BÀI BÁO CÁO MÔN PHÂN TÍCH VÀ THIẾT KẾ THUẬT TOÁN

Ngày 23 tháng 1 năm 2021

I Giới thiệu bài toán:

1 Mô tả:

Bài toán đường đi ngắn nhất của tất cả các cặp đỉnh là bài toán tìm một đường đi giữa cho tất cả các cặp đỉnh sao cho tổng các trọng số của các cạnh tạo nên đường đi đó là nhỏ nhất. Cho trước một đồ thị có trọng số (nghĩa là một tập đỉnh V, một tập cạnh E, và một hàm trong số có giá trị thực $f : E \to \mathbb{R}$), xét tất cả đỉnh v thuộc V, tìm một đường đi P từ v tới mỗi đỉnh v' thuộc V sao cho:

$$\sum_{p \in P} (f(p))$$

là nhỏ nhất trong tất cả các đường nối từ v tới v'.

2 Lịch sử:

- Rất khó để truy ngược lịch sử của bài toán tim đường đi ngắn nhất của tất cả các cặp đỉnh.
- So với các bài toán tối ưu hóa tổ hợp khác: như cây bao trùm nhỏ nhất, bài toán vận chuyển, việc nghiên cứu toán học trong bài toán đường đi ngắn nhất bắt đầu khá muộn.
- Các phương pháp tiếp cận không tối ưu đã được nghiên cứu như: Rosenfeld [1956], người đã đưa ra phương pháp phỏng đoán để xác định một tuyến đường vận tải đường bộ tối ưu thông qua một mô hình tắc nghẽn giao thông nhất định.
- Việc tìm đường đi, đặc biệt là tìm kiếm trong mê cung, thuộc về các bài toán đồ thị cổ điển, và các tài liệu tham khảo cổ điển là Wiener [1873], Lucas [1882] (mô tả một phương pháp do CP Trémaux), và Tarry [1895] - xem Biggs, Lloyd và Wilson [1976]. Chúng tạo cơ sở cho các kỹ thuật tìm kiếm theo chiều sâu.
- Các vấn đề về đường đi cũng được nghiên cứu vào đầu những năm 1950 trong bối cảnh 'định tuyến thay thế', tức là tìm đường ngắn thứ hai nếu đường ngắn nhất bị chặn. Điều này áp dụng cho việc sử dụng đường cao tốc (Trueblood [1952]), cũng như định tuyến cuộc gọi điện thoại.

3 Ứng dụng:

- Ứng dụng trong viễn thông, bài toán đường đi ngắn nhất đôi khi được gọi là bài toán đường đi có độ trễ nhỏ nhất (min-delay path problem) và thường được gắn với một bài toán đường đi rộng nhất

(widest path problem). ví dụ đường đi rộng nhất trong các đường đi ngắn nhất (độ trễ nhỏ nhất) hay đường đi ngắn nhất trong các đường đi rộng nhất.

- Kết hợp với bản đồ số.
- Ứng dụng trong bài toán xe buýt.
- Ứng dụng tron bài toán taxi.

4 Đề bài liên hệ với bản thân:

Bài toán tìm đường đi đến mấy chỗ giao hàng sao cho tốn ít chi phí nhất:

- Em đang thực hiện việc bán hàng online (tai nghe, chuột , cáp sạc), mọi người thường chọn mặt hàng mua sau đó gửi thông tin liên lạc gồm tên , số điện thoại , địa chỉ để em chốt đơn. Vì là sinh viên nghèo nên em không thể công tác với các công ty vận chuyển được mà phải tự mình đi giao hàng.
- Vào một ngày đẹp trời, em đang đi giao hàng cho khách. hôm nay, em phải giao 8 đơn hàng. Trước khi đi, em đã tính xem đi giao lần lượt những đơn nào là tối ưu nhất để có tiết kiệm thời gian và xăng. Tuy nhiên , khi đi giao đến bạn thứ 3 em lại nhận ra mình đã bỏ quên 1 gói hàng của bạn thứ 7 ở nhà. Việc này làm kế hoạch ban đầu bị rối loạn. em phải suy nghĩ xem, ở địa điểm hiện tại có đường nào có thể giao các đơn tiếp theo mà có thể ghé ngang nhà lấy hàng để giao cho bạn thứ 7 không.
- Chính vì thế em đã tìm hiểu trên mạng và thấy có thuật toán Floyd-Warshall có thể tính được tất cả đường đi giữa tất cả các địa điểm với nhau sao cho nó là ngắn nhất. Em đã cài đặt để sau này gặp trường hợp tương tự em chỉ cần lấy kết quả đã tính trước khi đi ra để xem ở địa điểm hiện tại con đường nào là tối ưu nhất để em có thể ghé nhà.

II Ý tưởng thiết kế cho bài toán:

1 Dẫn nguồn và lịch sử phát triển thuật toán:

1.1 Dẫn nguồn:

- Khái niệm, lịch sử, mã giả và chi tiết về thuật toán Floyd-warshall được tham khảo trên WikipediA https://en.wikipedia.org/wiki/Floyd\T5\textendashWarshall_algorithm.
- Source code được tham khảo trên Techie Delight

  https://www.techiedelight.com/pairs-shortest-paths-floyd-warshall-algorithm/?fbc

1.2 Lịch sử phát triển:

- Thuật toán Floyd – Warshall là một ví dụ về lập trình động, và được Robert Floyd công bố và được công nhận vào năm 1962.
- Về cơ bản, nó giống với các thuật toán được Bernard Roy công bố trước đó vào năm 1959 và cũng bởi Stephen Warshall vào năm 1962 để tìm ra điểm bắc cầu của một đồ thị, và có liên quan chặt chẽ với thuật toán của Kleene (đã xuất bản vào năm 1956) để chuyển đổi một automaton hữu hạn xác định thành một biểu thức chính quy.
- Công thức hiện tại của thuật toán dưới dạng ba vòng for lồng nhau lần đầu tiên được Peter Ingerman mô tả vào năm 1962.

2 Phương pháp đã dùng để thiết kế:

Thuật toán Floyd-Warshall được thiết kế bằng phương pháp quy hoạch động (Dynamic Programing). Nó chia bài toán tìm đường đi ngắn nhất giữa các cặp điểm, thành các bài toán tìm đường ngắn nhất giữa hai điểm là đường nối giữa hai điểm đó hoặc nó sẽ đi qua một vài điểm trung gian. Do tất cả các bài toán con được giải quyết trước, sau đó dùng kết quả để tìm lời giải cho bài toán lớn nên nó được tiếp cận theo hướng bottom-up.

3 Mã giả:

Cho dist là một $|V| \times |V|$ mảng khoảng cách được tạo thành từ ∞.

for mỗi cạnh (u, v):

    dist[u][v] = w(u, v) // Trọng số của cạnh (u, v)

for mỗi vertex v:

    dist[v][v] ← 0

for k in range(1, |V|)

    for i in range(1, |V|)

        for j in range(1, |V|)

            if dist[i][j] > dist[i][k] + dist[k][j]:

                dist[i][j] = dist[i][k] + dist[k][j]

4 Phân tích độ phức tạp bằng phương pháp toán học:

```
(1) for k in range(N):
(2)     for v in range(N):
(3)         for u in range(N):
(4)             if cost[v,k] != float('inf') and cost[k,u] != float('inf') and (cost[v,k] + cost[k,u] < cost[v,u]):
(5)                 cost[v,u] = cost[v,k] + cost[k,u]
(6)                 path[v,u] = path[k,u]
(7)         if cost[v,v] < 0:
(8)             return
```

Ta có: $T(1) = 1(1 + 1 * 1)(1 + 9 * 1)$
$T(2) = 2(2 + 1 * 2)(2 + 9 * 2)$
$T(3) = 3(3 + 1 * 3)(3 + 9 * 3)$
$=> T(n) = n(n + n)(n + 9n)$
$= 20n^3$
Đặt: $C = 20$
$T(n) = Cn^3$
$=> T(n) = O(n^3)$
Với n là số đỉnh của đò thị.

5 Mã nguồn cài đặt:

```
for k in range(N):
    for v in range(N):
        for u in range(N):
            # nếu có đường đi từ v -> u đi qua đỉnh trung gian k , sao cho cost(v -> k) + cost(k -> u)
            # nhỏ hơn so với cost(v -> u)
            # thì gán cost(u->v) = cost(v -> k) + cost(k -> u) và gán path(u -> v) = path(k -> u)
            # hay nói cách khách đi đường v -> k -> u tốn ít chi phí hơn đi đường v -> u
            # thì cập nhật lại chi phí đi từ v -> u và cập nhật lại đỉnh đi đến u
            if cost[v,k] != float('inf') and cost[k,u] != float('inf') and (cost[v,k] + cost[k,u] < cost[v,u]):
                cost[v,u] = cost[v,k] + cost[k,u]
                path[v,u] = path[k,u]
        # nếu phần từ đường chéo trờ thành âm, đồ thị chứa chu kì trọng số âm
        if cost[v,v] < 0:
            print("Phát hiện chu kì có trọng số âm")
```

6 Cách thức phát sinh Input/Output đã dùng để kiểm tra tính đúng đắn của cài đặt:

Input: Đầu vào của bài toán là một tra trận đồ thị có hướng, trong đó trọng số của cạnh có thể chứa giá trị âm

- Ta qui ước:
  - Đường chéo của ma trận có giá trị là 0.
  - Nếu không có cạnh nối giữa hai đỉnh của ma trận thì giá trị của nó là inf.
  - Nếu tồn tại cạnh nối giữa hai đỉnh thì giá trị của nó bằng chi phí của cạnh đó
- Ta sử dụng hàm random.random() để random ngẫu nhiên n đỉnh và p là khả năng xuất hiện cạnh nối giữa 2 đỉnh
- Sử dụng thư viện networkx.binomial_graph để tạo một đồ thị có hướng từ n và p
- Sau đó ta chuẩn hóa nó về dạng ma trận bằng hàm networkx.convert_matrix.to_numpy_matrix
- Tiếp tục chuẩn hóa ma trận thu được về dạng như ban đầu đã quy ước trong đó giá trị của cạnh sẽ được chọn ngẫu nhiên từ [-10,100]

Output:

- ma trận cost chứa giá trị đường đi ngắn nhất từ một đỉnh đến một đỉnh khác

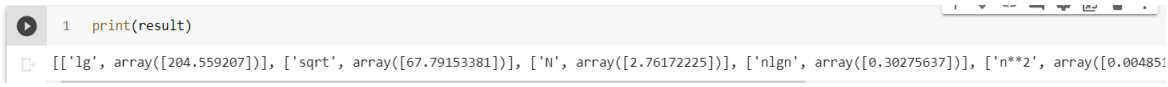
- ma trận path chứa đỉnh là đường đi đến đỉnh đang xét sau cho chi phí là ngắn nhất
- thử nghiệm vẽ một số đồ thị có số đỉnh nhỏ để kiểm chứng

7 Phân tích độ phức tạp bằng cài đặt:

- Sử dụng hàm CProfile để tiến hành đo thời gian thực hiện của các đồ thị có số đỉnh tăng dần:

```
for graph in list_Graph:
  cProfile.run('floydWarshall(graph,len(graph))')
```

- Sau khi thực hiện thu được 50 cặp giá trị là số đỉnh và thời gian thực thi
- Sử dụng LinearRegression để tìm hệ số X của các hàm độ phức tạp thuật toán cơ bản



- Thông qua việc tính toán ta tìm được mean square error của giá trị thời gian tính được và giá trị thời gian ban đầu , xét hàm nào có giá trị MSE nhỏ nhất thì có khả năng hàm đó là độ phức tạm của thuật toán cần tìm

665	2215.773		1918.195	88552.81	1748.181	218642.7	1836.545	143813.7	1887.942	107473.3	2145.561	4929.764	2232.398	276.3778
731	2950.225		1946.121	1008226	1832.88	1248459	2018.819	867517.2	2105.53	713510.1	2592.581	127909.5	2965.232	225.2236
		X	204.5592		67.79153		2.761722		0.302756		0.004852		7.59E-06	
		MSE		1.32E+06		2.78E+05		1.20E+05		1.01E+05		22214.65		5.89E+01
					Min	5.89E+01	suy ra	N**3 la do phuc tap can tim						