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

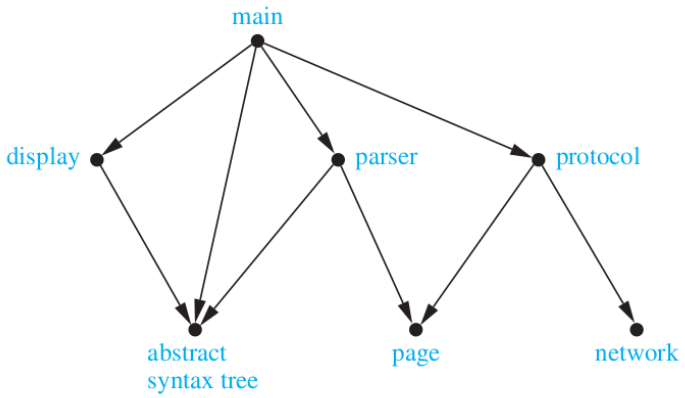
1 Bài 1

1.1 Những khái niệm cơ bản về đồ thị

1.2 Cấu trúc dữ liệu biểu diễn đồ thị

1.2.1 Thiết kế phần mềm

Đồ thị các thành phần phụ thuộc (Module Dependency Graphs)
Một trong những nhiệm vụ quan trọng trong thiết kế phần mềm đó là chia chương trình thành nhiều thành phần hoặc module khác nhau để tiện cho việc phát triển mà mở rộng cũng như bảo trì sau này. Việc hiểu được sự tương tác giữa các modules trong một chương trình tương tác với nhau như thế nào là cực kì quan trọng không chỉ trong việc thiết kế chương trình mà còn trong việc kiểm thử và bảo trì nữa. Một đồ thị các thành phần phụ thuộc giúp ích rất nhiều trong việc này. Trong đồ thị các thành phần phụ thuộc (dependencies), mỗi đỉnh biểu thị một module, một cạnh nối có hướng chỉ sự phụ thuộc của module này vào module kia. Một ví dụ về đồ thị biểu diễn sự phụ thuộc của các modules trong một ứng dụng web:



Hình 1: Ví dụ đồ thị sự phụ thuộc của các modules trong một ứng dụng web

1.2.2 Mạng giao thông

Mô hình đồ thị được sử dụng trong nhiều loại mạng giao thông khác nhau như đường bộ, hàng không, mạng đường sắt và mạng chuyển phát.

Định tuyến trong hàng không
Mô hình mạng hàng không có thể được biểu diễn bằng đồ thị với mỗi sân bay là một đỉnh. Các chuyến bay từ sân bay này tới sân bay khác có thể được biểu diễn bằng một cạnh có hướng từ sân bay cất cánh (đỉnh bắt đầu) đến sân bay hạ cánh (đỉnh kết thúc).

Mạng đường bộ
Mô hình đồ thị có thế được sử dụng để biểu diễn mạng đường bộ mà trong đó các đỉnh thể hiện các giao lộ và các cạnh thể hiện đường đi. Khi tất cả các cong đường trong mạng đều là đường 2 chiều thì ta có thể biểu diễn mạng bằng một đơn đồ thị vô hướng. Tuy nhiên trong thực tế ta thường gặp trong mạng giao thông thì một số đường là 2 chiều và một số khác là 1 chiều. Để biểu diễn mạng này ta dùng cạnh vô hướng để biểu diễn đường 2 chiều và cạnh có hướng để biểu diễn đường một chiều.

1.2.3 Mạng sinh học

Đồ thị chồng chéo ngách trong hệ sinh thái

1.3 Mô hình và ứng dụng của đồ thị trong thực tế

2 Bài 2

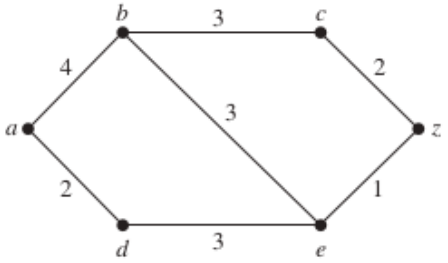
Trong phần này chúng ta sẽ xem xét bài toán "Travelling salesman problem" (bài toán người giao hàng) - là một trong những bài toán kinh điển và nổi tiếng trong lớp các bài toán về đồ thị nói chung và bài toán tìm đường đi ngắn nhất nói riêng.

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 2: Ví dụ đơn đồ thị với trọng số

Tìm đường đi ngắn nhất từ đỉnh a đến đỉnh z trong đồ thị ở hình 2.

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 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.

2.2 Bài toán người đưa hàng (TSP)

2.2.1 Thuật toán tìm nghiệm chính xác

Trong mục này chúng ta sẽ xem xét bài toán người đưa hàng đã đề cập ở đầu bài. Sau khi giải bài toán tìm đường đi ngắn nhất ở mục trước, ta biết rằng bài toán người đưa hàng không gì khác ngoài việc tìm được đi ngắn nhất khi ta đặt điểm kết thúc chính là điểm bắt đầu.
python code cho bài toán.

Nhận xét: Như đã nói ở phần trước, thuật toán này không làm giảm độ phức tạp tính toán so với cách vét cạn và là $O(n!)$. Độ phức tạp này tăng rất nhanh theo n . Ví dụ khi bắt đầu từ 1 đỉnh, ta có $(n-1)!$ chu trình Hamilton, tới đỉnh thứ 2 ta còn $(n-2)!$ chu trình Hamilton và tiếp tục như vậy. Bởi vì ta có thể đi theo thứ tự ngược lại trong chu trình Hamilton nên số chu trình cần sinh ra để có được lời giải là $(n-1)!/2$. Với số đỉnh bằng 25, số chu trình được sinh ra là $24!/2 \sim 3.1 \times 10^{23}$ - một con số lớn khủng khiếp. Trong thực tế, một đơn vị vận chuyển như *Giao hàng tiết kiệm* giao hàng triệu đơn hàng mỗi ngày tới các địa điểm khác nhau! Do đó ta gần như không thể tìm được lời giải chính xác của bài toán TSP trong thực tế mà cần phải sử dụng các phương pháp xấp xỉ để tìm được một đường đi *đủ tốt*.

2.2.2 Phương pháp xấp xỉ