Lab 3: Phân hoạch đồ thị

Phương pháp toán cho Trí tuệ nhân tạo

Nguyễn Bảo Long, Nguyễn Thị Thu Hằng

29/03/2023

Contents

1 Thông tin chung

- Thành viên:
  - Nguyễn Bảo Long - 22C11065
  - Nguyễn Thị Thu Hằng - 22C15027
- Bảng phân công công việc:

Công việc	Người thực hiện
Viết hàm và tài liệu cho thuật toán `Kernighan-Lin`	Bảo Long
Viết hàm và tài liệu cho thuật toán `Kosaraju`	Thu Hằng

2 Phát biểu bài toán

- Cho trước đồ thị $G = (V, E)$, trong đó:
  - V là tập hợp chứa n đỉnh.
  - E là tập cạnh.
- Tiêu chí phân hoạch: (1) - Phân hoạch dựa trên trọng số các cạnh giữa các phân hoạch; (2) - Phân hoạch để tìm thành phần liên thông.
- **Tiêu chí (1)**. Xét bài toán phân hoạch cân bằng (k, v), mục tiêu của bài toán là phân hoạch đồ thị G thành k thành phần $V_1, V_2, ..., V_k$ không giao nhau, mỗi thành phần chứa tối đa $v \times \frac{n}{k}$ đỉnh, sao cho tổng trọng số của các cạnh nối giữa các phân hoạch là nhỏ nhất. Trong trường hợp đồ thị không có trọng số, thuật toán sẽ cực tiểu số lượng cạnh nối giữa các phân hoạch.
- **Tiêu chí (2)**. Phân hoạch đồ thị thành các thành phần sao cho các thành phần này liên thông mạnh. Nếu mỗi thành phần liên thông mạnh được co lại thành một đỉnh, thì đồ thị sẽ trở thành một đồ thị có hướng không có chu trình. Vì vậy, tùy vào mục đích, chúng ta vẫn có thể sử dụng các thuật toán tìm thành phần liên thông mạnh để phân hoạch đồ thị.
- Ứng với mỗi tiêu chí sẽ là các thuật toán được trình bày trong phần **Các thuật toán phân hoạch đồ thị**.

3 Cấu trúc chương trình

- Cấu trúc source code được sử dụng lại từ Lab 2 và bổ sung thêm các thuật toán phân hoạch đồ thị vào lớp `Graph` như hình dưới:

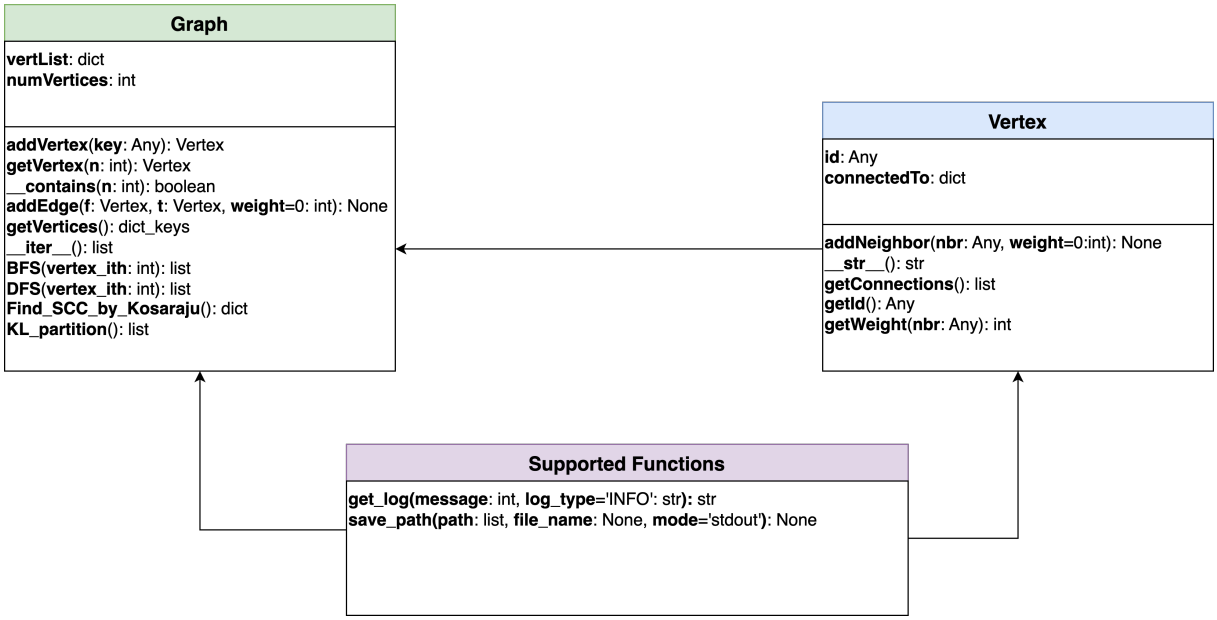

Figure 1: Tổ chức dữ liệu đồ thị.

4 Các thuật toán phân hoạch đồ thị

4.1 Tiêu chí (1) - Thuật toán `Kernighan–Lin`

- Ý tưởng: Thuật toán được đề xuất vào năm 1970, thực hiện phân hoạch tham lam. Cụ thể:
  - Thuật toán chia ngẫu nhiên các đỉnh thành hai nhóm bằng nhau (V_1, V_2) ($|V_1| = |V_2| = \frac{n}{2}$).
  - Thuật toán sẽ hoán đổi các cặp đỉnh của hai nhóm để sinh ra hai nhóm mới (V_1', V_2') sao cho $|V_1'| = |V_2'| = \frac{n}{2}$ và chi phí của phân hoạch mới phải nhỏ hơn hoặc bằng chi phí của phân hoạch cũ.
  - Thuật toán lặp lại quá trình hoán đổi các cặp đỉnh cho đến khi đã lặp đủ số lần quy định trước hoặc cho đến khi đạt được cực tiểu cục bộ (không hoán đổi được cặp đỉnh nào cho chi phí phân hoạch nhỏ hơn).
- Kí hiệu:
  - Chi phí phân hoạch: Tổng trọng số của các cạnh nối các phân hoạch.
  - Với mỗi đỉnh v trong đồ thị, gọi E_v, I_v lần lượt là external cost và internal cost của đỉnh v (được tính bằng cách lấy tổng trọng số của các đỉnh nối từ v đến các đỉnh nằm ngoài/trong phân hoạch mà v đang thuộc về). Giá trị D_v của đỉnh v là:

$$D_v = E_v - I_v$$

  - Gọi $gain$ là chi phí giảm sút sau khi hoán đổi 2 đỉnh $v_1 \in V_1$ và $v_2 \in V_2$. Quy ước $w_{1,2} = 0$ nếu v_1 không liên kết với v_2. $gain$ được tính như sau:

$$gain = D_1 + D_2 - 2w_{1,2}$$

- Thuật toán:
  - Khởi tạo ngẫu nhiên 2 nhóm A, B có cùng kích thước.
  - Lặp lại các bước:
    - Tính giá trị D cho từng đỉnh trong mỗi nhóm
    - Tìm các cặp đỉnh thuộc 2 nhóm A, B có $gain$ lớn nhất
    - Hoán đổi 2 đỉnh và cập nhật lại các giá trị D.
    - Dừng thuật toán khi $gain < 0$
- Nhận xét: Thuật toán hoán đổi một cách tham lam (thể hiện ở bước chọn $gain$ lớn nhất). Do đó, có thể dễ dàng rơi vào các lời giải cục bộ tùy theo trạng thái khởi tạo ngẫu nhiên ban đầu. Ví

dụ, khi thực hiện phân hoạch trên đồ thị hình 2, có thể sinh ra 2 output (lời giải cục bộ và lời giải toàn cục) như hình 3 và 4.

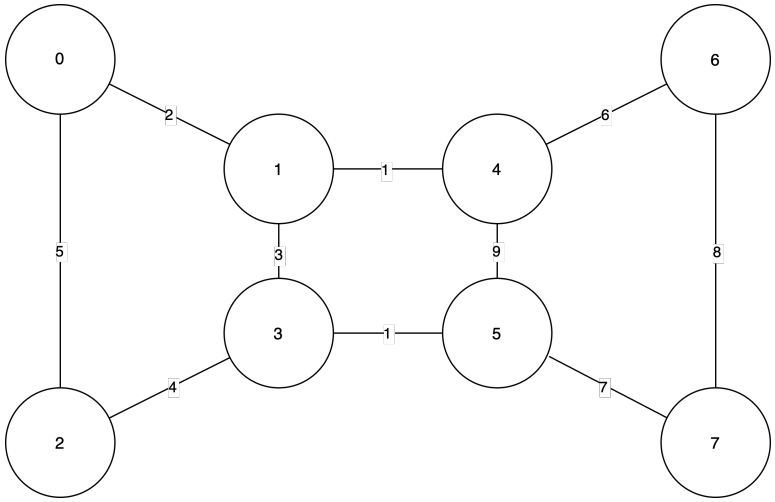

Figure 2: Đồ thị cần phân hoạch.

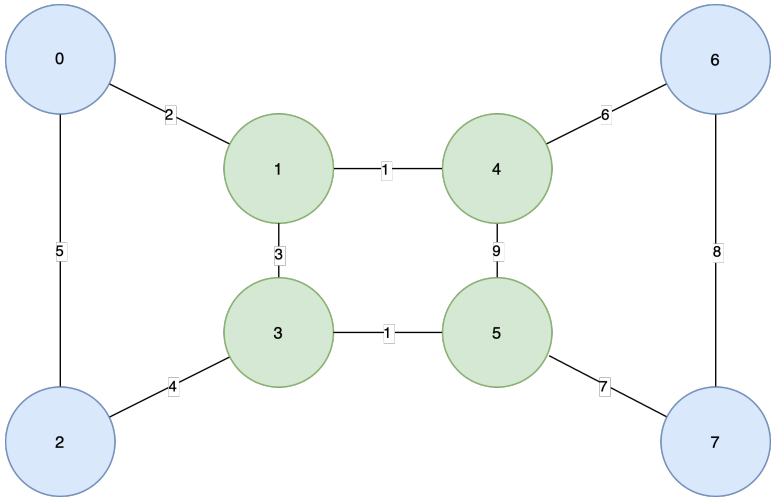

Figure 3: Lời giải cục bộ. Màu sắc của đỉnh thể hiện phân hoạch mà đỉnh thuộc về.

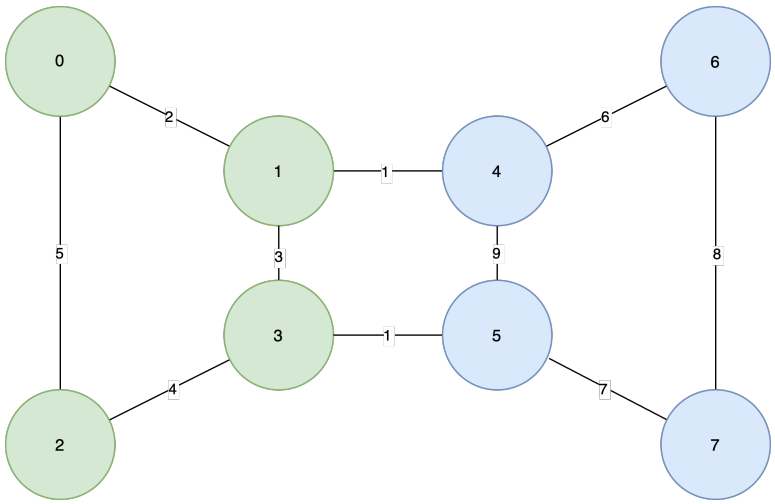

Figure 4: Lời giải toàn cục. Màu sắc của đỉnh thể hiện phân hoạch mà đỉnh thuộc về.

4.2 Tiêu chí (2) - Thuật toán `Kosaraju`

- Ý tưởng của thuật toán này xuất phát ở một định lý:

> Cho một thành phần liên thông mạnh G. Tạo dựng đồ thị có hướng H bằng cách đảo chiều tất cả các cạnh của G. Ta kết luận được H cũng là một thành phần liên thông mạnh.

- Mở rộng ra:

> Trong một đồ thị có hướng, các thành phần liên thông sẽ không thay đổi nếu ta thực hiện đảo chiều tất cả các cạnh của đồ thị đó.

- Thuật toán: thuật toán `Kosaraju` bao gồm 3 bước chính:
  - Duyệt đồ thị ưu tiên chiều sâu (DFS), các đỉnh đã được duyệt qua sẽ được thêm vào một ngăn xếp
  - Tìm chuyển vị của đồ thị
  - Duyệt DFS qua chuyển vị của đồ thị:
    * Nếu đỉnh đó chưa được duyệt: ta gọi hàm DFS từ đỉnh này – tất cả các đỉnh được duyệt trong hàm DFS này sẽ cùng thuộc 1 thành phần liên thông manh.
    * Nếu đỉnh đó đã được duyệt tức là nó đã thuộc 1 thành phần liên thông mạnh đã xét trước đó – ta bỏ qua đỉnh này.
- Cài đặt thuật toán:

- Đầu tiên, nhóm sẽ tiến hành cài đặt hàm `FillOrder` trong lớp `Graph` có chức năng duyệt đồ thị ưu tiên chiều sâu (DFS), các đỉnh đã được duyệt qua sẽ được thêm vào một ngăn xếp.

```python
def FillOrder(self, vertex_ith: int, visited:list, stack:list):
    """ Depth-first traverse through graph once. Note that a
        vertex is added to stack if and only if it and its
        child vertices are visited.

    Arg:
        + vertex_ith (int): id of vertex
        + stack (list): store visited vertices
    """
    # get vertex object from id of vertex
    vertex: Vertex = self.getVertex(vertex_ith)
    if vertex is None:
        message = 'Invalid vertex id, could not found vertex
            id `' + \
            str(vertex_ith) + '` in Graph'
        raise ValueError(get_log(message, log_type='ERROR'))
    visited.append(vertex_ith)

    # Recur for all the vertices adjacent to this vertex
    for i in vertex.getConnections():
        # if it is not visited, then visit
        if i.id not in visited:
            self.FillOrder(i.id, visited, stack)
    stack.append(vertex_ith)
```

- Thứ 2, nhóm sẽ cài đặt một hàm có chức năng lấy chuyển vị của đồ thị trong lớp `Graph`

```python
def get_transpose(self):
    """ Module is used for building graph from edge list
        Return a graph
    """
    g = Graph()
    for i in self.vertList.keys():
        for j in self.getVertex(i).getConnections():
            g.addEdge(j.id, i)
    return g
```

- Cuối cùng, nhóm sẽ thực hiện xây dựng hàm lấy các thành phần liên thông mạnh.

```python
def Find_SCC_by_Kosaraju(self):
    """ Module is used for find strong connect components by
        Kosaraju algorithm.
```

```
        Return strong connect components (SCCs)
    """
    # init
    SCCs = {}
    stack = []
    visited = []

    # Step 1: DFS
    for i in self.getVertices():
        if i not in visited:
            self.FillOrder(i, visited, stack)

    # Step 2: Compute transposed graph
    g_T = self.get_transpose()

    # Step 3: Run DFS again
    visited = []
    index = 0
    while stack:
        i = stack.pop()
        if i not in visited:
            index += 1
            scc = g_T.DFS(i, visited)
            SCCs[index] = scc

    return SCCs
```

5 Tài liệu tham khảo

- Kernighan, Brian W., and Shen Lin. “An efficient heuristic procedure for partitioning graphs.” The Bell system technical journal 49.2 (1970): 291-307.
- Phân hoạch đồ thị: https://patterns.eecs.berkeley.edu/?page_id=571
- Phân hoạch đồ thị: https://en.wikipedia.org/wiki/Graph_partition