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.

‹ Tính tổng, tích modulo và bán tổng-- computing modulo sum, modulo product and semi-sum • Cây khung nhỏ nhất: thuật toán Prim --- Prim Algorithm ›

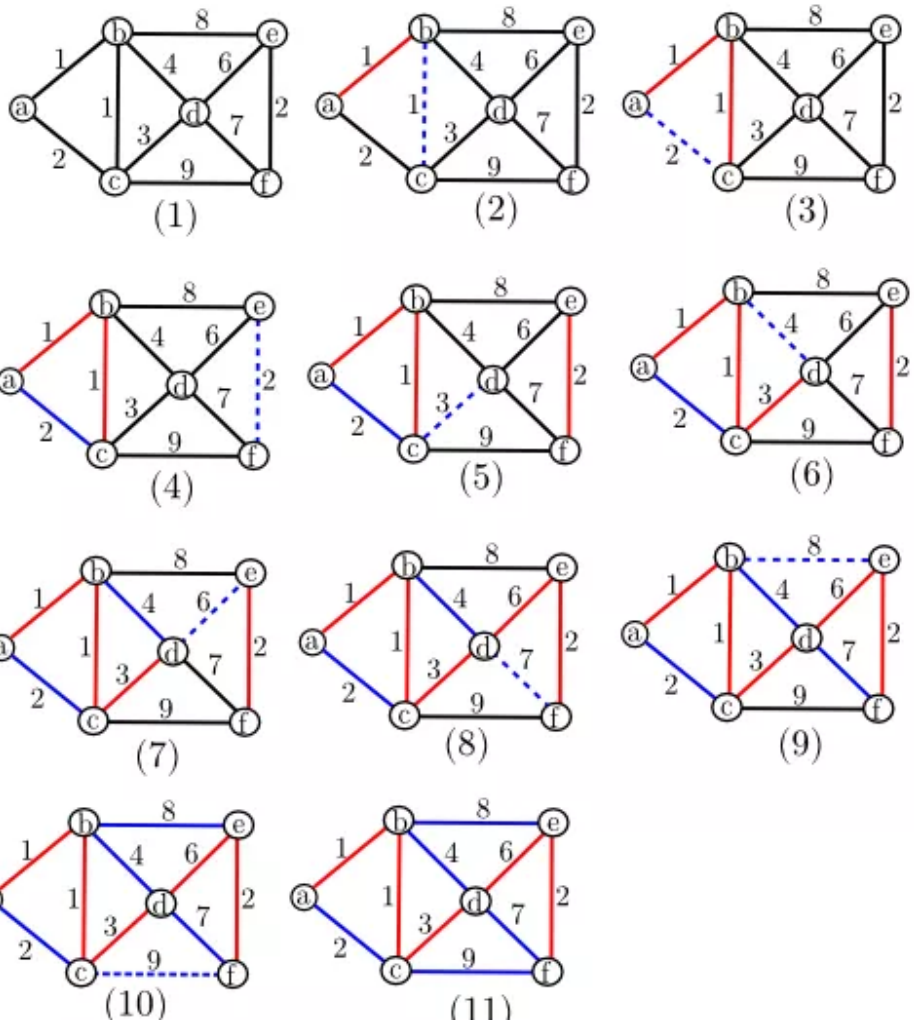
Cây khung nhỏ nhất: thuật toán Kruskal --- Kruskal Algorithm

May 22, 2016 in Uncategorized | 1 comment

Thuật toán Kruskal là một thuật toán tham lam. Thuật toán sẽ duy trì một đồ thị con phi chu trình T, gọi là một rừng (forest), của $G(V, E)$. Ban đầu, khởi tạo $T = V$, i.e, T là tập chỉ gồm các đỉnh mà không có cạnh nào cả. Tại mỗi bước, thuật toán sẽ cố gắng thêm cạnh vào T, để cuối cùng T sẽ là một cây khung. Do ta muốn tổng trọng số của T nhỏ nhất có thể, ta sẽ chọn cạnh cạnh có **trọng số** nhỏ nhất, gọi là e, trong số các cạnh **không** ở trong T và thêm vào T. Sẽ có hai trường hợp:

1. Nếu $T \cup \{e\}$ không có chu trình, ta sẽ thêm e vào T. Thao tác thêm e vào T sẽ tương đương với gộp hay cây trong T lại với nhau thành một cây mới.
2. Nếu $T \cup \{e\}$ có chu trình, ta sẽ thử cạnh tiếp theo.

Tính chất (6) cho chúng ta biết thêm cạnh theo thứ tự tăng dần của trọng số như vậy **có lẽ** sẽ cho chúng ta cây khung nhỏ nhất.
Ví dụ: các bước chạy của thuật toán với đồ thị cho trong hình trên. Các cạnh màu đỏ là các cạnh nằm trong cây khung nhỏ nhất. Các cạnh màu xanh là các cạnh đã được xét và không nằm trong cây khung. Các cạnh màu xanh gạch ngang là cạnh sẽ được xem xét đưa vào cây khung trong bước tiếp theo. Thứ tự các cạnh được xét là theo chiều tăng dần của trọng số.



Giả mã:

```
KRUSKAL(G(V, E), w):
    T ← V                    ≪ there is no edge in T ≫
    sort edges in E in ↑ order of weight
    let e_1, ..., e_m be edges after sorting
    for i ← 1 to m
        if T ∪ {e_i} has no cycle
            T ← T ∪ {e_i}
    return T.
```

Tạm thời chưa quan tâm đến thời gian thực thi KRUSKAL. Câu hỏi đầu tiên mà ta quan tâm đó là: thuật toán KRUSKAL có thực sự trả về cây khung nhỏ nhất? Để trả lời câu hỏi này, ta sẽ trả lời một loạt câu hỏi (dễ hơn) sau.

Câu hỏi 1: Đồ thị T đầu ra của KRUSKAL có phải là chu trình hay không?

Câu trả lời là **có** vì điều kiện trong dòng màu đỏ luôn đảm bảo điều này.

Câu hỏi 2: Đồ thị T đầu ra của KRUSKAL có phải là một cây hay không?

Ta đã biết T không có chu trình, do đó, nó sẽ là cây nếu liên thông (chỉ có duy nhất một thành phần liên thông). Giả sử đầu ra không liên thông, T sẽ có ít nhất hai thành phần liên thông (mỗi thành phần liên thông có thể

chỉ là 1 đỉnh). Chọn hai thành phần liên thông F_1, F_2 của T, sao cho tồn tại một cạnh của G nối hai thành phần này. Ta luôn chọn được vì G liên thông. Gọi e_k là cạnh có trọng số nhỏ nhất nối F_1, F_2. Cạnh e_k này không thuộc T vì nếu không F_1, F_2 sẽ là chỉ là một thành phần mà thôi. Nhưng khi kiểm tra điều kiện ở dòng màu đỏ, $T \cup \{e_k\}$ rõ ràng không có chu trình. Điều đó có nghĩa ta sẽ thêm e_k vào trong T ngay sau đó. Điều này trái với giả sử e_k không thuộc T. Do đó T phải liên thông, i.e, T là một cây.

Giờ ta sẽ chứng minh:

> **Theorem 1:** Đầu ra T của thuật toán KRUSKAL là cây khung có trọng số nhỏ nhất trong số các cây khung của $G(V, E)$.

Chứng minh: Gọi F là một cây khung nhỏ nhất của G sao cho số cạnh chung giữa F và T là lớn nhất. Nếu $F = T$ thì rõ ràng T là một cây khung nhỏ nhất. Do đó, giả sử $F \neq T$.

Gọi $T = \{t_1, t_2, \ldots, t_{n-1}\}$ và $F = \{f_1, f_2, \ldots, f_{n-1}\}$ lần lượt là thứ tự các cạnh của T và F sắp xếp theo chiều tăng của trọng số. Gọi i là chỉ số nhỏ nhất sao cho cạnh t_i và cạnh f_i là hai cạnh khác nhau. Theo cách chọn i, ta suy ra $i \leq n-2$ và $f_k = t_k, 1 \leq k \leq i-1$. Do đó, $\{t_1, \ldots, t_{i-1}\} \cup \{f_i\}$ không có chu trình.

Ta sẽ có $w(t_i) \leq w(f_i)$, vì nếu không, dòng màu đỏ trong thuật toán sẽ kiểm tra cạnh f_i **trước khi** kiểm tra cạnh t_i, và do $\{t_1, \ldots, t_{i-1}\} \cup \{f_i\}$ không có chu trình, f_i sẽ được thêm vào T trước t_i. Hay nói cách khác, f_i chính là cạnh thứ i của T; trái với giả thiết $f_i \neq t_i$.

Xét đồ thị $H = F \cup \{t_i\}$. H sẽ có một chu trình C. Chu trình C có hai tính chất sau:

1. C chứa t_i. Nếu không, C sẽ là một chu trình của F, trái với giả thiết F là một cây.
2. C chứa ít nhất một cạnh f_j sao cho $j \geq i$. Nếu không, C là tập con của $\{f_1, \ldots, f_{i-1}\} \cup t_i$; tập này cũng chính là tập $\{t_1, \ldots, t_i\}$. Hay nói cách khác C là tập con của T, trái với tính chất T là một cây.

Gọi $F' = H - \{f_j\}$; F' là một cây (tại sao?). Do $w(f_j) \geq w(f_i) \geq w(t_i)$, $w(F') \leq w(F)$. Từ đó suy ra F' cũng là một cây khung nhỏ nhất. Nhưng rõ ràng F' có số cạnh chung với T nhiều hơn F. Điều này trái với giả thiết F là cây khung nhỏ nhất có số cạnh chung nhiều nhất với T. Như vậy, $F = T$, i.e, T là cây khung nhỏ nhất.

Thực thi thuật toán Kruskal

Phần khó thực thi nhất của thuật toán KRUSKAL có lẽ là dòng màu đỏ. Theo tính chất (1) của cây khung, T có tối đa $n-1$ cạnh, do đó, $T \cup \{e\}$ có tối đa n cạnh. Dòng màu đỏ thực ra có thể quy về bài toán:

> Kiểm tra xem một đồ thị có chu trình hay không?

Ta có thể giải bài toán này bằng bất kì thuật toán duyệt đồ thị (http://www.giaithuatlaptrinh.com/?p=553) nào (BFS/DFS). Do đó, ta có thể

thực hiện dòng màu đỏ trong thời gian $O(n)$, qua đó thu được thuật toán $O(n^2)$. Tuy nhiên, ta muốn thực hiện nhanh hơn thế.

Ta nhận xét thấy trong mỗi bước trung gian, T là một rừng, i.e, mỗi thành phần liên thông của T là một cây. Phép kiểm tra xem $T \cup \{e\}$ có chu trình hay không tương đương với kiểm tra xem hai đầu mút, gọi là u, v, của e có thuộc cùng một cây trong rừng T hay không.

a. Nếu u, v thuộc cùng một cây, $T \cup \{e\}$ sẽ có chu trình.
2. Ngược lại, $T \cup \{e\}$ không có chu trình, và phép thêm e vào T sẽ tương đương với gộp 2 cây con của T thành một cây mới bằng cạnh e.

Giờ nếu biểu diễn mỗi cây trong T bằng cấu trúc tập hợp. Các đỉnh thuộc cùng một cây khi và chỉ khí nó nằm trong cùng một tập. Thao tác (a) sẽ tương đương với so sánh $\textsc{Find}(u)$ và $\textsc{Find}(v)$. Thao tác (b) sẽ tương đương với $\textsc{Union}(u, v)$. Các bạn xem thêm tại đây (http://www.giaithuatlaptrinh.com/?p=218).

Do đó, ta có thể viết lại thuật toán Kruskal như sau:

```
KruskalUnionFind(G(V, E), w):
    T ← V                          ≪ there is no edge in T ≫
    for each v ∈ V
        MakeSet(v)
    sort edges in E in ↑ order of weight
    let e_1, ..., e_m be edges after sorting
    for i ← 1 to m
        let u, v be two endpoints of e_i
        if Find(u) ≠ Find(v)
            T ← T ∪ {e_i}
            Union(u, v)
    return T.
```

Để thực thi giả mã trên, chúng ta có thể sử dụng biểu diễn dạng ma trận kề hay danh sách kề. Tuy nhiên, trong giả mã trên, không có truy vấn dạng "tìm hàng xóm của một đỉnh". Do đó, ta có thể sử dụng cấu trúc danh sách cạnh để biểu diễn đồ thị (gọi là danh sách chứ thực ra ta có thể dùng một mảng các struct, mỗi struct biểu diễn một cạnh). Phép biểu diễn này cho phép chúng ta có thể gọi thủ tục sắp xếp cạnh một cách dễ dàng. Trong Code dưới đây, mình sử dụng danh sách cạnh để cài đặt.

Code C:

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

Phân tích Sắp xếp các cạnh của đồ thị theo trọng số mất thời gian $O(m \log m) = O(m \log n)$ (vd sử dụng merge sort (http://www.giaithuatlaptrinh.com/?p=41)). Như ta đã biết, mỗi thao tác của cấu trúc Union-Find (kết hợp nén đường đi) mất thời gian $\alpha(n)$ trong đó $\alpha(.)$ là hàm Ackerman (https://en.wikipedia.org/wiki/Ackermann_function) ngược. Hàm này có giá trị ≤ 5 với mọi giá trị thực tế của n. Do đó, vòng lặp for (có chứa dòng màu đỏ) sẽ mất thời gian $m\alpha(n)$. Do đó, tổng thời gian của thuật toán là $O(m \log n + m\alpha(n)) = O(m \log n)$.

Theorem 2: Thuật toán KruskalUnionFind tìm cây khung nhỏ nhất của đồ thị $G(V, E)$ trong thời gian $O(E \log V)$.

Nhắc lại cấu trúc Union-Find

Trong cấu trúc Union-Find, giả sử ta có một tập $X[1, 2, \ldots, n]$ gồm n phần tử. Ta muốn có một cấu trúc dữ liệu biểu diễn các tập con không giao nhau (disjoint subsets) của X. Cấu trúc này hỗ trợ các thao tác sau:

1. **MakeSet(x)**: Tạo tập hợp $\{x\}$.
2. **Find(x)**: Trả lại id của tập hợp chứa x.
3. **Union(x, y)**: Thay thế hai tập hợp A, B lần lượt chứa x và y bằng tập hợp $A \cup B$. Ở đây ta giả sử phép Union luôn gộp 2 phần tử thuộc hai tập khác nhau.

Mỗi tập hợp con của X, ta sẽ dùng một cây để biểu diễn. Mảng $P[1, \ldots, n]$ sẽ biểu diễn cấu trúc cây này. i.e, $P[v] = u$ nếu như u là nút cha của v trong cây (biểu diễn tập hợp chứa u, v). Do đó, để kiểm tra xem hai phần tử u, v có trong cùng một cây hay không, ta chỉ việc tìm xem nút gốc của cây biểu diễn có trùng nhau hay không.

Trong thực thi dưới đây, ta sẽ sử dụng nén đường đi. Về cơ bản, khi ta đã tìm được gốc, gọi là x, của cây chứa nút u, ta đặt luôn $P[u] = x$, để các phép tìm gốc sau đó của u sẽ nhanh hơn.

Để hợp 2 cây, ta sẽ hợp theo hạng (rank) và ta sẽ dùng một mảng $R[1, 2 \ldots, n]$. Về mặt trực quan, khi gộp 2 cây, ta nên gộp cây có độ sâu (depth) nhỏ hơn vào cây có độ sâu lớn hơn vì làm như vậy, chiều sâu của cây không tăng lên tối đa 1 sau khi gộp, và do đó, các phép tìm gốc sau này sẽ nhanh hơn. Hạng là một chỉ số, về mặt trực quan (nhưng không hoàn toàn), đo xem cây nào có chiều sâu nhỏ hơn cây nào. Do đó, ta sẽ gộp cây của nút có hạng nhỏ hơn vào cây của nút có hạng lớn hơn.

Giả mã:

```
MakeSet(x):
    P[x] ← x
    R[x] ← 0
```

```
Find(x):
    if P[x] ≠ x
        P[x] ← Find(P[x])
    return P[x]
```

```
Union(x, y):
    x̄ ← Find(x)        [[the root of A]]
    ȳ ← Find(y)        [[the root of B]]
    if R[x̄] > R[ȳ]
        P[ȳ] ← x̄
    else
```

$P[\bar{x}] \leftarrow \bar{y}$
if $R[\bar{x}] = R[\bar{y}]$
$R[\bar{y}] \leftarrow R[\bar{y}] + 1$

Code C:

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

Code đầy đủ: Kruskal-mst (http://www.giaithuatlaptrinh.com/wp-content/uploads/2016/05/kruskal_mst.c) .

Tham khảo

[1] Kruskal, Joseph B. *On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem.* Proceedings of the American Mathematical society 7.1 (1956): 48-50.
[2] Prim, Robert Clay. *Shortest Connection Networks and some Generalizations.* Bell System Technical Journal 36.6 (1957): 1389-1401.
[3] Borůvka, Otakar (1926). *Příspěvek k řešení otázky ekonomické stavby elektrovodních sítí (Contribution to the Solution of a Problem of Economical Construction of Electrical Networks)*. Elektronický Obzor (in Czech) 15: 153–154.
[4] Nešetřil, Jaroslav, Eva Milková, and Helena Nešetřilová. *Otakar Borůvka on minimum spanning tree problem translation of both the 1926 papers, comments, history.* Discrete Mathematics 233.1 (2001): 3-36.
[5] Karger, David R.; Klein, Philip N.; Tarjan, Robert E. *A Randomized Linear-time Algorithm to Find Minimum Spanning Trees*. Journal of the ACM 42 (2): 321–328, 1995.
[6] Jeff Erickson. Lecture Notes on Minimum Spanning Tree (http://jeffe.cs.illinois.edu/teaching/algorithms/notes/20-mst.pdf) . UIUC, 2014.

Tổng quan về cây khung nhỏ nhất. (http://www.giaithua...p=1266)
June 17, 2016
In "minimum-spanning-tree"

Cây khung nhỏ nhất: thuật toán Borůvka -- Borůvka Algorithm (http://www.giaithua...p=1204)
June 1, 2016
In "boruvka-algorithm"

Cây khung nhỏ nhất: thuật toán KKT -- KKT Algorithm (http://www.giaithua...p=1297)
July 8, 2016
In "boruvka-algorithm"

Tags: greedy algorithm, Kruskal, minimum-spanning-tree, union-find

1 comment

Comments feed for this article

Trackback link: http://www.giaithuatlaptrinh.com/wp-trackback.php?p=1140

138bet on September 4, 2016 at 11:55 am

Thank you share it

Reply

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.