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.

‹ Lát cắt cực tiểu I: Thuật toán Stoer-Wagner -- MinCut I: Stoer-Wagner Algorithm • Giới thiệu về P và NP --- P vs NP ›

Thuật toán Kosaraju tìm thành phần liên thông mạnh -- Kosaraju's Algorithm

December 7, 2016 in Uncategorized | No comments

Trong bài này, chúng ta sẽ tìm hiểu thuật toán Kosaraju; thuật toán dễ hiểu nhất trong số các thuật toán tìm thành phần liên thông mạnh. Thuật toán này về cơ bản sẽ duyệt qua đồ thị 2 lần, do đó có thời gian tiệm cận $O(V+E)$. So với các thuật toán tìm thành phần liên thông mạnh khác như thuật toán cầu-khớp của Tarjan thì thuật toán Kosaraju chậm hơn, nhưng có cùng thời gian tiệm cận.

Đồ thị trong phần này của chúng ta là đồ thị có hướng và có thể có cung song song ngược chiều. Mình khuyến khích bạn đọc xem lại một số khái niệm cơ bản về đồ thị (http://www.giaithuatlaptrinh.com/?p=553) có hướng. Ta sẽ kí hiệu đồ thị đầu vào là $G(V,\vec{E})$ và kí hiệu đồ thị có hướng thu được bằng cách đảo chiều các cạnh của đồ thị $G(V,\vec{E})$. là $H(V,\overleftarrow{E})$. Ta gọi $H(V,\overleftarrow{E})$ là đồ thị ngược (reversed graph) của đồ thị $G(V,\vec{E})$.

Một đồ thị có hướng được gọi là **liên thông mạnh** (strongly connected) nếu tồn tại một đường đi có hướng từ u tới v, với mọi cặp đỉnh u,v của đồ thị. Một đồ thị con C của đồ thị có hướng $G(V,\vec{E})$ được gọi là một thành phần liên thông mạnh (strongly connected component) nếu nó liên thông mạnh và tối đại (maximal), i.e, **không** tồn tại một đồ thị con liên thông mạnh D của G mà C là đồ thị con của D. Xem ví dụ minh họa trong Figure 1.

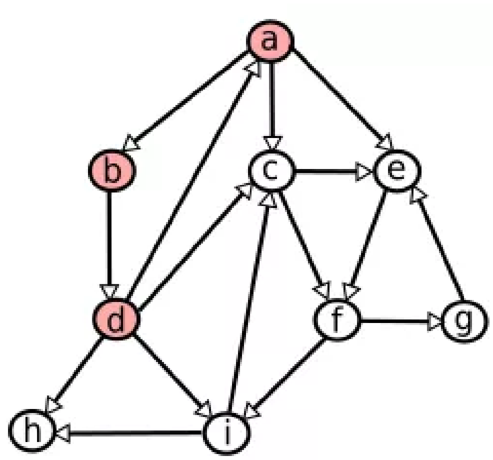

Figure 1: Ba đỉnh $\{a,b,d\}$ tạo nên một thành phần liên thông mạnh của

đồ thị. Ba đỉnh $\{e, f, g\}$ không tạo nên thành phần liên thông mạnh vì tồn tại một đồ thị con liên thông mạnh lớn hơn chứa ba đỉnh này.

> **Observation 1:** Thành phần liên thông mạnh của một đồ thị có hướng không thay đổi nếu ta đảo chiều tất cả các cung của đồ thị.

Trong nhận xét trên, không thay đổi được hiểu theo nghĩa là **tập đỉnh** của thành phần liên thông mạnh không thay đổi. Hay nói cách khác, nếu $X \subseteq V$ là tập đỉnh của một thành phần liên thông mạnh trong $G(V, \vec{E})$ thì tập đỉnh này cũng là tập đỉnh của một thành phần liên thông mạnh trong $H(V, \overleftarrow{E})$. Chứng minh nhận xét trên coi như bài tập cho bạn đọc.

Thuật toán Kosaraju

Thuật toán Kosaraju gồm hai bước chính (xem thêm giả mã dưới đây):

1. Duyệt đồ thị $G(V, \vec{E})$ sử dụng duyệt theo chiều sâu (http://www.giaithuatlaptrinh.com/?p=553) DFS. Trong bước này, thuật toán còn duy trì một ngăn xếp S. Mỗi đỉnh v sau khi đã duyệt xong thì sẽ được đẩy vào S. Một đỉnh được gọi là duyệt xong khi mà mọi hàng xóm được thăm sau nó đã được đánh dấu là đã duyệt (visited). Nói cách khác, một đỉnh được duyệt xong khi mà mọi đỉnh nằm trong cây con DFS của nó đã được duyệt.
2. Bước này ta in ra các thành phần liên thông mạnh qua thủ tục PrintConnectedComponent$(H(V, \overleftarrow{E}), S)$. Chú ý ở đây ta sẽ thao tác trên đồ thị ngược của G. Ta sẽ bắt đầu từ đỉnh v nằm trên đầu của ngăn xếp S. Ta tìm các đỉnh nằm trong tầm với (reachable) của v trong H (thủ tục Reachable(H, v)). Một đỉnh u được gọi là nằm trong tầm với của v nếu như tồn tại một đường đi có hướng từ v tới u trong H. Tập các đỉnh này (bao gồm cả v) sẽ là một thành phần liên thông mạnh của G chứa v (ta sẽ chứng minh ở dưới đây). Sau khi đã tìm được tập đỉnh này rồi, ta xóa nó khỏi ngăn xếp S và đồ thị H. Chúng ta cần phải cẩn thận trong việc thực thi phép xóa một tập đỉnh ra khỏi S và H để thuật toán cuối cùng vẫn là tuyến tính. Ta sẽ bàn thêm vấn đề này ở dưới đây.

```
Kosaraju(G(V, E→)):
    S ← EmptyStack()
    mark all vertices of V unvisited
    for each v ∈ V
        if v is unvisited
            Dfs(G, v, S)
    PrintConnectedComponent(H(V, E←), S)
```

```
Dfs(G(V, E→), v, S):
    mark v visited
    for each v → u ∈ E
        if u is unvisited
            Dfs(G, u, S)
    Push(v, S)        ≪ push v to stack S ≫
```

<u>PrintConnectedComponent</u>$(H(V, \overleftarrow{E}), S)$:
  while $S \neq \emptyset$
    $v \leftarrow$ Pop(S)
    $CC \leftarrow$ Reachable(H, v)
    $S \leftarrow S \setminus CC$ $\ll$ remove CC from stack $S \gg$
    $H \leftarrow H \setminus CC$ $\ll$ remove CC from graph $H \gg$
    Print CC

Trước hết, ta hãy xem ví dụ áp dụng của thuật toán Kosaraju.

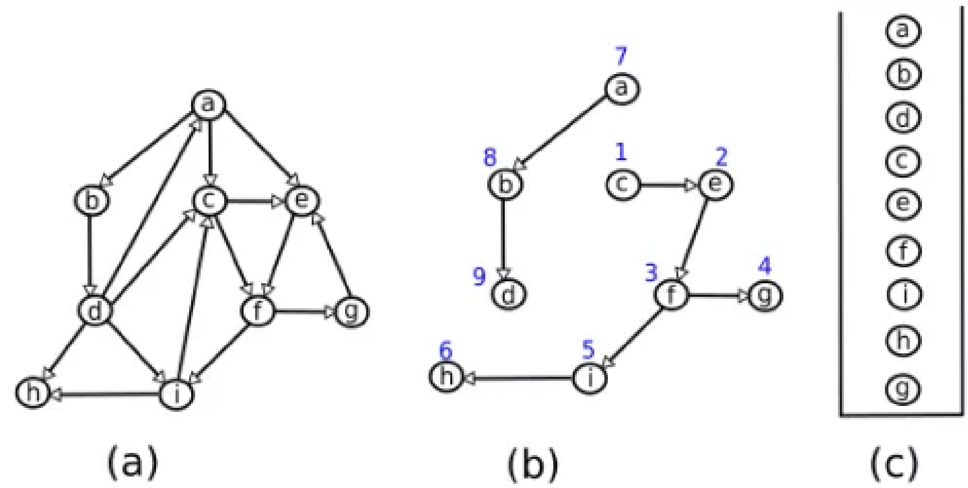

Figure 2: Bước 1 của thuật toán áp dụng cho đồ thị đầu vào ở hình (a). Hình (b) là rừng DFS, thu được bằng cách áp dụng thuật toán DFS bắt đầu từ đỉnh c; con số đính bên cạnh mỗi đỉnh là thứ tự đỉnh được thăm bởi DFS. Hình (c) là thứ tự các đỉnh được đẩy vào ngăn xếp khi thăm Dfs.

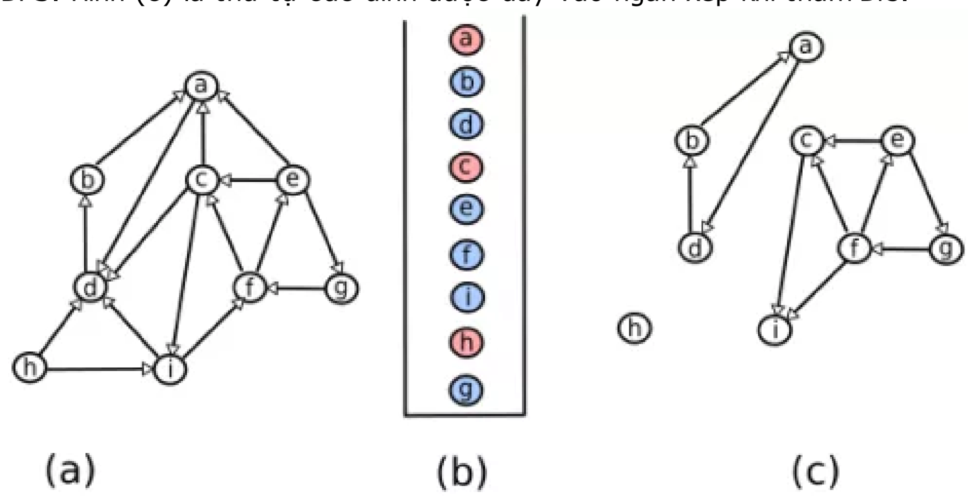

Figure 3: Bước 2 của thuật toán áp dung cho đồ thị ngược (a) của đồ thị trong Figure 2(a). Các đỉnh màu hồng $\{a, c, h\}$ lần lượt là các đỉnh được lấy ra khỏi ngăn xếp trong vòng lặp while của thủ tục PrintConnectedComponent qua phương thức Pop. Các đỉnh còn lại sẽ bị xóa khỏi ngăn xếp sau khi thành phần liên thông mạnh của nó đã được xác định. Hình (c) là các thành phần liên thông mạnh tương ứng.

Để thực thi thủ tục Reachable(H, v) ở trên, ta chỉ cần thực hiện duyệt đồ thị H từ đỉnh v (sử dụng Dfs hoặc Bfs). Để xóa một tập ra khỏi ngăn xếp

S, ta sẽ **không** duyệt qua ngăn xếp, vì như vậy sẽ mất thời gian $O(V)$, do đó, tổng thời gian thuật toán sẽ là $O(KV)$ với K là số thành phần liên thông mạnh. Ta chỉ cần **đánh dấu** tập đỉnh đó bị xóa sử dụng một mảng đánh dấu mà thôi. Do đó, "xóa" tập CC_c chỉ mất thời gian $|CC_c|$. Xóa tập đỉnh CC_c ra khỏi đồ thị H ta làm tương tự. Như vậy, về cơ bản, thuật toán Kosaraju chỉ duyệt qua đồ thị 2 lần. Do đó, thủ tục PrintConnectedComponent có thể được viết lại như sau:

```
PrintConnectedComponent(H(V, ←E), S):
    while S ≠ ∅
        v ← Pop(S)
        if v is not deleted
            DfsAndPrint(H, v)
```

```
DfsAndPrint(H(V, ←E), v):
    Print v
    mark v deleted
    for each v → u ∈ ←E
        if u is not deleted
            DfsAndPrint(H, u)
```

Code C:

+ expand source (#)

Từ phần tích ở trên, ta có:

Theorem 1: Thuật toán Kosaraju có thể được thực thi trong thời gian $O(V+E)$.

Tính đúng đắn của thuật toán Kosaraju

Phần này có lẽ là phần khó nhất của bài này. Gọi v là một đỉnh được lấy ra khỏi ngăn xếp bên trong vòng lặp while và $CC_c = \text{Reachable}(H, v)$ trong thủ tục PrintConnectedComponent$(H(V, \overleftarrow{E}), S)$. Gọi X là thành phần liên thông mạnh của G chứa đỉnh v. Ta sẽ chứng minh $X = CC_c$.

Theo Observation 1, mọi đỉnh nằm trong X sẽ nằm trong tầm với của v trong H và ngược lại. Do đó, mọi đỉnh trong X vẫn chưa bị xóa khỏi H tại thời điểm ta bắt đầu thăm v. Từ đó ta suy ra $X \subseteq CC_c$.

Ta chỉ còn phải chứng minh $CC_c \subseteq X$. Giả sử tồn tại một đỉnh $h \in CC_c \setminus X$. Theo định nghĩa, tồn tại một đường đi từ v tới h trong H. Do đó, tồn tại một đường đi từ h tới v trong G vì H là đồ thị ngược của G. Do $h \notin X$, không tồn tại đường đi từ v tới h trong G. Ta xét hai trường hợp

1. Đỉnh h được thăm **sau** v trong bước 1. Dễ thấy đỉnh v sẽ bị đưa vào ngăn xếp S trước h. Do đó, h sẽ được lấy ra khỏi ngăn xếp trước v. Hay nói

cách khác, h đã bị xóa tại thời điểm ta thăm v, trái với giải thiết $h \in CC_c$.

2. Đỉnh h được thăm **trước** v trong bước 1. Do tồn tại một đường đi từ h tới v trong G, đỉnh h sẽ được duyệt xong khi và chỉ khi đỉnh v được duyệt xong. Nhắc lại, một đỉnh được gọi là duyệt xong khi mà mọi hàng xóm được thăm sau nó đã được đánh dấu là đã duyệt (visited). Do đó, đỉnh v cũng sẽ vẫn bị đưa vào ngăn xếp S trước h. Lập luận tương tự như trường hợp 1, ta suy ra h đã bị xóa tại thời điểm ta thăm v, trái với giải thiết $h \in CC_c$.

Hai trường hợp trên cho ta biết h không tồn tại, i.e, $X = CC_c$. Do đó, thuật toán Kosaraju cho ta tập các thành phần liên thông mạnh của đồ thị $G(V, \vec{E})$.

Code đầy đủ: Kosaraju (http://www.giaithuatlaptrinh.com/wp-content/uploads/2016/12/Kosaraju.c) .

Tham khảo

[1] R. S. Kosaraju. *Unpublished*. 1978.
[2] M. C. Chu-Carroll. Good Math, Bad Math (http://scienceblogs.com/goodmath/2007/10/30/computing-strongly-connected-c/) . Accessed 11 Dec 2016.

Tags: graph algorithm, graph intro, Kosaraju, strongly-connected-component

No comments

Comments feed for this article

Trackback link: http://www.giaithuatlaptrinh.com/wp-trackback.php?p=1680

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.