Ứng dụng thuật toán đồ thị

# Thuật toán A\*:

## Bài toán:

Tìm đường đi ngắn nhất từ đỉnh A đến đỉnh B của đồ thị.

## Phương án giải quyết:

* Dùng BFS, Dijkstra, Bellman – Ford, …

## A\* là gì?

* Là giải thuật tìm kiếm trong đồ thị, được phát minh vào năm 1968 trong dự án Shakey, một dự án robot tự di chuyển.
* Là thuật toán tìm đường đi từ một đỉnh đến đỉnh đích có sử dụng hàm để ước lượng khoảng cách còn lại, ở đây là hàm heuristic (đọc thêm ở mục tiếp theo).
* A\* dùng hàm ước lượng khoảng cách để đánh giá đường đi tốt nhất có thể đi.
* Nếu tồn tại nhiều đường đi từ một đỉnh đến đỉnh khác, A\* luôn đảm bảo tìm được đường đi tối ưu nhất.

## Heuristic và hàm heuristic:

* Heuristic là phương pháp giải quyết vấn đề dựa trên phỏng đoán, ước chừng, kinh nghiệm để tìm ra giải pháp tốt nhất cho bài toán.
* Hàm heuristic là hàm ứng với mỗi trạng thái hay mỗi sự lựa chọn một giá trị ý nghĩa đối với vấn đề, dựa vào giá trị hàm này ta lựa chọn hành động.
* Đối với thuật toán A\*, hàm heuristic ở đây thường là ước lượng khoảng cách còn lại để đến đích.

## A\* hoạt động như thế nào?

* A\* lưu giữ một tập các lời giải chưa hoàn chỉnh, nghĩa là các đường đi qua đồ thị, bắt đầu từ nút xuất phát. Tập lời giải này được lưu trong một hàng đợi ưu tiên (priority queue).
* Thứ tự ưu tiên của một đường đi đến đỉnh  �xđược quyết định bởi hàm . Trong đó:
  + : độ ưu tiên của .
  + : khoảng cách điểm bắt đầu đến .
  + : khoảng cách ước lượng để đến đích từ .
* Tùy mỗi dạng bài khác nhau mà hàm ước lượng khoảng cách sẽ khác nhau.
* Thông thường có giá trị càng thấp thì độ ưu tiên càng cao.

## Cụ thể:

* Để thực hiện thuật toán này thì chúng ta cần có 2 danh sách:
  + : tập các đỉnh cần xét.
  + : tập các đỉnh đã được xét.
* Đầu tiên, chúng ta sẽ thêm đỉnh bắt đầu vào.
* Tiếp theo thực hiện các bước:
  + B1: Nếu rỗng => Không tìm thấy đường đi. Thoát.

Nếu không rỗng:

* + - Chọn đỉnh có thứ tự ưu tiên cao nhất trong , xóa khỏi .
    - Nếu là đỉnh cần tìm thì thoát.
    - Nếu không thì thêm vào , sau đó xét các đỉnh kề của
      * Nếu có trong thì bỏ qua.
      * Nếu chưa có trong thì thêm vào và tính toán , và .
      * Nếu đã có trong và (đường đi mới tối ưu hơn đường đi cũ) thì cập nhật lại và .
  + B2: Thực hiện B1.

## Độ phức tạp:

* Độ phức tạp về không gian lẫn thời gian phụ thuộc vào hàm heuristic.

## Ưu nhược điểm:

### Ưu điểm:

* + Luôn tìm được lời giải nếu bài toán có lời giải.
  + Là một thuật toán linh động, tùy bài toán mà chúng ta sẽ sử dụng hàm heuristic phù hợp.
* Nhược điểm:
  + Phụ thuộc vào hàm heuristic: A\* dựa vào hàm heuristic để ước lượng chi phí còn lại từ điểm hiện tại đến điểm đích. Tuy nhiên, nếu hàm heuristic không được thiết kế tốt, nó có thể dẫn đến việc tìm kiếm không chính xác hoặc tốn nhiều thời gian hơn để tìm đường đi tối ưu.
  + Chi phí tính toán: Thuật toán A\* phải tính toán chi phí từ điểm bắt đầu đến điểm hiện tại và từ điểm hiện tại đến điểm đích cho mỗi nút trong quá trình tìm kiếm. Điều này đặc biệt quan trọng khi áp dụng A\* cho các đồ thị lớn hoặc phức tạp, vì nó có thể yêu cầu rất nhiều phép tính.
  + Không đảm bảo tìm kiếm tối ưu: Mặc dù A\* có xu hướng tìm đường đi tối ưu, nó không đảm bảo tìm ra đường đi tối ưu. Nếu hàm heuristic không được chọn cẩn thận, A\* có thể rơi vào các đường cụt hoặc lạc hậu trong quá trình tìm kiếm.
  + Lưu trữ toàn bộ cây tìm kiếm: A\* yêu cầu lưu trữ toàn bộ cây tìm kiếm trong bộ nhớ. Điều này đòi hỏi một lượng lớn bộ nhớ, đặc biệt là đối với các đồ thị lớn. Việc lưu trữ toàn bộ cây tìm kiếm cũng tăng đáng kể việc tiêu thụ bộ nhớ.

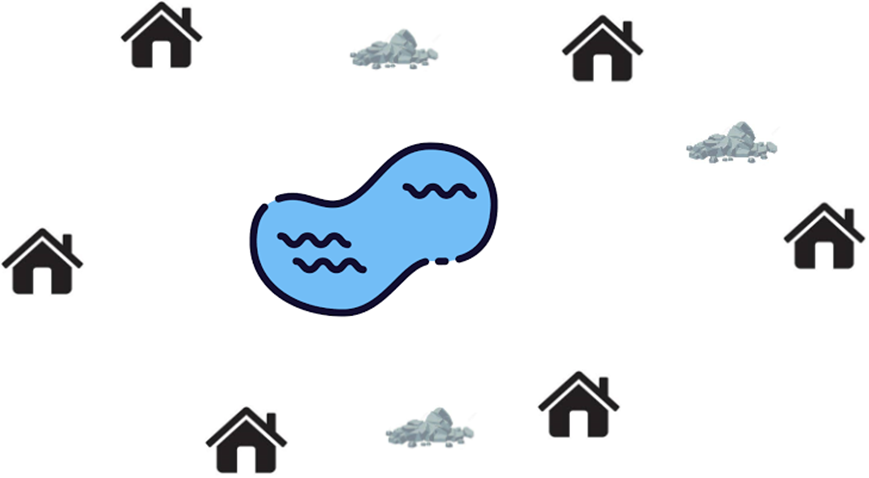
## Ứng dụng:

* Thuật toán A\* được sử dụng trong một số giao thức định tuyến mạng như RIP, OSPF, và BGP để tính toán đường đi tốt nhất giữa 2 nodes.
* Ứng dụng trong lập trình game để tìm đường đi ngắn nhất giữa 2 đối tượng.
* Ứng dụng trong tự động hóa, giúp robot tìm đường đi tốt nhất để đến đích.

# Cây khung nhỏ nhất:

## Bài toán:

Xây dựng các con đường đi qua tất cả nhà dân với chi phí nhỏ nhất?

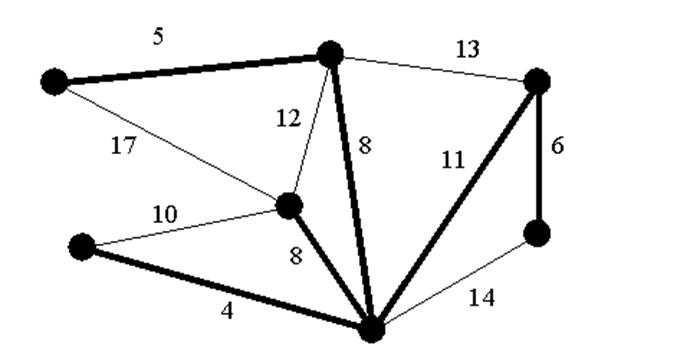


## Phương án giải quyết:

* Xây dựng đồ thị biểu diễn các nhà dân và khoảng cách giữa chúng. Mỗi nhà dân được biểu diễn bởi một đỉnh trong đồ thị và khoảng cách giữa các nhà dân là trọng số của cạnh nối giữa các đỉnh tương ứng.
* Có thể dùng cây khung nhỏ nhất để tối ưu chi phí cho bài toán.

## Cây khung nhỏ nhất là gì?

* Cây khung là tập hợp các cạnh của đồ thị sao cho các cạnh này liên thông (từ một đỉnh có thể đi đến bất kỳ đỉnh nào khác) và không chứa chu trình.
* Cây khung của đồ thị sẽ có đúng cạnh.
* Cây khung nhỏ nhất là cây khung có tổng trọng số là nhỏ nhất.



## Giải thuật tìm cây khung nhỏ nhất:

* Có rất nhiều thuật toán được đưa ra để tìm cây khung nhỏ nhất. Ví dụ: thuật Boruvkal, thuật toán Prim, thuật toán Kruskal.
* Phổ biến nhất là thuật toán Kruskal.

## Thuật toán Kruskal:

* Thuật toán Kruskal là một thuật toán trong lý thuyết đồ thị để tìm cây bao trùm (cây khung) nhỏ nhất của một đồ thị liên thông có trọng số.
* Thuật toán này xuất bản lần đầu tiên năm 1956, bởi Joseph Kruskal.

## Ý tưởng thuật toán Kruskal:

Ta cần tìm cây khung nhỏ nhất của đồ thị liên thông . Thuật toán sẽ bao gồm các bước sau:

* B1: Sắp xếp tập hợp các cạnh tăng dần theo trọng số.
* B2: Xét cạnh nhỏ nhất trong:
  + Chọn nó là một cạnh của cây khung nếu nó không tạo chu trình với các cạnh được chọn khác.
  + Ngược lại loại nó.
* B3: Lặp lại B2 cho đến khi hoàn thành cây khung.
* Thuật toán Kruskal là một giải thuật tham lam.

Ở B2, chúng ta có thể sử dụng CTDL Disjoint set để kiểm tra các các cạnh có tạo chu chu trình hay không.

## Độ phức tạp:

* Thời gian:
  + Độ phức tạp của thuật toán sắp xếp các cạnh theo thứ tự tăng dần: .
  + Ta duyệt E cạnh, mỗi cạnh dùng Disjoint set tốn . Vậy ta tốn .
* Độ phức tạp của thuật toán Kruskal là .
* Không gian:
  + để lưu các cạnh.
  + cho Disjoint set.
* Độ phức tạp không gian là .

## Ưu nhược điểm:

* Ưu điểm: dễ cài đặt, hiệu quả cao
* Nhược điểm: Có nhiều giải pháp đơn giản hơn Kruskal khi áp dụng với đồ thị không trọng số.

## Ứng dụng:

* Thiết kế mạng: Thuật toán Kruskal có thể được sử dụng để thiết kế mạng với chi phí thấp nhất. Nó có thể được sử dụng để tìm các kết nối mạng rẻ nhất có thể kết nối tất cả các nút trong mạng.
* Thuật toán xấp xỉ: Thuật toán Kruskal có thể được sử dụng để tìm các giải pháp xấp xỉ cho một số bài toán tối ưu phức tạp ví dụ như bài toán người du lịch (Traveling Salesman Problem).
* Dùng trong các giao thức kết nối giữa các thiết bị, máy móc để tránh tạo chu trình.
* Thiết kế hệ thống tưới tiêu với chi phí thấp cho nông trại.

# Sắp xếp Tô-pô (Topological sort):

## Bài toán sắp xếp Tô-pô:

* Thứ tự tô-pô của một đồ thị có hướng, không chu trình (DAG) là một thứ tự sắp xếp của các đỉnh sao cho với mọi cung từ đỉnh  đến đỉnh  trong đồ thị,  luôn nằm trước .
* Bài toán xác định thứ tự Tô-pô là bài toán sắp xếp Tô-pô.
* Các thuật toán sắp xếp tô pô được nghiên cứu lần đầu tiên vào những năm 1960 trong phương pháp PERT cho việc lập kế hoạch trong quản lý dự án.
* Định lý : Một đồ thị tồn tại thứ tự Tô-pô khi và chỉ khi đồ thị đó là DAG. Đồng nghĩa, mọi DAG đều luôn tồn tại ít nhất một thứ tự Tô-pô, và có thuật toán để tìm thứ tự Tô-pô trong thời gian tuyến tính.

## Bài toán:

Sắp xếp công việc cần làm để chuẩn bị cho một buổi họp

Một số yêu cầu cần có để chuẩn bị cho 1 buổi họp bao gồm:

- Công việc A: Chuẩn bị tài liệu

- Công việc B: Đặt phòng họp

- Công việc C: Chuẩn bị báo cáo

- Công việc D: Kiểm tra thiết bị

- Công việc E: Gửi lời mời tham dự

- Công việc F: Chuẩn bị danh sách tham dự

- Công việc G: Chuẩn bị bản trình bày

Mối quan hệ phụ thuộc giữa các công việc như sau:

- Công việc B phải hoàn thành trước khi Công việc A và C được thực hiện.

- Công việc D phải hoàn thành trước khi Công việc B và G được thực hiện.

- Công việc E phải hoàn thành trước khi Công việc B, C và F được thực hiện.

- Công việc F phải hoàn thành trước khi Công việc G được thực hiện.

## Phương án giải quyết:

- Xây dựng đồ thị biểu diễn mối quan hệ phụ thuộc giữa các công việc.

- Áp dụng Topological Sort để xác định thứ tự thực hiện công việc.

## Giải thuật Kahn:

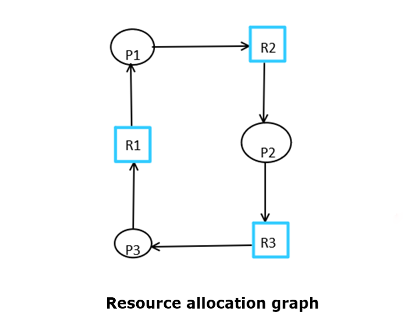
* Được đưa ra bởi nhà khoa học cùng tên vào năm 1962 dựa trên nhận xét: *Một DAG luôn luôn tồn tại ít nhất một đỉnh có bậc vào bằng 0 (đỉnh nguồn) và một đỉnh có bậc ra bằng 0*.
* Giải thuật:
  + B1: tính bậc vào của từng đỉnh trên đồ thị.
  + Sử dụng hàng đợi, nạp các đỉnh có bậc vào bằng 0 vào hàng đợi. Đánh dấu toàn bộ đỉnh này đã được thăm.
  + Với mỗi đỉnh trong hàng đợi, xét các đỉnh mà nó hướng đến, loại bỏ đường đi , giảm bậc vào của đi 1. Nếu bậc cửa đỉnh bằng 0 thì đẩy vào hàng đợi.
  + Lặp lại bước trên đến khi hàng đợi rỗng, ta thu được thứ tự Tô-pô.
  + Thứ tự các đỉnh được lấy ra khỏi hàng đợi chính là thứ tự Tô-pô của đồ thị.

## Độ phức tạp:

* Thời gian: .
* Không gian: .
* Trong đó, là số đỉnh của đồ thị, là số cạnh của đồ thị.

## Ứng dụng:

* Ứng dụng trong trình biên dịch của các ngôn ngữ lập trình, xác định thứ tự biên dịch của các đoạn code.
* Phát hiện deadlock trong hệ điều hành.



* Lập lịch các khóa học trong chương trình học.
* Tìm đường đi Hamilton trong DAG. Nếu đường đi Hamilton tồn tại thì thứ tự Tô-pô tìm được là duy nhất.
* Dùng để tạo sơ đồ luồng mạng (netwok flow diagram).

# Một số ứng dụng khác

## Thuật toán PageRank trong các công cụ tìm kiếm:

* PageRank là thuật toán phân tích các liên kết được dùng trong Google Search để xếp hạng các trang web. Giúp kết quả tìm kiếm được tốt hơn, hạn chế những trang spam.
* Thuật toán này xếp hạng một trang web dựa trên khái niệm: “số trang liên kết tới một trang web sẽ đo độ quan trọng của trang web đó”.
* Các trang web được nhiều trang web khác liên kết tới hơn sẽ quan trọng hơn và sẽ được ưu tiên xuất hiện trên kết quả tìm kiếm.

## Các thuật toán tìm thành phần liên thông mạnh:

* Một thành phần liên thông mạnh của một đồ thị có hướng là một đồ thị con mà giữa hai đỉnh bất kì luôn có đường đi đến nhau.
* Các thuật toán như Tarjan, Kosaraju, Gabow được sử dụng để tìm các thành phần liên thông mạnh.
* Các thuật toán này được áp dụng vào trong các thuật toán gợi ý bạn bè của các trang mạng xã hội như Facebook, Instagram …
* Các này sẽ tìm kiếm những nhóm người liên thông mạnh và gợi ý cho nhau dựa vào sở thích like, share.