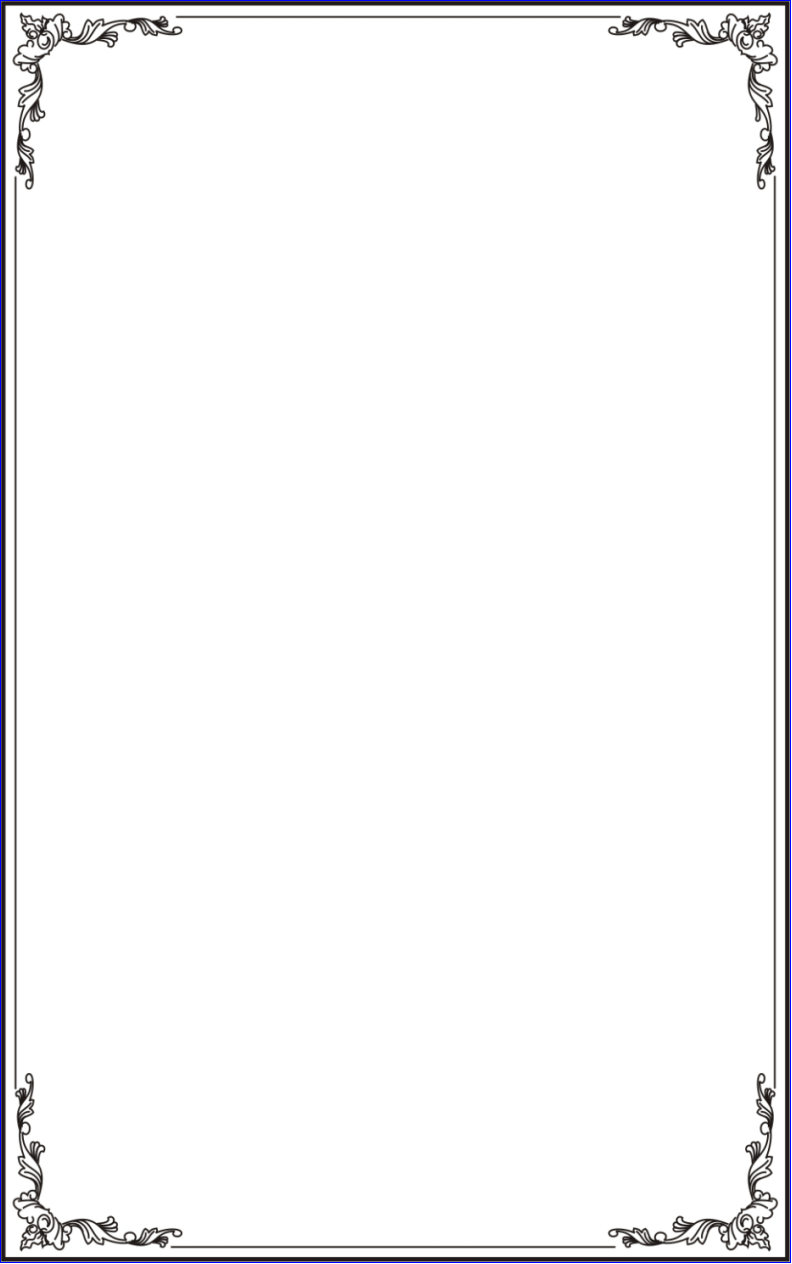
**TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP HÀ NỘI**

**KHOA CÔNG NGHỆ THÔNG TIN**



**BÁO CÁO BÀI TẬP LỚN**

**HỌC PHẦN : THỰC TẬP CƠ SỞ NGÀNH**

**ÁP DỤNG CÁC THUẬT TOÁN ĐỂ GIẢI BÀI TOÁN**

**TÔ MÀU ĐỒ THỊ**

| **Giảng viên hướng dẫn:** | TS. Nguyễn Thị Mỹ Bình |
| --- | --- |
| **Nhóm:**  **Thành viên:**  **Lớp - khóa:** | 13  Nguyễn Thị Trang - 2021604432  Đàm Thị Thủy - 2021605460  Nguyễn Thành Trung - 2021608050  Phạm Bảo Trung - 2020603277  Lê Anh Tú  Nguyễn Văn Tuấn - 2021608377  20222IT6040001 - K16 |

**Hà Nội - 2023**

# **MỤC LỤC**

[**MỤC LỤC 2**](#_eg28jpze5wlf)

[**LỜI CẢM ƠN 4**](#_hkmd22j3k3j2)

[**MỞ ĐẦU 5**](#_hbqq5bmc0xti)

[**CHƯƠNG I: GIỚI THIỆU CHUNG 7**](#_ui691jwfq2zo)

[**1.1. Tên đề tài 7**](#_uqxdmbk8avn5)

[**1.2. Lý do chọn đề tài 7**](#_ygikn8j2qt39)

[**1.3. Công cụ sử dụng 8**](#_7jt5z5o6vbck)

[**1.3.1. Ngôn ngữ lập trình C++ 8**](#_125nyxxa7o5c)

[**1.3.2. Dev-C++ 9**](#_a60ha9ln3fzk)

[**2.1. Khái niệm đồ thị 11**](#_dyb3pray62xp)

[**2.2. Đường đi, chu trình, đồ thị liên thông, đồ thị phẳng 12**](#_j1ofsxohrsqp)

[**2.2.1. Đường đi, chu trình: 12**](#_9014z79ibkba)

[**2.2.1.1. Đường đi 12**](#_z7zrih52ros9)

[**2.2.1.2. Chu trình 13**](#_ctwx68oh5ysl)

[**2.2.2. Đồ thị phẳng, đồ thị liên thông: 14**](#_z53sronsroi5)

[**2.2.2.1. Đồ thị phẳng 14**](#_w28589t563t)

[**2.2.2.2. Đồ thị liên thông 15**](#_jbimooy4s7fz)

[**2.3. Đơn đồ thị vô hướng liên thông 16**](#_4eqyz8rghed0)

[**2.4. Đồ thị có hướng liên thông 18**](#_urs494k3kvdh)

[**CHƯƠNG III : ÁP DỤNG CÁC THUẬT TOÁN ĐỂ CÀI ĐẶT BÀI TOÁN TÔ MÀU ĐỒ THỊ 20**](#_8wly2ieqvs66)

[**3.1. Phát biểu bài toán 20**](#_nrp2vtydcszp)

[**3.1.1. Bài toán 20**](#_qxfewxe6hlzd)

[**3.1.2. Xác định đầu vào, đầu ra của bài toán: 20**](#_kxe8y69klbg2)

[**3.1.3. Mô hình hóa bài toán 21**](#_h83x576h9cst)

[**3.2. Thuật toán tham lam 22**](#_5gv7oy4b7rji)

[**3.2.1. Giới thiệu chung 22**](#_muryh3xuirxv)

[**3.2.2. Lưu đồ thuật toán 23**](#_55d7q6tumidd)

[**3.2.3. Cài đặt thuật toán 24**](#_o4xdu2z5g2n3)

[**3.2.4. Đánh giá thuật toán 28**](#_8z2br2rmteq)

[**3.3. Thuật toán thu gọn (hòa nhập) đỉnh (contraction algorithm) 30**](#_v801mui4vrva)

[**3.3.1. Giới thiệu chung 30**](#_d1ajxf6bgkii)

[**3.3.2 Lưu đồ thuật toán 30**](#_fanfyyvk5q5)

[**3.3.3. Cài đặt thuật toán 30**](#_3p4g5b5nf941)

[**3.3.4. Đánh giá thuật toán 33**](#_rxiism7i4s1r)

[**3.4. Thuật toán tô màu tuần tự 33**](#_m6y6bc6kfr0x)

[**3.4.1. Giới thiệu chung 33**](#_rmeu51r1y7no)

[**3.4.2 Lưu đồ thuật toán 34**](#_7w2wxgezs43y)

[**3.4.3. Cài đặt thuật toán 36**](#_dsvrp859c5n)

[**3.4.4. Đánh giá thuật toán 36**](#_uhhesy3sjf50)

[**THỰC NGHIỆM THUẬT TOÁN 38**](#_u685ho7orrgs)

[**KẾT LUẬN 46**](#_ulau1f56bl7b)

[**TÀI LIỆU THAM KHẢO 47**](#_olyn0fpald4q)

# 

# 

# 

# 

# **LỜI CẢM ƠN**

Đối với một sinh viên trường Đại học Công nghiệp Hà Nội, bài tập lớn đóng một vai trò vô cùng quan trọng. Nó giúp chúng em củng cố kiến thức cũng như giúp em cải thiện các kỹ năng còn thiếu. Nhóm em xin gửi lời cảm ơn đến trường Đại học Công nghiệp Hà Nội, khoa Công nghệ thông tin đã tạo điều kiện học tập cho chúng em để chúng em có điều kiện học tập tốt nhất để hoàn thiện bài tập lớn một cách tốt nhất. Chúng em xin chân thành cảm ơn thầy cô trong khoa Công nghệ thông tin đã đã giảng dạy với tất cả tâm huyết, không ngại khó khăn giúp đỡ chúng em trong suốt những năm học vừa qua. Đặc biệt nhóm 13 xin chân thành cảm ơn đến giáo viên ***Nguyễn Thị Mỹ Bình***. Trong suốt kỳ học vừa qua, thực hiện bài tập lớn cô đã luôn giúp đỡ, hướng dẫn tận tình để chúng em có thể hoàn thành bài tập lớn đầu tiên này.

Tròng kỳ học vừa qua. Chúng em đã cố gắng hoàn thiện báo cáo bài tập lớn môn “ Thực tập cơ sở ngành” một cách tốt nhất trong khả năng của mình, tuy nhiên không thể tránh được những thiếu sót về mặt nội dung cũng như trình bày. Chúng em rất mong muốn nhận được sự góp ý của các thầy cô để bài tập lớn này của chúng em được hoàn thiện hơn

Chúng em xin chân thành cảm ơn!

# **MỞ ĐẦU**

Lý thuyết đồ thị là một lĩnh vực khoa học đã có từ lâu và có nhiều ứng dụng hiện đại. Những ý tưởng cơ bản của lý thuyết đồ thị được đề xuất vào những năm đầu của thế kỷ 18 bởi nhà toán học lỗi lạc người Thụy Sĩ Leonhard Euler. Chính ông là người đã sử dụng đồ thị để giải bài toán nổi tiếng về những cái cầu ở thành phố Konigsberg.

Thuật toán tô màu đồ thị là một trong những bài toán cơ bản trong lĩnh vực đồ thị. Ứng dụng thuật toán greedy để giải quyết bài toán tô màu đồ thị là một cách tiếp cận thường được sử dụng. Thuật toán này tập trung vào việc chọn màu cho từng đỉnh sao cho màu của các đỉnh kề nhau không trùng nhau. Với cách tiếp cận này, chúng ta có thể tối ưu hóa số lượng màu cần sử dụng để tô đồ thị mà vẫn đảm bảo tính đúng đắn của giải pháp.

Ngoài ra, thuật toán tô màu đồ thị được sử dụng để giải các bài toán thực tế về lập lịch, lập thời khóa biểu, sắp xếp lịch thi... Cùng với sự phát triển khoa học kỹ thuật và công nghệ thông tin như hiện nay thì lý thuyết đồ thị ngày càng có nhiều ứng dụng trong cuộc sống.

Trên cơ sở lý thuyết đã học, chúng tôi đã nghiên cứu, và muốn tìm hiểu hơn nữa về những ứng dụng hữu ích, thực tế của bài toán tô màu đồ thị. Vì vậy chúng tôi quyết định lựa chọn chủ đề cho bài báo cáo là: **“Áp dụng các thuật toán để giải bài toán tô màu đồ thị”**.

Để hiện thực các giải thuật giải quyết bài toán tô màu đồ thị, chúng tôi sử dụng ngôn ngữ lập trình C++. Đây là một ngôn ngữ lập trình được sử dụng rộng rãi trong lĩnh vực khoa học máy tính và có khả năng xử lý nhanh, đồng thời hỗ trợ nhiều thư viện hữu ích giúp cho việc lập trình trở nên dễ dàng và hiệu quả hơn.

Trong đề tài này, chúng tôi hy vọng sẽ tìm hiểu thêm về các ứng dụng thực tế của bài toán tô màu đồ thị và áp dụng các kiến thức và kỹ năng đã học được để giải quyết các vấn đề trong thực tế. Chúng tôi cũng hy vọng rằng, với sự phát triển của khoa học kỹ thuật và công nghệ thông tin, lý thuyết đồ thị sẽ có nhiều ứng dụng mới và hữu ích trong cuộc sống.

# **CHƯƠNG I: GIỚI THIỆU CHUNG**

# **1.1. Tên đề tài**

* “Áp dụng các thuật toán để giải bài toán tô màu đồ thị”.

# **1.2. Lý do chọn đề tài**

* Bài toán tô màu đồ thị là một bài toán cơ bản trong lý thuyết đồ thị và có tầm quan trọng to lớn trong nhiều lĩnh vực khác nhau, bao gồm khoa học máy tính, mạng lưới, tối ưu hóa, vật lý, sinh học, kinh tế và xã hội học. Ví dụ, trong khoa học máy tính, bài toán tô màu đồ thị được sử dụng rộng rãi để thiết kế và phân tích các thuật toán tìm kiếm đường đi ngắn nhất, phân tích độ phức tạp thuật toán, xác định tối ưu hoá bộ phận mạng và phát hiện lỗi mạng. Trong sinh học, bài toán tô màu đồ thị giúp phân tích sự tương tác giữa các phân tử và tìm kiếm các cấu trúc phức tạp trong protein và DNA.
* Với sự quan trọng của bài toán, ngày nay, nhiều nghiên cứu cố gắng tìm kiếm lời giải tối ưu cho bài toán này, dẫn đến việc phát triển và ứng dụng rộng rãi các thuật toán giải quyết vấn đề tô màu đồ thị. Việc áp dụng các thuật toán này mang lại nhiều giá trị thực tiễn và là một chủ đề nghiên cứu hấp dẫn. Nhờ kinh nghiệm tính toán và khả năng xử lý dữ liệu của máy tính, các thuật toán tô màu đồ thị có thể giúp tìm ra lời giải tối ưu và nhanh chóng cho rất nhiều bài toán ứng dụng, tiết kiệm thời gian và tài nguyên. Hơn nữa, việc áp dụng các thuật toán này còn giúp chúng ta hiểu thêm về cấu trúc và tính chất của đồ thị, đóng góp vào việc phát triển lý thuyết đồ thị.
* Nhóm chúng em quyết định chọn đề tài "Áp dụng các thuật toán để giải bài toán tô màu đồ thị" vì nó là một lựa chọn hợp lý và mang lại nhiều giá trị trong việc nghiên cứu và ứng dụng trong thực tiễn.

# **1.3. Công cụ sử dụng**

# **1.3.1. Ngôn ngữ lập trình C++**

* C++ là một ngôn ngữ lập trình phổ biến và mạnh mẽ được sử dụng cho nhiều mục đích, bao gồm cả giải quyết các bài toán đồ thị.
* Một số lý do tại sao C++ là một lựa chọn phổ biến cho giải bài toán đồ thị bao gồm:

1. Hiệu suất: C++ là một ngôn ngữ mạnh với nhiều tính năng tối ưu hóa, giúp cải thiện hiệu suất của chương trình. Điều này đặc biệt quan trọng trong việc xử lý các bài toán có kích thước lớn hoặc khi cần sử dụng các thuật toán phức tạp.

2. Sự linh hoạt: C++ cung cấp nhiều tính năng để thao tác và xử lý bài toán đồ thị, bao gồm các thư viện đồ thị tiêu chuẩn như Standard Template Library (STL) và C++ Graph Template Library (Boost.Graph), cùng với khả năng tùy chỉnh và mở rộng để tương thích với các yêu cầu cụ thể.

3. Khả năng tái sử dụng mã: C++ là một ngôn ngữ lập trình hướng đối tượng cho phép viết mã dễ sử dụng và tái sử dụng. Điều này có nghĩa là bạn có thể xây dựng một số lượng mã cơ bản có thể được sử dụng cho nhiều bài toán đồ thị khác nhau.

4. Chỉnh sửa và bảo trì: Khi giải quyết một bài toán đồ thị, việc phát hiện và sửa lỗi là rất quan trọng. C++ cung cấp các tính năng để tạo ra các lỗi chương trình nhanh chóng và gỡ rối chúng cũng rất dễ dàng.

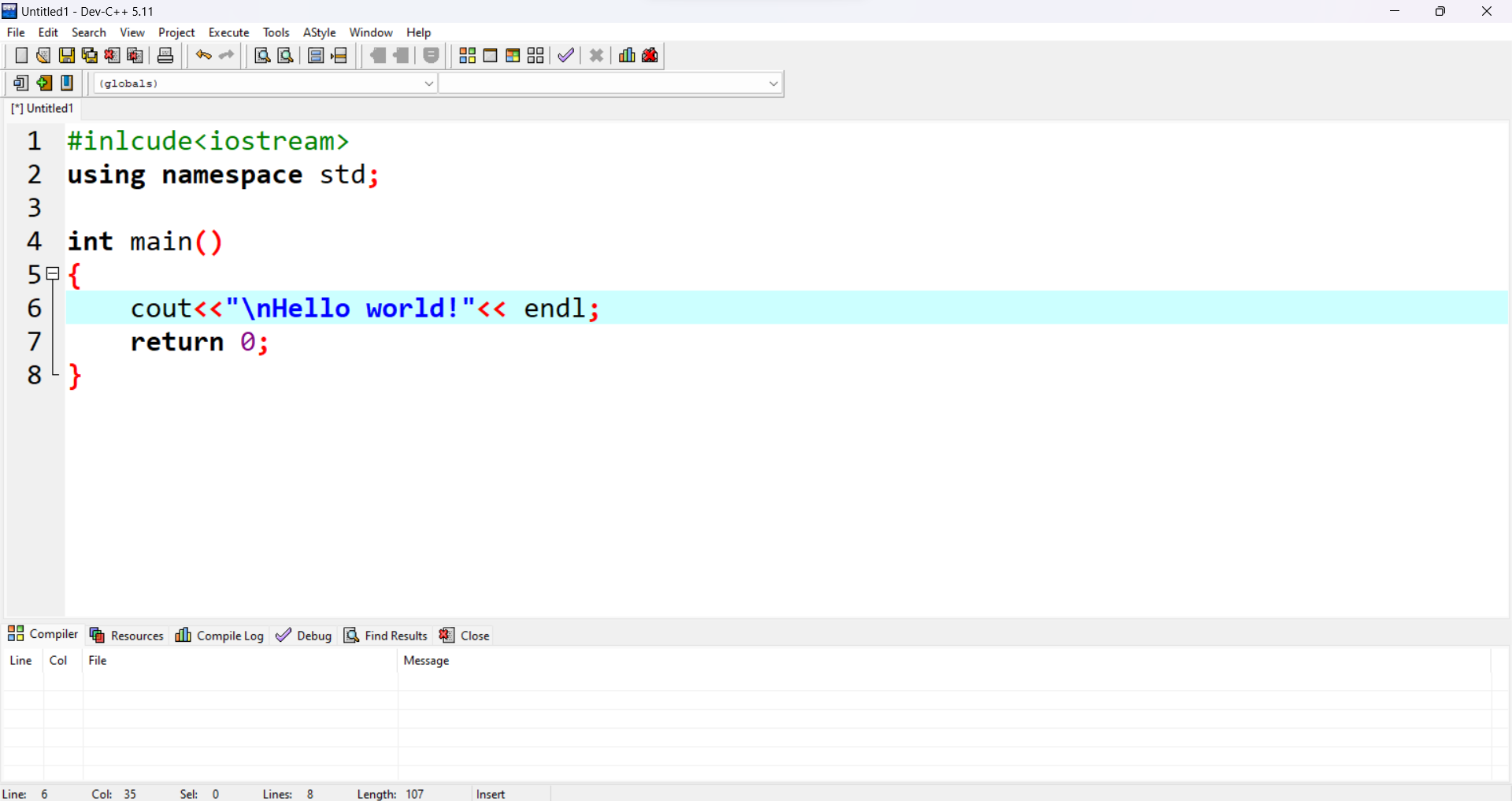
* Vì những lý do này và nhiều lý do khác, C++ là một lựa chọn phổ biến cho việc giải quyết các bài toán đồ thị. Nó cung cấp nhiều tính năng để tạo các chương trình có hiệu quả và có thể dễ dàng bảo trì.

# **1.3.2. Dev-C++**

* Dev C++ là một môi trường phát triển tích hợp tự do (IDE), hỗ trợ việc lập trình bằng ngôn ngữ C hay C++. Dev C++ được phát triển bởi lập trình viên Colin Laplace và được viết bằng ngôn ngữ lập trình Delphi.
* Dev C++ nằm trong bộ trình dịch mã nguồn mở MinGW và chỉ hỗ trợ trên hệ điều hành Microsoft Windows. Nhà sản xuất cũng đã cho ra đời một phiên bản chạy trên Linux tuy nhiên vẫn đang nằm trong giai đoạn thử nghiệm.
* Một điểm đáng chú ý của phần mềm này chính là tích hợp DevPaks, bao gồm các gói mở rộng như thư viện, mẫu và nhiều tiện ích khác. Thông thường, DevPaks sở hữu thêm tiện ích (GUI) với các công cụ phổ biến như GTK+, FLTK và wxWidgets. Từ năm 2005 đến nay, phần mềm này chưa được cập nhật lên phiên bản mới nhất. Tuy nhiên, phiên bản cũ cũng đã sở hữu đầy đủ tính năng để bạn có thể lập trình bằng ngôn ngữ C++.



Hình 1.5.2.1: Biểu tượng icon Dev C++

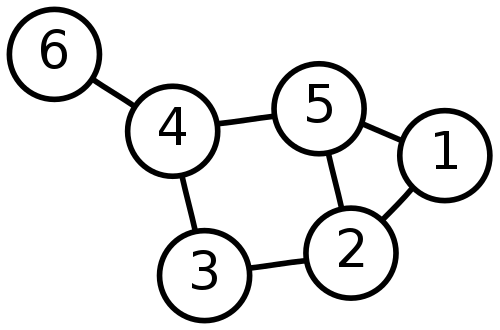


Hình 1.5.2.2: Hình ảnh giao diện Dev C++

**CHƯƠNG II: CƠ SỞ LÝ THUYẾT**

# **2.1. Khái niệm đồ thị**

* Đồ thị là một cách biểu diễn trực quan của các đối tượng và mối quan hệ giữa chúng. Nó bao gồm một tập hợp các đối tượng được gọi là đỉnh hoặc nút, và một tập hợp các cạnh hoặc liên kết giữa các đỉnh. Các đỉnh thường được biểu diễn bằng các hình tròn hoặc hình vuông, và các cạnh được biểu diễn bằng các đường thẳng hoặc cong nối giữa các đỉnh.
* Đồ thị được sử dụng rộng rãi trong nhiều lĩnh vực, chẳng hạn như khoa học máy tính, mạng lưới, toán học, kinh tế học, v.v. Nó giúp mô hình hóa và phân tích các quan hệ phức tạp giữa các đối tượng, cho phép ta tìm kiếm thông tin, định vị và giải quyết các vấn đề phức tạp.
* Ví dụ:



Hình 2.1.1: Hình ảnh minh họa đồ thị

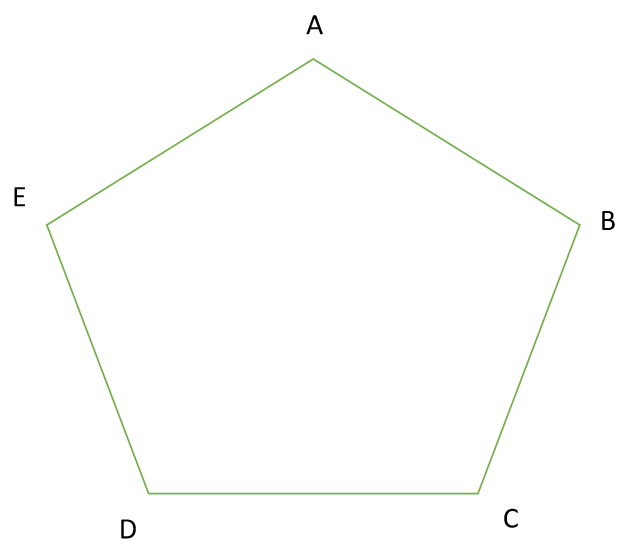
# **2.2. Đường đi, chu trình, đồ thị liên thông, đồ thị phẳng**

# **2.2.1. Đường đi, chu trình:**

# **2.2.1.1. Đường đi**

**- Khái niệm:** Đường đi trong đồ thị là một chuỗi các đỉnh mà từ một đỉnh đến đỉnh kế tiếp đều có một cạnh nối. Cụ thể, đường đi trong đồ thị G được định nghĩa là một chuỗi đỉnh (v1, v2, v3,..., vk) sao cho (vi, vi+1) là một cạnh của G với i = 1, 2, ..., k-1.

**- Hình vẽ:** Hình vẽ của một đường đi trong đồ thị có thể được biểu diễn bởi một loạt các đỉnh được nối với nhau bởi các cạnh. Ví dụ, đường đi từ đỉnh A đến đỉnh E trong hình dưới đây được biểu diễn bởi chuỗi các đỉnh A, B, C, D,E:

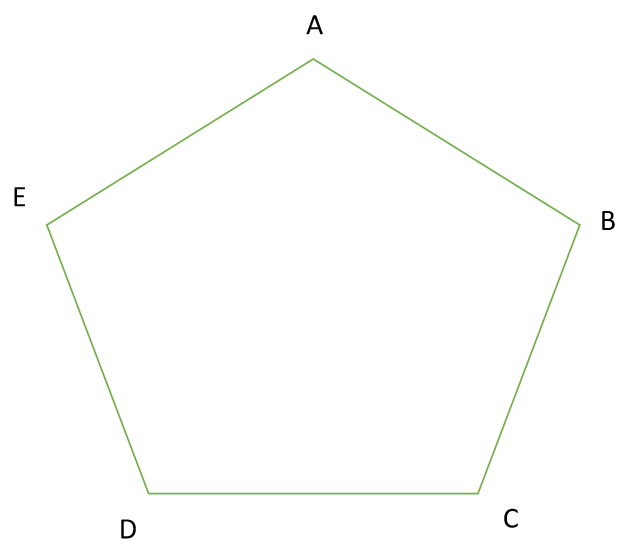
****

**- Định lý:**  Một đồ thị G có một đường đi duy nhất từ đỉnh u đến đỉnh v khi và chỉ khi không có chu trình nào đi qua u và v. Định lý này nói rằng, nếu tồn tại một đường đi từ đỉnh u đến đỉnh v trong một đồ thị G mà không có chu trình nào đi qua qua u và v thì đường đi đó sẽ là đường đi đơn, tức là không biết đỉnh. lặp lại trong con đường đó.

# **2.2.1.2. Chu trình**

**- Khái niệm:** Chu trình trong đồ thị là một đường đi đơn bắt đầu và kết thúc tại cùng một đỉnh. Cụ thể, chu trình trong đồ thị G được định nghĩa là một chuỗi đỉnh (v1, v2, v3,..., vk, v1) sao cho (vi, vi+1) là một cạnh của G với i = 1, 2, ..., k-1 và (vk, v1) cũng là một cạnh của G.

**- Hình vẽ:** Hình vẽ của một chu trình trong đồ thị có thể được biểu diễn bởi một chuỗi các đỉnh được nối với nhau bởi các cạnh, bắt đầu và kết thúc tại cùng một đỉnh. Ví dụ, chu trình từ đỉnh A đến đỉnh A trong hình dưới đây được biểu diễn bởi chuỗi các đỉnh A, B, C, D, E,A:

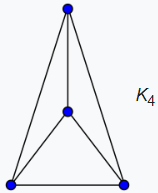
****

**- Định lý:** Một đồ thị G có chu trình đơn nếu và chỉ nếu không có hai đường đi khác nhau giữa hai đỉnh trong chu trình đó. Định lý này cho biết rằng, nếu có một chu trình trong đồ thị G mà không có hai đường đi khác nhau giữa hai đỉnh trong chu trình đó, thì chu trình đó sẽ là chu trình đơn, tức là không có đỉnh nào được lặp lại trong chu trình đó ngoại trừ đỉnh bắt đầu và kết thúc.

# **2.2.2. Đồ thị phẳng, đồ thị liên thông:**

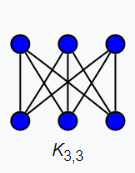
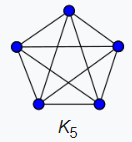
# **2.2.2.1. Đồ thị phẳng**

* **Khái niệm:** Một đồ thị được gọi là một đồ thị phẳng nếu ta có thể vẽ nó trên mặt phẳng sao cho các cạnh của nó không cắt nhau ngoài ở đỉnh. Cách vẽ như vậy sẽ được gọi là biểu diễn phẳng của đồ thị.



*Hình 2.2.2.1: Ví dụ đồ thị phẳng*

* Điều kiện cần và đủ để đồ thị là phẳng được chỉ ra trong định lý Kuratowski:
* *Đồ thị là phẳng khi và chỉ khi nó không chứa đồ thị con đồng phôi với K3,3 hoặc K5.*

*Hình 2.2.2.2: Đồ thị không phẳng*

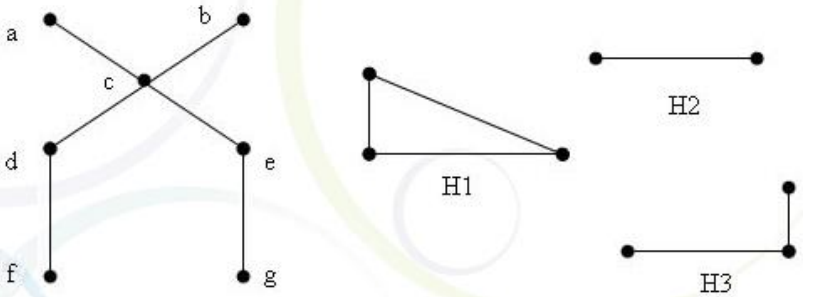
* Trong thực tế việc sử dụng định lý Kuratowski để kiểm tra đồ thị có phải là đồ thị phẳng hay không thì rất khó khăn. Tuy nhiên, tồn tại thuật toán để kiểm tra vấn đề này. Xét đồ thị phẳng với n đỉnh và p cạnh, ta có:
* Định lý 1. *Nếu n ≥ 3 thì p ≤ 3n - 6.*
* Định lý 2. *Nếu n ≥ 3 và không có chu trình có độ*

*dài 3, thì p ≤ 2n - 4.*

# **2.2.2.2. Đồ thị liên thông**

* **Khái niệm:** Một đồ thị được gọi là liên thông nếu có đường đi giữa mọi cặp đỉnh phân biệt của đồ thị. Ngược lại, đồ thị này được gọi là không liên thông.

Đồ thị vô hướng G = (V, E) được gọi là liên thông nếu luôn tìm được đường đi giữa hai đỉnh bất kỳ của nó.



*Hình 2.2.2.3: Đồ thị G và H*

Đồ thị G là liên thông, còn đồ thị H là không liên thông.

* Ta gọi đồ thị con của đồ thị G = (V, E) là đồ thị H = (W, F), trong đó WV và FE.

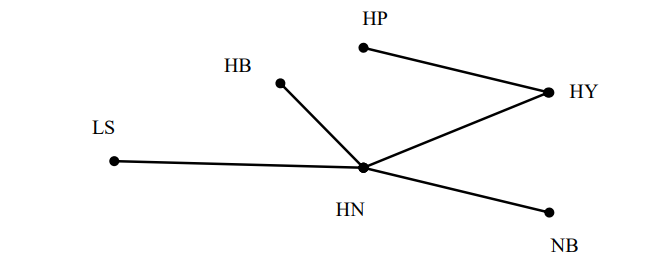
Trong trường hợp đồ thị là không liên thông, nó sẽ rã ra thành một số đồ thị con liên thông không có đỉnh chung. Những đồ thị con liên thông như vậy ta sẽ gọi là các thành phần liên thông của đồ thị.

Ví dụ: Đồ thị H trong hình 2.2.2.3 gồm 3 thành phần liên thông H1, H2, H3.

# **2.3. Đơn đồ thị vô hướng liên thông**

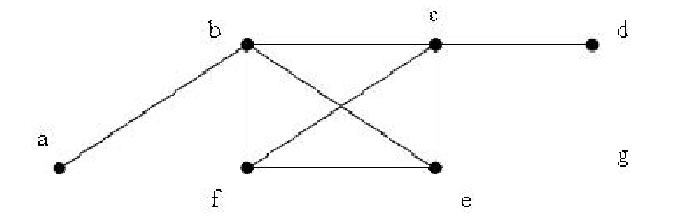
* **Định nghĩa:** Đơn đồ thị vô hướng G = (V,E) bao gồm V là tập các đỉnh, và E là tập các cặp không có thứ tự gồm hai phần tử khác nhau của V gọi là các cạnh.
* Người ta thường ký hiệu đồ thị là G (Graph), tập đỉnh là V (Vertex),và tập cạnh là E (edge).

Ví dụ: Mạng lưới giao thông thể hiện giữa 2 địa điểm khác nhau có đường đi trực tiếp đến nhau hay không, là một Đơn đồ thị vô hướng.



Hình 2.3.1: Đơn đồ thị vô hướng

* **Định nghĩa** Hai đỉnh u và v của đồ thị vô hướng G được gọi là kề nhau nếu (u,v) là cạnh của đồ thị G. Nếu e = (u, v) là cạnh của đồ thị ta nói cạnh này là liên thuộc với hai đỉnh u và v, hoặc cũng nói là nối đỉnh u và đỉnh v, đồng thời các đỉnh u và v sẽ được gọi là các đỉnh đầu của cạnh (u, v).
* **Định nghĩa** Ta gọi bậc của đỉnh v trong đồ thị vô hướng là số cạnh kề với v và sẽ ký hiệu là deg(v).



*Hình 3. Đồ thị vô hướng.*

**Ví dụ** . Xét đồ thị cho trong hình 3, ta có

deg(a) = 1, deg(b) = 4, deg(c) = 4, deg(f) = 3,

deg(d) = 1, deg(e) = 3, deg(g) = 0

Đỉnh bậc 0 gọi là đỉnh cô lập. Đỉnh bậc 1 được gọi là đỉnh treo. Trong ví dụ trên đỉnh g là đỉnh cô lập, a và d là các đỉnh treo.

**Định lý :** Giả sử G = (V, E) là đồ thị vô hướng với m cạnh. Khi đó tổng bậc của tất cả các đỉnh bằng hai lần số cung.

Thật vậy: Giả sử G = (V, E) là đồ thị vô hướng có m cạnh. Rõ ràng mỗi cạnh e = (u, v) được tính một lần trong deg(u) và một lần trong deg(v). Từ đó suy ra tổng tất cả các bậc của các đỉnh bằng hai lần số cạnh. Tức là deg(v) = 2|E| = 2m

**Hệ quả :** Trong đồ thị vô hướng, số đỉnh bậc lẻ (nghĩa là có bậc là số lẻ) là một số chẵn.

Thật vậy: Gọi L và C tương ứng là tập đỉnh bậc lẻ và tập đỉnh bậc chẵn của đồ thị. Ta có 2m = deg(v) + deg(v)

vC vL

Do deg(v) là chẵn với v là đỉnh trong C nên tổng thứ nhất ở trên là số chẵn. Từ đó suy ra tổng thứ hai (chính là tổng bậc của các đỉnh bậc lẻ) cũng phải là số chẵn, do tất cả các số hạng của nó là số lẻ, nên tổng này phải gồm một số chẵn các số hạng. Vì vậy, số đỉnh bậc lẻ phải là số chẵn.

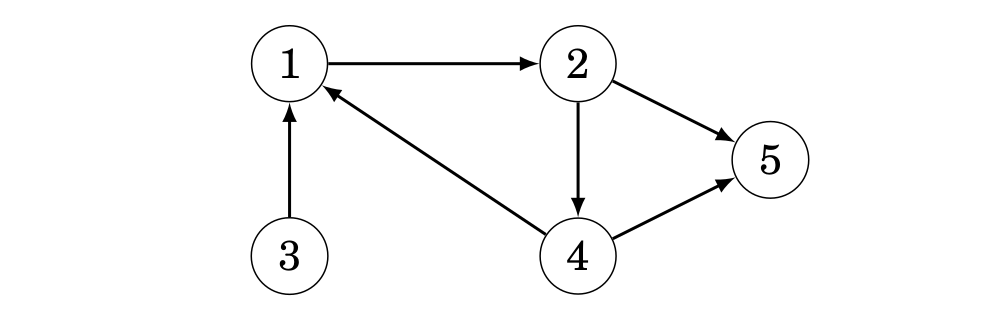
**Định nghĩa:** Đồ thị con:

* G’ được gọi là đồ thị con của G, ký hiệu G’ ≤ G nếu V’⊆ Vvà E’⊆ E.
* Nếu V’=V và E’⊆E thì G’ được gọi là đồ thị con khung của G .

**Định nghĩa :**Liên thông: Đồ thị G=(V,E) được gọi là liên thông nếu có đường đi giữa mọi cặp đỉnh u, v bất kỳ trong G.

# **2.4. Đồ thị có hướng liên thông**

* **Khái niệm:** Đồ thị có hướng liên thông là đồ thị mà từ mỗi đỉnh trong đó đều có đường đi đến tất cả các đỉnh khác trong đồ thị đó. Tức là, đồ thị có hướng liên thông là đồ thị mà không có đỉnh nào bị cô lập và không liên thông với các đỉnh khác.
* **Hình vẽ:** Hình vẽ của đồ thị có hướng liên thông có thể được biểu diễn bằng các đỉnh và cạnh được nối với nhau theo hướng từ một đỉnh đến các đỉnh khác trong đồ thị. Ví dụ, đồ thị có hướng liên thông dưới đây có 5 đỉnh và 6 cạnh:



**- Định lý:**

+) Một đồ thị có hướng G được gọi là có thành phần liên thông mạnh nếu và chỉ nếu từ mỗi đỉnh trong G đều có đường đi đến tất cả các đỉnh khác trong G và ngược lại.

+) Định lý này cho biết rằng, nếu đồ thị G có thành phần liên thông mạnh, thì từ mỗi đỉnh trong G đều có thể đi đến tất cả các đỉnh khác trong G và ngược lại. Nếu đồ thị không có thành phần liên thông mạnh, nghĩa là có ít nhất một đỉnh không thể đi đến một số đỉnh khác trong G hoặc ngược lại, thì đồ thị đó không là đồ thị có hướng liên thông.

# 

# 

# 

# **CHƯƠNG III : ÁP DỤNG CÁC THUẬT TOÁN ĐỂ CÀI ĐẶT BÀI TOÁN TÔ MÀU ĐỒ THỊ**

# **3.1. Phát biểu bài toán**

# **3.1.1. Bài toán**

Bài toán tô màu đồ thị là bài toán tìm cách gán màu cho các đỉnh của đồ thị sao cho hai đỉnh kề nhau không có cùng màu. Nói cách khác, mỗi đỉnh của đồ thị được gán một màu, sao cho các đỉnh kề nhau có màu khác nhau.

Bài toán tô màu đồ thị là một trong những bài toán quan trọng trong lĩnh vực đồ thị và được ứng dụng rộng rãi trong nhiều lĩnh vực khác nhau như mạng máy tính, lập trình phân tán, lập lịch và xếp hàng.

Bài toán tô màu đồ thị được phát biểu như sau: Cho một đồ thị G = (V, E), hãy tìm cách gán màu cho các đỉnh của G sao cho mỗi đỉnh được gán một màu và hai đỉnh kề nhau không có cùng màu.

# **3.1.2. Xác định đầu vào, đầu ra của bài toán:**

**Đầu vào:** Đồ thị G = (V, E), trong đó V là tập hợp các đỉnh và E là tập hợp các cạnh của đồ thị. Để đơn giản hóa, ta cũng có thể giả sử rằng đồ thị G là đồ thị vô hướng, tức là mỗi cạnh chỉ có thể nối giữa hai đỉnh duy nhất.

**Đầu ra:** Là một phân hoạch của tập đỉnh V thành các tập con có kích thước bằng nhau hoặc không bằng nhau, mỗi tập con được gọi là một lớp màu. Mỗi đỉnh trong đồ thị được gán một màu sao cho đỉnh kề nhau không có cùng màu và số lượng lớp màu được sử dụng là nhỏ nhất.

Tùy vào cách biểu diễn và giải quyết bài toán, đầu ra có thể được biểu diễn dưới dạng danh sách các đỉnh với màu tương ứng hoặc dưới dạng ma trận kề biểu diễn mối liên hệ giữa các đỉnh và màu được gán cho chúng.

# **3.1.3. Mô hình hóa bài toán**

Bài toán tô màu đồ thị có thể được mô hình hóa bằng cách sử dụng các biến quyết định và ràng buộc. Mô hình hóa cơ bản nhất có thể được mô tả như sau:

1. Biến quyết định: Mỗi đỉnh của đồ thị được gán một biến quyết định, giá trị của biến này là màu được gán cho đỉnh đó. Các biến này được đặt tên là x\_1, x\_2, ..., x\_n, tương ứng với các đỉnh của đồ thị.

2. Tập giá trị của biến quyết định: Các biến quyết định được giới hạn trong một tập giá trị có kích thước bằng số lượng màu tối đa được sử dụng. Tập giá trị này được đặt tên là {1, 2, ..., k}, tương ứng với các màu khác nhau được sử dụng.

3. Ràng buộc: Ràng buộc được đặt ra bằng cách yêu cầu các biến quyết định của các đỉnh kề nhau có giá trị khác nhau. Các ràng buộc này có thể được biểu diễn bằng cách sử dụng các phương trình hoặc bất phương trình. Ví dụ, nếu đỉnh i và đỉnh j kề nhau, thì ta có ràng buộc:

**x\_i ! = x\_j**

4. Mục tiêu: Mục tiêu của bài toán là tìm cách gán màu cho các đỉnh của đồ thị sao cho số lượng màu được sử dụng là ít nhất.

Sau khi xác định các biến, tập giá trị của biến và ràng buộc, ta có thể sử dụng các phương pháp tối ưu hóa hoặc các giải thuật để giải quyết bài toán.

# **3.2. Thuật toán tham lam**

# **3.2.1. Giới thiệu chung**

* Khái niệm**:** Thuật toán tô màu tham lam xem xét đồ thị G(V) với tập hợp các đỉnh V=[v1, v2,...vn] và tập các đỉnh kề Avj. Đầu tiên ta xét các đỉnh theo thứ tự và gán cho mỗi đỉnh một màu riêng theo nguyên tắc: các đỉnh không kề với đỉnh đang xét (không có cạnh nối trực tiếp) thì được phép tô cùng một màu, cấm tô màu đó cho các đỉnh có cạnh kề với đỉnh đang xét. Thuật toán lặp lại cho đến khi tất cả các đỉnh được tô màu.

# 

* Ứng dụng**:** Giải thuật Greedy algorithm được ứng dụng rộng rãi trong nhiều lĩnh vực khác nhau như:

1. Tối ưu hóa về chi phí, thời gian, năng suất sản xuất, hiệu quả sử dụng tài nguyên trong các công ty, doanh nghiệp và các lĩnh vực kinh doanh.

2. Điều khiển, quản lý các mạng máy tính, các hệ thống truyền thông và các hệ thống quản lý kho.

3. Tự động đăng ký phân công công việc cho các công nhân, tài xế, nhân viên văn phòng,...

4. Tối ưu hóa các hệ thống lựa chọn và sắp xếp, ví dụ như tìm kiếm và giải quyết bài toán ghép đôi, phân bổ đồ vật trong một cây túi,...

5. Giải quyết các vấn đề trong các lĩnh vực kỹ thuật, từ viễn thông, điện tử, thống kê, hóa học, sinh học đến y tế, giáo dục,…

* Giải thuật Greedy algorithm có thể được sử dụng độc lập hoặc kết hợp với các giải thuật khác để giải quyết các vấn đề khác nhau. Việc lựa chọn và áp dụng đúng giải thuật này sẽ đem lại những kết quả tốt nhất theo những yêu cầu của mỗi bài toán.

# **3.2.2. Lưu đồ thuật toán**

Thuật toán Greedy algorithm được thực hiện theo các bước sau:

- Bước 1: Sắp xếp các đỉnh theo thứ tự giảm dần của bậc

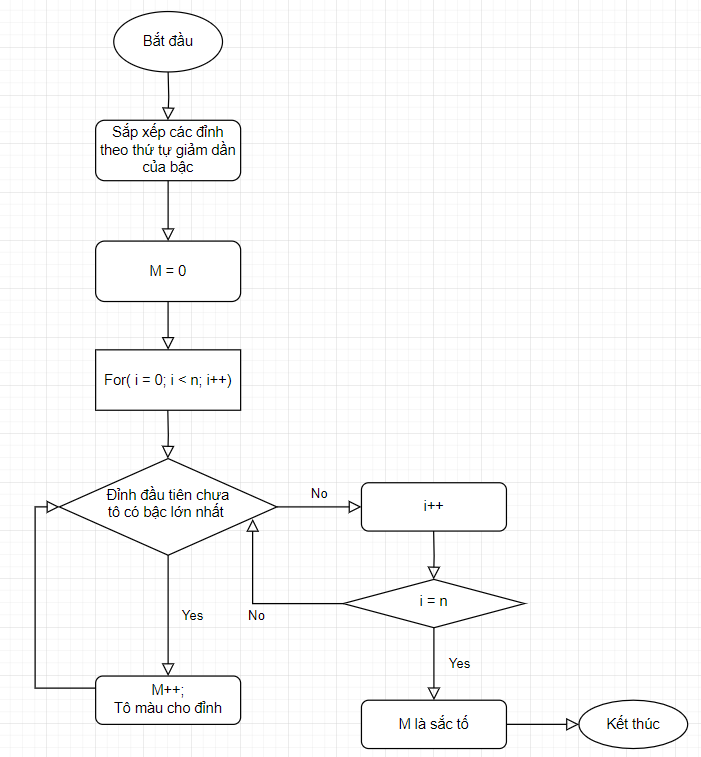
- Bước 2: Bắt đầu từ màu M=0;

- Bước 3: Làm đi làm lại các việc sau:

+) Tìm đỉnh đầu tiên chưa tô có bậc lớn nhất, nếu tìm được thì tăng M lên 1, ngược lại thì kết thúc

+) Tô đỉnh đó bằng màu M.

+) Tìm các đỉnh chưa tô mà không kề với các đỉnh có màu M, nếu tìm được thì tô đỉnh đó bằng màu M.

- Kết luận M là sắc số của đồ thị. 

Hình 3.2.2.1: Lưu đồ thuật toán tham lam

# **3.2.3. Cài đặt thuật toán**

#include <iostream>

#include <vector>

#include <set>

using namespace std;

vector<vector<int> > readGraph() {

int numVertices, numEdges;

cout << "Nhap so dinh cua do thi: ";

cin >> numVertices;

cout << "Nhap so canh cua do thi: ";

cin >> numEdges;

vector<vector<int> > graph(numVertices);

cout << "Nhap cac canh cua do thi:\n";

for (int i = 0; i < numEdges; i++) {

int u, v;

cout << "Canh " << i + 1 << ": ";

cin >> u >> v;

graph[u - 1].push\_back(v - 1);

graph[v - 1].push\_back(u - 1);

}

return graph;

}

void printColors(const vector<int>& colors) {

cout << "\nMau cua cac dinh:\n";

for (int vertex = 0; vertex < colors.size(); vertex++) {

cout << "Dinh " << vertex + 1 << ": Mau " << colors[vertex] << "\n";

}

}

vector<int> greedyColoring(const vector<vector<int> >& graph) {

int numVertices = graph.size();

vector<int> vertexColors(numVertices, -1);

set<int> usedColors;

int numUsedColors = 0;

for (int vertex = 0; vertex < numVertices; vertex++) {

for (int i = 0; i < graph[vertex].size(); i++) {

int neighbor = graph[vertex][i];

if (vertexColors[neighbor] != -1) {

usedColors.insert(vertexColors[neighbor]);

}

}

for (int color = 0; color < numVertices; color++) {

if (usedColors.find(color) == usedColors.end()) {

vertexColors[vertex] = color;

numUsedColors = max(numUsedColors, color + 1);

break;

}

}

usedColors.clear();

}

cout << "\nSo mau can dung: " << numUsedColors << "\n";

return vertexColors;

}

int main() {

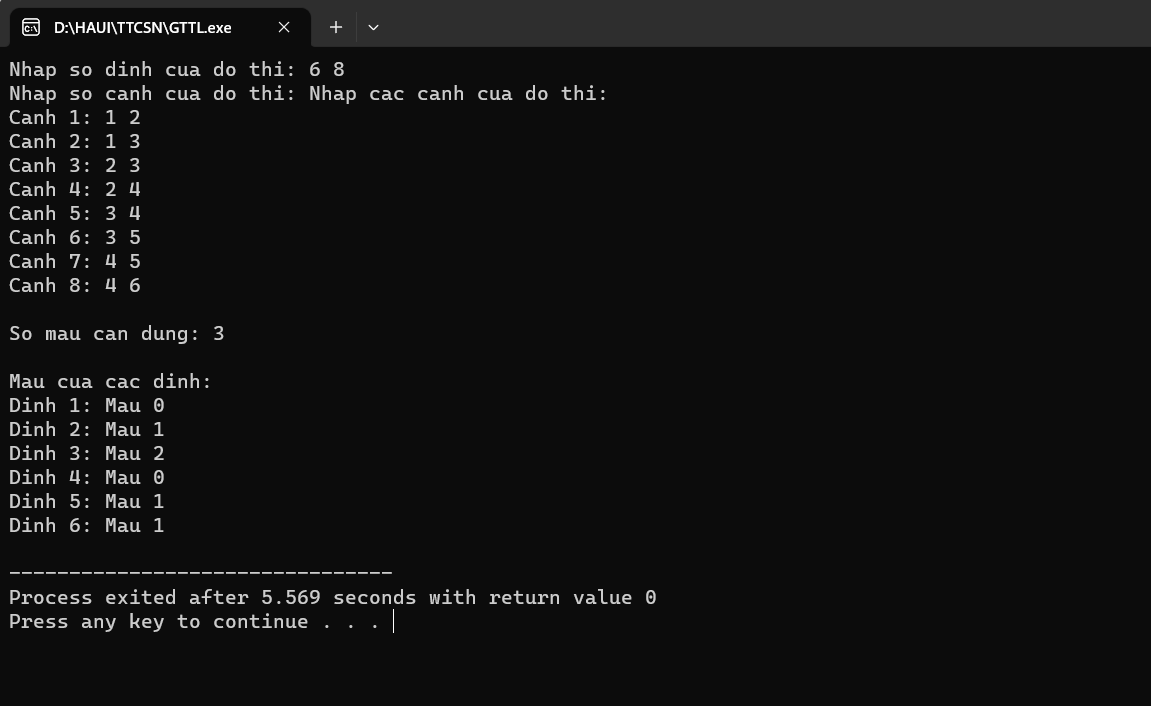
vector<vector<int> > graph = readGraph();

vector<int> colors = greedyColoring(graph);

printColors(colors);

return 0;

}



Hình 3.2.3.1: Hình ảnh minh họa kết quả của một trường hợp

# **3.2.4. Đánh giá thuật toán**

* Các ưu điểm của thuật toán Greedy bao gồm:
* Đơn giản và dễ hiểu: Thuật toán Greedy có cấu trúc rõ ràng và đơn giản, dễ dàng hiểu và cài đặt. Điều này giúp cho việc áp dụng thuật toán Greedy trở nên thuận tiện và nhanh chóng.
* Tính hiệu quả cao: Trong một số trường hợp, thuật toán Greedy có thể cho kết quả tối ưu hoặc gần tối ưu trong thời gian ngắn. Việc lựa chọn phương án tốt nhất tại từng bước giúp giảm thiểu thời gian tính toán và tăng hiệu quả của thuật toán.
* Tính khả thi cao: Thuật toán Greedy thường được sử dụng để giải quyết các bài toán tối ưu đơn giản và có cấu trúc rõ ràng, trong đó việc tính toán giá trị tối ưu có thể được thực hiện một cách dễ dàng. Điều này giúp cho việc áp dụng thuật toán Greedy trở nên khả thi và dễ dàng.
* Tính linh hoạt cao: Thuật toán Greedy có thể được sử dụng để giải quyết nhiều loại bài toán tối ưu khác nhau, từ các bài toán đơn giản đến các bài toán phức tạp hơn.
* Tuy nhiên, cũng cần lưu ý rằng thuật toán Greedy cũng có một số hạn chế, đặc biệt là không phải lúc nào cũng cho kết quả tối ưu và có thể bị ảnh hưởng bởi các yếu tố không đồng nhất.
* Mặc dù thuật toán Greedy rất đơn giản và hiệu quả trong nhiều trường hợp, tuy nhiên nó cũng có một số nhược điểm như sau:
* Không luôn tìm được giải pháp tối ưu: Thuật toán Greedy chỉ đảm bảo tìm được giải pháp tối ưu trong một số trường hợp đặc biệt. Trong nhiều bài toán phức tạp, thuật toán Greedy có thể dẫn đến việc tìm được giải pháp chưa tối ưu.
* Phụ thuộc vào thứ tự xử lý: Kết quả của thuật toán Greedy phụ thuộc vào thứ tự xử lý các phần tử trong dữ liệu đầu vào. Điều này có thể dẫn đến việc tìm được các giải pháp khác nhau tùy thuộc vào thứ tự xử lý.
* Không thể giải quyết được một số bài toán phức tạp: Thuật toán Greedy không phải là giải pháp cho tất cả các bài toán. Trong một số trường hợp, nó có thể không thể giải quyết được các bài toán phức tạp hoặc các bài toán có ràng buộc phức tạp.
* Độ phức tạp tính toán không được tối ưu: Trong một số trường hợp, thuật toán Greedy có thể dẫn đến độ phức tạp tính toán không được tối ưu. Việc xử lý các phần tử của dữ liệu đầu vào một cách tương đối đơn giản có thể dẫn đến việc tăng độ phức tạp tính toán.
* Không đảm bảo tính khả thi của giải pháp: Thuật toán Greedy không đảm bảo tính khả thi của giải pháp. Trong một số trường hợp, giải pháp được tìm thấy bởi thuật toán có thể không đáp ứng được các ràng buộc của bài toán.

# **3.3. Thuật toán thu gọn (hòa nhập) đỉnh (contraction algorithm)**

# **3.3.1. Giới thiệu chung**

Thuật toán thu gọn đỉnh (contraction algorithm) là một thuật toán được sử dụng trong lý thuyết đồ thị để giảm kích thước của đồ thị bằng cách gộp các đỉnh thành một đỉnh duy nhất. Phương pháp này giúp giảm độ phức tạp tính toán và cải thiện hiệu suất của các thuật toán trên đồ thị.

Thuật toán thu gọn đỉnh thường được áp dụng khi ta có một đồ

thị lớn và muốn giảm số lượng đỉnh trong đồ thị đó. Ý tưởng chính của thuật

toán là thực hiện các bước gộp đỉnh sao cho đồ thị sau khi thu gọn vẫn giữ được các tính chất quan trọng của đồ thị gốc.

# **3.3.2 Lưu đồ thuật toán**

* Thuật toán thu gọn (hòa nhập) đỉnh (contraction algorithm) được thực hiện theo các bước sau:
* Bước 1: Khởi tạo đồ thị gốc và đồ thị thu gọn ban đầu là đồ thị gốc.
* Bước 2: Lặp lại các bước sau cho đến khi không còn đỉnh nào có thể thu gọn nữa:
* Chọn một đỉnh trong đồ thị thu gọn và tìm các đỉnh kề với nó trong đồ thị gốc.
* Xác định các cạnh kết nối từ các đỉnh kề này đến các đỉnh khác trong đồ thị thu gọn.
* Gộp các đỉnh kề và đỉnh gốc thành một đỉnh mới trong đồ thị thu gọn, và cập nhật các cạnh kết nối theo cách thích hợp.
* Bước 3: Đồ thị thu gọn sau khi hoàn thành các bước trên là đồ thị đã được thu gọn. Thực hiện thuật toán tô màu trên đồ thị này để tô màu các đỉnh.

# **3.3.3. Cài đặt thuật toán**

#include <iostream>

#include <vector>

using namespace std;

// Hàm thu gọn đỉnh

void contractVertex(vector<vector<int>>& graph, int u, int v) {

int n = graph.size();

// Hợp nhất các cạnh kết nối với đỉnh u và v

for (int i = 0; i < n; i++) {

if (i != u && i != v) {

graph[u][i] += graph[v][i];

graph[i][u] += graph[i][v];

}

}

// Di chuyển đỉnh cuối cùng của đồ thị lên vị trí đỉnh u

graph[u][u] = 0;

for (int i = 0; i < n; i++) {

graph[i].erase(graph[i].begin() + v);

}

graph.erase(graph.begin() + v);

}

// Hàm in đồ thị

void printGraph(const vector<vector<int>>& graph) {

int n = graph.size();

for (int i = 0; i < n; i++) {

for (int j = 0; j < n; j++) {

cout << graph[i][j] << " ";

}

cout << endl;

}

}

int main() {

// Khởi tạo đồ thị

vector<vector<int>> graph = {

{0, 1, 1, 0, 0},

{1, 0, 1, 1, 1},

{1, 1, 0, 0, 1},

{0, 1, 0, 0, 1},

{0, 1, 1, 1, 0}

};

int u = 2; // Đỉnh u cần thu gọn

int v = 4; // Đỉnh v cần thu gọn

cout << "Graph before contraction:" << endl;

printGraph(graph);

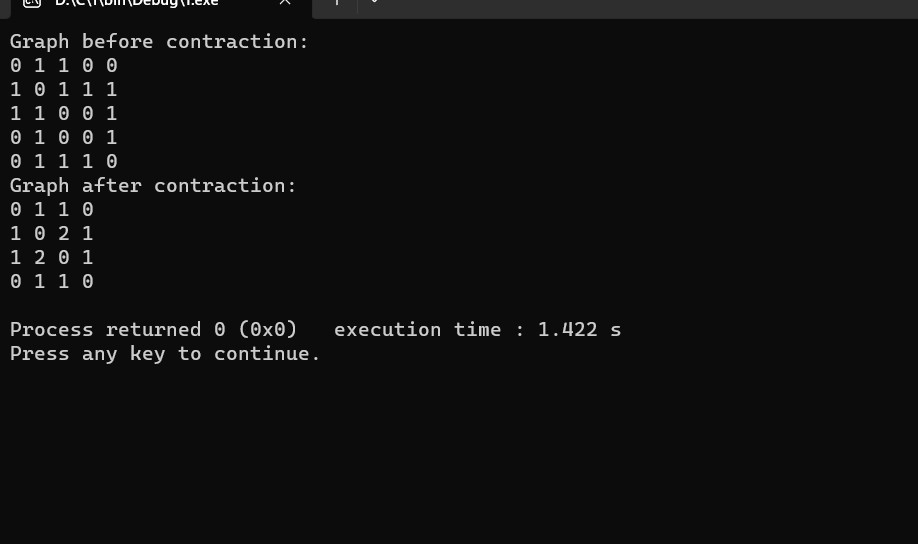
contractVertex(graph, u, v);

cout << "Graph after contraction:" << endl;

printGraph(graph);

return 0;

}



# **3.3.4. Đánh giá thuật toán**

1. Ưu điểm của thuật toán thu gọn (hòa nhập) đỉnh là:

* Giảm kích thước đồ thị: Thuật toán giúp giảm số lượng đỉnh trong đồ thị, làm cho đồ thị trở nên nhỏ hơn. Điều này giảm bộ nhớ và tài nguyên tính toán cần thiết cho việc xử lý đồ thị.
* Tăng tốc độ xử lý: Khi số lượng đỉnh giảm, các phép toán trên đồ thị có thể được thực hiện nhanh hơn. Việc hợp nhất đỉnh có thể giảm độ phức tạp của các thuật toán phụ thuộc vào số lượng đỉnh.
* Giảm độ phức tạp: Sau khi áp dụng thuật toán, các thuật toán khác có thể được áp dụng trên đồ thị thu gọn để tìm kiếm, phân tích hoặc thực hiện các phân tích khác một cách hiệu quả hơn.

1. Nhược điểm của thuật toán thu gọn (hòa nhập) đỉnh là:

* Mất mát thông tin: Khi hợp nhất các đỉnh, một số thông tin có thể bị mất đi. Điều này có thể làm mất một phần kiến thức về cấu trúc và mối quan hệ trong đồ thị ban đầu.
* -Khó khăn trong việc phục hồi: Khi đồ thị đã được thu gọn, việc phục hồi đồ thị ban đầu từ đồ thị thu gọn có thể trở nên khó khăn và tốn thời gian.
* Ảnh hưởng đến tính chất của đồ thị: Thuật toán thu gọn đỉnh có thể thay đổi các tính chất của đồ thị, chẳng hạn như đường đi ngắn nhất, chu trình Euler, hay tính liên thông của đồ thị.

# **3.4. Thuật toán tô màu tuần tự**

# **3.4.1. Giới thiệu chung**

* Khái niệm:
* Một trong những phương pháp tô màu đơn giản nhất đó là phương pháp tô màu tuần tự dựa trên một tập đỉnh
* O=[ a0, a1, …,an-1] đã được sắp thứ tự:

o F(a0)=1.

o Nếu a1, …,ai-1 (i ≥ 1) đã được tô màu, đặt F(ai) là màu nhỏ nhất chưa được dùng bởi các đỉnh kề với ai.

* Ứng dụng: Thuật toán này được sử dụng để giải quyết bài toán tô màu đồ thị, trong đó mỗi đỉnh của đồ thị được tô màu sao cho hai đỉnh kề nhau không cùng màu.

# **3.4.2 Lưu đồ thuật toán**

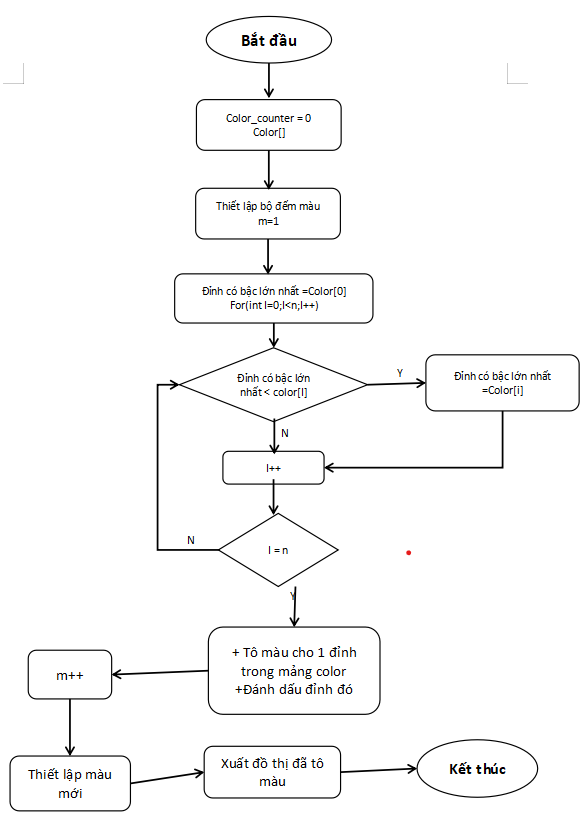
* Thuật toán DSATUR (Degree of Saturation) là thuật toán tô

màu tuần tự trong đó các đỉnh được sắp xếp động sau mỗi bước của thuật toán. Giả sử *F* là một phép tô màu bộ phận các đỉnh của *G*. Bậc bão hòa của đỉnh *x*, *degs*(*x*) là số lượng các màu khác nhau ở các đỉnh kề với *x*:

+ Bước 1: Gán màu 1 cho một đỉnh có bậc lớn nhất.

+ Bước 2: Đỉnh được tô màu tiếp theo là đỉnh *x* với bậc *degs(x)* lớn nhất.

+ Bước 3: Khi tất cả các đỉnh đều đã được tô màu thì kết thúc thuật toán.



hình 3.4.2.1: Lưu đồ thuật toán tô màu tuần tự

# **3.4.3. Cài đặt thuật toán**

# **3.4.4. Đánh giá thuật toán**

* Thuật toán tô màu tuần tự (sequential coloring) là một thuật toán đơn giản và hiệu quả để tô màu các đồ thị. Thuật toán này được sử dụng để giải quyết bài toán tô màu đồ thị, trong đó mỗi đỉnh của đồ thị được tô màu sao cho hai đỉnh kề nhau không cùng màu.
* Thuật toán tô màu tuần tự có thời gian chạy tuyến tính O(n+m), trong đó n là số đỉnh của đồ thị và m là số cạnh của đồ thị. Thuật toán này cũng có tính gọn nhẹ và dễ hiểu, nên được sử dụng rộng rãi trong thực tế.
* Tuy nhiên, thuật toán tô màu tuần tự có một số hạn chế. Một trong những hạn chế đó là nó không đảm bảo tìm được tô màu tối ưu cho một đồ thị. Ngoài ra, thuật toán này có thể tạo ra nhiều màu hơn cần thiết, do việc tô màu từng đỉnh một có thể dẫn đến sự lãng phí màu.
* Ưu điểm:

+) Thuật toán tô màu tuần tự đơn giản và dễ hiểu.

+) Thuật toán này cho kết quả tốt trong các trường hợp đồ thị có số đỉnh nhỏ và mật độ cạnh thấp.

+) Với đồ thị có số đỉnh ít, thuật toán tô màu tuần tự đem lại kết quả tốt về mặt thời gian và không tốn nhiều tài nguyên tính toán.

* Nhược điểm:

+) Thuật toán tô màu tuần tự không cho kết quả tối ưu trong trường hợp đồ thị có số đỉnh lớn và mật độ cạnh cao.

+) Tốc độ thực thi của thuật toán tô màu tuần tự giảm đáng kể khi đồ thị có số đỉnh lớn.

+) Trong một số trường hợp, thuật toán tô màu tuần tự không thể tìm ra phương án tô màu cho đồ thị.

# **THỰC NGHIỆM THUẬT TOÁN**

* **Bộ dữ liệu có 6 đỉnh, 8 cạnh:**

1 2

1 3

2 3

2 4

3 4

3 5

4 5

4 6

* **Bộ dữ liệu có 10 đỉnh, 15 cạnh:**

1 1

1 2

1 3

2 2

2 3

2 4

3 3

3 4

3 5

4 4

4 5

5 5

5 6

6 6

7 7

* **Bộ dữ liệu có 20 đỉnh, 30 cạnh:**

1 1

1 2

1 3

2 2

2 4

2 5

3 3

3 5

3 6

4 6

4 7

4 5

5 8

5 9

5 6

6 9

6 10

7 7

7 10

7 11

8 11

8 12

8 9

9 13

9 14

9 10

10 14

10 15

10 11

11 15

11 16

11 12

12 16

12 17

12 17

13 18

13 14

13 19

14 20

14 15

14 20

15 21

15 16

15 21

16 22

16 17

16 22

17 23

17 18

17 23

18 24

18 24

18 20

19 25

19 21

20 25

20 26

21 22

21 26

22 27

22 23

23 27

23 28

24 24

24 28

25 29

25 29

26 26

26 30

26 27

27 30

27 28

28 30

28 29

29 30

30 30

* **Bộ dữ liệu có 30 đỉnh, 45 cạnh:**

1 1

1 2

1 3

2 2

2 4

2 5

3 3

3 5

3 6

4 6

4 7

4 5

5 8

5 9

5 6

6 9

6 10

6 7

7 10

7 11

7 11

8 12

8 9

8 13

9 14

9 10

9 14

10 15

10 11

10 15

11 16

11 12

11 16

12 17

12 17

12 18

13 14

13 19

13 20

14 15

14 20

14 21

15 16

15 21

15 22

16 17

16 22

16 23

17 18

17 23

17 24

18 24

19 20

19 25

20 21

20 25

20 26

21 22

21 26

21 27

22 23

22 27

22 28

23 24

23 28

23 29

24 29

25 26

25 30

26 27

26 30

27 28

27 30

28 29

29 30

30 30

# 

# **KẾT LUẬN**

**Một số kết quả trong quá trình nghiên cứu đề tài:**

1. Hệ thống cơ bản được kiến thức lý thuyết đồ thị, tô màu đồ thị và

hiểu sâu hơn về các định nghĩa, định lí, thuật toán, các bài toán tô màu đồ thị.

1. Đưa ra được các thuật toán khác nhau và vận dụng thuật toán đó để giải quyết các bài toán tô màu đồ thị.

**Hướng phát triển :**

Các phương pháp sử dụng các thuật toán này có thể được mở rộng cho các bài toán tô màu đồ thị có nhiều ràng buộc hơn. Các thuật toán này nên được được cài đặt và phát triển thành một phần mềm để có thể sử dụng thuận tiện trong thực tế.

|  |  |
| --- | --- |
|  |  |

# 

# **TÀI LIỆU THAM KHẢO**

1. **Luanvan.co: (https://luanvan.co/luan-van/bai-toan-to-mau-do-thi-va-ung-dung-46084/)**

* **Nguyễn, H. H. (2017). Bài toán tô màu đồ thị và ứng dụng (Doctoral dissertation, Đại học Quốc gia Hà Nội).**
* **Behzad, M. (1960). A property of the perfect graph. Canadian Mathematical Bulletin, 3(2), 175-176.**
* **Chudnovsky, M., Robertson, N., Seymour, P., & Thomas, R. (2006). The strong perfect graph theorem. Annals of mathematics, 164(1), 51-229.**

1. **Xemtailieu.net: (https://xemtailieu.net/tai-lieu/thuat-toan-to-mau-do-thi-va-ung-dung-lv01834-1113146.html)**

* **Nguyễn, H. H. (2017). Bài toán tô màu đồ thị và ứng dụng (Doctoral dissertation, Đại học Quốc gia Hà Nội).**
* **Behzad, M. (1960). A property of the perfect graph. Canadian Mathematical Bulletin, 3(2), 175-176.**
* **Chudnovsky, M., Robertson, N., Seymour, P., & Thomas, R. (2006). The strong perfect graph theorem. Annals of mathematics, 164(1), 51-229.**

1. **Wikipedia: (https://vi.wikipedia.org)**

* **Korte, B., & Vygen, J. (2012). Combinatorial optimization: theory and algorithms. Springer Science & Business Media.**
* **Bondy, J. A., & Murty, U. S. R. (2008). Graph theory (Vol. 244). Springer Science & Business Media.**
* **Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to algorithms (3rd ed.). MIT Press.**