Report

Kruskal’s Algorithm for finding Minimum Spanning Tree

Họ và tên: Nguyễn Đăng Khánh

MSSV: 20224440

Mã lớp học: 152455

Tóm tắt nội dung

Bài toán tìm cây khung nhỏ nhất (Minimum Spanning Tree - MST) là một bài toán cơ bản trong lý thuyết đồ thị, trong đó cây khung nhỏ nhất của một đồ thị liên thông là cây con bao gồm tất cả các đỉnh và một tập hợp các cạnh sao cho không có chu trình và tổng trọng số của các cạnh là nhỏ nhất. Để giải bài toán này, thuật toán Kruskal sử dụng phương pháp "Tham lam" (greedy) với ý tưởng chính là sắp xếp các cạnh của đồ thị theo thứ tự tăng dần của trọng số. Sau đó, lần lượt thêm các cạnh có trọng số nhỏ nhất vào cây khung, miễn là việc thêm cạnh không tạo chu trình. Quá trình sẽ dừng lại khi cây khung chứa đúng $V-1$ cạnh, trong đó V là số đỉnh của đồ thị.

1. Giới thiệu

1.1 Lịch sử ngắn gọn

- Thuật toán Kruskal được đặt theo tên của **Joseph Kruskal**, người đã phát triển thuật toán này vào năm 1956.
- Đây là một trong những thuật toán cơ bản để tìm cây khung nhỏ nhất và được sử dụng rộng rãi trong lý thuyết đồ thị, cũng như các ứng dụng thực tế.

1.2 Các đặc điểm chính

- **Đồ thị đầu vào:**
  - Đồ thị là đồ thị vô hướng có trọng số.
  - Đồ thị phải liên thông để cây khung nhỏ nhất tồn tại (mỗi cặp đỉnh trong đồ thị đều phải có ít nhất một đường đi nối nhau).
- **Đầu ra của bài toán:**
  - Một tập hợp các cạnh của đồ thị tạo thành cây khung nhỏ nhất.

- Tổng trọng số của các cạnh trong cây khung nhỏ nhất.

- **Cây khung nhỏ nhất:** Là cây khung có tổng trọng số của các cạnh là nhỏ nhất trong tất cả các cây khung có thể có của đồ thị.
- **Thuật toán giải bài toán:** Sắp xếp các cạnh theo trọng số và lần lượt chọn các cạnh nhỏ nhất không tạo chu trình

2. Động lực

2.1 Phát biểu Bài toán

Bài toán tìm cây khung nhỏ nhất (Minimum Spanning Tree - MST) yêu cầu tìm một cây khung của đồ thị liên thông có trọng số sao cho tổng trọng số của các cạnh trong cây là nhỏ nhất. Cây khung là một cây con của đồ thị bao gồm tất cả các đỉnh của đồ thị, không chứa chu trình, và có số cạnh bằng V−1, trong đó V V là số đỉnh của đồ thị. Điều kiện tiên quyết để bài toán có lời giải là đồ thị phải liên thông, và các trọng số của các cạnh phải không âm. Mục tiêu của bài toán là đảm bảo chi phí tối thiểu khi kết nối tất cả các đỉnh trong đồ thị.

2.2 Ứng dụng Thực tế

- **Thiết kế mạng lưới:** Sử dụng trong thiết kế mạng cáp quang, mạng điện, hoặc hệ thống đường truyền dữ liệu để giảm thiểu chi phí lắp đặt. Đảm bảo kết nối tất cả các điểm trong mạng lưới với tổng chi phí nhỏ nhất.
- **Quy hoạch giao thông:** Áp dụng để xây dựng hệ thống đường giao thông, đường sắt, hoặc đường ống dẫn dầu. Giảm chi phí xây dựng và bảo trì mà vẫn kết nối tất cả các địa điểm cần thiết.
- **Đồ họa dữ liệu:** Tối ưu hóa việc kết nối các điểm trong không gian 3D. Hỗ trợ xây dựng cấu trúc đồ thị hiệu quả để giảm số lượng cạnh mà vẫn đảm bảo kết nối giữa các điểm.

3. Bối cảnh và Công việc liên quan

Trong bài toán tìm cây khung nhỏ nhất (MST) và thuật toán Kruskal, một số khái niệm và phương pháp có liên quan được áp dụng để tối ưu hóa quá trình giải quyết bài toán:

3.1 Lý thuyết Greedy (Tham lam)

- Thuật toán Kruskal sử dụng phương pháp tham lam, nghĩa là tại mỗi bước, nó chọn cạnh có trọng số nhỏ nhất mà không tạo chu trình.
- Đây là cốt lõi của thuật toán, đảm bảo từng bước đưa ra lựa chọn tối ưu cục bộ để đạt được kết quả tối ưu toàn cục (cây khung nhỏ nhất).

3.2 Union-Find (Disjoint Set Union - DSU):

- Một cấu trúc dữ liệu quan trọng trong thuật toán Kruskal, được sử dụng để kiểm tra xem hai đỉnh của cạnh đang xét có thuộc cùng một tập hợp hay không (kiểm tra chu trình).
- Union-Find giúp quản lý các tập hợp đỉnh và tối ưu hóa bằng Path Compression và Rank Optimization, đảm bảo hiệu suất cao.

3.3 Các ứng dụng chiến lược của thuật toán Kruskal:

- **Lý thuyết tối ưu:** Thuật toán này hoạt động trên giả định rằng việc chọn cạnh nhỏ nhất tại mỗi bước là lựa chọn tối ưu cục bộ và phù hợp với bài toán cây khung nhỏ nhất.
- **Mở rộng bài toán:** Thuật toán có thể được kết hợp với các phương pháp khác, như dynamic programming hoặc lý thuyết trò chơi, khi bài toán liên quan đến việc tìm cây khung tối ưu trong các đồ thị phức tạp hơn.

4. Giao diện (Giả định và Dữ liệu đầu vào)

Trong bài toán tìm cây khung nhỏ nhất (MST) với thuật toán Kruskal, giao diện đầu vào và đầu ra có thể được mô tả như sau:

4.1 Giả định

- Đồ thị đồ thị vô hướng liên thông có trọng số
- Trọng số các cạnh không âm
- Đầu vào gồm thông tin về:
  - Số lượng đỉnh (V)
  - Số lượng cạnh (E)
  - Danh sách các cạnh với thông tin: Đỉnh đầu(u), Đỉnh cuối(v), Trọng số của cạnh (weight).

4.2 Dữ liệu đầu vào

- Số đỉnh và số cạnh của đồ thị:
  - V: Số lượng đỉnh($V \geq 1$)
  - E: Số lượng cạnh($E \geq V-1$ cho đồ thị liên thông)
- Danh sách các cạnh
  - Mỗi cạnh được biểu diễn dưới dạng (u,v,weight) trong đó: u,v là 2 đỉnh của cạnh; weight là trọng số của cạnh, là số nguyên dương hoặc bằng 0

4.3 Dữ liệu đầu ra

➔Danh sách các cạnh thuộc cây khung nhỏ nhất, Tổng trọng số của các cạnh trong cây khung nhỏ nhất

5. Minh họa thuật toán Kruskal

Giả sử đồ thị có:

- V=4 (số đỉnh).
- E=5 (số cạnh).

Danh sách các cạnh (u,v, weight): Edges={(0,1,10),(0,2,6),(0,3,5),(1,3,15),(2,3,4)}

Các bước thực hiện thuật toán Kruskal:

5.1 Sắp xếp các cạnh theo trọng số tăng dần: Sau khi sắp xếp, thứ tự các cạnh là:

(2,3,4),(0,3,5),(0,2,6),(0,1,10),(1,3,15)}

5.2 Khởi tạo Union-Find:

- Mỗi đỉnh ban đầu là một tập hợp riêng: parent=[0,1,2,3]
- Cấp bậc của mỗi tập hợp: rank=[0,0,0,0]

5.3 Xét từng cạnh (theo thứ tự đã sắp xếp):

- Cạnh (2,3,4) :
  - find(2) ≠ find(3) → thêm cạnh vào cây khung.
  - Hợp nhất tập hợp của 2 và 3: parent=[0,1,2,2]
- Cạnh (0,3,5):
  - find(0) ≠ find(3) → thêm cạnh vào cây khung.
  - Hợp nhất tập hợp của 0 và 3: parent=[0,1,0,2]
- Cạnh (0,2,6):
  - find(0) ≠ find(2) → thêm cạnh vào cây khung.
  - Hợp nhất tập hợp của 0 và 2: parent=[0,0,0,2]
- Cạnh (0,1,10):
  - find(0)=find(1) → không thêm cạnh (tạo chu trình).
- Cạnh (1,3,15):
  - Không cần xét vì đã đủ V−1=3 cạnh trong cây khung.

5.4 Kết quả:

Các cạnh thuộc cây khung nhỏ nhất: (MST)

{(2,3,4),(0,3,5),(0,2,6)}

Tổng trọng số: (W_{MST})

4+5+6=15

6. Phân tích

6.1 Công thức

Thuật toán Kruskal hoạt động dựa trên các nguyên tắc sau:

Công thức chính:

- Tổng trọng số của cây khung nhỏ nhất(MST):

$$W_{MST} = \sum_{e \in MST} w(e)$$

- Trong đó:
  - **MST:** Tập hợp các cạnh thuộc cây khung nhỏ nhất.
  - **w(e):** Trọng số của cạnh e
- Điều kiện để thêm cạnh vào MST:
  - Một cạnh e(u,v) chỉ được thêm vào nếu hai đỉnh u và v thuộc hai tập hợp khác nhau.
  - find(u)≠find(v), trong đó **Union-Find** được sử dụng để kiểm tra tập hợp.

6.2 Giả mã

Kruskal(V, E):

1. Sắp xếp danh sách các cạnh E theo trọng số tăng dần.
2. Khởi tạo Union-Find để quản lý tập hợp các đỉnh.
3. MST = {} (Tập hợp các cạnh trong cây khung nhỏ nhất).
4. Tổng trọng số MST = 0.
5. Duyệt qua từng cạnh (u, v, w) trong danh sách E:
   - Nếu find(u) ≠ find(v): (u và v thuộc hai tập hợp khác nhau)
     a. Thêm cạnh (u, v) vào MST.
     b. Hợp nhất hai tập hợp chứa u và v (union(u, v)).
     c. Cập nhật Tổng trọng số MST += w.
   - Nếu MST đã có đủ V-1 cạnh: Dừng lại.
6. Trả về MST và Tổng trọng số MST.

6.3 Tối ưu hóa

a) Sắp xếp danh sách cạnh (Step 1):

- Sử dụng thuật toán sắp xếp nhanh (Quick Sort) hoặc Merge Sort:

➔Độ phức tạp: **O(ElogE),** với E là số cạnh.

b) Kiểm tra và hợp nhất tập hợp (Step 5):

- Path Compression: Giảm chiều cao của cây khi tìm gốc (find), bằng cách gán trực tiếp mỗi đỉnh con về gốc.

➔Độ phức tạp: **O(log∗V),** gần như hằng số.

- Rank Optimization: Luôn gắn cây nhỏ hơn vào gốc của cây lớn hơn để giảm chiều cao cây.

➔Độ phức tạp: **O(log∗V).**

c) Độ phức tạp tổng:

- Thời gian:
  - Sắp xếp cạnh: O(ElogE).
  - Kiểm tra và hợp nhất tập hợp cho E cạnh: O(E·log∗V).
  - Tổng độ phức tạp: O(ElogE) (vì log∗V rất nhỏ).
- Không gian:
  - O(V+E): Dùng để lưu Union-Find và danh sách các cạnh.

7. Kết luận

Bài toán tìm cây khung nhỏ nhất (MST) minh họa rõ ràng cách tối ưu hóa chi phí khi kết nối tất cả các đỉnh trong một đồ thị liên thông. Với thuật toán Kruskal, vấn đề được giải quyết hiệu quả bằng tư duy tham lam (Greedy), kết hợp với cấu trúc dữ liệu Union-Find giúp tối ưu hóa việc kiểm tra chu trình và hợp nhất các tập hợp. Độ phức tạp của thuật toán O(ElogE), nhờ sắp xếp danh sách cạnh và tối ưu hóa Union-Find, đảm bảo rằng ngay cả với đồ thị lớn, thuật toán vẫn hoạt động tốt.

Kruskal không chỉ mang tính ứng dụng thực tiễn cao, như trong thiết kế mạng lưới giao thông, hệ thống điện, hay mạng truyền dữ liệu, mà còn là ví dụ điển hình về cách sử dụng thuật toán tham lam để giải quyết bài toán tối ưu toàn cục. Việc áp dụng thành công tư duy này không chỉ giúp giảm độ phức tạp của bài toán mà còn minh chứng tầm quan trọng của các cấu trúc dữ liệu trong xử lý đồ thị. Qua đó, bài toán MST là một trường hợp điển hình để hiểu rõ hơn về các chiến lược tối ưu và kỹ thuật xử lý đồ thị trong khoa học máy tính.

8. Tài liệu tham khảo

- **Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C.**

Introduction to Algorithms (3rd Edition)

- **GeeksforGeeks**

Kruskal's Minimum Spanning Tree Algorithm