CHƯƠNG 5
CÁC BÀI TOÁN ĐƯỜNG ĐI

Bùi Tiến Lên

Đại học Khoa học Tự nhiên TPHCM

1/1/2018

Nội dung

ĐƯỜNG ĐI TRÊN ĐỒ THỊ TỔNG QUÁT

Đường đi là gì

Định nghĩa 5.1

Cho đồ thị $G = (V, E)$, **đường đi** (path) P trong G là một dãy luân phiên các "đỉnh - cạnh"

$$P = v_1 e_1 v_2 e_2 v_3 \ldots v_m$$

sao cho $e_i = (v_i, v_{i+1})$ hoặc

$$P = v_1 v_2 v_3 \ldots v_m$$

Các loại đường đi

- Đường đi sơ cấp là đường đi không có đỉnh lặp lại
- Đường đi đơn là đường đi không có cạnh lặp lại
- Đường đi tổng quát không ràng buộc

Các bài toán đường đi

Bài toán 5.1

- **Bài toán tìm một đường đi sơ cấp**: Cho đồ thị $G = (V, E)$ và hai đỉnh s và t. Hãy tìm **đường đi sơ cấp** đi từ s cho đến t
- **Bài toán tìm tất cả đường đi sơ cấp**: Cho đồ thị $G = (V, E)$ và hai đỉnh s và t. Hãy tìm tất cả **đường đi sơ cấp** từ s cho đến t
- **Bài toán tìm tất cả đường đi đơn**: Cho đồ thị $G = (V, E)$ và hai đỉnh s và t. Hãy tìm **đường đi đơn** từ s cho đến t
- **Bài toán tìm một đường đi có chiều dài cho trước**: Cho đồ thị $G = (V, E)$ và hai đỉnh s và t và một số dương k. Hãy tìm **đường đi** có chiều dài k đi từ s cho đến t

Các bài toán chu trình

Bài toán 5.2

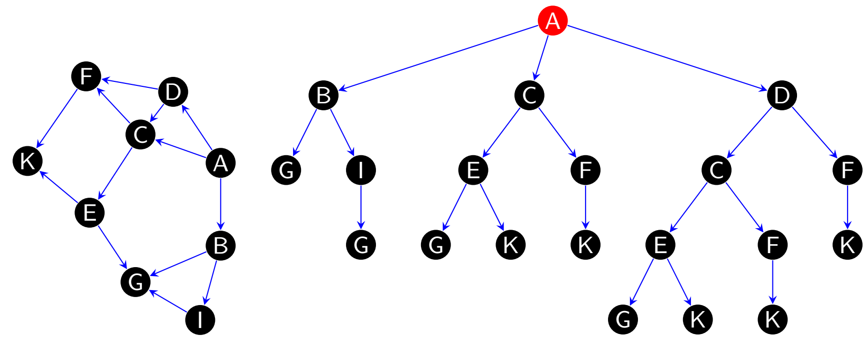
- **Bài toán tìm chu trình sơ cấp**: Cho đồ thị $G = (V, E)$ và hai s. Hãy tìm **chu trình sơ cấp** đi qua s.
- **Bài toán tìm tất cả chu trình sơ cấp**: Cho đồ thị $G = (V, E)$ và hai s. Hãy tìm tất cả **chu trình sơ cấp** đi qua s.

Cây đường đi sơ cấp

Định nghĩa 5.2

Cho đồ thị $G = (V, E)$ và đỉnh s, cây đường đi sơ cấp bắt đầu từ đỉnh s sẽ liệt kê tất cả đường đi sơ cấp từ s tới các đỉnh của đồ thị

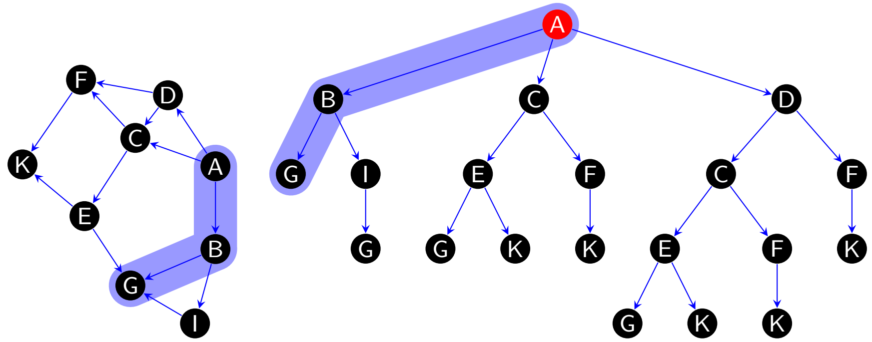
Cây đường đi sơ cấp (cont.)



(a) đồ thị **(b)** cây đường đi bắt đầu từ đỉnh A

Hình 5.1: Đồ thị và cây đường đi

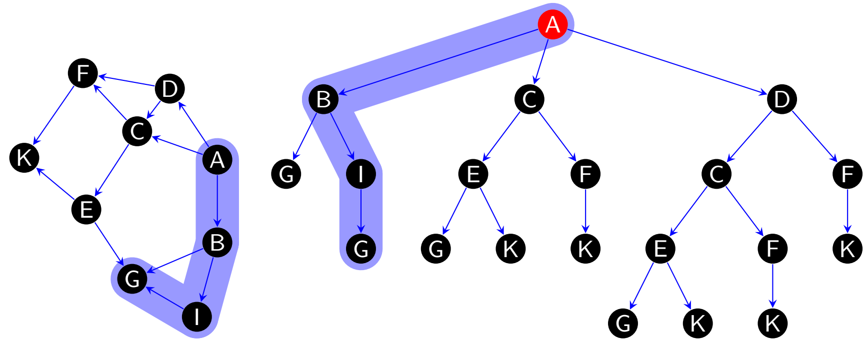
Cây đường đi sơ cấp (cont.)



(a) đường đi trên đồ thị **(b)** đường đi trên cây đường đi

Hình 5.2: Đường đi từ A đến G

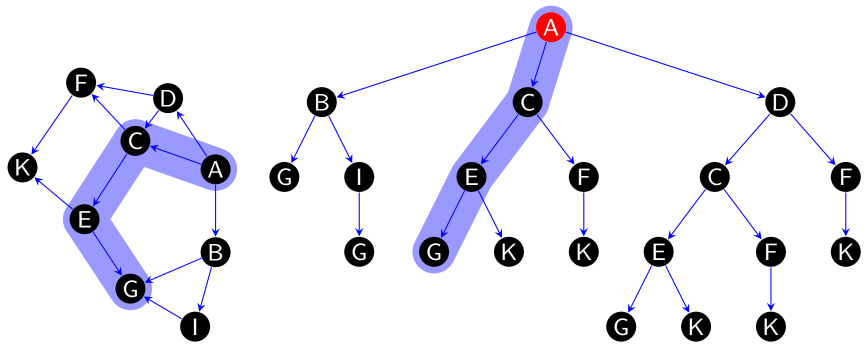
Cây đường đi sơ cấp (cont.)



(a) đường đi trên đồ thị **(b)** đường đi trên cây đường đi

Hình 5.3: Đường đi từ A đến G

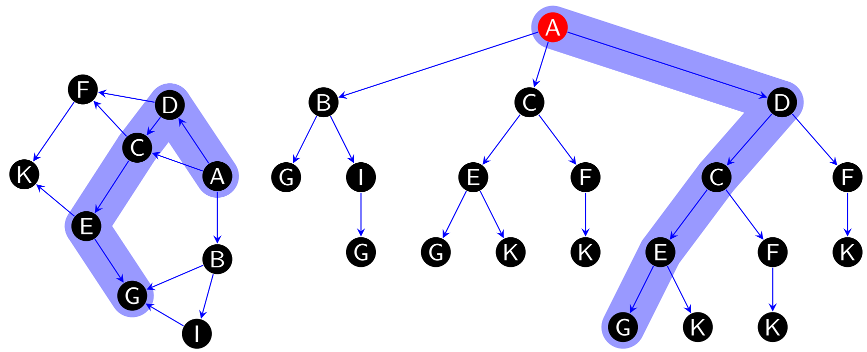
Cây đường đi sơ cấp (cont.)



(a) đường đi trên đồ thị **(b)** đường đi trên cây đường đi

Hình 5.4: Đường đi từ A đến G

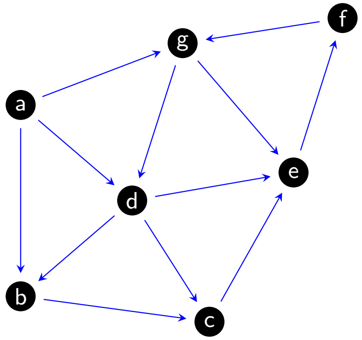
Cây đường đi sơ cấp (cont.)



(a) đường đi trên đồ thị **(b)** đường đi trên cây đường đi

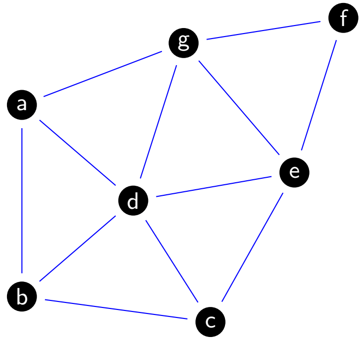
Hình 5.5: Đường đi từ A đến G

Bài tập minh họa



Hình 5.6: Hãy vẽ cây đường đi sơ cấp cho đồ thị có hướng trên bắt đầu từ đỉnh a

Bài tập minh họa (cont.)



Hình 5.7: Hãy vẽ cây đường đi sơ cấp cho đồ thị vô hướng trên bắt đầu từ đỉnh a

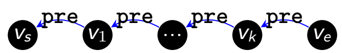
DFS tìm đường đi sơ cấp

Algorithm 1 Tìm đường đi P từ đỉnh v_s đến v_e

```
function DFS_FIND_PATH(v_s, v_e)
    Duyệt v_s
    if v_s == v_e then
        return true
    for mỗi đỉnh v kề với đỉnh v_s do
        if v chưa được duyệt then
            pre[v] = v_s
            if DFS_FIND_PATH(v, v_e) then
                return true
    return false
```


DFS tìm đường đi sơ cấp (cont.)

- Trong hàm trên đã sử dụng kỹ thuật lưu vết của đường đi thông qua việc lưu lại đỉnh trước của đỉnh v bằng phép gán $pre[v] = v_s$
- Để xác định đường đi ta sử dụng cách lần ngược từ đỉnh v_e cho đến v_s



DFS tìm tất cả đường đi sơ cấp

Algorithm 2 Tìm tất cả đường đi từ đỉnh v_s đến v_e

procedure $\text{DFS\_FIND\_ALL\_PATHS}(v_s, v_e)$
    Duyệt v_s
    if $v_s == v_e$ **then**
        In ra đường đi
    for mỗi đỉnh v kề với đỉnh v_s **do**
        if v chưa được duyệt **then**
            $\text{PUSH}(pre[v])$
            $pre[v] = v_s$
            $\text{DFS\_FIND\_ALL\_PATHS}(v, v_e)$
            $\text{POP}(pre[v])$
    v_s trở lại trạng thái chưa duyệt

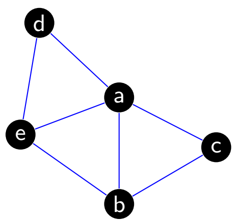
BFS tìm đường đi sơ cấp

Algorithm 3 Tìm đường đi từ đỉnh v_s đến v_e

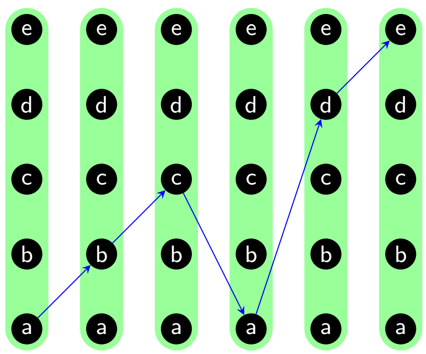
procedure $\text{BFS\_FIND\_PATH}(v_s, v_e)$
    $queue \leftarrow v_s$
    while $queue \neq \emptyset$ **do**
        $v \leftarrow queue$
        Duyệt đỉnh v
        if $v == v_e$ **then**
            In ra đường và kết thúc
        for mỗi đỉnh u kề với đỉnh v **do**
            if đỉnh u chưa duyệt và không có trong *queue* **then**
                $pre[u] = v$
                $queue \leftarrow u$

Đồ thị đường đi tổng quát

- Đồ thị đường đi tổng quát nhằm mục đích biểu diễn các đường đi không phải sơ cấp hay đơn



Hình 5.8: Đồ thị



Hình 5.9: Đồ thị đường đi a-b-c-a-d-e

ĐƯỜNG ĐI TRÊN ĐỒ THỊ CÓ TRỌNG SỐ

Bài toán tìm đường đi

Đối với đồ thị có trọng số, thường xét đến

- Tìm đường đi ngắn nhất
- Tìm đường đi dài nhất
- Tìm đường đi thỏa mãn những yêu cầu nào đó

Bài toán đường đi ngắn nhất

Bài toán 5.3

Cho đồ thị $G = (V, E, L)$ là một đồ thị có trọng số, hai đỉnh s và t, tập hợp $\mathcal{P}$ là tất cả các đường đi từ s đến t. Bài toán tìm **đường đi ngắn nhất** (**shortest path**) từ s đến t có thể phát biểu qua công thức sau

$$P_{min} = \arg\min_{P \in \mathcal{P}}(P) \tag{5.1}$$

Bài toán đường đi ngắn nhất (cont.)

Một số lưu ý

- Các thuật toán tìm đường đi ngắn nhất trên đồ thị là các thuật toán tối ưu rời rạc
- Các thuật toán nói chung đều có thể áp dụng cho cả đồ thị có hướng và vô hướng
- Khi giải bài toán đường đi ngắn nhất
  - Bỏ đi các cạnh song song và chỉ giữ lại cạnh có trọng số nhỏ nhất
  - Bỏ đi các cạnh khuyên và chỉ giữ lại cạnh khuyên có trọng số âm bé nhất

Nguyên lý Bellman

Nguyên lý

- P là đường đi từ đỉnh s đến đỉnh t
- P_1 và P_2, đường đi con của P từ đỉnh s đến k và từ k đến t với k là một đỉnh nằm trên đường đi P

Nếu P là đường đi ngắn nhất **thì** P_1 và P_2 cũng là những đường đi ngắn nhất.

Hình 5.10: Nguyên lý Bellman

Nguyên lý Bellman (cont.)

Chứng minh

- Giả sử tồn tại một đường đi P_1' từ s đến k ngắn hơn đường đi P_1. Nghĩa là
$$L(P_1') < L(P_1)$$
- Suy ra
$$L(P_1' \oplus P_2) < L(P_1 \oplus P_2) = L(P)$$
- Trái với giả thiết P là con đường ngắn nhất để đi từ s cho đến t

■

Nguyên lý Bellman (cont.)

Lưu ý

- Nguyên lý Bellman không có phát biểu ngược lại
- Các đường đi P, P_1, P_2 là những đường đi bất kỳ

Các thuật toán tìm đường đi ngắn nhất

- Thuật toán cây đường đi ngắn nhất
- Thuật toán Dijkstra
- Thuật toán A*
- Thuật toán Bellman
- Thuật toán Floyd

Thuật toán cây đường đi ngắn nhất

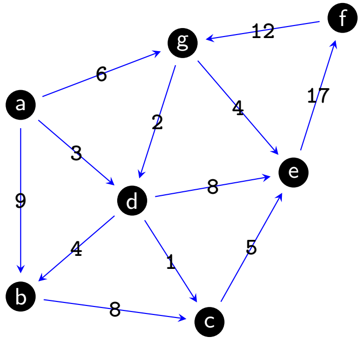
Cho **đồ thị có trọng số không âm** $G = (V, E, L)$ với n đỉnh. Hãy xây dựng cây **đường đi sơ cấp ngắn nhất** $T = (V, E, D)$ bắt đầu từ đỉnh s đến các đỉnh trong đồ thị

Algorithm 4 Thuật toán cây đường đi ngắn nhất

- **Bước 1**: Khởi tạo $V_T = \{s\}, E_T = \emptyset, d(s) = 0$
- **Bước 2**: Chọn đỉnh $y, y \in V_G - V_T$ sao cho $d(x) + l(x, y), x \in X_T$ nhỏ nhất
$$V_T = V_T + \{y\}$$
$$E_T = E_T + \{(x, y)\}$$
$$d(y) = d(x) + l(x, y)$$
- **Bước 3**: Nếu không tìm được đỉnh y nào thì DỪNG ngược lại quay lại bước 2

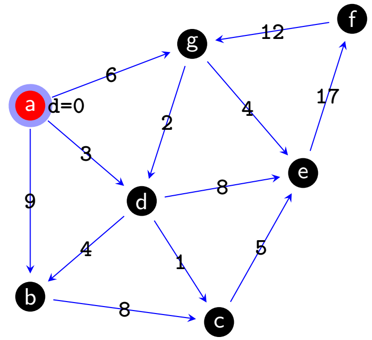
Minh họa thuật toán

Áp dụng thuật toán cây đường đi ngắn nhất để tìm đường đi ngắn từ đỉnh a đến các đỉnh còn lại



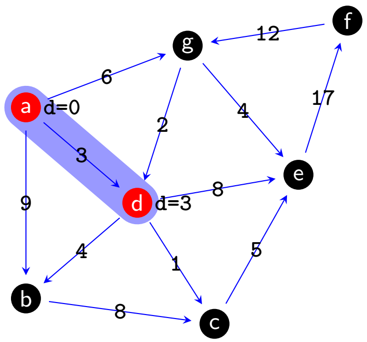
Hình 5.11: Tìm các đường đi ngắn nhất từ đỉnh a đến các đỉnh còn lại

Minh họa thuật toán (cont.)



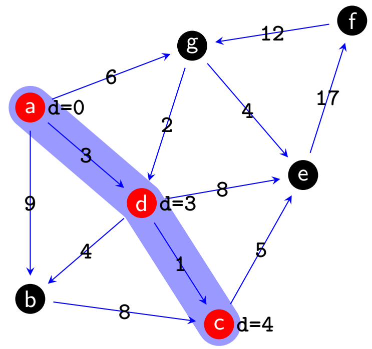
Hình 5.12: Đỉnh đầu tiên a

Minh họa thuật toán (cont.)



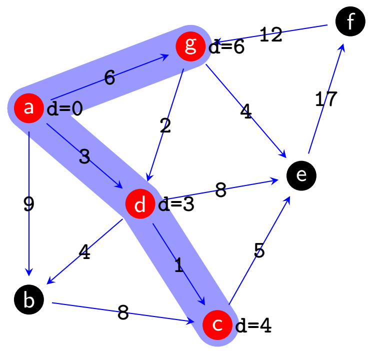
Hình 5.13: Thêm đỉnh d và cạnh ad

Minh họa thuật toán (cont.)



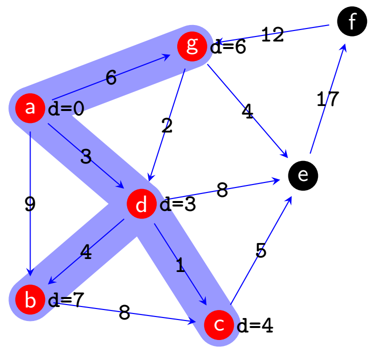
Hình 5.14: Thêm đỉnh c và cạnh dc

Minh họa thuật toán (cont.)



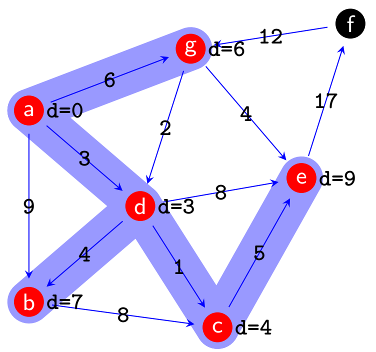
Hình 5.15: Thêm đỉnh g và cạnh ag

Minh họa thuật toán (cont.)



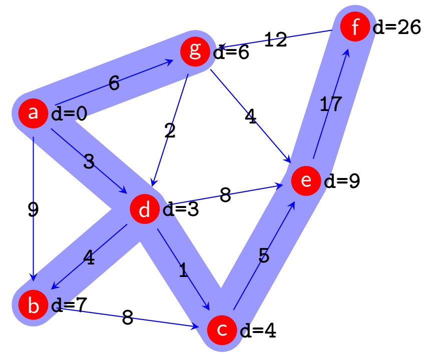
Hình 5.16: Thêm đỉnh d và cạnh db

Minh họa thuật toán (cont.)



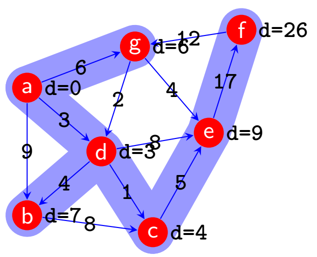
Hình 5.17: Thêm đỉnh e và cạnh ce

Minh họa thuật toán (cont.)



Hình 5.18: Thêm đỉnh f và cạnh ef

Minh họa thuật toán (cont.)



Hình 5.19: Cây đường đi ngắn nhất từ đỉnh a

Vậy đường đi ngắn nhất

- Từ a đến d: a d với trọng số 3
- Từ a đến c: a d c với trọng số 4
- Từ a đến g: a g với trọng số 6
- Từ a đến b: a d b với trọng số 7
- Từ a đến e: a d c e với trọng số 9
- Từ a đến f: a d c e f với trọng số 26

Thuật toán Dijkstra

Cho **đồ thị có trọng số không âm** $G = (V, E, L)$ với n đỉnh. Hãy tìm **đường đi sơ cấp ngắn nhất** từ đỉnh s đến các đỉnh

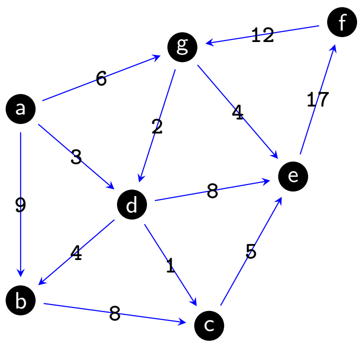
Algorithm 5 Thuật toán Dijkstra

- **bước 1**: $d(s) \leftarrow 0$ và $d(x) \leftarrow \infty, \forall x \neq s$
- **bước 2**: $T \leftarrow \emptyset$
- **bước 3**: Lặp nếu còn đỉnh
  - Chọn đỉnh $y, y \notin T$ sao cho $d(y)$ nhỏ nhất
  - Cập nhật T: $T \leftarrow T + \{y\}$
  - Cập nhật các giá trị d cho các đỉnh còn lại

$$\forall x \notin T, d(x) > d(y) + l(y, x) \Rightarrow d(x) \leftarrow d(y) + l(y, x)$$

Minh họa thuật toán Dijkstra

Áp dụng thuật toán Dijkstra để tìm đường đi ngắn từ đỉnh a đến các đỉnh còn lại

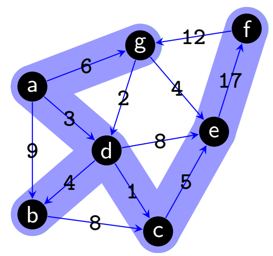


Hình 5.20: Tìm các đường đi ngắn nhất từ đỉnh a đến các đỉnh còn lại

Minh họa thuật toán Dijkstra

Bảng 5.1: Bảng tính trọng số đường đi từ 1 đến các đỉnh

d(a)	d(b)	d(c)	d(d)	d(e)	d(f)	d(g)
0	∞	∞	∞	∞	∞	∞
	9	∞	3	∞	∞	6
	7	4		11	∞	6
	7			9	∞	6
	7			9	∞	
				9	∞	
					26	



Hình 5.21: Đồ thị và các đỉnh được chọn

Thuật toán Dijkstra cập nhật

Để xác định đường đi ta cần một biến $prev(x)$ để lưu lại thông tin đỉnh trước của đỉnh x trong đường đi

Algorithm 6 Thuật toán Dijkstra cập nhật

- **bước 1**: $d(s) \leftarrow 0, d(x) \leftarrow \infty, \forall x \neq s$ và $prev(x) \leftarrow \infty, \forall x$
- **bước 2**: $T \leftarrow \emptyset$
- **bước 3**: Lặp nếu còn đỉnh
  - Chọn đỉnh $y, y \notin T$ sao cho $d(y)$ nhỏ nhất
  - Cập nhật T: $T \leftarrow T + \{y\}$
  - Cập nhật các giá trị d và $prev$ cho các đỉnh còn lại

$$\forall x \notin T, d(x) > d(y) + l(y,x) \Rightarrow \begin{cases} d(x) \leftarrow d(y) + l(y,x) \\ prev(x) \leftarrow y \end{cases}$$

Minh họa thuật toán Dijkstra cập nhật

Bảng 5.2: Bảng xác định trọng số & đỉnh trước trong đường đi

a	b	c	d	e	f	g
(0;null)	(∞;null)	(∞;null)	(∞;null)	(∞;null)	(∞;null)	(∞;null)
	(9;a)	(∞;null)	**(3;a)**	(∞;null)	(∞;null)	(6;a)
	(7;d)	**(4;d)**		(11;d)	(∞;null)	(6;a)
	(7;d)			(9;c)	(∞;null)	**(6;a)**
	(7;d)			(9;c)	(∞;null)	
				(9;c)	(∞;null)	
					(26;e)	

Cài đặt Dijkstra bằng hàng đợi ưu tiên

Vấn đề

- Trong thực tế, đồ thị có số đỉnh rất lớn
- Do đó thuật toán Dijkstra nên được cài đặt bằng hàng đợi ưu tiên

Định nghĩa 5.3

Hàng đợi ưu tiên (priority queue) là một hàng đợi trong đó mỗi phần tử được gắn với một con số được gọi là độ ưu tiên

- Độ ưu tiên sẽ do ứng dụng xác định
- Việc lấy một phần tử ra khỏi hàng đợi sẽ được dựa trên độ ưu tiên và quy tắc FIFO. Nghĩa là phần tử nào có độ ưu tiên cao nhất sẽ được lấy ra trước nhất. Trong trường hợp có nhiều phần tử có cùng độ ưu tiên thì sử dụng quy tắc FIFO

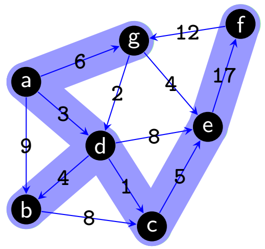
Cài đặt Dijkstra bằng hàng đợi ưu tiên (cont.)

Algorithm 7 Thuật toán Dijsktra

procedure DIJKSTRA_FIND_PATH(v_s,v_e)
  $priority\_queue \leftarrow v_s$ (với $v_s.d = 0$ và $v_s.prev = null$)
  while $priority\_queue \neq \emptyset$ **do**
    $v \leftarrow priority\_queue$
    Duyệt đỉnh v
    if $v = v_e$ **then**
      In ra đường và kết thúc
    for mỗi đỉnh u kề với đỉnh v **do**
      if đỉnh u chưa duyệt **then**
        if $u \in priority\_queue$ **then**
          cập nhật $u.d$ và $u.pre$ nếu tốt hơn
        else
          $u.pre \leftarrow v$ và $u.d \leftarrow v.d + l(v,u)$
          $priority\_queue \leftarrow u$

Minh họa

Hình 5.22: Tìm đường đi bắt đầu từ đỉnh a đến các đỉnh còn lại



Bảng 5.3: Bảng tính cho thuật toán Dijkstra sử dụng hàng đợi ưu tiên. Mỗi đỉnh sẽ có hai thuộc tính *d* độ dài tính từ đỉnh xuất phát (dùng làm độ ưu tiên) và *pre* đỉnh trước của đỉnh này

v	*priority_queue*
	a(0;null)
a(0;null)	b(9;a) d(3;a) g(6;a)
d(3;a)	b(7;d) g(6;a) c(4;d) e(11;d)
c(4;d)	b(7;d) g(6;a) e(9;c)
g(6;a)	b(7;d) e(9;c)
b(7;d)	e(9;c)
e(9;c)	f(26;e)
f(26;e)	$\emptyset$

Tìm tất cả đường đi ngắn nhất

- Hiệu chỉnh thuật toán Dijkstra để tìm tất cả đường đi ngắn nhất từ v_s đến v_e. Thay vì chỉ lưu lại một **đỉnh trước** *pre* của v, ta sẽ lưu lại một **danh sách đỉnh trước** *pre_list* của v.

Tìm tất cả đường đi ngắn nhất (cont.)

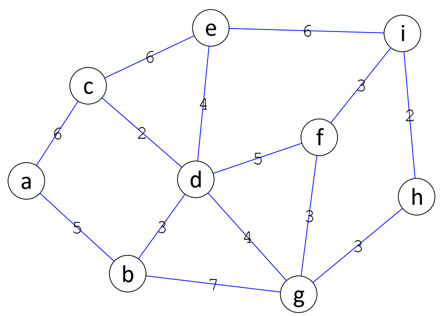
Algorithm 8 Tìm tất cả đường đi ngắn nhất

```
procedure DijkstraFindAllPath(v_s, v_e)
    priority_queue ← v_s (với v_s.d ← 0 và v_s.prev_list ← null)
    while priority_queue ≠ ∅ do
        v ← priority_queue
        Duyệt đỉnh v
        if v = v_e then
            In ra đường và kết thúc
        for mỗi đỉnh u kề với đỉnh v do
            if đỉnh u chưa duyệt then
                if u ∈ priority_queue then
                    if u.d > v.d + l(v, u) then
                        u.d ← v.d + l(v, u)
                        u.pre_list ← v
                    if u.d = v.d + l(v, u) then
                        Add(u.pre_list, v)
                else
                    u.pre_list ← v và u.d ← v.d + l(v, u)
                    priority_queue ← u
```


Ví dụ minh họa

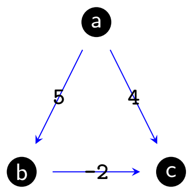
Ví dụ 5.1

Tìm tất cả đường đi ngắn nhất từ đỉnh *a* đến đỉnh *i* trên đồ thị vô hướng dưới

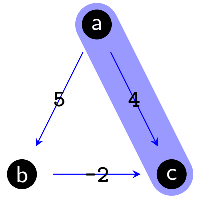


Thuật toán Dijkstra và đồ thị có trọng số âm

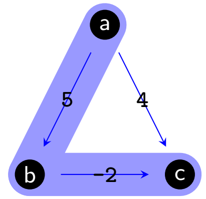
Thuật toán Dijkstra áp dụng cho đồ thị có trọng số âm



Hình 5.23: đồ thị có trọng số âm



Hình 5.24: đường đi ngắn nhất từ a - c theo Dijkstra

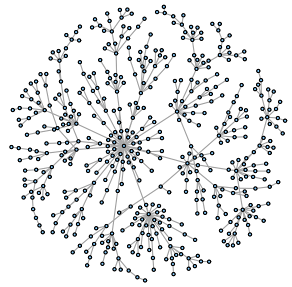


Hình 5.25: đường đi ngắn nhất từ a - c thật sự

Thuật toán A*

Vấn đề

- Thuật toán Dijkstra là một thuật toán vét cạn. Do đó, sẽ gặp rất nhiều khó khăn trong các bài toán có độ phức tạp lớn.



Hình 5.26: Bài toán với đồ thị phức tạp

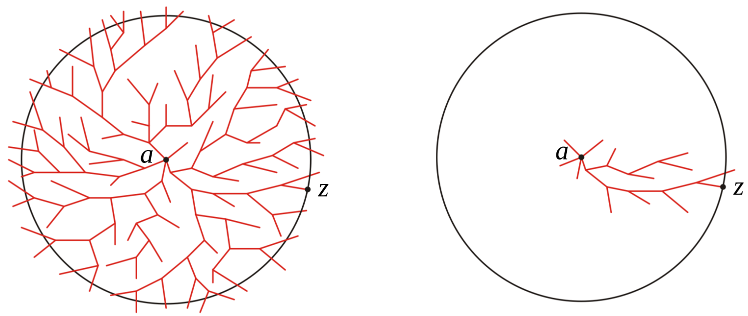
Thuật toán A* (cont.)

Ý tưởng thuật toán A*

- Được Peter Hart, Nils Nilsson, và Bertram Raphael đề xuất vào năm 1968
- Là sự cải tiến đột phá từ thuật toán Dijkstra bằng cách mỗi đỉnh có thêm *thông tin ước lượng h* cho các đỉnh là khoảng cách từ nó đến *đỉnh kết thúc*
- Độ ưu tiên trong hàng đợi sẽ được tính dựa trên d và h. Thông thường là

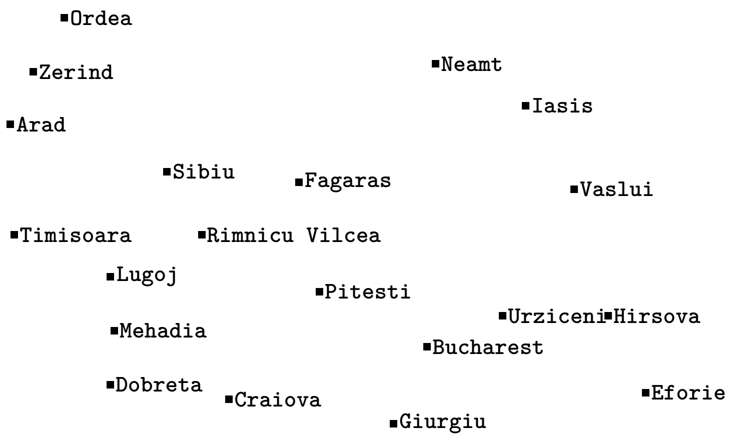
$$d + h \tag{5.2}$$

Thuật toán A* (cont.)



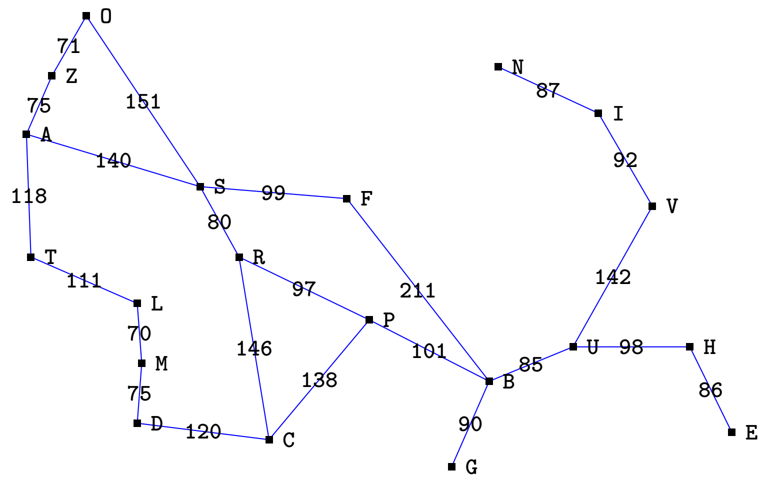
Hình 5.27: Tìm theo mọi hướng vs tìm với định hướng

Minh họa thuật toán A*



Hình 5.28: Các thành phố của Romania

Minh họa thuật toán A*



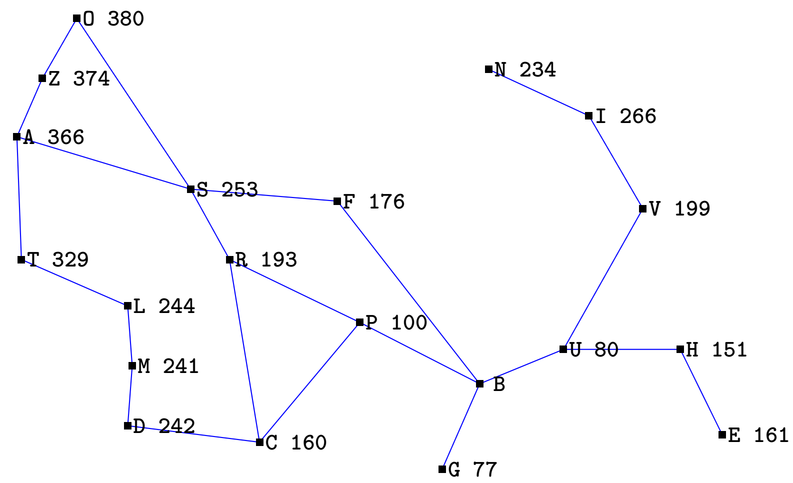
Hình 5.29: Bản đồ đường bộ

Minh họa thuật toán A*

Bảng 5.4: Khoảng cách theo đường chim bay từ các thành phố đến thành phố Bucharest

thành phố	h	thành phố	h
Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Dobreta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugo	244	Zerind	374

Minh họa thuật toán A*



Thuật toán Bellman

Thuật toán Bellman là **thuật toán qui hoạch động** (dynamic algorithm) sử dụng nguyên lý Bellman để tìm **đường đi** ngắn nhất. Cho đồ thị có trọng số bất kỳ $G = (V, E, L)$ với n đỉnh. Ý tưởng của thuật toán được thể hiện qua hàm đệ qui $\mathcal{P}$

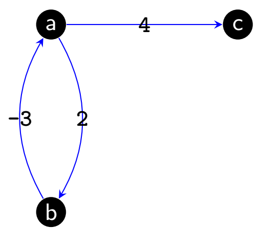
- Hàm $\mathcal{P}_k(t)$ là hàm trả về đường đi ngắn nhất từ đỉnh s đến đỉnh t và đi qua tối đa k đỉnh không tính đỉnh đầu
- Hàm $\mathcal{P}_{k+1}(t)$ là hàm trả về đường đi ngắn nhất đi từ đỉnh s đến đỉnh t và đi qua tối đa $k+1$ đỉnh không tính đỉnh đầu.
- Hàm $\mathcal{P}_k(t)$ được định nghĩa đệ qui như sau

$$\begin{cases} \mathcal{P}_0(t) = \begin{cases} 0 & t = s \\ \infty & t \neq s \end{cases} \\ \mathcal{P}_{k+1}(t) = \min\left(\mathcal{P}_k(t), \min\left(\mathcal{P}_k(v) + l(v,t), \forall v\right)\right) \end{cases} \tag{5.3}$$

Minh họa thuật toán Bellman

Ví dụ 5.2

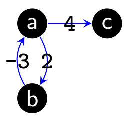
Áp dụng thuật toán Bellman tìm đường đi ngắn nhất cho đồ thị sau từ đỉnh a. Lưu ý đồ thị có mạch âm



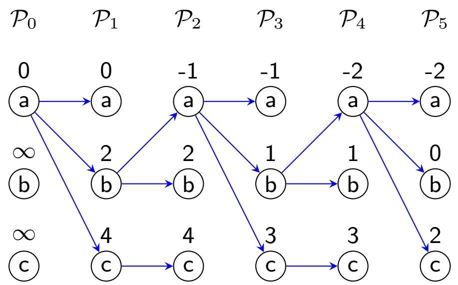
Hình 5.30: Đồ thị có trọng số âm 3 đỉnh

Minh họa thuật toán Bellman (cont.)

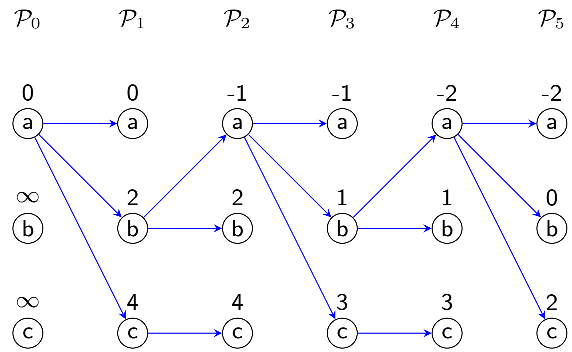
Bảng 5.5: Đồ thị, ma trận trọng số và bảng tính



	a	b	c
a	0	2	4
b	-3	0	∞
c	∞	∞	0



Minh họa thuật toán Bellman (cont.)



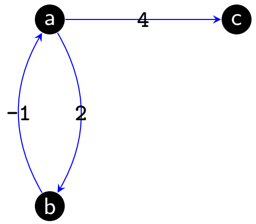
Vậy đường đi ngắn nhất từ a đi qua tối đa 6 đỉnh

- đến đỉnh c: a b a b a c có trọng số là 2
- đến đỉnh b: a b a b a b có trọng số là 0
- đến đỉnh a: a b a b a có trọng số là -2

Minh họa thuật toán Bellman

Ví dụ 5.3

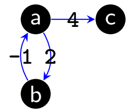
Áp dụng thuật toán Bellman tìm đường đi ngắn nhất cho đồ thị sau từ đỉnh a. Lưu ý đồ thị không có mạch âm



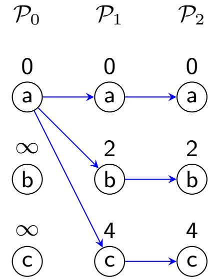
Hình 5.31: Đồ thị có trọng số âm 3 đỉnh

Minh họa thuật toán Bellman (cont.)

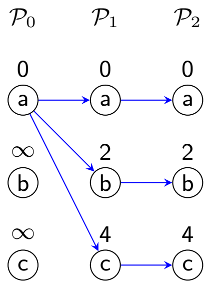
Bảng 5.6: Đồ thị, ma trận trọng số và bảng tính



$$\begin{array}{cccc} & a & b & c \\ a & 0 & 2 & 4 \\ b & -1 & 0 & \infty \\ c & \infty & \infty & 0 \end{array}$$



Minh họa thuật toán Bellman (cont.)



Vậy đường đi ngắn nhất từ a

- đến đỉnh c: a c có trọng số là 4
- đến đỉnh b: a b có trọng số là 2
- đến đỉnh a: a là 0

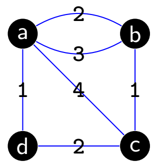
Nhận xét về thuật toán Bellman

Nhận xét

- Nếu trong đồ thị tồn tại **mạch âm** thì trọng số đường đi sẽ càng lúc càng giảm
- Điều kiện để dừng thuật toán Bellman
  - Nếu P_k và P_{k+1} là hoàn toàn giống nhau
  - Nếu điều kiện trên không xảy ra nghĩa là trong đồ thị tồn tại **những mạch âm** thì việc dừng thuật toán tại bước lặp k với đường đi P_k được hiểu là một lời giải cho đường đi ngắn nhất đi qua tối đa k đỉnh không kể đỉnh bắt đầu

Bài tập

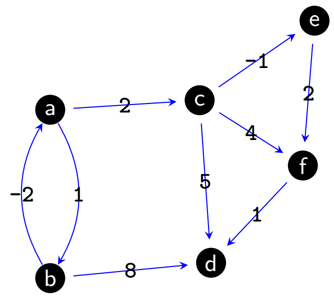
- Áp dụng thuật toán Bellman tìm đường đi ngắn nhất cho đồ thị sau bắt đầu từ đỉnh a



Hình 5.32: Đồ thị có trọng số không âm 4 đỉnh

Bài tập (cont.)

- Áp dụng thuật toán Bellman tìm đường đi ngắn nhất cho đồ thị sau bắt đầu từ đỉnh a



Hình 5.33: Đồ thị có trọng số âm 6 đỉnh

Thuật toán Floyd

Thuật toán Floyd cũng là một thuật toán qui hoạch động dùng để tìm trọng số đường đi ngắn nhất cho tất cả các cặp đỉnh của đồ thị có trọng số $G = (V, E, L)$ có n đỉnh $\{1, ..., n\}$. Ý tưởng của thuật toán được trình bày qua hàm đệ qui $\mathcal{D}$ như sau

- Hàm $\mathcal{D}_k(i, j)$ là hàm trả về đường đi ngắn nhất từ đỉnh i đến đỉnh j sử dụng các đỉnh trung gian $\{1, ..., k\}$
- Hàm $\mathcal{D}_{k+1}(i, j)$ là hàm trả về đường đi ngắn nhất từ đỉnh i đến đỉnh j sử dụng các đỉnh trung gian $\{1, ..., k, k+1\}$
- Hàm $\mathcal{D}_k(i, j)$ được định nghĩa đệ qui như sau

$$\begin{cases} \mathcal{D}_0(i,j) = l(i,j) \\ \mathcal{D}_{k+1}(i,j) = \min\left(\mathcal{D}(i,j), \mathcal{D}_k(i,k+1) + \mathcal{D}_k(k+1,j)\right) \end{cases} \tag{5.4}$$

Thuật toán Floyd (cont.)

Sau đây là cài đặt thuật toán Floyd với

- biến n là số đỉnh của đồ thị
- biến l, d và pre lưu ma trận trọng số, thông tin độ dài quãng đường ngắng nhất và vết đường đi

```
for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
    {
        d[i][j] = l[i][j];
        if (d[i][j] == INFINITY)
            pre[i][j] = NO;
        else
            pre[i][j] = i;
    }
for (k = 0; k < n; k++)
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
        if (d[i][j] > d[i][k] + d[k][j])
```

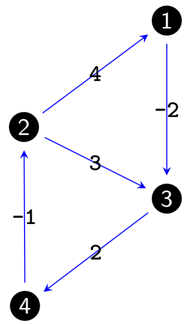
Thuật toán Floyd (cont.)

```
        {
            d[i][j] = d[i][k] + d[k][j];
            pre[i][j] = pre[k][j];
        }
```


Minh họa thuật toán Floyd

Ví dụ 5.4

Hãy áp dụng thuật toán Floyd cho đồ thị dưới đây



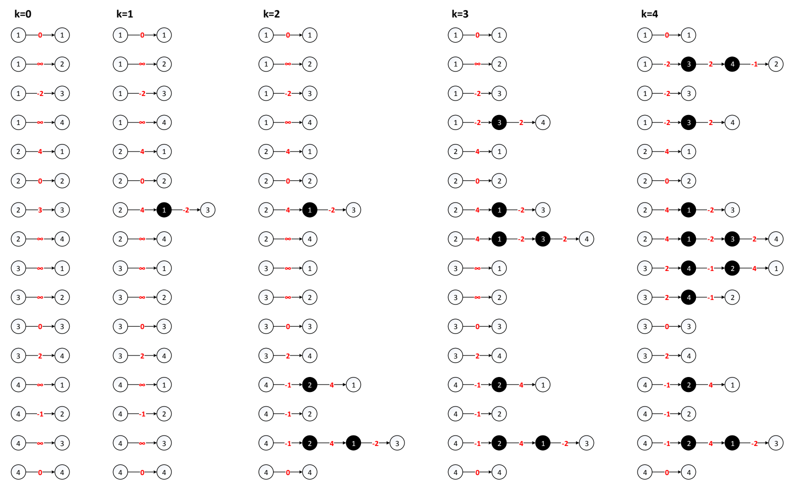
Hình 5.34: Đồ thị có trọng số âm

Minh họa thuật toán Floyd (cont.)

Ma trận trọng số của đồ thị là

$$L = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} 0 & \infty & -2 & \infty \\ 4 & 0 & 3 & \infty \\ \infty & \infty & 0 & 2 \\ \infty & -1 & \infty & 0 \end{pmatrix} \end{array}$$

Minh họa thuật toán Floyd (cont.)



Minh họa thuật toán Floyd (cont.)

- Khi $k = 0$ các đường đi khởi tạo: 1 3, 2 1, 2 3, 3 4, 4 2
- Khi $k = 1$ cập nhật đường đi: 2 1 3
- Khi $k = 2$ cập nhật đường đi: 4 2 1, 4 2 1 3
- Khi $k = 3$ cập nhật đường đi: 1 3 4, 2 1 3 4
- Khi $k = 4$ cập nhật đường đi: 1 3 4 2, 3 4 2 1, 3 4 2

MỘT SỐ KHÁI NIỆM LIÊN QUAN ĐẾN ĐƯỜNG ĐI

Ma trận khoảng cách

Định nghĩa 5.4

Gọi G là một đồ thị có trọng số không âm có n đỉnh $v_1, v_2, ..., v_n$, **ma trận khoảng cách** (distance matrix) d là ma trận vuông cấp n với

$$d(v_i, v_j) = \text{trọng số đường đi ngắn nhất từ } v_i \text{ đến } v_j$$

Lưu ý

Ma trận khoảng cách cho đồ thị vô hướng là một trận đối xứng

Một số khái niệm

Định nghĩa 5.5

Cho đồ thị có trọng số G, **độ lệch** (eccentricity) của đỉnh v

$$\epsilon(v) = \max\{d(v, u) | \forall u \in V, u \neq v\}$$

Định nghĩa 5.6

Cho đồ thị có trọng số G, **tâm** (center) của đồ thị là

$$center(G) = \arg\min_{v \in V}(\epsilon(v))$$

Một số khái niệm (cont.)

Định nghĩa 5.7

Cho đồ thị có trọng số G, **bán kính** (radius) của đồ thị là

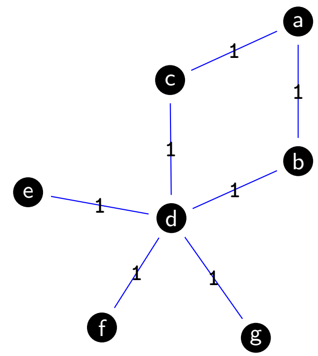
$$rad(G) = \min\{\epsilon(v))|\forall v \in V\}$$

Định nghĩa 5.8

Cho đồ thị có trọng số G, **đường kính** (diameter) của đồ thị là

$$diam(G) = \max\{\epsilon(v))|\forall v \in V\}$$

Ví dụ



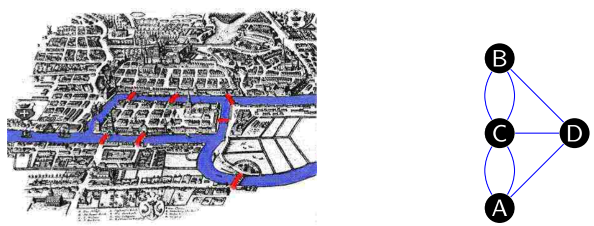
Hình 5.35: Xác định độ lệch, tâm, bán kính và đường kính

ĐƯỜNG ĐI CÓ RÀNG BUỘC

Bài toán về đồ thị Euler

Lịch sử

Bài toán này có nguồn gốc từ bài toán dân gian. Làm sao đi qua 7 chiếc cầu đúng một lần và trở về nơi xuất phát. Bài toán này đã được nhà bác học Euler giải quyết trọn vẹn vào năm 1736.



(a) bảy cây cầu **(b)** biểu diễn đồ thị

Hình 5.36: Bài toán bảy cây cầu

Định nghĩa về đồ thị Euler

Định nghĩa về đồ thị Euler (cont.)

Định nghĩa 5.9

Cho một đồ thị $G = (V, E)$

- **Dây chuyền Euler** (Euler path) là dây chuyền đi qua tất cả các cạnh trong đồ thị và mỗi cạnh đi qua đúng một lần
- **Đường đi Euler** (Euler path) là đường đi qua tất cả các cạnh của đồ thị và mỗi cạnh được đi qua đúng một lần
- **Chu trình Euler** (Euler circuit) là dây chuyền Euler có đỉnh đầu trùng với đỉnh cuối
- **Mạch Euler** (Euler circuit) là đường đi Euler có đỉnh đầu trùng với đỉnh cuối
- **Đồ thị Euler vô hướng** (undirected Euler graph) là đồ thị vô hướng có chứa ít nhất một chu trình Euler
- **Đồ thị Euler có hướng** (directed Euler graph) là đồ thị có hướng chứa ít nhất một chu trình Euler

Định lý về chu trình Euler

Định lý 5.1

Một đa đồ thị vô hướng liên thông $G = (V, E)$ có chu trình Euler khi và chỉ khi mỗi đỉnh của nó đều có bậc chẵn

Chứng minh

Sinh viên tự chứng minh ■

Thuật toán Hierholzer tìm chu trình Euler

Cho một đồ thị liên thông $G = (V, E)$ có tất cả các đỉnh bậc chẵn. Thuật toán Hierholzer thực hiện theo nguyên lý tắc sau

- Từ đỉnh v tìm một chu trình C trên đồ thị G
- Nếu C không phải chu trình Euler thì chu trình C sẽ có một đỉnh u có một cạnh không thuộc chu trình C
  - Loại bỏ các cạnh của C khỏi đồ thị G
  - Từ đỉnh u tìm một chu trình C' trên đồ thị G
  - Kết hợp chu trình C' vào chu trình C
  - Quay lại kiểm tra C có phải là chu trình Euler hay không?

Cài đặt thuật toán Hierholzer

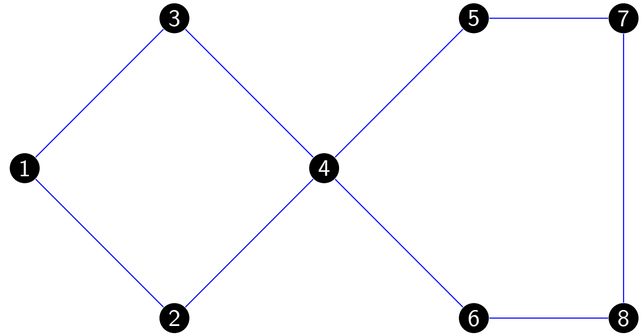
Thuật toán có thể cài đặt bằng stack như sau

Algorithm 9 Tìm chu trình Euler C bắt đầu từ đỉnh *start*

```
procedure FIND_EULER_CIRCLE(G, start, C)
    stack.PUSH(start)
    while stack.NOT_EMPTY do
        v = stack.TOP
        if không còn cạnh kề với v then
            stack.POP
            C = C + {v}
        else
            Lấy cạnh (v, u) đầu tiên kề với đỉnh v
            Xóa cạnh (v, u)
            stack.PUSH(u)
```


Minh họa cài đặt thuật toán Hierholzer

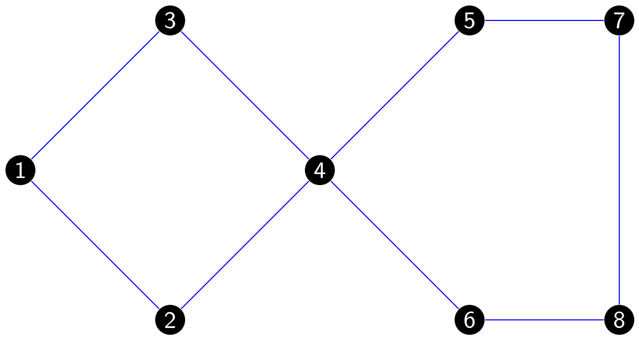
Tìm chu trình Euler của đồ thị dưới đây bắt đầu từ đỉnh 1



Hình 5.37: Đồ thị liên thông có các đỉnh đều bậc chẵn

Minh họa cài đặt thuật toán Hierholzer (cont.)

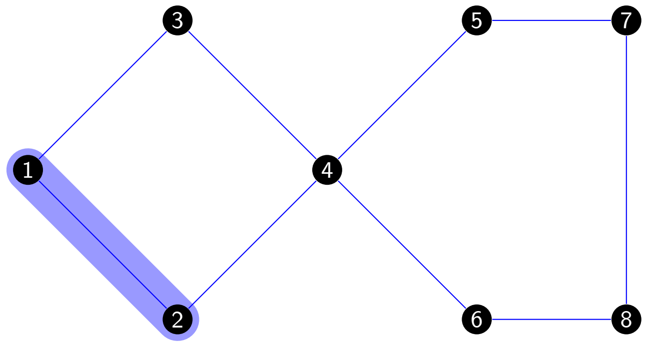
$stack = \{1\}$
$C = \{\emptyset\}$



Hình 5.38: Đưa đỉnh đầu vào *stack*

Minh họa cài đặt thuật toán Hierholzer (cont.)

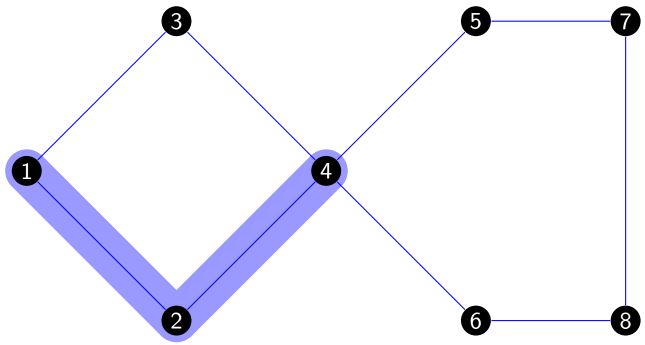
$stack = \{1, 2\}$
$C = \{\emptyset\}$



Hình 5.39: Đưa đỉnh 2 vào *stack* và xóa cạnh (1,2)

Minh họa cài đặt thuật toán Hierholzer (cont.)

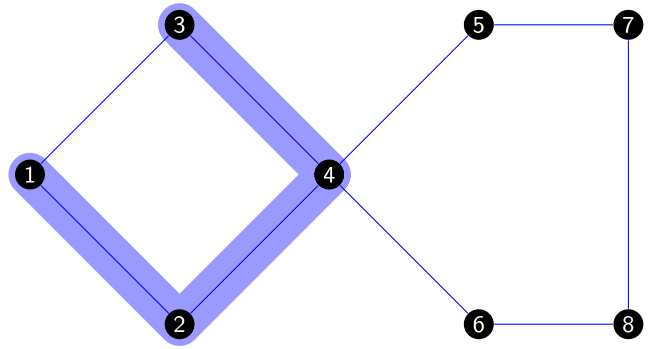
stack $= \{1, 2, 4\}$
$C = \{\emptyset\}$



Hình 5.40: Đưa đỉnh 4 vào *stack* và xóa cạnh (1,4)

Minh họa cài đặt thuật toán Hierholzer (cont.)

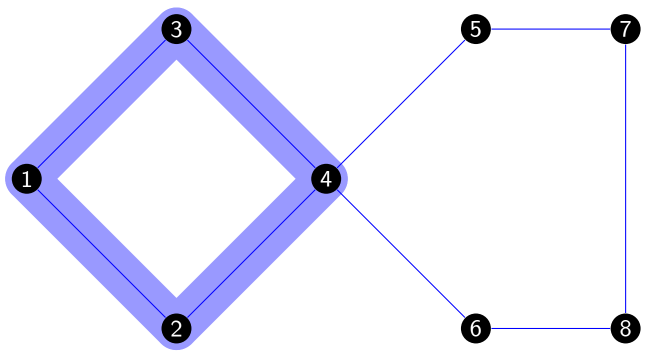
stack $= \{1, 2, 4, 3\}$
$C = \{\emptyset\}$



Hình 5.41: Đưa đỉnh 3 vào *stack* và xóa cạnh (4,3)

Minh họa cài đặt thuật toán Hierholzer (cont.)

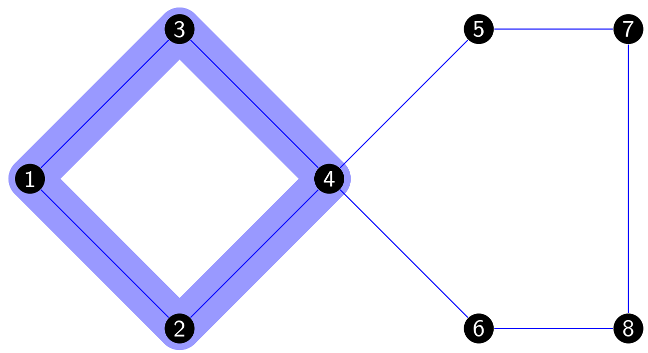
stack $= \{1, 2, 4, 3, 1\}$
$C = \{\emptyset\}$



Hình 5.42: Đưa đỉnh 1 vào *stack* và xóa cạnh (3,1)

Minh họa cài đặt thuật toán Hierholzer (cont.)

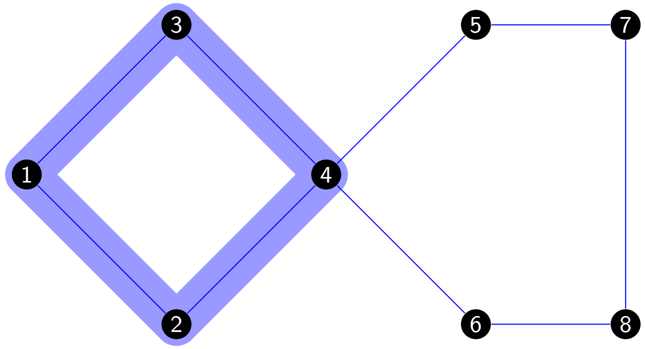
stack $= \{1, 2, 4, 3\}$
$C = \{1\}$



Hình 5.43: Lấy đỉnh 1 ra khỏi *stack* và thêm vào chu trình *C*

Minh họa cài đặt thuật toán Hierholzer (cont.)

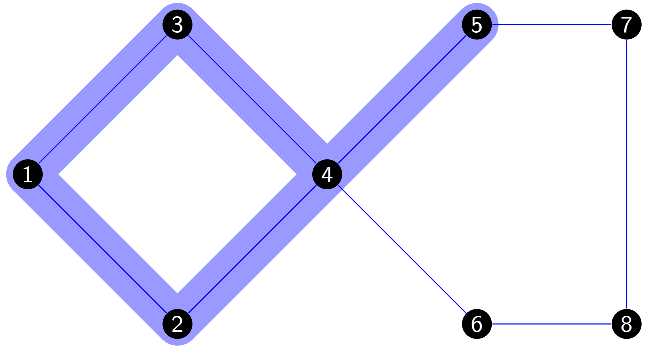
stack $= \{1, 2, 4\}$
$C = \{1, 3\}$



Hình 5.44: Lấy đỉnh 3 ra khỏi *stack* và thêm vào chu trình *C*

Minh họa cài đặt thuật toán Hierholzer (cont.)

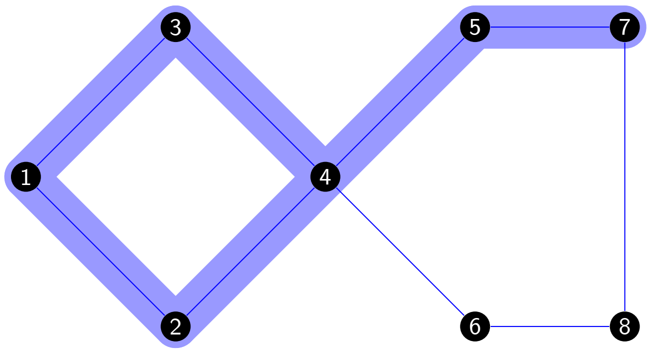
stack $= \{1, 2, 4, 5\}$
$C = \{1, 3\}$



Hình 5.45: Đưa đỉnh 5 vào *stack* và xóa cạnh (4,5)

Minh họa cài đặt thuật toán Hierholzer (cont.)

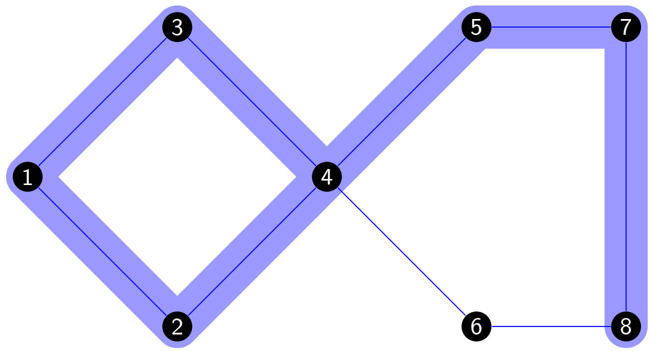
stack $= \{1, 2, 4, 5, 7\}$
$C = \{1, 3\}$



Hình 5.46: Đưa đỉnh 7 vào *stack* và xóa cạnh (5,7)

Minh họa cài đặt thuật toán Hierholzer (cont.)

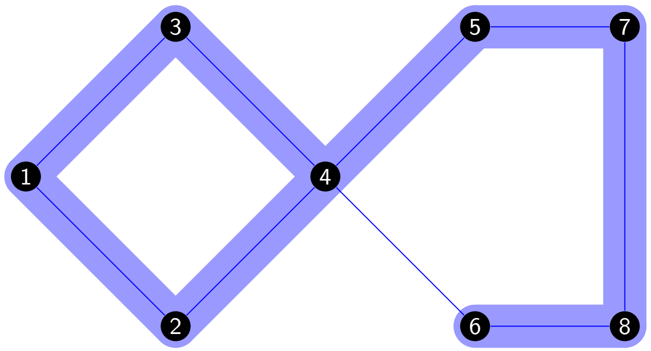
stack $= \{1, 2, 4, 5, 7, 8\}$
$C = \{1, 3\}$



Hình 5.47: Đưa đỉnh 8 vào *stack* và xóa cạnh (7,8)

Minh họa cài đặt thuật toán Hierholzer (cont.)

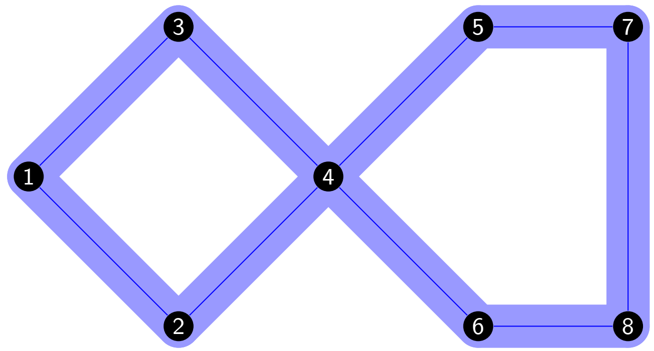
stack $= \{1, 2, 4, 5, 7, 8, 6\}$
$C = \{1, 3\}$



Hình 5.48: Đưa đỉnh 6 vào *stack* và xóa cạnh (8,6)

Minh họa cài đặt thuật toán Hierholzer (cont.)

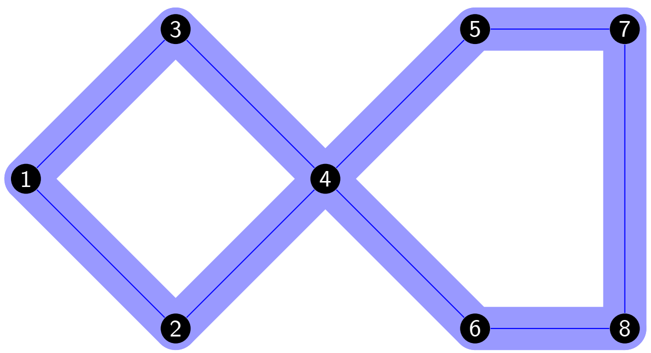
stack $= \{1, 2, 4, 5, 7, 8, 6, 4\}$
$C = \{1, 3\}$



Hình 5.49: Đưa đỉnh 4 vào *stack* và xóa cạnh (6,4)

Minh họa cài đặt thuật toán Hierholzer (cont.)

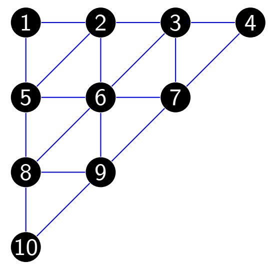
stack $= \{\emptyset\}$
$C = \{1, 3, 4, 6, 8, 7, 5, 4, 2, 1\}$



Hình 5.50: Lần lượt lấy các đỉnh ra khỏi *stack* và thêm vào chu trình *C*

Bài tập

Áp dụng thuật toán Hierholzer tìm chu trình Euler cho đồ thị sau



Hình 5.51: Đồ thị Euler?

Thuật toán Fleury tìm chu trình Euler

Cho một đồ thị liên thông $G = (V, E)$ có tất cả các đỉnh bậc chẵn.

Algorithm 10 Thuật toán Fleury

Thuật toán thực hiện theo hai qui tắc

- **Quy tắc 1**: Mỗi khi đi qua một cạnh thì xóa cạnh đó và xóa đỉnh cô lập (nếu có)
- **Quy tắc 2**: Không đi qua cạnh cầu trừ phi không còn cách nào khác

Định lý về đường đi Euler

Định lý 5.2

Một đa đồ thị vô hướng liên thông $G = (V, E)$ có đường đi Euler khi và chỉ khi đồ thị có đúng 2 đỉnh bậc lẻ

Chứng minh

Sinh viên tự chứng minh ■

Đường đi Euler cho đồ thị có hướng

Định lý 5.3

Một đa đồ thị có hướng liên thông $G = (V, E)$ có chu trình Euler khi và chỉ tại mỗi đỉnh v của đồ thị đều có $d^+(v) = d^-(v)$

Chứng minh

Sinh viên tự chứng minh ■

Định lý 5.4

Một đa đồ thị có hướng liên thông $G = (V, E)$ có đường đi Euler khi và chỉ tại tất cả các đỉnh v của đồ thị đều có $d^+(v) = d^-(v)$ trừ duy nhất 2 đỉnh x, y

$$d^+(x) = d^-(x) + 1, d^+(y) + 1 = d^-(y)$$

Chứng minh

Sinh viên tự chứng minh ■

Bài toán người đưa thư Trung Hoa

Bài toán 5.4

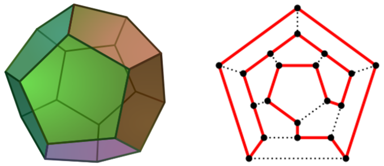
Bài toán người đưa thư Trung Hoa (**Chinese postman problem**) được phát biểu như sau: Cho một đồ thị liên thông tìm một chu trình ngắn nhất đi qua các cạnh

- Nếu đồ thị không có trọng số thì chu trình ngắn nhất là chu trình có số cạnh ít nhất
- Nếu đồ thị có trọng số thì chu trình ngắn nhất là chu trình có trọng số nhỏ nhất

Bài toán về đồ thị Hamilton

Lịch sử

- Bài toán này xuất phát từ trò đố vui do William Rowan Hamilton, nhà toán học người Ailen đưa ra vào năm 1857.
- Giả sử có một khối thập nhị diện đều với mỗi mặt là một ngũ giác đều. Mỗi đỉnh trong 20 đỉnh khối này được đặt tên một thành phố. Hãy tìm một cách đi khép kín ghé thăm 20 thành phố chỉ một lần



Hình 5.52: Khối thập nhị diện đều và chu trình Hamilton

Định nghĩa về đồ thị Hamilton

Định nghĩa 5.10

Cho một đồ thị $G = (V, E)$

- **Dây chuyền Hamilton** (**Hamilton path**) là dây chuyền đi qua tất cả các đỉnh trong đồ thị và mỗi đỉnh đi qua đúng một lần
- **Đường đi Hamilton** (**Hamilton path**) là đường đi qua tất cả các đỉnh của đồ thị và mỗi đỉnh được đi qua đúng một lần
- **Chu trình Hamilton** (**Hamilton circuit**) là dây chuyền Hamilton có đỉnh đầu trùng với đỉnh cuối
- **Mạch Hamilton** (**Hamilton circuit**) là đường đi Hamilton có đỉnh đầu trùng với đỉnh cuối
- **Đồ thị Hamilton** (**Hamilton graph**) là đồ thị có chứa ít nhất một chu trình Hamilton

Các điều kiện cần

1. Đồ thị G có đỉnh treo thì sẽ không có chu trình Hamilton
2. Giả sử G là đồ thị hai phía và tập đỉnh thứ nhất có m đỉnh và tập đỉnh thứ hai có n đỉnh
   - Nếu $m \neq n$ thì đồ thị G không có chu trình Hamilton
   - Nếu m và n khác nhau từ 2 đơn vị trở lên thì đồ thị G không có đường đi Hamilton

Các điều kiện đủ

Định lý 5.5 (Ore, 1960)

Cho một đơn đồ thị liên thông $G = (V, E)$ với số đỉnh $n \geq 3$, nếu $\forall u, v \in V, d(u) + d(v) \geq n$ thì đồ thị có chu trình Hamilton

Chứng minh

Sinh viên tự chứng minh ■

Định lý 5.6 (Dirac, 1952)

Cho một đơn đồ thị liên thông $G = (V, E)$ với số đỉnh $n \geq 3$, nếu $\forall v \in V, d(v) \geq \frac{n}{2}$ thì đồ thị có chu trình Hamilton

Chứng minh

Sinh viên tự chứng minh ■

Các điều kiện đủ (cont.)

Định lý 5.7 (Posa)

Cho một đơn đồ thị liên thông $G = (V, E)$ với số đỉnh $n \geq 3$; giả sử có không quá $k - 1$ đỉnh có bậc không lớn hơn k với $k \in \left[1 ... \frac{n-1}{2}\right)$ và có không quá $\frac{n-1}{2}$ đỉnh có bậc vượt quá $\frac{n-1}{2}$ với n lẻ. Khi đó đồ thị G có một chu trình Hamilton

Chứng minh

Sinh viên tự chứng minh ■

Phương pháp tìm chu trình Hamilton

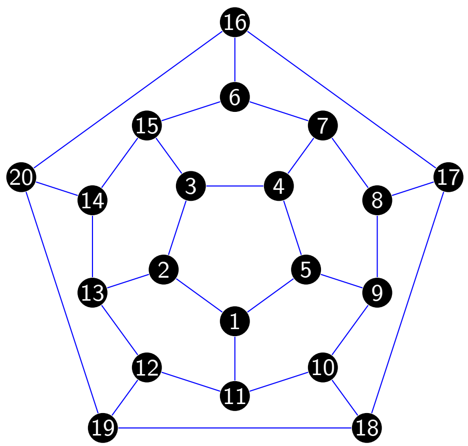
- Thuật toán vét cạn có độ phức tạp lũy thừa

Algorithm 11 Tìm chu trình Hamilton

- **Quy tắc 1**: Nếu đồ thị có đỉnh cô lập hoặc đỉnh treo thì đồ thị không có chu trình Hamilton
- **Quy tắc 2**: Nếu đỉnh v có bậc $d(v) = 2$ thì cả hai cạnh kề với nó đều phải thuộc chu trình Hamilton
- **Quy tắc 3**: Khi hai cạnh có chung một đỉnh v thuộc chu trình Hamilton thì các cạnh kề còn lại của v sẽ không thuộc chu trình Hamilton
- **Quy tắc 4**: Tránh tạo ra chu trình con
- **Quy tắc 5**: Tận dụng tính đối xứng của đồ thị để giảm bớt trường hợp

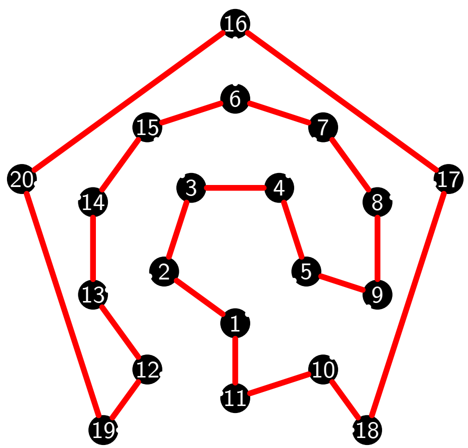
Minh họa

Tìm chu trình Hamilton của đồ thị dưới đây



Hình 5.53: Đồ thị biểu diễn khối thập nhị diện đều

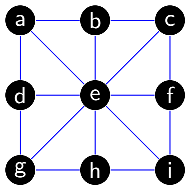
Minh họa



Nhánh	Thực hiện
	Bắt đầu tại đỉnh 1
✓	Chọn đỉnh 2, cạnh (1,2)
✓	Chọn đỉnh 3, cạnh (2,3)
	xóa cạnh (2,13)
	chọn cạnh (12, 13), (13,14)
✓	Chọn đỉnh 4, cạnh (3,4)
	xóa cạnh (3,15)
	chọn cạnh (14, 15), (15,6)
	xóa cạnh (14,20)
	chọn cạnh (19, 20), (20,16)
✓	Chọn đỉnh 5, cạnh (4,5)
	xóa cạnh (4,7)
	chọn cạnh (6, 7), (7,8)
	xóa cạnh (6,16)
	chọn cạnh (16, 17)
✓	...

Bài tập

Hãy tìm chu trình Hamilton (nếu có) của các đồ thị sau



Hình 5.54: Đồ thị Hamilton?

Bài toán người bán hàng

Bài toán 5.5

Bài toán người bán hàng (**travelling salesman problem**) được phát biểu như sau: Cho một đồ thị liên thông tìm một chu trình ngắn nhất đi qua các đỉnh

- Nếu đồ thị không có trọng số thì chu trình ngắn nhất là chu trình có số cạnh ít nhất
- Nếu đồ thị có trọng số thì chu trình ngắn nhất là chu trình có trọng số nhỏ nhất

Tài liệu tham khảo

Diestel, R. (2005).
Graph theory. 2005.
Springer-Verlag.

Moore, E. F. (1959).
The shortest path through a maze.
Bell Telephone System.

Rosen, K. H. and Krithivasan, K. (2012).
Discrete mathematics and its applications.
McGraw-Hill New York.

Tarjan, R. (1972).
Depth-first search and linear graph algorithms.
SIAM journal on computing, 1(2):146–160.

Tài liệu tham khảo (cont.)

Trần, T. and Dương, D. (2013).
Giáo trình lý thuyết đồ thị. 2013.
NXB Đại Học Quốc Gia TPHCM.

West, D. B. et al. (2001).
Introduction to graph theory.
Prentice hall Englewood Cliffs.