Chương 3

Các bài toán về đường đi

Trong các ứng dụng thực tế, ta cần tìm đường đi (nếu có) giữa hai đỉnh của đồ thị. Đặc biệt, bài toán tìm đường đi ngắn nhất giữa hai đỉnh của một đồ thị có ý nghĩa to lớn. Có thể dẫn về bài toán như vậy từ nhiều bài toán thực tế. Ví dụ, bài toán tìm hành trình tiết kiệm nhất (theo tiêu chuẩn khoảng cách, thời gian hoặc chi phí) trên một bản đồ giao thông; bài toán chọn phương pháp tiết kiệm nhất để đưa một hệ động lực từ trạng thái này sang trạng thái khác v.v... Hiện nay có rất nhiều phương pháp dựa trên lý thuyết đồ thị tỏ ra là các phương pháp có hiệu quả nhất.

Chương này trình bày các thuật toán tìm đường đi ngắn nhất trên đồ thị có trọng số.

3.1 Đường đi giữa hai đỉnh

3.1.1 Đường đi giữa hai đỉnh

Trong nhiều trường hợp, chúng ta cần trả lời câu hỏi: *Tồn tại đường đi μ từ đỉnh s đến đỉnh t của đồ thị có hướng $G := (V, E)$? Nếu có, hãy chỉ ra cách đi của đường đi μ.*

Lời giải của bài toán này khá đơn giản: chúng ta chỉ cần áp dụng thuật toán tìm kiếm theo chiều rộng (hoặc chiều sâu) trên đồ thị có hướng G như sau. Gán mỗi đỉnh của G một chỉ số. Bằng phương pháp lặp, dần dần ta sẽ cho mỗi đỉnh v một chỉ số nào đó bằng độ dài đường đi ngắn nhất (số cung ít nhất) từ s tới v. Đánh dấu đỉnh s bằng chỉ số 0. Nếu các đỉnh được đánh dấu bằng chỉ số m lập thành một tập hợp $P(m)$ đã biết, thì ta đánh dấu chỉ số $(m + 1)$ cho mọi đỉnh của tập hợp:

$$P(m+1) := \{v_j \text{ chưa được đánh dấu} \mid \text{tồn tại } v_i \in P(m) \text{ với } (v_i, v_j) \in E\}.$$

Thuật toán dừng khi không thể đánh dấu được nữa. Có hai trường hợp xảy ra:

1. Đỉnh t được đánh dấu, chẳng hạn $t \in P(m)$ với m nào đó, thì ta xét các đỉnh $v_1, v_2, ...,$ sao cho
$$v_1 \in P(m-1), v_2 \in P(m-2), \ldots, v_m \in P(0).$$
Khi đó $\mu := \{s = v_m, v_{m-1}, ..., v_1, t\}$ là đường đi phải tìm.

2. Đỉnh t không được đánh dấu. Trong trường hợp này, ta kết luận không tồn tại đường đi từ s đến t.

Theo cách xây dựng của thuật toán, dễ dàng chứng minh rằng

Mệnh đề 3.1.1 *Nếu đồ thị được xác định bởi dãy liên tiếp các đỉnh, thì thuật toán có thời gian $O(m)$.*

3.1.2 Đồ thị liên thông mạnh

Nhắc lại là đồ thị có hướng G gọi là *liên thông mạnh* nếu hai đỉnh s và t tùy ý của G luôn luôn tồn tại một đường đi từ s đến t. Hiển nhiên rằng

Định lý 3.1.2 *Cho $G = (V, E)$ là đồ thị có hướng, và $v \in V$. Khi đó G liên thông mạnh nếu và chỉ nếu mọi cặp đỉnh $a, b \in V$, tồn tại một đường đi từ a đến v và một đường đi từ v đến b.*

Dựa trên thuật toán tìm kiếm theo chiều sâu, ta có thể mô tả cách xác định một đồ thị có hướng có liên thông mạnh hay không thông qua định lý sau:

Định lý 3.1.3 *Cho $G = (V, E)$ là đồ thị có hướng, và $v \in V$. Ký hiệu $G' := (V, E')$ là đồ thị có hướng nhận được từ G bằng cách đảo hướng mỗi cung trong E. Khi đó G là liên thông mạnh nếu và chỉ nếu thuật toán tìm kiếm theo chiều sâu trên đồ thị có hướng G, khởi đầu từ v, đạt được mọi đỉnh của G và thuật toán tìm kiếm theo chiều sâu trên đồ thị có hướng G', khởi đầu từ v, đạt được mọi đỉnh của G'.*

Định nghĩa 3.1.4 Đồ thị vô hướng G gọi là *được định hướng mạnh* nếu có thể định hướng trên các cạnh của G sao cho đồ thị có hướng tương ứng nhận được là liên thông mạnh.

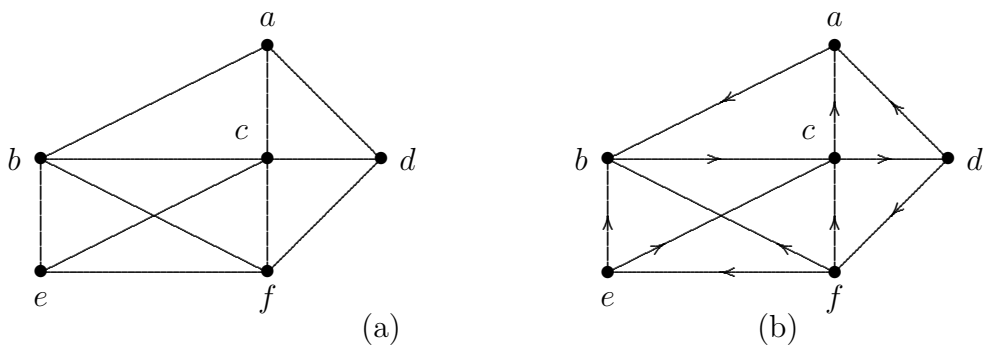
Định lý sau cho chúng ta một đặc trưng của đồ thị vô hướng được định hướng mạnh. Ta nói *cầu* trong đồ thị vô hướng liên thông là một cạnh mà bỏ đi thì đồ thị sẽ mất tính liên thông.

Định lý 3.1.5 *Đồ thị vô hướng G được định hướng mạnh nếu và chỉ nếu nó liên thông và không có cầu.*

Chứng minh. Bài tập. ◁

Dựa trên thuật toán tìm kiếm theo chiều sâu, ta có thể định hướng các cạnh của đồ thị vô hướng được định hướng mạnh như sau. Lấy một đỉnh bất kỳ trong đồ thị vô hướng G làm đỉnh khởi đầu và thực hiện thuật toán tìm kiếm theo chiều sâu. Vì đồ thị vô hướng là liên thông, kết quả của việc tìm kiếm này cho ta một *cây bao trùm*[1] $T = (V_n, E_n)$, trong đó $V_n = \{v_1, v_2, \ldots, v_n\}$ là các đỉnh của G. Nếu $e = (v_i, v_j)$ là một cạnh trong cây bao trùm T, ta định hướng nó từ đỉnh có chỉ số nhỏ hơn đến đỉnh có chỉ số lớn hơn. Tức là nếu $i < j$, định hướng cạnh e từ v_i đến v_j, và nếu $j < i$ thì định hướng cạnh e từ v_j đến v_i. Nếu cạnh $e = (v_i, v_j)$ của G không thuộc cây T, thì ta định hướng cạnh này từ đỉnh có chỉ số lớn hơn đến đỉnh có chỉ số nhỏ hơn. Tức là nếu $i > j$, định hướng cạnh e từ v_i đến v_j, và nếu $j > i$ thì định hướng cạnh e từ v_j đến v_i.

Hình 3.1 minh họa đồ thị vô hướng và cách định hướng nó.



Hình 3.1: (a) Đồ thị vô hướng G. (b) Đồ thị G được định hướng.

[1]Khái niệm này sẽ được trình bày trong **??**.

3.2 Đường đi ngắn nhất giữa hai đỉnh

Cho đồ thị có hướng $G = (V, E)$ mà các cung của nó được gán các trọng lượng cho bởi ma trận vuông cấp $n : W = (w_{ij} = (w(v_i, v_j))$. Bài toán đặt ra là tìm đường đi μ từ một đỉnh xuất phát $s \in V$ đến một đỉnh cuối là $t \in V$ sao cho tổng các trọng lượng trên đường đi μ :

$$\sum_{e_k \in \mu} w(e_k)$$

là nhỏ nhất.

Các phần tử $w_{ij}, i, j = 1, ..., n$, của ma trận trọng lượng có thể dương, âm hoặc bằng không. Một điều kiện duy nhất đặt ra là đồ thị có hướng G không chứa các mạch μ với tổng trọng lượng trên mạch μ âm. Vì rằng, nếu có một mạch μ như vậy và v là một đỉnh nào đó của nó, thì xuất phát từ nút s ta đi đến đỉnh v, và sau đó đi vòng quanh mạch μ một số đủ lớn lần rồi mới đến t ta sẽ thu được một đường đi có trọng lượng đủ nhỏ. Vì vậy, trong trường hợp này, đường đi ngắn nhất là không tồn tại.

Nhận xét rằng, nếu ma trận trọng lượng W thoả mãn

$$w_{ij} := \begin{cases} 1 & \text{nếu} \quad (v_i, v_j) \in E, \\ 0 & \text{nếu ngược lại,} \end{cases}$$

thì đường đi ngắn nhất từ s đến t được xác định bằng thuật toán tìm kiếm theo chiều rộng như đã trình bày trong phần trước.

Trước tiên chúng ta xét thuật toán Dijkstra, đơn giản và rất hiệu quả, để giải bài toán đặt ra trong trường hợp ma trận trọng lượng W có các phần tử không âm. Sau đó phát triển nó để giải quyết bài toán trong trường hợp tổng quát.

3.2.1 Trường hợp ma trận trọng lượng không âm

Thuật toán Dijkstra [20] tìm đường đi ngắn nhất từ đỉnh s đến đỉnh t trong đồ thị có hướng có trọng lượng không âm. Phương pháp này dựa trên việc gán các *nhãn tạm thời* cho các đỉnh: *Nhãn* của đỉnh v_i, ký hiệu $L(v_i)$, là một cận trên của độ dài đường đi từ s đến v_i. Các nhãn này sau đó tiếp tục được giảm bớt bởi một thủ tục lặp và tại mỗi bước lặp có đúng một nhãn tạm thời trở thành *nhãn cố định*. Khi đỉnh v_i được gán nhãn cố định thì không thể giảm $L(v_i)$; số này chính là độ dài đường đi ngắn nhất từ s đến v_i.

Thuật toán Dijkstra ($w_{ij} \geq 0$)

1. [Khởi tạo] Đặt
$$L(v_i) := \begin{cases} 0 & \text{nếu } v_i = s, \\ +\infty & \text{nếu ngược lại.} \end{cases}$$
với mọi $v_i \in V$. Đánh dấu đỉnh s có nhãn cố định, các đỉnh khác có nhãn tạm thời. Đặt $c = s$.

2. [Cập nhật] Với mọi $v_i \in \Gamma(c)$ sao cho v_i có nhãn tạm thời đặt
$$L(v_i) := \min\{L(v_i), L(c) + w(c, v_i)\};$$

3. Tìm đỉnh v_{i^*} có nhãn tạm thời sao cho
$$L(v_{i^*}) := \min\{L(v_i) < +\infty \mid v_i \text{ có nhãn tạm thời}\}.$$

4. Đánh dấu đỉnh v_{i^*} có nhãn cố định và đặt $c := v_{i^*}$.

5. (a) *(Nếu tìm đường đi từ s đến t)* Nếu $v_i = t$ thì thuật toán dừng, $L(t)$ là đường đi ngắn nhất từ s đến t; ngược lại, chuyển sang Bước 2.

   (b) *(Nếu tìm tất cả các đường đi xuất phát từ s)* Nếu không thể gán nhãn cố định cho đỉnh v_{i^*} trong Bước 4 thì thuật toán dừng; giá trị $L(v_i)$ của đỉnh v_i có nhãn cố định cho ta độ dài đường đi ngắn nhất từ s đến v_i. Ngược lại chuyển sang Bước 2.

Định lý 3.2.1 *Thuật toán Dijkstra cho ta đường đi ngắn nhất từ s đến t (nếu có).*

Chứng minh. Trước hết chú ý rằng các đỉnh không được gán nhãn cố định sẽ không tồn tại đường đi từ s đến nó (những đỉnh này có nhãn L bằng ∞).

Giả sử $L(v_i)$ của các đỉnh có nhãn cố định ở bước nào đó là độ dài đường đi ngắn nhất từ s đến v_i. Ký hiệu S_1 là tập tất cả các đỉnh này và S_2 là tập các đỉnh có nhãn tạm thời. Cuối Bước 2 của mỗi lần lặp, nhãn tạm thời $L(v_i)$ là độ dài đường đi ngắn nhất μ_i từ s đến v_i qua các đỉnh trong tập S_1. (Vì trong mỗi lần lặp chỉ có một đỉnh được đưa vào S_1 nên cập nhật lại $L(v_i)$ chỉ xảy ra trong Bước 2).

Xét đường đi ngắn nhất từ s đến v_{i^*} không hoàn toàn đi qua các đỉnh của S_1 mà chứa ít nhất một đỉnh thuộc S_2 và giả sử $v_j \in S_2$ là đỉnh đầu tiên như vậy trên đường đi này. Do w_{ij} không âm, đoạn đường đi từ v_j đến v_{i^*} phải có độ dài Δ không âm sao cho $L(v_j) < L(v_{i^*}) - \Delta < L(v_{i^*})$. Tuy nhiên điều này mâu thuẫn với khẳng định rằng $L(v_{i^*})$ có nhãn tạm thời nhỏ nhất, và do đó các đỉnh trên đường đi ngắn nhất đến v_{i^*} thuộc S_1 và do đó $L(v_{i^*})$ là độ dài của đường đi này.

Vì S_1 khởi tạo là $\{s\}$ và trong mỗi bước lặp, đỉnh v_{i^*} được thêm vào S_1, nên bằng qui nạp, $L(v_i)$ là độ dài đường đi ngắn nhất từ s đến v_i với mọi đỉnh $v_i \in S_1$. Do đó thuật toán cho lời giải tối ưu. ◁

Mệnh đề 3.2.2 *Thuật toán Dijkstra đòi hỏi thời gian $O(n^2)$. Nếu đồ thị thưa và được xác định bởi dãy liên tiếp các đỉnh, thì thời gian cực đại của thuật toán là $O(m \log n)$.*

Chứng minh. Trong trường hợp đồ thị liên thông mạnh đầy đủ n đỉnh và cần tìm đường đi ngắn nhất từ s đến mọi đỉnh khác, thuật toán cần $n(n-1)/2$ phép cộng và so sánh trong Bước 2 và $n(n-1)/2$ phép so sánh khác trong Bước 3. Ngoài ra, các Bước 2 và 3 cần xác định các đỉnh được gán nhãn tạm thời nên cần thêm $n(n-1)/2$ phép so sánh. Các số này cũng là cận trên cho số các phép toán cần thiết để tìm đường đi ngắn nhất từ s đến t, và thật vậy, các giá trị này đạt được khi t là đỉnh cuối cùng được gán nhãn cố định. ◁

Khi Thuật toán Dijkstra kết thúc, các đường đi ngắn nhất được xác định bằng đệ qui theo Phương trình (3.1) dưới đây. Do đó nếu v_i' là đỉnh trước đỉnh v_i trong đường đi ngắn nhất từ s đến v_i thì với mọi đỉnh v_i cho trước, đỉnh v_i' có thể lấy là đỉnh mà

$$L(v_i) = L(v_i') + w(v_i', v_i). \tag{3.1}$$

Nếu tồn tại duy nhất đường đi ngắn nhất từ s đến v_i thì các cung (v_i', v_i) trên đường đi ngắn nhất tạo thành một cây có hướng (xem Chương 4) với gốc là đỉnh s. Nếu có nhiều hơn một đường đi ngắn nhất từ s đến bất kỳ một đỉnh khác, thì Phương trình 3.1 sẽ được thoả mãn bởi hơn một đỉnh khác v_i' với v_i nào đó.

Thủ tục sau minh họa thuật toán Dijkstra. Trong thủ tục này, mảng $Mark[]$ được sử dụng để đánh dấu các đỉnh có nhãn tạm thời hay cố định: $Mark[i]$ = FALSE nếu đỉnh v_i có nhãn tạm thời; ngược lại bằng TRUE. Giá trị $Label[i]$ tương ứng nhãn $L_(v_i)$ và $Pred[i]$, sau khi kết thúc thuật toán, là đỉnh liền trước đỉnh v_i trong đường đi ngắn nhất từ s đến v_i. Nếu tồn tại đường đi ngắn nhất từ s đến t, thì thủ tục PathTwoVertex() sẽ in ra thông tin của đường đi này.

```
void Dijkstra(byte Start, byte Terminal)
{
  byte i, Current;
  AdjPointer Tempt;
  Path Pred;
  int Label[MAXVERTICES], NewLabel, Min;
  Boolean Mark[MAXVERTICES];
```



```
for(i = 1; i <= NumVertices; i++)
{
  Mark[i] = FALSE;
  Pred[i] = 0;
  Label[i] = +Infty;
}

Mark[Start] = TRUE;
Pred[Start] = 0;
Label[Start] = 0;
Current = Start;

while (Mark[Terminal] == FALSE)
{
  Tempt = V_out[Current]->Next;
  while (Tempt != NULL)
  {
    if (Mark[Tempt->Vertex] == FALSE)
    {
      NewLabel = Label[Current] + Tempt->Length;
      if (NewLabel < Label[Tempt->Vertex])
      {
        Label[Tempt->Vertex] = NewLabel;
        Pred[Tempt->Vertex] = Current;
      }
    }
    Tempt = Tempt->Next;
  }

  Min = +Infty;
  for (i = 1; i <= NumVertices; i++)
  if ((Mark[i] == FALSE) && (Label[i] < Min))
  {
    Min = Label[i];
    Current = i;
  }

  if (Min == +Infty)
  {
    printf("Khong ton tai duong di tu %d", Start);
    printf(" den %d", Terminal);
```



```
      return;
    }
    Mark[Current] = TRUE;
  }

  printf("Ton tai duong di tu %d", Start);
  printf(" den %d", Terminal);
  printf("\nDuong di qua cac dinh:");
  printf("\n % d " , Start);
  PathTwoVertex(Pred, Start, Terminal);
  printf("\nDo dai la: %d ", Label[Terminal]);
}
```

3.2.2 Trường hợp ma trận trọng lượng tuỳ ý

Thuật toán Dijkstra chỉ áp dụng trong trường hợp ma trận trọng lượng W không âm. Tuy nhiên, nhiều bài toán thực tế, W là ma trận chi phí, cho nên những cung mang lại lợi nhuận phải có chi phí âm. Trong trường hợp này, phương pháp cho dưới đây tìm đường đi ngắn nhất từ s đến tất cả các đỉnh khác. Đây cũng là phương pháp lặp và dựa trên cách đánh nhãn, trong đó cuối bước lặp thứ k các nhãn biểu diễn giá trị độ dài của các đường đi ngắn nhất (từ s đến tất cả các đỉnh khác) có số cung không vượt quá $(k+1)$. Thuật toán này đưa ra lần đầu tiên bởi Ford [26] vào giữa năm 1950. Sau đó được Moore [45] và Bellman [3] cải tiến như sau.

Thuật toán Ford, Moore, Bellman

Ký hiệu $L^k(v_i)$ là nhãn của đỉnh v_i ở cuối lần lặp thứ $(k+1)$.

1. [Khởi tạo] Đặt $S = \Gamma(s), k = 1$; và

$$L^1(v_i) := \begin{cases} 0 & \text{nếu} \quad v_i = s, \\ w(s, v_i) & \text{nếu} \quad v_i \neq s, v_i \in \Gamma(s), \\ +\infty & \text{nếu ngược lại.} \end{cases}$$

2. [Cập nhật nhãn] Với mỗi $v_i \in \Gamma(S), v_i \neq s$, thay đổi nhãn của nó theo quy tắc:

$$L^{k+1}(v_i) := \min[L^k(v_i), \min_{v_j \in T_i}\{L^k(v_j) + w(v_j, v_i)\}], \tag{3.2}$$

trong đó $T_i := \Gamma^{-1}(v_i) \cap S$. Tập S chứa tất cả các đỉnh v_i sao cho đường đi ngắn nhất (hiện hành) từ s đến v_i có số cung là k.

Tập T_i chứa tất cả các đỉnh v_j sao cho đường đi ngắn nhất từ s đến v_i có số cung là k (tức là $v_j \in S$) và kết thúc bằng cung (v_j, v_i). Chú ý rằng, nếu $v_i \notin \Gamma(S)$ thì đường đi ngắn nhất từ s đến v_i không thể có số cung là $(k+1)$ và ta không thay đổi nhãn của đỉnh này:

$$L^{k+1}(v_i) := L^k(v_i), \quad \text{với mọi } v_i \notin \Gamma(S).$$

3. [Kiểm tra kết thúc] (a) Nếu $k \leq n-1$ và $L^{k+1}(v_i) = L^k(v_i)$, với mọi $v_i \in V$, thì thuật toán dừng; nhãn của đỉnh v_i cho ta độ dài đường đi ngắn nhất từ s đến v_i.

   (b) Nếu $k < n-1$ và $L^{k+1}(v_i) \neq L^k(v_i)$, với v_i nào đó, thì chuyển sang Bước 4.

   (c) Nếu $k = n-1$ và $L^{k+1}(v_i) \neq L^k(v_i)$, với v_i nào đó, thì đồ thị G có mạch với độ dài âm. Thuật toán kết thúc.

4. [Cập nhật S] Đặt

$$S := \{v_i \mid L^{k+1}(v_i) \neq L^k(v_i)\}.$$

   (Tập S bây giờ chứa tất cả các đỉnh mà đường đi ngắn nhất từ s đến nó có số cung là $(k+1)$).

5. Thay k bởi $(k+1)$ và lặp lại Bước 2.

Khi thuật toán kết thúc và đồ thị không có mạch độ dài âm, chúng ta có thể tìm tất cả các đường đi (nếu tồn tại) ngắn nhất từ s đến tất cả các đỉnh khác. Hơn nữa các đường đi có thể nhận được trực tiếp nếu, trong phép cộng vào các nhãn $L^k(v_i)$ ở Phương trình 3.2, ta thêm một nhãn $P^k(v_i)$ lưu trữ thông tin mỗi đỉnh trong suốt quá trình tính toán, trong đó $P^k(v_i)$ là đỉnh kề trước đỉnh v_i trên đường đi ngắn nhất từ s đến v_i ở bước lặp thứ k. Ta có thể khởi tạo

$$P^1(v_i) := \begin{cases} s & \text{nếu} \quad v_i \in \Gamma(s), \\ 0 & \text{nếu ngược lại.} \end{cases}$$

Nhãn $P^k(v_i)$ được cập nhật theo Phương trình 3.2 trong Bước 2 sao cho $P^{k+1}(v_i) = P^k(v_i)$ nếu giá trị nhỏ nhất đạt được ở phần tử đầu tiên trong dấu ngoặc của Phương trình 3.2; hoặc $P^{k+1}(v_i) = v_j$ nếu ngược lại. Nếu ký hiệu $\mathbf{P}(v_i)$ là vector được xây dựng từ các nhãn P khi kết thúc thuật toán, thì đường đi ngắn nhất từ s đến v_i nhận được theo thứ tự ngược:

$$s, \ldots, P^3(v_i), P^2(v_i), P(v_i), v_i,$$

trong đó $P^2(v_i)$ có nghĩa là $P(P(v_i)), \ldots$.

Đoạn chương trình sau minh họa thuật toán đã trình bày.


```
void Ford_Moore_Bellman(byte Start)
{
  byte i, k, Terminal;
  AdjPointer Tempt1, Tempt2;
  Path OldPred, NewPred;
  int OldLabel[MAXVERTICES], NewLabel[MAXVERTICES], Min;
  Boolean Done, Mark[MAXVERTICES];

  for (i = 1; i <= NumVertices; i++)
  {
    Mark[i] = FALSE;
    OldPred[i] = 0;
    OldLabel[i] = +Infty;
  }

  OldLabel[Start] = 0;
  Tempt1 = V_out[Start]->Next;
  while (Tempt1 != NULL)
  {
    Mark[Tempt1->Vertex] = TRUE;
    if (Tempt1->Vertex != Start)
    {
      OldPred[Tempt1->Vertex] = Start;
      OldLabel[Tempt1->Vertex] = Tempt1->Length;
    }
    Tempt1 = Tempt1->Next;
  }

  for (i = 1; i <= NumVertices; i++)
  {
    NewPred[i] = OldPred[i];
    NewLabel[i] = OldLabel[i];
  }

  for (k = 1; k < NumVertices; k++)
  {
    printf("\n ");
    for (i = 1; i <= NumVertices; i++)
    {
      if (Mark[i] == TRUE)
      {
        Tempt1 = V_out[i]->Next;
```



```
    while (Tempt1 != NULL)
    {
      NewPred[Tempt1->Vertex] = Tempt1->Vertex;
      Min = OldLabel[Tempt1->Vertex];
      Tempt2 = V_in[Tempt1->Vertex]->Next;
      while (Tempt2 != NULL)
      {
        if (Mark[Tempt2->Vertex] == TRUE)
        {
          if ((OldLabel[Tempt2->Vertex] + Tempt2->Length) < Min)
          {
            NewPred[Tempt1->Vertex] = Tempt2->Vertex;
            Min = OldLabel[Tempt2->Vertex] + Tempt2->Length;
          }
        }
        Tempt2 = Tempt2->Next;
      }
      NewLabel[Tempt1->Vertex] = Min;

      Tempt1 = Tempt1->Next;
    }
  }
}

Done = TRUE;
for (i = 1; i <= NumVertices; i++)
{
  if (OldLabel[i] != NewLabel[i])
  {
    Done = FALSE;
    break;
  }
}

if (Done == TRUE)
{
  for (Terminal = 1; Terminal <= NumVertices; Terminal++)
  if (OldLabel[Terminal] < +Infty)
  {
    printf("\n ");
    printf("Ton tai duong di tu %d", Start);
    printf(" den %d", Terminal);
```



```
        printf("\n Duong di qua cac dinh:");
        printf(" % d " , Start);
        PathTwoVertex(OldPred, Start, Terminal);
        i = Terminal;
        printf(" Do dai la: %d ", OldLabel[Terminal]);
        printf("\n ");
        getch();
      }
      else
      {
        printf("\n ");
        printf("Khong ton tai duong di tu %d", Start);
        printf(" den %d", Terminal);
      }
      return;
    }

    for (i = 1; i <= NumVertices; i++)
    if (OldLabel[i] != NewLabel[i])
    {
      Mark[i] = TRUE;
      OldPred[i] = NewPred[i];
      OldLabel[i] = NewLabel[i];
    }
    else Mark[i] = FALSE;
  }
  printf("\n Ton tai mach co do dai am");
}
```

Để chứng minh thuật toán tìm được cho lời giải tối ưu của bài toán cần sử dụng nguyên lý tối ưu của quy hoạch động và từ nhận xét là nếu không có đường đi tối ưu qua k cung thì sẽ không có đường đi tối ưu qua $(k+1)$ cung. Chúng ta không trình bày chứng minh đó; bạn đọc quan tâm có thể xem [6].

Đồ thị có hướng có trọng số âm nhưng không có mạch độ dài âm

Trong trường hợp đồ thị có hướng có trọng số tuỳ ý nhưng không có mạch có độ dài âm thì *thuật toán Moore-Bellman-d'Usopo* tìm đường đi ngắn nhất từ s đến t trình bày dưới đây sẽ hiệu quả hơn.

Nhãn của mọi đỉnh $v_i \in V$ là cặp $(P(v_i), L(v_i))$. Kết thúc thuật toán, $L(t)$ là độ dài của đường đi ngắn nhất từ s đến t và vector $\mathbf{P}$ cho ta thông tin của đường đi này.

1. [Khởi tạo] Đặt
$$L(v_i) := \begin{cases} +\infty & \text{nếu} \quad v_i \neq s, \\ 0 & \text{nếu} \quad v_i = s, \end{cases}$$
và
$$P(v_i) := \begin{cases} -1 & \text{nếu} \quad v_i \neq s, \\ 0 & \text{nếu} \quad v_i = s, \end{cases}$$
Khởi tạo Stack chỉ có một phần tử là s.

2. [Bước lặp] Trong khi Stack khác NULL

{
lấy ra phần tử v_i *ở đầu* Stack;
với mọi đỉnh $v_j \in V$ sao cho $v_j \in \Gamma(v_i)$,
{
nếu $[L(v_i) + w(v_i, v_j) < L(v_j)]$ thì gán
{
$L(v_j) = L(v_i) + w(v_i, v_j)$;
$P(v_j) = v_i$;
nếu đỉnh v_j chưa bao giờ ở trong Stack thì chèn v_j *vào cuối* Stack;
ngược lại, nếu v_j đã từng ở trong Stack, nhưng hiện tại không có trong đó thì chèn v_j *vào đầu* của Stack.
}
}
}

3.3 Đường đi ngắn nhất giữa tất cả các cặp đỉnh

Giả sử cần phải tìm đường đi ngắn nhất giữa hai đỉnh tùy ý. Rõ ràng ta có thể giải bài toán này bằng cách sử dụng n lần thuật toán mô tả ở phần trước, mà trong mỗi lần thực hiện, ta sẽ chọn s lần lượt là các đỉnh của đồ thị. Trong trường hợp đồ thị đầy đủ có ma trận trọng lượng không âm, số phép tính cần thiết tỉ lệ với n^3; trái lại với ma trận trọng lượng tổng quát, số phép tính tỉ lệ với n^4.

Trong phần này chúng ta sẽ trình bày hai cách hoàn toàn khác để giải bài toán tìm đường đi ngắn nhất giữa tất cả các cặp đỉnh. Những phương pháp này có thể áp dụng cho các ma trận trọng lượng tổng quát và chỉ đòi hỏi số phép tính tỉ lệ với n^3. Với trường hợp ma trận trọng lượng không âm, nói chung, các phương pháp này sẽ nhanh hơn 50% cách áp dụng thuật toán Dijkstra n lần.

3.3.1 Thuật toán Hedetniemi (trường hợp ma trận trọng lượng không âm)

Một trong những mục đích của các thuật toán tìm đường đi ngắn nhất là xác định tất cả các độ dài của các đường đi ngắn nhất có thể có trong một đồ thị có trọng lượng tại cùng một thời điểm. Trong phần này, chúng ta thảo luận cách tiếp cận khác giải quyết bài toán này (xem [2]).

Thuật toán được xây dựng trên cơ sở một cách tính mới lũy thừa của các ma trận, mà chúng ta sẽ gọi là *tổng ma trận Hedetniemi*. Tổng này được Hedetniemi trình bày với Nystuen tại trường đại học Michigan.

Xét đồ thị liên thông có trọng lượng $G = (V, \Gamma)$ với tập các đỉnh $V = \{v_1, v_2, \ldots, v_n\}$ và ma trận trọng lượng không âm $W := [w_{ij}]$ cấp $n \times n$.

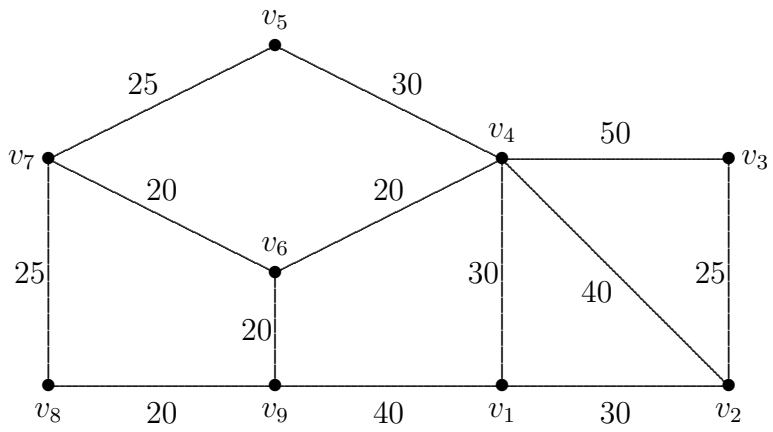
Chẳng hạn, đồ thị trong Hình 3.2 có ma trận trọng lượng

$$W = \begin{pmatrix} 0 & 30 & \infty & 30 & \infty & \infty & \infty & \infty & 40 \\ 30 & 0 & 25 & 40 & \infty & \infty & \infty & \infty & \infty \\ \infty & 25 & 0 & 50 & \infty & \infty & \infty & \infty & \infty \\ 30 & 40 & 50 & 0 & 30 & 20 & \infty & \infty & \infty \\ \infty & \infty & \infty & 30 & 0 & \infty & 25 & \infty & \infty \\ \infty & \infty & \infty & 20 & \infty & 0 & 20 & \infty & 20 \\ \infty & \infty & \infty & \infty & 25 & 20 & 0 & 25 & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & 25 & 0 & 20 \\ 40 & \infty & \infty & \infty & \infty & 20 & \infty & 20 & 0 \end{pmatrix}.$$

Chúng ta giới thiệu một phép toán mới trên các ma trận gọi là tổng ma trận Hedetniemi.

Định nghĩa 3.3.1 Giả sử A là ma trận cấp $m \times n$ và B là ma trận cấp $n \times p$. *Tổng ma trận Hedetniemi* của hai ma trận A và B là ma trận C cấp $m \times p$, ký hiệu $A \oplus B$, xác định bởi:

$$c_{ij} := \min\{a_{i1} + b_{1j}, a_{i2} + b_{2j}, \ldots, a_{in} + b_{nj}\}.$$



Hình 3.2:

Ví dụ 3.3.2 Tìm tổng ma trận $A \oplus B$ nếu $A = \begin{pmatrix} 0 & 1 & 2 \\ 2 & 0 & 3 \\ 5 & 6 & 0 \end{pmatrix}$ và $B = \begin{pmatrix} 0 & 3 & 4 \\ 5 & 0 & 4 \\ 3 & 1 & 0 \end{pmatrix}$.

Ta có

$$A \oplus B = \begin{pmatrix} 0 & 1 & 2 \\ 2 & 0 & 3 \\ 5 & 6 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 3 & 4 \\ 5 & 0 & 4 \\ 3 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 2 \\ 2 & 0 & 3 \\ 3 & 1 & 0 \end{pmatrix}.$$

Chẳng hạn, phần tử c_{23} xác định bởi

$$c_{23} = \min\{2+4, 0+4, 3+0\} = 3.$$

Ví dụ 3.3.3 Tìm tổng ma trận $A \oplus B$ nếu $A = \begin{pmatrix} 0 & 1 & \infty \\ 1 & 0 & 4 \\ \infty & 4 & 0 \end{pmatrix}$ và $B = \begin{pmatrix} 1 & 0 & \infty \\ 1 & 0 & 4 \\ \infty & 4 & 0 \end{pmatrix}$.

Ta có

$$A \oplus B = \begin{pmatrix} 0 & 1 & \infty \\ 1 & 0 & 4 \\ \infty & 4 & 0 \end{pmatrix} + \begin{pmatrix} 1 & 0 & \infty \\ 1 & 0 & 4 \\ \infty & 4 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 5 \\ 1 & 0 & 4 \\ 5 & 4 & 0 \end{pmatrix}.$$

Chẳng hạn, phần tử c_{13} xác định bởi

$$c_{23} = \min\{0+\infty, 1+4, \infty+0\} = 5.$$

Bây giờ áp dụng tổng ma trận Hedetniemi vào việc tìm đường đi ngắn nhất. Xét ví

dụ trong Hình 3.2. Ký hiệu $W^2 := W \oplus W, W^k := W^{k-1} \oplus W, k \geq 2$. Khi đó

$$W^2 = \begin{pmatrix} 0 & 30 & 55 & 30 & 60 & 50 & \infty & 60 & 40 \\ 30 & 0 & 25 & 40 & 70 & 60 & \infty & \infty & 70 \\ 55 & 25 & 0 & 50 & 80 & 70 & \infty & \infty & \infty \\ 30 & 40 & 50 & 0 & 30 & 20 & 40 & \infty & 40 \\ 60 & 70 & 80 & 30 & 0 & \infty & 25 & \infty & \infty \\ \infty & \infty & \infty & 20 & \infty & 0 & 20 & \infty & 20 \\ \infty & \infty & \infty & \infty & 25 & 20 & 0 & 25 & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & 25 & 0 & 20 \\ 40 & \infty & \infty & \infty & \infty & 20 & \infty & 20 & 0 \end{pmatrix}.$$

Chúng ta hãy xét cách xác định một phần tử của ma trận $W^2 = [a_{ij}^{(2)}]$:

$$\begin{aligned} a_{13}^{(2)} &= \min\{0+\infty, 30+25, \infty+0, 30+50, \infty+\infty, \infty+\infty, \infty+\infty, \infty+\infty, 40+\infty\} \\ &= 55. \end{aligned}$$

Chú ý rằng giá trị 55 là tổng của 30, độ dài của đường đi ngắn nhất với số cung một từ đỉnh v_1 đến đỉnh v_2, và của 25, độ dài của cung nối đỉnh v_2 và đỉnh v_3. Do đó $a_{13}^{(2)}$ là độ dài của đường đi ngắn nhất từ v_1 đến v_2 với số cung nhiều nhất hai. Suy ra W^2 cho ta thông tin của tất cả các độ dài đường đi ngắn nhất giữa hai đỉnh có số cung nhiều nhất hai.

Tương tự, W^3 cho ta thông tin của tất cả các độ dài đường đi ngắn nhất giữa hai đỉnh có số cung nhiều nhất ba, và vân vân. Do đồ thị có n đỉnh nên có nhiều nhất $(n-1)$ cung trên đường đi ngắn nhất giữa hai đỉnh. Vậy

Định lý 3.3.4 *Trong đồ thị có trọng số không âm n đỉnh, phần tử hàng i cột j của ma trận Hedetniemi W^{n-1} là độ dài của đường đi ngắn nhất giữa đỉnh v_i và v_j.*

Với đồ thị trong Hình 3.2 có chín đỉnh, ta có

$$W^8 = \begin{pmatrix} 0 & 30 & 55 & 30 & 60 & 50 & 70 & 60 & 40 \\ 30 & 0 & 25 & 40 & 70 & 60 & 80 & 90 & 70 \\ 55 & 25 & 0 & 50 & 80 & 70 & 90 & 110 & 90 \\ 30 & 40 & 50 & 0 & 30 & 20 & 40 & 60 & 40 \\ 60 & 70 & 80 & 30 & 0 & 45 & 25 & 50 & 65 \\ 50 & 60 & 70 & 20 & 45 & 0 & 20 & 40 & 20 \\ 70 & 80 & 90 & 40 & 25 & 20 & 0 & 25 & 40 \\ 60 & 90 & 110 & 60 & 50 & 40 & 25 & 0 & 20 \\ 40 & 70 & 90 & 40 & 65 & 20 & 40 & 20 & 0 \end{pmatrix}.$$

Do đó, đường đi ngắn nhất từ v_1 đến v_7 có độ dài 70.

Điều lý thú nhất trong ví dụ này là $W^4 = W^8$. Thật vậy, đẳng thức suy trực tiếp từ đồ thị trong Hình 3.2: mọi đường đi ngắn nhất đi qua nhiều nhất bốn cung. Bởi vậy độ dài của đường đi ngắn nhất được xác định bởi ma trận W^4 thay vì phải tính đến W^8. Tổng quát ta có

Định lý 3.3.5 *Trong đồ thị có trọng số không âm n đỉnh, nếu ma trận Hedetniemi* $W^k \neq W^{k-1}$, *còn* $W^k = W^{k+1}$, *thì* W^k *biểu thị tập các độ dài của các đường đi ngắn nhất, và số cung trên mỗi đường đi ngắn nhất không vượt quá k.*

Do đó, thuật toán này có thể dừng ở bước lặp thứ $k < (n-1)$. Dưới đây là đoạn chương trình minh họa tính ma trận luỹ thừa của ma trận trọng lượng W.

```
void Hetdetniemi()
{
  int i, j, k, t;
  int Old[MAXVERTICES][MAXVERTICES], New[MAXVERTICES][MAXVERTICES];
  Boolean Flag;

  for (i = 1; i <= NumVertices; i++)
    for  (j = 1; j <= NumVertices; j++)  Old[i][j] = w[i][j];

  for (t = 1; t < NumVertices; t++)
  {
    Flag = TRUE;

    for (i = 1; i <= NumVertices; i++)
    {
      for (j = 1; j <= NumVertices; j++)
      {
        New[i][j] = Old[i][1] + w[1][j];
        for (k = 2; k <= NumVertices; k++)
          New[i][j] = min(New[i][j], Old[i][k] + w[k][j]);

        if (New[i][j] != Old[i][j])
        {
          Flag = FALSE;
          Old[i][j] = New[i][j];
        }
      }
    }
```



```
    if (Flag == TRUE) break;
  }
}
```

Xác định đường đi ngắn nhất

Bây giờ ta tìm đường đi ngắn nhất từ v_1 đến v_7 (độ dài 70). Ta có

$$w_{14}^{(4)} = w_{1k}^{(3)} \oplus w_{k7}$$

với k nào đó. Nhưng các phần tử $w_{1k}^{(3)}$ tạo thành vector hàng

$$(0, 30, 55, 30, 60, 50, 70, 60, 40)$$

và các phần tử w_{k7} tạo thành vector cột

$$(\infty, \infty, \infty, \infty, 25, 20, 0, 25, \infty).$$

Vì giá trị nhỏ nhất đạt được tại $k = 6$ ứng với $70 = 50 + 20$ (và $k = 7$) nên đường đi ngắn nhất từ v_1 đến v_7 sẽ đi qua nhiều nhất ba cung từ v_1 đến v_6 và kết thúc với cung độ dài 20 từ v_6 đến v_7. (Thật ra, do giá trị nhỏ nhất cũng đạt được tại $k = 7$ (ứng với $70 + 0$) nên tồn tại đường đi ngắn nhất từ v_1 đến v_7 với tổng số cung đi qua không vượt quá ba).

Tiếp đến ta tìm cung trước khi đến đỉnh v_6. Chú ý rằng $w_{16}^{(4)} = 70 - 20 = 50$. Các phần tử w_{k6} tạo thành vector cột

$$(\infty, \infty, \infty, 20, \infty, 0, 20, \infty, 20).$$

Lần này giá trị nhỏ nhất đạt tại $k = 4$ (ứng với $50 = 30 + 20$) nên đường đi ngắn nhất độ dài 50 từ v_1 đến v_6 kết thúc với cung (v_4, v_6) độ dài 20. Cuối cùng, các phần tử w_{k4} tạo thành vector cột

$$(30, 40, 50, 0, 30, 20, \infty, \infty, \infty),$$

và giá trị nhỏ nhất đạt tại $k = 1$ hoặc $k = 4$ (ứng với $30 + 0$ hoặc $0 + 30$) nên tồn tại cung độ dài 30 từ v_1 đến v_4. Vậy đường đi ngắn nhất từ v_1 đến v_7 là v_1, v_4, v_6, v_7 (các cung có độ dài 30, 20, 20).

Do đó thuật toán Hedetniemi cho một minh họa hình học trong mỗi bước lặp, và sử dụng các ma trận có thể phục hồi được đường đi ngắn nhất. Như vậy, chúng ta cần thêm một ma trận P lưu trữ thông tin của các đường đi ngắn nhất. Ma trận này sẽ được cập nhật trong mỗi bước lặp khi tính W^k từ W^{k-1}.

3.3.2 Thuật toán Floyd (trường hợp ma trận trọng lượng tuỳ ý)

Thuật toán dưới đây được đưa ra lần đầu tiên bởi Floyd [25] và được làm mịn hơn bởi Murchland [46].

Thuật toán Floyd

1. [Khởi tạo] Đặt $k := 0$.
2. $k := k + 1$.
3. [Bước lặp] Với mọi $i \neq k$ sao cho $w_{ik} \neq \infty$ và với mọi $j \neq k$ sao cho $w_{kj} \neq \infty$, thực hiện phép gán
$$w_{ij} := \min\{w_{ij}, (w_{ik} + w_{kj})\}. \tag{3.3}$$
4. [Điều kiện kết thúc] (a) Nếu tồn tại chỉ số i sao cho $w_{ii} < 0$ thì tồn tại mạch với độ dài âm chứa đỉnh v_i. Bài toán vô nghiệm; thuật toán dừng.

   (b) Nếu $w_{ii} \geq 0$ với mọi $i = 1, 2, \ldots, n$, và $k = n$, bài toán có lời giải: ma trận w_{ij} cho độ dài đường đi ngắn nhất từ v_i đến v_j. Thuật toán dừng.

   (c) Nếu $w_{ii} \geq 0$ với mọi $i = 1, 2, \ldots, n$, nhưng $k < n$, chuyển sang Bước 2.

Chứng minh tính đúng đắn của thuật toán Floyd là hoàn toàn đơn giản [35], [25] và dành cho người đọc. Phép toán cơ bản của Phương trình 3.3 trong thuật toán này gọi là *phép toán bộ ba* và có nhiều ứng dụng trong những bài toán tương tự bài toán tìm đường đi ngắn nhất (xem [14], [30]).

Các đường đi ngắn nhất có thể nhận được đồng thời cùng với các độ dài đường đi ngắn nhất bằng cách sử dụng quan hệ đệ qui tương tự Phương trình 3.2. Bằng cách áp dụng cơ chế của Hu [35] để lưu giữ thông tin các đường đi ngắn nhất cùng với độ dài của nó. Cụ thể là đưa vào ma trận vuông cấp n : $P := [\theta_{ij}]$; và biến đổi nó đồng thời với việc biến đổi ma trận W. Phần tử θ_{ij} của ma trận P sẽ chỉ ra đỉnh đi trước v_j trong đường đi ngắn nhất từ v_i đến v_j; ở bước đầu tiên ta gán cho ma trận P các giá trị đầu là $\theta_{ij} := v_i$ với mọi đỉnh v_i và v_j.

Cùng với việc biến đổi ma trận W theo Phương trình 3.3 trong Bước 3, ta biến đổi P theo quy tắc

$$\theta_{ij} := \begin{cases} \theta_{kj} & \text{nếu} \quad (w_{ik} + w_{kj}) < w_{ij}, \\ \theta_{ij} & \text{nếu ngược lại.} \end{cases}$$

Kết thúc thuật toán, ma trận P thu được sẽ giúp cho ta việc tìm các đường đi ngắn nhất. Chẳng hạn đường đi ngắn nhất giữa hai đỉnh v_i và v_j là

$$v_i, v_\nu, \ldots, v_\gamma, v_\beta, v_\alpha, v_j$$

trong đó $v_\alpha = \theta_{ij}, v_\beta = \theta_{i\alpha}, v_\gamma = \theta_{i\beta}, ...$, cho đến khi $v_i = \theta_{i\nu}$.

Cần chú ý ở đây là nếu tất cả các giá trị w_{ii} được khởi tạo bằng ∞ (thay cho bằng 0) lúc xuất phát thuật toán, thì các giá trị cuối cùng của w_{ii} là độ dài của mạch ngắn nhất xuất phát từ đỉnh v_i. Ngoài ra cũng có thể dễ dàng xác định mạch có độ dài âm khi $w_{ii} < 0$ dựa vào ma trận đường đi P.

Thủ tục Floyd() sau minh họa thuật toán đã trình bày.

```
void Floyd()
{
  byte i, j, k;
  AdjPointer Tempt;
  byte Pred[MAXVERTICES][MAXVERTICES];
  int Weight[MAXVERTICES][MAXVERTICES], NewLabel;
  byte Start, Terminal;

  for (i = 1; i <= NumVertices; i++)
  {
    for (j = 1; j <= NumVertices; j++)
    {
      Weight[i][j] = +INFTY;
      Pred[i][j] = i;
    }
    Weight[i][i] = 0;
  }

  for (i = 1; i <= NumVertices; i++)
  {
    Tempt = V_out[i]->Next;
    while (Tempt != NULL)
    {
      Weight[i][Tempt->Vertex] = (long)Tempt->Length;
      Tempt = Tempt->Next;
    }
  }

  for (k = 1; k <= NumVertices; k++)
  {
    for (i = 1; i <= NumVertices; i++)
    {
      if ((i != k) && (Weight[i][k] < +INFTY))
```



```
      for (j = 1; j <= NumVertices; j++)
        if ((j != k) && (Weight[k][j] < +INFTY))
        {
          NewLabel = Weight[i][k] + Weight[k][j];
          if (Weight[i][j] > NewLabel)
          {
            Weight[i][j] = NewLabel;
            Pred[i][j] = Pred[k][j];
          }
        }
        if (Weight[i][i] < 0)
        {
          printf("Ton tai mach do dai am qua dinh %d", i);
          printf(" %6d ", Weight[i][i]);
          return;
        }
      }
    }

    // Vi du minh hoa
    Start = 1;
    Terminal = 3;

    if (Weight[Start][Terminal] < +INFTY)
    {
      printf("Ton tai duong di tu %d", Start);
      printf(" den %d", Terminal);
      printf("\nDuong di qua cac dinh:");

      i = Terminal;
      while (i != Start)
      {
        printf("%3d ", i);
        i = Pred[Start][i];
      }
      printf("%3d ", Start);
      printf("\nTrong luong la %3d ", Weight[Start][Terminal]);
    }
    else
      printf("\n Khong ton tai duong di tu %d den %d", Start, Terminal);
}
```


3.4 Phát hiện mạch có độ dài âm

Vấn đề phát hiện các mạch có độ dài âm trong một đồ thị tổng quát là quan trọng không những trong chính bài toán tìm đường đi ngắn nhất mà còn là một bước cơ bản trong những thuật toán khác (xem Phần 3.4.1 và [14]).

Ta biết rằng Thuật toán Floyd tìm đường đi ngắn nhất giữa các cặp đỉnh có thể phát hiện các mạch độ dài âm trong đồ thị. Hơn nữa, nếu đồ thị chứa một đỉnh s mà có thể đến tất cả các đỉnh khác từ đó, thì có thể áp dụng Thuật toán Ford-Moore-Bellman (tìm tất cả các đường đi ngắn nhất xuất phát từ đỉnh s) để phát hiện các mạch có độ dài âm như đã thực hiện trong Bước 3 của phần này. Nếu tồn tại đỉnh không thể đến được từ s (chẳng hạn, khi G là đồ thị vô hướng không liên thông) thì Thuật toán Ford-Moore-Bellman sẽ kết thúc và chỉ các đỉnh có thể đến được từ s có nhãn hữu hạn, các đỉnh khác không đến được từ s có nhãn bằng ∞. Trong trường hợp này có thể tồn tại các mạch có độ dài âm trong thành phần liên thông không chứa s và không được phát hiện. Tuy nhiên, nhiều ứng dụng cần kiểm tra có mạch độ dài âm hay không, đồ thị được xét có đỉnh s đến được tất cả các đỉnh khác, và do đó với những trường hợp như vậy, để xác định sự tồn tại của mạch độ dài âm, Thuật toán Ford-Moore-Bellman sẽ cho phép tính toán hiệu quả hơn Thuật toán Floyd.

Các nguyên tắc kết thúc của Thuật toán Ford-Moore-Bellman được trình bày để tính toán ít nhất các đường đi ngắn nhất từ s khi không có mạch độ dài âm. Các nguyên tắc này có thể cải biên để phát hiện mạch độ dài âm sớm hơn như sau.

Sau khi gán lại nhãn của đỉnh v_i từ một đỉnh v_{j^*} theo Phương trình 3.2 ta kiểm tra đỉnh v_i có thuộc đường đi ngắn nhất hiện hành (mà có thể suy từ các nhãn hiện hành θ) từ s đến v_{j^*} hay không. Nếu đúng, thì đỉnh v_{j^*} đã được gán nhãn thông qua v_i và điều này dẫn đến $L^k(v_{j^*}) + w(v_{j^*}, v_i) < L^k(v_i)$; do đó một phần của đường đi ngắn nhất hiện hành từ v_i đến v_{j^*} cộng thêm cung (v_{j^*}, v_i) sẽ tạo thành mạch có độ dài âm và thuật toán kết thúc. Nếu mặt khác, nhãn của đỉnh v_i hoặc không thay đổi bởi Phương trình 3.2, hoặc nhận nhãn mới từ một đỉnh v_{j^*} nhưng v_i không thuộc đường đi ngắn nhất từ s đến v_{j^*}, thì thuật toán tiếp tục Bước 3 như trước. Cần chú ý rằng, cải tiến trên chính là giảm những dư thừa không cần thiết trong Thuật toán Ford-Moore-Bellman, do một mạch có độ dài âm được xác định ngay khi nó được tạo ra và không cần đợi kết thúc thủ tục.

3.4.1 Mạch tối ưu trong đồ thị có hai trọng lượng

Một vấn đề nảy sinh trong nhiều lĩnh vực liên quan đến một đồ thị có hướng mà mỗi cung (v_i, v_j) được gán hai trọng lượng: w_{ij} và b_{ij}. Bài toán là tìm một mạch Φ mà cực tiểu (hoặc

cực đại) hàm mục tiêu

$$z(\Phi) := \frac{\sum_{(v_i,v_j)\in\Phi} w_{ij}}{\sum_{(v_i,v_j)\in\Phi} b_{ij}}.$$

Chẳng hạn, xét hành trình của một con tàu hay một máy bay trên mạng giao thông và giả sử rằng w_{ij} là "lợi nhuận" và b_{ij} là "thời gian" cần thiết để di chuyển trên cung (v_i, v_j). Bài toán đặt ra là tìm hành trình để tối đa lợi nhuận với thời gian nhỏ nhất.

Các vấn đề khác có thể phát biểu dạng bài toán tìm các mạch tối ưu trong đồ thị có hai trọng lượng: Lập lịch để tính toán song song, tối ưu đa mục tiêu trong xử lý công nghiệp.

Bài toán tìm một mạch Φ trong đồ thị có hai trọng lượng để cực tiểu hàm mục tiêu $z(\Phi)$ có thể giải quyết nhờ thuật toán phát hiện các mạch có độ dài âm như sau. Giả sử rằng các trọng lượng w_{ij} và b_{ij} là các số thực tuỳ ý (dương, âm hoặc bằng không) nhưng thoả mãn ràng buộc $\sum_{(v_i,v_j)\in\Phi} b_{ij} > 0$, với mọi mạch Φ trong G. (Trong hầu hết các tình huống, chẳng hạn các vấn đề nêu trên, $w_{ij} \in \mathbf{R}$ nhưng $b_{ij} \geq 0$ với mọi i và j).

Chúng ta chọn một giá trị thử z^k đối với hàm mục tiêu $z(\Phi)$ và xét đồ thị có trọng lượng

$$w_{ij}^k = w_{ij} - z^k b_{ij}.$$

Với ma trận trọng lượng w_{ij}^k mới, xét bài toán đường đi ngắn nhất trên G. Có ba trường hợp xảy ra:

Trường hợp A. Tồn tại mạch có độ dài âm Φ^- sao cho

$$\sum_{(v_i,v_j)\in\Phi^-} w_{ij}^k < 0.$$

Trường hợp B. Không tồn tại mạch có độ dài âm và

$$\sum_{(v_i,v_j)\in\Phi} w_{ij}^k > 0$$

với mọi mạch Φ.

Trường hợp C. Tồn tại mạch có độ dài bằng không (nhưng không có mạch độ dài âm); tức là

$$\sum_{(v_i,v_j)\in\Phi^0} w_{ij}^k = 0$$

với mạch Φ^0 nào đó.

Trong Trường hợp A chúng ta có thể nói rằng z^* (giá trị cực tiểu của z) nhỏ hơn z^k vì

$$\sum_{(v_i,v_j)\in\Phi^-} w_{ij}^k \equiv \sum_{(v_i,v_j)\in\Phi^-} w_{ij} - z^k \sum_{(v_i,v_j)\in\Phi^-} b_{ij} < 0$$

chỉ có thể đúng nếu

$$\frac{\sum_{(v_i,v_j)\in\Phi^{-1}} w_{ij}}{\sum_{(v_i,v_j)\in\Phi^{-1}} b_{ij}} < z^k$$

mà hiển nhiên chỉ ra rằng $z^* < z^k$.

Tương tự, trong Trường hợp B ta có thể nói rằng $z^* > z^k$; và trong Trường hợp C thì $z^* = z^k$.

Do đó thuật toán tìm kiếm nhị phân để giải quyết bài toán như sau: Khởi đầu với giá trị thử z^1; nếu nó quá lớn (tức là Trường hợp A) thử với $z^2 < z^1$; nếu nó quá nhỏ (tức là Trường hợp B) thử với $z^2 > z^1$. Khi đã có các cận trên và dưới (z_u và z_l tương ứng) ta có thể xác định z^* bằng cách thử $z^k = (z_u + z_l)/2$ và thay z_u bằng z^k nếu xảy ra Trường hợp A, hoặc thay z_l bằng z^k nếu xảy ra Trường hợp B. Do số các phép thử tỉ lệ với $1/\eta$, trong đó $0 < \eta \ll 1$ là sai số cho trước, và do mỗi lần thử (xác định mạch độ dài âm hoặc tính toán ma trận trọng lượng) đòi hỏi n^3 phép toán, nên tìm nghiệm của bài toán trên đòi hỏi $O[n^3 \log 1/\eta]$ phép toán.

