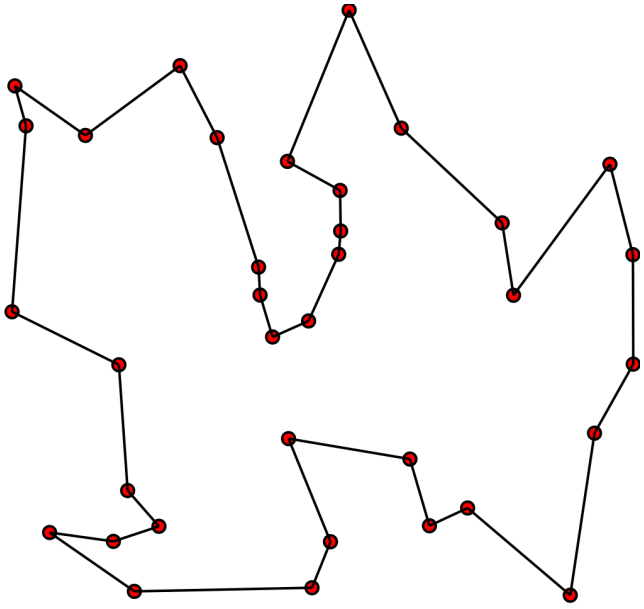
Bài tập Thực hành Nhập môn Trí tuệ Nhân tạo tuần 4

Nguyễn Lê Ngọc Duy - 20280023 - 20KDL1

Mục lục

1 Bài toán người đi du lịch - Travelling salesman problem (TSP).

Trong các bài toán về đường đi, bài toán người đi du lịch (còn gọi tắt là TSP) là một trong những bài toán được nghiên cứu nhiều nhất. Bài toán này được đề xuất vào thế kỉ XVII bởi hai nhà toán học vương quốc Anh là Sir William Rowan Hamilton và Thomas Penyngton Kirkman, và được ghi trong cuốn giáo trình Lý thuyết đồ thị nổi tiếng của Oxford. Nó nhanh chóng trở thành bài toán khó thách thức toàn thế giới bởi độ phức tạp thuật toán tăng theo hàm số mũ (trong chuyên ngành thuật toán người ta còn gọi chúng là những bài toán NP-hard). Người ta bắt đầu thử và công bố các kết quả giải bài toán này trên máy tính từ năm 1954 (49 đỉnh), cho đến năm 2004 bài toán giải được với số đỉnh lên tới 24.978, và dự báo sẽ còn tiếp tục tăng cao nữa.
Bài toán này được đặt ra như sau: Cho một tập các thành phố và khoảng cách giữa các cặp thành phố, đồng thời giữa hai thành phố bất kì **luôn tồn tại đường đi giữa hai thành phố đó với nhau**, nhiệm vụ của chúng ta tìm một đường đi bắt đầu từ thành phố xuất phát, đi qua tất cả các thành phố đúng 1 lần và trở lại thành phố xuất phát sao cho tổng đường đi là nhỏ nhất.
Ta có thể phát biểu lại bài toán dưới đạng đồ thị như sau: Cho một đồ thị vô hướng có trọng số, giữa hai đỉnh của đồ thị luôn được nối với nhau bởi ít nhất một cạnh, hãy tìm một đường đi bắt đầu từ đỉnh xuất phát, đi qua tất cả các đỉnh của đồ thị đúng 1 lần và quay trở lại đinh xuất phát sao cho độ dài của dường đi là nhỏ nhất.
Cho đến ngày này, các nhà khoa học đã tìm ra được nhiều phương pháp để giải quyết bài toán. Trong khuôn khổ của bài báo cáo tuần này, chúng ta sẽ cùng đi sâu vào cách giải sử dụng thuật toán A*, kết hợp bởi **cây khung nhỏ nhất** (Minimum Spanning Tree - MST) và thuật heuristic **chèn gần nhất** (Nearest Insertion).



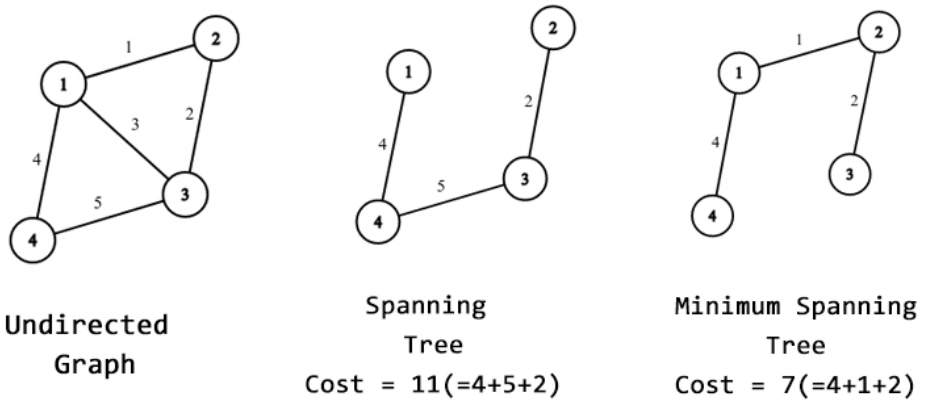
Hình 1: Một lời giải cho bài toán TSP

2 Cây khung nhỏ nhất (MST) - Thuật toán Prim.

2.1 Cây khung nhỏ nhất (MST).

Trong lý thuyết 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* - nghĩa là từ một đỉnh bất kỳ có thể đi tới các đỉnh khác mà chỉ dùng các cạnh trên tập hợp đó.
- Tổng trọng số của các cạnh trong tập hợp này là *nhỏ nhất*.



Hình 2: Minh hoạ cây khung và cây khung nhỏ nhất của đồ thị vô hướng có trọng số

Trong bài báo cáo này, chúng ta sẽ tìm hiểu một thuật toán có thể tìm được cây khung nhỏ nhất của đồ thị, đó là thuật toán Prim.

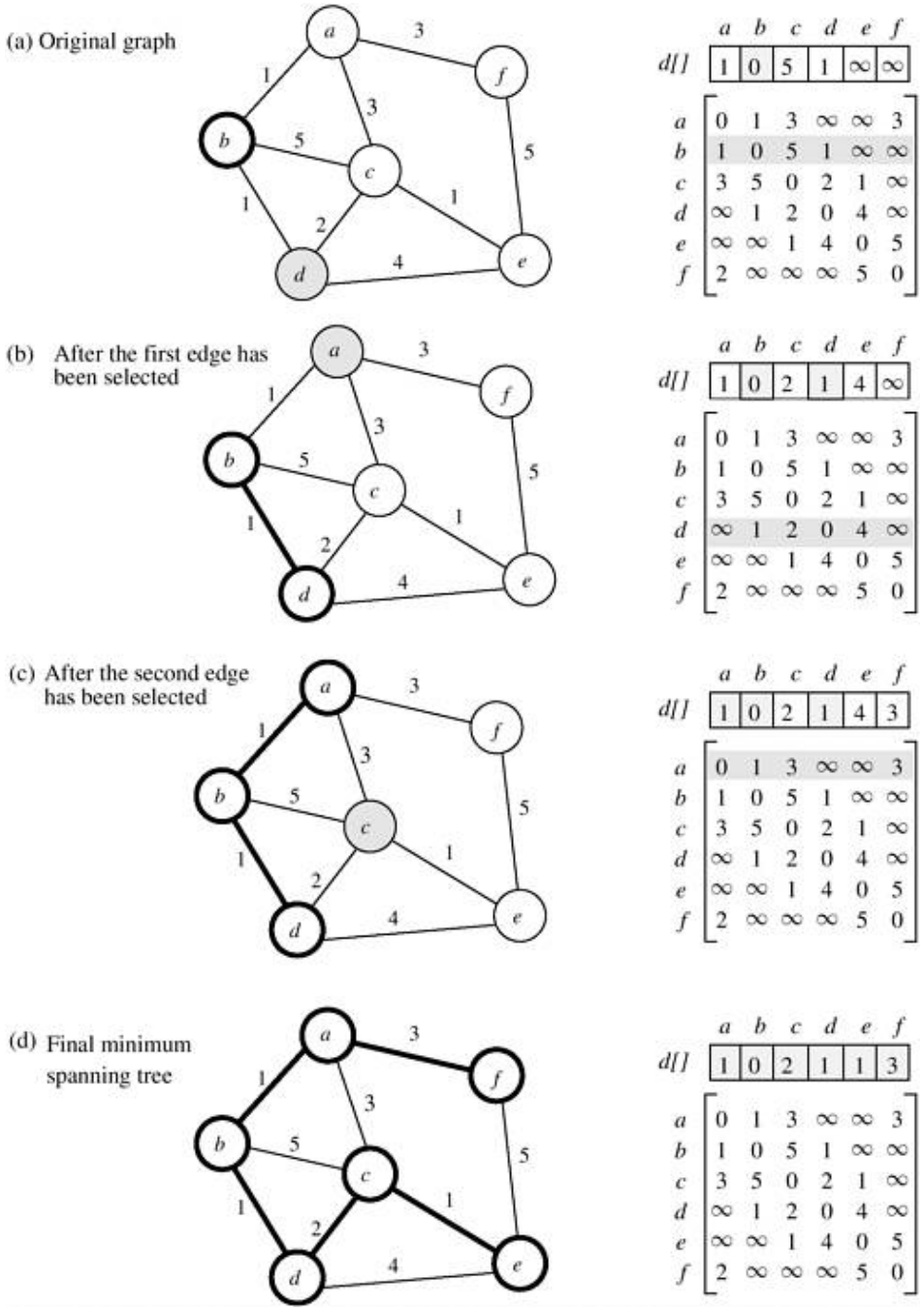
2.2 Thuật toán Prim.

Về mặt ý tưởng, thuật toán Prim rất giống với ý tưởng của thuật toán Dijikstra (tìm đường đi ngắn nhất trên đồ thị). Thuật toán Prim kết nạp từng đỉnh của đồ thị theo tiêu chí: đỉnh được kết nạp vào tiếp theo phải *chưa được nạp* và *gần nhất* với các đỉnh đã được nạp vào đò thị. Ta có thể dùng một hàng đợi ưu tiên lưu lại các cạnh có trọng số nhỏ nhất, khi đó độ phức tạp thời gian của thuật toán Prim là $O((V + E) \log V)$ vì mỗi đỉnh được thêm vào hàng đợi chỉ một lần và thao tác thêm phần tử vào hàng đợi có chi phí thời gian là hàm logarit.
Mã giả cho thuật toán Prim và hình ảnh minh hoạ thuật toán được thể hiện dưới đây.

Thuật toán 1 Thuật toán Prim

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
```



Hình 3: Minh hoạ thuật toán Prim

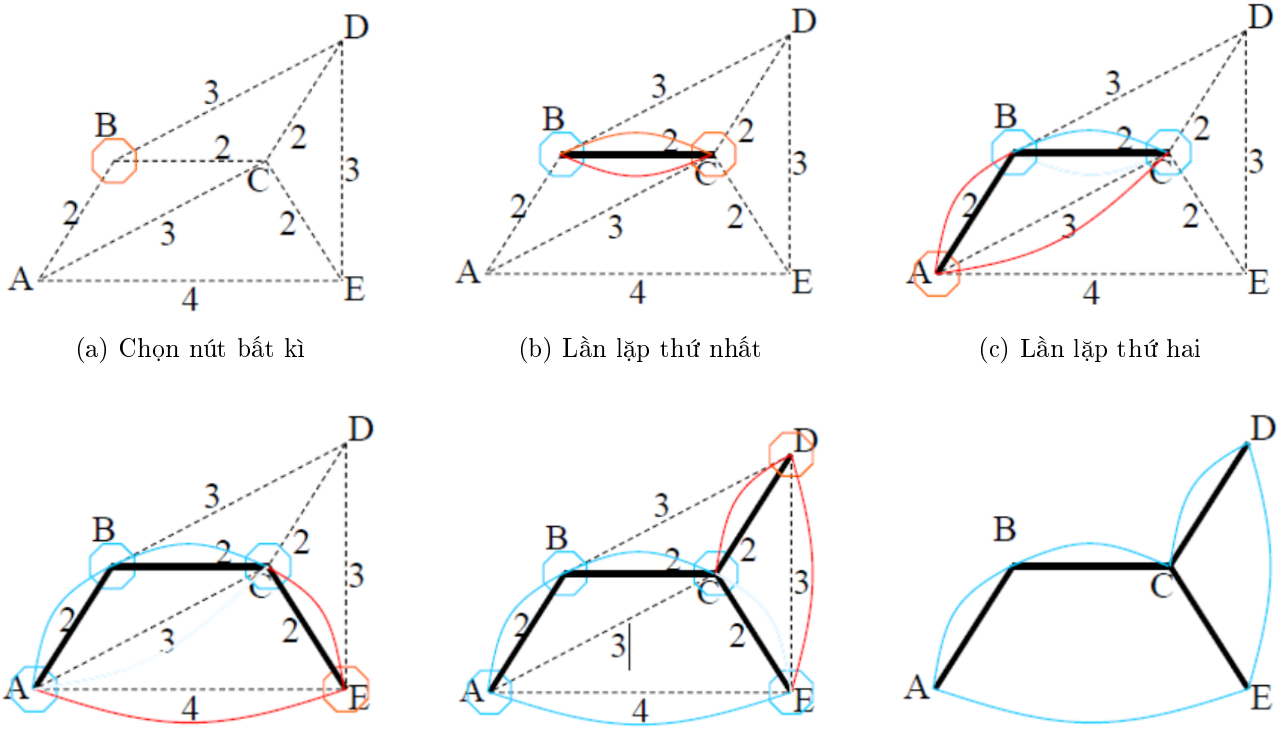
3 Giải thuật heuristic Chèn gần nhất - Nearest Insertion.

Cho đến thời điểm hiện tại, lời giải nhanh và tối ưu nhất cho bài toán TSP vẫn chưa được tìm ra. Mặc dù vậy, ta có thể tìm ra được lời giải *tạm chấp nhận được* cho bài toán một cách nhanh chóng. Một trong số các kĩ thuật để có thể tìm ra được những lời giải như vậy chính là kỹ thuật heuristic, và trong bài báo cáo này, chúng ta sẽ dùng kĩ thuật Chèn gần nhất (Nearest Insertion) để tìm một lời giải *tạm chấp nhận được* cho bài toán TSP.
Nearest Insertion là kỹ thuật mở rộng đường đi, gọi là *tour*, bằng cách chèn những điểm mới vào những điểm trong tour trước đó. Giải thuật này tìm những điểm nào không nằm trong tour gần nhất với bất kì điểm nào trong tour, sau đó chèn giữa hai điểm nào đó trong tour sao cho tổng trọng số trong tour là nhỏ nhất. Độ phức tạp của giải thuật này là $O(n^2)$ vì các bước tìm đỉnh và cạnh để chèn có độ phức tạp $O(n)$.
Mã giả và minh hoạ cho giải thuật Nearest Insertion được minh hoạ như hình dưới

Thuật toán 2 Giải thuật Nearest Insertion

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**



(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

Hình 4: Minh hoạ giải thuật heuristic Nearest Insertion

4 Kết quả trên Python.

Vận dụng thuật toán Prim và giải thuật heuristic Nearest Insertion, ta có thể cài đặt chương trình giải bài toán TSP bằng thuật toán A* trên Python 3.9.13.
Một số hàm cần lưu ý trong chương trình gồm:

- `Graph.printMST(self, parent, g, d_temp, t)`: in ra cây khung nhỏ nhất của đồ thị và trả về trọng số của cây khung nhỏ nhất.
- `Graph.minKey(self, key, mstSet)`: tìm giá trị nhỏ nhất trong tập hợp các đỉnh của cây khung nhỏ nhất.
- `Graph.primMST(self, g, d_temp, t)`: thực hiện thuật toán tìm cây khung nhỏ nhất của đồ thị.
- `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 của bài toán.
- `startTSP(graph, tree, V)`: thực hiện việc giải bài toán TSP bằng thuật toán A*.

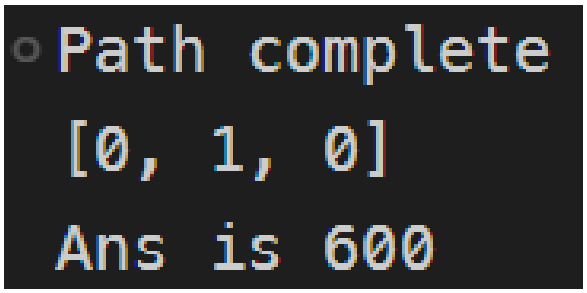
Các test case và kết quả của chương trình được minh hoạ như hình 5 và hình 6.

```
# Test case 1:
# V = 2
# graph = [[0, 300], [300, 0]]

# Test case 2:
# V = 3
# graph = [[0, 300, 200], [300, 0, 500], [200, 500, 0]]

# Test case 3:
# V = 4
# graph = [[0, 5, 2, 3], [5, 0, 6, 3], [2, 6, 0, 4], [3, 3, 4, 0]]

# Test case 4:
# V = 4
# graph = [[0, 10, 15, 20], [10, 0, 35, 25], [15, 35, 0, 30], [20, 25, 30, 0]]
```

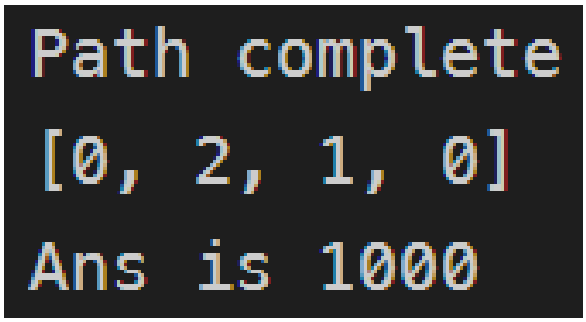
Hình 5: Các test case của chương trình

5 Nhận xét.

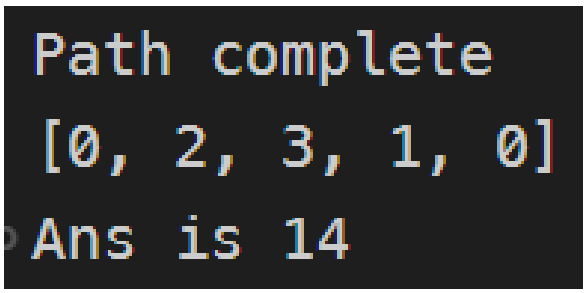
- Chương trình trên chỉ giải được khi từng cặp đỉnh trong đò thị được nối với nhau bởi 1 cạnh.
- Thuật toán trên có thể chạy chậm nếu số lượng đỉnh rất lớn.
- Ngoài giải thuật heuristic Nearest Insertion, ta có thể tối ưu lời giải bằng các giải thuật heuristic khác như giải thuật 3-Opt, giải thuật Lin-Kernighan,...



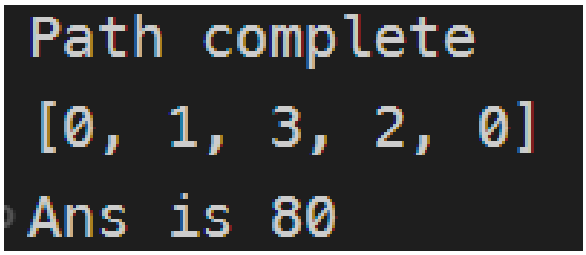
(a) Test case 1



(b) Test case 2



(c) Test case 3



(d) Test case 4

Hình 6: Kết quả chạy chương trình với các test case ở hình 5