Trường Đại học Khoa học Tự Nhiên
ĐHQG.TPHCM

CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM
Độc lập - Tự do - Hạnh phúc

TPHCM, ngày 2 tháng 12 năm 2024

BÁO CÁO THỰC HÀNH – NHẬP MÔN TRÍ TUỆ NHÂN TẠO – LẦN 5

Họ và tên: Nguyễn Thanh Kiên.
MSSV: 22110092.

Nhắc lại kiến thức

1. **Bài toán Travelling salesman problem (TSP).**

Cho trước n thành phố và các khoảng cách d_{ij} giữa mỗi cặp thành phố, tìm tour ngắn nhất sao cho mỗi thành phố được viếng thăm chỉ một lần.

Bài toán người du hành (TSP) là một trong những vấn đề nổi bật và khó khăn nhất về lý thuyết đồ thị, được giới thiệu từ thế kỷ XVII bởi Hamilton và Kirkman. Đây là bài toán NP-hard với độ phức tạp tăng theo hàm số mũ, lần đầu được giải trên máy tính vào năm 1954 (49 đỉnh) và đến năm 2004 đạt tới 24.978 đỉnh, dự kiến sẽ còn tăng nữa.

TSP yêu cầu tìm hành trình đi qua tất cả các thành phố đúng 1 lần và quay lại điểm xuất phát sao cho tổng quãng đường là nhỏ nhất. Bài toán được biểu diễn dưới dạng đồ thị có trọng số, trong đó mỗi cặp đỉnh được nối với nhau bằng một cạnh.

Ngày nay, nhiều phương pháp đã được phát triển để giải quyết TSP. Trong bài báo cáo này, chúng ta sẽ tìm hiểu cách giải sử dụng thuật toán A*, kết hợp với cây khung nhỏ nhất (MST) và thuật toán heuristic chèn gắn gần nhất (Nearest Insertion).

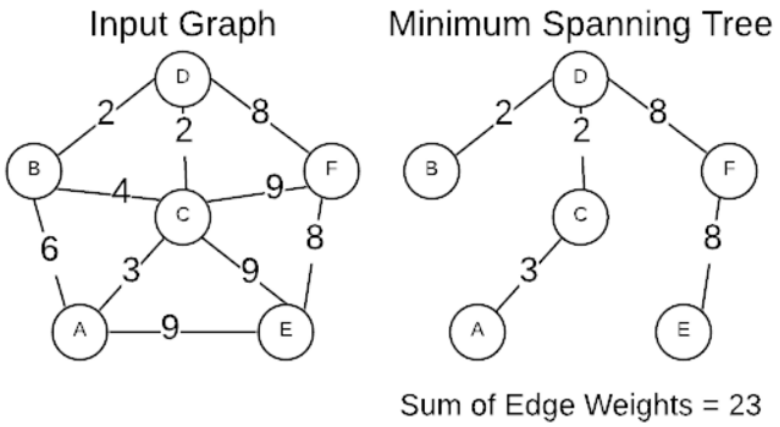
Một lời giải trực quan cho bài toán TSP

2. Cây khung nhỏ nhất – Minimum Spanning Tree – MST

Trong lý thuyết về đồ thị, cây khung nhỏ nhất của đồ thị có trọng số G = [V, E] là một tập hợp các cạnh của đồ thị sao cho:

- Tập hợp các cạnh này không chứa chu trình và liên thông, hay nói cách khác là từ một đỉnh bất kỳ có thể đi tới các đỉnh khác mà chỉ dùng các cạnh trên tập hợp đỉnh đó.
- Tổng trọng số của các cạnh trong tập hợp này là nhỏ nhất.

Minh họa cây khung và MST của nó với đồ thị vô hướng có trọng số



3. Thuật toán Prim

Trong khoa học máy tính, **thuật toán Prim** là một thuật toán tham lam để tìm cây bao trùm nhỏ nhất của một đồ thị vô hướng có trọng số liên thông. Nó khá giống với thuật toán Dijikstra (thuật toán tìm 1 đường đi ngắn nhất trên đồ thị), thuật toán Prim kết nạp từng đỉnh của đồ thị theo điều kiện độc nhất: đỉnh được kết nạp vào tiếp theo phải chưa được nạp và gần nhất với các đỉnh đã được nạp và đồ thị. Ta sẽ sử dụng một hàng đợi ưu tiên PriorQueue để lưu lại các cạnh có trọng số nhỏ nhất. Nhận xét một chút về độ phức tạp lúc này, là O((V + E)logV), vì mỗi đỉnh được thêm vào PO chỉ một lần và thao tác thêm phần tử vào hàng đợi có chi phí thời gian là hàm Logarit.

Đầu vào: Đồ thị vô hướng, liên thông, có trọng số $G = (V, U)$
Đầu ra: Cây khung nhỏ nhất T của đồ thị G.

```
1: Chọn một đỉnh bất kì s ∈ G.
2: D[s] = 0
3: for each v ∈ V \ {s} do
4:     D[v] = ∞
5:     v.parent = null
6: end for
7: Khởi tạo T = ∅
8: Khởi tạo hàng đợi ưu tiên Q = (D[v], v) for each v ∈ V.
9: T.connect(u)
10: while Q không rỗng do
11:     u = Q.removeMin()
12:     for each v ∈ G.adjacent[u] do
13:         if v ∈ Q và w(u, v) < D[v] then
14:             D[v] = w(u, v)
15:             v.parent = u
16:             T.connect(v)
17:         end if
18:     end for
19: end while
```

Minh họa thuật toán Prim

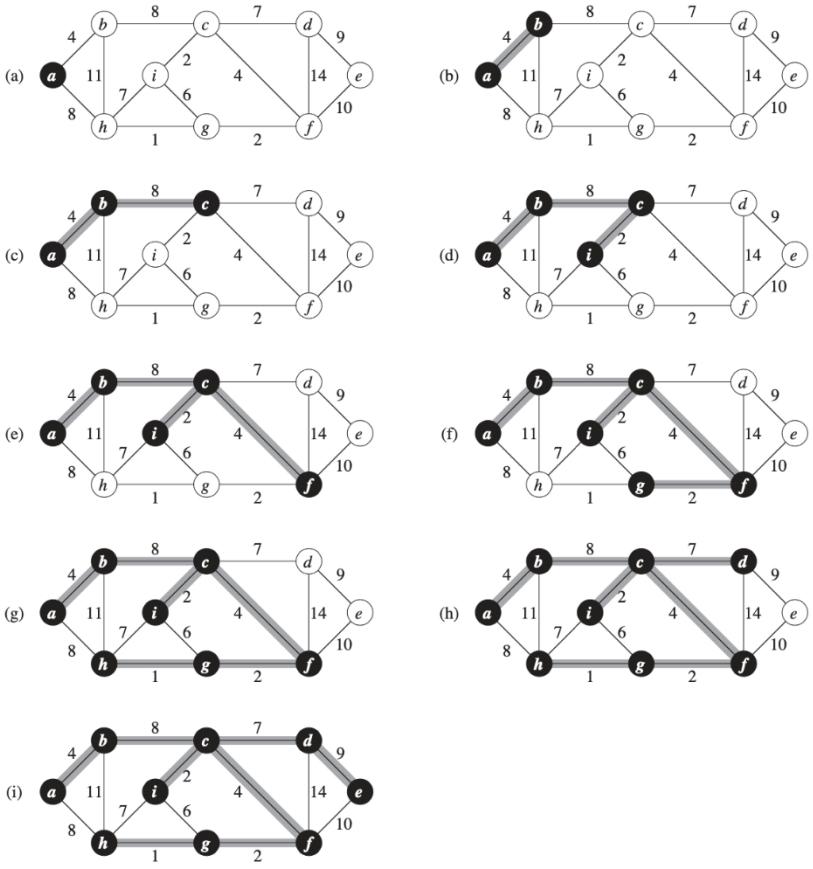


Figure 23.5 The execution of Prim's algorithm on the graph from Figure 23.1. The root vertex is a. Shaded edges are in the tree being grown, and black vertices are in the tree. At each step of the algorithm, the vertices in the tree determine a cut of the graph, and a light edge crossing the cut is added to the tree. In the second step, for example, the algorithm has a choice of adding either edge (b, c) or edge (a, h) to the tree since both are light edges crossing the cut.

Minh họa Trực quan thuật toán Prim bằng Graph

4. Giải thuật Heuristic Nearest Insertion

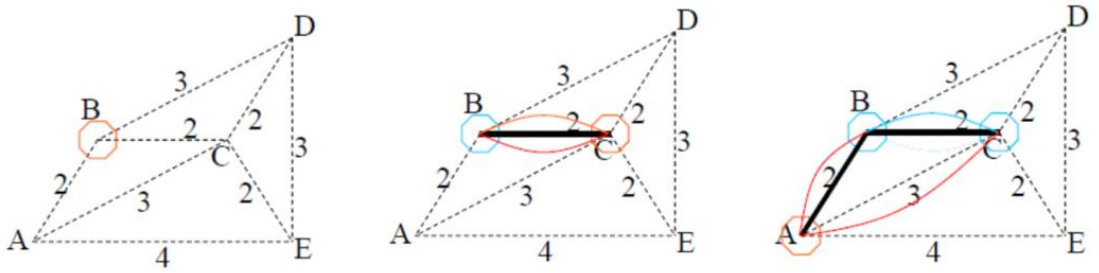
Hiện tại, lời giải tối ưu cho bài toán TSP vẫn chưa được tìm ra, nhưng tuy vậy, có một vài lời giải mà chúng ta đều có thể tạm chấp nhận. Một trong số đó là sử dụng giải thuật heuristic, cụ thể là phương pháp Chèn gắn gần nhất (Nearest Insertion).

Nearest Insertion mở rộng hành trình (tour) bằng cách chèn điểm mới vào giữa các điểm hiện có, đảm bảo tổng trọng số là nhỏ nhất. Phương pháp này có độ phức tạp $O(n^2)$ vì các bước tìm đỉnh và cạnh để chèn có độ phức tạp là O(n).

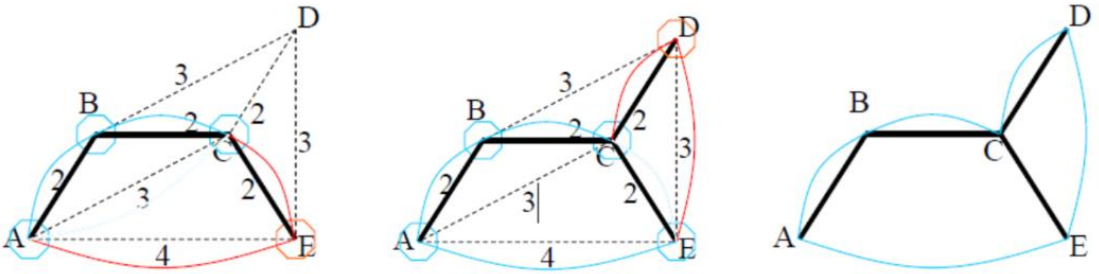
Đầu vào: Đồ thị vô hướng, liên thông, có trọng số $G = (V, U)$
Đầu ra: Đường đi ngắn nhất P

1: P.addTour(k)
2: Tìm node r sao cho c_{kr} nhỏ nhất.
3: P.addTour(r)
4: **for each** $v \in V \setminus \{i, r\}$ **do**
5: Tìm node $r \notin P.V$ sao cho c_{vr} nhỏ nhất.
6: Tìm cạnh $(i, j) \in P.E$ sao cho $c_{ir} + c_{rj} - cij$ nhỏ nhất.
7: Chèn r vào giữa i và j.
8: **end for**

Minh họa cho giải thuật.



(a) Chọn nút bất kì (b) Lần lặp thứ nhất (c) Lần lặp thứ hai



(d) Lần lặp thứ ba (e) Lần lặp thứ tư (f) Kết quả cuối cùng

Minh họa bằng trực quan cho giải thuật.

Thực hành và Coding:

1. **Nhận xét về Code:**

- **Graph.printMST(self, parent, g, d_temp, t)**: In ra cây khung và trả về trọng số của cây khung nhỏ nhất.
- **Graph.minKey(self, key, mstSet):** Tìm giá trị nhỏ nhất trong tập đỉnh thuộc cây khung.
- **Graph.primMST(self, g, d_temp, t):** Thực hiện thuật toán Prim tìm cây khung nhỏ nhất.
- **heuristic(tree, p_id, t, V, graph):** Thực hiện giải thuật heuristic Nearest Insertion.
- **checkPath(tree, toExpand, V):** Kiểm tra và in ra đường đi ngắn nhất.
- **tartTSP(graph, tree, V):** Giải bài toán TSP bằng thuật toán A*.

2. **Sửa lỗi – Debug**

Đoạn code cho trong File thực hành xuất hiện một số lỗi sau

a. Phải import thư viện TreeLib

Ta sử dụng: !pip install treelib trên colab hoặc pip install treelib trong cmd để sử dụng trong các IDE.

b. def minKey

Ta phải khai báo biến min_index = một giá trị cụ thể nào đó, ở đây em khai báo = -1

min_index = -1

c. def Heuristic

- Với l = len(visited), ta phải khai báo num = V – l, không phải V – 1.

Sau khi debug 3 vấn đề trên, đoạn code chạy không bị lỗi.

3. Chạy và kết quả

Xây dựng hàm run_test_cases để chạy 6 Test Case:

```
def run_test_cases():
    test_cases = [
        # Test case 1: Smallest graph
        (2, [[0, 100], [100, 0]]),
        # Test case 2: Simple 3-node graph
        (3, [[0, 300, 200], [300, 0, 500], [200, 500, 0]]),
        # Test case 3: Original 4-node graph
        (4, [[0, 5, 2, 3], [5, 0, 6, 3], [2, 6, 0, 4], [3, 3, 4, 0]]),
        # Test case 4: Fully connected graph with random distances
        (4, [[0, 10, 15, 20], [10, 0, 35, 25], [15, 35, 0, 30], [20, 25, 30, 0]]),
        # Test case 5: Symmetric graph with equal distances
        (4, [[0, 5, 5, 5], [5, 0, 5, 5], [5, 5, 0, 5], [5, 5, 5, 0]]),
        # Test case 6: Asymmetric graph
        (3, [[0, 10, 20], [5, 0, 25], [15, 10, 0]]),
    ]

    for i, (V, graph) in enumerate(test_cases):
        print(f"Test Case {i + 1}: V = {V}")
        for row in graph:
            print(row)
        print("-" * 30)
        tree = Tree()
        ans = startTSP(graph, tree, V)
        print("Ans is " + str(ans))
        print("-" * 30)
```

Kết quả:

```
Test Case 1: V = 2
[0, 100]
[100, 0]
--------------------------
Path complete
[0, 1, 0]
Ans is 200
```

```
Test Case 2: V = 3
[0, 300, 200]
[300, 0, 500]
[200, 500, 0]
----------------------------
Path complete
[0, 2, 1, 0]
Ans is 1000
```

```
Test Case 3: V = 4
[0, 5, 2, 3]
[5, 0, 6, 3]
[2, 6, 0, 4]
[3, 3, 4, 0]
----------------------------
Path complete
[0, 2, 3, 1, 0]
Ans is 14
```

```
Test Case 4: V = 4
[0, 10, 15, 20]
[10, 0, 35, 25]
[15, 35, 0, 30]
[20, 25, 30, 0]
----------------------------
Path complete
[0, 1, 3, 2, 0]
Ans is 80
```

```
Test Case 5: V = 4
[0, 5, 5, 5]
[5, 0, 5, 5]
[5, 5, 0, 5]
[5, 5, 5, 0]
----------------------------
Path complete
[0, 1, 2, 3, 0]
Ans is 20
```

```
Test Case 6: V = 3
[0, 10, 20]
[5, 0, 25]
[15, 10, 0]
----------------------------
Path complete
[0, 1, 2, 0]
Ans is 35
```

Nhận xét:

- Chương trình cho ở trên chỉ giải được các bài toán khi và chỉ khi từng cặp đỉnh trong đồ thị được nối với nhau bởi 1 cạnh.
- Thuật toán trên sẽ chạy rất chậm nếu số lượng đỉnh lớn (5, 6, 7,…)
- Ngoài HNI (heuristic nearest insertion), ta có thể sử dụng các giải thuật khác như Lin-Kernighan hay 3-opt,…

Người báo cáo
Nguyễn Thanh Kiên