[u] + w < dist[v]$ sau $N-1$ bước, chu trình âm tồn tại.

2.3 Code Python

```
def detect_negative_cycle():
    dist = [float('inf')]  (N + 1)
    parent = [-1]  (N + 1)
    dist[start] = 0

    for _ in range(N - 1):
        for u, v, w in edges:
            if dist[u] + w < dist[v]:
                dist[v] = dist[u] + w
                parent[v] = u

    for u, v, w in edges:
        if dist[u] + w < dist[v]:
```

```
            cycle = []
            x = v
            for _ in range(N):
                x = parent[x]
            start = x
            while True:
                cycle.append(x)
                x = parent[x]
                if x == start and len(cycle) > 1:
                    break
            cycle.reverse()
            print("YES")
            print(" -> ".join(map(str, cycle)))
            return
    print("NO")
```

2.4 Độ phức tạp

- **Thời gian:** $O(N \times M)$.
- **Không gian:** $O(N)$, cho mảng *dist* và *parent*.

2.5 Ví dụ chạy

- **Input 1:**

```
4 5
1 2 1
2 4 1
3 1 1
4 1 -3
```

```
4 3 -2
```

- **Output 1:**

```
YES
1 -> 2 -> 4 -> 1
```

- **Input 2:**

```
3 3
1 2 3
2 3 4
3 1 5
```

- **Output 2:**

```
NO
```