Data Warehousing

Lecture 11 – Intro to Graph Data Science

CITS3401
CITS5504

Dr. Sirui Li

Computer Science and Software Engineering

School of Maths, Physics and Computing

Acknowledgement: The lecture slides are adopted from online sources.

Outline of This Week's Lectures

- What is Search
- Breadth and Depth First Search algorithms
- A* Search algorithm
- Minimum spanning tree and random walk
- Implementation with Neo4j

Outline of This Week's Lectures

- What is Search
- Breadth and Depth First Search algorithms
- A* Search algorithm
- Minimum spanning tree and random walk
- Implementation with Neo4j

Search Problems

- Search: Search is the process of systematically exploring a problem space to find a solution or achieve a goal.
- Steps:
  1. Formulating problems into states, actions, a transition model and action a cost function **(outside the scope of this unit)**
  2. Using search algorithms to find the solution
- Solution: A path from the initial state to a goal state

Application: Route Finding

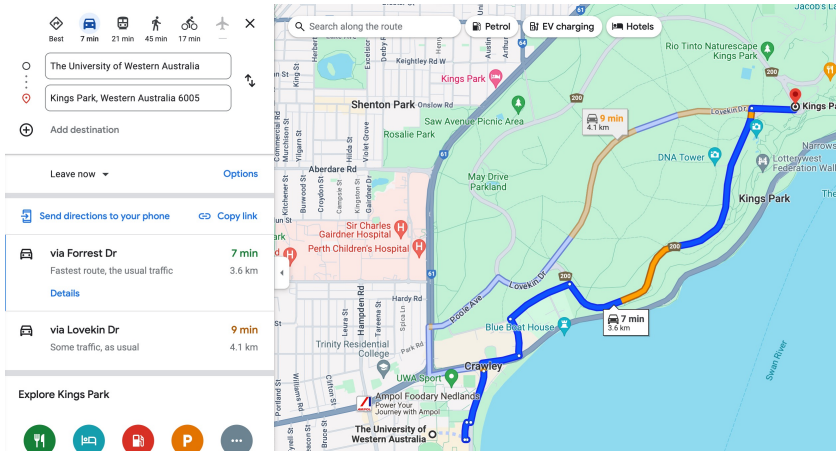
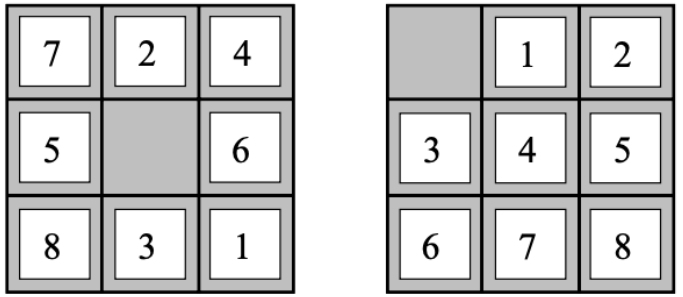


Image from Google Map

Application: Solving Puzzles



Application: Video Games

Image from https://www.retrogamer.net/retro_games90/the-making-of-the-sims/

Application: Robot Motion Planning

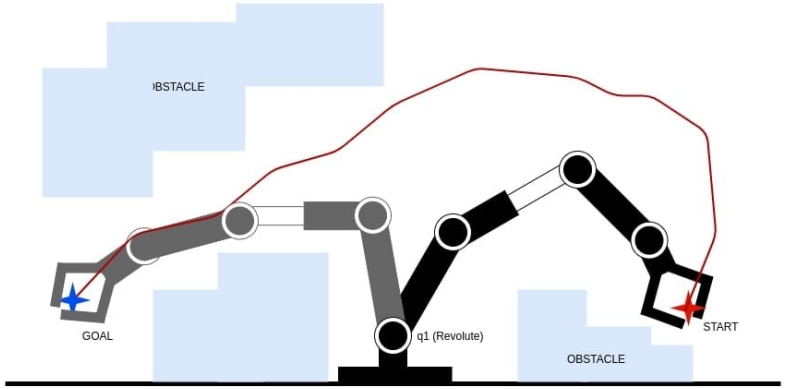
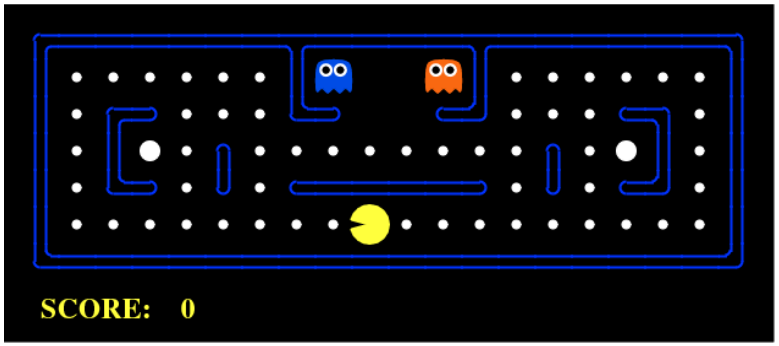


Image from: https://control.com/technical-articles/how-does-motion-planning-for-autonomous-robot-manipulation-work/

Example: Pac-Man

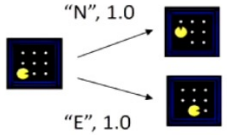


GIF from https://github.com/junthbasnet/Pacman-Search

Example: Formulating Pac-Man into a Search Problem (outside the scope of this unit)

Actions: North, South, East, West

Transition model:



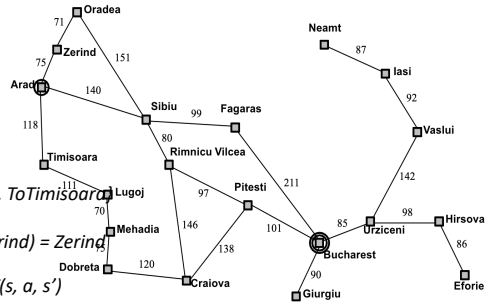
Action cost function: $f($  ,N,  $) = 1$

Solution: A sequence of actions which transforms the initial state to the goal state.

Example: Formulating Traveling in Romania into a Search Problem (outside the scope of this unit)

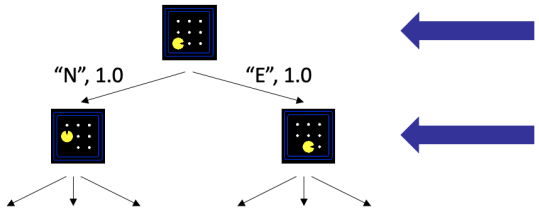
Formulate Problem:

- Initial state: In Arad
- Goal state: In Bucharest
- State space: Various cities
- Actions: drive between cities.
  E.g. ACTIONS(Arad) = {ToZerind, ToTimisoara}
- Transition model: *RESULT(Arad, ToZerind) = Zerind*
- Action cost function: *ACTION-COST(s, a, s')*
- Solution: sequence of drive actions leading from Arad to Bucharest



Scenario: On holiday in Romania; currently in Arad; flight leaves tomorrow from Bucharest; drive from Arad to Bucharest.

Search Tree



- A search tree:
  - A "what if" tree of plans and their outcomes
  - The initial/start state is the root node
  - Children correspond to successors
  - Nodes show states

Outline of This Week's Lectures

- What is Search
- Breadth and Depth First Search algorithms
- A* Search algorithm
- Minimum spanning tree and random walk
- Implementation with Neo4j

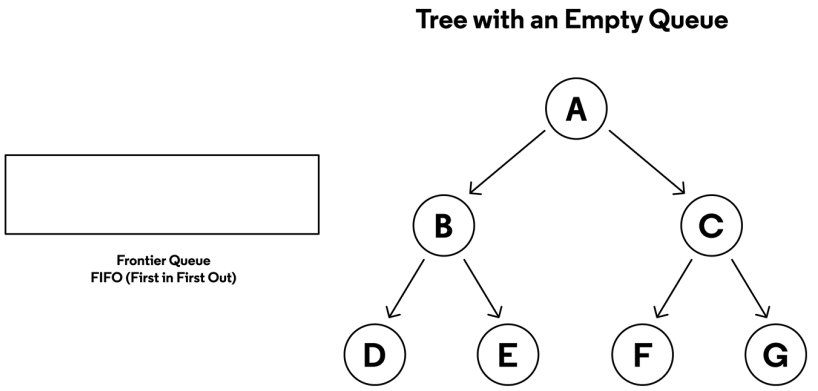
Pseudocode for Searching

```
// start is the initial or start state, goal is the goal state
function search_algorithm(start, goal):
    frontier = Queue()
    frontier.enqueue(start)

    while not frontier.isEmpty():
        current = frontier.dequeue() //BFS, DFS, A* have different dequeue techniques
        if current == goal:
          return "Goal found"
        for neighbor in current.neighbors():
          frontier.enqueue(neighbor)
    return "Goal not found"

```

Pseudocode for Searching

1. Frontier: the set of nodes that are considered but have not yet been fully explored. These nodes are usually candidates for further exploration in the search for the goal state.
2. What if multiple nodes share the same priority during dequeue? A common practice is alphabetical tie-breaking, prioritizing nodes earlier in the alphabet for expansion.
3. **Goal check occurs after dequeue**, not during enqueue.

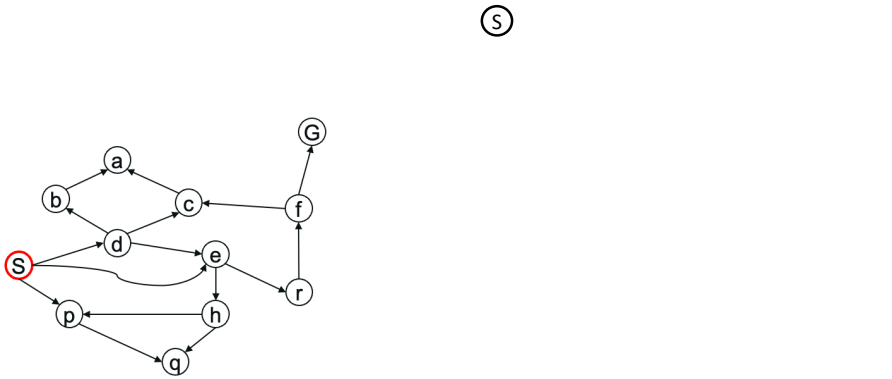
Breadth First Search (BFS)



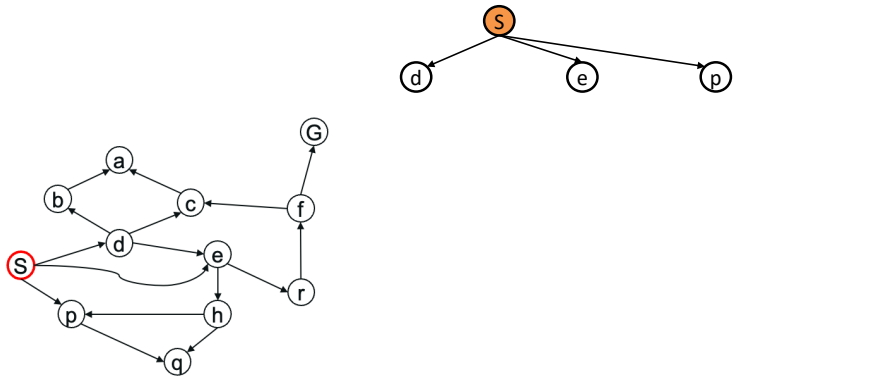
https://www.codecademy.com/article/tree-traversal

FIFO: new successors go at end

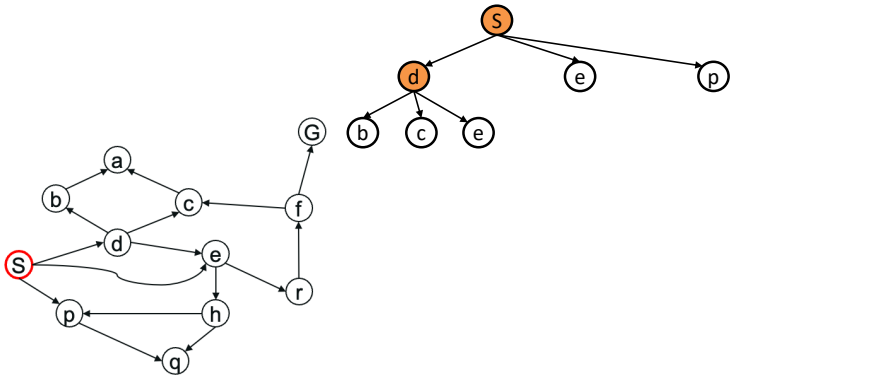
Example: Breadth First Search (BFS)



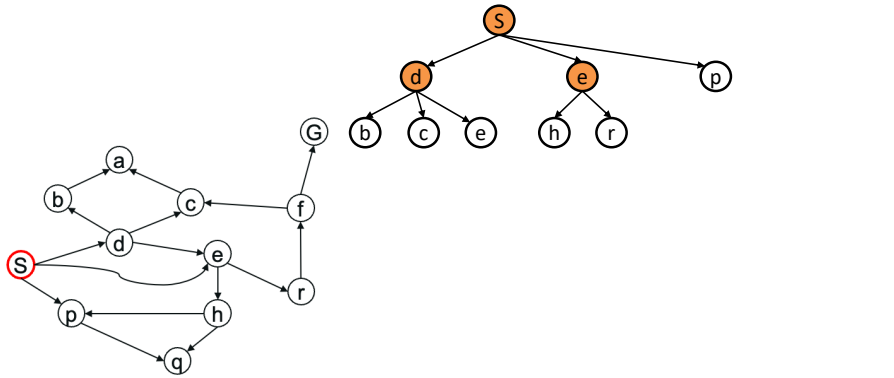
Example: Breadth First Search (BFS)



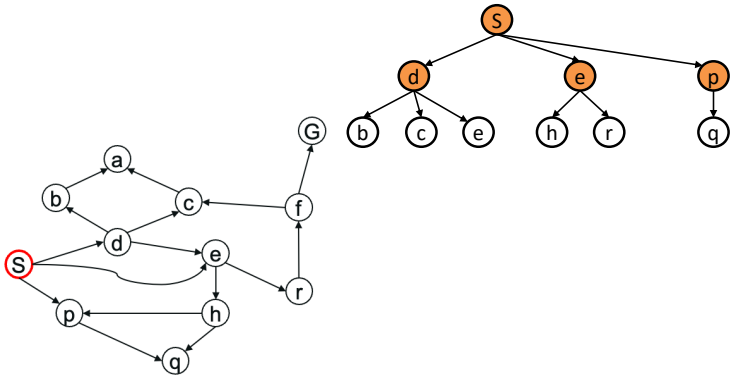
Example: Breadth First Search (BFS)



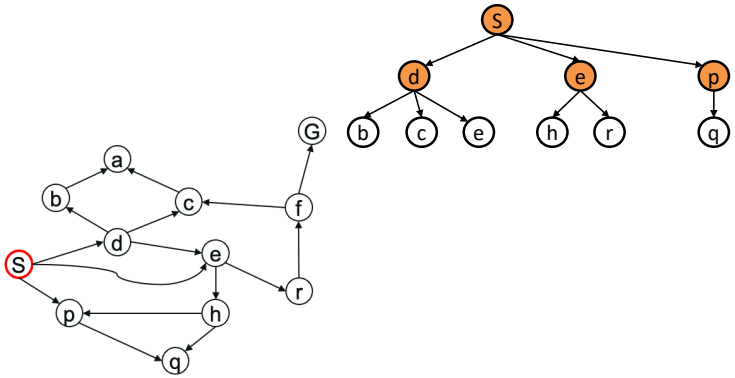
Example: Breadth First Search (BFS)



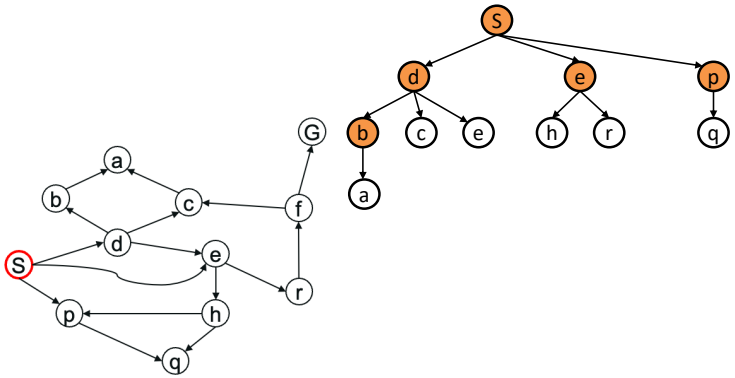
Example: Breadth First Search (BFS)



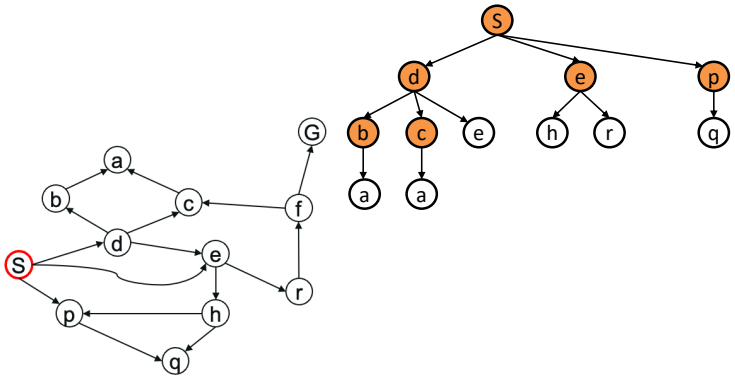
Example: Breadth First Search (BFS)



Example: Breadth First Search (BFS)

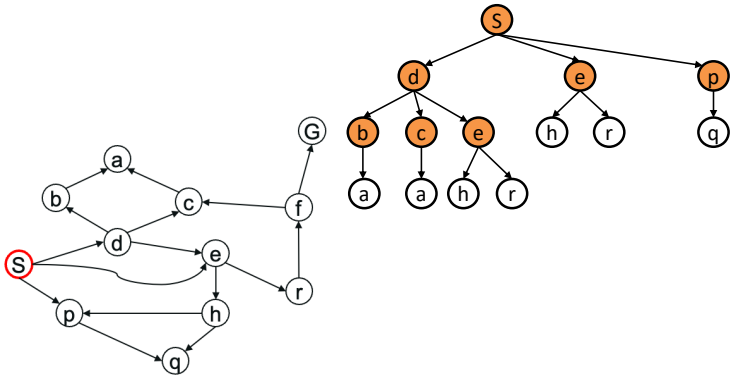


Example: Breadth First Search (BFS)



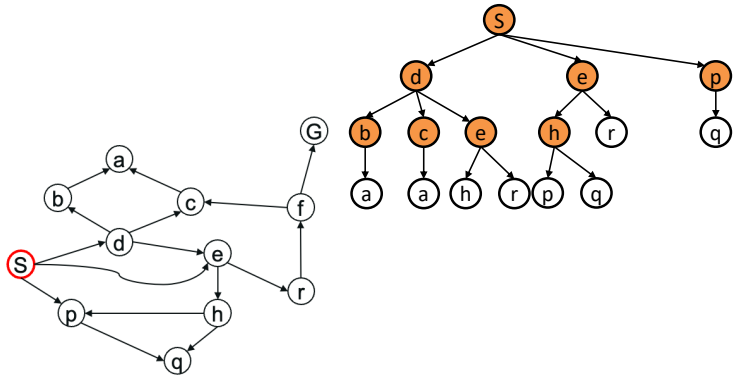
Example: Breadth First Search (BFS)

THE UNIVERSITY OF WESTERN AUSTRALIA



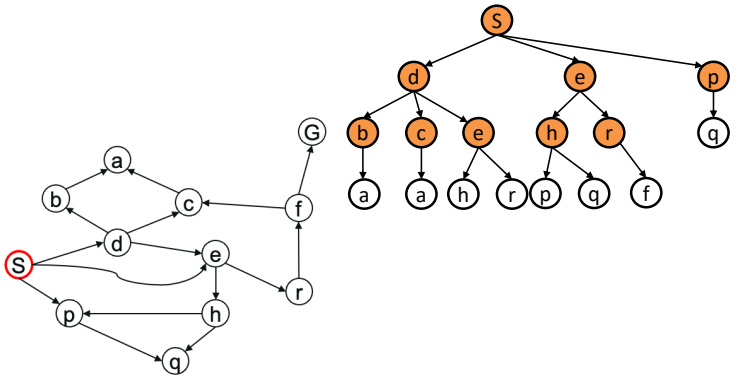
Example: Breadth First Search (BFS)

THE UNIVERSITY OF WESTERN AUSTRALIA



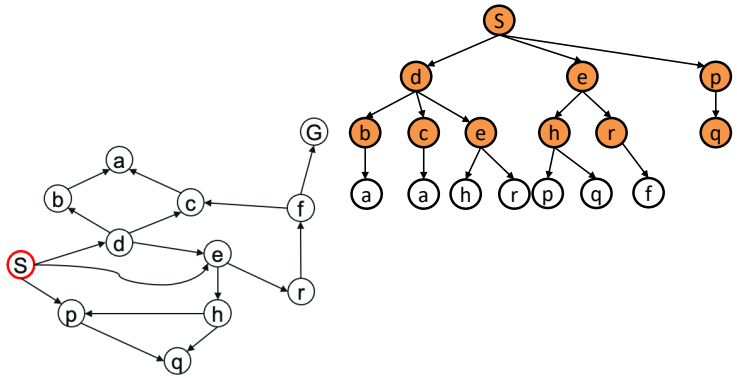
Example: Breadth First Search (BFS)

THE UNIVERSITY OF WESTERN AUSTRALIA



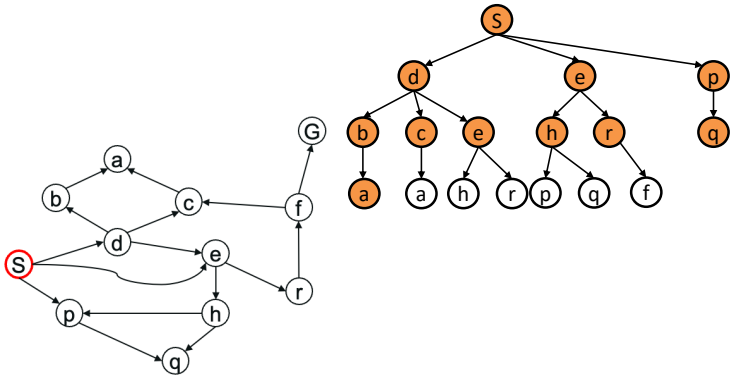
Example: Breadth First Search (BFS)

THE UNIVERSITY OF WESTERN AUSTRALIA



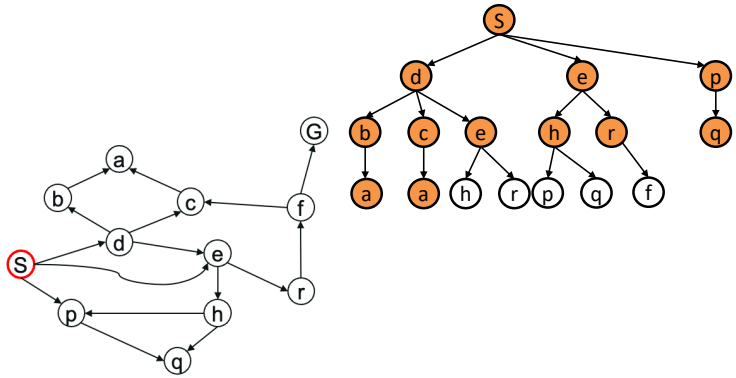
Example: Breadth First Search (BFS)

THE UNIVERSITY OF WESTERN AUSTRALIA



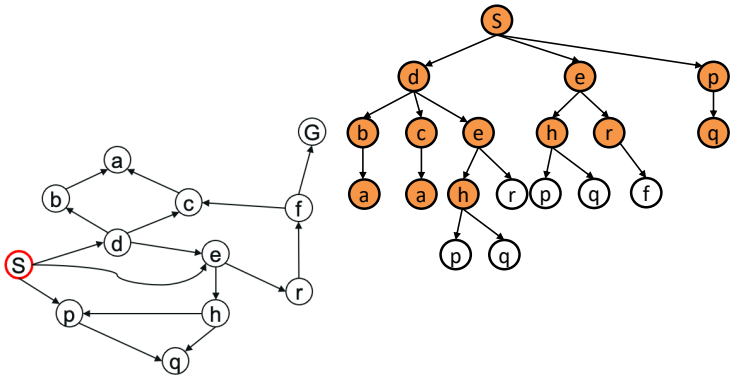
Example: Breadth First Search (BFS)

THE UNIVERSITY OF WESTERN AUSTRALIA



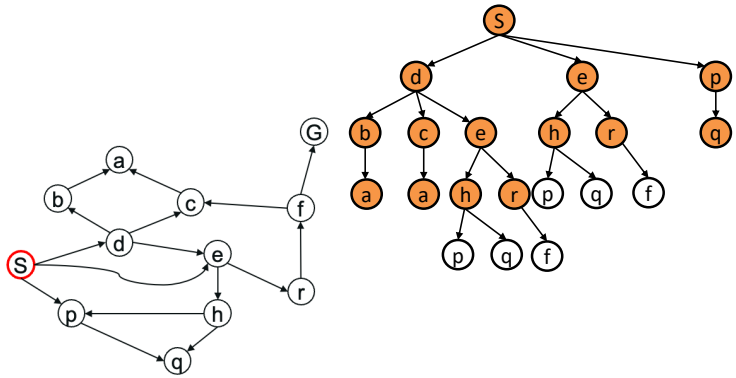
Example: Breadth First Search (BFS)

THE UNIVERSITY OF WESTERN AUSTRALIA



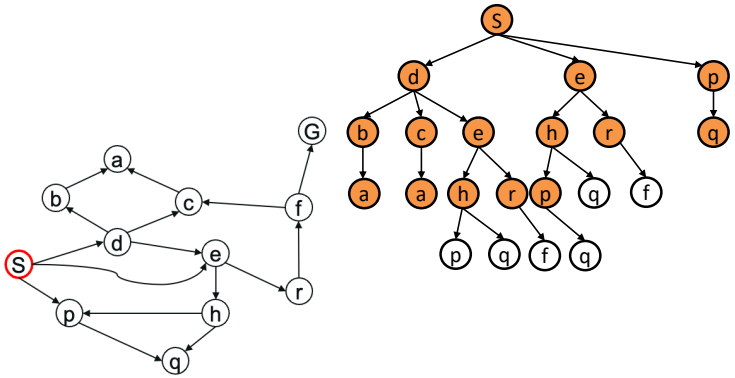
Example: Breadth First Search (BFS)

THE UNIVERSITY OF WESTERN AUSTRALIA



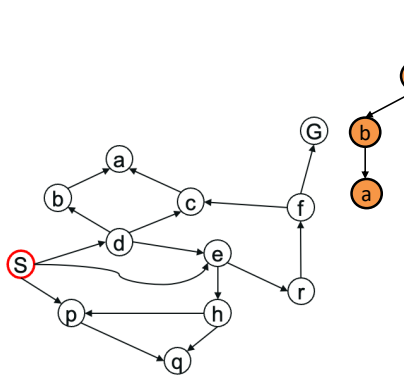
Example: Breadth First Search (BFS)

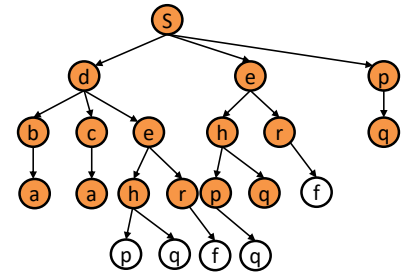
THE UNIVERSITY OF WESTERN AUSTRALIA



Example: Breadth First Search (BFS)

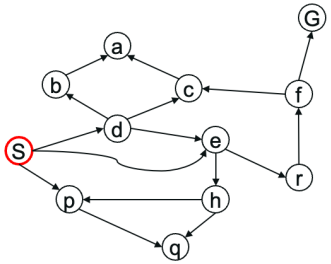
THE UNIVERSITY OF WESTERN AUSTRALIA

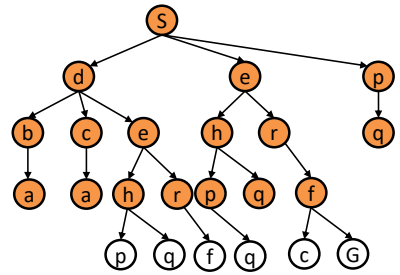




Example: Breadth First Search (BFS)

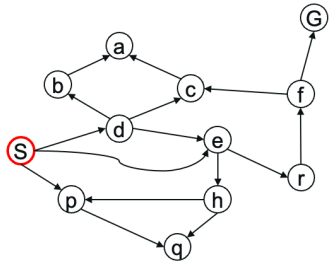
THE UNIVERSITY OF WESTERN AUSTRALIA

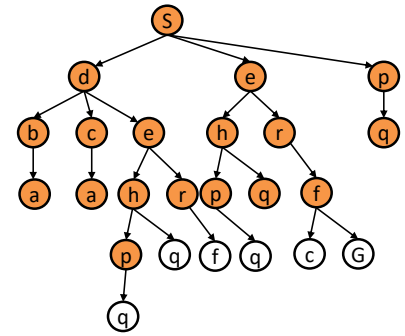




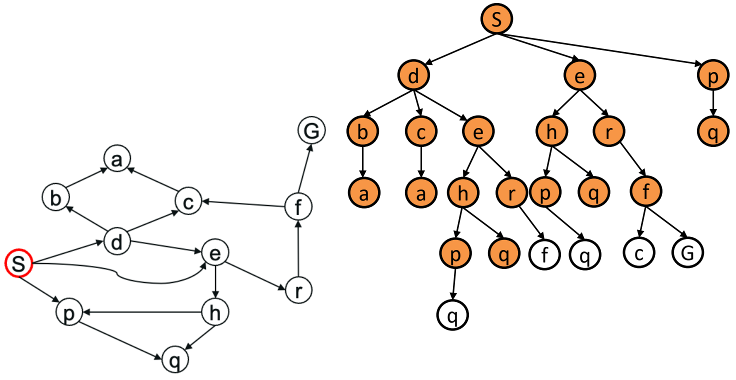
Example: Breadth First Search (BFS)

THE UNIVERSITY OF WESTERN AUSTRALIA

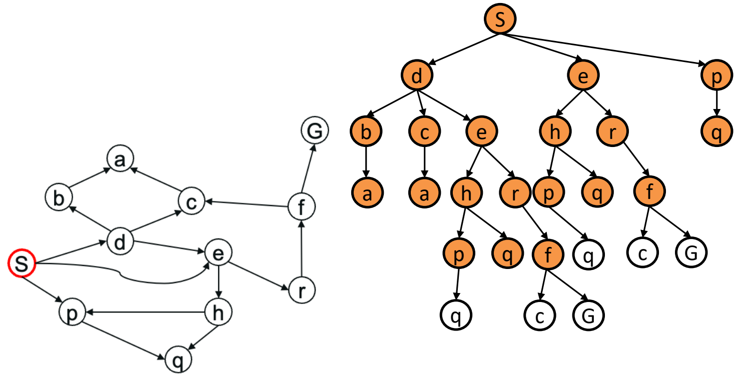




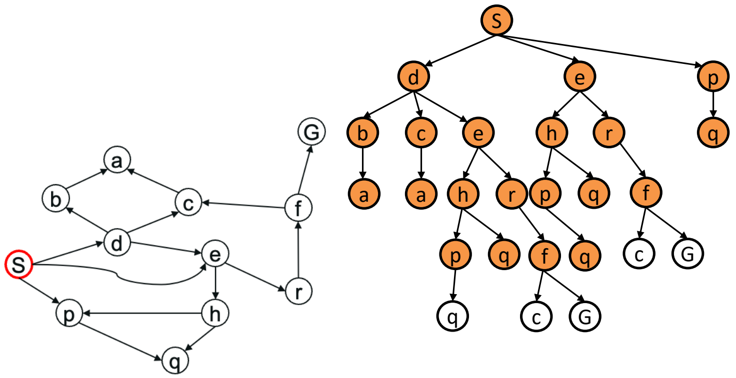
Example: Breadth First Search (BFS)



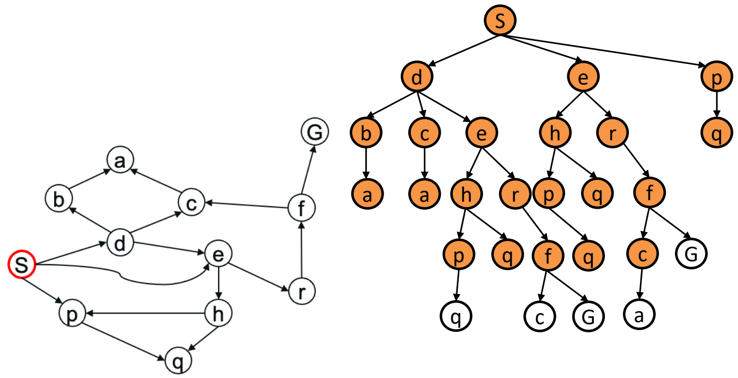
Example: Breadth First Search (BFS)



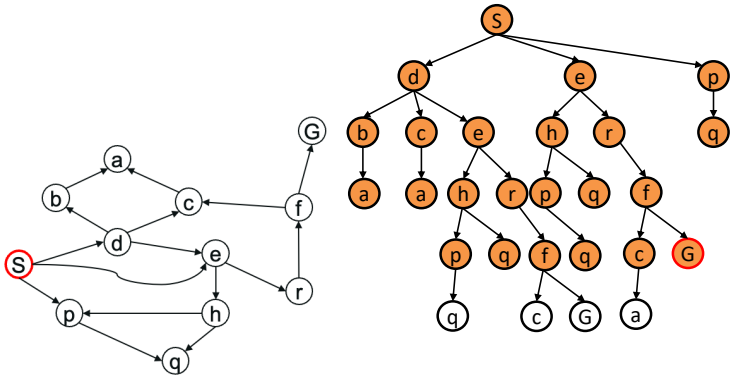
Example: Breadth First Search (BFS)



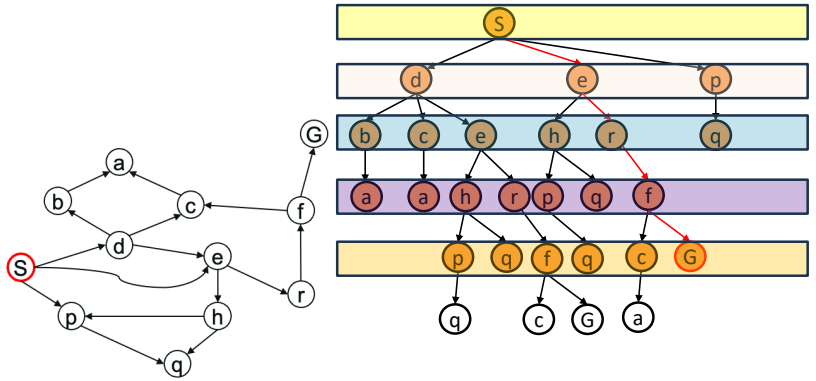
Example: Breadth First Search (BFS)



Example: Breadth First Search (BFS)

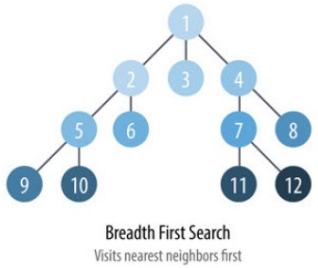


Example: Breadth First Search (BFS)



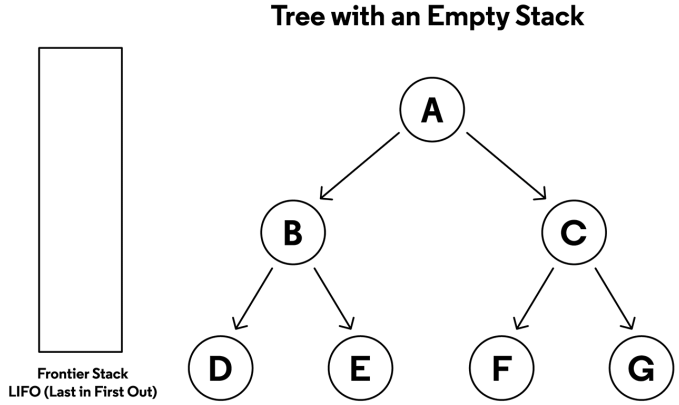
Breadth First Search

- Breadth First Search (BFS) starts from a chosen node and **explores all of its neighbours** at one hop away before visiting all neighbours two hops away, and so on.
- It is commonly used as a basis for other pathfinding algorithms (shortest path, closeness centrality etc).



https://neo4j.com/docs/graph-data-science/current/algorithms/bfs/
https://www.cs.usfca.edu/~galles/visualization/BFS.html

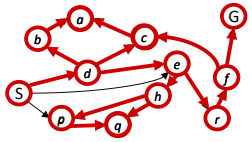
Depth First Search (DFS)



https://www.codecademy.com/article/tree-traversal

Example: DFS

Expand **deepest** unexpanded node
Implementation:
frontier = LIFO queue, i.e., put successors at front



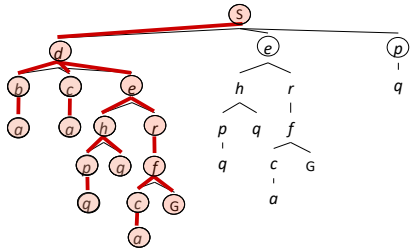
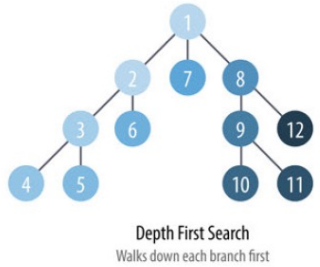


Image from UC berkeley

Depth First Search

- Depth First Search is similar to BFS but instead of exploring neighbours first, it **walks down each branch first**.
- Starting from a chosen node, it picks one of its neighbours, then traverses as far as it can along that path before backtracking.



https://neo4j.com/docs/graph-data-science/current/algorithms/dfs/
https://www.cs.usfca.edu/~galles/visualization/DFS.html

Quiz: BFS vs DFS (part 1)

Video from UC berkeley

Quiz: BFS vs DFS (part 2)

Video from UC berkeley

Outline of This Week's Lectures

- What is Search
- Breadth and Depth First Search algorithms
- A* Search algorithm
- Minimum spanning tree and random walk
- Implementation with Neo4j

A* Search

- BFS and DFS don't have information about Goal location
- A* search allows the inclusion of **extra information** that the algorithm can use as part of a **heuristic function** to determine which nodes to explore next.
- A* selects the path that minimises the evaluation function:

  $f(n) = g(n) + h(n)$

  where:

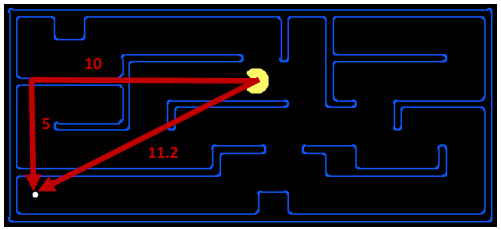
  - $g(n)$ is the cost of the path from the source to node n so far
  - $h(n)$ is the **estimated** cost of the path from node n to the destination node, as computed by a heuristic.
  - *f(n) is the* estimated total cost of path through n to goal (heuristic).

The lower $f(n)$, the more desirable is.

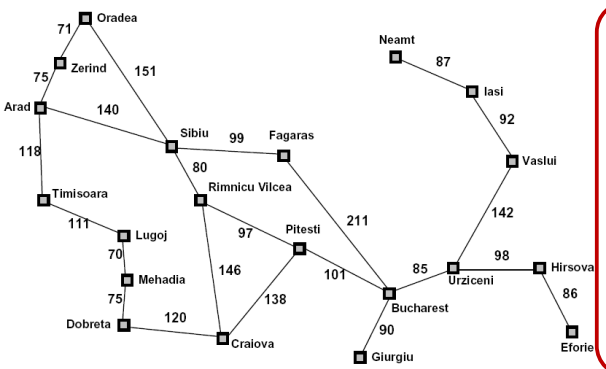
Search Heuristics

A heuristic is:

1. A function that *estimates* how close a state is to a goal
2. Designed for a particular search problem
3. Examples: Manhattan distance, Euclidean distance for pathing



Example: Heuristic Function



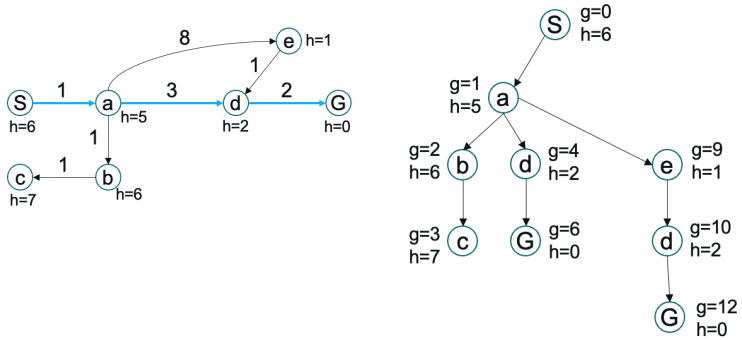
Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

h(x)

A* search

Idea: avoid expanding paths that are already expensive

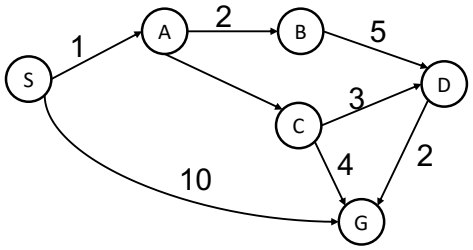
Evaluation function $f(n) = g(n) + h(n)$



A* search: orders by the evaluation function $\mathrm{f(n)} = g(n) + h(n)$

Your Turn

Given the directed graph below, use BFS, DFS and A* to find the path from S to G. Each edge has a cost associated with it (g(n)). If there are ties, assume alphabetical tie-breaking (i.e., nodes earlier in the alphabet are expanded first).



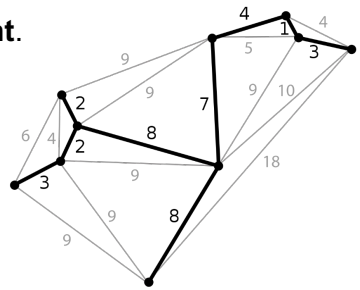
node	h(n)
S	5
A	3
B	4
C	2
D	6
G	0

Outline of This Week's Lectures

- What is Search
- Breadth and Depth First Search algorithms
- A* Search algorithm
- Minimum spanning tree and random walk
- Implementation with Neo4j

Minimum Spanning Tree

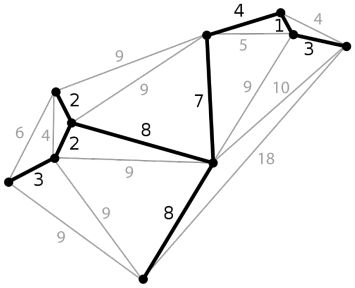
- The **Minimum Spanning Tree** algorithm starts from a given node and finds **all its reachable nodes** and the set of relationships that connect the nodes together with the **minimum possible weight**.
- It is often calculated using **Prim's Algorithm**, invented in 1957.



https://en.wikipedia.org/wiki/Minimum_spanning_tree

Minimum Spanning Tree

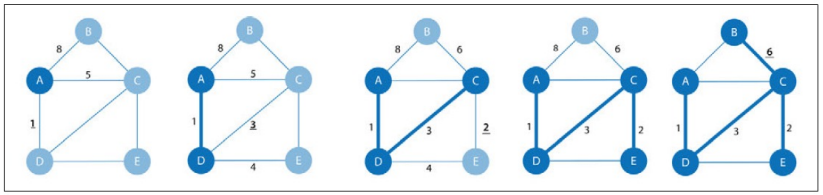
- Minimum Spanning Tree is useful when we need to find the best route to visit all nodes, i.e. visit all nodes in a single ‘walk’.
- It can be used for the Traveling Salesman Problem (TSP), as well as other industry-focused uses such as optimising water pipes and circuit design.
- Other uses include:
  - Minimising travel cost for exploring a country.
  - Tracing the history of infection transmission in an outbreak.



https://en.wikipedia.org/wiki/Minimum_spanning_tree

Minimum Spanning Tree

- **Prim’s Algorithm** operates as follows:
  - Start with a chosen node.
  - The relationship with the smallest weight connected to that node is added to the tree, along with its connected node.
  - Repeat this process, always choosing the minimal-weight relationship that joins any node **not** already in the tree.
  - When there are no more nodes, the tree is the MST.



Random Walk

- The **Random Walk algorithm** provides a set of nodes on a **random path** in a graph.
- It is often described as being similar to how a drunk person traverses a city. They know where they want to go, but have no strategy for how to get there.
- It is surprisingly useful, particularly for machine learning applications:
  - It is the cornerstone of the *node2vec* and *graph2vec* algorithms, which create **graph embeddings**.
  - It can also be used as a way to measure *semantic similarity*.

Random Walk

- The algorithm is relatively straightforward:
  - Start at a chosen node, or a random node.
  - Move to a connected node at random.
  - Repeat until the chosen path length is reached.

Outline of This Week's Lectures

- What is Search
- Breadth and Depth First Search algorithms
- A* Search algorithm
- Minimum spanning tree and random walk
- Implementation with Neo4j

Intro to Graph Data Science

- Graph data science allows us to **answer business critical questions** and **make predictions** using a graph database.
- Neo4j has a **Graph Data Science** library that provides a range of useful graph algorithms: https://neo4j.com/docs/graph-data-science/current/.
- Note for this unit we require **Neo4j 5.30** and **GDS 2.3+.** The one I am using is Neo4j 5.30 and GDS 2.4.6.

Demo graph – the Transport Graph

- We will be using the **Transport graph** from the *Graph Algorithms* book by Needham & Hodler.

Table 4-2. transport-nodes.csv

id	latitude	longitude	population
Amsterdam	52.379189	4.899431	821752
Utrecht	52.092876	5.104480	334176
Den Haag	52.078663	4.288788	514861
Immingham	53.61239	-0.22219	9642
Doncaster	53.52285	-1.13116	302400
Hoek van Holland	51.9775	4.13333	9382
Felixstowe	51.96375	1.3511	23689
Ipswich	52.05917	1.15545	133384
Colchester	51.88921	0.90421	104390
London	51.509865	-0.118092	8787892
Rotterdam	51.9225	4.47917	623652
Gouda	52.01667	4.70833	70939

Table 4-3. transport-relationships.csv

src	dst	relationship	cost
Amsterdam	Utrecht	EROAD	46
Amsterdam	Den Haag	EROAD	59
Den Haag	Rotterdam	EROAD	26
Amsterdam	Immingham	EROAD	369
Immingham	Doncaster	EROAD	74
Doncaster	London	EROAD	277
Hoek van Holland	Den Haag	EROAD	27
Felixstowe	Hoek van Holland	EROAD	207
Ipswich	Felixstowe	EROAD	22
Colchester	Ipswich	EROAD	32
London	Colchester	EROAD	106
Gouda	Rotterdam	EROAD	25
Gouda	Utrecht	EROAD	35
Den Haag	Gouda	EROAD	32
Hoek van Holland	Rotterdam	EROAD	33

Creating the demo graph

- We can create the graph using a LOAD CSV:

```
WITH "https://github.com/neo4j-graph-analytics/book/raw/master/data/" AS base
WITH base + "transport-nodes.csv" AS uri
LOAD CSV WITH HEADERS FROM uri AS row
MERGE (place:Place {name:row.id})
SET place.latitude = toFloat(row.latitude),
place.longitude = toFloat(row.latitude),
place.population = toInteger(row.population)
```

```
WITH "https://github.com/neo4j-graph-analytics/book/raw/master/data/" AS base
WITH base + "transport-relationships.csv" AS uri
LOAD CSV WITH HEADERS FROM uri AS row
MATCH (origin:Place {name: row.src})
MATCH (destination:Place {name: row.dst})
MERGE (origin)-[:EROAD {distance: toInteger(row.cost)}]->(destination)
```

*Note: I have changed some things (id -> name, added a missing slash).

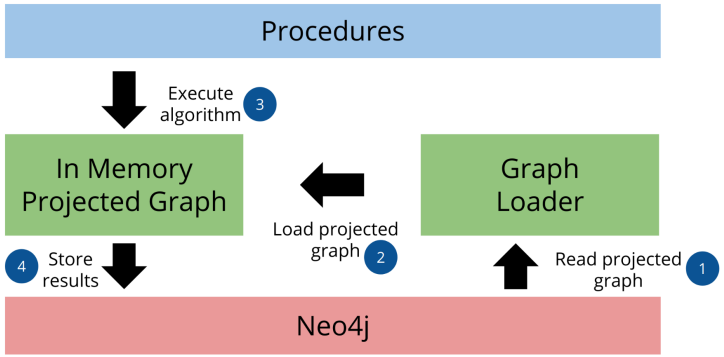
Projecting our graph

- We must first *project* our graph and store it in the graph catalog. This is necessary before running any GDS algorithms.

```
CALL gds.graph.project('Transport','Place','EROAD')
```

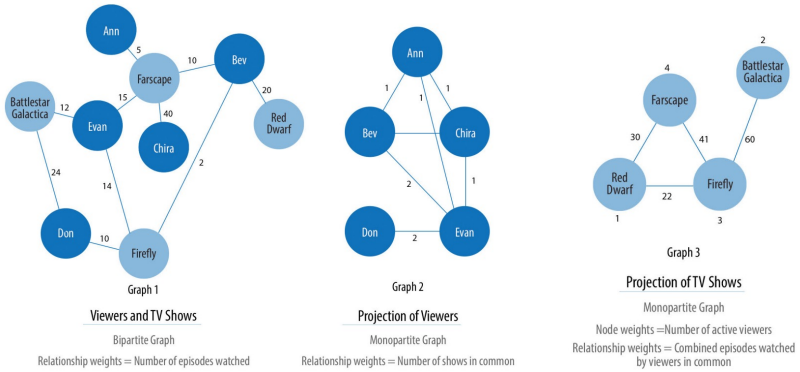
- Note we have not specified the **orientation** of the graph, so it defaults to “NATURAL” (same orientation as in the underlying Neo4j graph). This would make our graph **directed**, which is not ideal for our domain.

Neo4j’s Graph Data Science library



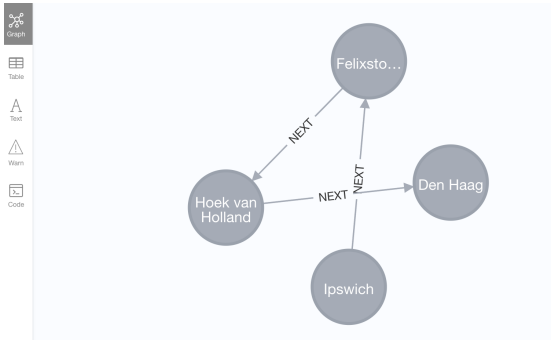
https://neo4j.com/docs/graph-data-science/current/common-usage/

Why Graph Projection?



Breadth First Search

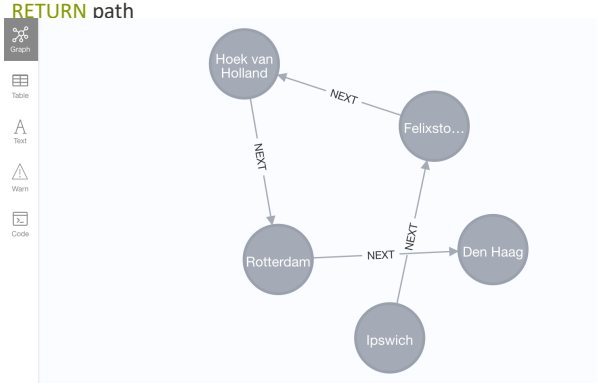
```
MATCH (source:Place{name:"Ipswich"}), (target:Place{name:'Den Haag'})
CALL gds.bfs.stream('Transport', {sourceNode:source, targetNodes:target})
YIELD path
RETURN path
```



Note that BFS and DFS in Neo4j GDS are **not weighted**.
Remember to **ignore the arrows** on the road relationships, as our graph is undirected.

Depth First Search

```
MATCH (source:Place{name:"Ipswich"}), (target:Place{name:'Den Haag'})
CALL gds.dfs.stream('Transport', {sourceNode:source, targetNodes:target})
YIELD path
RETURN path
```



Note that BFS and DFS in Neo4j GDS are **not weighted**.
Remember to **ignore the arrows** on the road relationships, as our graph is undirected.

A* Shortest Path in Neo4j

- Before we can run A* in Neo4j, we need to project the graph again, this time including the **distance, latitude and longitude** properties on the EROAD relationship.
- We must first **delete our projection** from before:

```
CALL gds.graph.drop('Transport')
```

We can then **project the graph** once more, this time with **distance, latitude and longitude** properties:

```
CALL gds.graph.project(
  'Transport',
  'Place',
  { EROAD: { orientation: 'UNDIRECTED' } },
  { relationshipProperties: 'distance',
    nodeProperties: ['latitude', 'longitude'] }
)
```


A* Shortest Path in Neo4j

We can run A* via the following command:

```
MATCH (source:Place {name: 'Ipswich'}), (target:Place {name: 'Utrecht'})
CALL gds.shortestPath.astar.stream('Transport', {
    sourceNode: source,
    targetNode: target,
    latitudeProperty: 'latitude',
    longitudeProperty: 'longitude',
    relationshipWeightProperty: 'distance'
})
YIELD index, sourceNode, targetNode, totalCost, nodeIds, costs, path
RETURN
    index,
    gds.util.asNode(sourceNode).name AS sourceNodeName,
    gds.util.asNode(targetNode).name AS targetNodeName,
    totalCost,
    [nodeId IN nodeIds | gds.util.asNode(nodeId).name] AS nodeNames,
    costs,
    nodes(path) as path
ORDER BY index
```


Minimum Spanning Tree in Neo4j

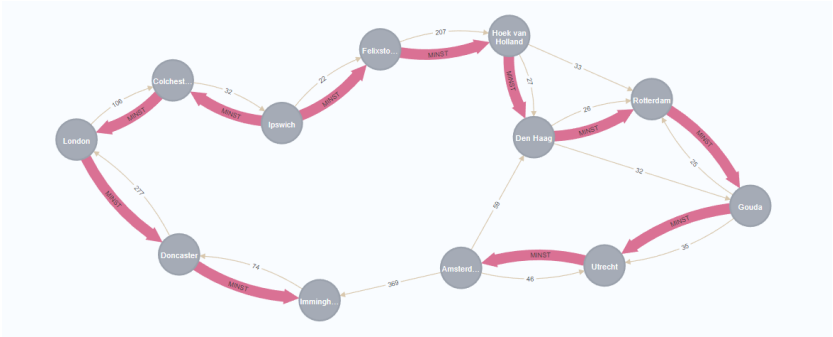
```
MATCH (n:Place{name: 'Ipswich'})
CALL gds.beta.spanningTree.write(
    'Transport',
    {sourceNode: id(n),
    relationshipWeightProperty: 'distance',
    writeProperty: 'writeCost',
    writeRelationshipType: 'MINST'})

YIELD preProcessingMillis, computeMillis, writeMillis, effectiveNodeCount
RETURN preProcessingMillis, computeMillis, writeMillis, effectiveNodeCount;
```

This will create a relationship of type *MINST* between each node in the graph where the minimum spanning tree exists.

Minimum Spanning Tree in Neo4j

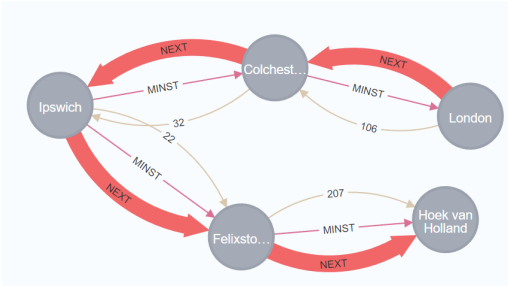
We can **display the MST** via the query (with *Connect result nodes OFF)*:

```
MATCH (p1:Place)-[m:MINST]->(p2:Place) RETURN p1, m, p2
```



Random Walk in Neo4j

```
MATCH (source:Place {name: "London"})
CALL gds.randomWalk.stream('Transport',
  {sourceNodes: [id(source)], walkLength: 5, walksPerNode: 1})
YIELD nodeIds, path
RETURN nodeIds,
[nodeId IN nodeIds | gds.util.asNode(nodeId).name] AS nodeNames,
path
```



Summary

Algorithm type	What it does	Example use	Spark example	Neo4j example
Breadth First Search	Traverses a tree structure by fanning out to explore the nearest neighbors and then their sublevel neighbors	Locating neighbor nodes in GPS systems to identify nearby places of interest	Yes	No
Depth First Search	Traverses a tree structure by exploring as far as possible down each branch before backtracking	Discovering an optimal solution path in gaming simulations with hierarchical choices	No	No
Shortest Path Variations: A*, Yen's	Calculates the shortest path between a pair of nodes	Finding driving directions between two locations	Yes	Yes
All Pairs Shortest Path	Calculates the shortest path between *all pairs of nodes* in the graph	Evaluating alternate routes around a traffic jam	Yes	Yes
Single Source Shortest Path	Calculates the shorest path between a *single root* node and *all* other nodes	Least cost routing of phone calls	Yes	Yes
Minimum Spanning Tree	Calculates the path in a connected tree structure with the smallest cost for visiting all nodes	Optimizing connected routing, such as laying cable or garbage collection	No	Yes
Random Walk	Returns a list of nodes along a path of specified size by randomly choosing relationships to traverse.	Augmenting training for machine learning or data for graph algorithms.	No	Yes

Copyright Notice

Copyright Notice