Giải Thuật Lập Trình

Nơi tổng hợp và chia sẻ những kiến thức liên quan tới giải thuật nói chung và lý thuyết khoa học máy tính nói riêng.

‹ Cây khung nhỏ nhất: thuật toán Kruskal --- Kruskal Algorithm • Cây khung nhỏ nhất: thuật toán Borůvka -- Borůvka Algorithm ›

Cây khung nhỏ nhất: thuật toán Prim --- Prim Algorithm

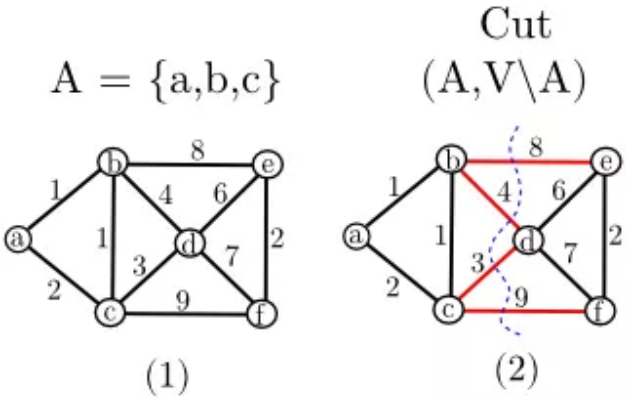
May 26, 2016 in Uncategorized | No comments

Bài này là bài thứ hai trong chuỗi bài mình viết về các thuật toán tìm cây khung nhỏ nhất. Trong bài viết trước (http://www.giaithuatlaptrinh.com/?p=1140), mình giới thiệu bài toán và một số tính chất cơ bản của cây khung nhỏ nhất. Mình sẽ không nhắc lại các tính chất đó ở đây nữa.

Thuật toán Prim, theo Wikipedia (https://en.wikipedia.org/wiki/Prim's_algorithm), được tìm ra đầu tiên bởi Jarník [2] năm 1930, sau đó được Prim [1] độc lập phát triển năm 1957. Thuật toán Prim xây dựng cây khung dựa trên lát cắt (cut of graphs).

Lát cắt (Cut): Gọi A là một tập đỉnh của đồ thị. Lát cắt tương ứng với A, kí hiệu là $(A, V \setminus A)$, là tập **tất cả** các cạnh của G có đúng một đầu mút (incident vertex) nằm trong A (đầu mút còn lại hiển nhiên là nằm trong $V \setminus A$).

Ví dụ lát cắt tương ứng với $\{a, b, c\}$ của đồ thị trong hình (a) là các cạnh màu đỏ trong hình (b).



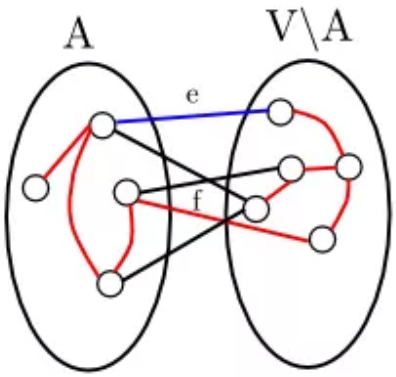
Xét một chu trình C của đồ thị và một lát cắt $(A, V \setminus A)$. Nếu C chỉ đi qua các đỉnh trong trong A hoặc chỉ đi qua các đỉnh trong $V \setminus A$, thì C sẽ không chứ cạnh nào của lát cắt, i.e, $|C \cap (A, V \setminus A)| = \emptyset$. Nếu C chứa đỉnh của cả A lần $V \setminus A$ thì ta sẽ thấy C chứa một **số chẵn** cạnh của lát cắt $(A, V \setminus A)$. Do đó, ta có:

Fact 1: Nếu C là một chu trình của đồ thị và $(A, V \setminus A)$ là một lắt cắt, thì $|C \cap (A, V \setminus A)|$ là một số chẵn.

Fact 1 cho chúng ta biết, nếu $|C \cap (A, V \setminus A)| \neq \emptyset$ thì C chứa ít nhất hai cạnh của của lát cắt $(A, V \setminus A)$. Sau đây chúng ta sẽ thảo luận quan hệ giữa cây khung và cắt. Bổ đề dưới đây tương tự như tính chất (5) của cây khung nhỏ nhất mà chúng ta đã thảo luận ở bài trước (http://www.giaithuatlaptrinh.com/?p=1140).

Lemma 1: Gọi e là cạnh có trọng số nhỏ nhất của một lát cắt $(A, V \setminus A)$. Tồn tại một cây khung nhỏ nhất T của đồ thị chứa cạnh e.

Chứng minh: Thật vậy, gọi F là một cây khung nhỏ nhất nhất không chứa e. Ta có $F \cup \{e\}$ sẽ chứa một chu trình C. Chu trình này phải chứ e. Do đó, $|C \cap (A, V \setminus A)| \neq \emptyset$. Theo Fact 1, C sẽ chứa một cạnh f khác e nằm trong cắt $(A, V \setminus A)$. Thêm nữa, $w(f) \geq w(e)$, do e là cạnh có trọng số nhỏ nhất.



Gọi $T = F \cup \{e\} \setminus \{f\}$. T là một cây khung của đồ thị (tại sao?). Do $w(e) \leq w(f)$, ta suy ra $w(T) \leq w(F)$. Do đó, T cũng phải là một cây khung nhỏ nhất vì F là cây khung nhỏ nhất. Như vậy, T là một cây khung nhỏ nhất chứa e.

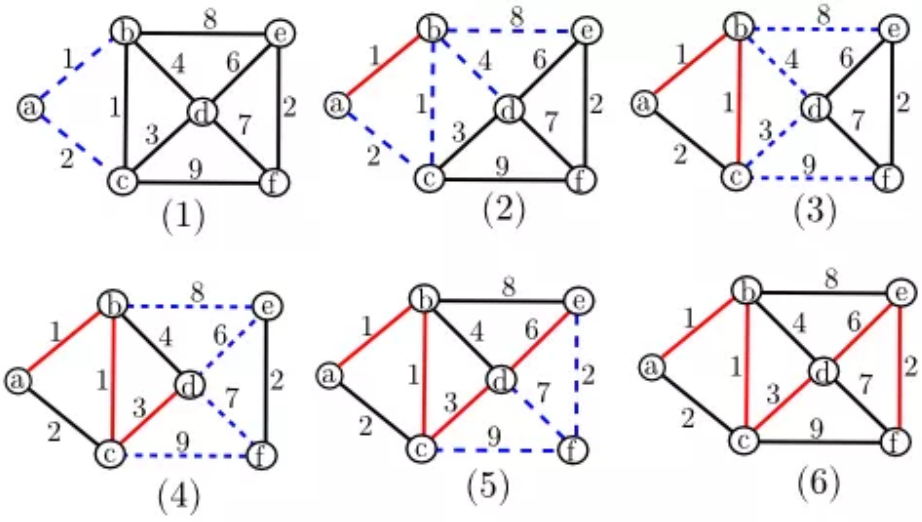
Lemma 1 cho chúng ta biết cạnh nhỏ nhất của một lát cắt luôn nằm trong một cây khung nhỏ nhất nào đó. Ta sẽ sử dụng tính chất này để xây dựng cây khung.

Gọi u là một đỉnh bất kì thuộc đồ thị và $(\{u\}, V \setminus \{u\})$ là cắt tương ứng với u. Cắt này chính là tập các cạnh kề với u. Gọi e là cạnh có trọng số nhỏ nhất trong cắt $(\{u\}, V \setminus \{u\})$. Theo Lemma 1, tồn tại cây khung nhỏ nhất chứa cạnh này. Do đó, ta có thể lấy e là cạnh đầu tiên trong cây khung mà ta muốn tìm.

Gọi v là đầu mút khác u của e. Cây hiện tại giờ chỉ gồm một cạnh e. Xét lát cắt $(\{u, v\}, V \setminus \{u, v\})$ và một cạnh e' là cạnh nhỏ nhất trong lát cắt. Lemma 1 cho chúng ta biết tồn tại cây khung nhỏ nhất chứa e' (và cũng chứa e vì theo chứng minh của Lemma 1, ta chỉ thay đổi cạnh khác trong cùng lát cắt bằng e' để thu được cây khung chứa e'). Do đó, ta có thể thêm tiếp e' vào cây khung hiện tại. Gọi w là đầu mút khác v của e' (đương nhiên $w \neq u$), và ta lại có thể tiếp tục thêm cạnh nhỏ nhất của cắt $(\{u, v, w\}, V \setminus \{u, v, w\})$ vào cây hiện tại. Cứ lặp lại như vậy cho đến

khi cây hiện tại chứa toàn bộ tập đỉnh của G và cây đó cũng là một cây khung nhỏ nhất.

Ví dụ thực hiện quá trình trên với đồ thị dưới đây. Các cạnh màu đỏ là các cạnh mà ta đã thêm vào cây khung nhỏ nhất. Các cạnh màu xanh gạch ngang là các cạnh của lát cắt mà ta xét để tìm ra cạnh cần thêm vào cây khung.



Thuật toán Prim chính là thuật toán thực hiện quá trình mà chúng ta vừa thảo luận. Giả mã:

```
PRIM(G(V, E), w):
    pick any vertex u
    add the smallest edge of the cut ({u}, V \ {u}) to T
    while T ≠ V          ≪ T does not contain all vertices≫
        e ← the smallest edge of the cut (T, V \ T)
        T ← T ∪ {e}
    return T
```

Tính đúng đắn của giải thuật Prim có thể được suy ra trực tiếp từ các thảo luận ở trên. Chi tiết cụ thể hơn coi như bài tập cho bạn đọc.

Giờ chúng ta thảo luận cách thực thi thuật toán (đây luôn là phần thú vị nhất). Gọi n, m lần lượt là số đỉnh và số cạnh của đồ thị. Ta thấy thời gian thực thi thuật toán là $O(nF(n,m))$ trong đó $F(n,m)$ là thời gian để thực thi dòng màu đỏ. Thuật toán nhanh hay chậm quyết định bởi dòng này.

Cách thực thi dòng màu đỏ đơn giản nhất đó là duyệt qua tất cả các cạnh của lát cắt và tìm cạnh có trọng số nhỏ nhất. Thời gian của cách thực thi này có thể lên tới $O(m)$, và như vậy, tổng thời gian của thuật toán là $O(nm)$. Với đồ thị dầy ($m = O(n^2)$), cách này cho chúng ta thuật toán $O(n^3)$ (còn xa mới bằng Kruskal).

Thực thi trong thời gian $O(n^2)$

Với cách thực thi này, ta sẽ sử dụng biểu diễn là ma trận kề. Mỗi đỉnh u **không thuộc** cây hiện tại T ($u \in V \setminus T$), ta sẽ lưu một nhãn $d[u]$. Nhãn $d[u]$ là trọng số của cạnh nhỏ nhất trong số các cạnh từ u tới các đỉnh

trong T hiện tại. Ta cũng lưu thêm một nhãn $M[u]$, trong đó $M[u] = v$ nếu v là đỉnh trong T mà $w(u, v) = d[u]$.

Để tìm cạnh nhỏ nhất của lát cắt (thực hiện dòng màu đỏ), ta sẽ duyệt qua tất cả các đỉnh trong $V \setminus T$, tìm ra đỉnh u có $d[u]$ nhỏ nhất. Khi đó, cạnh $(u, M[u])$ chính là cạnh nhỏ nhất của lát cắt $(T, V \setminus T)$. Do đó, $F(n) \leq n$. Liệu chúng ta đã xong?

Còn một vấn đề nữa, đó là sau khi thêm một cạnh vào trong cây khung hiện tại T, ta phải cập nhật lại nhãn cho các đỉnh còn lại. Giả sử cạnh (u, v), với $v = M[u]$, sẽ được thêm vào cây khung. Giả sử trước khi thêm cạnh này, $u \notin T$. Để cập nhật nhãn, ta duyệt qua các đỉnh còn lại trong $V \setminus T$ và với mỗi đỉnh x, ta so sánh trọng số $w(u, x)$ với $d[x]$. Nếu $w(u, x) < d[x]$, ta sẽ cập nhật $d[x] = w(u, x)$ và $M[x] = u$. Nếu không ta sẽ giữ nguyên trọng số. Thao tác cập nhật này có thể thực hiện trong thời gian $O(n)$, do đó, tổng thời gian của thuật toán sẽ là $O(n^2)$.

```
FastPrim(G(V, E), w):
    pick any vertex u
    M[1, ..., n] ← {0, ..., 0}
    d[1, ..., n] ← {∞, ..., ∞}
    for each neighbor x of u
        M[x] ← u
        d[x] ← w(u, x)
    T ← ∅
    while T ≠ V          ≪ T does not contain all vertices≫
        u ← FindSmallestVertex(M[1, ..., V])
        for each neighbor x of u
            if x ∉ T and w(u, x) < d[x]
                M[x] ← u
                d[x] ← w(u, x)
        T ← T ∪ {(u, M[u])}          ≪ add edge (u, M[u]) to T ≫
    return T
```

```
FindSmallestVertex(d[1, ..., n]):
    min ← +∞
    u ← 0
    for each v ∈ V
        if d[v] < min and v ∉ T
            min ← d[v]
            u ← v
    return u
```

Khi code giả mã trên, ta có thể nhận thấy từ mảng M, ta có thể xây dựng lại được cây T. Với mỗi x mà $M[x] \neq 0$, $(x, M[x])$ chính là cạnh của cây T. Ta sẽ dùng thêm một mảng $inT[1, \ldots, n]$ trong đó $inT[u] =$ True nếu u ở trong cây hiện tại và $inT[u] =$ False nếu ngược lại. Code C:

```
+ expand source (#)
```

Thực thi trong thời gian $O((m+n)\log n)$

Thao tác FindSmallestVertex tìm một đỉnh có nhãn nhỏ nhất trong tập các đỉnh không nằm trong T. Thao tác này gợi lại cho chúng ta cấu trúc Heap (http://www.giaithuatlaptrinh.com/?p=736). Thực sự, chúng ta có thể thực thi thao tác này sử dụng Heap nhị phân với khóa là nhãn $d[1,\ldots,n]$.

```
PrimHeap(G(V,E), w):
    pick any vertex u
    M[1,...,n] ← {0,...,0}
    d[1,...,n] ← {∞,...,∞}
    for each neighbor x of u
        M[x] ← u
        d[x] ← w(u,x)
    initialize heap H with keys d[1,2,...,n]
    T ← ∅
    while T ≠ V          ≪ T does not contain all vertices≫
        u ← ExtractMin(H)
        for each neighbor x of u
            if x ∉ T and w(u,x) < d[x]
                M[x] ← u
                d[x] ← w(u,x)
                DecreaseKey(H, x, d[x])
        T ← T ∪ {(u, M[u])}      ≪ add edge (u, M[u]) to T ≫
    return T
```

Hai dòng màu đỏ chính là thay đổi so với thuật toán trước.

Khi thực thi giả mã này, ta sẽ dùng cấu trúc dữ liệu danh sách kề để biểu diễn đồ thị. Với mỗi đỉnh, ta sử dụng một danh sách liên kết lưu các đỉnh kề với nó. Code C:

```
+ expand source (#)
```

Phân tích thuật toán: Thao tác ExtractMin(H) mất thời gian $O(\log n)$ và thao tác DecreaseKey$(H,x,d[x])$ mất thời gian $O(\log n)$ nếu ta sử dụng Heap nhị phân và mất $O(1)$ nếu ta sử dụng Fibonacci Heap (https://en.wikipedia.org/wiki/Fibonacci_heap). Do đó, mỗi vòng lặp sẽ mất $O(\deg(u)\log(n)+\log n)$ nếu sử dụng Heap nhị phân mất $O(\deg(u)+\log n)$ nếu sử dụng Fibonacci Heap. Do đó, tổng thời gian của thuật toán là:

$$\sum_{u\in V} O(\deg(u)\log(n)+\log n) = O((m+n)\log n) \qquad (1)$$

nếu sử dụng Heap nhị phân, và tổng thời gian của thuật toán là:

$$\sum_{u\in V} O(\deg(u)+\log n) = O(m+n\log n) \qquad (2)$$

(http://www.cs.tau.ac.il/~zwick/grad-algo-13/mst.pdf)
nếu sử dụng Fibonacci Heap (https://en.wikipedia.org/wiki/Fibonacci_heap).

Final Remark Với đồ thị dầy, ta nên sử dụng thuật toán $O(n^2)$ của Prim, vì thực thi đơn giản mà thời gian chạy lại tối ưu. Với đồ thị thưa, ta có hai

lựa chọn: hoặc sử dụng thuật toán Prim, hoặc sử dụng thuật toán Kruskal. Trong thực tế, có lẽ không nên chọn Fibonacci Heap để thực thi Prim vì mặc dù thời gian tiệm cận nhỏ hơn, hằng số ẩn sau O khá lớn và Fibonacci Heap cũng rất khó thực thi. Sử dụng Heap nhị phân, ta sẽ thu được thuật toán với thời gian giống Kruskal. Tuy nhiên, hằng số ẩn sau Big O của Kruskal nhỏ hơn của Prim nhiều do nhân tử $\log n$ trong độ phức tạp là do bước sắp xếp. Do đó, ta nên chọn thực thi Kruskal với đồ thị thưa.

Code đầy đủ: Prim (http://www.giaithuatlaptrinh.com/wp-content/uploads/2016/05/prim.c) , Prim-and-Heap (http://www.giaithuatlaptrinh.com/wp-content/uploads/2016/05/prim_heap.c) .

Tham khảo

[1] Prim, Robert Clay. *Shortest Connection Networks and some Generalizations*. Bell System Technical Journal 36.6 (1957): 1389-1401
[2]Jarník, V. *O jistém problému minimálním" [About a certain minimal problem]*, Práce Moravské Přírodovědecké Společnosti (in Czech) 6: 57–63, 1930.
[3] Uri Zwick. *Notes on Minimum Spanning Tree* (http://www.cs.tau.ac.il/~zwick/grad-algo-13/mst.pdf) . Tel Aviv University, 2013.

In "global-min-cut" In "boruvka-algorithm"

Tags: binary heap, cut-property, minimum-spanning-tree, prim-algorithm

No comments

Comments feed for this article

Trackback link: http://www.giaithuatlaptrinh.com/wp-trackback.php?p=1175

Reply

Your email address will not be published. Required fields are marked *

Your comment

Name *

Email *

Website

Post Comment

Notify me of follow-up comments by email.

Notify me of new posts by email.