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.

‹ Heap nhị phân và thuật toán Heapsort -- Binary Heap and Heapsort • Thuật toán Bellman-Ford tìm đường đi ngắn nhất từ một đỉnh trong đồ thị có trọng số âm -- Bellman Ford Algorithm ›

Thuật toán Dijkstra tìm đường đi ngắn nhất từ một đỉnh-- Dijkstra Algorithm

January 12, 2016 in Uncategorized | No comments

Trong bài này chúng ta sẽ tìm hiểu một thuật toán kinh điển tìm đường đi ngắn nhất từ một đỉnh tới mọi đỉnh khác trong đồ thị có hướng, có trọng số. Thuật toán này cũng áp dụng được cho đồ thị vô hướng nếu như ta coi mỗi cạnh vô hướng uv là 2 cung có hướng ngược chiều nhau $u \to v$ và $v \to u$ với cùng trọng số của cạnh uv. Do đó, ta chỉ phát biểu thuật toán trong bài này cho đồ thị có hướng mà thôi.

Các khái niệm cơ bản về cạnh, đường đi, chu trình, etc., mình đã viết ở bài trước (http://www.giaithuatlaptrinh.com/?p=553) nên ở đây mình không nhắc lại nữa. Ta gọi $w : \vec{E} \to \mathbb{R}^+$ là một *hàm trọng số* gán cho mỗi cung $(u \to v)$ một trọng số $w(u \to v)$ không âm. Đôi khi ta cũng có thể gọi trọng số của một cung là chiều dài của cung đó. *Chiều dài của một đường đi* $P(u, v)$ giữa hai điểm được định nghĩa là tổng chiều dài của các cạnh nằm trên đường đi đó và ta kí hiệu là $w(P(u, v))$. Bài toán mà chúng ta sẽ tìm hiểu trong bài này như sau:

Problem 1: Cho một đồ thị có hướng $G(V, \vec{E})$, một hàm trọng số $w : \vec{E} \to \mathbb{R}^+$ và một đỉnh s. Tìm đường đi ngắn nhất từ s tới mọi đỉnh trong G.

Để đơn giản, ta sẽ dùng V và E lần lượt để chỉ số đỉnh và số cạnh của đồ thị. Năm 1959, Dijkstra[1] công bố *thuật toán Dijkstra* giải bài toán này trong thời gian $O(V^2)$. Thuật toán Dijkstra là tối ưu với đồ thị dày ($E = O(V^2)$) nhưng với đồ thị thưa thì thuật toán này khá chậm. Tuy nhiên, ý tưởng của Dijkstra sau này đã được cải tiến [3] và có thời gian chạy $O(E + V \log V)$. Trong bài này, chúng ta sẽ tìm hiểu một phiên bản thực thi thuật toán đó sử dụng Heap nhị phân (http://www.giaithuatlaptrinh.com/?p=736) trong thời gian $O((V + E) \log V)$. Cuối bài, mình sẽ thảo luận ngắn gọn thuật toán $O(E + V \log V)$ với Fibonacci Heap (https://en.wikipedia.org/wiki/Fibonacci_heap) của Fredman và Tarjan.

Theorem 1: Thuật toán Dijkstra tìm đường đi ngắn nhất từ một đỉnh tới mọi đỉnh khác trong đồ thị với thời gian $O((V + E) \log V)$.

Tính chất của đường đi ngắn nhất

Gọi $\delta(s,v)$ là khoảng cách ngắn nhất từ s tới v trong G và $SP(s,v)$ là một đường đi ngắn nhất bất kì từ s tới v. Ta có:

Lemma 1: Đường đi con của $SP(s,v)$ từ s tới một đỉnh $u \in SP(s,v)$ cũng là đường đi ngắn nhất từ s tới u.

Chứng minh: Gọi $P(s,u)$ là đường đi con của $SP(s,v)$ từ s tới u. Giả sử tồn tại một đường đi khác $P'(s,u)$ sao cho $w(P'(s,u)) < w(P(s,u))$. Thay thế đường đi $P(s,u)$ trong $SP(s,v)$ bằng đường đi $P'(s,u)$, ta sẽ thu được đường đi khác từ s tới v với chiều dài nhỏ hơn, trái ngược với định nghĩa $SP(s,v)$ là đường đi ngắn nhất.

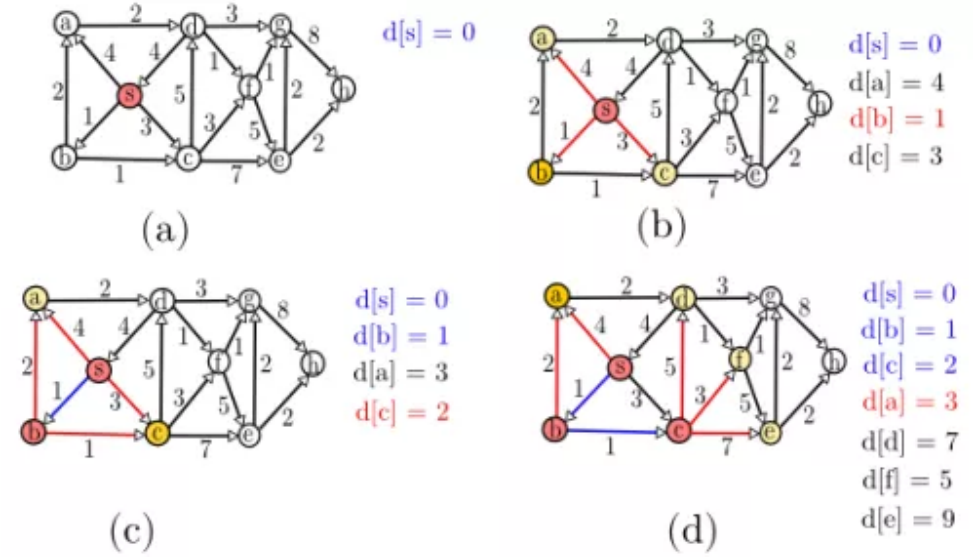
Thuật toán Dijkstra

Experimental Thought Trước hết ta xét s và các hàng xóm của nó, gọi là tập $N(s)$. Gọi u là một hàng xóm với độ dài cạnh $w(s \to u)$ nhỏ nhất. Khoảng cách ngắn nhất từ s tới u là bao nhiêu? Câu trả lời chính là $w(s \to u)$, vì nếu không, đường đi ngắn nhất từ s tới u đó phải đi qua một hàng xóm khác của s, và do đó, sẽ dài hơn $w(s \to u)$. Từ đó ta suy ra $\delta(s,u) = w(s \to u)$. Xem hình (b) ở hình dưới đây. Đỉnh được tô màu vàng đậm (đỉnh b) là đỉnh có trọng số cạnh từ s nhỏ nhất.

Tiếp theo ta xét tập các hàng xóm của cả s và u (u chính là b trong hình minh họa (c)), gọi là tập $N(s,u)$. Với mỗi $v \in N(s,u)$, ta gọi $d[v] = \min(w(s \to v), \delta(s,u) + w(u \to v))$. Dễ thấy nếu $d[v] = w(s \to v)$, độ dài cạnh $w(s \to v)$ sẽ nhỏ hơn độ dài đường đi $s \to u \to v$, và ngược lại. Trong số các đỉnh của tập $N(s,u)$, gọi v là đỉnh có $d[v]$ nhỏ nhất, i.e, $v = \text{argmin}_{x \in N(s,u)}(d[x])$. Câu hỏi cũ vẫn là khoảng cách ngắn nhất từ s tới v là bao nhiêu? Dễ dàng thấy rằng $\delta(s,v) = d[v]$ với cùng lí do như trên. Xem hình (c) trong hình dưới đây. Đỉnhh được tô màu hồng đậm là các đỉnh đã tìm được đường đi ngắn nhất. Đỉnh c là đỉnh có $d[.]$ nhỏ nhất, do đó, $d = \delta(s,c)$.

Tiếp theo ta lại xét $N(s,u,v)$, blah. blah...(Xem hình (d)) Cứ tiếp tục lặp lại như vậy, ta sẽ tìm ra được đường đi ngắn nhất (chính xác phải là độ dài của đường đi ngắn nhất) từ s tới mọi đỉnh trong đồ thị. Đó chính là tư

tưởng của thuật toán Dijkstra.



Giả mã "thô" của thủ tục trên như sau:

DIJKSTRA$(G(V,\vec{E}),w,s)$:
  for $v \leftarrow 1$ to V
    $d[v] \leftarrow +\infty$
  $d[s] \leftarrow 0$
  for every neighbor v of s
    $d[v] \leftarrow w(s \rightarrow v)$
  $B \leftarrow V \backslash \{s\}$ $\ll B$ *contains all vertices except* $s \gg$
  repeat
    $u \leftarrow \text{argmin}_{x \in B} d[x]$
    $B \leftarrow B \setminus \{u\}$ $\ll d[u] = \delta(s,u) \gg$
    for every neighbor v of u
      $d[v] \leftarrow \min(d[v], d[u] + w(u \rightarrow v))$
  until $B = \emptyset$

Ta có:

Theorem 2: Với mỗi đỉnh u, giá trị $d[u]$ sau khi thực hiên DIJKSTRA($G(V,\vec{E}),w,s$) là đường đi ngắn nhất từ s tới u.

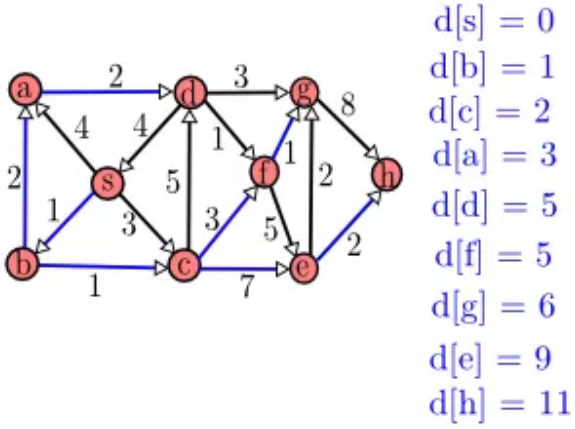
Chứng minh Nhận thấy mỗi bước trong vòng lặp, ta sẽ lấy ra đúng một đỉnh khỏi tập B, do đó vòng lặp này thực hiện $n-1$ lần, trong đó $n = V$ là số đỉnh. Gọi $u_1, u_2, \ldots, u_{n-1}$ là thứ tự các đỉnh được lấy ra khỏi B theo thuật toán trên. Theo thảo luận ở phần experimental thought, $d[u_1] = \delta(s,u_1)$ do u_1 là đỉnh có $w(s \rightarrow u_1)$ nhỏ nhất trong số các hàng xóm của s.

Gọi $d^i[v]$ là giá trị $d[v]$ ngay trước vòng lặp i của thuật toán. Ta có $d^i[v]$ chính là chiều dài của đường đi ngắn nhất từ s tới v đi qua một hoặc nhiều (hoặc 0) điểm trong tập $\{u_1, u_2, \ldots, u_{i-1}\}$. Vì u_i là đỉnh được lấy ra khỏi B ở bước thứ i, ta có:

$$d^i[u_i] \leq d^i[u_j] \text{ với mọi } i+1 \leq j \leq n-1 \qquad (1)$$

Giả sử $d^i[u] \neq \delta(s, u_i)$, đường đi ngắn nhất từ s tới u_i, $SP(s, u_i)$, sẽ phải đi qua một đỉnh nào đó không thuộc $\{u_1, \ldots, u_{i-1}\}$. Gọi x là điểm đầu tiên trên đường đi từ s tới u thỏa mãn tính chất đó. Theo Lemma 1, đường đi con của $SP(s, u_i)$ từ s tới x sẽ là đường đi ngắn nhất từ s tới x đi qua các điểm $\{u_1, \ldots, u_{i-1}\}$. Do đó $d^i[x] = \delta(s, x)$. Từ đó ta suy ra $d^i[x] < d^i[u]$ vì $\delta(s, x) \leq w(SP(s, u_i)) < d^i[u]$. Điều này trái với bất đẳng thức (1).

Áp dụng thuật toán Dijkstra với hình trên, ta thu được (cây) đường đi ngắn nhất trong hình sau:



Các cung màu xanh là các cung của cây đường đi ngắn nhất. thứ tự các đỉnh được lấy ra từ B sẽ có chiều tăng dần của khoảng cách $\delta(s, .)$.

Thực thi thuật toán Dijkstra

Bây giờ ta sẽ thực thi thuật toán trong giả mã ở trên. Như đã phân tích, vòng lặp repeat/until thực hiện $V - 1$ lần lặp. Hai thao tác tốn kém nhất của mỗi lần lặp là:

1. Lấy đỉnh u có giá trị $d[u]$ nhỏ nhất ra khỏi B
2. Cập nhật nhãn $d[v]$ của các hàng xóm v của u.

Ta có thể thực thi tập hợp B mảng. Do đó, thao tác đầu tiên có thể thực hiện trong thời gian $O(V)$ còn thao tác thứ 2 có thể thực hiện trong thi gian $O(\deg(u))$ trong đó $\deg(u)$ là bậc của đỉnh u trong G (nếu sử dụng cấu trúc danh sách kề). Như vậy, ta có thể thực thi thuật toán trong thời gian:

$$O(V^2 + \sum_{u \in G} \deg(u)) = O(V^2 + E) = O(V^2) \qquad (2)$$

Tuy nhiên, nếu biểu diễn B bằng một Heap nhị phân (http://www.giaithuatlaptrinh.com/?p=736), thao tác đầu tiên chính là **ExtractMin** và thao tác thứ 2 là **DecreaseKey**. Do mỗi thao tác **ExtractMin** và **DecreaseKey** mất thời gian $O(\log n)$, thời gian để thực hiện thuật toán Dijkstra với Heap nhị phân là:

$$O(V \log V + \sum_{u \in G} \log(V) \deg(u)) = O((V + E) \log V) \qquad (3)$$

Từ đó ta suy ra Theorem 1.

Giả mã của thuật toán:

```
DijkstraHeapA(G(V, E), w, s):
    for v ← 1 to V
        d[v] ← +∞
    d[s] ← 0
    for every neighbor v of s
        d[v] ← w(s → v)
    B ← BuildHeap(V \ {s})          ≪ B contains all vertices except s ≫
    repeat
        u ← ExtractMin(B)           ≪ d[u] = δ(s, u) ≫
        for every neighbor v of u
            d[v] ← min(d[v], d[u] + w(u → v))
            DecreaseKey(B, v, d[v])
    until B = ∅
```

Remark Nếu ta biểu diễn B bằng Fibonacci Heap (https://en.wikipedia.org/wiki/Fibonacci_heap), thao tác **ExtractMin** có thời gian $O(\log n)$ (khấu trừ) và thao tác **DecreaseKey** có thời gian $O(1)$. Do đó, tổng thời gian của thuật toán Dijkstra với Fibonacci Heap là:

$$O(V \log V + \sum_{u \in G} \deg(u)) = O((V \log V + E) \qquad (4)$$

Để in ra đường đi ngắn nhất, mỗi khi cập nhật lại giá trị $d[v] \leftarrow \min(d[v], d[u] + w(u \rightarrow v))$ trong thuật toán trên, ta sẽ đánh dấu u là hàng xóm làm thay đổi nhãn $d[v]$ của v. Ta sẽ dùng một mảng P và đánh dấu $P[v] = u$. Như vậy, thuật toán cuối cùng như sau:

```
DijkstraHeapB(G(V, E), w, s):
    for v ← 1 to V
        d[v] ← +∞
    d[s] ← 0
    P[s] ← s
    for every neighbor v of s
        d[v] ← w(s → v)
        P[v] ← s
    B ← BuildHeap(V \ {s})          ≪ B contains all vertices except s ≫
    repeat
        u ← ExtractMin(B)           ≪ d[u] = δ(s, u) ≫
        for every neighbor v of u
            if d[v] > d[u] + w(u → v))
                d[v] ← d[u] + w(u → v)
                P[v] ← u
            DecreaseKey(B, v, d[v])
    until B = ∅
```

```
FindReverseShortestPath(G(V, E), w, s, t):
    DijkstraHeapB(G, w, s)
    print t
    while P[t] ≠ t
```

$t \leftarrow P[t]$
print t

Code C với biểu diễn ma trận kề:

+ expand source (#)

Nhắc lại Heap nhị phân

Chúng ta sẽ phải sửa đổi lại một chút mã của Heap nhi phân (http://www.giaithuatlaptrinh.com/?p=736) trong bài trước để áp dụng vào thuật toán Dijkstra. Trong bài trước, chúng ta sử dụng mảng $H[1, 2, \ldots, n]$ để thực thi Heap, trong đó phần tử $H[i]$ lưu **giá trị khóa** tương ứng với nút thứ i. Do đó, khi cập nhật một giá trị khóa sử dụng **DecreaseKey** ta cần phải có địa chỉ của nút cần cập nhật và giá trị khóa mới của nút đó.

Với thuật toán Dijkstra, ta vẫn sử dụng mảng $H[1, 2, \ldots, n]$ để thực thi Heap. Tuy nhiên, mỗi phần tử $H[i]$ sẽ lưu một đỉnh của đồ thị (chứ không phải khóa) còn khóa sẽ được lưu trong mảng d. Như vậy, khóa của nút thứ i của Heap là $d[H[i]]$. Để thực hiện **DecreaseKey**, như đã nói ở đoạn văn trước, ta sẽ lưu một mảng $pos[1, 2, \ldots, V]$ trong đó $pos[v]$ sẽ lưu vị trí của đỉnh $v \in G$ trong Heap H. Hay nói cách khác,

$$pos[v] = i \text{ khi và chỉ khi } H[i] = v \qquad (4)$$

Giả mã của Heap:

DecreaseKey(Heap H, Vertex u, Key K):
$D[u] \leftarrow K$
UpHeapify$(pos[u])$

ExtractMin(Heap H):
$n \leftarrow$ length of the heap H
$tmp \leftarrow H[1]$
$H[1] \leftarrow H[n]$
$pos[H[n]] \leftarrow 1$ *≪ update the position of the vertex $H[n]$ ≫*
$n \leftarrow n - 1$
DownHeapify(1)
return tmp

DownHeapify(Heap Node u):
$m \leftarrow 2 * u$ *≪ v is the left child of u ≫*
if $m \leq n$ *≪ u is not a leaf≫*
if $d[H[m]] > d[H[2 * u + 1]]$
$m \leftarrow 2 * u + 1$
if $d[H[u]] > d[H[m]]$
swap $H[u] \leftarrow H[m]$
update pos $H[u]$ and $H[m]$ after swap
DownHeapify(m)

```
PARENT( u):
    if u is even
        return u/2
    else
        return (u − 1)/2
```

```
BUILDHEAP(V \ {s}):
    for i ← 1 to s − 1
        H[i] ← i
        pos[i] ← i
    for i ← s + 1 to V
        H[i − 1] ← i
        pos[i] ← i − 1
    for i ← V − 1 down to 1
        DOWNHEAPIFY(i)
```

Code C:

+ expand source (#)

Code đầy đủ: Dijkstra-with-adjacency-matrix (http://www.giaithuatlaptrinh.com/wp-content/uploads/2016/02/dijkstra_matrix.c) , Dijkstra-with-adjacency-list (http://www.giaithuatlaptrinh.com/wp-content/uploads/2016/02/dijkstra_adj_list.c)

Tham khảo

[1] Dijkstra, E. W. (1959). *A note on two problems in connexion with graphs*. Numerische Mathematik 1: 269–271
[2] Jeff Erickson. Lecture Notes on Single Source Shortest Paths (http://jeffe.cs.illinois.edu/teaching/algorithms/notes/21-sssp.pdf) . UIUC, 2014.
[3] Fredman, Michael L., and Robert Endre Tarjan. *Fibonacci heaps and their uses in improved network optimization algorithms.* Journal of the ACM (JACM) 34.3 (1987): 596-615.

RELATED

Đường đi ngắn nhất giữa mọi cặp đỉnh trong đồ thị thưa -- Johnson's Algorithm (http://www.giaithua...p=874)
February 29, 2016
In "all-pairs-shortest-paths"

Thực thi thuật toán Dijkstra bằng các cấu trúc dữ liệu (http://www.giaithua...p=838)
February 19, 2016
In "binary heap"

Thuật toán Bellman-Ford tìm đường đi ngắn nhất từ một đỉnh trong đồ thị có trọng số âm -- Bellman Ford Algorithm (http://www.giaithua...p=789)
January 20, 2016
In "Bellman-Ford"

Tags: binary heap, Dijkstra, Fibonacci, Fibonacci Heap, graph algorithm, heap, shortest-paths

No comments

Comments feed for this article

Trackback link: http://www.giaithuatlaptrinh.com/wp-trackback.php?p=764

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.