Graphs Are Everywhere

Christopher Hoult
@choult

Who is this guy?

Practicing PHP commercially since 2004

Senior Software Engineer at Datto

Formerly at DataSift and Time Out Digital

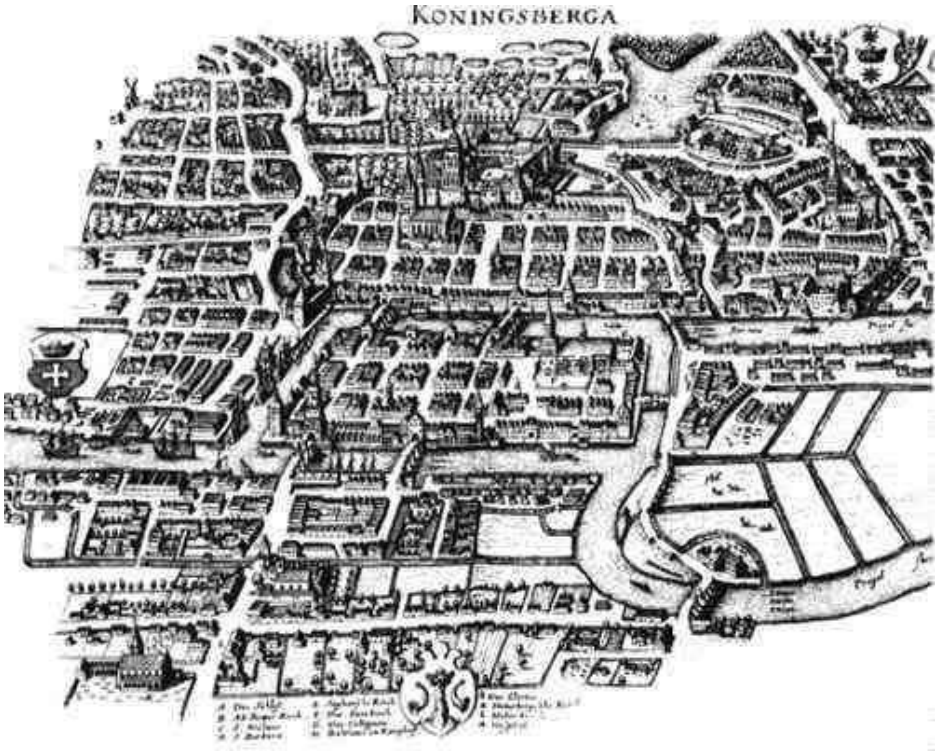
Founder/co-organizer of the PHP Berkshire User Group

I also act

Graphs are everywhere

The Seven Bridges of Königsberg

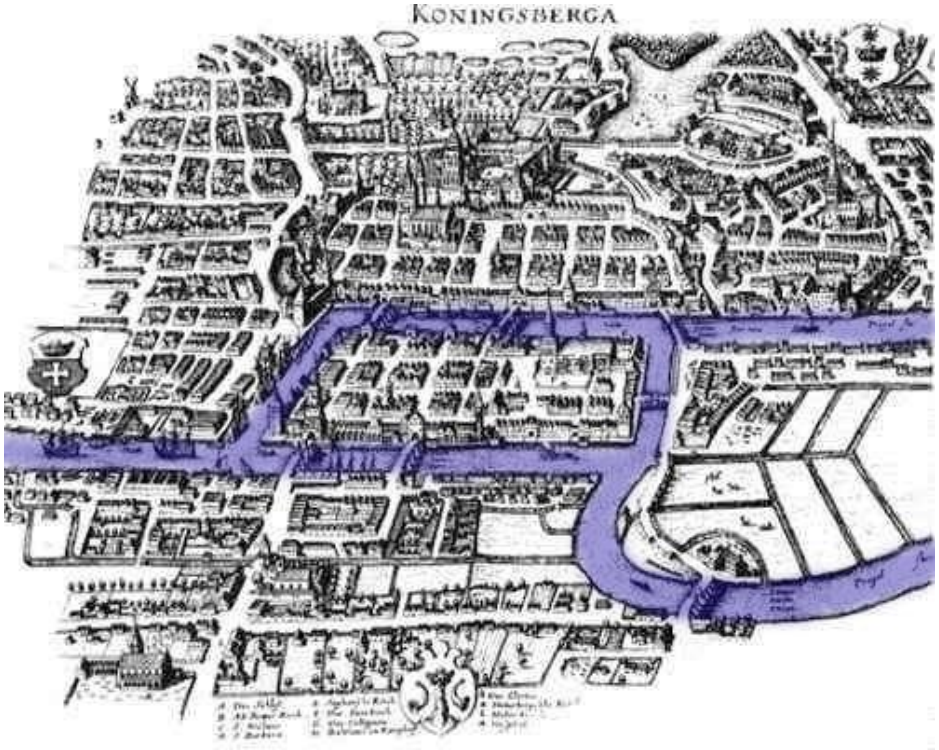
Leonhard Euler, Prussia,
1736



The Seven Bridges of Königsberg

Leonhard Euler, Prussia,
1736

Now Kaliningrad, Russia;
built around the Pregel River

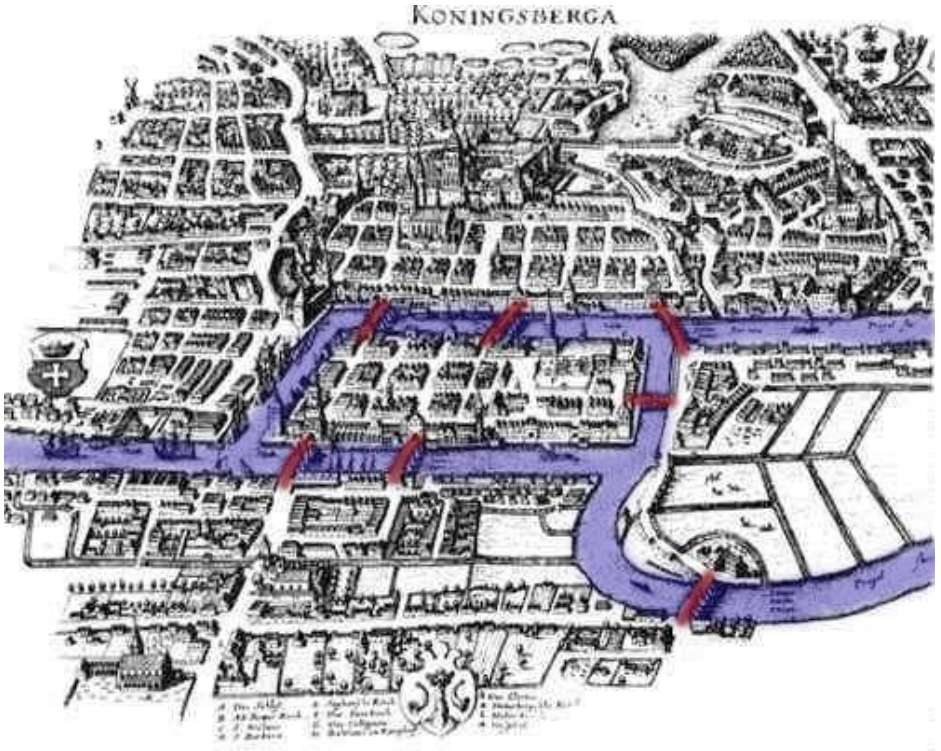


The Seven Bridges of Königsberg

Leonhard Euler, Prussia, 1736

Now Kaliningrad, Russia; built around the Pregel River

“Can you walk through the city, crossing each bridge only once?”



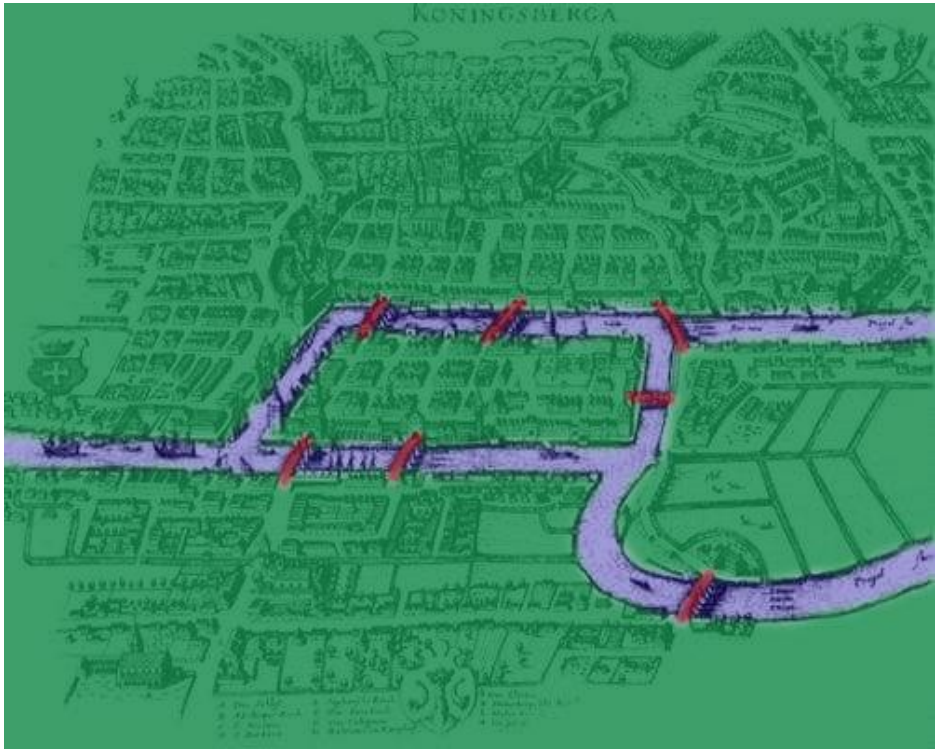
The Seven Bridges of Königsberg

Leonhard Euler, Prussia, 1736

Now Kaliningrad, Russia; built around the Pregel River

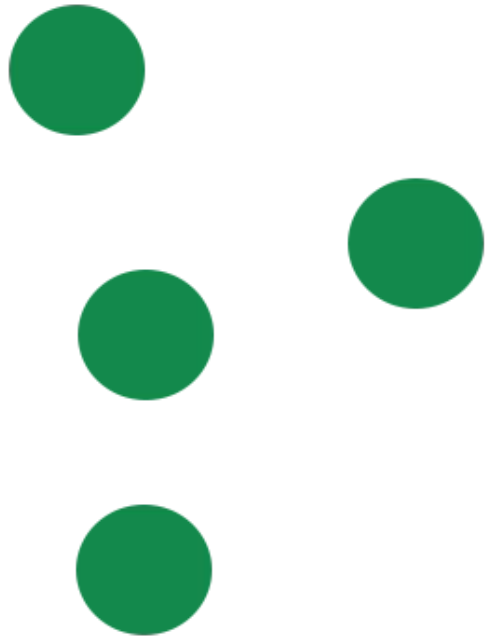
“Can you walk through the city, crossing each bridge only once?”

Reduce to four land-masses, and seven connections



The Seven Bridges of Königsberg

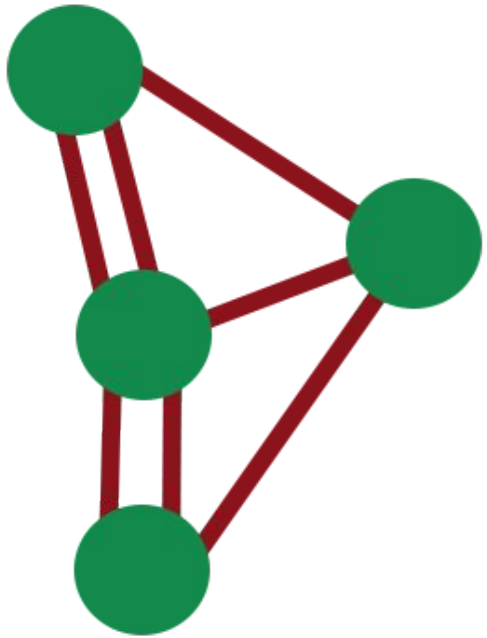
Nodes (or vertices)



The Seven Bridges of Königsberg

Nodes (or vertices)

Edges

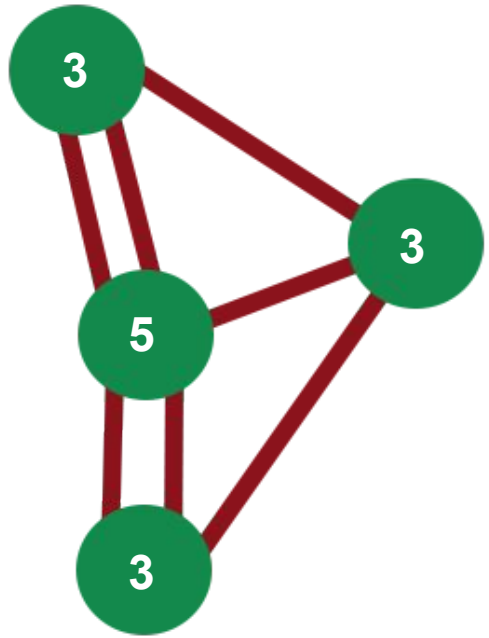


The Seven Bridges of Königsberg

Nodes (or vertices)

Edges

Degrees



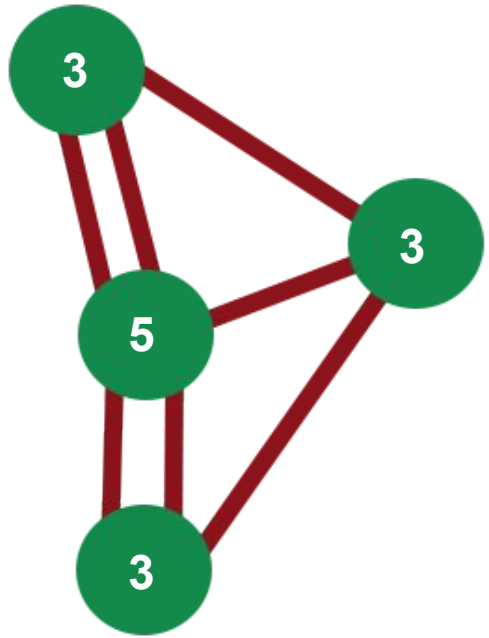
The Seven Bridges of Königsberg

Nodes (or vertices)

Edges

Degrees

Example of an undirected graph - you can go either way over each bridge.



The Seven Bridges of Königsberg

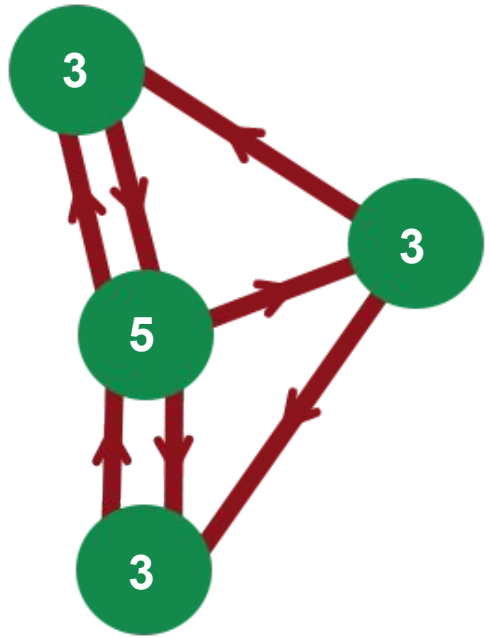
Nodes (or vertices)

Edges

Degrees

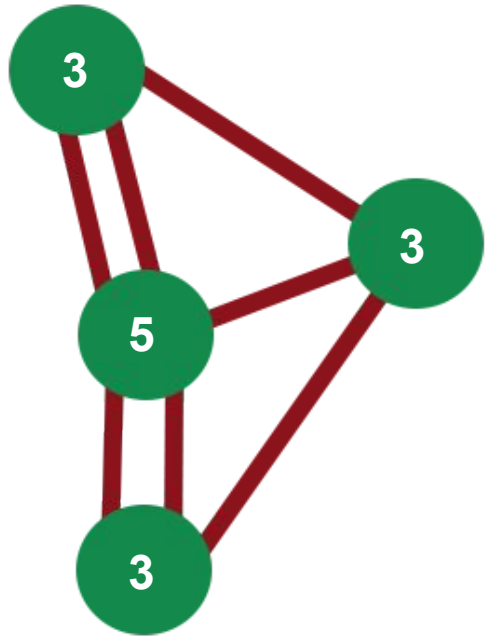
Example of an undirected graph - you can go either way over each bridge.

A one-way system would be a directed graph.



The Seven Bridges of Königsberg

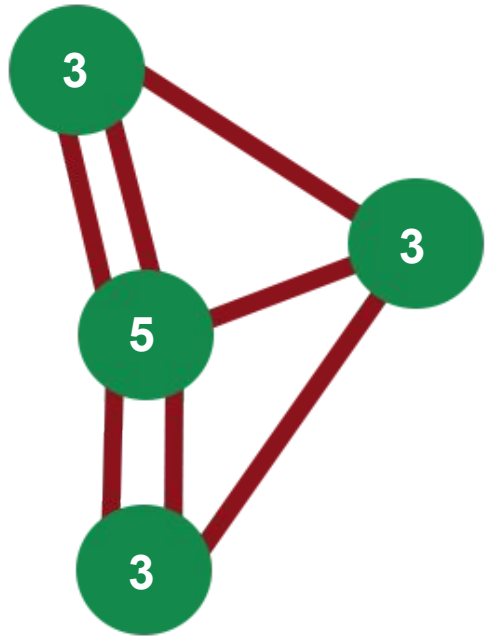
To get into and out of a node, there must be an even number of edges; its *degree* must be even.



The Seven Bridges of Königsberg

To get into and out of a node, there must be an even number of edges; its *degree* must be even.

Only the start and end can have odd degrees.

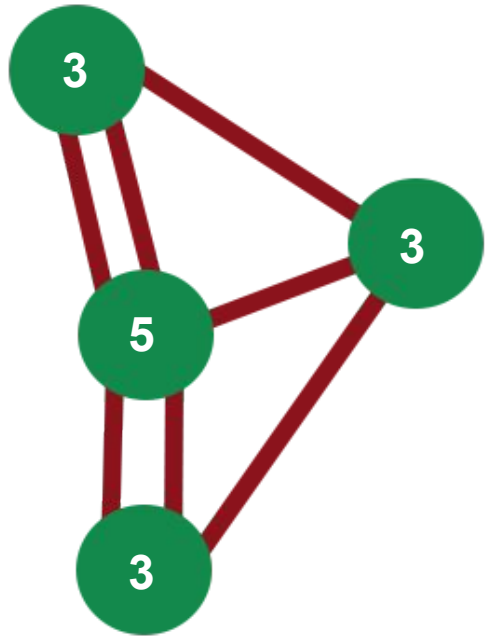


The Seven Bridges of Königsberg

To get into and out of a node, there must be an even number of edges; its *degree* must be even.

Only the start and end can have odd degrees.

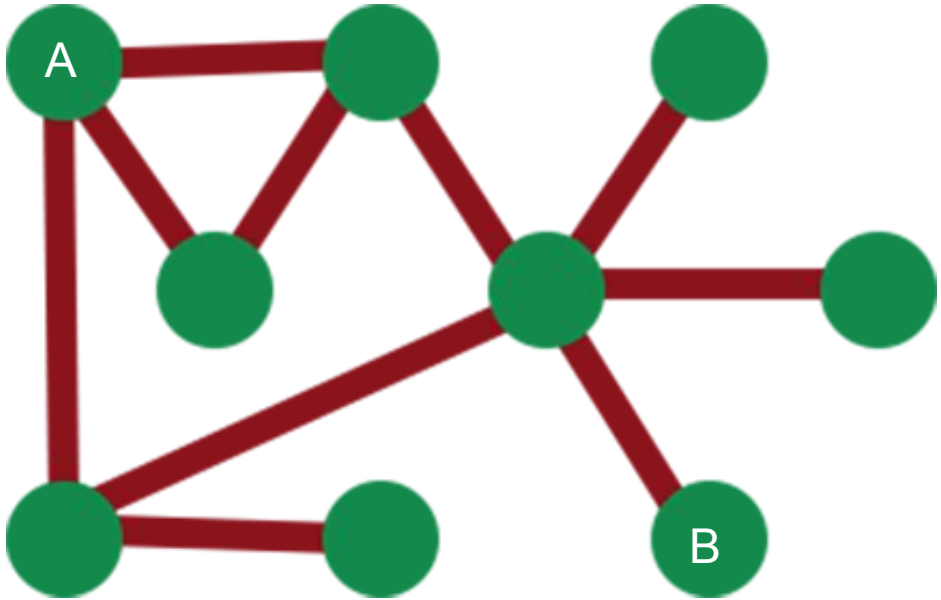
Therefore, only possible if zero or two nodes of odd degree - these will be start and end.



Pathfinding

Pathfinding

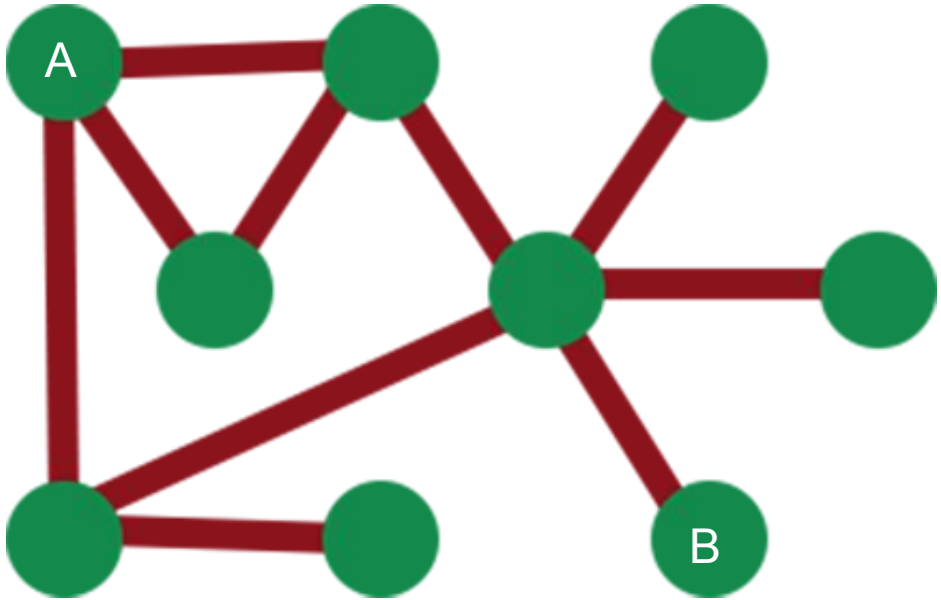
Problem: Find your way from arbitrary node A to arbitrary node B.



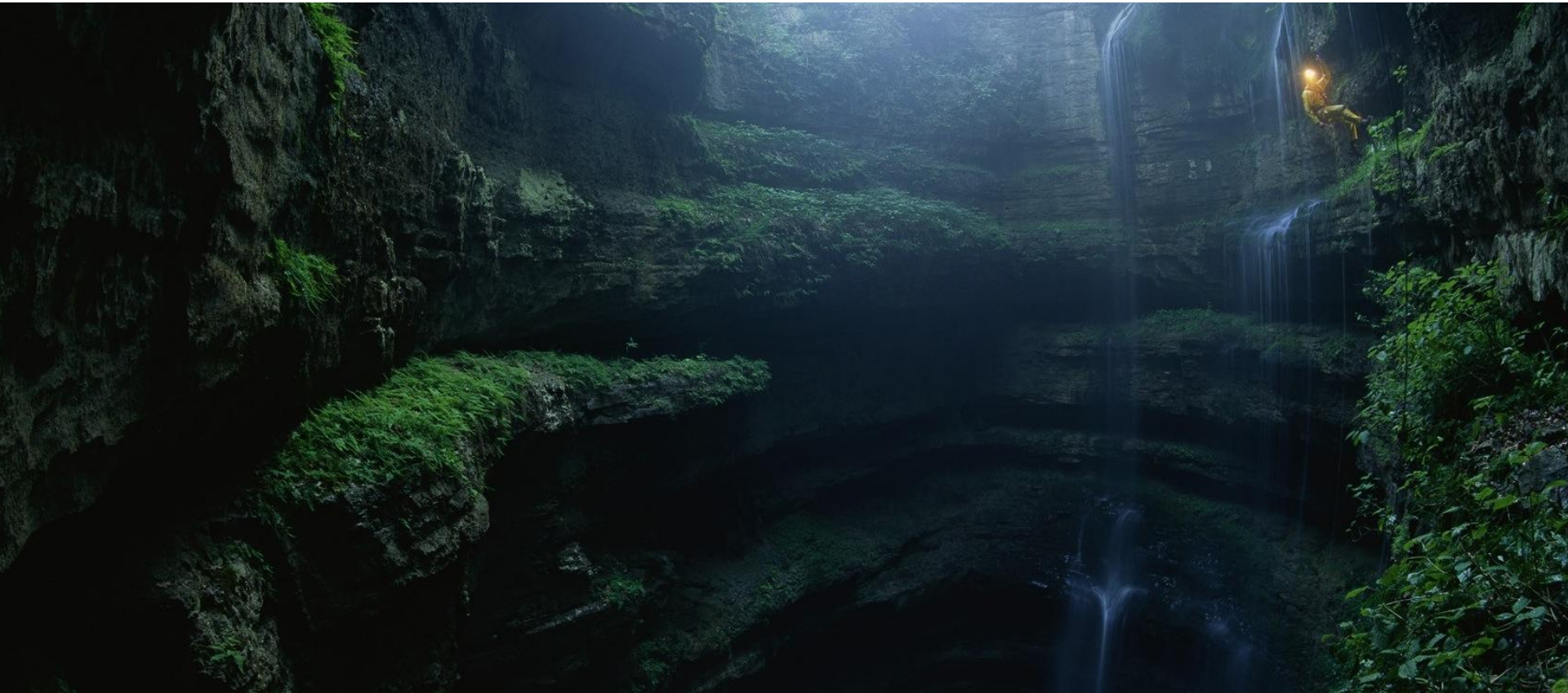
Pathfinding

Problem: Find your way from arbitrary node A to arbitrary node B.

Two approaches: depth-first and breadth-first.



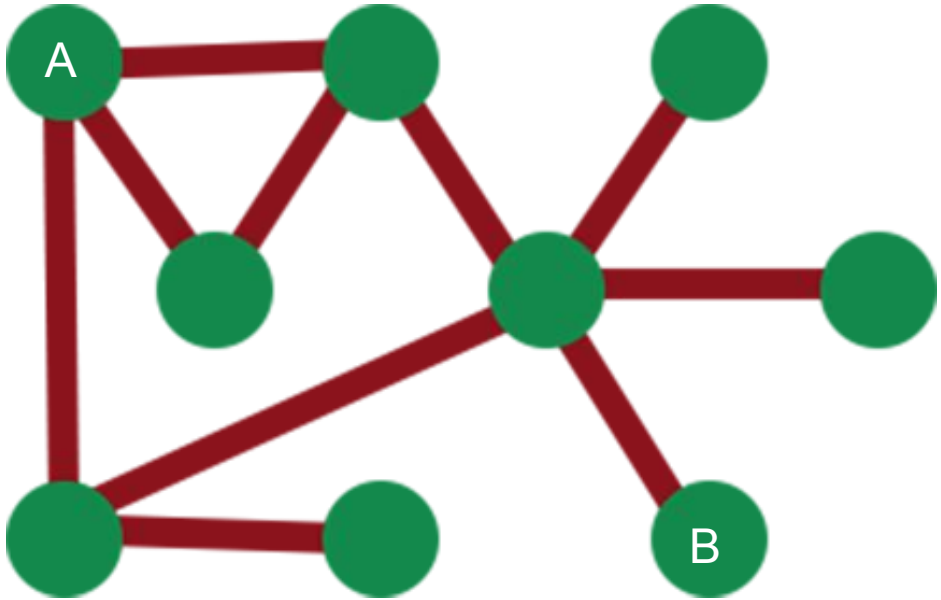
Pathfinding: Depth-first



Pathfinding: Depth-first

Problem: Find your way from arbitrary node A to arbitrary node B.

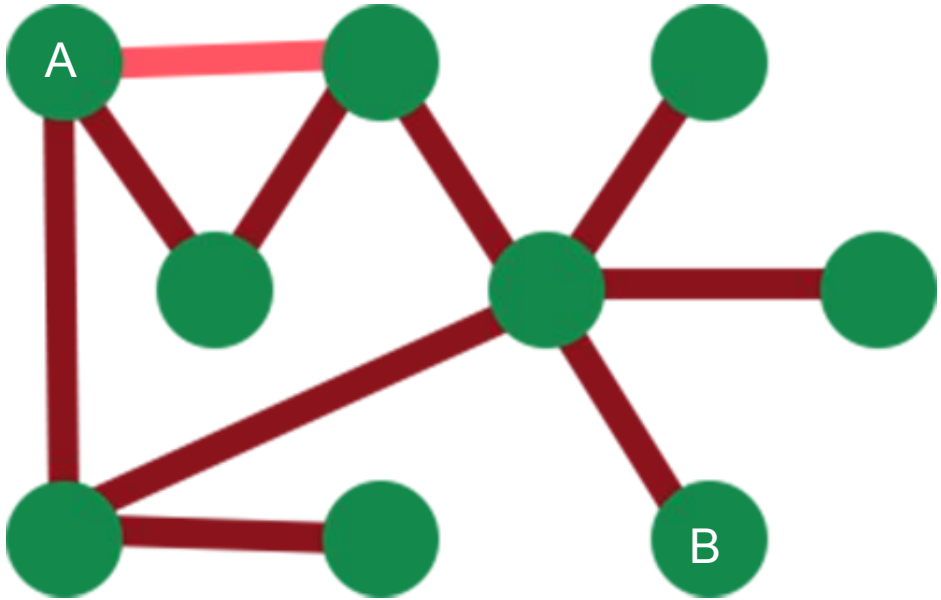
Depth-first: follow the graph from node A until you reach node B or a dead end and you backtrack.



Pathfinding: Depth-first

Problem: Find your way from arbitrary node A to arbitrary node B.

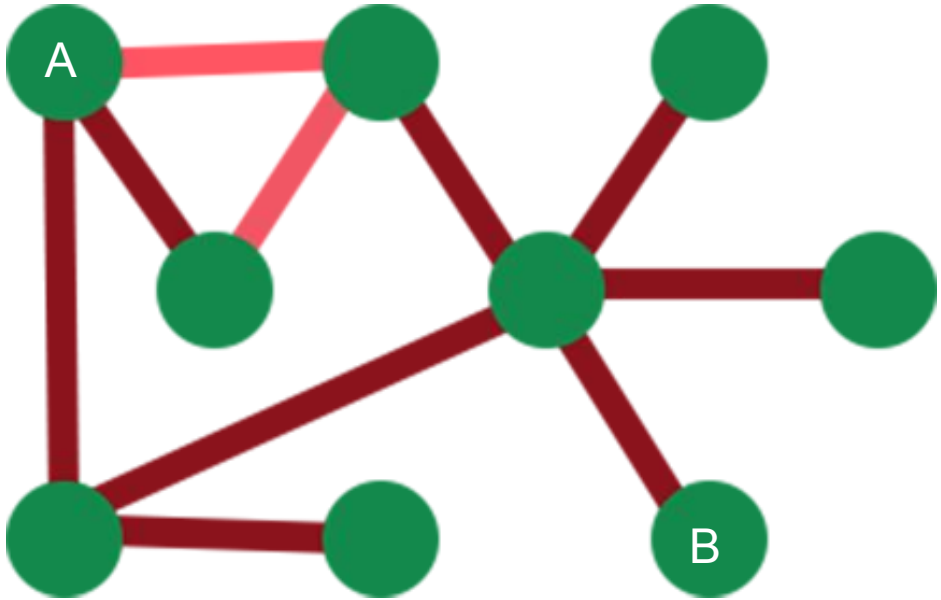
Depth-first: follow the graph from node A until you reach node B or a dead end and you backtrack.



Pathfinding: Depth-first

Problem: Find your way from arbitrary node A to arbitrary node B.

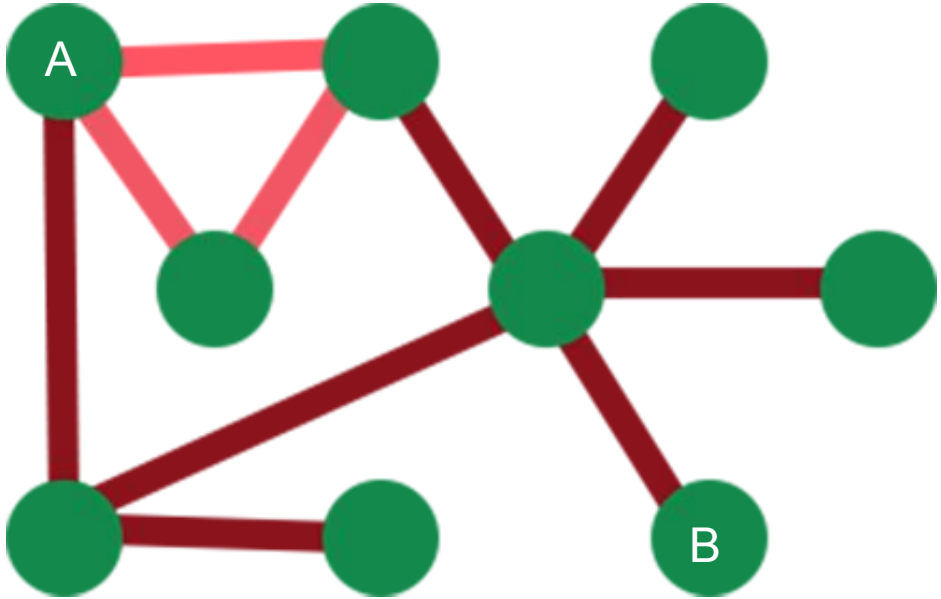
Depth-first: follow the graph from node A until you reach node B or a dead end and you backtrack.



Pathfinding: Depth-first

Problem: Find your way from arbitrary node A to arbitrary node B.

A cycle is where a collection of nodes are interconnected such that a path will loop back on itself.

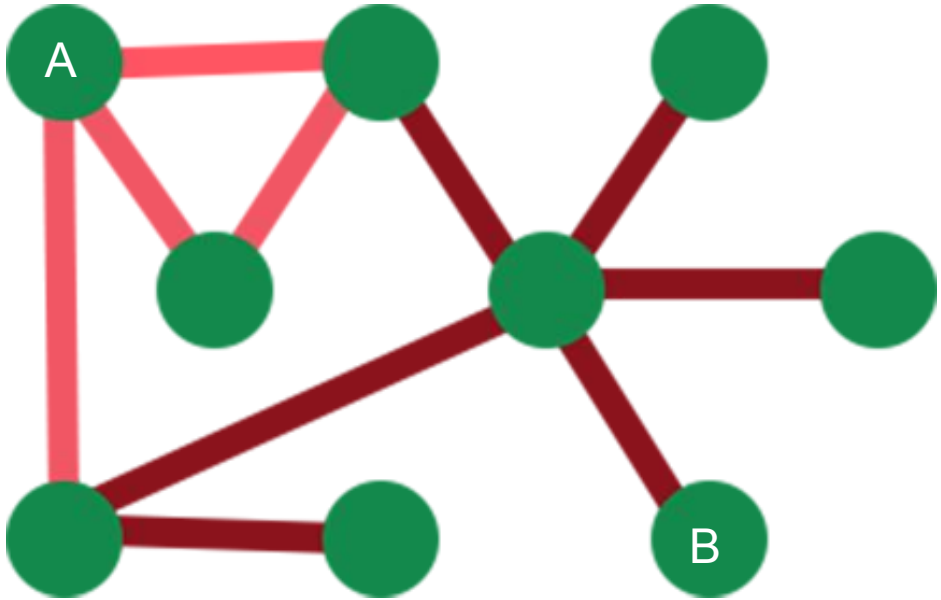


Pathfinding: Depth-first

Problem: Find your way from arbitrary node A to arbitrary node B.

A cycle is where a collection of nodes are interconnected such that a path will loop back on itself.

Cope by tracking which nodes have already been traversed.

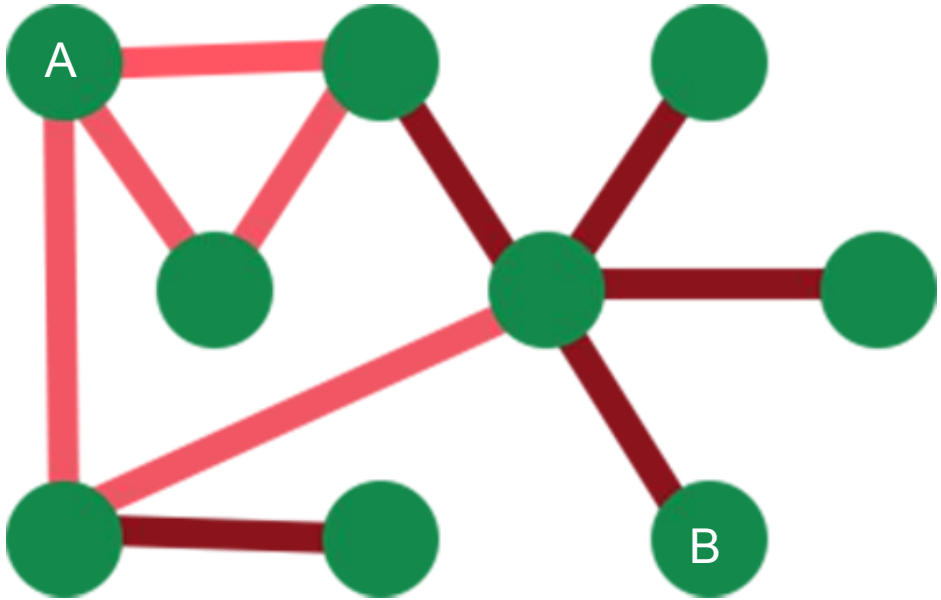


Pathfinding: Depth-first

Problem: Find your way from arbitrary node A to arbitrary node B.

A cycle is where a collection of nodes are interconnected such that a path will loop back on itself.

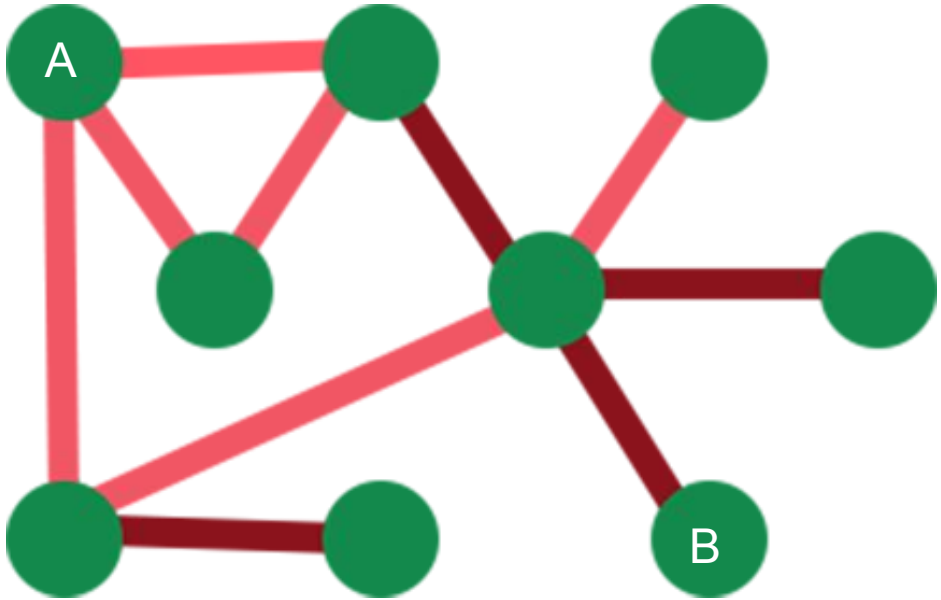
Cope by tracking which nodes have already been traversed.



Pathfinding: Depth-first

Problem: Find your way from arbitrary node A to arbitrary node B.

We've now hit a dead-end, and can go no further - so backtrack and try again.

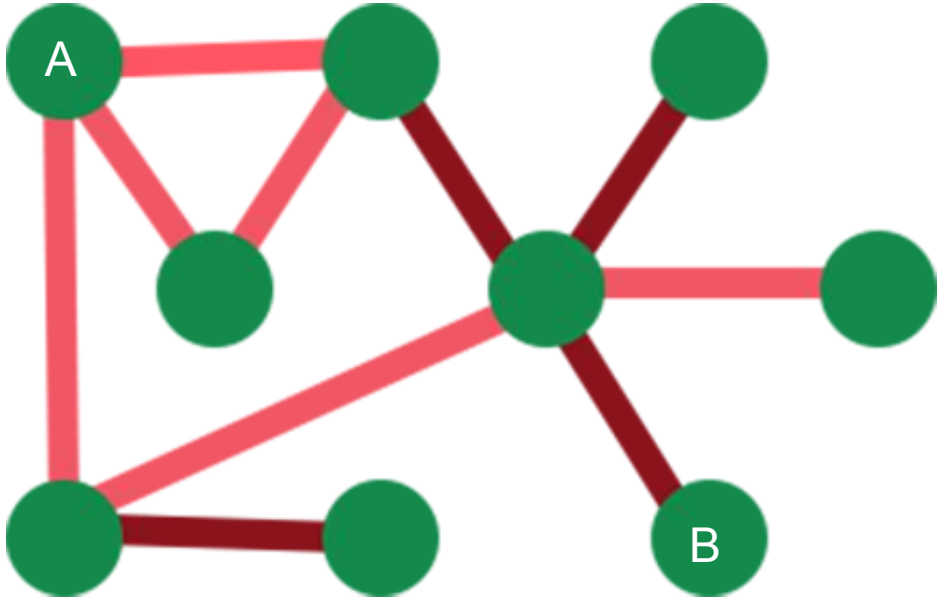


Pathfinding: Depth-first

Problem: Find your way from arbitrary node A to arbitrary node B.

Best-case: we get to node B as quickly as possible.

Worst-case: we get to node B last, having explored the whole graph!

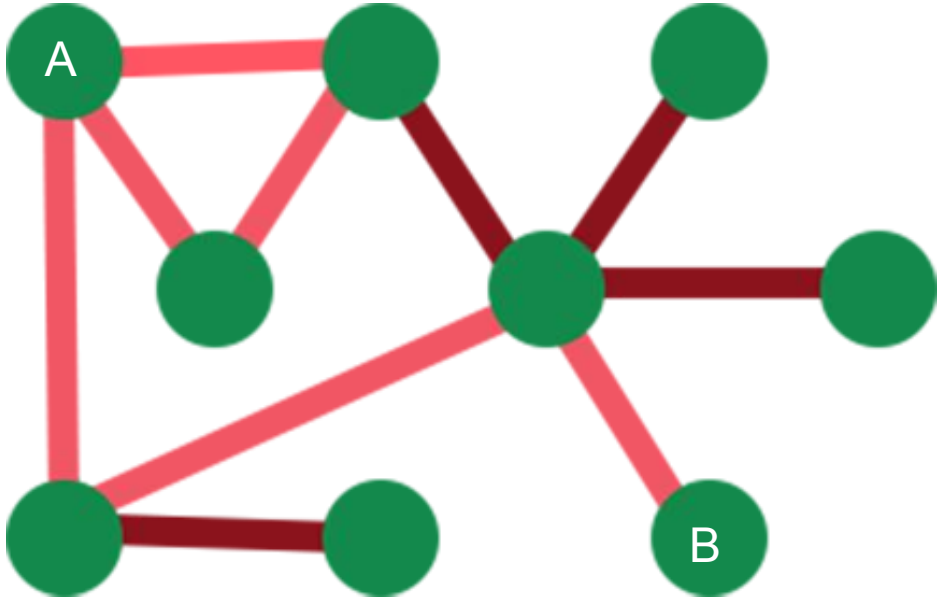


Pathfinding: Depth-first

Problem: Find your way from arbitrary node A to arbitrary node B.

Best-case: we get to node B as quickly as possible.

Worst-case: we get to node B last, having explored the whole graph!



Depth-first: Mazes

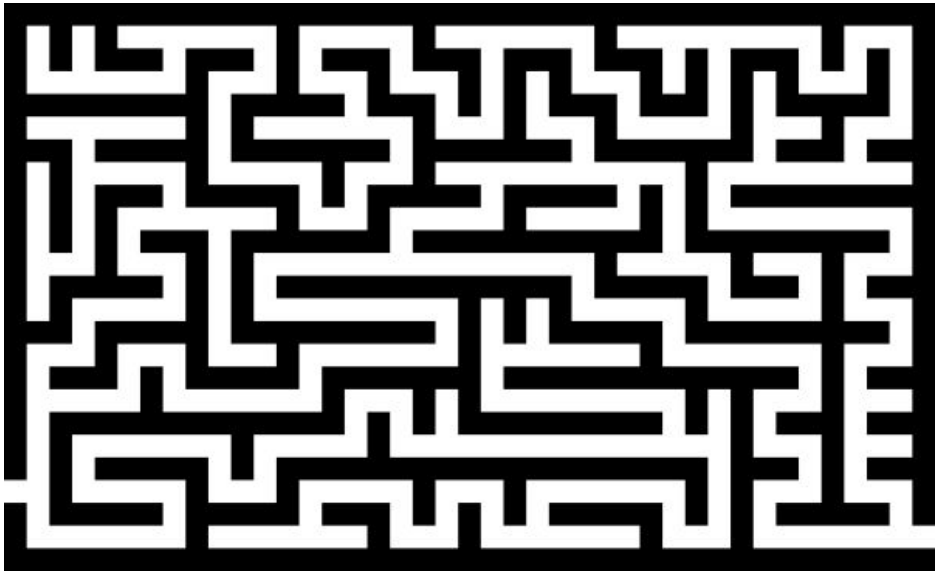
Depth-first: Mazes

Solving mazes with depth-first:

At a junction, take first path on left, keep going.

If you hit a dead end, backtrack until you can take another turn.

Repeat...

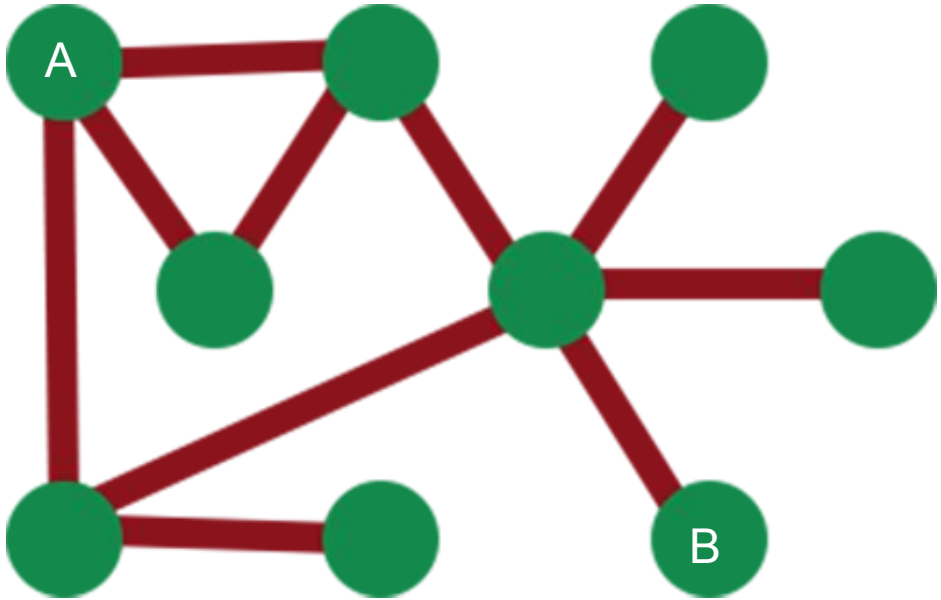


Pathfinding: Breadth-first

Pathfinding: Breadth-first

Problem: Find your way from arbitrary node A to arbitrary node B.

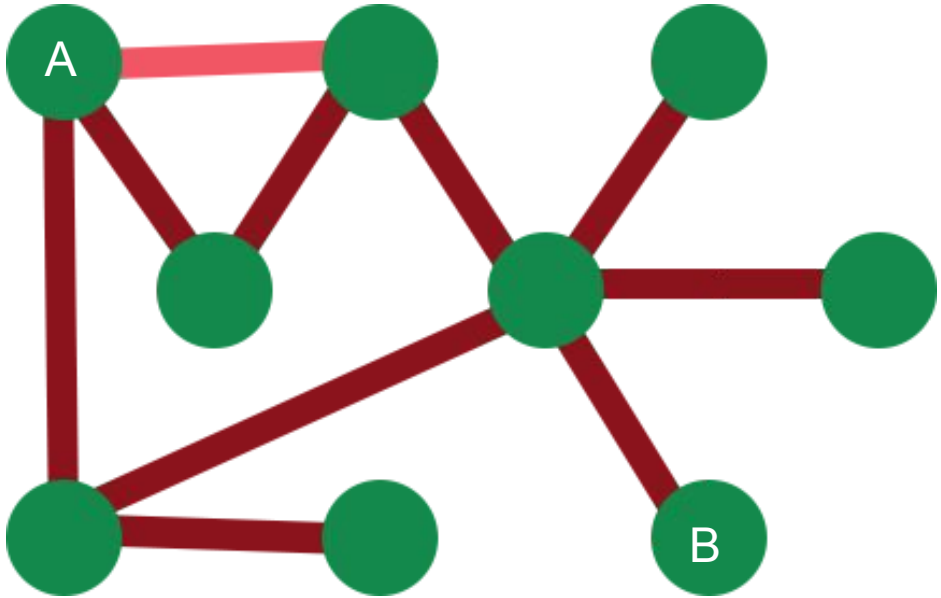
Breadth-first: check each node at depth 1, depth 2 etc. until node B is found.



Pathfinding: Breadth-first

Problem: Find your way from arbitrary node A to arbitrary node B.

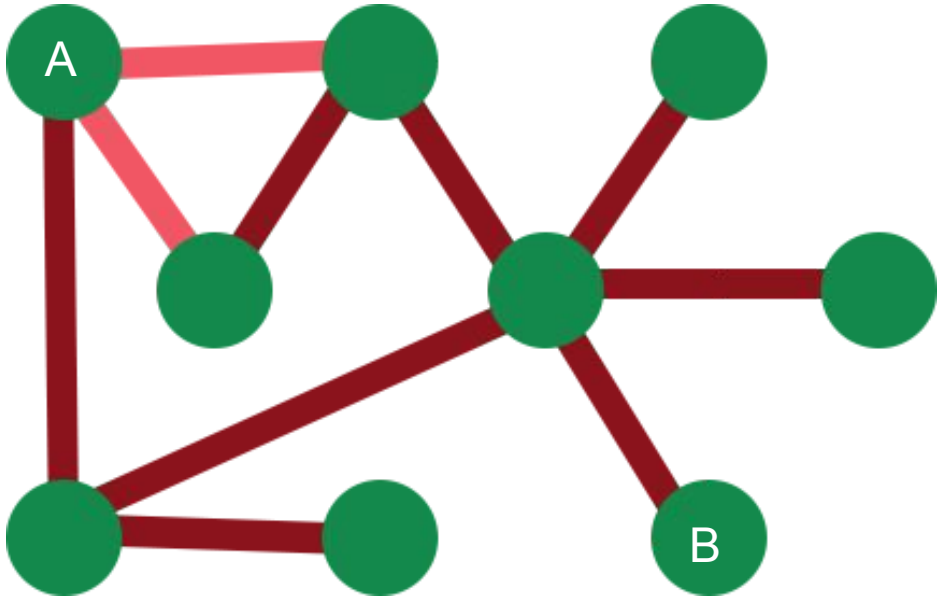
Breadth-first: check each node at depth 1, depth 2 etc. until node B is found.



Pathfinding: Breadth-first

Problem: Find your way from arbitrary node A to arbitrary node B.

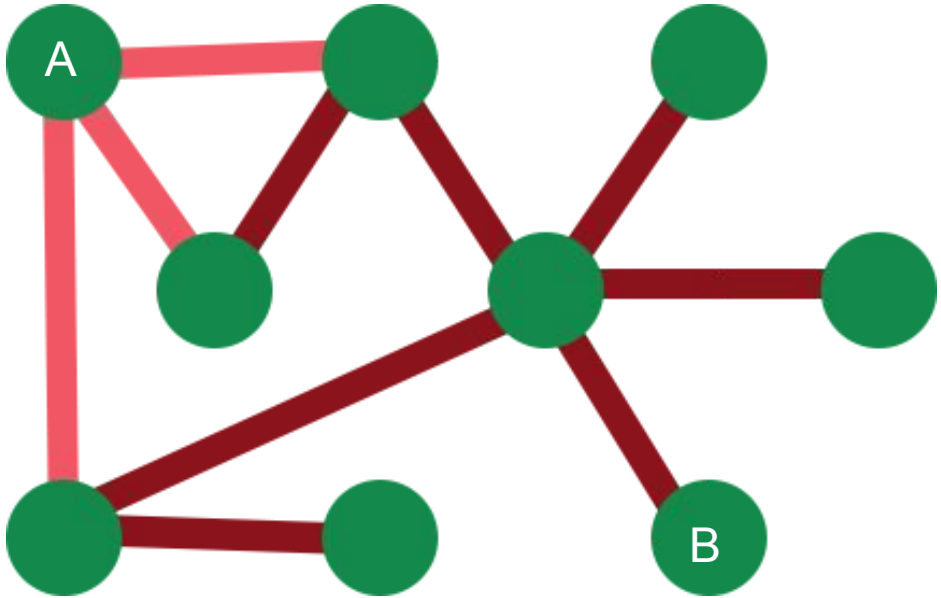
Breadth-first: check each node at depth 1, depth 2 etc. until node B is found.



Pathfinding: Breadth-first

Problem: Find your way from arbitrary node A to arbitrary node B.

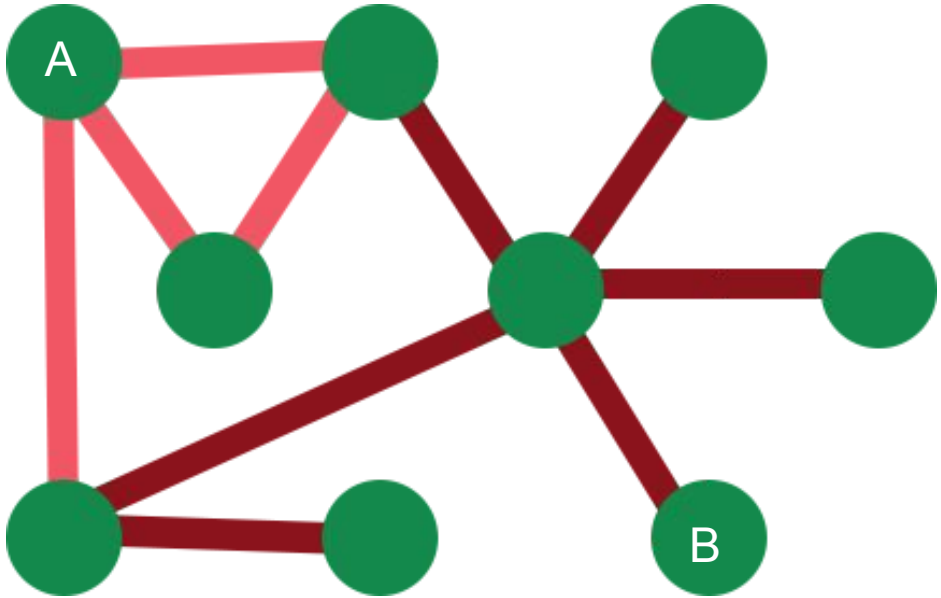
Breadth-first: check each node at depth 1, depth 2 etc. until node B is found.



Pathfinding: Breadth-first

Problem: Find your way from arbitrary node A to arbitrary node B.

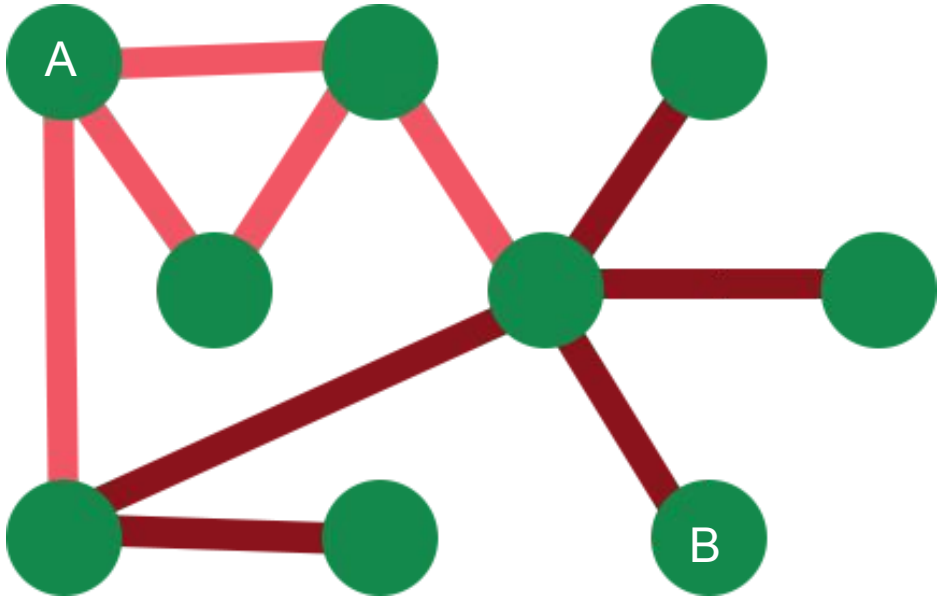
Breadth-first: check each node at depth 1, depth 2 etc. until node B is found.



Pathfinding: Breadth-first

Problem: Find your way from arbitrary node A to arbitrary node B.

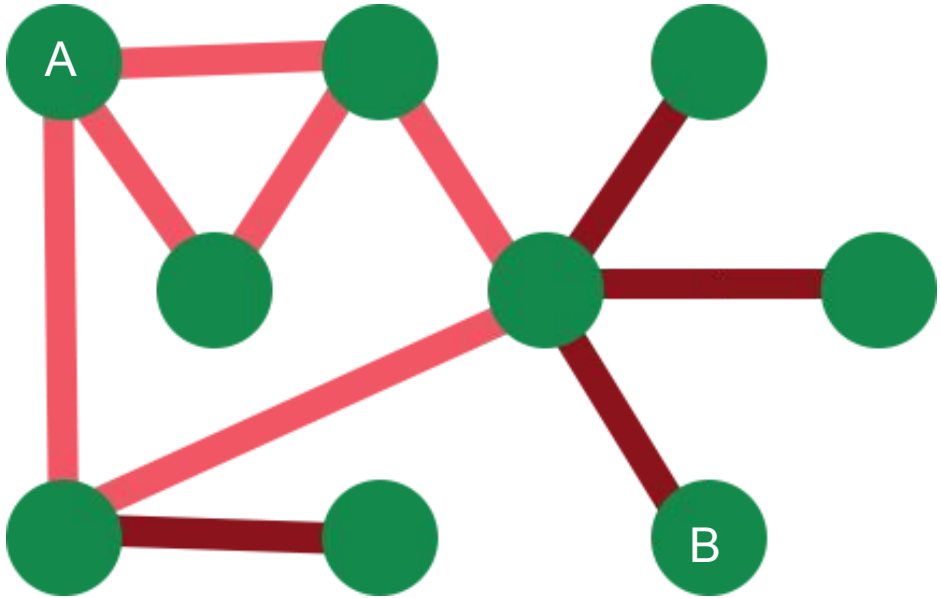
Breadth-first: check each node at depth 1, depth 2 etc. until node B is found.



Pathfinding: Breadth-first

Problem: Find your way from arbitrary node A to arbitrary node B.

Breadth-first: check each node at depth 1, depth 2 etc. until node B is found.

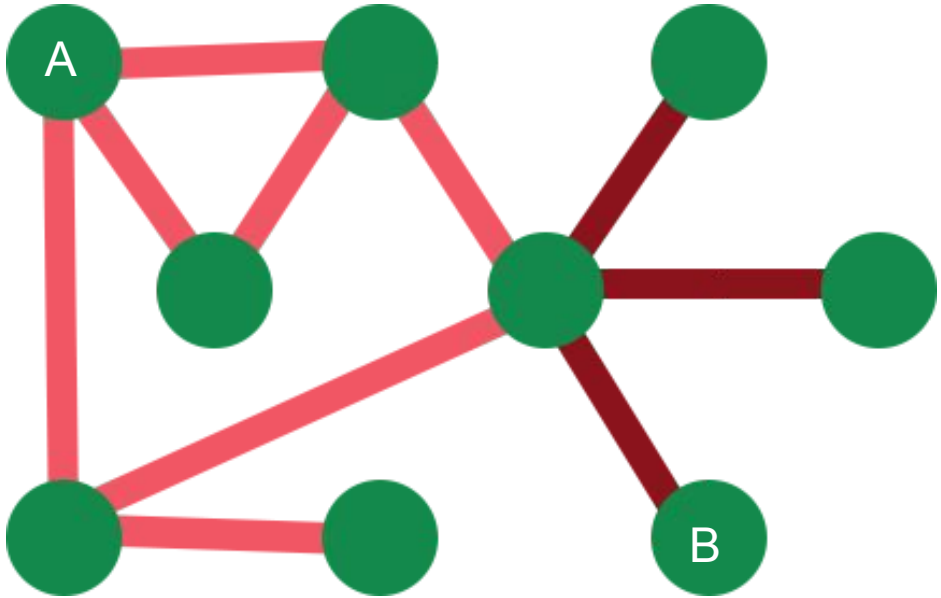


Pathfinding: Breadth-first

Problem: Find your way from arbitrary node A to arbitrary node B.

Breadth-first: check each node at depth 1, depth 2 etc. until node B is found.

This approach is still susceptible to cycles.

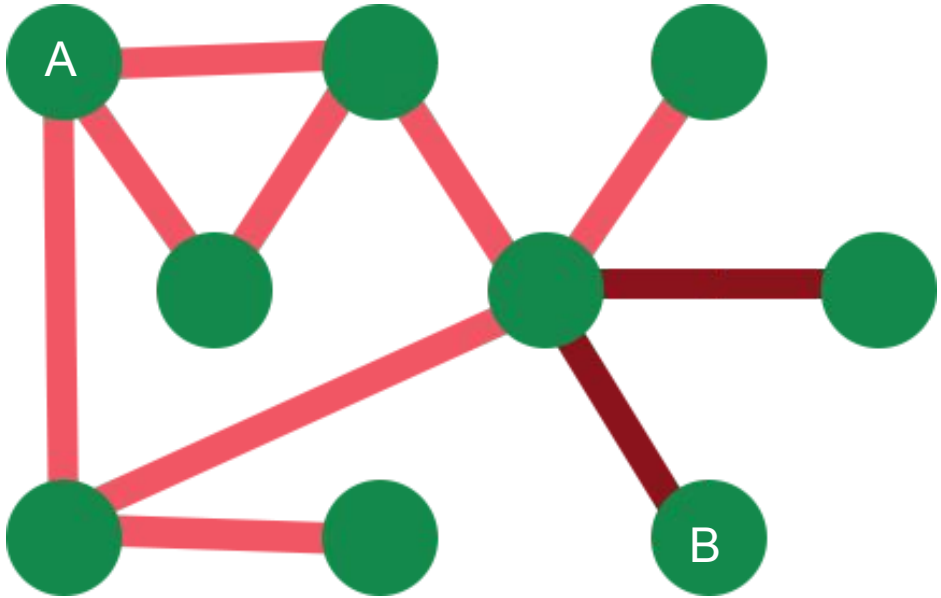


Pathfinding: Breadth-first

Problem: Find your way from arbitrary node A to arbitrary node B.

Breadth-first: check each node at depth 1, depth 2 etc. until node B is found.

This approach is still susceptible to cycles.

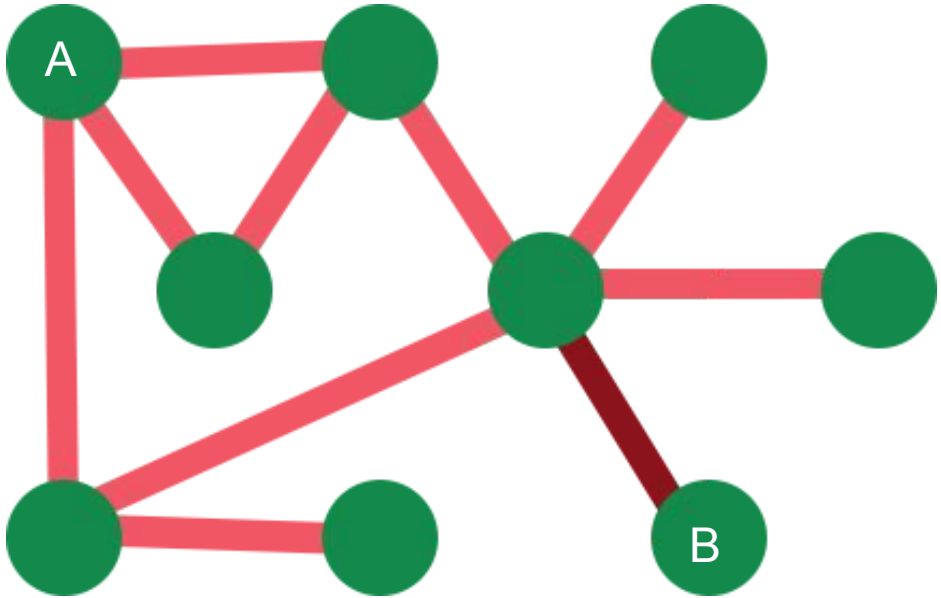


Pathfinding: Breadth-first

Problem: Find your way from arbitrary node A to arbitrary node B.

Breadth-first: check each node at depth 1, depth 2 etc. until node B is found.

This can be slower, but will eventually find the shortest path.

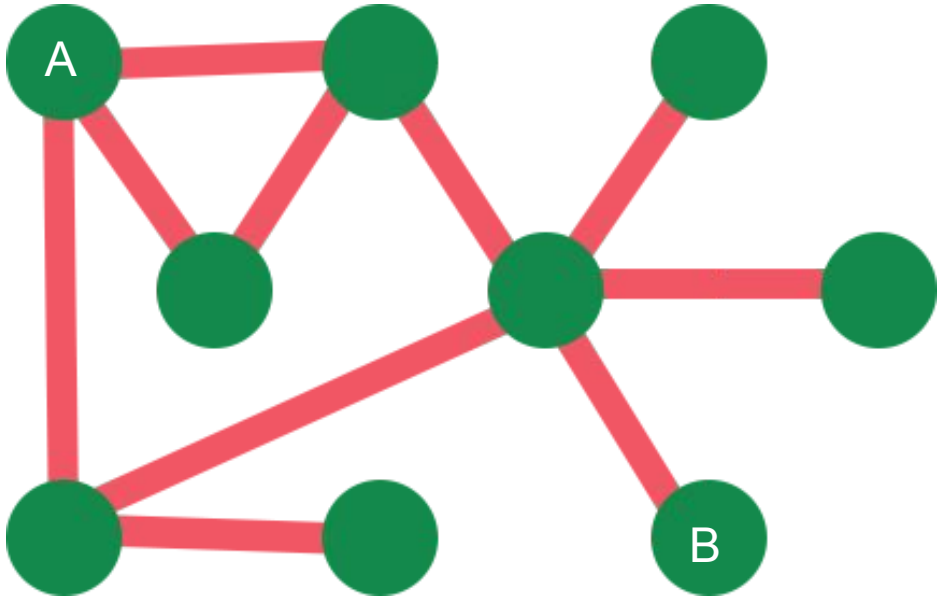


Pathfinding: Breadth-first

Problem: Find your way from arbitrary node A to arbitrary node B.

Breadth-first: check each node at depth 1, depth 2 etc. until node B is found.

This can be slower, but will eventually find the shortest path.



Breadth-first: The six degrees of Kevin Bacon

Breadth-first: The six degrees of Kevin Bacon

An actor’s Bacon Number is the number of steps between them and Kevin Bacon via working with other actors on movies.

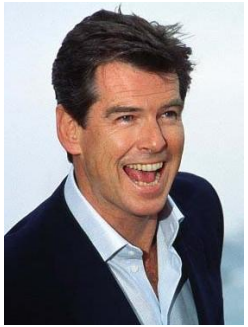
Breadth-first: The six degrees of Kevin Bacon

An actor's Bacon Number is the number of movies between them and Kevin Bacon via working with other actors.

Eg. Actor A worked with Actor B on movie 1; Actor B worked with Actor C on movie 2; Actor C worked with Kevin Bacon on movie 3 therefore Actor A's *Bacon Number* is 3

Breadth-first: The six degrees of Kevin Bacon

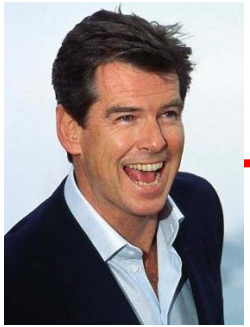
Pierce Brosnan has a Bacon Number of 2



Pierce Brosnan

Breadth-first: The six degrees of Kevin Bacon

Pierce Brosnan has a Bacon Number of 2



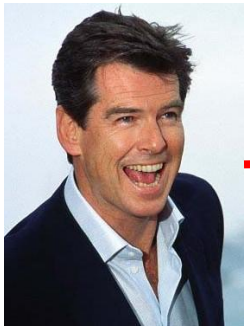
Was in

Pierce Brosnan

After the Sunset

Breadth-first: The six degrees of Kevin Bacon

Pierce Brosnan has a Bacon Number of 2



Pierce Brosnan

Was in

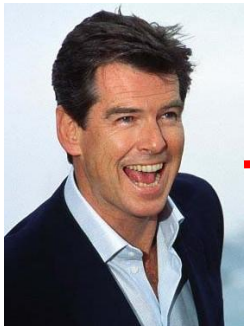
After the Sunset

With

Chris Penn

Breadth-first: The six degrees of Kevin Bacon

Pierce Brosnan has a Bacon Number of 2



Pierce Brosnan

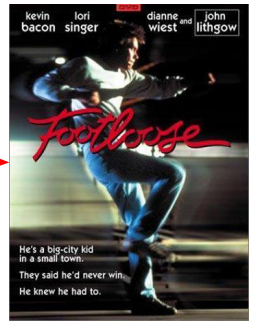
Was in

After the Sunset

With

Chris Penn

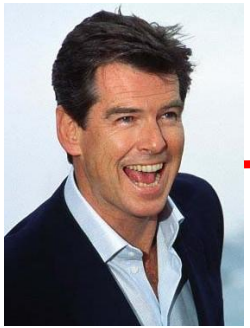
Was in



Footloose

Breadth-first: The six degrees of Kevin Bacon

Pierce Brosnan has a Bacon Number of 2



Pierce Brosnan

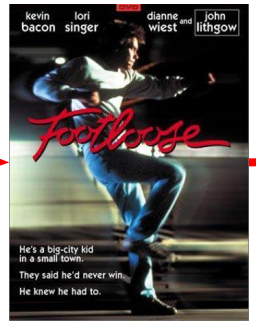
Was in

After the Sunset

With

Chris Penn

Was in



Footloose

With

Kevin Bacon

Breadth-first: The six degrees of Kevin Bacon

A graph to represent the search-space of IMDB (say) would be huge: 3m+ actors; 2m+ movies and TV shows.

Breadth-first: The six degrees of Kevin Bacon

A graph to represent the search-space of IMDB (say) would be huge: 3m+ actors; 2m+ movies and TV shows.

Depth-first search would be more susceptible to its worst-case scenario - it would just get lost without supreme luck!

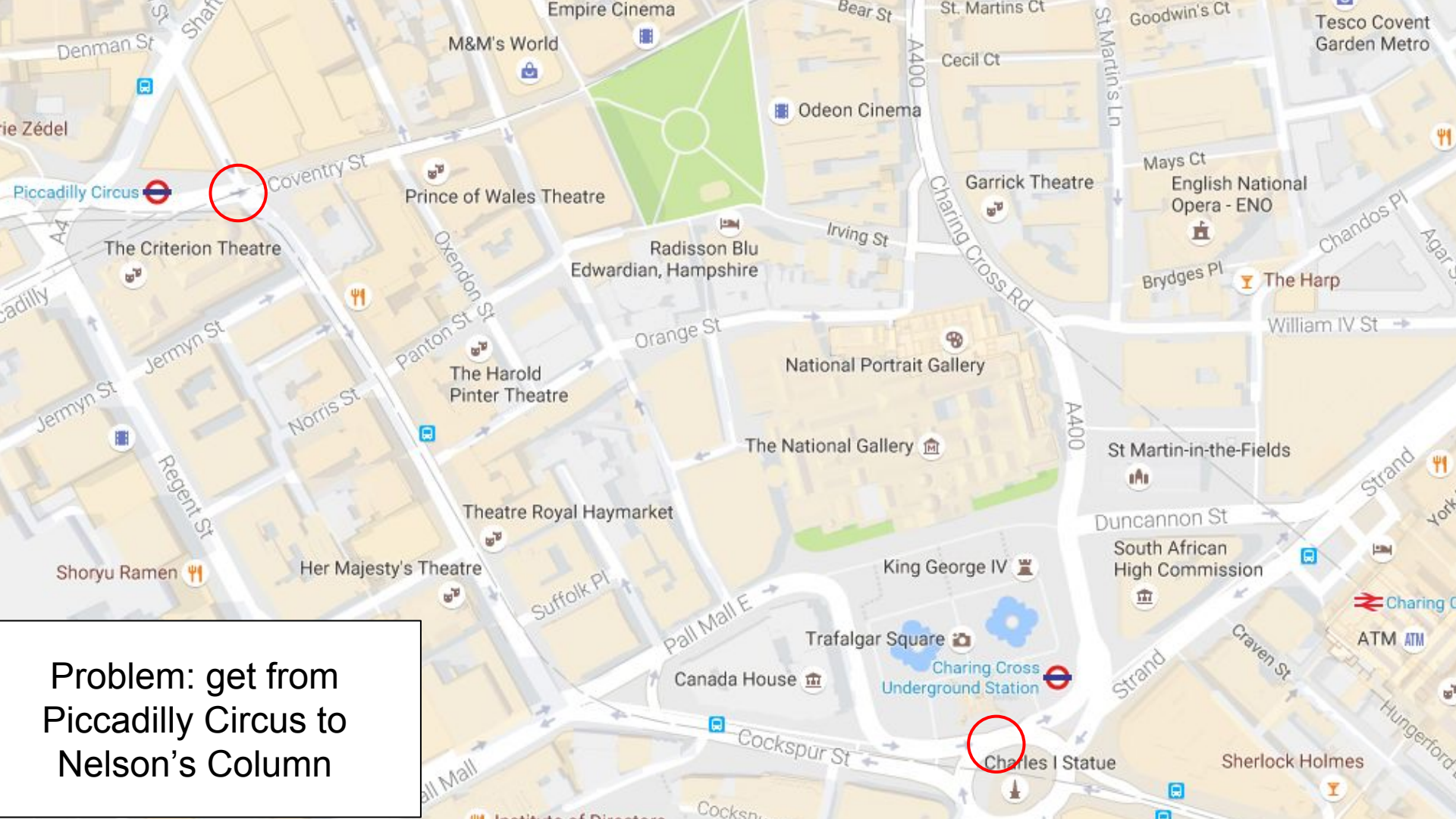
Breadth-first: The six degrees of Kevin Bacon

A graph to represent the search-space of IMDB (say) would be huge: 3m+ actors; 2m+ movies and TV shows.

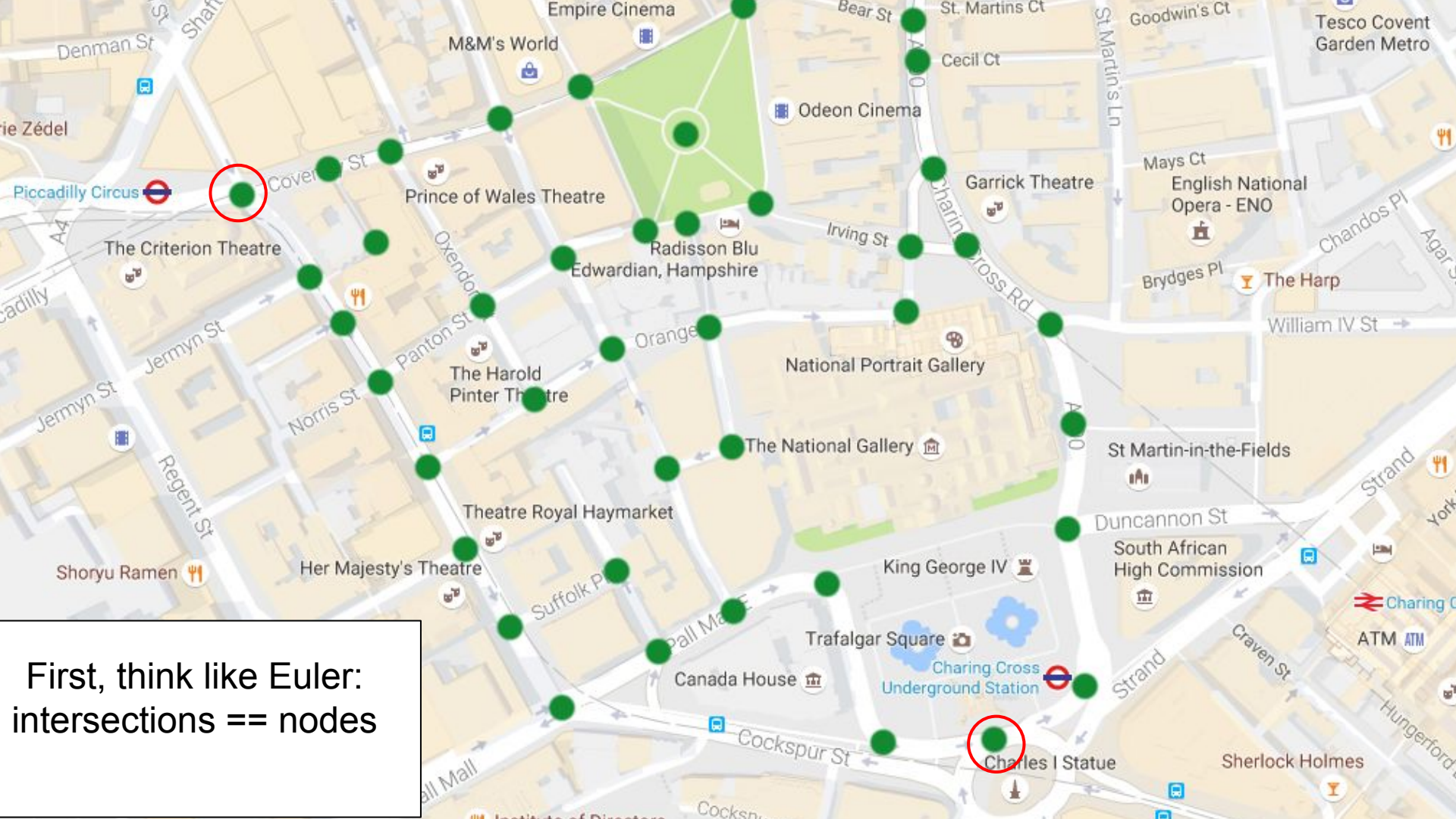
Depth-first search would be more susceptible to its worst-case scenario - it would just get lost without supreme luck!

Breadth-first approaches such as Dijkstra’s algorithm or A* search would fare much better.

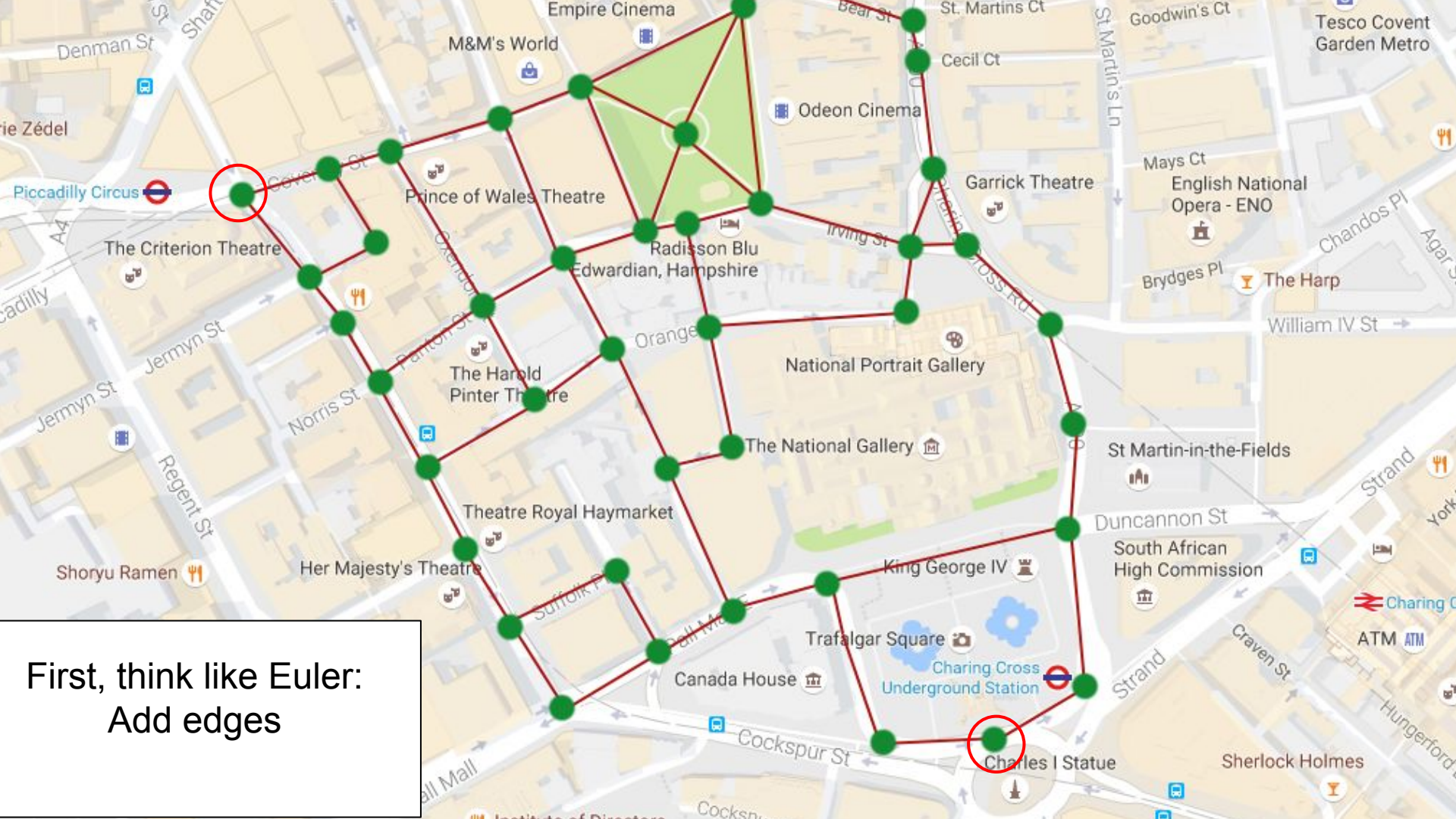
Breadth-first: Dijkstra's algorithm



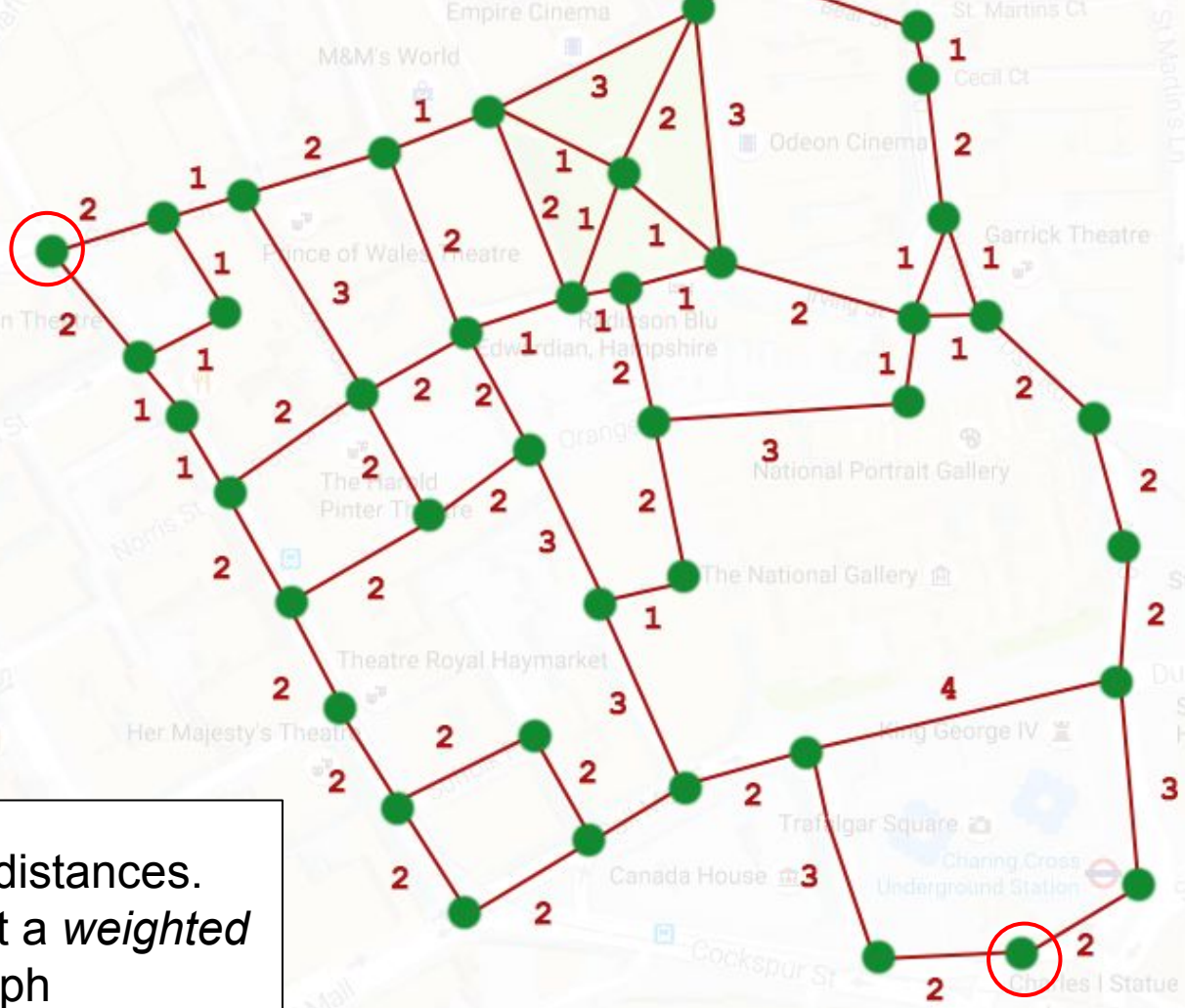
Problem: get from Piccadilly Circus to Nelson's Column



First, think like Euler:
intersections == nodes



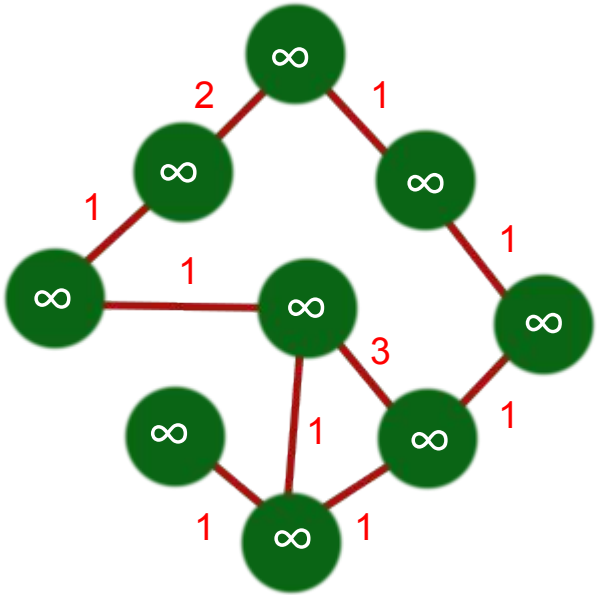
First, think like Euler:
Add edges



Next, add distances. This makes it a *weighted* graph

Dijkstra’s algorithm

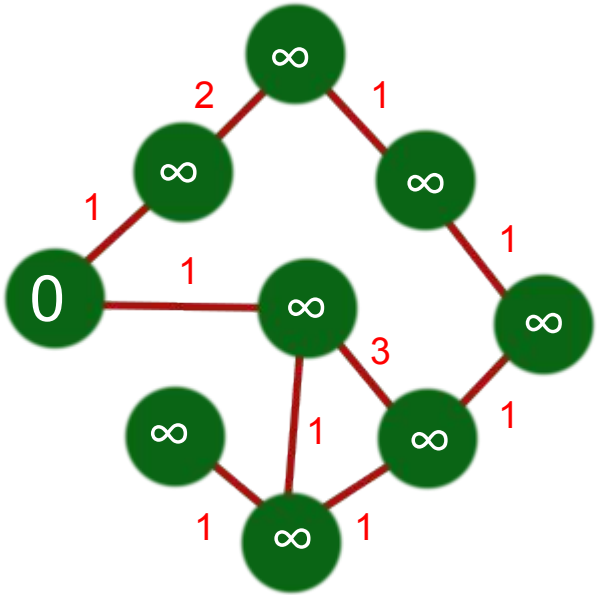
All nodes start with a score of “infinity”



Dijkstra’s algorithm

All nodes start with a score of “infinity”

The start node has score 0

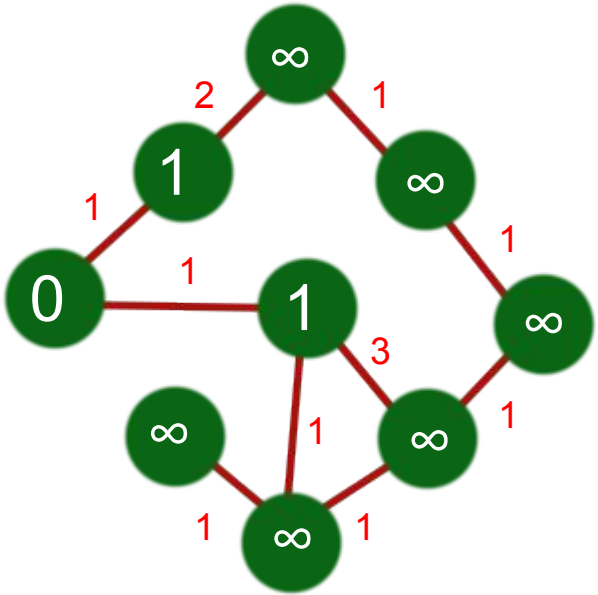


Dijkstra’s algorithm

All nodes start with a score of “infinity”

The start node has score 0

Visit all connected nodes and set their score to node score plus weight if this number is less than current score



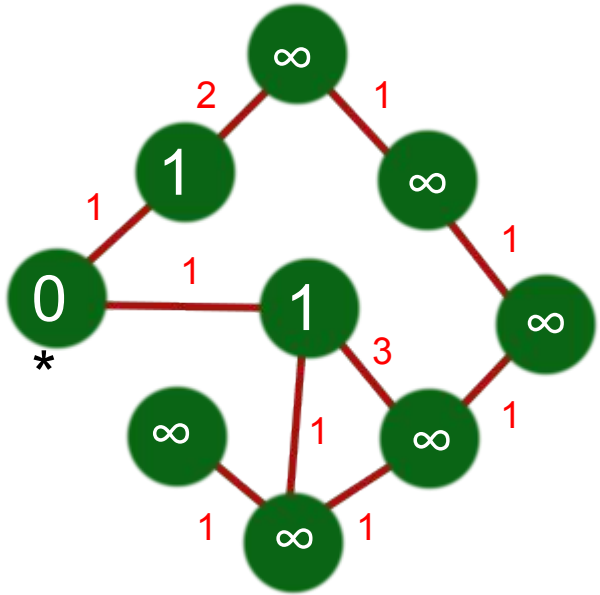
Dijkstra’s algorithm

All nodes start with a score of “infinity”

The start node has score 0

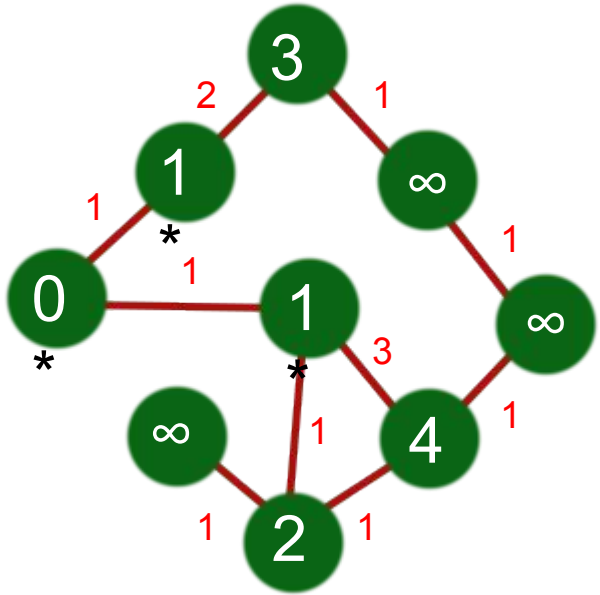
Visit all connected nodes and set their score to node score plus weight if this number is less than current score

Mark as visited



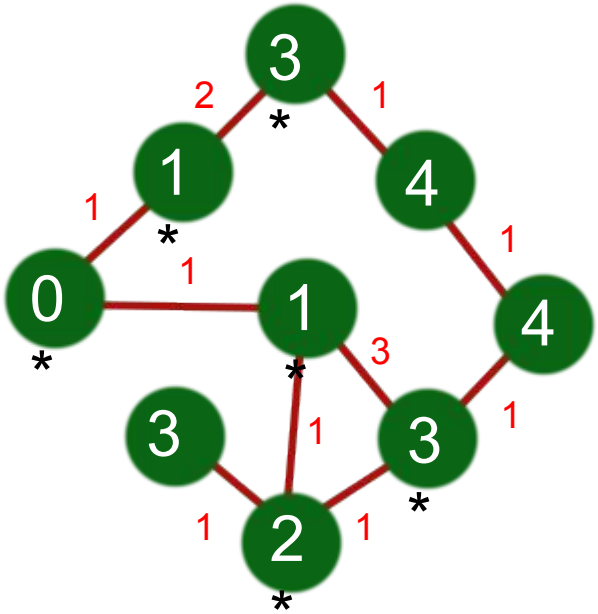
Dijkstra’s algorithm

Once all nodes at current depth *with a score* are marked as visited, proceed to next depth.



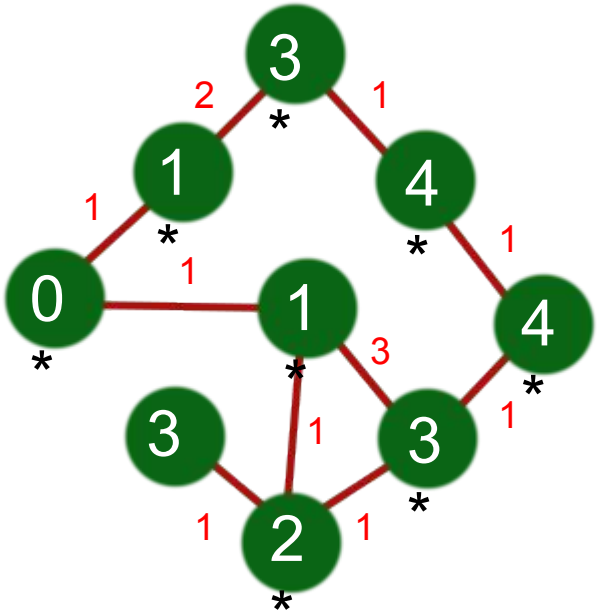
Dijkstra’s algorithm

Once all nodes at current depth *with a score* are marked as visited, proceed to next depth.



Dijkstra’s algorithm

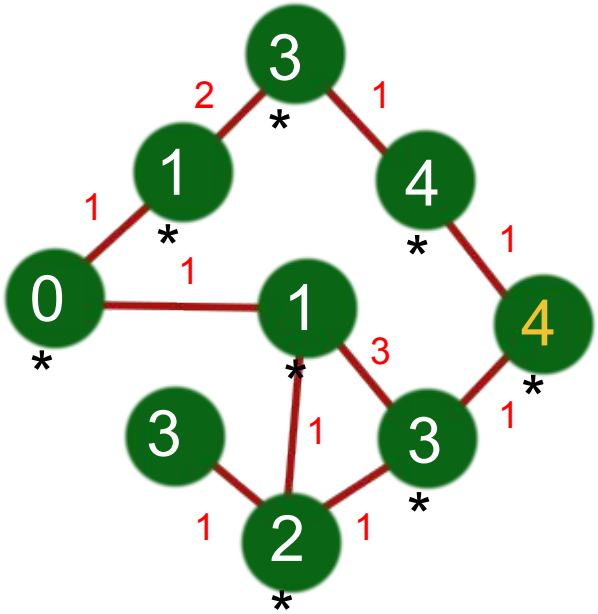
If the current node is the end node, complete the process and then finish.



Dijkstra’s algorithm

If the current node is the end node, complete the process and then finish.

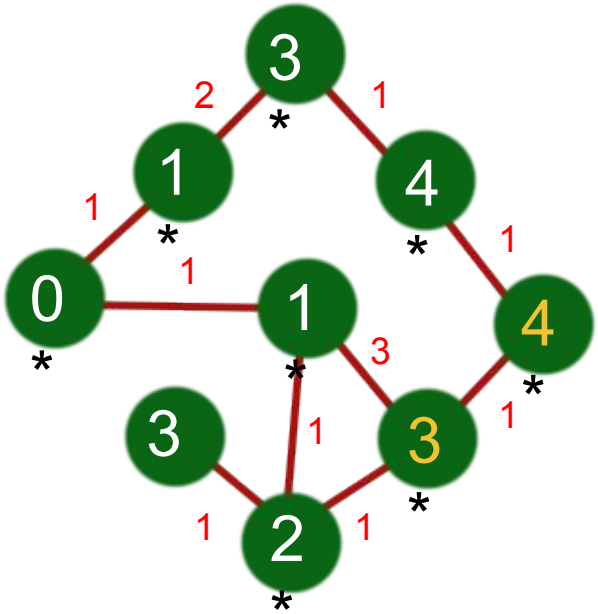
The shortest path can then be found by visiting the connected nodes with lowest total score and distance.



Dijkstra’s algorithm

If the current node is the end node, complete the process and then finish.

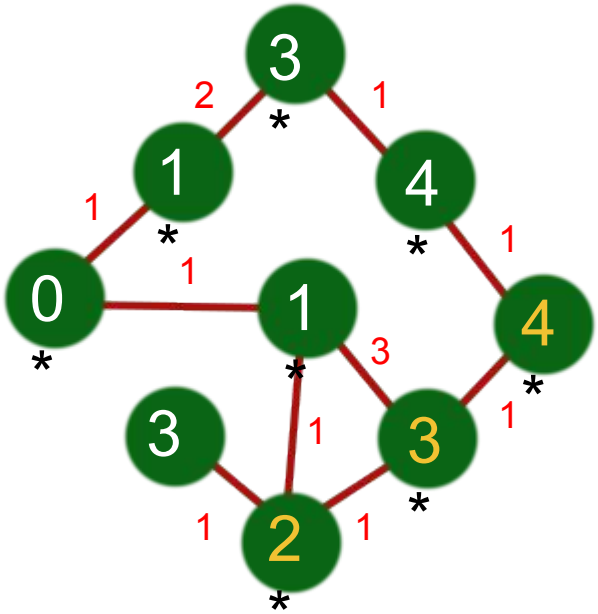
The shortest path can then be found by visiting the connected nodes with lowest total score and distance.



Dijkstra’s algorithm

If the current node is the end node, complete the process and then finish.

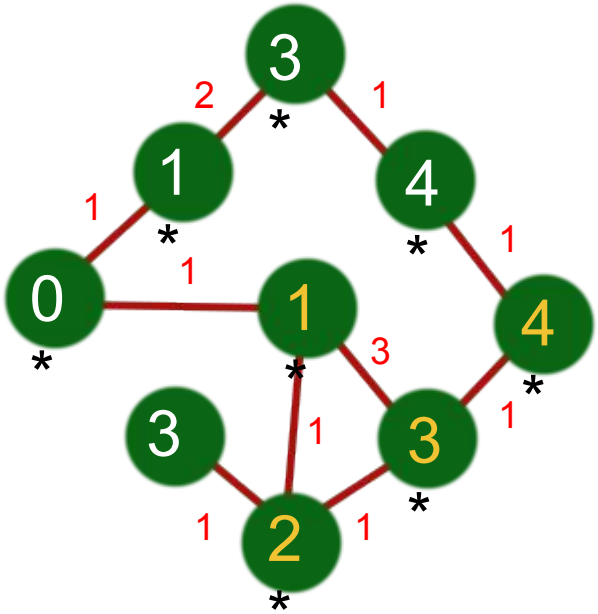
The shortest path can then be found by visiting the connected nodes with lowest total score and distance.



Dijkstra’s algorithm

If the current node is the end node, complete the process and then finish.

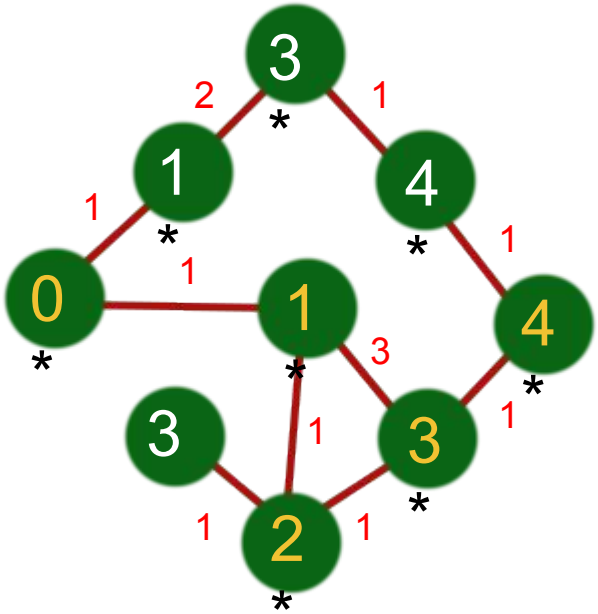
The shortest path can then be found by visiting the connected nodes with lowest total score and distance.



Dijkstra’s algorithm

If the current node is the end node, complete the process and then finish.

The shortest path can then be found by visiting the connected nodes with lowest total score and distance.

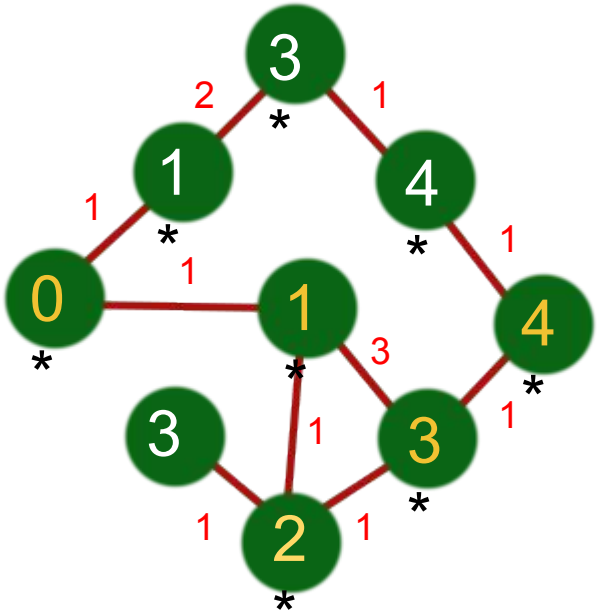


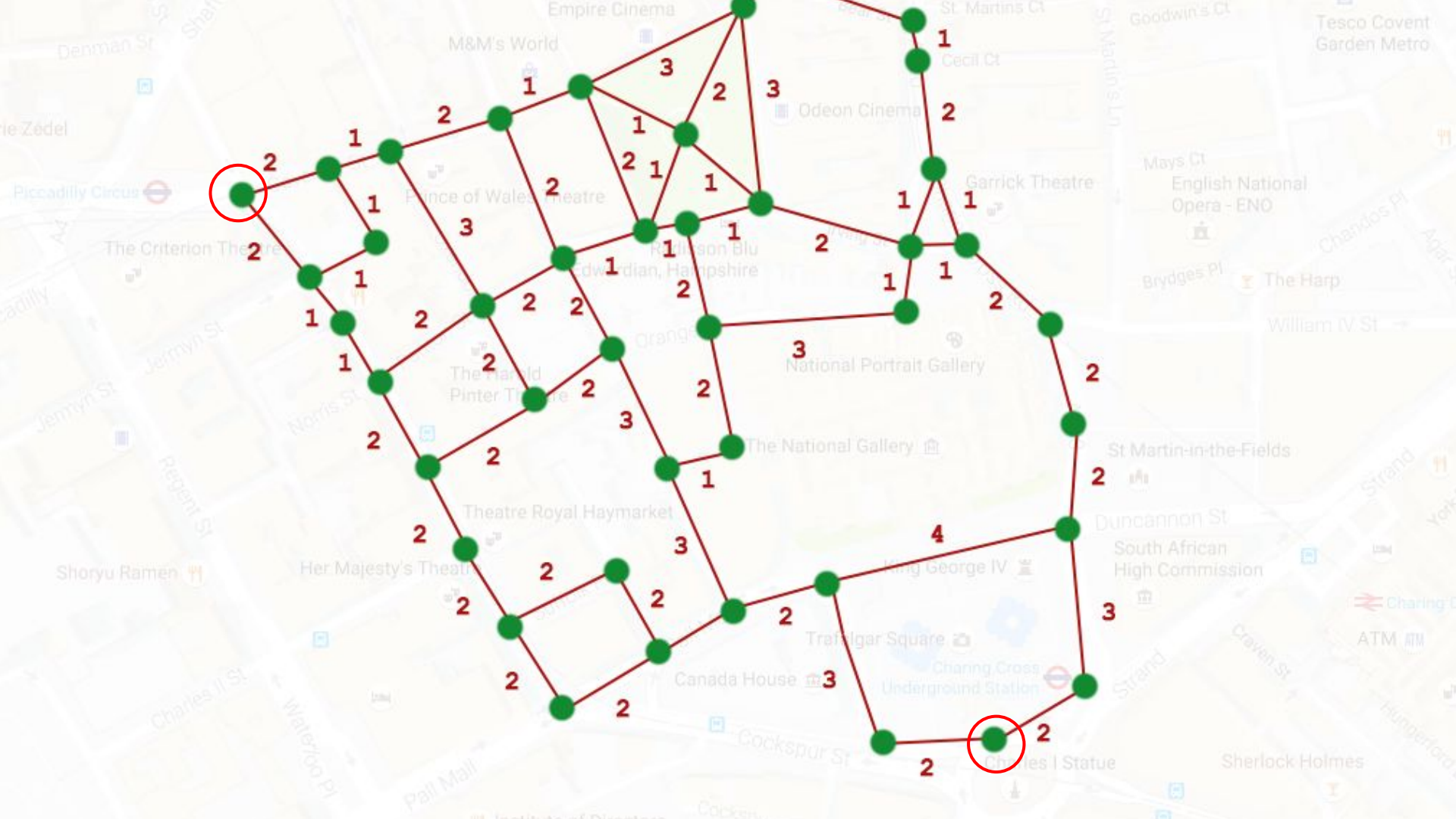
Dijkstra’s algorithm

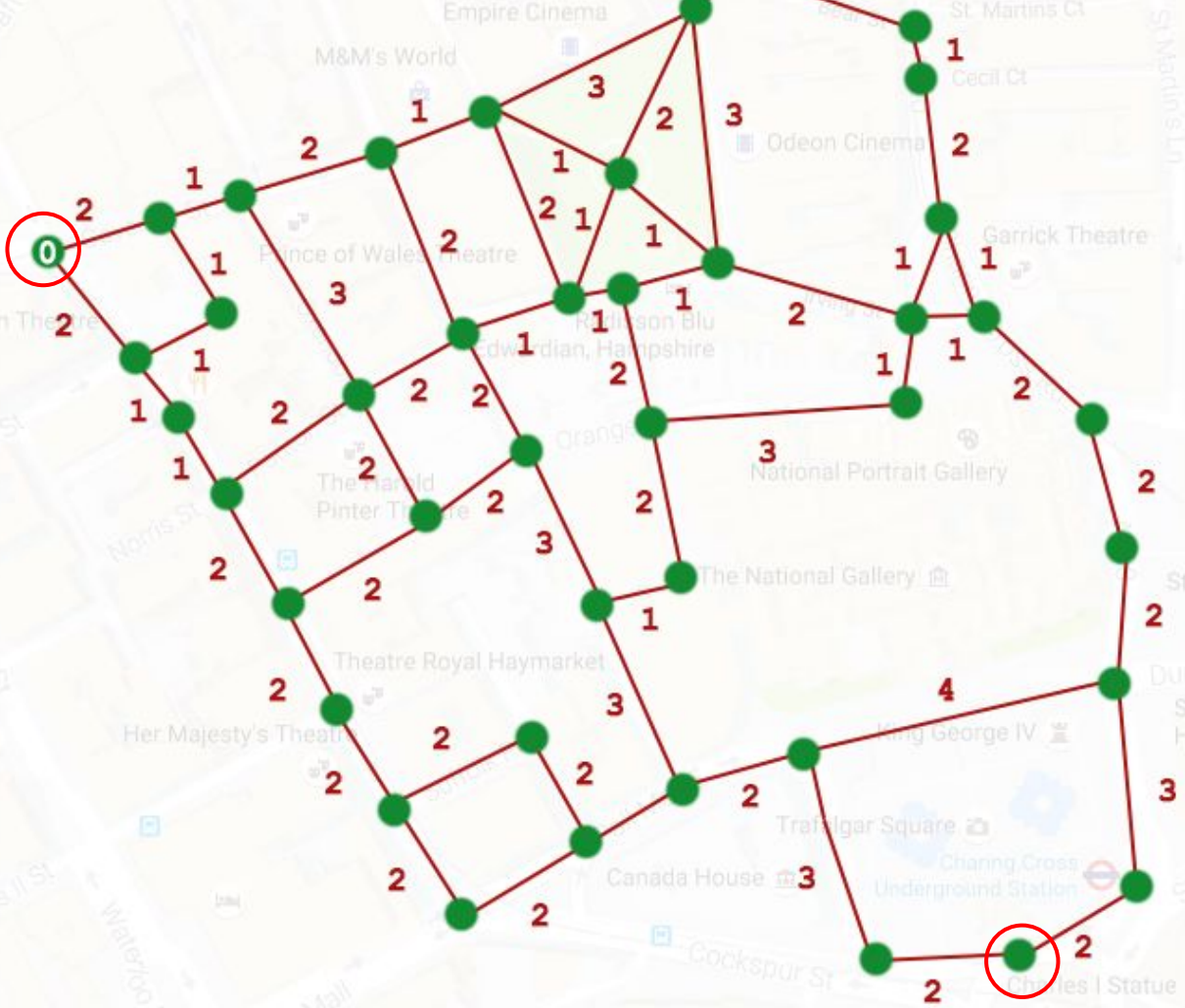
If the current node is the end node, complete the process and then finish.

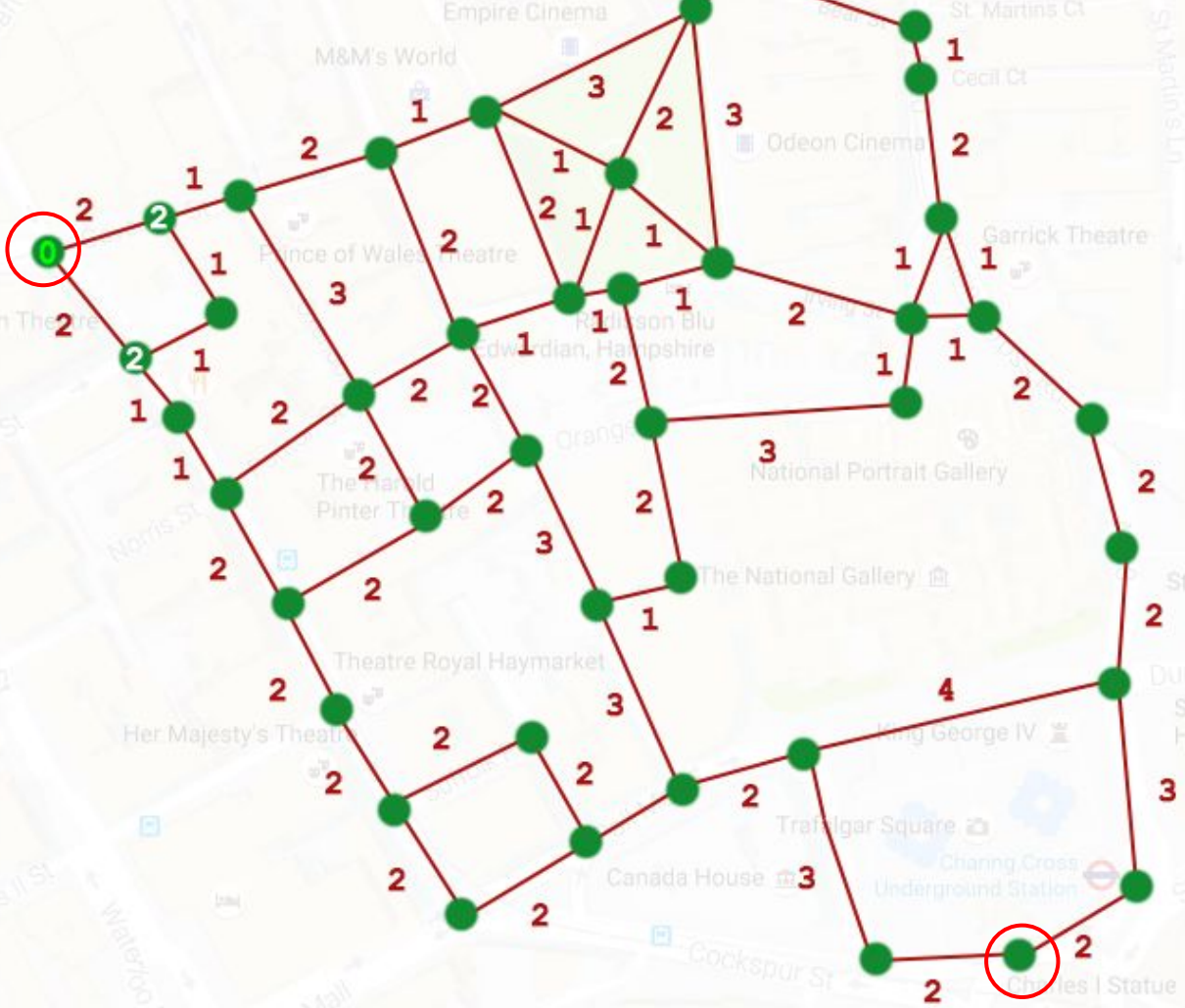
The shortest path can then be found by visiting the connected nodes with lowest total score and distance.

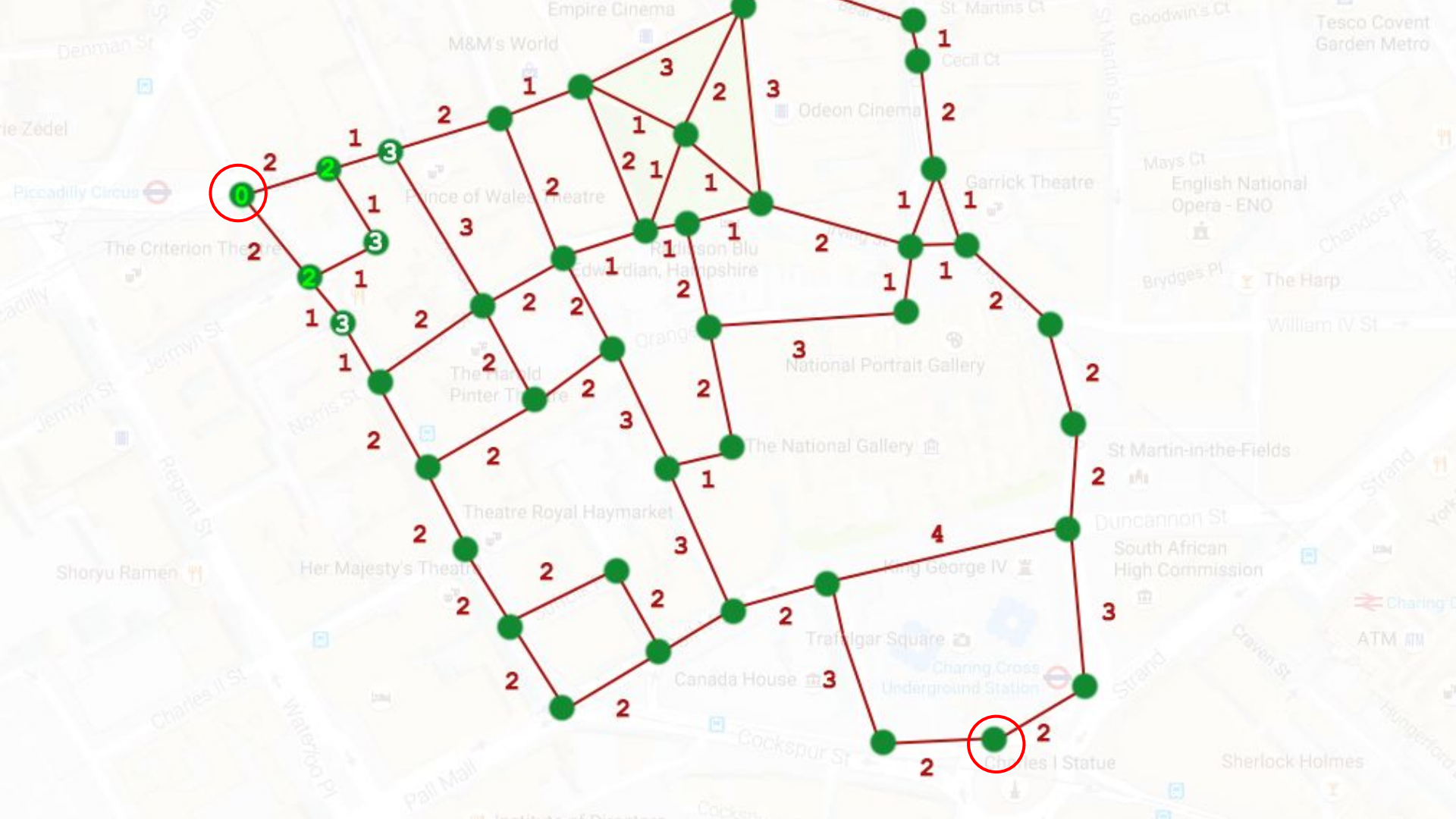
If no nodes with a score are left, and the end has not been found, then there is no path.

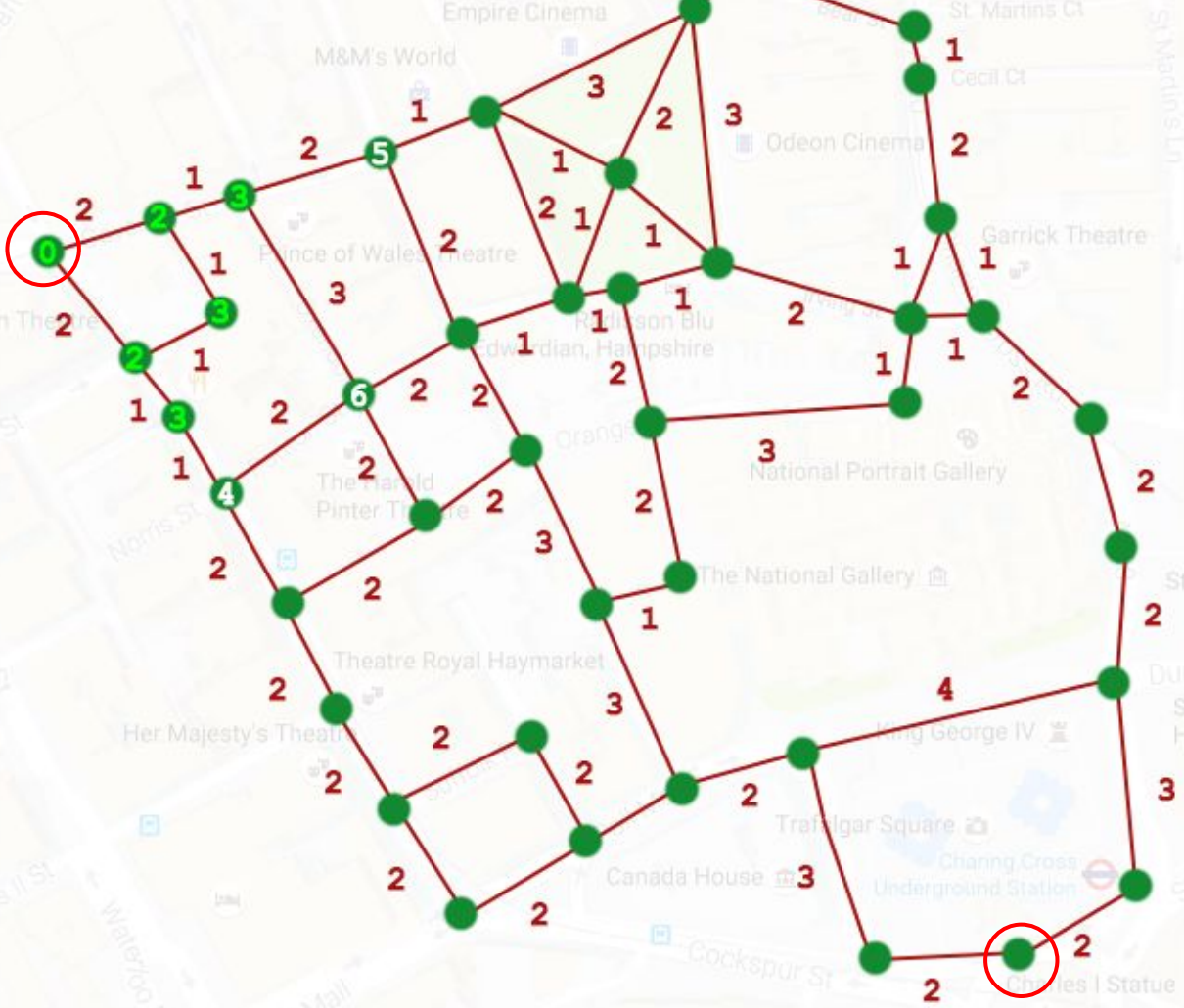


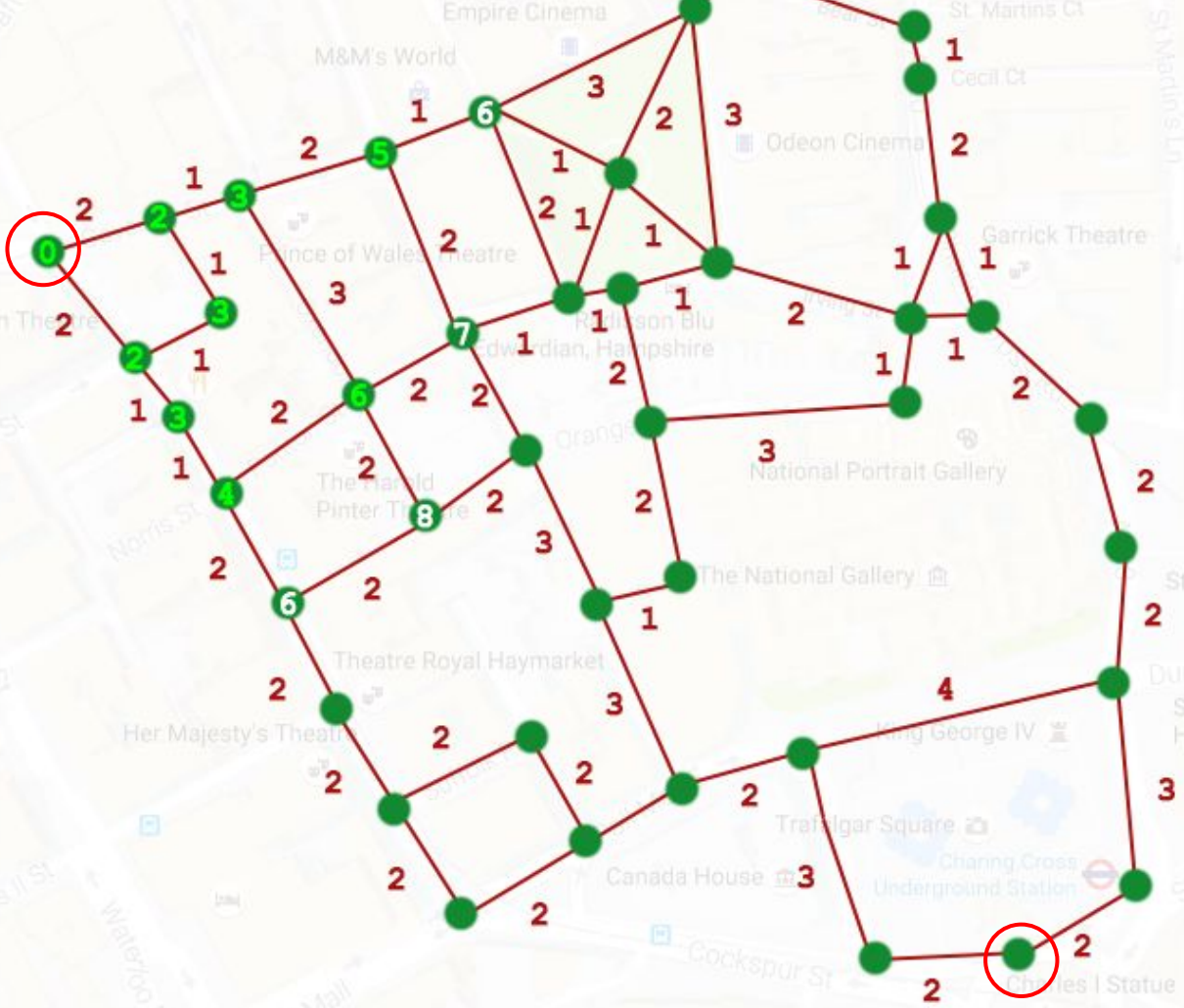

Empire Cinema
M&M's World
Odeon Cinema
Garrick Theatre
National Portrait Gallery
The National Gallery
St Martin-in-the-Fields
Theatre Royal Haymarket
Her Majesty's Theatre
Canada House
Trafalgar Square
South African High Commission
Charles I Statue
Sherlock Holmes
The Criterion Theatre
Shoryu Ramen
Tesco Covent Garden Metro
English National Opera - ENO
The Harp
ATM

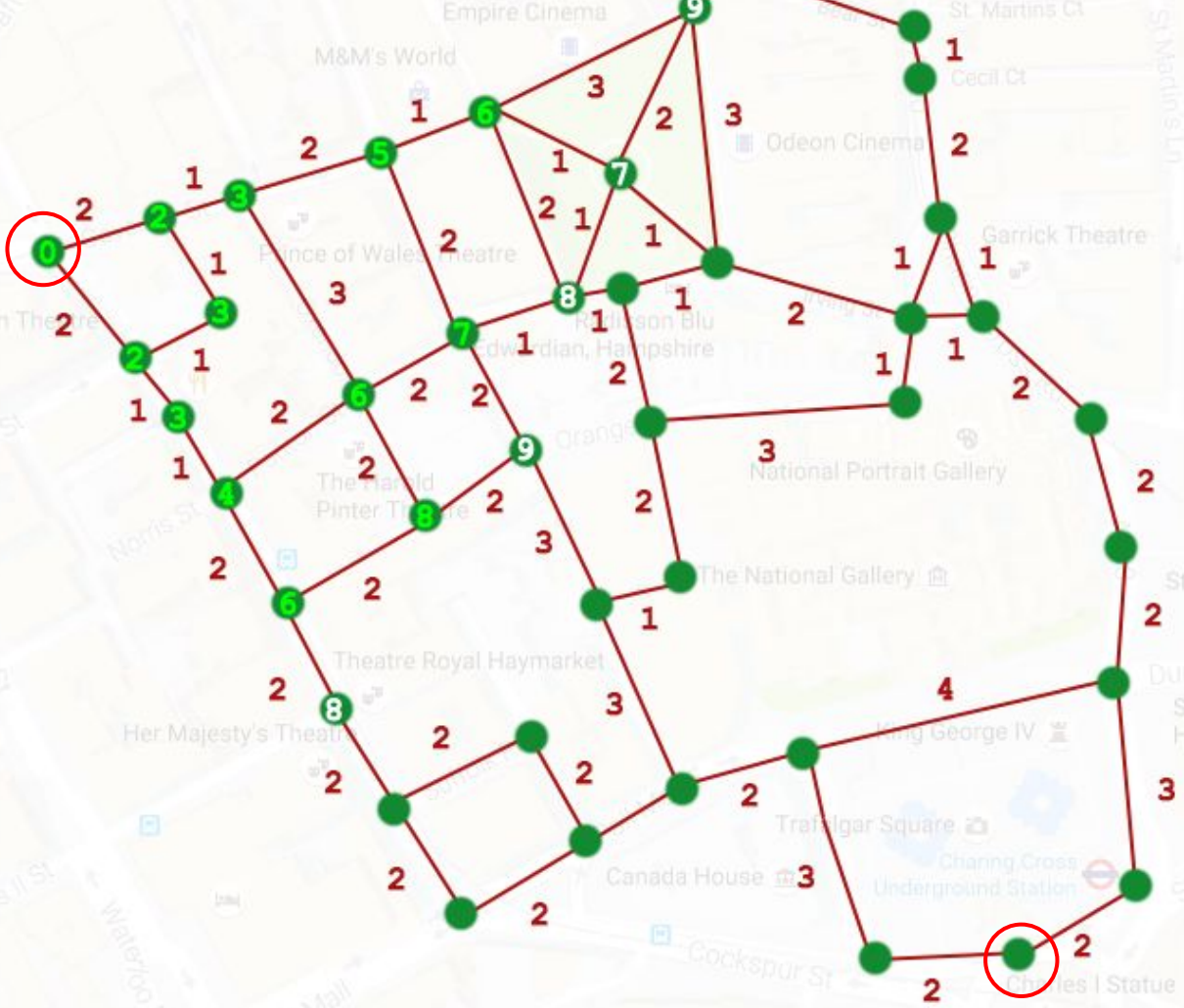

0

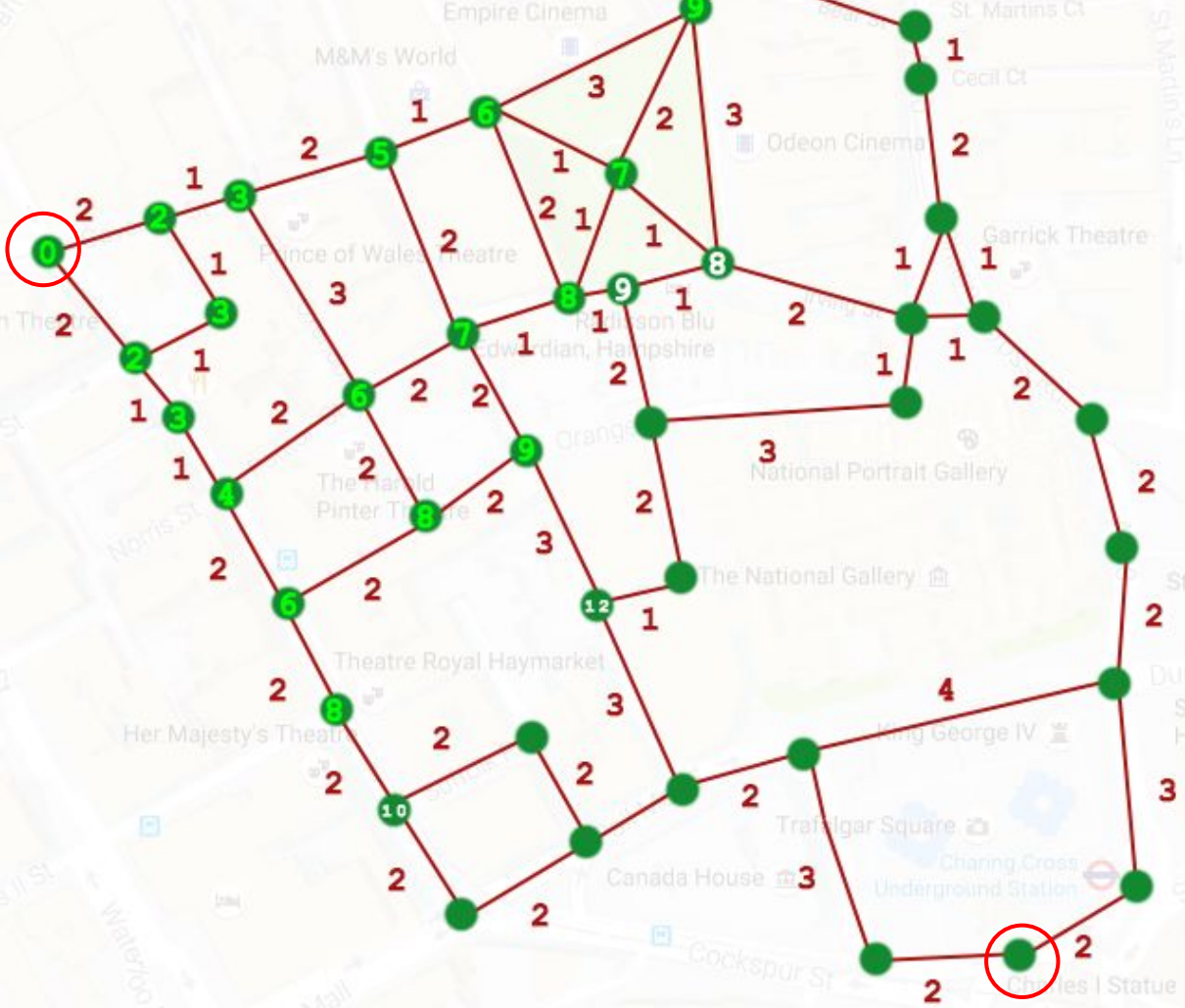

0
Empire Cinema
M&M's World
Odeon Cinema
Garrick Theatre
English National Opera - ENO
The Criterion Theatre
Prince of Wales Theatre
National Portrait Gallery
The National Gallery
St Martin-in-the-Fields
Theatre Royal Haymarket
Her Majesty's Theatre
South African High Commission
Trafalgar Square
Canada House
Charing Cross Underground Station
Tesco Covent Garden Metro
The Harp
Shoryu Ramen
Sherlock Holmes
ATM

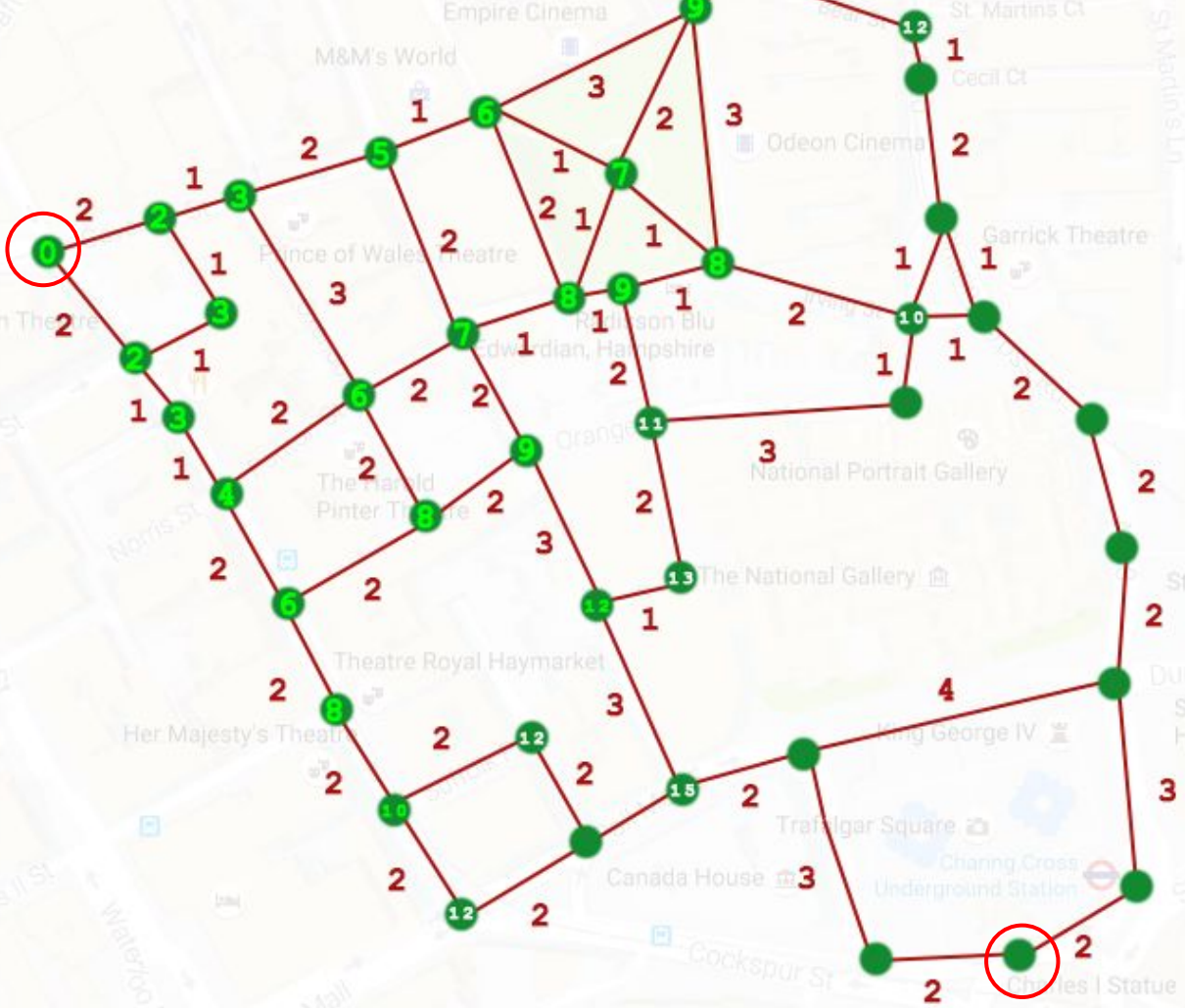

Empire Cinema
M&M's World
Odeon Cinema
Garrick Theatre
English National Opera - ENO
The Harp
The Criterion Theatre
National Portrait Gallery
The National Gallery
St Martin-in-the-Fields
Theatre Royal Haymarket
Her Majesty's Theatre
South African High Commission
Trafalgar Square
Canada House
Charing Cross Underground Station
Charles I Statue
Sherlock Holmes
Tesco Covent Garden Metro
Shoryu Ramen
ATM

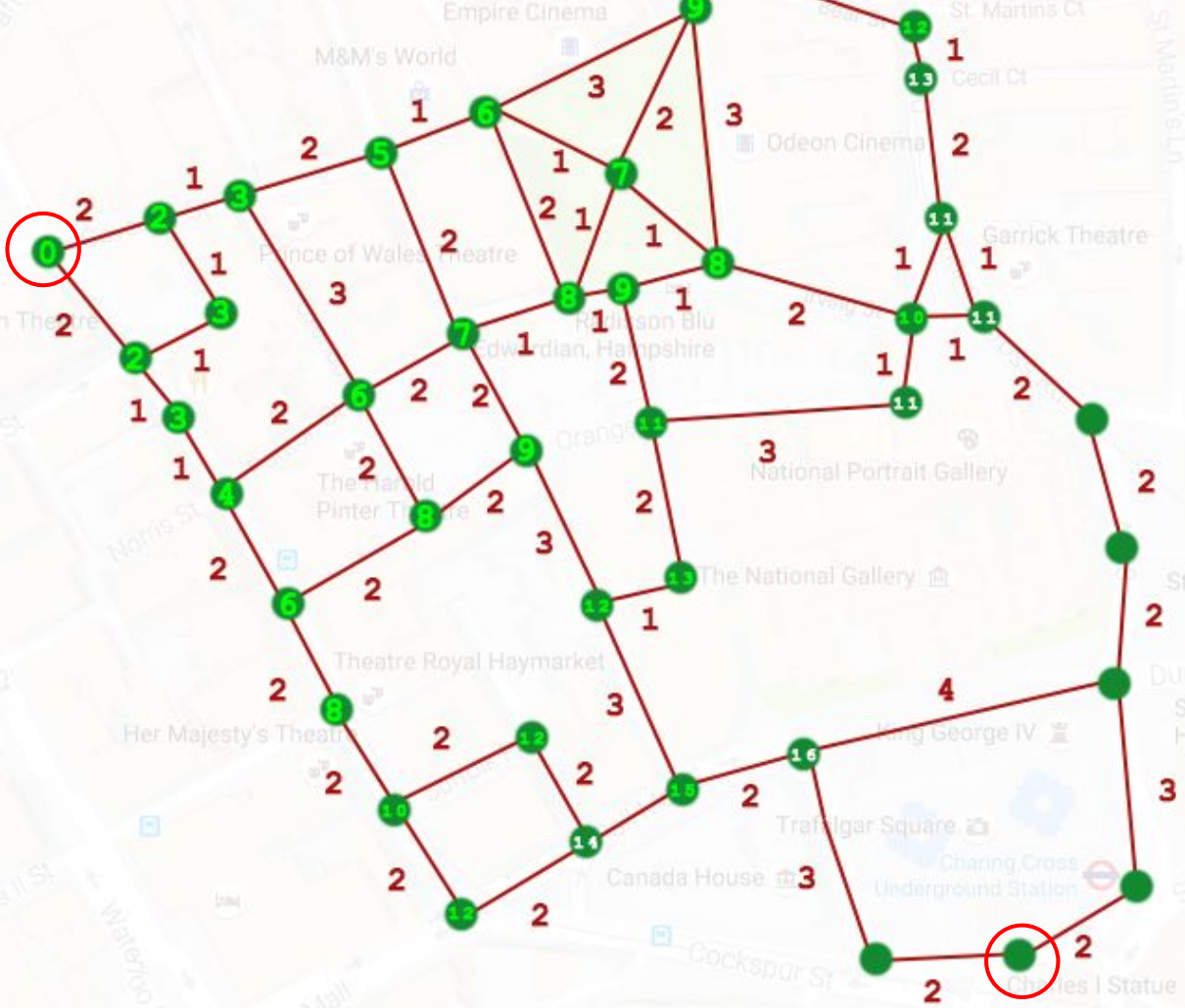

0
1
2
3
4
5
6
Empire Cinema
M&M's World
Odeon Cinema
Garrick Theatre
English National Opera - ENO
The Criterion Theatre
Prince of Wales Theatre
The Harp
National Portrait Gallery
The National Gallery
St Martin-in-the-Fields
Theatre Royal Haymarket
Her Majesty's Theatre
South African High Commission
Trafalgar Square
Canada House
Charing Cross Underground Station
Shoryu Ramen
Tesco Covent Garden Metro
Sherlock Holmes

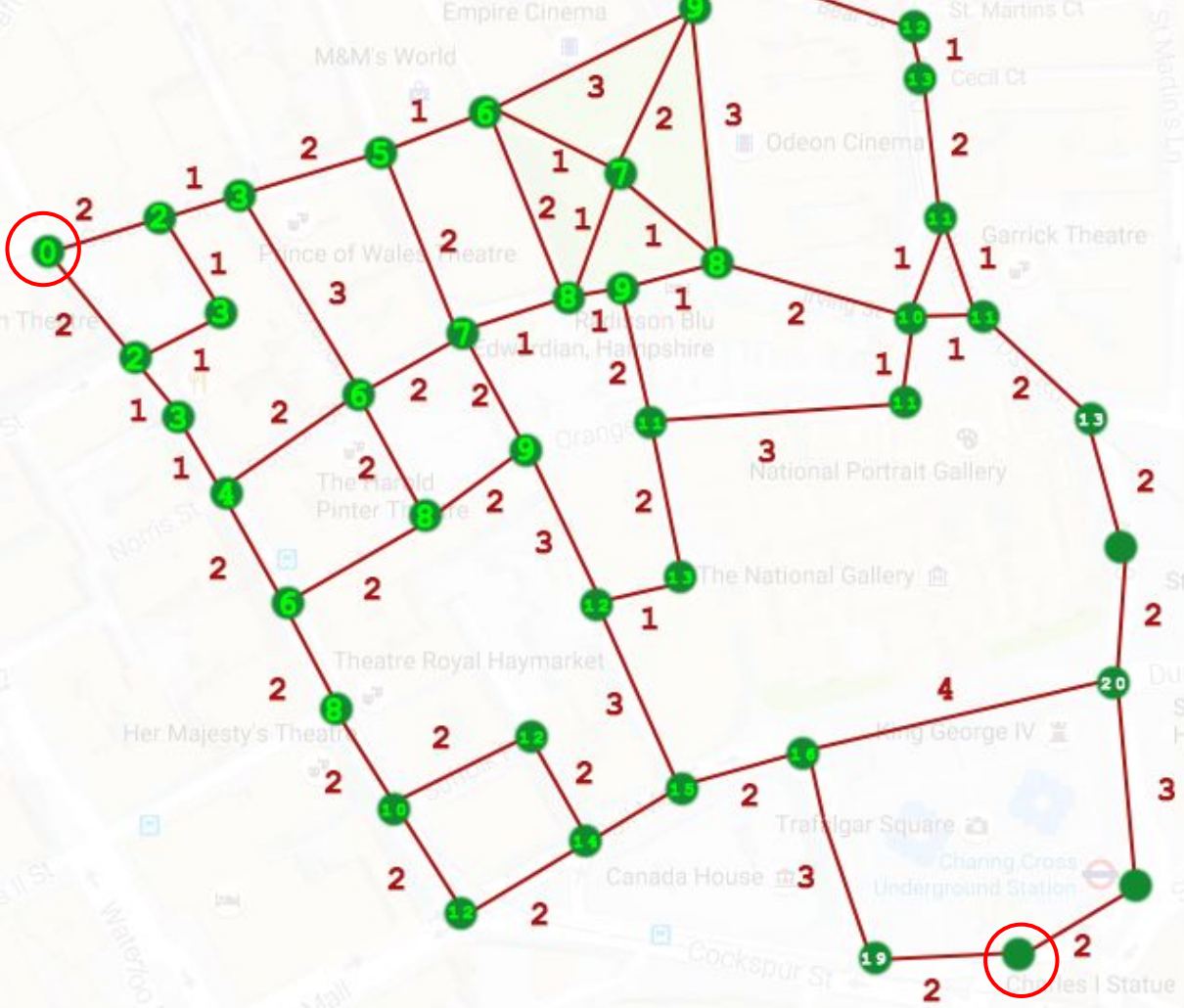

0
1
2
3
4
5
6
7
8

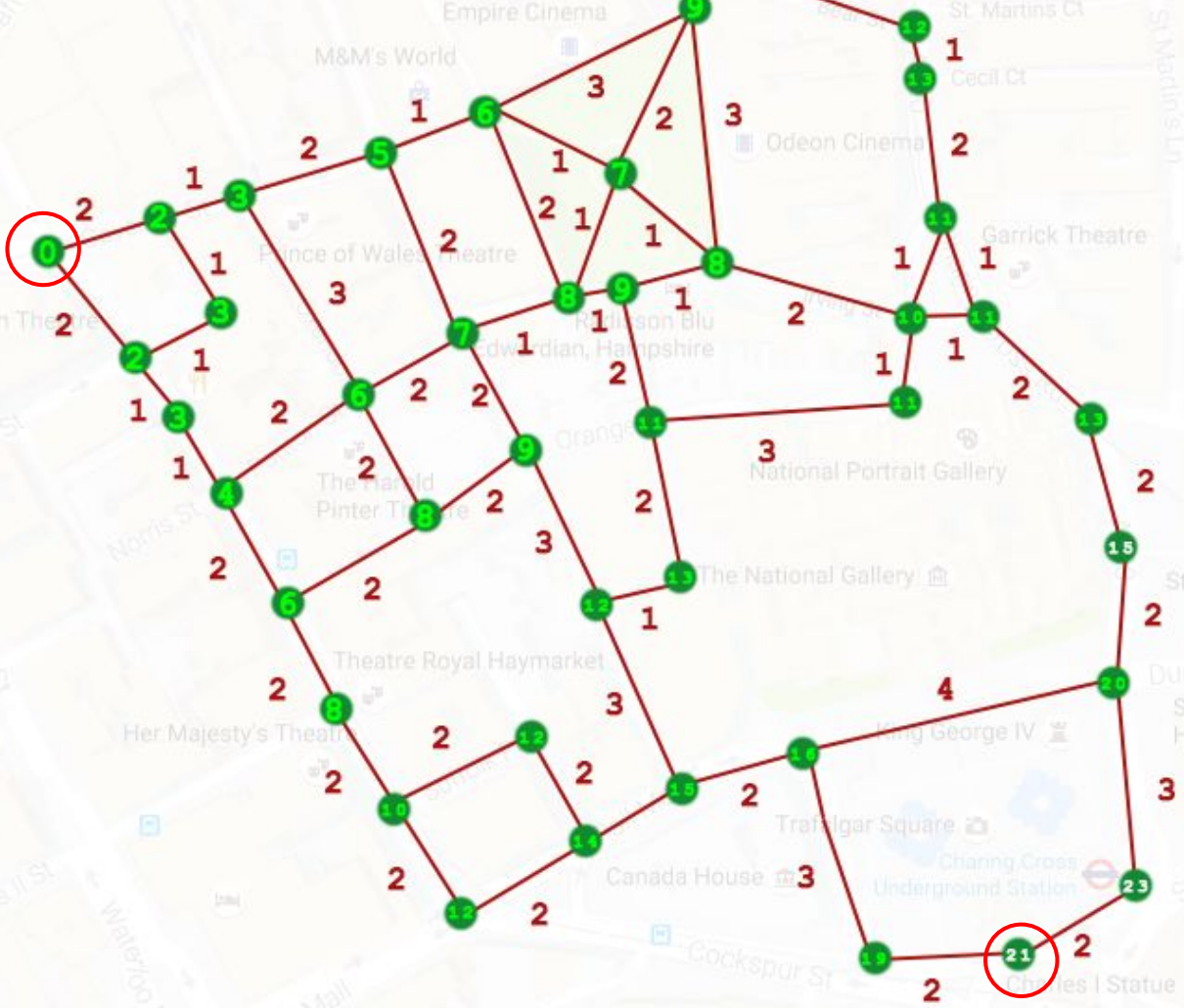

0
1
2
3
4
5
6
7
8
9

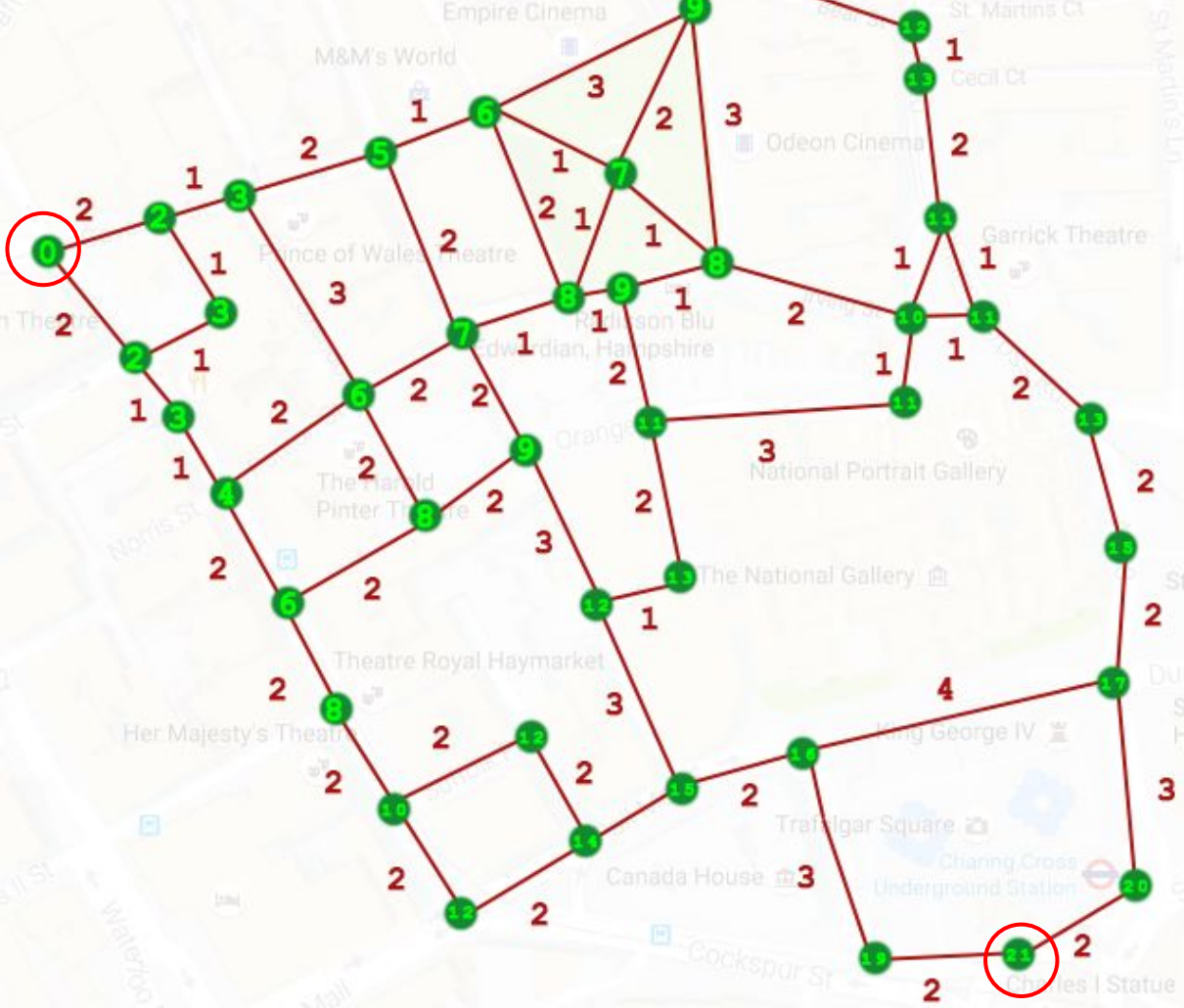

0
2
3
5
6
7
8
9
10
12
Empire Cinema
M&M's World
Odeon Cinema
Garrick Theatre
National Portrait Gallery
The National Gallery
St Martin-in-the-Fields
Theatre Royal Haymarket
Her Majesty's Theatre
Trafalgar Square
Canada House
South African High Commission
Tesco Covent Garden Metro
English National Opera - ENO
The Harp
Shoryu Ramen
Sherlock Holmes

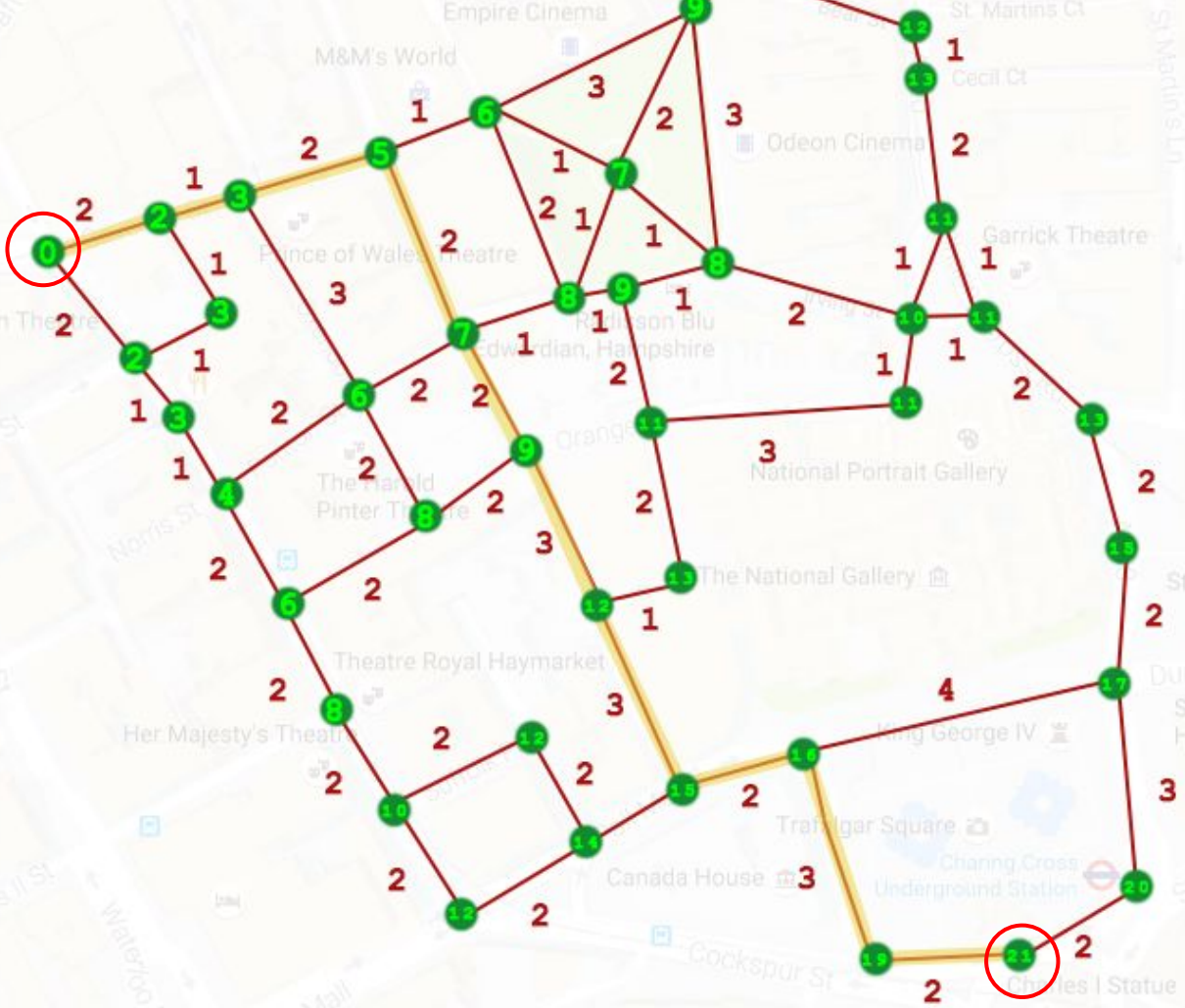

0
2
3
4
5
6
7
8
9
10
11
12
13
15


0
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16


0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
19
20


0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
18
19
20
21
23


0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21


0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21

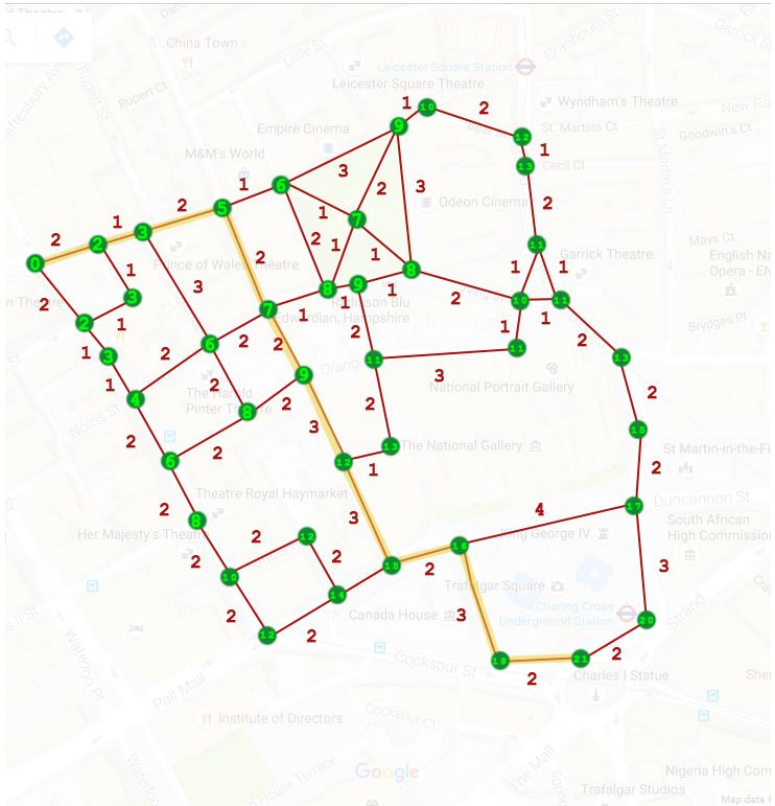
Weighted graphs

Useful for modelling physical systems and biased networks.

Weighted graphs

Useful for modelling physical systems and biased networks.

We could have used a beauty score to provide a scenic route.



Weighted graphs

Useful for modelling physical systems and biased networks.

We could have used a beauty score to provide a scenic route.

Or perhaps use crime figures to avoid dangerous areas.

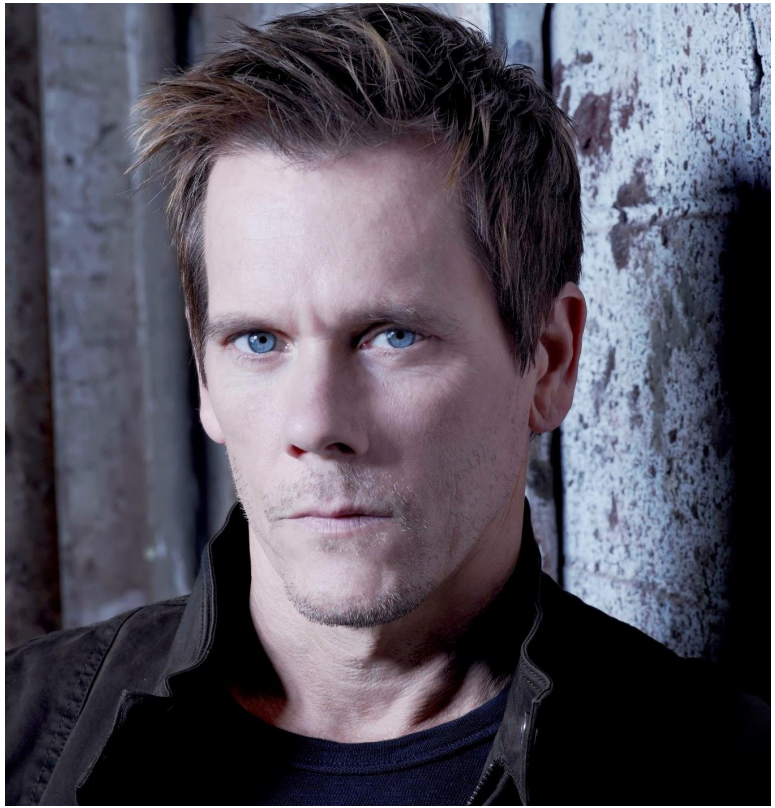
Weighted graphs

Useful for modelling physical systems and biased networks.

We could have used a beauty score to provide a scenic route.

Or perhaps use crime figures to avoid dangerous areas.

We could even use IMDB ratings to weight our Bacon Number search - going via the worst movies.

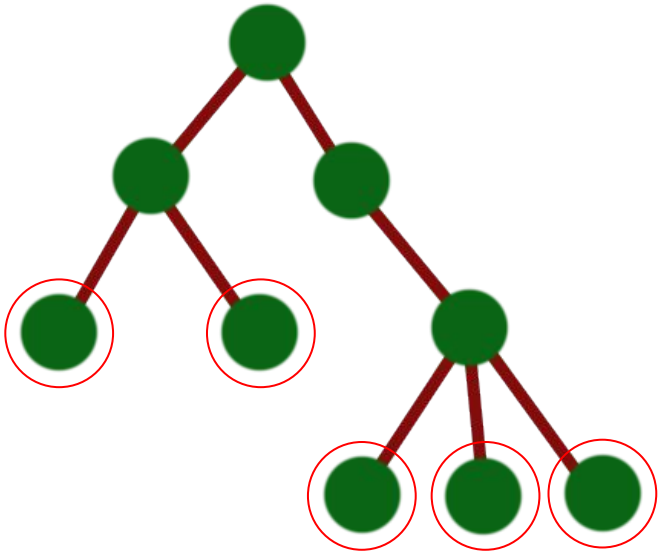


Trees

Trees

A tree is a special kind of graph; it still has nodes and edges.

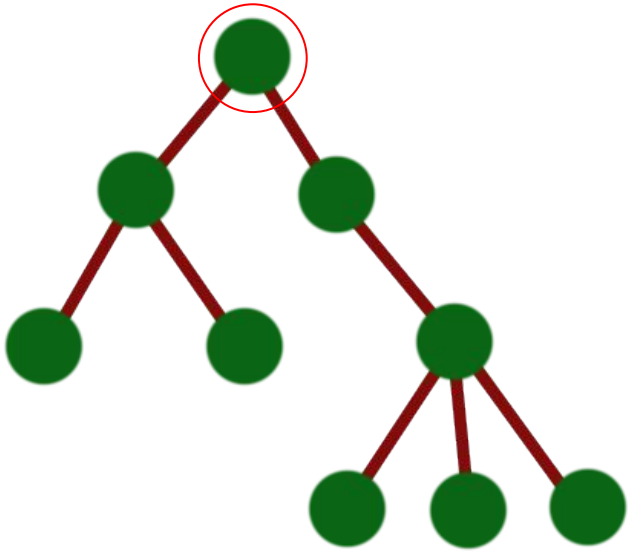
Where a node has a degree of one it is called a *leaf*.



Trees

A tree is a special kind of graph; it still has nodes and edges.

Where a node has a degree of one it is called a *leaf*. If it represents a hierarchy, one node is the *root*.

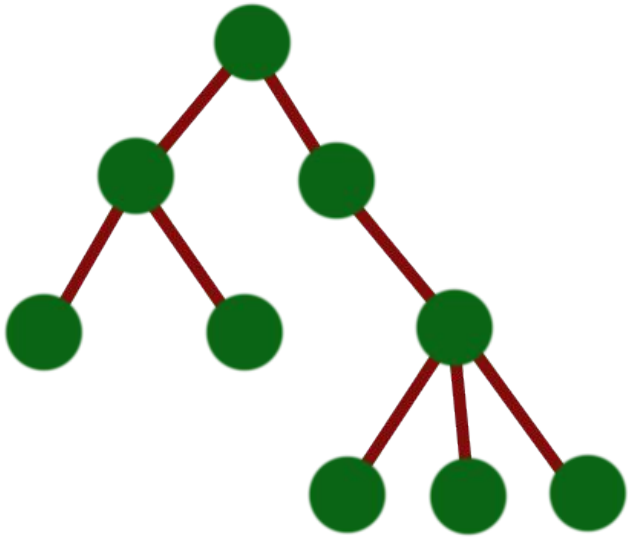


Trees

A tree is a special kind of graph; it still has nodes and edges.

Where a node has a degree of one it is called a *leaf*. If it represents a hierarchy, one node is the *root*.

Trees are constructed in such a way that there is only one path between any two nodes.



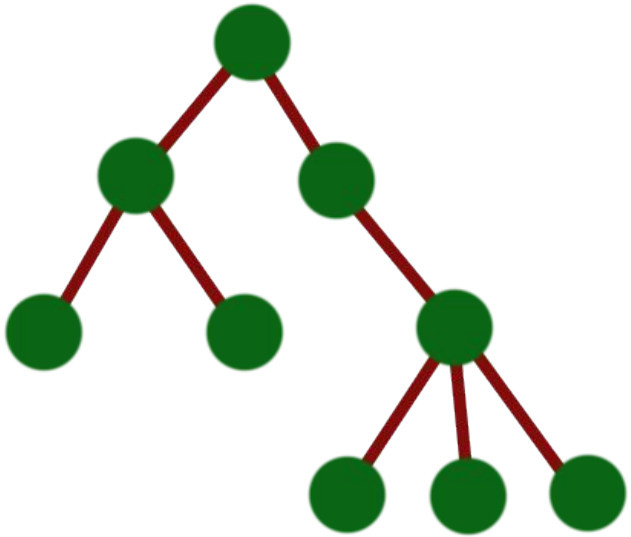
Trees

A tree is a special kind of graph; it still has nodes and edges.

Where a node has a degree of one it is called a *leaf*. If it represents a hierarchy, one node is the *root*.

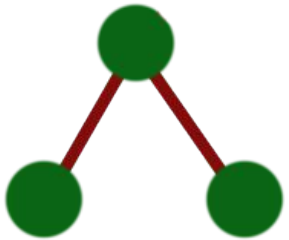
Trees are constructed in such a way that there is only one path between any two nodes.

It is an example of an Acyclic Graph - it has no cycles.



Trees

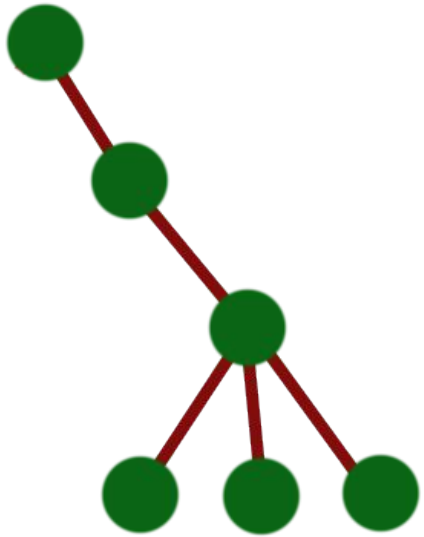
Much like real trees, they are *self-similar*



Trees

Much like real trees, they are *self-similar*

When you look at smaller pieces, they look a lot like the bigger piece.

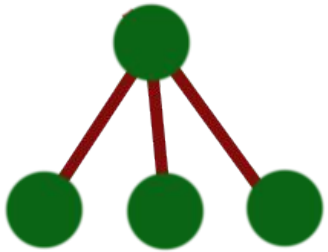


Trees

Much like real trees, they are *self-similar*

When you look at smaller pieces, they look a lot like the bigger piece.

A tree with depth greater than one is a *forest* of other trees.



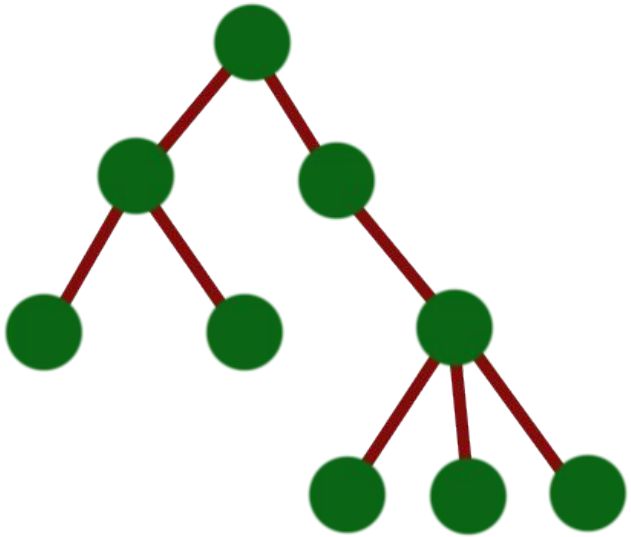
Trees

Much like real trees, they are *self-similar*

When you look at smaller pieces, they look a lot like the bigger piece.

A tree with depth greater than one is a *forest* of other trees.

Therefore any operation you perform on a small tree can be performed on any tree.

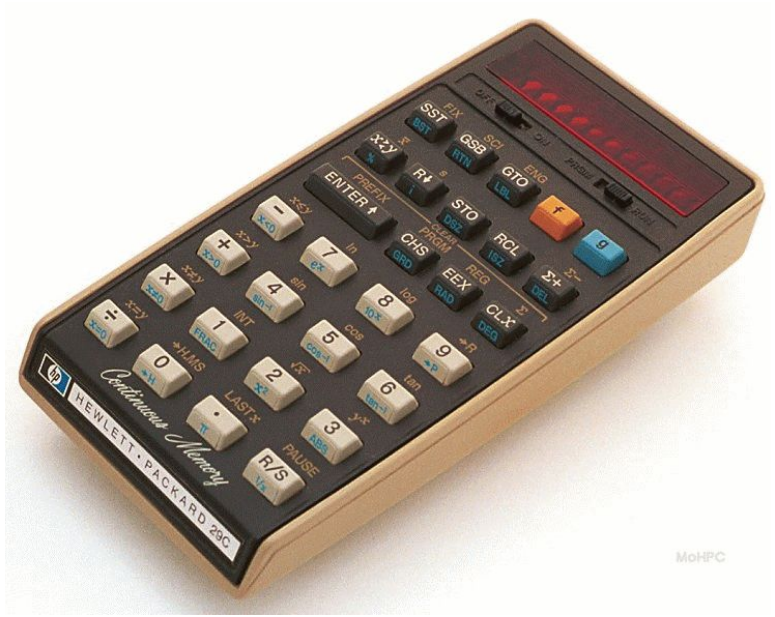


Walking trees: Arithmetic

HEWLETT · PACKARD
Continuous Memory
GSB
RTN
SCI
GTO
LBL
ENG
ENTER
PREFIX
R/S
PAUSE
LAST x
CLEAR PRGM
CHS
STO
RCL
EEX
REG
CLX
DEG
RAD
INT
FRAC
→H.MS
cos
cos⁻¹
sin
sin⁻¹
tan
tan⁻¹
ln
log
ABS
f
g
RUN

Reverse Polish Notation

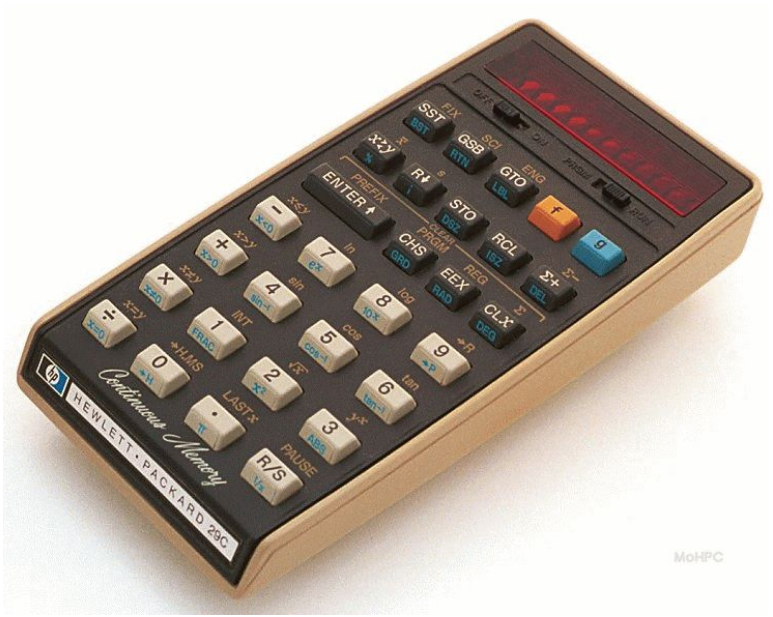
The first pocket calculators used RPN to input arithmetic.



Reverse Polish Notation

The first pocket calculators used RPN to input arithmetic.

Instead of typing “one plus two equals” you’d type “one two plus equals.”

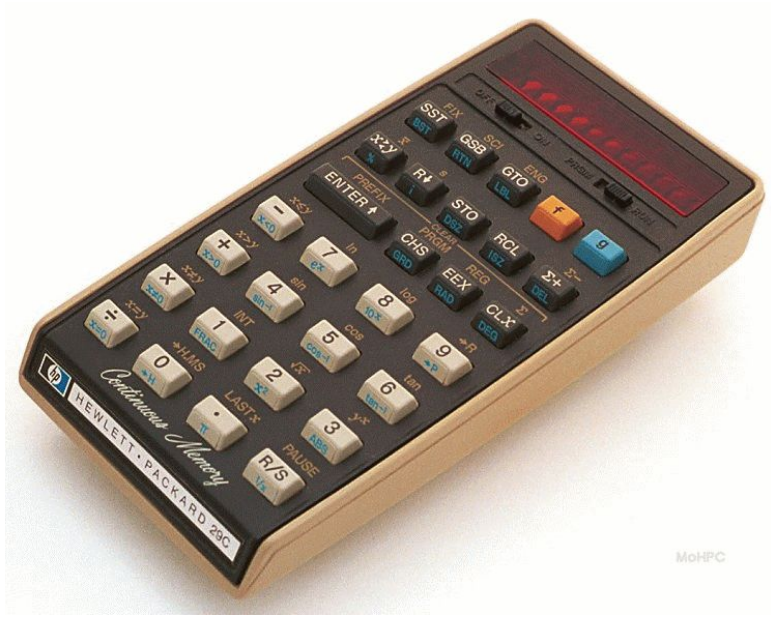


Reverse Polish Notation

The first pocket calculators used RPN to input arithmetic.

Instead of typing “one plus two equals” you’d type “one two plus equals.”

This is somewhat… counter-intuitive.



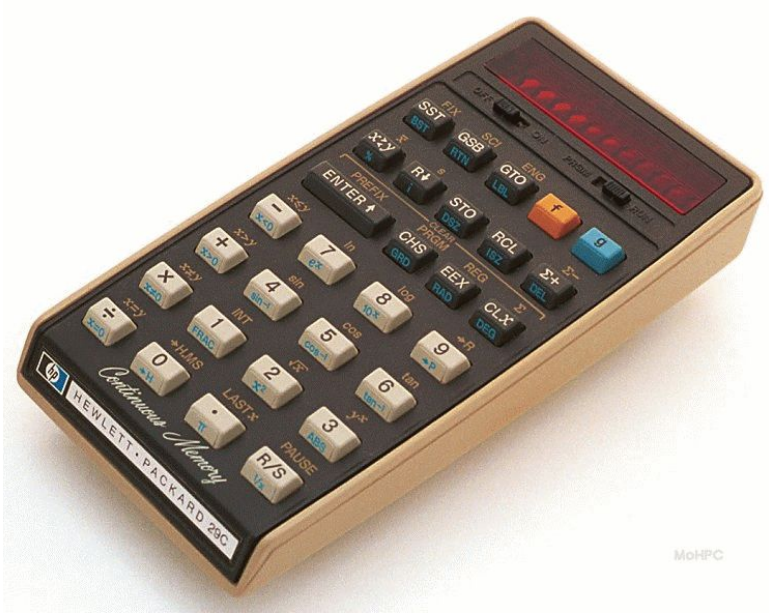
Reverse Polish Notation

The first pocket calculators used RPN to input arithmetic.

Instead of typing “one plus two equals” you’d type “one two plus equals.”

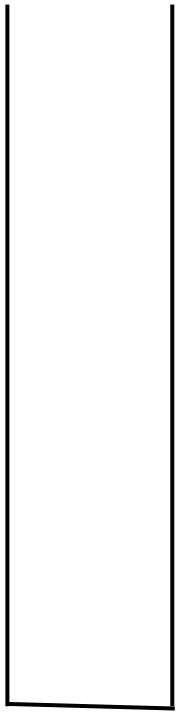
This is somewhat… counter-intuitive.

The reason for this is rooted in how computers process arithmetic.



Reverse Polish Notation

Essentially, stack arithmetic uses a queue and a stack.

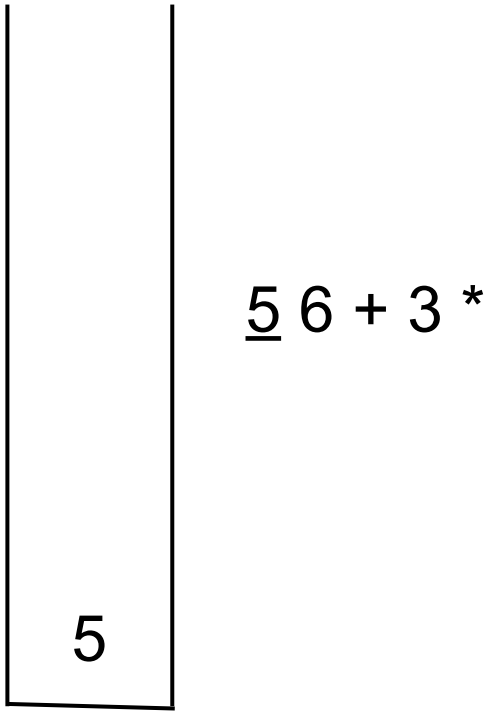


5 6 + 3 *

Reverse Polish Notation

Essentially, stack arithmetic uses a queue and a stack.

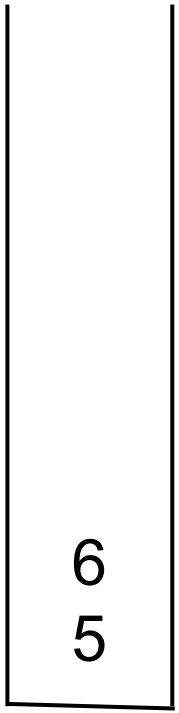
Read each item from the queue in turn.



Reverse Polish Notation

Essentially, stack arithmetic uses a queue and a stack.

Read each item from the queue in turn.



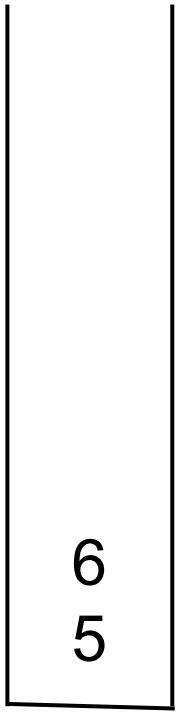
5 6 + 3 *

Reverse Polish Notation

Essentially, stack arithmetic uses a queue and a stack.

Read each item from the queue in turn.

When you reach an operator, pop the last two items from the stack, perform the operation and return the result to the stack.



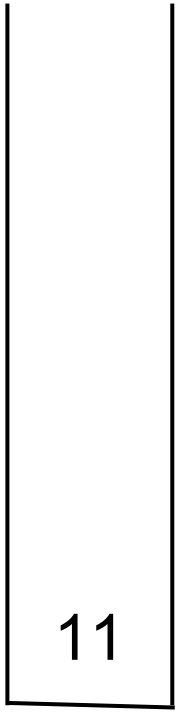
5 6 + 3 *

Reverse Polish Notation

Essentially, stack arithmetic uses a queue and a stack.

Read each item from the queue in turn.

When you reach an operator, pop the last two items from the stack, perform the operation and return the result to the stack.



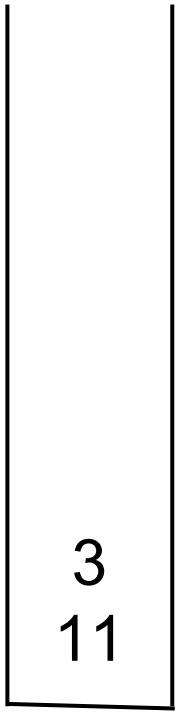
5 6 + 3 *

Reverse Polish Notation

Essentially, stack arithmetic uses a queue and a stack.

Read each item from the queue in turn.

When you reach an operator, pop the last two items from the stack, perform the operation and return the result to the stack.



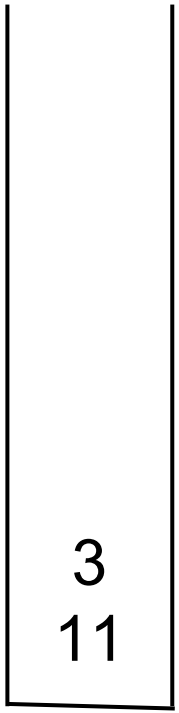
5 6 + 3 *

Reverse Polish Notation

Essentially, stack arithmetic uses a queue and a stack.

Read each item from the queue in turn.

When you reach an operator, pop the last two items from the stack, perform the operation and return the result to the stack.



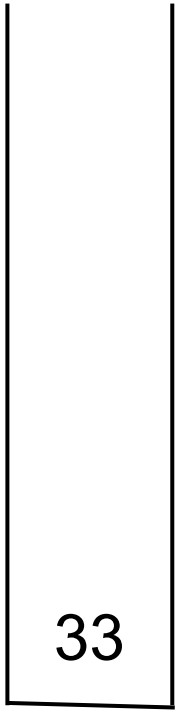
5 6 + 3 *

Reverse Polish Notation

Essentially, stack arithmetic uses a queue and a stack.

Read each item from the queue in turn.

When you reach an operator, pop the last two items from the stack, perform the operation and return the result to the stack.



5 6 + 3 *

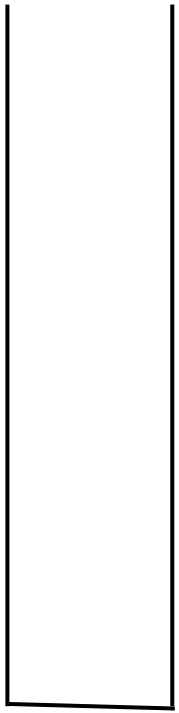
Reverse Polish Notation

Essentially, stack arithmetic uses a queue and a stack.

Read each item from the queue in turn.

When you reach an operator, pop the last two items from the stack, perform the operation and return the result to the stack.

When your queue is finished, you have your answer!



5 6 + 3 * = 33

Reverse Polish Notation

Another property of RPN is there is no need for operator precedence.

3 5 7 + * 23 / 7 +

Reverse Polish Notation

Another property of RPN is there is no need for operator precedence.

But: RPN is pretty illegible!

3 5 7 + * 23 / 7 + = ?!

Reverse Polish Notation

Another property of RPN is there is no need for operator precedence.

But: RPN is pretty illegible!

3 5 7 + * 23 / 7 + = ?!

How can we transpose it to “natural”?

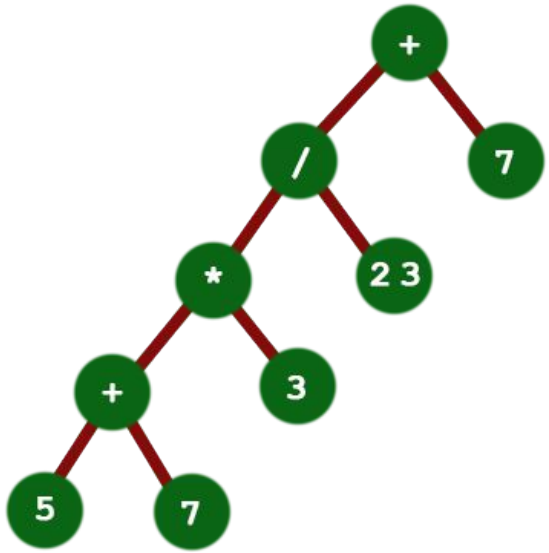
(Not that it’s much better in this example!)

(((5 + 7) * 3) / 23) + 7 = ?

Reverse Polish Notation

Answer: TREES!

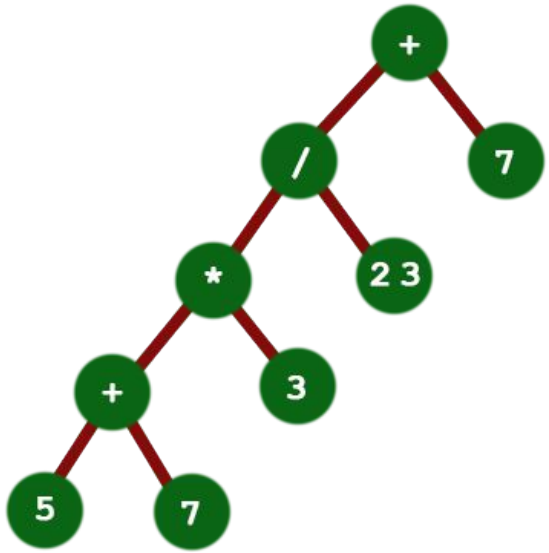
(Bet you didn’t see that coming!)



Reverse Polish Notation

Answer: TREES!

Operators become nodes; operands become leaves.

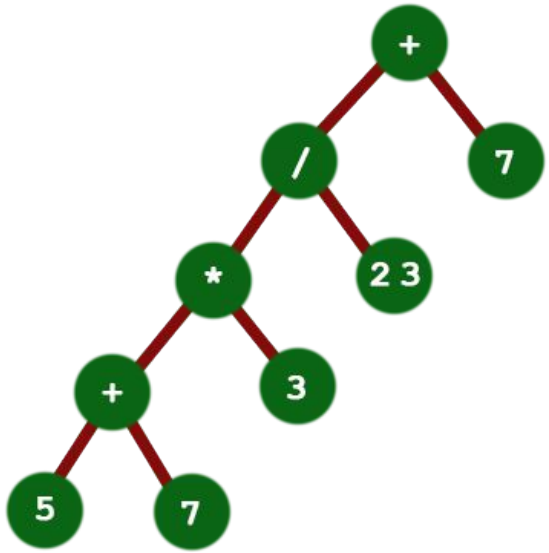


Reverse Polish Notation

Answer: TREES!

Operators become nodes; operands become leaves.

We can then walk the tree to produce our output. There are three methods.



Reverse Polish Notation

Answer: TREES!

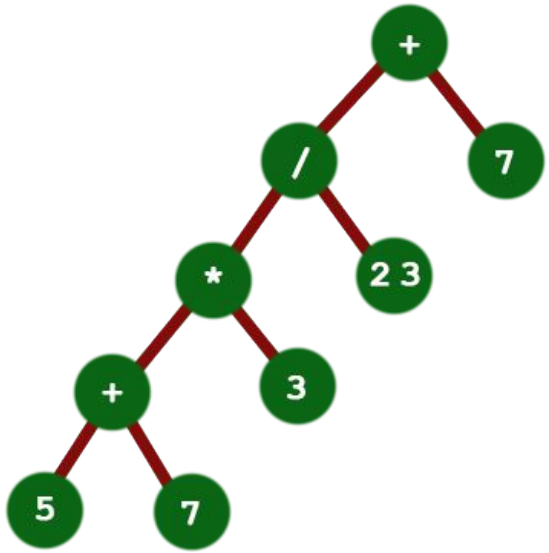
Operators become nodes; operands become leaves.

We can then walk the tree to produce our output. There are three methods.

Pre-order: Node, Left child, Right child

In-order: Left child, Node, Right child

Post-order: Left child, Right child, Node



Reverse Polish Notation

Answer: TREES!

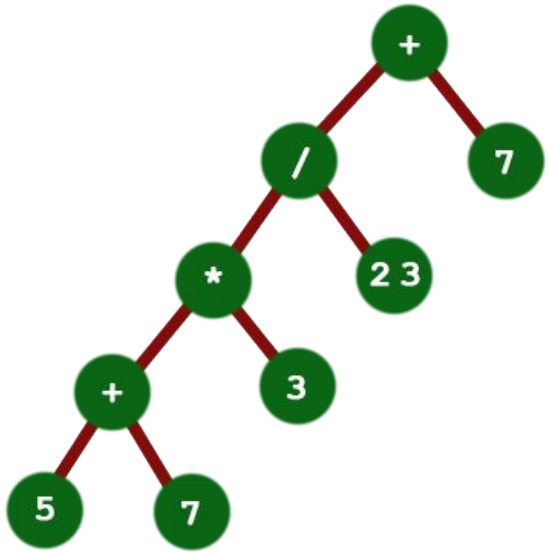
Operators become nodes; operands become leaves.

We can then walk the tree to produce our output. There are three methods.

Pre-order: Node, Left child, Right child

In-order: Left child, Node, Right child

Post-order: Left child, Right child, Node



+

Reverse Polish Notation

Answer: TREES!

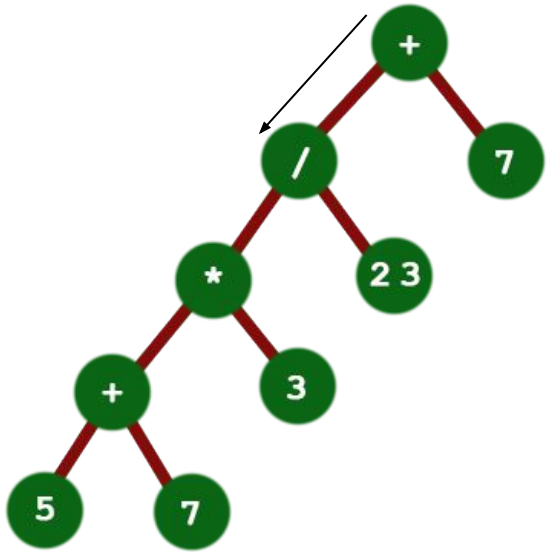
Operators become nodes; operands become leaves.

We can then walk the tree to produce our output. There are three methods.

Pre-order: Node, Left child, Right child

In-order: Left child, Node, Right child

Post-order: Left child, Right child, Node



+ /

Reverse Polish Notation

Answer: TREES!

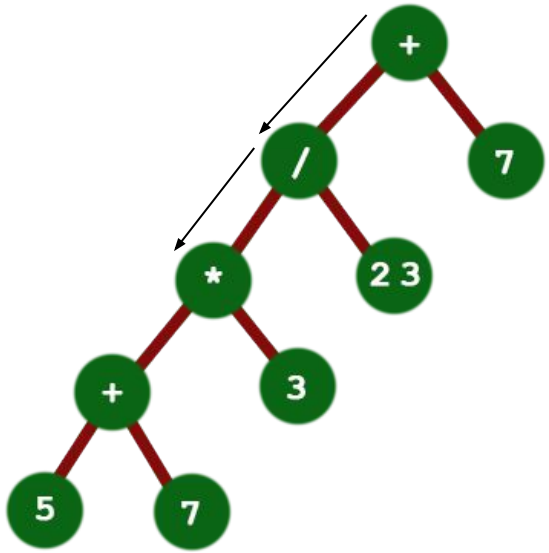
Operators become nodes; operands become leaves.

We can then walk the tree to produce our output. There are three methods.

Pre-order: Node, Left child, Right child

In-order: Left child, Node, Right child

Post-order: Left child, Right child, Node



+ / *

Reverse Polish Notation

Answer: TREES!

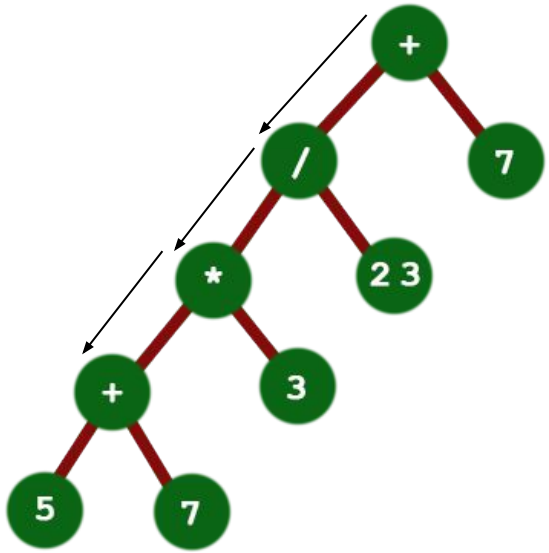
Operators become nodes; operands become leaves.

We can then walk the tree to produce our output. There are three methods.

Pre-order: Node, Left child, Right child

In-order: Left child, Node, Right child

Post-order: Left child, Right child, Node



+ / * +

Reverse Polish Notation

Answer: TREES!

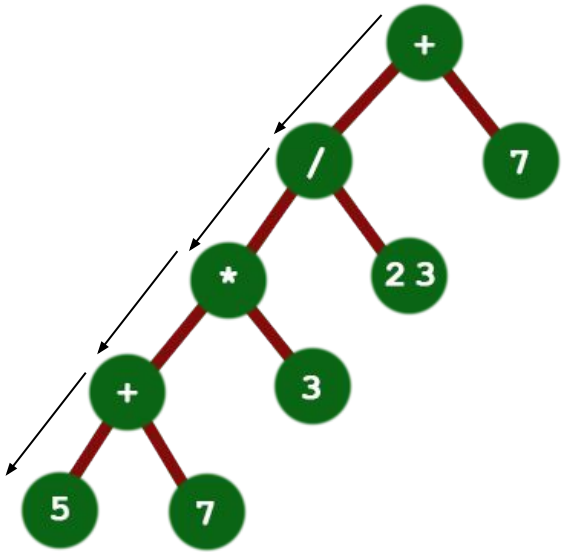
Operators become nodes; operands become leaves.

We can then walk the tree to produce our output. There are three methods.

Pre-order: Node, Left child, Right child

In-order: Left child, Node, Right child

Post-order: Left child, Right child, Node



+ / * + 5

Reverse Polish Notation

Answer: TREES!

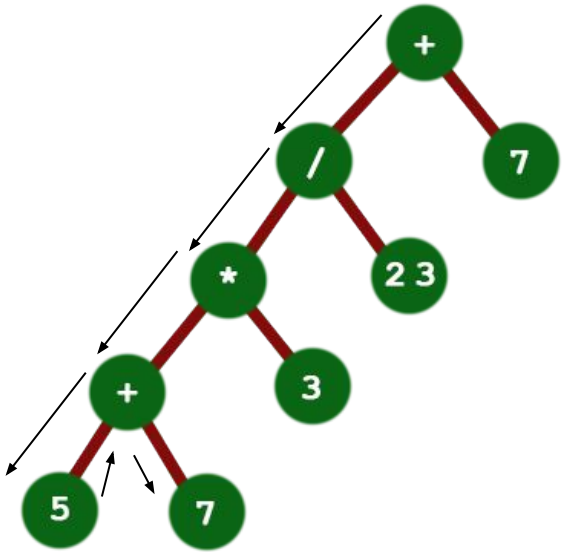
Operators become nodes; operands become leaves.

We can then walk the tree to produce our output. There are three methods.

Pre-order: Node, Left child, Right child

In-order: Left child, Node, Right child

Post-order: Left child, Right child, Node



+ / * + 5 7

Reverse Polish Notation

Answer: TREES!

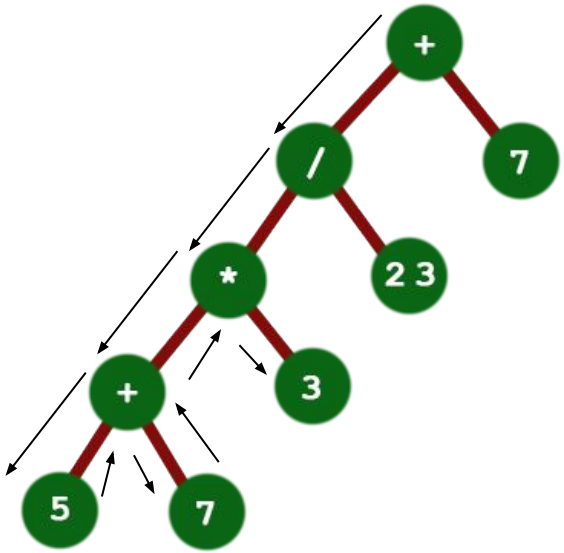
Operators become nodes; operands become leaves.

We can then walk the tree to produce our output. There are three methods.

Pre-order: Node, Left child, Right child

In-order: Left child, Node, Right child

Post-order: Left child, Right child, Node



+ / * + 5 7 3

Reverse Polish Notation

Answer: TREES!

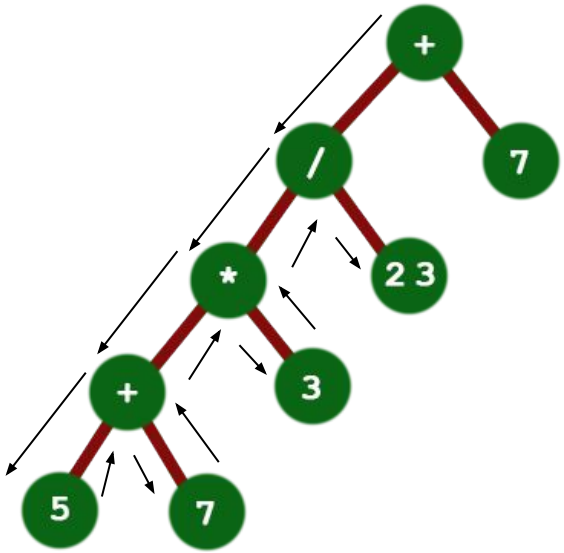
Operators become nodes; operands become leaves.

We can then walk the tree to produce our output. There are three methods.

Pre-order: Node, Left child, Right child

In-order: Left child, Node, Right child

Post-order: Left child, Right child, Node



+ / * + 5 7 3 23

Reverse Polish Notation

Answer: TREES!

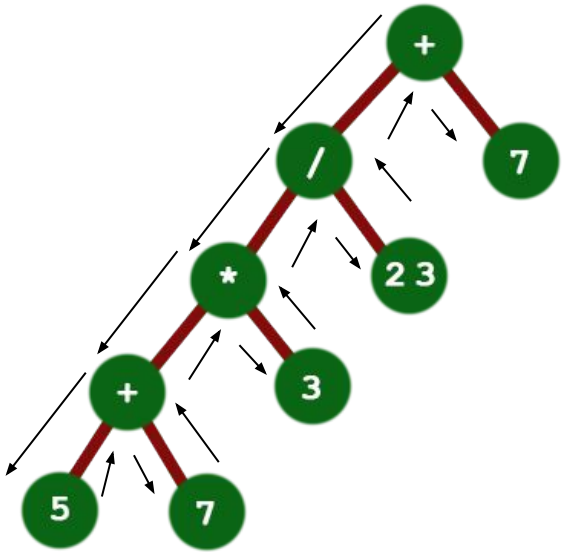
Operators become nodes; operands become leaves.

We can then walk the tree to produce our output. There are three methods.

Pre-order: Node, Left child, Right child

In-order: Left child, Node, Right child

Post-order: Left child, Right child, Node



+ / * + 5 7 3 23 7

Reverse Polish Notation

Answer: TREES!

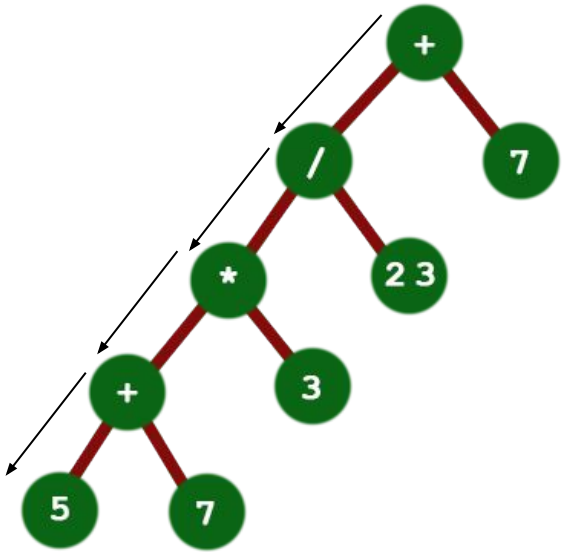
Operators become nodes; operands become leaves.

We can then walk the tree to produce our output. There are three methods.

Pre-order: Node, Left child, Right child

In-order: Left child, Node, Right child

Post-order: Left child, Right child, Node



Reverse Polish Notation

Answer: TREES!

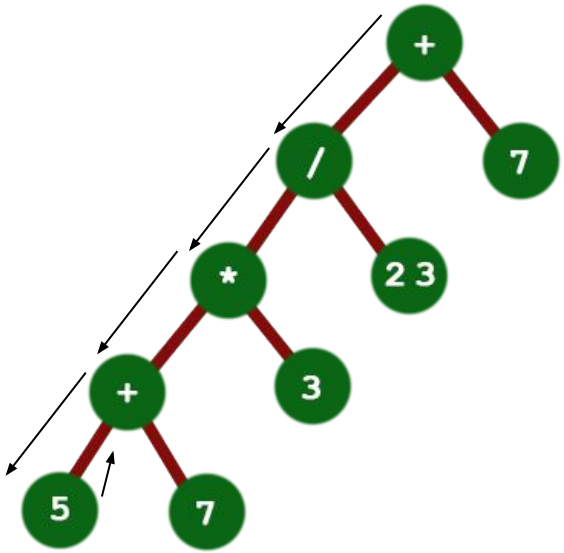
Operators become nodes; operands become leaves.

We can then walk the tree to produce our output. There are three methods.

Pre-order: Node, Left child, Right child

In-order: Left child, Node, Right child

Post-order: Left child, Right child, Node



5 +

Reverse Polish Notation

Answer: TREES!

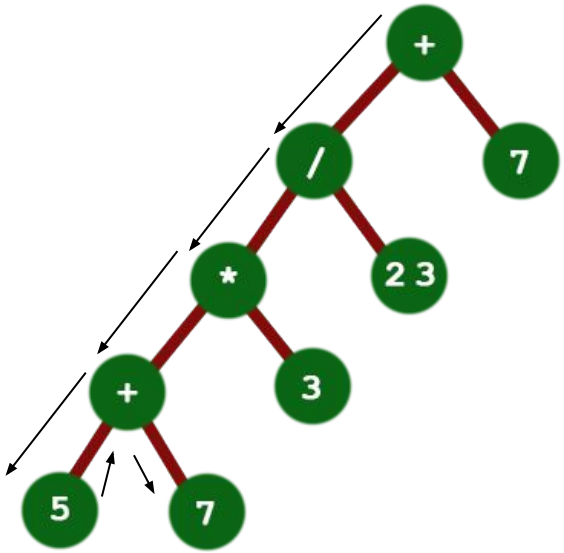
Operators become nodes; operands become leaves.

We can then walk the tree to produce our output. There are three methods.

Pre-order: Node, Left child, Right child

In-order: Left child, Node, Right child

Post-order: Left child, Right child, Node



5 + 7

Reverse Polish Notation

Answer: TREES!

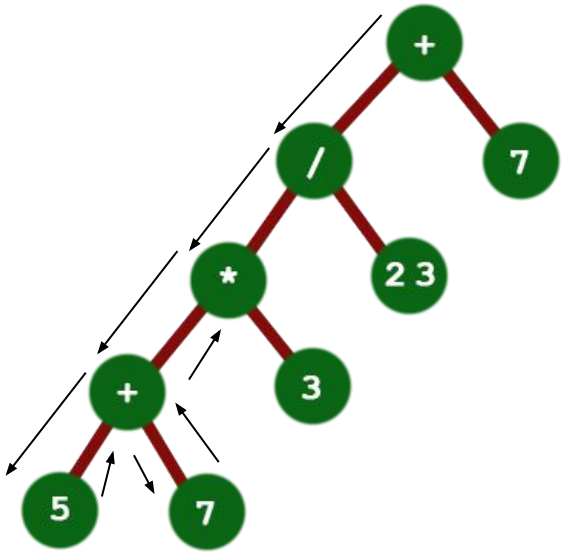
Operators become nodes; operands become leaves.

We can then walk the tree to produce our output. There are three methods.

Pre-order: Node, Left child, Right child

In-order: Left child, Node, Right child

Post-order: Left child, Right child, Node



5 + 7 *

Reverse Polish Notation

Answer: TREES!

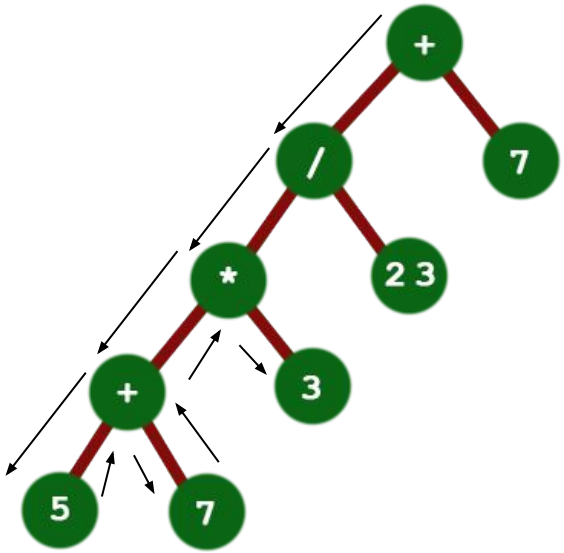
Operators become nodes; operands become leaves.

We can then walk the tree to produce our output. There are three methods.

Pre-order: Node, Left child, Right child

In-order: Left child, Node, Right child

Post-order: Left child, Right child, Node



5 + 7 * 3

Reverse Polish Notation

Answer: TREES!

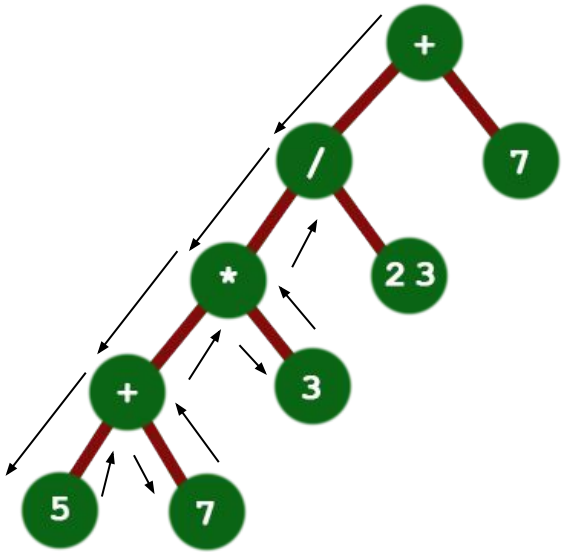
Operators become nodes; operands become leaves.

We can then walk the tree to produce our output. There are three methods.

Pre-order: Node, Left child, Right child

In-order: Left child, Node, Right child

Post-order: Left child, Right child, Node



5 + 7 * 3 /

Reverse Polish Notation

Answer: TREES!

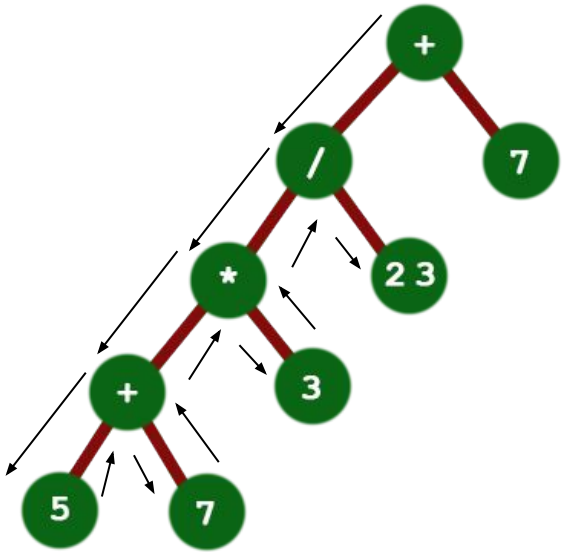
Operators become nodes; operands become leaves.

We can then walk the tree to produce our output. There are three methods.

Pre-order: Node, Left child, Right child

In-order: Left child, Node, Right child

Post-order: Left child, Right child, Node



5 + 7 * 3 / 23

Reverse Polish Notation

Answer: TREES!

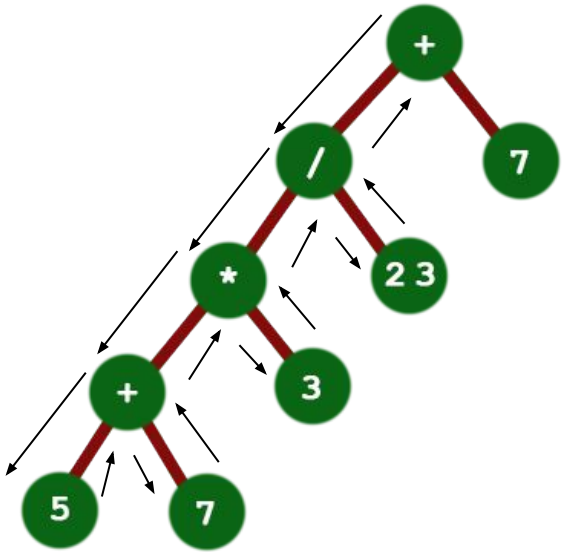
Operators become nodes; operands become leaves.

We can then walk the tree to produce our output. There are three methods.

Pre-order: Node, Left child, Right child

In-order: Left child, Node, Right child

Post-order: Left child, Right child, Node



5 + 7 * 3 / 23 +

Reverse Polish Notation

Answer: TREES!

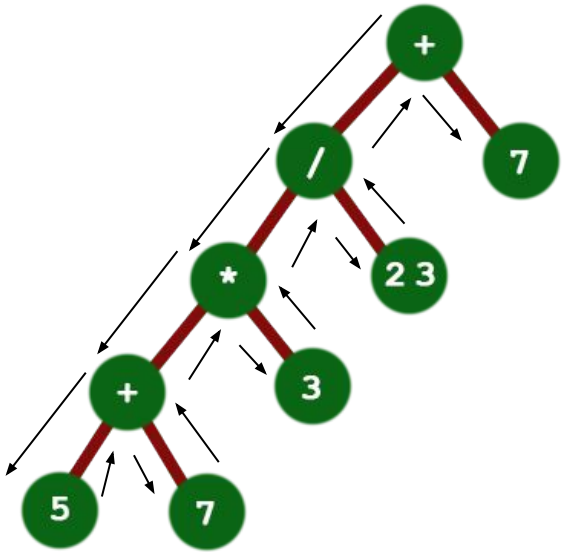
Operators become nodes; operands become leaves.

We can then walk the tree to produce our output. There are three methods.

Pre-order: Node, Left child, Right child

In-order: Left child, Node, Right child

Post-order: Left child, Right child, Node



5 + 7 * 3 / 23 + 7

Reverse Polish Notation

Answer: TREES!

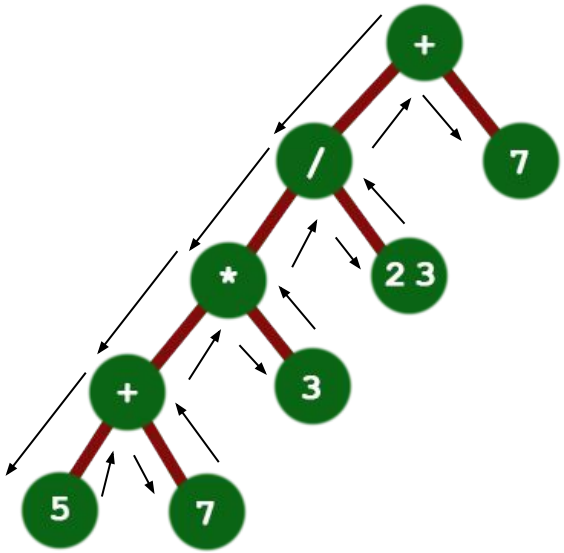
Operators become nodes; operands become leaves.

We can then walk the tree to produce our output. There are three methods.

Pre-order: Node, Left child, Right child

In-order: Left child, Node, Right child

Post-order: Left child, Right child, Node



(((5 + 7) * 3) / 23) + 7

(Operator precedence matters!)

Reverse Polish Notation

Answer: TREES!

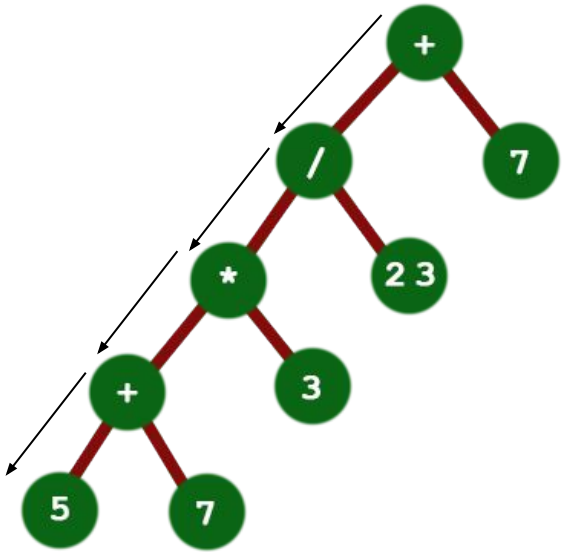
Operators become nodes; operands become leaves.

We can then walk the tree to produce our output. There are three methods.

Pre-order: Node, Left child, Right child

In-order: Left child, Node, Right child

Post-order: Left child, Right child, Node



Reverse Polish Notation

Answer: TREES!

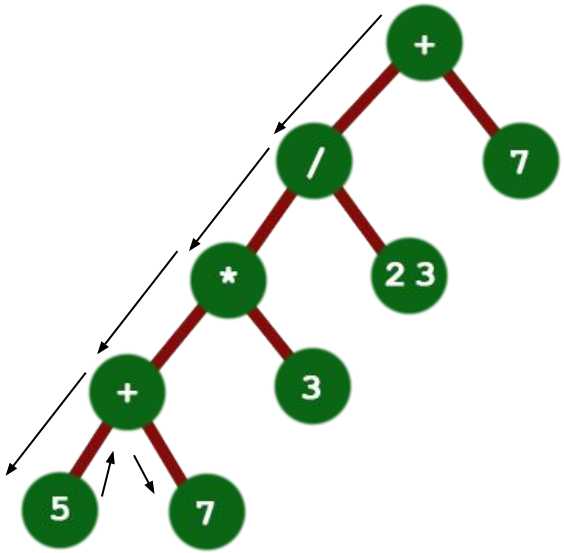
Operators become nodes; operands become leaves.

We can then walk the tree to produce our output. There are three methods.

Pre-order: Node, Left child, Right child

In-order: Left child, Node, Right child

Post-order: Left child, Right child, Node



5 7

Reverse Polish Notation

Answer: TREES!

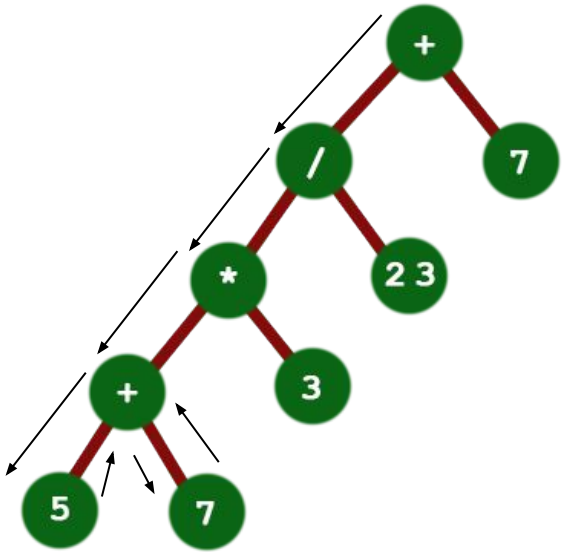
Operators become nodes; operands become leaves.

We can then walk the tree to produce our output. There are three methods.

Pre-order: Node, Left child, Right child

In-order: Left child, Node, Right child

Post-order: Left child, Right child, Node



5 7 +

Reverse Polish Notation

Answer: TREES!

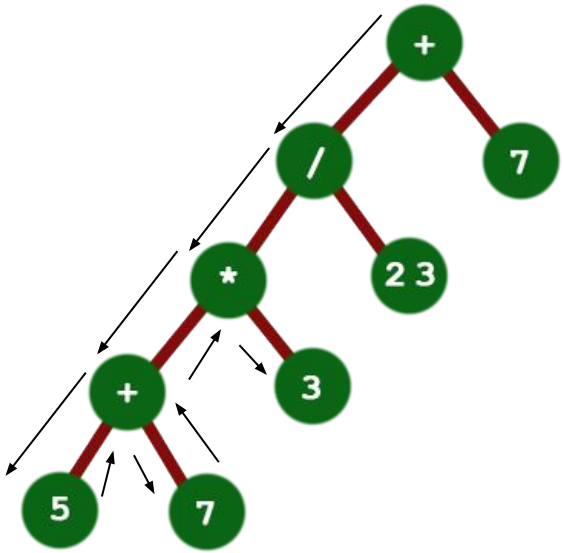
Operators become nodes; operands become leaves.

We can then walk the tree to produce our output. There are three methods.

Pre-order: Node, Left child, Right child

In-order: Left child, Node, Right child

Post-order: Left child, Right child, Node



5 7 + 3

Reverse Polish Notation

Answer: TREES!

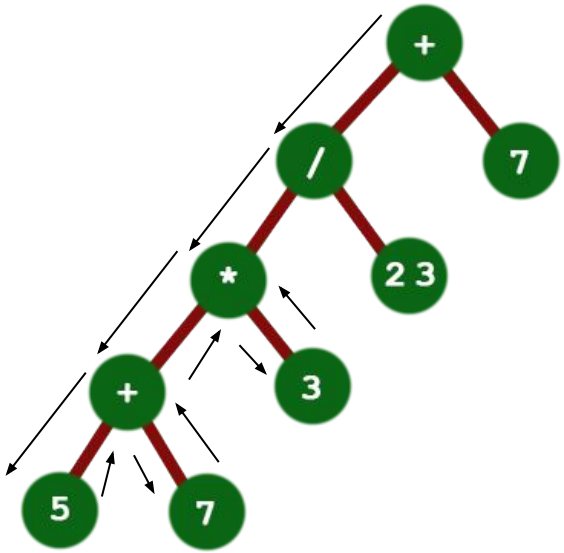
Operators become nodes; operands become leaves.

We can then walk the tree to produce our output. There are three methods.

Pre-order: Node, Left child, Right child

In-order: Left child, Node, Right child

Post-order: Left child, Right child, Node



5 7 + 3 *

Reverse Polish Notation

Answer: TREES!

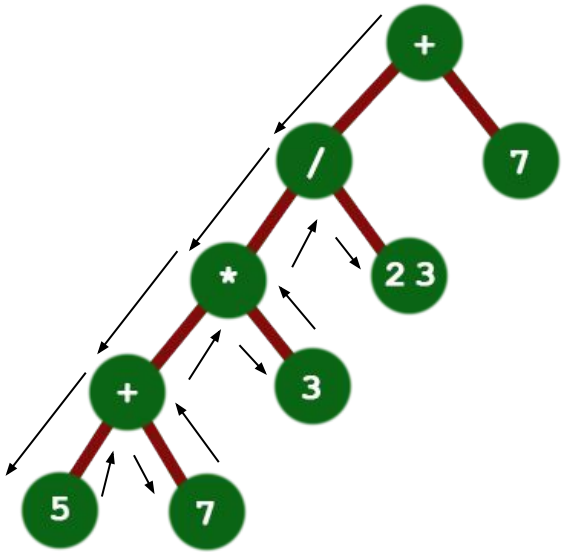
Operators become nodes; operands become leaves.

We can then walk the tree to produce our output. There are three methods.

Pre-order: Node, Left child, Right child

In-order: Left child, Node, Right child

Post-order: Left child, Right child, Node



5 7 + 3 * 23

Reverse Polish Notation

Answer: TREES!

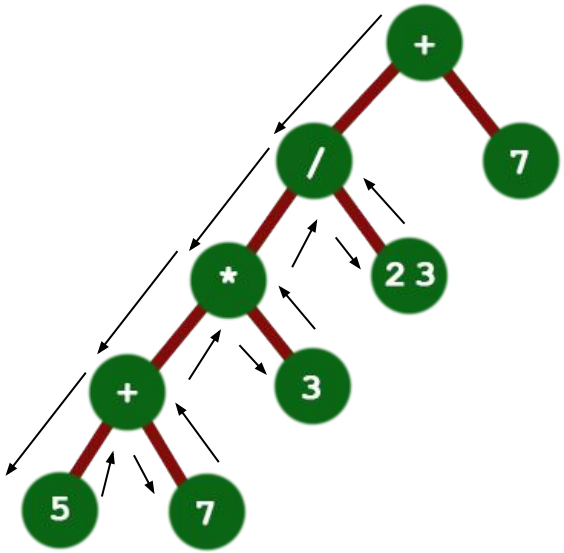
Operators become nodes; operands become leaves.

We can then walk the tree to produce our output. There are three methods.

Pre-order: Node, Left child, Right child

In-order: Left child, Node, Right child

Post-order: Left child, Right child, Node



5 7 + 3 * 23 /

Reverse Polish Notation

Answer: TREES!

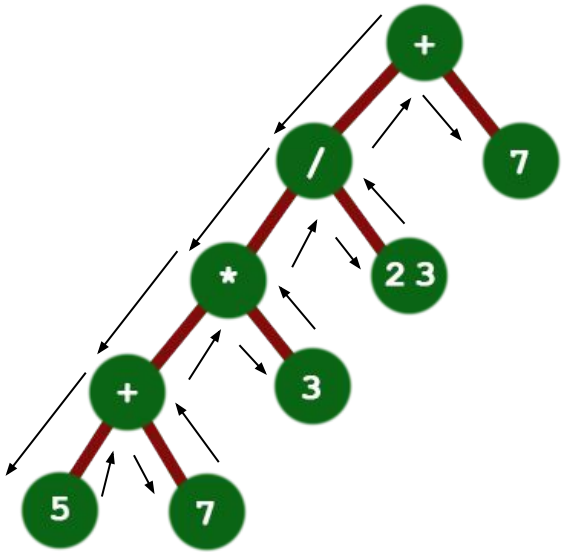
Operators become nodes; operands become leaves.

We can then walk the tree to produce our output. There are three methods.

Pre-order: Node, Left child, Right child

In-order: Left child, Node, Right child

Post-order: Left child, Right child, Node



5 7 + 3 * 23 / 7

Reverse Polish Notation

Answer: TREES!

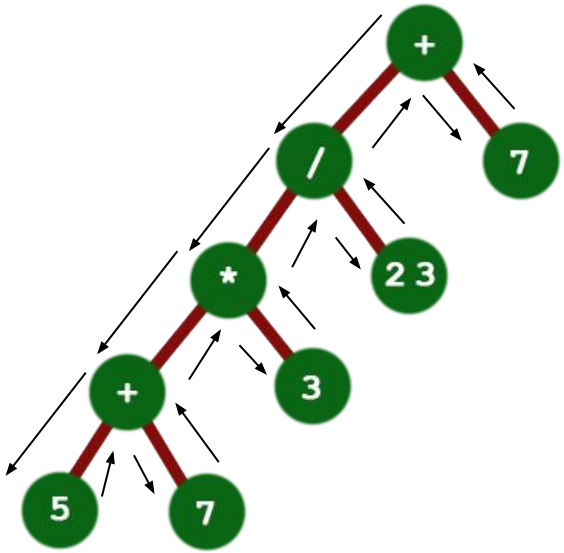
Operators become nodes; operands become leaves.

We can then walk the tree to produce our output. There are three methods.

Pre-order: Node, Left child, Right child

In-order: Left child, Node, Right child

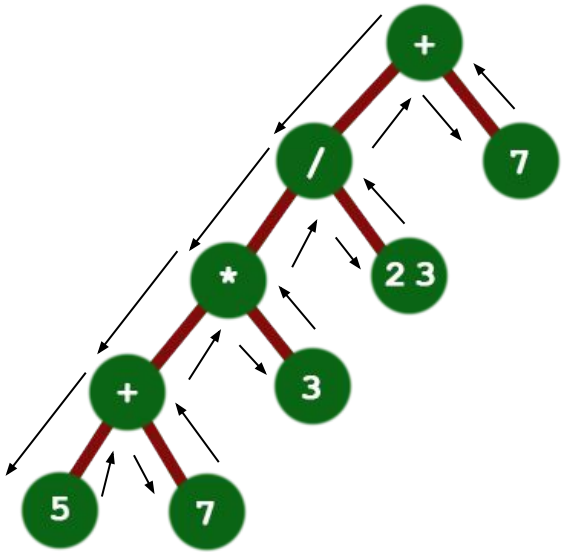
Post-order: Left child, Right child, Node



5 7 + 3 * 23 / 7 +

Reverse Polish Notation

How do we get it into the tree in the first place?

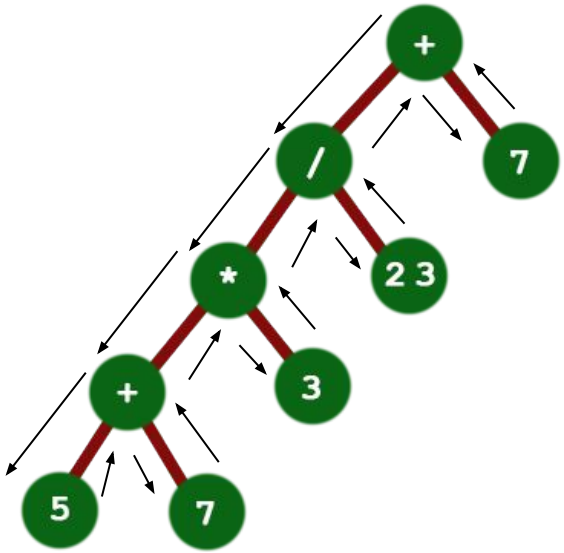


5 7 + 3 * 23 / 7 +

Reverse Polish Notation

How do we get it into the tree in the first place?

Parsers will do it - creating an *Abstract Syntax Tree* - but operator precedence is messy.



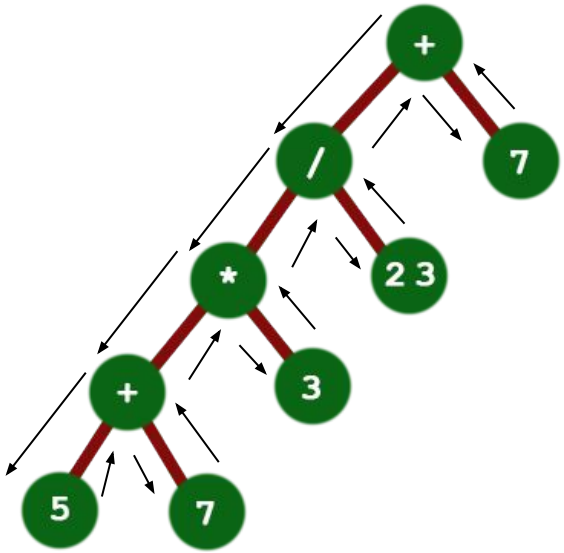
5 7 + 3 * 23 / 7 +

Reverse Polish Notation

How do we get it into the tree in the first place?

Parsers will do it - creating an *Abstract Syntax Tree* - but operator precedence is messy.

Enter Edsger Dijkstra again - his “shunting yard” algorithm is outside the scope of this talk.

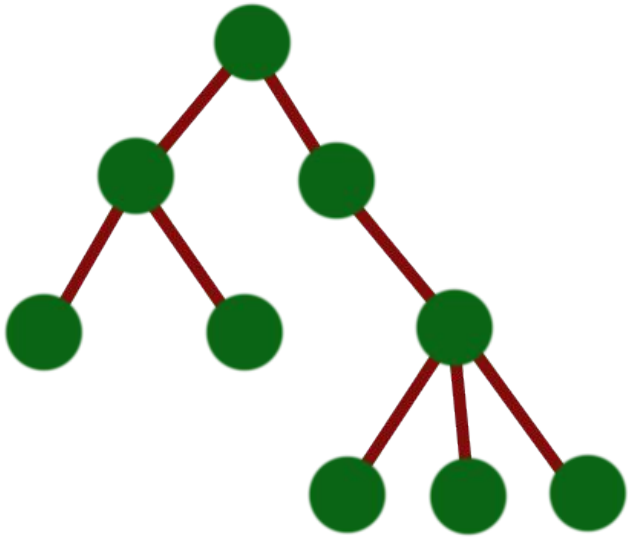


5 7 + 3 * 23 / 7 +

Nested sets

Nested sets

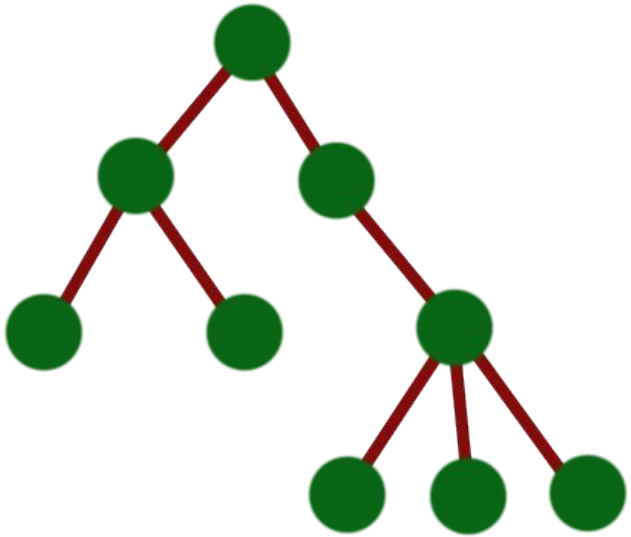
Problem: how do you store a tree in a database?



Nested sets

Problem: how do you store a tree in a database?

Answer: use a GraphDB like neo4j

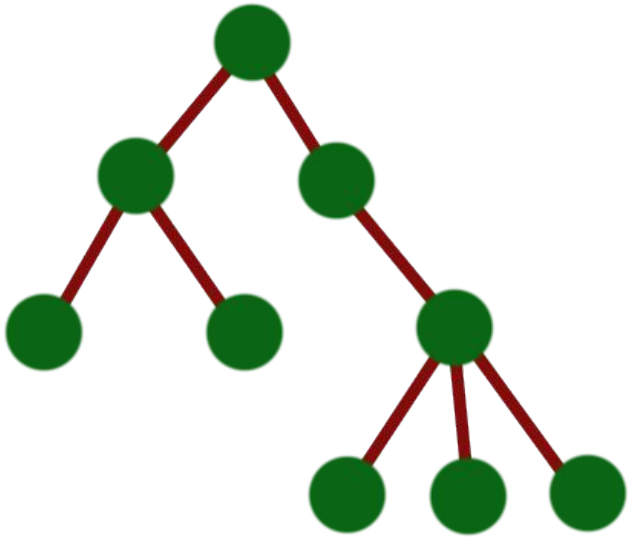


HIPSTER OVERLOAD!

(Fine, we’re just afraid of change)

Nested sets

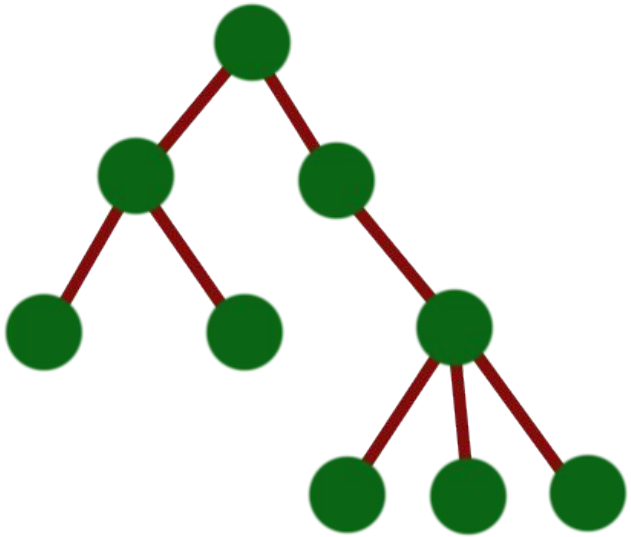
Problem: how do you store a tree in a RDBMS?



Nested sets

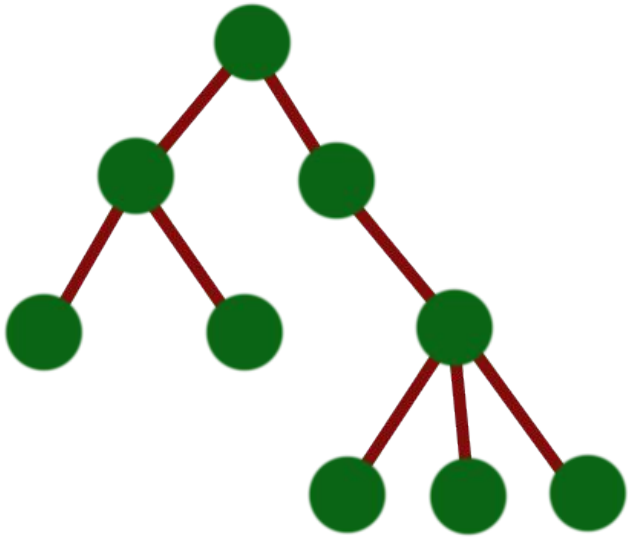
Problem: how do you store a tree in a RDBMS?

Easy! Each record has an id and a parent id; the root is the record with no parent id.



Nested sets

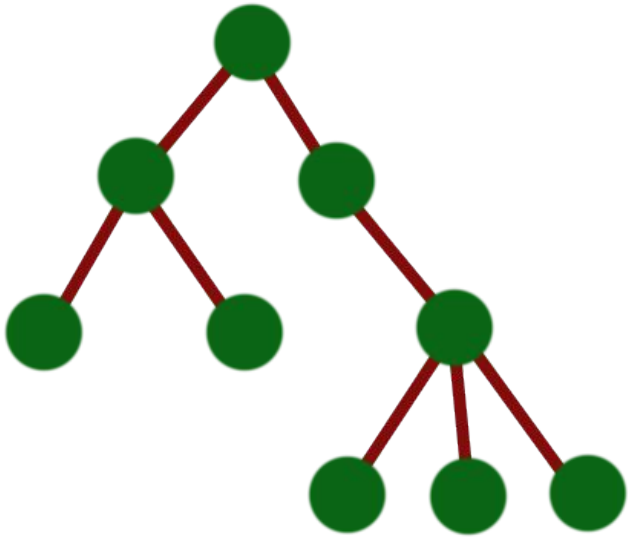
Problem: how do you retrieve all of a node’s antecedents in a RDBMS?



Nested sets

Problem: how do you retrieve all of a node's antecedents in a RDBMS?

Easy! Get that node's id, find all records with that as a parent id. Then for each of those records find all records with the parent id matching that record's id. Then for each of *those* records…

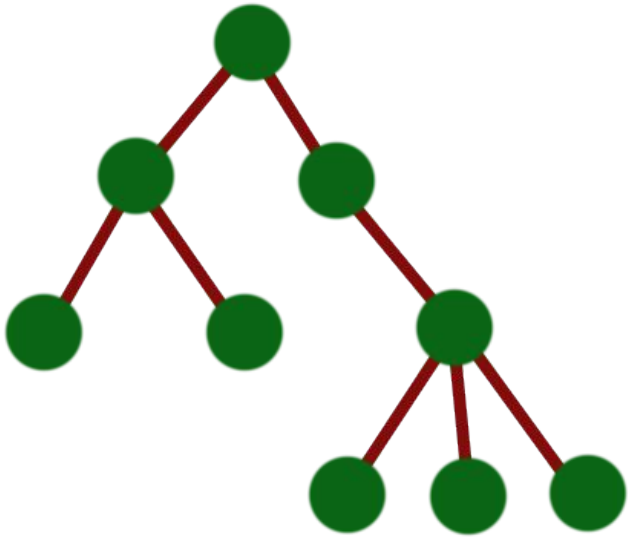


Nested sets

Problem: how do you retrieve all of a node's antecedents in a RDBMS?

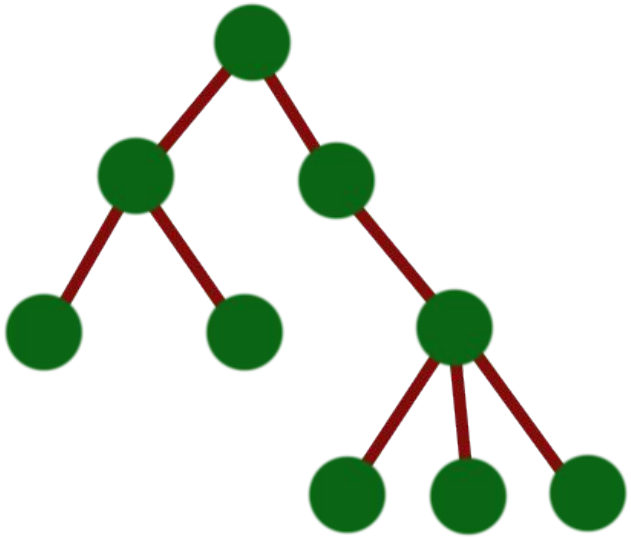
Easy! Get that node's id, find all records with that as a parent id. Then for each of those records find all records with the parent id matching that record's id. Then for each of *those* records…

The above solution can easily spiral out of control and generate huge numbers of queries.



Nested sets

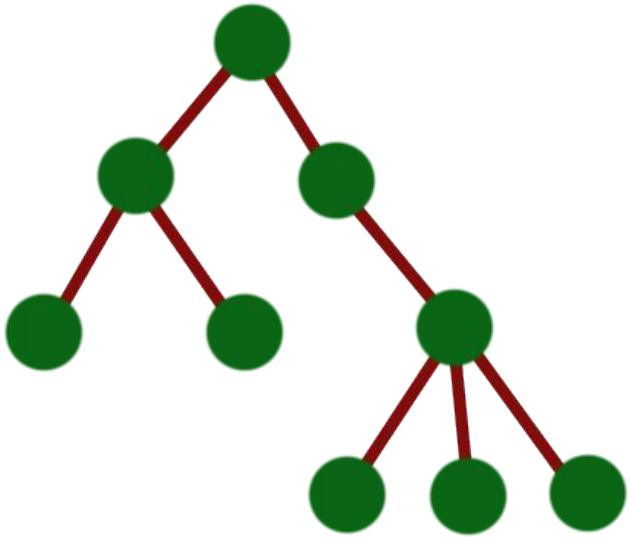
This is where *nested sets* come in.



Nested sets

This is where *nested sets* come in.

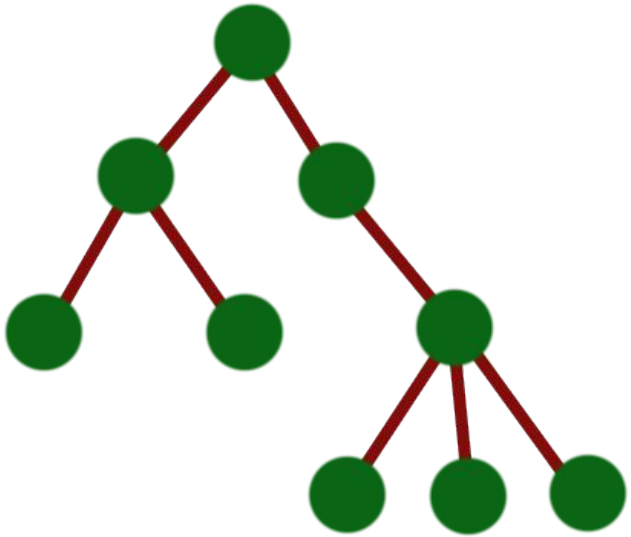
Add two more columns to your records, *left* and *right*.



Nested sets

This is where *nested sets* come in.

Add two more columns to your records, *left* and *right*.

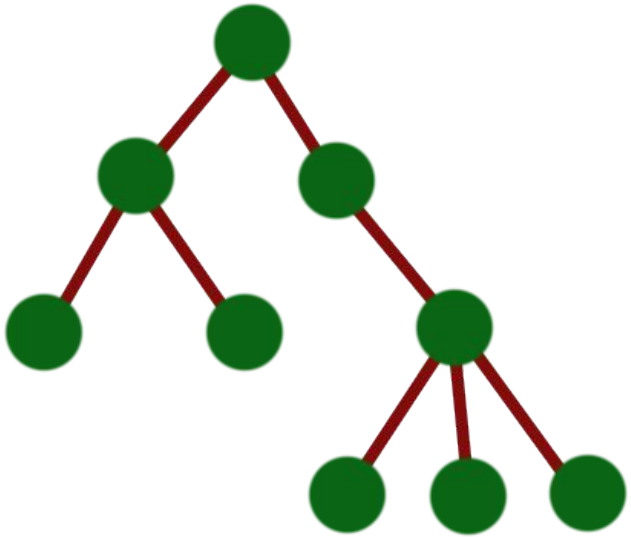


Nested sets

This is where *nested sets* come in.

Add two more columns to your records, *left* and *right*.

Then to store the tree, walk it while counting the times you enter a node. If it’s your first time there, set its left value to the counter; if you’ve been there before, set its right value to the counter.

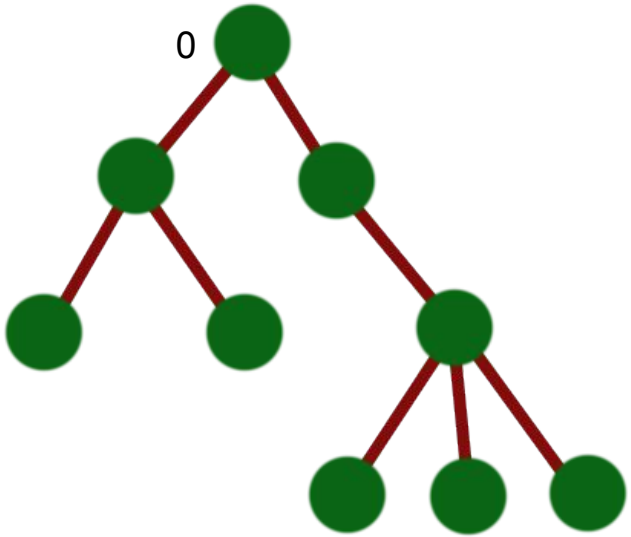


Nested sets

This is where *nested sets* come in.

Add two more columns to your records, *left* and *right*.

Then to store the tree, walk it while counting the times you enter a node. If it’s your first time there, set its left value to the counter.

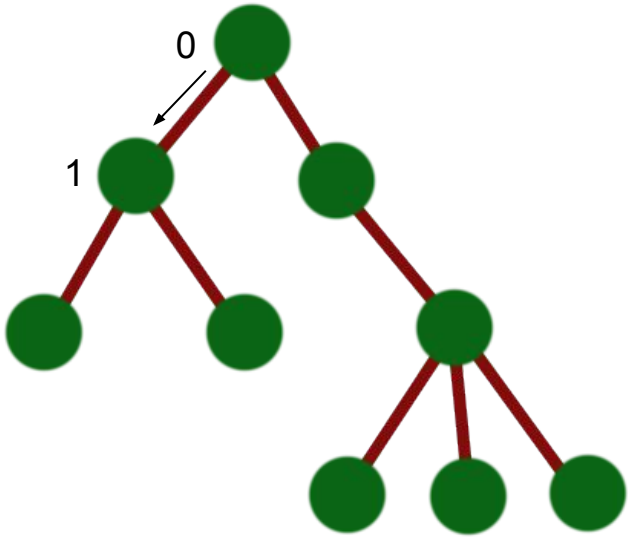


Nested sets

This is where *nested sets* come in.

Add two more columns to your records, *left* and *right*.

Then to store the tree, walk it while counting the times you enter a node. If it’s your first time there, set its left value to the counter.

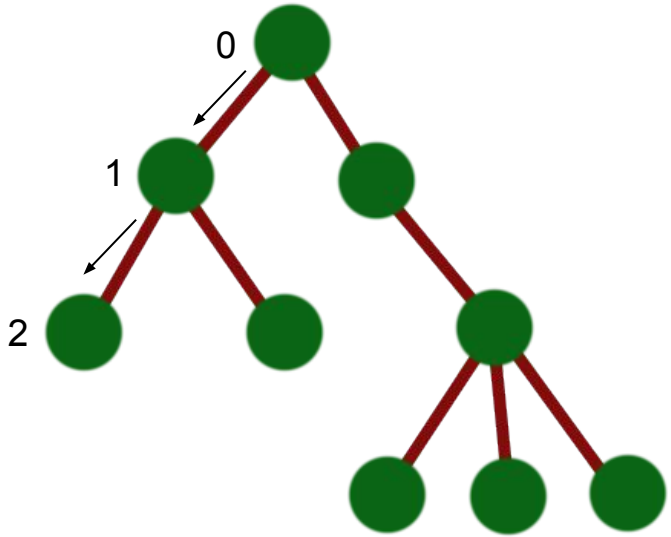


Nested sets

This is where *nested sets* come in.

Add two more columns to your records, *left* and *right*.

Then to store the tree, walk it while counting the times you enter a node. If it’s your first time there, set its left value to the counter.



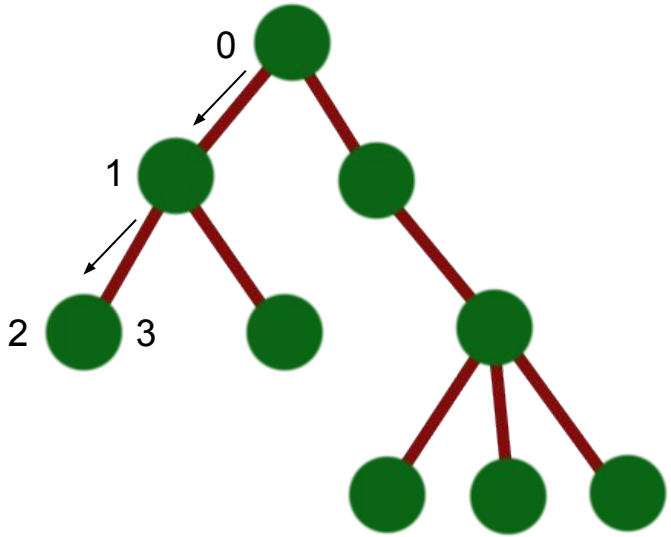
Nested sets

This is where *nested sets* come in.

Add two more columns to your records, *left* and *right*.

Then to store the tree, walk it while counting the times you enter a node. If it’s your first time there, set its left value to the counter.

If there are no more children to walk, set the node’s right value to the counter.



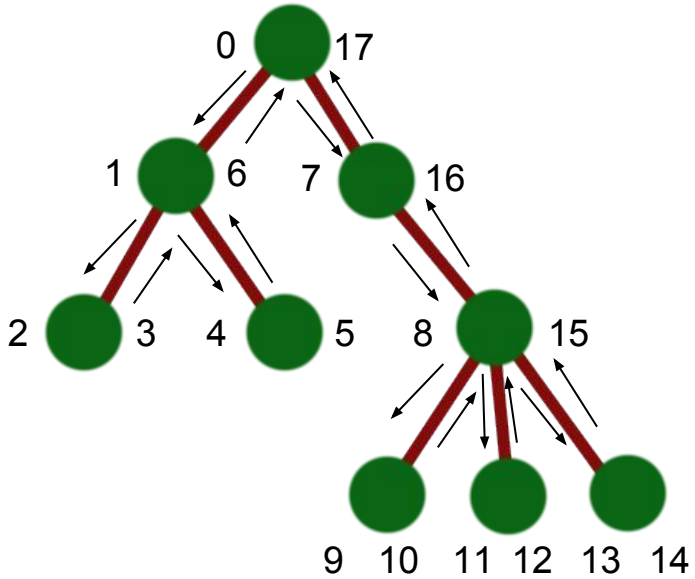
Nested sets

This is where *nested sets* come in.

Add two more columns to your records, *left* and *right*.

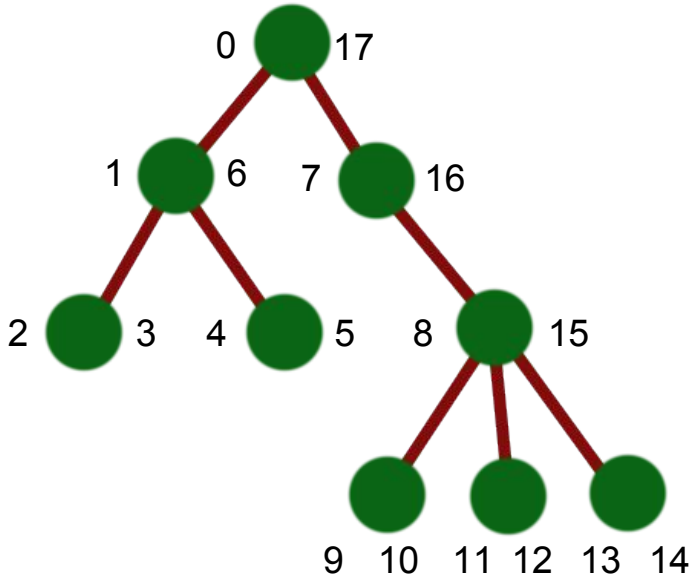
Then to store the tree, walk it while counting the times you enter a node. If it’s your first time there, set its left value to the counter.

If there are no more children to walk, set the node’s right value to the counter.

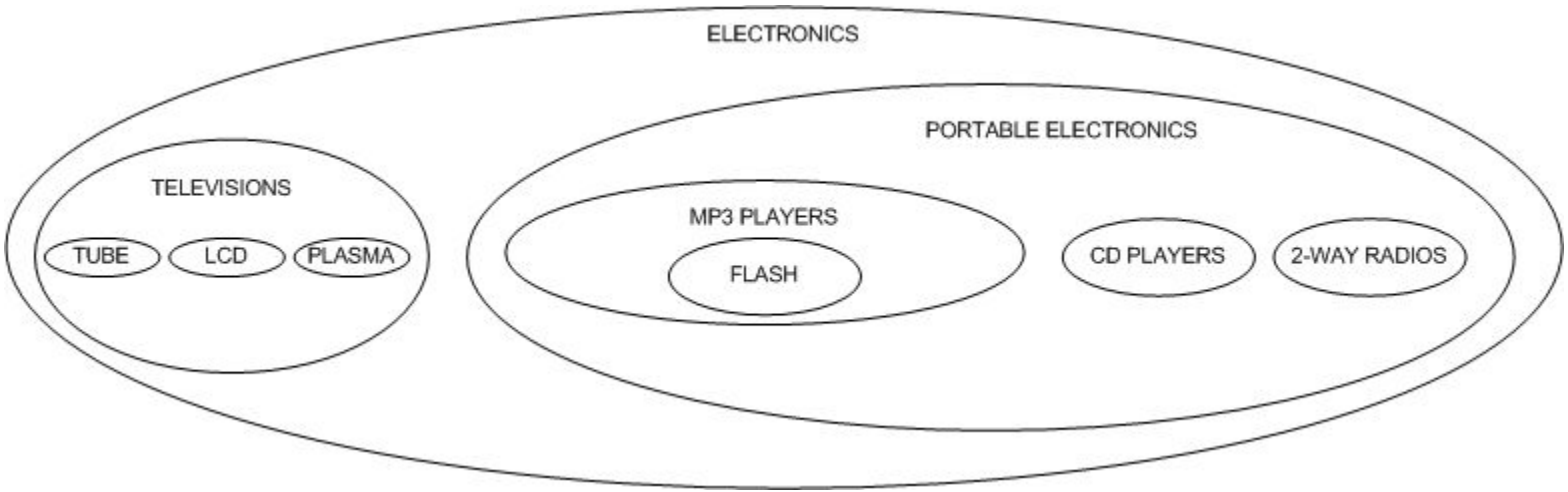


Nested sets

Every node now has the property that its children have left and right values between its own.



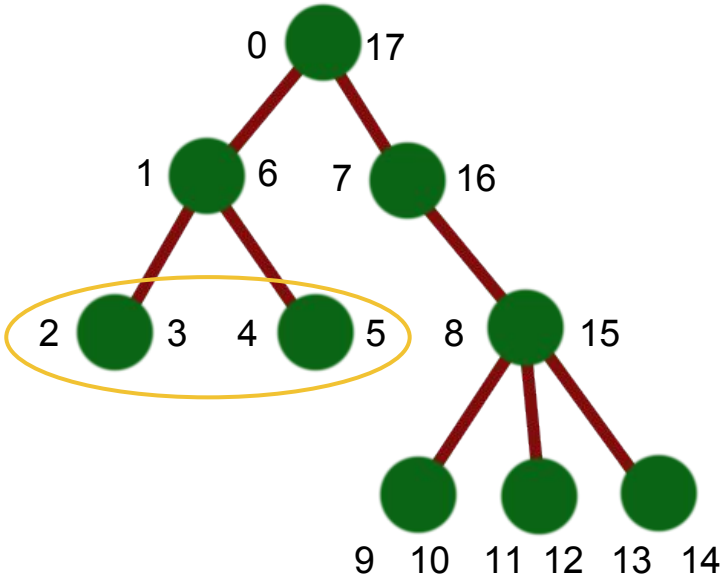
Nested sets


ELECTRONICS
PORTABLE ELECTRONICS
TELEVISIONS
TUBE
LCD
PLASMA
MP3 PLAYERS
FLASH
CD PLAYERS
2-WAY RADIOS

Nested sets

Every node now has the property that its children have left and right values between its own.

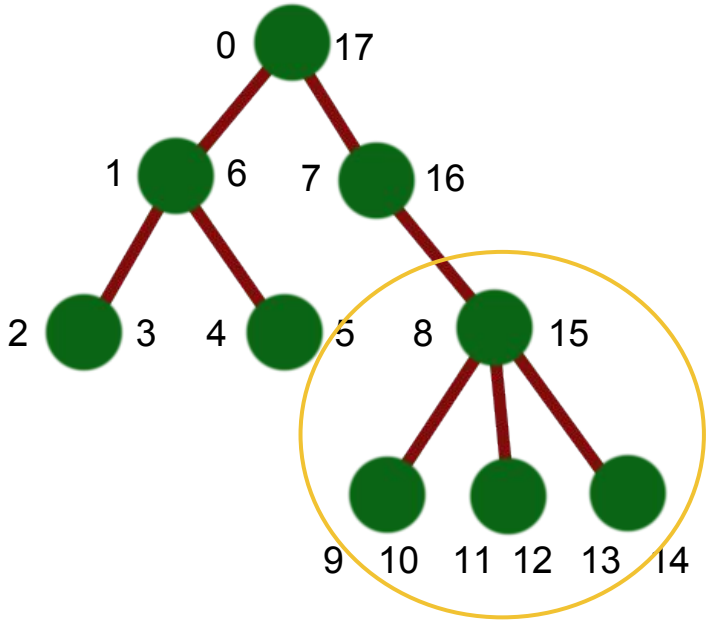
```
SELECT * FROM nodes
WHERE left > 1 AND right < 6
```



Nested sets

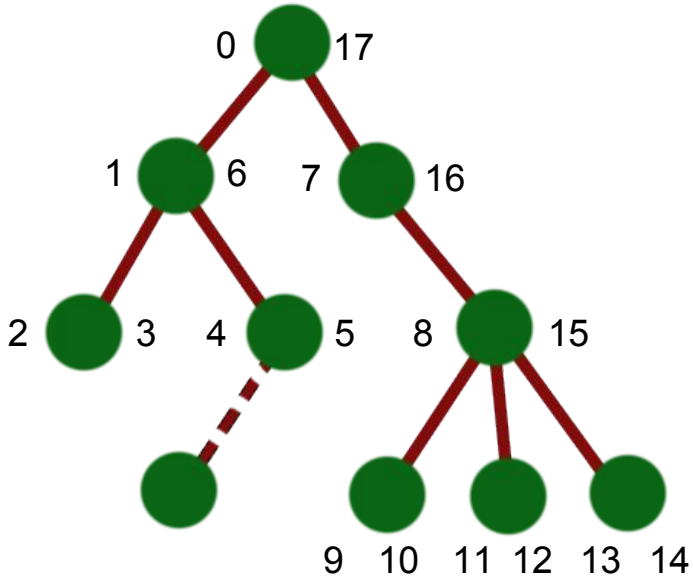
Every node now has the property that its children have left and right values between its own.

```
SELECT * FROM nodes
WHERE left > 7 AND right < 16
```



Nested sets

Insertion in SQL is then easy - with the left value for the parent node:



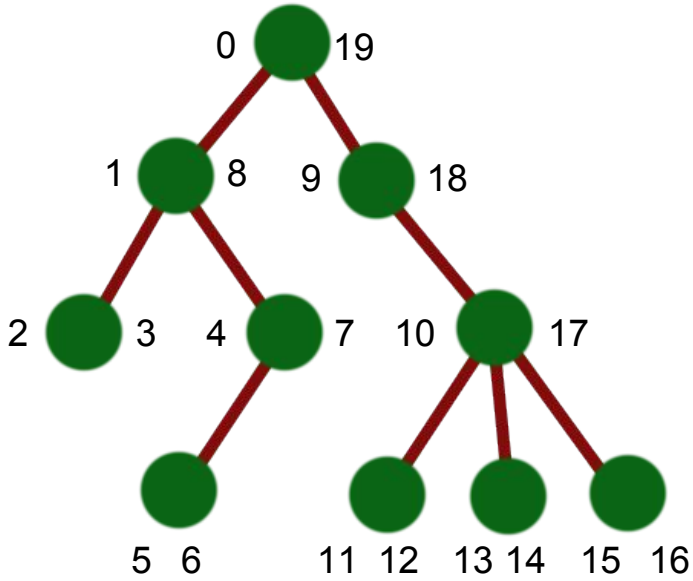
Nested sets

Insertion in SQL is then easy - with the left value for the parent node as *pleft*:

```
UPDATE node SET left = left + 2
WHERE left > pleft;

UPDATE node SET right = right + 2
WHERE right > pleft;

INSERT INTO node (left, right)
VALUES (pleft + 1, pleft + 2);
```

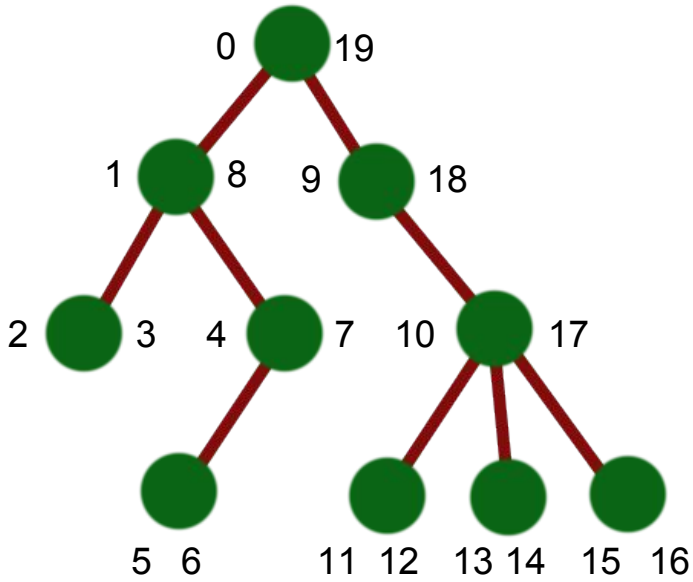


Nested sets

Deletion in SQL is easy too - with the left value for the node to be deleted as *pleft*:

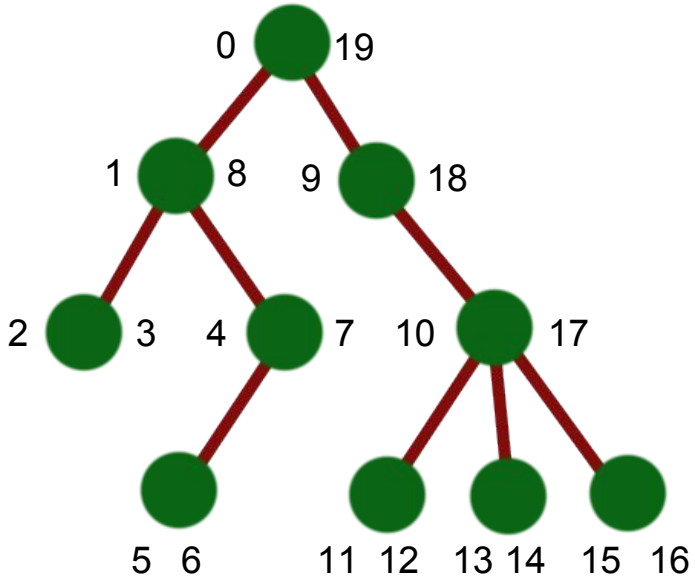
```
UPDATE node SET left = left - 2
WHERE left > pleft;

UPDATE node SET right = right - 2
WHERE right > pleft;
```



Nested sets

There are extensions to do this for you in both versions of Doctrine, Eloquent and Propel.



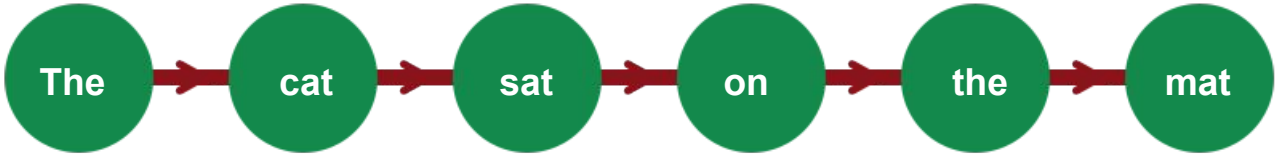
Depth-first: Modelling language

Depth-first: Modelling language

Sentences can be modelled as directed graphs:

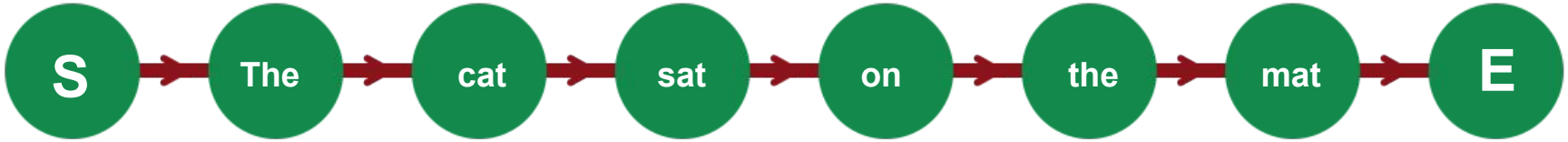
Depth-first: Modelling language

Sentences can be modelled as directed graphs:



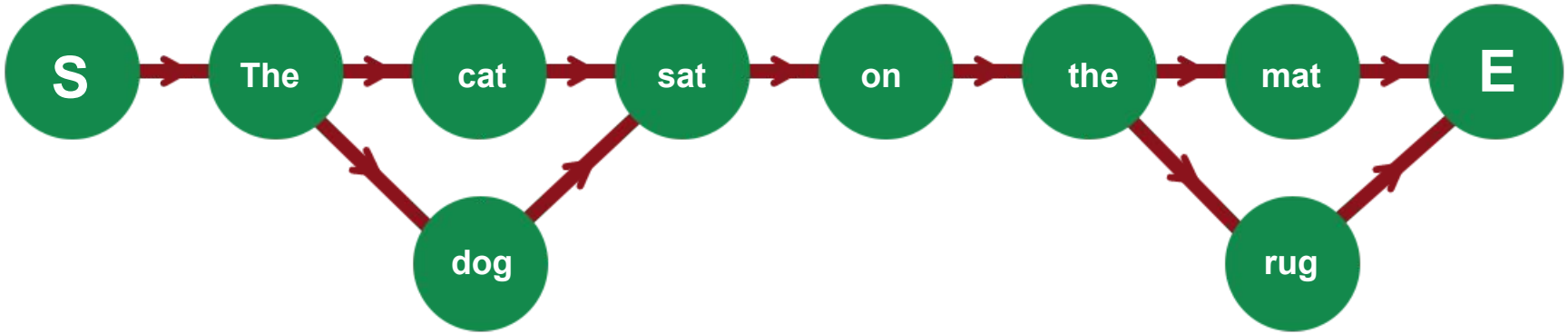
Depth-first: Modelling languages

We can then add special start and end nodes:



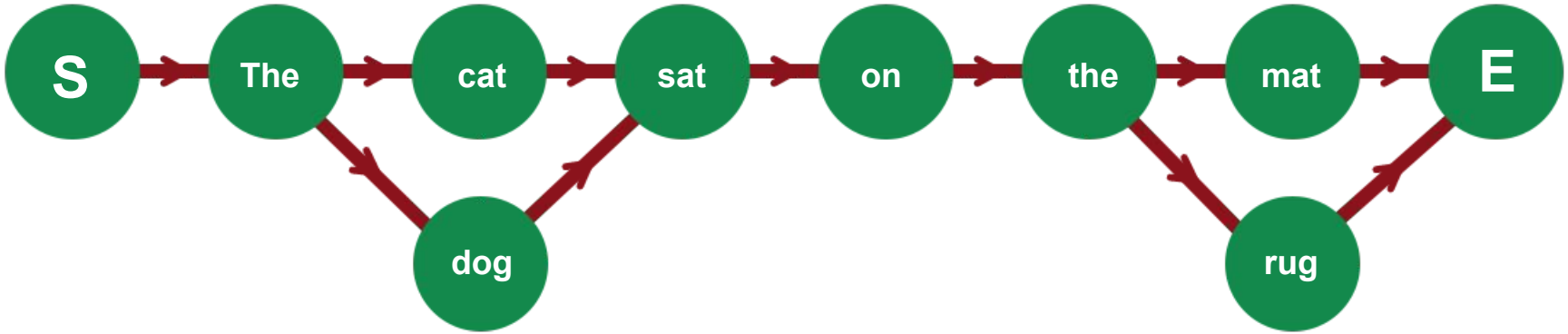
Depth-first: Modelling languages

Now add another sentence:



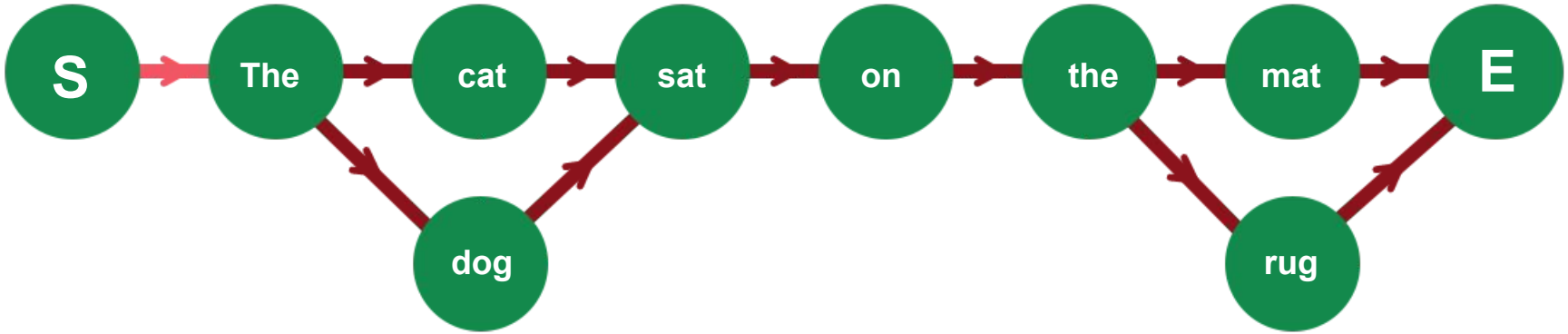
Depth-first: Modelling languages

We can now generate a novel sentence by randomly traversing it:



Depth-first: Modelling languages

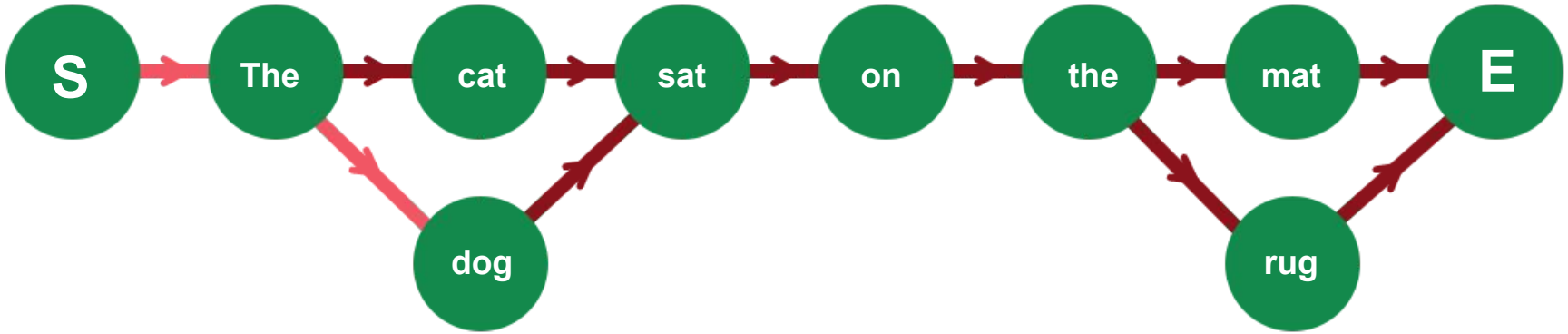
We can now generate a novel sentence by randomly traversing it:



The

Depth-first: Modelling languages

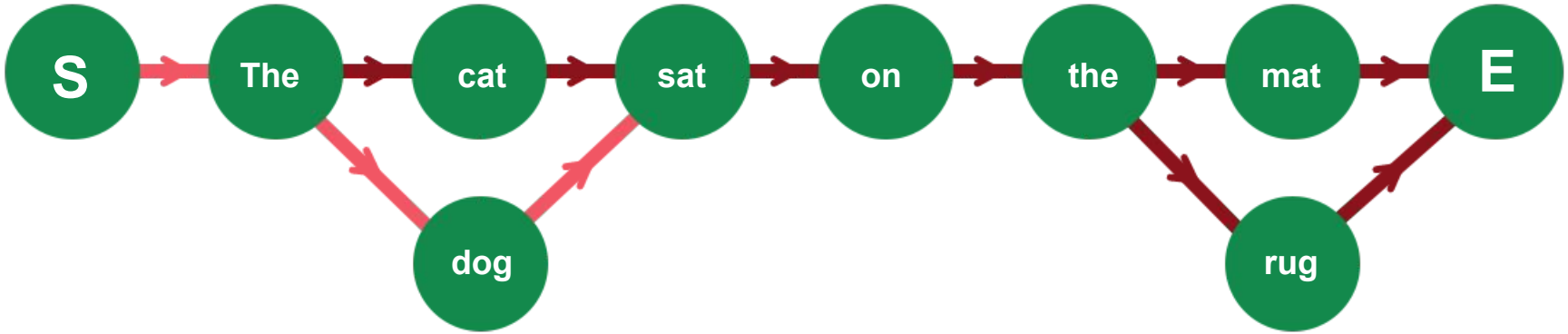
We can now generate a novel sentence by randomly traversing it:



The dog

Depth-first: Modelling languages

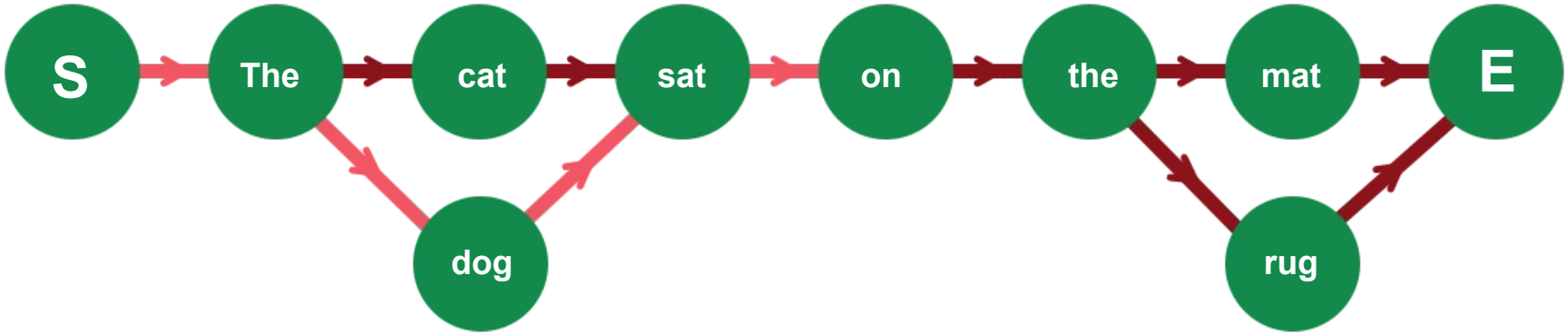
We can now generate a novel sentence by randomly traversing it:



The dog sat

Depth-first: Modelling languages

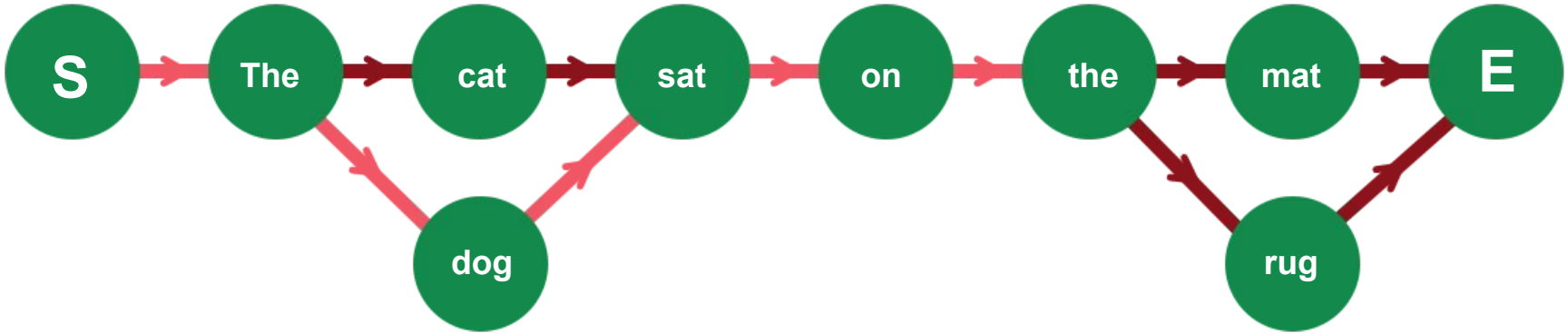
We can now generate a novel sentence by randomly traversing it:



The dog sat on

Depth-first: Modelling languages

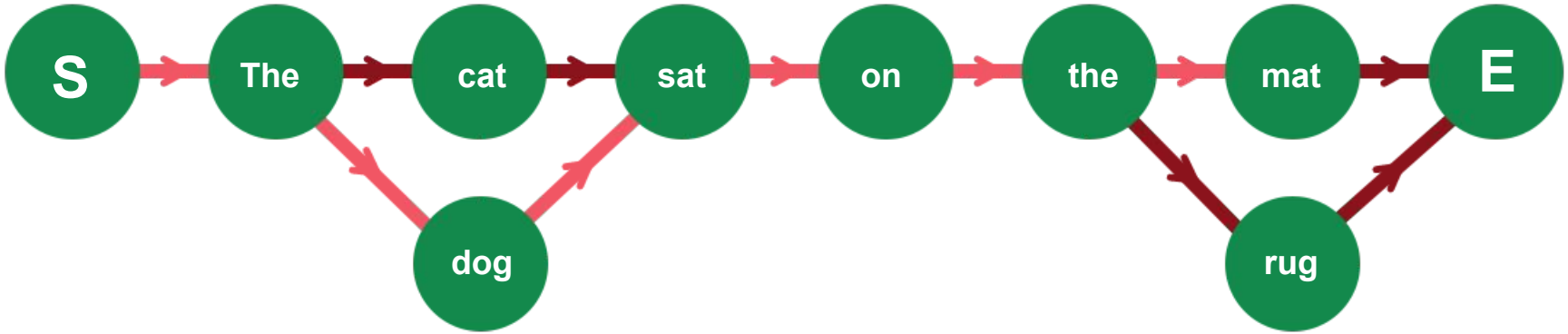
We can now generate a novel sentence by randomly traversing it:



The dog sat on the

Depth-first: Modelling languages

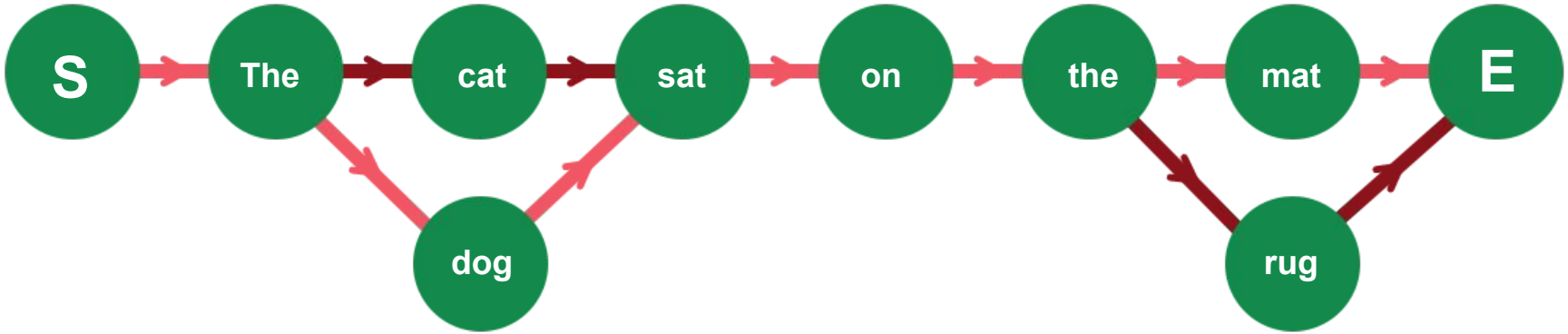
We can now generate a novel sentence by randomly traversing it:



The dog sat on the mat

Depth-first: Modelling languages

When we find the end node, we have a (structurally sound) sentence.



The dog sat on the mat.

Depth-first: Modelling sentences

Daily Mail Headline Generator

---

http://choult.com/dm

Source: http://bit.ly/2aNrWF4

Depth-first: Modelling sentences

NORTH KOREA FIRES A 60FT WELL IN THE FIRST JOB - SAYS HE WAS SHOT DEAD HIS EMOTIONAL VISIT TO LONG-FORGOTTEN 1846 WILD WEST SETTLERS WHO WON BY ISIS ATTACK

Depth-first: Modelling sentences

MICHAEL PHELPS GETS HIS FRIEND'S JAPANESE AKITA DOG

Depth-first: Modelling sentences

AWE-INSPIRING MOMENT AMERICA'S GOT TALENT ACT GOES GLUTEN-FREE WITH HER BROTHER

In Summary

Graphs are cool!

Questions?

Have you been affected by any of the issues in this talk?

Email: chris@choult.com

Twitter: @choult

LinkedIn: Christopher Hoult