BÀI TẬP TOÁN TỔ HỢP VÀ LÝ THUYẾT ĐỒ THỊ

Giảng viên: Nguyễn Quản Bá Hồng
Sinh viên: Võ Huỳnh Thái Bảo

Ngày 23 tháng 7 năm 2025

Mục lục

1 Bài Toán 1: Ferrers & Ferrers Transpose Diagrams 3

2 Bài Toán 2: Đếm số phân hoạch p(n, k) 4

3 Bài Toán 3: Số phân hoạch tự liên hợp 5

4 Bài Toán 4: Chuyển Đổi Giữa Các Biểu Diễn Đồ Thị và Cây 6

5 Bài Toán 5: Giải Các Bài Tập Đồ Thị (Val21) 8

6 Bài Toán 6: Tree Edit Distance 9

7 Bài Toán 7: Tree Traversal – Duyệt Cây 10

8 Bài Toán 8: Thuật Toán BFS Trên Đồ Thị Đơn Hữu Hạn 11

9 Bài Toán 9: Thuật Toán BFS Trên Multigraph (Đồ Thị Đa Cung) 12

10 Bài Toán 10: Thuật Toán BFS Trên Đồ Thị Tổng Quát 14

11 Bài Toán 11: Thuật Toán DFS Trên Đồ Thị Đơn Hữu Hạn 16

12 Bài Toán 12: Thuật Toán DFS Trên Multigraph (Đồ Thị Đa Cung) 17

13 Bài Toán 13: Thuật Toán DFS Trên Đồ Thị Tổng Quát 19

14 Bài Toán 14: Dijkstra trên Đồ Thị Đơn 21

15 Bài Toán 15: Dijkstra trên Đồ Thị Đa Cung (Multigraph) 22

16 Bài Toán 16: Dijkstra trên Đồ Thị Tổng Quát 23

1 Bài Toán 1: Ferrers & Ferrers Transpose Diagrams

Đồ Án Phân Hoạch Số Nguyên

Phát biểu bài toán

Cho một phân hoạch $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_k) \in \mathbb{N}^k$. Hãy vẽ biểu đồ Ferrers tương ứng và biểu đồ chuyển vị (transpose) của nó. Mỗi phần tử λ_i thể hiện số dấu sao trong dòng thứ i.

Biểu đồ Ferrers

Biểu đồ Ferrers biểu diễn mỗi phần tử của phân hoạch bằng một dòng các dấu *, số lượng dấu * tương ứng với giá trị của phần tử.

Ví dụ: $\lambda = (4, 3, 1)$

Ferrers diagram:

```
****
***
*
```

Biểu đồ Ferrers chuyển vị

Chuyển vị tương ứng là việc lấy cột của biểu đồ Ferrers làm dòng.

Transpose của ví dụ trên:

```
***
***
*
*
```

Thuật toán

Bước 1: Nhập phân hoạch λ

Kiểm tra xem λ có phải là dãy không tăng.

Bước 2: Vẽ biểu đồ Ferrers

Duyệt từng dòng, in ra số lượng dấu * tương ứng với phần tử.

Bước 3: Chuyển vị Ferrers

- Tìm giá trị lớn nhất $m = \max(\lambda)$
- Duyệt từng dòng từ 1 đến m (theo chiều dọc), in dấu * xem xét xem dòng đó có đủ chiều dài hay không?

Chú thích các biến

- `partition` – danh sách chứa phân hoạch
- `max_row` – phần tử lớn nhất, xác định chiều cao transpose
- `rows[i]` – số lượng dấu * trong dòng thứ i

2 Bài Toán 2: Đếm số phân hoạch p(n, k)

Đồ Án Phân Hoạch Số Nguyên

Phát biểu bài toán

Cho hai số nguyên dương $n, k \in \mathbb{N}$. Đếm số phân hoạch của n sao cho phần tử lớn nhất trong mỗi phân hoạch không vượt quá k. Ký hiệu hàm đếm là $p(n, k)$.

Ví dụ: $p(5, 3) = 5$, gồm các phân hoạch: $(3, 2), (3, 1, 1), (2, 2, 1), (2, 1, 1, 1), (1, 1, 1, 1, 1)$

Công thức đệ quy

Công thức đệ quy:

$$p(n, k) = \begin{cases} 1 & \text{if } n = 0 \\ 0 & \text{if } n < 0 \text{ or } k = 0 \\ p(n-k, k) + p(n, k-1) & \text{otherwise} \end{cases}$$

Giải thích:

- $p(n-k, k)$ là số phân hoạch của n có ít nhất một phần tử bằng k
- $p(n, k-1)$ là số phân hoạch của n có tất cả phần tử nhỏ hơn k

Thuật toán (Quy hoạch động)

Khởi tạo mảng hai chiều $dp[n+1][k+1]$, với:

- $dp[i][j]$ lưu giá trị $p(i, j)$
- Gán $dp[0][j] = 1$ với mọi $j \geq 0$
- Duyệt i từ 1 đến n, j từ 1 đến k, cập nhật:
  $dp[i][j] = dp[i-j][j] + dp[i][j-1]$ nếu $i \geq j$
  ngược lại thì:
  $dp[i][j] = dp[i][j-1]$

Chú thích các biến

- n, k – đầu vào bài toán
- $dp[i][j]$ – số phân hoạch của i với phần tử lớn nhất $\leq j$

So sánh

Có thể so sánh $p(n)$ (tổng phân hoạch của n) với:

$$p(n) = \sum_{k=1}^{n} p(n, k)$$

3 Bài Toán 3: Số phân hoạch tự liên hợp

Đồ Án Phân Hoạch Số Nguyên

Phát biểu bài toán

Cho $n, k \in \mathbb{N}$. Hãy thực hiện các yêu cầu sau:

(a) Đếm số phân hoạch tự liên hợp của n có đúng k phần tử, ký hiệu $p_k^{self}(n)$

(b) Với k bất kỳ, in ra tất cả các phân hoạch tự liên hợp của n

(c) Thiết lập công thức đệ quy truy hồi tính $p_k^{self}(n)$

Định nghĩa: Phân hoạch tự liên hợp

Phân hoạch $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_r)$ là tự liên hợp nếu biểu đồ Ferrers của nó bằng chính chuyển vị của nó. Ví dụ: $(3, 1, 1)$ và $(5, 3, 1)$ là phân hoạch tự liên hợp.

Công thức đếm (derivation)

Số phân hoạch tự liên hợp của n bằng số tập các số nguyên dương lẻ phân biệt sao cho tổng của chúng là n.

Công thức tổng quát:

$$p^{self}(n) = \text{số phân hoạch của } n \text{ thành tổng của các số lẻ phân biệt}$$

Công thức đệ quy

Gọi $dp[i][j]$ là số phân hoạch của i dùng các số lẻ phân biệt $\leq j$. Ta có:
$dp[i][j] = dp[i][j-2] + dp[i-j][j]$ nếu $j \leq i$
Ngược lại:
$dp[i][j] = dp[i][j-2]$
Với khởi tạo: $dp[0][j] = 1$, $dp[i][0] = 0$ với $i > 0$

Chú thích các biến

- n – tổng cần phân hoạch
- $dp[i][j]$ – số phân hoạch của i dùng số lẻ phân biệt $\leq j$
- j tăng theo bước 2 (chỉ số lẻ)

4 Bài Toán 4: Chuyển Đổi Giữa Các Biểu Diễn Đồ Thị và Cây

Đồ Án 4: Duyệt Đồ Thị & Cây

Phát biểu bài toán

Viết chương trình C/C++, Python để chuyển đổi giữa:

- 4 dạng biểu diễn đồ thị:
  (a) adjacency matrix
  (b) adjacency list
  (c) extended adjacency list (với trọng số)
  (d) adjacency map
- 3 dạng biểu diễn cây:
  (a) array of parents
  (b) first-child next-sibling
  (c) graph-based tree

  Tổng cộng có $\binom{4}{2} \cdot 2 + \binom{3}{2} \cdot 2 = 42$ chuyển đổi.

Phân tích tổng quát

- Đồ thị: Dùng cấu trúc lớp 'Graph' để chứa dữ liệu dưới các dạng khác nhau.
- Cây: Sử dụng cây gốc không có chu trình, áp dụng các phương pháp duyệt và ánh xạ.

Thuật toán

Mỗi chuyển đổi cần:

1. Hàm đọc từ định dạng A
2. Hàm sinh định dạng B

Ví dụ: để chuyển từ adjacency matrix → adjacency list:

1. Duyệt từng dòng i
2. Với mỗi cột j nếu $matrix[i][j] == 1$ thì thêm j vào $list[i]$

Chú thích các biến số

- `matrix[i][j]`: lưu 1 nếu có cạnh từ i đến j
- `adjList[i]`: vector danh sách kề
- `parent[i]`: cây biểu diễn bằng mảng cha

Ví dụ minh họa

Cho đồ thị 4 đỉnh:

$$A = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix}$$

$$\Rightarrow \text{adjList} = \{0 : [1, 3], 1 : [0, 2], 2 : [1, 3], 3 : [0, 2]\}$$

5 Bài Toán 5: Giải Các Bài Tập Đồ Thị (Val21)

Đồ Án 4: Duyệt Đồ Thị & Cây

Phát biểu bài toán

Giải các bài toán cơ bản từ mục 1.1 đến 1.6 và bài tập 1.1 đến 1.10 theo tài liệu Val21. Chúng bao gồm các thao tác với đồ thị đơn như:

- Tính bậc của các đỉnh
- Kiểm tra đồ thị vô hướng / có hướng
- Kiểm tra liên thông
- Kiểm tra đồ thị có phải cây hay không
- Đếm số thành phần liên thông
- Duyệt đồ thị bằng DFS / BFS

Ý tưởng thuật toán

1. **Bậc đỉnh:** Với đồ thị vô hướng, bậc đỉnh là số lượng đỉnh kề. Với đồ thị có hướng, dùng in-degree và out-degree.
2. **Liên thông:** Dùng DFS để kiểm tra xem có thể duyệt hết tất cả các đỉnh từ một đỉnh gốc.
3. **Cây:** Đồ thị là cây nếu:
   - Liên thông
   - Không có chu trình
   - Có đúng $n - 1$ cạnh với n đỉnh
4. **DFS/BFS:** Tiêu chuẩn để duyệt toàn bộ đồ thị, áp dụng để kiểm tra tính liên thông hoặc in thứ tự duyệt.

Chú thích các biến số

- `n`: số lượng đỉnh trong đồ thị
- `adj`: danh sách kề, kiểu `adj[i]` là danh sách các đỉnh kề với đỉnh i
- `visited[i]`: mảng đánh dấu đỉnh i đã được duyệt trong DFS/BFS
- `inDeg[i]`: số lượng cung đi vào đỉnh i (đồ thị có hướng)
- `outDeg[i]`: số lượng cung đi ra từ đỉnh i
- `parent[i]`: đỉnh cha của i trong DFS tree
- `component_count`: số thành phần liên thông
- `isTree`: cờ kiểm tra đồ thị có phải là cây hay không

6 Bài Toán 6: Tree Edit Distance

Đồ Án 4: Duyệt Đồ Thị & Cây

Phát biểu bài toán

Cho hai cây có gốc T_1 và T_2. Tính số phép biến đổi tối thiểu cần thiết để biến T_1 thành T_2 bằng ba thao tác:

- Insert (thêm nút)
- Delete (xóa nút)
- Rename (đổi nhãn)

Mục tiêu

Tìm khoảng cách edit $TED(T_1, T_2)$ sao cho tổng chi phí nhỏ nhất.

Thuật toán áp dụng

(a) Brute-force / Backtracking: thử tất cả phép biến đổi

(b) Branch-and-bound: loại bỏ nhánh có chi phí tạm tính > best hiện tại

(c) Divide and Conquer: chia cây thành các subtree nhỏ

(d) Dynamic Programming: giải bài toán con bằng quy hoạch động

Chi tiết thuật toán (Zhang Shasha)

- Định nghĩa post-order cho mỗi node i trong cây
- Hàm $ted(i, j)$ = chi phí biến subtree rooted tại i thành subtree rooted tại j
- Dựng bảng $dp[i][j]$ với công thức:

$$dp[i][j] = \begin{cases} dp[i-1][j-1] & \text{nếu } label(i) = label(j) \\ \min \begin{cases} dp[i-1][j] + cost(\text{delete}) \\ dp[i][j-1] + cost(\text{insert}) \\ dp[i-1][j-1] + cost(\text{rename}) \end{cases} & \text{ngược lại} \end{cases}$$

Chú thích các biến số

- T_1, T_2: hai cây gốc
- $dp[i][j]$: khoảng cách edit từ subtree i của T_1 tới subtree j của T_2
- $label[i]$: nhãn (tên) của node i
- $l[i], r[i]$: chỉ số preorder hoặc postorder của node i
- $cost\_ins, cost\_del, cost\_ren$: chi phí các thao tác

7 Bài Toán 7: Tree Traversal – Duyệt Cây

Đồ Án 4: Duyệt Đồ Thị & Cây

Phát biểu bài toán

Cho một cây gốc T, viết chương trình C/C++, Python để duyệt cây theo 4 cách:

(a) Preorder traversal (tiền thứ tự): Duyệt node $\rightarrow$ con trái $\rightarrow$ con phải

(b) Postorder traversal (hậu thứ tự): Duyệt con trái $\rightarrow$ con phải $\rightarrow$ node

(c) Top-down traversal: Duyệt theo tầng từ gốc xuống lá (BFS)

(d) Bottom-up traversal: Duyệt theo tầng từ lá lên gốc

Ý tưởng

- Preorder và Postorder: dùng đệ quy truyền thống
- Top-down: dùng queue để duyệt theo tầng (BFS)
- Bottom-up: dùng BFS để lưu tầng, sau đó in ngược lại

Chú thích các biến số

- `Node`: class chứa label và children
- `tree`: cây gốc cần duyệt
- `result`: mảng kết quả duyệt
- `queue`: dùng trong top-down duyệt theo tầng
- `levels`: danh sách lưu từng tầng

8 Bài Toán 8: Thuật Toán BFS Trên Đồ Thị Đơn Hữu Hạn

Đồ Án 5.1: Breadth-first Search

Phát biểu bài toán

Cho một đồ thị đơn hữu hạn $G = (V, E)$ (finite simple graph). Viết chương trình C/C++, Python để thực hiện thuật toán tìm kiếm theo chiều rộng (Breadth-First Search - BFS) trên G.

Ý tưởng

- Duyệt đồ thị bắt đầu từ đỉnh nguồn s
- Dùng hàng đợi (queue) để duyệt từng đỉnh theo thứ tự vào-trước-ra-trước
- Đánh dấu các đỉnh đã thăm để tránh lặp

Thuật toán BFS (pseudocode)

BFS(G, s):
Tạo hàng đợi Q
$visited[s] \leftarrow true$
Q.enqueue(s)
while Q không rỗng **do**
  $u \leftarrow Q$.dequeue()
  xử lý đỉnh u
  for mỗi đỉnh v kề với u **do**
    if not $visited[v]$ **then**
      $visited[v] \leftarrow true$
      Q.enqueue(v)
    end if
  end for
end while

Chú thích các biến số

- G: đồ thị đầu vào, dưới dạng danh sách kề (adjacency list)
- s: đỉnh bắt đầu BFS
- $visited[i]$: mảng boolean đánh dấu đỉnh i đã được thăm
- `queue`: hàng đợi FIFO lưu các đỉnh đang chờ duyệt
- `res`: danh sách thứ tự các đỉnh được duyệt

9 Bài Toán 9: Thuật Toán BFS Trên Multigraph (Đồ Thị Đa Cung)

Đồ Án 5.1: Breadth-first Search

Phát biểu bài toán

Cho một đồ thị đa cung hữu hạn $G = (V, E)$, trong đó có thể tồn tại nhiều cạnh nối giữa cùng một cặp đỉnh (u, v). Yêu cầu: Triển khai thuật toán tìm kiếm theo chiều rộng (Breadth-First Search – BFS) trên G.

Đặc điểm của multigraph

- Có thể có nhiều hơn một cạnh giữa 2 đỉnh u và v
- Có thể có cạnh tự khép (self-loop), ví dụ (u, u)
- Danh sách kề có thể chứa nhiều lần cùng một đỉnh kề

Ý tưởng

- Duyệt theo chiều rộng như đồ thị đơn
- Để tránh duyệt lặp qua nhiều cạnh trùng nhau, cần dùng mảng $visited[]$ để đánh dấu đã thăm
- Mỗi đỉnh chỉ được duyệt đúng một lần, bỏ qua các cạnh trùng nếu đỉnh kề đã thăm

Thuật toán BFS (pseudocode)

BFS_Multigraph(G, s):
$visited[v] \leftarrow False$ với mọi $v \in V$
$Q \leftarrow$ hàng đợi rỗng
$visited[s] \leftarrow True$
enqueue(Q, s)
while Q không rỗng **do**
  $u \leftarrow$ dequeue(Q)
  xử lý đỉnh u
  for mỗi $v \in adj[u]$ **do**
    {cho phép lặp} **if** not $visited[v]$ **then**
    $visited[v] \leftarrow True$
    enqueue(Q, v)
   end if
  end for
end while

Chú thích các biến số

- G: đồ thị đa cung, biểu diễn bằng danh sách kề có thể chứa trùng
- $adj[u]$: danh sách các đỉnh kề với u, có thể chứa v nhiều lần
- s: đỉnh bắt đầu BFS

- $visited[v]$: boolean đánh dấu đỉnh v đã được duyệt
- Q: hàng đợi FIFO dùng để duyệt BFS
- `res`: danh sách kết quả duyệt

Giải thích xử lý trùng cạnh

- Tại vì một đỉnh v có thể xuất hiện nhiều lần trong $adj[u]$, ta chỉ xét $visited[v]$ đúng một lần
- Ví dụ: nếu $adj[0] = [1, 1, 2]$, thì đỉnh 1 chỉ được duyệt 1 lần duy nhất
- Điều này sẽ giúp BFS vẫn có độ phức tạp $O(V + E)$ với E là tổng số cung (bao gồm trùng cạnh)

10 Bài Toán 10: Thuật Toán BFS Trên Đồ Thị Tổng Quát

Đồ Án 5.1: Breadth-first Search

Phát biểu bài toán

Cho một đồ thị tổng quát $G = (V, E)$, không giả định đơn, đa cung hay hướng/vô hướng. Viết thuật toán Breadth-First Search (BFS) có khả năng hoạt động chính xác trên mọi loại đồ thị.

Giả thuyết

- Đồ thị có thể có nhiều thành phần liên thông
- Đỉnh có thể có self-loop (tức là $(v, v) \in E$)
- Có thể có nhiều cạnh nối giữa một cặp đỉnh
- Có thể là đồ thị hữu hướng hoặc vô hướng
- Có thể rỗng

Ý tưởng

- Sử dụng BFS tiêu chuẩn cho từng thành phần liên thông
- Duyệt BFS từ mọi đỉnh chưa được thăm để bao phủ toàn bộ đồ thị
- Sử dụng mảng $visited[]$ để tránh lặp chu kỳ, đa cung hoặc self-loop

Thuật toán BFS tổng quát (pseudocode)

General_BFS(G):
$n \leftarrow$ số lượng đỉnh
$visited[v] \leftarrow False\ \forall v \in V$
for mỗi đỉnh u từ 0 đến $n-1$ **do**
  if not $visited[u]$ **then**
    $Q \leftarrow$ hàng đợi mới
    enqueue(Q, u)
    $visited[u] \leftarrow True$
    while Q không rỗng **do**
      $v \leftarrow$ dequeue(Q)
      xử lý v
      for mỗi đỉnh $w \in adj[v]$ **do**
        if not $visited[w]$ **then**
          $visited[w] \leftarrow True$
          enqueue(Q, w)
        end if
      end for
    end while
  end if
end for

Chú thích các biến số

- G: đồ thị tổng quát, có thể có hướng, đa cung, self-loop
- $adj[v]$: danh sách các đỉnh kề với v (có thể trùng)
- $visited[v]$: boolean kiểm tra đã duyệt đỉnh v
- Q: hàng đợi BFS dùng cho từng thành phần liên thông
- `res`: danh sách kết quả BFS từ từng component

Xử lý bài toán

- Với self-loop: (v, v) không ảnh hưởng nếu đã kiểm tra $visited[v]$ đúng cách
- Với multigraph: dù có nhiều cạnh trùng, BFS chỉ duyệt 1 lần
- Với đồ thị không liên thông: dùng BFS trên từng thành phần

11 Bài Toán 11: Thuật Toán DFS Trên Đồ Thị Đơn Hữu Hạn

Đồ Án 5.2: Depth-first Search

Phát biểu bài toán

Cho đồ thị đơn hữu hạn $G = (V, E)$ (finite simple graph). Yêu cầu: Triển khai thuật toán duyệt theo chiều sâu (Depth-First Search – DFS) bắt đầu từ một đỉnh s.

Ý tưởng

- DFS đi càng sâu càng tốt trước khi quay lại duyệt các đỉnh còn lại
- Sử dụng đệ quy hoặc stack để triển khai
- Trên đồ thị đơn, mỗi cạnh tồn tại một lần duy nhất → DFS sẽ duyệt tối đa $O(V + E)$

Thuật toán DFS (pseudocode)

DFS(G, u):
$visited[u] \leftarrow True$
xử lý đỉnh u
for mỗi đỉnh $v \in adj[u]$ **do**
  if not $visited[v]$ **then**
    DFS(G, v)
  end if
end for

Chú thích các biến số

- G: đồ thị đầu vào, là đồ thị đơn
- $adj[u]$: danh sách kề của đỉnh u
- $visited[u]$: đánh dấu đỉnh đã được duyệt
- u: đỉnh bắt đầu DFS
- `res`: danh sách kết quả thứ tự duyệt

Đặc điểm đồ thị đơn

- Mỗi cặp đỉnh chỉ có nhiều nhất một cạnh
- Không có cạnh tự khép (u, u)
- Đồ thị vô hướng (auto mặc định)

12 Bài Toán 12: Thuật Toán DFS Trên Multigraph (Đồ Thị Đa Cung)

Đồ Án 5.2: Depth-first Search

Phát biểu bài toán

Cho một đồ thị đa cung $G = (V, E)$, trong đó có thể tồn tại nhiều cạnh giữa cùng một cặp đỉnh (u, v). Yêu cầu: Triển khai thuật toán tìm kiếm theo chiều sâu (Depth-First Search – DFS) trên G.

Đặc điểm của multigraph

- Có thể tồn tại nhiều cạnh giữa một cặp đỉnh
- Có thể tồn tại cạnh tự khép (u, u)
- Danh sách kề có thể chứa trùng lặp các đỉnh

Ý tưởng thuật toán

- DFS được triển khai tương tự như với đồ thị đơn
- Mỗi đỉnh của một đồ thị được duyệt đúng một lần, điều này vẫn đúng bất kể đồ thị có chứa các cạnh bội (cạnh trùng) hay không.
- Dùng mảng $visited[]$ để ngăn việc lặp lại duyệt đỉnh

Thuật toán DFS (pseudocode)

```
DFS_Multigraph(G, u):
visited[u] ← True
xử lý đỉnh u
for mỗi v ∈ adj[u] do
    {có thể trùng} if not visited[v] then
    DFS_Multigraph(G, v)
  end if
end for
```

Chú thích các biến số

- G: đồ thị đa cung, biểu diễn dưới dạng danh sách kề (có thể chứa trùng)
- $adj[u]$: danh sách các đỉnh kề với u (có thể có lặp)
- $visited[v]$: boolean kiểm tra đỉnh v đã được duyệt chưa
- u: đỉnh hiện tại trong DFS
- `res`: danh sách các đỉnh được duyệt theo thứ tự

Xử lý cạnh trùng và self-loop

- Cạnh trùng: nếu $adj[u] = [v, v, v]$ thì DFS vẫn chỉ gọi một lần cho v
- Cạnh tự khép (u, u): sẽ không gây ra vòng lặp vô hạn nếu kiểm tra $visited[u]$ đúng

13 Bài Toán 13: Thuật Toán DFS Trên Đồ Thị Tổng Quát

Đồ Án 5.2: Depth-first Search

Phát biểu bài toán

Cho một đồ thị tổng quát $G = (V, E)$, không giới hạn kiểu đồ thị. Yêu cầu: Viết thuật toán DFS để duyệt toàn bộ đồ thị, bao gồm cả các thành phần rời rạc, cạnh trùng, cạnh tự khép và đồ thị có hướng hoặc vô hướng.

Tính chất đồ thị tổng quát

- Có thể là đồ thị vô hướng hoặc hữu hướng
- Cho phép nhiều cạnh giữa cùng một cặp đỉnh (đa cung)
- Cho phép self-loop (u, u)
- Có thể gồm nhiều thành phần liên thông
- Có thể rỗng

Ý tưởng thuật toán

- Khởi tạo mảng $visited[]$ cho tất cả các đỉnh
- Duyệt DFS từ từng đỉnh chưa thăm → đảm bảo bao phủ toàn bộ đồ thị (gồm nhiều thành phần)
- Mỗi đỉnh chỉ được duyệt đúng một lần, bất kể có bao nhiêu cạnh đi tới nó

Thuật toán DFS tổng quát (pseudocode)

DFS_Component(G, u):
$visited[u] \leftarrow True$
xử lý u
for mỗi $v \in adj[u]$ **do**
  if not $visited[v]$ **then**
    DFS_Component(G, v)
  end if
end for
General_DFS(G):
$visited[v] \leftarrow False$ với mọi $v \in V$
for mỗi đỉnh u **do**
  if not $visited[u]$ **then**
    DFS_Component(G, u)
  end if
end for

Chú thích các biến số

- G: đồ thị tổng quát (đa cung, self-loop, rời rạc)
- $adj[u]$: danh sách các đỉnh kề với u (có thể trùng)
- $visited[u]$: true nếu đã thăm đỉnh u
- `DFS_Component`: hàm xử lý một thành phần liên thông
- `General_DFS`: vòng lặp chính để bao phủ toàn bộ đồ thị

Xử lý các trường hợp đặc biệt

- Cạnh trùng (multiedge): $visited[]$ đảm bảo duyệt đúng 1 lần
- Self-loop: DFS không gọi lại chính nó nếu đã được đánh dấu
- Disconnected graph: Mỗi thành phần liên thông được duyệt riêng

14 Bài Toán 14: Dijkstra trên Đồ Thị Đơn

Phát biểu bài toán

Cho đồ thị đơn $G = (V, E)$ với trọng số không âm. Hãy cài đặt thuật toán Dijkstra để tìm đường đi ngắn nhất từ đỉnh nguồn s đến các đỉnh còn lại.

Ý tưởng thuật toán

- Khởi tạo khoảng cách ban đầu là ∞ (vô cực), riêng đỉnh nguồn s là 0.
- Sử dụng hàng đợi ưu tiên để chọn đỉnh u có khoảng cách nhỏ nhất.
- Với mỗi đỉnh v kề với u, nếu $dist[v] > dist[u] + w(u, v)$ thì cập nhật.

Thuật toán Dijkstra

Input: Đồ thị $G = (V, E)$, trọng số không âm, đỉnh bắt đầu s
Khởi tạo $dist[v] \leftarrow \infty$ với mọi $v \in V$, $dist[s] \leftarrow 0$
Sử dụng hàng đợi ưu tiên Q
while Q không rỗng **do**
    $u \leftarrow$ đỉnh có $dist[u]$ nhỏ nhất trong Q
    for all đỉnh v kề với u **do**
        if $dist[v] > dist[u] + w(u, v)$ **then**
            $dist[v] \leftarrow dist[u] + w(u, v)$
        end if
    end for
end while

15 Bài Toán 15: Dijkstra trên Đồ Thị Đa Cung (Multigraph)

Phát biểu bài toán

Cho đồ thị đa cung $G = (V, E)$ với trọng số không âm. Có thể có nhiều cung giữa một cặp đỉnh (u, v). Hãy cài đặt thuật toán Dijkstra để tìm đường đi ngắn nhất từ một đỉnh nguồn s.

Lưu ý

- Với mỗi cặp (u, v) có thể có nhiều trọng số khác nhau.
- Cần xét tất cả các cạnh giữa u và v khi cập nhật khoảng cách.

Thuật toán Dijkstra (Multigraph)

Input: Đồ thị $G = (V, E)$ với đa cung, đỉnh bắt đầu s
Khởi tạo $dist[v] \leftarrow \infty$ với mọi $v \in V$, $dist[s] \leftarrow 0$
Khởi tạo hàng đợi ưu tiên Q
while Q không rỗng **do**
  $u \leftarrow$ đỉnh có $dist[u]$ nhỏ nhất
  for all mỗi cạnh (u, v, w) **do**
    if $dist[v] > dist[u] + w$ **then**
      $dist[v] \leftarrow dist[u] + w$
    end if
  end for
end while

Độ phức tạp

Không thay đổi: $O((V + E) \log V)$ với Priority Queue.

16 Bài Toán 16: Dijkstra trên Đồ Thị Tổng Quát

Phát biểu bài toán

Cho đồ thị tổng quát $G = (V, E)$ có thể là:

- Có hướng hoặc vô hướng
- Có thể có đa cung (nhiều cạnh giữa cùng một cặp đỉnh)
- Có thể có self-loop (cạnh từ đỉnh đến chính nó)

Hãy cài đặt thuật toán Dijkstra để tìm đường đi ngắn nhất từ một đỉnh s đến các đỉnh còn lại.

Lưu ý đặc biệt

- Dijkstra không áp dụng cho cạnh trọng số âm.
- Self-loop không ảnh hưởng kết quả vì không làm thay đổi $dist[u]$.
- Đa cung vẫn xử lý bình thường: luôn chọn trọng số nhỏ nhất khi cập nhật.

Thuật toán Dijkstra Tổng Quát

Input: Đồ thị tổng quát $G = (V, E)$ với trọng số không âm, nguồn s
$dist[v] \leftarrow \infty$, $dist[s] \leftarrow 0$
Priority Queue $Q \leftarrow \{(0, s)\}$
while Q không rỗng **do**
  lấy (d_u, u) từ Q
  for all cạnh (u, v, w) **do**
    if $dist[v] > d_u + w$ **then**
      $dist[v] \leftarrow d_u + w$, thêm $(dist[v], v)$ vào Q
    end if
  end for
end while

Tài liệu

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

[2] Boris Goldengorin. *Optimization Problems in Graph Theory*. Springer, 2018.

[3] Shahriar Shahriari. *An Invitation To Combinatorics*. Cambridge University Press, 2022.

[4] Gabriel Valiente. *Algorithms on Trees and Graphs with Python Code*. Online Lecture Notes, 2021.