

**TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI
PHÂN HIỆU TẠI TP. HỒ CHÍ MINH
BỘ MÔN CÔNG NGHỆ THÔNG TIN**



**BÁO CÁO ĐỒ ÁN TỐT NGHIỆP
ĐỀ TÀI: TÌM HIỂU VÀ ỨNG DỤNG
THUẬT TOÁN XÁC ĐỊNH ĐƯỜNG ĐI HAMILTON**

Giảng viên hướng dẫn: NGUYỄN LÊ MINH

Sinh viên thực hiện: PHẠM VĂN TỊNH

Lớp : CÔNG NGHỆ THÔNG TIN

Khoá : 57

Tp. Hồ Chí Minh, năm 2021

**TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI
PHÂN HIỆU TẠI TP. HỒ CHÍ MINH
BỘ MÔN CÔNG NGHỆ THÔNG TIN**



**BÁO CÁO ĐỒ ÁN TỐT NGHIỆP
ĐỀ TÀI: TÌM HIỂU VÀ ỨNG DỤNG
THUẬT TOÁN XÁC ĐỊNH ĐƯỜNG ĐI HAMILTON**

Giảng viên hướng dẫn: NGUYỄN LÊ MINH

Sinh viên thực hiện: PHẠM VĂN TỊNH

Lớp : CÔNG NGHỆ THÔNG TIN

Khoá : 57

Tp. Hồ Chí Minh, năm 2021

TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI
PHÂN HIỆU TẠI TP. HỒ CHÍ MINH
BỘ MÔN CÔNG NGHỆ THÔNG TIN

CỘNG HOÀ XÃ HỘI CHỦ NGHĨA VIỆT NAM
Độc lập – Tự do - Hạnh phúc

NHIỆM VỤ THIẾT KẾ TỐT NGHIỆP
BỘ MÔN: CÔNG NGHỆ THÔNG TIN

-----***-----

Mã Sinh Viên: 5751071043

Khóa: 57

Họ tên SV: Phạm Văn Tịnh

Lớp: CQ.CNTT.57

1. Tên đề tài:

TÌM HIỂU VÀ ỨNG DỤNG THUẬT TOÁN XÁC ĐỊNH ĐƯỜNG ĐI HAMILTON.

2. Mục đích, yêu cầu:

a. Mục đích

- Tìm hiểu lý thuyết đồ thị và thuật toán xác định đồ thị Hamilton.
- Ứng dụng thuật toán xác định đồ thị Hamilton vào bài toán “Người du lịch”.

b. Yêu cầu

- Tìm hiểu thuật toán xác định đồ thị Hamilton.
- Ứng dụng thuật toán xác định đồ thị Hamilton vào bài toán “Người du lịch” và xây dựng trang web ứng dụng thuật toán.

3. Nội dung và phạm vi đề tài:

a. Nội dung đề tài:

- Tìm hiểu ngôn ngữ C# (C-Sharp) và Framework ASP.NET Core 5 và ứng dụng vào việc triển khai thuật toán xác định đồ thị Hamilton.
- Tìm hiểu Angular và ứng dụng vào việc xây dựng trang web áp dụng thuật toán xác định đồ thị Hamilton vào bài toán “Người du lịch”.
- Tìm hiểu Azure Devops, Google Compute Engine, Docker và ứng dụng vào việc triển khai trang web.

b. Phạm vi đề tài:

- Tìm hiểu và ứng dụng thuật toán xác định đồ thị Hamilton vào bài toán “Người du lịch”.

4. Công nghệ, công cụ và ngôn ngữ lập trình:

a. Công nghệ:

- ASP.NET Core
- Angular

b. Công cụ:

- IDE: Visual Studio Code
- Một số thư viện xử lý dữ liệu từ Google Maps APIs, Angular Maps Component v.v...

c. Ngôn ngữ lập trình: C#, Typescript

5. Các kết quả chính dự kiến sẽ đạt được và ứng dụng:

- Hoàn chỉnh cuốn báo cáo đề tài.
- Xây dựng hoàn chỉnh trang web ứng dụng thuật toán xác định đồ thị Hamilton vào bài toán “Người du lịch”.

6. Giảng viên và cán bộ hướng dẫn

Họ tên: NGUYỄN LÊ MINH

Đơn vị công tác: Bộ môn Công Nghệ Thông Tin – Trường Đại học Giao thông
Vận tải Phân hiệu tại TP HCM.

Điện thoại: 0931385579

Email: nlminh@utc2.edu.vn

Ngày ... tháng ... năm 2021

Trưởng BM Công nghệ Thông tin

Đã giao nhiệm vụ TKTN

Giảng viên hướng dẫn

ThS.Nguyễn Lê Minh

Đã nhận nhiệm vụ TKTN

Sinh viên: Phạm Văn Tịnh

Điện thoại: 0973957027

Ký Tên:

Email: 5751071043@st.utc2.edu.vn

LỜI CẢM ƠN

Ba tháng làm đồ án tốt nghiệp vừa qua là ba tháng khó quên nhất, trọn vẹn nhất trong quãng đời sinh viên của tôi, là quãng thời gian quý báu để tôi có thể vận dụng những kiến thức mà thầy cô đã truyền dạy trong gần suốt 4 năm tại trường. Trong suốt ba tháng, tôi đã được các thầy cô, bạn bè trong lớp đóng góp nhiều ý kiến, đưa ra nhiều nhận xét giúp tôi có thể hoàn thành đồ án của mình một cách tốt nhất có thể. Tôi muốn gửi lời cảm ơn chân thành nhất đến toàn thể quý thầy cô trong bộ môn Công nghệ thông tin Trường Đại học Giao thông Vận tải Phân hiệu tại thành phố Hồ Chí Minh, các thầy cô đã giảng dạy các môn học đến từ các trường lân cận, và đặc biệt là thầy Nguyễn Lê Minh đã quan tâm giúp đỡ, hướng dẫn đồ án tốt nghiệp cho tôi, để tôi có thể hoàn thành xuất sắc nhất đồ án tốt nghiệp.

Tôi xin cảm ơn ban lãnh đạo Trường Đại học Giao thông Vận tải Phân hiệu tại thành phố Hồ Chí Minh, các khoa, phòng ban chức năng đã tạo điều kiện tốt nhất có thể để tôi có thể hoàn thành đồ án tốt nghiệp, đưa đồ án vào thử nghiệm trong trường. Tôi mong sau khi hoàn thành đồ án tốt nghiệp tôi sẽ có thể bước ra ngoài xã hội với một công việc ổn định, đúng ngành nghề đã theo học và không ngừng phát triển hoàn thiện bản thân trên con đường sự nghiệp của mình.

Trong suốt quá trình làm đồ án, với điều kiện thời gian cũng như kinh nghiệm còn hạn chế, chắc chắn không thể tránh khỏi những thiếu sót, tôi mong thầy cô đóng góp ý kiến để tôi có thể bổ sung, hoàn thiện đồ án tốt nghiệp tốt hơn.

Tôi xin chân thành cảm ơn!

NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Tp. Hồ Chí Minh, ngày tháng năm
Giáo viên hướng dẫn

Nguyễn Lê Minh

PHẦN MỞ ĐẦU

Trên thực tế có nhiều bài toán liên quan tới một tập các đối tượng và những mối liên hệ giữa chúng, đòi hỏi toán học phải đặt ra một mô hình biểu diễn một cách chặt chẽ và tổng quát bằng ngôn ngữ ký hiệu, đó là đồ thị. Những ý tưởng cơ bản của nó được đưa ra từ thế kỷ thứ XVIII bởi nhà toán học Thụy Sĩ Leonhard Euler, ông đã dùng mô hình đồ thị để giải bài toán về những cây cầu Königsberg nổi tiếng.

Mặc dù lý thuyết đồ thị đã được khoa học phát triển từ rất lâu nhưng lại có rất nhiều ứng dụng hiện đại. Đặc biệt trong khoảng vài năm trở lại đây, cùng với sự ra đời của máy tính điện tử và sự phát triển nhanh chóng của Tin học, Lý thuyết đồ thị càng được quan tâm đến nhiều hơn. Đặc biệt là các thuật toán trên đồ thị đã có nhiều ứng dụng trong nhiều lĩnh vực khác nhau như: Mạng máy tính, Lý thuyết mã, Tối ưu hóa, Kinh tế học v.v... Chẳng hạn như trả lời câu hỏi: Hai máy tính trong mạng có thể liên hệ được với nhau hay không? Hay vấn đề phân biệt hai hợp chất hóa học có cùng công thức phân tử nhưng lại khác nhau về công thức cấu tạo cũng được giải quyết nhờ mô hình đồ thị. Hiện nay, môn học này là một trong những kiến thức cơ sở của bộ môn khoa học máy tính.

Trong thị trường thương mại hiện nay, ngành vận tải nói riêng và ngành du lịch nói riêng. Các doanh nghiệp này đã phát triển lộ trình vận chuyển hàng hóa, lộ trình kinh doanh tour du lịch v.v... và vẫn đang phát triển và tối ưu những lộ trình nhằm giảm thiểu chi phí. Vậy câu hỏi đặt ra rằng: “Chúng ta có thể tìm kiếm thêm những lộ trình mới tối ưu hơn.”.

Mặc dù đã cố gắng để hoàn thành công việc, nhưng do thiếu kinh nghiệm cũng như kỹ năng chưa cao, thời gian hạn chế nên việc phân tích, thiết kế và xây dựng chương trình còn mắc phải nhiều thiếu sót. Kính mong thầy cô góp ý, bổ sung để tôi hoàn thiện tốt hơn.

Tôi xin chân thành cảm ơn!

Tp. Hồ Chí Minh, 10/06/2021

MỤC LỤC

NHIỆM VỤ THIẾT KẾ TỐT NGHIỆP	
LỜI CẢM ƠN	
NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN	
PHẦN MỞ ĐẦU	
DANH MỤC BẢNG BIỂU	
DANH MỤC HÌNH VẼ	
CHƯƠNG 1. TỔNG QUAN ĐỀ TÀI.....	1
1.1. Lý do chọn đề tài.....	1
1.2. Mục đích nghiên cứu.....	2
1.3. Đối tượng và phạm vi nghiên cứu.....	2
1.4. Phương pháp nghiên cứu.....	2
1.5. Cấu trúc báo cáo đồ án tốt nghiệp.....	2
1.6. Kết luận	3
CHƯƠNG 2. CƠ SỞ LÝ THUYẾT.....	4
2.1. Tổng quan về .NET Core	4
2.1.1. Giới thiệu	4
2.1.2. Tại sao chọn .NET Core?	4
2.1.3. Đặc điểm cốt lõi của .NET Core	5
2.1.4. Lịch sử phiên bản .NET Core	6
2.1.5. Thành phần cốt lõi của .NET Core	6
2.2. Ngôn ngữ lập trình C#.....	7
2.2.1. Giới thiệu ngôn ngữ C#.....	7
2.2.2. Đặc trưng của ngôn ngữ C#	8
2.3. Tổng quan về Angular Framework	9
2.3.1. Giới thiệu Angular Framework.....	9

2.3.3.	Những tính năng nổi bật của Angular Framework	10
2.4.	Tổng quan về Google Maps API.....	11
2.4.1.	Giới thiệu	11
2.4.2.	Các dịch vụ đang được Google Maps API cung cấp	12
2.5.	Tổng quan về lý thuyết đồ thị	13
2.5.1.	Tổng quan	13
2.5.2.	Định nghĩa đồ thị.....	13
2.5.6.	Biểu diễn đồ thị trên máy vi tính	25
2.5.7.	Các thuật toán tìm kiếm trên đồ thị và ứng dụng.....	33
CHƯƠNG 3. PHÂN TÍCH THUẬT TOÁN.....		40
3.1.	Tổng quan thuật toán Hamilton.....	40
3.1.1.	Giới thiệu thuật toán.....	40
3.1.2.	Mô tả thuật toán	40
3.1.3.	Qui tắc tìm chu trình Hamilton	46
3.1.4.	Ví dụ về thuật toán	46
3.2.	Kết luận	47
CHƯƠNG 4. THỬ NGHIỆM VÀ ĐÁNH GIÁ.....		48
4.1.	Giới thiệu bài toán.....	48
4.2.	Môi trường thử nghiệm	48
4.3.	Giao diện trang web thử nghiệm.....	49
4.4.	Kết quả và đánh giá.....	52
CHƯƠNG 5. KẾT LUẬN VÀ KIẾN NGHỊ		54
5.1.	Kết quả đạt được	54
5.2.	Hướng phát triển	55
TÀI LIỆU THAM KHẢO		56

DANH MỤC CHỮ VIẾT TẮT

Từ viết tắt	Mô tả	Ý nghĩa	Ghi chú
API	Application Programming Interface	Giao diện lập trình ứng dụng.	
IoT	Internet of Thing	Internet vạn vật.	
MIT	Massachusetts Institute of Technology	Giấy phép phần mềm.	
MVC	Model-View-Controller	Kiến trúc phần mềm.	
DOM	Document object Model	Giao diện lập trình ứng dụng.	
SPA	Single Page Application	Ứng dụng web.	
SDK	Software Development Kit	Bộ công cụ phát triển phần mềm.	
HTTP	Hyper Text Transfer Protocol	Giao thức truyền tải siêu văn bản.	

DANH MỤC BẢNG BIỂU

Bảng 1.1: Bảng theo dõi ngày phát hành các phiên bản .NET Core và .NET 5 được hỗ trợ (nguồn: Microsoft).

Bảng 1.2: Bảng theo dõi các phiên bản .NET Core kết thúc hỗ trợ (nguồn Microsoft).

DANH MỤC HÌNH VẼ

Hình 1.1: Các thành phần cơ bản của .NET Core.

Hình 1.2: Sơ đồ mạng máy tính.

Hình 1.3: Sơ đồ mạng máy tính với đa kênh thoại.

Hình 1.4: Sơ đồ mạng máy tính với kênh thoại thông báo.

Hình 1.5: Mạng máy tính với kênh thoại một chiều.

Hình 1.6: Đồ thị vô hướng.

Hình 1.7: Đồ thị có hướng.

Hình 1.8: Đường đi trên đồ thị.

Hình 1.9: Đồ thị G và H .

Hình 1.10: Đồ thị liên thông mạnh G và đồ thị liên thông yếu H .

Hình 1.11: Đồ thị đầy đủ.

Hình 1.12: Đồ thị lập phương Q_1, Q_2, Q_3 .

Hình 1.13: Đồ thị hai phía.

Hình 1.14: Đồ thị K_4 là đồ thị phẳng.

Hình 1.15: Các miền tương ứng với biểu diễn phẳng của đồ thị.

Hình 1.16: Đồ thị vô hướng G và Đồ thị có hướng G_1

Hình 1.17: Ma trận kề cho đồ thị có hướng G_1 cho trong hình 1.

Hình 1.18: Danh sách cạnh của G .

Hình 1.19: Danh sách cung của G_1 .

Hình 1.20: Danh sách kề của đồ thị vô hướng G cho trong hình 1.

Hình 1.21: Danh sách kề của đồ thị có hướng G_1 cho trong hình 1.

Hình 1.22: Đồ thị H .

Hình 1.23: Chỉ số mới (trong dấu ngoặc) của các đỉnh được đánh giá lại theo thứ tự chúng được thăm trong thuật toán tìm kiếm theo chiều sâu.

Hình 1.24: Chỉ số mới (trong dấu ngoặc) của các đỉnh được đánh giá lại theo thứ tự chúng được thăm trong thuật toán tìm kiếm theo chiều sâu.

Hình 2.1: Đường đi Hamilton.

Hình 2.2: Hành trình của con mã trên bàn cờ 8×8 .

Hình 2.3: Đồ thị vô hướng $ayba'b'$.

Hình 2.4: Đồ thị G .

Hình 2.5: Đồ thị G' .

Hình 2.6: Đồ thị vô hướng G' .

Hình 2.7: Đồ thị đầy đủ K_n .

Hình 2.8: Đồ thị H .

Hình 2.9: Chu trình Hamilton H tìm được.

Hình 3.1: Giao diện trang web thử nghiệm.

Hình 3.2: Hệ thống tự động đánh dấu các địa điểm mà người dùng nhập vào lên Google Maps.

Hình 3.3: Các kết quả sau khi hệ thống xử lý xong được thể hiện.

Hình 3.4: Chi tiết lộ trình đường đi 1.

Hình 3.5: Chi tiết được đi từ điểm ‘Thành phố Hồ Chí Minh, Việt Nam’ đến ‘Thành phố Vũng Tàu, Bà Rịa – Vũng Tàu, Việt Nam’

CHƯƠNG 1. TỔNG QUAN ĐỀ TÀI

1.1. Lý do chọn đề tài

Trên thực tế có nhiều bài toán liên quan tới một tập các đối tượng và những mối liên hệ giữa chúng, đòi hỏi toán học phải đặt ra một mô hình biểu diễn một cách chặt chẽ và tổng quát bằng ngôn ngữ ký hiệu, đó là đồ thị. Những ý tưởng cơ bản của nó được đưa ra từ thế kỷ thứ XVIII bởi nhà toán học Thụy Sĩ Leonhard Euler, ông đã dùng mô hình đồ thị để giải bài toán về những cây cầu Königsberg nổi tiếng.

Mặc dù lý thuyết đồ thị đã được khoa học phát triển từ rất lâu nhưng lại có rất nhiều ứng dụng hiện đại. Đặc biệt trong khoảng vài năm trở lại đây, cùng với sự ra đời của máy tính điện tử và sự phát triển nhanh chóng của Tin học, Lý thuyết đồ thị càng được quan tâm đến nhiều hơn. Đặc biệt là các thuật toán trên đồ thị đã có nhiều ứng dụng trong nhiều lĩnh vực khác nhau như: Mạng máy tính, Lý thuyết mã, Tối ưu hóa, Kinh tế học v.v... Chẳng hạn như trả lời câu hỏi: Hai máy tính trong mạng có thể liên hệ được với nhau hay không? Hay vấn đề phân biệt hai hợp chất hóa học có cùng công thức phân tử nhưng lại khác nhau về công thức cấu tạo cũng được giải quyết nhờ mô hình đồ thị. Hiện nay, môn học này là một trong những kiến thức cơ sở của bộ môn khoa học máy tính.

Trong thị trường thương mại hiện nay, ngành vận tải nói riêng và ngành du lịch nói riêng. Các doanh nghiệp này đã phát triển lộ trình vận chuyển hàng hóa, lộ trình kinh doanh tour du lịch v.v... và vẫn đang phát triển và tối ưu những lộ trình nhằm giảm thiểu chi phí. Vậy câu hỏi đặt ra rằng: “Chúng ta có thể tìm kiếm thêm những lộ trình mới tối ưu hơn.”.

Từ bài toán thực tế trên, em quyết định sử dụng thuật toán xác định đường đi Hamilton kết hợp với dữ liệu từ Google Maps API, phân tích xem những lộ trình nào sẽ tối ưu nhất. Thông qua kết quả đó các doanh nghiệp có thể điều chỉnh lại lộ trình kinh doanh cho phù hợp.

Với mong muốn đóng góp thêm một giải pháp về ứng dụng công nghệ thông tin vào nghiên cứu và ứng dụng thuật toán xác định đường đi Hamilton để phân tích và tìm kiếm các lộ trình đường đi tối ưu, em đã chọn đề tài “TÌM HIỂU VÀ ỨNG DỤNG THUẬT TOÁN XÁC ĐỊNH ĐƯỜNG ĐI HAMILTON”.

1.2. Mục đích nghiên cứu

- Nghiên cứu cơ sở lý thuyết đồ thị và ứng dụng thuật toán xác định đường đi Hamilton để hỗ trợ việc phân tích và tìm ra lộ trình đường đi tối ưu.
- Nghiên cứu thuật toán xác định đường đi Hamilton và tìm hiểu cách cải tiến thuật toán, từ đó triển khai ứng dụng đơn giản để kết hợp với dữ liệu thực tế từ Google Maps API.

1.3. Đối tượng và phạm vi nghiên cứu

- Nghiên cứu các vấn đề về kết hợp dữ liệu từ Google Maps API với thuật toán xác định đường đi Hamilton trong quá trình phân tích dữ liệu.
- Đồ án tập trung nghiên cứu thuật toán xác định đường đi Hamilton và khả năng ứng dụng để phát triển và ứng dụng thực tế.

1.4. Phương pháp nghiên cứu

- Phương pháp nghiên cứu tài liệu: chọn lọc, phân tích và tổng hợp các tài liệu về lý thuyết đồ thị nói chung và thuật toán xác định đường đi Hamilton nói riêng, từ kho thông tin khổng lồ của các tác giả trong nước hay ngoài nước đã và đang nghiên cứu. Từ đó rút ra các kiến thức cần thiết để hoàn thành nhiệm vụ.
- Phương pháp thực nghiệm: tổng hợp là làm sạch dữ liệu. Áp dụng vào xây dựng ứng dụng mô phỏng lại thuật toán xác định đường đi Hamilton từ dữ liệu thật ở Google Maps API để tìm ra các lộ trình phù hợp.

1.5. Cấu trúc báo cáo đồ án tốt nghiệp

Cấu trúc đồ án được chia thành các chương như sau:

Chương 1: Tổng quan đề tài.

Chương 2: Cơ sở lý thuyết

Chương 3: Phân tích thuật toán

Chương 4: Thử nghiệm và đánh giá

Chương 5: Kết luận và kiến nghị

1.6. Kết luận

Phần này em đã trình bày tổng quan về đề tài đồ án tốt nghiệp. Nói lên lý do chọn đề tài, mục đích nghiên cứu và phương pháp nghiên cứu thuật toán. Qua chương 1, em sẽ trình bày cơ sở lý thuyết phục vụ cho thuật toán cũng như ứng dụng.

CHƯƠNG 2. CƠ SỞ LÝ THUYẾT

2.1. Tổng quan về .NET Core

2.1.1. Giới thiệu

.NET Core là phiên bản mới nhất của .NET Framework, là một nền tảng phát triển miễn phí, mã nguồn mở được xây dựng và phát triển bởi Microsoft và cộng đồng .NET trên Github. .NET Core là một cross-platform chạy trên các hệ điều hành Windows, macOS và Linux.

.NET Core có thể được sử dụng để xây dựng các ứng dụng khác nhau như ứng dụng mobile, web, cloud, IoT, machine learning, microservice, game v.v...

.NET Core được xây dựng lại từ đầu để làm cho nó trở thành một Framework mô-đun, nhẹ, nhanh và đa nền tảng. Nó bao gồm các tính năng cốt lõi được yêu cầu để chạy một ứng dụng .NET Core cơ bản. Các tính năng khác được cung cấp dưới dạng gói NuGet, nhà phát triển có thể thêm nó vào ứng dụng nếu cần. Bằng cách này, các ứng dụng .NET Core có thể tăng hiệu suất, giảm dung lượng bộ nhớ và trở nên dễ bảo trì.

2.1.2. Tại sao chọn .NET Core?

Có một số hạn chế với .NET Framework. Ví dụ, nó chỉ chạy trên nền tảng Windows. Ngoài ra, nhà phát triển cần sử dụng các API .NET khác nhau cho các thiết bị Windows khác nhau như Windows Desktop, Windows Store, Windows Phone và các ứng dụng Web. Ngoài ra, bất kỳ thay đổi được thực hiện trên .NET Framework đều ảnh hưởng đến tất cả các ứng dụng phụ thuộc vào nó.

Ngày nay, việc có một ứng dụng chạy trên các thiết bị là điều phổ biến; chương trình phụ trợ trên máy chủ web, giao diện người dùng quản trị trên máy tính để bàn windows, web và ứng dụng di động cho người tiêu dùng. Vì vậy, cần có một khuôn khổ duy nhất hoạt động ở mọi nơi. Vì vậy, xem xét điều này, Microsoft đã tạo ra .NET Core. Mục tiêu chính của .NET Core là làm cho .NET Framework mã nguồn mở, tương thích đa nền tảng, có thể được sử dụng trong nhiều lĩnh vực.

2.1.3. Đặc điểm cốt lõi của .NET Core

Khuôn khổ mã nguồn mở:

.NET Core là một khuôn khổ mã nguồn mở được duy trì bởi Microsoft và có sẵn trên GitHub theo giấy phép MIT và Apache 2. Nó là một dự án .NET Foundation.

Đa nền tảng:

.NET Core chạy trên hệ điều hành Windows, macOS và Linux.

Nhất quán giữa các kiến trúc:

Thực thi mã với cùng một hành vi trong các kiến trúc tập lệnh khác nhau, bao gồm x64, x86 và ARM.

Nhiều loại ứng dụng:

Nhiều loại ứng dụng khác nhau có thể được phát triển và chạy trên nền tảng .NET Core như mobile, desktop, cloud, IoT, machine-learning, microservices, game v.v...

Hỗ trợ nhiều ngôn ngữ:

Nhà phát triển có thể sử dụng ngôn ngữ lập trình C#, F# và Visual Basic để phát triển các ứng dụng .NET Core. Có thể sử dụng IDE yêu thích bao gồm Visual Studio 2017/2019, Visual Studio Code, Sublime Text, Vim v.v...

Kiến trúc mô-đun:

.NET Core hỗ trợ cách tiếp cận kiến trúc mô-đun bằng cách sử dụng các gói NuGet. Các gói NuGet khác nhau cho các tính năng khác nhau có thể được thêm vào dự án .NET Core nếu cần. Ngay cả thư viện .NET Core cũng được cung cấp dưới dạng gói NuGet. Bằng cách này, nó giảm dung lượng bộ nhớ, tăng tốc hiệu suất và dễ bảo trì.

Công cụ CLI:

.NET Core bao gồm các công cụ CLI (Command Line Interface) để phát triển và tích hợp liên tục.

Triển khai linh hoạt:

Ứng dụng .NET Core có thể được triển khai trên toàn người dung hoặc toàn hệ thống với Docker Containers.

Khả năng tương thích:

Tương thích với .NET Framework và Mono API.

2.1.4. Lịch sử phiên bản .NET Core

Các phiên bản được hỗ trợ:

Phiên bản	Ngày phát hành	Phiên bản mới nhất	Kết thúc hỗ trợ
.NET 5	10/11/2020	5.0.7	Khoảng tháng 2/2022
.NET Core 3.1	3/12/2019	3.1.16	3/12/2022
.NET Core 2.1	20/5/2018	2.1.18	21/8/2021

Bảng 1.1: Bảng theo dõi ngày phát hành các phiên bản .NET Core và .NET 5 được hỗ trợ (nguồn: Microsoft)

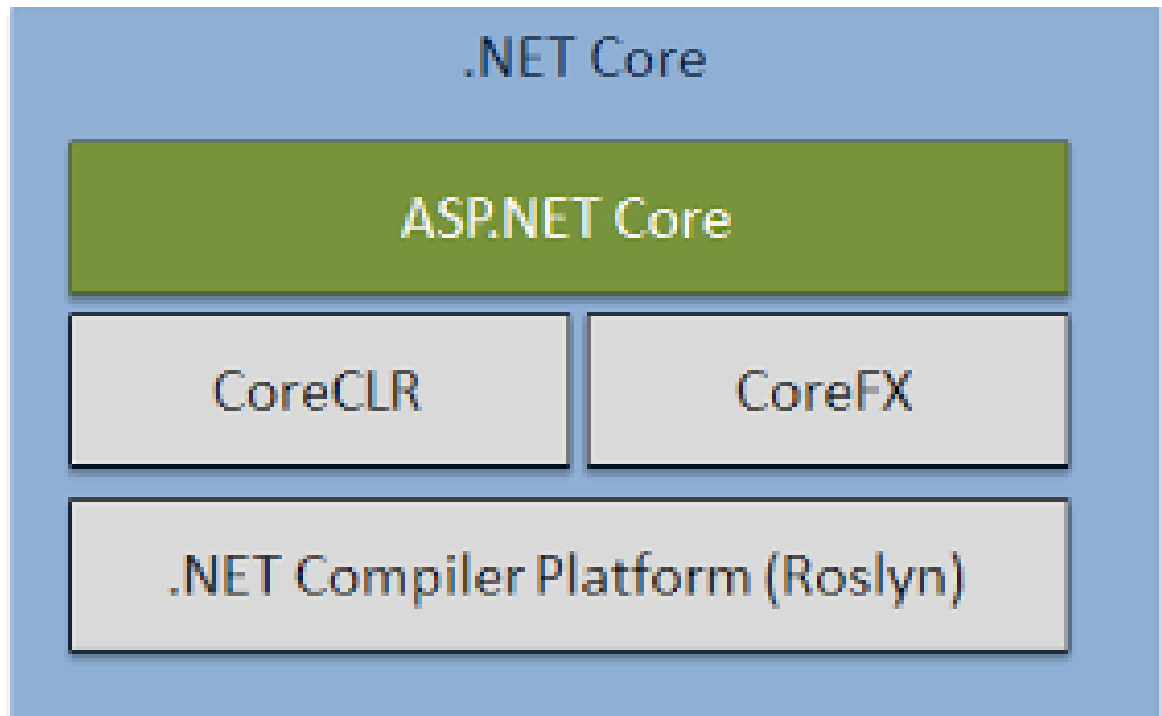
Các phiên bản không hỗ trợ:

Phiên bản	Ngày phát hành	Phiên bản mới nhất	Kết thúc hỗ trợ
.NET Core 3.0	23/9/2019	3.0.3	3/3/2020
.NET Core 2.2	4/12/2018	2.2.8	23/12/2019
.NET Core 2.0	14/8/2017	2.0.9	1/10/2018
.NET Core 1.1	16/11/2016	1.1.13	27/6/2019
.NET Core 1.0	27/6/2016	1.0.16	27/6/2019

Bảng 1.2: Bảng theo dõi các phiên bản .NET Core kết thúc hỗ trợ (nguồn Microsoft)

2.1.5. Thành phần cốt lõi của .NET Core

.NET Core Framework bao gồm các thành phần sau:



Hình 1.1: Các thành phần cơ bản của .NET Core

Công cụ CLI: Một bộ công cụ để phát triển và triển khai.

Roslyn: Trình biên dịch ngôn ngữ cho C# và Visual Basic.

CoreFx: Tập hợp các thư viện.

CoreCLR: Một trình biên dịch dựa trên JIT

2.2. Ngôn ngữ lập trình C#

2.2.1. Giới thiệu ngôn ngữ C#

Ngôn ngữ C# là một ngôn ngữ lập trình hiện đại được phát triển bởi Microsoft và được phê duyệt bởi European Computer Manufacturers Association (ECMA) và International Standards Organization (ISO). C# là ngôn ngữ lập trình hướng đối tượng (Object-oriented language) bao gồm: tính đóng gói (encapsulation), tính kế thừa (inheritance), tính đa hình (Polymorphism), tính trừu tượng (Abstraction). Nó được xây dựng trên nền tảng của hai ngôn ngữ C++ và Java.

C# được thiết kế cho các ngôn ngữ chung cơ sở hạ tầng (Common language infrastructure), trong đó bao gồm các mã (Executable code) và môi trường thực thi (Runtime environment) cho phép sử dụng các ngôn ngữ cấp cao khác nhau trên các nền tảng và kiến trúc máy tính khác nhau.

C# là một trong nhiều ngôn ngữ lập trình được hỗ trợ bởi .NET Core.

2.2.2. Đặc trưng của ngôn ngữ C#

C# là một ngôn ngữ đơn giản:

Ngôn ngữ C# đơn giản vì nó dựa trên nền tảng C và C++. Nếu nhà phát triển đã làm quen với C, C# hoặc thậm chí là Java, nhà phát triển sẽ thấy C# khá giống về diện mạo, cú pháp, biểu thức, toán tử và các chức năng khác được lấy trực tiếp từ ngôn ngữ C và C++, nhưng nó đã được cải tiến để đơn giản hơn.

C# là ngôn ngữ hiện đại:

Ngôn ngữ C# chứa những đặc tính như là xử lý ngoại lệ, thu gom bộ nhớ tự động, kiểu dữ liệu mở rộng, bảo mật mã nguồn đều là những đặc tính của một ngôn ngữ hiện đại.

C# là ngôn ngữ hướng đối tượng:

Những đặc điểm chính của ngôn ngữ hướng đối tượng là tính đóng gói, tính kế thừa, tính đa hình và tính trừu tượng. C# hỗ trợ tất cả những đặc tính trên, giúp cho các lập trình viên có thể tối ưu và tái sử dụng các mã nguồn một cách dễ dàng.

C# là ngôn ngữ mạnh mẽ và mềm dẻo:

Ngôn ngữ C# chỉ bị giới hạn ở chính bởi bản thân hay là trí tưởng tượng của chúng ta. Ngôn ngữ này không đặt những ràng buộc lên những việc có thể làm. C# được sử dụng cho nhiều dự án khác nhau như tạo ra ứng dụng xử lý văn bản, ứng dụng đồ họa, bản tính, hay thậm chí là trình biên dịch cho các ngôn ngữ khác.

C# là ngôn ngữ có ít từ khóa:

C# là ngôn ngữ sử dụng giới hạn những từ khóa. Phần lớn các từ khóa được sử dụng để mô tả thông tin. Chúng ta có thể nghĩ rằng một ngôn ngữ có nhiều từ khóa thì mạnh hơn. Điều này không phải sự thật, ít nhất trong trường hợp ngôn ngữ C#, chúng ta có thể tìm thấy ngôn ngữ này có thể sử dụng để làm bất cứ nhiệm vụ nào.

C# là ngôn ngữ hướng module:

Mã nguồn C# có thể được viết trong những phần được gọi là những lớp, những lớp này chứa các phương thức thành viên của nó. Những lớp và những phương thức có thể được sử dụng lại trong ứng dụng hay các chương trình khác. Bằng cách truyền các mẫu thông tin đến những lớp hay phương thức có thể tạo ra nhưng mã nguồn dùng lại một cách hiệu quả [5].

2.3. Tổng quan về Angular Framework

2.3.1. Giới thiệu Angular Framework

Angular là một JavaScript Framework dùng để tạo giao diện web (Front-end). Đây là một sản phẩm được viết bởi Misko Hevery và Adam Abrons. Chính thức ra mắt vào ngày 20/10/2010. Hiện tại Angular đang được Google duy trì.

Hiểu đơn giản, Angular là một khung làm việc của JavaScript MVC phía máy khách (client) nhằm phát triển web động.

AngularJS là từ dùng để nói về Angular 1 (ra đời năm 2009), được viết bằng JavaScript. Angular là từ gọi chung cho Angular 2 trở lên (ra đời năm 2016), được viết bằng TypeScript – phiên bản nâng cao của JavaScript.

Angular được thay đổi rất nhiều từ AngularJS. Angular được thiết kế lại từ đầu nên có nhiều khái niệm đã thay đổi từ AngularJS. Kiến trúc của Angular và AngularJS là hoàn toàn khác nhau.

Hiện tại AngularJS không còn được Google hỗ trợ nâng cấp.

2.3.2. Các phiên bản của Angular Framework

- **AngularJS:** Đây là phiên bản đầu tiên của Angular được cho ra mắt vào 20/10/2010, hoạt động theo kiểu MVC (Model View Controller), do Misko Hevery làm việc tại Google sáng tạo ra.
- **Angular 2:** Ra mắt vào 14/09/2016, là phiên bản thay thế cho AngularJS, sử dụng các khái niệm mới để tối đa quá trình phát triển trong Framework này. Phiên bản này được viết bằng TypeScript, có tốc độ làm việc nhanh và hỗ trợ đa nền tảng, cộng thêm cấu trúc code đơn giản, dễ sử dụng.
- **Angular 4:** Với phiên bản này, số lượng code đã được lược bớt, làm cho kích thước tệp đóng gói giảm xuống 60%, giúp đẩy nhanh quá trình phát triển ứng dụng. Được trình làng vào 23/03/2017.
- **Angular 5:** Ra mắt vào ngày 11/01/2017, ứng dụng HTTPClient thay cho HTTP, làm tăng tốc độ và khả năng bảo mật cao cho chương trình. Ngoài ra, sử dụng công cụ Build Optimizer được thiết lập sẵn trong CLI, hỗ trợ việc tối ưu Tree Shark và bỏ bớt những dòng code không cần thiết.

- **Angular 6:** Version 6 hỗ trợ ra mắt vào ngày 03/05/2018. Điều đặc biệt ở phiên bản này là được cập nhật thêm CLI (Command Line Interface), cùng một số lệnh mới như ng-update để thuận tiện cho việc chuyển đổi giữa các phiên bản và ng-add để dễ dàng thêm những tính năng quan trọng trong ứng dụng. Từ đó, khiến nó trở thành ứng dụng web tiên bộ hơn.
- **Angular 7:** Phát hành 18/10/2018, được ứng dụng các công nghệ mới. Version này đã được cập nhật trên RxJS 6.3.
- **Angular 8:** Ra mắt vào ngày 25/8/2019 với CLI workflow improvements và Dynamic imports for lazy routers, cùng với nhiều công cụ khác.
- **Angular 9:** Phiên bản này mới được ra mắt vào 6/2/2020, hỗ trợ quá trình di chuyển các ứng dụng để có thể sử dụng trình biên dịch Ivy theo thời gian mặc định. Việc cập nhật này nhằm mục đích giúp nó có thể hoạt động được trên TypeScript 3.6 và 3.7.
- **Angular 9.1:** Được ra mắt sau ngày 25/3/2020.
- **Angular 10:** Được ra mắt sau version 9.1 khoảng 1 tháng (chính xác là vào ngày 8/4/2020).
- **Angular 11:** Được phát hành vào tháng 11/2020. Bản phát hành chính Angular 11 cung cấp các bản cập nhật trên toàn bộ nền tảng, bao gồm CLI và các components.

2.3.3. Những tính năng nổi bật của Angular Framework

Cơ chế Two-Way Data Binding:

Đây là tính năng được nhiều lập trình viên đánh giá là ấn tượng nhất trong Angular. Với tính năng này, mọi thay đổi trên view, dù là nhỏ cũng đều được cập nhật tự động trên model vào Component Class và ngược lại

Ngoài ra, nó còn hỗ trợ Property Binding. Nhờ có tính năng này, các lập trình viên có thể tạo ra mối liên quan giữa các thuộc tính HTML và Component Class, lúc này mọi dữ liệu sẽ được xuất hiện tự động trong view thông qua việc điều khiển DOM.

Hỗ trợ cơ chế Routing mạnh mẽ:

Angular cung cấp cơ chế Routing đồng bộ trên các trang và cho phép chúng ta có thể tạo ra SPA. Đồng thời, hỗ trợ và hiển thị đúng view nào đúng thời điểm và mục đích điều hướng.

Đặc biệt, Angular còn giúp định nghĩa các Route cho mỗi Page View trên từng ứng dụng. Các lập trình viên sẽ kích hoạt Route dựa vào sự tương tác của các người dùng.

Mở rộng HTML:

Thông qua Angular, các lập trình viên dễ dàng sử dụng được cấu trúc tương tự như điều kiện IF, vòng lặp FOR, cùng những biến địa phương “local variables”.

Thiết kế Module hóa:

Người dùng chỉ có thể tổ chức và quản lý tốt các source code thông qua quản trình tạo Angular Module. Do Angular hoạt động dựa trên việc tiếp cận các thiết kế Module hóa.

Hỗ trợ quá trình làm việc với Back End:

Nhờ có Angular mà việc kết nối với Back End Server trở nên dễ dàng, thực thi việc nhận dữ liệu một cách logic.

Cộng đồng hỗ trợ lớn:

Cung cấp nhiều người dữ liệu đa dạng, từ cơ bản đến nâng cao qua các API, có hẳn một khóa học căn bản được tạo nên bởi cộng đồng những người sử dụng Angular.

Sử dụng mã nguồn mở.

Được phát triển bởi Google và thường xuyên cập nhật phiên bản mới.

2.4. Tổng quan về Google Maps API

2.4.1. Giới thiệu

Google Maps là một dịch vụ ứng dụng vào công nghệ bản đồ trực tuyến trên web miễn phí được cung cấp bởi Google, hỗ trợ nhiều dịch vụ khác của Google đặc biệt là dò đường và chỉ đường; hiển thị bản đồ đường sá, các tuyến đường tối ưu cho từng loại phương tiện, cách bắt xe và chuyển tuyến cho các loại phương tiện công cộng (xe bus, xe khách v.v ...), và những địa điểm (kinh doanh, trường học, bệnh viện, cây ATM v.v...) trong khu vực cũng như khắp nơi trên thế giới.

Maps API là phương thức cho phép 1 website B sử dụng dịch vụ bản đồ của website A (gọi là Maps API) và nhúng vào website của mình (site B). Sute ở đây là Google Maps, site B là các website cá nhân hoặc tổ chức muốn sử dụng dịch vụ Google (di chuột, zoom, đánh dấu trên bản đồ v.v...).

Các ứng dụng xây dựng trên Google Maps được nhúng vào trang web cá nhân thông qua các thẻ JavaScript do vậy việc sử dụng Google Maps API rất dễ dàng.

Google Maps API đã được nâng cấp lên phiên bản v3 không chỉ hỗ trợ cho các máy để bàn truyền thống mà cho cả các thiết bị di động; các ứng dụng nhanh hơn và nhiều hơn.

Các dịch vụ hoàn toàn miễn phí với việc xây dựng một ứng dụng nhỏ. Trả phí nếu đó là việc sử dụng cho mục đích kinh doanh, doanh nghiệp.

2.4.2. Các dịch vụ đang được Google Maps API cung cấp

Maps:

Maps SDKs: Mang thế giới thực đến với người dùng của với bản đồ tĩnh và động cho trang web và thiết bị di động.

Embeddeb maps: Thêm một bản đồ tương tác vào web với một yêu cầu HTTP đơn giản.

Street view imagery: Thêm chế độ xem phố 360 ° vào ứng dụng.

Elevation: Nhận độ cao của một hoặc một loạt vị trí.

Routes:

Directions: Cung cấp chỉ đường cho nhiều phương thức giao thông, có thông tin giao thông thời gian thực.

Distance Matrix: Tính toán thời gian và khoảng cách di chuyển cho nhiều điểm xuất phát và điểm đến.

Roads: Xác định các con đường lân cận bằng cách xử dụng tọa độ vị trí.

Places:

Places API & SDKs: Tích hợp chi tiết Địa điểm, tìm kiếm và tự động hoàn thành tìm kiếm của Google vào ứng dụng.

Geocoding: Chuyển đổi tọa độ thành địa chỉ và ngược lại.

Geolocation: Nhận vị trí thiết bị gần đúng bằng cách sử dụng các tháp di động và WiFi gần đó.

Time zones: Xác định múi giờ cho một hoặc một loạt tọa độ.

2.5. Tổng quan về lý thuyết đồ thị

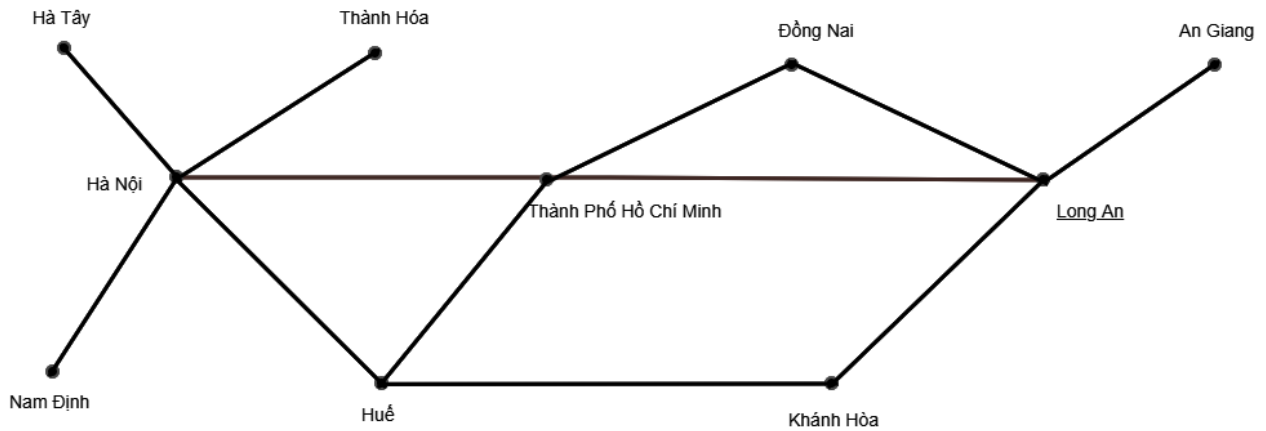
2.5.1. Tổng quan

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

Đồ thị được sử dụng để giải các bài toán trong nhiều lĩnh vực khác nhau. Chẳng hạn, đồ thị có thể sử dụng để xác định các mạch vòng trong vấn đề giải tích mạch điện. Chúng ta có thể phân biệt các hợp chất hóa học hữu cơ khác nhau với cùng công thức phân tử nhưng khác nhau về cấu trúc phân tử nhờ đồ thị. Chúng ta có thể xác định hai máy tính trong mạng có thể trao đổi thông tin được với nhau hay không nhờ mô hình đồ thị của mạng máy tính. Đồ thị có trọng số trên các cạnh có thể sử dụng để giải các bài toán như: Tìm đường đi ngắn nhất giữa hai thành phố trong mạng giao thông. Chúng ta còn có thể sử dụng đồ thị để giải các bài toán về lập lịch, thời khóa biểu, phân bố tần số cho các trạm phát thanh và truyền hình v.v...

2.5.2. Định nghĩa đồ thị

Đồ thị là một cấu trúc rời rạc bao gồm các đỉnh và các cạnh nối các đỉnh này. Chúng ta phân biệt các loại đồ thị khác nhau bởi kiểu và số lượng cạnh nối hai đỉnh nào đó của đồ thị. Để có thể hình dung được tại sao lại cần đến các loại đồ thị khác nhau, chúng ta sẽ nêu ví dụ sử dụng chúng để mô tả một mạng máy tính. Giả sử ta có một mạng gồm các máy tính và các kênh điện thoại (gọi tắt là kênh thoại) nối các máy tính này. Chúng ta có thể biểu diễn các vị trí đặt máy tính bởi các điểm và các kênh thoại nối chúng bởi các đoạn nối.



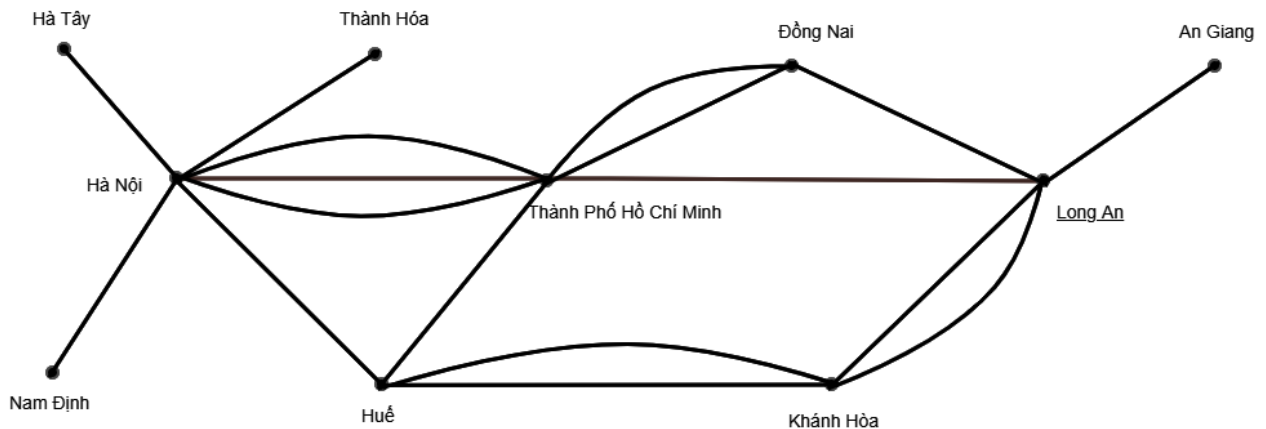
Hình 1.2: Sơ đồ mạng máy tính.

Nhận thấy rằng trong mạng ở hình 1.2, giữa hai máy bất kỳ chỉ có nhiều nhất là một kênh thoại nối giữa chúng, kênh thoại này cho phép liên lạc cả hai chiều và không có máy tính nào lại được nối với chính nó. Sơ đồ mạng trong hình 1.2 được gọi là đơn đồ thị vô hướng. Ta có định nghĩa sau:

Định nghĩa 1:

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.

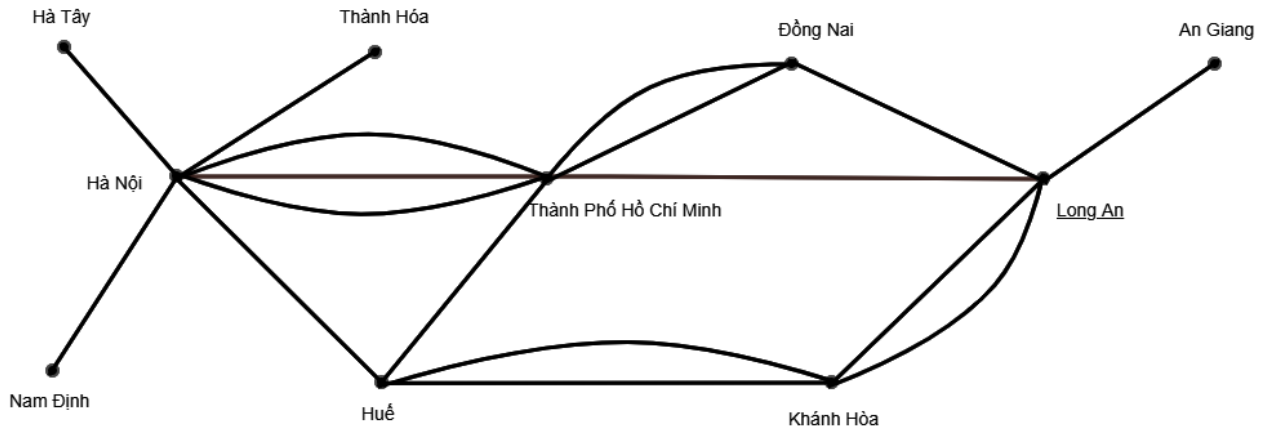
Trong trường hợp giữa hai máy tính nào đó thường xuyên phải truyền tải nhiều thông tin người ta phải nối hai máy bởi nhiều kênh thoại. Mạng với đa kênh thoại giữa các máy được cho trong hình 1.3.



Hình 1.3: Sơ đồ mạng máy tính với đa kênh thoại.

Định nghĩa 2:

Đa đồ thị vô hướng $G = (V, E)$ bao gồm V là các tập đỉnh, và E là tập các cặp không có thứ tự gồm hai phần tử khác nhau của V gọi là các cạnh. Hai cạnh e_1 và e_2 được gọi là cạnh lặp nếu chúng cùng tương ứng với một cặp đỉnh.

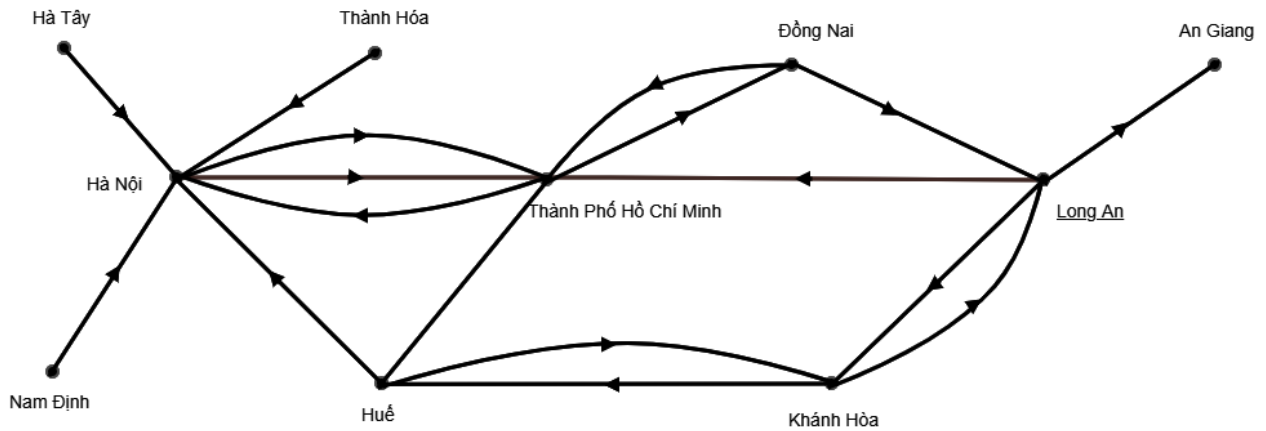


Hình 1.4: Sơ đồ mạng máy tính với kênh thoại thông báo.

Rõ ràng mỗi đơn đồ thị đề là đa đồ thị, nhưng không phải đa đồ thị nào cũng là đơn đồ thị, vì trong đa đồ thị có thể có hai (hoặc nhiều hơn) cạnh nối một cặp đỉnh nào đó. Trong mạng máy tính có thể có những kênh thoại nối một máy nào đó với chính nó (chẳng hạn với mục đích thông báo). Mạng như vậy được cho trong hình 1.4. Khi đó đa đồ thị không thể mô tả được mạng như vậy, bởi vì những khuyên (cạnh nối một đỉnh với chính nó). Trong trường hợp này chúng ta cần sử dụng đến khái niệm giả đồ thị vô hướng.

Định nghĩa 3:

Giả 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 (không nhất thiết phải khác nhau) của V gọi là cạnh. Cạnh e được gọi là khuyên nếu nó có dạng $e = (u, u)$.



Hình 1.5: Mạng máy tính với kênh thoại một chiều.

Các kênh thoại trong mạng máy tính có thể chỉ cho phép truyền tin theo một chiều. Chẳng hạn, trong hình 1.5 máy chủ ở Hà Nội chỉ có thể nhận tin từ các máy ở địa phương, có một số máy chỉ có thể gửi tin đi, còn các kênh thoại cho phép truyền tin theo cả hai chiều được thay thế bởi hai cạnh có hướng ngược chiều nhau.

Định nghĩa 4:

Đơn đồ thị vô hướng $G = (V, E)$ bao gồm V là các đỉnh và E là tập các cặp có thứ tự gồm hai phần tử khác nhau của V gọi là các cung.

Nếu trong mạng có thể có đa kênh thoại một chiều, ta sẽ phải sử dụng đến khái niệm đa đồ thị có hướng.

Định nghĩa 5:

Đa đồ thị có hướng $G = (V, E)$ bao gồm V là tập các đỉnh và E là tập các cặp có thứ tự gồm hai phần tử khác nhau của V gọi là các cung. Hai cung e_1, e_2 tương ứng cùng với một cặp đỉnh được gọi là cung lặp.

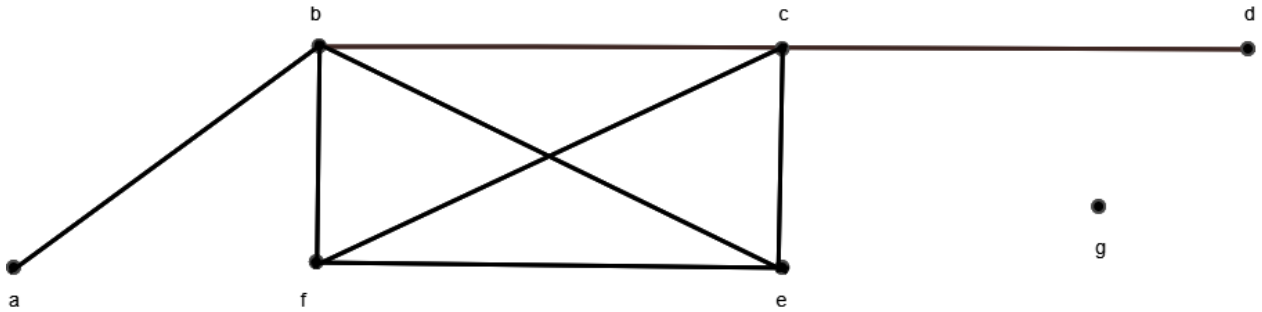
2.5.3. Các thuật ngữ cơ bản

Định nghĩa 1:

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 đỉnh đầu của cạnh (u, v) .

Định nghĩa 2:

Ta gọi bậc đỉnh của v trong đồ thị vô hướng là số cạnh liên thuộc với nó và sẽ ký hiệu là $\deg(v)$.



Hình 1.6: Đồ thị vô hướng.

Thí dụ 1: Xét đồ thị trong hình 1.6, 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. Bậc của các đỉnh có tính chất sau:

Định lý 1: 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.

Chứng minh: 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.

Thí dụ 2: Đồ thị với n đỉnh có bậc là 6 có bao nhiêu cạnh?

Giải: Theo định lý 1 ta có $2m = 6n$. Từ đó suy ra tổng các cạnh của đồ thị là $3n$.

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

Chứng minh: Thực vậy, gọi O và U tương ứng là tập đỉnh bậc lẻ và tập đỉnh bậc chẵn của đồ thị ta có:

$$2m = \sum_{v \in U} \deg(v) + \sum_{v \in O} \deg(v) \quad v \in U \quad v \in O$$

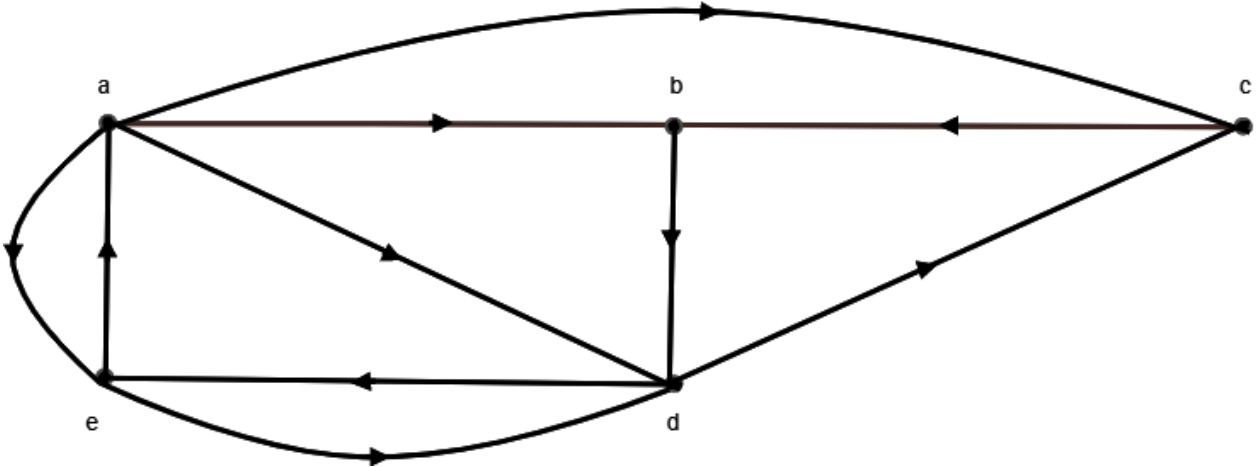
Do $\deg(v)$ là chẵn với v là đỉnh trong U nên tổng thứ nhất ở trên là số chẵn. Từ đó suy ra tổng thứ hai (chính là tổng bậc của đỉnh bậc lẻ) cũng phải là số chẵn, do tất cả các số hạng của nó là số lẻ, nên tổng này phải gồm một số chẵn các số hạng. Vì vậy, số đỉnh bậc lẻ phải là số chẵn.

Định nghĩa 3:

Nếu $e = (u, v)$ là cung của đồ thị vô hướng G thì ta nói hai đỉnh u và v là kề nhau, và nói cung (u, v) nối đỉnh u với đỉnh v hoặc cũng nói cung này là đi ra khỏi đỉnh u và vào đỉnh v . Đỉnh $u(v)$ sẽ được gọi là đỉnh đầu (cuối) của cung (u, v) .

Định nghĩa 4:

Ta gọi bán bậc ra (bán bậc vào) của đỉnh v trong đồ thị có hướng là số cung của đồ thị đi ra khỏi nó (đi vào nó) và ký hiệu là $\deg^+(v)$ ($\deg^-(v)$).



Hình 1.7: Đồ thị có hướng.

Thí dụ 3: Xét đồ thị cho trong hình 1.7. Ta có:

- $\deg^-(a) = 1, \deg^-(b) = 2, \deg^-(c) = 2, \deg^-(d) = 2, \deg^-(e) = 2$
- $\deg^+(a) = 3, \deg^+(b) = 1, \deg^+(c) = 1, \deg^+(d) = 2, \deg^+(e) = 2$

Do mỗi cung $(u,$

$v)$ sẽ được tính một lần trong bán bậc vào của đỉnh v và một lần trong bán bậc ra của đỉnh u nên ta có:

Định lý 2: Giả sử $G = (V, E)$ là đồ thị có hướng. Khi đó:

$$2m = \sum \deg^+(v) = \sum \deg^-(v)$$

$$v \in V$$

Rất nhiều tính chất của đồ thị có hướng không phụ thuộc vào hướng trên các cung của nó. Vì vậy, trong nhiều trường hợp sẽ thuận tiện hơn nếu ta bỏ qua hướng trên các cung của đồ thị. Đồ thị vô hướng thu được bằng cách bỏ qua các hướng trên các cung được gọi là đồ thị vô hướng tương ứng với đồ thị có hướng đã cho.

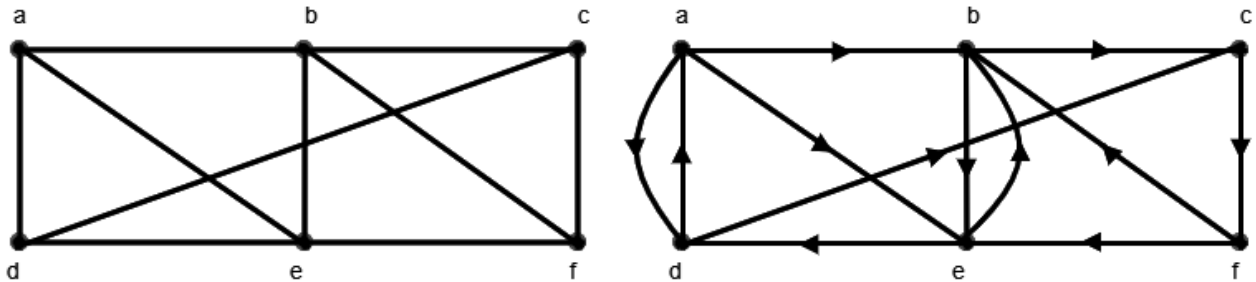
2.5.4. Đường đi, chu trình, độ liên thông

Định nghĩa 1:

Đường đi độ dài n từ đỉnh u đến đỉnh v , trong đó n là số nguyên dương, trên đồ thị vô hướng $G = (V, E)$ là dãy $x_0, x_1, \dots, x(n-1), x(n)$ trong đó $u = x_0, v = x(n), (x(i), x(i+1)) \in E, i = 0, 1, 2, \dots, n-1$. Đường đi nói trên còn có thể biểu diễn dưới dạng dãy các cạnh: $(x_0, x_1), (x_1, x_2), \dots, (x(n-1), x(n))$.

Đỉnh u gọi là đỉnh đầu, còn đỉnh v gọi là đỉnh cuối của đường đi. Đường đi có đỉnh đầu trùng với đỉnh cuối (tức là $u = v$) được gọi là chu trình. Đường đi hay chu trình được gọi là đơn nếu như không có cạnh nào bị lặp lại.

Thí dụ 1: Trên đồ thị vô hướng cho trong hình 1.8: a, d, c, f, e là đường đi đơn độ dài 4. Còn d, e, c, a không là đường đi, do (c, e) không phải là cạnh của đồ thị. Dãy b, c, f, e, b là chu trình độ dài 4. Đường đi a, b, e, d, a, b có độ dài là 5 không phải là đường đi đơn, do cạnh (a, b) có mặt trong nó 2 lần.



Hình 1.8: Đường đi trên đồ thị.

Khái niệm đường đi và chu trình trên đồ thị có hướng được định nghĩa hoàn toàn tương tự như trong trường hợp đồ thị vô hướng, chỉ khác là ta có chú ý đến hướng trên các cung.

Định nghĩa 2:

Đường đi độ dài n từ đỉnh u đến đỉnh v , trong đó n là số nguyên dương, trên đồ thị có hướng $G = (V, E)$ là dãy $x_0, x_1, \dots, x(n-1), x(n)$, trong đó $u = x_0, v = x(n), (x(i), x(i+1)) \in E, i = 0, 1, 2, \dots, n-1$. Đường đi nói trên có thể biểu diễn dưới dạng dãy các cung: $(x_0, x_1), (x_1, x_2), \dots, (x(n-1), x(n))$.

Đỉnh u gọi là đỉnh đầu, còn đỉnh v gọi là đỉnh cuối của đường đi. Đường đi có đỉnh đầu trùng với đỉnh cuối (tức là $u = v$) được gọi là chu trình. Đường đi hay chu trình được gọi là đơn nếu như không có cạnh nào bị lặp lại.

Vấn đề: Xét một mạng máy tính. Một câu hỏi đặt ra là hai máy tính bất kỳ trong mạng này có thể trao đổi thông tin được với nhau hoặc là trực tiếp qua kênh nối cùng

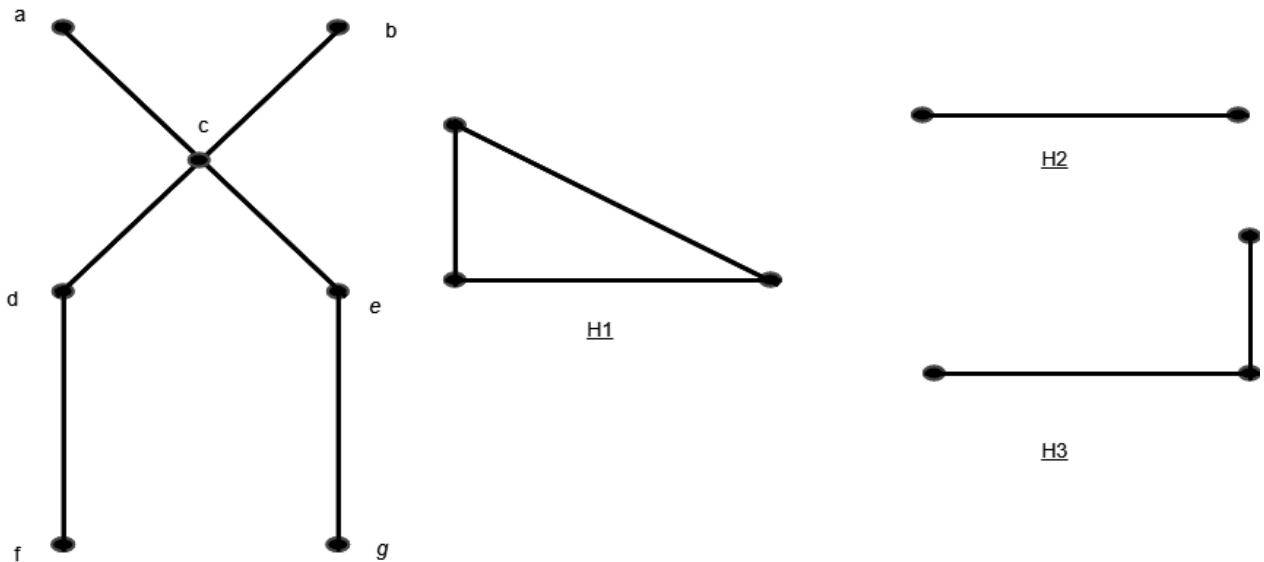
hoặc thông qua một hoặc vài máy tính trung gian trong mạng? Nếu sử dụng đồ thị để biểu diễn mạng máy tính này (trong đó các đỉnh của đồ thị tương ứng với các máy tính, còn các cạnh tương ứng với các kênh nối) câu hỏi đó được phát biểu trong ngôn ngữ đồ thị như sau: Tồn tại hay không đường đi giữa mọi cặp đỉnh của đồ thị.

Định nghĩa 3:

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.

Như vậy hai máy tính bất kỳ trong mạng có thể trao đổi thông tin với nhau khi và chỉ khi đồ thị tương ứng với mạng này là đồ thị liên thông.

Thí dụ 3: Trong hình 1.9: Đồ thị G là liên thông, còn đồ thị H là không liên thông.



Hình 1.9: Đồ thị G và H .

Định nghĩa 4:

Ta gọi đồ thị con của đồ thị $G = (V, E)$ là đồ thị $H = (W, F)$, trong đó $W \subseteq V$ và $F \subseteq E$.

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

Thí dụ 4: Đồ thị H trong hình 1.9 gồm 3 thành phần liên thông $H1$, $H2$, $H3$.

Trong mạng máy tính có thể có những máy (những kênh nối) mà sự hỏng hóc của nó sẽ ảnh hưởng đến việc trao đổi thông tin trong mạng. Các khái niệm tương ứng với tình huống này được đưa ra trong định nghĩa sau.

Định nghĩa 5:

Đỉnh v được gọi là đỉnh rẽ nhánh nếu việc loại bỏ v cùng với các cạnh liên thuộc với nó khỏi đồ thị làm tăng số thành phần liên thông của đồ thị. Cạnh e được gọi là cầu nếu việc loại bỏ nó khỏi đồ thị làm tăng số phần số phần liên thông của đồ thị.

Thí dụ 5: Trong đồ thị G ở hình 1.9, đỉnh d và e là đỉnh rẽ nhánh, còn các cạnh (d, g) và (e, f) là cầu

Đối với đồ thị có hướng có hai khái niệm liên thông phụ thuộc vào việc ta có xét đến hướng trên các cung hay không.

Định nghĩa 6:

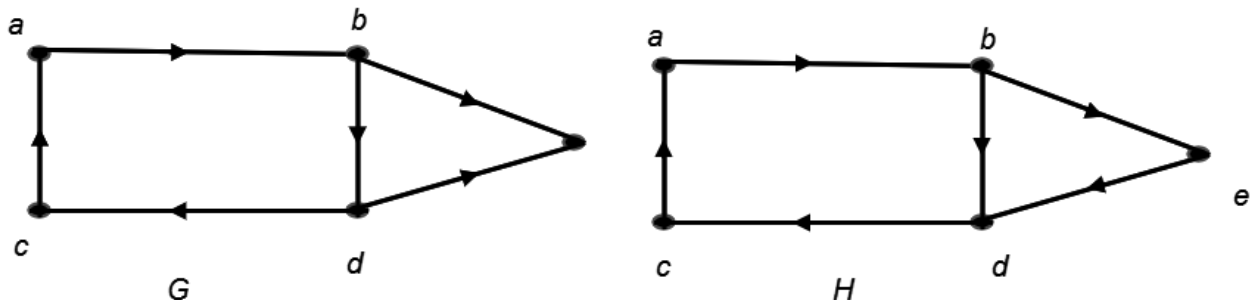
Đồ thị có hướng $G = (V, A)$ được gọi là liên thông mạnh nếu luôn tìm được đường đi giữa hai đỉnh bất kỳ của nó.

Định nghĩa 7:

Đồ thị có hướng $G = (V, A)$ được gọi là liên thông yếu nếu đồ thị vô hướng tương ứng với nó là vô hướng liên thông.

Rõ ràng nếu đồ thị là liên thông mạnh thì nó cũng là liên thông yếu, nhưng điều ngược lại rõ ràng lại là không luôn đúng, như chỉ ra trong ví dụ dưới đây.

Thí dụ 6: Trong hình 1.10 đồ thị G là liên thông mạnh, còn H là liên thông yếu nhưng không là liên thông mạnh.



Hình 1.10: Đồ thị liên thông mạnh G và đồ thị liên thông yếu H .

Một câu hỏi đặt ra là khi nào có thể định hướng các cạnh của một đồ thị vô hướng liên thông để có thể thu được đồ thị có hướng liên thông mạnh? Ta sẽ gọi đồ thị như vậy

là đồ thị định hướng được. Định lý dưới đây cho ta tiêu chuẩn nhận biết một đồ thị có là định hướng được hay không.

Định lý 1: Đồ thị vô hướng liên thông là định hướng được khi và chỉ khi mỗi cạnh của nó nằm trên ít nhất một chu trình.

Chứng minh:

Điều kiện cần. Giả sử (u, v) là một cạnh của một đồ thị. Từ sự tồn tại đường đi có hướng từ u đến v và ngược lại suy ra (u, v) phải nằm trên ít nhất một chu trình.

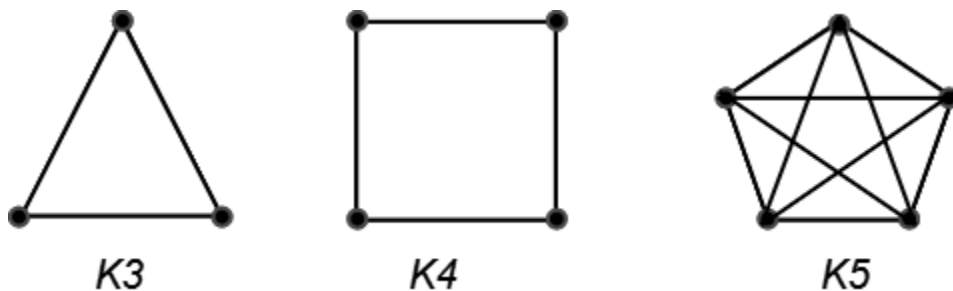
Điều kiện đủ. Thủ tục sau đây cho phép định hướng các cạnh của đồ thị để thu được đồ thị có hướng liên thông mạnh. Giả sử C là một chu trình nào đó trong đồ thị. Định hướng các cạnh trên chu trình này theo một hướng đi vòng theo nó. Nếu tất cả các cạnh của đồ thị là đã được định hướng thì kết thúc thủ tục. Ngược lại, chọn e là một cạnh chưa định hướng có chung đỉnh với ít nhất một trong số các cạnh đã định hướng. Theo giả thiết tìm được chu trình C' chứa cạnh e . Định hướng các cạnh chưa được định hướng của C' theo một hướng dọc theo chu trình này (không định hướng lại các cạnh đã có định hướng). Thủ tục trên sẽ được lặp lại cho đến khi tất cả các cạnh của đồ thị được định hướng. Khi đó ta thu được đồ thị có hướng liên thông mạnh.

2.5.5. Một số dạng đồ thị đặc biệt

Đồ thị đầy đủ:

Đồ thị đầy đủ n đỉnh, ký hiệu bởi K_n , là đơn đồ thị vô hướng mà giữa hai đỉnh bất kỳ của nó luôn có cạnh nối.

Các đồ thị K_3, K_4, K_5 cho trong hình dưới đây

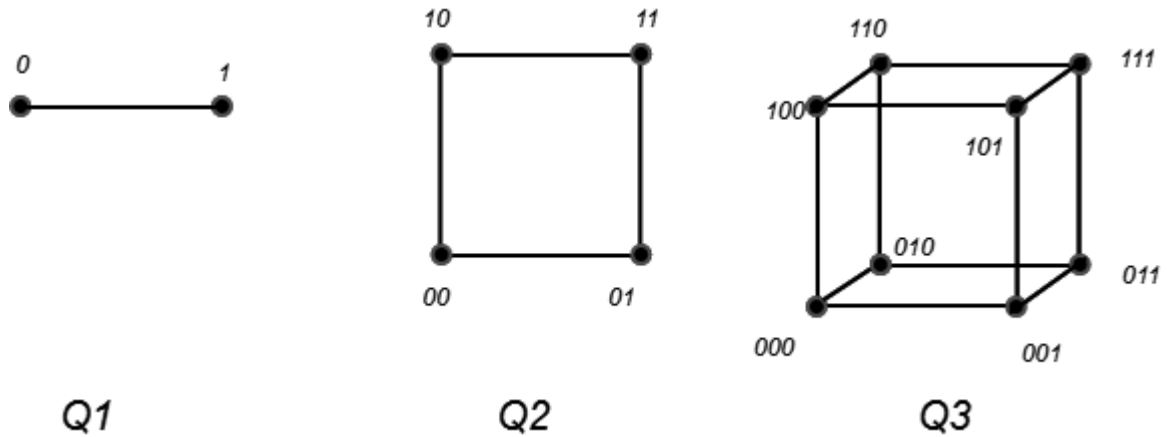


Hình 1.11: Đồ thị đầy đủ.

Đồ thị đầy đủ K_n có tất cả $n(n-1)/2$ cạnh, nó là đơn đồ thị có nhiều cạnh nhất.

Đồ thị vòng:

Đồ thị lập phương n đỉnh Q_n là đồ thị với các đỉnh biểu diễn 2^n xâu nhị phân độ dài n . Hai đỉnh của nó gọi là kề nhau nếu như hai xâu nhị phân tương ứng chỉ khác nhau 1 bit. Hình 1.12 cho thấy Q_n với $n = 1, 2, 3$.



Hình 1.12: Đồ thị lập phương Q_1, Q_2, Q_3 .

Đồ thị hai phía:

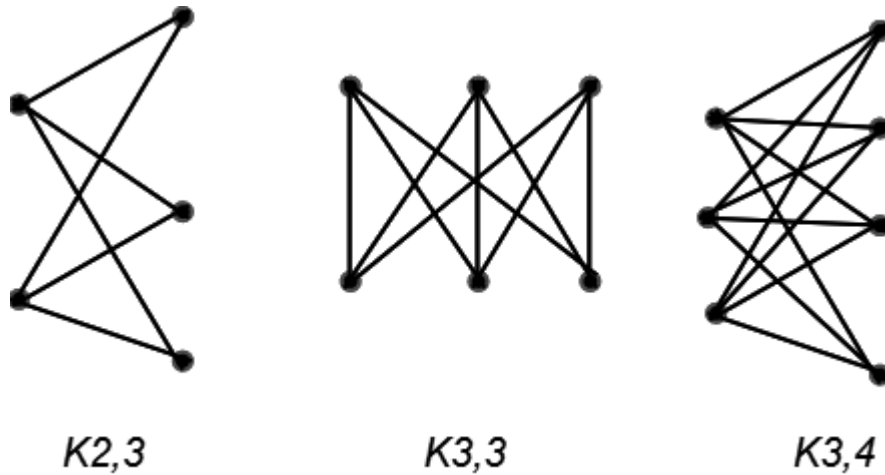
Đơn đồ thị $G = (V, E)$ được gọi là hai phía nếu như tập đỉnh V của nó có thể phân hoạch thành hai tập X và Y sao cho mỗi cạnh của đồ thị chỉ nối một đỉnh nào đó trong X với một đỉnh nào đó trong Y . Khi đó ta sẽ sử dụng ký hiệu $G = (V? Y, E)$ để chỉ đồ thị hai phía với tập đỉnh $X?Y$.

Định lý sau đây cho phép nhận biết một đơn đồ thị có phải là hai phía hay không.

Định lý 1: Đơn đồ thị là đồ thị hai phía khi và chỉ khi nó không chứa chu trình độ dài lẻ.

Để kiểm tra xem một đồ thị liên thông có phải là hai phía hay không có thể áp dụng thủ tục sau. Cho v là một đỉnh bất kỳ của đồ thị. Đặt $X = \{v\}$, còn Y là tập các đỉnh kề của v . Khi đó các đỉnh kề của các đỉnh trong Y phải buộc vào X . Ký hiệu tập các đỉnh như vậy là T . Vì thế nếu phát hiện $T ? Y \neq \emptyset$ thì đồ thị không phải là hai phía, kết thúc ngược lại, đặt $X = X ? T$. Tiếp tục xét như vậy đối với T' là tập các đỉnh kề của T v.v...

Đồ thị hai phía $G = (X ? Y, E)$ với $?X? = m, ?Y? = n$ được gọi là đồ thị hai phía đầy đủ và ký hiệu là $K_{2,3}, K_{3,3}, K_{3,4}$ được cho trong hình 1.13.

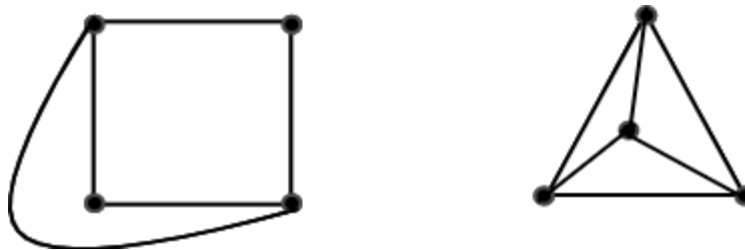


Hình 1.13: Đồ thị hai phai.

Đồ thị phẳng:

Đồ thị được gọi là 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.

Thí dụ đồ thị K_4 là phẳng, vì có thể vẽ nó trên mặt phẳng sao cho các cạnh của nó không cắt nhau ngoài ở đỉnh (hình 1.14).



Hình 1.14: Đồ thị K_4 là đồ thị phẳng.

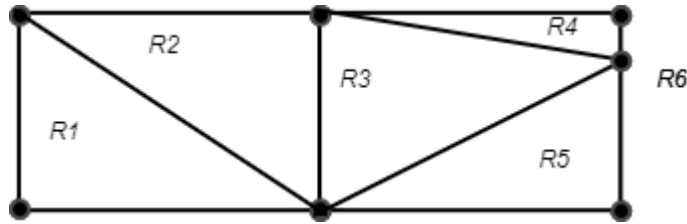
Một điều đáng lưu ý, nếu đồ thị là phẳng thì luôn có thể vẽ nó trên mặt phẳng với các cạnh nối là các đoạn thẳng không cắt nhau ngoài ở đỉnh (ví dụ xem cách vẽ K_4 trong hình 1.14).

Để nhận biết xem một đồ thị có phải là đồ thị phẳng có thể sử dụng định lý Kuratovski, mà để phát biểu nó ta cần một số khái niệm sau: Ta gọi một phép chia cạnh (u, v) của đồ thị là việc loại bỏ cạnh này khỏi đồ thị và thêm vào đồ thị một đỉnh mới w cùng với hai cạnh (u, w) , (w, v) . Hai đồ thị $G(V, E)$ và $H = (W, F)$ được gọi là đồng cấu nếu chúng có thể thu được từ cùng một đồ thị nào đó nhờ phép chia cạnh.

Định lý 2 (Kuratowski): Đồ thị là phẳng khi và chỉ khi nó không chứa đồ thị con đồng cấu với $K_{3,3}$ hoặc K_5 .

Trong trường hợp riêng, đồ thị $K_{3,3}$ hoặc K_5 không phải là đồ thị phẳng. Bài toán về tính phẳng của đồ thị $K_{3,3}$ là bài toán nổi tiếng về ba căn hộ và ba hệ thống cung cấp năng lượng cho chúng. Cần xây dựng hệ thống đường cung cấp năng lượng với mỗi một căn hộ nói trên sao cho chúng không cắt nhau. Đồ thị phẳng còn tìm được những ứng dụng quan trọng trong công nghệ chế tạo mạch in.

Biểu diễn phẳng của đồ thị sẽ chia mặt phẳng ra thành các miền, trong đó có thể cả miền không bị chặn. Thí dụ, biểu diễn phẳng của đồ thị cho trong hình 1.15 chia mặt phẳng ra thành 6 miền R_1, R_2, \dots, R_6 .



Hình 1.15: Các miền tương ứng với biểu diễn phẳng của đồ thị.

Euler đã chứng minh được rằng các cách biểu diễn khác nhau của một đồ thị đều chia mặt phẳng ra thành cùng một số miền. Để chứng minh được điều đó, Euler đã tìm được mối liên hệ giữa số miền, số đỉnh của đồ thị và số cạnh của đồ thị phẳng sau đây.

Định lý 3 (Công thức Euler): Giả sử G là đồ thị phẳng liên thông với n đỉnh, m cạnh. Gọi r là số miền của mặt phẳng bị chia bởi biểu diễn phẳng của G . Khi đó:

$$r = m - n + 2$$

Có thể chứng minh định lý bằng qui nạp. Xét thí dụ minh họa cho áp dụng công thức Euler.

Thí dụ: Cho G là đồ thị phẳng liên thông với 20 đỉnh, mỗi đỉnh đều có bậc là 3. Hỏi mặt phẳng bị chia làm bao nhiêu phần bởi biểu diễn phẳng của đồ thị G ?

Giải: Do mỗi đỉnh của đồ thị đều có bậc là 3, nên tổng bậc của các đỉnh là $3 \times 20 = 60$. Từ đó suy ra số cạnh của đồ thị $m = 60/2 = 30$. Vì vậy, theo công thức Euler, số miền cần tìm là $r = 30 - 20 + 2 = 12$

2.5.6. Biểu diễn đồ thị trên máy vi tính

Để lưu trữ đồ thị và thực hiện các thuật toán khác nhau với đồ thị trên máy tính cần phải tìm những cấu trúc dữ liệu thích hợp để mô tả đồ thị. Việc chọn cấu trúc dữ liệu nào để biểu diễn đồ thị có tác động rất lớn đến hiệu quả của thuật toán. Vì vậy, việc chọn lựa cấu trúc dữ liệu để biểu diễn đồ thị phụ thuộc vào từng tình huống cụ thể (bài toán và thuật toán cụ thể). Trong mục này chúng ta sẽ xét một số phương pháp cơ bản được sử dụng để biểu diễn đồ thị trên máy tính, đồng thời cũng phân tích một cách ngắn gọn những ưu điểm cũng như những nhược điểm của chúng.

Ma trận kề. Ma trận trọng số:

Xét đơn đồ thị vô hướng $G = (V, E)$, với tập đỉnh $V = \{1, 2, \dots, n\}$, tập cạnh $E = \{e_1, e_2, \dots, e_m\}$. Ta gọi ma trận kề của đồ thị G là ma trận.

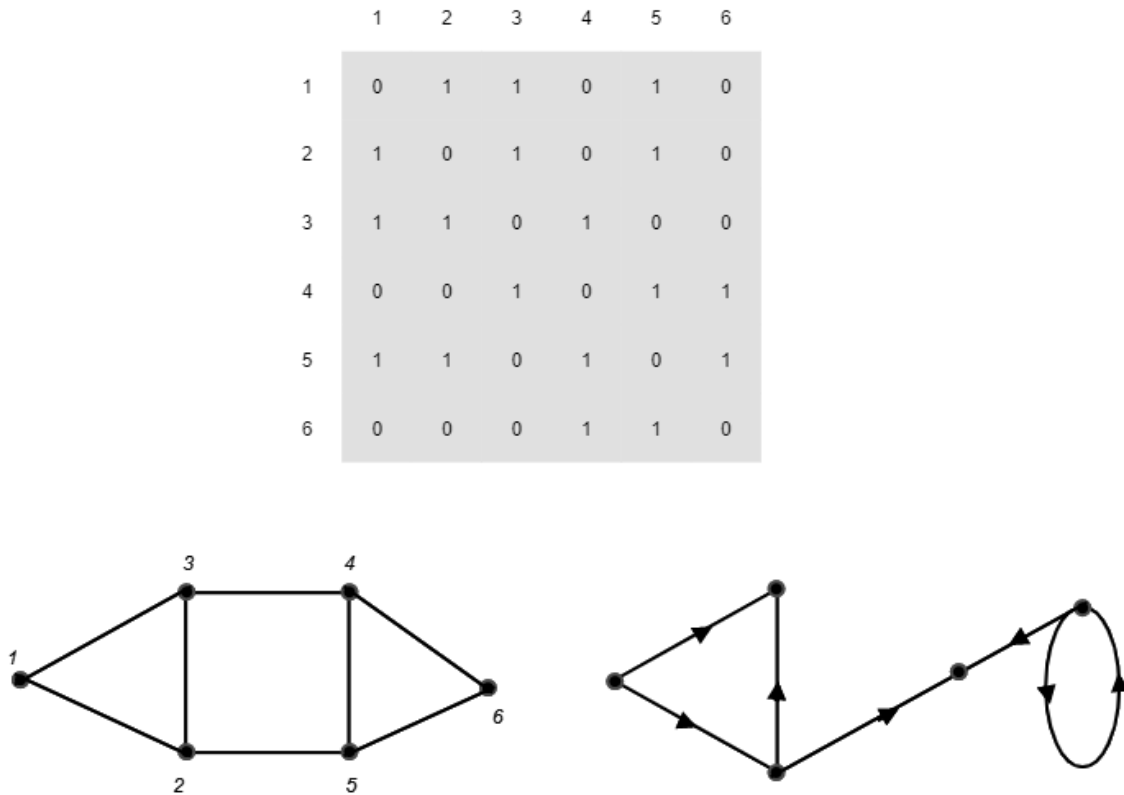
$$A = \{a_{ij} : i, j = 1, 2, \dots, n\}$$

Với các phần tử được xác định theo qui tắc sau đây:

$$a_{ij} = 0, \text{ nếu } (i, j) \notin E \text{ và}$$

$$a_{ij} = 1, \text{ nếu } (i, j) \in E, i, j = 1, 2, \dots, n.$$

Thí dụ 1: Ma trận kề của đồ thị vô hướng cho trong hình 1.16 là:



Hình 1.16: Đồ thị vô hướng G và Đồ thị có hướng G_1 .

Các tính chất của ma trận kề:

- Rõ ràng ma trận kề của đồ thị vô hướng là ma trận đối xứng, tức là:
 $a[i,j] = a[j,i]$, $i, j = 1, 2, \dots, n$.
 ngược lại, mỗi $(0,1)$ mã trận đối xứng cấp n sẽ tương ứng, chính xác đến cách đánh số đỉnh (còn nói là: chính xác đến đẳng cấu), với một đơn đồ thị vô hướng n đỉnh
- Tổng các phần tử trên dòng i (cột j) của ma trận kề chính bằng bậc của đỉnh i (đỉnh j).
- Nếu ký hiệu a_{ij}^p , $i, j = 1, 2, \dots, n$. cho ta số đường đi khác nhau từ đỉnh i đến đỉnh j qua $p-1$ đỉnh trung gian. Ma trận kề của đồ thị có hướng được định nghĩa một cách hoàn toàn tương tự.

Thí dụ 2: Đồ thị có hướng G_1 cho trong hình 1.16 có ma trận kề là ma trận sau:

	1	2	3	4	5	6
1	0	1	1	0	0	0
2	0	0	0	0	0	0
3	0	1	0	1	0	0
4	0	0	0	0	0	0
5	0	0	0	1	0	1
6	0	0	0	0	1	0

Hình 1.17: Ma trận kề cho đồ thị có hướng G_1 cho trong hình 1.16.

Lưu ý rằng ma trận kề của đồ thị có hướng không phải là ma trận đối xứng.

Chú ý: Trên đây chúng ta chỉ xét đơn đồ thị. Ma trận kề của đa đồ thị có thể xây dựng hoàn toàn tương tự, chỉ khác là thay vì ghi vào vị trí $a[i,j]$ nếu (i,j) là cạnh của đồ thị, chúng ta sẽ ghi k là số cạnh nối hai đỉnh i, j .

Trong rất nhiều vấn đề ứng dụng của lý thuyết đồ thị, mỗi cạnh $e = (u, v)$ của đồ thị được gán với một con số $c(e)$ (còn viết là $c(u, v)$) gọi là trọng số của cạnh e . Đồ thị trong trường hợp như vậy được gọi là đồ thị có trọng số.

$C = \{c[i,j], i,j = 1, 2, \dots, n\}$ với

$c[i,j] = c(i,j)$ nếu $(i,j) \in E$ và

$c[i,j] = 0$ nếu $(i,j) \notin E$

trong đó số θ , tùy trường hợp cụ thể, có thể được đặt bằng một trong các giá trị sau: $0, +\infty, -\infty$.

Ưu điểm lớn nhất của phương pháp biểu diễn đồ thị bằng ma trận kề (hoặc ma trận trọng số) là để trả lời câu hỏi: Hai đỉnh u, v có kề nhau trên đồ thị hay không, chúng ta chỉ phải thực hiện một phép so sánh, nhược điểm lớn nhất của phương pháp này là: không phụ thuộc vào số cạnh của đồ thị, ta luôn phải sử dụng n^2 đơn vị bộ nhớ để lưu trữ ma trận kề của nó.

Danh sách cạnh chung (cung):

Trong trường hợp đồ thị thưa (đồ thị có số cạnh m thỏa mãn bất đẳng thức : $m < 6n$) người ta thường dùng cách biểu diễn đồ thị dưới dạng danh sách cạnh.

Trong cách biểu diễn đồ thị bởi danh sách cạnh (cung) chúng ta sẽ lưu trữ danh sách tất cả các cạnh (cung) của đồ thị vô hướng (có hướng). Một cạnh (cung) $e = (e, y)$ của đồ thị sẽ tương ứng với hai biến $Dau(e)$, $Cuoi(e)$. Như vậy, để lưu trữ đồ thị ta cần sử dụng $2m$ đơn vị bộ nhớ. Nhược điểm của cách biểu diễn này là để xác định những đỉnh nào của đồ thị là kề với một đỉnh cho trước chúng ta phải làm cỡ m phép so sánh (khi duyệt qua danh sách tất cả các cạnh của đồ thị).

Chú ý: Trong trường hợp đồ thị có trọng số ta cần thêm m đơn vị bộ nhớ để lưu trữ trọng số của các cạnh.

Thí dụ 3: Danh sách cạnh (cung) của đồ thị G (G_1) cho trong hình 1.16 là:

Đầu	Cuối
1	2
1	3
1	5
2	3
2	5
3	4
4	5
4	6
5	6

Hình 1.18: Danh sách cạnh của G .

Đầu	Cuối
1	2
1	3
3	2
3	4
5	4
5	6
6	5

Hình 1.19: Danh sách cung của G_1 .

Danh sách kề:

Trong rất nhiều vấn đề ứng dụng của lý thuyết đồ thị, cách biểu diễn đồ thị dưới dạng danh sách kề là cách biểu diễn thích hợp nhất được sử dụng.

Trong cách biểu diễn này, với mỗi đỉnh v của đồ thị chúng ta lưu trữ danh sách các đỉnh kề với nó, mà ta sẽ ký hiệu là

$$Ke(v) = \{ u \in V : (v, u) \in E \}$$

Khi đó vòng lặp được thực hiện với mỗi phần tử trong danh sách này theo thứ tự các phần tử được sắp xếp trong nó sẽ được viết như sau:

for $u \in Ke(v)$ do...

Chẳng hạn, trên PASCAL có thể mô tả danh sách này như sau (Gọi là **Forward Star**):

Const

$m = 1000$; ? m – số cạnh?

$n = 100$; ? n – số đỉnh ?

var

Ke: array[1.. m] of integer;

Tro: array[1..n+1] of interger;

Trong đó Tro[i] ghi nhận vị trí bắt đầu của danh sách kề của đỉnh i, $i = 1, 2, \dots, n$

$\text{Tro}[n+1] = 2m+1$.

Khi đó dòng lệnh qui ước

for $i \in \text{Ke}(v)$ do

begin

.....

end.

Có thể thay thế bởi cấu trúc lệnh trên PASCAL như sau:

for $i := \text{Tro}[v]$ to $\text{Tro}[v+1] - 1$ do

begin

$U := \text{Ke}[i]$;

.....

end.

Trong rất nhiều thuật toán làm việc với đồ thị chúng ta thường xuyên phải thực hiện các thao tác: Thêm hoặc bớt một số cạnh. Trong trường hợp này cấu trúc dữ liệu dùng ở trên là không thuận tiện. Khi đó nên chuyển sang sử dụng danh sách kề liên kết (Linked Adjacency) như mô tả trong chương trình nhập danh sách kề của đồ thị từ bàn phím và đưa danh sách đó ra màn hình sau đây:

Program AdjList;

Const

maxV = 100;

Type

Link = ^node;

node = record;

v: interger;

next: link;

end;

Var

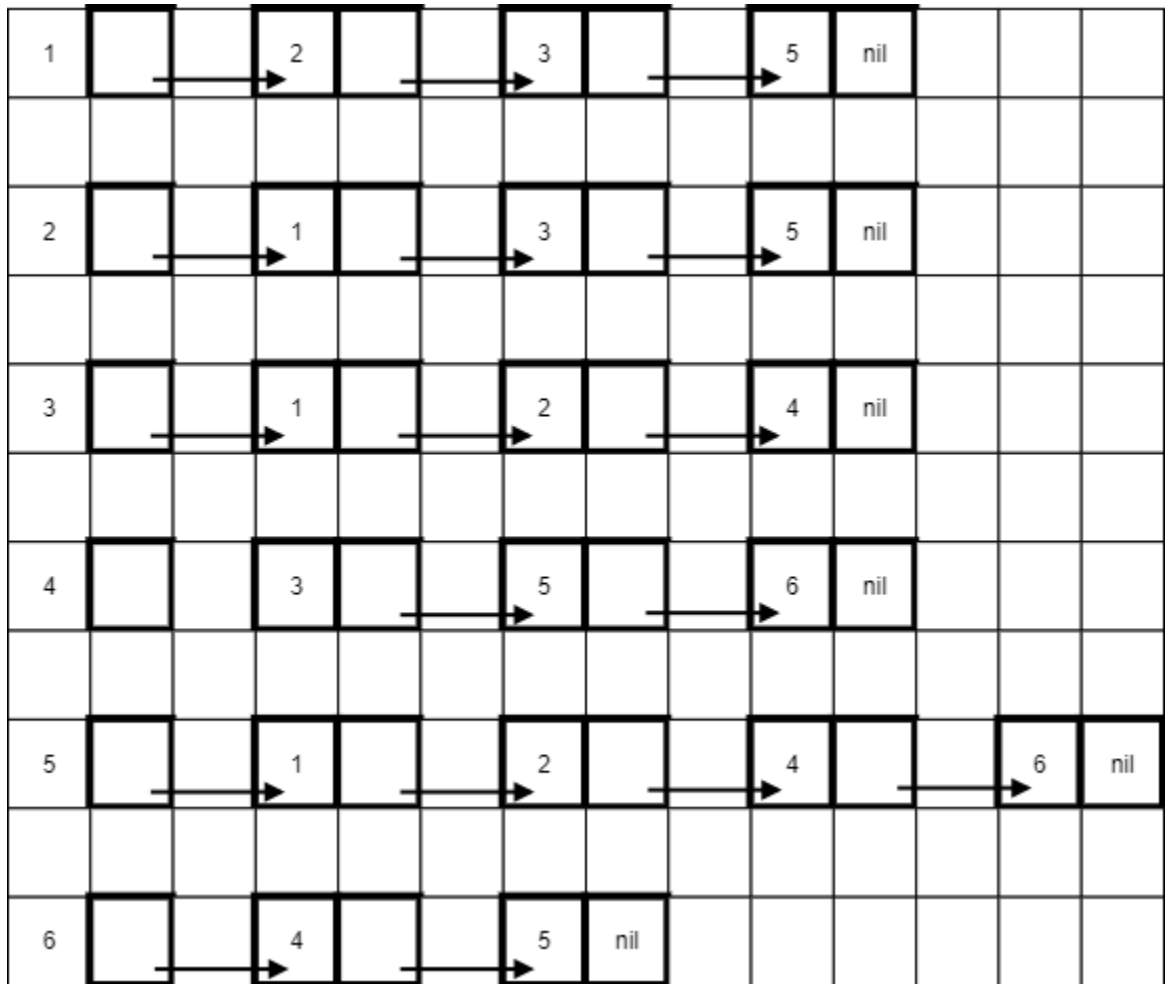
j,x,y,m,n,u,v:interger;

```

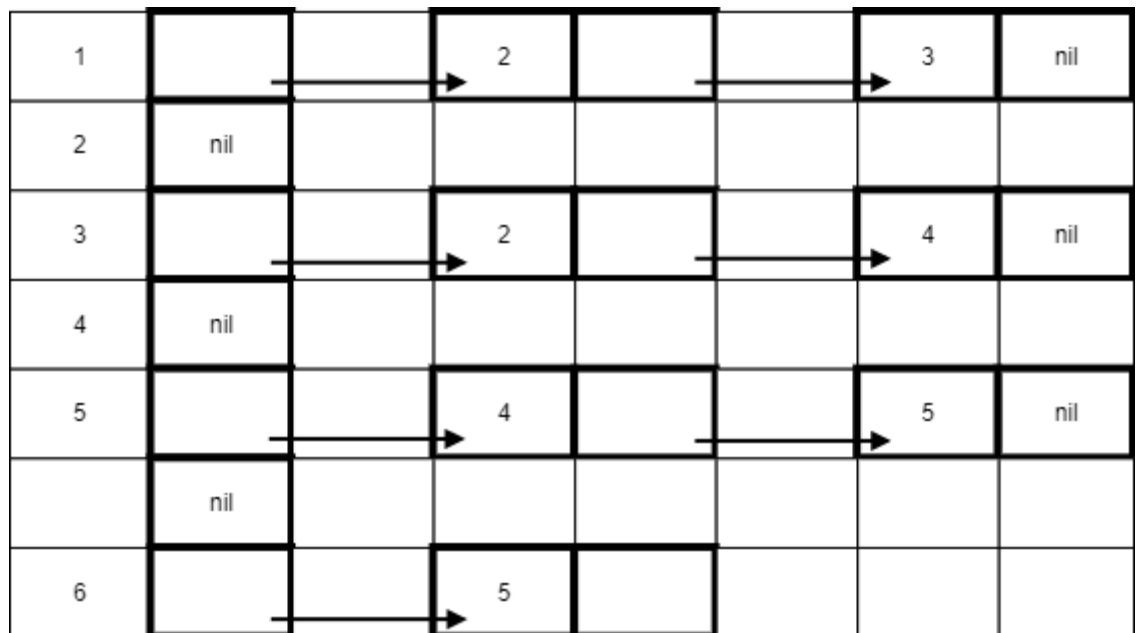
t:link;
Ke:array[1..Vmax] of link;
begin
write('Cho so canh va dinh cua do thi:'); readln(m,n);
(*Khoi tao*)
for j:=1 to n do Ke[j]: = nil;
for j:=1 to m do
begin
write('Cho dinh dau va cuoi cua canh 'j,':');readln(x,y);
new(t); t^.v:=x, t^.next:=Ke[y]; Ke[y]:=t;
new(t); t^.v:=y, t^.next:=Ke[x]; Ke[x]:=t;
end;
writeln('Danh sach ke cua cac dinh cua do thi:');
for J:=1 to m do
begin
writeln('Danh sachcac dinh ke cua dinh 'j,':');
t:=Ke[j];
while t^.next<>nil do
begin
write(t^.v:4);
t:=t^.next;
end;
end;
readln;
End.

```

Thí dụ 4: Danh sách kề của các đồ thị trong hình 1.16 được mô tả trong hình sau:



Hình 1.20: Danh sách kề của đồ thị vô hướng G cho trong hình 1.16.



Hình 1.21: Danh sách kề của đồ thị có hướng G_1 cho trong hình 1.16.

Đề ý rằng trong cách biểu diễn này chúng ta cần phải sử dụng cỡ $m+n$ đơn vị bộ nhớ. Trong các thuật toán mô tả ở các phần tiếp theo hai cấu trúc danh sách kề và ma trận trọng số được sử dụng thường xuyên.

2.5.7. Các thuật toán tìm kiếm trên đồ thị và ứng dụng

Rất nhiều thuật toán trên đồ thị được xây dựng dựa trên cơ sở duyệt tất cả các đỉnh của đồ thị sao cho mỗi đỉnh của nó được viếng thăm đúng một lần. Vì vậy, việc xây dựng những thuật toán cho phép duyệt một cách hệ thống tất cả các đỉnh của đồ thị là một vấn đề quan trọng thu hút sự quan tâm nghiên cứu của nhiều tác giả. Những thuật toán như vậy chúng ta sẽ gọi là thuật toán tìm kiếm trên đồ thị. Trong mục này chúng ta sẽ giới thiệu hai thuật toán cơ bản trên đồ thị: Thuật toán tìm kiếm theo chiều sâu (Depth First Search) và Thuật toán tìm kiếm theo chiều rộng (Breadth First Search) và ứng dụng của chúng vào việc giải một số bài toán trên đồ thị.

Trong mục này chúng ta sẽ xét đồ thị vô hướng $G = (V, E)$, với đỉnh n và m cạnh.

Chúng ta sẽ quan tâm đến việc đánh giá hiệu quả của các thuật toán trên đồ thị, mà một trong những đặc trưng quan trọng nhất là độ phức tạp tính toán, tức là số phép toán mà thuật toán cần phải thực hiện trong tính huống xấu nhất được biểu diễn như hàm của kích thước đầu vào của bài toán. Trong các thuật toán trên đồ thị, đầu vào là đồ thị $G = (V, E)$, vì vậy, kích thước của bài toán sẽ là số đỉnh n và số cạnh m của đồ thị. Khi đó độ phức tạp tính toán của thuật toán sẽ được biểu diễn như là hàm của hai biến số $f(n, m)$ là số phép toán nhiều nhất cần thực hiện theo thuật toán đối với mọi đồ thị n đỉnh và m cạnh. Khi so sánh tốc độ tăng của hai hàm nhận giá trị không âm $f(n)$ và $g(n)$ chúng ta sẽ sử dụng ký hiệu sau:

$f(n)=O(g(n)) \Leftrightarrow$ tìm được các hằng số $C, N \geq 0$ sao cho $f(n) \leq C g(n)$ với mọi $n \leq N$

Tương tự như vậy nếu $f(n_1, n_2, \dots, n_k), g(n_1, n_2, \dots, n_k)$ là các hàm nhiều biến ta viết:

$f(n_1, n_2, \dots, n_k) = O(g(n_1, n_2, \dots, n_k))$

\Leftrightarrow tìm được các hằng số $C, N > 0$ sao cho

$f(n_1, n_2, \dots, n_k) \leq C g(n_1, n_2, \dots, n_k)$ với mọi $n_1, n_2, \dots, n_k \geq N$

Nếu độ phức tạp tính toán của thuật toán là $O(g(n))$ thì ta sẽ còn nói là nó đòi hỏi thời gian tính cỡ $O(g(n))$.

Tìm kiếm theo chiều sâu trên đồ thị:

Ý tưởng chính của thuật toán có thể trình bày như sau. Ta sẽ bắt đầu tìm kiếm từ một đỉnh v_0 nào đó của đồ thị. Sau đó chọn u là một đỉnh tùy ý kề với v_0 và lặp lại quá trình đối với u . Ở bước tổng quát, giả sử ta đang xét đỉnh v . Nếu như trong các số các đỉnh kề với v tìm được đỉnh w là chưa được xét thì ta sẽ xét đỉnh này (nó sẽ trở thành đã xét) và bắt đầu từ nó ta sẽ bắt đầu quá trình tìm kiếm còn nếu như không còn đỉnh nào kề với v là chưa xét thì ta nói rằng đỉnh này đã duyệt xong và quay trở lại tiếp tục tìm kiếm từ đỉnh mà trước đó ta đến được đỉnh v (nếu $v = v_0$, thì kết thúc tìm kiếm). Có thể nói nôm na là tìm kiếm theo chiều sâu bắt đầu từ đỉnh v được thực hiện trên cơ sở tìm kiếm theo chiều sâu từ tất cả các đỉnh chưa xét kề với v . Quá trình này có thể mô tả bởi thủ tục đệ qui sâu đây:

Procedure DFS(v);

(*tìm kiếm theo chiều sâu bắt đầu từ đỉnh v ; các biến Chuaxet, Ke là biến toàn cục*)

Begin

Tham_dinh(v);

Chuaxet[v]:=false;

For $u \in Ke(v)$ do

If Chuaxet[u] then DFS(u);

End; (*đỉnh v đã duyệt xong*)

Khi đó, tìm kiếm theo chiều sâu trên đồ thị được thực hiện nhờ thuật toán sau:

Begin

(*Initialization*)

for $v \in V$ do Chuaxet[v]:=true;

for $v \in V$ do

if Chuaxet[v] then DFS(v);

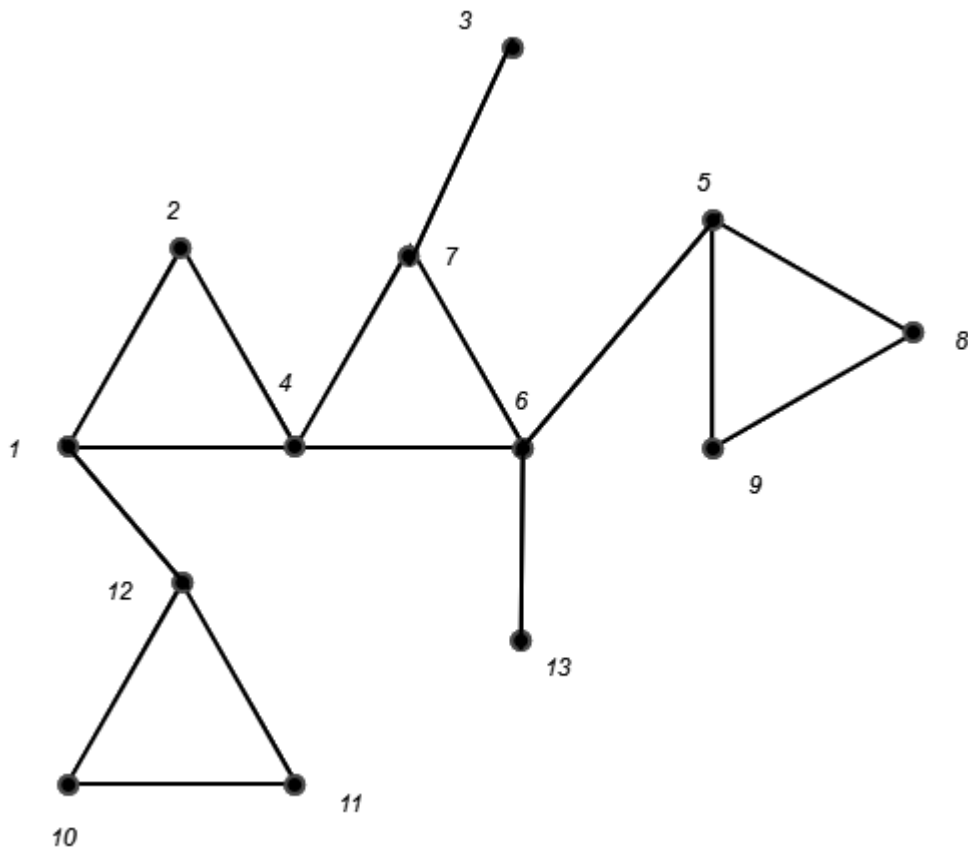
End.

Rõ ràng lệnh gọi DFS(v) sẽ cho phép đến thăm tất cả các đỉnh thuộc cùng thành phần liên thông với đỉnh v , bởi vì sau khi thăm đỉnh là lệnh gọi đến thủ tục DFS đối với tất cả các đỉnh kề với nó. Mặt khác, do mỗi khi thăm đỉnh v xong, biến Chuaxet[v] được

đặt lại giá trị false nên mỗi đỉnh sẽ được thăm đúng một lần. Thuật toán lần lượt sẽ tiến hành tìm kiếm từ các đỉnh chưa được thăm, vì vậy, nó sẽ xét qua tất cả các đỉnh của đồ thị (không nhất thiết phải là liên thông).

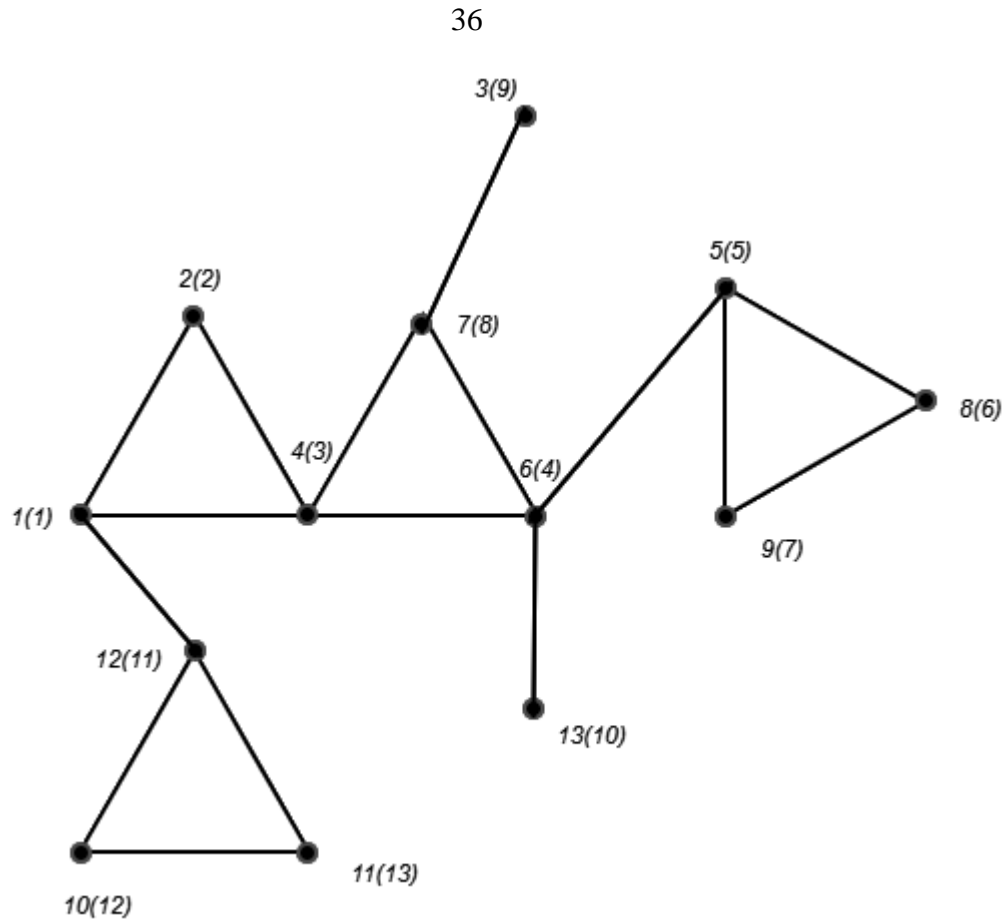
Để đánh giá độ phức tạp tính toán của thủ tục, trước hết nhận thấy rằng số phép toán cần thực hiện trong hai chu trình của thuật toán (hai vòng for ở chương trình chính) là cỡ n . Thủ tục DFS phải thực hiện không quá n lần. Tổng số phép toán cần phải thực hiện trong các thủ tục này là $O(n+m)$, do trong các thủ tục này ta phải xét qua tất cả các cạnh và các đỉnh của đồ thị. Vậy độ phức tạp tính toán của thuật toán là $O(n+m)$.

Thí dụ 1: Xét đồ thị cho trong hình 1.22 gồm 13 đỉnh, các đỉnh được đánh số từ 1 đến 13 như sau:



Hình 1.22: Đồ thị H .

Khi đó các đỉnh của đồ thị được đánh số lại theo thứ tự chúng được thăm theo thủ tục tìm kiếm theo chiều sâu mô tả ở trên như hình 1.22. Giả thiết rằng các đỉnh trong danh sách kề của đỉnh v ($Ke(v)$) được sắp xếp theo thứ tự tăng dần của chỉ số.



Hình 1.23: Chỉ số mới (trong dấu ngoặc) của các đỉnh được đánh giá lại theo thứ tự chúng được thăm trong thuật toán tìm kiếm theo chiều sâu.

Thuật toán tìm kiếm theo chiều sâu trên đồ thị vô hướng trình bày ở trên dễ dàng có thể mô tả lại cho đồ thị có hướng. Trong trường hợp đồ thị có hướng, DFS(v) sẽ cho phép thăm tất cả các đỉnh u nào mà từ v có đường đi đến u. Độ phức tạp tính toán của thuật toán là $O(n+m)$.

Tìm kiếm theo chiều rộng trên đồ thị:

Để ý rằng trong thuật toán tìm kiếm theo chiều sâu đỉnh được thăm càng muộn sẽ càng sớm trở thành đã duyệt xong. Điều đó là hệ quả tất yếu của việc các đỉnh được thăm sẽ được kết nạp vào trong ngăn xếp (STACK). Tìm kiếm theo chiều rộng trên đồ thị, nếu nói một cách ngắn gọn, được xây dựng trên cơ sở thay thế ngăn xếp (STACK) bởi hàng đợi (QUEUE). Với sự cải biên như vậy, đỉnh được thăm càng sớm sẽ càng sớm trở thành đã duyệt xong (tức là càng sớm rời khỏi hàng đợi). Một đỉnh sẽ trở thành đã duyệt xong ngay sau khi ta xét xong tất cả các đỉnh kề (chưa được thăm) với nó. Thủ tục có thể mô tả như sau:

Procedure BFS(v);
 (*Tìm kiếm theo chiều rộng bắt đầu từ đỉnh v, các biến Chuaxet, Ke là biến cục
 bộ*)

```
begin
  QUEUE:=  $\emptyset$  ;
  QUEUE  $\leftarrow$  v; (*kết quả nạp vào QUEUE*)
  Chuaxet[v]:=false;
  While QUEUE  $\neq \emptyset$  do
  Begin
    p  $\leftarrow$  QUEUE; (*lấy p từ QUEUE:*)
    Tham_dinh(p);
    For u  $\in$  Ke(v) do
    If Chuaxet[u] then
    Begin
      QUEUE  $\leftarrow$  u;
      Chuaxet[u]:=false;
    End;
  End;
end;
```

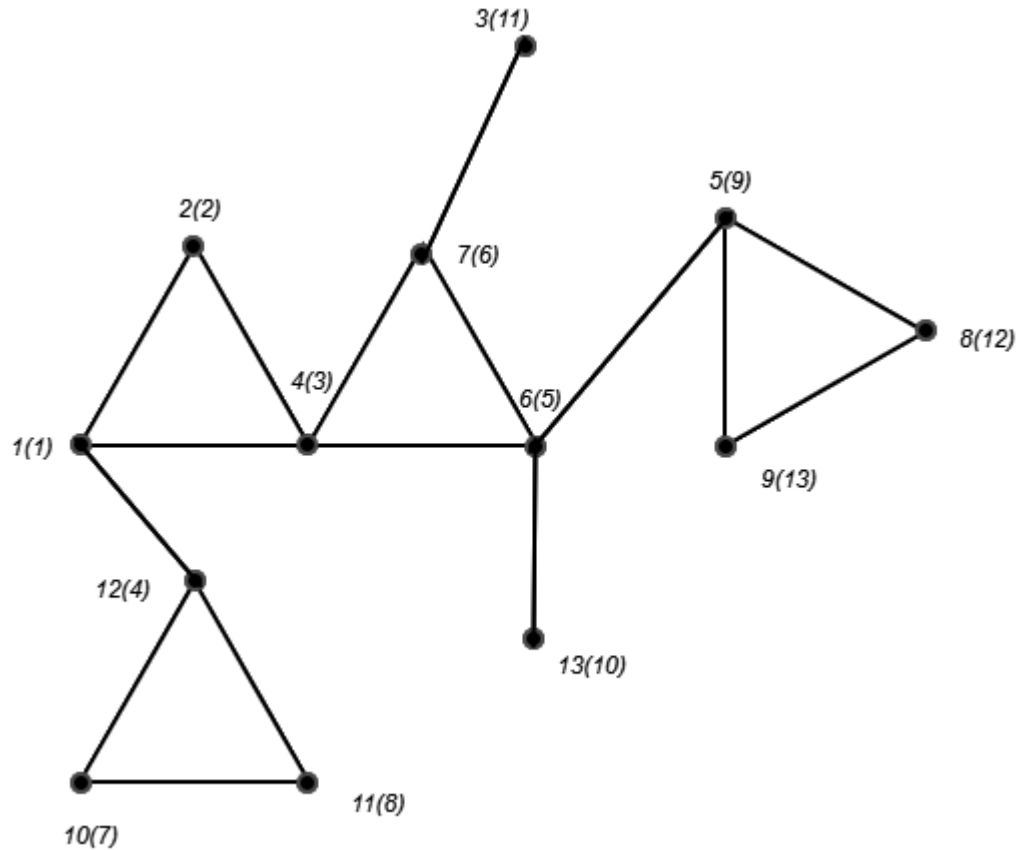
Khi đó, tìm kiếm theo chiều rộng trên đồ thị được thực hiện nhờ thuật toán sau:

```
Begin
  (*Initialization*)
  for f  $\in$  V do
    Chuaxet[v]:=true;
  for v  $\in$  V do
    if Chuaxet[v] then BFS(v);
  End.
```

Lập luận tương tự như trong thủ tục tìm kiếm theo chiều sâu, có thể chỉ ra được rằng lệnh gọi BFS(v) sẽ cho phép thăm đến tất cả các đỉnh thuộc cùng thành phần liên

thông với đỉnh v , và mỗi đỉnh của đồ thị sẽ được thăm đúng một lần. Độ phức tạp tính toán của thuật toán là $O(m+n)$.

Thí dụ 2: Xét đồ thị trong hình 1.24. Thứ tự thăm đỉnh của đồ thị theo thuật toán tìm kiếm theo chiều rộng được ghi trong ngoặc.



Hình 1.24: Chỉ số mới (trong dấu ngoặc) của các đỉnh được đánh lại theo thứ tự chúng được thăm trong thuật toán tìm kiếm theo chiều sâu.

Bài toán tìm đường đi giữa hai đỉnh:

Giả sử s và t là hai đỉnh nào đó của đồ thị. Hãy tìm đường đi từ s đến t . Như trên đã phân tích, thủ tục DFS(s) (BS(s)) sẽ cho thăm tất cả các đỉnh thuộc cùng một thành phần liên thông với s . vì vậy, sau khi thực hiện xong thủ tục, nếu Chuaxet[t]=true, thì điều đó có nghĩa là không có đường đi từ s đến t , còn nếu Chuaxet[t]=false thì t thuộc cùng thành phần liên thông với s , hay nói một cách khác: tồn tại đường đi từ s đến t . Trong trường hợp tồn tại đường đi, để ghi nhận đường đi, ta dùng thêm biểu thức Truoc[v] để ghi nhận đỉnh đi trước đỉnh v trong đường đi tìm kiếm từ s đến v . Khi đó, đối với thủ tục DFS(v) cần sửa đổi câu lệnh if trong nó như sau:

If Chuaxet[u] then

Begin

Truoc[u]:=v;

DFS(u);

End;

Còn đối với thủ tục BFS(v) cần sửa đổi câu lệnh if trong nó như sau:

If Chuaxet [u] then

Begin

QUEUE \leftarrow u;

Chuaxet[u]:=false;

Truoc[u]:=p;

End;

Chú ý: Đường đi tìm được theo thuật toán tìm kiếm theo chiều rộng là đường đi ngắn nhất (theo số cạnh) từ s đến t. Điều này suy trực tiếp từ thứ tự thăm đỉnh theo thuật toán tìm kiếm theo chiều rộng.

CHƯƠNG 3. PHÂN TÍCH THUẬT TOÁN

3.1. Tổng quan thuật toán Hamilton

3.1.1. Giới thiệu thuật toán

Năm 1857, nhà toán học người Ailen là Hamilton (1805-1865) đưa ra trò chơi “đi vòng quanh thế giới” như sau.

Cho một hình thập nhị diện đều (đa diện đều có 12 mặt, 20 đỉnh và 30 cạnh), mỗi đỉnh của hình mang tên một thành phố nổi tiếng, mỗi cạnh của hình (nối hai đỉnh) là đường đi lại giữa hai thành phố tương ứng. Xuất phát từ một thành phố, hãy tìm đường đi thăm tất cả các thành phố khác, mỗi thành phố chỉ một lần, rồi trở về chỗ cũ.

Trước Hamilton, có thể là từ thời Euler, người ta đã biết đến một câu đố học búa về “đường đi của con mã trên bàn cờ”. Trên bàn cờ, con mã chỉ có thể đi theo đường chéo của hình chữ nhật 2×3 hoặc 3×2 ô vuông. Giả sử bàn cờ có 8×8 ô vuông. Hãy tìm đường đi của con mã qua được tất cả các ô của bàn cờ, mỗi ô chỉ một lần rồi trở lại ô xuất phát.

Bài toán này được nhiều nhà toán học chú ý, đặc biệt là Euler, De Moivre, Vandermonde, v.v...

Hiện nay đã có nhiều lời giải và phương pháp giải cũng có rất nhiều, trong đó có quy tắc: mỗi lần bố trí con mã ta chọn vị trí mà tại vị trí này số ô chưa dùng tới do nó không chẵn là ít nhất.

Một phương pháp khác dựa trên tính đối xứng của hai nửa bàn cờ. Ta tìm hành trình của con mã trên một nửa bàn cờ, rồi lấy đối xứng cho nửa bàn cờ còn lại, sau đó nối hành trình của hai nửa đã tìm lại với nhau.

Trò chơi và câu đố trên dẫn tới việc khảo sát một lớp đồ thị đặc biệt, đó là đồ thị Hamilton.

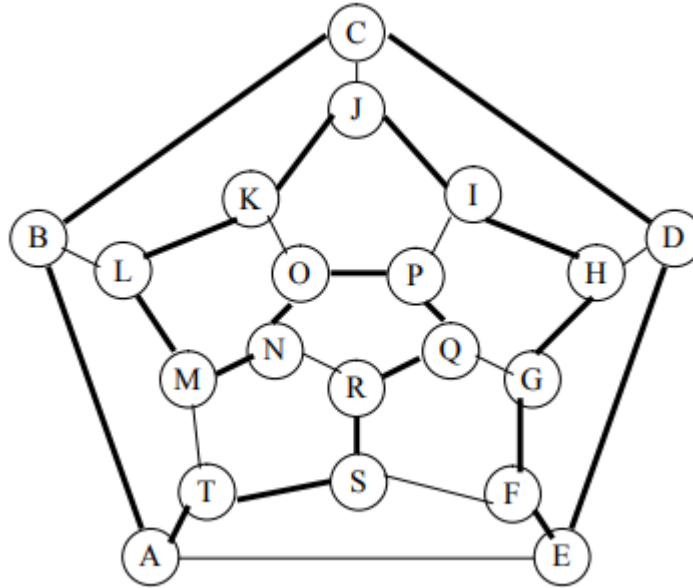
3.1.2. Mô tả thuật toán

Định nghĩa:

Chu trình (tương ứng đường đi) sơ cấp chứa tất cả các đỉnh của đồ thị (vô hướng hoặc có hướng) G được gọi là chu trình (tương ứng đường đi) Hamilton. Một đồ thị có

chứa một chu trình (tương ứng đường đi) Hamilton được gọi là đồ thị Hamilton (tương ứng nửa Hamilton).

Thí dụ 1: Đồ thị Hamilton (hình thập nhị diện đều biểu diễn trong mặt phẳng) với chu trình Hamilton A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, A (đường tô đậm).



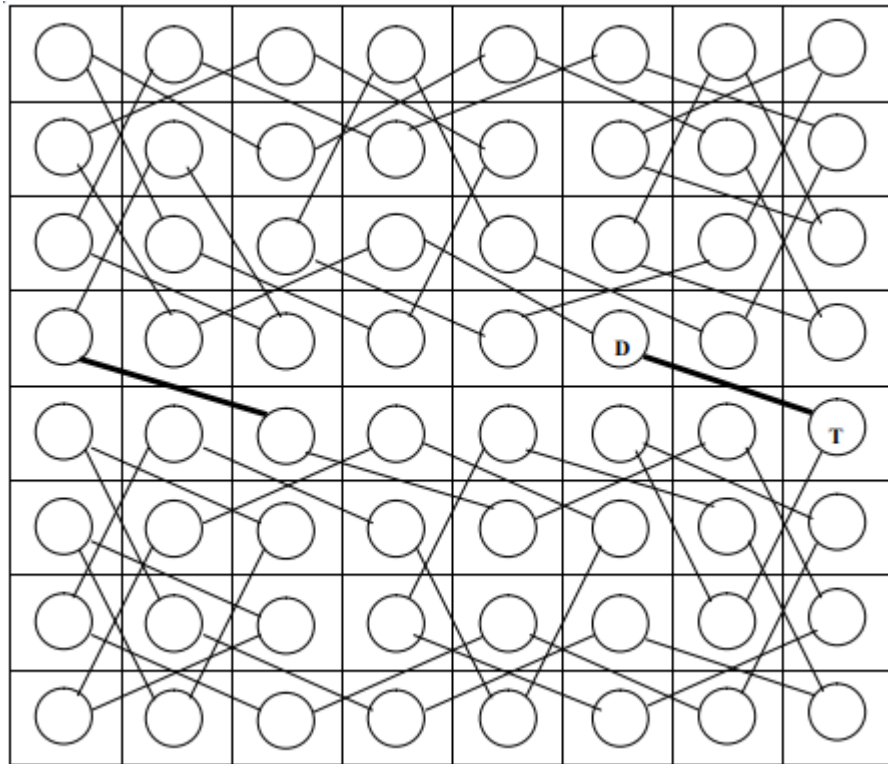
Hình 2.1: Đường đi Hamilton.

Thí dụ 2: Trong một đợt thi đấu bóng bàn có n ($n \geq 2$) đấu thủ tham gia. Mỗi đấu thủ gặp từng đấu thủ khác đúng một lần. Trong thi đấu bóng bàn chỉ có khả năng thắng hoặc thua.

Chứng minh rằng sau đợt thi đấu có thể xếp tất cả các đấu thủ đứng thành một hàng dọc, để người đứng sau thắng người đứng ngay trước anh (chị) ta.

Người đứng sau thắng người đứng ngay trước anh (chị) ta. Xét đồ thị có hướng G gồm n đỉnh sao cho mỗi đỉnh ứng với một đấu thủ và có một cung nối từ đỉnh u đến đỉnh v nếu đấu thủ ứng với u thắng đấu thủ ứng với v . Như vậy, đồ thị G có tính chất là với hai đỉnh phân biệt bất kỳ u và v , có một và chỉ một trong hai cung (u, v) hoặc (v, u) , đồ thị như thế được gọi là đồ thị có hướng đầy đủ. Từ Mệnh đề 4.2.2 dưới đây, G là một đồ thị nửa Hamilton. Khi đó đường đi Hamilton trong G cho ta sự sắp xếp cần tìm.

Thí dụ 3: Một lời giải về hành trình của con mã trên bàn cờ 8×8 :



Hình 2.2: Hành trình của con mã trên bàn cờ 8x8.

Đường đi Hamilton tương tự đường đi Euler trong cách phát biểu: Đường đi Euler qua mọi cạnh (cung) của đồ thị đúng một lần, đường đi Hamilton qua mọi đỉnh của đồ thị đúng một lần. Tuy nhiên, nếu như bài toán tìm đường đi Euler trong một đồ thị đã được giải quyết trọn vẹn, dấu hiệu nhận biết một đồ thị Euler là khá đơn giản và dễ sử dụng, thì các bài toán về tìm đường đi Hamilton và xác định đồ thị Hamilton lại khó hơn rất nhiều. Đường đi Hamilton và đồ thị Hamilton có nhiều ý nghĩa thực tiễn và đã được nghiên cứu nhiều, nhưng vẫn còn những khó khăn lớn chưa ai vượt qua được.

Người ta chỉ mới tìm được một vài điều kiện đủ để nhận biết một lớp rất nhỏ các đồ thị Hamilton và đồ thị nửa Hamilton. Sau đây là một vài kết quả.

Định lý (Rédei): Nếu G là một đồ thị có hướng đầy đủ thì G là đồ thị nửa Hamilton.

Chứng minh: Giả sử $G=(V, E)$ là đồ thị có hướng đầy đủ và $\alpha=(v_1, v_2, \dots, v_{k-1}, v_k)$ là đường đi sơ cấp bất kỳ trong đồ thị G .

- Nếu α đã đi qua tất cả các đỉnh của G thì nó là một đường đi Hamilton của G .
- Nếu trong G còn có đỉnh nằm ngoài α , thì ta có thể bổ sung dần các đỉnh này vào α và cuối cùng nhận được đường đi Hamilton.

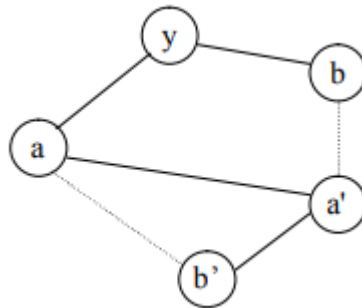
Thật vậy, giả sử v là đỉnh tùy ý không nằm trên α

- Nếu có cung nối v với v_1 thì bổ sung v vào đầu của đường đi α để được $\alpha_1 = (v, v_1, v_2, \dots, v_{k-1}, v_k)$.
- Nếu tồn tại chỉ số i ($1 \leq i \leq k-1$) mà từ v_i có cung nối tới v và từ v có cung nối tới v_{i+1} thì ta chen v vào giữa v_i và v_{i+1} để được đường đi sơ cấp $\alpha_2 = (v_1, v_2, \dots, v_i, v, v_{i+1}, \dots, v_k)$.
- Nếu cả hai khả năng trên đều không xảy ra nghĩa là với mọi i ($1 \leq i \leq k$) v_i đều có cung đi tới v . Khi đó bổ sung v vào cuối của đường đi α và được đường đi $\alpha_3 = (v_1, v_2, \dots, v_{k-1}, v_k, v)$.

Nếu đồ thị G có n đỉnh thì sau $n-k$ bổ sung ta sẽ nhận được đường đi Hamilton.

Định lý (Dirac, 1954): Nếu G là một đơn đồ thị có n đỉnh và mọi đỉnh của G đều có bậc không nhỏ hơn $2n/3$ thì G là một đồ thị Hamilton.

Chứng minh: Định lý được chứng minh bằng phản chứng. Giả sử G không có chu trình Hamilton. Ta thêm vào G một số đỉnh mới và nối mỗi đỉnh mới này với mọi đỉnh của G , ta được đồ thị G' . Giả sử k (>0) là số tối thiểu các đỉnh cần thiết để G' chứa một chu trình Hamilton. Như vậy, G' có $n+k$ đỉnh.



Hình 2.3: Đồ thị vô hướng $ayba'b'$

Gọi P là chu trình Hamilton $ayb \dots a$ trong G' , trong đó a và b là các đỉnh của G , còn y là một trong các đỉnh mới. Khi đó b không kề với a , vì nếu trái lại thì ta có thể bỏ đỉnh y và được chu trình $ab \dots a$, mâu thuẫn với giả thiết về tính chất nhỏ nhất của k .

Ngoài ra, nếu a' là một đỉnh kề nào đó của a (khác với y) và b' là đỉnh nối tiếp ngay a' trong chu trình P thì b' không thể là đỉnh kề với b , vì nếu trái lại thì ta có thể thay P bởi chu trình $aa' \dots bb' \dots a$, trong đó không có y , mâu thuẫn với giả thiết về tính chất nhỏ nhất của k .

Như vậy, với mỗi đỉnh kề với a , ta có một đỉnh không kề với b , tức là số đỉnh không kề với b không thể ít hơn số đỉnh kề với a (số đỉnh kề với a không nhỏ hơn $2n+k$).
 a b' a' b y 62 Mặt khác, theo giả thiết số đỉnh kề với b cũng không nhỏ hơn $2n+k$. Vì không có đỉnh nào vừa kề với b lại vừa không kề với b , nên số đỉnh của G' không ít hơn $2(2n+k) = n+2k$, mâu thuẫn với giả thiết là số đỉnh của G' bằng $n+k$ ($k>0$). Định lý được chứng minh.

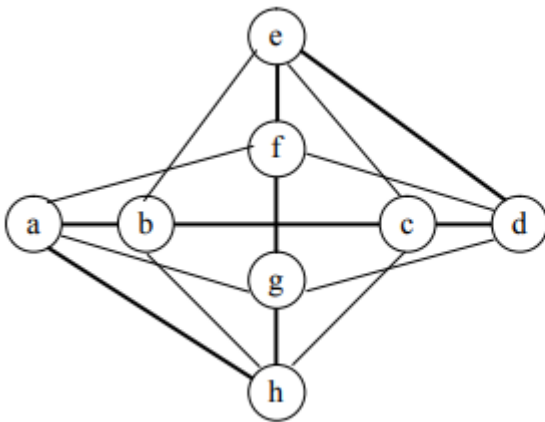
Hệ quả: Nếu G là đơn đồ thị có n đỉnh và mọi đỉnh của G đều có bậc không nhỏ hơn $2n-1$ thì G là đồ thị nửa Hamilton

Chứng minh: Thêm vào G một đỉnh x và nối x với mọi đỉnh của G thì ta nhận được đơn đồ thị G' có $n+1$ đỉnh và mỗi đỉnh có bậc không nhỏ hơn $2n+1$. Do đó theo Định lý 4.2.3, trong G' có một chu trình Hamilton. Bỏ x ra khỏi chu trình này, ta nhận được đường đi Hamilton trong G .

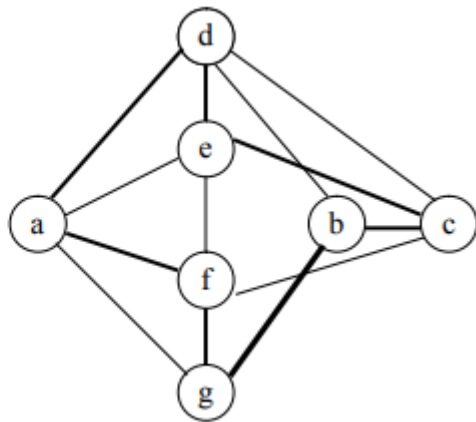
Định lý (Ore, 1960): Nếu G là một đơn đồ thị có n đỉnh và bất kỳ hai đỉnh nào không kề nhau cũng có tổng số bậc không nhỏ hơn n thì G là một đồ thị Hamilton.

Định lý: 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 $2n$ thì G là một đồ thị Hamilton.

Thí dụ 2:



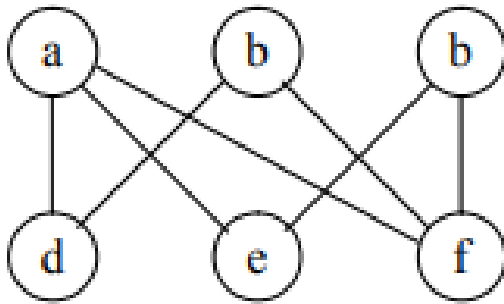
Hình 2.4: Đồ thị G .



Hình 2.5: Đồ thị G' .

Hình 2.4, đồ thị G có 8 đỉnh, đỉnh nào cũng có bậc 4, nên theo định lý Dirac, G là đồ thị Hamilton

Hình 2.5, đồ thị G' có 5 đỉnh bậc 4 và 2 đỉnh bậc 2 kề nhau nên tổng số bậc của đỉnh không kề nhau bất kỳ bằng 7 hoặc 8, nên theo định lý Ore, G' là đồ thị Hamilton.



Đồ thị phân đôi này có bậc của mỗi đỉnh bằng 2 hoặc 3 ($> 3/2$), nên theo Định lý 4.2.6, nó là đồ thị Hamilton.

Hình 2.6: Đồ thị vô hướng G' .

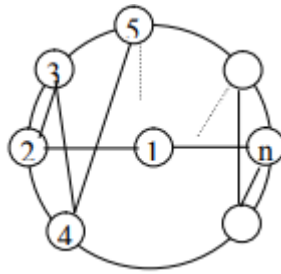
Bài toán sắp xếp chỗ ngồi:

Có n đại biểu từ n nước đến dự hội nghị quốc tế. Mỗi ngày họp một lần ngồi quanh một bàn tròn. Hỏi phải bố trí bao nhiêu ngày và bố trí như thế nào sao cho trong mỗi ngày, mỗi người có hai người kế bên là bạn mới. Lưu ý rằng n người đều muốn làm quen với nhau.

Xét đồ thị gồm n đỉnh, mỗi đỉnh ứng với mỗi người dự hội nghị, hai đỉnh kề nhau khi hai đại biểu tương ứng muốn làm quen với nhau. Như vậy, ta có đồ thị đầy đủ K_n . Đồ thị này là Hamilton và rõ ràng mỗi chu trình Hamilton là một cách sắp xếp như yêu cầu của bài toán. Bài toán trở thành tìm các chu trình Hamilton phân biệt của đồ thị đầy đủ K_n (hai chu trình Hamilton gọi là phân biệt nếu chúng không có cạnh chung).

Định lý: Đồ thị đầy đủ K_n với n lẻ và $n \geq 3$ có đúng $2n - 1$ chu trình Hamilton phân biệt.

Chứng minh: K_n có $2n - 1$ cạnh và mỗi chu trình Hamilton có n cạnh, nên số chu trình Hamilton phân biệt nhiều nhất là $2n - 1$.



Hình 2.7: Đồ thị đầy đủ K_n .

Giả sử các đỉnh của K_n là $1, 2, \dots, n$. Đặt đỉnh 1 tại tâm của một đường tròn và các đỉnh $2, \dots, n$ đặt cách đều nhau trên đường tròn (mỗi cung là $3600/(n-1)$) sao cho đỉnh lẻ nằm ở nửa đường tròn trên và đỉnh chẵn nằm ở nửa đường tròn dưới. Ta có ngay chu trình Hamilton đầu tiên là $1, 2, \dots, n, 1$. Các đỉnh được giữ cố định, xoay khung theo chiều kim đồng hồ với các góc quay:

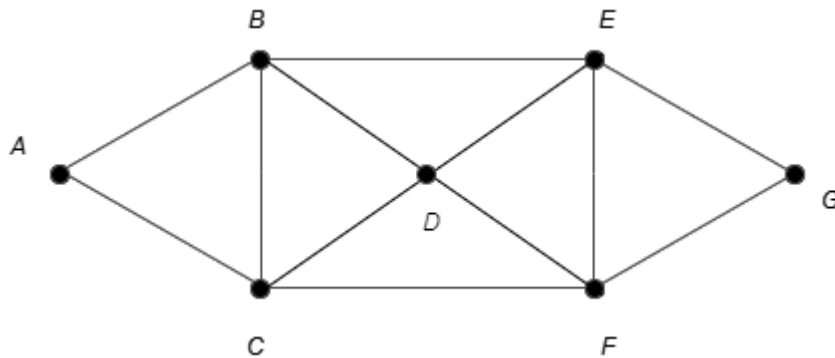
$$\frac{360^0}{n-1}, 2 \cdot \frac{360^0}{n-1}, 3 \cdot \frac{360^0}{n-1}, \dots, \frac{n-3}{2} \cdot \frac{360^0}{n-1}$$

ta nhận được $2n - 3$ khung phân biệt với khung đầu tiên. Do đó ta có $2n - 1$ chu trình Hamilton phân biệt.

3.1.3. Quy tắc tìm chu trình Hamilton

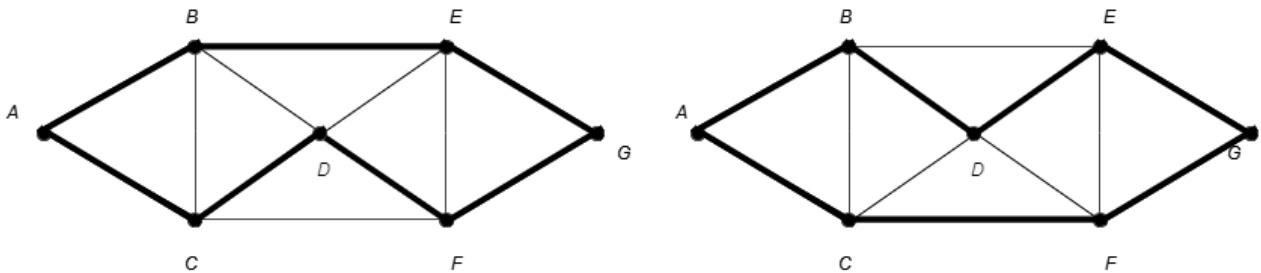
- Lấy hết cả cạnh kề với đỉnh bậc 2.
- Không cho phát sinh chu trình ít hơn n cạnh.
- Nếu đã lấy 2 cạnh kề với đỉnh x thì có thể bỏ tất cả các cạnh còn lại kề với x .
- Duy trì tính liên thông và đảm bảo bậc mỗi đỉnh luôn lớn hơn hay bằng 2.

3.1.4. Ví dụ về thuật toán



Hình 2.8: Đồ thị H .

- Gọi H là chu trình Hamilton cần tìm.
- Do $\deg(A) = \deg(G) = 2$, nên cạnh AB, AC, GE, GF chắc chắn thuộc chu trình Hamilton H .
- Cạnh BC và EF không thuộc chu trình Hamilton H , vì nếu thuộc chu trình Hamilton H sẽ tạo chu trình Hamilton con là ABC và EFG .
- Giả sử BE (hoặc CF) thuộc H chu trình Hamilton H , thì $\deg(B) = \deg(E) = 2$, nên ta sẽ xóa các cạnh còn lại tiến tới B và E (hoặc C và F).
- Lúc này, $\deg(D) = 2$ nên cạnh DC và DF (hoặc DB hoặc DE) chắc chắn thuộc chu trình Hamilton H .
- Lúc này $\deg(C) = \deg(F) = 2$ (hoặc $\deg(B) = \deg(E) = 2$), nên ta xóa các cạnh còn lại tiến tới C và F (hoặc B và E).
- Ta được các chu trình Hamilton H cần tìm là: $ABEGFDC, ABDEGFCA$



Hình 2.9: Chu trình Hamilton H tìm được.

3.2. Kết luận

Hầu hết các bài toán tối ưu liên quan đến đường đi Hamilton hay chu trình Hamilton trong đồ thị là những vấn đề phức tạp. Phương pháp tiếp cận để giải quyết các bài toán được tối ưu lần vấn đề cần thiết.

CHƯƠNG 4. THỬ NGHIỆM VÀ ĐÁNH GIÁ

4.1. Giới thiệu bài toán

Một người xuất phát từ một thành phố nào đó và muốn tới thăm n thành phố khác, mỗi thành phố đúng một lần, rồi quay về thành phố ban đầu. Hỏi nên đi theo trình tự nào để độ dài tổng cộng các đoạn đường đi qua là ngắn nhất (khoảng cách giữa các thành hai thành phố có thể hiểu là cự ly thông thường hoặc thời gian cần đi hoặc chi phí của hành trình v.v... và xem như cho trước).

Đầu vào bài toán: Tập dữ liệu gồm:

- Điểm xuất phát (tương ứng với dữ liệu đầu vào Google Maps API: địa chỉ, latitude-longitude, place_id)
- Danh sách các điểm đến (tương ứng với dữ liệu đầu vào Google Maps API: địa chỉ, latitude-longitude, place_id)

Đầu ra bài toán:

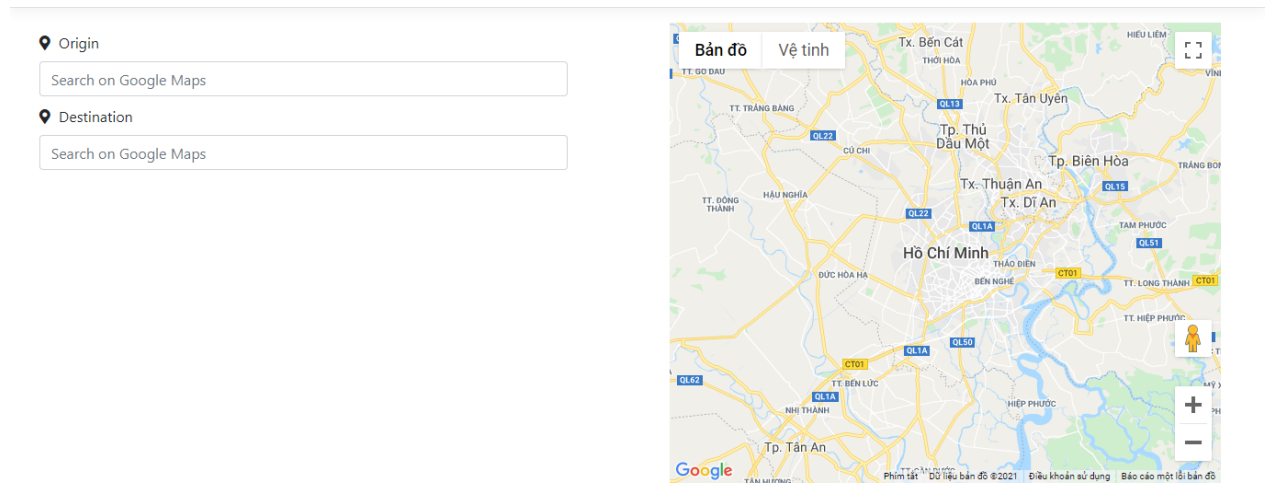
- Danh sách các điểm người xuất phát có thể đi qua phù hợp với yêu cầu chỉ thăm mỗi thành phố đúng một lần.
- Khoảng cách giữa các thành phố (tương ứng với dữ liệu được trả về từ Google Maps API).
- Mô tả đường đi cơ bản mà người xuất phát có thể đi qua (tương ứng với dữ liệu được trả về từ Google Maps API).

4.2. Môi trường thử nghiệm

- Công nghệ: .NET Core 5.x, Angular 12.x.
- Ngôn ngữ lập trình: C#.
- Công cụ: Visual Studio Code

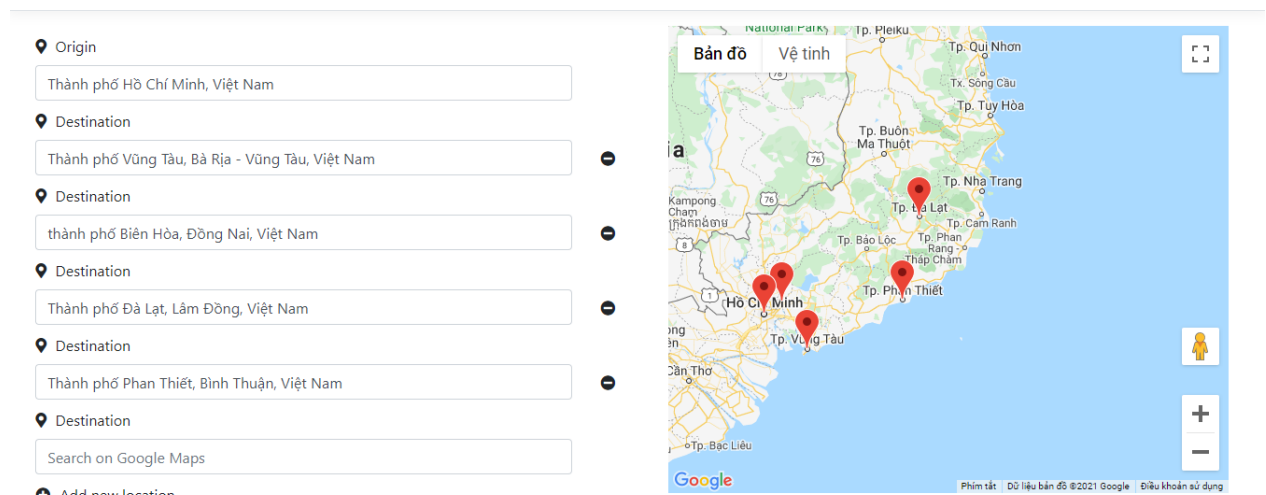
4.3. Giao diện trang web thử nghiệm

Trang web gồm các chức năng sau:



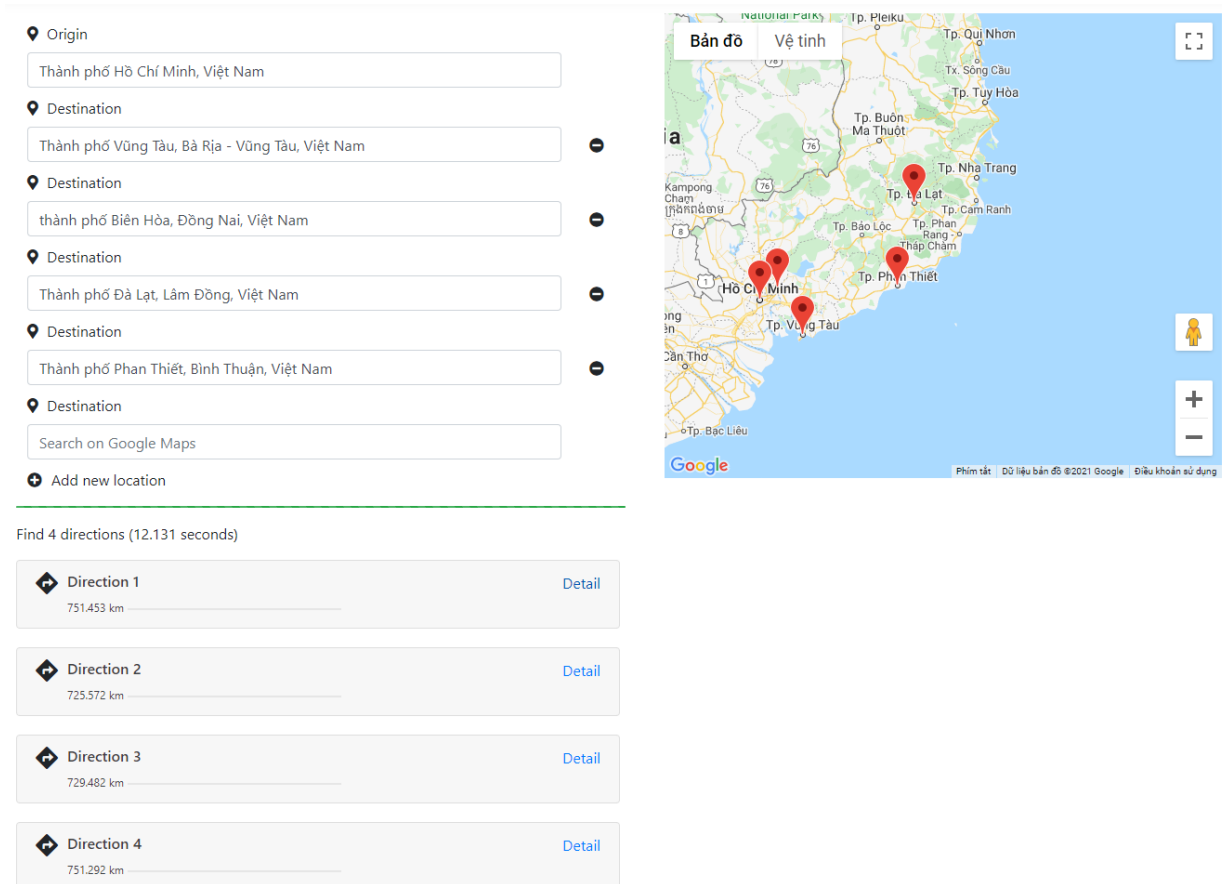
Hình 3.1: Giao diện trang web thử nghiệm .

Người dùng cần điền thông tin địa điểm xuất phát và danh sách các địa điểm cần đến (thông tin có thể là địa chỉ, latitude-longitude, place_id) với sự gợi ý từ dữ liệu Google Maps API. Hệ thống sẽ tự động đánh dấu các địa điểm mà người dùng nhập vào lên Google Maps.



Hình 3.2: Hệ thống tự động đánh dấu các địa điểm mà người dùng nhập vào lên Google Maps.

Sau khi nhập xong các địa điểm, hệ thống tự động tính toán, tìm kiếm các lộ trình (chu trình/ đường đi) với dữ liệu được lấy từ Google Maps API và xử lý bằng thuật toán xác định đường đi Hamilton. Kết quả được hiển thị như hình dưới:




Hình 3.3: Các kết quả sau khi hệ thống xử lý xong được thể hiện.

Tốc độ xử lý của hệ thống sẽ chịu ảnh hưởng từ một số yếu tố:


- Số lượng địa điểm người dùng nhập vào, tốc độ xử lý tỷ lệ thuận với số lượng địa điểm được nhập
- Tốc độ internet. Vì hệ thống sử dụng dữ liệu từ Google Maps API nên tốc độ mạng quyết định đến tốc độ xử lý của hệ thống.


Người dùng chọn ‘Detail’ chi tiết lộ trình, trang web sẽ hiển thị lộ trình chi tiết như hình dưới (hình 3.3). Các địa điểm thuộc điểm đến sẽ được xuất hiện duy nhất một lần (phù hợp với yêu cầu bài toán ‘Mỗi thành phố chỉ được thăm một lần’) và địa điểm xuất phát sẽ xuất hiện ở bắt đầu và kết thúc (phù hợp với yêu cầu của bài toán: ‘Quay về địa điểm ban đầu’).


Find 4 directions (12.131 seconds)



Direction 1
[Detail](#)


751.453 km


Thành phố Hồ Chí Minh, Việt Nam | Thành phố Vũng Tàu, Bà Rịa - [Detail](#)
 Vũng Tàu, Việt Nam
 QL51
 95.972 km


Thành phố Vũng Tàu, Bà Rịa - Vũng Tàu, Việt Nam | Thành phố Đà [Detail](#)
 Lạt, Lâm Đồng, Việt Nam
 QL20
 282.955 km


Thành phố Đà Lạt, Lâm Đồng, Việt Nam | Thành phố Phan Thiết, [Detail](#)
 Bình Thuận, Việt Nam
 Lương Sơn - Đại Ninh/QL28B
 159.546 km

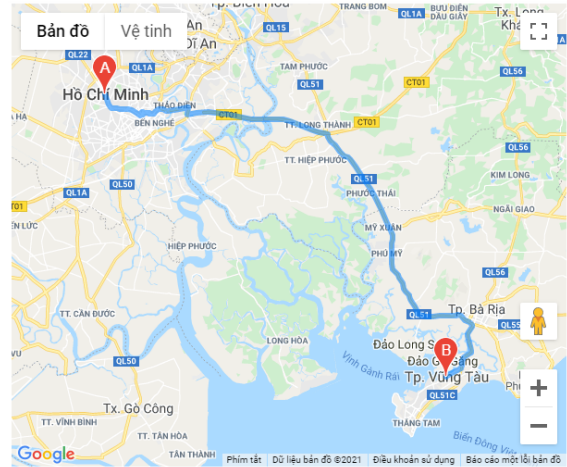
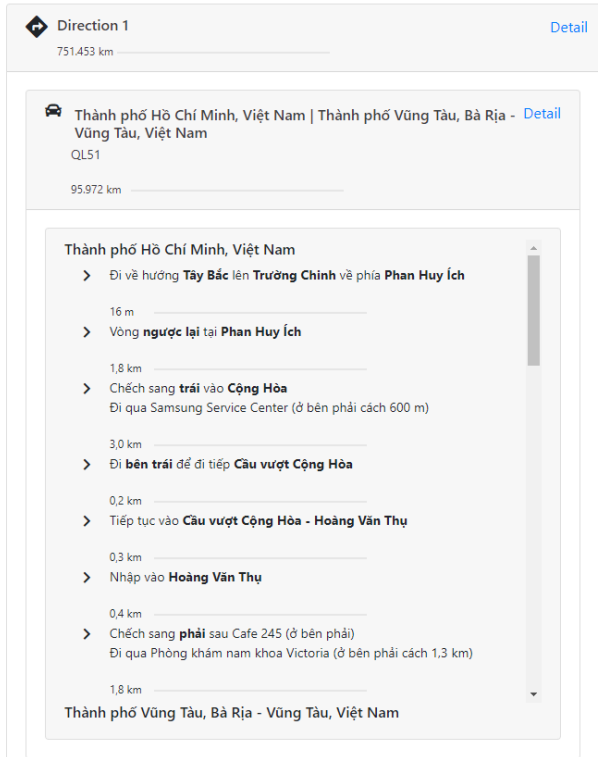

Thành phố Phan Thiết, Bình Thuận, Việt Nam | thành phố Biên [Detail](#)
 Hòa, Đồng Nai, Việt Nam
 QL1A
 177.29 km


thành phố Biên Hòa, Đồng Nai, Việt Nam | Thành phố Hồ Chí [Detail](#)
 Minh, Việt Nam
 Quốc lộ 1K và Xa lộ Đại Hàn/QL1A
 35.69 km

Hình 3.4: Chi tiết lộ trình đường đi 1.

Người dùng chọn ‘Detail’ chi tiết lộ trình đường đi giữa các điểm, trang web sẽ hiển thị đường đi giữa hai điểm đó (Ví dụ chi tiết đường đi từ ‘Thành phố Hồ Chí Minh, Việt Nam’ đến ‘Thành phố Vũng Tàu, Bà Rịa – Vũng Tàu, Việt Nam’ như hình dưới) với chi tiết đường đi tương ứng với dữ liệu đường đi của Google Maps API – Directions API. Tại bản đồ Google Maps sẽ hiển thị đường đi trực quan lên bản đồ (Với điểm xuất phát đánh dấu là ‘A’ và điểm đến đánh dấu là ‘B’).

Find 4 directions (12,131 seconds)



Hình 3.5: Chi tiết được đi từ điểm ‘Thành phố Hồ Chí Minh, Việt Nam’ đến ‘Thành phố Vũng Tàu, Bà Rịa – Vũng Tàu, Việt Nam’.

4.4. Kết quả và đánh giá

Với trang web xác định các lộ trình đường đi dựa trên dữ liệu Google Maps API và thuật toán xác định đường đi Hamilton, bước đầu sẽ giúp ích cho các doanh nghiệp vận tải trong việc xác định lộ trình đường đi tối ưu cho việc kinh doanh của chính mình, từ đó tối ưu được chi phí vận chuyển và chi phí đưa hàng hóa đến người tiêu dùng.

Bên cạnh đó trang web xác định lộ trình đường đi còn có một số vấn đề chưa được tối ưu:

- Chưa xử lý được vấn đề khoảng cách giữa các điểm gần nhau cho tối ưu với thuật toán. Không thể xác định lộ trình đường đi chính xác với những địa điểm này.
- Việc phân tích, tính toán được thực hiện với dữ liệu được lấy từ bên thứ 3 (cụ thể là Google Maps API) nên tốc độ xử lý của hệ thống không thể can thiệp được. Tốc độ xử lý của hệ thống sẽ tỉ lệ thuận với số lượng địa điểm người dùng nhập vào.
- Giao diện trang web chưa có tính thẩm mỹ cao.

CHƯƠNG 5. KẾT LUẬN VÀ KIẾN NGHỊ

5.1. Kết quả đạt được

Trong suốt thời gian nghiên cứu và thực hiện đề tài ‘Tìm hiểu và ứng dụng thuật toán xác định đường đi Hamilton’ em đã cố gắng làm hết khả năng của mình, bản thân đã lĩnh hội được rất nhiều tri thức và nắm bắt các kiến thức chuyên môn cũng như một số công nghệ mới, cụ thể như sau:

Về mặt khoa học:

- Nắm bắt được các kiến thức cơ bản về lý thuyết đồ thị và thuật toán xác định đường đi, cụ thể là thuật toán xác định đường đi Hamilton.
- Hiểu rõ các quy trình và phương pháp ứng dụng thuật toán xác định đường đi Hamilton để áp dụng vào bài toán thực tế.
- Nắm bắt được cách tích hợp các công nghệ của Google vào đề tài. Cụ thể là Google Maps API, Google Compute Engine v,v...

Về mặt ứng dụng:

Từ những kết quả về mặt kiến thức đã đạt được ở trên, em đã xây dựng thành công trang web ứng dụng thuật toán xác định đường đi Hamilton. Trang web có chức năng: tìm kiếm địa chỉ, đánh dấu địa chỉ trên bản đồ, tìm kiếm các lộ trình đường đi giữa các địa chỉ với dữ liệu từ Google Maps APIs. Nhằm giúp các doanh nghiệp vận tải có thể tối ưu được lộ trình đường đi vận chuyển hàng hóa nhằm tối ưu chi phí.

Về mặt con người:

Qua quá trình làm đồ án, mặc dù thời gian thực hiện không quá nhiều nhưng em đã cố gắng học hỏi, rèn luyện thêm cho bản thân một số kỹ năng về tìm kiếm, nghiên cứu tài liệu, phân tích bài toán, cách nhìn nhận và xử lý vấn đề, làm việc nhóm, lập trình, rèn luyện tính kiên nhẫn và cách trình bày văn bản hợp lý hơn v,v...rất hữu ích cho bản thân em trong công việc và cuộc sống hằng ngày.

Như vậy, em đã hoàn thành cơ bản những mục tiêu được đặt ra ban đầu với đề tài nghiên cứu này.

Tồn tại:

Bên cạnh những khía cạnh đạt được, trong quá trình làm đồ án, mặc dù đã rất cố gắng nhưng vẫn còn một số vấn đề chưa được hoàn thiện như:

- Xử lý dữ liệu từ bên thứ ba còn hạn chế.
- Giao diện chưa tối ưu về mặt thẩm mỹ.
- Tốc độ hệ thống cần được cải thiện.

5.2. Hướng phát triển

Trong quá trình làm đồ án, bản thân cũng đã tự học hỏi, tìm hiểu trau dồi kiến thức từ việc tra cứu đọc tài liệu, cũng như các kiến thức học tập được từ người thầy của mình. Mong muốn trong tương lai sắp tới, đồ án của em được mở rộng theo các hướng như sau:

- Tiếp tục nghiên cứu và phát triển thuật toán để hoàn thiện hệ thống này, và tích hợp vào hệ thống vào hệ thống quản lý doanh nghiệp giao thông vận tải với đầy đủ các chức năng nhằm giải quyết vấn đề tối ưu quãng đường di chuyển vận chuyển hàng hóa.
- Cần phải tối ưu về thuật toán, tích hợp API từ Google để giải quyết vấn đề tốc độ xử lý kết quả.
- Về mặt thẩm mỹ, hiện tại em đang sử dụng Bootstrap 4.x, mà giao diện chưa được đẹp và chưa tối ưu hiển thị cho mọi thiết bị. Vì thế, có thể sử dụng Bootstrap 5.x hoặc một số bộ Framework Front End giúp chúng ta phát triển giao diện đẹp hơn.

TÀI LIỆU THAM KHẢO

- [1]. Thạc sĩ Nguyễn Thanh Hùng, *Giáo trình Lý Thuyết Đồ Thị*.
- [2]. Lê Minh Hoàng, *Giáo trình Lý thuyết đồ thị*.
- [3]. Lê Minh Hoàng, Bài giảng Giải Thuật và Lập Trình.
- [4]. <https://dotnet.microsoft.com/platform/support/policy/dotnet-core>, “.NET Core and .NET 5 Support Policy”
- [5]. <https://tedu.com.vn/lap-trinh-aspnet-core/gioi-thieu-ve-aspnet-core-203.html>, “Giới thiệu về ASP.NET Core”
- [6]. <https://www.tutorialsteacher.com/core/dotnet-core>, “.NET Core Overview”
- [7]. <https://docs.microsoft.com/en-us/dotnet/core/introduction>, “Introduction to .NET”
- [8]. <https://devblogs.microsoft.com/premier-developer/net-core-overview>, “.NET Core Overview”
- [9]. <https://tedu.com.vn/lap-trinh-angular-2-can-ban/gioi-thieu-ve-260.html>, “Giới thiệu về Angular”
- [10]. <https://viblo.asia/p/gioi-thieu-tong-quan-ve-angular-07LKX9j2ZV4>, “Giới thiệu tổng quan về Angular”
- [11]. <https://blog.tinohost.com/angular-la-gi>, “Angular là gì? Giới thiệu toàn tập Angular”
- [12]. https://vi.wikipedia.org/wiki/Google_Maps, “Google Maps”
- [13]. <https://developers.google.com/maps/documentation>, “Google Maps Platform Documentation”
- [14]. William L. Hamilton, *Graph Representation Learning (2020)*, McGill University
- [15]. <https://github.com/angular/components/tree/master/src/google-maps#readme>, “Angular Google Maps Component”
- [16]. <https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-ubuntu-20-04>, “How To Install and Use Docker on Ubuntu 20.04”
- [17]. <https://expressmagazine.net/development/4013/thuat-toan-ve-tim-duong-di-va-chu-trinh-hamilton-cai-dat-bang-c-c++>, “Thuật toán về tìm đường đi và chu trình Hamilton cài đặt bằng C/C++”