WIKIPEDIA

Thuật toán Prim

Bách khoa toàn thư mở Wikipedia

Trong khoa học máy tính, **thuật toán Prim** là một thuật toán tham lam để tìm cây bao trùm nhỏ nhất của một đồ thị vô hướng có trọng số liên thông. Nghĩa là nó tìm một tập hợp các cạnh của đồ thị tạo thành một cây chứa tất cả các đỉnh, sao cho tổng trọng số các cạnh của cây là nhỏ nhất. Thuật toán được tìm ra năm 1930 bởi nhà toán học người Séc Vojtěch Jarník và sau đó bởi nhà nghiên cứu khoa học máy tính Robert C. Prim năm 1957 và một lần nữa độc lập bởi Edsger Dijkstra năm 1959. Do đó nó còn được gọi là **thuật toán DJP**, **thuật toán Jarník**, hay **thuật toán Prim–Jarník**.

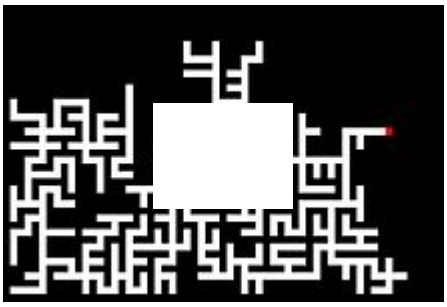
Một vài thuật toán đơn giản khác cho bài toán này bao gồm thuật toán Kruskal và thuật toán Borůvka.

Mục lục

Mô tả

Thuật toán xuất phát từ một cây chỉ chứa đúng một đỉnh và mở rộng từng bước một, mỗi bước thêm một cạnh mới vào cây, cho tới khi bao trùm được tất cả các đỉnh của đồ thị.

- Dữ liệu vào: Một đồ thị có trọng số liên thông với tập hợp đỉnh V và tập hợp cạnh E (trọng số có thể âm). Đồng thời cũng dùng V và E để ký hiệu số đỉnh và số cạnh của đồ thị.
- Khởi tạo: $V_{mới} = \{x\}$, trong đó x là một đỉnh bất kì (đỉnh bắt đầu) trong V, $E_{mới} = \{\}$
- Lặp lại cho tới khi $V_{mới} = V$:
  - Chọn cạnh (u, v) có trọng số nhỏ nhất thỏa mãn u thuộc $V_{mới}$ và v không thuộc $V_{mới}$ (nếu có nhiều cạnh như vậy thì chọn một cạnh bất kì trong chúng)
  - Thêm v vào $V_{mới}$, và thêm cạnh (u, v) vào $E_{mới}$
- Dữ liệu ra: $V_{mới}$ và $E_{mới}$ là tập hợp đỉnh và tập hợp cạnh của một cây bao trùm nhỏ nhất



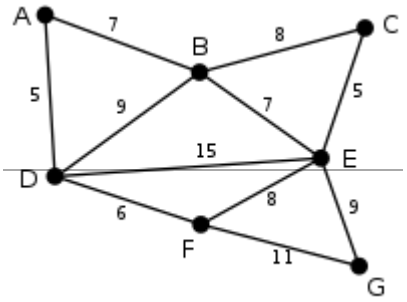
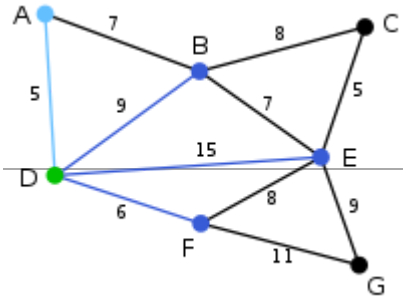
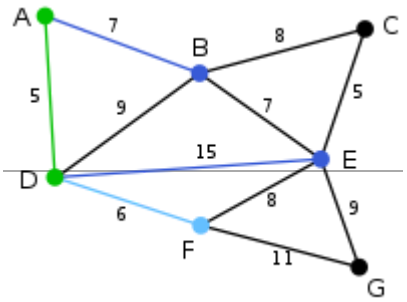
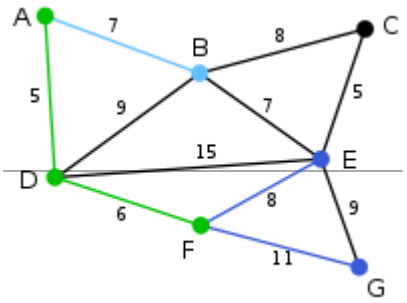
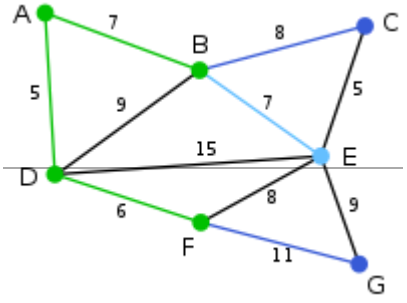
Thuật toán Prim có nhiều ứng dụng, chẳng hạn như xây dựng mê cung trên, bằng cách áp dụng thuật toán Prim cho một đồ thị lưới có trọng số ngẫu nhiên.

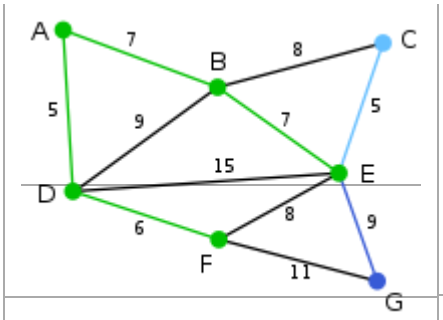
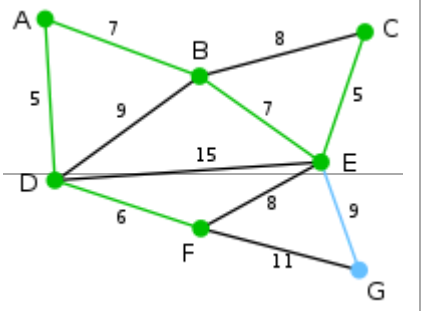
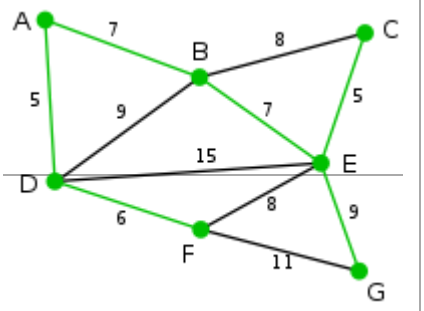
Độ phức tạp tính toán

Cấu trúc dữ liệu tìm cạnh có trọng số nhỏ nhất	Độ phức tạp thời gian (tổng cộng)
Tìm kiếm trên ma trận kề	$O(V^2)$
Đống nhị phân và danh sách kề	$O((V + E) \log V) = O(E \log V)$
Đống Fibonacci và danh sách kề	$O(E + V \log V)$

Một cách lập trình đơn giản sử dụng ma trận kề và tìm kiếm toàn bộ mảng để tìm cạnh có trọng số nhỏ nhất có thời gian chạy $O(V^2)$. Bằng cách sử dụng cấu trúc dữ liệu đống nhị phân và danh sách kề, có thể giảm thời gian chạy xuống $O(E \log V)$. Bằng cách sử dụng cấu trúc dữ liệu đống Fibonacci phức tạp hơn, có thể giảm thời gian chạy xuống $O(E + V \log V)$, nhanh hơn thuật toán trước khi đồ thị có số cạnh $E=\omega(V)$.

Ví dụ

Hình minh họa	U	Cạnh (u,v)	V \ U	Mô tả
	{}		{A,B,C,D,E,F,G}	Đây là đồ thị có trọng số ban đầu. Các số là các trọng số của các cạnh.
	{D}	(D,A) = 5 **V** (D,B) = 9 (D,E) = 15 (D,F) = 6	{A,B,C,E,F,G}	Chọn một cách tùy ý đỉnh **D** là đỉnh bắt đầu. Các đỉnh **A**, **B**, **E** và **F** đều được nối trực tiếp tới **D** bằng cạnh của đồ thị. **A** là đỉnh gần **D** nhất nên ta chọn **A** là đỉnh thứ hai của cây và thêm cạnh **AD** vào cây.
	{A,D}	(D,B) = 9 (D,E) = 15 (D,F) = 6 **V** (A,B) = 7	{B,C,E,F,G}	Đỉnh được chọn tiếp theo là đỉnh gần **D** hoặc **A** nhất. **B** có khoảng cách tới **D** bằng 9 và tới **A** bằng 7, **E** có khoảng cách tới cây hiện tại bằng 15, và **F** có khoảng cách bằng 6. **F** là đỉnh gần cây hiện tại nhất nên chọn đỉnh **F** và cạnh **DF**.
	{A,D,F}	(D,B) = 9 (D,E) = 15 (A,B) = 7 **V** (F,E) = 8 (F,G) = 11	{B,C,E,G}	Thuật toán tiếp tục tương tự như bước trước. Chọn đỉnh **B** có khoảng cách tới **A** bằng 7.
	{A,B,D,F}	(B,C) = 8 (B,E) = 7 **V** (D,B) = 9 chu trình (D,E) = 15 (F,E) = 8 (F,G) = 11	{C,E,G}	Ở bước này ta chọn giữa **C**, **E**, và **G**. **C** có khoảng cách tới **B** bằng 8, **E** có khoảng cách tới **B** bằng 7, và **G** có khoảng cách tới **F** bằng 11. **E** là đỉnh gần nhất, nên chọn đỉnh **E** và cạnh **BE**.
	{A,B,D,E,F}	(B,C) = 8	{C,G}	Ở bước này ta chọn

		(D,B) = 9 chu trình (D,E) = 15 chu trình (E,C) = 5 **V** (E,G) = 9 (F,E) = 8 chu trình (F,G) = 11		giữa **C** và **G**. **C** có khoảng cách tới **E** bằng 5, và **G** có khoảng cách tới **E** bằng 9. Chọn **C** và cạnh **EC**.
	{A,B,C,D,E,F}	(B,C) = 8 chu trình (D,B) = 9 chu trình (D,E) = 15 chu trình (E,G) = 9 **V** (F,E) = 8 chu trình (F,G) = 11	{G}	Đỉnh **G** là đỉnh còn lại duy nhất. Nó có khoảng cách tới **F** bằng 11, và khoảng cách tới **E** bằng 9. **E** ở gần hơn nên chọn đỉnh **G** và cạnh **EG**.
	{A,B,C,D,E,F,G}	(B,C) = 8 chu trình (D,B) = 9 chu trình (D,E) = 15 chu trình (F,E) = 8 chu trình (F,G) = 11 chu trình	{}	Hiện giờ tất cả các đỉnh đã nằm trong cây và cây bao trùm nhỏ nhất được tô màu xanh lá cây. Tổng trọng số của cây là 39.

Chứng minh

Đặt G là một đồ thị có trọng số liên thông. Trong mỗi bước, thuật toán Prim chọn một cạnh nối một đồ thị con với một đỉnh không thuộc đồ thị con đó. Vì G liên thông nên luôn tồn tại đường đi từ mỗi đồ thị con tới tất cả các đỉnh còn lại. Kết quả T của thuật toán Prim là một cây, vì các cạnh và đỉnh được thêm vào T là liên thông và cạnh mới thêm không bao giờ tạo thành chu trình với các cạnh cũ. Đặt T_1 là một cây bao trùm nhỏ nhất của G. Nếu $T_1=T$ thì T là cây bao trùm nhỏ nhất. Nếu không, đặt e là cạnh đầu tiên được thêm vào T mà không thuộc T_1, và V là tập hợp các đỉnh thuộc T trước khi thêm e. Một đầu của e thuộc V và đầu kia không thuộc V. Vì T_1 là một cây bao trùm của G, nên tồn tại đường đi trong T_1 nối hai đầu của e. Trên đường đi đó, nhất định tồn tại cạnh f nối một đỉnh trong V với một đỉnh ngoài V. Trong bước lặp khi e được thêm vào Y, thuật toán cũng có thể chọn f thay vì e nếu như trọng số của nó nhỏ hơn e. Vì f không được chọn nên

$$w(f) \geq w(e).$$

Đặt T_2 là đồ thị thu được bằng cách xóa f và thêm e vào T_1. Dễ thấy T_2 liên thông, có cùng số cạnh như T_1, và tổng trọng số các cạnh không quá trọng số của T_1, nên nó cũng là một cây bao trùm nhỏ nhất của G và nó chứa e cũng như tất cả các cạnh được thuật toán chọn trước nó. Lặp lại lập luận trên nhiều lần, cuối cùng ta thu được một cây bao trùm nhỏ nhất của G giống hệt như T. Vì vậy T là một cây bao trùm nhỏ nhất.

Tham khảo

- V. Jarník (1930), "O jistém problému minimálním", *Práce Moravské Přírodovědecké Společnosti* **6**: 57–63 **(tiếng Séc)**
- R. C. Prim (1957), "Shortest connection networks and some generalizations", *Bell System Technical Journal* **36**: 1389–1401**(tiếng Anh)**
- Cheriton, David; Tarjan, Robert E. (tháng 12 1976), "Finding minimum spanning trees", *SIAM Journal on Computing* **5** (4): 724–741 Kiểm tra giá trị ngày tháng trong: |date= (trợ giúp)**(tiếng Anh)**
- Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford (2009), *Introduction to Algorithms* (ấn bản 3), MIT Press, ISBN 0-262-03384-4 Phần 23.2: The algorithms of Kruskal and Prim, tr. 631–638.**(tiếng Anh)**

Liên kết ngoài

- Applet mô tả thuật toán Prim (http://students.ceid.upatras.gr/~papagel/project/prim.htm)
- Một applet khác mô tả thuật toán Prim (http://www.mincel.com/java/prim.html)
- Chương trình Python minh họa thuật toán tìm cây bao trùm nhỏ nhất viết bởi Ronald L. Rivest (http://people.csail.mit.edu/rivest/programs.html)

Wikimedia Commons có thư viện hình ảnh và phương tiện truyền tải về ***Thuật toán Prim (https://commons.wikimedia.org/wiki/Category:Prim%27s_Algorithm?uselang=vi)***