CHƯƠNG IV. CÁC BÀI TOÁN ĐƯỜNG ĐI

IV.1 Bài toán đường đi ngắn nhất

IV.1.1 Phát biểu bài toán

Cho G=(X, E) là một đồ thị có hướng. Ta định nghĩa ánh xạ trọng lượng như sau:

$$L: E \longrightarrow \mathbb{R}$$
$$e \longmapsto L(e)$$

Xét hai đỉnh i, j $\in$X, gọi P là một đường đi từ đỉnh i đến đỉnh j, trọng lượng (hay giá) của đường đi P được định nghĩa là:

$$L(P) = \sum (e \in P)\ L(e)$$

Mục đích của bài toán đường đi ngắn nhất là tìm đường đi P từ i đến j mà có trọng lượng nhỏ nhất trong số tất cả những đường đi có thể có.

Nhận xét.

- Mặc dù bài toán được phát biểu cho đồ thị có hướng có trọng, nhưng các thuật toán sẽ trình bày đều có thể áp dụng cho các đồ thị vô hướng có trọng bằng cách xem mỗi cạnh của đồ thị vô hướng như hai cạnh có cùng trọng lượng nối cùng một cặp đỉnh nhưng có chiều ngược nhau.
- Khi làm bài toán tìm đường đi ngắn nhất thì chúng ta có thể bỏ bớt đi các cạnh song song và chỉ chừa lại một cạnh có trọng lượng nhỏ nhất trong số các cạnh song song.

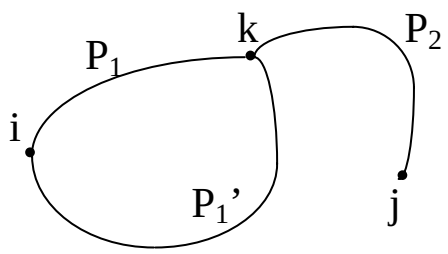
- Đối với các khuyên có trọng lượng không âm thì cũng có thể bỏ đi mà không làm ảnh hưởng đến kết quả của bài toán. Đối với các khuyên có trọng lượng âm thì có thể đưa đến bài toán đường đi ngắn nhất không có lời giải (xem IV.1.3).
- Do các nhận xét vừa nêu, có thể xem dữ liệu nhập của bài toán đường đi ngắn nhất là ma trận L được định nghĩa như sau:

$$L_{ij} = \begin{cases} \text{trọng lượng cạnh nhỏ nhất nối i đến j nếu có,} \\ 0 \text{ nếu không có cạnh nối i đến j.} \end{cases}$$

Trong quá trình bày các thuật toán, để cho tổng quát, giá trị 0 trong ma trận L có thể thay thế bằng $+\infty$. Tuy nhiên khi cài đặt chương trình, chúng ta vẫn có thể dùng 0 thay vì $+\infty$ bằng cách đưa thêm một số lệnh kiểm tra thích hợp trong chương trình.

IV.1.2 Nguyên lý Bellman

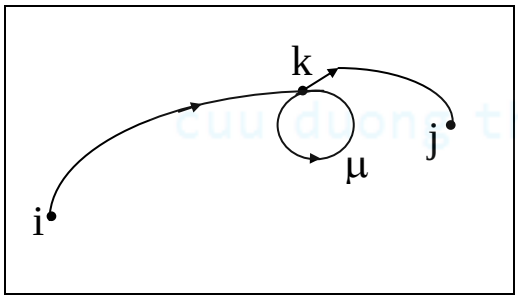
Hầu hết các thuật toán tìm đường đi ngắn nhất đều đặt cơ sở trên nguyên lý Bellman, đây là nguyên lý tổng quát cho các bài toán tối ưu hóa rời rạc, đối với trường hợp bài toán đường đi ngắn nhất thì có thể trình bày nguyên lý nầy như sau.



$$L(P1') < L(P1) \Rightarrow L(P_1' \oplus P_2) < L(P_1 \oplus P_2) = L(P)$$

Giả sử P là đường đi ngắn nhất từ đỉnh i đến đỉnh j và k là một đỉnh nằm trên đường đi P. Giả sử $P=P_1 \oplus P_2$ với P_1 là đường đi con của P từ i đến k và P_2 là đường đi con của P từ k đến j. Nguyên lý Bellman nói rằng P_1 cũng là đường đi ngắn nhất từ i đến k, vì nếu có một đường đi khác là P_1' từ i đến k có trọng lượng nhỏ hơn hơn P_1 thì $P_1' \oplus P_2$ là đường đi từ i đến j mà có trọng lượng nhỏ hơn P, điều nầy mâu thuẫn với tính ngắn nhất của P.

IV.1.3 Điều kiện tồn tại lời giải



Gọi P là một đường đi từ i đến j, giả sử P có chứa một mạch μ. Có 2 trường hợp sau đây.

- Nếu $L(\mu) \geq 0$ thì có thể cải tiến đường đi P bằng cách bỏ đi mạch μ.
- Nếu $L(\mu) < 0$ thì không tồn tại đường đi ngắn nhất từ đỉnh i đến đỉnh j vì nếu quay vòng tại μ càng nhiều vòng thì trọng lượng đường đi P càng nhỏ đi, tức là $L(P) \rightarrow -\infty$.

IV.1.4 Thuật toán Dijkstra

Xét đồ thị G=(X, E) có trọng với X={1, 2, ..., n} và giả sử các cạnh không âm.

- Dữ liệu nhập cho thuật toán là ma trận trọng lượng L (với qui ước $L_{hk}=+\infty$ nếu không có cạnh nối từ đỉnh h đến đỉnh k) và hai đỉnh i, j cho trước.
- Dữ liệu xuất là đường đi ngắn nhất từ i đến j.

Bước 1. Gán T:=X và gắn các nhãn:
Dodai[i]=0; Dodai[k]= $+\infty$, $\forall k \in X\backslash\{i\}$;
Nhan[k]=-1, $\forall k \in X$.

Bước 2. Nếu $j \notin T$ thì dừng và giá trị Dodai[j] chính là độ dài đường đi ngắn nhất từ i đến j và Nhan[j] là đỉnh nằm ngay trước j trên đường đi đó.

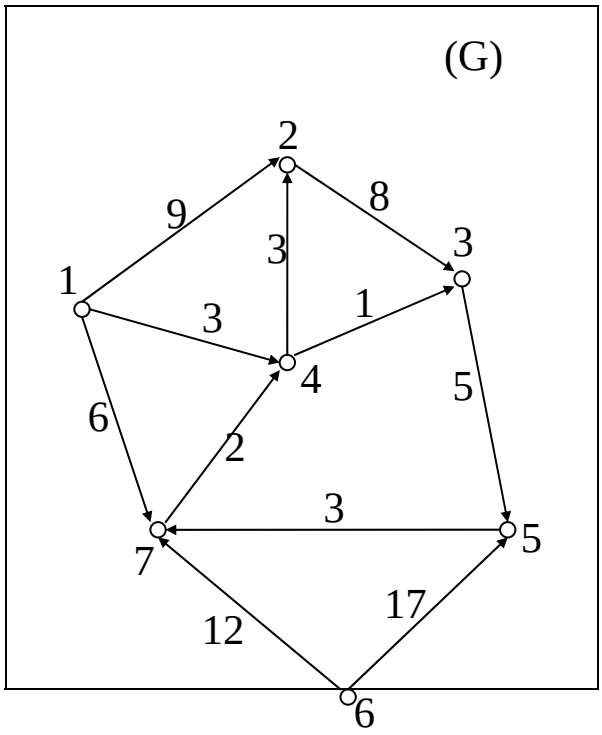
Bước 3. Chọn đỉnh $v \in T$ sao cho Dodai[v] nhỏ nhất và gán T := T\{v}.

Bước 4. Với mọi đỉnh $k \in T$ và có cạnh nối từ v đến k,
nếu Dodai[k]>Dodai[v]+L_{vk} thì
Dodai[k]= Dodai[v]+L_{vk} và Nhan[k]=v
Cuối với mọi.
Trở về bước 2.

Ghi chú: Khi thuật toán dừng, nếu Dodai[j]= $+\infty$ thì không tồn tại đường đi từ i đến j, nếu ngược lại thì Dodai[j] là độ dài đường đi ngắn nhất và ta lần ngược ra đường đi ngắn nhất (đi ngược từ j trở lại i) như sau:

```
write(j);
k:= Nhan[j];
while k<>i do
  begin
    write('<---', k);
    k := Nhan[k];
  end;
write('<---', i);
```

Ví dụ cho thuật toán Dijkstra.



Ta tìm đường đi ngắn nhất từ đỉnh 1 đến đỉnh 5 cho đồ thị (G) trong hình vẽ. Quá trình thực hiện thuật toán được mô tả trong các bảng sau đây, chúng ghi lại giá trị của các biến T, Dodai, Nhan. Đường đi ngắn nhất từ 1 đến 5 có độ dài là 9 và đi qua các đỉnh 1,4,3,5.

Các đỉnh	1	2	3	4	5	6	7
T	~~(1)~~	2	3	4	5	6	7
		2	3	~~(4)~~	5	6	7
		2	~~(3)~~		5	6	7
		~~(2)~~			5	6	7
					5	6	~~(7)~~
					~~(5)~~	6	

Dodai	(0)	+∞	+∞	+∞	+∞	+∞	+∞
		9	+∞	(3)	+∞	+∞	6
		6	(4)		+∞	+∞	6
		(6)			9	+∞	6
					9	+∞	(6)
					(9)	+∞	

Các đỉnh	1	2	3	4	5	6	7
Nhan	-1	-1	-1	-1	-1	-1	-1
		[1]	-1	[1]	-1	-1	[1]
		[4]	[4]	1	-1	-1	1
		4	4	1	[3]	-1	1

CÀI ĐẶT THUẬT TOÁN DIJKSTRA

Đoạn chương trình Pascal sau đây gồm hai thủ tục: Dijkstra (tìm đường đi ngắn nhất) và Induongdi (in ra đường đi ngắn nhất). Chúng ta dùng một cấu trúc tích hợp tên là DOTHI bao gồm cả thông tin về dữ liệu nhập của đồ thị và các biến cần thiết cho quá chạy của thuật toán Dijkstra. Thủ tục Dijkstra giả sử rằng đồ thị G đã có sẵn số đỉnh G.n và ma trận trọng lượng G.L; chúng ta qui ước một giá trị đặc biệt cho +∞ là VOCUC=-1và thêm một vài lệnh kiểm tra thích hợp trong chương trình.

```
const   MAXV=20;
        VOCUC=-1;
type    DINH = 1..MAXV;

        DOTHI=record
                    n: byte;
                    L: array[DINH, DINH] of real;
                    T, X: set of DINH;
                    Dodai: array[DINH] of real;
                    Nhan: array[DINH] of integer;
              end;
procedure InDuongDi (G: DOTHI; i, j: integer);
  var k: integer;
begin
  writeln('Duong ngan nhat tu ', i,' den ',j,' la:');
  write(j);
  k:=G.Nhan[j];
  while k<>i do
    begin
            write('<---', k);
            k := G.Nhan[k];
    end;
  writeln('<---', i);
end;
```

```
procedure Dijstra(var G: DOTHI; i, j: integer);
var min: real;
    k, v: DINH;
begin
   G.X := [1..G.n];
   G.T := G.X;
   G.Dodai[i] := 0;
   for k:=1 to G.n do
       begin
            if k<>i then
               G.Dodai[k]:=VOCUC;
            G.Nhan [k] := -1;
       end;
   while j in G.T  do
       begin
             {----------------------------------------}
             min:=-1;
             for k:=1 to G.n do
                  if (k in G.T) and (G.Dodai[k] <> VOCUC) then
                     if (min=-1) or (min>G.Dodai[k]) then
                         begin
                           min := G.Dodai[k];
                           v := k;
                         end;
             {----------------------------------------}

             G.T := G.T-[v];
             for k:=1 to G.n do
                  if (G.L[v, k] > 0) and (k in G.T) then
                     if (G.Dodai[k]=VOCUC) or
                        (G.Dodai[k] > G.Dodai[v]+G.L[v,k]) then
                         begin
                               G.Dodai[k] := G.Dodai[v]+G.L[v,k];
                               G.Nhan[k] := v;
                         end;
       end;
end;
```

IV.1.5 Thuật toán Floyd

Thuật toán Floyd được dùng để tìm ra đường đi ngắn nhất giữa tất cả cặp đỉnh bất kỳ của một đồ thị G với các cạnh có trọng lượng dương. Dữ liệu nhập cho thuật toán là ma trận trọng lượng L (với qui ước Lij=0 nếu không có cạnh nối từ đỉnh i đến đỉnh j). Thuật toán được thuật hiện bằng 3 vòng lặp lồng nhau, khi thuật toán kết thúc thì Lij sẽ là độ dài đường đi ngắn nhất từ đỉnh i đến đỉnh j nếu Lij>0 và đường đi không tồn tại

nếu Lij=0. Trong phần cài đặt, chúng ta sẽ bổ sung thêm kỹ thuật để chỉ ra cụ thể đường ngắn nhất.

```
Lặp i=1, 2, ..., n làm
    Lặp j=1, 2, ..., n làm
        Nếu L[j, i]>0 thì
            Lặp k=1, 2, ..., n làm
                Nếu L[i, k]>0 thì
                    Nếu L[j, k]=0 hay L[j, i]+L[i,k]<L[j, k] thì
                        L[j, k] = L[j, i]+L[i,k]
            Cuối lặp k.
    Cuối lặp j.
Cuối lặp i.
```

CÀI ĐẶT THUẬT TOÁN FLOYD

Trong cài đặt thuật toán Floyd, ngoài trọng lượng đường đi nối từ đỉnh i (gọi là nút 1 trên đường đi) đến đỉnh j (gọi là nút 2 trên đường đi), chúng ta bổ sung thêm một trường tên là *sau_nut1* để lưu chỉ số của đỉnh ngay sau i trên đường đi từ i đến j. Do đó mỗi phần tử L[i, j] là một mẫu tin gồm 2 trường: trường *dodai* là trọng lượng đường đi và trường *sau_nut1*. Mỗi khi đường đi được cải tiến thì giá trị của trường *sau_nut1* cũng thay đổi. Thủ tục *Floyd* nhận vào một tham số đồ thị G có kiểu cấu trúc FLOYD_GRAPH, trong đó giả sử các trường: G.n đã được khởi tạo là số đỉnh đồ thị, G.L[i,j].dodai được khởi tạo giá trị Lij của ma trận trọng lượng, G.L[i,j].sau_nut1 được khởi tạo giá trị là j nếu có cạnh nối i đến j và được khởi tạo giá trị 0 nếu ngược lại. Thủ tục *Induongdi* dùng để in ra đường đi ngắn nhất từ đỉnh i đến đỉnh j. Chú ý rằng với mỗi đồ thị G thì chỉ

cần gọi thủ tục *Floyd* một lần để tìm ra tất cả các đường đi, trong khi đó thủ tục *Induongdi* phải được gọi nhiều lần để in ra từng đường đi cụ thể.

```
const MaxN=20;
type  VERTEX=1..MaxN;
      PATH=record
              sau_nut1: integer;
              dodai: real;
           end;
      FLOYD_GRAPH=record
                     n: byte;
                     L: array[VERTEX, VERTEX] of PATH;
                  end;

function Floyd_init (filename: string; var g: FLOYD_GRAPH): boolean;
 var f: TEXT;
     i, j: integer;
begin
   assign(f, filename);
   {$I-}
   reset(f);
   if IOresult<>0 then
      writeln('Khong the mo tap tin ', filename)
   else
     begin
        read(f, g.n);
        for i:=1 to g.n do
          for j:=1 to g.n do
            begin
              read(f, g.L[i, j].dodai);
              if g.L[i, j].dodai >0 then
                 g.L[i, j].sau_nut1 := j
              else
                 g.L[i, j].sau_nut1 := 0;
            end;
     end;
   {$I+}
end;


procedure Floyd(var g: FLOYD_GRAPH);
 var i, j, k: VERTEX;
 begin
  for i:=1 to g.n do
   for j:=1 to g.n do
      if g.L[j, i].dodai > 0 then
        for k:=1 to g.n do
          if g.L[i, k].dodai > 0 then
            if (g.L[j, k].dodai=0) or
               (g.L[j, i].dodai +g.L[i, k].dodai < g.L[j, k].dodai) then
             begin
               g.L[j, k].dodai := g.L[j, i].dodai+g.L[i, k].dodai;
               g.L[j, k].sau_nut1 := g.L[j, i].sau_nut1;
             end;
end;
procedure Induongdi(var g: FLOYD_GRAPH; i, j: integer);
 var k: integer;
```

```
begin
   k := i;
   repeat
     write(k);
     if k<>j then
            write('--->');
     k := g.L[k, j].sau_nut1;
   until k=j;
   write(j);
   writeln('   ( do dai la:', g.L[i, j].dodai:0:2,' )')
end;
```

IV.1.4 Thuật toán Bellman

Thuật toán Bellman được dùng cho các đồ thị có trọng lượng âm. Thuật toán nầy tìm đường đi ngắn nhất từ một đỉnh của đồ thị đến mỗi đỉnh khác nếu đồ thị không có mạch âm. Nếu phát hiện đồ thị có mạch âm thì thuật toán dừng. Dữ liệu nhập cho thuật toán là ma trận trọng lượng L (với qui ước Lij=0 nếu không có cạnh nối từ đỉnh i đến đỉnh j).

Cho trước đỉnh $x \in X$.

Bước 1. Khởi tạo $\pi(0, x)=0$; $\pi(0, i)=+\infty$, $\forall i \neq x$ và k=1.

Bước 2. Với mỗi $i \in X$ ta đặt

$\pi(k, i)=\min(\{\pi(k-1, i)\} \cup \{\pi(k-1, j)+Lji$/có cạnh nối j đến i$\}$

Bước 3. Nếu $\pi(k, i)=\pi(k-1, i)$ với mọi $i \in X$ thì $\pi(k, i)$ chính là độ dài đường đi ngắn từ x đến i. Ngược lại nếu k<n thì tăng k:=k+1 và trở lại bước 2; nếu k=n thì dừng vì từ x đi tới được một mạch âm.

CÀI ĐẶT THUẬT TOÁN BELLMAN

Khi cài đặt thuật toán Bellman, ma trận π được cài đặt như một mảng 2 chiều *Pi*, mỗi phần tử bao gồm hai trường: trường *dodai* và trường *truoc_nut2*. Cụ thể *Pi[k,i].dodai* là giá trị của π(k, i) trong thuật toán; và *Pi[k,i].truoc_nut2* là chỉ số của nút đi ngay trước nút i trên đường đi ngắn nhất từ x đến i. Vì thuật toán Bellman làm việc trên cả các số âm nên không thể dùng giá trị đặc biệt là -1 cho +∞, chúng ta sẽ dùng giá trị lớn nhất của số nguyên 4 byte (maxlongint) để thay cho +∞. Đoạn chương trình sau đây gồm hai chương trình con Bellman và Induongdi với một số điểm cần lưu ý như sau.

- Hàm Bellman gồm 3 tham số: biến cấu trúc đồ thị G, một đỉnh x và chỉ số dòng k. Dữ liệu vào cho hàm là số đỉnh đồ thị G.n, ma trận trọng lượng G.L. Nếu hàm trả về FALSE thì từ đỉnh x có thể đi đến một mạch âm. Ngược lại nếu hàm trả về TRUE thì dữ liệu ra của hàm là một chỉ số k và ma trận G.Pi; trong đó G.Pi[k, i] chứa thông tin về đường đi ngắn nhất từ x đến i nếu G.Pi[k, i].dodai ≠ +∞.
- Hàm Induongdi cũng nhận vào 3 tham số như hàm Bellman và in ra tất cả các đường đi ngắn nhất nếu có từ đỉnh x đến tất cả các đỉnh khác của đồ thị.

```
const VOCUC=maxlongint;
      MAXV=20;
type
      BELL_ITEM=record
                  truoc_nut2: integer;
                  dodai: real;
            end;

      GRAPH=record
                n: integer;
                L:  array[1..MAXV, 1..MAXV] of real;
                Pi: array[0..MAXV, 1..MAXV] of BELL_ITEM;
            end;
function Bellman(var G: GRAPH; x: integer; var k: integer): boolean;
  (* tra ve false neu phat hien x di den chu trinh do dai am;
```

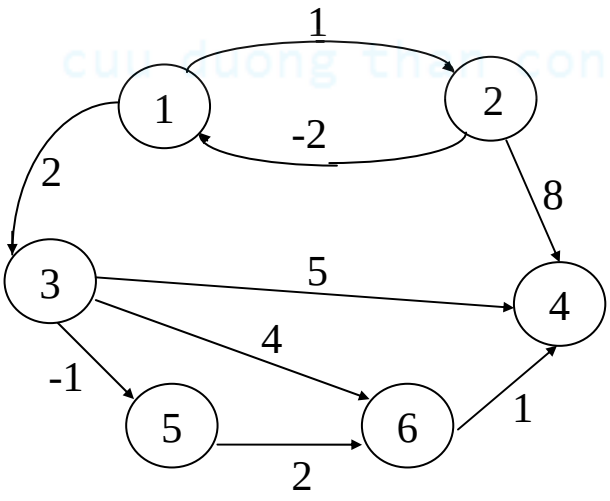
```
    tra ve true neu co duong di ngan nhat tu x den cac dinh khac *)
var i, j, j_min: integer;
    continue: boolean;
    min: real;
begin
   G.Pi[0, x].dodai := 0;
   G.Pi[0, x].truoc_nut2 := -1;
   for i:=1 to G.n do
      if i<>x then
         begin
            G.Pi[0, i].dodai := VOCUC;
            G.Pi[0, i].truoc_nut2 := -1;
         end;
  k := 1;
  continue := TRUE;
  while (k<G.n) and continue do
    begin
       for i:=1 to G.n do
         begin
           min := G.Pi[k-1, i].dodai;
           j_min := i;
           for j:=1 to G.n do
             begin
               if (G.L[j, i]<>0) and (G.Pi[k-1,j].dodai<>VOCUC) then
                  if min>G.Pi[k-1,j].dodai+G.L[j, i] then
                     begin
                       min := G.Pi[k-1,j].dodai+G.L[j, i];
                       j_min := j;
                     end;
             end;
           if j_min<>i then
             begin
               G.Pi[k, i].dodai := min;
               G.Pi[k, i].truoc_nut2 := j_min;
             end
           else
             begin
               G.Pi[k, i].dodai := min;
               G.Pi[k, i].truoc_nut2 := G.Pi[k-1, i].truoc_nut2;
             end
         end;
       continue := FALSE;
       i := 1;
       while (not continue) and (i<=G.n) do
         begin
           if G.Pi[k, i].dodai <> G.Pi[k-1, i].dodai then
              continue := TRUE;
           i := i+1;
         end;
       if continue then
          k := k+1;
    end;
  Bellman := not continue;
end;



procedure Induongdi(var G: GRAPH; x, k: integer);
```

```
var i, j: integer;
begin
   for i:=1 to G.n do
     if i<>x then
       begin
          if G.Pi[k, i].Dodai=VOCUC then
             writeln('   Khong co duong di tu ',x,' den ',i)
          else
            begin
              write('   Duong di tu ',x,' den ',i,': ');
              write(i);
              j := G.Pi[k, i].truoc_nut2;
              while j<>x do
               begin
                 write('<---', j);
                 j := G.Pi[k, j].truoc_nut2;
               end;
              writeln('<---', x);
            end;
       end;
end;
```

VÍ DỤ CHO THUẬT TOÁN BELLMAN



Xem đồ thị trong hình vẽ trên, chúng ta sẽ tính toán cho 2 trường hợp: các đường đi khởi đầu từ đỉnh 1 và các đường đi khởi đầu từ đỉnh 3.

- Trường hợp đường đi khởi đầu từ đỉnh 1, thuật toán dừng và phát hiện ra từ 1 có thể đến mạch âm, thực ra trường hợp nầy thì đỉnh 1 nằm ngay chính trên mạch âm. Tính toán chi tiết được cho trong bảng sau:

π và k	③	1	2	4	5	6
π(0, ?); k=1	0	∞	∞	∞	∞	∞
π(1, ?); k=2	0	∞	∞	5(3)	-1(3)	4(3)
π(2, ?); k=3	0	∞	∞	5(3)	-1(3)	1(5)
π(3, ?); k=4	0	∞	∞	2(6)	-1(3)	1(5)
π(4, ?); k=5	0	∞	∞	2(6)	-1(3)	1(5)

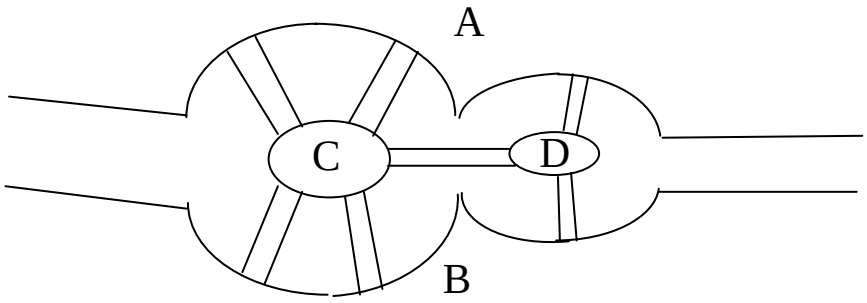
- Trường hợp đường đi khởi đầu từ đỉnh 3, thuật toán dừng và cho biết có đường đi ngắn nhất từ đỉnh 3 đến mỗi đỉnh còn lại hay không. Bảng sau đây cho biết kết quả tính chi tiết, các số trong ngoặc là các giá trị của trường *truoc_nut2*.

π và k	①	2	3	4	5	6
π(0, ?); k=1	0	∞	∞	∞	∞	∞
π(1, ?); k=2	0	1	2	∞	∞	∞
π(2, ?); k=3	-1	1	2	7	1	6
π(3, ?); k=4	-1	0	1	7	1	3
π(4, ?); k=5	-2	0	1	4	0	3
π(5, ?); k=6	-2	-1	0	4	0	2

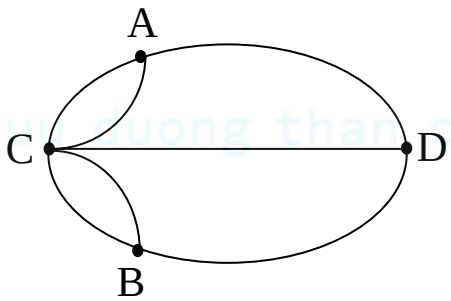
Dựa vào bảng trên có thể suy ra:
- đường đi từ 3 đến 1 hay 2: không có;
- đường đi ngắn nhất từ 3 đến 4 (độ dài 2): 4← 6← 5← 3;
- đường đi ngắn nhất từ 3 đến 5 (độ dài -1):5← 3;
- đường đi ngắn nhất từ 3 đến 6 (độ dài 1): 6← 5← 3.

IV.2 Đồ thị Euler

IV.2.1 Bài toán 7 chiếc cầu



Đây là một tình huống có thật ở Konigsberg (nước Đức), có hai vùng bị ngăn cách bởi một dòng sông và có 2 cù lao (đảo) ở giữa sông, 7 chiếc cầu nối những vùng nầy với nhau như minh họa trong hình vẽ trên. Người dân trong vùng thách đố nhau là thử tìm cách xuất phát từ một vùng đi dạo qua mỗi chiếc cầu đúng một lần và trở về nơi xuất phát. Năm 1736, nhà toán học Euler đã mô hình bài toán nầy bằng một đồ thị vô hướng với mỗi đỉnh ứng với một vùng, mỗi cạnh ứng với một chiếc cầu. Bài toán được phát biểu lại cho đồ thị trong hình vẽ bên dưới, hãy tìm một đường đi trong đồ thị qua tất cả các cạnh, mỗi cạnh chỉ một lần sau đó trờ về đỉnh xuất phát. Việc giải bài toán đưa đến các định lý liên quan đến đồ thị Euler.



IV.2.2 Các định nghĩa

(a) Dây chuyền Euler là dây chuyền đi qua tất cả các cạnh trong đồ thị và mỗi cạnh được đi qua đúng một lần.

(b) Chu trình Euler là dây chuyền Euler có đỉnh đầu trùng với đỉnh cuối.

(c) Đường đi Euler (đồ thị có hướng) là đường đi qua tất cả các cạnh của đồ thị và mỗi cạnh được đi qua đúng một lần.

(d) Mạch Euler là đường đi Euler có đỉnh đầu trùng với đỉnh cuối.

(e) Đồ thị Euler vô hướng là đồ thị vô hướng có chứa một chu trình Euler.

(f) Đồ thị Euler có hướng là đồ thị có hướng có chứa một mạch Euler.

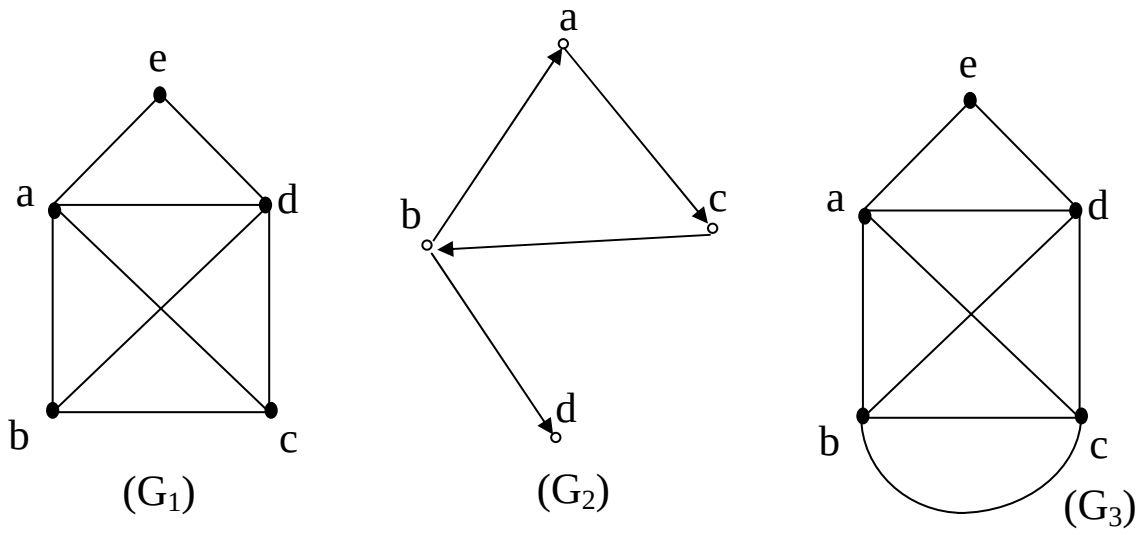
IV.2.3 Định lý Euler

Định lý 1: Cho G=(X, E) là một đồ thị vô hướng. Khi đó: G là đồ thị Euler $\Leftrightarrow$ G liên thông và d(x) chẵn $\forall x \in X$.

Định lý 2: Cho G=(X, E) là một đồ thị vô hướng. Khi đó G có chứa dây chuyền Euler và không chứa chu trình chu trình Euler khi và chỉ khi G liên thông có chứa đúng hai đỉnh bậc chẵn.

Định lý 3: Cho G=(X, E) là một đồ thị có hướng.
G là đồ thị Euler ⇔ G liên thông và $d^+(x)=d^-(x)$ $\forall x \in X$.

Ví dụ



Đồ thị (G1) có 2 đỉnh bậc lẻ nên không phải là đồ thị Euler. Tuy nhiên do thỏa mãn điều kiện của định lý 2, đồ thị nầy có dây chuyền Euler: bcadbaedc. Đồ thị (G2) có dây chuyền Euler nhưng không có đường đi Euler. Đồ thị vô hướng (G3) có mọi đỉnh đều bậc chẵn nên là đồ thị Euler vô hướng.

IV.3 Đồ thị Hamilton

Khái niệm đường đi Hamilton được xuất phát từ bài toán: "Xuất phát từ một đỉnh của khối thập nhị diện đều, hãy đi dọc theo các cạnh của khối đó sao cho đi qua tất cả các đỉnh khác, mỗi đỉnh qua đúng một lần, sau đó trở về đỉnh xuất phát". Bài toán nầy được nhà toán học Hamilton đưa ra vào năm 1859.

IV.3.1 Định nghĩa

(a) Dây chuyền Hamilton là dây chuyền đi qua tất cả các đỉnh của đồ thị và đi qua mỗi đỉnh đúng một lần.
(b) Chu trình Hamilton là dây chuyền Hamilton và có một cạnh trong đồ thị nối đỉnh đầu của dây chuyền với đỉnh cuối của nó.
(c) Đồ thị Hamilton là đồ thị có chứa một chu trình Hamilton.

IV.3.2 Vài kết quả liên quan đến đồ thị Hamilton

Không giống như đồ thị Euler, chúng ta chưa có điều kiện cần và đủ để kiểm tra xem một đồ thị có là Hamilton hay không. Các kết quả có được hiện nay chỉ là các điều kiện đủ để một đồ thị là đồ thị Hamilton hay có dây chuyền Hamilton.

(a) Đồ thị đủ luôn là đồ thị Hamilton. Với n lẻ ≥ 3 thì K_n có (n-1)/2 chu trình Hamilton đôi một không có cạnh chung.
(b) Giả sử G là đồ thị lưỡng phân với hai tập đỉnh X_1, X_2 và $|X_1|=|X_2|=n$. Nếu $d(x)\geq n/2$ với mọi đỉnh x của G thì G là đồ thị Hamilton.
(c) Giả sử G là đồ thị vô hướng đơn gồm n đỉnh với $n\geq 3$. Nếu $d(x)\geq n/2$ với mọi đỉnh x của G thì G là đồ thị Hamilton.
(d) Giả sử G là đồ thị vô hướng đơn gồm n đỉnh với $n\geq 3$. Nếu $d(x)\geq (n-1)/2$ với mọi đỉnh x của G thì G có dây chuyền Hamilton.

(e) Giả sử G là đồ thị vô hướng đơn gồm n đỉnh với $n \geq 3$. Nếu $d(x)+d(y) \geq n$ với mọi cặp đỉnh x, y không kề nhau của G thì G là đồ thị Hamilton.

(f) Giả sử G là đồ thị vô hướng đơn gồm n đỉnh và m cạnh. Nếu $m \geq (n^2-3n+6)/2$ thì G là đồ thị Hamilton.

IV.3.3 Qui tắc để tìm dây chuyền Hamilton