Breadth-first and depth-first traversal

Prof. Noah Snavely
CS1114
http://cs1114.cs.cornell.edu

Cornell University
Computer Science

Administrivia

- Assignment 2, Part 2, due Friday
  - Please sign up for a Friday slot
- Assignment 3 will be out Friday

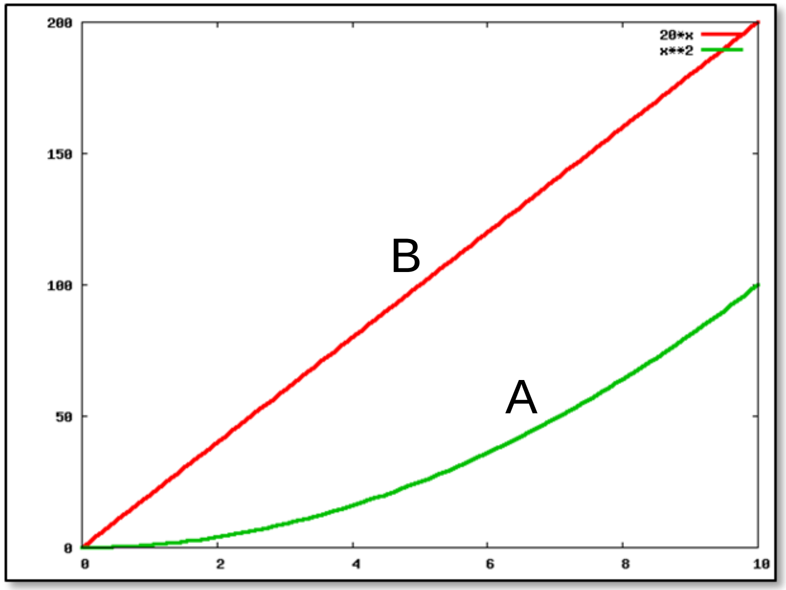
- Survey:
  - Should we move lecture closer to the lab?

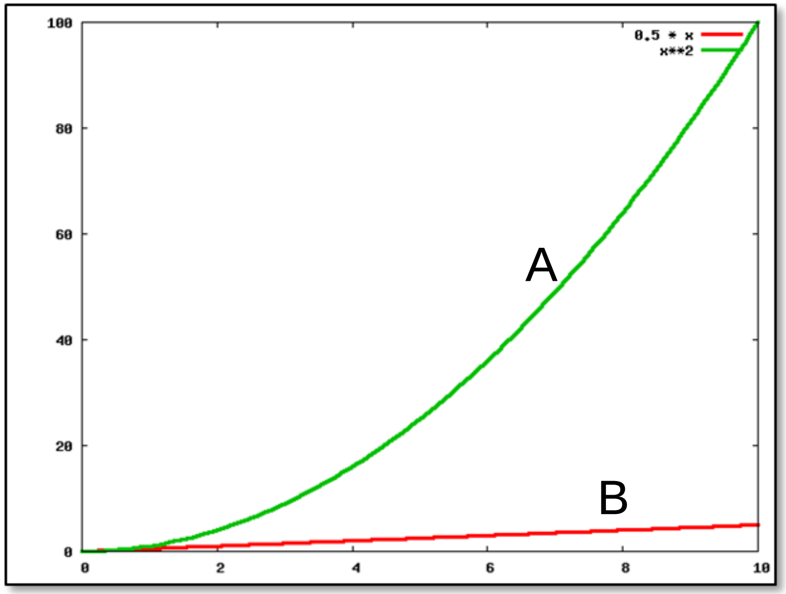
- Prelim 1! Next Thursday, 2/26, in class
  - There will be a review session TBA

Final notes on Big-O Notation

- If algorithm **A** is $O(n^2)$ and algorithm **B** is $O(n)$, we know that:
  - For large n, **A** will eventually run much slower than **B**
  - For small n, we know very little:
    - **A** could be slower
    - **B** could be slower
    - They could have similar runtimes
    - Or difference could be very large

Final notes on Big-O Notation





Finding blobs

1	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0
0	0	0	1	1	1	0	0	0	0
0	0	0	1	1	1	0	0	0	0
0	0	0	1	1	1	0	0	0	0
0	0	0	1	1	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Finding blobs

1	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0
0	0	0	1	1	1	0	0	0	0
0	0	0	1	1	1	0	0	0	0
0	0	0	1	1	1	0	0	0	0
0	0	0	1	1	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Blobs are connected components!

Finding components

1. Pick a 1 to start with, where you don't know which component it is in
   – When there aren't any, you're done
2. Give it a new component color
3. Assign the same component color to each pixel that is part of the same component
   – Basic strategy: color any neighboring 1's, have them color their neighbors, and so on

Finding components

- For each vertex we visit, we color its neighbors and remember that we need to visit them at some point
  - Need to keep track of the vertices we still need to visit in a todo list
  - After we visit a vertex, we'll pick one of the vertices in the todo list to visit next

- This is also called *graph traversal*

Stacks and queues

- Two ways of representing a "todo list"
- Stack: Last In First Out (LIFO)
  - (Think cafeteria trays)
  - The newest task is the one you'll do next
- Queue: First In First Out (FIFO)
  - (Think a line of people at the cafeteria)
  - The oldest task is the one you'll do next

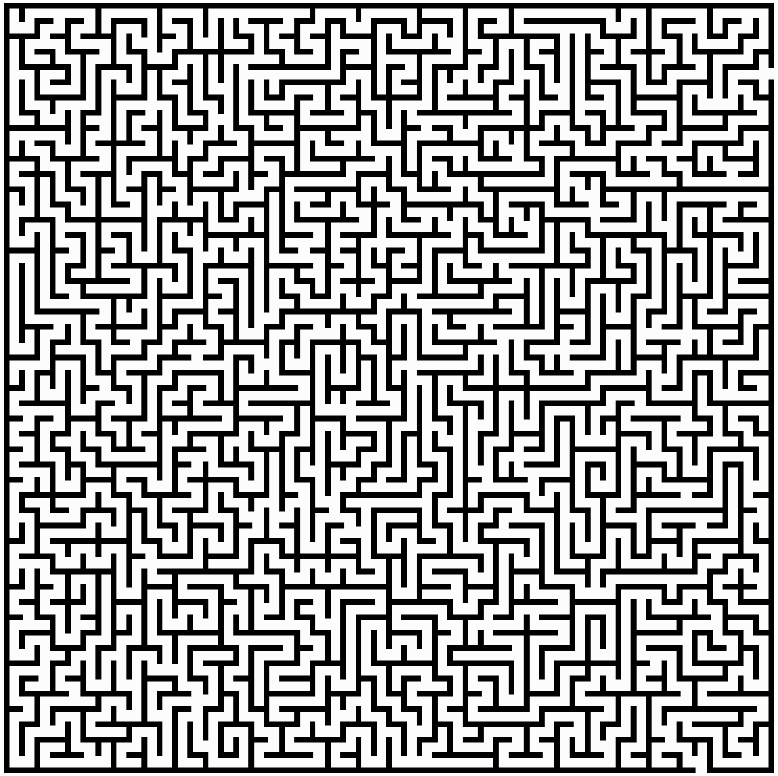
Stacks

- Two operations:
- Push: add something to the top of the stack
- Pop: remove the thing on top of the stack

Queue

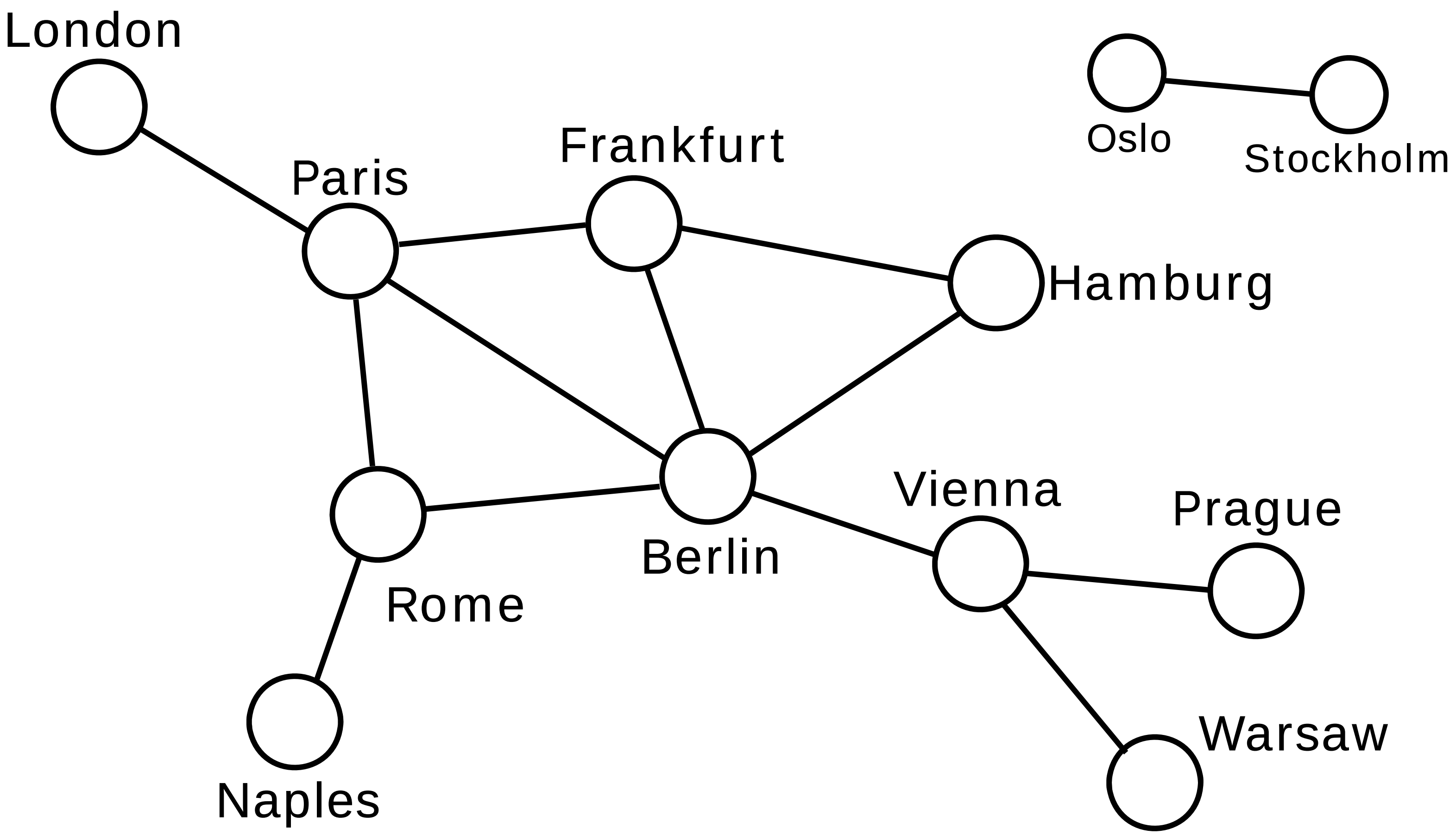
- Two operations:
- Enqueue: add something to the end of the queue
- Dequeue: remove something from the front of the queue

Graph traversal

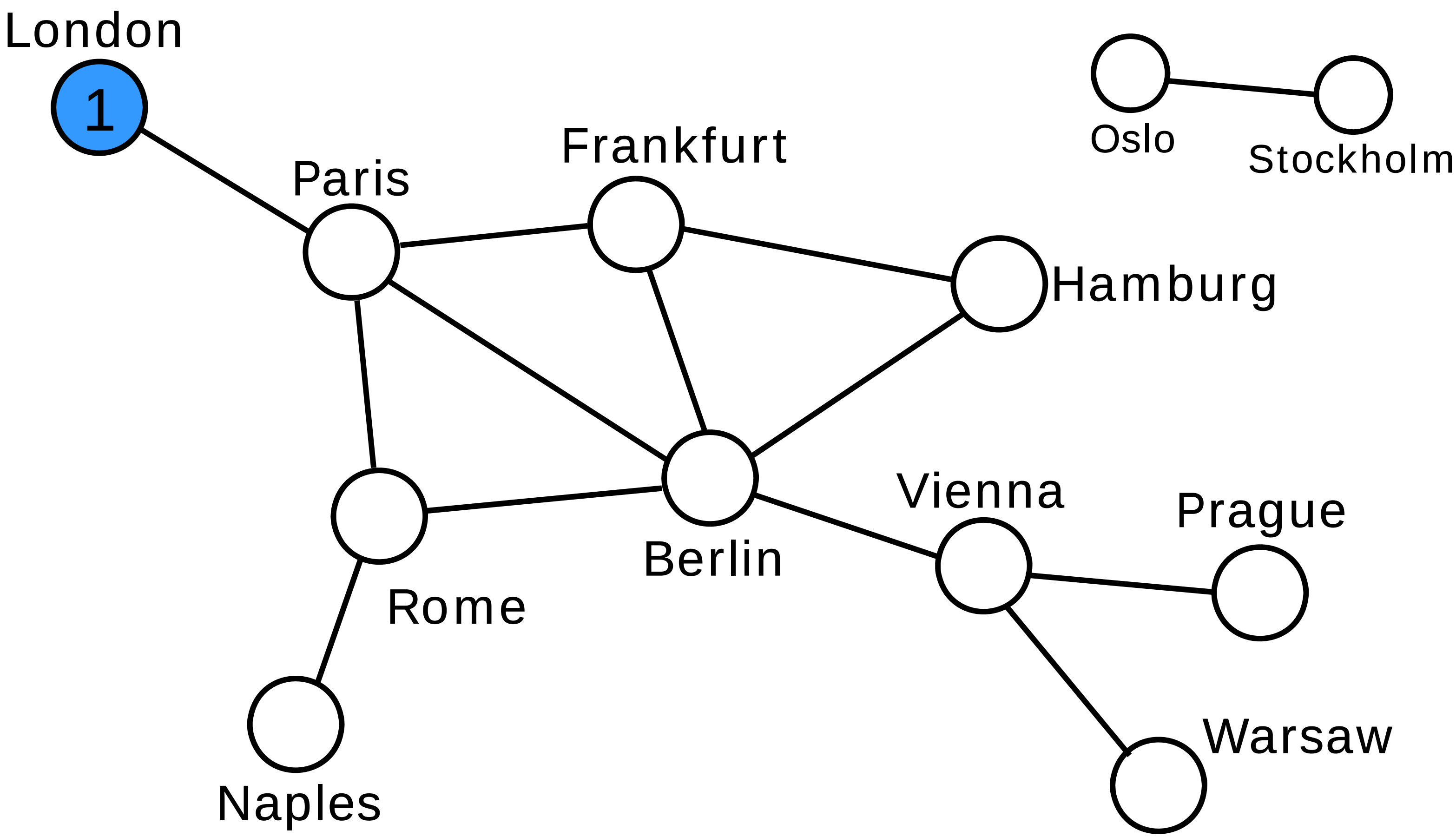


- Suppose you're in a maze
- What strategy can you use to find the exit?

Graph traversal

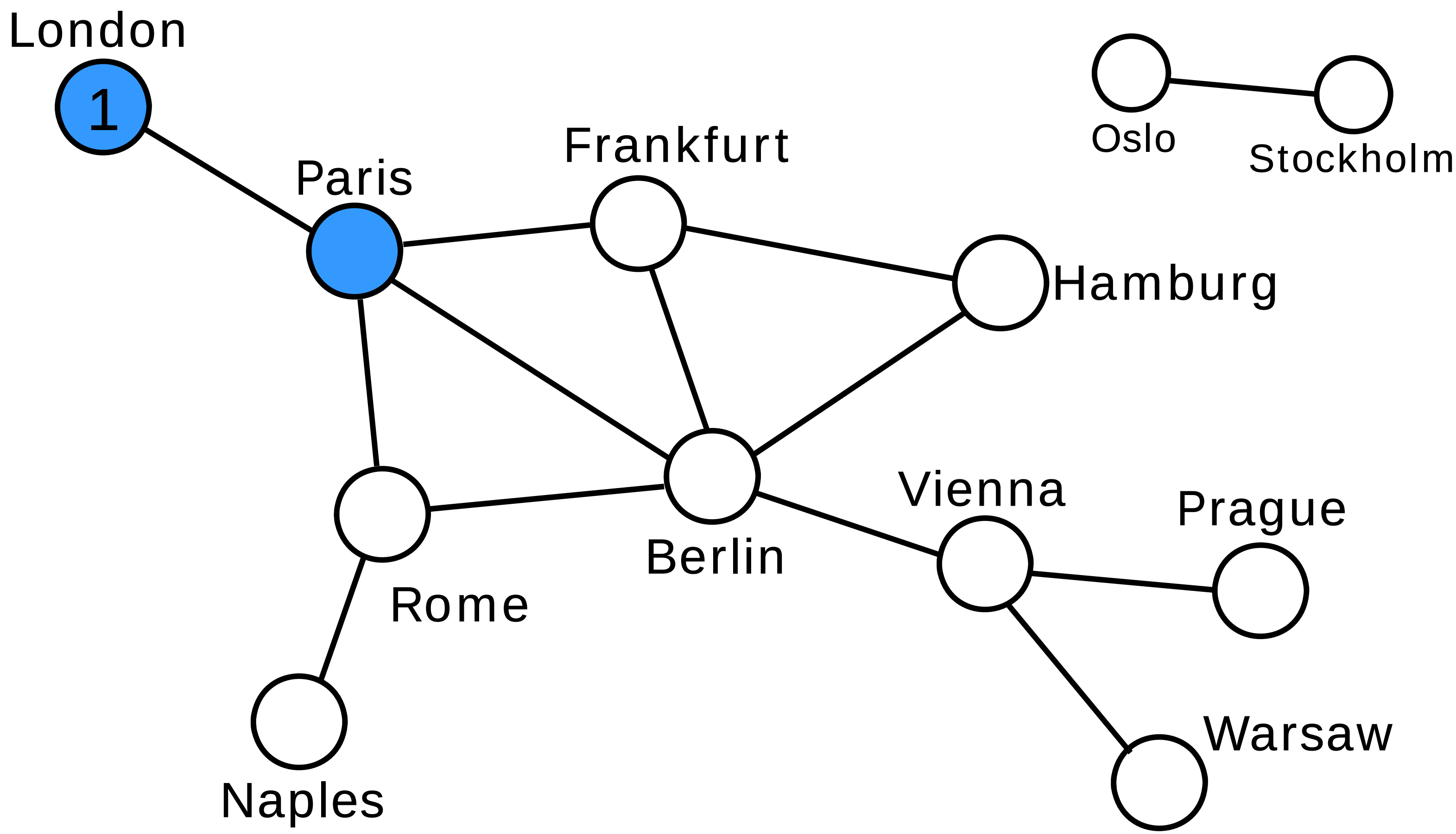


Graph traversal (stack)



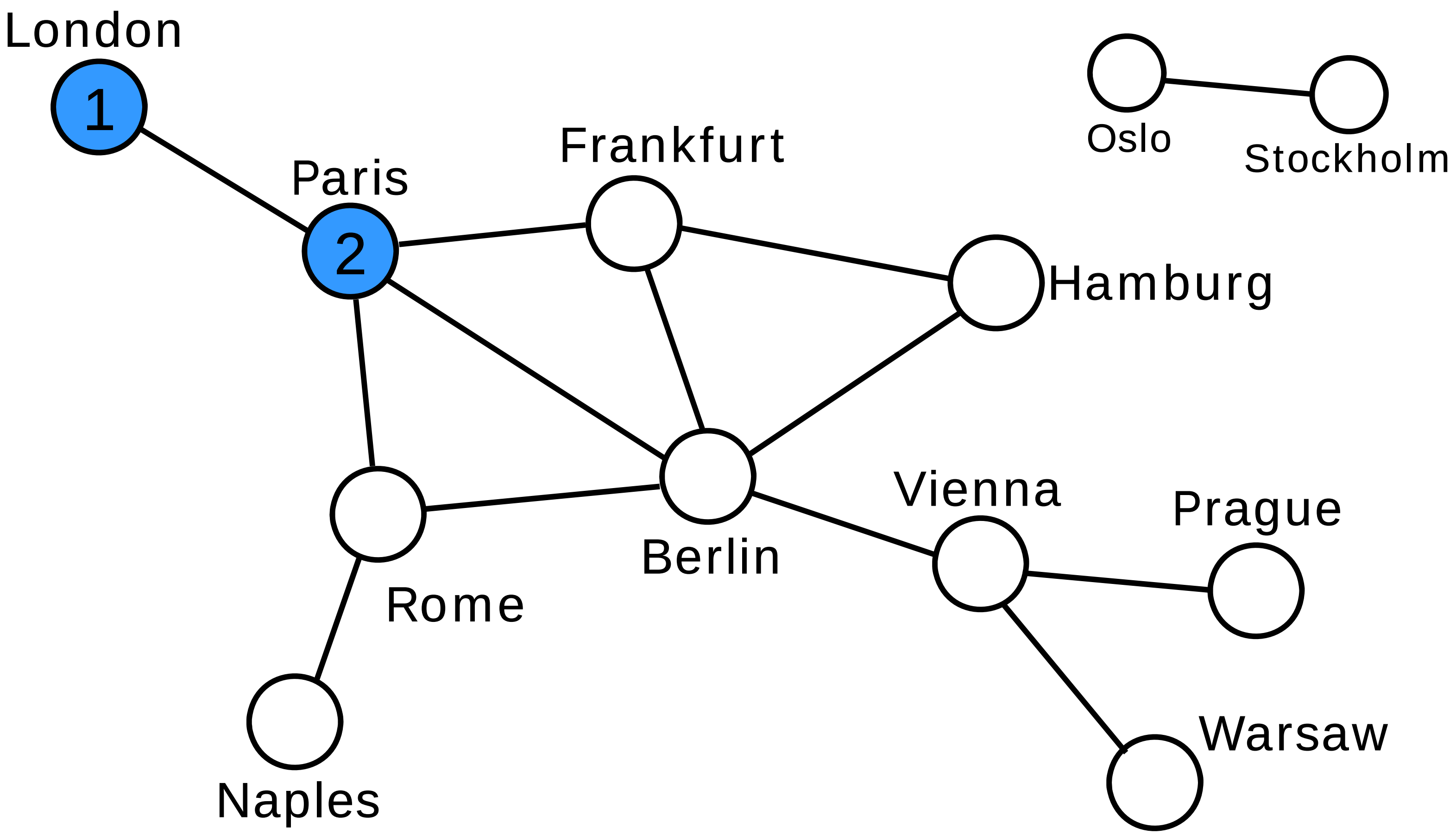
Current node: London
Todo list: [ ]

Graph traversal (stack)



Current node: London
Todo list: [Paris]

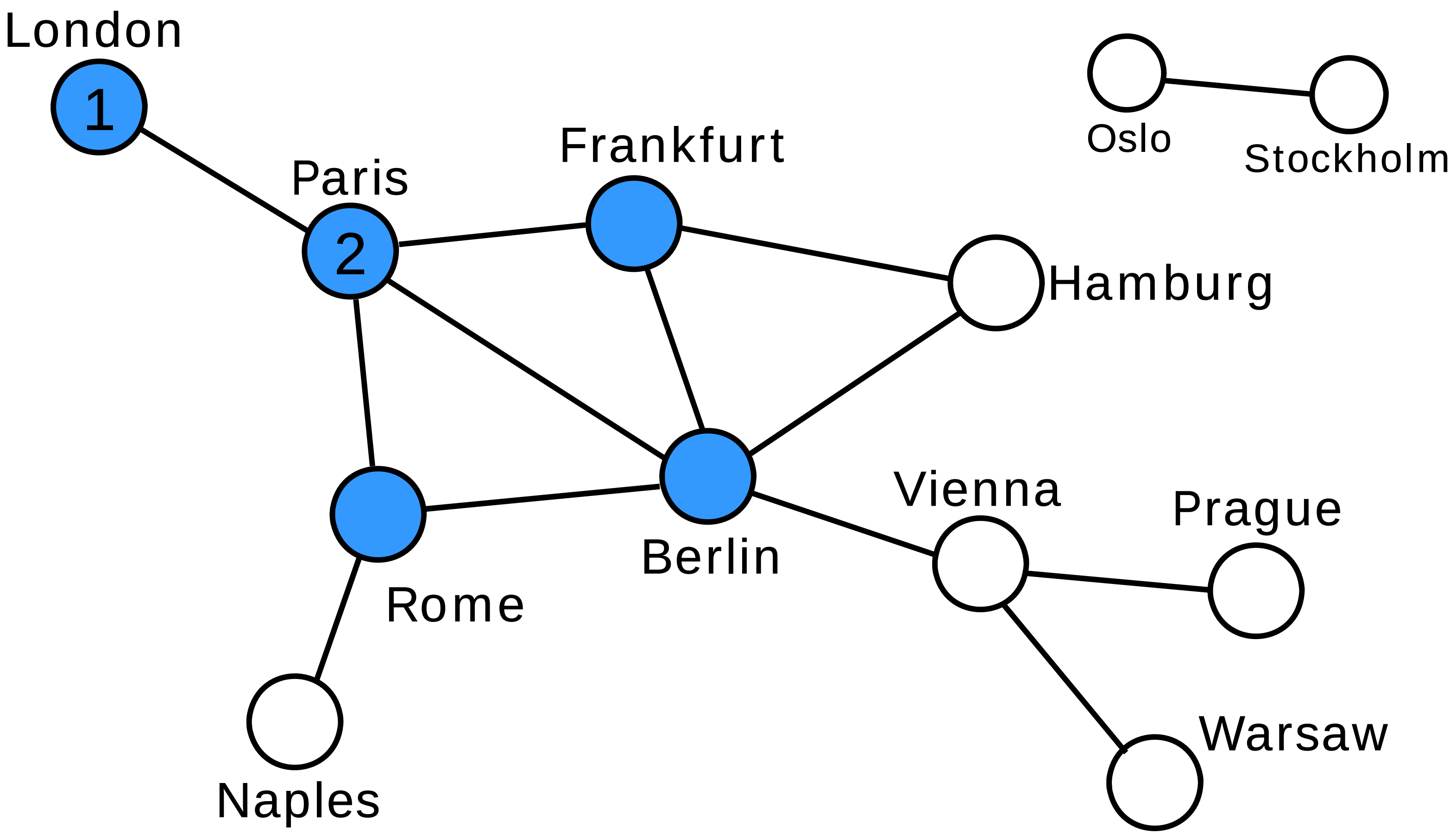
Graph traversal (stack)



Current node: Paris

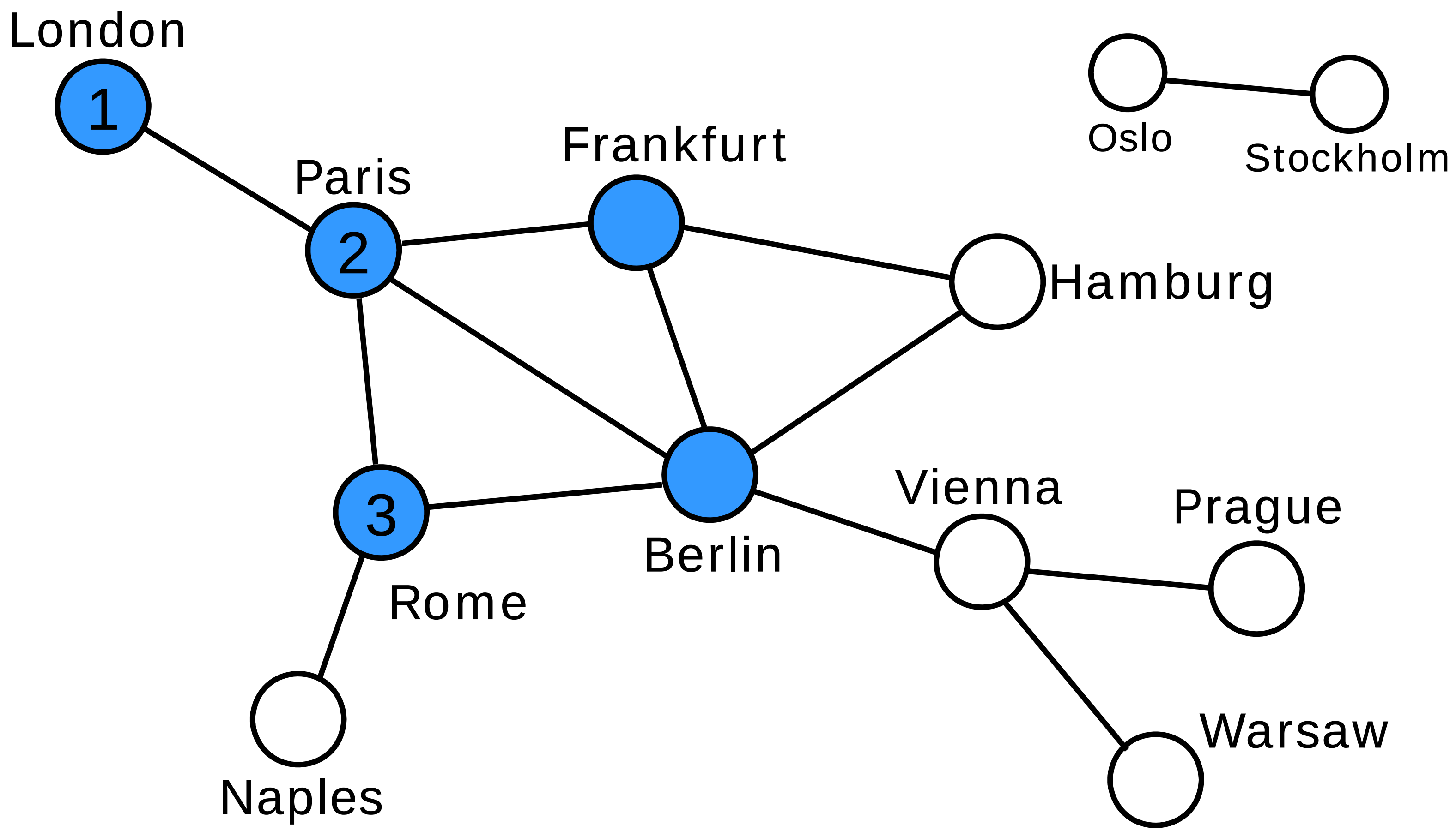
Todo list: [ ]

Graph traversal (stack)



Current node: Paris
Todo list: [Frankfurt, Berlin, Rome]

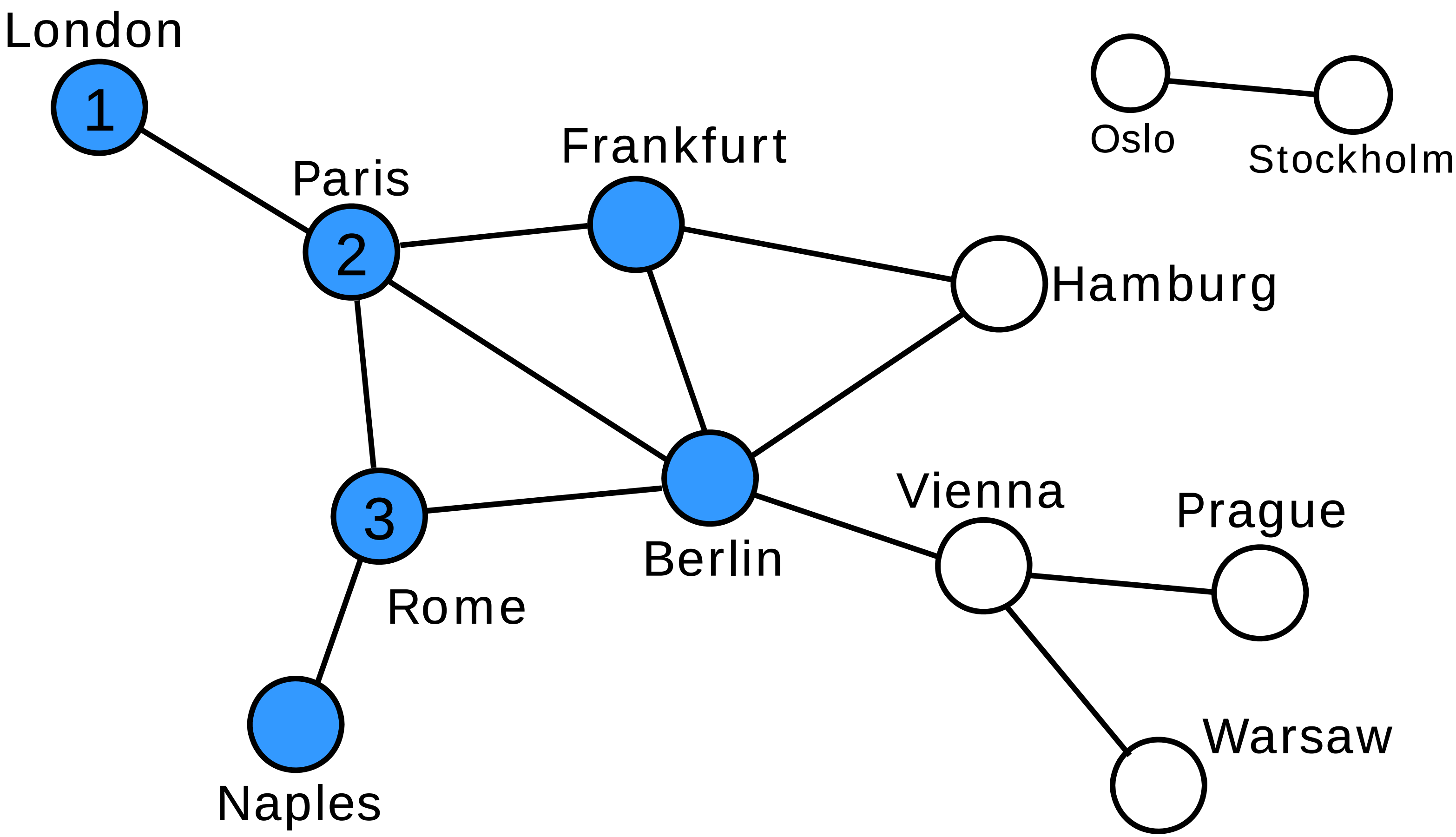
Graph traversal (stack)



Current node: Rome

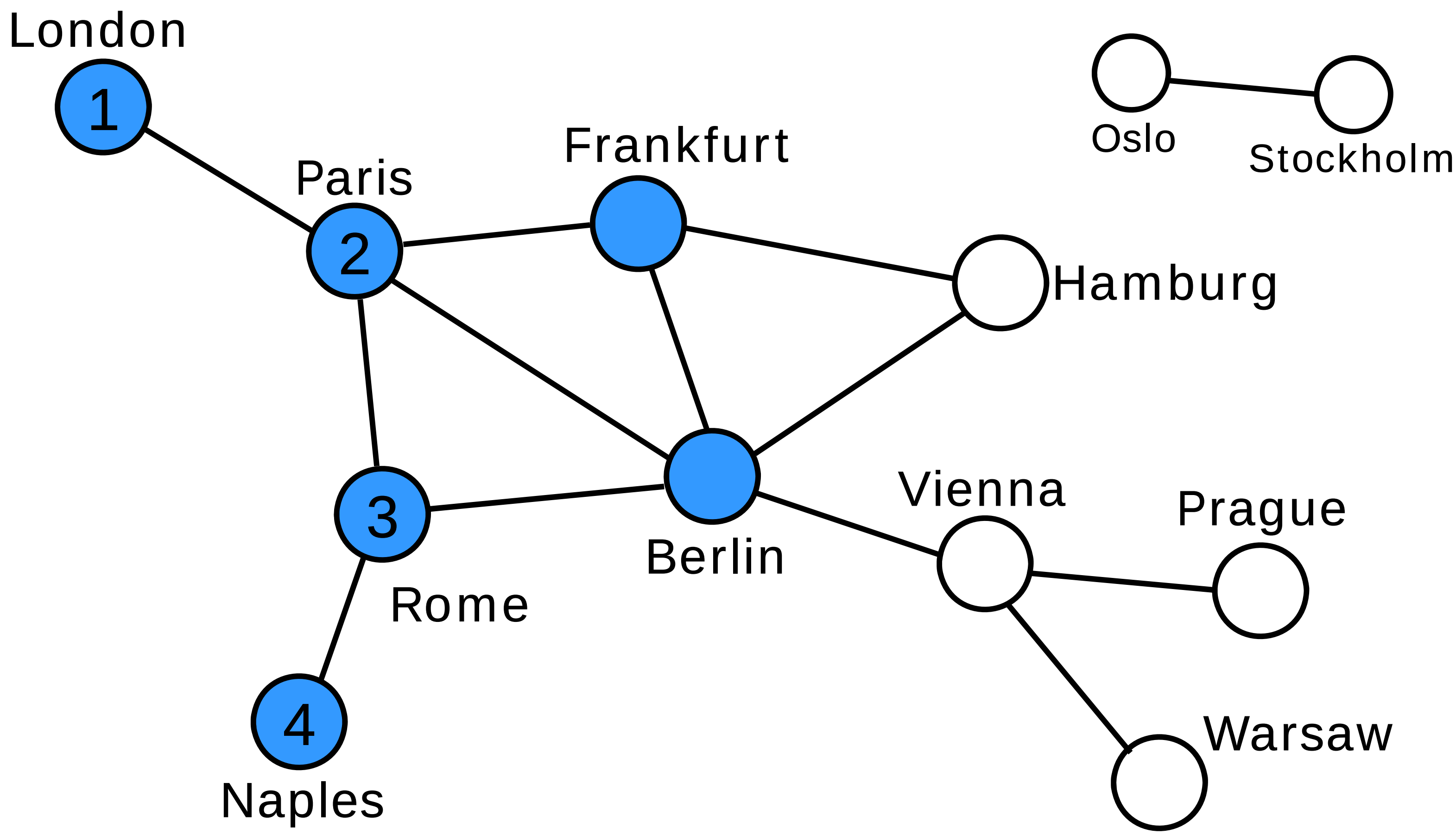
Todo list: [Frankfurt, Berlin]

Graph traversal (stack)



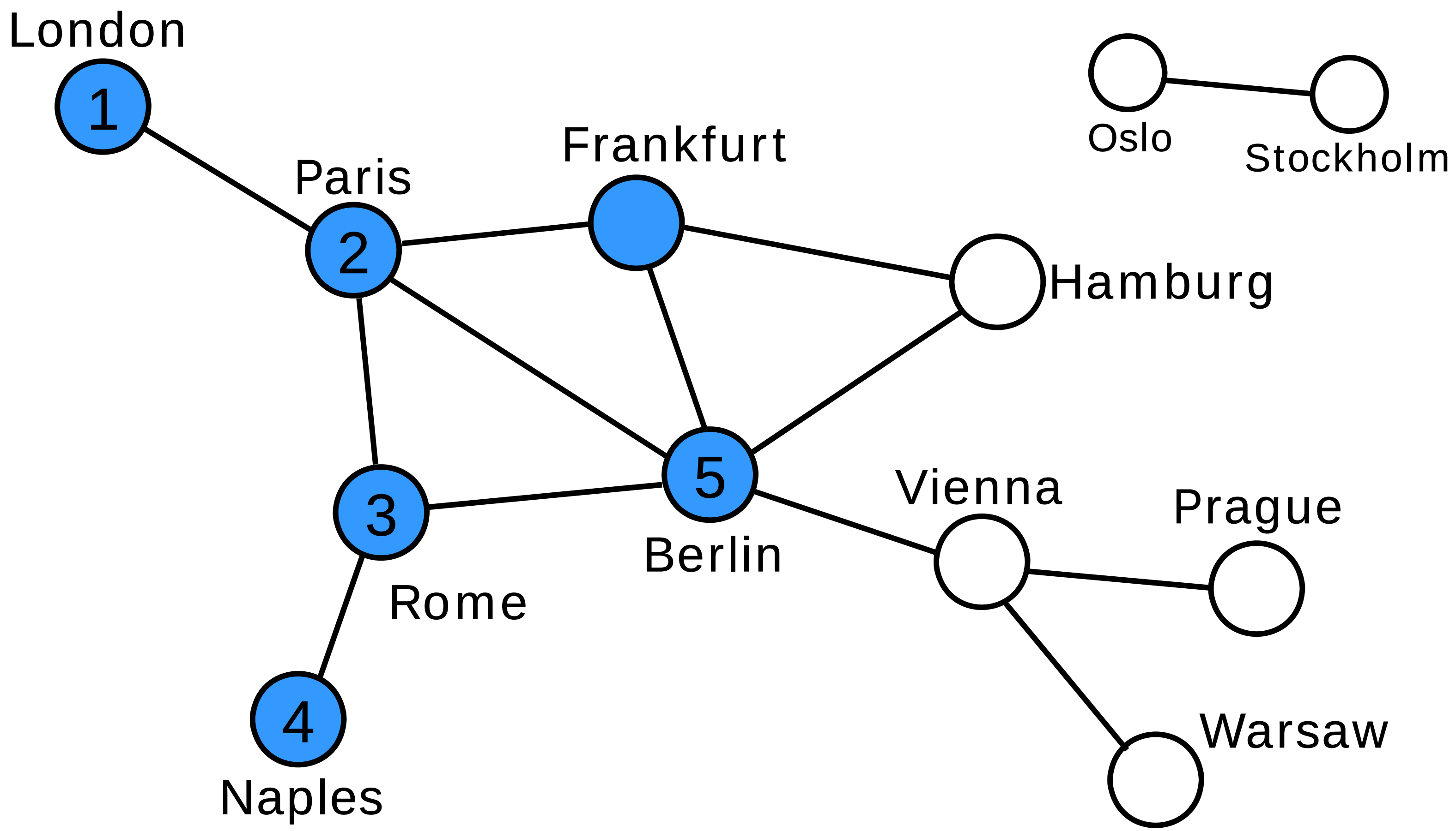
Current node: Rome
Todo list: [Frankfurt, Berlin, Naples]

Graph traversal (stack)



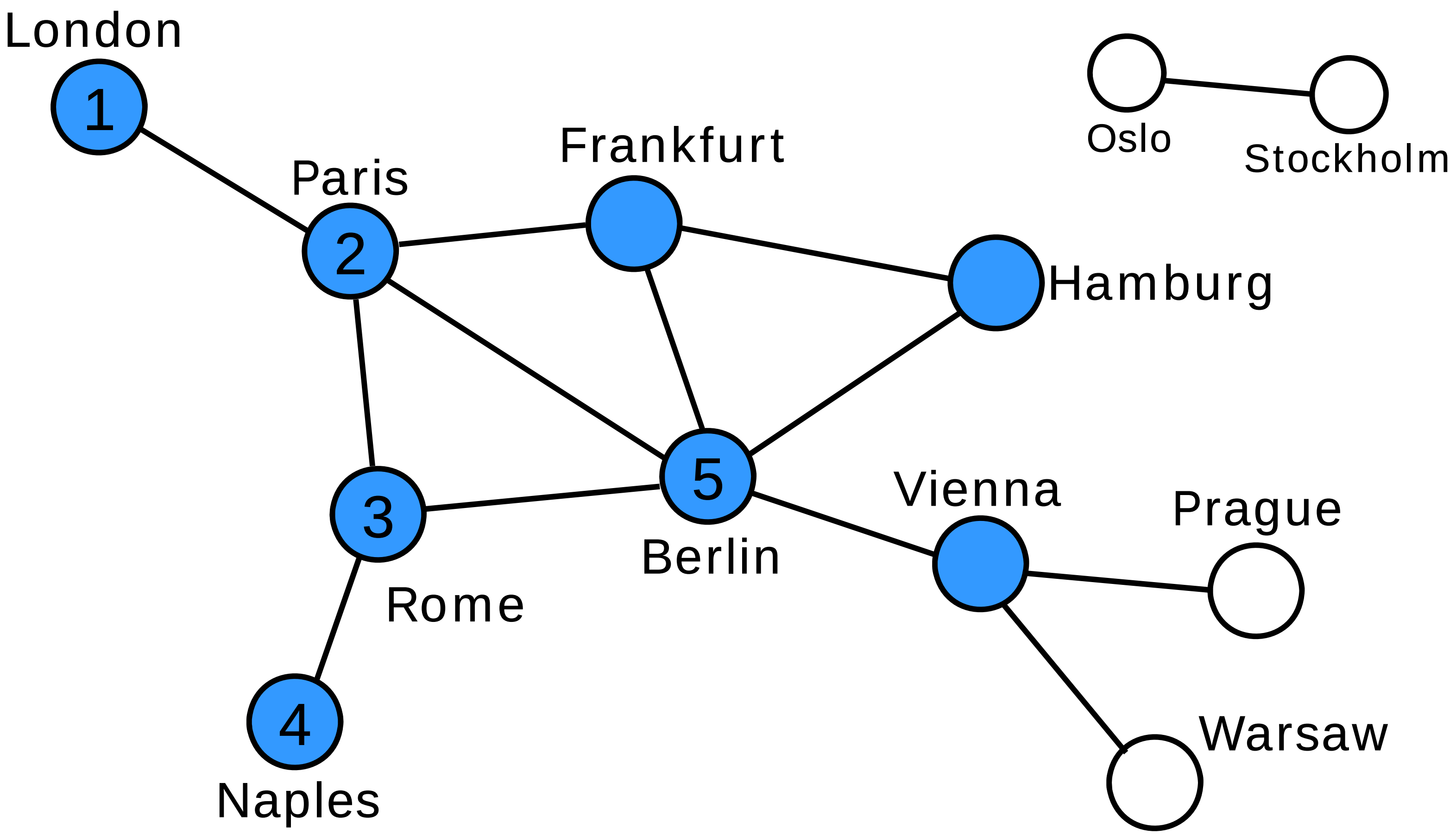
Current node: Naples
Todo list: [Frankfurt, Berlin]

Graph traversal (stack)



Current node: Berlin
Todo list: [Frankfurt]

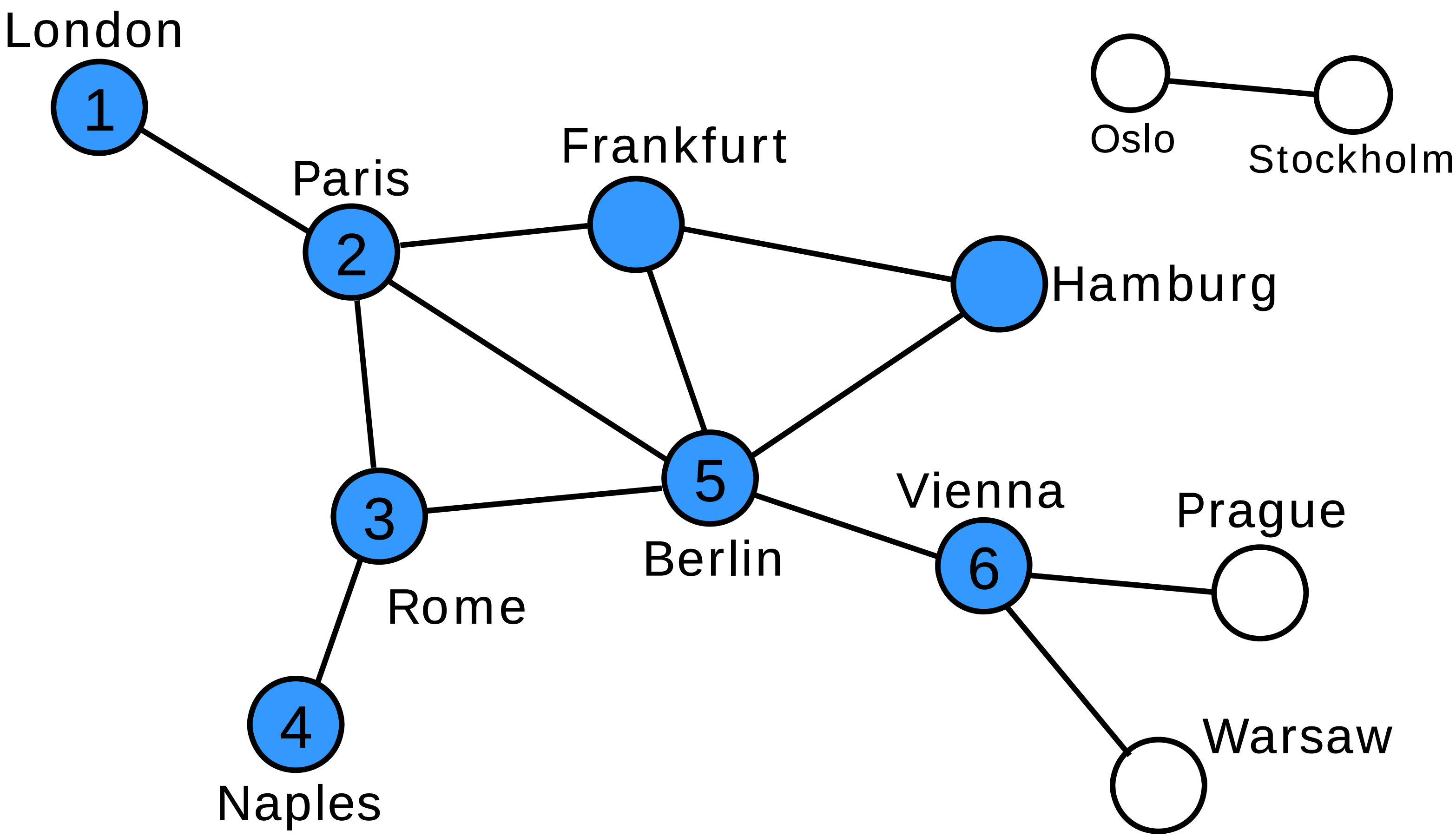
Graph traversal (stack)



Current node: Berlin

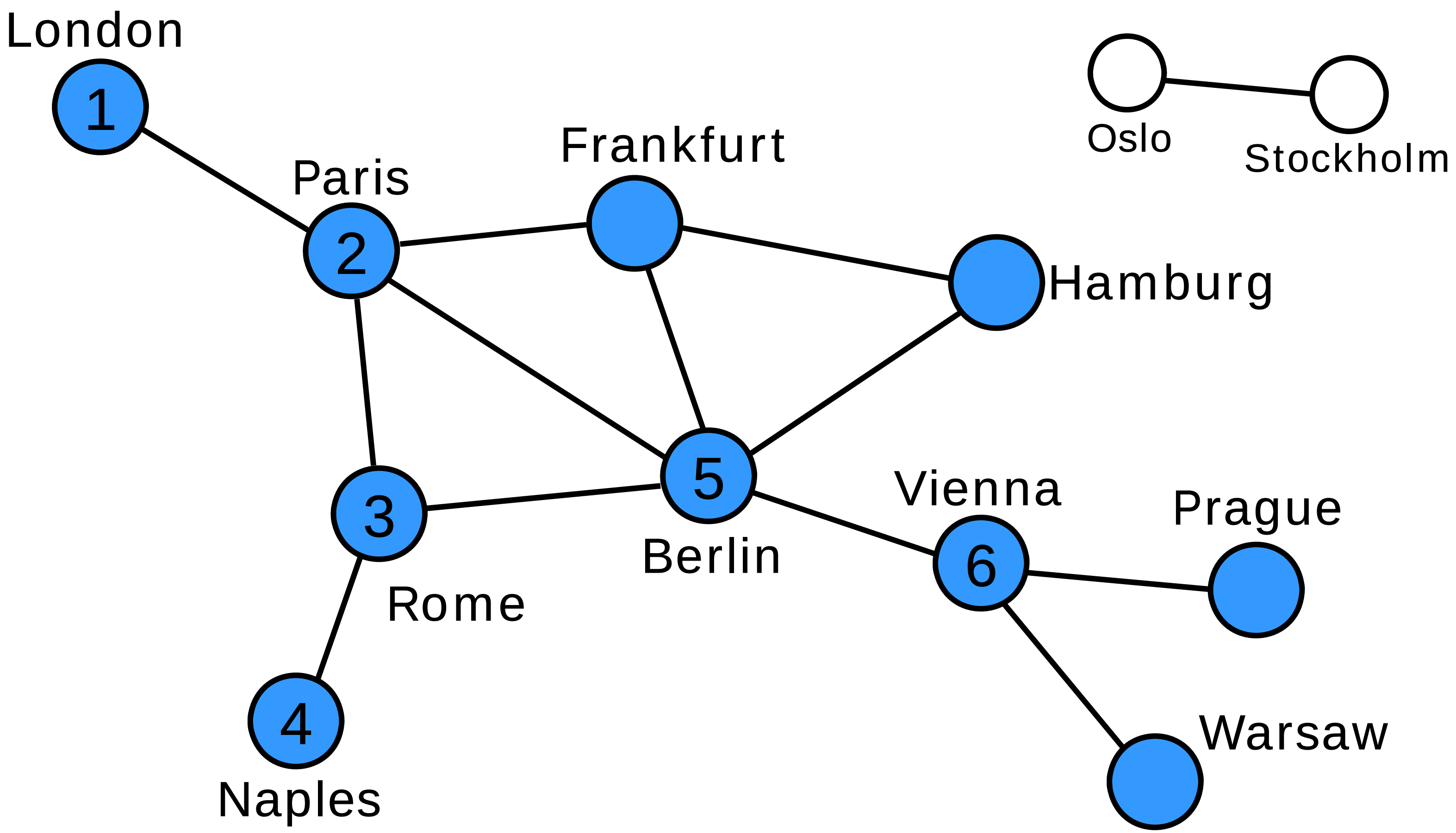
Todo list: [Frankfurt, Hamburg, Vienna]

Graph traversal (stack)



Current node: Vienna
Todo list: [Frankfurt, Hamburg]

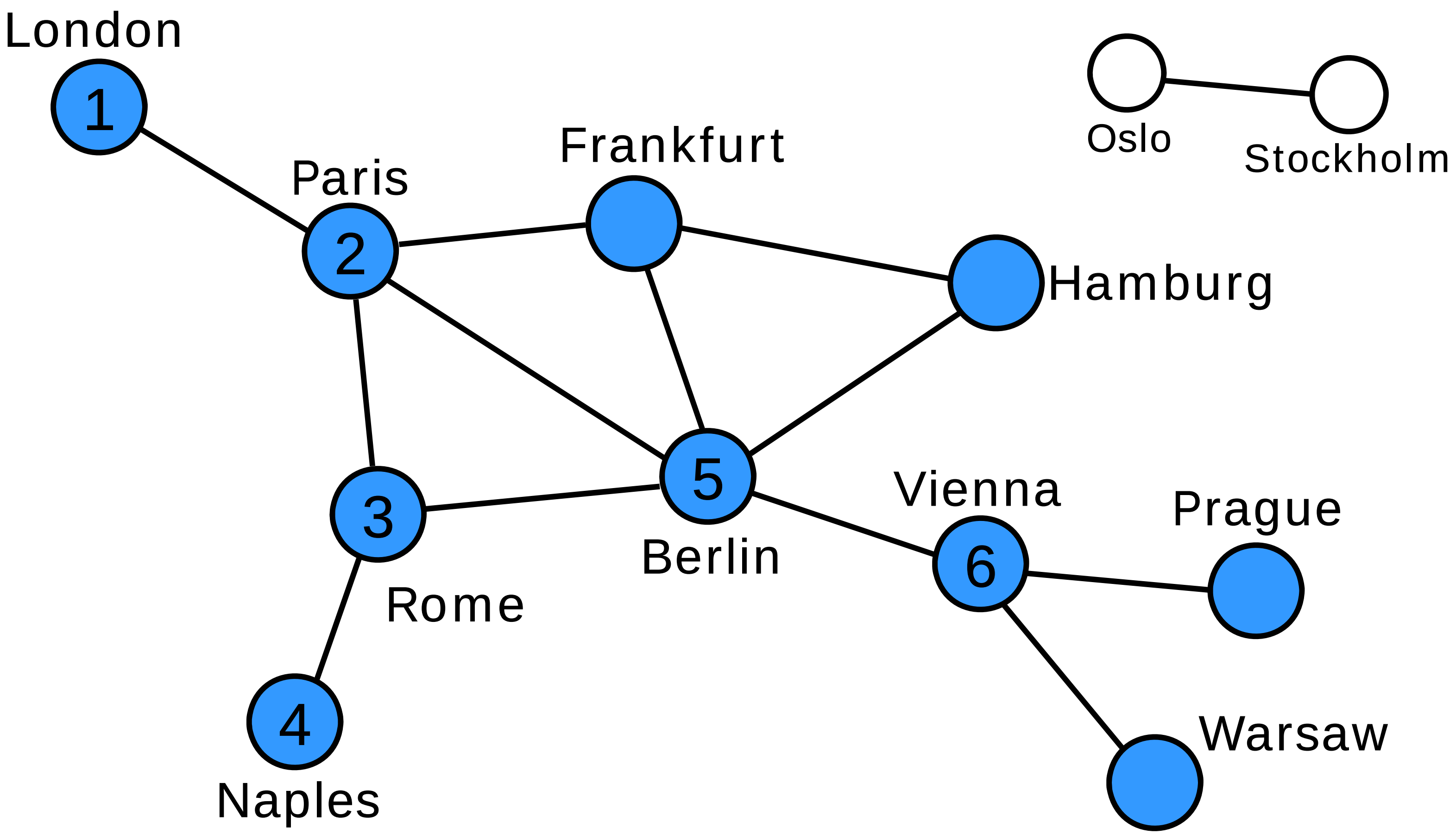
Graph traversal (stack)



Current node: Vienna

Todo list: [Frankfurt, Hamburg, Prague, Warsaw]

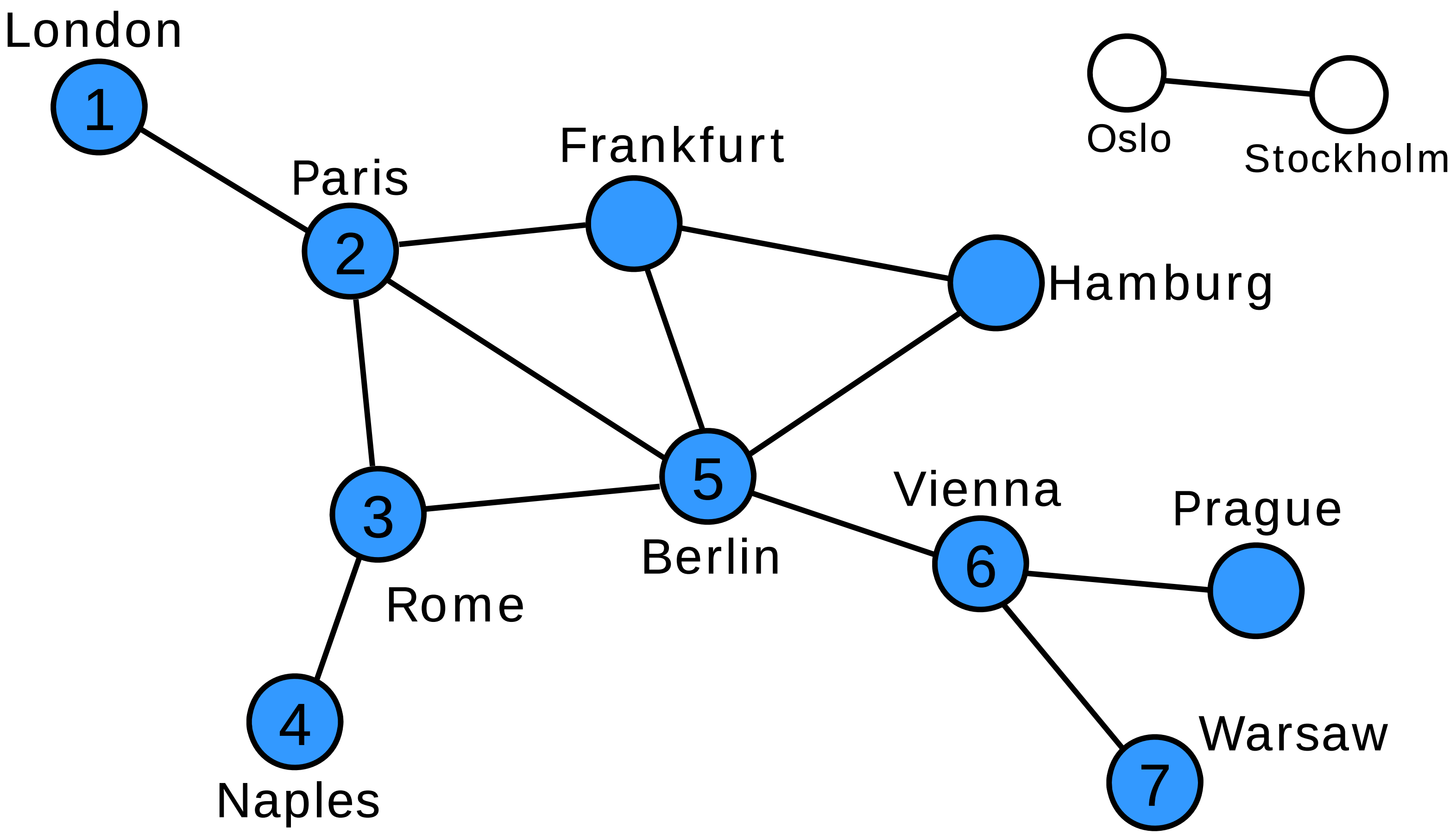
Graph traversal (stack)



Current node: Vienna

Todo list: [Frankfurt, Hamburg, Prague, Warsaw]

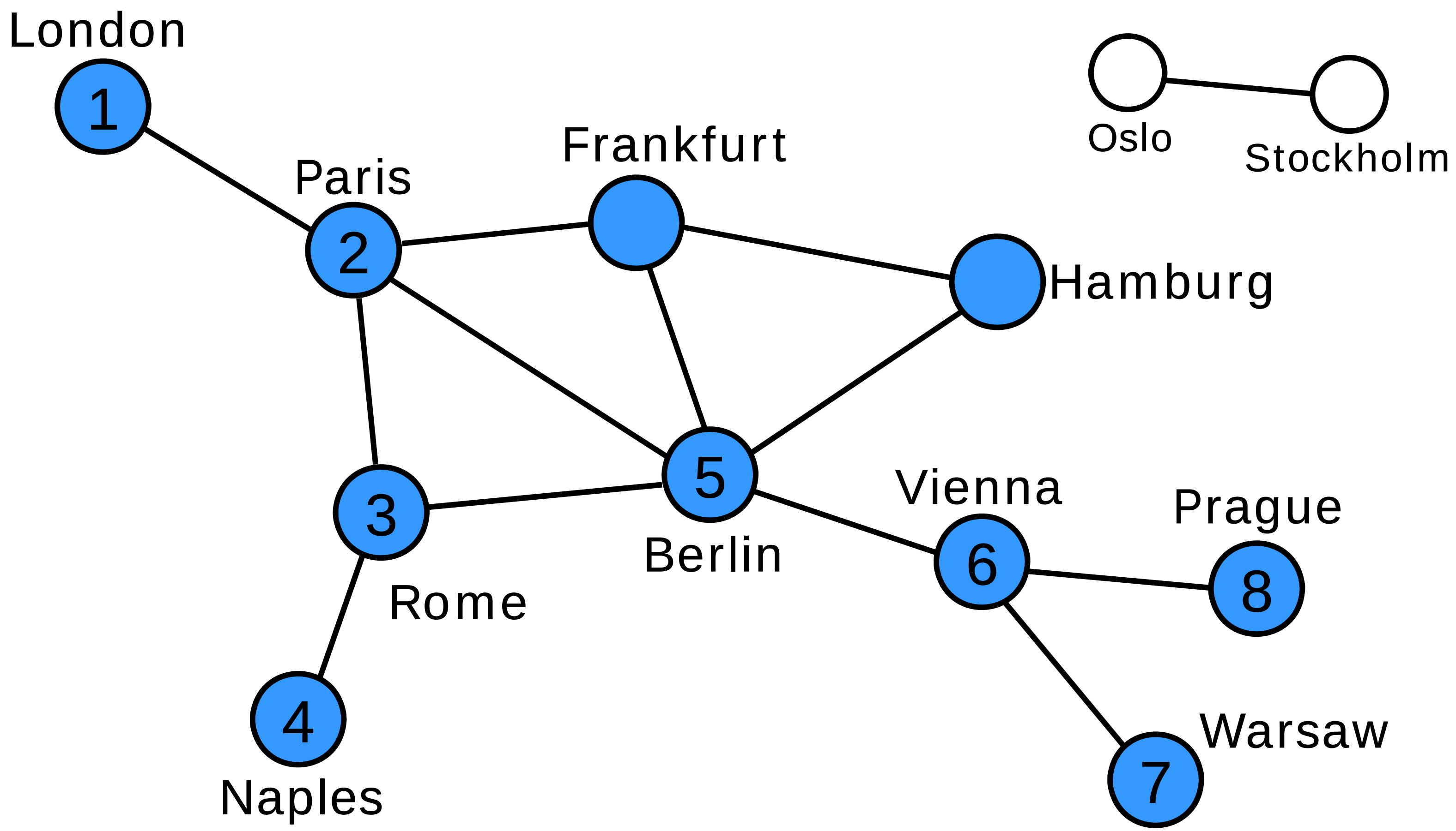
Graph traversal (stack)



Current node: Warsaw

Todo list: [Frankfurt, Hamburg, Prague]

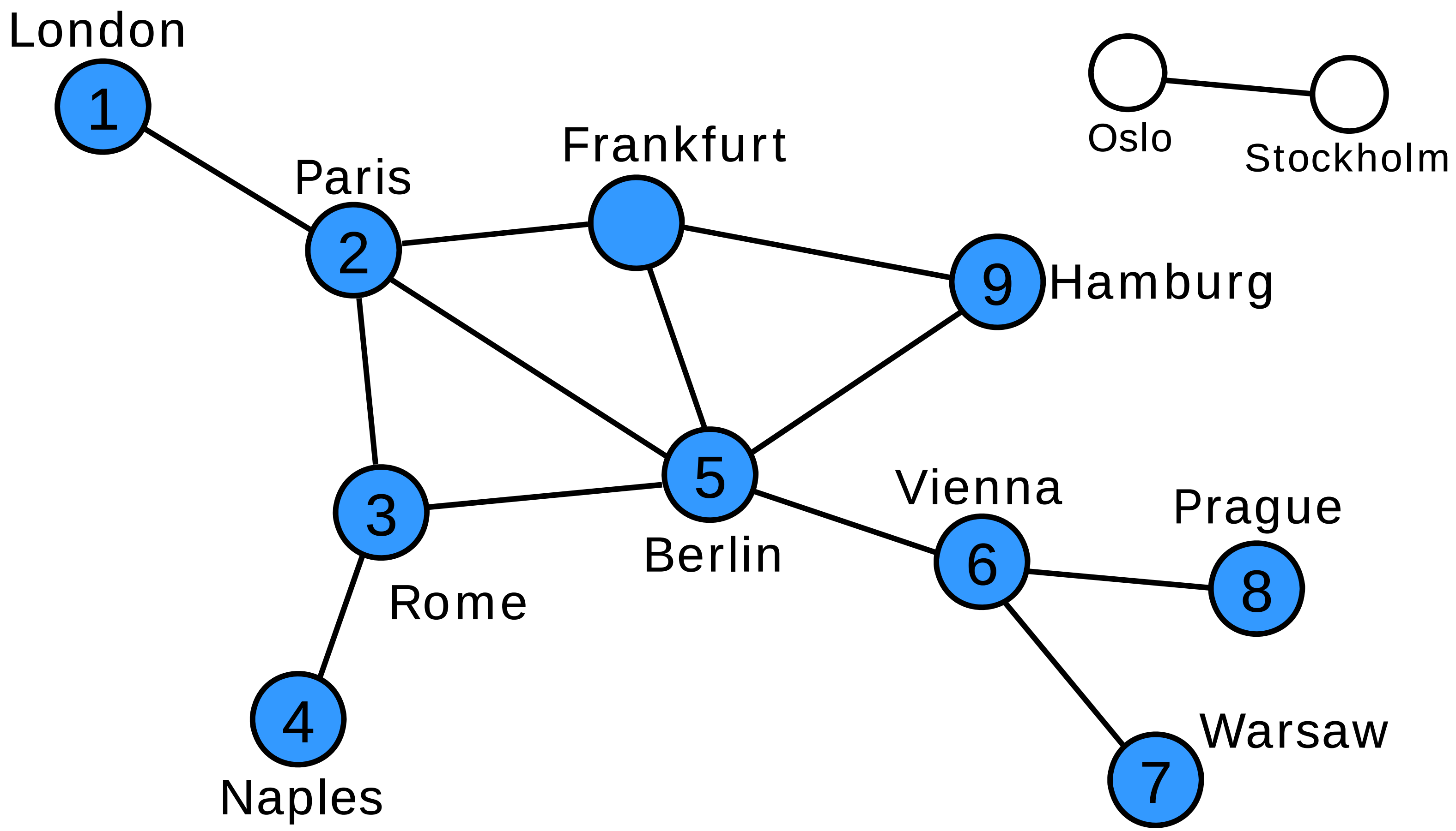
Graph traversal (stack)



Current node: Prague

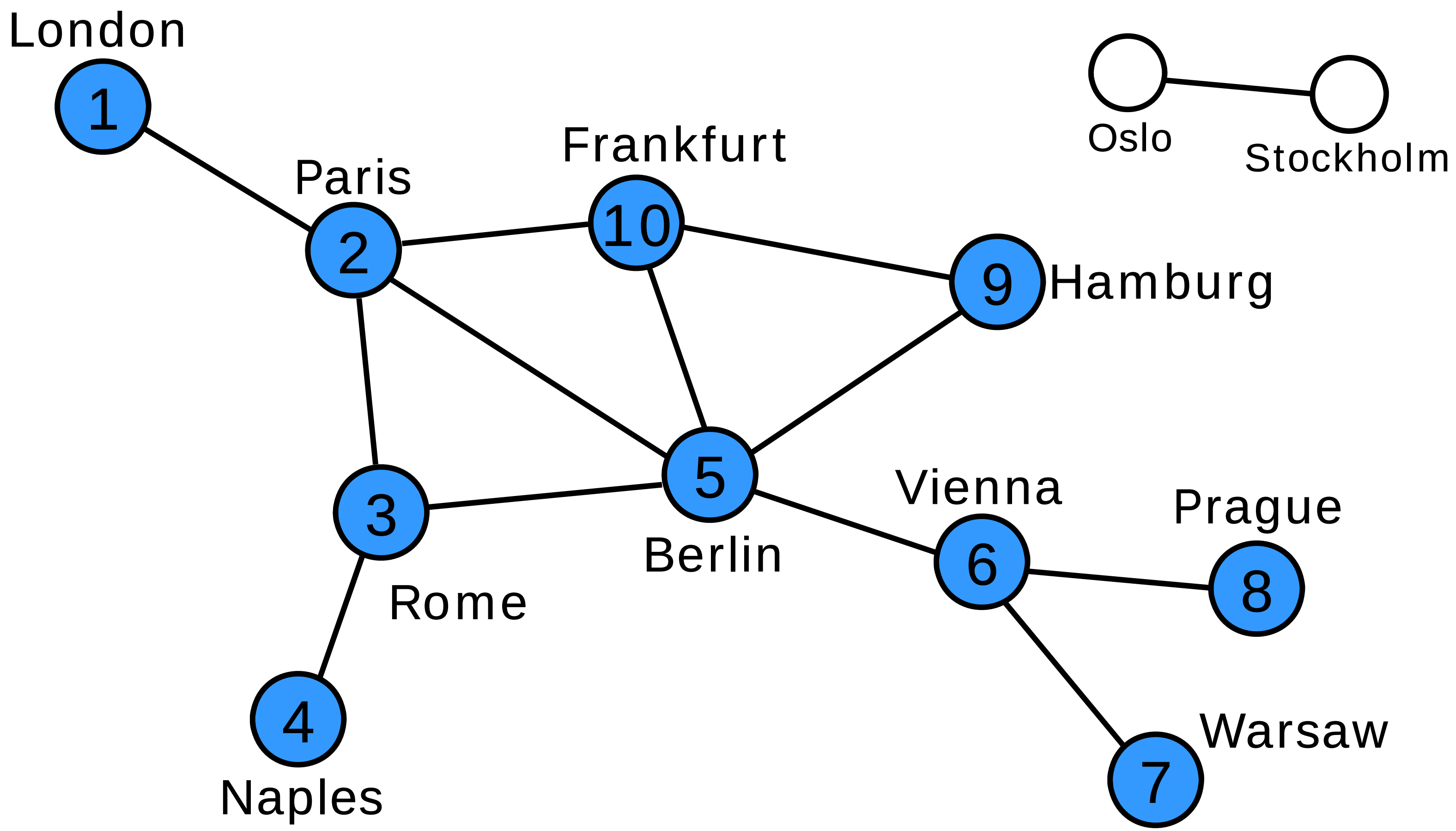
Todo list: [Frankfurt, Hamburg]

Graph traversal (stack)



Current node: Hamburg
Todo list: [Frankfurt]

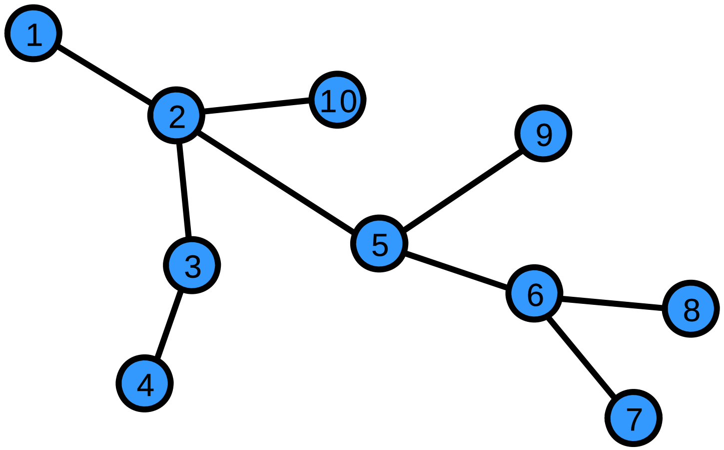
Graph traversal (stack)



Current node: Frankfurt

Todo list: [ ]

Depth-first search (DFS)



- Call the starting node the *root*
- We traverse paths all the way until we get to a dead-end, then backtrack (until we find an unexplored path)

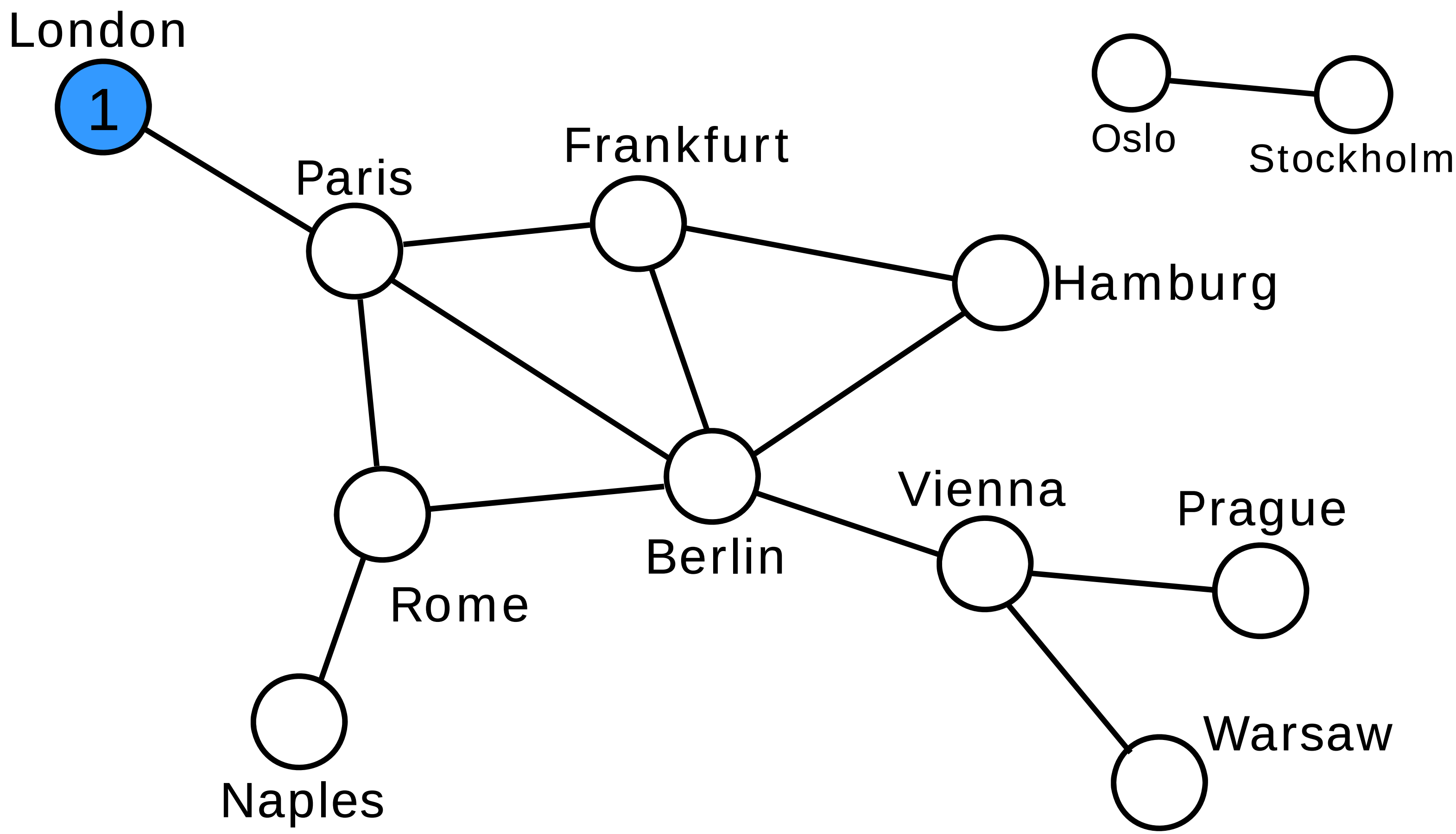
Another strategy

1. Explore all the cities that are one hop away from the root
2. Explore all cities that are two hops away from the root
3. Explore all cities that are three hops away from the root

   ...

- This corresponds to using a *queue*

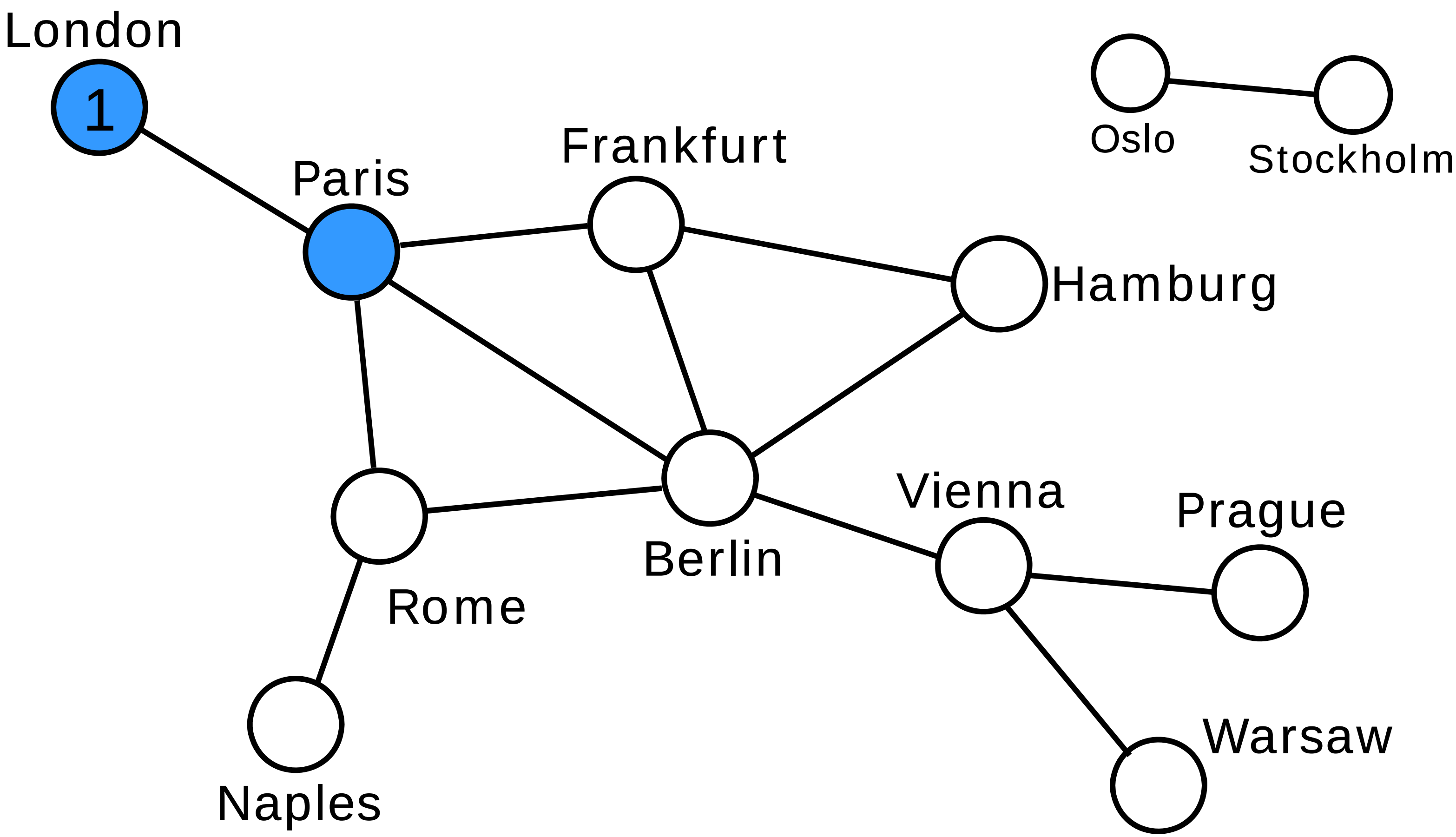
Graph traversal (queue)



Current node: London

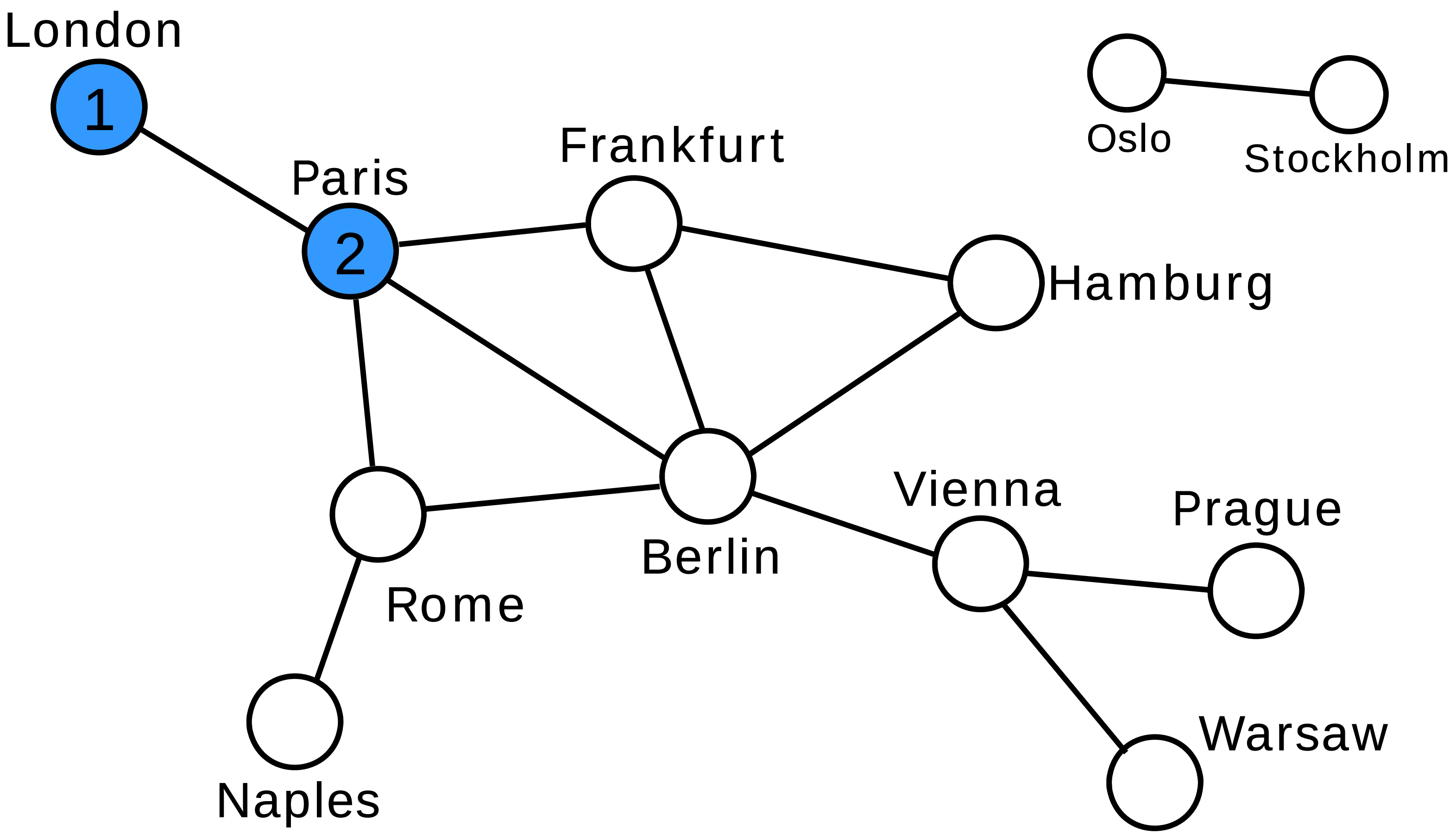
Todo list: [ ]

Graph traversal (queue)



Current node: London
Todo list: [Paris]

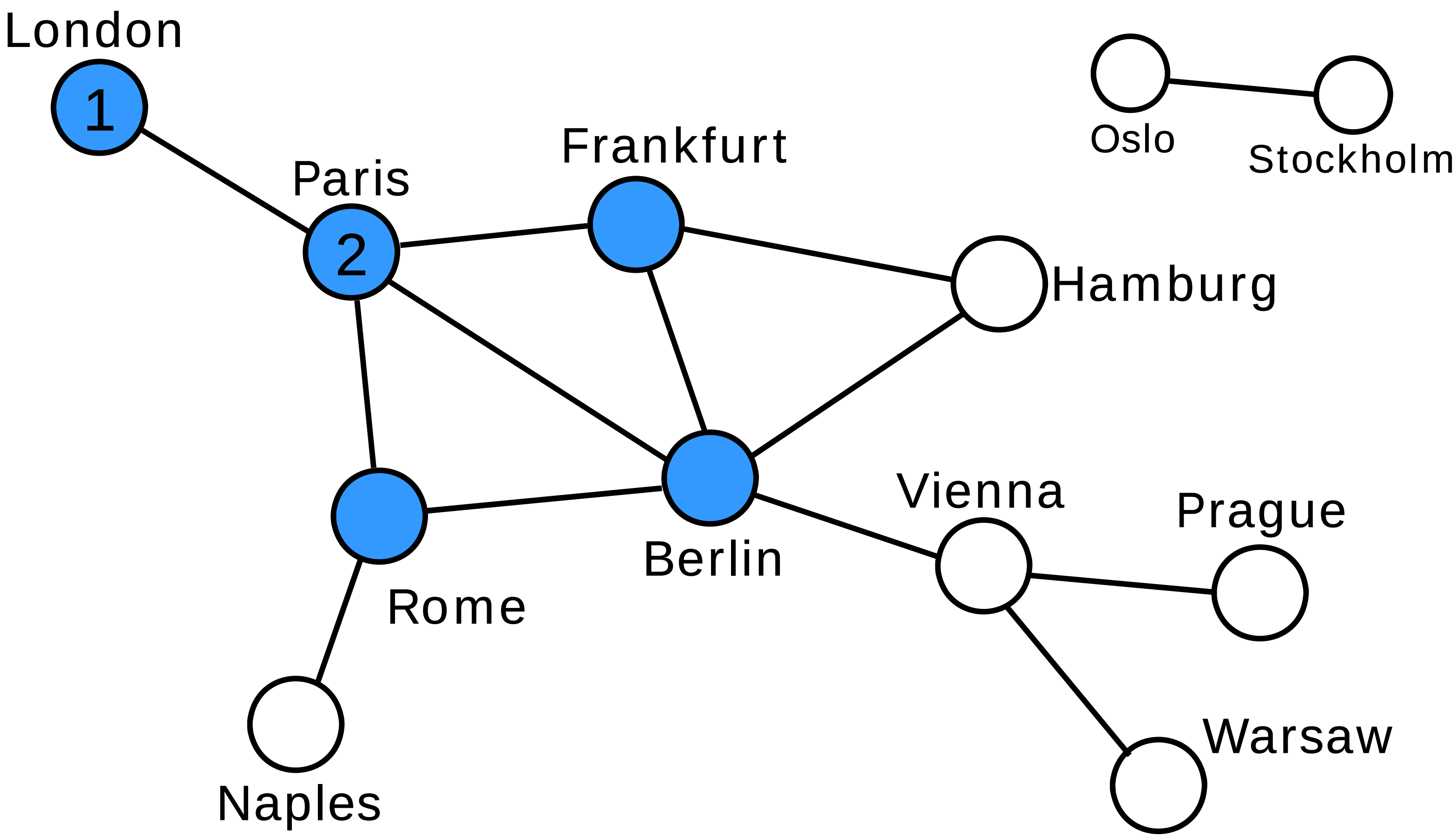
Graph traversal (queue)



Current node: Paris

Todo list: [ ]

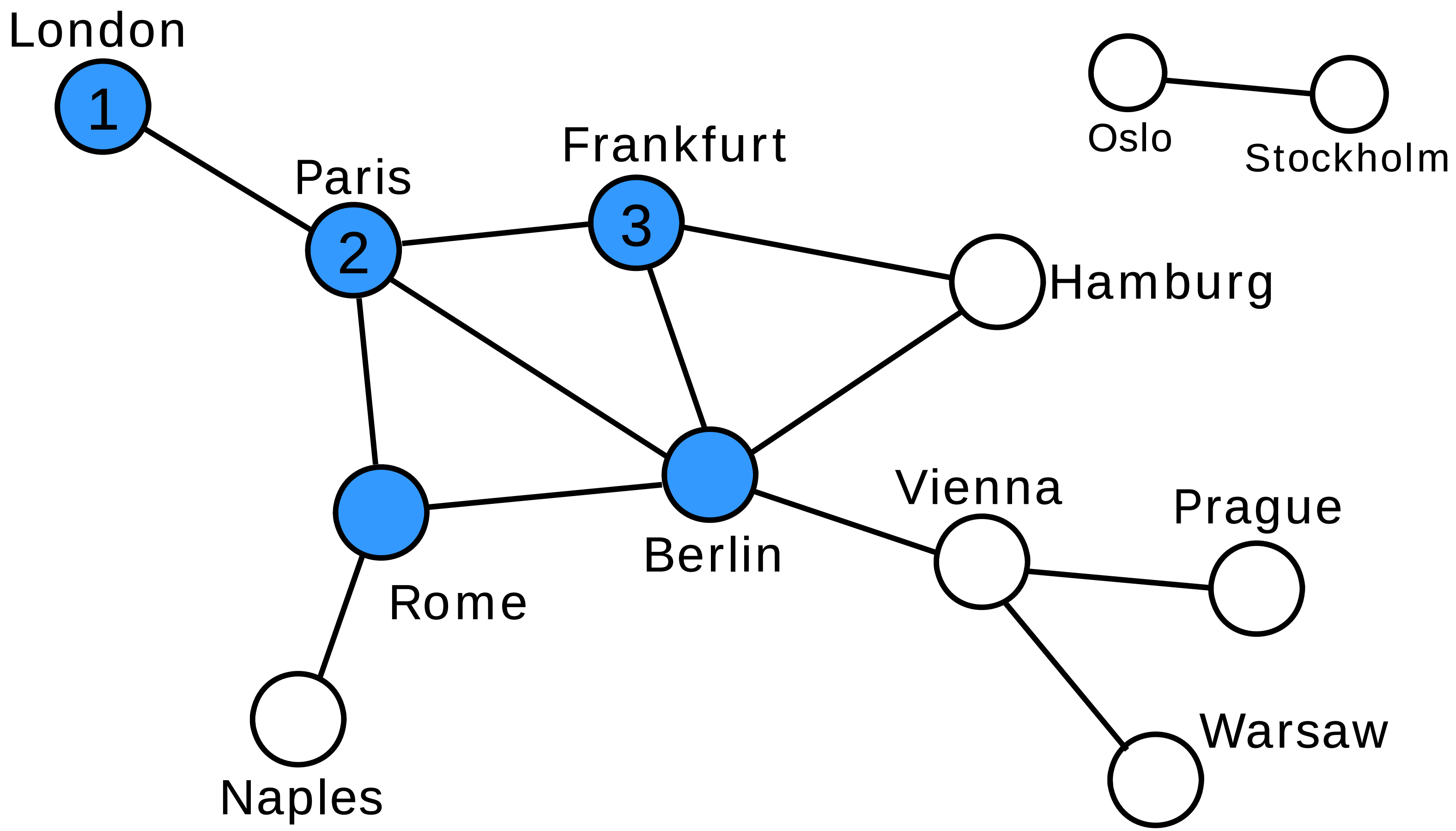
Graph traversal (queue)



Current node: Paris

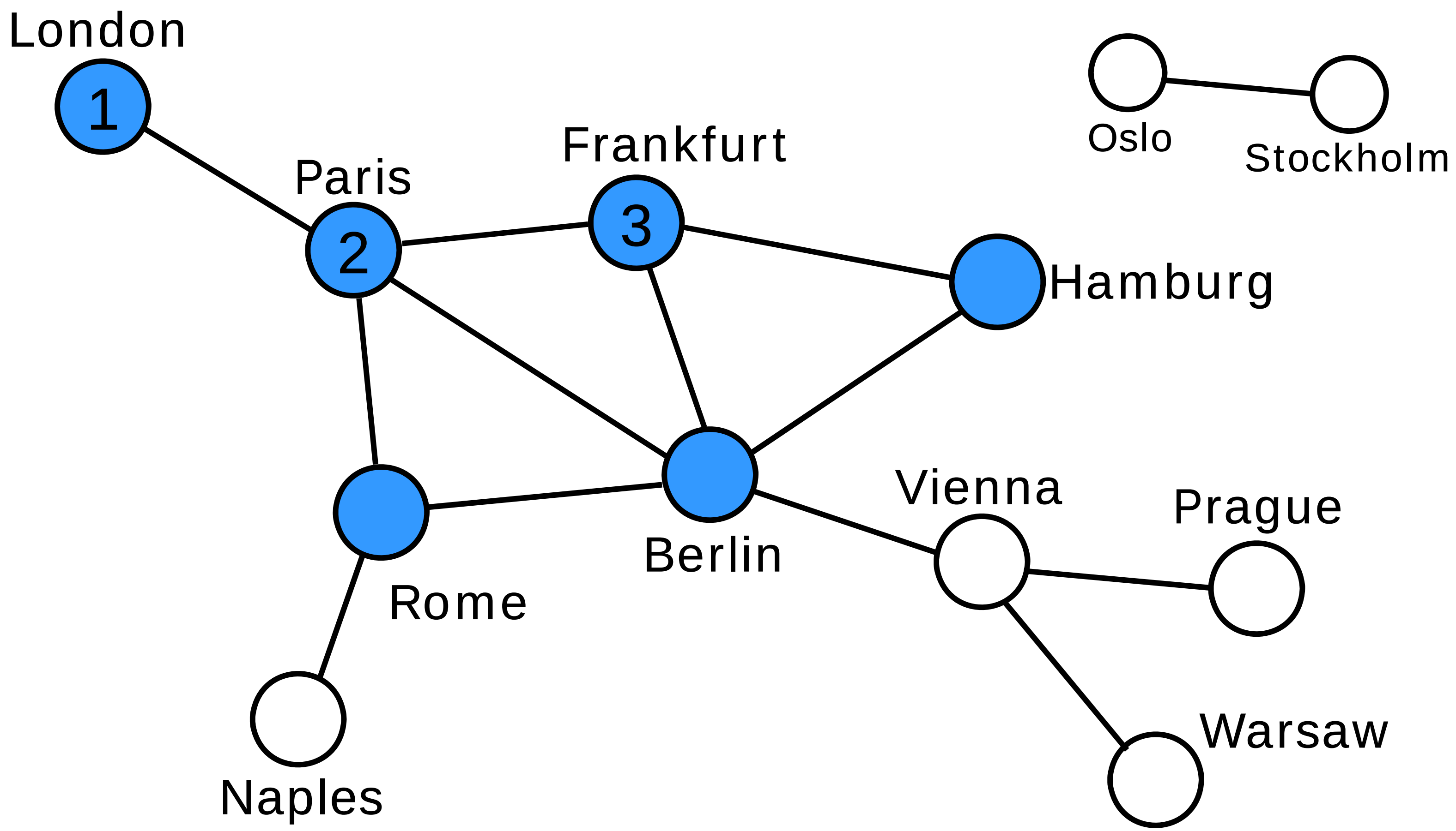
Todo list: [Frankfurt, Berlin, Rome]

Graph traversal (queue)



Current node: Frankfurt
Todo list: [Berlin, Rome]

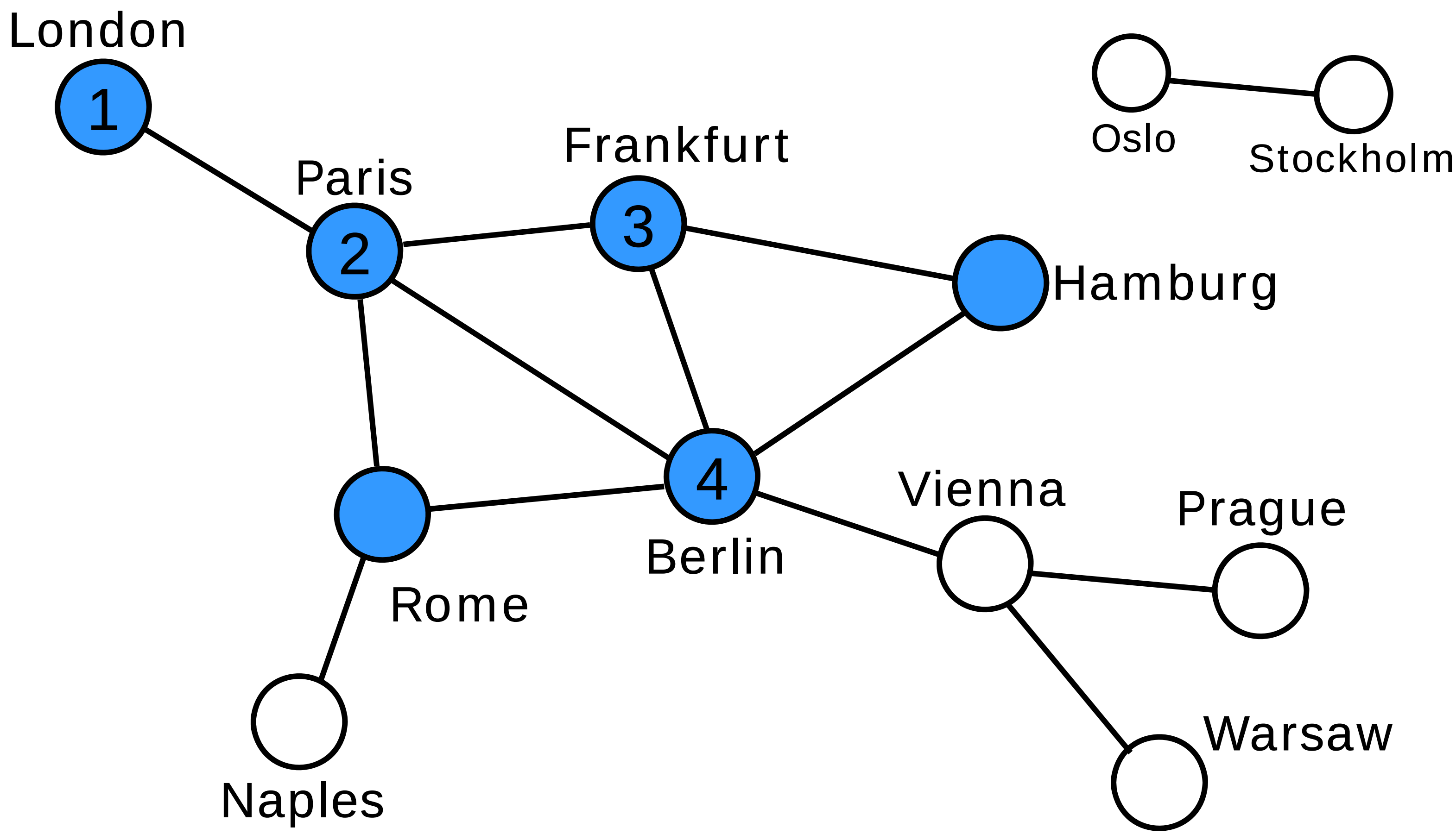
Graph traversal (queue)



Current node: Frankfurt

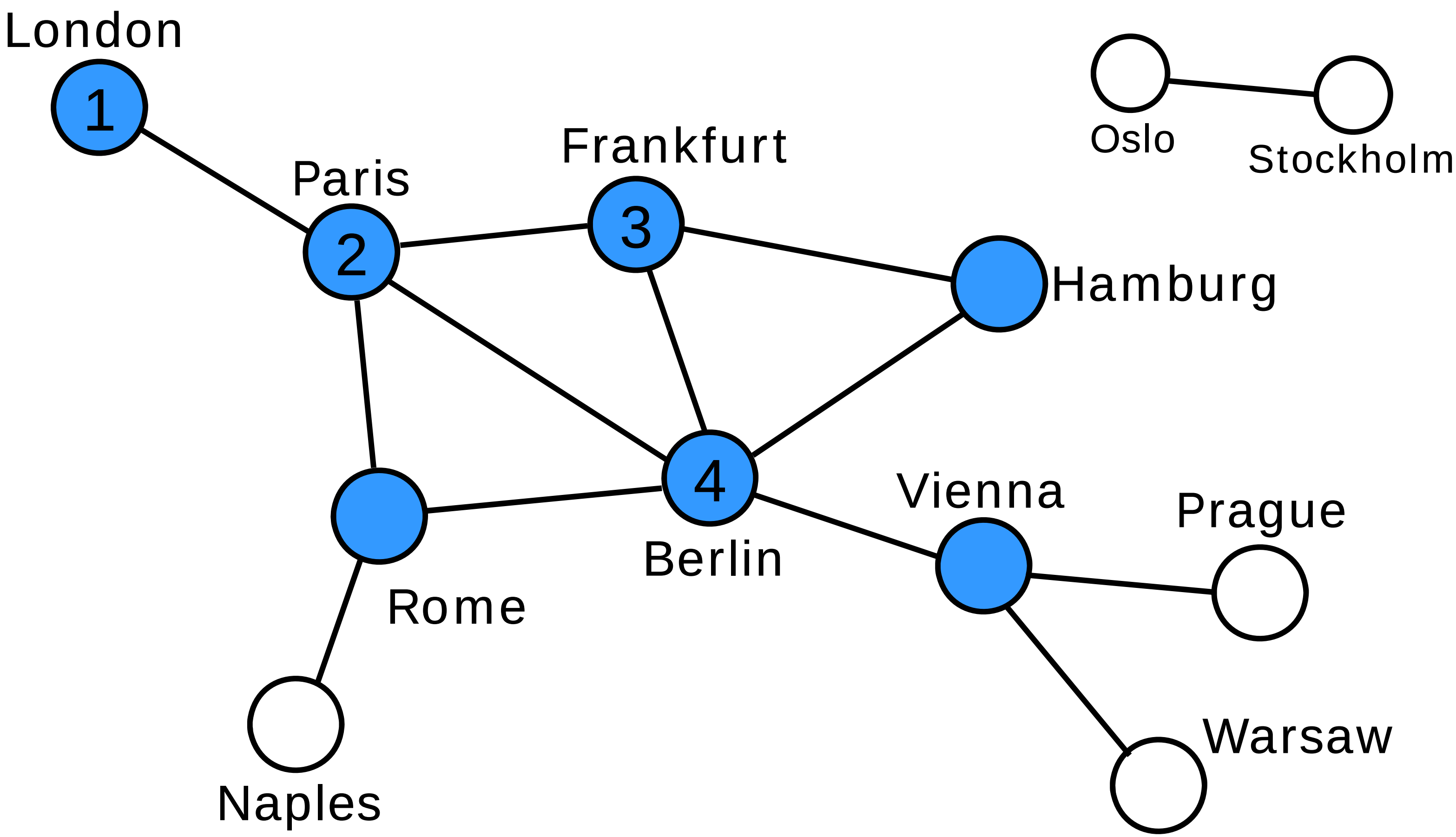
Todo list: [Berlin, Rome, Hamburg]

Graph traversal (queue)



Current node: Berlin
Todo list: [Rome, Hamburg]

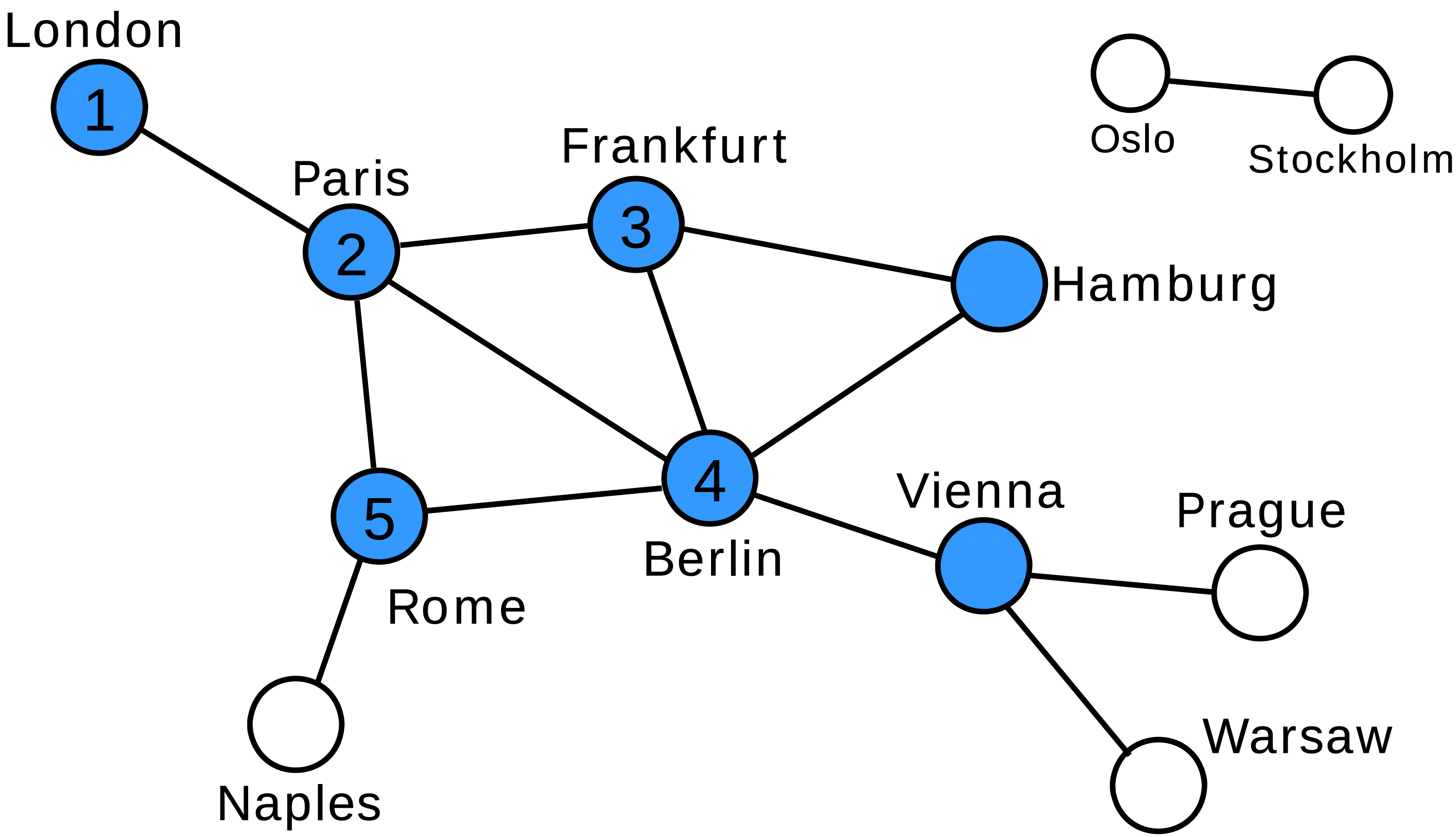
Graph traversal (queue)



Current node: Berlin

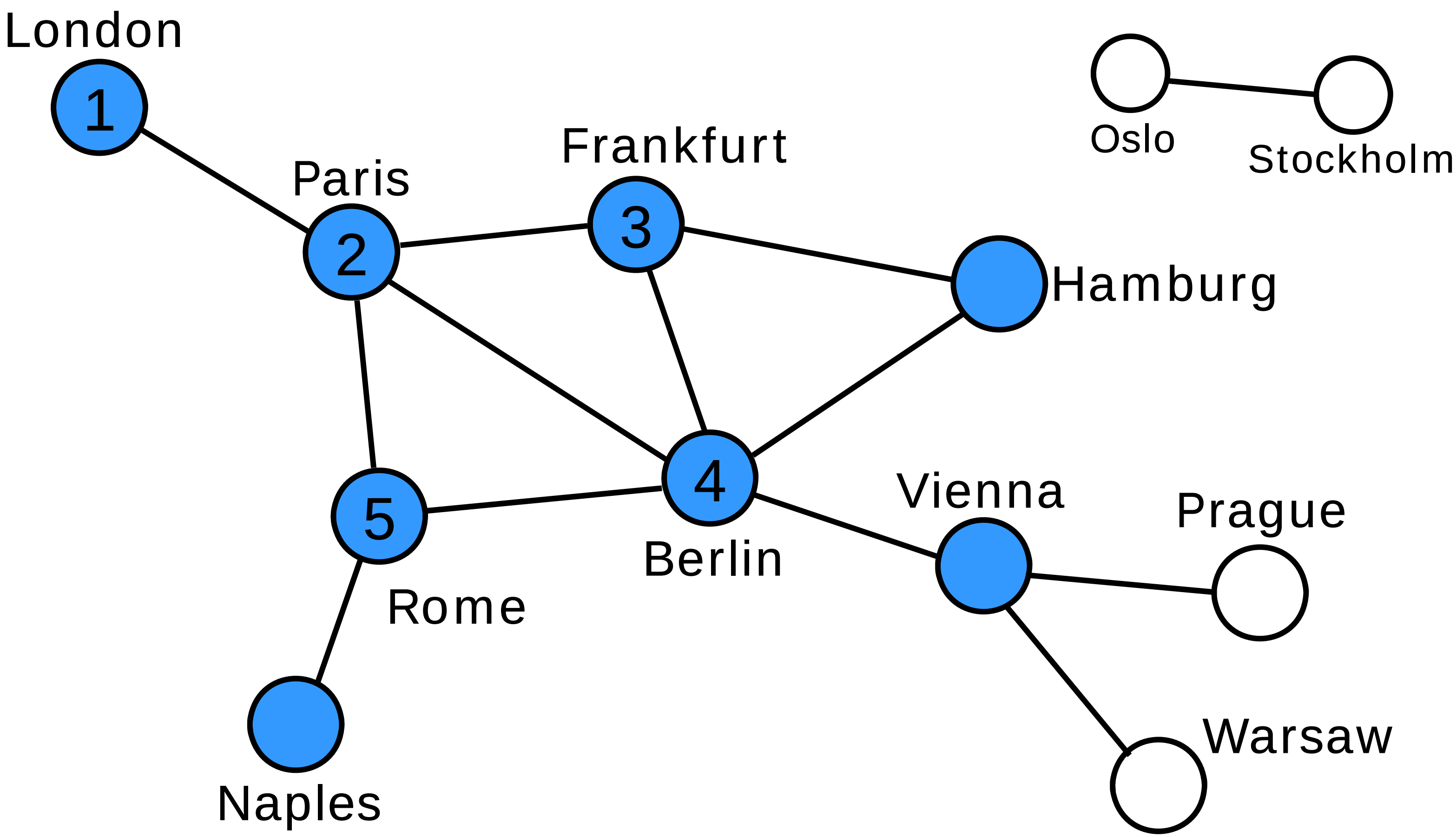
Todo list: [Rome, Hamburg, Vienna]

Graph traversal (queue)



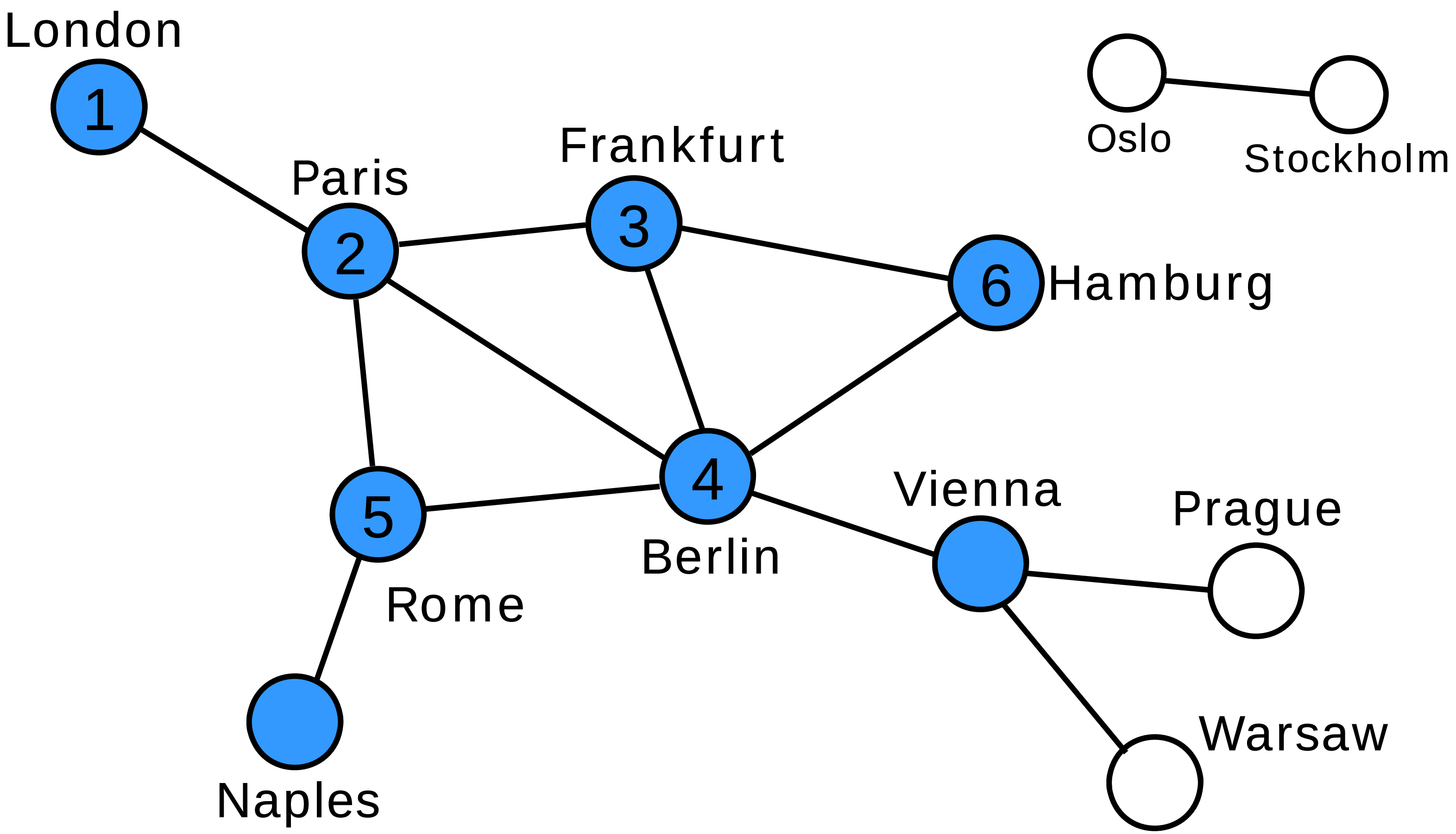
Current node: Rome
Todo list: [Hamburg, Vienna]

Graph traversal (queue)



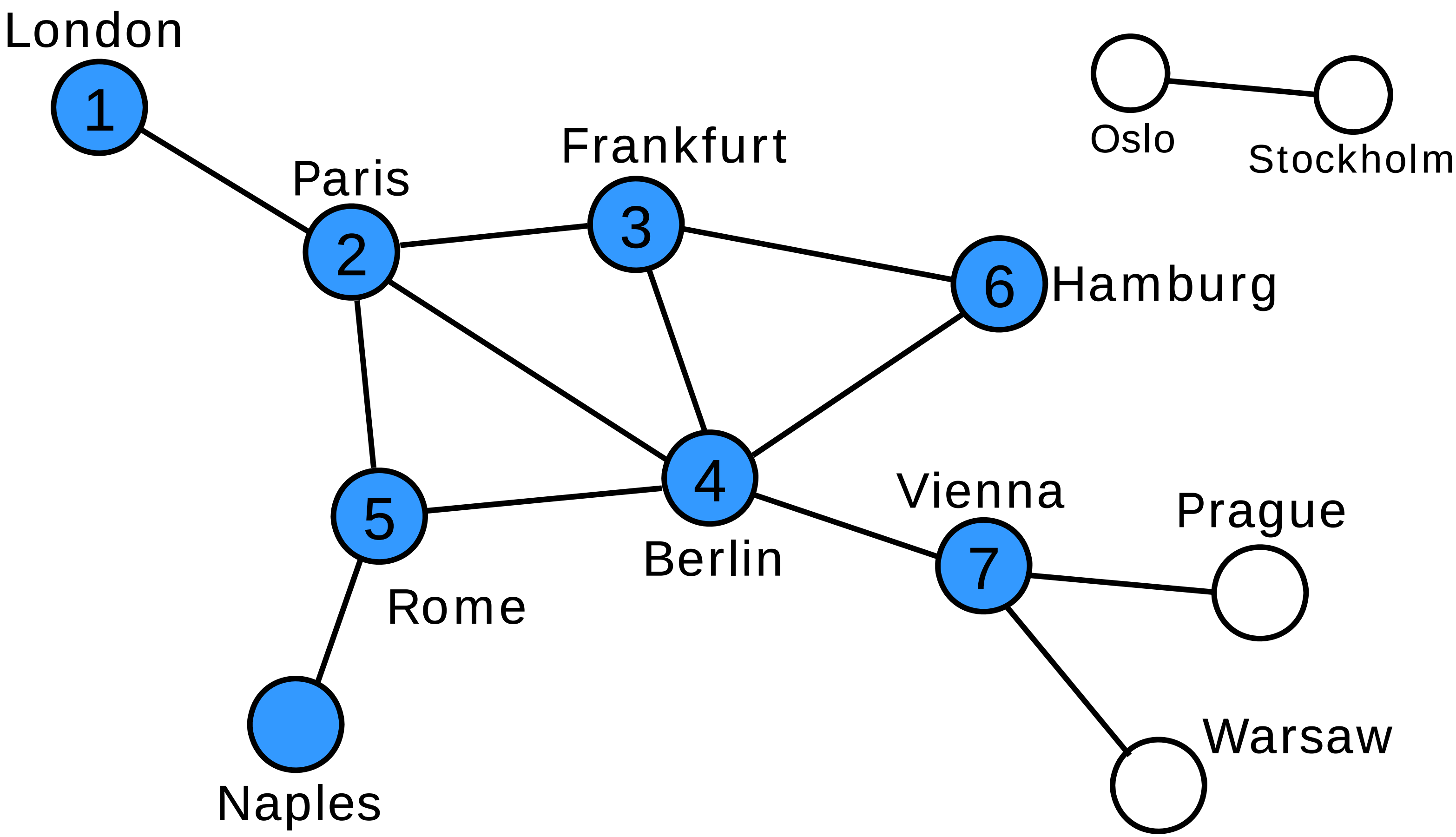
Current node: Rome
Todo list: [Hamburg, Vienna, Naples]

Graph traversal (queue)



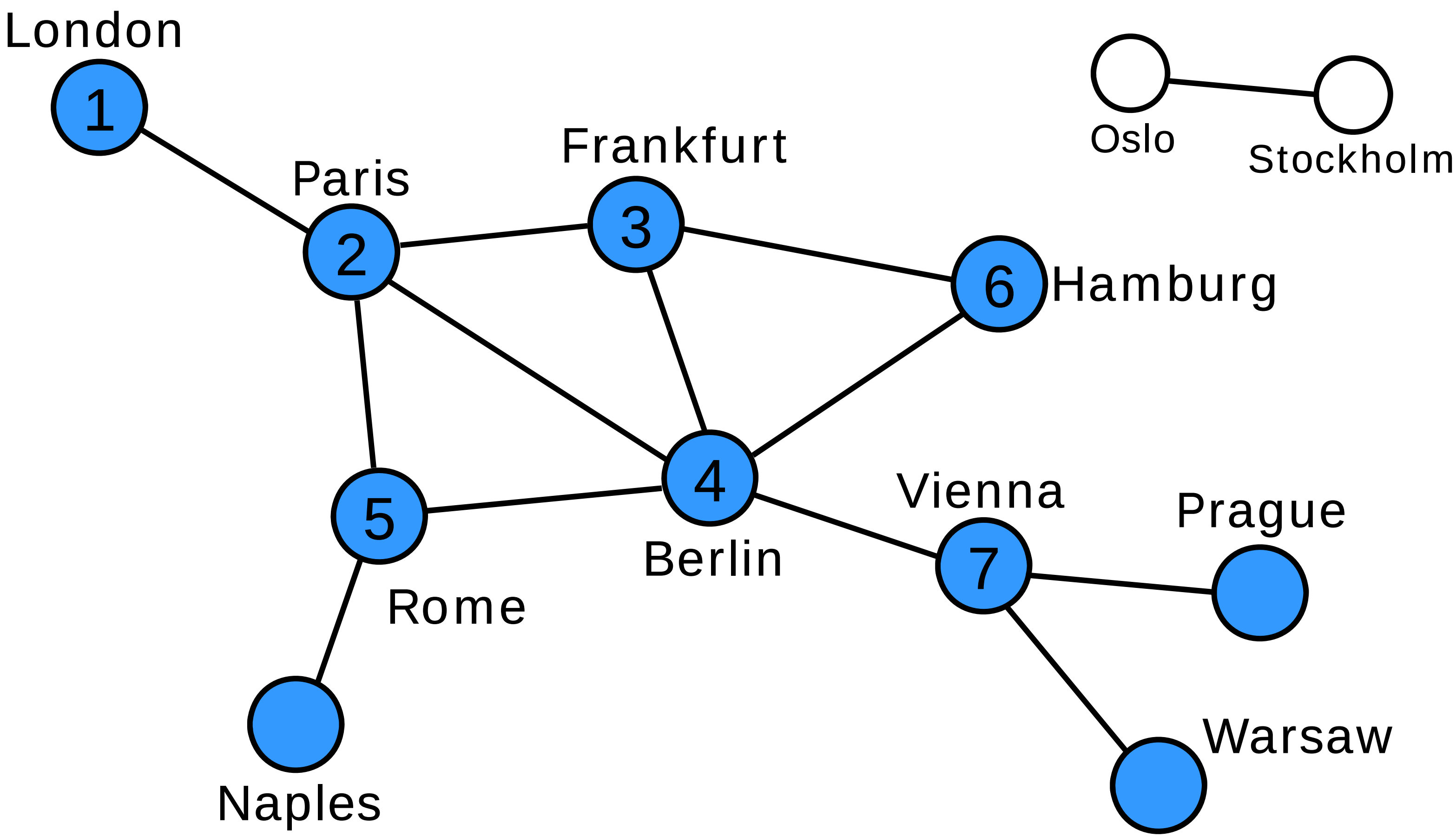
Current node: Hamburg
Todo list: [Vienna, Naples]

Graph traversal (queue)



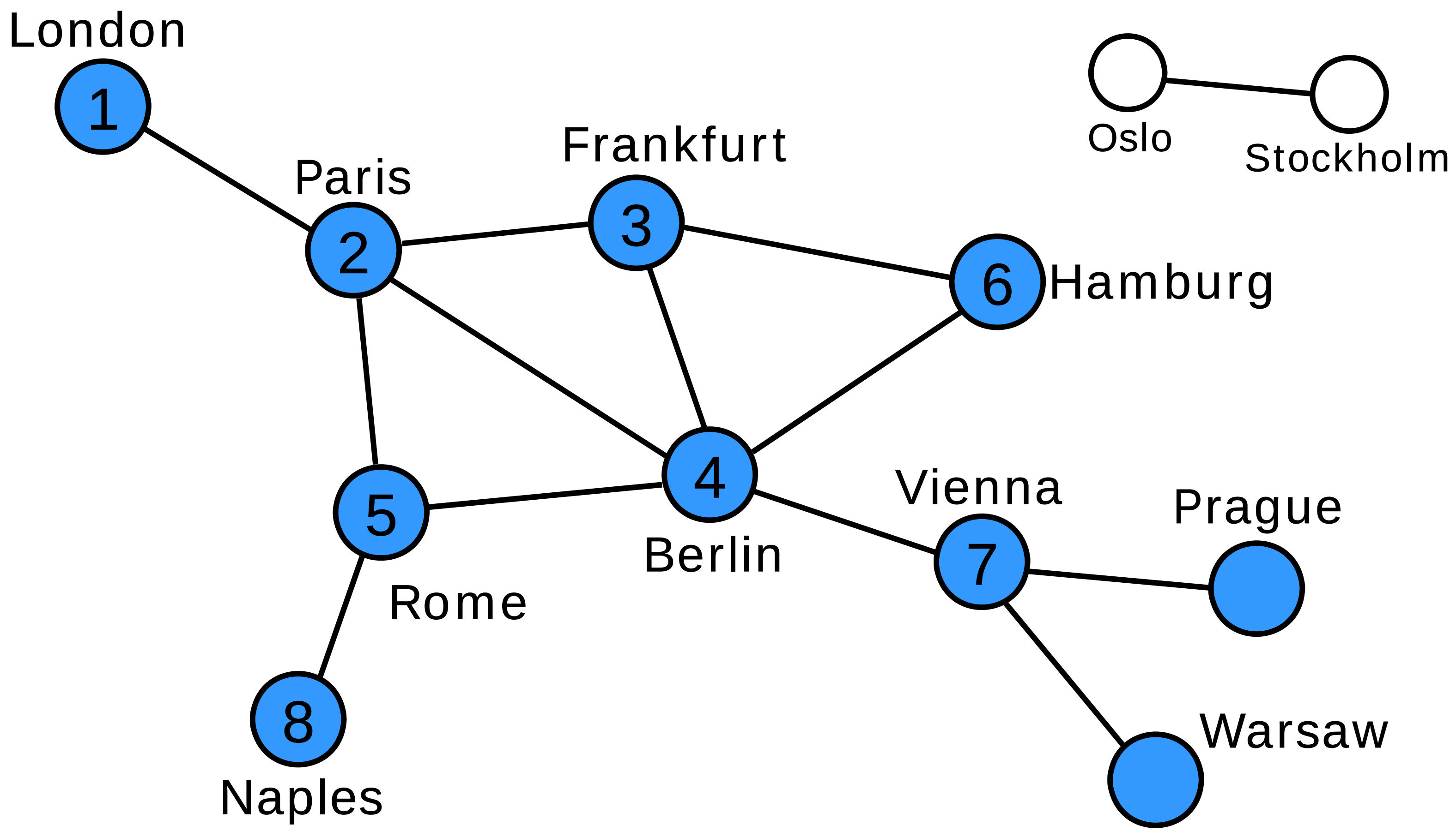
Current node: Vienna
Todo list: [Naples]

Graph traversal (queue)



Current node: Vienna
Todo list: [Naples, Prague, Warsaw]

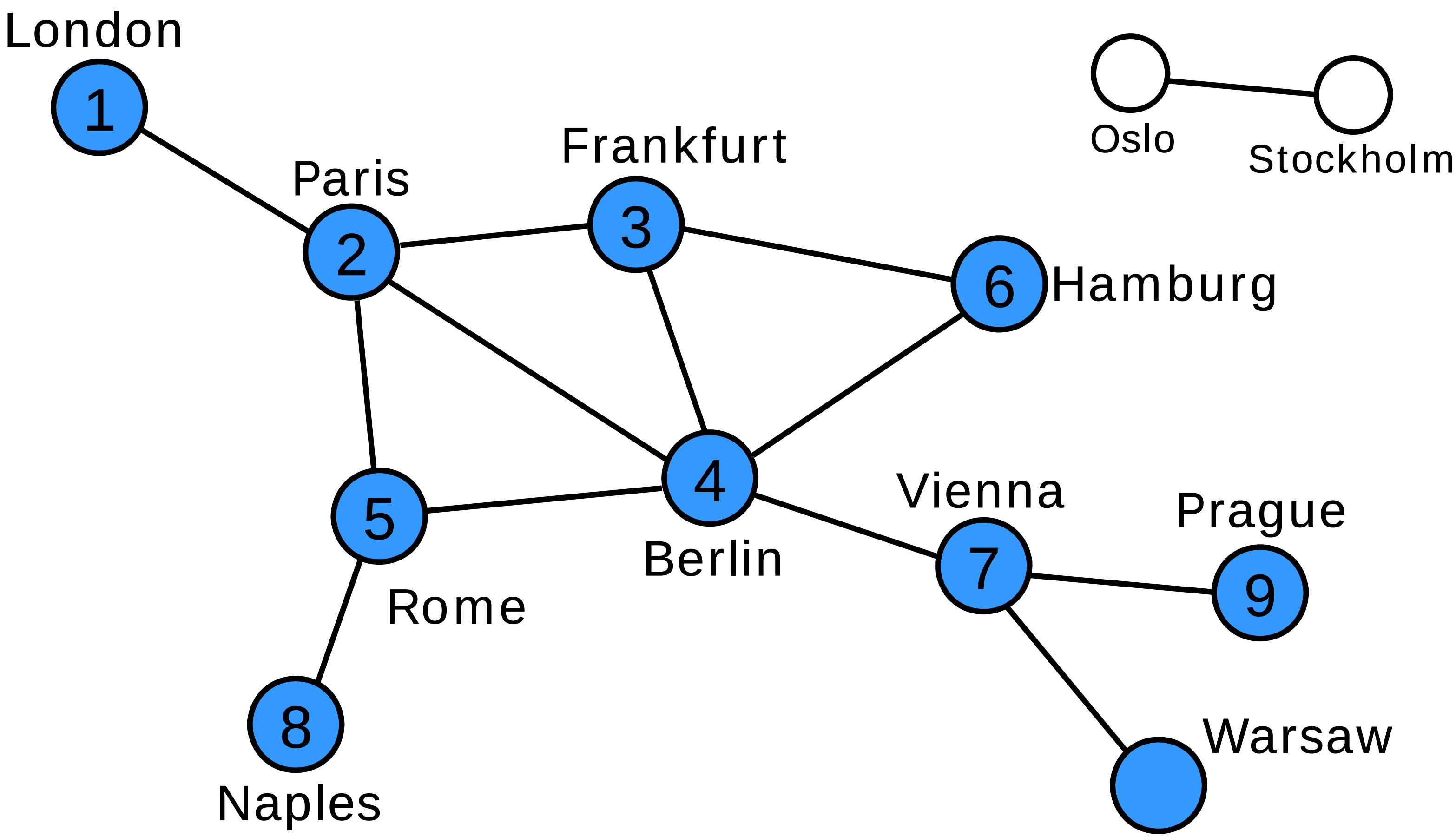
Graph traversal (queue)



Current node: Naples

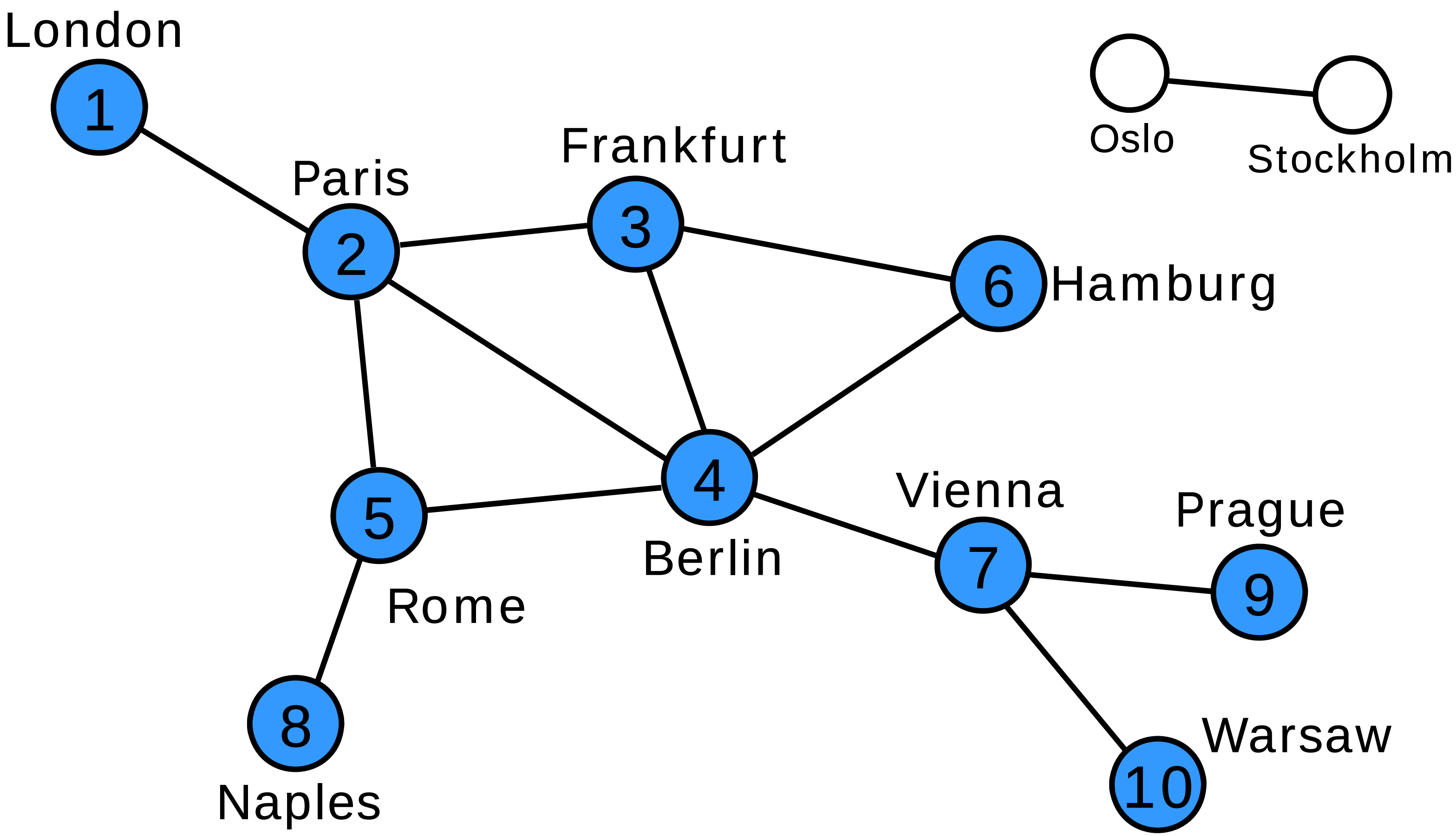
Todo list: [Prague, Warsaw]

Graph traversal (queue)



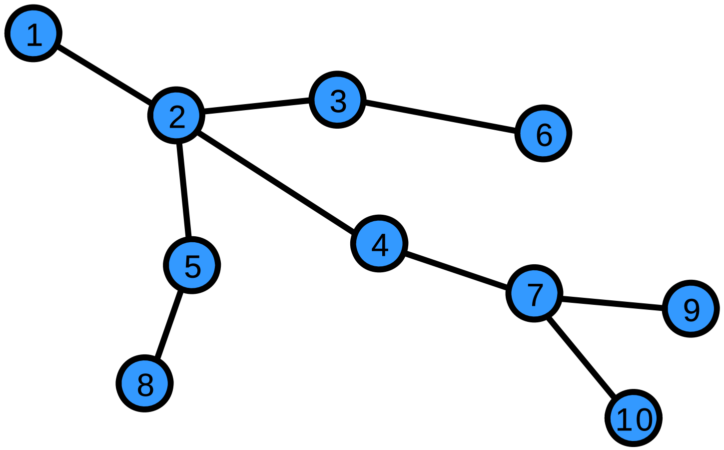
Current node: Prague
Todo list: [Warsaw]

Graph traversal (queue)



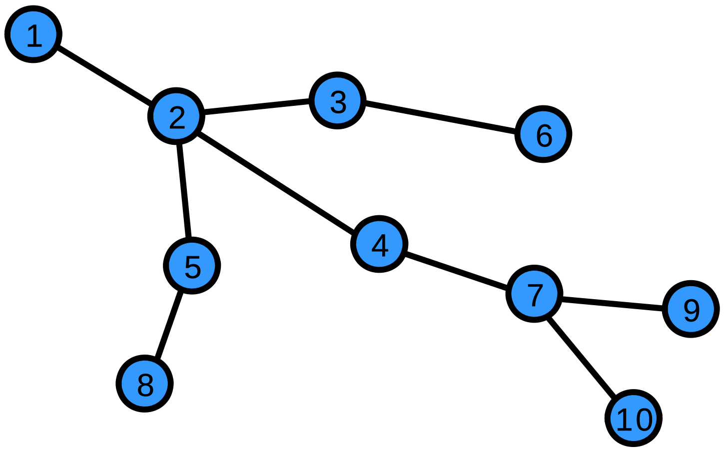
Current node: Warsaw
Todo list: [ ]

Breadth-first search (BFS)

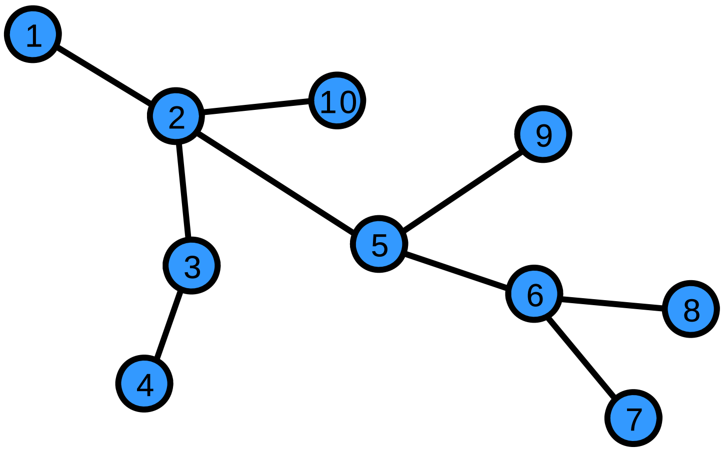


- We visit all the vertices at the same level (same distance to the root) before moving on to the next level

BFS vs. DFS

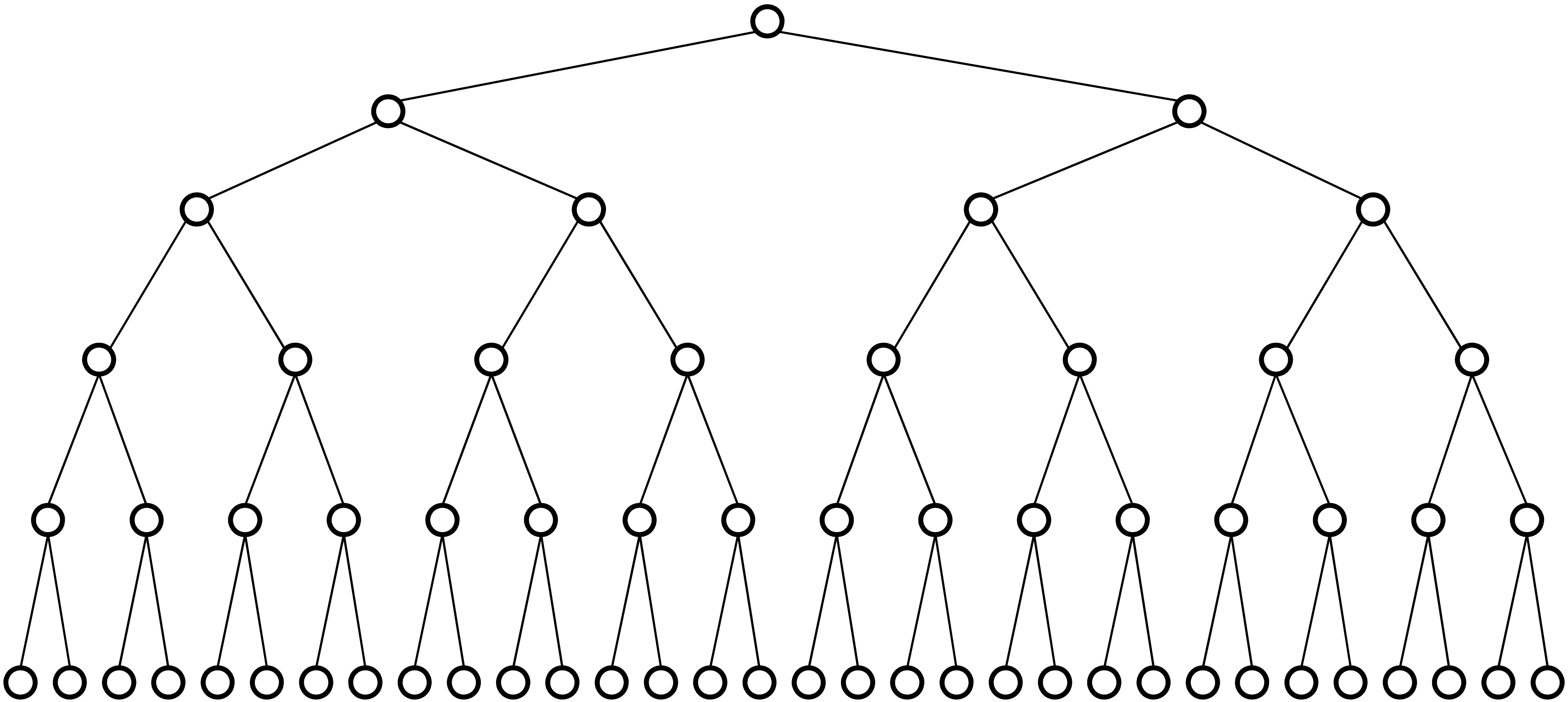


Breadth-first (queue)



Depth-first (stack)

BFS vs. DFS



(tree = graph with no cycles)

Basic algorithms

BREADTH-FIRST SEARCH (Graph G)

- While there is an uncolored node **r**
  - Choose a new color
  - Create an empty queue **Q**
  - Let **r** be the root node, color it, and add it to **Q**
  - While **Q** is not empty
    - Dequeue a node **v** from **Q**
    - For each of **v**'s neighbors **u**
      - If **u** is not colored, color it and add it to **Q**

Basic algorithms

DEPTH-FIRST SEARCH (Graph G)

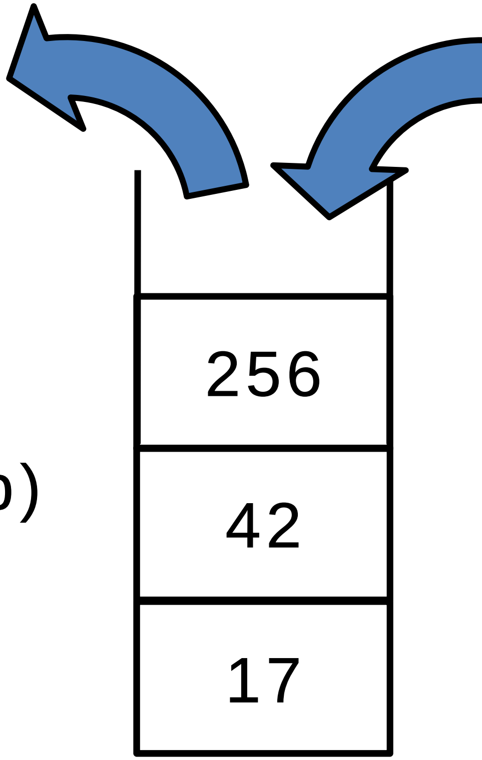
- While there is an uncolored node **r**
  - Choose a new color
  - Create an empty stack **S**
  - Let **r** be the root node, color it, and push it on **S**
  - While **S** is not empty
    - Pop a node **v** from **S**
    - For each of **v**'s neighbors **u**
      - If **u** is not colored, color it and push it onto **S**

Queues and Stacks

- Examples of Abstract Data Types (ADTs)
- ADTs fulfill a contract:
  - The contract tells you what the ADT can do, and what the behavior is
  - For instance, with a stack:
    - We can push and pop
    - If we push X onto S and then pop S, we get back X, and S is as before
- Doesn't tell you *how* it fulfills the contract

Implementing DFS

- How can we implement a stack?
  - Needs to support several operations:
  - Push (add an element to the top)
  - Pop (remove the element from the top)
  - IsEmpty



Implementing a stack

- IsEmpty

```
function e = IsEmpty(S)
    e = (length(S) == 0);
```

- Push (add an element to the top)

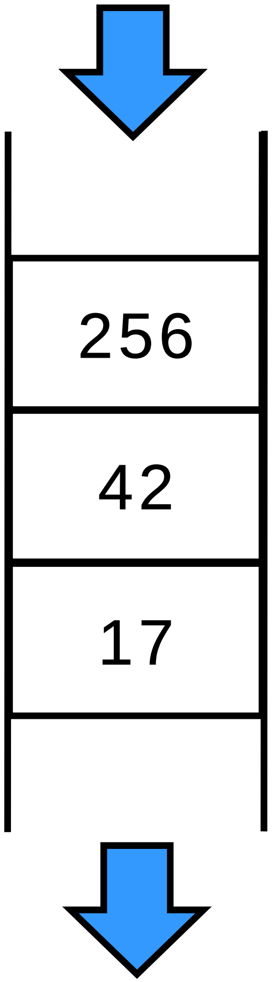
```
function S = push(S, x)
    S = [ S  x ]
```

- Pop (remove an element from the top)

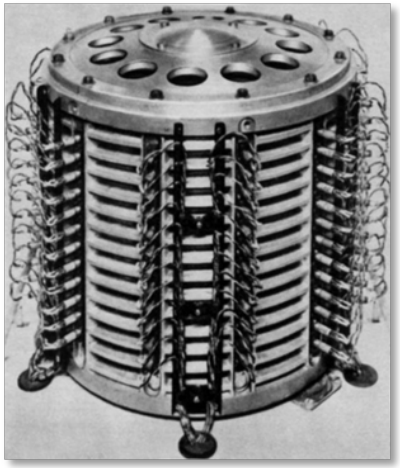
```
function [S, x] = pop(S)
    n = length(S); x = S(n); S = S(1:n-1);
    % but what happens if n = 0?
```

Implementing BFS

- How can we implement a queue?
  - Needs to support several operations:
  - Enqueue (add an element to back)
  - Dequeue (remove an element from front)
  - IsEmpty
- Not quite as easy as a stack...



Efficiency



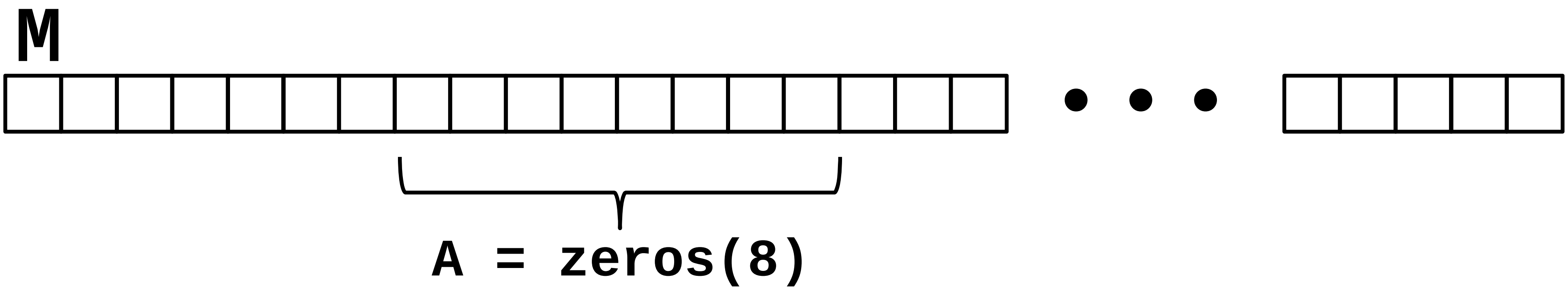
- Ideally, all of the operations (push, pop, enqueue, dequeue, IsEmpty) run in constant (O(1)) time
- To figure out running time, we need a model of how the computer's memory works

Computers and arrays

- Computer memory is a large array
  - We will call it M
- In constant time, a computer can:
  - Read any element of M (random access)
  - Change any element of M to another element
  - Perform any simple arithmetic operation
- This is more or less what the hardware manual for an x86 describes

Computers and arrays

- Arrays in Matlab are consecutive subsequences of M



Memory manipulation

- How long does it take to:
  - Read A(8)?
  - Set A(7) = A(8)?
  - Copy all the elements of an array (of size *n*) A to a new part of M?
  - Shift all the elements of A one cell to the left?

Implementing a queue: Take 1

- First approach: use an array
- Add (enqueue) new elements to the end of the array
- When removing an element (dequeue), shift the entire array left one unit

```
Q = [];
```

Implementing a queue: Take 1

- IsEmpty

```
function e = IsEmpty(Q)
    e = (length(S) == 0);
```

- Enqueue (add an element)

```
function Q = enqueue(Q,x)
    Q = [ Q x ];
```

- Dequeue (remove an element)

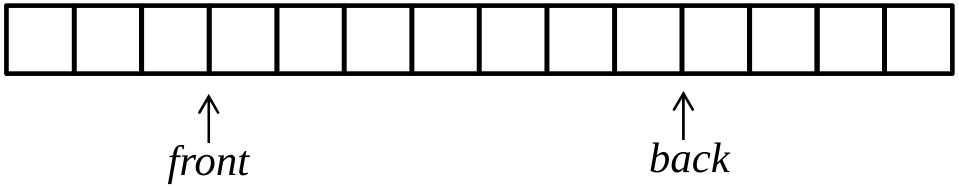
```
function [Q, x] = dequeue(Q)
    n = length(Q); x = Q(1);
    for i = 1:n-1
        Q(i) = Q(i+1); % everyone steps forward one step
```

What is the running time?

- IsEmpty
- Enqueue (add an element)
- Dequeue (remove an element)

Implementing a queue: Take 2

- Second approach: use an array AND
- Keep two pointers for the front and back of the queue



- Add new elements to the back of the array
- Take old elements off the front of the array

```
Q = zeros(1000000);
front = 1; back = 1;
```

Implementing a queue: Take 2

- IsEmpty
- Enqueue (add an element)
- Dequeue (remove an element)