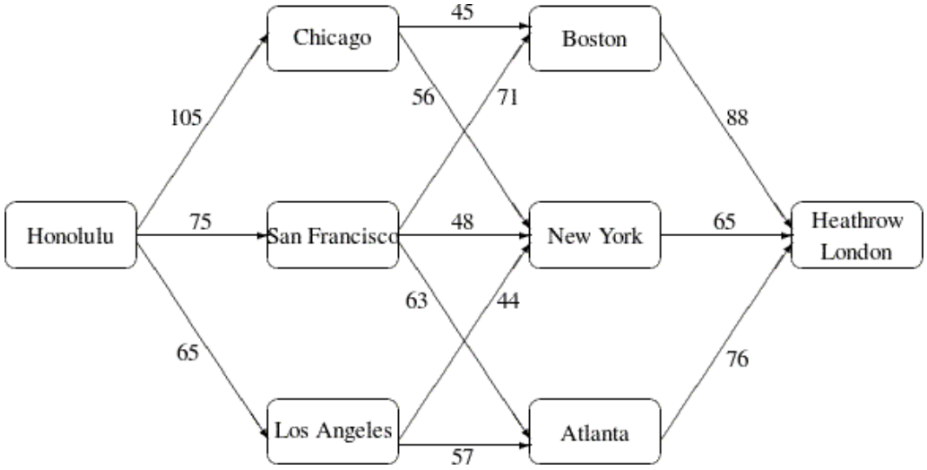
CHƯƠNG 6. BÀI TOÁN TÌM ĐƯỜNG ĐI NGẮN NHẤT TRÊN ĐỒ THỊ (SHORTEST-PATH PROBLEMS) (SPP)

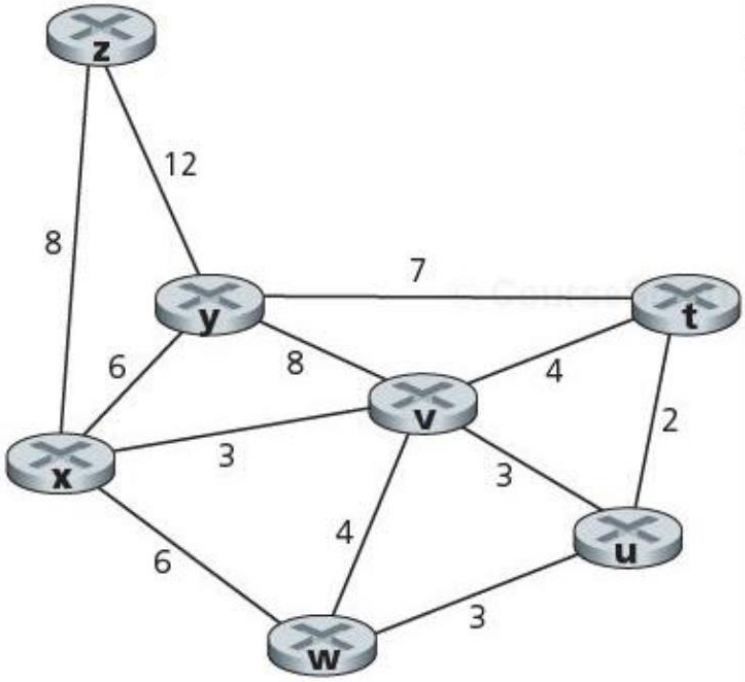
Contents

6.1 Bài toán đường đi ngắn nhất

6.1.1 Mở đầu



Yêu cầu: tìm đường đi ngắn nhất từ Chicago tới Atlanta?



Yêu cầu: tìm đường đi ngắn nhất để truyền gói tin từ router z đến router u?

Trong các ứng dụng thực tế, Bài toán tìm đường đi ngắn nhất giữa hai đỉnh của một đồ thị liên thông có một ý nghĩa to lớn. Có thể dẫn về bài toán như vậy nhiều bài toán thực tế quan trọng. Ví dụ:

- Bài toán chọn một hành trình tiết kiệm nhất (theo tiêu chuẩn khoảng cách hoặc thời gian hoặc chi phí) trên một mạng giao thông đường bộ, đường thuỷ hoặc đường không.
- Bài toán chọn một phương pháp tiết kiệm nhất để đưa một hệ động lực lực từ trạng thái xuất phát đến một trạng thái đích.
- Bài toán lập lịch thi công các công đoạn trong một công trình thi công lớn.
- Bài toán lựa chọn đường truyền tin với chi phí nhỏ nhất trong mạng thông tin.
- Bài toán vẽ đường di chuyển cho robot v ... v ...

6.1.2 Các khái niệm

- G = (V, E, W) with weight function W: E $\rightarrow \mathbb{R}$ (assigning real values to edges) *(Ánh xạ từ tập E vào tập số thực $\mathbb{R}$, mỗi cạnh tương ứng một trọng số).*
- Weight of path p = $v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_k$ is

$$w(p)=\sum_{i=1}^{k-1} w(v_i + v_{i+1})$$

- Shortest path = a path of the *minimum weight*
- Bài toán tìm đường đi ngắn nhất trên đồ thị dưới dạng tổng quát có thể phát biểu như sau: Tìm đường đi có độ dài nhỏ nhất từ một đỉnh xuất phát s ∈ V đến đỉnh cuối (đích) t ∈ V.

Đường đi như vậy ta sẽ gọi là đường đi ngắn nhất từ s đến t còn độ dài của nó ta sẽ ký hiệu là **d(s,t)** và còn gọi là khoảng cách từ s đến t (khoảng cách định nghĩa như vậy ***có thể là số âm***).

Nếu như *không tồn tại đường đi* từ s đến t thì ta sẽ đặt **d(s,t) = ∞.**

Rõ ràng, nếu như mỗi chu trình trong đồ thị đều có *độ dài dương*, *thì trong đường đi ngắn nhất không có đỉnh nào bị lặp lại* (đường đi không có đỉnh lặp lại sẽ được gọi là *đường đi cơ bản/ đường đi đơn*).

Mặt khác, nếu trong đồ thị có chu trình với *độ dài âm* (chu trình như vậy, để ngắn gọn, ta sẽ gọi là chu trình âm) thì *khoảng cách giữa một số cặp đỉnh nào đó của đồ thị có thể là không xác định*, bởi vì, bằng cách đi vòng theo chu trình này một số đủ lớn lần, ta có thể chỉ ra *đường đi giữa các đỉnh này có độ dài nhỏ hơn bất cứ số thực cho trước nào.*

Trong những trường hợp như vậy, có thể đặt vấn đề tìm đường đi cơ bản ngắn nhất, tuy nhiên bài toán đặt ra sẽ trở nên phức tạp hơn rất nhiều, bởi vì nó chứa bài toán xét sự tồn tại đường đi Hamilton trong đồ thị như là một trường hợp riêng. Hiện chưa có ai chứng minh được sự tồn tại hay không một thuật toán đa thức tìm đường đi đơn ngắn nhất trên đồ thị có chu trình âm.

- ***Yêu cầu:***

■ **1. Single-source (single-destination).** Find a shortest path from a *given source (vertex s) to each of the vertices. (Tìm đường đi ngắn nhất từ một đỉnh đến mọi đỉnh còn lại của đồ thị)*

■ **2. Single-pair.** Given two vertices, *find a shortest path between them*. Solution to single-source problem solves this problem efficiently, too. *(Tìm đường đi ngắn nhất giữa 2 đỉnh bất kỳ)*

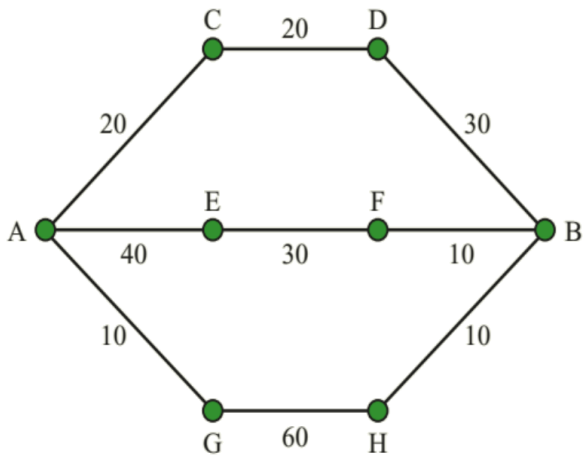
■ **3. All-pairs.** Find shortest-paths for every pair of vertices. Dynamic programming algorithm. *(Tìm đường đi ngắn nhất giữa mọi cặp đỉnh của đồ thị)*

** 2 bài toán 1, 3 đều quy về bài toán 2. (Bài toán 2 là bài toán tổng quát).*

■ Unweighted shortest-paths – BFS.

6.1.3 Giải bài toán tìm đường đi ngắn nhất bằng các cách khác nhau

- Dùng vét cạn (Phương pháp được sử dụng đầu tiên)

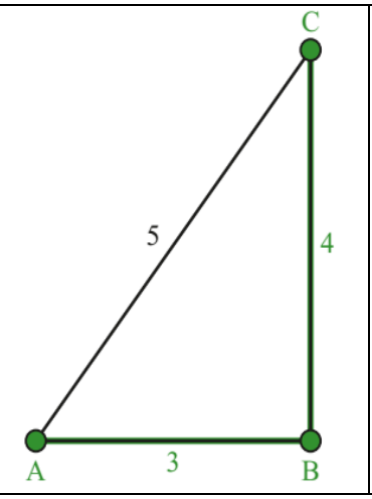


The network below shows cities connected by highways. The length of each highway is indicated by an edge weight. What is the shortest path between cities A and B?

There are three paths from A to B.
1/ Path ACDB has a length of 20 + 20 + 30 = 70.
2/ Path AEFB has a length of 40 + 30 + 10 = 80.
3/ Path AGHB has a length of 10 + 60 +10 = 80.
The path with the shortest length is ACDB.

One way to find the shortest path is to calculate the length of every path, and then select the path with the shortest length. This method only works in practice *when there is a small number of possible paths to consider.*

- Dùng cây khung cực tiểu



+ The minimum spanning tree *does not always provide the shortest path between two points.*
+ For example, in the network on the left, the minimum spanning tree contains the path ABC (highlighted in green), w/ total weight is 7, whereas the shortest path between vertices A and C is AC, w/ weight just only 5.

- Dùng tham lam: *Tối ưu cục bộ → Tối ưu toàn cục*

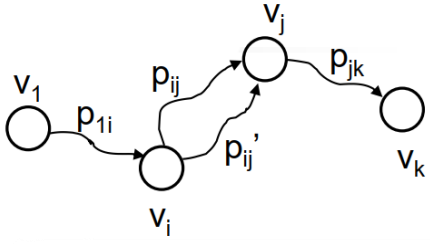
Một tính chất của đường đi ngắn nhất nữa có thể phát biểu một cách không hình thức như sau: *Mọi đoạn đường con của đường đi ngắn nhất cũng là đường đi ngắn nhất.* ***(subpaths of shortest paths are shortest paths).***

Tính chất tuy rất hiển nhiên này nhưng lại hàm chứa một nội dung rất sâu sắc, và người ta thường gọi nó là ***nguyên lý tối ưu.*** Việc chứng minh tính đúng đắn của hầu hết các thuật toán tìm đường đi ngắn nhất đều được xây dựng dựa vào nguyên lý này.

Chứng minh: Dùng phương pháp chứng minh phản chứng

Given:

- A weighted graph G = (V, E)
- A weight function w: E → $\mathbb{R}$
- A shortest path p = $\langle v_1, v_2, \dots, v_k \rangle$ from v_1 to v_k
- A subpath *(đường đi con)* of p: $p_{ij} = \langle v_i, v_{i+1}, \dots, v_j \rangle$, with $1 \le i \le j \le k$



Then: **p_{ij} is a shortest path from v_i to v_j**

Proof: $p = v_1 \overset{p_{1i}}{\rightsquigarrow} v_i \overset{p_{ij}}{\rightsquigarrow} v_j \overset{p_{jk}}{\rightsquigarrow} v_k$

$w(p) = w(p_{1i}) + w(p_{ij}) + w(p_{jk})$

Assume $\exists\ p_{ij}'$ from v_i to v_j with $w(p_{ij}') < w(p_{ij})$

$\Rightarrow w(p') = w(p_{1i}) + w(p_{ij}') + w(p_{jk}) <$ **w(p) contradiction!** *(do p đã là đường đi ngắn nhất nên p' ngắn hơn là điều không thể!).*

*** Điều kiện tồn tại đường đi ngắn nhất:** Đồ thị **không** chứa **chu trình âm.** Chu trình âm là chu trình có tổng trọng số âm.

6.2 Thuật toán Dijkstra

- (/ˈdaɪkstrə/ *DYKE-strə*) Thuật toán tìm đường đi ngắn nhất từ một đỉnh xuất phát đến tất cả các đỉnh còn lại trong đồ thị *vô hướng/ có hướng*, có **trọng số không âm (≥ 0).**

❖ Ý tưởng: sử dụng nguyên lý tham lam (greedy)

- Tại mỗi bước luôn chọn đường đi ngắn nhất có thể.
- Đường đi ngắn nhất tới đỉnh chưa xét được xây dựng từ đường đi ngắn nhất qua các đỉnh đã được xét.

- For larger networks it is difficult to consider all possible paths between two vertices. Fortunately, Dijkstra's algorithm gives us a method to construct *a special spanning tree that contains all the shortest paths from a particular start vertex to every other vertex.*

Starting with any vertex, let's call it A, Dijkstra's algorithm *grows a tree of shortest paths from A.* The steps are as follows:

1/ Make a tree that ***only contains the vertex A***.

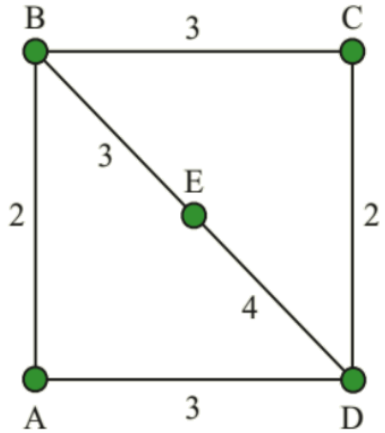
2/ Make a list of all paths ***starting from A*** that extend the tree by exactly ***one edge.*** Choose the path that has the *minimum* ***total weight*** and add the vertex and edge to the tree.

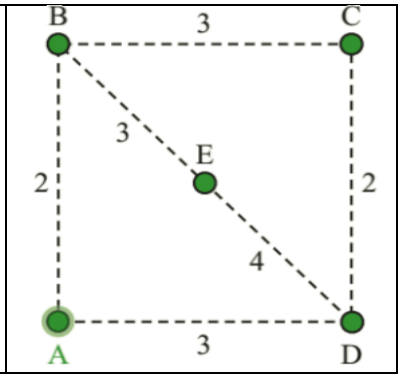
3/ Repeat step 2 until the tree includes ***all vertices.***

This is very similar to Prim's algorithm, *except in step 2 you select a new edge based on the* ***total distance from A*** *rather than the individual edge weight.*

Ví dụ:

a) Use Dijkstra's algorithm to find the shortest path from **A to E** in the following network.

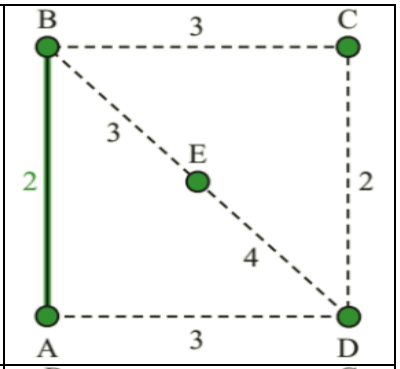


Make a tree with only vertex A.	

The tree contains only the vertex A.
The list of all paths ***from A*** with *exactly one vertex outside* this tree is:

Path	**AB**	AD
Total weight	**2**	3

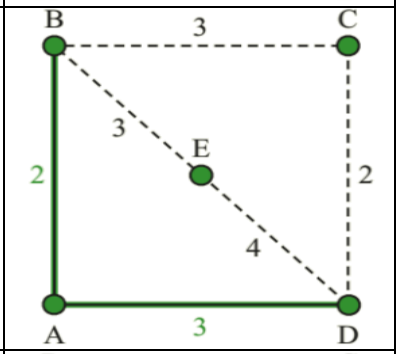
Path AB has minimum total weight, so add it to the tree.



The tree contains vertices A and B. The list of all paths starting at A with exactly one vertex outside this tree is:

Path	**AD**	ABC	ABE
Total weight	**2**	2 + 3 = 5	2 + 3 = 5

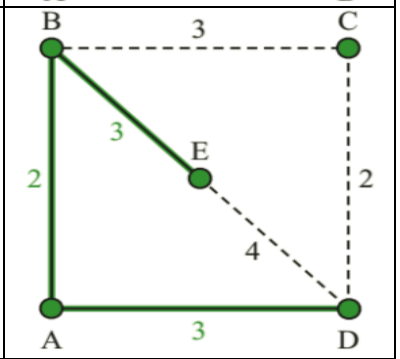
Path AD has minimum total weight, so add it to the tree.



The tree contains vertices A, B and D. The list of all paths from A with exactly one vertex outside this tree is:

Path	**ABE**	ABC	ADE	ADC
Total weight	**2 + 3 = 5**	2 + 3 = 5	3 + 4 = 7	3 + 2 = 5

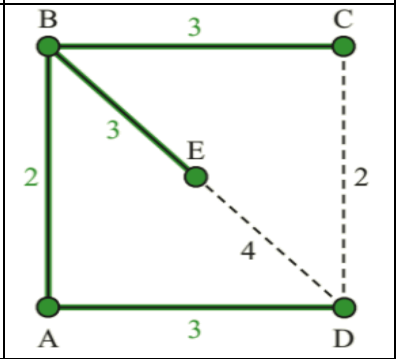
Paths ABE, ABC and ADC all have the same minimum total weight, so we are *free to choose any one* and add it to the tree. In this example *we have chosen to add ABE.*



The tree contains vertices A, B, D and E. ***Only vertex C remains outside the tree.*** The paths from A with exactly one vertex outside the tree are:

Path	**ABC**	ADC
Total weight	**2 + 3 = 5**	3 + 2 = 5

All paths have the same weight, so we are again free to choose any one and add it to the tree. In this example *we have chosen to add ABC.*



The resulting tree is a spanning tree that contains the shortest path from A to any other vertex.
In particular, ***the shortest path from A to E in this tree is ABE, which has total weight 2 + 3 = 5.***

- Thuật toán Dijkstra bình thường sẽ có độ phức tạp là $O(V^2 + E)$. Tuy nhiên ta có thể sử dụng kết hợp với cấu trúc heap, khi đó độ phức tạp sẽ là $O((E + V)\log(V))$, nếu dùng Fibonacci Heap thì độ phức tạp giảm xuống còn $O(E + V\log V)$.

**** Các thuật toán liên quan đến tìm đường đi ngắn nhất đều theo cơ chế Relaxation.***

■ For each vertex v in the graph, we maintain **d(v),** the estimate of the shortest path **from s**, *initialized to ∞ at the start.*

■ Relaxing an edge (u, v) means testing whether we can improve the shortest path to v found so far by going through u

```
if (d(u) + w (u, v) < d(v)) then
    d(v) = d(u) + w(u, v);
    previous(v) = v;
```

với d(v) là khoảng cách từ đỉnh v đến đỉnh xuất phát, tương tự với d(u).

**** Chạy thuật toán Dijkstra theo cách đầy đủ (cũng gần như là cách cài đặt thuật toán trên máy).***

- Đầu tiên ta gán cho mỗi đỉnh một nhãn gồm 2 thành phần: **(Previous(v), d(v)).**

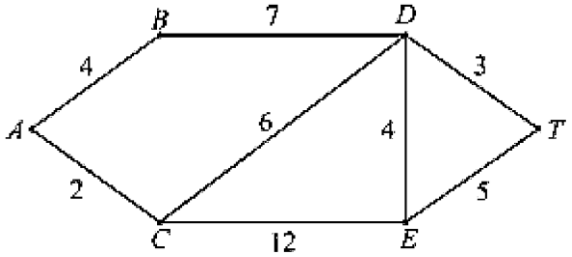
+ Thành phần thứ nhất là đỉnh nằm trước & kề với đỉnh v (đỉnh đang xét) (Parent hoặc Previous(v))

+ Thành phần thứ hai là khoảng cách từ đỉnh nguồn A đến các đỉnh v còn lại. Cụ thể ta có d(A) = 0, còn d(B), … = ∞ (ngầm hiểu là khoảng cách chưa xác định, khoảng cách ở xa vô cực).

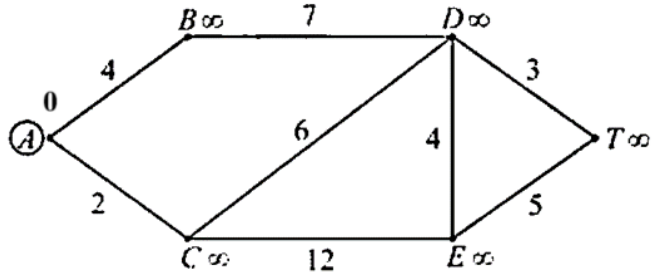
- Tiếp đó, ta thực hiện các bước lặp lần lượt điều chỉnh nhãn (còn gọi là *nhãn tạm thời*) của từng đỉnh về nhãn cố định đúng bằng độ dài của đường đi ngắn nhất từ đỉnh đó đến đỉnh A, khi đó các đỉnh được gắn *nhãn vĩnh viễn*, dựa theo cơ chế Relaxation.

Ví dụ: *(Sách Chuyên đề Toán 11 Chân trời sáng tạo)*

b) Tìm đường đi ngắn nhất từ đỉnh A đến các đỉnh còn lại.



- Đầu tiên, gán nhãn cho đỉnh A bằng 0 (d(A) = 0), cho các đỉnh khác đều bằng ∞. Previous của các đỉnh ta gán là NULL. *(Tạm thời chưa thể hiện trên đồ thị để đỡ rối).*
- Khoanh tròn đỉnh A và viết nhãn của mỗi đỉnh khác bên cạnh đỉnh đó.
- Mỗi lần thay đổi nhãn của đỉnh thì ta *gạch nhãn cũ và viết thêm nhãn mới.*



Bước 1. Tìm đỉnh khác A gần A nhất.

- Ta chỉ cần tìm trong các đỉnh kề với A, gồm hai đỉnh B và C.
- Ta dựa theo cơ chế Relaxation:

```
if (d(u) + w (u, v) < d(v)) then
      d(v) = d(u) + w(u, v);
      previous(v) = v;
```

+ Xét A, B:

d(A) + w(A, B) = 0 + 4 = 4 < d(B) = ∞

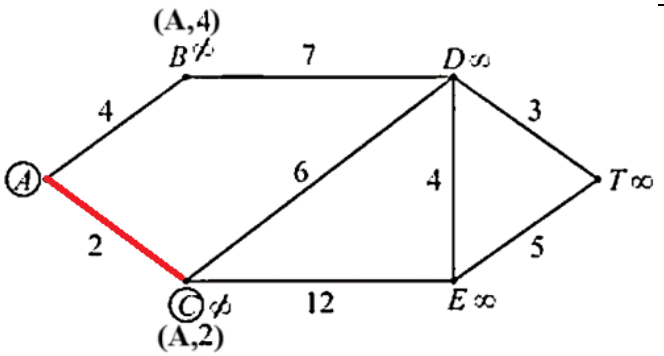
nên ta đổi nhãn của B thành (A, 4). *Với A là Previous (B) (Đỉnh trước B) (Previous (B) = A); d(B) = 4.*

+ Xét A, C:

d(A) + w(A, C) = 0 + 2 = 2 < d(B) = ∞

Tương tự đổi nhãn của C thành (A, 2).

(Previous (C) = A; d(C) = 2).



Ta thấy đỉnh C có nhãn nhỏ nhất trong các đỉnh khác A nên C là đỉnh gần A nhất. Ta cố định nhãn và khoanh tròn đỉnh C.

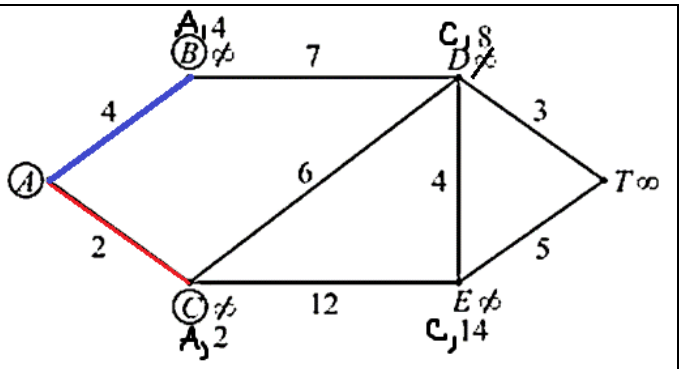
Bước 2. Tìm đỉnh gần A thứ hai.

- Trước hết, xét các đỉnh kề với C (là đỉnh vừa được khoanh tròn ở cuối bước trước) trong các đỉnh chưa khoanh tròn, gồm D và E.
- Xét C, D:

$$d(C) + w(C, D) = 2 + 6 = 8 < d(D) = \infty$$

nên ta đổi nhãn của D thành (C, 8).

- Tương tự, xét ACE thỏa điều kiện Relaxation, đổi nhãn E thành (C, 14). Với d(E) = 14.
- Sau đó, xét các đỉnh kề với A, ta thấy chỉ còn B chưa khoanh tròn, mà đỉnh B ta đã gán nhãn ở bước 1.



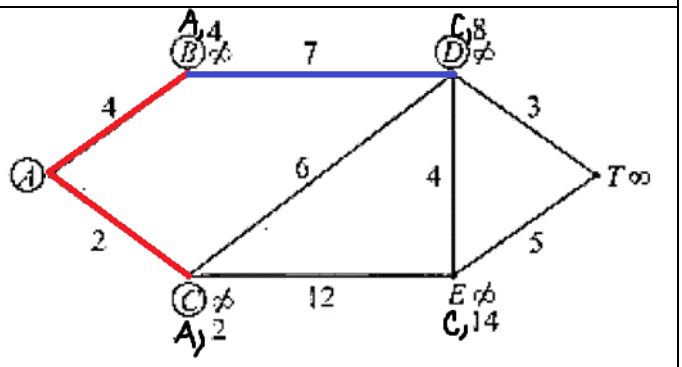
Ta thấy, trong các đỉnh chưa khoanh tròn, đỉnh B có nhãn nhỏ nhất = 4, nên nó là đỉnh gần A thứ hai. Ta cố định nhãn và khoanh tròn đỉnh B.

Bước 3. Tìm đỉnh gần A thứ ba.

- Trước hết, xét các đỉnh kề với A, ta thấy 2 đỉnh B, C đã khoanh tròn.
- Xét các đỉnh kề với C, ta thấy D, E chưa khoanh tròn, và ta đã gán nhãn ở bước 2 với D (C, 8) và E (C, 14).
- Xét các đỉnh kề với B, chỉ có đỉnh D kề với B và chưa khoanh tròn.

$$d(B) + w(B, D) = 4 + 7 = 11 > d(D) = 8.$$

Số này lớn hơn nhãn hiện tại của D nên ta giữ nguyên nhãn này.



Trong các đỉnh chưa khoanh tròn, đỉnh D có nhãn nhỏ nhất, nên nó là đỉnh gần A thứ ba. Ta cố định nhãn và khoanh tròn đỉnh D.

Bước 4. Tìm đỉnh gần A thứ tư.

Trong các đỉnh chưa khoanh tròn, xét các đỉnh kề với D, gồm E và T.

+ Xét D, E:

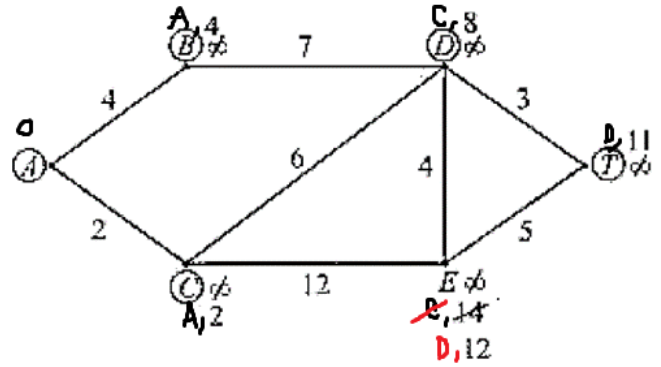
$$d(D) + w(D, E) = 8 + 4 = 12 < d(E) = 14$$

Số này nhỏ hơn nhãn hiện tại của E (là 14) nên ta đổi nhãn của E thành (**D, 12**). Lúc này d(E) = 12.

+ Xét D, T:

$$d(D) + w(D, T) = 8 + 3 = 11 < d(E) = \infty$$

nên đổi nhãn của T thành (D, 11).



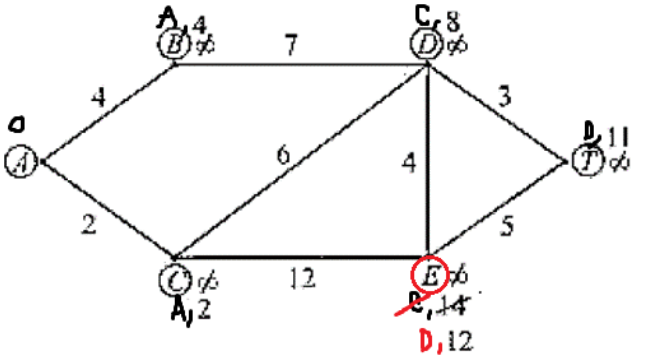
Trong các đỉnh chưa khoanh tròn, đỉnh T có nhãn nhỏ nhất, nên T là đỉnh gần A thứ tư. Ta cố định nhãn và khoanh tròn đỉnh T.

Bước 5. Tìm đỉnh gần A thứ năm.

- Lúc này chỉ tồn tại một đỉnh chưa khoanh tròn là E.
- Có 3 đỉnh đã được khoanh và kề với E là C, D và T.
- Từ bước 4 ta thấy chỉ còn cần xét E và T.

$$d(T) + w(T, E) = 11 + 5 = 16 > d(E) = 12$$

nên giữ nguyên nhãn của E.



Vậy ta khoanh tròn đỉnh E. Kết thúc thuật toán.

Tổng kết: Đường đi ngắn nhất:

A – B: A → B: 4

A – C: A → C: 2

A – D: D ← C ← A: 8 *(D ; Pre(D) = C ; Pre (C) = A)*

A – E: E ← D ← C ← A: 12

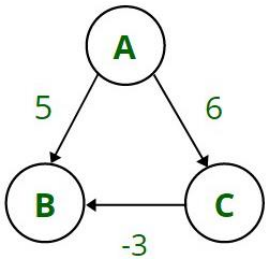
A – T: T ← D ← C ← A: 11

- **Chú ý:** Về sau, khi giải bài toán tìm đường đi ngắn nhất, ta chỉ cần vẽ trên một hình chung cho tất cả các bước.
- Từ lời giải trên, ta có thể mô tả khái quát thuật toán (thuật toán Dijkstra) tìm đường đi ngắn nhất từ đỉnh A đến đỉnh T trên một đồ thị có trọng số như sau.

Mở đầu (cũng gọi là Bước 0): Gán nhãn của A bằng 0, các đỉnh khác bằng 0. Khoanh tròn đỉnh A.
Các bước lặp: Trong mỗi bước lặp thực hiện các thao tác dưới đây:
- Gọi U là đỉnh vừa được khoanh tròn ở bước trước. Trong các đỉnh chưa khoanh tròn, xét lần lượt từng đỉnh V kề với đỉnh U, kiểm tra **d(U) + w(U, V) < d(V).**

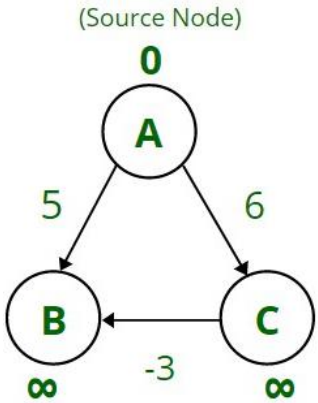
Nếu thỏa điều kiện kiểm tra, ta thực hiện gán d**(V) = d(U) + w(U, V)** & đổi nhãn của V theo cấu trúc: **(đỉnh_trước_V, d(V)).**
- So sánh nhãn của tất cả các đỉnh chưa khoanh tròn. Đỉnh nào có nhãn nhỏ nhất thì khoanh tròn đỉnh đó (nếu có nhiều đỉnh như vậy thì khoanh một đỉnh tùy ý trong số đó).
- Nếu đỉnh T chưa được khoanh tròn thì thực hiện bước lặp tiếp theo, trái lại thì kết thúc các bước lặp.
Kết luận: Từ đỉnh T đi lùi theo Previous (T) = T', rồi Previous(T') = ... Từ đó nhận được đường đi ngắn nhất từ A đến T cùng với độ dài của nó chính là d(T) được lấy từ nhãn.

- Why does Dijkstra's Algorithm fail on negative weights?
+ Let's take a simple example for a better understanding of why **Dijkstra's Algorithm** fails for negative weights.

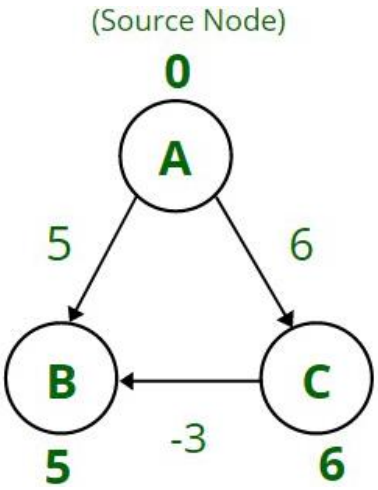


+ Consider directed graph with nodes **A, B,** and **C** which is connected by edges having weights that represent the cost to use this edge. The following are the weights as mentioned in the above diagram: **A –>B = 5, A –>C = 6, C –>B = -3**. Here one weight **C -> B** is negative.

- Consider node **A** as the source node and the task is to find the shortest distance from source node **A** to all the other nodes present in the graph i.e., nodes **B** and **C**.

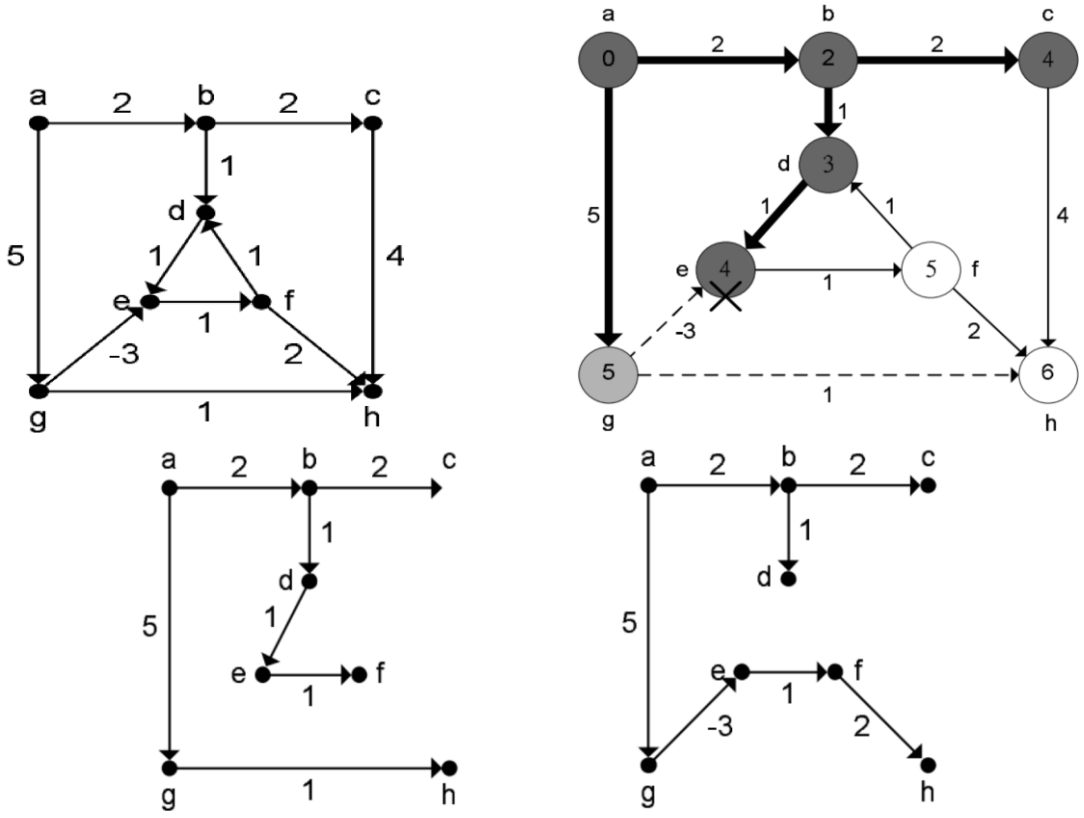


- So, first mark the distance as **0**, at node **A** (as the distance from **A** to **A** is **0**), and then mark this node as visited meaning that it has been included in the shortest path.
- Since in the beginning, the distance of the source node to all other nodes is not known so initialize it as **infinity**. Update this distance if any distance shorter than infinity is found (which is basically the greedy approach)



- Then, update the distance from source node **A** to **B** with the weight of the edge that connects it with **A** which is **5** (because 5 < infinity).
  In a similar way, also update the distance from **A** to **C** which was previously infinity to **6** (as 6 < infinity).
- Now check for the shortest distance from the source node **A** and as 5 is the least distance from **A** to **B**, so mark node **B** as '**visited**'.
  Similarly, the next shortest is **6** so mark node **C** also as visited. At this point, all three nodes of the graph are visited.
- Now the most important step arises here, as it can be seen that by following this algorithm, the shortest distance from **A –> B** is **5** but if traveled the distance via node **C** that is the path **A –> C –> B** the distance will be as **3** (as A–>C = 6 and C–>B = -3), so 6 + (-3) = 3. As **3** is less than **5**, but Dijkstra's algorithm gives the incorrect answer as **5**, which is not the shortest distance. Therefore **Dijkstra's Algorithm** fails for negative cases.

** Another example of Dijkstra's algorithm failure on negative weight:*



- Để giải quyết khuyết điểm này, ta có thuật toán Bellman – Ford.

6.3 Thuật toán Bellman – Ford

- Áp dụng cho đồ thị **có hướng**, có trọng số G (V, E, C, s), trong đó s là đỉnh xuất phát, C là trọng số (có thể có cạnh có trọng số âm).
- Đi tìm đường đi ngắn nhất từ s đến mọi đỉnh còn lại *(khi đồ thị không có chu trình âm)*, hoặc kiểm tra ***đồ thị chứa chu trình âm*** hay không.
- *Dựa trên "Principle of Relaxation".*

```
Algorithm: Initialize-Single-Source (G, S)
{
      for all vertices q ∈ V do{
            d[q] := ∞;
            prev[q]: = NULL;
      }
      d[S] := 0; //S is the start vertice
}
```

```
Algorithm: Relax (p, q, w)
{
      if (d[q] > d[p] + w[p, q]){
            Then d[q] := d[p] + w[p, q];
            prev[q]: = p;
      }
}
```

```
Algorithm: BELLMAN-FORD (G,W,S)
{
      Initialize-Single-Source(G,S)
      for j: =1 to |V|-1
            do for every edge (p, q) ∈ E
                  do RELAX (p, q, W);

      for every edge (p, q) ∈ E
            do if d[q] > d[p] + w(p, q)
                  then return FALSE
      return TRUE
      // no negative weight cycle
}
```

- Nếu sau **|V|-1 vòng lặp Relaxation**, vẫn có sự thay đổi giá trị **`d[v]`** cho một đỉnh nào đó, thì **chắc chắn** tồn tại **chu trình âm** trong đồ thị.
- Lý do là vì sau |V|-1 vòng lặp, giá trị **`d[v]`** đã được **cập nhật** dựa trên tất cả các đường đi tiềm năng. Nếu vẫn có thay đổi sau vòng lặp tiếp theo, thì có nghĩa là có **đường đi** mới từ đỉnh nguồn đến **`v`** **ngắn hơn** so với tất cả các đường đi đã được tìm thấy trước đó. Đường đi này **chắc chắn** phải **bao gồm** một chu trình âm.

- Why Relaxing Edges N-1 times, gives us Single Source Shortest Path?

In the worst-case scenario, a shortest path between two vertices can have at most N-1 edges, where N is the number of vertices. This is because a ***simple path in a graph with N vertices can have at most N-1 edges***, as it's *impossible to form a closed loop without revisiting a vertex (because any path with more than n − 1 edges has a cycle in it).*

By relaxing edges N-1 times, the Bellman-Ford algorithm ensures that the distance estimates (ước tính) for all vertices have been updated to their optimal (tối ưu) values, assuming (giả sử) the graph doesn't contain any negative-weight cycles reachable from the source vertex. If a graph contains a negative-weight cycle reachable from the source vertex, the algorithm can detect (phát hiện) it after N-1 iterations, since the negative cycle disrupts (gián đoạn) the shortest path lengths.

In summary, relaxing edges N-1 times in the Bellman-Ford algorithm guarantees (đảm bảo) that the algorithm has explored all possible paths of length up to N-1, which is the maximum possible length of a shortest path in a graph with N vertices. This allows the algorithm to correctly calculate the shortest paths from the source vertex to all other vertices, given that there are no negative-weight cycles.

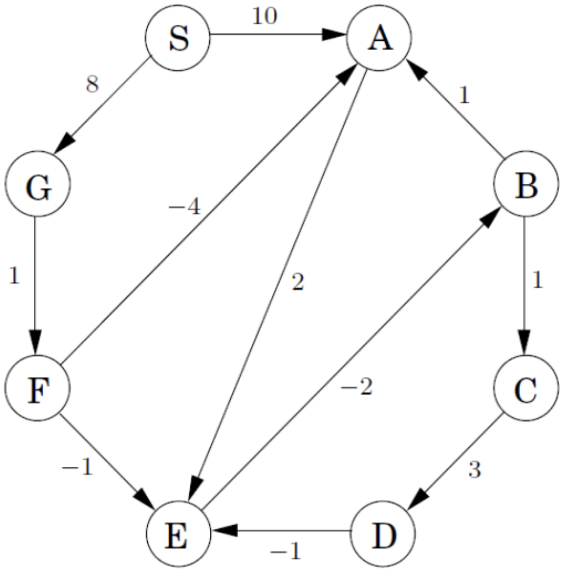
*** Sau |V|-1 vòng lặp:**

- **Tất cả các cạnh** trong đồ thị đã được **xét qua ít nhất một lần**.
- Do đó, **tất cả các đường đi tiềm năng** đã được **xét qua**.
- Giá trị **`d[v]`** cho mỗi đỉnh v đã được **cập nhật** chính xác, thể hiện **khoảng cách ngắn nhất** từ đỉnh nguồn đến **`v`**.

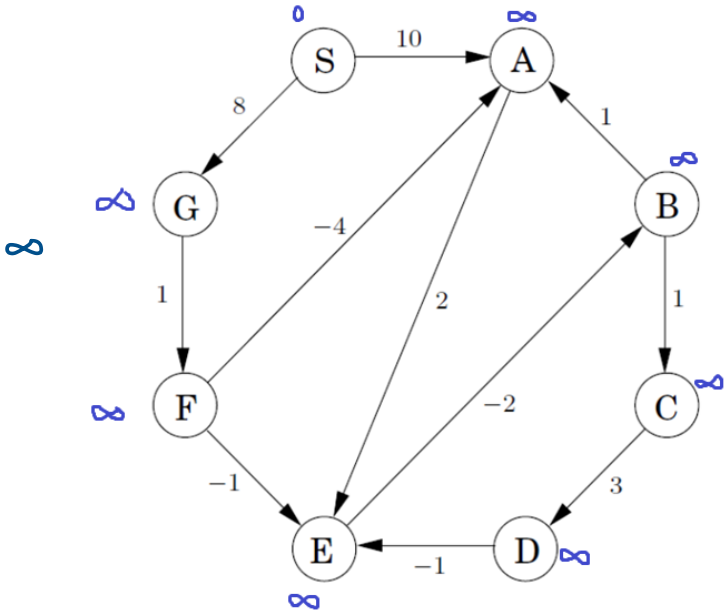
- Đối với cách chạy thuật toán bằng tay:

- **B1:** Liệt kê tất cả các cạnh.
- **B2:** Vận dụng cơ chế Relaxation đối với từng cạnh, lặp lại bước này |số_đỉnh| - 1 lần.
- **B3:** Nếu thấy các đỉnh có nhãn âm cứ giảm giá trị theo các lần lặp, dừng lại và kiểm tra xem tại đó có chu trình âm hay không. Nếu có thì kết luận đồ thị không tồn tại đường đi ngắn nhất. Ngược lại thực hiện tiếp bước 2.

Ví dụ:



- Danh sách các cạnh: *E = {(S,A),(S,G),(A,E),(B,A),(B,C),(C,D),(D,E),(E,B),(F,A),(F,E),(G,F)}*
- Khởi tạo nhãn cho các đỉnh:



@ Đồ thị có 8 đỉnh, do đó cần relax 7 lần.

- Relax các cạnh lần 1:

E = {(S,A),(S,G),(A,E),(B,A),(B,C),(C,D),(D,E),(E,B),(F,A),(F,E),(G,F)}

d(S) + w(S, A) = 0 + 10 = 10 < d(A) = ∞ → A: S, 10

d(S) + w(S, G) = 0 + 8 = 8 < d(A) = ∞ → G: S, 8

d(A) + w(A, E) = 10 + 2 = 12 < d(E) = ∞ → E: A, 12

d(B) + w(B, A) = ∞ + 1 = ∞ > d(A) = 10

d(B) + w(B, C) = ∞ + 1 = ∞ = d(C) = ∞

d(C) + w(C, D) = ∞ + 3 = ∞ = d(D) = ∞

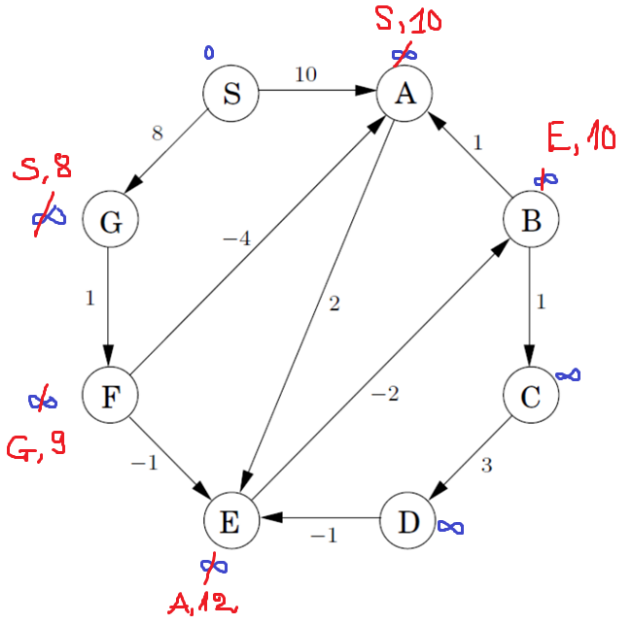
d(D) + w(D, E) = ∞ - 1 = ∞ > d(E) = 12

d(E) + w(E, B) = 12 - 2 = 10 < d(B) = ∞ → **B: E, 10**

d(F) + w(F, A) = ∞ - 4 = ∞ > d(A) = 10

d(F) + w(F, E) = ∞ - 1 = ∞ > d(E) = 12

d(G) + w(G, F) = 8 + 1 = 9 < d(F) = ∞ → **F: G, 9**



- Relax các cạnh lần 2:

E = {(S,A),(S,G),(A,E),(B,A),(B,C),(C,D),(D,E),(E,B),(F,A),(F,E),(G,F)}

d(S) + w(S, A) = 0 + 10 = 10 = d(A)

d(S) + w(S, G) = 0 + 8 = 8 = d(A)

d(A) + w(A, E) = 10 + 2 = 12 = d(E)

d(B) + w(B, A) = 10 + 1 = 11 > d(A) = 10

d(B) + w(B, C) = 10 + 1 = 11 < d(C) = ∞ → **C: B, 11**

d(C) + w(C, D) = 11 + 3 = 14 < d(D) = ∞ → **D: C, 14**

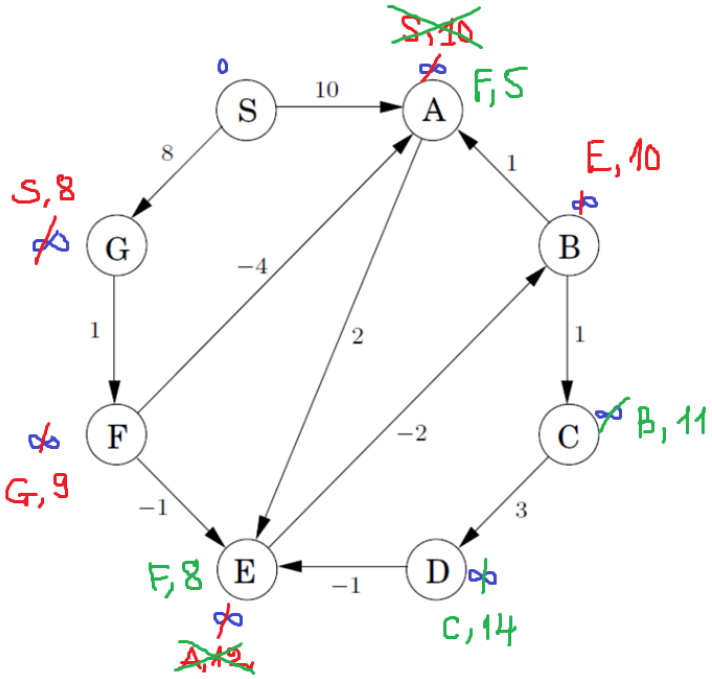
d(D) + w(D, E) = 14 - 1 = 13 > d(E) = 12

d(E) + w(E, B) = 12 - 2 = 10 = d(B)

d(F) + w(F, A) = 9 - 4 = 5 < d(A) = 10 → **A: F, 5**

d(F) + w(F, E) = 9 - 1 = 8 < d(E) = 12 → **E: F, 8**

d(G) + w(G, F) = 8 + 1 = 9 = d(F)



- Relax các cạnh lần 3:

E = {(S,A),(S,G),(A,E),(B,A),(B,C),(C,D),(D,E),(E,B),(F,A),(F,E),(G,F)}

d(S) + w(S, A) = 0 + 10 = 10 > d(A) = 5

d(S) + w(S, G) = 0 + 8 = 8 = d(A)

d(A) + w(A, E) = 5 + 2 = 7 < d(E) = 8 → **E: A, 7**

d(B) + w(B, A) = 10 + 1 = 11 > d(A) = 5

d(B) + w(B, C) = 10 + 1 = 11 = d(C)

d(C) + w(C, D) = 11 + 3 = 14 = d(D)

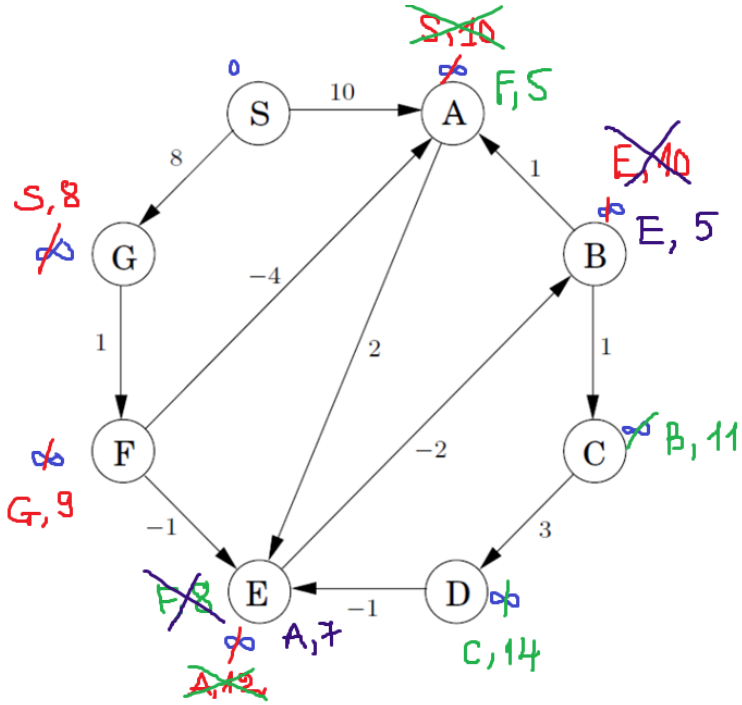
d(D) + w(D, E) = 14 - 1 = 13 > d(E) = 7

d(E) + w(E, B) = 7 - 2 = 5 < d(B) = 10 → **B: E, 5**

d(F) + w(F, A) = 9 - 4 = 5 = d(A)

d(F) + w(F, E) = 9 - 1 = 8 > d(E) = 7

d(G) + w(G, F) = 8 + 1 = 9 = d(F)



- Relax các cạnh lần 4:

E = {(S,A),(S,G),(A,E),(B,A),(B,C),(C,D),(D,E),(E,B),(F,A),(F,E),(G,F)}

d(S) + w(S, A) = 0 + 10 = 10 > d(A) = 5

d(S) + w(S, G) = 0 + 8 = 8 = d(A)

d(A) + w(A, E) = 5 + 2 = 7 = d(E)

d(B) + w(B, A) = 5 + 1 = 6 > d(A) = 5

d(B) + w(B, C) = 5 + 1 = 6 < d(C) = 11 → **C: B, 6**

d(C) + w(C, D) = 6 + 3 = 9 < d(D) = 14 → **D: C, 9**

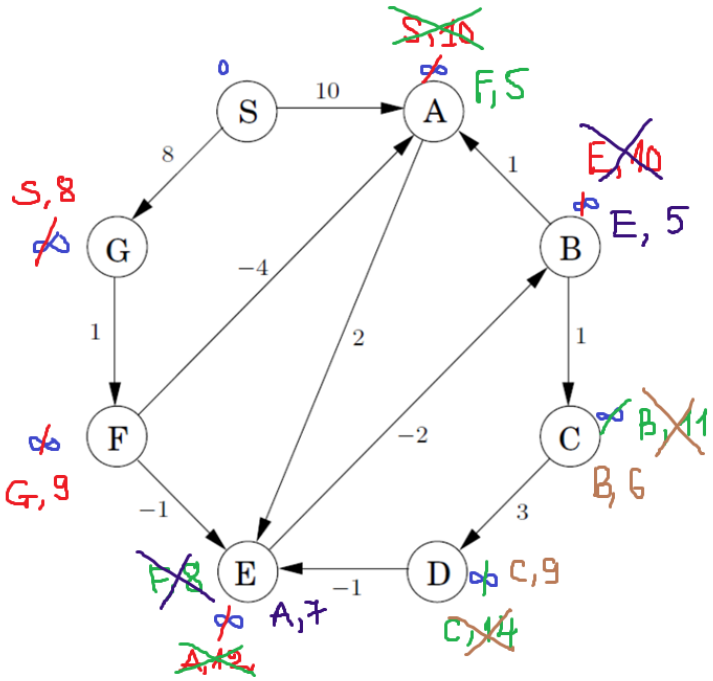
d(D) + w(D, E) = 9 - 1 = 8 > d(E) = 7

d(E) + w(E, B) = 7 - 2 = 5 = d(B)

d(F) + w(F, A) = 9 - 4 = 5 = d(A)

d(F) + w(F, E) = 9 - 1 = 8 > d(E) = 7

d(G) + w(G, F) = 8 + 1 = 9 = d(F)



- Relax các cạnh lần 5:

E = {(S,A),(S,G),(A,E),(B,A),(B,C),(C,D),(D,E),(E,B),(F,A),(F,E),(G,F)}

d(S) + w(S, A) = 0 + 10 = 10 > d(A) = 5

d(S) + w(S, G) = 0 + 8 = 8 = d(A)

d(A) + w(A, E) = 5 + 2 = 7 = d(E)

d(B) + w(B, A) = 5 + 1 = 6 > d(A) = 5

d(B) + w(B, C) = 5 + 1 = 6 = d(C)

d(C) + w(C, D) = 6 + 3 = 9 = d(D)

d(D) + w(D, E) = 9 - 1 = 8 > d(E) = 7

d(E) + w(E, B) = 7 - 2 = 5 = d(B)

d(F) + w(F, A) = 9 - 4 = 5 = d(A)

d(F) + w(F, E) = 9 - 1 = 8 > d(E) = 7

→ Đồ thị không thay đổi về nhãn của các đỉnh nữa.

- Relax các cạnh lần 6, 7: Kết quả tương tự như lần 5.

Vậy đường đi ngắn nhất từ đỉnh S đến các đỉnh còn lại là:

- **S – A:** A ← F ← G ← S.
- **S – B:** B ← E ← A ← F ← G ← S.
- **S – C:** C ← B ← E ← A ← F ← G ← S.
- **S – D:** D ← C ← B ← E ← A ← F ← G ← S.
- **S – E:** E ← A ← F ← G ← S.
- **S – F:** F ← G ← S.
- **S – G:** G ← S.

Given a graph G = (V, E) with positive edge weights, the Bellman-Ford algorithm and Dijkstra's algorithm can *produce different shortest-path trees* despite *always producing the same shortest-path* ***weights.***

Solution: True. Both algorithms are guaranteed *(đảm bảo)* to produce the same shortest-path weight, but if there are multiple shortest paths, Dijkstra's will choose the shortest path according to the ***greedy strategy***, and Bellman-Ford will choose the shortest path depending on the ***order of relaxations***, and the two shortest path trees may be different.

6.4 So sánh các thuật toán

	Thuật toán Dijikstra	Thuật toán Bellman-Ford
Mục đích	Tìm đường đi ngắn nhất từ đỉnh nguồn đến các đỉnh còn lại (Khi đồ thị có toàn bộ các cạnh là trọng số dương thì hiệu quả như nhau)	
Ưu điểm	Đơn giản, dễ hiểu, hiệu quả cho đồ thị không có chu trình âm.	Có thể xử lý đồ thị có chu trình âm, có thể tìm đường đi với trọng số âm & dương.
Nhược điểm	Không thể xử lý đồ thị có chu trình âm chính xác.	Phức tạp hơn Dijkstra, có thể tốn thời gian hơn cho đồ thị lớn.
Thời gian code	Lâu	Nhanh
Tìm được chu trình âm	Không	Có
Thuật toán cơ sở	Tham lam (Chọn cạnh nào làm tổng trọng số đang xét là nhỏ nhất)	Cơ chế Relaxation & thứ tự chọn cạnh
Độ phức tạp thời gian	O (E x logV)	O (E x V)
Ứng dụng	Hệ thống định vị GPS, lập kế hoạch mạng, v.v.	Hệ thống định tuyến giao thông, hệ thống mạng viễn thông, v.v.

6.5 Ứng dụng về đồ thị có trọng số âm

- Thu chi: Chi tiền thì thể hiện trọng số âm, thu tiền thì thể hiện trọng số dương.
- Gói tin qua một cổng/ nhấn một nút… thì sẽ có độ trễ, được thể hiện bằng trọng số âm.
- Trạng thái của một vấn đề nào đó: Tích cực thì trọng số dương, tiêu cực thì trọng số âm.
    - Đánh giá rủi ro trong các khoản đầu tư. Trọng số âm có thể được sử dụng để biểu thị các mối tương quan tiêu cực giữa các khoản đầu tư, chẳng hạn như rủi ro thị trường hoặc rủi ro tín dụng. Các thuật toán như thuật toán PageRank có thể được sử dụng để xác định các khoản đầu tư có rủi ro cao trong danh mục đầu tư.
    - Xây dựng hệ thống đề xuất, chẳng hạn như hệ thống đề xuất phim hoặc sách. Trọng số âm có thể được sử dụng để biểu thị các mối quan hệ tiêu cực giữa các mục, chẳng hạn như khi hai phim có cùng thể loại nhưng được đánh giá thấp.
    - Mô hình hóa các mạng xã hội, trong đó các cá nhân hoặc nhóm được biểu thị bằng các nút và các mối quan hệ giữa họ được biểu thị bằng các cạnh. Trọng số âm có thể được sử dụng để mô tả các hiện tượng như sự thù hận hoặc bất đồng chính kiến.
    - Mô hình hóa các cuộc bầu cử, trong đó các ứng cử viên và cử tri được biểu thị bằng các nút và các mối quan hệ giữa họ được biểu thị bằng các cạnh. Trọng số âm có thể được sử dụng để mô tả các hiện tượng như chiến dịch tranh cử tiêu cực hoặc sự phân cực chính trị.
- Mô tả mạng lưới giao thông, trong đó thời gian di chuyển giữa hai địa điểm được biểu thị bằng trọng số cạnh. Trọng số âm có thể được sử dụng để mô tả các hiện tượng như tắc đường hoặc phí cầu đường.
- Mô hình hóa mạng xã hội, trong đó các cá nhân được biểu thị bằng các nút và các mối quan hệ giữa họ được biểu thị bằng các cạnh. Trọng số âm có thể được sử dụng để mô tả các hiện tượng như thông tin sai lệch hoặc thù địch.
- Mô hình hóa mạng lưới điện, trong đó các trạm phát điện và trạm biến áp được biểu thị bằng các nút và đường dây điện được biểu thị bằng các cạnh. Trọng số âm có thể được sử dụng để mô tả các chi phí liên quan đến việc truyền tải điện, chẳng hạn như tổn thất điện năng trên đường dây.
- Mô hình hóa mạng lưới kho hàng, trong đó các kho hàng được biểu thị bằng các nút và các tuyến vận chuyển giữa các kho hàng được biểu thị bằng các cạnh. Trọng số âm có thể được sử dụng để mô

tả các chi phí liên quan đến việc vận chuyển hàng hóa, chẳng hạn như chi phí nhiên liệu hoặc phí đường bộ.

- Mô hình hóa chuỗi cung ứng, trong đó các nhà cung cấp, nhà sản xuất, nhà phân phối và khách hàng được biểu thị bằng các nút và các mối quan hệ giữa họ được biểu thị bằng các cạnh. Trọng số âm có thể được sử dụng để mô tả các chi phí liên quan đến việc di chuyển hàng hóa qua chuỗi cung ứng, chẳng hạn như chi phí lưu kho hoặc chi phí sản xuất.