Toán rời rạc và thuật toán
Đại học Khoa học Tự nhiên
Khoa Toán - Cơ - Tin học
Khoa học dữ liệu K4
Tháng 10 năm 2022

Bài tập số 2 - Lý thuyết đồ thị

Nguyễn Mạnh Linh, Nguyễn Thị Đông, Triệu Hồng Thúy

1 Bài 1

2 Bài 2

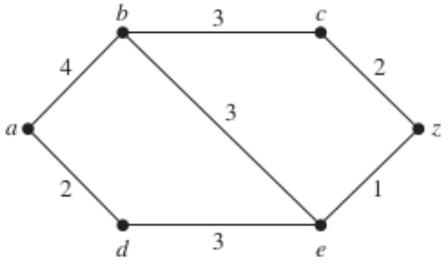
Trong phần này chúng ta sẽ xem xét bài toán "Travelling salesman problem" (bài toán người giao hàng) - là một trong những bài toán kinh điển và nổi tiếng trong lớp các bài toán về đồ thị nói chung và bài toán tìm đường đi ngắn nhất nói riêng.

Bài toán: Cho một danh sách các thành phố và khoảng cách từng cặp một, tìm đường đi ngắn nhất đi qua mỗi thành phố đúng một lần và quay về điểm xuất phát

Chúng ta có thể mô hình bài toán trên bằng đồ thị như sau: *Cho một đơn đồ thị đầy đủ, với các cạnh có trọng số, tìm đường đi xuất phát từ một đỉnh đi qua tất cả các đỉnh của đồ thị mỗi đỉnh đúng một lần với tổng trọng số (chi phí) nhỏ nhất.*
Trước hết chúng ta giải bài toán con đơn giản hơn là tìm đường đi với chi phí nhỏ nhất từ một đỉnh a tới đỉnh z cho trước. Trong bài toán này, không nhất thiết ta phải đi qua tất cả các đỉnh của đồ thị. Có nhiều cách tiếp cận bài toán, cách đầu tiên mà ta có thể nghĩ ngay tới là *vét cạn (brute force)*, tuy nhiên việc liệt kê toàn bộ cấu hình (toàn bộ các đường đi khả dĩ từ a đến z) của bài toán rất tốn thời gian. Sau đây ta cùng xem xét một thuật toán hiệu quả hơn là thuật toán Dijkstra.

2.1 Thuật toán Dijkstra tìm đường đi ngắn nhất trên đồ thị

Trước khi giải bài toán tổng quát, ta hãy xem xét một ví dụ đơn giản sau đây:



Hình 1: Ví dụ đơn đồ thị với trọng số

Tìm đường đi ngắn nhất từ đỉnh a đến đỉnh z trong đồ thị ở hình 1.

Trước hết, bắt đầu từ đỉnh a, có 2 đỉnh kết nối với a là b và d với trọng số lần lượt là 4 và 2, đương nhiên d là đỉnh gần a nhất. Chúng ta có thể tìm đỉnh gần nhất thứ 2 bằng cách liệt kê tất cả các đường đi với d là đỉnh bắt đầu và đỉnh kết thúc không nằm trong tập $\{a, d\}$. Tính từ đỉnh d, chỉ có một đường khả dĩ là e. Từ đó ta có tập $\{a, d, e\}$. Tương tự như vậy ta tìm đỉnh tiếp theo với cạnh có trọng số nhỏ nhất với e là đỉnh bắt đầu và đỉnh kết thúc không thuộc tập $\{a, d, e\}$, ta được đỉnh z. Đường đi cuối cùng là $a \to d \to e \to z$.

Ví dụ trên thể hiện nguyên lý tổng quát của thuật toán Dijkstra đó là đường đi từ đỉnh a tới đỉnh z có thể được xây dựng bằng cách liệt kê các đường đi và tìm đường có trọng số nhỏ nhất để thêm đỉnh đó vào tập đường đi.
Một cách tổng quát ta có thể trình bày thuật toán Dijkstra với lược đồ sau:

Thuật toán Dijkstra
G là một đơn đồ thị đầy đủ kết nối với trọng số mỗi cạnh dương
G có các đỉnh $a = v_0, v_1, ..., v_n = z$, trọng số $w(v_i, v_j)$ với $w(v_i, v_j) = \infty$ nếu $\{v_i, v_j\}$ không thuộc tập cạnh của G

for i = 1 to n

$L(v_i) = \infty$

$L(a) = 0$

$S = \emptyset$ (Khởi tạo các nhãn với nhãn của a là 0, các nhãn khác bằng ∞ và tập S rỗng)

while $z \notin S$

$u = a$ đỉnh không thuộc S với $L(u)$ nhỏ nhất

$S = S \cup \{u\}$

for $v \notin S$

if $L(u) + w(u, v) < L(v)$ then $L(v) = L(u) + w(u, v)$ (Bước này thêm đỉnh gần nhất vào tập S và cập nhật lại hàm chi phí)

return $L(z)$ ($L(z)$ là độ dài của đường đi ngắn nhất từ a tới z)

Nhận xét: Thực chất thuật toán Dijkstra vẫn phải duyệt qua toàn bộ cấu hình của bài toán và không làm giảm độ phức tạp so với việc liệt kê toàn bộ đường đi và tính giá trị chi phí cho mỗi cấu hình. Tuy nhiên thuật toán Dijkstra đem lại sự tường minh mặc dù quét toàn bộ các đỉnh không thuộc tập đã đi qua (S) nhưng ta chỉ cập nhật đỉnh gần với đỉnh cuối cùng nhất, điều đó làm giảm không gian lưu trữ so với cách vét cạn (khi ta phải lưu toàn bộ đường đi và chi phí của mọi đường đi khả dĩ từ đỉnh a tới đỉnh z).

Trong thực tế lập trình, chúng ta không nhất thiết phải lưu giá trị cạnh bằng một số lớn (vô cùng) mà có thể đánh bằng 0 và kiểm tra thêm một điều kiện trong vòng lặp để tránh sự cồng kềnh của ma trận kề.

python code cho bài toán