Cấu trúc Đồ thị

Phần 2: Các bài toán đồ thị cơ bản

Nguyễn Thanh Bình

Viện Điện tử Viễn thông – ĐHBK Hà Nội

Nội dung chính

Tìm đường đi ngắn nhất

Bài toán này được chia thành một số bài toán con:

1. Tìm đường đi ngắn nhất giữa hai nút cho trước;
2. Tìm đường đi ngắn nhất từ một nút đến tất cả các nút còn lại;
3. Tìm đường đi ngắn nhất giữa mọi cặp nút;

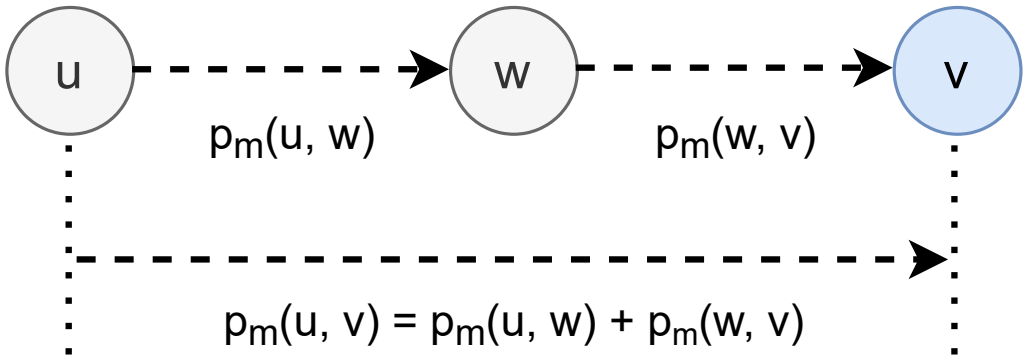
Lưu ý:

Hiện đã có các giải thuật cho các bài toán 2 và 3. Việc giải bài toán 1 hiện nay phải thông qua bài toán 2 chứ chưa có một lời giải trực tiếp.

Tìm đường đi ngắn nhất

Tính chất đường đi ngắn nhất giữa hai nút

Cho đồ thị G(V, E), và hai đỉnh u, v $\in$ V. Gọi $p_m(u, v)$ là đường đi ngắn nhất từ u đến v. Khi đó, với mọi đỉnh w nằm trên $p_m(u, v)$, ta đều có $p(u, w)$ và $p(w, v)$ đều là các đường đi ngắn nhất. Tính chất này còn được gọi là tính chất bộ phận.



Tìm đường đi ngắn nhất

Chiến lược chung

Từ tính chất trên, có thể suy ra một chiến lược cơ bản để giải quyết bài toán tìm đường đi ngắn nhất giữa hai đỉnh u (đỉnh nguồn) và v (đỉnh đích) như sau:

- Vì có thể có nhiều đường đi giữa u và v, nên phương pháp tìm kiếm cần kiểm tra và so sánh tất cả đường dẫn. Tất nhiên, trong trường hợp đặc biệt, nếu chỉ có một đường dẫn giữa u và v thì đó chính là đường dẫn phải tìm.
- Vì tất cả các đường dẫn đều bắt đầu từ u và kết thúc ở v, nên phương pháp duyệt theo chiều sâu bắt đầu từ nút u sẽ được sử dụng để tìm kiếm và so sánh các đường dẫn này.

Tìm đường đi ngắn nhất

Lưu ý

Nếu trong đồ thị tồn tại một chu trình mà có tổng trọng số âm thì đường dẫn mà chứa chu trình này sẽ không có giá trị nhỏ nhất (vì cứ đi một vòng chu trình thì giá trị lại giảm đi và giảm đến âm vô cùng). Chính vì vậy, các đồ thị mà ta xét trong phần này sẽ không có chu trình như vậy.

Tìm đường đi ngắn nhất từ một nút đến tất cả các nút còn lại

Phát biểu bài toán

Cho đồ thị G(V, E) với $V = v_0, v_1, ..., v_{M-1}$ (M đỉnh) và $E = e_0, e_1, ..., e_{N-1}$ (N cạnh). Chọn một đỉnh bất kỳ làm nguồn, ta giả sử là đỉnh $s = v_0$. Yêu cầu tìm các đường đi ngắn nhất từ đỉnh s đến các đỉnh còn lại.

Các giải thuật tìm đường đi ngắn nhất từ một nút đến tất cả các nút còn lại

- Giải thuật Dijkstra: chỉ áp dụng cho đồ thị không có trọng số âm;
- Giải thuật Bellman-Ford: áp dụng cả cho đồ thị có trọng số âm, và có thể dùng để phát hiện chu trình âm;

Giải thuật Dijkstra

Ý tưởng chính

Áp dụng chiến lược quy nạp, giải thuật sẽ tìm và cập nhật dần dần các thông tin về đường đi ngắn nhất giữa nút nguồn s và các nút còn lại (sau này ta nói ngắn gọn là đường đi ngắn nhất hoặc đường đi ngắn nhất đến nút đích). Gọi F là tập các nút đích mà đường đi ngắn nhất đã tìm được. Ban đầu ta có $F = \{s\}$. Nhiệm vụ chính của giải thuật là bổ sung dần các nút thích hợp vào F cho đến khi $F = V$.

Giải thuật Dijkstra

Chi tiết giải thuật

Ta giả sử ở một bước nào đó, ta đã có tập $F \neq \emptyset$ và $F \subset (V \backslash s)$ (chưa tìm hết tất cả các nút đích), và ta cần tìm một nút $v_m \in V \backslash F$ phù hợp để đưa vào F. Tức là, thông tin về đường đi ngắn nhất đến v_m là xác định được.

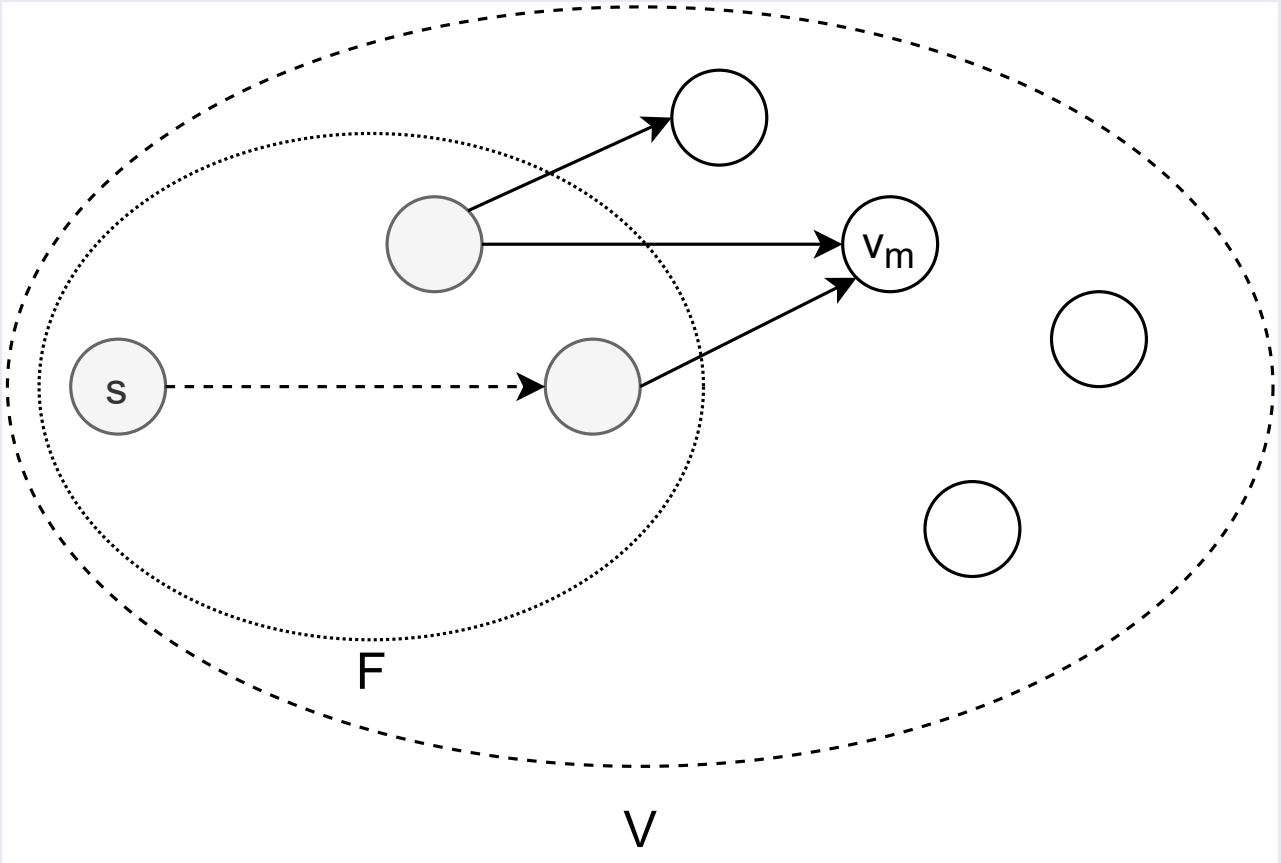
Nút v được gọi là nút kề của F nếu tồn tại ít nhất một nút $v_k \in F$ sao cho v là nút kề của v_k. Còn trái lại thì v không kề với F.

Gọi d(v) là độ dài đường đi ngắn nhất từ nút nguồn s đến nút v. Do đó, nếu $v \in F$ thì d(v) đã xác định. Còn trái lại giá trị này chưa xác định (để tiện thì ban đầu giá trị này ở mỗi nút được đặt bằng ∞).

Người ta đã chứng minh rằng v_m phải là nút kề của F và $d(v_m)$ có giá trị nhỏ nhất trong số các nút thuộc tập $V \backslash F$.

Giải thuật Dijkstra

Chi tiết giải thuật



Giải thuật Dijkstra

Giải thuật tìm nút v_m

Vì tập F đã xác định, nên tập các đỉnh kề với F cũng hoàn toàn xác định. Từ đó, giá trị độ dài đường đi ngắn nhất đến các đỉnh kề này cũng có thể xác định được.
Giả sử v_k là một đỉnh kề của F. Khi đó ta có:

$$d(v_k) = min\{(d(v_f) + |v_f, v_k|) \mid \forall v_f \in F\}$$

Từ các đỉnh kề của F, ta cũng dễ dàng tìm được v_m vì:

$$d(v_m) = min\{d(v_k) \mid \forall v_k \in \text{tập kề của } F\}$$

Ghi chú

Ký hiệu $|v_f, v_k|$ để chỉ độ dài cạnh nối hai đỉnh v_f và v_k.

Giải thuật Dijkstra

Cài đặt

Chuẩn bị: Ta sử dụng hai cấu trúc dữ liệu để lưu trữ các thông tin về tập đường đi ngắn nhất này:

- Mảng $dist[M]$ để lưu các khoảng cách ngắn nhất từ nguồn đến các đỉnh còn lại, trong đó $dist[i]$ là khoảng cách từ nguồn đến đỉnh v_i ($dist[0] = 0$);
- Mảng $prev[M]$ để lưu các nút đứng trước nhằm tìm lại con đường ngắn nhất từ nguồn đến các nút khác, trong đó $prev[k] = v_i$, nghĩa là nút v_i đứng ngay trước nút v_k trên đường đi ngắn nhất từ nút nguồn đến một nút khác có đi qua v_i và v_k.

Giải thuật Dijkstra

Cài đặt

Xử lý: Gọi F là tập các nút mà ta đã tìm được đường đi ngắn nhất xuất phát từ nút nguồn v_0. Quá trình xử lý gồm các bước như sau:

- Khởi tạo ban đầu:

```
F = {v0};
dist[0] = 0;
for (i = 1; i<M; i++) {
  if (v0, vi) ∈ E {
    dist[i] = |v0, vi| ;
    prev[i] = v0;
  } else dist[i] = ∞ ;
}
```

- Tìm và cập nhật lần lượt các nút đích:

Giải thuật Dijkstra

Cài đặt (tiếp)

- Tìm và cập nhật lần lượt các nút đích: giai đoạn này thực hiện trong M-1 bước, mỗi bước sẽ tìm được một nút đích. Ta đánh số các bước là i = 1, 2, .., (M-1).

  Ở bước thứ i, chọn đỉnh $v_m \in V \backslash F$ có độ dài đến s ngắn nhất, tức là:

  $$dist[m] = MIN(dist[k], \forall v_k \in V \backslash F).$$

  Sau đó, cập nhật lại các trạng thái về độ dài tại các $dist[k]$ và $prev[k]$, $\forall v_k \in F$ nếu cần.
  Tức là nếu $\exists$ cạnh (v_m, v_k) và
  $dist[k] > dist[m] + |(v_m, v_k)|$ thì cập nhật:
  $dist[k] = dist[m] + |(v_m, v_k)|$ và $prev[k] = v_m$.

Giải thuật Dijkstra

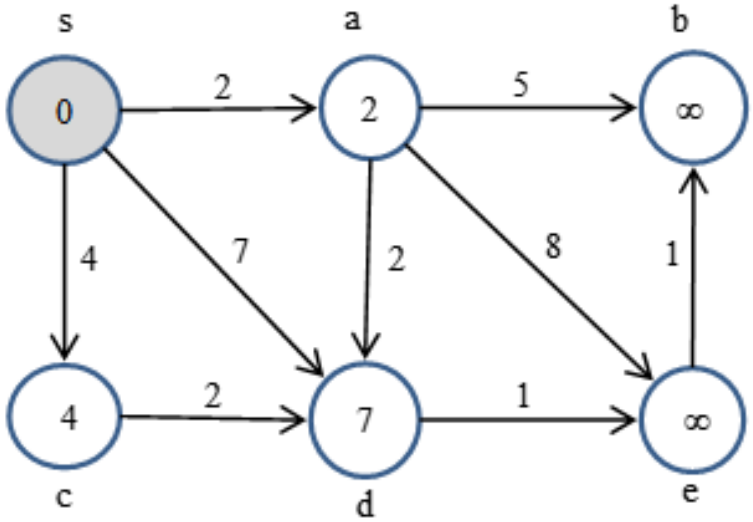
Thủ tục mô tả cài đặt như sau:

```
F = {v0}; dist[0] = 0;
for (i=1; i<M;i++) {
    if (v0, vi) ∈ E {
        dist[i] = |v0,vi| ; prev[i]=v0;
    } else dist[i] = ∞;
}
for (i=1; i<M; i++){
    Tìm vm∈V\F sao cho dist[m] = MIN (dist[k] ∀vk∈V\F);
    F = F∪ vm;
    for each vk∈V\F
    if ((vm, vk) ∈ E) &&(dist[k] > dist[m] + | vm, vk|){
        dist[k] = dist[m] + |vm, vk|;
        prev[k] = vm ;
    }
}
```

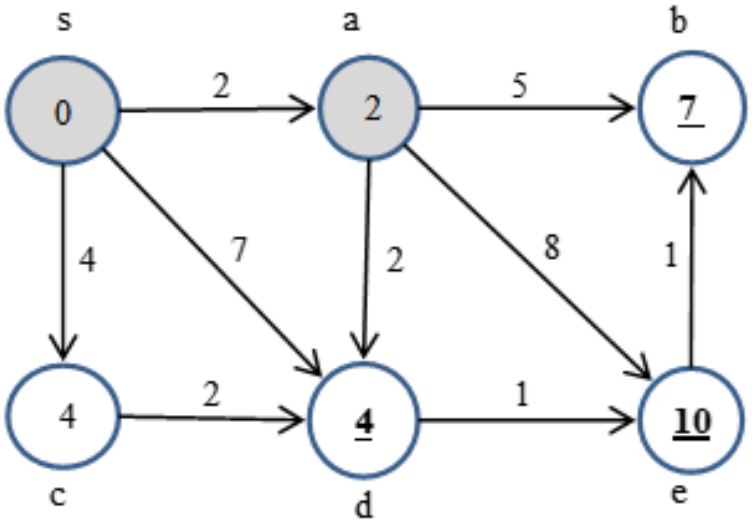
Ví dụ: minh họa giải thuật Dijkstra

- Trong các hình sau, giá trị của mỗi nút chính là độ dài đường đi hiện tại từ nút nguồn (nút s) tới nút đó. Các giá trị này có thể được cập nhật lại nhiều lần trong suốt quá trình tìm kiếm (các giá trị được tô đậm và gạch chân là vừa được cập nhật lại).
- Có hai loại nút được biểu diễn trong hình: nút xám, biểu diễn các nút trong tập F, tập các nút đích đã tìm được đường đi ngắn nhất; Nút trắng là nút đang trong quá trình kiểm tra.
- Các bước được đánh số bên dưới, trong đó bước (0) là bước khởi tạo.

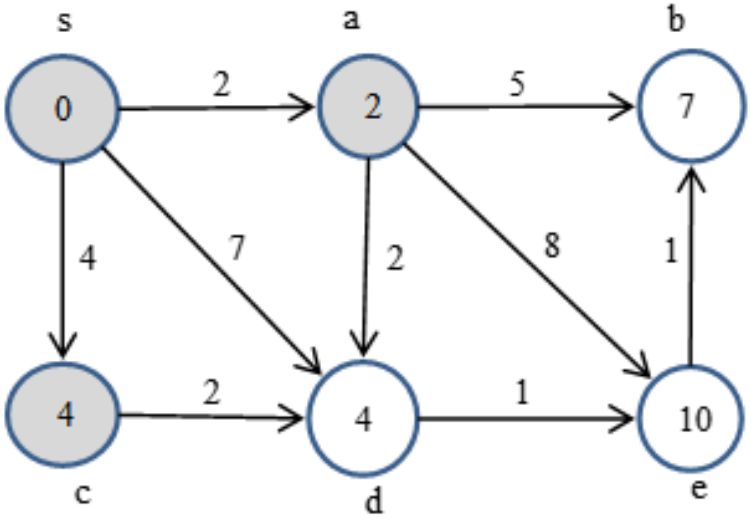
Ví dụ: minh họa giải thuật Dijkstra



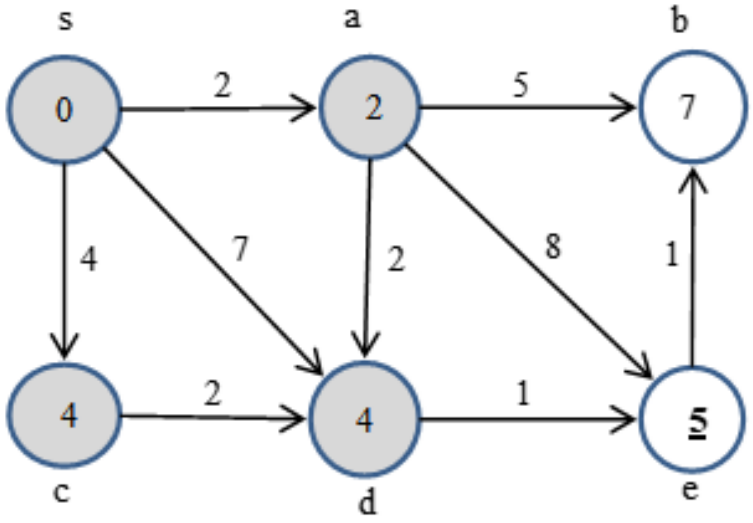
(0)



(1)

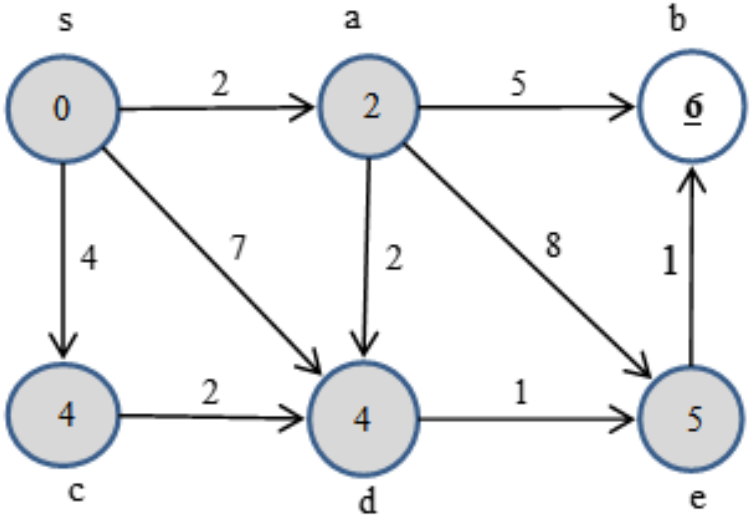


(2)

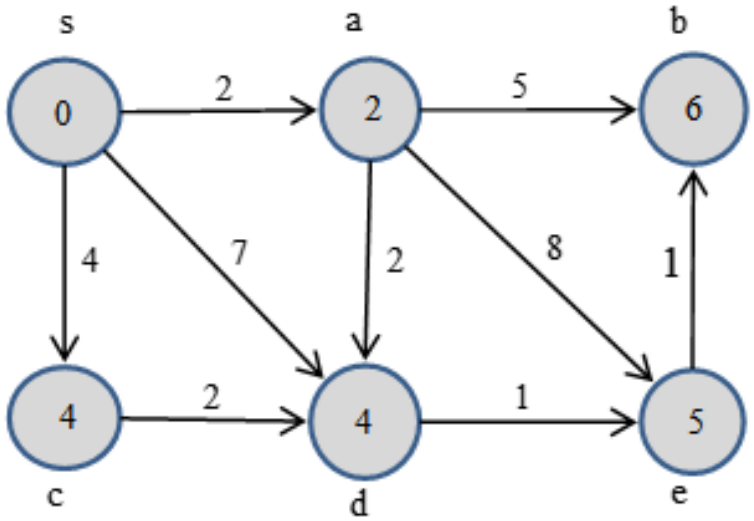


(3)

Ví dụ: minh họa giải thuật Dijkstra



(4)



(5)

Giải thuật Bellman-Ford

Ý tưởng giải thuật

- Đầu vào: đồ thị G(V, E) với $V = v_0, v_1, ..., v_{M-1}$ (M đỉnh) và $E = e_0, e_1, ..., e_{N-1}$ (N cạnh). Đỉnh nguồn $s = v_0$.
- Đầu ra: Các đường đi ngắn nhất từ đỉnh s đến các đỉnh còn lại.
- Ý tưởng: Giả sử d(v) là độ dài đường đi ngắn nhất đến v mà ta muốn đạt được. Giá trị này sẽ được cập nhật dần dần cho từng nút cho đến khi chúng thực sự đạt đến giá trị mong muốn. Ban đầu ta sẽ có: d(s) = 0; còn lại d(v) = ∞ với $v \neq s$; Giải thuật này sử dụng chiến lược vét cạn bằng cách kiểm tra từng cạnh của đồ thị và cập nhật lại độ dài đường đi ngắn nhất cho các nút nếu cần.

Giải thuật Bellman-Ford

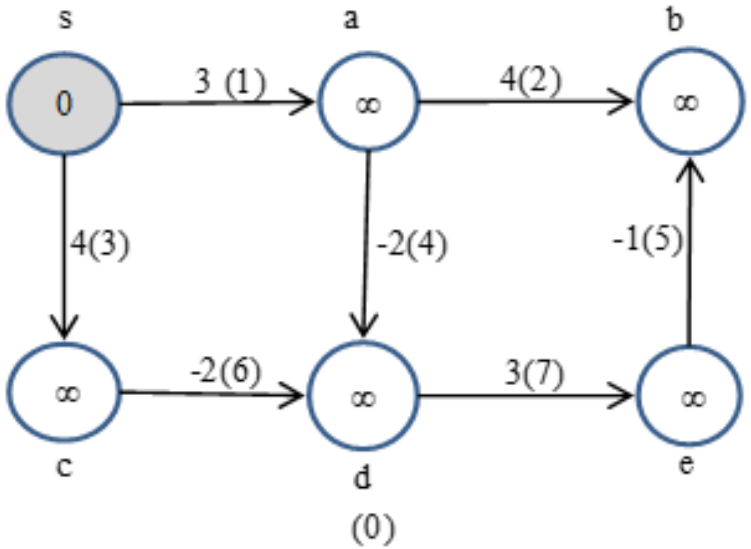
Cài đặt

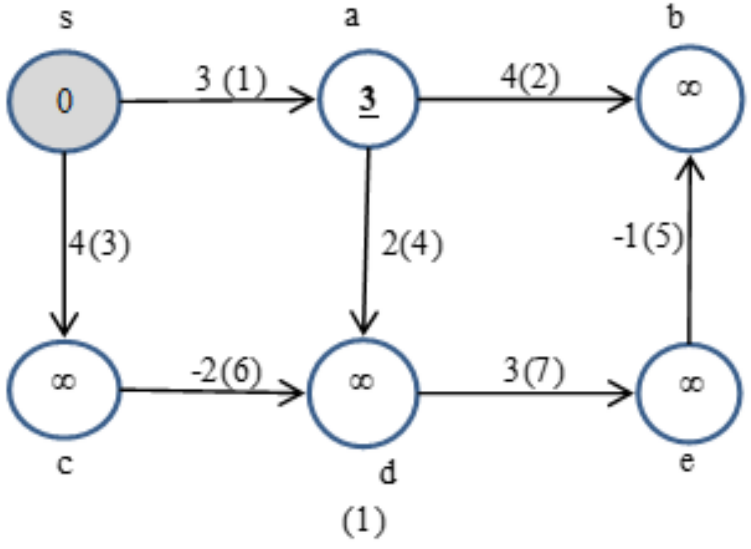
Sử dụng hai mảng dist[M] và prev[M] có ý nghĩa tương tự như trong cài đặt giải thuật Dijkstra. Khi đó, thủ tục cài đặt cho giải thuật có dạng sau:

```
dist[0] = 0;
for (i=1; i<M; i++) dist[i] = ∞;
while ∃ cạnh (u, v) ∈E sao cho dist[u] + |u,v| < dist[v] {
      dist[v] = dist[u] + |u,v| ;
      prev[v] = u;
}
```

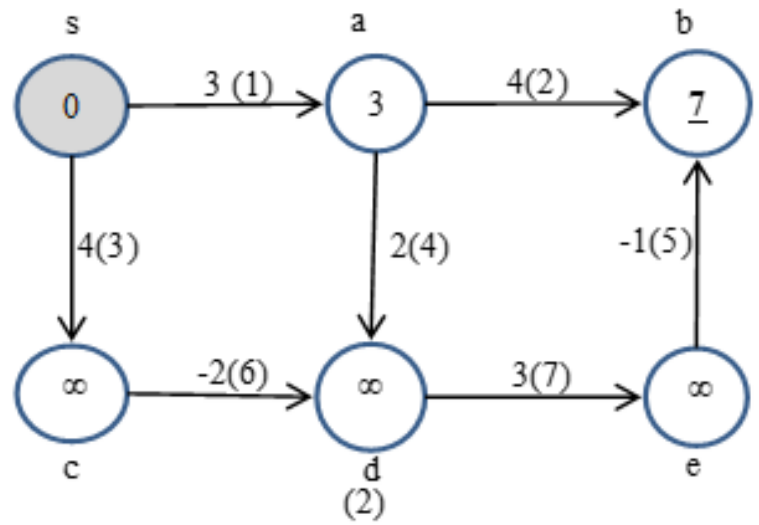
Ví dụ: minh họa hoạt động của giải thuật Bellman-Ford

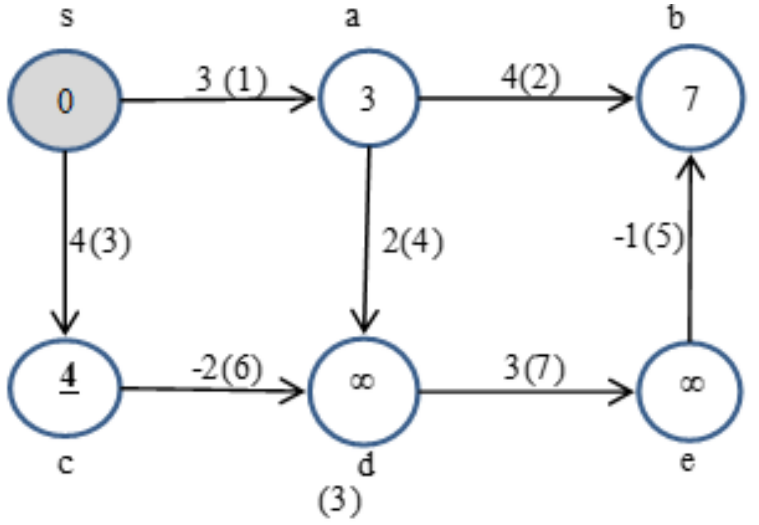
Các hình minh họa hoạt động của giải thuật qua một đồ thị có cả trọng số âm và dương. Ở hình này, ở mỗi cạnh, ngoài giá trị trọng số, còn thứ tự mà cạnh đó sẽ được kiểm tra (giá trị nằm trong ngoặc đơn). Ở trong mỗi đỉnh là giá trị đường đi ngắn nhất ở thời điểm hiện tại (giá trị tô đậm và gạch dưới là giá trị vừa được cập nhật).

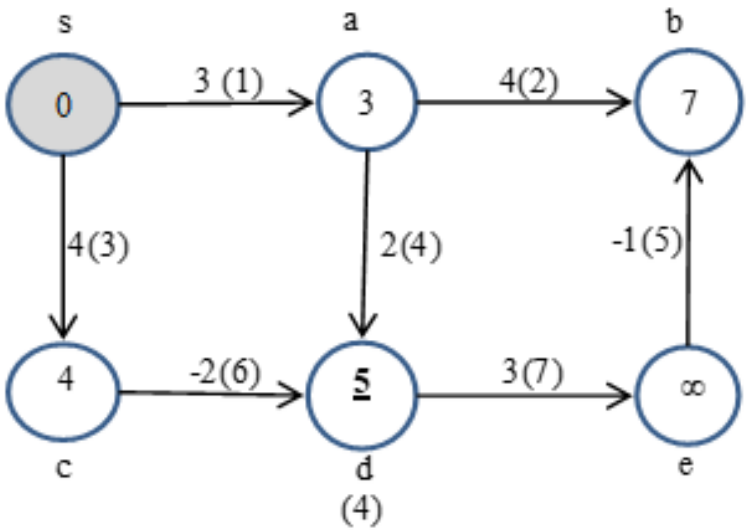


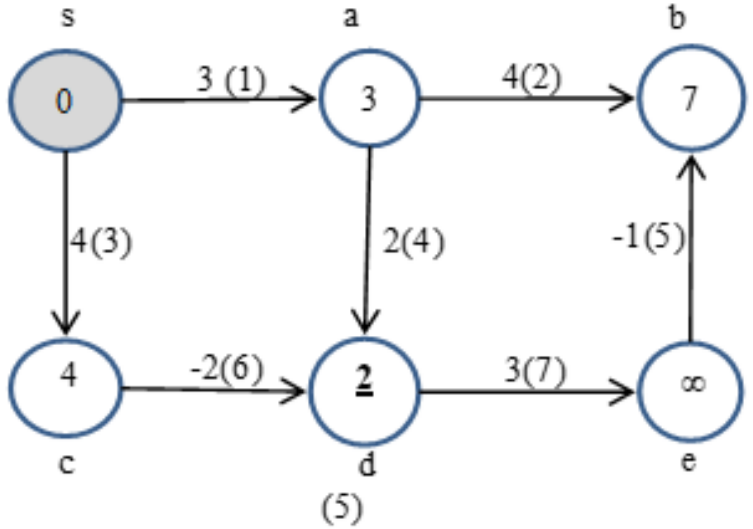


Ví dụ: minh họa hoạt động của giải thuật Bellman-Ford (tiếp)

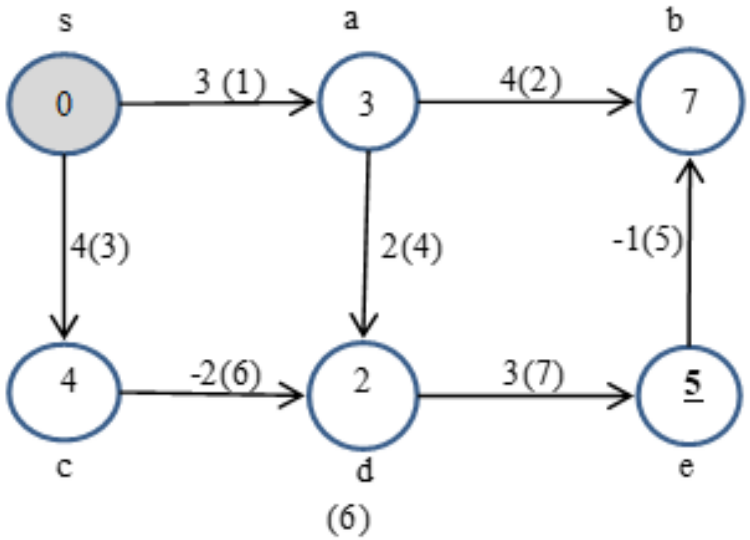


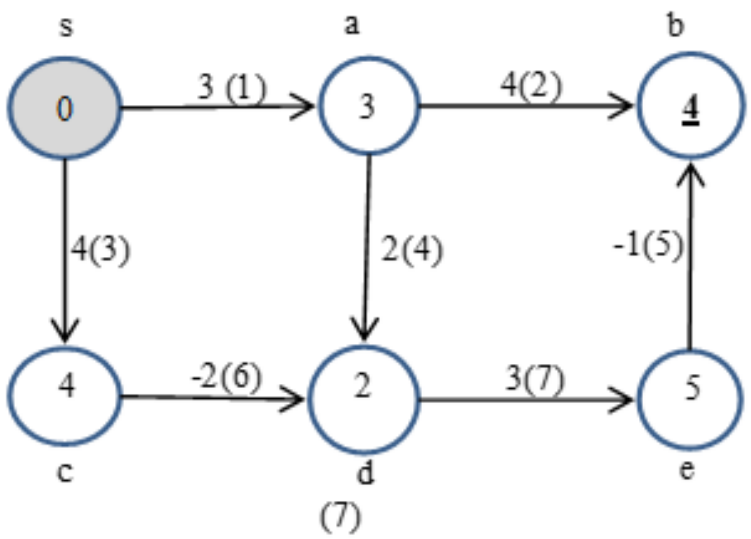






Ví dụ: minh họa hoạt động của giải thuật Bellman-Ford (tiếp)





Tìm đường đi ngắn nhất giữa mọi cặp nút

Phát biểu bài toán

Cho đồ thị có trọng số G(V, E) với $V = v_0, v_1, ..., v_{M-1}$ (M đỉnh) và $E = e_0, e_1, ..., e_{N-1}$ (N cạnh). Tìm đường đi ngắn nhất giữa mọi cặp đỉnh của đồ thị.

Giới thiệu giải thuật

Một ý tưởng đơn giản là có thể mở rộng các giải thuật tìm đường đi ngắn nhất giữa một nút và các nút còn lại (Bellman-Ford và Dijkstra) cho mọi nút. Tuy nhiên, giải thuật này sẽ có độ phức tạp rất cao.
Một chiến lược tốt hơn được sử dụng là giải thuật Floyd, cho phép tìm đường đi ngắn nhất giữa mọi cặp nút của đồ thị mà có thể chứa trọng số âm.

Giải thuật Floyd

Ý tưởng giải thuật

Giải thuật này có ý tưởng khá giống với giải thuật Bellman-Ford bằng cách cập nhật dần độ dài đường đi ngắn nhất bằng chiến lược vét cạn các cạnh của đồ thị. Tuy nhiên, giải thuật này không cần có điểm nguồn vì nó sẽ tìm luôn đường đi ngắn nhất giữa các cặp đỉnh.

Giải thuật Floyd

Cài đặt giải thuật

- Đầu vào: Cho đồ thị có trọng số G(V, E) với $V = v_0, v_1, ..., v_{M-1}$ (M đỉnh) và $E = e_0, e_1, ..., e_{N-1}$ (N cạnh).
- Đầu ra: Đường đi ngắn nhất giữa cặp đỉnh, được lưu trong 2 mảng D[M,M] và P[M,M], trong đó:
  - D[i,j] là độ dài đường đi ngắn nhất giữa 2 đỉnh v_i và v_j;
  - P[i,j] = k, tức là đỉnh v_k nằm trên đường đi ngắn nhất giữa 2 đỉnh v_i và v_j. Mảng P có vai trò như mảng Prev trong 2 giải thuật Bellman-Ford và Dijkstra nhằm tìm lại chính đường đi ngắn nhất giữa các cặp đỉnh (chứ không chỉ là các độ dài đó);

Giải thuật Floyd

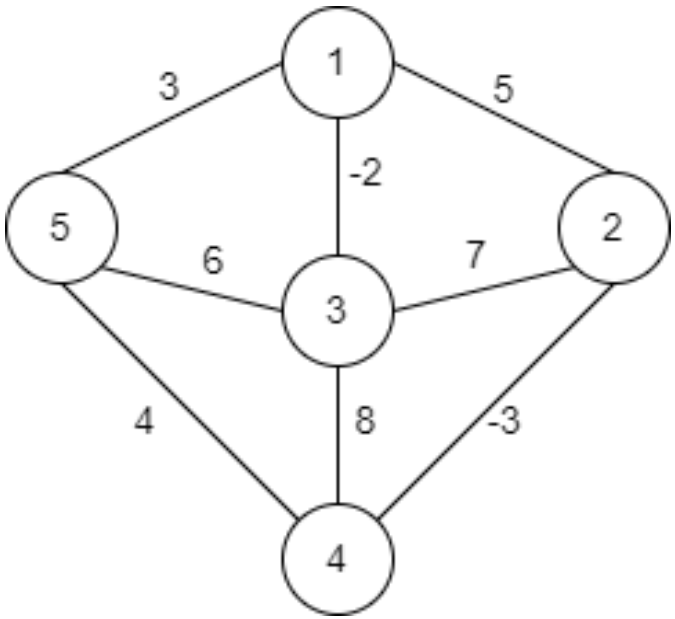
Cài đặt giải thuật

- Nội dung:

```
Khởi tạo:
  P = -1;
  D = ∞;
  for (i=0;i<M;i++)
    for (j=0;j<M;j++)
      if (i,j) ∈ E D[i,j] = |i,j|
Cập nhật D và P:
  for (i=0;i<M;i++)
    for (j=0;j<M;j++)
      for (k=0;k<M;k++)
        if D[i,j] > D[i,k] + D[k,j]{
          D[i,j] = D[i,k] + D[k,j]
          P[i,j] = k
        }
```

Ví dụ: minh họa hoạt động của giải thuật Floyd

Giả sử cho đồ thị:



Tìm chu trình của đồ thị

Bài toán tìm chu trình của đồ thị có thể chia làm một số loại bài toán con như sau:

1. Kiểm tra sự tồn tại của chu trình trong đồ thị;
2. Kiểm tra sự tồn tại của chu trình xuất phát từ một đỉnh/cạnh cho trước;
3. Tìm tất cả các chu trình trong đồ thị.

Quá trình kiểm tra chu trình thực ra là quá trình duyệt đồ thị. Mà trong các phương pháp duyệt đồ thị, chỉ có duyệt theo chiều sâu thì thứ tự các nút được duyệt đi theo các đường dẫn, qua đó ta mới có thể kiểm tra xem đường dẫn đó có khép kín không, tức là sự tồn tại của chu trình hay không. Do đó, duyệt theo chiều sâu thường được sử dụng trong các bài toán tìm chu trình.

Tìm chu trình của đồ thị

Trong phần này, chúng ta sẽ xét giải thuật cho bài toán 2, là tìm sự tồn tại của chu trình từ một đỉnh cho trước. Giải thuật cho bài toán này vừa là cơ sở cho hai bài toán còn lại, vừa được sử dụng trong nhiều ứng dụng, trong đó có bài toán tìm cây khung có giá cực tiểu mà sẽ được trình bầy ở phần sau.

Giải thuật tìm chu trình của đồ thị

Ý tưởng giải thuật

- Đầu vào: Cho đồ thị có trọng số G(V, E) với $V = v_0, v_1, ..., v_{M-1}$ (M đỉnh) và $E = e_0, e_1, ..., e_{N-1}$ (N cạnh), và một đỉnh ban đầu $s \in V$.
- Đầu ra: trả lời G có tồn tại một chu trình mà chứa s hay không?

Giải thuật tìm chu trình của đồ thị

Ý tưởng giải thuật

- Nội dung: áp dụng chiến lược duyệt theo chiều sâu, bắt đầu từ s. Giả sử quá trình kiểm tra đang đến nút c. Gọi tập các đỉnh hiện tại kề với c là A_c. Ta có giải thuật đệ quy như sau:
  - Điểm dừng: có 2 khả năng xảy ra:
    - Nếu A_c rỗng thì kết luận không có chu trình;
    - Nếu tồn tại $v_i \in A_c$ và $v_i = s$ thì kết luận có tồn tại chu trình.
  - Trường hợp đệ quy: là khi $A_c \neq \emptyset$, và $\forall v_i \in A_c$ đều có $v_i \neq s$. Khi đó, lặp lại giải thuật cho từng đỉnh $v_i \in A_c$.

Giải thuật tìm chu trình của đồ thị

Thủ tục

```
bool CycleDetection(G, s, c){
    Ac = Tập đỉnh kề của c;
    if (Ac là rỗng) return false;
    else
        if ∃ v_i ∈ Ac and v_i = s return true;
        else
            for each v_i ∈ Ac
                if (CycDetection(G, s, v_i) == true) return true;
    return false;
}
```

Tìm cây khung

- Khái niệm cây khung: cây khung của một đồ thị G(V, E), (với V gồm M nút, và E gồm N cạnh) là một đồ thị con liên thông T bao gồm M nút và không có chu trình (tức là nó sẽ có $M - 1$ cạnh). Như vậy, khi $N \geq M - 1$ thì G có thể có một hoặc nhiều cây khung.
- Bài toán: Bài toán tìm cây khung có giá cực tiểu (sau này được gọi ngắn gọn là cây khung cực tiểu) áp dụng cho đồ thị G là đồ thị liên thông, vô hướng và có trọng số. Khi đó G sẽ có ít nhất một cây khung. Yêu cầu của bài toán là tìm ra cây khung Tm có tổng giá trị các trọng số trên các cạnh là nhỏ nhất trong số tất cả các cây khung của G.

Giải thuật tìm cây khung

- Đối với đồ thị có trọng số G có M nút và N cạnh, thì cây khung sẽ có đúng $M - 1$ cạnh.
- Việc tìm cây khung có giá cực tiểu thực ra là bài toán cần chọn ra $M - 1$ giá trị (ứng với $M - 1$ trọng số của $M - 1$ cạnh) từ N giá trị sao cho tổng của $M - 1$ giá trị này là nhỏ nhất và không tạo ra chu trình.
- Do tổng trọng số tất cả N cạnh của đồ thị là không đổi, nên bài toán trên thực ra cũng tương đương với bài toán làm thể nào để loại bỏ đúng $N - M + 1$ cạnh, sao cho tổng các cạnh bị loại bỏ có giá trị lớn nhất và các cạnh còn lại không tạo ra chu trình.
- Việc giải hai bài toán tương đương ở trên dẫn đến hai giải thuật khác nhau: giải thuật Kruskal và giải thuật Dijkstra.

Giải thuật Kruskal

Ý tưởng giải thuật

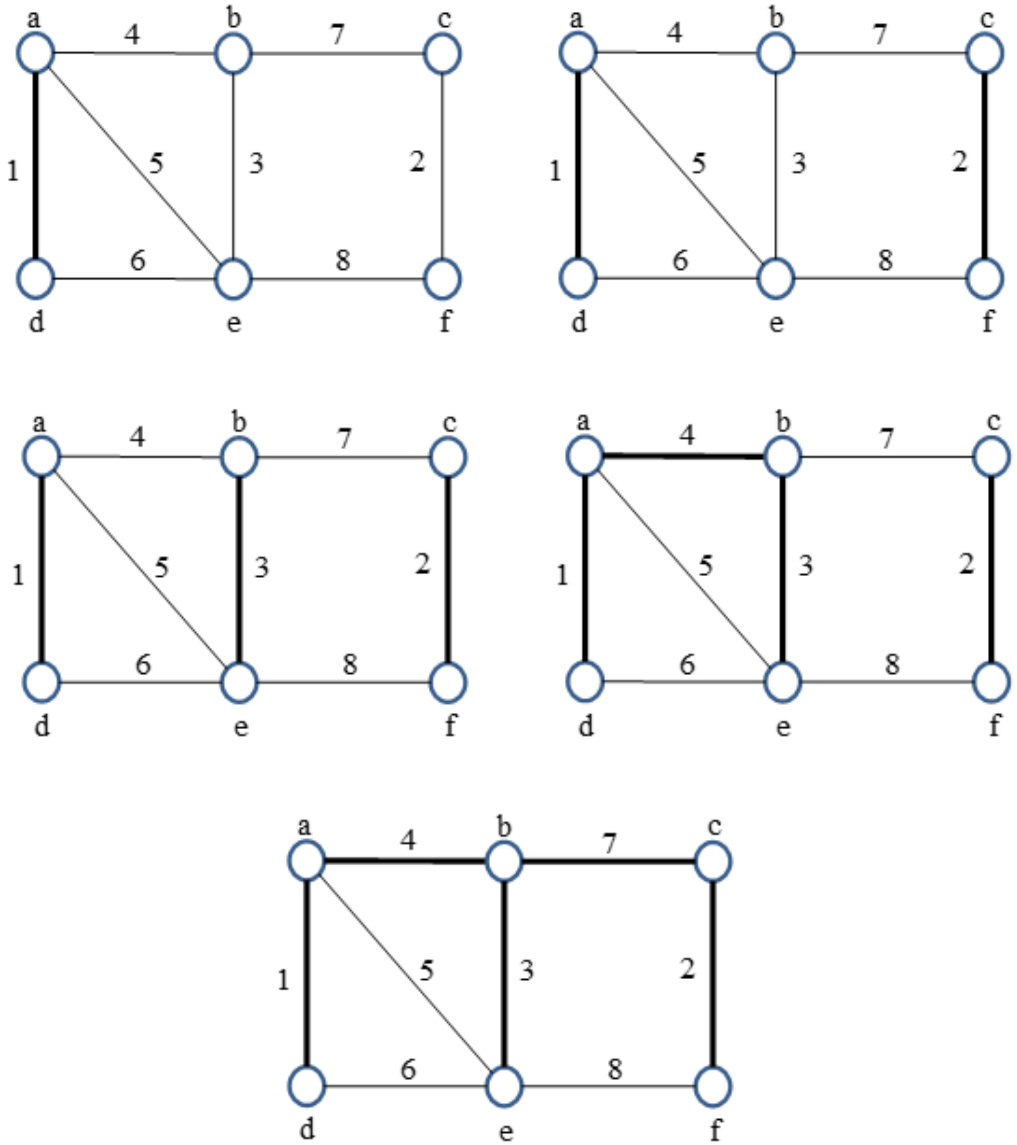
Xuất phát từ cây khung rỗng (chỉ có các đỉnh mà chưa có cạnh) bổ sung dần các cạnh có giá trị nhỏ nhất có thể vào (không tạo thành chu trình), cho đến khi tạo thành cây khung đầy đủ (đúng M-1 cạnh).
Để rút ngắn thời gian tìm kiếm các cạnh phù hợp, đầu tiên các cạnh của đồ thị sẽ được sắp xếp theo giá trị trọng số theo thứ tự tăng dần. Sau đó, quá trình chọn các cạnh sẽ lần lượt từ nhỏ đến lớn, cạnh được chọn cũng phải thỏa mãn điều kiện là không tạo ra chu trình với các cạnh đã được chọn trước đó.

Giải thuật Kruskal

Thủ tục

```
T = ϕ;
S = tập các cạnh được sắp xếp của E;
{ giả sử: S = (e0, e1, ..., eN-1) }
i = 0;
Chừng nào (i < N) và (|T|<M-1)
      Nếu ei không tạo ra chu trình thì: T = T ∪ ei;
      i++;
```

Ví dụ minh họa Giải thuật Kruskal



Giải thuật Dijkstra

Ý tưởng giải thuật

- Sử dụng chiến lược thứ hai để tìm cây khung cực tiểu. Đó là, bắt đầu từ đồ thị ban đầu, tiến hành loại bỏ dần các cạnh có giá trị lớn nhất có thể (những cạnh mà gây ra chu trình), cho đến khi còn lại cây khung - là cây khung cực tiểu.
- Thông thường, giống như ở giải thuật Kruskal, để chọn được các cạnh có giá trị lớn nhất, thì các cạnh của đồ thị cần được sắp xếp, mà như chúng ta đã biết giải thuật sắp xếp tốt nhất thì độ phức tạp cũng là N.Log(N), với N là số cạnh của đồ thị.

Giải thuật Dijkstra

Ý tưởng giải thuật (tiếp)

- Để tránh thao tác sắp xếp, giải thuật Dijkstra đưa ra một cải tiến quan trọng mà kế thừa từ giải thuật Kruskal. Đó là cũng xuất phát từ cây khung rỗng ban đầu, tiến hành bổ sung lần lượt từng cạnh của đồ thị theo thứ tự bất kì (không cần sắp xếp), cho đến khi đủ thành cây khung.
- Đồng thời, mỗi lần bổ sung một cạnh, cần kiểm tra xem có tạo thành chu trình không. Nếu có, thì chu trình phải chứa cạnh vừa được bổ sung. Khi đó, tìm và loại bỏ cạnh có trọng số lớn nhất khỏi chu trình .

Giải thuật Dijkstra

Thủ tục

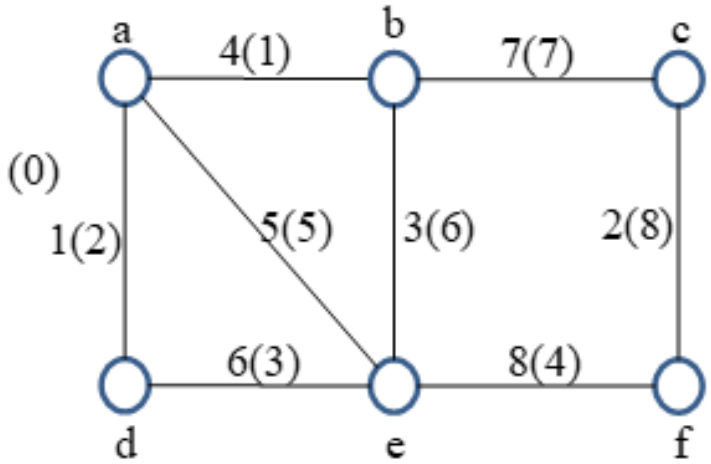
- Đầu vào: Cho đồ thị có trọng số G(V, E) có M đỉnh $V = v_0, v_1, ..., v_{M-1}$, và N cạnh $E = e_0, e_1, ..., e_{N-1}$;
- Đầu ra: Cây khung cực tiểu
- Nội dung:

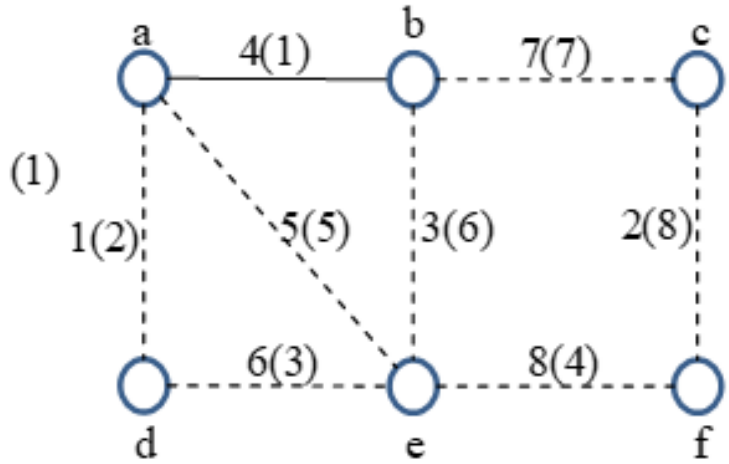
```
T = ϕ;
for (i = 0; i < N; i++)
    T = T ∪ e_i;
    Nếu e_i tạo ra chu trình c_i
        Tìm e_m ∈ c_i và e_m là cạnh max trong c_i;
        Loại bỏ e_m;
```

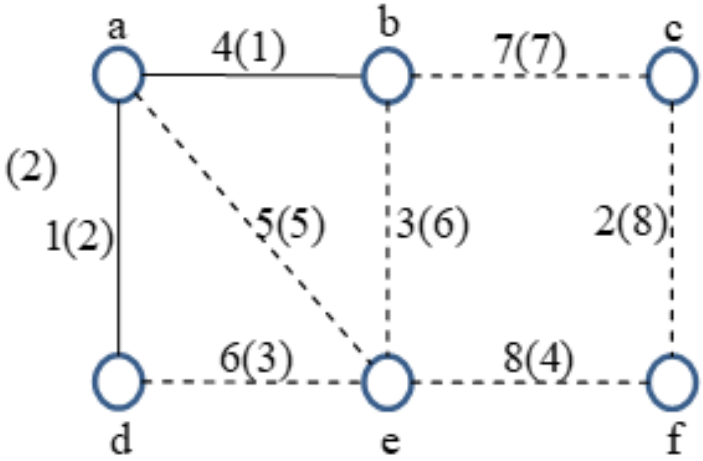
Ví dụ minh họa giải thuật Dijkstra

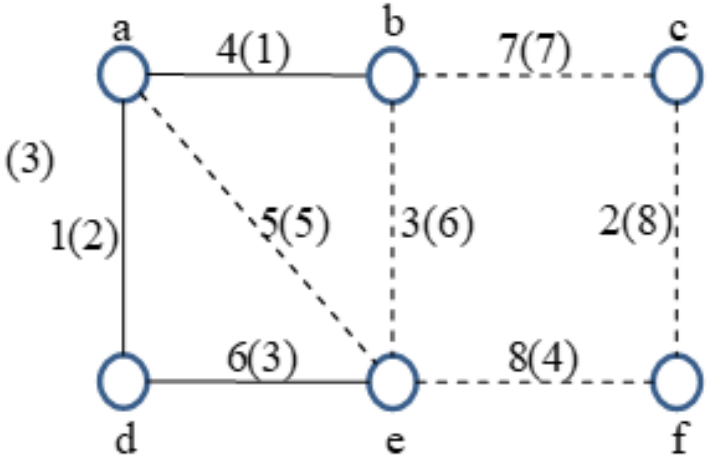
- Trong hình, mỗi cạnh ngoài phần trọng số, còn có thêm thông tin về thứ tự được chọn (phần để trong ngoặc) để đưa vào cây khung.
- Các cạnh với nét liền là các cạnh đã được chọn, cạnh với nét đứt thường là chưa được chọn, cạnh với nét đứt đậm là cạnh bị loại ra do là cạnh lớn nhất trong một chu trình.
- Hình đầu tiên (0) là đồ thị ban đầu.
- Hình cuối cùng (9) là cây khung cực tiểu tìm được.

Ví dụ minh họa giải thuật Dijkstra (tiếp)

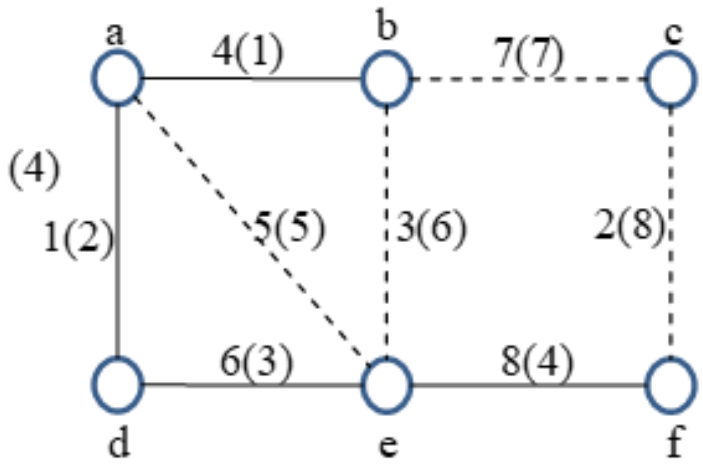


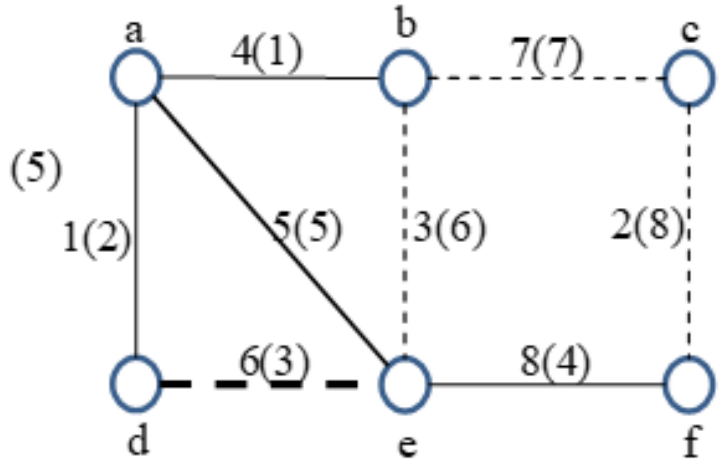


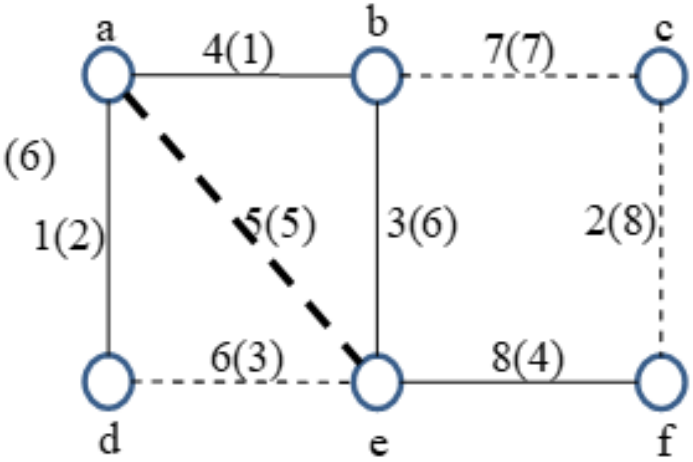


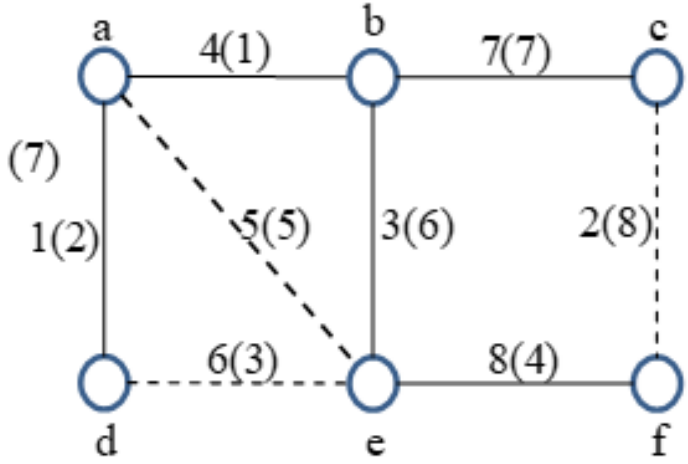


Ví dụ minh họa giải thuật Dijkstra (tiếp)

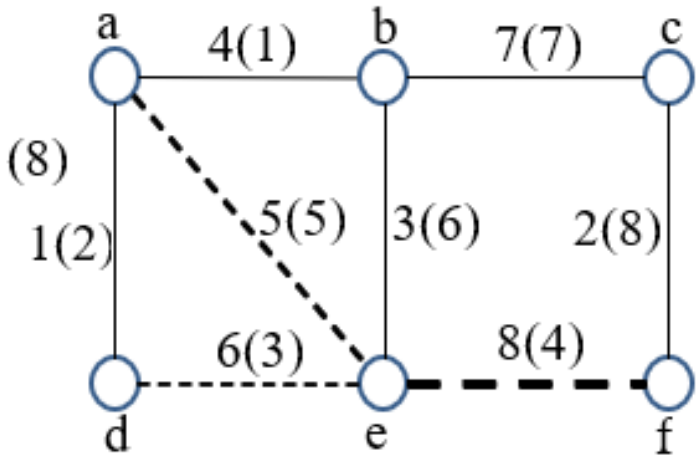


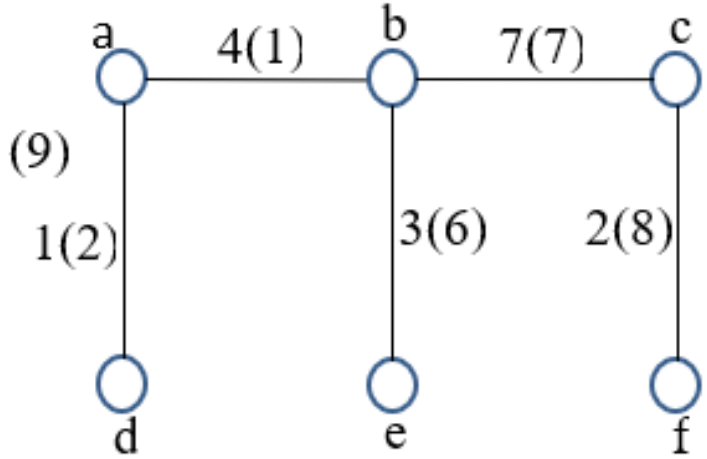






Ví dụ minh họa giải thuật Dijkstra (tiếp)





Tham khảo

1. "Data Structures using C, 2nd Edition "; Reema Thareja; Oxford University Press, 2014.
2. "Ngôn ngữ lập trình C và Cấu trúc dữ liệu"; Nguyễn Thanh Bình, Nguyễn Hoài Giang; NXB Giáo Dục Việt Nam, 2017.