|  |
| --- |
| <https://www.geeksforgeeks.org/bellman-ford-algorithm-dp-23/>  <http://www.giaithuatlaptrinh.com/?p=764>  <https://www.giaithuatlaptrinh.com/?p=789> |

**Sơ lược về giải thuật Bellman-Ford:**

Ta sẽ quy ước nếu trong đồ thị mà cung u→v không tồn tại, ta sẽ coi như nó tồn tại với trọng số +∞.

Gọi δ(s,v) là khoảng cách ngắn nhất từ s (source) đến v (vertices). với mỗi đỉnh v, ta sẽ lưu một nhãn tạm thời d[v] và sau đó cập nhật d[v] trong các vòng lặp. Ta có thể coi nhãn tạm thời này là một ước lượng tạm thời của khoảng cách ngắn nhất từ s tới v. Mỗi giá trị của d[v] sẽ có ý nghĩa như sau:

**Ý nghĩa nhãn:** Giá trị d[v] sẽ tương ứng với độ dài của một đường đi nào đó (có thể chưa phải là ngắn nhất) từ s tới v.

Ta khởi tạo d[v]=w(s→v). Ý nghĩa của việc khởi tạo này là tạm thời coi cạnh s→v là ước lượng của đường đi ngắn nhất từ s tới v.

Ta sẽ sử dụng ý nghĩa của nhãn trên để tìm đường đi ngắn nhất. Dễ thấy, nếu tồn tại cung u→v mà d[u]+w(u→v)<d[v] thì đường đi từ v qua u và tới s sẽ ngắn hơn đường đi hiện tại tương ứng với ước lượng d[v]. Do đó, ta sẽ cập nhật lại:

d[v]=d[u]+w(u→v)(1)

Ý nghĩa của việc cập nhật lại là đường đi từ v qua u tới s là một ước lượng tốt hơn với ước lượng hiện tại của v. Ta gọi cung (u→v) thỏa mãn d[u]+w(u→v)<d[v] là cung bị căng (tense), và thao tác gán lại nhãn của v theo phương trình (1) là thao tác nới lỏng (relax) cung (u→v).

Như vậy, thuật toán tìm đường đi ngắn nhất có thể được phát biểu đơn giản trong một câu như sau:

**Luật nới lỏng:** Tìm các cung căng và nới lỏng chúng.

Giải mã:

|  |
| --- |
| GenericBellmanFord(G(V,E→),w,s):     d[s]←0     **for** each v∈V         d[v]←w(s→v)     **repeat**         for every tense arc (u→v)             Relax(u→v)     **until** there is no tense edge |

|  |
| --- |
| Relax(u→v):     d[v]←d[u]+w(u→v) |

Hiển nhiên, ta nới lỏng đến khi nào đồ thị không còn cung nào căng thì lúc đó ước lượng d[v] cũng chính là khoảng cách ngắn nhất, i.e, d[v]=δ(s,v). Nếu G có chu trình âm (như trong ví dụ trên) thì luôn tồn tại một cung căng trong mỗi vòng lặp của thuật toán trên (tại sao?). Do đó, thuật toán sẽ không bao giờ kết thúc.

Bây giờ ta sẽ giả sử G không có chu trình âm, thuật toán GenericBellmanFord sẽ luôn kết thúc do mỗi bước, ít nhất một đỉnh v sẽ có nhãn d[v] giảm xuống ít nhất là một đơn vị (giả sử trọng số các cạnh là các số nguyên) và nhãn đó không thể giảm tới âm vô hạn được. Vấn đề còn lại là thuật toán trên sẽ kết thúc sao bao nhiêu vòng lặp?

Để trả lởi câu hỏi trên thì ta hãy xét một trường hợp đơn giản sau: Giả sử u là đỉnh mà cung s→u cũng chính là đường đi ngắn nhất từ s tới u (luôn tồn tại đỉnh như vậy trong đồ thị, nhưng không nhất thiết là đỉnh có d[u] nhỏ nhất;

Do đó d[u]=w(s→u)=δ(s,u) và từ đó suy ra ta sẽ không bao giờ phải cập nhật lại nhãn d[u] sau khi nó được khởi tạo. Như vậy, các đỉnh có đường đi ngắn nhất từ s mà không đi qua đỉnh nào khác sẽ không cần phải cập nhật lại nhãn sau khi khởi tạo.

Mở rộng ví dụ trên ra một chút, xét đỉnh v mà đường đi ngắn nhất từ s tới v chỉ đi qua một đỉnh u nào đó. Hay nói cách khác s→u→v là đường đi ngắn nhất. Do đó, s→u là đường đi ngắn nhất từ s tới u, và theo như trên d[u] sẽ không cần phải cập nhật lại sau khi nó được khởi tạo. Trong vòng lặp đầu tiên, cung u→v sẽ căng (tại sao?) và ta sẽ cập nhật lại d[v]=d[u]+w(u→v). Do đó, d[v] chính là độ dài đường đi s→u→v và từ đó suy ra d[v]=δ(s,v). Như vậy, các đỉnh v có đường đi ngắn nhất từ s đi qua một đỉnh khác sẽ có nhãn được cập nhật trong vòng lặp đầu tiên và nhãn này sẽ không bao giờ được cập nhật ở các vòng lặp sau đó nữa.

Tổng quát hóa lên ta sẽ thấy, nếu một đỉnh v có đường đi ngắn nhất từ s đi qua k đỉnh khác thì nhãn của v sẽ không bao giờ được cập nhật sau k vòng lặp của thuật toán. Do đường đi ngắn nhất từ s tới một đỉnh bất kì không đi qua nhiều hơn |V|−2 đỉnh khác, thuật toán trên sẽ lặp sau |V|−2 bước.

Điều gì xảy ra nếu sau |V|−2 bước mà vẫn có một cung bị căng? Dễ thấy trường hợp này chỉ xảy ra khi đồ thị có chu trình âm. Qua đó, ta có thể phát hiện được đồ thị có chu trình âm hay không. Giả mã của thuật toán như sau:

|  |
| --- |
| BellmanFord(G(V,E→),w,s):     d[s]←0     P[s]←s     **for** each v∈V         d[v]←w(s→v)         P[v]←s     **for** i←1 to V−2         for every tense arc (u→v)             Relax(u→v)     **if** there is a tense arc         print G has a negative cycle |

|  |
| --- |
| Relax((u→v)):     d[v]←d[u]+w(u→v)     P[v]←u |

|  |
| --- |
| FindReverseShortestPath(t,P[1,2,…,V]):     print t     **while** P[t]≠t         t←P[t]         print t |

Mảng P[1,2,…,V] ở trên được dùng để lưu vết đường đi. Ta sử dụng mảng này để in ra đường đi ngắn nhất từ s tới t.

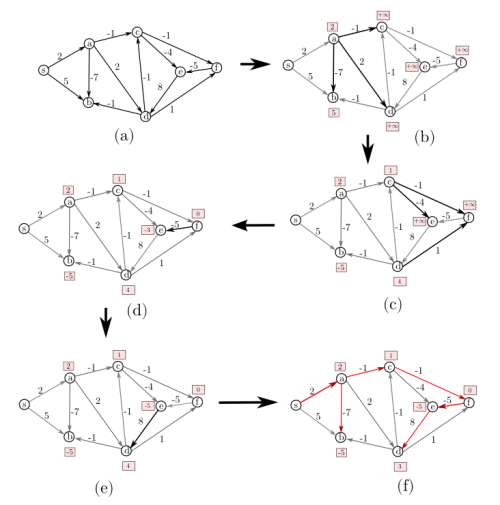
**Phân tích thuật toán:**Để phát hiện ra các cung căng, ta chỉ cần duyệt qua tất cả các cung. Sử dụng danh sách kề, các cung có thể được duyệt qua trong thời gian O(E), do đó, tổng thời gian của thuật toán là O(VE). So với Dijkstra (O(Vlog⁡V+E)) thì thuật toán Bellman Ford chậm hơn rất nhiều.

Một ví dụ thực thi thuật toán Bellman-Ford được minh họa trong hình dưới đây.

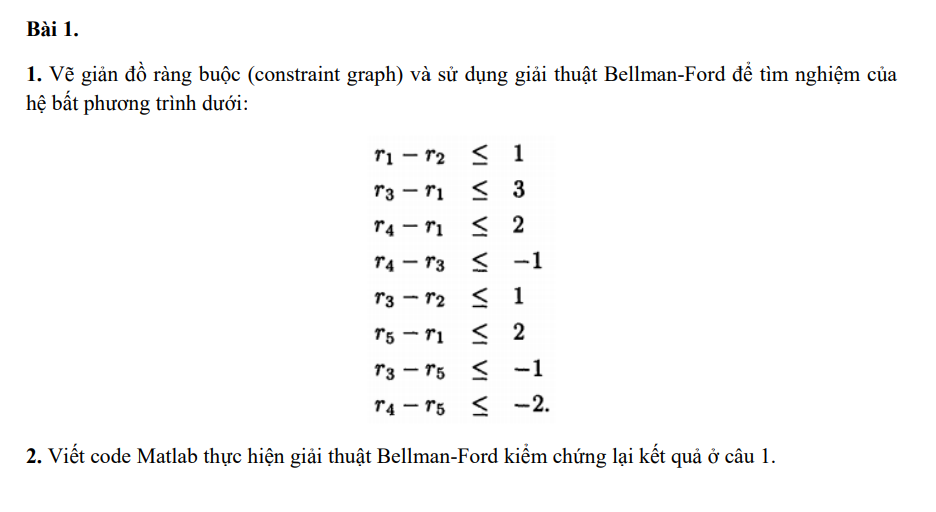
Các cung tô đậm là các cung bị căng.

Trong hình (f), các cung màu đỏ là các cung trong cây đường đi ngắn nhất xuất phát tại s.

Các số trong hộp được tô màu hồng là nhãn d[v] của đỉnh.



**Remark** Nếu chúng ta biết trước được G không có chu trình âm thì chúng ta không nhất thiết phải chạy hết |V|−2 vòng lặp mà ta có thể dừng ngay khi đồ thị G không còn cung căng nữa.

**Giản đồ ràng buộc:**

|  |
| --- |
| **-1**  **2**  **1**  **2**  **1**  **-1**  **2**  **-2** |