Toán rời rạc và thuật toán
Đại học Khoa học Tự nhiên
Khoa Toán - Cơ - Tin học
Khoa học dữ liệu K4
Tháng 10 năm 2022

Bài tập số 2 - Lý thuyết đồ thị

Nguyễn Mạnh Linh, Nguyễn Thị Đông, Triệu Hồng Thúy

1 Bài 1

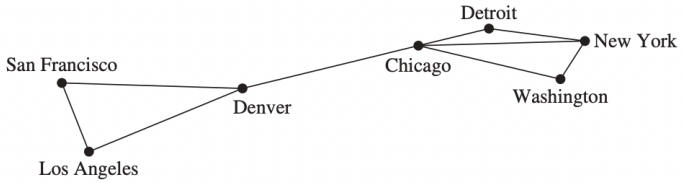
1.1 Những khái niệm cơ bản về đồ thị

1.1.1 Khái niệm đồ thị

Một đồ thị là một cấu trúc rời rạc gồm tập hợp các đỉnh và các cạnh nối giữa các đỉnh đó. Có thể mô tả đồ thị theo cách hình thức như sau:

$$\mathbf{G = (V,E)}$$

tức là đồ thị **G** có tập các đỉnh là **V**, tập các cạnh là **E**. Có thể hiểu **E** là tập hợp các cặp (u, v) với u và v là hai đỉnh thuộc V.
Tập các đỉnh V của đồ thị G có thể là vô hạn. Một đồ thị có tập đỉnh vô hạn hoặc vô số cạnh được gọi là đồ thị vô hạn, và khi so sánh, đồ thị có tập đỉnh hữu hạn và tập hợp cạnh hữu hạn được gọi là đồ thị hữu hạn

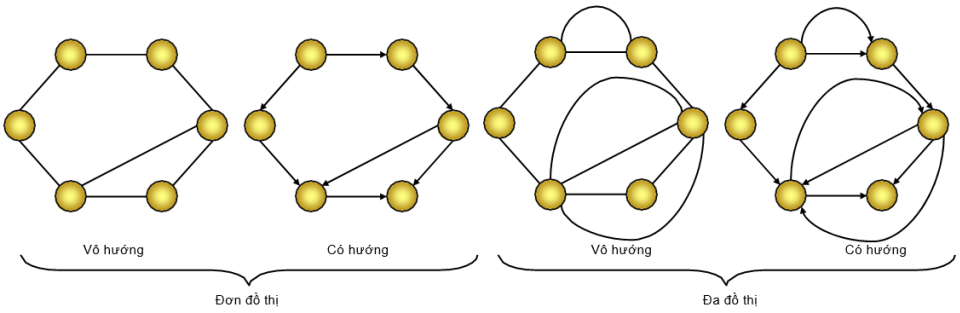


Hình 1: Mạng máy tính

Bây giờ, giả sử rằng một mạng được tạo thành từ các trung tâm dữ liệu và các liên kết giao tiếp giữa các máy tính. Chúng ta có thể biểu diễn vị trí của mỗi trung tâm dữ liệu bằng một điểm và mỗi liên kết truyền thông liên kết bằng một đoạn thẳng, như thể hiện trong Hình 1.

1.1.2 Phân loại đồ thị

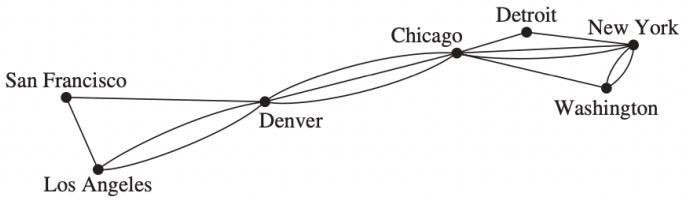
Có thể phân loại đồ thị G theo tính chất các tập cạnh như sau:



Hình 2: Phân loại đồ thị

1. **Đơn đồ thị**
   G được gọi là đơn đồ thị nếu như giữa hai đỉnh (u, v) của V có nhiều nhất một cạnh trong E nối từ u tới v.
   Như Hình 1 là một đơn đồ thị. Mạng máy tính này có thể được mô hình hóa bằng cách sử dụng một đồ thị trong đó các đỉnh của đồ thị đại diện cho các trung tâm dữ liệu và các cạnh đại diện cho các liên kết truyền thông. Lưu ý rằng mỗi cạnh của đồ thị đại diện cho mạng máy tính này kết nối hai đỉnh khác nhau. Có nghĩa là, không có cạnh nào kết nối một đỉnh với chính nó. Hơn nữa, không có hai cạnh khác nhau nối cùng một cặp đỉnh.

2. **Đa đồ thị**
   G được gọi là đa đồ thị nếu như giữa hai đỉnh (u, v) của V có thể có nhiều hơn 1 cạnh nối trong E nối từ u tới v. Hiển nhiên đơn đồ thị cũng là đa đồ thị.



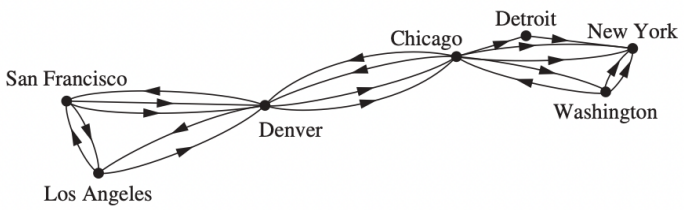
Hình 3: Đa đồ thị

Một mạng máy tính có thể chứa nhiều liên kết giữa các trung tâm dữ liệu, như trong Hình 3. Để mô hình hóa các mạng như vậy, chúng ta cần các đồ thị có nhiều hơn một cạnh nối cùng một cặp đỉnh. Các đồ thị có thể có nhiều cạnh nối các đỉnh giống nhau được gọi là đồ thị đa phương. Khi có m cạnh khác nhau liên kết với cùng một cặp đỉnh không có thứ tự u, v, ta cũng nói rằng u, v là một cạnh bội số m. Tức là, chúng ta có thể coi tập hợp các cạnh này là m bản sao khác nhau của một cạnh u, v.

3. **Đồ thị vô hướng**
   G được gọi là đồ thị vô hướng (undirected graph) nếu như các cạnh trong E là không định hướng, tức là cạnh (u, v) là cạnh hai chiều.

4. **Đồ thị có hướng**
   G được gọi là đồ thị có hướng (directed graph) nếu như các cạnh trong E là có định

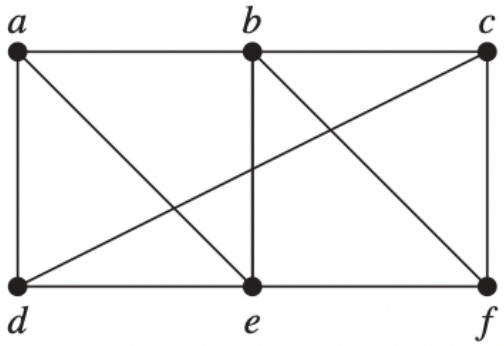
hướng.
Khi chúng ta mô tả một đồ thị có hướng bằng một bản vẽ đường thẳng, chúng ta sử dụng một mũi tên chỉ từ u đến v để chỉ ra hướng của một cạnh bắt đầu tại u và kết thúc tại v. Một biểu đồ có hướng có thể chứa các vòng lặp và nó có thể chứa nhiều cạnh có hướng bắt đầu và kết thúc ở cùng một đỉnh. Một đồ thị có hướng cũng có thể chứa các cạnh có hướng nối các đỉnh u và v theo cả hai hướng; nghĩa là, khi một đồ thị chứa một cạnh từ u đến v, nó cũng có thể chứa một hoặc nhiều cạnh từ v đến u.
Khi một đồ thị có hướng không có vòng lặp và không có nhiều cạnh có hướng, nó được gọi là **đồ thị có hướng đơn giản**. Bởi vì một đồ thị có hướng đơn giản có nhiều nhất một cạnh liên kết với mỗi cặp đỉnh có thứ tự (u, v), chúng ta gọi (u, v) là một cạnh nếu có một cạnh liên kết với nó trong đồ thị.
Các đồ thị có hướng có thể có nhiều cạnh có hướng từ một đỉnh đến một đỉnh thứ hai (có thể giống nhau) được sử dụng để mô hình hóa các mạng như vậy. Những đồ thị như vậy là **đồ thị đa hướng**. Khi có m cạnh có hướng, mỗi cạnh liên kết với một cặp đỉnh có thứ tự (u, v), chúng ta nói rằng (u, v) là một cạnh của m.



Hình 4: Đồ thị đa hướng

1.1.3 Đường đi và chu trình

Gọi n là một số nguyên không âm và G là một đồ thị vô hướng. Đường đi có độ dài n từ u đến v trong G là dãy gồm n cạnh $e_1, ..., e_n$ của G mà trong đó tồn tại dãy $x_0 = u, x_1, ..., x_{n-1}, x_n = v$ thuộc các đỉnh sao cho e_i, với i = 1, ..., n, các điểm cuối x_{i-1} và x_i. Khi đồ thị đơn giản, ta biểu thị đường đi này bằng dãy đỉnh $x_0, x_1, ..., x_n$ (vì liệt kê các đỉnh này xác định duy nhất đường đi).
Đường đi là **một chu trình** nếu nó bắt đầu và kết thúc tại cùng một đỉnh, nghĩa là, nếu u = v, và có độ dài lớn hơn 0. Đường đi hoặc chu trình được cho là đi qua các đỉnh $x_1, x_2, ..., x_{n-1}$ hoặc đi qua các cạnh $e_1, ..., e_n$. Một đường đi hoặc chu trình sẽ đơn giản nếu nó không chứa cùng một cạnh nhiều hơn một lần.
Đường đi được gọi là đường đi đơn giản (simple path) nếu tất cả các đỉnh trên đường đi đó đều phân biệt. Đường đi được gọi là đường đi đơn nếu như không có cạnh nào trên đường đi đó đi qua hơn một lần.
Chu trình gọi là chu trình đơn giản (simple circuit) nếu như $x_1, x_2, ..., x_k$v đôi một khác nhau. Chu trình mà trong đó không có cạnh nào đi qua hơn một lần được gọi là chu trình đơn.
Ví dụ:



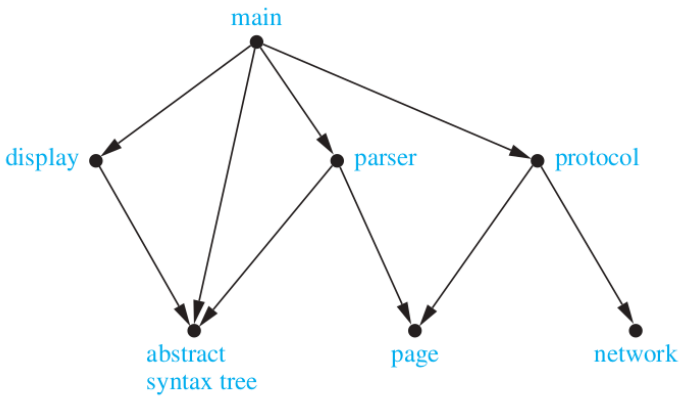
Hình 5: Đơn đồ thị

Trong đồ thị trên, a, d, c, f, e là một đường đi đơn giản có độ dài 4, bởi vì a, d, d, c, c, f và f, e là tất cả các cạnh. Tuy nhiên, d, e, c, a không phải là đường đi, vì e, c không phải là cạnh. Lưu ý rằng b, c, f, e, b là một chu trình có độ dài 4 vì b, c, c, f, f, e và e, b là các cạnh và đường đi này bắt đầu và kết thúc ở b. Đường đi a, b, e, d, a, b có độ dài 5 không phải là đường đi đơn vì nó chứa cạnh a, b hai lần.

1.2 Cấu trúc dữ liệu biểu diễn đồ thị

1.2.1 Thiết kế phần mềm

Đồ thị các thành phần phụ thuộc (Module Dependency Graphs)
Một trong những nhiệm vụ quan trọng trong thiết kế phần mềm đó là chia chương trình thành nhiều thành phần hoặc module khác nhau để tiện cho việc phát triển mà mở rộng cũng như bảo trì sau này. Việc hiểu được sự tương tác giữa các modules trong một chương trình tương tác với nhau như thế nào là cực kì quan trọng không chỉ trong việc thiết kế chương trình mà còn trong việc kiểm thử và bảo trì nữa. Một đồ thị các thành phần phụ thuộc giúp ích rất nhiều trong việc này. Trong đồ thị các thành phần phụ thuộc (dependencies), mỗi đỉnh biểu thị một module, một cạnh nối có hướng chỉ sự phụ thuộc của module này vào module kia. Một ví dụ về đồ thị biểu diễn sự phụ thuộc của các modules trong một ứng dụng web:



Hình 6: Ví dụ đồ thị sự phụ thuộc của các modules trong một ứng dụng web

1.2.2 Mạng giao thông

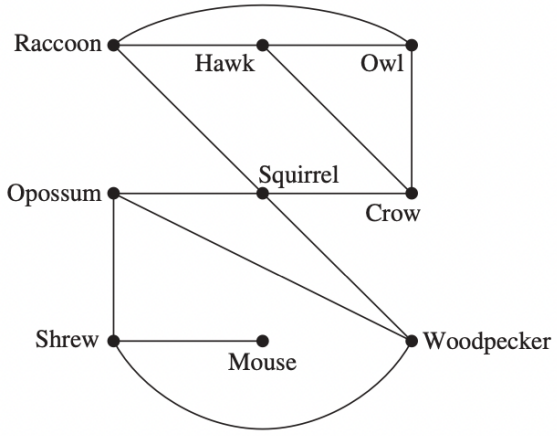
Mô hình đồ thị được sử dụng trong nhiều loại mạng giao thông khác nhau như đường bộ, hàng không, mạng đường sắt và mạng chuyển phát.

Định tuyến trong hàng không
Mô hình mạng hàng không có thể được biểu diễn bằng đồ thị với mỗi sân bay là một đỉnh. Các chuyến bay từ sân bay này tới sân bay khác có thể được biểu diễn bằng một cạnh có hướng từ sân bay cất cánh (đỉnh bắt đầu) đến sân bay hạ cánh (đỉnh kết thúc).

Mạng đường bộ
Mô hình đồ thị có thế được sử dụng để biểu diễn mạng đường bộ mà trong đó các đỉnh thể hiện các giao lộ và các cạnh thể hiện đường đi. Khi tất cả các cong đường trong mạng đều là đường 2 chiều thì ta có thể biểu diễn mạng bằng một đơn đồ thị vô hướng. Tuy nhiên trong thực tế ta thường gặp trong mạng giao thông thì một số đường là 2 chiều và một số khác là 1 chiều. Để biểu diễn mạng này ta dùng cạnh vô hướng để biểu diễn đường 2 chiều và cạnh có hướng để biểu diễn đường một chiều.

1.2.3 Mạng sinh học

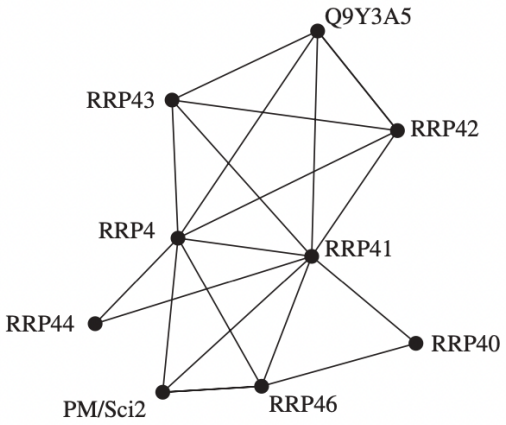
Nhiều khía cạnh của khoa học sinh học có thể được mô hình hóa bằng cách sử dụng đồ thị.
Đồ thị chồng chéo ngách trong hệ sinh thái
Đồ thị được sử dụng trong nhiều mô hình liên quan đến sự tương tác của các loài động vật khác nhau.
Ví dụ, sự cạnh tranh giữa các loài trong hệ sinh thái có thể được mô hình hóa bằng cách sử dụng đồ thị chồng chéo thích hợp. Mỗi loài được biểu diễn bằng một đỉnh. Một cạnh vô hướng nối hai đỉnh nếu hai loài đại diện bởi các đỉnh này cạnh tranh (nghĩa là một số nguồn thức ăn mà chúng sử dụng là giống nhau). Đồ thị chồng chéo ngách là một đồ thị đơn giản vì không cần vòng lặp hoặc nhiều cạnh trong mô hình này. Biểu đồ trong Hình 11 mô hình hệ sinh thái của một khu rừng. Từ biểu đồ này, chúng ta thấy rằng sóc và gấu trúc cạnh tranh nhưng quạ và chuột thì không.



Hình 7: Đồ thị chồng chéo ngách

Đồ thị tương tác protein

Tương tác protein trong tế bào sống xảy ra khi hai hoặc nhiều protein trong tế bào đó liên kết với nhau để thực hiện một chức năng sinh học. Bởi vì các tương tác protein là quan trọng đối với hầu hết các chức năng sinh học, nhiều nhà khoa học đang nghiên cứu để phát hiện ra các protein mới và các tương tác cơ bản giữa các protein. Tương tác protein trong tế bào có thể được mô hình hóa bằng cách sử dụng đồ thị tương tác protein, một đồ thị vô hướng trong đó mỗi protein được biểu diễn bằng một đỉnh, với một cạnh nối các đỉnh biểu thị từng cặp protein tương tác với nhau. Việc xác định tương tác protein thực sự trong tế bào là một vấn đề khó khăn, vì các thí nghiệm thường tạo ra kết quả dương tính giả, kết luận rằng hai protein tương tác khi chúng thực sự không tương tác. Đồ thị tương tác protein có thể được sử dụng để suy ra thông tin sinh học quan trọng, chẳng hạn như bằng cách xác định các protein quan trọng nhất cho các chức năng khác nhau và chức năng của các protein mới được phát hiện.



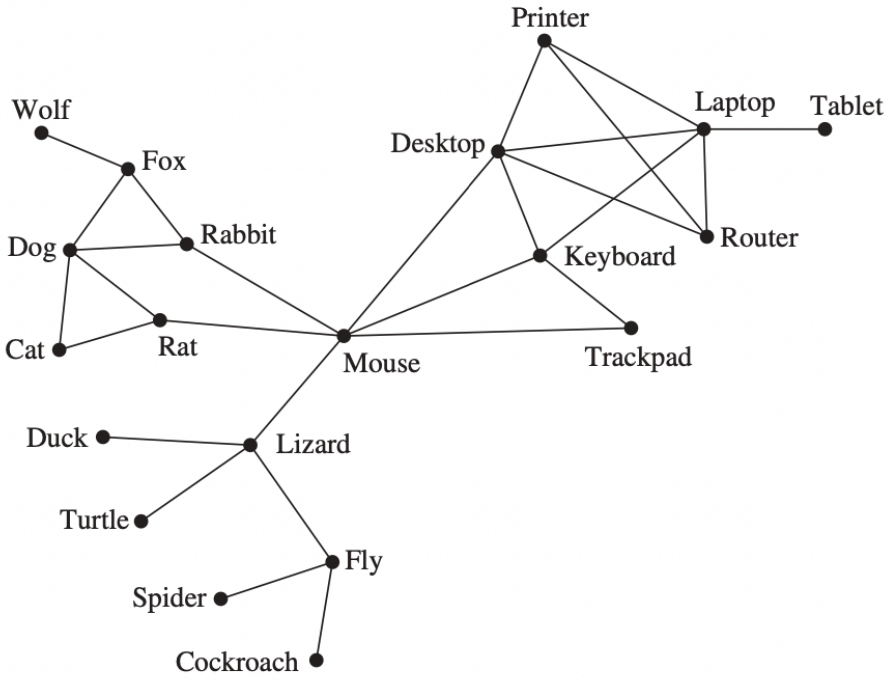
Hình 8: Đồ thị tương tác protein

Bởi vì có hàng ngàn loại protein khác nhau trong một tế bào điển hình, nên đồ thị tương tác protein của một tế bào là cực kỳ lớn và phức tạp. Ví dụ, tế bào nấm men có hơn 6.000 protein, và hơn 80.000 tương tác giữa chúng đã được biết đến, và tế bào người có hơn 100.000 protein, có lẽ khoảng 1.000.000 tương tác giữa chúng. Các đỉnh và cạnh bổ sung được thêm vào đồ thị tương tác protein khi các protein mới và tương tác giữa các protein được phát hiện. Do sự phức tạp của đồ thị tương tác protein, chúng thường được chia thành các đồ thị nhỏ hơn được gọi là mô-đun đại diện cho các nhóm protein có liên quan đến một chức năng cụ thể của tế bào. Hình trên minh họa một mô-đun của đồ thị tương tác protein được mô tả trong [Bo04], bao gồm phức hợp của các protein phân loại RNA trong tế bào người. Để tìm hiểu thêm về đồ thị tương tác protein, hãy xem [Bo04], [Ne10] và [Hu07].

1.2.4 Mạng ngữ nghĩa

Các mô hình đồ thị được sử dụng rộng rãi trong việc hiểu ngôn ngữ tự nhiên và trong việc truy xuất thông tin. Hiểu ngôn ngữ tự nhiên (NLU) là chủ đề của máy móc điều khiển để tháo rời và phân tích cú pháp lời nói của con người. Mục tiêu của nó là cho phép máy móc hiểu và giao tiếp như con người. Truy xuất thông tin (IR) là chủ đề thu thập thông tin từ tập hợp các nguồn dựa trên nhiều loại tìm kiếm khác nhau. Sự hiểu biết ngôn ngữ tự nhiên là công nghệ cho phép khi chúng tôi trò chuyện với các nhân viên dịch vụ khách

hàng tự động. Những tiến bộ trong NLU được thể hiện rõ khi giao tiếp giữa con người và máy móc liên tục được cải thiện. Khi chúng tôi thực hiện tìm kiếm trên web, chúng tôi tận dụng lợi thế của nhiều tiến bộ trong việc truy xuất thông tin được thực hiện trong những thập kỷ gần đây.
Trong các mô hình đồ thị cho các ứng dụng NLU và IR, các đỉnh thường đại diện cho các từ, cụm từ hoặc câu, và các cạnh thể hiện các kết nối liên quan đến ý nghĩa của các đối tượng này.
Trong mạng ngữ nghĩa, các đỉnh được sử dụng để biểu diễn các từ và các cạnh vô hướng được sử dụng để kết nối các đỉnh khi một quan hệ ngữ nghĩa giữ giữa các từ này. Quan hệ ngữ nghĩa là quan hệ giữa hai hoặc nhiều từ dựa trên nghĩa của từ. Ví dụ, chúng ta có thể xây dựng một đồ thị trong đó các đỉnh đại diện cho danh từ và hai đỉnh được nối với nhau khi chúng có ý nghĩa tương tự. Ví dụ, tên của các quốc gia khác nhau có ý nghĩa tương tự, tên của các loại rau khác nhau cũng vậy. Để xác định danh từ nào có nghĩa tương tự, có thể kiểm tra phần lớn văn bản. Các danh từ trong văn bản được phân tách bằng dấu liên kết (chẳng hạn như "hoặc" hoặc "và") hoặc dấu phẩy, hoặc xuất hiện trong danh sách, được cho là có nghĩa tương tự.
Ví dụ, xem xét các sách về nông nghiệp, chúng ta có thể xác định rằng các danh từ đại diện cho tên của các loại trái cây như bơ, bưởi, ổi, xoài, đu đủ và mãng cầu xiêm cũng có nghĩa tương tự. Các nhà nghiên cứu thực hiện phương pháp này bằng cách sử dụng British National Corpus, một tập hợp các văn bản tiếng Anh với 100.000.000 từ, tạo ra một biểu đồ với gần 100.000 đỉnh, đại diện cho danh từ và 500.000 liên kết, kết nối các đỉnh đại diện cho các cặp từ có nghĩa tương tự



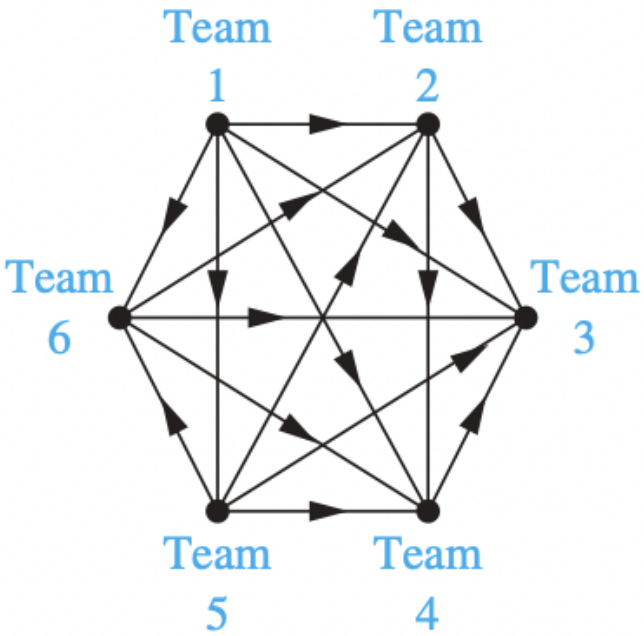
Hình 9: Đồ thị mạng ngữ nghĩa

Hình trên hiển thị một đồ thị nhỏ trong đó các đỉnh đại diện cho danh từ và các cạnh nối các từ có nghĩa tương tự. Biểu đồ này tập trung xung quanh từ chuột. Biểu đồ minh họa rằng có hai ý nghĩa riêng biệt đối với chuột. Nó có thể đề cập đến một con vật hoặc nó

có thể đề cập đến phần cứng máy tính. Khi một chương trình NLU gặp từ mouse trong một câu, nó có thể xem những từ nào có nghĩa tương tự sẽ phù hợp với câu để giúp xác định xem con chuột ám chỉ động vật hay phần cứng máy tính trong câu đó.

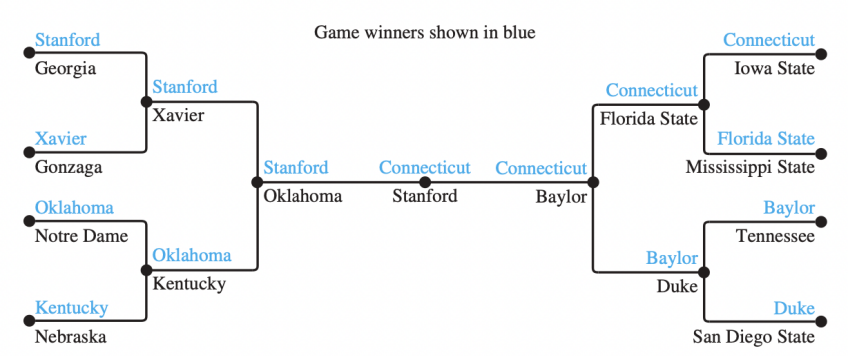
1.2.5 Mạng thi đấu

Thi đấu vòng tròn
Một giải đấu mà mỗi đội đấu với các đội khác đúng một lần và không cho phép hòa được gọi là giải đấu vòng tròn một lượt. Các giải đấu như vậy có thể được mô hình hóa bằng cách sử dụng đồ thị có hướng trong đó mỗi đội được biểu diễn bằng một đỉnh. Lưu ý rằng (a, b) là một cạnh nếu đội a thắng đội b. Biểu đồ này là một biểu đồ có hướng đơn giản, không chứa vòng lặp hoặc nhiều cạnh có hướng (vì không có hai đội chơi với nhau nhiều hơn một lần). Mô hình đồ thị có hướng như vậy được trình bày trong hình. Chúng tôi thấy rằng Đội 1 là đội bất bại trong giải đấu này, và Đội 3 bất phân thắng bại.



Hình 10: Thi đấu vòng tròn

Thi đấu loại trực tiếp
Một giải đấu mà mỗi thí sinh bị loại sau một lần thua được gọi là giải đấu loại trực tiếp. Các giải đấu loại trực tiếp thường được sử dụng trong thể thao, bao gồm giải vô địch quần vợt và giải vô địch bóng rổ NCAA hàng năm. Chúng ta có thể lập mô hình một giải đấu như vậy bằng cách sử dụng một đỉnh để đại diện cho mỗi trò chơi và một cạnh có hướng để kết nối trò chơi với trò chơi tiếp theo mà người chiến thắng trong trò chơi này đã chơi. Biểu đồ trong hình dưới thể hiện các trò chơi của 16 đội cuối cùng trong năm 2010 NCAA của giải đấu bóng rổ.



Hình 11: Thi đấu loại trực tiếp

1.3 Mô hình và ứng dụng của đồ thị trong thực tế

2 Bài 2

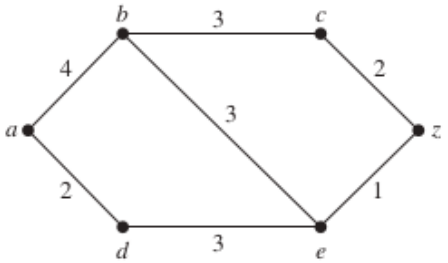
Trong phần này chúng ta sẽ xem xét bài toán "Travelling salesman problem" (bài toán người giao hàng) - là một trong những bài toán kinh điển và nổi tiếng trong lớp các bài toán về đồ thị nói chung và bài toán tìm đường đi ngắn nhất nói riêng.

Bài toán: Cho một danh sách các thành phố và khoảng cách từng cặp một, tìm đường đi ngắn nhất đi qua mỗi thành phố đúng một lần và quay về điểm xuất phát

Chúng ta có thể mô hình bài toán trên bằng đồ thị như sau: *Cho một đơn đồ thị đầy đủ, với các cạnh có trọng số, tìm đường đi xuất phát từ một đỉnh đi qua tất cả các đỉnh của đồ thị mỗi đỉnh đúng một lần với tổng trọng số (chi phí) nhỏ nhất.*
Trước hết chúng ta giải bài toán con đơn giản hơn là tìm đường đi với chi phí nhỏ nhất từ một đỉnh a tới đỉnh z cho trước. Trong bài toán này, không nhất thiết ta phải đi qua tất cả các đỉnh của đồ thị. Có nhiều cách tiếp cận bài toán, cách đầu tiên mà ta có thể nghĩ ngay tới là *vét cạn (brute force)*, tuy nhiên việc liệt kê toàn bộ cấu hình (toàn bộ các đường đi khả dĩ từ a đến z) của bài toán rất tốn thời gian. Sau đây ta cùng xem xét một thuật toán hiệu quả hơn là thuật toán Dijkstra.

2.1 Thuật toán Dijkstra tìm đường đi ngắn nhất trên đồ thị

Trước khi giải bài toán tổng quát, ta hãy xem xét một ví dụ đơn giản sau đây:



Hình 12: Ví dụ đơn đồ thị với trọng số

Tìm đường đi ngắn nhất từ đỉnh a đến đỉnh z trong đồ thị ở hình 12.

Trước hết, bắt đầu từ đỉnh a, có 2 đỉnh kết nối với a là b và d với trọng số lần lượt là 4 và 2, đương nhiên d là đỉnh gần a nhất. Chúng ta có thể tìm đỉnh gần nhất thứ 2 bằng cách liệt kê tất cả các đường đi với d là đỉnh bắt đầu và đỉnh kết thúc không nằm trong tập $\{a, d\}$. Tính từ đỉnh d, chỉ có một đường khả dĩ là e. Từ đó ta có tập $\{a, d, e\}$. Tương tự như vậy ta tìm đỉnh tiếp theo với cạnh có trọng số nhỏ nhất với e là đỉnh bắt đầu và đỉnh kết thúc không thuộc tập $\{a, d, e\}$, ta được đỉnh z. Đường đi cuối cùng là $a \to d \to e \to z$.

Ví dụ trên thể hiện nguyên lý tổng quát của thuật toán Dijkstra đó là đường đi từ đỉnh a tới đỉnh z có thể được xây dựng bằng cách liệt kê các đường đi và tìm đường có trọng số nhỏ nhất để thêm đỉnh đó vào tập đường đi.
Một cách tổng quát ta có thể trình bày thuật toán Dijkstra với lược đồ sau:

Thuật toán Dijkstra
G là một đơn đồ thị kết nối với trọng số mỗi cạnh dương
G có các đỉnh $a = v_0, v_1, ..., v_n = z$, trọng số $w(v_i, v_j)$ với $w(v_i, v_j) = \infty$ nếu $\{v_i, v_j\}$ không thuộc tập cạnh của G

for i = 1 to n

$L(v_i) = \infty$

$L(a) = 0$

$S = \emptyset$ (Khởi tạo các nhãn với nhãn của a là 0, các nhãn khác bằng ∞ và tập S rỗng)

while $z \notin S$

$u = a$ đỉnh không thuộc S với $L(u)$ nhỏ nhất

$S = S \cup \{u\}$

for $v \notin S$

if $L(u) + w(u, v) < L(v)$ then $L(v) = L(u) + w(u, v)$ (Bước này thêm đỉnh gần nhất vào tập S và cập nhật lại hàm chi phí)

return $L(z)$ ($L(z)$ là độ dài của đường đi ngắn nhất từ a tới z)

Nhận xét: Thực chất thuật toán Dijkstra vẫn phải duyệt qua toàn bộ cấu hình của bài toán và không làm giảm độ phức tạp so với việc liệt kê toàn bộ đường đi và tính giá trị chi phí cho mỗi cấu hình. Tuy nhiên thuật toán Dijkstra đem lại sự tường minh mặc dù quét toàn bộ các đỉnh không thuộc tập đã đi qua (S) nhưng ta chỉ cập nhật đỉnh gần với đỉnh cuối cùng nhất, điều đó làm giảm không gian lưu trữ so với cách vét cạn (khi ta phải lưu toàn bộ đường đi và chi phí của mọi đường đi khả dĩ từ đỉnh a tới đỉnh z).

Trong thực tế lập trình, chúng ta không nhất thiết phải lưu giá trị cạnh bằng một số lớn (vô cùng) mà có thể đánh bằng 0 và kiểm tra thêm một điều kiện trong vòng lặp để tránh sự cồng kềnh của ma trận kề.

python code cho bài toán.

2.2 Bài toán người đưa hàng (TSP)

2.2.1 Thuật toán tìm nghiệm chính xác

Trong mục này chúng ta sẽ xem xét bài toán người đưa hàng đã đề cập ở đầu bài. Sau khi giải bài toán tìm đường đi ngắn nhất ở mục trước, ta biết rằng bài toán người đưa hàng không gì khác ngoài việc tìm được đi ngắn nhất khi ta đặt điểm kết thúc chính là điểm bắt đầu.
python code cho bài toán.

Nhận xét: Như đã nói ở phần trước, thuật toán này không làm giảm độ phức tạp tính toán so với cách vét cạn và là $O(n!)$. Độ phức tạp này tăng rất nhanh theo n . Ví dụ khi bắt đầu từ 1 đỉnh, ta có $(n-1)!$ chu trình Hamilton, tới đỉnh thứ 2 ta còn $(n-2)!$ chu trình Hamilton và tiếp tục như vậy. Bởi vì ta có thể đi theo thứ tự ngược lại trong chu trình Hamilton nên số chu trình cần sinh ra để có được lời giải là $(n-1)!/2$. Với số đỉnh bằng 25, số chu trình được sinh ra là $24!/2 \sim 3.1 \times 10^{23}$ - một con số lớn khủng khiếp. Trong thực tế, một đơn vị vận chuyển như *Giao hàng tiết kiệm* giao hàng triệu đơn hàng mỗi ngày tới các địa điểm khác nhau! Do đó ta gần như không thể tìm được lời giải chính xác của bài toán TSP trong thực tế mà cần phải sử dụng các phương pháp xấp xỉ để tìm được một đường đi *đủ tốt*.

2.2.2 Phương pháp xấp xỉ