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.

Floyd-Warshall

You are currently browsing articles tagged **Floyd-Warshall**.

Quy hoạch động tìm đường đi ngắn nhất giữa mọi cặp đỉnh---Floyd-Warshall Algorithm

January 30, 2016 in Uncategorized | No comments

Trong bài này, chúng ta sẽ thảo luận thuật toán tìm đường đi ngắn nhất giữa mọi cặp đỉnh của đồ thị $G(V, \vec{E})$ (có hướng, có trọng số và không có chu trình âm). Chúng ta sẽ áp dụng quy hoạch động để giải bài toán này trong thời gian $O(n^3)$ và bộ nhớ $O(n^2)$. Mình sẽ giới thiệu một vài cách phát triển thuật toán quy hoạch động khác nhau, mỗi cách hi vọng sẽ mang đến cho bạn đọc một cách nhìn khác về bài toán. Cách phát biểu cuối cùng sẽ đưa chúng ta đến thuật toán nổi tiếng Floyd-Warshall. Tài liệu tham khảo chính là notes của Jeff Erickson [1]. Trước khi đọc tiếp, mình khuyến khích bạn đọc xem lại cách phát triển thuật toán quy hoạch động cho bài toán đường đi ngắn nhất từ một đỉnh tới mọi đỉnh khác trong bài này (http://www.giaithuatlaptrinh.com/?p=789) .

Remark: Đồ thị G trong bài này sẽ được biểu diễn bằng ma trận kề W, i.e, $W[u, v] = w(u \to v)$ nếu tồn tại một cung từ u tới v với trọng số $w(u \to v)$.

Problem 1: Cho đồ thị có hướng, có trọng số và không có chu trình âm $G(V, \vec{E})$, tìm khoảng cách ngắn nhất giữa mọi cặp đỉnh của đồ thị.

Nếu trọng số của đồ thị G là không âm, ta có thể áp dụng thuật toán Dijkstra (http://www.giaithuatlaptrinh.com/?p=764) V lần, mỗi lần áp dụng cho một đỉnh của đồ thị. Do đó, ta có:

Theorem 1: Ta có thể tìm được khoảng cách ngắn nhất giữa mọi cặp đỉnh trong đồ thị không có trọng số âm trong thời gian $O(V^2 \log V)$ sử dụng $O(V^2)$ bộ nhớ.

Tuy nhiên trong đồ thị có trọng số âm, bài toán trở nên phức tạp hơn nhiều. Như ta đã biết, thuật toán Bellman-Ford (http://www.giaithuatlaptrinh.com/?p=789) có thể tìm đường đi ngắn nhất từ

một đỉnh tới mọi đỉnh khác trong đồ thị trong thời gian $O(VE)$ và bộ nhớ $O(V)$. Do đó, nếu áp dụng thuật toán Bellman-Ford cho mọi đỉnh của đồ thị, ta sẽ thu được thuật toán với thời gian $O(V^2E)$ và bộ nhớ $O(V^2)$. Với đồ thị dầy $E = O(V^2)$, thời gian của thuật toán này là $O(V^4)$.

Cũng trong bài trước (http://www.giaithuatlaptrinh.com/?p=789) , chúng ta cũng đã tìm hiểu phương pháp quy hoạch động áp dụng cho bài toán đường đi ngắn nhất từ một đỉnh tới mọi đỉnh khác và cuối cùng ta lại quy thuật toán này về thuật toán Bellman-Ford. Trong bài này, chúng ta cũng áp dụng tư tưởng tương tự để giải quyết bài toán đường đi ngắn nhất giữa mọi cặp đỉnh.

Trong các giả mã dưới đây, ta quy ước $w(u \to u) = 0$ và $w(u \to v) = +\infty$ nếu như cung $u \to v$ không tồn tại trong đồ thị G.

Thuật toán quy hoạch động trâu bò

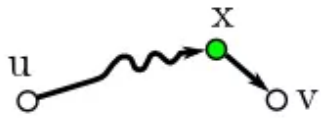
Ta sẽ xây dựng bảng bảng quay hoạc động tương tự như bảng quy hoạch động trong bài thuật toán Bellman-Ford.

> Gọi $D[1, \ldots, V][1, \ldots, V][0, \ldots, V-1]$ là mảng 3 chiều trong đó mỗi ô $D[u, v, i]$ tương ứng với khoách cách ngắn nhất từ u tới v đi qua tối đa i đỉnh

Từ định nghĩa trên, ta có công thức truy hồi:

$$D[u,v,i] = \begin{cases} w(u \to v) & \text{if } i = 0 \\ \min_{(x \to v) \in \vec{E}}(D[u,x,i-1] + w(x \to v)), & \text{otherwise} \end{cases} \quad (1)$$

Ý nghĩa của công thức truy hồi trên như sau: đường đi ngắn nhất từ u tới v sẽ phải đi qua một hàng xóm x nào đó của v mà $(x \to v) \in \vec{E}$ (xem hình dưới đây). Nếu đường đi ngắn nhất từ u tới v đi qua tối đa i đỉnh thì đường đi con từ u tới x sẽ đi qua không quá $i-1$ đỉnh. Do đó $D[u,v,i] = D[u,x,i-1] + w(x \to v)$. Do ta không biết x, ta sẽ duyệt qua tất cả các hàng xóm có thể của v và lấy giá trị nhỏ nhất để tìm $D[u,v,i]$.



Khoảng cách ngắn nhất từ u tới v chính là $D[u,v,V-2]$ vì đường đi ngắn nhất từ u tới v sẽ đi qua tối đa $V-2$ đỉnh.

Giả mã:

```
DummyAllPairsSPs(A[1,2,...,n]):
    initialize all entries of D[1,...,V][1,...,V][0,...,V-1] to +∞
    for each u ∈ V
        for each v ∈ V
            D[u,v,0] ← w(u → v)
    for i ← 1 to V − 2
        for each u ∈ V
            for each v ∈ V
                for each in-neghbor x of v        ≪ x → v ∈ E ≫
```

$D[u,v,i] \leftarrow \min(D[u,v,i], D[u,x,i] + w(x \rightarrow v))$
output $D[u,v,V-2]$ as the shortest distance between u,v

Tính đúng đắn của thuật toán trên có thể chứng minh bằng phương pháp quy nạp. Bạn đọc xem thêm tại các các bài viết trước về quy hoạch động (http://www.giaithuatlaptrinh.com/?page_id=4) .

Phân tích thời gian Bước khởi tạo đầu tiên mất thời gian $O(V^3)$ để khởi tạo các phần tử của bảng D thành $+\infty$. Bước khởi tạo thứ hai (2 vòng lặp for) mất thời gian $O(V^2)$. Do đó ta mất tổng thời gian $O(V^3)$ để khởi tạo. Tiếp theo ta sẽ phân tích 4 vòng for cuối cùng. Vòng for trong cùng duyệt qua tất các cảc đỉnh x mà có cung $x \rightarrow v \in \vec{E}$; do đó, vòng for này có $d^+(v)$ vòng lặp trong đó $d^+(v)$ là bậc tới (http://www.giaithuatlaptrinh.com/?p=553) của v. Tổng số bước lặp của hai vòng for trong cùng là:

$$\sum_{v \in V} d^+(v) = O(E) \qquad (2)$$

Do hai vòng lặp ngoài cùng có $(V-2)V$ bước lặp, thời gian của thuật toán là $O(V^2E)$. Như vậy, thuật toán DummyAllPairsSPs có thời gian chạy bằng thời gian thực hiện V lần thuật toán Bellman-Ford.

Nhanh hơn trâu bò một chút

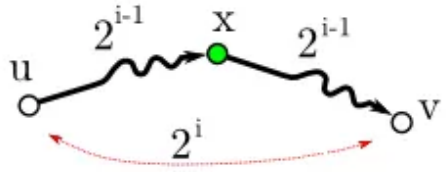
Ta thay đổi bảng quy hoạch động trong thuật toán toán trâu bò một chút:

> Gọi $D[1,\ldots,V][1,\ldots,V][0,\ldots,\lceil \log_2(V-2) \rceil]$ là mảng 3 chiều trong đó mỗi ô $D[u,v,i]$ tương ứng với khoách cách ngắn nhất từ u tới v đi qua tối đa 2^i đỉnh.

Chú ý tới số 2^i trong định nghĩa trên. Với định nghĩa trên, ta có công thức truy hồi:

$$D[u,v,i] = \min_{x \in V}(D[u,x,i-1] + D[x,v,i-1]) \qquad (3)$$

Ý nghĩa của công thức truy hồi trên là nếu có một đường đi từ u tới v qua tối đa 2^i đỉnh thì đường đi đó sẽ đi qua một đỉnh x sao cho đường đi từ u tới x sẽ qua tối đa 2^{i-1} đỉnh và đường đi từ x tới v đi qua tối đa 2^{i-1} đỉnh (xem hình dưới đây).



Từ định nghĩa ta suy ra, $D[u,v,\lceil \log_2(V-2) \rceil]$ sẽ là đường đi ngắn nhất từ u tới v đi qua tối đa $2^{\lceil \log_2(V-2) \rceil} \geq 2^{\log_2(V-2)} = V-2$ đỉnh. Do đó, $D[u,v,\lceil \log_2(V-2) \rceil]$ sẽ là đường đi ngắn nhất từ u tới v.

Chú ý với định nghĩa bảng D như trên, trường hợp cơ bản sẽ là $D[u,v,1]$. Do đó, ta sẽ áp dụng một vòng lặp trong thuật toán DummyAllPairsSPs để tính các giá trị đó.

Giả mã:

```
NotSoDumbAllPairsSPs(A[1,2,...,n]):
    initialize entries of D[1,...,V][1,...,V][0,...,⌈log2(V-2)⌉] to +∞
    for each u ∈ V
        for each v ∈ V
            D[u,v,0] ← w(u → v)
    for each u ∈ V
        for each v ∈ V
            for each in-neghbor x of v        ≪ x → v ∈ E⃗ ≫
                D[u,v,1] ← min(D[u,v,1], D[u,x,0] + w(x → v))
    for i ← 2 to ⌈log2(V-2)⌉
        for each u ∈ V
            for each v ∈ V
                for each x ∈ V
                    D[u,v,i] ← min(D[u,v,i], D[u,x,i-1] + D[x,v,i-1])
    output D[u,v,⌈log2(V-2)⌉] as the shortest distance between u,v
```

Phân tích thuật toán: Bước khởi tạo đầu tiên mất thời gian $O(V^2 \log_2(V))$ để khởi tạo bảng D và bước thứ hai (2 vòng lặp for đầu tiên) mất thời gian $O(V^2)$. Phân tích bước thứ ba (2 vòng lặp for tiếp theo), tương tự như phân tích ở trên, mất thời gian $O(VE)$. Bước cuối cùng thực hiện 4 vòng lặp lồng nhau, 3 vòng lặp trong cùng có V^3 bước lặp và vòng lặp ngoài cùng có $\log_2(V-2)$ bước lặp. Như vậy, tổng thời gian của thuật toán NotSoDumbAllPairsSPs là $O(VE + V^3 \log_2(V)) = O(V^3 \log_2(V))$.

Thuật toán Floyd-Warshall

Floy[2] và Warshall[3] thay đổi cách xây dựng bảng quy hoạch động theo một hướng khác để giảm thời gian tính toán của thuật toán. Giả sử ta đánh số các đỉnh từ 1 đến V.

> Gọi $D[1,\ldots,V][1,\ldots,V][0,\ldots,V]$ là mảng 3 chiều trong đó mỗi ô $D[u,v,i]$ tương ứng với khoách cách ngắn nhất từ u tới v đi qua các đỉnh trung gian nằm trong tập hợp $\{1,2,\ldots,i\}$

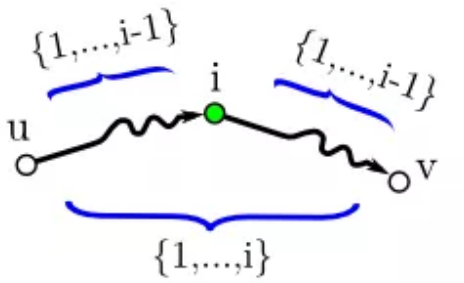
Như vậy, $D[u,v,0]$ sẽ có ý nghĩa là đường đi ngắn nhất từ u tới v mà không đi qua đỉnh trung gian nào cả và ta sẽ khởi tạo $D[u,v,0] = w(u \to v)$. Với định nghĩa như trên, ta có công thức truy hồi:

$$D[u,v,i] = \min(D[u,v,i-1], D[u,i,i-1] + D[i,v,i-1]) \qquad (4)$$

Ý nghĩa của công thức truy hồi trên là nếu có một đường đi từ u tới v qua các đỉnh trong tập hợp $\{1,2,\ldots,i\}$ thì hoặc đường đi đó đi qua đỉnh i hoặc nó không đi qua đỉnh i (xem hình dưới đây):

1. Nếu đường đi đó **không** đi qua đỉnh i thì đường đi ngắn nhất đó cũng chính là đường đi ngắn nhất từ u tới v đi qua các đỉnh trung gian trong $\{1,2,\ldots,i-1\}$. Do đó $D[u,v,i] = D[u,v,i-1]$
2. Nếu đường đi đó đi qua đỉnh i thì đường đi con từ u tới i và đường đi con từ i tới v sẽ chỉ đi qua các đỉnh thuộc tập $\{1,\ldots,i-1\}$. Do đó $D[u,v,i] = D[u,v,i-1] + D[i,v,i-1]$

Ta sẽ lấy min trong hai trường hợp để xác định giá trị $D[u,v,i]$.



Từ định nghĩa ta suy ra, $D[u,v,V]$ sẽ là đường đi ngắn nhất từ u tới v.

Giả mã:

```
FloydWarshall(A[1,2,...,n]):
    initialize entries of D[1,...,V][1,...,V][0,...,V] to +∞
    for each u ∈ V
        for each v ∈ V
            D[u,v,0] ← w(u → v)
    for i ← 1 to V
        for each u ∈ V
            for each v ∈ V
                D[u,v,i] ← min(D[u,v,i−1], D[u,i,i−1] + D[i,v,i−1])
    output D[u,v,V] as the shortest distance between u,v
```

Do 3 vòng for lồng nhau cuối cùng có $O(V^3)$ bước lặp, ta có:

Theorem 2: Thuật toán Floyd-Warshall tìm khoảng cách ngắn nhất giữa mọi cặp đỉnh trong đồ thị không có chu trình âm trong thời gian $O(V^3)$.

Giảm bộ nhớ

Thuật toán Floyd-Warshall trình bày ở trên có bộ nhớ $O(V^3)$ vì đầu ra của chúng ta là mảng 3 chiều D. Để giảm bộ nhớ chúng ta có nhận xét: Các giá trị ở bảng con 2 chiều thứ i (là các gía trị $D[u,v,i]$ với **mọi** cặp đỉnh u,v) chỉ phụ thuộc vào giá trị của bảng con hai chiều thứ $i-1$ (là các giá trị $D[u,v,i-1]$ với **mọi** cặp đỉnh u,v). Do đó, thay vì lưu cả bảng 3 chiều, ta có thể lưu hai bảng hai chiều, một bảng tương ứng với bảng hai chiều thứ i với i chẵn, và một với bảng hai chiều thứ i với i lẻ và tái sử dụng các bảng hai chiều đó qua các vòng lặp. Ý tưởng này giống như cách giảm bộ nhớ của thuật toán tính số Fibonacci (http://www.giaithuatlaptrinh.com/?p=66).

Tuy nhiên, ta có thể đơn giản hóa bằng cách bỏ hẳn chiều thứ 3 của bảng 3 chiều D. Tính chất nhỏ nhất của khoảng cách ngắn nhất sẽ đảm bảo thuật toán luôn đúng mặc dù ta bỏ hẳn một chiều của bảng quy hoạch động. Giả mã:

```
MemFloydWarshall(A[1,2,...,n]):
    initialize entries of D[1,...,V][1,...,V] to +∞
    for each u ∈ V
```

```
    for each v ∈ V
        D[u,v] ← w(u → v)
for i ← 1 to V
    for each u ∈ V
        for each v ∈ V
            D[u,v] ← min(D[u,v], D[u,i] + D[i,v])
output D[u,v] as the shortest distance between u,v
```

Truy vết đường đi

Để in ra đường đi ngắn nhất giữa hai đỉnh (u, v) bất kì, chúng ta cần phải lưu vết trong quá trình quy hoạch động. Mỗi khi có thay đổi trong bảng quy hoạch động ($D[u,v] > D[u,i] + D[i][v]$), ta phải ghi lại sự thay đổi đó. Ta sẽ dùng một mảng 2 chiều $T[1,\ldots,V][1,\ldots,V]$ trong đó:

$$T[u,v] = i \text{ nếu đường đi ngắn nhất từ } u \text{ tới } v \text{ đi qua đỉnh } i$$

Có hai giá trị đặc biệt của $T[u,v]$ mà ta cần phải quan tâm: nếu không tồn tại đường đi nào từ u tới v, ta sẽ gán $T[u,v] = \text{Null}$. Nếu đường đi ngắn nhất từ u đến v không đi qua đỉnh trung gian nào, ta sẽ gán $T[u,v] = 0$.

Giả mã cập nhật bảng T:

```
MemAndTraceFloydWarshall(A[1,2,...,n]):
    initialize entries of D[1,...,V][1,...,V] to +∞
    initialize entries of T[1,...,V][1,...,V] to Null
    for each u ∈ V
        for each v ∈ V
            if w(u → v) < +∞
                D[u,v] ← w(u → v)
                T[u,v] ← 0
    for i ← 1 to V
        for each u ∈ V
            for each v ∈ V
                if D[u,v] > D[u,i] + D[i,v]
                    D[u,v] ← D[u,i] + D[i,v]
                    T[u,v] ← i
    output D[u,v] as the shortest distance between u,v
```

Code C:

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

Khi đã có bảng $T[1,\ldots,V][1,\ldots,V]$, ta có thể in ra đường đi ngắn nhất giữa hai đỉnh u, v như sau:

```
PrintReversePath(T,u,v):
    if T[u,v] = Null
        print the path between u,v does not exists
    else
        print v
        FindReversePath(T,u,v)
```

```
FindReversePath(T, u, v):
    i ← T[u, v]
    if i = 0
        print u
    else
        FindReversePath(T, i, v)
        FindReversePath(T, u, i)
```

Code C:

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

Remark Mặc dù sau rất nhiều năm nghiên cứu, cho đến nay vẫn chưa có thuật toán nào tốt hơn thời gian $O(V^3)$ một cách **đáng kể** và cho **mọi trường hợp**. Do đó, người ta giả thuyết rằng không tồn tại thuật toán như vậy (gọi là giả thuyết đường đi ngắn nhất giữa mọi cặp đỉnh). Thuật toán (ngẫu nhiên) tốt nhất cho đến nay được phát triển bởi Ryan Williams (http://web.stanford.edu/~rrwill/) [5] có thời gian $O(V^3/2^{\Omega(\sqrt{\log V})})$.

Code đầy đủ: Floyd-Warshall (http://www.giaithuatlaptrinh.com/wp-content/uploads/2016/02/FW.c)

Tham khảo

[1] Jeff Erickson, Lecture Notes on All Pairs Shortest Paths (http://jeffe.cs.illinois.edu/teaching/algorithms/notes/22-apsp.pdf) , UIUC, 2014.
[2] Floyd, Robert W. *Algorithm 97: shortest path.* Communications of the ACM 5.6 (1962): 345.
[3] Warshall, Stephen. *A theorem on boolean matrices.* Journal of the ACM (JACM) 9.1 (1962): 11-12.
[4] Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L. *Introduction to Algorithms (1st ed.)*. Chapter 26.2, MIT Press and McGraw-Hill, 1990.
[5] Williams, Ryan. *Faster all-pairs shortest paths via circuit complexity.* Proceedings of the 46th Annual ACM Symposium on Theory of Computing. ACM, 2014.

Tags: all-pairs-shortest-paths, dynamic programming, Floyd-Warshall, Ryan Williams, shortest-paths