BIỄU DIỄN TRI THỨC

Bài tập 3

Nhóm 07

June 19th, 2021

Bài 3

Thu thập tri thức

Lý thuyết đồ thị

Bài toán tìm đường đi ngắn nhất

Câu a. Xác định nguồn tri thức

- **Xác định bài toán**

  Cho đồ thị vô hướng G không có trọng số âm, đỉnh bắt đầu x_0. Tìm đường đi ngắn nhất từ x_0 đến đỉnh bất kỳ x_k theo thuật toán Dijsktra.

- **Nguồn thu thập tri thức**

  - Kenneth H. Rosen. 2018. Discrete mathematics and its applications. Boston: McGraw-Hill

- **Phân loại tri thức**

  - **Khái niệm:** Đỉnh, Cạnh
  - **Quan hệ:** Cạnh là khoảng cách giữa hai đỉnh liền kề
  - **Hàm:** Dijkstra(Đỉnh bắt đầu, Đỉnh kết thúc); Degree(Đỉnh)

- Trình bày thuật giải

  - Thuật toán Dijkstra

    Mô tả bài toán: Cho một đồ thị có hướng $G = (V, E)$, một hàm trọng số $w : E \to [0, \infty]$ và một đỉnh nguồn s. Cần tính toán được đường đi ngắn nhất từ đỉnh nguồn s đến mỗi đỉnh của đồ thị.

    Ứng dụng thực tế:

    * Được sử dụng để tìm con đường đi ngắn nhất.
    * Được sử dụng trong các ứng dụng mạng xã hội.
    * Được ứng dụng trong mạng điện thoại.
    * Được ứng dụng để tìm các vị trí trong bản đồ.

    Thuật toán:

```
    def dijkstra(self, src):

      dist = [sys.maxsize] * self.V
      dist[src] = 0
      sptSet = [False] * self.V

      for cout in range(self.V):

        # Pick the minimum distance vertex from
        # the set of vertices not yet processed.
        # u is always equal to src in first iteration
        u = self.minDistance(dist, sptSet)

        # Put the minimum distance vertex in the
        # shotest path tree
        sptSet[u] = True

```

```
          # Update dist value of the adjacent vertices
          # of the picked vertex only if the current
          # distance is greater than new distance and
          # the vertex in not in the shotest path tree
          for v in range(self.V):
            if self.graph[u][v] > 0 and
              sptSet[v] == False and
              dist[v] > dist[u] + self.graph[u][v]:
                dist[v] = dist[u] + self.graph[u][v]

        self.printSolution(dist)

```

Listing 1: Mã giải của thuật toán Dijkstra

– Thuật toán A*

Mô tả bài toán: Cho một đồ thị có hướng $G = (V, E)$, một hàm trọng số $w : E \to [0, \infty]$ và một đỉnh nguồn s. Cần tính toán được đường đi ngắn nhất từ đỉnh nguồn s đến mỗi đỉnh của đồ thị tuy nhiên dựa vào heuristics để tăng tốc độ tìm kiếm.

Ứng dụng của thuật tóan:

* Được sử dụng để tìm con đường đi ngắn nhất.
* Được sử dụng trong các ứng dụng mạng xã hội.
* Được ứng dụng trong mạng điện thoại.
* Được ứng dụng để tìm các vị trí trong bản đồ.

Thuật toán:

```
      function A*(điểm_xuất_phát,đích)
        var đóng:= tập rỗng
        var q:= tạo_hàng_đợi(tạo_đường_đi(điểm_xuất_phát))
        while q không phải tập rỗng
          var p:= lấy_phần_tử_đầu_tiên(q)
          var x:= nút cuối cùng của p
          if x in đóng
            continue
          if x = đích
            return p
          bổ sung x vào tập đóng
          foreach y in các_đường_đi_tiếp_theo(p)
            đưa_vào_hàng_đợi(q, y)
        return failure

```

Listing 2: Mã giải của thuật toán A*

Câu b. Mô hình biểu diễn tri thức

Mô hình Hệ luật dẫn dạng (F,R) cho kiến thức thu thập được thể hiện như sau:

- **Facts (F)** = tập đỉnh bắt đầu (`start_node`), đỉnh kết thúc (`end_node`), và các cặp đỉnh cùng khoảng cách của nó trong đồ thị gọi là (`path`):
  - `start_node`
  - `end_node`
  - `path_1`, `path_2`, `path_3`, ...`path_n`

  với `path` là bộ các (`from_node`, `to_node`, `distance`).

- **Rules (R)**
  - $\forall$`i` $\in$ {`1,n`}, `from_node_i`, `to_node_j` $\in$ `path_i`, `adjacency_list[from_node_i]` = {`to_node_j`|`from_node_j` = `from_node_i`, $\forall$`j` $\in$ {`1,n`}} $\Rightarrow$ `adjacency_list`
  - $\exists$!`i` $\in$ `n`, min(`path_i[distance]`) $\Rightarrow$ `min_path`

Bài toán

Cho đồ thị vô hướng, trọng số không âm. Tìm đường đi ngắn nhất giữa hai đỉnh lần lượt là đỉnh bắt đầu và đỉnh kết thúc.

- **GT**: Danh sách các cặp đỉnh cùng trọng số (khoảng cách) của chúng, và hai đỉnh bắt đầu, kết thúc.

  Ví dụ: A B 3, B C 5,. . . ; `start_node` = A, `end_node` = C

- **KL**: Trả về đường đi ngắn nhất, và khoảng cách của hai đỉnh đã cho.

  Ví dụ: [A,B,C], 8

```
def shortest_path(self, start_node, end_node):
  # Ban đầu, tất cả các đỉnh đều được coi như chưa được xét.
  unvisited_nodes = self.nodes.copy()

  # Khởi tạo một CTDL từ điển distance_from_start chứa các khoảng cách ban đầu (INFINITY)
  # từ từng đỉnh đến đỉnh start_node. Từ điển này sẽ được cập nhật khi tìm đường ngắn nhất.
  distance_from_start = {
    node: (0 if node == start_node else INFINITY) for node in self.nodes
  }

  # Khởi tạo một CTDL từ điển previous_node chứa các đỉnh truớc đó khi xét tới từng điểm
  # có đường đi ngắn nhất.
  previous_node = {node: None for node in self.nodes}

  # Xét tập các đỉnh chưa được xét
  while unvisited_nodes:
```

```
    # B1: Chọn một đỉnh trong danh sách đỉnh chưa xét mà có khoảng cách ngắn nhất trong
    # từ điển [distance_from_start]
    current_node = min(
    unvisited_nodes, key=lambda node: distance_from_start[node]
    )

    # B2: Loại đỉnh đang xét khỏi danh sách đỉnh chưa xét
    unvisited_nodes.remove(current_node)

    # B3: Nếu đỉnh đang xét có khoảng cách tới đỉnh bắt đầu là INFINITY,
    # nghĩa là hai đỉnh này không liên thông với nhau, quay lại B1
    if distance_from_start[current_node] == INFINITY:
      break

    # B4: Xét các đỉnh láng giềng của đỉnh đang xét,
    # nếu tổng [khoảng cách của đỉnh láng giềng so với đỉnh đang xét] và [khoảng cách đỉnh bắt
     đầu so với đỉnh đang xét] bé hơn khoảng cách [đỉnh bắt đầu so với đỉnh láng giềng]
    # thì cập nhật khoảng cách ngắn nhất, và đỉnh trước đó của láng giềng là đỉnh đang xét
    for neighbor, distance in self.adjacency_list[current_node]:
      new_path = distance_from_start[current_node] + distance
      if new_path < distance_from_start[neighbor]:
        distance_from_start[neighbor] = new_path
        previous_node[neighbor] = current_node

    # B5: Nếu đỉnh đang xét là đỉnh kết thúc thì kết thúc bài toán.
    if current_node == end_node:
      break

# Xây dựng CTDL hàng đợi hai đầu cho đường đi
path = deque()

# Thêm vào đầu hàng đợi lần lượt các đỉnh từ cuối đến đầu
current_node = end_node
while previous_node[current_node] is not None:
  path.appendleft(current_node)
  current_node = previous_node[current_node]
path.appendleft(start_node)

# Trả về đường đi ngắn nhất và khoảng cách cần tìm
return path, distance_from_start[end_node]
```

Listing 3: Thuật toán Dijkstra

Câu c. Tổ chức lưu trữ miền tri thức

Tổ chức lưu trữ miền tri thức trong hệ quản trị cơ sở dữ liệu `(CSDL) MS SQL Server` với các bảng sau:

- Bảng **`Vertices`**: mỗi dòng định danh đỉnh của đồ thị (a, b, c, ...) với:
  - `id` là dãy số duy nhất cho từng đỉnh, `name` là tên đỉnh thường được đặt theo ký hiệu toán học.
  - `degree` là thuộc tính bậc của đỉnh ($deg_a = 2$, $deg_b = 1$, ...), trong đó một số trường hợp đặc biệt số bậc của đỉnh như $deg_b = 1$, cho biết b là đỉnh treo hay nếu $deg_b = 0$ thì b là đỉnh cô lập.

Vertices	
PK	**`id int NOT NULL`**
	`name char(50) NOT NULL`
	`degree int NOT NULL`

- Bảng **`Edges`**: mỗi dòng chứa thông tin của một cạnh trong đồ thị bao gồm hai cặp `id` của đỉnh từ bảng `vertices`, khoảng cách (`distance`) giữa hai đỉnh đó và tên (`name`) của cạnh đó.

Edges	
FK	**`from_id char(50) NOT NULL`**
FK	**`to_id char(50) NOT NULL`**
	`distance int NOT NULL`
	`name char(100) NOT NULL`

Miền tri thức trên cho phép giải các bài toán liên quan tới phân loại đồ thị, tìm đường đi và chu trình đường đi trong đồ thị.

Ví dụ minh họa chi tiết miền tri thức lý thuyết đồ thị bằng cấu trúc bảng dữ liệu với đồ thị gồm 5 đỉnh A,B,C,D,E với độ dài các cạnh tương ứng như bên dưới:

vertices		
id	name	degree
1	A	2
2	B	3
3	C	4
4	D	5
5	E	2

edges			
from_id	to_id	distance	name
1	2	4	Nguyễn Văn Trỗi
1	3	1	Cộng Hòa
1	4	1	Trường Sơn
2	3	2	Lý Chính Thắng
3	4	5	Nam Kì Khởi Nghĩa
4	5	3	Hoàng Văn Thụ

Hình 1: Minh họa miền tri thức định dạng bảng trong CSDL