0.1 Bài toán đường đi ngắn nhất

Cho đơn đồ thị có hướng $G = (V, E)$ với hàm trọng số $w : E \to R$ ($w(e)$ được gọi là độ dài hay trọng số của cạnh e).

Độ dài của đường đi $P = v_1 \to v_2 \to ... \to v_k$ là số:

$$w(P) = \sum_{i=1}^{k-1} w(v_i, v_{i+1})$$

Đường đi ngắn nhất từ đỉnh u đến đỉnh v là đường đi có độ dài ngắn nhất trong số các đường đi nối u với v.

Độ dài của đường đi ngắn nhất từ u đến v còn được gọi là **Khoảng cách từ** u **tới** v và ký hiệu là $\delta(u, v)$.

Các dạng bài toán ĐĐNN

1. **Bài toán một nguồn một đích** : Cho hai đỉnh s và t, cần tìm đường đi ngắn nhất từ s đến t.
2. **Bài toán một nguồn nhiều đích** : Cho s là đỉnh nguồn, cần tìm đường đi ngắn nhất từ s đến tất cả các đỉnh còn lại.
3. **Bài toán mọi cặp** : Tìm đường đi ngắn nhất giữa mọi cặp đỉnh của đồ thị.

Ta thấy các bài toán được xếp theo thứ tự từ đơn giản đến phức tạp. Hễ có thuật toán hiệu quả để giải một trong ba bài toán thì thuật toán đó cũng có thể sử dụng để giải hai bài toán còn lại.

Nếu đồ thị có chu trình âm thì độ dài đường đi giữa hai đỉnh nào đó có thể làm nhỏ tuỳ ý. Vậy để thực hiện bài toán tìm đường đi ngắn nhất ta giả thiết đồ thị không chứa chu trình âm.

Biểu diễn đường đi ngắn nhất : các thuật toán tìm đường đi ngắn nhất làm việc với hai mảng:

- $d(v)$: độ dài đường đi từ s đến v ngắn nhất hiện biết (cận trên cho độ dài đường đi ngắn nhất thực sự).
- $p(v)$: đỉnh đi trước v trong đường đi nói trên (sẽ sử dụng để truy ngược đường đi từ s đến v).

Giảm cận trên - Relaxation: sử dụng cạnh (u, v) để kiểm tra xem đường đi đến v đã tìm được có thể làm ngắn hơn nhờ đi qua u hay không. Các thuật toán tìm đđnn khác nhau ở số

lần dùng các cạnh và trình tự duyệt chúng để thực hiện giảm cận.

Nhận xét chung

- Việc cài đặt các thuật toán được thể hiện nhờ **thủ tục gán nhãn**: Mỗi đỉnh v sẽ có nhãn gồm 2 thành phần $(d[v], p[v])$. Nhãn sẽ biến đổi trong quá trình thực hiện thuật toán.
- Nhận thấy rằng để tính khoảng cách từ s đến t, ở đây, ta phải tính khoảng cách từ s đến tất cả cách đỉnh còn lại của đồ thị.
- Hiện vẫn chưa biết thuật toán nào cho phép tìm đđnn giữa hai đỉnh làm việc thực sự hiệu quả hơn những thuật toán tìm đđnn từ một đỉnh đến tất cả các đỉnh còn lại.

0.2 Thuật toán Bellman-Ford

0.2.1 Ý tưởng thuật toán

Thuật toán Bellman-Ford tìm đường đi ngắn nhất từ đỉnh s đến tất cả các đỉnh còn lại của đồ thị.

Thuật toán làm việc trong trường hợp trọng số của các cung là tùy ý. Giả thiết rằng đồ thị không có chu trình âm.

Đầu vào :

- Đồ thị $G = (V, E)$ với n đỉnh xác định bởi ma trận trọng số $W[u, v], u, v \in V$, đỉnh nguồn $s \in V$.

Đầu ra : Với mỗi $v \in V$

- $d[v] = \delta[s, v]$.
- $p[v]$ - Đỉnh đi trước v trong đđnn từ s đến v.

0.2.2 Cài đặt

Em thực hiện cài đặt thuật toán theo mã giả:

Tính đúng đắn của thuật toán có thể chứng minh trên cơ sở nguyên lý tối ưu của quy hoạnh động.

Độ phức tạp của thuật toán là $O(n^3)$

Đối với đồ thị thưa, sử dụng danh sách kề sẽ tốt hơn, khi đó vòng lặp theo u viết lại thành: **For** $u \in Ke(v)$ **do**

Trong trường hợp này ta thu được thuật toán với độ phức tạp $O(n.m)$

Algorithm 1 Bellman-Ford algorithm

```
1:  procedure BELLMAN-FORD
2:      for v ∈ V do                                          ▷ Khởi tạo
3:          d[v] ← w[s, v]
4:          p[v] ← s
5:      end for
6:      d[s] ← 0
7:      p[s] ← 0
8:      for k = 1 to n − 2 do
9:          for v ∈ V\{s} do
10:             for u ∈ V do
11:                 if d[v] > d[u] + w[u, v] then             ▷ Bước Relaxation
12:                     d[v] ← d[u] + w[u, v]
13:                     p[v] ← u
14:                 end if
15:             end for
16:         end for
17:     end for
18: end procedure
```

0.3 Thuật toán Dijkstra

0.3.1 Ý tưởng thuật toán

Trong trường hợp trọng số trên các cung là không âm, thuật toán do Dijkstra là hiệu quả hơn nhiều so với thuật toán Bellman-Ford.

Thuật toán được cây dựng dựa trên thủ tục gán nhãn. Đầu tiên nhãn của các đỉnh là tạm thời. Ở mỗi bước lặp có một nhãn tạm thời trở thành nhãn cố định. Nếu nhãn của một đỉnh u trở thành cố định thì $d[u]$ sẽ cho ta độ dài của đđnn từ đỉnh s đến u. Thuật toán kết thúc khi nhãn của tất cả các đỉnh trở thành cố định.

Đầu vào :

- Đồ thị $G = (V, E)$ với n đỉnh xác định bởi ma trận trọng số $W[u, v], u, v \in V$, đỉnh nguồn $s \in V$.

Giả thiết: $w[u, v] \leqslant 0, u, v \in V$.

Đầu ra : Với mỗi $v \in V$

- $d[v] = \delta[s, v]$.
- $p[v]$ - Đỉnh đi trước v trong đđnn từ s đến v.

0.3.2 Cài đặt

Em thực hiện cài đặt thuật toán theo mã giả:

Algorithm 2 Dijkstra algorithm

```
1:  procedure DIJKSTRA
2:      for v ∈ V do                                              ▷ Khởi tạo
3:          d[v] ← w[s, v]
4:          p[v] ← s
5:      end for
6:      d[s] ← 0
7:      S ← {s}                                          ▷ S - Tập đỉnh có nhãn cố định
8:      T ← V\{s}                                        ▷ T - Tập đỉnh có nhãn tạm thời
9:      while T ≠ ∅ do Tìm đỉnh u ∈ T thỏa mãn d[u] = min{d[z] : z ∈ T}
10:         T ← T\{u}
11:         S ← S ∪ {u}                                        ▷ Cố định nhãn của đỉnh u
12:         for v ∈ T do                                  ▷ Gán nhãn lại cho các đỉnh trong T
13:             if d[v] > d[u] + w[u, v] then                          ▷ Bước relaxation
14:                 d[v] ← d[u] + w[u, v]
15:                 p[v] ← u
16:             end if
17:         end for
18:     end while
19: end procedure
```

Thuật toán Dijkstra theo mã giả trên tìm được đường đi ngắn nhất từ đỉnh s đến tất cả các đỉnh còn lại trên đồ thị sau thời gian $\mathcal{O}(n^2)$

0.4 So sánh với thư viện có sẵn

Để kiểm tra tính đúng đắn và hiệu quả của thuật toán đã cài đặt, em sử dụng thư viện *Networkx* để so sánh kết quả và thời gian chạy.

0.4.1 Dữ liệu

Dữ liệu được tạo ngẫu nhiên từ thư viện *networkx*. Do việc tạo một đồ thị lớn có trọng số âm mà không có chu trình âm rất khó khăn nên em chỉ tạo đồ thị có trọng số dương.

Và em cũng thực hiện lưu đồ thị đã tạo ngẫu nhiên dưới dạng ma trận trọng số trên một mảng trên numpy để tiện kiểm tra thuật toán và tránh phải lưu trữ các file dữ liệu.

0.4.2 So sánh thời gian

Sử dụng 2 đồ thị được tạo ngẫu nhiên, em thực hiện tổng hợp thời gian chạy dưới bảng sau:

	100 đỉnh, 4000 cạnh	2000 đỉnh, 2.10^6 cạnh
Dijsktra	0.0068361759185791016	2.077575206756592
Thư viện: Dijkstra	0.005284309387207031	2.4850828647613525
Bellman-Ford	0.8282551765441895	Không chạy được
Thư viện: Bellman-Ford	0.0045166015625	2.7311718463897705

Bảng 1: So sánh thời gian chạy với thư viện(Đơn vị: giây(s))

0.4.3 Phân tích lý do

Thời gian chạy giữa thư viện và tự cài đặt còn chênh lệch nhiều, có thể do các lý do:

- Sử dụng cấu trúc dữ liệu để lưu trữ chưa tốt
- Sử dụng nhiều vòng lặp lồng nhau nên không được nhanh
- Đối với đồ thị thưa, sử dụng ma trận kề là không hiệu quả (đặc biệt với thuật toán Bellman-Ford)