

GRAPH DATABASE

ỨNG DỤNG GRAPH DATABASE

VỚI NEO4J

Giảng viên hướng dẫn
Nguyễn Thanh Tuấn

Nhóm
Nguyễn Như Quỳnh (30%)
Hà Long Giang (30%)
Cao Bá Thiện (30%)
Hồ Xuân Anh Tú (10%)

1. Graph Database Neo4j

Graph Database

Giới thiệu

Cơ sở dữ liệu đồ thị (Graph Database) được xây dựng có mục đích để lưu trữ và điều hướng các mối quan hệ. Các mối quan hệ là công dân hạng nhất trong cơ sở dữ liệu đồ thị và hầu hết giá trị của cơ sở dữ liệu đồ thị được bắt nguồn từ các mối quan hệ này. Cơ sở dữ liệu đồ thị sử dụng các nút để lưu trữ các thực thể dữ liệu và các cạnh để lưu trữ các mối quan hệ giữa các thực thể. Một cạnh luôn có nút bắt đầu, nút kết thúc, kiểu và hướng, và một cạnh có thể mô tả mối quan hệ cha-con, hành động, quyền sở hữu và những thứ tương tự. Không có giới hạn về số lượng và loại mối quan hệ mà một nút có thể có.

Mỗi loại cơ sở dữ liệu đều có điểm mạnh và điểm yếu. Khía cạnh quan trọng nhất là biết sự khác biệt cũng như các tùy chọn có sẵn cho các vấn đề cụ thể.

Bảng 1. So sánh ưu, nhược điểm của cơ sở dữ liệu đồ thị

Ưu Điểm	Nhược Điểm
Các cấu trúc nhanh nhẹn và linh hoạt.	Không có ngôn ngữ truy vấn chuẩn hóa. Ngôn ngữ phụ thuộc vào nền tảng được sử dụng.
Biểu diễn mối quan hệ giữa các thực thể rõ ràng	Đồ thị không thích hợp cho các hệ thống dựa trên giao dịch.
Các truy vấn xuất ra kết quả theo thời gian thực. Tốc độ phụ thuộc vào số lượng các mối quan hệ	Cơ sở người dùng nhỏ nên khó tìm được hỗ trợ khi gặp sự cố.

So sánh cơ sở dữ liệu đồ thị (Graph) và cơ sở dữ liệu quan hệ (RDBMS)

Trong một RDBMS, các truy vấn mối quan hệ dữ liệu có thể được trả lời bằng cách tạo các JOIN giữa các bảng cơ sở dữ liệu. Tuy nhiên, các hoạt động này đòi hỏi nhiều máy tính và bộ nhớ, và tác động của hiệu suất tăng lên khi khối lượng dữ liệu tăng lên. Cơ sở dữ liệu đồ thị có thể phân tích dữ liệu phức tạp, được kết nối nhanh hơn và duy trì phân tích đó để tham khảo trong tương lai.

Độ sâu và mật độ kết nối và khối lượng dữ liệu ảnh hưởng đáng kể đến thời gian truy vấn. Với Neo4j, bạn có thể truy vấn dữ liệu với độ sâu hàng triệu hoặc hàng chục triệu kết nối mỗi giây trên mỗi lõi máy tính, điều này sẽ tương đương trong thế giới cơ sở dữ liệu quan hệ với hàng triệu hoạt động JOIN mỗi giây trên mỗi lõi, điều

này là không thể. Có một sự khác biệt lớn về tốc độ và sự khác biệt này làm tăng tốc độ kết nối của bạn càng chặt chẽ và bạn càng có nhiều dữ liệu hơn.

Về tốc độ, càng có nhiều dữ liệu, thì việc liên kết dữ liệu trong các loại cơ sở dữ liệu khác, bao gồm cả Oracle RDBMS càng trở nên chậm hơn. Sử dụng Neo4j, truy vấn đường dẫn ngắn nhất trên dữ liệu với hàng chục tỷ nút và mỗi quan hệ có thể mất một hoặc hai mili giây để chạy. Truy vấn SQL tương đương sẽ chạy chậm hơn hàng nghìn lần nếu một ứng dụng chỉ sử dụng một RDBMS

Cơ sở dữ liệu đồ thị không nhằm thay thế cơ sở dữ liệu quan hệ. Hiện tại, cơ sở dữ liệu quan hệ là tiêu chuẩn của ngành. Khía cạnh quan trọng nhất là biết mỗi loại cơ sở dữ liệu phải cung cấp những gì. Cơ sở dữ liệu quan hệ cung cấp cách tiếp cận có cấu trúc đối với dữ liệu, trong khi cơ sở dữ liệu biểu đồ nhanh nhẹn và tập trung vào thông tin chi tiết về mối quan hệ dữ liệu nhanh chóng.

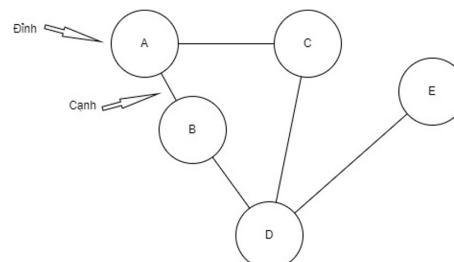
Cả cơ sở dữ liệu đồ thị và cơ sở dữ liệu quan hệ đều có miền của chúng. Các trường hợp sử dụng với các mối quan hệ phức tạp tận dụng sức mạnh của cơ sở dữ liệu đồ thị, vượt trội hơn so với cơ sở dữ liệu quan hệ truyền thống. Cơ sở dữ liệu quan hệ như MySQL hoặc SQL yêu cầu lập kế hoạch cẩn thận khi tạo mô hình cơ sở dữ liệu, trong khi đồ thị có cách tiếp cận dữ liệu tự nhiên và linh hoạt hơn nhiều.

Tiêu chí so sánh	Graph Database	RDBMS
Định dạng	Các nút và các cạnh có thuộc tính	Bảng có hàng và có cột
Các mối quan hệ	Được biểu diễn bằng các cạnh giữa các nút	Được tạo bằng khóa ngoại giữa các bảng
Tính linh hoạt	Linhh hoạt	Không linh hoạt
Mức độ phức tạp truy vấn	Nhanh chóng và đáp ứng	Yêu cầu các phép nối phức tạp
Các trường hợp sử dụng	Hệ thống có mối quan hệ kết nối cao	Hệ thống tập trung vào giao dịch với các mối quan hệ đơn giản hơn

Bảng 2. So sánh Graph Database và RDBMS

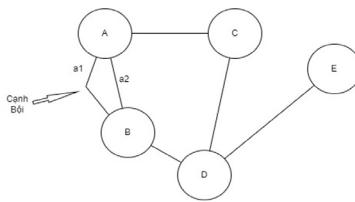
Graph Theory (Lý thuyết đồ thị)

Đơn đồ thị vô hướng ($G = \langle V, E \rangle$): Là đồ thị gồm tập V các đỉnh và tập E các cặp không có thứ tự gồm 2 phần tử khác nhau của tập V được gọi là tập cạnh



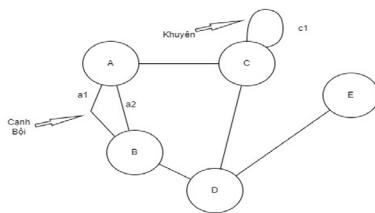
Hình 1. Ví dụ về đơn đồ thị vô hướng

Đa đồ thị vô hướng ($G = \langle V, E \rangle$): Đa đồ thị vô hướng gồm tập V các đỉnh và tập E là họ các cặp không có thứ tự gồm hai phần tử khác nhau của tập V gọi là các cạnh. Hai cạnh a_1, a_2 được gọi là cạnh bội nếu chúng cùng tương ứng với 1 cặp đỉnh



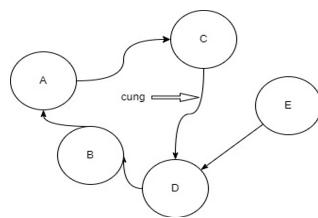
Hình 2. Ví dụ về đa đồ thị vô hướng

Giả đồ thị vô hướng ($G = \langle V, E \rangle$): Giả đồ thị vô hướng gồm tập V các đỉnh và tập E là họ các cặp không có thứ tự gồm hai phần tử khác nhau của tập V gọi là các cạnh. Cạnh c1 được gọi là khuyên



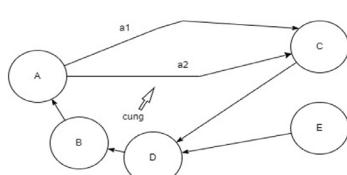
Hình 3. Ví dụ về giả đồ thị vô hướng

Đơn đồ thị có hướng ($G = \langle V, E \rangle$): Đơn đồ thị có hướng gồm tập V các đỉnh và tập E là họ các cặp có thứ tự gồm hai phần tử của tập V gọi là các cung.



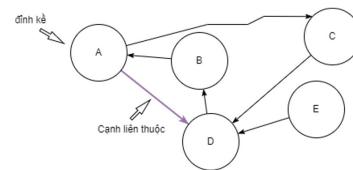
Hình 4. Ví dụ về đơn đồ thị có hướng

Đa đồ thị có hướng ($G = \langle V, E \rangle$): Đơn đồ thị có hướng gồm tập V các đỉnh và tập E là họ các cặp có thứ tự gồm hai phần tử của tập V gọi là các cung. Hai cạnh a_1, a_2 với 1 cặp đỉnh được gọi là cung lặp.



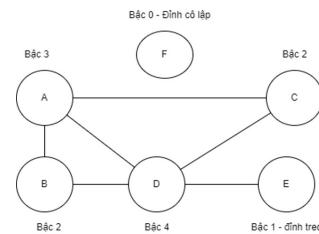
Hình 5. Ví dụ về đa đồ thị có hướng

Cạnh liên thuộc: nếu tồn tại cạnh $e = (a_1, a_2)$ là cạnh của đồ thị thì cạnh e được gọi là cạnh liên thuộc với các đỉnh.



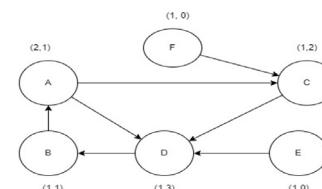
Hình 6. Ví dụ về đỉnh kề và cạnh liên thuộc của đồ thị

Bậc của đỉnh trên đồ thị vô hướng: Bậc của đỉnh trên đồ thị vô hướng là số cạnh liên thuộc với với đỉnh ký hiệu $\deg(u)$ – trong đó u là tên đỉnh . Đỉnh có bậc bằng 0 được gọi là đỉnh cô lập , đỉnh có bậc bằng 1 được gọi là đỉnh treo . trong 1 đồ thị $G = \langle V, E \rangle$ là đồ thị vô hướng gồm n cạnh , tổng số bậc của các đỉnh trong đồ thị lúc này bằng $2n$.



Hình 7. Ví dụ về bậc của đỉnh trên đồ thị vô hướng

Bán bậc của đỉnh trên đồ thị có hướng: Bán bậc ra của đỉnh a là số cung đi ra từ đỉnh a ký hiệu là $\deg^+(u)$, Bán bậc vào của đỉnh a là số cung đi vào đỉnh a ký hiệu là $\deg^-(u)$. Trên đồ thị có hướng , tổng số bán bậc ra bằng tổng số bán bậc vào và bằng n số cạnh của đỉnh



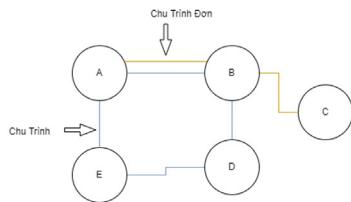
Hình 8. Ví dụ về bán bậc của đỉnh trên đồ thị có hướng

Chu trình và Đường đi (path):

Đường đi là dãy các đỉnh hoặc dãy các cạnh trong đó 2 đỉnh liên tiếp có cạnh nối. Độ dài đường đi là số cạnh trên đường đi. Đường đi đơn là trường hợp có các đỉnh trên đường đi là các đỉnh phân biệt

Chu trình là đường đi gồm các cạnh phân biệt có đỉnh đầu trùng đỉnh cuối

Chu trình đơn là 1 chu trình trong đó ngoại trừ đỉnh đầu và đỉnh cuối thì không còn đỉnh nào khác trùng nhau



Hình 9. Ví dụ về Chu Trình

Liên thông và Thành phần liên thông :

Đồ thị vô hướng được gọi là liên thông nếu luôn tìm được đường đi giữa 2 đỉnh bất kỳ của đồ thị.

Trong trường hợp đồ thị vô hướng không liên thông, nó phân rã thành các thành phần liên thông (TPLT). Như vậy đồ thị vô hướng liên thông nếu nó có số TPLT là 1. Đỉnh cô lập cũng là 1 thành phần liên thông.

Ứng dụng

Công cụ đề xuất thời gian thực: Các đề xuất về sản phẩm và thương mại điện tử trong thời gian thực mang lại trải nghiệm người dùng tốt hơn đồng thời tối đa hóa lợi nhuận. Các trường hợp đáng chú ý bao gồm Netflix, eBay và Walmart

Quản lý dữ liệu tổng thể: Liên kết tất cả dữ liệu của công ty với một vị trí cho một điểm tham chiếu duy nhất cung cấp tính nhất quán và độ chính xác của dữ liệu. Quản lý dữ liệu tổng thể là rất quan trọng đối với các công ty toàn cầu quy mô lớn

Quản lý tài sản kỹ thuật số: Lượng nội dung số khổng lồ và không ngừng tăng lên. Cơ sở dữ liệu đồ thị cung cấp một mô hình cơ sở dữ liệu có thể mở rộng và đơn giản để theo dõi các tài sản kỹ thuật số, chẳng hạn như tài liệu, đánh giá, hợp đồng, v.v.

Phát hiện gian lận: Việc tìm kiếm các mẫu đáng ngờ và phát hiện các giao dịch thanh toán gian lận được thực hiện trong thời gian thực bằng cách sử dụng cơ sở dữ liệu biểu đồ. Nhắm mục tiêu và cô lập các phần của biểu đồ giúp phát hiện hành vi lừa đảo nhanh hơn.

Tìm kiếm ngữ nghĩa: Xử lý ngôn ngữ tự nhiên là mơ hồ. Tìm kiếm theo ngữ nghĩa giúp cung cấp ý nghĩa đằng sau các từ khóa để có kết quả phù hợp hơn, dễ dàng lập bản đồ hơn bằng cách sử dụng cơ sở dữ liệu biểu đồ

Quản lý mạng: Về bản chất, mạng là các đồ thị được liên kết. Đồ thị làm giảm thời gian cần thiết để cảnh báo cho quản trị viên mạng về các sự cố trong mạng.

Định tuyến: Thông tin truyền qua mạng bằng cách tìm các đường dẫn tối ưu làm cho cơ sở dữ liệu đồ thị trở

thành lựa chọn hoàn hảo cho việc định tuyến.

Neo4j

Giới thiệu

Neo4j là hệ quản trị cơ sở dữ liệu đồ thị đầu tiên được giới thiệu vào năm 2007 và công bố phiên bản 1.0 vào năm 2010. Hiện nay neo4j là một trong những hệ quản trị cơ sở dữ liệu đồ thị được sử dụng nhiều nhất.

Neo4j được viết bởi ngôn ngữ java và ngôn ngữ truy vấn là Cypher.

Hệ sinh thái Neo4j

Hệ sinh thái ứng dụng của Neo4J bao gồm Neo4J Database, Neo4J Desktop, Neo4J Browser và Neo4J Bloom. Ngoài ra Neo4J còn có thể kết hợp sử dụng với GraphQL và Spark

Neo4j Database	Neo4j Desktop	Neo4j Brower	Neo4j Bloom
Sản phẩm chính và trọng tâm của Neo4j. Nó xử lý các transactional lần analytics workloads và được tối ưu hóa để duyệt các đường dẫn(traversing paths) thông qua dữ liệu bằng cách sử dụng các mối quan hệ (relationships) trong biểu đồ để tìm kết nối giữa các thực thể (entities).	Là một IDE dành riêng cho Neo4j, nơi quản lý các projects và database tại máy chủ local hoặc có thể kết nối đến Neo4j servers.	Là một cypher command shell giúp người dùng tương tác và visualize đồ thị(graph).	Là một trình khai phá dữ liệu (data exploration) giúp người dùng có thể visualize, navigate và query mà không cần viết code. Người dùng cũng có thể viết các câu trúc (patterns) gần với ngôn ngữ tự nhiên để truy xuất (retrieve) dữ liệu và duyệt (traverse) qua các lớp(layers) của biểu đồ. Người dùng cũng có thể chỉnh sửa (edit), cập nhật (update) trực tiếp khi gặp dữ liệu bị thiếu (missing data) hoặc dữ liệu lỗi (bad data).

Neo4j lưu trữ dữ liệu

Neo4j lưu trữ các nút, mối quan hệ, nhãn và thuộc tính trong các tệp riêng biệt.

Các nút được lưu trữ trong tệp neostore.nodestore.db. Tệp này có kích thước cố định theo mỗi nút được tạo mới. Đối với mỗi nút được thêm vào cơ sở dữ liệu, tệp này được tăng thêm 9 byte. Bằng cách này, một nút có id 100 có thể dễ dàng được tìm thấy trong 900 byte trong tệp (id 100 x 9 byte mỗi nút = 900 byte). Bản ghi nút có các con trỏ đến mối quan hệ nút đầu tiên, thuộc tính nút đầu tiên và cho các nhãn nút.

Các mối quan hệ được lưu trữ trong tệp neotore.relationshipstore.db. Đây cũng là một tệp có kích thước cố định. Mỗi mối quan hệ có các con trỏ đến các nút bắt đầu và kết thúc, kiểu quan hệ (trong neostore.relationshiptype-store.db), các bản ghi mối quan hệ tiếp theo và trước đó cho mỗi nút bắt đầu và kết thúc và một cờ cho biết liệu mối quan hệ có phải là mối quan hệ đầu tiên trong chuỗi quan hệ hay không.

Thuộc tính của các nút và các mối quan hệ được lưu trữ trong tệp neostore.propertystore.db. Mỗi bản ghi thuộc tính có một con trỏ đến thuộc tính tiếp theo và có thể chứa tối đa 4 thuộc tính. Mỗi thuộc tính có một con trỏ đến tên thuộc tính (neostore.propertystore.db.index-tệp), loại thuộc tính. Giá trị thuộc tính có thể là một giá Graph Database | 4

trị nội tuyến hoặc một con trỏ đến một tệp động cho chuỗi lớn (neostore.propertystore.db.strings) và mảng (neostore.propertystore.db.arrays file).

2. Cypher

Cypher Query

Giới thiệu

Cypher là một ngôn ngữ truy vấn cơ sở dữ liệu đồ thị, với ngôn ngữ này chúng ta có thể tương tác như là truy vấn, cập nhập hay là quản trị một cách hiệu quả với cơ sở dữ liệu đồ thị. Ngôn ngữ này được thiết kế giúp cho developer cũng như là các chuyên gia có thể thuận tiện khi làm việc với Neo4j. Cypher dựa trên ASCII, vì vậy cú pháp của nó dễ hiểu và làm cho các truy vấn dễ hiểu hơn. Nó tập trung vào việc diễn đạt rõ ràng những gì cần truy xuất từ một biểu đồ, chứ không phải về cách truy vấn nó. Cypher được coi là ngôn ngữ truy vấn đồ thị dễ tìm hiểu nhất.

Cypher được lấy cảm hứng từ rất nhiều các cách tiếp cận khác nhau, một số các từ khóa như là WHERE, ORDER BY được lấy cảm hứng từ ngôn ngữ SQL, trong khi đó pattern matching thì lại được mượn từ SPARQL. Ngoài ra một vài ngữ nghĩa thì lại được mượn từ các ngôn ngữ khác như là Haskell và Python. Cấu trúc của Cypher được xây dựng dựa trên ngôn ngữ Tiếng Anh với ngữ nghĩa thuận tiện cho người thao tác với ngôn ngữ, điều này giúp cho việc viết và đọc các câu query cũng dễ dàng hơn.

Định dạng

Định dạng nút (Nodes)

(): nút rỗng

(varname:NodeName): Nút có nhãn là NodeName, tên biến của nút là varname. Nút có thể không có tên biến

Truy vấn clean: Gợi ý sử dụng cách đặt tên nhãn theo kiểu CamelCase (viết hoa chữ cái đầu).

Nên đặt tên nhãn mang tính gợi nhớ tới đối tượng, nhãn nút nên là một danh từ

Tên nhãn có phân biệt chữ hoa chữ thường

Định dạng quan hệ (Relationship)

[varname:RELATIONSHIP_NAME]: mỗi quan hệ có nhãn là RELATIONSHIP_NAME và biến quan hệ là varname .

Truy vấn clean: Gợi ý cách đặt tên nhãn theo kiểu up-

per_case, tất cả được viết hoa và sử dụng dấu gạch nối giữa các từ. Nhãn của các quan hệ nên là động từ. Tên nhãn có phân biệt chữ hoa chữ thường.

Khóa thuộc tính, biến, tham số, bí danh và hàm

Ví dụ: title, size(), count(), firstName...

Có phân biệt chữ hoa chữ thường

Truy vấn sạch: nên viết theo định dạng camelCase(chữ cái đầu viết thường)

Mệnh đề

Ví dụ: MATCH, WHERE, WITH, UNWIND...

Các mệnh đề không phân biệt chữ hoa chữ thường

Truy vấn sạch: các mệnh đề nên được tạo kiểu là tất cả các chữ in hoa, được đặt ở đầu mỗi dòng để dễ đọc và dễ truy vấn

Keyword

Ví dụ: AND, OR, IN, NOT, DISTINCT, STARTS WITH, CONTAINS, ENDS WITH,...

Các keyword không phân biệt chữ hoa chữ thường

Truy vấn sạch: Nên được viết in hoa, không cần đặt ở đầu dòng mới

Thụt dòng và ngắt dòng

Mỗi mệnh đề nên được ngắt dòng, ngoài ra, các khối truy vấn con, ON CREATE, ON MATCH nên được ngắt dòng và thụt vào 2 khoảng trắng

Sử dụng dấu ngoặc nhọn để nhóm khối truy vấn con. Nếu truy vấn con chỉ có 1 dòng, không cần đặt nó xuống dòng riêng hoặc thụt lề

Metacharacter

Dấu nháy đơn: nên dùng cho các giá trị chuỗi bằng chữ

Ví dụ: 'Mats\' quote: "statement" ', "Cypher's a nice language"

Backticks

Dùng để tránh thoát các ký tự có khoảng trắng hoặc ký tự đặc biệt

Ví dụ: MATCH ('odd-ch@racter\$`'Spaced Label`{`&property`': 42})

Dấu chấm phẩy

Dùng trong trường hợp có 1 tập các câu lệnh Cypher và cần phân tách giữa các câu lệnh. Không nên dùng khi chỉ có 1 câu lệnh

Giá trị null và boolean

Nên được viết thường trong truy vấn

Ví dụ: p.birthday = null, missingBirth = true

Xử lý các mẫu

Nếu một mẫu(pattern) bị tràn dòng, nên ngắt dòng sau mũi tên, không phải trước

`MATCH (:Person)--(:Company)--> (:Position)`

Sử dụng các nút và mối quan hệ ẩn danh nếu không được sử dụng sau này trong truy vấn

`MATCH (:Person)-[:Likes]->(technology:Technology)
RETURN technology`

Các mẫu nên nối với nhau để tránh lặp lại biến

Các nút được đặt tên hoặc các điểm chung nên được đặt ở đầu mệnh đề MATCH. Các mối quan hệ mẫu đi (->) nên để trước các mối quan hệ mẫu đến (<--)

Khoảng trắng

Không nên để khoảng trắng giữa không gian vị từ nhãn.
Không có khoảng trắng giữa các mẫu

Một khoảng trắng ở hai bên toán tử. Một khoảng trắng sau dấu phẩy

Không nên để khoảng trắng giữa 2 dấu ngoặc đơn

Dùng khoảng trắng giữa truy vấn con và dấu ngoặc nhọn

Cypher Tutorial

CREATE

Mệnh đề CREATE được dung để tạo ra các nút (nodes) và các mối quan hệ (relationships).

1. Tạo nút

1.1. Tạo nút đơn

Tạo nút với cypher command sau đây:

`CREATE (p)`

1.2. Tạo nhiều nút

Tạo các nút với cypher command sau đây:

`CREATE (p), (q)`

1.3. Tạo 1 nút với 1 label, 1 property

Tạo 1 nốt với label :Person, property {name: 'Johnny Depp'}

`CREATE (:Person {name: 'Johnny Depp'})`

1.4. Tạo các nốt với nhiều labels và properties

Khi tạo các nốt mới với nhiều labels, ta có thể them properties vào cùng lúc

Tạo 2 node có label :Person:Actor với property {name: 'Johnny Depp', born: 1963}, {name: 'Amber Heard', born: 1986}

`CREATE (p:Person:Actor {name: 'Johnny Depp',
born: 1963}), (q:Person:Actor {name: 'Amber
Heard', born: 1986})`

1.5. Trả về nút đã tạo ra

`CREATE (a {name: 'Johnny Depp'})
RETURN a.name`

Tên của nút mới được tạo ra được trả về

2. Tạo mối quan hệ giữa 2 nút

Để tạo mối quan hệ giữa hai nút, trước tiên ta lấy ra được hai nút. Khi các nút được lấy ra, ta chỉ cần tạo mối quan hệ giữa chúng.

Tạo 1 mối quan hệ rằng p và q đã li hôn với nhau

`MATCH (p:Person:Actor {name: 'Johnny Depp', born: 1963}), (q:Person:Actor {name: 'Amber Heard', born: 1986})`

`CREATE (p)-[rel:IS_DIVORCED_WITH]->(q)`

Mối quan hệ được tạo ra được trả lại bởi truy vấn

3. Tạo một đường dẫn đầy đủ.

Khi bạn sử dụng create và một mẫu, tất cả các phần của mẫu chưa có phạm vi tại thời điểm này sẽ được tạo

`CREATE a = (p:Person:Actor {name: 'Johnny Depp', born: 1963, status:Divorced})-[:WORKS_AT]->(FILM_STUDIO)<-[:WORKS_AT]-(q:Person:Actor {name: 'Amber Heard', born: 1986, status:Divorced})`

```
RETURN a
```

Truy vấn này tạo ra ba nút và hai mối quan hệ trong một lần, gán nó cho một biến đường dẫn và trả về nó.

4. Sử dụng các tham số với CREATE

4.1. Tạo nút với một tham số cho các thuộc tính

Bạn cũng có thể tạo graph entity từ bản đồ. Tất cả các cặp khóa/giá trị trong bản đồ sẽ được đặt làm thuộc tính trên mỗi quan hệ hoặc nút được tạo. Trong trường hợp này, chúng tôi cũng thêm một nhãn Position vào nút.

Parameter

```
{
  "props" : {
    "name" : "Johnny Depp",
    "position" : "Dior's ambassador"
  }
}
CREATE (p:Person $props)
RETURN p
```

4.2. Tạo nhiều nút với một tham số cho các thuộc tính của chúng

Bằng cách cung cấp cho cypher một mảng bản đồ, nó sẽ tạo một nút cho mỗi bản đồ. Parameter

```
{
  "props" : [ {
    "name" : "Johnny Depp",
    "position" : "Dior's ambassador"
  }, {
    "name" : "Amber Heard",
    "position" : "In Debt"
  } ]
}
UNWIND $props AS map
CREATE (n)
SET n = map
```

MATCH

Mệnh đề MATCH được sử dụng để tìm kiếm mẫu được mô tả trong đó.

1. Giới thiệu

Mệnh đề MATCH cho phép bạn chỉ định các mẫu mà Neo4j sẽ tìm kiếm trong cơ sở dữ liệu. Đây là cách chính để đưa dữ liệu vào tập hợp các ràng buộc hiện tại.

MATCH thường được kết hợp với phần WHERE để thêm

các hạn chế hoặc vị từ (predicates) vào các mẫu MATCH, làm cho chúng cụ thể hơn. Các vị từ là một phần của mô tả mẫu và không nên được coi là một bộ lọc chỉ được áp dụng sau khi đối sánh xong. Điều này có nghĩa là WHERE luôn phải được đặt cùng với mệnh đề MATCH mà nó thuộc về.

MATCH có thể xảy ra ở đầu truy vấn hoặc sau đó, có thể sau WITH. Nếu đó là mệnh đề đầu tiên, sẽ không có gì bị ràng buộc và Neo4j sẽ thiết kế một tìm kiếm để tìm các kết quả phù hợp với mệnh đề và bất kỳ vị từ liên quan nào được chỉ định trong bất kỳ phần WHERE nào. Điều này có thể liên quan đến việc quét cơ sở dữ liệu, tìm kiếm các nút có một nhãn nhất định hoặc tìm kiếm một mục để cho khớp mẫu. Các nút và mối quan hệ được tìm thấy bởi tìm kiếm này có sẵn dưới dạng các phần tử mẫu liên kết và có thể được sử dụng để đối sánh mẫu của các đường dẫn. Chúng cũng có thể được sử dụng trong bất kỳ mệnh đề MATCH nào khác, trong đó Neo4j sẽ sử dụng các phần tử đã biết và từ đó tìm ra các phần tử chưa biết khác.

2. Tìm các node cơ bản

2.1. Lấy ra tất cả các nốt

Bằng cách chỉ định một mẫu với một nút duy nhất và không có nhãn, tất cả các nút trong biểu đồ sẽ được trả về.

Tạo 7 nodes bao gồm: 5 nodes có label :Person, property {name:}; 2 nodes có label :movie và property {title:....}. Sau đó lấy ra tất cả các node.

```
CREATE (a:Person {name:"Lưu Diệc Phi"}),
       (b:Person {name:"Huỳnh Hiểu Minh"}),
       (c:Person {name:"Dương Mịch"}),
       (d:Person {name:"Hoắc Kiến Hoa"}),
       (e:Person {name:"Trịnh Hiểu Long"}),
       (f:Person {name:"Nguyễn Như Quỳ"}),
       (m1:Movie {title:"Thần Đèo Đại Hiệp"}),
       (m2:Movie {title:"Tiên Kiếm Kì Hiệp"}),
       (a)-[:ACTED_IN]->(m1),
       (b)-[:ACTED_IN]->(m1),
       (c)-[:ACTED_IN]->(m1),
       (d)-[:ACTED_IN]->(m2),
       (c)-[:ACTED_IN]->(m2),
       (a)-[:ACTED_IN]->(m2),
       (e)-[:DIRECTED]->(m1),
       (f)-[:DIRECTED]->(m2);
```

MATCH (n)

RETURN n

2.2. Lấy tất cả các node với 1 label

Lấy tất cả các node có label được thực hiện với một node

Lấy label :movie với property {title:...}

```
MATCH (movie:Movie)  
RETURN movie.title
```

Returns all the movies in the database.

2.3. Lấy các node có mối liên hệ

Biểu tượng -- nghĩa là liên kết với(related to), mà không liên quan đến loại hoặc hướng của mối quan hệ

Trả lại tất cả các phim được đạo diễn bởi 'Nguyễn Như Quỳ'

```
MATCH (director {name: 'Nguyễn Như Quỳ'})--(movie)  
RETURN movie.title
```

2.4. Match với labels

Kết hợp với các label để ràng buộc mẫu của ta với các label trên các nút

Lấy ra các node liên quan đến label :Person 'Huỳnh Hiểu Minh' mà có label :movie

```
MATCH (:Person {name: 'Huỳnh Hiểu Minh'})--(movie:Movie)  
RETURN movie.title
```

3. Tìm các mối quan hệ cơ bản

3.1. Mối quan hệ Outgoing

Khi hướng của một mối quan hệ được quan tâm, nó được hiển thị bằng cách sử dụng-> hoặc <-.

Trả về bất kỳ node nào được kết nối với :Person 'Lưu Diệc Phi' bởi một mối quan hệ hướng ngoại.

```
MATCH (:Person {name: 'Lưu Diệc Phi'})-->(movie)  
RETURN movie.title
```

3.2. Các mối quan hệ và biến được định hướng

Nếu cần có một biến, để lọc các thuộc tính của mối quan hệ hoặc để trả về mối quan hệ, đây là cách ta giới thiệu biến

Trả về loại mối quan hệ từ 'Dương Mịch'

```
MATCH (:Person {name: 'Dương Mịch'})-[r]->(movie)  
RETURN type(r)
```

3.3. Ghép các loại mối quan hệ

Khi ta biết loại mối quan hệ cần muốn nối đến, ta có thể chỉ định nó bằng cách sử dụng dấu hai chấm cùng với loại mối quan hệ.

Trả về tất cả các diễn viên đã ACTED_IN 'Thần ĐIÊN ĐẠI HIỆP'

```
MATCH (thandieudaihiep:Movie {title: 'Thần ĐIÊN  
ĐẠI HIỆP'})<-[ACTED_IN]-(actor)  
RETURN actor.name
```

3.4. Kết hợp nhiều loại mối quan hệ

Để nối một trong nhiều loại, ta có thể chỉ định điều này bằng cách cho chúng vào cùng chuỗi với ký hiệu |.

Trả về các node với label ACTED_IN hoặc DIRECTED có mối quan hệ với 'Tiên Kiếm Kì Hiệp'.

```
MATCH (tienkiemkihiệp {title: 'Tiên Kiếm Kì  
Hiệp'})<-[ACTED_IN|DIRECTED]-(person)  
RETURN person.name
```

4. Relationships in depth

4.1. Các loại mối quan hệ với các ký tự không phổ biến

Đôi khi cơ sở dữ liệu của bạn sẽ có các loại có ký tự không chữ hoặc có khoảng trống trong đó. Sử dụng `\``(backtick) để trích dẫn những điều này.

Ta có thể thêm một mối quan hệ bổ sung giữa 'Lưu Diệc Phi' và 'Dương Mịch'. Trả lại loại mối quan hệ với khoảng trống trong nó

```
MATCH  
    (ludiph:Person {name: 'Lưu Diệc Phi'}),  
    (dumi:Person {name: 'Dương Mịch'})  
CREATE (dumi)-[:`IS FRIEND WITH`]->(ludiph)
```

Tạo nên graph mới như dưới đây:

```
MATCH (n {name: 'Rob Reiner'})-[r: 'TYPE INCLUDING A SPACE']->()
RETURN type(r)
```

4.2. Multiple relationships

Nhiều mối quan hệ có thể được thể hiện bằng cách sử dụng nhiều câu dưới dạng ()-() hoặc chúng có thể được xâu chuỗi lại với nhau.

Trả lại bộ phim 'Hoắc Kiến Hoa' đã diễn ra và đạo diễn của bộ phim

```
MATCH (hokiko {name: 'Hoắc Kiến Hoa'})-[:ACTED_IN]->(movie)<-[:DIRECTED]-(director)
RETURN movie.title, director.name
```

4.3. Variable length relationships

Các node là một số lượng thay đổi của relationship->node các bước nhảy của node có thể được tìm thấy bằng cách sử dụng cú pháp sau:

-[:TYPE*minHops..maxHops]->. minHops và maxHops là tùy chọn. Mặc định tương ứng là 1 và vô cùng. Khi không có giới hạn nào được đưa ra, các dấu chấm có thể bị bỏ qua. Các dấu chấm cũng có thể bị bỏ qua khi chỉ đặt một giới hạn và điều này ngụ ý một mẫu độ dài cố định.

Trả lại các bộ phim liên quan đến 'Dương Mịch' từ 1 đến 2 hops.

```
MATCH (dumi {name: 'Dương Mịch'})-[:ACTED_IN-*1..2]->(movie:Movie)
```

```
RETURN movie.title
```

4.4. Mỗi quan hệ có độ dài thay đổi với nhiều kiểu quan hệ

Các mối quan hệ có độ dài thay đổi có thể được kết hợp với nhiều kiểu quan hệ. Trong trường hợp này, *minHops..maxHops áp dụng cho tất cả các kiểu quan hệ cũng như bất kỳ sự kết hợp nào của chúng.

Trả lại tất cả label :Person liên quan đến 'Huỳnh Hiểu Minh' bởi 2 hops với tất cả mọi quan hệ loại ACTED_IN và DIRECTED

```
MATCH (huhumi {name: 'Huỳnh Hiểu Minh'})-[:ACTED_IN|DIRECTED*2]-(person:Person)
```

```
RETURN person.name
```

4.5. Đường dẫn có độ dài bằng không

Sử dụng các đường dẫn độ dài thay đổi có giới hạn 0 có nghĩa là hai biến có thể trở đến cùng một nút. Nếu độ

dài đường dẫn giữa hai nút bằng 0, theo định nghĩa thì chúng là cùng một nút.

Lưu ý rằng khi đối sánh các đường dẫn có độ dài bằng 0, kết quả có thể chứa một kết quả phù hợp ngay cả khi đối sánh trên một kiểu quan hệ không được sử dụng.

Trả lại chính bộ phim cũng như các diễn viên và đạo diễn của nó

```
MATCH (thandieudaihiep:Movie {title: 'Thần ĐIÊN ĐẠI HIỆP'})-[*0..1]-(x)
```

```
RETURN x
```

4.6. Đặt tên cho đường dẫn

Nếu muốn trả lại hoặc lọc một đường dẫn, ta có thể đặt tên cho đường dẫn.

Trả lại 2 đường dẫn bắt đầu bằng 'Lưu Diệc Phi'

```
MATCH p = (ludipi {name: 'Lưu Diệc Phi'})-->()
```

```
RETURN p
```

WHERE

WHERE thêm các ràng buộc vào mẫu trong mệnh đề MATCH hoặc mệnh đề OPTIONAL MATCH hoặc lọc kết quả mẫu với mệnh đề WITH..

1. Tìm các mẫu với WHERE

1.1 Lọc các mẫu

Các mẫu là các biểu thức trong Cypher, các biểu thức trả về một danh sách các đường dẫn. Biểu thức danh sách cũng là các vị từ (predicates) - một danh sách trống biểu thị false và một biểu thức không trống biểu thị true.

Vì vậy, các mẫu không chỉ là biểu thức, chúng còn là vị từ. Hạn chế duy nhất đối với mẫu là phải thể hiện nó trong một đường dẫn duy nhất. Không thể sử dụng dấu phẩy giữa nhiều đường dẫn như bạn làm trong MATCH. Bạn có thể đạt được hiệu ứng tương tự bằng cách kết hợp nhiều mẫu với AND.

Lưu ý rằng bạn không thể giới thiệu các biến mới ở đây. MATCH (a) - [*] -> (b) rất khác với WHERE (a) - [*] -> (b). Đường đầu tiên sẽ tạo ra một đường dẫn cho mọi đường đi mà nó có thể tìm thấy giữa a và b, trong khi đường dẫn thứ hai sẽ loại bỏ bất kỳ đường dẫn phù hợp nào trong đó a và b không có chuỗi quan hệ định hướng giữa chúng.

Tên và tuổi của các node có mối quan hệ outgoing với node 'Johnny Depp' được trả về.

```
CREATE (p:Person:Actor {name: 'Johnny Depp', born: 1963, age: 58}), (q:Person:Actor {name: 'Amber Heard', born: 1986, age: 36}), (z:Person {name: 'Elon Musk', born: 1971, age: 50})
```

```
MATCH (p:Person:Actor {name: 'Johnny Depp', born: 1963}), (q:Person:Actor {name: 'Amber Heard', born: 1986}), (z:Person {name: 'Elon Musk', born: 1971, age: 50})
```

```
CREATE (p)-[:IS_DIVORCED_WITH]->(q),
       (q)-[:HAVE_AN_AFFAIR_WITH]->(z),
       (z)-[:IS_FRIEND_WITH]->(p),
       (p)-[:IS_FRIEND_WITH]->(z);
MATCH
  (p:Person {name: 'Johnny Depp'}),
  (other:Person)
WHERE other.name IN ['Amber Heard', 'Elon Musk']
AND (other)-->(p)
RETURN other.name, other.age
```

1.2. Lọc các mẫu dùng NOT

Toán tử NOT có thể được sử dụng để loại trừ một mẫu.

Giá trị tên và tuổi cho các node không có mối quan hệ gửi đi với nút 'Amber Heard' được trả về.

```
MATCH
  (person:Person),
  (q:Person {name: 'Amber Heard'})
WHERE NOT (person)-->(q)
RETURN person.name, person.age
```

1.3. Lọc các mẫu với properties

Ta cũng có thể thêm thuộc tính vào các mẫu của mình

Ex: Tìm tất cả các giá trị tên và tuổi cho các nút có mối quan hệ "IS_FRIEND_WITH" với node 'Elon Musk'

Cypher:

```
MATCH (n:Person)
WHERE (n)-[:IS_FRIEND_WITH]-({name: 'Elon Musk'})
RETURN n.name, n.age
```

2. Lists

2.1. Toán tử IN

Để kiểm tra xem một phần tử có tồn tại trong danh sách hay không, ta có thể sử dụng toán tử IN.

Ex: Truy vấn này cho biết cách kiểm tra xem một thuộc tính có tồn tại trong danh sách theo nghĩa đen hay không.

```
MATCH (a:Person)
```

```
WHERE a.name IN ['Johnny Depp', 'Amber Heard']
```

```
RETURN a.name, a.age
```

This query shows how to check if a property exists in a literal list.

3. Using ranges

3.1. Simple range

Để kiểm tra một phần tử nằm trong một phạm vi cụ thể, hãy sử dụng các toán tử bất đẳng thức <, <=, >=, >.

Giá trị tên và tuổi của các node có thuộc tính tên c lớn hơn hoặc bằng 'Elon Musk' được trả về.

```
MATCH (a:Person)
WHERE a.name >= 'Elon Musk'
RETURN a.name, a.age
DELETE
```

Câu lệnh DELETE được sử dụng để xóa các node, mối quan hệ hoặc đường dẫn.

1. Xoá node đơn

Để xoá một node, dùng câu truy vấn DELETE

```
CREATE (p:Person:Actor {name: 'Johnny Depp', born: 1963, age: 58}), (q:Person:Actor {name: 'Amber Heard', born: 1986, age: 36}), (z:Person {name: 'Elon Musk', born: 1971, age: 50}), (n:Person {name: 'UNKNOWN'})
```

```
MATCH (p:Person:Actor {name: 'Johnny Depp', born: 1963}), (q:Person:Actor {name: 'Amber Heard', born: 1986}), (z:Person {name: 'Elon Musk', born: 1971, age: 50})
```

```
CREATE (p)-[:IS_DIVORCED_WITH]->(q),
       (q)-[:HAVE_AN_AFFAIR_WITH]->(z),
       (z)-[:IS_FRIEND_WITH]->(p),
```

```
(p) - [:IS_FRIEND_WITH] -> (z) ;
MATCH (n:Person {name: 'UNKNOWN'})
DELETE n
```

2. Xoá tất cả các node và các mối quan hệ

Truy vấn này không phải để xóa một lượng lớn dữ liệu, nhưng rất hữu ích khi thử nghiệm với các tập dữ liệu mẫu nhỏ.

Ex cypher:

```
MATCH (n)
DETACH DELETE n
```

3. Xoá 1 node với tất cả các mối quan hệ của nó

Khi bạn muốn xóa một node và bất kỳ mối quan hệ nào đi đến hoặc từ nó đi, hãy sử dụng DETACH DELETE.

Ex: Xoá các mối quan hệ của node 'Amber Heard'

Cypher:

```
MATCH (n {name: 'Amber Heard'})
DETACH DELETE n
```

Cũng có thể chỉ xóa các mối quan hệ, để (các) nút không bị ảnh hưởng.

Truy vấn này sẽ xóa tất cả các mối quan hệ IS_FRIEND_WITH gửi đi khỏi node có tên 'Elon Musk'

```
MATCH (n {name: 'Elon Musk'})-[r:KNOWS]->()
DELETE r
```

3. Pathfinding Graph Search Algorithms

Graph Search algorithms

Giới thiệu

Graph Search algorithm (thuật toán tìm kiếm đồ thị) khám phá một đồ thị để khám phá chung toàn đồ thị hoặc tìm kiếm các nút rõ ràng. Các thuật toán này tạo ra các đường dẫn trong mạng lưới, nhưng không có kỳ vọng rằng các đường dẫn đó là tối ưu về mặt tính toán.

Breadth First Search (tìm kiếm theo chiều rộng) và Depth First Search (tìm kiếm theo chiều sâu) là hai thuật toán cơ bản của các thuật toán tìm kiếm đồ thị. Chúng là những bước quan trọng để duyệt qua một đồ thị và thường là những bước bắt buộc đầu tiên trong nhiều loại phân tích đồ thị khác nhau.

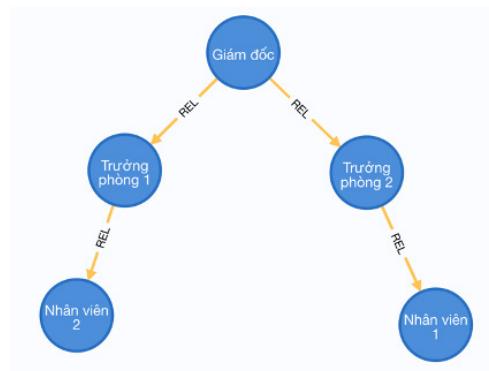
Breadth First Search và Depth First Search

Breadth First Search (BFS) là một trong những thuật toán duyệt đồ thị cơ bản. Nó bắt đầu từ một nút đã chọn và khám phá tất cả các hàng xóm của nó ở cách đó một bước trước khi thăm tất cả các hàng xóm ở cách đó hai bước,...

Tìm kiếm đầu tiên theo chiều sâu (DFS) là một thuật toán duyệt đồ thị cơ bản khác. Nó bắt đầu từ một nút đã chọn, chọn một trong các nút lân cận của nó, sau đó đi càng sâu càng tốt dọc theo con đường đó trước khi quay ngược lại và duyệt tiếp.

Giả sử chúng ta tạo một đồ thị gồm các node Giám đốc, Trưởng phòng 1, Trưởng phòng 2, Nhân viên 1, Nhân viên 2. Thông qua ví dụ này, chúng ta sẽ làm rõ được sự khác biệt trong cách duyệt đồ thị của hai thuật toán BFS và DFS.

```
CREATE (nA:Node {name: 'Giám đốc'}),
       (nB:Node {name: 'Trưởng phòng 1'}),
       (nC:Node {name: 'Trưởng phòng 2'}),
       (nD:Node {name: 'Nhân viên 1'}),
       (nE:Node {name: 'Nhân viên 2'}),
       (nA)-[:REL]->(nB), (nA)-[:REL]->(nC),
       (nB)-[:REL]->(nE), (nC)-[:REL]->(nD);
```

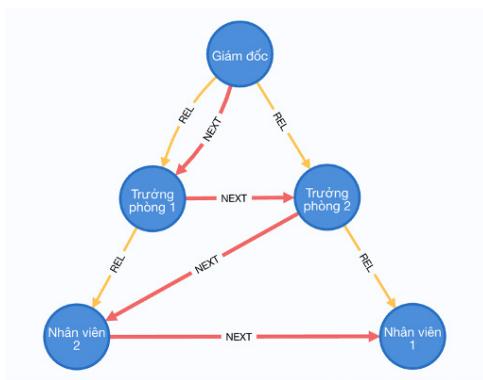


Chạy câu truy vấn trên sẽ cho chúng ta một mạng lưới như trên. Chúng ta tiến hành tạo đồ thị có tên gọi "my-Graph" bằng câu truy vấn dưới đây.

```
CALL gds.graph.create('myGraph', 'Node', 'REL')
```

Chúng ta tiến hành duyệt đồ thị bằng thuật toán BFS:

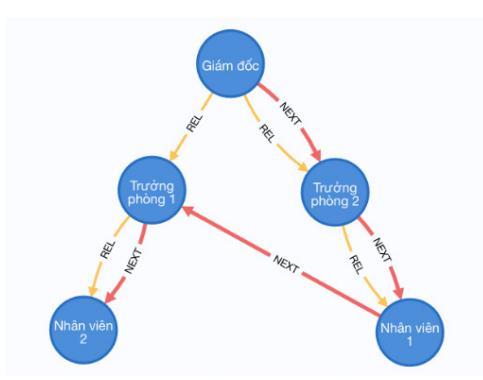
```
MATCH (source:Node{name:'Giám đốc'})
CALL gds.alpha.bfs.stream('myGraph', {
    startNode: id(source)})
YIELD path
RETURN path
```



Hình Duyệt đồ thị bằng Breadth First Search

Thuật toán DFS trong Neo4j:

```
MATCH (source:Node{name:'Giám đốc'})
CALL gds.alpha.dfs.stream('myGraph', {
    startNode: id(source)})
YIELD path
RETURN path
```



Hình: Duyệt đồ thị bằng Depth First Search

Pathfinding algorithms

Giới thiệu

Các thuật toán tìm đường xây dựng dựa trên các thuật toán tìm kiếm đồ thị và khám phá các đường đi giữa các nút, bắt đầu từ một nút và đi qua các mối quan hệ cho đến khi đến đích. Các thuật toán này được sử dụng để xác định các tuyến đường tối ưu thông qua một biểu đồ. Nó cũng được dùng cho các cuộc gọi hoặc định tuyến IP với chi phí thấp nhất và mô phỏng trò chơi.

Loại thuật toán	Mục đích	Ví dụ thực tế
Dijkstra Shortest Path	Tìm đường ngắn nhất giữa nút gốc và nút đích đến	Tìm đường ngắn nhất giữa 2 địa điểm trên bản đồ
Yen's k-Shortest Paths	Tìm các đường ngắn nhất giữa nút gốc và nút đích đến	Tìm các đường ngắn nhất 2 địa điểm trên bản đồ

Bảng: Tổng quan về các thuật toán tìm đường

Dijkstra's Shortest Path algorithm

Giới thiệu

Thuật toán Dijkstra (hay thuật toán Dijkstra's Shortest Path First, thuật toán SPF) là một thuật toán để tìm các đường đi ngắn nhất giữa các nút trong biểu đồ. Thuật toán Dijkstra có thể được sử dụng để xác định đường đi ngắn nhất từ một nút trong biểu đồ đến mọi nút khác trong cùng một cấu trúc dữ liệu đồ thị, miễn là có thể truy cập được các nút từ nút bắt đầu.

Theo nhiều cách, thuật toán Dijkstra là một công thức phức tạp dựa trên hình thức duyệt biểu đồ Breath First Search (tìm kiếm theo chiều rộng) mà chúng ta đã nêu trên. Sự khác biệt chính là thực tế nó thông minh hơn một chút và có thể xử lý các đồ thị có trọng số rất tốt. Tuy nhiên, nếu chúng ta nhìn vào thuật toán của Dijkstra được trực quan hóa, chúng ta sẽ thấy rằng về cơ bản nó hoạt động giống như BFS, trải rộng thay vì theo đuổi sâu một con đường cụ thể.

Google Maps sử dụng thuật toán Dijkstra để tính toán đường đi ngắn nhất giữa hai điểm.

Thuật toán

Thuật toán Dijkstra hoạt động theo các bước sau:

Bước 1: Chọn nút bắt đầu và nút kết thúc và thêm nút bắt đầu vào tập hợp các nút đã ghé thăm có giá trị bằng 0. Các nút đã ghé thăm là tập hợp các nút có đường đi ngắn nhất đã biết từ nút bắt đầu. Nút bắt đầu có giá trị 0 vì nó cách chính nó 0 độ dài đường dẫn.

Bước 2: Duyệt theo chiều rộng từ nút bắt đầu đến các nút lân cận gần nhất của nó và ghi lại độ dài đường dẫn so với mỗi nút lân cận.

Loại thuật toán	Ý tưởng	Ví dụ thực tế
Breadth First Search	Duyệt qua biểu đồ từ nút gốc và khám phá tất cả các nút lân cận	Định vị các nút lân cận trong hệ thống GPS để xác định các địa điểm ưa thích lân cận
Depth First Search	Một thuật toán bắt đầu với nút ban đầu của biểu đồ và sau đó đi sâu hơn xuống các lớp cho đến khi tìm thấy nút cụ thể hoặc nút không có nút con	Phát hiện các chu trình trong một đồ thị.

Bước 3: Chọn đường dẫn ngắn nhất đến một trong những vùng lân cận này và đánh dấu nút đó là đã được ghé thăm. Trong trường hợp độ dài các đường dẫn tiếp theo bằng nhau, thuật toán Dijkstra chọn ngẫu nhiên.

Bước 4: Ghé thăm các nút lân cận gần nhất với tập hợp các nút đã ghé thăm và ghi lại độ dài đường dẫn từ nút bắt đầu so với các nút lân cận mới này. Không truy cập vào bất kỳ nút lân cận nào đã được ghé thăm, vì chúng ta đã biết các đường dẫn ngắn nhất đến chúng.

Bước 5: Lặp lại các bước 3 và 4 cho đến khi nút kết thúc được đánh dấu là đã ghé thăm.

Một lưu ý khi sử dụng thuật toán Dijkstra là thuật toán này không hỗ trợ đồ thị có trọng số âm. Nếu trong số là giá trị âm thì thuật toán không còn chính xác nữa.

Một số ứng dụng của thuật toán tìm đường ngắn nhất

Tìm chỉ đường giữa các địa điểm: Các công cụ lập bản đồ web như Google Maps sử dụng thuật toán Dijkstra hoặc một biến thể gần giống để chỉ đường cho người dùng.



Hình: Google Maps sử dụng thuật toán Dijkstra để tính toán đường đi ngắn nhất và chỉ đường cho người dùng

Tìm mức độ tách biệt giữa mọi người trong mạng xã hội: Giả sử khi bạn xem hồ sơ của ai đó trên LinkedIn, nó sẽ cho biết có bao nhiêu người tách bạn ra trong biểu đồ, cũng như liệt kê các mối quan hệ chung của bạn.

Yen's k-Shortest Paths algorithm

Giới thiệu

Thuật toán k-Shortest Path của Yen tương tự như thuật toán Dijkstra, nhưng thay vì chỉ tìm đường đi ngắn nhất giữa hai cặp nút, nó còn tính toán đường đi ngắn nhất thứ hai, đường đi ngắn nhất thứ ba,... lên đến $k-1$ độ lệch của những con đường ngắn nhất.

Jin Y. Yen đã phát minh ra thuật toán vào năm 1971 và mô tả nó trong "Tìm K đường dẫn không vòng ngắn nhất

trong mạng". Thuật toán này hữu ích để tìm các đường đi thay thế thay vì chỉ tìm kiếm đường đi ngắn nhất. Nó trở nên đặc biệt hữu ích khi chúng ta cần nhiều hơn một kế hoạch dự phòng.

Thuật toán

Như đã đề cập ở trên, thuật toán k-Shortest Path của Yen sử dụng chính thuật toán Dijkstra. Thuật toán này đảm bảo rằng một đường đi ngắn nhất đã được phát hiện sẽ không được duyệt lại.

Đối với $k = 1$, thuật toán hoạt động giống hệt như thuật toán đường đi ngắn nhất của Dijkstra và trả về đường đi ngắn nhất. Với $k = 2$, thuật toán trả về đường đi ngắn nhất và đường đi ngắn thứ hai giữa cùng một nút bắt đầu và nút kết thúc. Nói chung, với $k = n$, thuật toán tính toán nhiều nhất n đường đi được phát hiện theo thứ tự tổng chi phí của chúng.

Bài toán tìm đường ngắn nhất giữa hai địa điểm

Mục tiêu bài toán

Các thuật toán Dijkstra và Yen's có ứng dụng rất lớn trong các bài toán tìm đường ngắn nhất giữa hai điểm. Ở phần này, chúng ta đặt ra một bài toán thực tế để tìm đường đi ngắn nhất và các đường đi ngắn nhất từ "Times City" tới "Vincom Bà Triệu" ở thành phố Hà Nội. Hai thuật toán trên sẽ được áp dụng thông qua công cụ Neo4j.

Giới thiệu tập dữ liệu

Tập dữ liệu bao gồm hai file "transport_node.csv" và "transport_relationship.csv". File "transport_node.csv" chứa dữ liệu về các nút là các địa điểm cụ thể nối với nhau và nối liền hai địa điểm từ "Times City" tới "Vincom Bà Triệu". File "transport_relationship.csv" chứa thông tin về khoảng cách giữa các nút là các địa điểm đã nêu trên. Lưu ý bài toán đặt trong giả thuyết lý tưởng là chỉ có các nút và các đường đi trên giữa 2 địa điểm này.

Tên trường dữ liệu	Ý nghĩa trường dữ liệu	Kiểu dữ liệu (điều kiện)
place	Tên địa điểm làm mốc	String (không trùng lặp)
street	Tên phố	String

Bảng: Các trường dữ liệu, ý nghĩa và kiểu dữ liệu file "transport_node.csv"

Tên trường dữ liệu	Ý nghĩa trường dữ liệu	Kiểu dữ liệu (điều kiện)
source	Tên địa điểm đầu	String
destination	Tên địa điểm đích	String
street	Tên phố nằm giữa hai địa điểm	String
distance	Khoảng cách giữa hai địa điểm	Integer

Bảng: Các trường dữ liệu, ý nghĩa và kiểu dữ liệu file “transport_relationship.csv”

Biểu diễn dữ liệu dưới dạng đồ thị

Chúng ta tiến hành import các nút từ file “transport_node.csv” vào Neo4j. Có 13 địa điểm được thêm vào từ dữ liệu.

```
WITH 'https://raw.githubusercontent.com/
leslien10/Essay-Dataset/main/transport_node.csv'
AS url

LOAD CSV WITH HEADERS FROM url AS row

MERGE (p:Place {id:row.place, street:row.street})
```



Tiếp đến, chúng ta sẽ tạo các liên kết và truyền khoảng cách giữa các địa điểm vào các liên kết đó. Tạo một đồ thị có tên “transport_graph” để chuẩn bị áp dụng các thuật toán tìm đường ngắn nhất.

```
WITH 'https://raw.githubusercontent.com/
leslien10/Essay-Dataset/main/transport_relationship.csv'
AS url

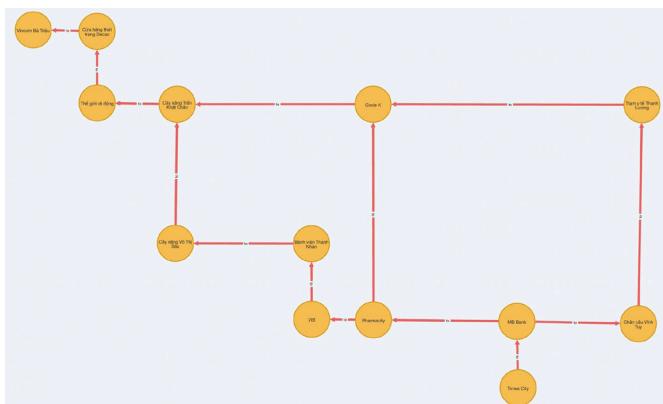
LOAD CSV WITH HEADERS FROM url AS row

MATCH (source:Place {id: row.source})
MATCH (destination:Place {id: row.destination})
MERGE (source)-[:to {cost: toInteger(row.distance), street:row.street}]->(destination);
```

```
CALL gds.graph.create(
    'transport_graph',
    'Place',
    { to: {orientation: "NATURAL"} },
    {relationshipProperties: 'cost'})
```

Tìm đường ngắn nhất với Dijkstra

Trước khi áp dụng thuật toán Dijkstra, chúng ta sẽ làm rõ cấu trúc của thuật toán trong Neo4j.



Bây giờ chúng ta tiến hành áp dụng thuật toán shortestPath.dijkstra để tìm đường đi ngắn nhất giữa Times City đến Vincom Bà Triệu. Kết quả trả ra của thuật toán này chính là danh sách các nút theo thứ tự và khoảng cách cộng dồn khi đi qua từng nút. Chúng ta gán địa điểm xuất phát và đích đến là các nút source và destination. Tham số relationshipWeightProperty sẽ được gán là “cost” vì trọng số của chúng ta chính là khoảng cách giữa các nút hay các địa điểm.

```
MATCH (source:Place {id: "Times City"}), (destination:Place {id: "Vincom Bà Triệu"})
CALL gds.shortestPath.dijkstra.stream(
    'transport_graph',
    {
        sourceNode: source,
        targetNode: destination,
        relationshipWeightProperty: 'cost'
    }
)
YIELD nodeIds, costs
WITH [nodeId in nodeIds|gds.util.asNode(nodeId).id] AS nodeName, costs AS costs, size(nodeIds) AS size
UNWIND range(0,siz-1) AS n
RETURN nodeName[n] AS place, costs[n] AS distance
```

Hình: Kết quả thuật toán Dijkstra cho bài toán tìm đường ngắn nhất

Tổng độ dài quãng đường ngắn nhất giữa hai điểm được tính ra là 4220m tương ứng với 4,22km. Thời gian chạy hết câu lệnh là 46ms. Tuyến đường ngắn nhất được mô

Place	Distance
1. Times City	0.0
2. MB Bank	450.0
3. Pharmacy	1050.0
4. VIB	1170.0
5. Bệnh viện Thanh Nhàn	1870.0
6. Cây xăng Võ Thị Sáu	2570.0
7. Cây xăng Trần Khát Chân	3320.0
8. Thế giới di động	3720.0
9. Cửa hàng thời trang DECAO	4070.0
10. Vincom Bà Triệu	4220.0

tả dưới hình sau.

Tìm các đường ngắn nhất với Yen's k shortest path

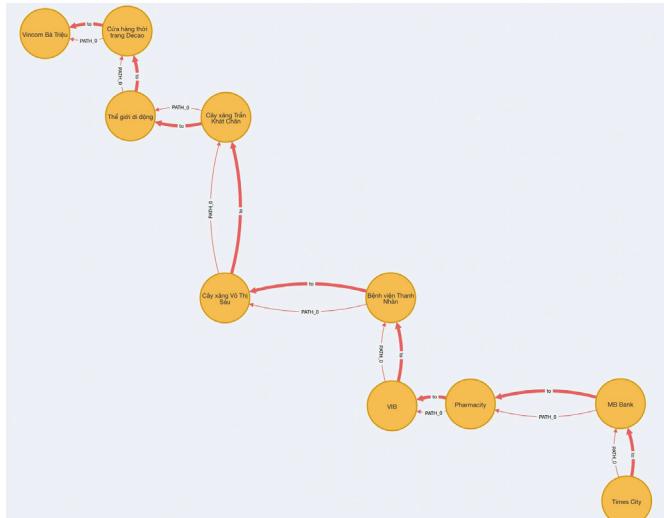
Tương tự, trước khi áp dụng thuật toán Yen's k shortest path, chúng ta sẽ làm rõ cấu trúc của thuật toán này trong Neo4j.

Tiến hành áp dụng thuật toán Yen's k shortest path để tìm các đường đi ngắn nhất giữa hai điểm chúng ta cần. Lưu ý ở đây, tham số k chúng ta lựa chọn sẽ là số đường đi ngắn nhất.

```
CALL gds.graph.create(
  'transport_graph_yen',
  'Place',
  'to',
  {
    relationshipProperties: 'cost'
  }
);
```

```
MATCH (source:Place {id: 'Times City'}), (target:Place {id: 'Vincom Bà Triệu'})
CALL gds.shortestPath.yens.stream('transport_
graph_yen', {
  sourceNode: source,
  targetNode: target,
  k: 3,
  relationshipWeightProperty: 'cost'
})
YIELD index, sourceNode, targetNode, totalCost,
nodeIds, costs, path
RETURN
index,
gds.util.asNode(sourceNode).id AS source,
gds.util.asNode(targetNode).id AS destination,
totalCost,
[nodeId IN nodeIds | gds.util.asNode(nodeId).id]
AS nodeNames,
costs,
nodes(path) as path
ORDER BY index;
```

Bảng: Kết quả thuật toán Yen's k shortest path cho bài toán tìm đường ngắn nhất



Vì chúng ta đặt $k = 3$ nên thuật toán này trả ra kết quả là 3 đường đi ngắn nhất từ Times City tới Vincom Bà Triệu. Độ dài 3 đường lần lượt là 4220m, 4850m và 4850m. Có thể dễ dàng nhận thấy đường đi ngắn nhất của thuật toán này chính là kết quả trả ra của thuật toán Dijkstra. Thời gian chạy hết câu lệnh là 82ms.

Nhận xét và kết luận

Có thể nhận thấy rằng thuật toán Dijkstra được sử dụng trong các bài toán một đường đi ngắn nhất giữa hai nút. Trong khi đó, thuật toán của Yen sẽ thích hợp cho các bài toán khi ta cần nhiều phương án hơn là một phương án duy nhất. Chính vì thế, thời gian thực hiện của thuật toán Yen cũng mất nhiều hơn so với thuật toán Dijkstra.

4. Centrality Algorithm PageRank

Centrality algorithms

Giới thiệu

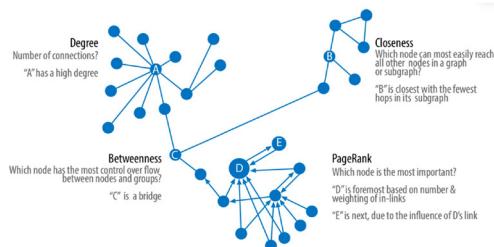
Centrality algorithms (thuật toán trung tâm) được sử dụng để phân tích các vai trò của các node (nút) cụ thể trong một đồ thị và tác động của chúng tới toàn mạng lưới. Các thuật toán trung tâm trở nên hữu ích vì nó xác định được các nút quan trọng nhất và giúp chúng ta hiểu được credibility (độ tin cậy), accessibility (khả năng tiếp cận), the speed at which things spread (độ lan truyền) và bridges between groups (cầu nối giữa các nhóm). Mặc dù nhiều thuật toán trong số này được phát minh để phân tích mạng xã hội, nhưng kể từ khi ra đời, chúng cũng được sử dụng trong rất nhiều ngành và lĩnh vực

khác nhau.

Bảng: Tổng quan về các thuật toán tập trung

Loại thuật toán	Mục đích	Ví dụ thực tế
Degree Centrality	Đo lường số relationship (mối quan hệ) của một node (nút)	Ước tính mức độ nổi tiếng của một người
Closeness Centrality	Tính toán nút nào có đường đi ngắn nhất tới tất cả các nút khác	Tìm kiếm địa điểm tối ưu nhất cho các dịch vụ công cộng mới nhằm tối đa khả năng truy cập
Betweenness Centrality	Đo lường số đường đi ngắn nhất qua một nút	Cải thiện việc xác định thuộc đặc trị nhỏ tìm ra các gen kiểm soát các chứng bệnh cụ thể
PageRank (được phổ biến bởi Google)	Ước lượng tầm quan trọng của mỗi nút từ những nút liên kết tới nó và các nút liên kết của những nút ấy	Dự đoán lưu lượng giao thông, chuyển động của con người cho mang lưới giao thông và không gian đô thị

Hình: Các thuật toán tập trung và câu hỏi mà chúng trả lời



Trong nghiên cứu này, chúng ta chỉ tập trung phân tích thuật toán PageRank với cơ sở toán học và các ứng dụng thực tế của chúng.

PageRank algorithm

Giới thiệu

PageRank (PR) là thuật toán được biết đến nhiều nhất trong các loại thuật toán trung tâm. Google đã tạo ra PageRank để giải quyết vấn đề mà họ gặp phải với công cụ tìm kiếm của mình. Chính vì thế, PageRank đã được đặt theo tên của Larry Page, một trong những người sáng lập Google. Mục đích của thuật toán này là để xếp hạng các trang web trong kết quả công cụ tìm kiếm của họ. Khi đưa ra một truy vấn tìm kiếm từ người dùng, họ có thể ngay lập tức tìm thấy một tập hợp lớn các trang web chứa các từ hoặc nội dung gần như giống như người dùng đã nhập.

"PageRank hoạt động bằng cách đếm số lượng và chất lượng của các liên kết đến một trang để xác định ước tính sơ bộ về mức độ quan trọng của trang web. Giả định cơ bản là các trang web quan trọng hơn có khả năng nhận được nhiều liên kết hơn từ các trang web khác." – Google

Hình: Cách thuật toán PageRank hoạt động (mô hình đơn giản)

How PageRank Works (A Simplified View)

PageRank is divided equally between the total number of links on a page.



Ví dụ, tưởng tượng bạn là một người bạn của Lionel Messi – một trong những cầu thủ vĩ đại nhất thế giới. Khả năng rất cao rằng mức độ ảnh hưởng của chính bạn cũng sẽ tăng lên. Người ta đặt ra giả thuyết có một vài người bạn rất quyền lực có thể khiến bạn có ảnh hưởng hơn là có nhiều người bạn ít quyền lực hơn. PageRank được tính toán bằng cách phân phối lặp đi lặp lại thứ hạng của một nút trên các nút lân cận hoặc bằng cách duyệt ngẫu nhiên biểu đồ và đếm tần suất mỗi nút được truy cập trong những lần duyệt này.

Thuật toán PageRank

PageRank được định nghĩa trong tài liệu của Google như sau:

$$PR(u) = (1 - d) + d \left(\frac{PR(T1)}{C(T1)} + \dots + \frac{PR(Tn)}{C(Tn)} \right)$$

Trong đó:

Chúng ta giả sử rằng trang A có các trang từ T1 đến Tn trỏ đến nó.

PR(...): PageRank hay tầm quan trọng của nút đó

d: damping factor (hệ số giảm xóc) được đặt trong khoảng từ 0 tới 1 và thường được đặt mặc định là 0,85. Bạn có thể hiểu chúng như là xác suất mà một người xem sẽ tiếp tục trỏ tới các trang khác. Điều này giúp giảm thiểu rank sink (chìm thứ hạng).

1 - d: là xác suất mà một nút được tiếp cận trực tiếp mà không tuân theo bất kỳ mối quan hệ nào.

C(T...): out-degree (số lượng liên kết bắt đầu tại nút T) của nút T

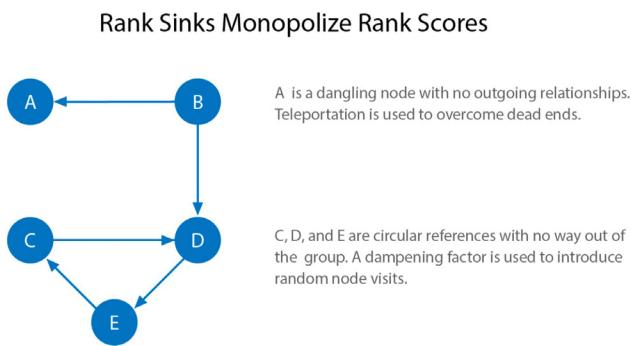
Iteration, Random Surfers và Rank Sinks

PageRank là một thuật toán lặp lại cho đến khi điểm số hội tụ hoặc cho đến khi đạt đến một số lần lặp lại (iteration) nhất định.

Về mặt khái niệm, PageRank giả định rằng có một người lướt web truy cập các trang bằng cách nhấp vào các liên kết hoặc bằng cách sử dụng một URL ngẫu nhiên. Damping factor (hệ số giảm xóc) xác suất một người

dùng ngẫu nhiên tiếp tục nhấp vào các liên kết khi họ duyệt web. Điều này được cho là sẽ giảm với mỗi lần nhấp vào liên kết.

Một nút hoặc một nhóm các nút không có outgoing relationships (các mối quan hệ ra ngoài) có thể độc quyền điểm PageRank bằng cách từ chối chia sẻ. Đây chính là khái niệm Rank Sink (thứ hạng chìm). Bạn có thể tưởng tượng tình huống này giống như một người lướt web bị mắc kẹt ở một trang hoặc một tập hợp con gồm các trang, không có lối ra. Một khó khăn khác xảy ra khi các nút chỉ trỏ đến nhau trong một nhóm. Điều này làm tăng thứ hạng của chúng khi người truy cập lướt qua lại giữa các trang này. Tình huống này được miêu tả trong hình dưới đây.



Hình: Xếp hạng chìm là do một nút hoặc một nhóm nút, không có mối quan hệ ra ngoài gây ra.

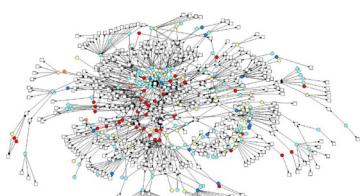
Có 2 chiến lược để tránh Rank Sink (thứ hạng chìm):

Thứ nhất, PageRank sẽ giả định tất cả các nút đều có outgoing relationships (các mối quan hệ ra ngoài), dù cho có các nút không có mối quan hệ này. Việc di chuyển qua các liên kết vô hình này đôi khi được gọi là teleportation (dịch chuyển tức thời).

Thứ hai, hệ số giảm xóc tránh thứ hạng chìm bằng cách đưa ra xác suất cho liên kết trực tiếp so với lượt truy cập các nút ngẫu nhiên. Khi bạn đặt α thành 0,85, một nút hoàn toàn ngẫu nhiên được truy cập 15% thời gian.

Một số ứng dụng của PageRank

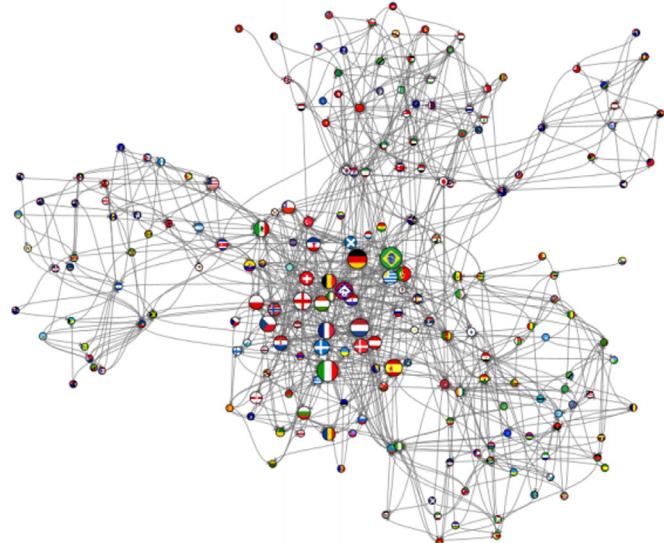
Tìm hiểu mức độ kết nối của một người trên Mạng xã hội: Một trong những lãnh thổ chưa được khám phá trong phân tích mạng xã hội là thông tin toàn mạng. Sử dụng thông tin toàn mạng này, chúng ta có thể ước tính mức độ ảnh hưởng của từng người dùng. Có thể dễ dàng phân tích mạng bằng thuật toán PageRank.



Hình: Một ví dụ minh họa về một mạng lưới của mạng xã hội

Đường và Không gian đô thị: Một ứng dụng đáng ngạc nhiên khác của PageRank là dành cho mạng lưới đường và không gian đô thị, nơi nó giúp dự đoán cả lưu lượng giao thông và chuyển động của con người. Mạng lưới đường bộ sử dụng một cấu trúc thú vị được gọi là biểu đồ đường tự nhiên. Đó là một con đường liên tục được xây dựng từ các đoạn đường bằng cách nối các đoạn liền kề với nhau nếu góc đầu nhỏ và không có giải pháp thay thế nào tốt hơn.

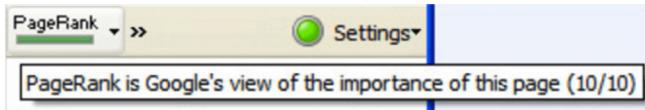
Xếp hạng trong thể thao: Ma trận ngẫu nhiên và phương pháp xếp hạng eigenvector không phải thứ gì quá mới trong lĩnh vực xếp hạng thể thao. Một trong những cấu trúc mạng tự nhiên cho thể thao là mạng chiến thắng, trong đó mỗi đội là một nút và nút i trỏ đến nút j nếu j thắng trong trận đấu giữa i và j. Các mạng này thường được tính theo tỷ số bởi đội j thắng đội i. Govan, Meyer và Albright (2008) đã sử dụng PageRank với hệ số giảm xóc = 0,85 để xếp hạng các đội bóng đá có các mạng chiến thắng này.



Hình: Các cường quốc bóng đá như Brazil, Đức, Ý, Tây Ban Nha có điểm PageRank cao nhất toàn mạng

Tại sao Google lựa chọn loại bỏ thanh công cụ PageRank?

Như đã nói ở trên, thuật toán PageRank là một trong những thuật toán được quan tâm và nổi tiếng nhất trong các thuật toán trung tâm. Chính vì sức mạnh và độ phổ biến của nó, những người làm SEO trở nên bị ám ảnh bởi PageRank và nó nhanh chóng trở thành tâm điểm của các chiến thuật SEO, thậm chí vượt lên cả việc tạo ra nội dung tuyệt vời và tối ưu trải nghiệm người dùng.



Hình: Google từng công khai xếp hạng PageRank

SEOs biết cách để có thể sử dụng PageRank xếp hạng trang web của họ cao hơn và họ đã tận dụng điều này. Nếu chúng ta xem xét vấn đề này từ góc độ của Google, thanh công cụ PageRank công khai chính là khởi nguồn vấn đề. Nếu không có nó, sẽ không có thước đo chính xác về độ quan trọng của một trang web. SEOs đã lạm dụng PageRank và sử dụng nó để thao túng thứ hạng các trang web. Cuối cùng, Google đã không có lựa chọn thực sự nào khác ngoài việc gỡ bỏ thanh công cụ PageRank vào năm 2016.

Bài toán xếp hạng tay vợt Tennis xuất sắc nhất (2010 - 2019)

Mục tiêu bài toán

Như đã nêu, Govan, Meyer và Albright vào năm 2008 đã sử dụng PageRank với hệ số giảm xóc = 0,85 để xếp hạng các đội bóng đá dựa vào số lần chiến thắng của họ trên toàn mạng. Ở bài toán này, mục tiêu của chúng ta là dựa vào thông tin toàn bộ các trận đấu ATP trong vòng 10 năm, từ năm 2010 đến năm 2019 cùng với danh tính tay vợt thắng và thua để xác định xếp hạng PageRank cho toàn bộ các tay vợt chuyên nghiệp. Tay vợt có xếp hạng PageRank cao nhất được cho là tay vợt thi đấu xuất sắc nhất trong giai đoạn này. Chúng ta sẽ sử dụng công cụ Neo4j để phân tích và áp dụng thuật toán PageRank.

Giới thiệu tập dữ liệu

Tập dữ liệu “match_scores_2010-2019.csv” được Kevin Lin tổng hợp từ kết quả tất cả các trận đấu ATP trên trang chủ chính thức của ATP World Tour – Hiệp hội Quần vợt chuyên nghiệp thế giới.

2018		Tournament	Draw	Surface	Total Financial Commitment	Winner	Results	Go
Tournament								
	Brisbane Brisbane, Australia 2017.12.31	SGL 28 DBL 16	Outdoor Hard	\$528,910	SGL: Nick Kyrgios DBL: Henri Kontinen, John Peers		RESULTS	
	Doha Doha, Qatar 2018.01.01	SGL 32 DBL 16	Outdoor Hard	\$1,386,665	SGL: Gael Monfils DBL: Oliver Marach, Mate Pavic		RESULTS	
	Pune Pune, India 2018.01.01	SGL 28 DBL 16	Outdoor Hard	\$561,345	SGL: Gilles Simon DBL: Robin Haase, Matwe Middelkoop		RESULTS	
	Sydney Sydney, Australia 2018.01.07	SGL 28 DBL 16	Outdoor Hard	\$528,910	SGL: Daniil Medvedev DBL: Lukasz Kubot, Marcelo Melo		RESULTS	
	Auckland Auckland, New Zealand 2018.01.08	SGL 28 DBL 16	Outdoor Hard	\$561,345	SGL: Roberto Bautista Agut DBL: Oliver Marach, Mate Pavic		RESULTS	
	Australian Open Melbourne, Australia 2018.01.15	SGL 128 DBL 64	Outdoor Hard	AS\$25,096,000	SGL: Roger Federer DBL: Oliver Marach, Mate Pavic		RESULTS	

Hình: Dữ liệu các giải đấu và trận đấu trên trang chính thức của ATP World Tour

Tập dữ liệu này bao gồm 36 trường và 42827 quan sát. Mỗi quan sát là thông tin của một trận đấu Tennis cụ thể với 36 thuộc tính. Do tính chất của bài toán xếp hạng PageRank mà chúng ta đang thực hiện, chúng ta sẽ chỉ sử dụng 8 trường dữ liệu được liệt kê dưới bảng sau.

Tên trường dữ liệu	Ý nghĩa trường dữ liệu	Kiểu dữ liệu (điều kiện)
tourney_year_id	ID giải đấu	String (không trùng lặp)
tourney_name	Tên giải đấu	String
tourney_round_name	Tên vòng đấu	String
winner_name	Tên tay vợt giành chiến thắng	String
winner_player_id	ID tay vợt giành chiến thắng	String
loser_name	Tên tay vợt thua trận	String
loser_player_id	ID tay vợt thua trận	String
match_id	ID trận đấu	String (không trùng lặp)

Bảng: Các trường dữ liệu, ý nghĩa và kiểu dữ liệu

Biểu diễn dữ liệu dưới dạng đồ thị

Đồ thị của chúng ta bao gồm hai loại nút chính: Match và Player. Trước khi import dữ liệu vào Neo4j, cần phải tạo một số ràng buộc để chúng ta không vô tình tạo ra các nút là các trận đấu và tay vợt trùng lặp.

```
CREATE CONSTRAINT FOR (p:Player) REQUIRE p.id IS UNIQUE;
CREATE CONSTRAINT FOR (m:Match) REQUIRE m.id IS UNIQUE;
```

Sau khi đã thiết lập ràng buộc cho các nút, chúng ta tiến hành import dữ liệu vào Neo4j. Đầu tiên, chúng ta truyền dữ liệu các trận đấu vào các nút Match. Các thuộc tính trận đấu bao gồm ID trận đấu, tên vòng đấu, ID tay vợt thắng và thua trận, ID giải đấu, tên giải đấu và năm tổ chức. Một lưu ý ở đây là cần trích xuất năm từ chính ID của giải đấu. VD: tourney_year_id = “2010-339” ® year = “2010”.

```

WITH
'https://raw.githubusercontent.com/leslien10/Essay-Dataset/main/match_scores_2010-2019.csv' AS url
LOAD CSV WITH HEADERS FROM url AS row
MERGE (m:Match {id: row.match_id,
    name: row.tourney_round_name,
    winner: row.winner_player_id,
    loser: row.loser_player_id,
    tour_id: row.tourney_year_id,
    tour_name: row.tourney_name,
    year: toInteger(split(row.tourney_year_id, "-")[0])})

```

Tiếp đến, chúng ta truyền dữ liệu các tay vợt vào các nút Player. Thông tin của các tay vợt chính là thông tin trích xuất người thắng và thua từ các trận đấu. Bất cứ Player nào đã được import trước đó sẽ không tạo thêm các nút mới vì ID của mỗi Player là duy nhất.

```

WITH
'https://raw.githubusercontent.com/leslien10/Essay-Dataset/main/match_scores_2010-2019.csv' AS url
LOAD CSV WITH HEADERS FROM url AS row
MERGE (winner:Player{name: row.winner_name, id: row.winner_player_id})
MERGE (loser:Player{name: row.loser_name, id: row.loser_player_id})

```

Có 42827 trận đấu khác nhau và 2125 tay vợt tham gia thi đấu ở các trận đấu này.

Sau khi tạo các nút Match và Player, chúng ta tiến hành tạo các liên kết giữa các nút này. Mỗi trận đấu sẽ trả tới một người thắng và một người thua bằng các liên kết WINNER và LOSER. Một tay vợt có thể được trả tới từ một hoặc rất nhiều các trận đấu khác nhau.

```

MATCH (m:Match), (p:Player)
WHERE m.winner = p.id
MERGE (m)-[:WINNER]->(p);
MATCH (m:Match), (p:Player)
WHERE m.loser = p.id
MERGE (m)-[:LOSER]->(p);

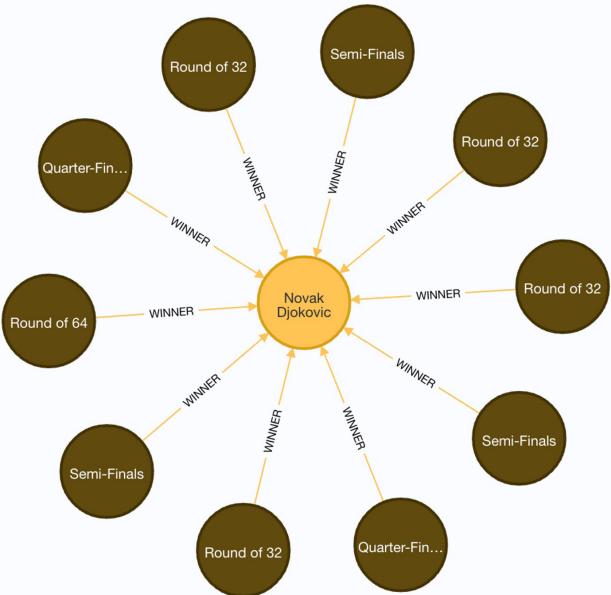
```

Thực hiện câu truy vấn dưới đây để biểu diễn 10 trận thắng của Novak Djokovic.

```

MATCH a=(m:Match)-[:WINNER]->(p:Player)
WHERE p.name="Novak Djokovic"
RETURN a
LIMIT 10

```



Thực hiện câu truy vấn dưới đây để tính số trận đã chơi của Dominic Thiem trong năm 2018. Kết quả trả ra 71 trận.

```

MATCH a=(m:Match)-[]->(p:Player)
WHERE p.name="Dominic Thiem"
AND m.year=2018
RETURN COUNT (a)

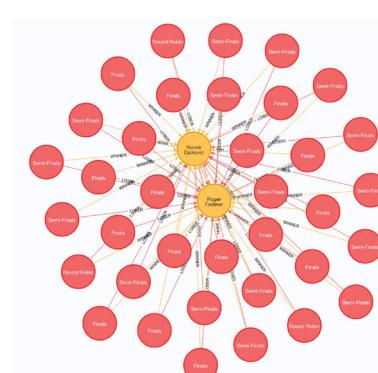
```

Trong giai đoạn 2010-2019, Novak Djokovic và Roger Federer đã đối đầu với nhau 36 lần. Có thể nhận thấy phần lớn các trận đối đầu giữa hai tay vợt này thường ở vòng bán kết và chung kết của các giải đấu. Điều này cũng chứng tỏ đây là hai tay vợt rất xuất sắc của giai đoạn này.

```

MATCH a=(p1)<-[]-()-[]-(p2)
WHERE p1.name="Novak Djokovic" AND p2.name =
"Roger Federer"
RETURN COUNT(a)

```



Những tay vợt giành nhiều chiến thắng nhất

Trước tiên, chúng ta tiến hành xếp hạng các tay vợt theo tỉ lệ phần trăm giành chiến thắng trong các trận đấu ở các giải đấu mà họ tham gia. Thực hiện câu truy vấn bên dưới sẽ cho ra top 10 tay vợt có tỉ lệ phần trăm giành chiến thắng cao nhất.

```
MATCH (p:Player)
WITH p,
    size((p)<-[:WINNER]-()) AS wins,
    size((p)<-[:LOSER]-()) AS defeats
RETURN p.name, wins, defeats,
CASE WHEN wins+defeats = 0 THEN 0
ELSE (wins * 100.0) / (wins + defeats) END
AS percentageWins
ORDER BY wins DESC
LIMIT 10
```

name	score
"Novak Djokovic"	43.994348273991484
"Rafael Nadal"	35.6866678714673
"Roger Federer"	35.500003086773866
"Andy Murray"	26.0437633838526
"Tomas Berdych"	20.22046305165025
"David Ferrer"	19.106808463615266
"Stan Wawrinka"	18.836110285726797
"Kei Nishikori"	18.449955484340858
"Jo-Wilfried Tsonga"	17.66566566801716
"Marin Cilic"	16.636721525112957

Bảng: Top 10 tay vợt có tỉ lệ phần trăm giành chiến thắng cao nhất

Không bất ngờ khi top 10 của chúng ta chính là những gương mặt vô cùng nổi tiếng trong làng quần vợt thế giới. Đầu tiên là Novak Djokovic – tay vợt người Serbia đã 5 lần kết thúc năm ở vị trí số 1 thế giới (tính tới thời điểm cuối năm 2019). Hai cái tên ở vị trí số 2 và 3 cũng là hai huyền thoại của làng quần vợt: Rafael Nadal – ông vua mặt sân đất nện và Roger Federer – tàu tốc hành Thụy Sĩ.

Tiếp đến, chúng ta sẽ áp dụng thuật toán PageRank để xếp hạng các tay vợt, so sánh kết quả bảng xếp hạng có những thay đổi nào.

PageRank và tay vợt xuất sắc nhất

Cách mà thuật toán PageRank hoạt động chính là việc các nút nhận được credibility (độ tin cậy) từ những mối quan hệ trỏ đến chúng. Ví dụ, trong biểu đồ các trang web, một trang web tạo uy tín cho trang khác bằng cách liên kết đến nó. Mức độ tin cậy mà nó mang lại có thể được xác định bởi weight property (trọng số) của mối quan hệ đó.

Trong biểu đồ quần vợt của chúng ta, độ tin cậy giữa các tay vợt có thể được mô hình hóa dựa trên số trận thắng mà họ có được trong các lần đối đầu với nhau. Ví dụ, truy vấn sau đây tìm xem Federer và Nadal đã đánh bại nhau bao nhiêu lần:

```
MATCH (p1:Player {name: "Roger Federer"}),
(p2:Player {name: "Rafael Nadal"})
RETURN p1.name, p2.name,
size((p1)<-[:WINNER]-()-[:LOSER]->(p2)) AS p1Wins,
size((p1)<-[:LOSER]-()-[:WINNER]->(p2)) AS p2Wins
```

p1.name	p2.name	p1Wins	p2Wins
"Roger Federer"	"Rafael Nadal"	10	11

Đồ thị của chúng ta nên có mối quan hệ giữa Federer và Nadal với trọng số bằng số lần họ đánh bại nhau. Trọng số của mối quan hệ từ Federer đến Nadal sẽ là 11 vì Federer đang mang lại cho Nadal sự tin tưởng rất nhiều cho 11 trận thắng đó. Tương tự, trọng số của mối quan hệ từ Nadal đến Federer sẽ là 10. Câu truy vấn dưới đây sẽ gán số lần thắng thành trọng số của các mối quan hệ.

```
MATCH (p1)<-[:WINNER]-(:match)-[:LOSER]->(p2)
WHERE p1.name IN ["Roger Federer", "Rafael Nadal"]
AND p2.name IN ["Roger Federer", "Rafael Nadal"]
RETURN p2.name AS source, p1.name AS target,
count(*) AS weight
```

source	target	weight
"Rafael Nadal"	"Roger Federer"	10
"Roger Federer"	"Rafael Nadal"	11

Chúng ta tiến hành tạo một đồ thị với tên gọi “ATP-graph_2010-2019” biểu thị mối quan hệ và trọng số của các mối quan hệ giữa các tay vợt như đã làm với ví dụ trên.

```
CALL gds.graph.create.cypher(
  'ATP-graph_2010-2019',
  'MATCH (n) RETURN id(n) AS id',
  'MATCH (p1)<-[:WINNER]-(:Match)-[:LOSER]->(p2)
  RETURN id(p2) AS source,
  id(p1) AS target, COUNT(*) AS weight'
)
```

Bây giờ chúng ta đã sẵn sàng để áp dụng thuật toán PageRank vào đồ thị trên. Thuật toán PageRank trong Neo4j được cấu trúc như sau:

Như thông tin từ bảng trên, PageRank trong Neo4j được mặc định không có trọng số. Ở bài toán này chúng ta sử dụng trọng số để xếp hạng PageRank, chính vì thế thuộc tính relationshipWeightProperty sẽ được gán là 'weight'. Hệ số giảm xóc được gán là 0.85 – thường được sử dụng. Thông số maxIteration cũng được gán mặc định là 20.

Chúng ta tiến hành áp dụng thuật toán PageRank cho đồ thị đã tạo bên trên.

```
CALL gds.pageRank.stream('ATP-graph_2010-2019', {
    maxIterations: 20,
    dampingFactor: 0.85,
    relationshipWeightProperty: 'weight'
})
YIELD nodeId, score
RETURN gds.util.asNode(nodeId).name AS name,
score
ORDER BY score DESC, name ASC;
```

	name	score
1	"Novak Djokovic"	43.994348273991484
2	"Rafael Nadal"	35.6866678714673
3	"Roger Federer"	35.500003086773866
4	"Andy Murray"	26.0437633838526
5	"Tomas Berdych"	20.22046305165025
6	"David Ferrer"	19.106808463615266
7	"Stan Wawrinka"	18.836110285726797
8	"Kei Nishikori"	18.440955484340858
9	"Jo-Wilfried Tsonga"	17.66566566801716
10	"Marin Cilic"	16.636721525112957

Bảng: Xếp hạng các tay vợt xuất sắc nhất giai đoạn 2010 – 2019 theo PageRank

Không bất ngờ khi vị trí số 1 chính là tay vợt người Serbia Novak Djokovic. Những tay vợt trong top 10 vẫn là những gương mặt quen thuộc với chúng ta. Chúng ta cũng có thể tiến hành xếp hạng các tay vợt theo từng năm theo câu lệnh dưới đây.

```
CALL gds.graph.create.cypher(
    'ATP-graph-2018',
    'MATCH (n) RETURN id(n) AS id',
    'MATCH (p1)-[:WINNER]->(Match)-[:LOSER]->(p2)'
    WHERE Match.year = 2018

    RETURN id(p2) AS source, id(p1) AS target,
    count(*) as weight'
);

CALL gds.pageRank.stream('ATP-graph-2018', {
    maxIterations: 20,
    dampingFactor: 0.85,
    relationshipWeightProperty: 'weight'
})
YIELD nodeId, score

RETURN gds.util.asNode(nodeId).name AS name,
score
ORDER BY score DESC, name ASC;
```

	name	score
1	"Novak Djokovic"	11.636491251046172
2	"Dominic Thiem"	9.205249357664956
3	"Juan Martin del Potro"	9.016499179936172
4	"Alexander Zverev"	8.651114654389229
5	"Rafael Nadal"	8.465505674232949
6	"Stefanos Tsitsipas"	7.534977307092328
7	"Marin Cilic"	7.31541216194673
8	"Roger Federer"	6.914717560012362
9	"Kevin Anderson"	6.528234731607003
10	"Kei Nishikori"	6.4525003027129095

Bảng: Xếp hạng các tay vợt xuất sắc nhất năm 2018 theo PageRank

Rank ^	+/- Rank ^	Player ^	Age ^	Points ^	Points ^	Tourn Played ^	Dropping ^	Next Best ^
1	-	Novak Djokovic	31	9,045	-	17	0	0
2	-	Rafael Nadal	32	7,480	-	13	0	0
3	-	Roger Federer	37	6,420	-	17	0	0
4	-	Alexander Zverev	21	6,385	-	21	0	0
5	-	Juan Martin del Potro	30	5,300	-	18	0	0
6	-	Kevin Anderson	32	4,710	-	20	0	0
7	-	Marin Cilic	30	4,250	-	19	0	0
8	-	Dominic Thiem	25	4,095	-	25	0	0
9	-	Kei Nishikori	29	3,590	-	24	0	0
10	-	John Isner	33	3,155	-	23	0	0

Bảng: Xếp hạng chính thức ATP World Tour cuối năm 2018

Nhận xét và kết luận

Mặc dù so sánh xếp hạng PageRank và xếp hạng ATP World Tour chính thức là không hoàn toàn giống nhau, tuy nhiên phần lớn các gương mặt trong top 10 đều có mặt trong hai bảng xếp hạng, và sự khác biệt là không quá lớn. Có thể lý do dẫn đến sự khác biệt cũng tới từ việc đánh trọng số. Trên thực tế, trọng số của các chiến thắng trong các giải đấu là khác nhau, các giải ATP trên thế giới có số điểm khác nhau. Hơn nữa, xét từ góc độ

năng lực, việc giành chiến thắng trước tay vợt top 10 thế giới so với việc giành chiến thắng trước các tay vợt top 1000 thế giới phải được đánh trọng số khác nhau. Chính vì thế, chúng ta có thể phát triển và hoàn thiện bài toán này hơn bằng cách nghiên cứu thêm về việc đánh trọng số cho các mối quan hệ giữa các tay vợt.

Từ ứng dụng về bài toán xếp hạng tay vợt Tennis, chúng ta nhận thấy rằng thuật toán PageRank có thể được sử dụng trong rất nhiều lĩnh vực và đạt được những kết quả bất ngờ. Trả lời cho câu hỏi Google có còn sử dụng PageRank, câu trả lời là có. Google vẫn sử dụng PageRank trong các thuật toán của họ. Cho đến nay, thuật toán này vẫn là một thuật toán vô cùng quan trọng trong nhiều lĩnh vực và ngành nghề.

Leonie Mann @leoniejmann · Feb 24, 2020
Replying to @JohnMu and @sayan29031995
But you use page rank..

johnmu.xml (personal) @JohnMu
Yes, we do use PageRank internally, among many, many other signals. It's not quite the same as the original paper, there are lots of quirks (eg, disavowed links, ignored links, etc.), and, again, we use a lot of other signals that can be much stronger.
1:47 AM · Feb 25, 2020

1 26 Reply Copy link

Hình: Một tweet của John Mueller - Nhà phân tích xu hướng quản trị trang web cấp cao tại Google khẳng định chắc chắn rằng PageRank vẫn được sử dụng như một tín hiệu xếp hạng của họ.

5. Community Detection Louvain Algorithm

Community Detection

Giới thiệu

Việc hình thành cộng đồng là phổ biến trong tất cả các loại kết nối trong mạng lưới và việc xác định chúng là điều cần thiết để đánh giá hành vi của nhóm và các hiện tượng mới nổi. Nguyên tắc chung trong việc tìm kiếm cộng đồng là các thành viên của nó sẽ có nhiều mối quan hệ trong nhóm hơn là với các nút bên ngoài nhóm của họ. Việc xác định các tập hợp liên quan này cho thấy các cụm nút, nhóm cô lập và cấu trúc mạng. Thông tin này giúp suy ra hành vi hoặc sở thích tương tự của các nhóm ngang hàng, ước tính khả năng phục hồi, tìm các mối quan hệ liên quan với nhau và chuẩn bị dữ liệu cho các phân tích khác.

Loại thuật toán	Mục đích	Ví dụ thực tế
Triangle Count and Clustering Coefficient	Đo lường số lượng nút tạo thành hình tam giác và mức độ mà các nút có xu hướng tập hợp lại với nhau	Ước tính độ ôn định của nhóm và liệu mạng lưới có biểu hiện "thế giới nhỏ" được thấy trong biểu đồ với các cụm đơn xen chất chẽ hay không
Strongly Connected Components	Tìm các nhóm trong đó mỗi nút có thể truy cập được từ mọi nút khác trong cùng nhóm đó, bất kể hướng của mối quan hệ	Đưa ra các đề xuất sản phẩm dựa trên nhóm liên kết hoặc các mặt hàng tương tự
Connected Components	Tìm các nhóm trong đó mỗi nút có thể truy cập được từ mọi nút khác trong cùng nhóm đó, bất kể hướng của mối quan hệ	Thực hiện phân nhánh nhanh cho các thuật toán khác và xác định các thuật toán biệt lập
Label Propagation	Tìm ra các cụm bằng cách trải rộng các nhãn dựa trên phần lớn các vùng lân cận	Tìm ra được sự đồng thuận trong cộng đồng xã hội hoặc tìm kiếm sự kết hợp nguy hiểm của các loại thuốc được kê đơn chung
Louvain Modularity	Tối đa hóa độ chính xác giả định của các nhóm bằng cách so sánh trọng số và mật độ mối quan hệ với một ước tính hoặc trung bình	Trong phân tích gian lận, đánh giá xem một nhóm có một số các hành vi xấu rời rạc hoặc đang hoạt động như một vòng lừa đảo

Bảng: Tổng quan về các thuật toán phát hiện cộng đồng

Trong nghiên cứu này, chúng ta chỉ tập trung phân tích thuật toán Louvain Modularity với cơ sở toán học và các ứng dụng thực tế của chúng.

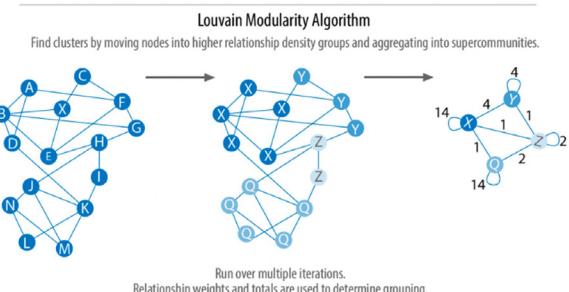
Louvain algorithm

Giới thiệu

Louvain là một thuật toán unsupervised algorithm (không giám sát) dùng để phát hiện các cộng đồng trong các mạng lớn. Nó không yêu cầu đầu vào về số lượng cộng đồng cũng như kích thước của chúng. Trước khi thực hiện nó được chia thành 2 giai đoạn: Tối ưu hóa mô đun và Tổng hợp cộng đồng.

Nguyên tắc của nó là tối đa hóa điểm môđun cho mỗi cộng đồng, trong đó môđun định lượng chất lượng của việc gán các nút cho cộng đồng. Điều này có nghĩa là đánh giá mức độ kết nối của các nút trong cộng đồng với mật độ cao hơn như thế nào so với mức độ kết nối của chúng trong một mạng ngẫu nhiên và sau đó thu được phân vùng cộng đồng tối ưu.

Thuật toán Louvain là một trong những thuật toán dựa trên môđun nhanh nhất và hoạt động tốt với các đồ thị lớn. Nó cũng tiết lộ một hệ thống phân cấp của các cộng đồng ở các quy mô khác nhau, điều này rất hữu ích cho việc hiểu hoạt động toàn cầu của một mạng.

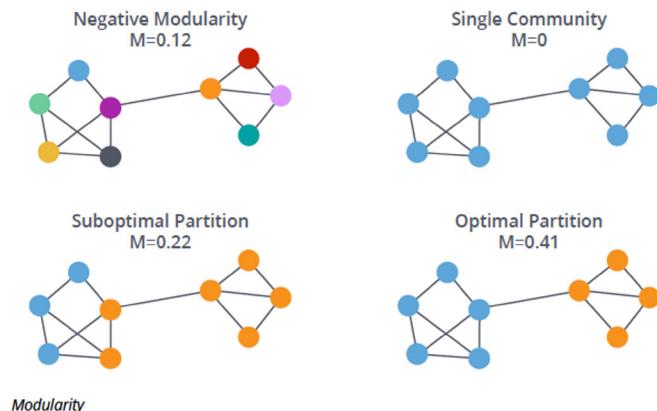


Hình Mô hình thuật toán Louvain Modularity

Thuật toán

Bất kỳ thuật toán học máy unsupervised (không giám sát) hoặc supervised (có giám sát) đều cần có hàm loss/optimization/cost function để quyết định tiêu chí hội tụ. Trong giả thiết phát hiện cộng đồng, Mô-đun là một trong những lựa chọn tối ưu hóa phù hợp nhất. Thuật toán của Louvain dựa trên việc tối ưu hóa Mô-đun rất hiệu quả.

Để hiểu được thuật toán mô đun Louvain, trước tiên chúng ta phải có cái nhìn tổng quan về mô đun nói chung.

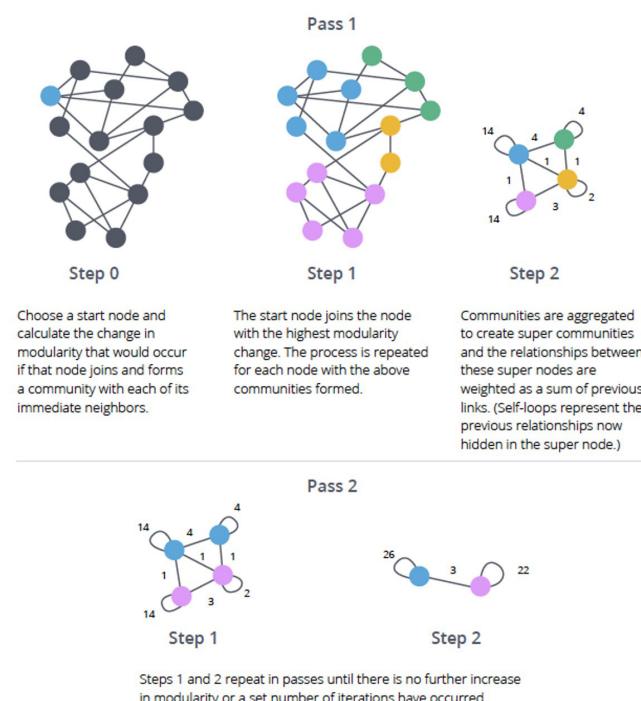


Mô-đun là thước đo mức độ tốt của các nhóm đã được phân chia thành các cụm. Nó so sánh các mối quan hệ trong một cụm so với những gì sẽ được mong đợi đối với một số lượng kết nối ngẫu nhiên (hoặc đường cơ sở khác).

Một phân vùng lý tưởng của cộng đồng phải có các đặc điểm sau:

(1) Nhiều cạnh hơn trong cộng đồng.

(2) Ít cạnh hơn giữa các cộng đồng.



Louvain Modularity Algorithm

Chất lượng của các cộng đồng được gọi là phân vùng sau đây được đo bằng Mô-đun phân vùng. Mô-đun Q được định nghĩa theo công thức hiển thị trong hình dưới đây:

$$Q = \frac{1}{2m} \sum_{i,j} \left[A_{ij} - \frac{k_i k_j}{2m} \right] \delta(c_i, c_j),$$

where

A_{ij} is the weight of the edge between i and j.
k_i is the sum of weights of the vertex attached to the vertex I, also called as degree of the node
c_i is the community to which vertex i is assigned
δ(x,y) is 1 if x = y and 0 otherwise
m = $(1/2)\sum_{i,j} A_{ij}$ i.e number of links

Để tối đa hóa mô đun, thuật toán Louvain có hai giai đoạn lặp lại. Giai đoạn đầu chỉ định mỗi nút trong mạng cho cộng đồng của chính nó. Sau đó, nó cố gắng tối đa hóa lợi ích mô-đun bằng cách hợp nhất các cộng đồng lại với nhau. Nút i có khả năng sẽ được hợp nhất với nút j (hàng xóm của i) để xác định xem có sự gia tăng tích cực trong mô-đun hay không. Nếu không có sự gia tăng theo mô-đun thì nút vẫn ở trong cộng đồng hiện tại của nó, nếu không thì các cộng đồng sẽ được hợp nhất. Sự hợp nhất các cộng đồng này có thể được biểu diễn bằng công thức sau:

$$\Delta Q = \left[\frac{\Sigma_{in} + 2k_{i,in}}{2m} - \left(\frac{\Sigma_{tot} + k_i}{2m} \right)^2 \right] - \left[\frac{\Sigma_{in}}{2m} - \left(\frac{\Sigma_{tot}}{2m} \right)^2 - \left(\frac{k_i}{2m} \right)^2 \right]$$

Trong đó:

Sigma_in là tổng của tất cả các trọng số của các liên kết bên trong cộng đồng mà i đang chuyển đến

Sigma_tot là tổng của tất cả các trọng số của các liên kết đến các nút trong cộng đồng mà i đang di chuyển vào

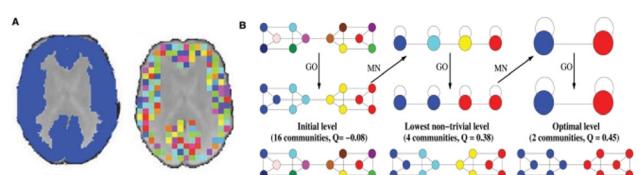
k_i là mức độ trọng số của i

k_i,in là tổng trọng số của các liên kết giữa i và các nút khác trong cộng đồng mà i đang di chuyển vào

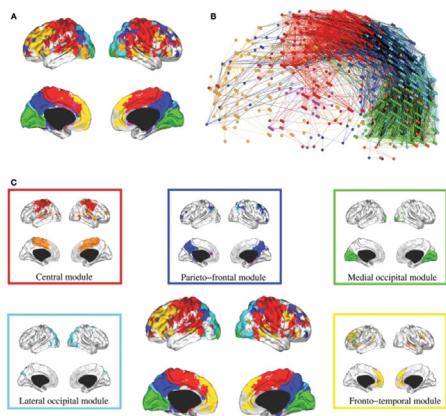
m là tổng trọng số của tất cả các liên kết trong mạng.

Một số ứng dụng của Louvain

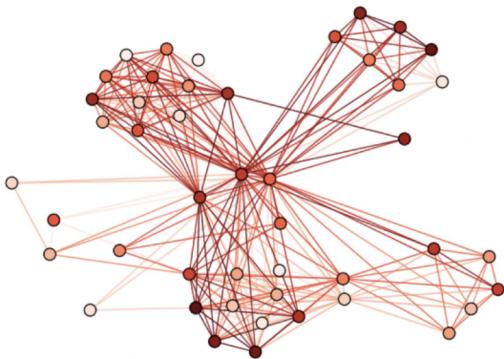
Thuật toán Louvain đã được sử dụng để nghiên cứu, tìm hiểu hoạt động não người. Và tìm ra các cấu trúc cộng đồng có thứ bậc trong mạng lưới chức năng của não.



Hình: Mô hình phân cấp của mạng chức năng não người.



Thuật toán Louvain đã được dùng nhằm để xuất và cung cấp các khuyến nghị, gợi ý cho người dùng Reddit. Với mục đích tìm các subreddits tương tự dựa trên hành vi chung của chính họ.



Hình: Trục quan hóa đồ thị con của biểu đồ Reddit mẫu thu được sau khi chạy thuật toán phân vùng.

Thuật toán Louvain đã được sử dụng để trích xuất các chủ đề từ các nền tảng xã hội trực tuyến, chẳng hạn như Twitter và YouTube. Dựa trên biểu đồ đồng thời cùng xuất hiện của các thuật ngữ như một phần của quá trình tạo ra mô hình chính.

Bài toán phát hiện cộng đồng với tập dữ liệu KEGG_DISEASE

Mục tiêu bài toán

COVID-19 đã chiếm lĩnh thế giới bởi những “sợi tóc ngắn” của mình. Sự bùng nổ của đại dịch COVID-19 đã đặt ra vô vàn thách thức cho lĩnh vực y tế trong việc nghiên cứu thuốc đặc trị, vaccine và mầm bệnh của các loại bệnh mới. Nó làm tan vỡ nhiều gia đình, gây thiệt hại to lớn về kinh tế, xã hội và có lẽ sẽ thay đổi hành vi của chúng ta mãi mãi. Do đó, việc cấp bách chính là nâng cao nhận thức của chúng ta về vấn đề sức khỏe cộng đồng. Rõ ràng là loài người chúng ta cần đầu tư nhiều hơn vào nghiên cứu y học để ngăn chặn những thảm họa tiếp

theo.

Trong bài toán này, chúng ta sẽ sử dụng tập dữ liệu Kegg_Disease để tìm ra những thông tin thú vị về các mầm bệnh, chứng bệnh, thuốc và các mối liên hệ giữa chúng. Việc tìm ra những loại mầm bệnh gây ra nhiều chứng bệnh khác nhau, các loại thuốc đa dụng chữa được nhiều chứng bệnh khác nhau giúp tăng hiểu biết của chúng ta về mạng lưới các mầm bệnh và giảm bớt chi phí khi phát triển một loại thuốc mới hoàn toàn. Đặc biệt, trong bài toán này chúng ta sẽ sử dụng thuật toán Louvain – một loại thuật toán phát hiện cộng đồng để phát hiện các cộng động mầm bệnh, bệnh và thuốc chữa.

Giới thiệu tập dữ liệu

Tập dữ liệu trong bài toán này được Sixing Huang tổng hợp từ trang “genome.jp/kegg/disease/”.

Biểu diễn dữ liệu dưới dạng đồ thị

Đầu tiên, chúng ta import dữ liệu vào Neo4j. Có ba loại nút chính là disease (bệnh), drug (thuốc) và pathogen (mầm bệnh). Cân tạo ràng buộc rằng “ko” của ba loại nút này là không trùng lặp và là duy nhất.

```
CREATE CONSTRAINT ON (n:disease) ASSERT n.ko IS UNIQUE;
```

```
CREATE CONSTRAINT ON (n:drug) ASSERT n.ko IS UNIQUE;
```

```
CREATE CONSTRAINT ON (n:pathogen) ASSERT n.ko IS UNIQUE;
```

Tiếp theo chúng ta truyền dữ liệu các loại bệnh, thuốc và mầm bệnh vào các nút từ tập dữ liệu. Sau đó tiến hành tạo các liên kết mầm bệnh gây nên bệnh (causes) và thuốc chữa bệnh (treats).

```
LOAD CSV WITH HEADERS FROM 'https://raw.githubusercontent.com/leslien10/Essay-Dataset/main/disease.csv' AS row
```

```
MERGE (n:disease {name: row.name, ko: row.ko, description: row.description, disease_category:row.disease_category});
```

```
LOAD CSV WITH HEADERS FROM 'https://raw.githubusercontent.com/leslien10/Essay-Dataset/main/drug.csv' AS row
```

```
MERGE (n:drug {name: row.name, ko: row.ko});
```

```
LOAD CSV WITH HEADERS FROM 'https://raw.githubusercontent.com/leslien10/Essay-Dataset/main/pathogen.csv' AS row
```

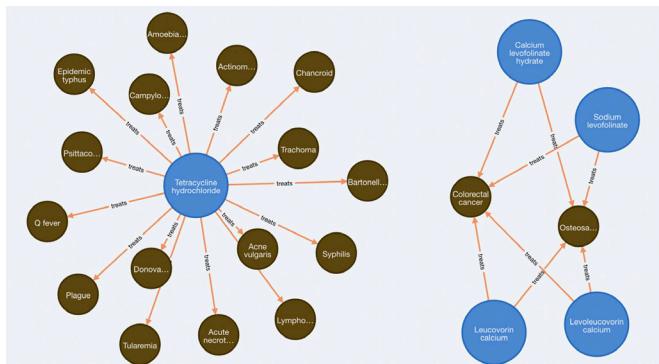
```
MERGE (n:pathogen {name: row.name, ko: row.ko, taxonomy: row.taxonomy});
```

LOAD CSV WITH HEADERS FROM ‘https://raw.githubusercontent.com/leslien10/Essay-Dataset/main/drug_disease.csv’ AS row

MERGE (n1:drug {ko: row.from})

MERGE (n2:disease {ko: row.to})

MERGE (n1)-[r:treats]->(n2);

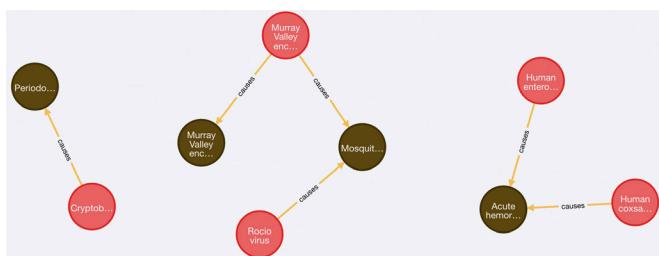


LOAD CSV WITH HEADERS FROM ‘https://raw.githubusercontent.com/leslien10/Essay-Dataset/main/pathogen_disease.csv’ AS row

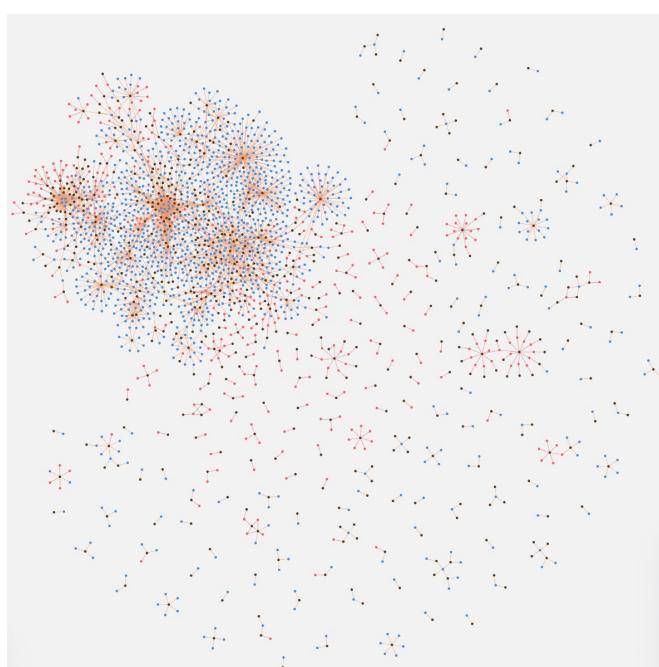
MERGE (n1:pathogen {ko: row.from})

MERGE (n2:disease {ko: row.to})

MERGE (n1)-[r:causes]->(n2);



Có 333 mầm bệnh, 2498 loại bệnh và 1339 thuốc điều trị trong đồ thị của chúng ta. Đây là một mạng lưới với độ lớn lý tưởng để tiến hành các phân tích sâu hơn. Chúng ta cũng có thể quan sát mạng lưới thông qua cấu trúc liên kết của nó.



Hình: Cấu trúc liên kết đồ thị từ dữ liệu KEGG_DISEASE. Nút màu đỏ thể hiện mầm bệnh, màu nâu thể hiện loại bệnh và màu xanh thể hiện thuốc chữa trị.

Chúng ta tiến hành kiểm tra 10 danh mục bệnh nhiều nhất trong tập dữ liệu bằng câu lệnh dưới.

MATCH (d:disease)

RETURN d.disease_category, COUNT(d.disease_category) as count

ORDER BY count DESC LIMIT 10;

d.disease_category	count
“Congenital malformation”	647
“Infectious disease”	345
“Nervous system disease”	234
“Inherited metabolic disease”	124
“Cancer”	97
“Congenital disorder of metabolism”	94
“Hematologic disease”	63
“Immune system disease”	56
“Cardiovascular disease”	54
“Nervous system disease; Musculoskeletal disease”	52

Khác với dự đoán của chúng ta, các “băng đảng” khét tiếng nhất không phải là các nhóm bệnh ung thư, bệnh truyền nhiễm hay bệnh tim mạch mà đứng đầu là nhóm các dị tật bẩm sinh, bao gồm hội chứng móng và xương bánh chè và hội chứng Meckel.

Các bệnh truyền nhiễm có thể do bacteria (vi khuẩn), viruses (vi rút) hoặc eukaryota (sinh vật nhân thực) gây ra. Câu lệnh sau sẽ hiển thị số lượng từng loại trong tập dữ liệu.

MATCH (p:pathogen)

RETURN split(p.taxonomy, “; ”)[0] as domain, COUNT(split(p.taxonomy, “; ”)[0]) as count

ORDER BY count DESC;

domain	count
“Bacteria”	147
“Viruses”	132
“Eukaryota”	54

Có thể nhận thấy vi rút là mầm bệnh chiếm số lượng nhiều nhất trong các loại mầm bệnh trong tập dữ liệu, theo sau là vi rút và ít hơn hẳn là các mầm bệnh từ sinh vật nhân thực. Chúng ta sẽ phân tích sâu hơn vào nhóm các mầm bệnh từ vi rút.

MATCH (p:pathogen)

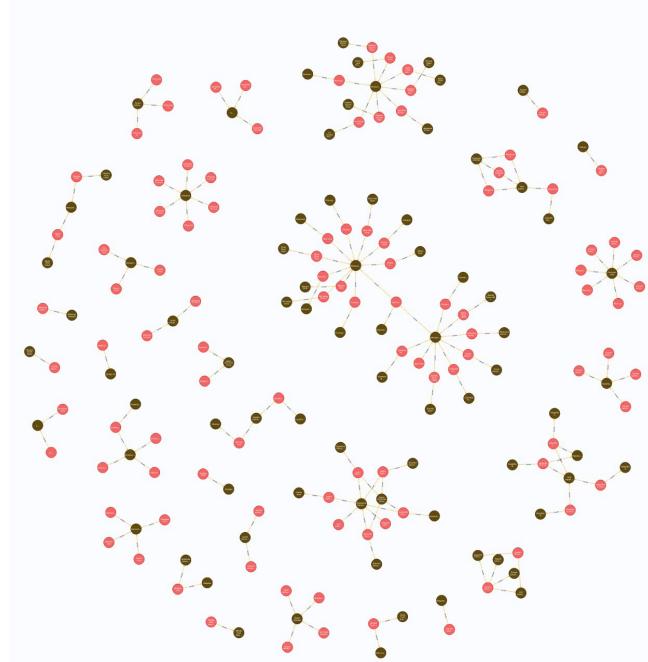
WHERE p.taxonomy contains “virus”

```
RETURN split(p.taxonomy, "; ")[0] + ";" +
+ split(p.taxonomy, "; ")[1] as domain,
COUNT(split(p.taxonomy, "; ")[0] + ";" +
split(p.taxonomy, "; ")[1]) as count
```

ORDER BY count DESC;

domain	count
"Viruses; Riboviria"	104
"Viruses; Duplodnaviria"	9
"Viruses; Varidnaviria"	9
"Viruses; Monodnaviria"	9
"Viruses; Deltavirus"	1

Như bạn có thể thấy, Riboviria là vùng virus phong phú nhất. Cả SARS-CoV-2 hay COVID-19 và SARS coronavirus tiền thân của nó, tác nhân gây ra đợt bùng phát SARS 2002–2004, đều thuộc về chủng vi rút Riboviria. Các vi rút suy giảm miễn dịch ở người (HIV) và vi rút viêm gan cũng vậy.



Cuối cùng, có thể lấy ra được số lượng thuốc chống lại các bệnh truyền nhiễm bằng câu truy vấn sau:

MATCH (dr:drug) -[]->(di:disease)

WHERE di.disease_category = “Infectious disease”

RETURN COUNT(DISTINCT(dr));

COUNT(DISTINCT(dr))
293

Như kết quả trên, thấy rằng hiện nay có 293 loại thuốc để chữa các bệnh truyền nhiễm. Cần lưu ý rằng từ khóa “DISTINCT” là cần thiết ở đây, vì một số loại thuốc có thể

được sử dụng để điều trị nhiều bệnh và chúng sẽ được tính hai lần nếu không có “DISTINCT”.

Tương tự bạn thực hiện lấy số lượng thuốc của bệnh ung thư như sau:

MATCH (dr:drug) -[]->(di:disease)

WHERE di.disease_category = “Cancer”

RETURN COUNT(DISTINCT(dr));

COUNT(DISTINCT(dr))
271

Câu truy vấn trên cho ra 271 loại thuốc điều trị bệnh ung thư.

Chúng ta tiếp tục tìm kiếm các loại thuốc đa dụng. Thuốc đa dụng là loại thuốc có thể được sử dụng để điều trị nhiều loại bệnh. Nói cách khác, chúng rất linh hoạt. Bởi vì để phát triển loại thuốc mới rất đắt và đòi hỏi các thử nghiệm lâm sàng, vì vậy nếu “một loại thuốc cũ có thể tiếp tục nghiên cứu chữa trị bệnh mới”, chắc chắn đó là một tin tuyệt vời cho cả bệnh nhân và các công ty dược phẩm. Trên thực tế, các loại thuốc được quản lý cẩn thận bởi các cơ quan chức năng như FDA ở Hoa Kỳ và các nhà sản xuất chỉ có thể tiếp thị chúng theo những chỉ định đã được FDA chấp thuận. Vì vậy, sẽ rất thú vị khi tìm ra những loại thuốc linh hoạt nhất trong dữ liệu và những dấu hiệu của chúng.

Thực hiện câu truy vấn sau để tìm ra các loại thuốc đa dụng:

MATCH (dr:drug) -[r:treats]->(di:disease)

RETURN dr.name as drug, COUNT(r) as count ORDER BY count DESC LIMIT 10;

drug	count
"Prednisolone sodium phosphate"	37
"Prednisone"	32
"Dexamethasone"	26
"Triamcinolone acetonide"	26
"Doxycycline hydrate"	26
"Doxycycline"	25

Danh sách top 10 về cơ bản bao gồm hai nhóm thuốc: hormone steroid và kháng sinh tetracycline doxycycline. Hormone steroid được sử dụng để ngăn chặn hệ thống miễn dịch và điều trị một loạt các tình trạng viêm nhiễm. Thuốc thứ ba trong danh sách, Dexamethasone, đã được nghiên cứu trên bệnh nhân COVID-19 và kết quả sơ bộ cho thấy nó làm giảm tỷ lệ tử vong ở những người được thở máy xâm nhập hoặc thở oxy đơn thuần nhưng không làm giảm tỷ lệ tử vong ở những người không được hỗ trợ hô hấp (The RECOVERY Collaborative Nhóm, năm 2020). Doxycycline là một loại kháng sinh phổ rộng và

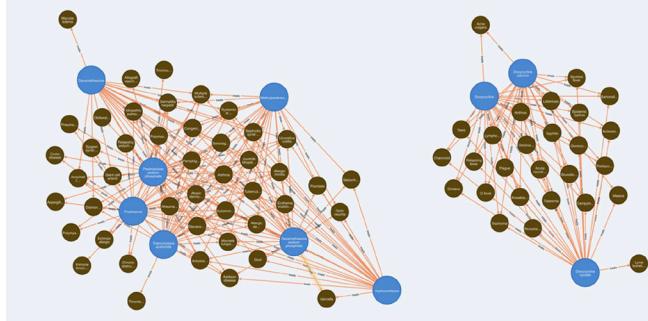
được sử dụng trong điều trị các bệnh truyền nhiễm như sốt rét, bệnh Lyme, bệnh tả và bệnh giang mai. Trên thực tế, truy vấn sau đây cho thấy rằng các chỉ định cho các hormone steroid này chồng chéo lên nhau rất nhiều, trong khi doxycycline và các dẫn xuất của nó cũng nhầm vào nhiều bệnh giống nhau.

Chúng ta cùng tìm hiểu top 10 loại thuốc đặc trị trên dành cho các loại bệnh nào với câu truy vấn dưới đây.

```

MATCH (a:drug)-[r:treats]->(b:disease)
WITH a, COUNT(r) AS indicationCount
    ORDER BY indicationCount DESC LIMIT 10
MATCH (a:drug)-[r:treats]->(b:disease)
RETURN (a) -[r] -> (b)

```



Hình: Top 10 loại thuốc đa dụng hàng đầu và các bệnh mà nó chữa trị

Ví dụ, hai loại thuốc hàng đầu trong danh sách: prednisone và prednisolone dạng hoạt động của nó, đã được chấp thuận sử dụng trong y tế ở Hoa Kỳ từ năm 1955. Chúng phổ biến 21 và là loại thuốc được kê toa nhiều thứ 134 ở Hoa Kỳ từ năm 2021 cho đến nay theo Cơ sở dữ liệu của DrugStats. Thực hiện lần lượt hai lệnh này để xem chi tiết về prednisolone natri phosphate:

```

MATCH (dr:drug) -[]->(di:disease)
WHERE dr.name="Prednisolone sodium phosphate"
RETURN di.disease_category, count(di.disease_cat-
egory) as count
ORDER BY count DESC LIMIT 10;

```

di.disease_category	count
"Immune system disease"	10
"Infectious disease"	4
"Musculoskeletal disease"	3
"Skin and connective tissue disease"	2
"Hematologic disease; Immune system disease"	2
"Lung disease"	2
"Skin and connective tissue disease; Immune system disease"	1
"Urinary system disease"	1
"Endocrine and metabolic diseases"	1
"Respiratory disease"	1

Hình: Top 10 loại bệnh mà thuốc Prednisolone chữa trị nhiều nhất

MATCH (dr:drug) -[]->(di:disease)

```

WHERE dr.name="Prednisolone sodium phosphate" AND
(di.disease_category = "Lung disease" or di.dis-
ease_category="Infectious disease")

```

```

RETURN di.name, di.disease_category
```

```

ORDER BY di.disease_category;
```

di.name	di.disease_category
"Trichinosis"	"Infectious disease"
"Pneumocystis pneumonia"	"Infectious disease"
"Tuberculosis"	"Infectious disease"
"Aspergillosis"	"Infectious disease"
"Chronic obstructive pulmonary disease (COPD)"	"Lung disease"
"Idiopathic pulmonary fibrosis"	"Lung disease"

Hình: Các bệnh truyền nhiễm và bệnh phổi thuốc Prednisolone đặc trị.

Hóa ra, prednisolone không chỉ được sử dụng để điều trị bệnh hệ thống miễn dịch mà còn cả các bệnh truyền nhiễm như bệnh giun xoắn và bệnh lao. Nó cũng được sử dụng trong bệnh phổi tắc nghẽn mãn tính (COPD) vì tác dụng chống viêm.

Tiếp đến, chúng ta cùng phân tích top 10 mầm bệnh gây ra nhiều bệnh nhất trong tập dữ liệu.

```

MATCH (p:pathogen) -[r:causes]->(di:disease)
```

```

RETURN p.name, COUNT(r) as count ORDER BY count
DESC LIMIT 10;

```

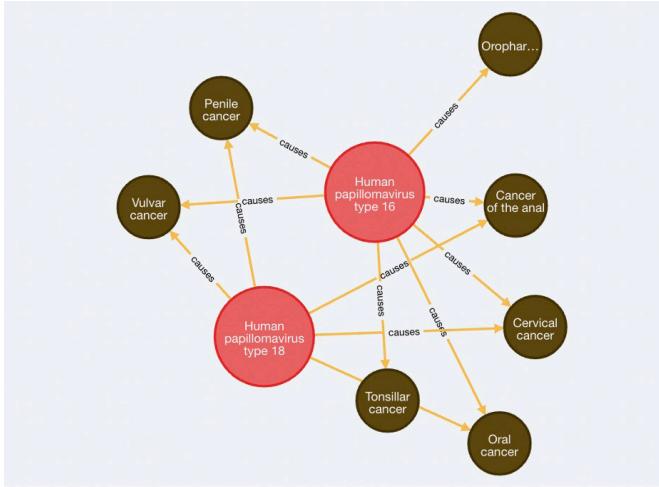
p.name	count
"Human papillomavirus type 16"	7
"Human papillomavirus type 18"	5
"Human herpesvirus 4 (Epstein-Barr virus)"	4
"Human immunodeficiency virus 1 (HIV-1)"	4
"Machupo mammarenavirus"	3
"Hepatitis B virus (HBV)"	3
"Sabin mammarenavirus"	3
"Guarantito mammarenavirus"	3
"Oropouche virus"	3
"Hepatitis C virus (HCV)"	3

Tất cả 10 mầm bệnh hàng đầu đều là virus. Hai loại hàng đầu là papillomavirus có thể dẫn đến các loại ung thư ở người. Theo Wikipedia, hầu hết các trường hợp nhiễm vi rút papillomavirus đều không có triệu chứng và 90% trường hợp tự khỏi trong vòng hai năm. Nhưng trong những trường hợp khác, chúng vẫn tồn tại và dẫn đến mụn cóc hoặc tổn thương. Những tổn thương này làm tăng nguy cơ ung thư dọc theo các vùng khác nhau trong cơ thể con người như cổ tử cung, âm hộ, âm đạo, dương vật, hậu môn, miệng, amidan hoặc họng. Gần như tất cả ung thư cổ tử cung là do HPV gây ra và hai chủng hàng đầu này chiếm 70% các trường hợp. Chỉ riêng HPV16 là

nguyên nhân gây ra gần 90% trường hợp ung thư hầu họng dương tính với HPV.

```

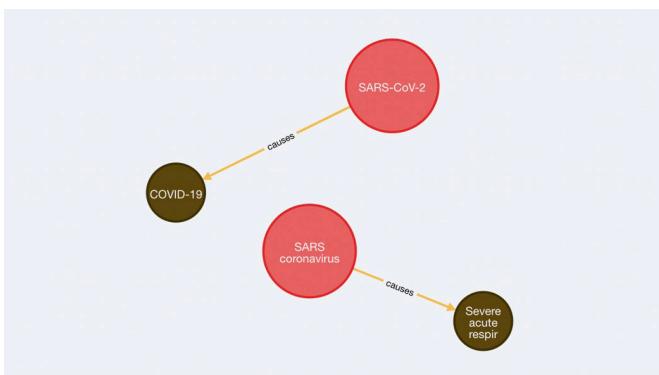
MATCH a=(p:pathogen)-[r:causes]->(di:disease)
WHERE p.name="Human papillomavirus type 16"
or p.name="Human papillomavirus type 18"
RETURN a;
  
```



Hình: Hai loại mầm bệnh hàng đầu là papillomavirus có thể dẫn đến các loại ung thư ở người

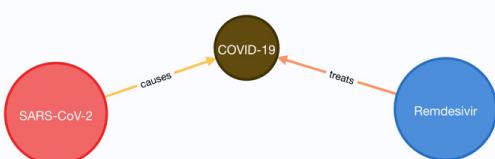
```

MATCH a=(p:pathogen)-[r:causes]->(di:disease)
WHERE di.name="COVID-19"
or di.name="Severe acute respiratory syndrome"
RETURN a;
  
```



Hình: Hai mầm bệnh dẫn đến SARS và COVID-19

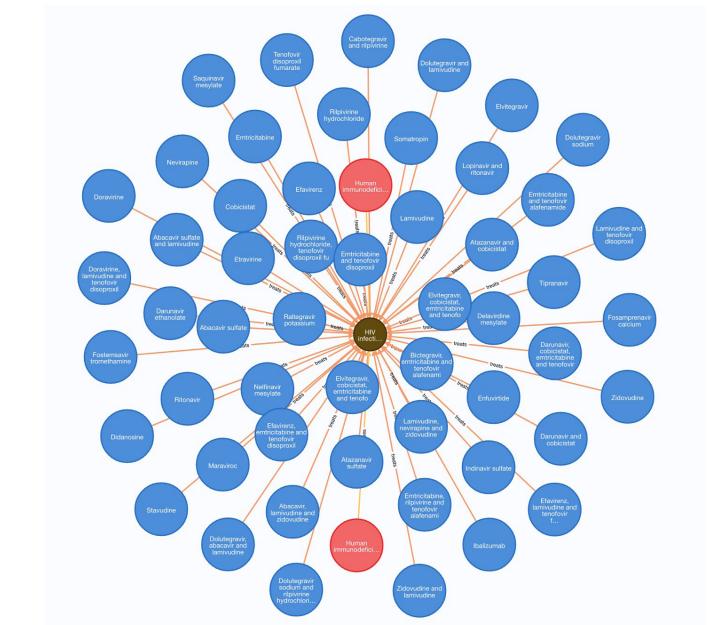
Ngược lại, cả mầm bệnh SARS và SARS-CoV-2 đều chỉ kết nối tới 1 bệnh của riêng chúng. SARS đã gây ra đại dịch SARS năm 2002–2004 ở Hồng Kông và cho đến nay vẫn chưa có liệu pháp kháng vi-rút nào cho bệnh này. Đối với COVID-19, KEGG liệt kê thuốc Remdesivir có thể rút ngắn các triệu chứng COVID.



Hình: Remdesivir được công nhận là có thể rút ngắn các triệu chứng COVID

```

MATCH a=(p:pathogen)-[r:causes]->(di:disease)
WHERE di.name="HIV infection"
RETURN a;
  
```



Phát hiện cộng đồng bệnh bằng thuật toán Louvain

Đầu tiên, chúng ta cho tất cả các nút và cạnh vào trong một đồ thị có tên là “disease-graph”.

```

CALL gds.graph.create.cypher(
  'disease-graph',
  'MATCH (n) RETURN id(n) AS id',
  'MATCH (n)--(m) RETURN id(n) AS source, id(m) AS target'
)
  
```

Bây giờ chúng ta có thể chạy thuật toán Louvain trên nó để xác định các cụm hoặc cộng đồng. Thuật toán này ngưng tụ các nút được kết nối gần nhau thành các nút lớn hơn và lặp lại cho đến khi không thể ngưng tụ. Đầu tiên, chúng ta cùng xem nhiều cộng đồng có thể được hình thành bằng câu lệnh dưới đây.

```

CALL gds.louvain.stats('disease-graph')
YIELD communityCount
  
```

Kết quả trả ra 2120 cộng đồng với tổng cộng 4170 nút. Khi chạy truy vấn này với các tham số khác nhau, chẳng hạn như “maxIterations” và “tolerances” nhưng không cho ra kết quả khác biệt lớn nào.

Bây giờ chúng ta có thể kiểm tra các cộng đồng lớn nhất và xem các thành viên của nó.

```
CALL gds.louvain.stream('disease-graph')
YIELD nodeId, communityId, intermediateCommunity-
Ids
RETURN communityId, COUNT(communityId) as count
ORDER BY count DESC LIMIT 10;
```

	communityId	count
1	3421	190
2	2767	189
3	2615	172
4	3549	160
5	3402	120
6	3561	116
7	3518	108
8	3190	90
9	2848	74
10	3239	73

Cộng đồng có ID 3421 là cộng đồng lớn nhất với 190 thành viên. Sau khi xem qua danh sách, rõ ràng là cộng đồng này chủ yếu chứa các mục liên quan đến khối u. Các mầm bệnh như vi rút Viêm gan B và D được biết là làm tăng nguy cơ ung thư cũng có trong cộng đồng này.

Nhận xét và kết luận

Chúng ta có thể tiến hành thêm nhiều phân tích thêm cho các cộng đồng khác. Các phân tích cộng đồng cho thấy rằng các bệnh trong các cụm lớn là những bệnh nhận được sự quan tâm nghiên cứu nhiều nhất và chúng có mối liên hệ chặt chẽ với nhau thông qua các mầm bệnh thông thường hoặc các loại thuốc đa dụng.

Ngược lại, nhiều “hòn đảo” biệt lập là những bệnh hiếm gặp như hội chứng Christianson hoặc Zimmermann-Laband. Đáng chú ý là SARS và COVID-19 cũng là hai “hòn đảo” biệt lập mặc dù chúng đã gây ra một đại dịch thay đổi thế giới của chúng ta mãi mãi. Thông điệp từ nghiên cứu này cũng rất rõ ràng: chúng ta còn biết quá ít về COVID-19.