Graphs-Part 2

To Eulerian Graphs

Maximum Flows of Minimum Cost

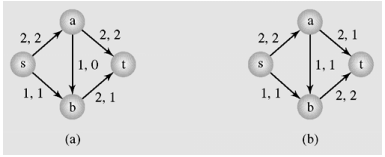


Figure 8-24 Two possible maximum flows for the same network

Maximum Flows of Minimum Cost (continued)

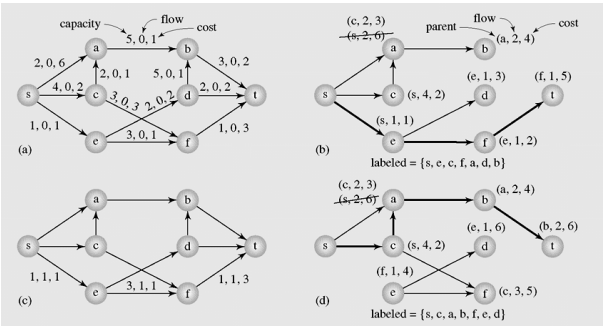


Figure 8-25 Finding a maximum flow of minimum cost

Maximum Flows of Minimum Cost (continued)

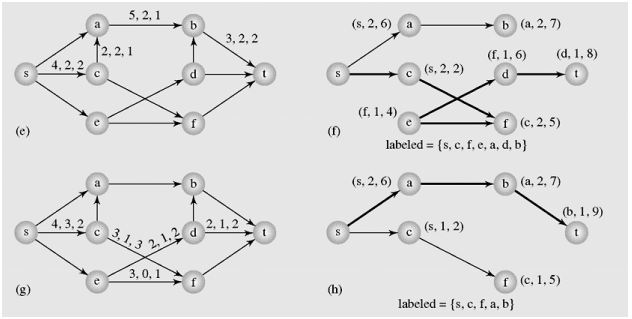


Figure 8-25 Finding a maximum flow of minimum cost (continued)

Maximum Flows of Minimum Cost (continued)

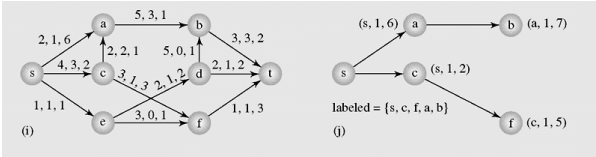


Figure 8-25 Finding a maximum flow of minimum cost (continued)

Matching

- **Maximum matching** is a matching that contains a maximum number of edges so that the number of unmatched vertices (that is, vertices not incident with edges in *M*) is minimal
- A **matching problem** consists in finding a maximum matching for a certain graph *G*
- The problem of finding a perfect matching is also called the **marriage problem**

Matching (continued)

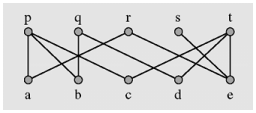


Figure 8-26 Matching five applicants with five jobs

Matching (continued)

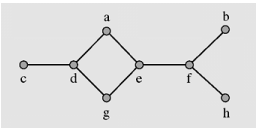


Figure 8-27 A graph with matchings M_1 = {*edge*(*ab*), *edge*(*ef*)} and M_2 = {*edge*(*ab*), *edge*(*de*), *edge*(*fh*)}

Matching (continued)

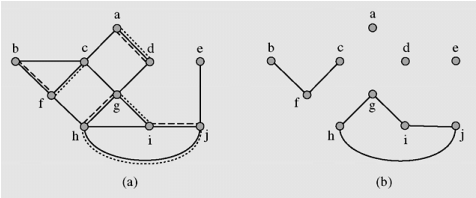


Figure 8-28 (a) Two matchings *M* and *N* in a graph *G* = (*V*,*E*) and (b) the graph *G'* = (*V*, *M* ⊕*N*)

Matching (continued)

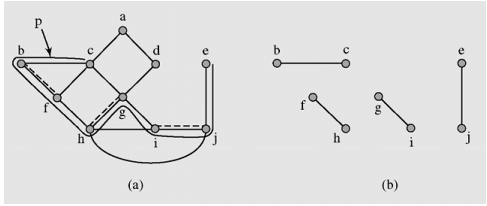


Figure 8-29 (a) Augmenting path *P* and a matching *M* and (b) the matching *M* ⊕ *P*

Matching (continued)

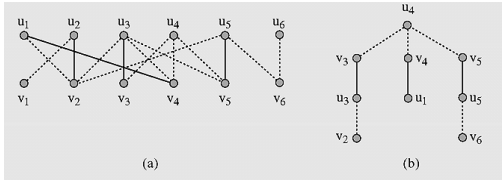


Figure 8-30 Application of the `findMaximumMatching()` algorithm. Matched vertices are connected with solid lines.

Matching (continued)

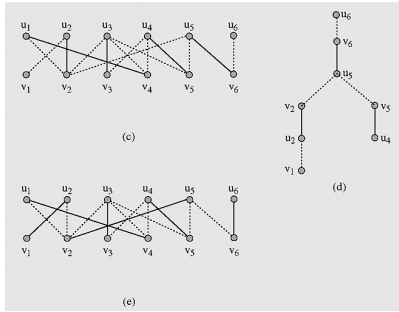


Figure 8-30 Application of the `findMaximumMatching()` algorithm. Matched vertices are connected with solid lines (continued).

Assignment Problem

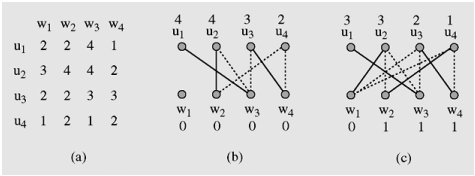


Figure 8-31 An example of application of the `optimalAssignment()` algorithm

Matching in Nonbipartite Graphs

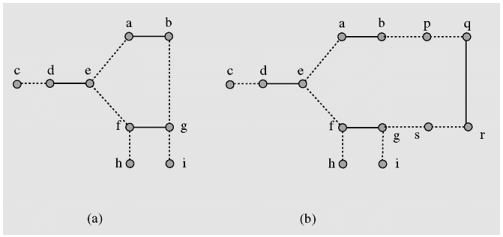


Figure 8-32 Application of the `findMaximumMatching()` algorithm to a nonbipartite graph

Matching in Nonbipartite Graphs (continued)

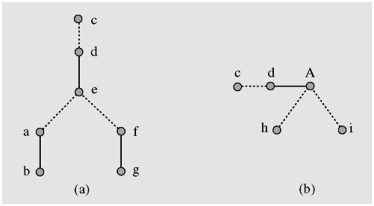


Figure 8-33 Processing a graph with a blossom

Matching in Nonbipartite Graphs (continued)

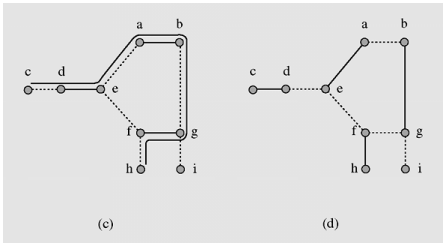
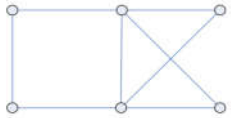


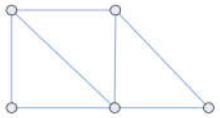
Figure 8-33 Processing a graph with a blossom (continued)

Euler Graph

- Đường đi qua môi cạnh của đô thị đúng một lần được gọi là đường đi Euler.
- Chu trình qua mỗi cạnh của đồ thị đúng một lần được gọi là chu trình Euler.
- Đồ thị được gọi là đồ thị Euler nếu nó có chu trình Euler, và gọi là đồ thị nửa Euler nếu nó có đường đi Euler
- Nhận xét: mọi đồ thị Euler luôn là nửa Euler, nhưng điều ngược lại không luôn đúng.



Đồ thị Euler



Đồ thị nửa Euler

17

8.10- Eulerian, Hamiltonian Graphs

- An **Eulerian trail** in a graph is a path that includes **all edges** of the graph **only once**
- An **Eulerian cycle** is a cycle that is also an Eulerian trail
- A graph that has an Eulerian cycle is called an **Eulerian graph**

Eulerian Graphs (continued)

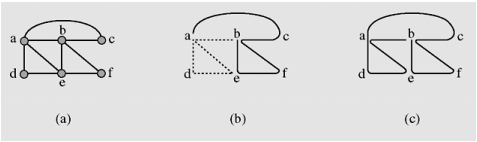


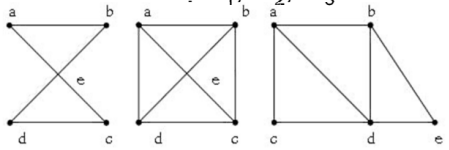
Figure 8-34 Finding an Eulerian cycle

A graph contains the Euler cycle if each of vertices having even degree.
A graph contains the Euler trail if it has exactly two odd-degree vertices.

Đồ thị vô hướng, liên thông G=(V, E) có chu trình Euler khi và chỉ khi mọi đỉnh của G đều có bậc chẵn.

Đồ thị vô hướng, liên thông G=(V, E) có đường đi Euler mà không có chu trình Euler khi và chỉ khi G có đúng hai đỉnh bậc lẻ.

Ví dụ 1: Xét 3 đồ thị G_1, G_2, G_3 bên dưới:

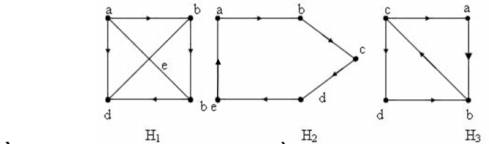


Đồ thị G_1 trong hình là đồ thị Euler vì nó có chu trình Euler a, e, c, d, e, b, a.

Đồ thị G_3 không có chu trình Euler nhưng nó có đường đi Euler a, c, d, e, b, d, a, b, vì thế G_3 là đồ thị nửa Euler.

Đồ thị G_2 không có chu trình cũng như đường đi Euler.

Ví dụ 2: Xét 3 đồ thị H_1, H_2, H_3 bên dưới:



Đồ thị H_2 trong hình là đồ thị Euler vì nó có chu trình Euler a, b, c, d, e, a.

Đồ thị H_3 không có chu trình Euler nhưng nó có đường đi Euler c, d, b, c, a, b, vì thế H_3 là đồ thị nửa Euler.

Đồ thị H_1 không có chu trình cũng như đường đi Euler.

Eulerian Graphs (continued)

Checking whether an undirected graph has Euler cycle (Fleury, 1883)

FleuryAlgorithm (undirected graph)

v = *a starting vertex* ; // any vertex
path = v;
Untraversed = graph;
while v *has untraveres edges*
  if edge(vu) *is the only one untraversed edge*
    e = edge (vu);
    remove v *from* untraversed;
  else e = edge (vu) *which is not a bridge* in untraversed;
  path = path + u;
  r*emove* e *from* untraversed;
  v = u;
If untraversed *has no edges* success;
Else failure;

Ý tưởng chủ đạo

❑ Thuật toán Flor:

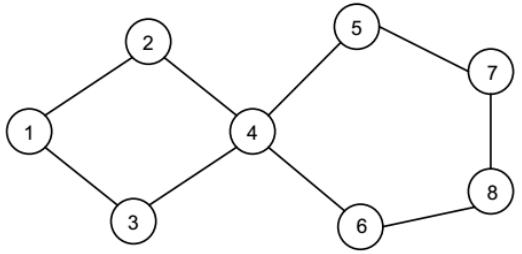
Xuất phát từ một đỉnh u nào đó của G ta đi theo các cạnh của nó một cách tuỳ ý chỉ cần tuân thủ 2 qui tắc sau:

(1) Xoá bỏ cạnh đã đi qua đồng thời xoá bỏ đỉnh cô lập tạo thành.

(2) Ở mỗi bước ta chỉ đi qua cầu khi không còn cách lựa chọn nào khác.

"một đi không trở lại"

Ví dụ 1

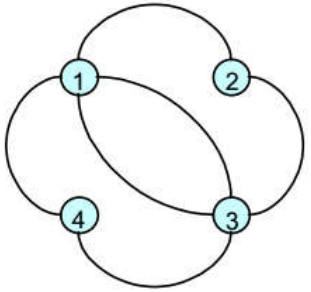


Đáp án

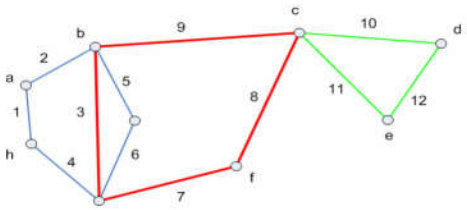
Nếu xuất phát từ đỉnh 1, có hai cách đi tiếp: hoặc sang 2 hoặc sang 3, giả sử ta sẽ sang 2 và xoá cạnh (1, 2) vừa đi qua. Từ 2 chỉ có cách duy nhất là sang 4, nên cho dù (2, 4) là cầu ta cũng phải đi sau đó xoá luôn cạnh (2, 4). Đến đây, các cạnh còn lại của đồ thị có thể vẽ như trên bằng nét liền, các cạnh đã bị xoá được vẽ bằng nét đứt.

Bây giờ đang đứng ở đỉnh 4 thì ta có 3 cách đi tiếp: sang 3, sang 5 hoặc sang 6. Vì (4, 3) là cầu nên ta sẽ không đi theo cạnh (4, 3) mà sẽ đi (4, 5) hoặc (4, 6). Nếu đi theo (4, 5) và cứ tiếp tục đi như vậy, ta sẽ được chu trình Euler là (1, 2, 4, 5, 7, 8, 6, 4, 3, 1). Còn đi theo (4, 6) sẽ tìm được chu trình Euler là: (1, 2, 4, 6, 8, 7, 5, 4, 3, 1).

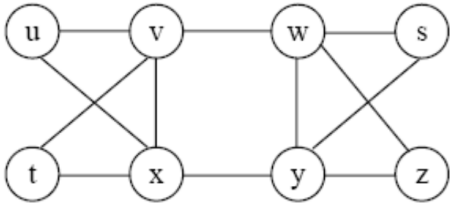
Ví dụ



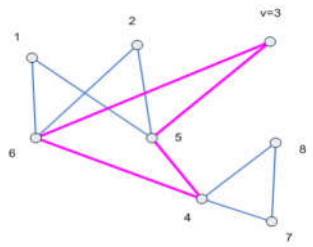
Ví dụ



Ví dụ



Ví dụ



❑ Bài toán người phát thư Trung Hoa (Guan 1960):

Một NV đi từ Sở BĐ, qua một số đường phố để phát thư, rồi quay về Sở. Phải đi qua các đường theo trình tự nào để đường đi là ngắn nhất?

Xét bài toán: Cho đồ thị liên thông G. Một chu trình qua mọi cạnh của G gọi là một hành trình trong G. Hãy tìm hành trình ngắn nhất (qua ít cạnh nhất).

Nếu G là đồ thị Euler thì chu trình Euler trong G là hành trình ngắn nhất cần tìm.

Xét trường hợp G có một số đỉnh bậc lẻ. Khi đó, mọi hành trình trong G phải đi qua ít nhất hai lần một số cạnh nào đó. 56

Ý tưởng

❑ Bài toán người phát thư Trung Hoa (Guan 1960):

Định lý (Gooodman và Hedetniemi, 1973). Nếu G là một đồ thị liên thông có q cạnh thì hành trình ngắn nhất trong G có chiều dài q + m(G),

trong đó m(G) là số cạnh mà hành trình đi qua hai lần và được xác định như sau:

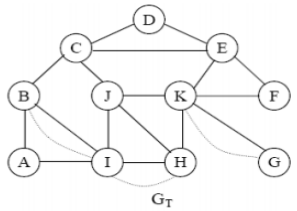
Gọi $V_0(G)$ là tập hợp các đỉnh bậc lẻ (2k đỉnh) của G. Ta phân 2k phần tử của G thành k cặp, mỗi tập hợp k cặp gọi là một phân hoạch cặp của $V_0(G)$.

Gọi độ dài đường đi ngắn nhất từ u đến v là khoảng cách d(u,v). Đối với mọi phân hoạch cặp P_i, tính khoảng cách giữa hai đỉnh trong từng cặp, tính tổng d(Pi).

Số m(G) bằng cực tiểu của các d(Pi): m(G)=min d(Pi). 57

Ví dụ

1. Đồ thị EULER:

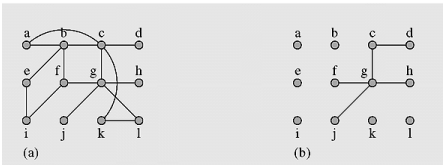


G_T

$V_O(G)$={B, G, H, K}
tập hợp các phân hoạch cặp là:
P={P_1, P_2, P_3}, trong đó:
P_1 = {(B, G), (H, K)} → d(P_1) = d(B, G)+d(H, K) = 4+1 = 5
P_2 = {(B, H), (G, K)} → d(P_2) = d(B, H)+d(G, K) = 2+1 = 3,
P_3 = {(B, K), (G, H)} → d(P_3) = d(B, K)+d(G, H) = 3+2 = 5.
m(G) = min(d(P_1), d(P_2), d(P_3)) = 3.
G_T có được từ G bằng cách thêm vào 3 cạnh: (B, I), (I, H), (G, K) và G_T là đồ thị Euler.
Vậy hành trình ngắn nhất cần tìm là đi theo chu trình Euler trong G_T:
A, B, C, D, E, F, K, G, K, E, C, J, K, H, J, I, H, I, B, I, A. 58

The Chinese Postman Problem

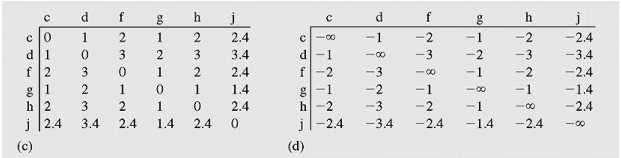
A postman picks mails at the post office, delivers them and returns to the first office.
Vertex: Street corner, edge: street with its length ➔ Initial graph G
→If G has no Eulerian cycle, construct an Eulerian graph with minimum cost G* from G by adding some edges to odd vertices such that the cost is minimum.



Set of odd vertices: { c,d,f,g,h,j}

Figure 8-35 Solving the Chinese postman problem

The Chinese Postman Problem



	c	d	f	g	h	j
c	0	1	2	1	2	2.4
d	1	0	3	2	3	3.4
f	2	3	0	1	2	2.4
g	1	2	1	0	1	1.4
h	2	3	2	1	0	2.4
j	2.4	3.4	2.4	1.4	2.4	0

(c)

	c	d	f	g	h	j
c	−∞	−1	−2	−1	−2	−2.4
d	−1	−∞	−3	−2	−3	−3.4
f	−2	−3	−∞	−1	−2	−2.4
g	−1	−2	−1	−∞	−1	−1.4
h	−2	−3	−2	−1	−∞	−2.4
j	−2.4	−3.4	−2.4	−1.4	−2.4	−∞

(d)

Figure 8-35 Solving the Chinese postman problem (continued)

The Chinese Postman Problem

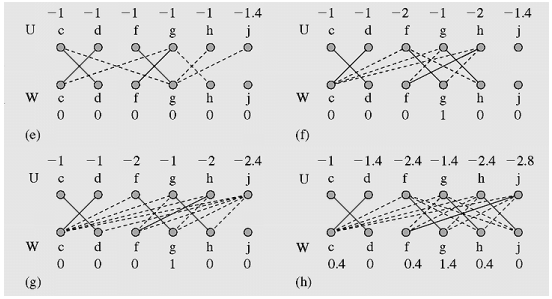


Figure 8-35 Solving the Chinese postman problem (continued)

The Chinese Postman Problem

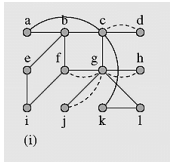


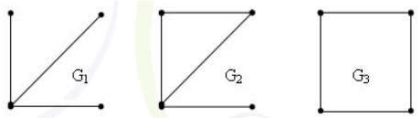
Figure 8-35 Solving the Chinese postman problem (continued)

2. Đồ thị HAMILTON

- Đường đi qua tất cả các đỉnh của đồ thị mỗi đỉnh đúng một lần được gọi là đường đi Hamilton.
- Chu trình bắt đầu từ một đỉnh v nào đó qua tất cả các đỉnh còn lại mỗi đỉnh đúng một lần rồi quay trở về v được gọi là chu trình Hamilton.
- Đồ thị G được gọi là đồ thị Hamilton nếu nó chứa chu trình Hamilton và gọi là đồ thị nửa Hamilton nếu nó có đường đi Hamilton.

2. Đồ thị HAMILTON

Ví dụ 3. Trong hình: Đồ thị G3 là Hamilton, G2 là nửa Hamilton còn G1 không là nửa Hamilton.

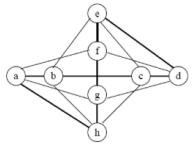


Cho đến nay việc tìm một tiêu chuẩn nhận biết đồ thị Hamilton vẫn còn là mở.
Phần lớn các phát biểu đều có dạng "nếu G có số cạnh đủ lớn thì G là Hamilton"

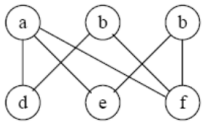
- Định lý 1 (Dirak 1952). Đơn đồ thị vô hướng G với n>2 đỉnh, mỗi đỉnh có bậc không nhỏ hơn n/2 là đồ thị Hamilton.
- Định lý 2 Nếu G là đồ thị phân đôi với hai tập đỉnh là V_1, V_2 có số đỉnh cùng bằng n ($n \geq 2$) và bậc của mỗi đỉnh lớn hơn n/2 thì G là một đồ thị Hamilton.
- Định lý 3. Giả sử G là đồ có hướng liên thông với n đỉnh. Nếu $\deg^+(v) \geq n/2$, $\deg^-(v) \geq n/2$, $\forall v$ thì G là Hamilton.

61

Ví dụ

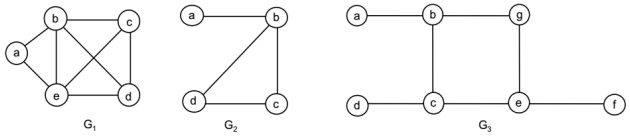


Đồ thị G này có 8 đỉnh, đỉnh nào cũng có bậc 4, nên G là đồ thị Hamilton.



Đồ thị phân đôi này có bậc của mỗi đỉnh bằng 2 hoặc 3 (> 3/2), nên nó là đồ thị Hamilton.

Ví dụ: Xét 3 đơn đồ thị G_1, G_2, G_3 sau:



Đồ thị G_1 có chu trình Hamilton (a, b, c, d, e, a). G_2 không có chu trình Hamilton vì deg(a) = 1 nhưng có đường đi Hamilton (a, b, c, d). G_3 không có cả chu trình Hamilton lẫn đường đi Hamilton

Nhận biết đồ thị Hamilton

- Chưa có chuẩn để nhận biết 1 đồ thị có là Hamilton hay không
- Chưa có thuật toán để kiểm tra
- Các kết quả thu được ở dạng điều kiện đủ
- Nếu G có số cạnh đủ lớn thì G là Hamilton

Hamiltonian Graphs

- A **Hamiltonian cycle** in a graph is a cycle that passes through all the vertices of the graph
- A graph is called a **Hamiltonian graph** if it includes at least one Hamiltonian cycle
- All complete graphs are Hamiltonian.
- Ore Theorem: if edge(vu)$\notin$ E, graph G'=(V, E $\cup${edge(vu)}) is Hamiltonian, and deg(v) + deg(u) $\geq$|V|, then graph G=(V,E) is also Hamiltonian.

Hamiltonian Graphs (continued)

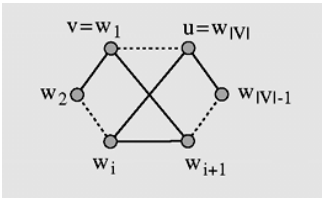


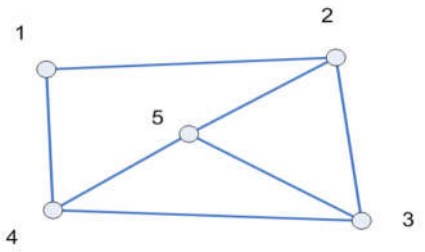
Figure 8-36 Crossover edges

→ We can easily expand a non-Hamiltonian to a Hamiltonian graph (algorithm: page 436)

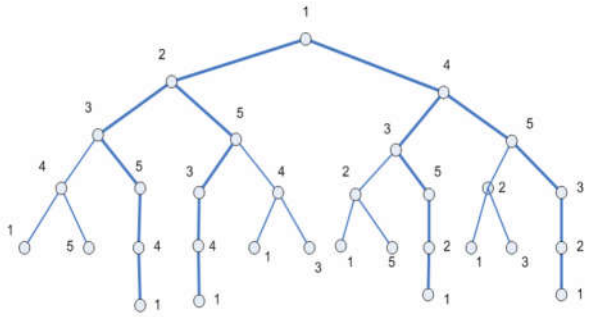
II.3. Giải thuật x/d chu trình Hamilton

- Dùng giải thuật quay lui
  - *Bắt đầu từ 1 đỉnh, đi theo con đường dài nhất có thể được (depth – first)*
  - *Nếu đường đó chứa mọi đỉnh và có thể nối 2 đỉnh đầu và cuối bằng 1 cạnh thì đó là chu trình Hamilton*
  - *Nếu trái lại ta lùi lại một đỉnh để mở con đường theo chiều sâu khác*
  - *Cứ tiếp tục quá trình trên cho đến khi thu được chu trình Hamilton.*

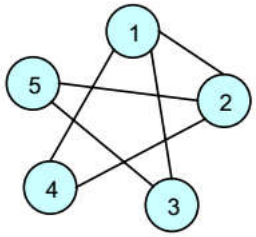
Ví dụ



Đáp án là cây tìm kiếm tất cả các chu trình Hamilton



Ví dụ



1 3 5 2 4 1

1 4 2 5 3 1

Hamiltonian Graphs (continued)

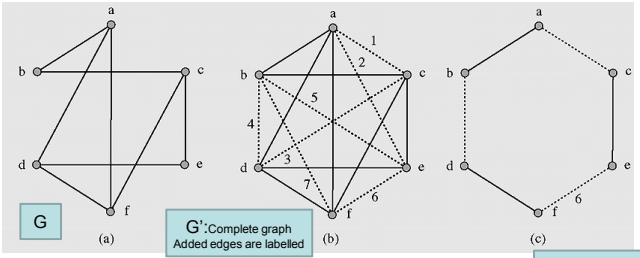


Figure 8-37 Finding a Hamiltonian cycle

Hamiltonian Graphs (continued)

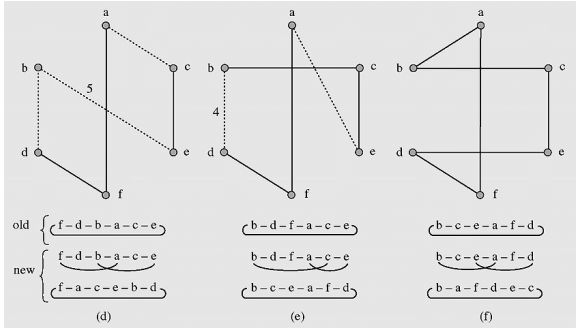


Figure 8-37 Finding a Hamiltonian cycle (continued)

The Traveling Salesman Problem

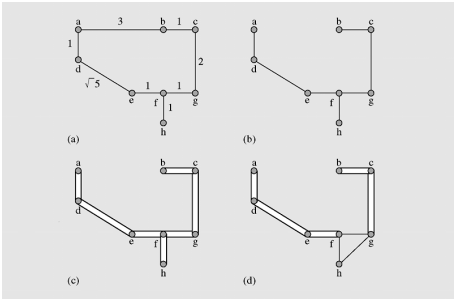


Figure 8-38 Using a minimum spanning tree to find a minimum salesman tour

The Traveling Salesman Problem (continued)

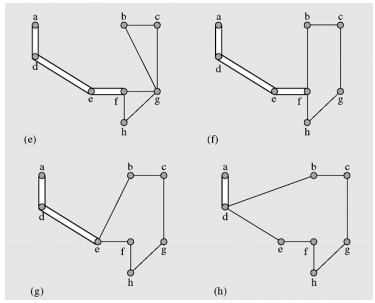


Figure 8-38 Using a minimum spanning tree to find a minimum salesman tour (continued)

The Traveling Salesman Problem (continued)

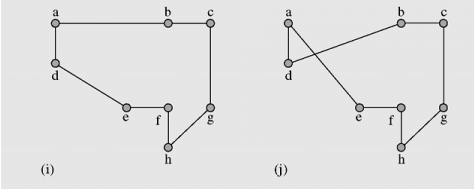


Figure 8-38 Using a minimum spanning tree to find a minimum salesman tour (continued)

The Traveling Salesman Problem (continued)

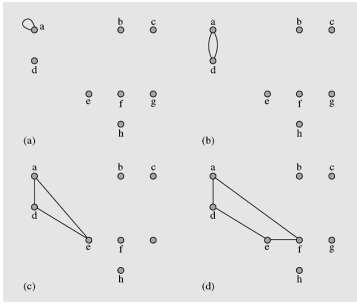


Figure 8-39 Applying the nearest insertion algorithm to the cities in Figure 8.38a

The Traveling Salesman Problem (continued)

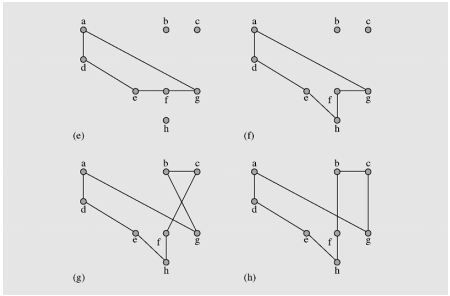
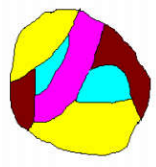
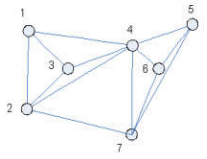


Figure 8-39 Applying the nearest insertion algorithm to the cities in Figure 8.38a (continued)

I. Định nghĩa

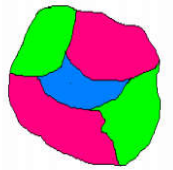
- **Cần phải tô màu một bản đồ với điều kiện:**
  - Hai miền chung biên giới được tô hai màu khác nhau
  - Số màu cần dùng là tối thiểu
- **Hãy xác định số màu tối thiểu cho mọi bản đồ**

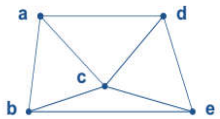




Bản đồ này cần dùng 4 màu để tô

I. Định nghĩa





Bài toán tô màu bản đồ quy về bài toán tô màu các **Đỉnh** của đồ thị

Định nghĩa 1

Tô màu một đơn đồ thị là sự gán màu cho các đỉnh của nó sao cho hai đỉnh liền kề nhau được gán màu khác nhau.

Định nghĩa 2

Số màu của một đồ thị là số tối thiểu các màu cần thiết để tô màu đồ thị này.

II. Định lý 4 màu

- *Định lý: Số màu của một đồ thị phẳng là không lớn hơn 4*
  - Định lý này được phát biểu lần đầu tiên năm 1850 và được 2 nhà toán học Mỹ Appel và Haken chứng minh năm 1976 bằng phản chứng.
  - Đối với các đồ thị không phẳng số màu có thể tuỳ ý lớn
  - Để chứng minh đồ thị G là n-màu ta phải
    - Chỉ ra 1 cách tô màu G với n màu
    - CMR không thể tô màu G với ít hơn n màu

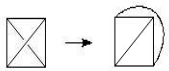
2. Đồ thị phẳng

2.1. Định nghĩa

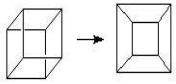
Một đồ thị được gọi là phẳng nếu nó có thể vẽ được trên một mặt phẳng mà không có các cạnh nào cắt nhau ở điểm không phải là điểm mút của mỗi cạnh. Hình vẽ như vậy được gọi là một biểu diễn phẳng của đồ thị.

2.2. Các ví dụ

Ví dụ 1: K_4 là đồ thị phẳng vì có thể vẽ lại như sau:



Ví dụ 2: Q_3 cũng là đồ thị phẳng vì:



II. Định lý 4 màu

- Các bài toán tô màu đồ thị
  1. Cho đồ thị G và số nguyên k. Xây dựng một thuật toán để kiểm tra xem có thể tô màu G bằng k màu, nếu được thì thực hiện việc đó.
  2. Cho đồ thị G hãy xác định số màu k của đồ thị và hãy tô màu G bằng k màu đó

Ý tưởng chủ đạo

Thuật toán **SequentialColor** tô màu 1 đồ thị với k màu:
Xem các đỉnh theo thứ tự từ 1 đến |V|, tại mỗi đỉnh v gán màu đầu tiên có sẵn mà chưa được gán cho 1 đỉnh nào liền v

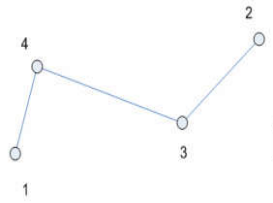
1. Xếp các đỉnh theo thứ tự bất kỳ 1,2,… n
2. Tạo tập L_i - tập các màu có thể gán cho đỉnh I
3. Bắt đầu tô từ đỉnh 1
4. Với đỉnh k ∈ {1,…,n} tô màu đầu tiên của L_k cho k
5. ∀ j > k và j kề k loại bỏ trong L_j màu đã được tô cho k
6. Giải thuật dừng lại khi tất cả các đỉnh đã được tô

8.11- Graph Coloring

- **Sequential coloring** establishes the sequence of vertices and a sequence of colors before coloring them, and then color the next vertex with the lowest number possible

```
sequentialColoringAlgorithm(graph = (V, E))
    put vertices in a certain order vP1, vP2, . . . , vPv ;
    put colors in a certain order cP1, c2, . . . , ck;
    for i = 1 to |V|
        j = the smallest index of color that does not appear in any neighbor of vPi;
        color(vPi) = cj;
```


Ví dụ

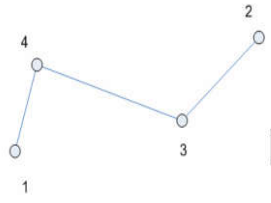


Các màu:
X: Xanh
Đ: Đỏ
T: Tím
V: Vàng

Thứ tự tô các đỉnh: 1, 2, 3, 4

Các bước	L1	L2	L3	L4	Màu tô
Khởi tạo	X, Đ, T, V	X, Đ, T, V	X, Đ, T, V	X, Đ, T, V	
B1	X	X, Đ, T, V	X, Đ, T, V	Đ, T, V	1 - Xanh
B2		X	Đ, T, V	Đ, T, V	2 - Xanh
B3			Đ	T, V	3 - Đỏ
B4				T	4 - Tím

Ví dụ



Các màu:
X: Xanh
Đ: Đỏ
T: Tím
V: Vàng

Thứ tự tô các đỉnh: 4, 3, 2, 1

Các bước	L4	L3	L1	L2	Màu tô
Khởi tạo	X, Đ, T, V	X, Đ, T, V	X, Đ, T, V	X, Đ, T, V	
B1	X	Đ, T, V	Đ, T, V	X, Đ, T, V	4 - Xanh
B2		Đ	Đ, T, V	X, T, V	3 - Đỏ
B3			Đ	X, T, V	1 - Đỏ
B4				X	2 - Xanh

- Nhận xét
  - Là dạng thuật toán tham lam → Lời giải tìm được chưa chắc tối ưu
  - Độ phức tạp của giải thuật $O(n^2)$

Graph Coloring (continued)

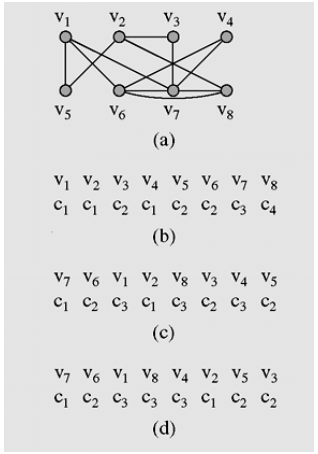


Figure 8-40
(a) A graph used for coloring;
(b) colors assigned to vertices with the sequential coloring algorithm that orders vertices by index number;
(c) vertices are put in the largest first sequence;
(d) graph coloring obtained with the Brélaz algorithm

End of session
Click to go to summary

NP-Complete Problems

- P Problems: Polynomial time problems with determonistic algorithms → tractable
- NP Problems: Polynomial time problems with non-determonistic algorithms → tractable
- NP Complete Problems: Problems when they are reduced to NP problems.

The Clique Problem

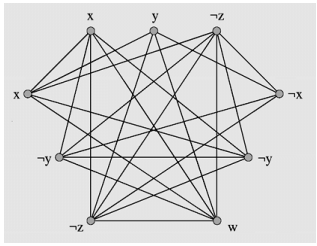


Figure 8-41 A graph corresponding to Boolean expression

The 3-Colorability Problem

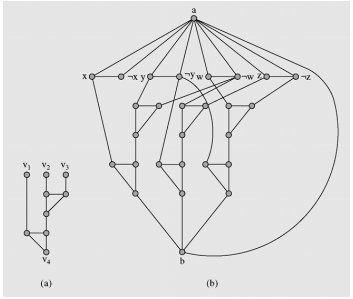


Figure 8-42 (a) A 9-subgraph; (b) a graph corresponding to Boolean expression

The Vertex Cover Problem

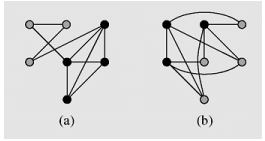


Figure 8-43 (a) A graph with a clique; (b) a complement graph

The Hamiltonian Cycle Problem

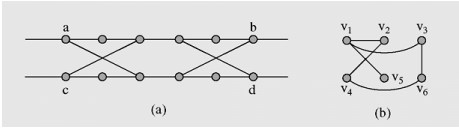


Figure 8-44 (a) A 12-subgraph; (b) a graph *G* and (c) its transformation, graph G_H

The Hamiltonian Cycle Problem

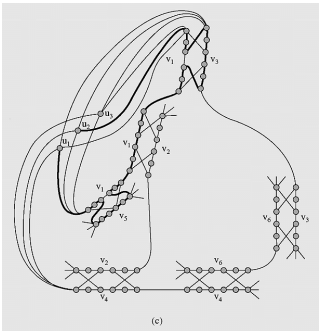


Figure 8-44 (a) A 12-subgraph; (b) a graph *G* and (c) its transformation, graph G_H (continued)

Case Study: Distinct Representatives

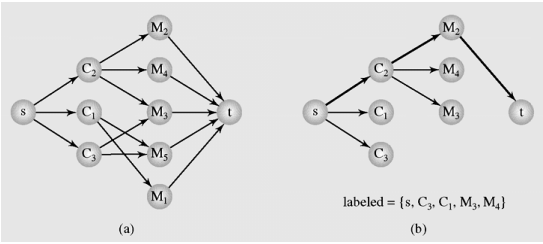


Figure 8-45 (a) A network representing membership of three committees, C_1, C_2, and C_3, and (b) the first augmenting path found in this network

Case Study: Distinct Representatives (continued)

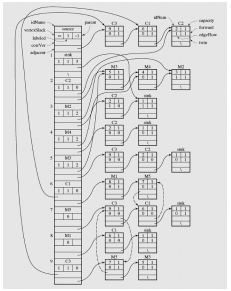


Figure 8-46 The network representation created by `FordFulkersonMaxFlow()`

Case Study: Distinct Representatives (continued)

```
import java.io.*;
import java.util.*;

class Vertex {
    public int idNum, capacity, edgeFlow;
    public boolean forward; // direction;
    public Vertex twin;     // edge in opposite direction;
    public Vertex() {
    }
    public Vertex(int id, int c, int ef, boolean f) {
        idNum = id; capacity = c; edgeFlow = ef; forward = f; twin = null;
    }
    public boolean equals(Object v) {
        return idNum == ((Vertex)v).idNum;
    }
    public String toString() {
        return (idNum + " " + capacity + " " + edgeFlow + " " + forward);
    }
}
```

Figure 8-47 An implementation of the distinct representatives problem

Case Study: Distinct Representatives (continued)

```
}

class VertexInArray {
    public String idName;
    public int vertexSlack;
    public boolean labeled = false;
    public int parent;
    public LinkedList adjacent = new LinkedList();
    public Vertex corrVer;   // corresponding vertex: vertex on parent's
    public VertexInArray() { // list of adjacent vertices with the same
    }                        // idNum as the cell's index;
    public VertexInArray(String s) {
        idName = s;
    }
    public boolean equals(Object v) {
        return idName.equals(((VertexInArray)v).idName);
    }
    public void display() {
        System.out.print(idName + ' ' + vertexSlack + ' '
             + labeled + ' ' + parent + ' ' + corrVer + "-> ");
        System.out.print(adjacent);
        System.out.println();
    }
}
```

Figure 8-47 An implementation of the distinct representatives problem (continued)

Case Study: Distinct Representatives (continued)

```
class Network {
    public Network() {
        vertices.add(source,new VertexInArray());
        vertices.add(sink,  new VertexInArray());
        ((VertexInArray)vertices.get(source)).idName = "source";
        ((VertexInArray)vertices.get(sink)).idName   = "sink";
        ((VertexInArray)vertices.get(source)).parent = none;
    }
    private final int sink = 1, source = 0, none = -1;
    private ArrayList vertices = new ArrayList();
    private int edgeSlack(Vertex u) {
        return u.capacity - u.edgeFlow;
    }
    private boolean labeled(Vertex p) {
        return ((VertexInArray)vertices.get(p.idNum)).labeled;
    }
    public void display() {
        for (int i = 0; i < vertices.size(); i++) {
            System.out.print(i + ": " );
            ((VertexInArray)vertices.get(i)).display();
        }
    }
```

Figure 8-47 An implementation of the distinct representatives problem (continued)

Case Study: Distinct Representatives (continued)

```
public void readCommittees(String fileName, InputStream fIn) {
    int ch = 1, pos;
    try {
        while (ch > -1) {
            while (true)
                if (ch > -1 && !Character.isLetter((char)ch)) // skip
                    ch = fIn.read();                  // nonletters;
                else break;
            if (ch == -1)
                break;
            String s = "";
            while (ch > -1 && ch != ':') {
                s += (char)ch;
                ch = fIn.read();
            }
            VertexInArray committee = new VertexInArray(s.trim());
            int commPos = vertices.size();
            Vertex commVer = new Vertex(commPos,1,0,false);
```

Figure 8-47 An implementation of the distinct representatives problem (continued)

Case Study: Distinct Representatives (continued)

```
vertices.add(committee);
for (boolean lastMember = false; !lastMember; ) {
    while (true)
        if (ch > -1 && !Character.isLetter((char)ch))
            ch = fIn.read();         // skip nonletters;
        else break;
    if (ch == -1)
        break;
    s = "";
    while (ch > -1 && ch != ',' && ch != ';') {
        s += (char)ch;
        ch = fIn.read();
    }
    if (ch == ';')
        lastMember = true;
```

Figure 8-47 An implementation of the distinct representatives problem (continued)

Case Study: Distinct Representatives (continued)

```
    VertexInArray member = new VertexInArray(s.trim());
    Vertex memberVer = new Vertex(0,1,0,true);
    if ((pos = vertices.indexOf(member)) == -1) {
        memberVer.idNum = vertices.size();
        member.adjacent.addFirst(new
                        Vertex(sink,1,0,true));
        member.adjacent.addFirst(commVer);
        vertices.add(member);
    }
    else {
        memberVer.idNum = pos;
        ((VertexInArray)vertices.get(pos)).
              adjacent.addFirst(commVer);
    }
    committee.adjacent.addFirst(memberVer);
    memberVer.twin = commVer;
    commVer.twin = memberVer;
```

Figure 8-47 An implementation of the distinct representatives problem (continued)

Case Study: Distinct Representatives (continued)

```
            }
            commVer = new Vertex(commPos,1,0,true);
            ((VertexInArray)vertices.get(source)).adjacent.
                                addFirst(commVer);
        }
    } catch (IOException io) {
    }
    display();
}
private void label(Vertex u, int v) {
```

Figure 8-47 An implementation of the distinct representatives problem (continued)

Case Study: Distinct Representatives (continued)

```
    VertexInArray uu = (VertexInArray) vertices.get(u.idNum);
    VertexInArray vv = (VertexInArray) vertices.get(v);
    uu.labeled = true;
    if (u.forward)
        uu.vertexSlack = Math.min(vv.vertexSlack,edgeSlack(u));
    else uu.vertexSlack = Math.min(vv.vertexSlack,u.edgeFlow);
    uu.parent  = v;
    uu.corrVer = u;
}
```

Figure 8-47 An implementation of the distinct representatives problem (continued)

Case Study: Distinct Representatives (continued)

```
}
private void augmentPath() {
    int sinkSlack = ((VertexInArray)vertices.get(sink)).vertexSlack;
    Stack path = new Stack();
    for (int i = sink; i != source;
         i = ((VertexInArray)vertices.get(i)).parent) {
        VertexInArray vv = (VertexInArray) vertices.get(i);
        path.push(vv.idName);
        if (vv.corrVer.forward)
            vv.corrVer.edgeFlow += sinkSlack;
        else vv.corrVer.edgeFlow -= sinkSlack;
        if (vv.parent != source && i != sink)
            vv.corrVer.twin.edgeFlow = vv.corrVer.edgeFlow;
    }
    for (int i = 0; i < vertices.size(); i++)
        ((VertexInArray)vertices.get(i)).labeled = false;
    System.out.print("  source");
    while (!path.isEmpty())
        System.out.print(" => " + path.pop());
    System.out.print(" (augmented by " + sinkSlack + ");\n");
```

Figure 8-47 An implementation of the distinct representatives problem (continued)

Case Study: Distinct Representatives (continued)

```
}
public void FordFulkersonMaxFlow() {
    Stack labeledS = new Stack();
    for (int i = 0; i < vertices.size(); i++)
        ((VertexInArray) vertices.get(i)).labeled = false;
    ((VertexInArray)vertices.get(source)).vertexSlack =
        Integer.MAX_VALUE;
    labeledS.push(new Integer(source));
    System.out.println("Augmenting paths:");
    while (!labeledS.isEmpty()) {   // while not stuck;
        int v = ((Integer) labeledS.pop()).intValue();
        for (Iterator it = ((VertexInArray)vertices.get(v)).
                                    adjacent.iterator();
```

Figure 8-47 An implementation of the distinct representatives problem (continued)

Case Study: Distinct Representatives (continued)

```
                it.hasNext(); ) {
               Vertex u = (Vertex) it.next();
               if (!labeled(u)) {
                   if (u.forward && edgeSlack(u) > 0 ||
                       !u.forward && u.edgeFlow > 0)
                        label(u,v);
                   if (labeled(u))
                        if (u.idNum == sink) {
                             augmentPath();
                             labeledS.clear(); // look for another path;
                             labeledS.push(new Integer(source));
                             break;
                        }
                        else {
                             labeledS.push(new Integer(u.idNum));
                             ((VertexInArray)vertices.get(u.idNum)).
                                                      labeled = true;
                        }
               }
          }
     }
  }
}
```

Figure 8-47 An implementation of the distinct representatives problem (continued)

Case Study: Distinct Representatives (continued)

```
public class DistinctRepresentatives {
    static public void main(String args[]) {
        String fileName = "";
        Network net = new Network();
        InputStream fIn;
        InputStreamReader isr = new InputStreamReader(System.in);
        BufferedReader buffer = new BufferedReader(isr);
        try {
            if (args.length == 0) {
                System.out.print("Enter a file name: ");
                fileName = buffer.readLine();
                fIn = new FileInputStream(fileName);
            }
            else {
                fIn = new FileInputStream(args[0]);
                fileName = args[0];
            }
            net.readCommittees(fileName,fIn);
```

Figure 8-47 An implementation of the distinct representatives problem (continued)

Case Study: Distinct Representatives (continued)

```
            fIn.close();
        } catch(IOException io) {
            System.err.println("Cannot open " + fileName);
        }
        net.FordFulkersonMaxFlow();
        net.display();
    }
}
```

Figure 8-47 An implementation of the distinct representatives problem (continued)

Summary

- A graph is a collection of vertices (or nodes) and the connections between them
- A multigraph is a graph in which two vertices can be joined by multiple edges
- A pseudograph is a multigraph with the condition $v_i \neq v_j$ removed, which allows for loops to occur
- The sets used to solve the union-find problem are implemented with circular linked lists

Summary (continued)

- A spanning tree is an algorithm that guarantees generating a tree (or a forest, a set of trees) that includes or spans over all vertices of the original graph
- An undirected graph is called connected when there is a path between any two vertices of the graph
- A network is a digraph with one vertex *s,* called the source*,* with no incoming edges, and one vertex *t,* called the sink*,* with no outgoing edges

Summary (continued)

- Maximum matching is a matching that contains a maximum number of edges so that the number of unmatched vertices (that is, vertices not incident with edges in *M*) is minimal
- A Hamiltonian cycle in a graph is a cycle that passes through all the vertices of the graph
- Sequential coloring establishes the sequence of vertices and a sequence of colors before coloring them, and then color the next vertex with the lowest number possible