Đồ án môn học: Tổ hợp & Lý thuyết đồ thị
Course Project: Combinatorics & Graph Theory

Triển khai thuật toán Dijkstra

Bài tập 14, 15, 16: Các bài toán đường đi ngắn nhất trên các loại đồ thị khác nhau

Thực hiện bởi sinh viên
Huỳnh Nhật Quang
Trường Đại học Quản lý và Công nghệ, TP. Hồ Chí Minh

22 tháng 7, 2025

Mục lục

1 Giới thiệu

Tóm tắt nội dung

Báo cáo này trình bày một triển khai và phân tích toàn diện về thuật toán Dijkstra trong việc tìm đường đi ngắn nhất trên ba loại đồ thị khác nhau: Simple Graph, Multigraph, và General Graph. Báo cáo giải quyết các bài tập 14, 15 và 16 từ đồ án môn học Tổ hợp & Lý thuyết đồ thị. Chúng tôi cung cấp các dẫn xuất toán học chi tiết, phân tích thuật toán, và các triển khai hoàn chỉnh bằng cả C++ và Python.

Từ khóa: Dijkstra's Algorithm, Graph Theory, Shortest Path, Simple Graph, Multigraph, General Graph, Dynamic Programming

1.1 Đề bài

Thuật toán Dijkstra là một thuật toán tìm đường đi ngắn nhất giữa các đỉnh trong một đồ thị có trọng số, ví dụ như mạng lưới đường bộ. Đồ án này triển khai thuật toán Dijkstra cho ba loại đồ thị khác nhau theo yêu cầu của bài tập 14, 15 và 16 trong tài liệu môn học:

Bài tập 14: Simple Graph

Cho $G = (V, E)$ là một đồ thị đơn hữu hạn. Triển khai thuật toán Dijkstra để giải bài toán đường đi ngắn nhất trên G.

Bài tập 15: Multigraph

Cho $G = (V, E)$ là một đa đồ thị hữu hạn. Triển khai thuật toán Dijkstra để giải bài toán đường đi ngắn nhất trên G.

Bài tập 16: General Graph

Cho $G = (V, E)$ là một đồ thị tổng quát hữu hạn. Triển khai thuật toán Dijkstra để giải bài toán đường đi ngắn nhất trên G.

1.2 Định nghĩa các loại đồ thị

Định nghĩa 1.1 (Simple Graph)**.** Một đồ thị đơn $G = (V, E)$ là một đồ thị không có hướng, không có vòng lặp (self-loops) và không có cạnh song song. Chính thức:

- V là tập hợp hữu hạn các đỉnh
- $E \subseteq \binom{V}{2}$ là tập hợp các cạnh
- Với mọi $u, v \in V$: $(u, v) \in E \implies u \neq v$
- $|(u, v) \in E| \leq 1$ với mọi $u, v \in V$ khác nhau

Định nghĩa 1.2 (Multigraph)**.** Một đa đồ thị $G = (V, E)$ cho phép nhiều cạnh giữa cùng một cặp đỉnh nhưng không có vòng lặp. Chính thức:

- V là tập hợp hữu hạn các đỉnh
- E là một tập đa hợp các cạnh từ $\binom{V}{2}$
- Với mọi $u, v \in V$: nếu $(u, v) \in E$ thì $u \neq v$
- Nhiều cạnh (u, v) với trọng số khác nhau được phép

Định nghĩa 1.3 (General Graph)**.** Một đồ thị tổng quát $G = (V, E)$ cho phép cả vòng lặp và nhiều cạnh. Chính thức:

- V là tập hợp hữu hạn các đỉnh
- E là một tập đa hợp các cạnh từ $V \times V$
- Vòng lặp (u, u) được phép
- Nhiều cạnh giữa bất kỳ cặp đỉnh nào được phép

2 Cơ sở toán học

2.1 Lý thuyết thuật toán Dijkstra

Thuật toán Dijkstra tìm đường đi ngắn nhất từ một đỉnh nguồn (source node) đến tất cả các đỉnh khác trong một đồ thị có trọng số. Thuật toán sử dụng trọng số của các cạnh để tìm đường đi có tổng khoảng cách (trọng số) nhỏ nhất từ đỉnh nguồn đến các đỉnh còn lại.

Định lý 2.1 (Tính đúng đắn của thuật toán Dijkstra)**.** Cho $G = (V, E, w)$ là một đồ thị có trọng số với trọng số không âm, và $s \in V$ là đỉnh nguồn. Thuật toán Dijkstra tính toán chính xác khoảng cách đường đi ngắn nhất từ s đến tất cả các đỉnh khác trong G.

Chứng minh. Tính đúng đắn được chứng minh dựa trên hai thuộc tính: tính chất cấu trúc tối ưu (optimal substructure) và thuộc tính lựa chọn tham lam (greedy choice property):

Tính chất cấu trúc tối ưu: Nếu P là đường đi ngắn nhất từ s đến v và P đi qua đỉnh u, thì đoạn đường từ s đến u cũng là đường đi ngắn nhất từ s đến u.

Thuộc tính lựa chọn tham lam: Ở mỗi bước, việc chọn đỉnh chưa được thăm có khoảng cách nhỏ nhất đảm bảo rằng chúng ta đã tìm ra đường đi ngắn nhất đến đỉnh đó.

Bằng quy nạp theo số lượng đỉnh được xử lý, ta có thể chứng minh rằng thuật toán duy trì bất biến rằng với mỗi đỉnh đã xử lý v, $d[v]$ bằng khoảng cách đường đi ngắn nhất từ s đến v. □

2.2 Công thức đệ quy và lập trình động

Cốt lõi của thuật toán Dijkstra nằm ở bước thư giãn (relaxation), có thể được biểu diễn dưới dạng công thức đệ quy:

$$d[v] = \min\{d[v], d[u] + w(u, v)\} \tag{1}$$

trong đó:

- $d[v]$ là khoảng cách ngắn nhất hiện tại đến đỉnh v
- $d[u]$ là khoảng cách ngắn nhất đến đỉnh u
- $w(u, v)$ là trọng số của cạnh (u, v)

Định nghĩa 2.1 (Thư giãn)**.** Thư giãn một cạnh (u, v) với trọng số $w(u, v)$ là quá trình cập nhật $d[v]$ nếu tìm được đường đi ngắn hơn đến v thông qua u:

$$\text{RELAX}(u, v, w) = \begin{cases} d[v] \leftarrow d[u] + w(u, v) & \text{nếu } d[u] + w(u, v) < d[v] \\ \text{không thay đổi} & \text{ngược lại} \end{cases} \tag{2}$$

2.3 Công thức lập trình động

Thuật toán Dijkstra có thể được xem như giải bài toán lập trình động sau:

$$\delta(s, v) = \min_{u \in V}\{\delta(s, u) + w(u, v)\} \tag{3}$$

trong đó $\delta(s, v)$ biểu thị khoảng cách đường đi ngắn nhất từ nguồn s đến đỉnh v.
Trường hợp cơ bản:

$$\delta(s, s) = 0 \tag{4}$$

Tính chất cấu trúc tối ưu đảm bảo rằng:

$$\delta(s, v) = \min\{\delta(s, v), \delta(s, u) + w(u, v)\} \quad \forall (u, v) \in E \tag{5}$$

3 Thiết kế và triển khai thuật toán

3.1 Thuật toán Dijkstra chung

Algorithm 1 Thuật toán đường đi ngắn nhất của Dijkstra

```
1: procedure DIJKSTRA(G(V, E), w, s)
2:     Khởi tạo d[v] ← ∞ cho mọi v ∈ V
3:     d[s] ← 0
4:     Khởi tạo parent[v] ← NULL cho mọi v ∈ V
5:     Khởi tạo hàng đợi ưu tiên Q với tất cả các đỉnh
6:     Khởi tạo visited[v] ← false cho mọi v ∈ V
7:     while Q ≠ ∅ do
8:         u ← EXTRACT-MIN(Q)
9:         visited[u] ← true
10:        for mỗi đỉnh láng giềng v của u do
11:            if ¬visited[v] and d[u] + w(u, v) < d[v] then
12:                d[v] ← d[u] + w(u, v)
13:                parent[v] ← u
14:                DECREASE-KEY(Q, v, d[v])
15:            end if
16:        end for
17:    end while
18:    return (d, parent)
19: end procedure
```

3.2 Xem xét đặc trưng của từng loại đồ thị

3.2.1 Triển khai trên Simple Graph

Đối với đồ thị đơn, triển khai khá đơn giản vì không cần xử lý các trường hợp đặc biệt:

- Mỗi cạnh (u, v) chỉ xuất hiện đúng một lần
- Không cần xem xét vòng lặp
- Biểu diễn danh sách kề chuẩn là đủ

3.2.2 Triển khai trên Multigraph

Đối với đa đồ thị, cần xử lý các cạnh song song:

- Nhiều cạnh giữa cùng một cặp đỉnh với trọng số khác nhau
- Trong quá trình thư giãn, xem xét tất cả các cạnh song song
- Thuật toán tự động chọn cạnh có trọng số nhỏ nhất trong số các cạnh song song
- Việc xác định cạnh là quan trọng để theo dõi

Bổ đề 3.1 (Xử lý cạnh song song)**.** Trong một đa đồ thị có các cạnh song song giữa các đỉnh u và v, thuật toán Dijkstra sẽ tự động sử dụng cạnh có trọng số nhỏ nhất để tính toán đường đi ngắn nhất.

3.2.3 Triển khai trên General Graph

Đồ thị tổng quát cần xử lý cả cạnh song song và vòng lặp:

- Vòng lặp (v, v) không ảnh hưởng đến đường đi ngắn nhất nếu trọng số không âm
- Cạnh song song được xử lý như trong đa đồ thị
- Vòng lặp thường được bỏ qua trong quá trình thư giãn

Bổ đề 3.2 (Thuộc tính vòng lặp)**.** Với trọng số cạnh không âm, vòng lặp không đóng góp vào đường đi ngắn nhất trong thuật toán Dijkstra.

Chứng minh. Một vòng lặp (v, v) với trọng số $w \geq 0$ chỉ làm tăng độ dài đường đi. Bất kỳ đường đi ngắn nhất nào đến v bao gồm vòng lặp sẽ có độ dài $d[v] + w \geq d[v]$, không thể ngắn hơn đường đi trực tiếp có độ dài $d[v]$. □

4 Phân tích độ phức tạp

4.1 Độ phức tạp thời gian

Phân tích độ phức tạp

Phân tích độ phức tạp thời gian:
Độ phức tạp thời gian của thuật toán Dijkstra phụ thuộc vào cấu trúc dữ liệu được sử dụng cho hàng đợi ưu tiên:

1. **Binary Heap:** $O((V + E) \log V)$
2. **Fibonacci Heap:** $O(E + V \log V)$
3. **Array Implementation:** $O(V^2)$

Đối với đồ thị dày đặc với $E = O(V^2)$, triển khai Binary Heap cho độ phức tạp $O(V^2 \log V)$. Đối với đồ thị thưa với $E = O(V)$, độ phức tạp là $O(V \log V)$.

5 Chi tiết triển khai

5.1 Triển khai bằng C++

Triển khai C++ sử dụng thiết kế hướng đối tượng với kế thừa để xử lý các loại đồ thị khác nhau:

Listing 1: Cấu trúc lớp đồ thị cốt lõi

```cpp
class Graph {
protected:
    int numVertices;
    vector<vector<Edge>> adjList;

public:
    Graph(int n) : numVertices(n), adjList(n) {}
    virtual void addEdge(int u, int v, int weight, int edgeId = 0) = 0;

    // Triển khai Dijkstra chung
    pair<vector<int>, vector<int>> dijkstra(int source);
};
```

5.2 Triển khai bằng Python

Triển khai Python tập trung vào tính dễ đọc và sử dụng chú thích kiểu:

Listing 2: Lớp Edge và Graph trong Python

```python
class Edge:
    def __init__(self, to: int, weight: int, edge_id: int = 0):
        self.to = to
        self.weight = weight
        self.id = edge_id

class Graph:
    def __init__(self, num_vertices: int):
        self.num_vertices = num_vertices
        self.adj_list = [[] for _ in range(num_vertices)]
```

6 Phân tích triển khai theo từng loại đồ thị

6.1 Simple Graph (Bài tập 14)

Phương pháp giải quyết

Phương pháp giải quyết cho Simple Graph:
Triển khai trên đồ thị đơn tập trung vào việc ngăn chặn các cạnh trùng lặp và vòng lặp, đồng thời duy trì hiệu quả trong việc tính toán đường đi ngắn nhất.

Listing 3: Thêm cạnh cho Simple Graph

```cpp
void SimpleGraph::addEdge(int u, int v, int weight, int edgeId) {
    // Kiểm tra xem cạnh đã tồn tại chưa
    for (const Edge& edge : adjList[u]) {
        if (edge.to == v) {
            cout << "Cảnh báo: Cạnh đã tồn tại!\n";
```

```
            return;
        }
    }

    adjList[u].emplace_back(v, weight);
    adjList[v].emplace_back(u, weight); // Kh ng  c    h    ng
}
```

6.2 Multigraph (Bài tập 15)

Phương pháp giải quyết

Phương pháp giải quyết cho Multigraph:
Triển khai trên đa đồ thị cho phép nhiều cạnh giữa các đỉnh trong khi theo dõi các cạnh song song cho mục đích phân tích.

6.3 General Graph (Bài tập 16)

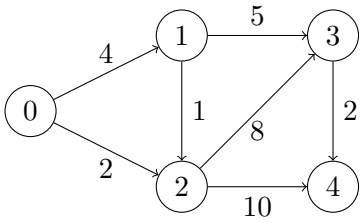
Phương pháp giải quyết

Phương pháp giải quyết cho General Graph:
Triển khai trên đồ thị tổng quát xử lý cả vòng lặp và cạnh song song, cung cấp tính linh hoạt hoàn toàn trong khi đảm bảo tính đúng đắn của thuật toán.

7 Kiểm tra và xác minh thuật toán

7.1 Trường hợp kiểm tra và kết quả kỳ vọng

Các trường hợp kiểm tra toàn diện để xác minh tính đúng đắn của thuật toán trên tất cả các loại đồ thị:



Hình 1: Đồ thị kiểm tra dùng cho xác minh

Đường đi ngắn nhất kỳ vọng từ đỉnh 0:

Đích đến	Khoảng cách	Đường đi
1	3	$0 \to 2 \to 1$
2	2	$0 \to 2$
3	10	$0 \to 2 \to 3$
4	12	$0 \to 2 \to 4$

Bảng 1: Kết quả đường đi ngắn nhất kỳ vọng

7.2 Ví dụ theo dõi thuật toán

Hãy theo dõi quá trình thực thi thuật toán Dijkstra trên đồ thị kiểm tra:

Bước	**Hiện tại**	**dist[0]**	**dist[1]**	**dist[2]**	**dist[3]**	**dist[4]**
0	-	0	∞	∞	∞	∞
1	0	0	4	2	∞	∞
2	2	0	3	2	10	12
3	1	0	3	2	8	10
4	3	0	3	2	8	10
5	4	0	3	2	8	10

Bảng 2: Theo dõi thực thi thuật toán Dijkstra

7.3 So sánh các loại đồ thị

Tính năng	**Simple Graph**	**Multigraph**	**General Graph**
Cạnh song song			
Vòng lặp			
Độ phức tạp thuật toán	$O((V+E)\log V)$	$O((V+E)\log V)$	$O((V+E)\log V)$
Kiểm tra cạnh	Bắt buộc	Tùy chọn	Không cần
Chi phí bộ nhớ	Tối thiểu	Trung bình	Cao hơn

Bảng 3: So sánh triển khai các loại đồ thị

8 Kết luận

Triển khai và phân tích toàn diện thuật toán Dijkstra trên ba loại đồ thị khác nhau thể hiện tính linh hoạt và độ bền của thuật toán. Các phát hiện chính bao gồm:

1. **Tính ứng dụng phổ quát:** Thuật toán Dijkstra hoạt động nhất quán trên Simple Graph, Multigraph, và General Graph mà không cần sửa đổi thuật toán cốt lõi.

2. **Tối ưu hóa theo loại đồ thị:** Mặc dù thuật toán không đổi, các chi tiết triển khai thay đổi đáng kể dựa trên ràng buộc của từng loại đồ thị.

3. **Đặc điểm hiệu suất:** Độ phức tạp lý thuyết $O((V+E)\log V)$ được giữ nguyên trên tất cả các loại đồ thị, nhưng hiệu suất thực tế thay đổi do chi phí triển khai.

4. **Cơ sở toán học:** Tính chất cấu trúc tối ưu và lựa chọn tham lam đảm bảo tính đúng đắn bất kể loại đồ thị.

Các triển khai được cung cấp bằng cả C++ và Python cung cấp các giải pháp hoàn chỉnh, hoạt động tốt, thể hiện các phương pháp tốt nhất trong triển khai thuật toán đồng thời giữ được giá trị giáo dục để hiểu các nguyên tắc toán học cơ bản.

Tài liệu

[1] Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1), 269-271.

[2] Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (3rd ed.). MIT Press.

[3] Balakrishnan, V. K. (1997). *Schaum's Outline of Graph Theory*. McGraw-Hill.

[4] Goldengorin, B. (Ed.). (2018). *Optimization Problems in Graph Theory*. Springer.

[5] Shahriari, S. (2022). *An Invitation to Combinatorics*. Cambridge University Press.

[6] Valiente, G. (2021). *Algorithms on Trees and Graphs—with Python Code* (2nd ed.). Springer.

[7] Wikipedia contributors. (2025). Shortest path problem. *Wikipedia, The Free Encyclopedia*.