Graph Algorithms

CMPE 180A DSA with Python

Sanja Damjanovic Week 9 9/9/2024

Week 9 Outline

- DFS & BFS applications:
    - The word ladder problem
    - The knight’s tour
- Graph algorithms:
    - Topological sorting
    - Strongly connected components
    - Dijkstra’s algorithm
- The final exam
    - 9/16 at 5.30 pm PT via Zoom (Week 10)
    - 150 minutes
- The term project presentations
    - 9/23 at 5.30 pm PT via Zoom (Week 11)
- Upload all deliverables
    - Before 9/28

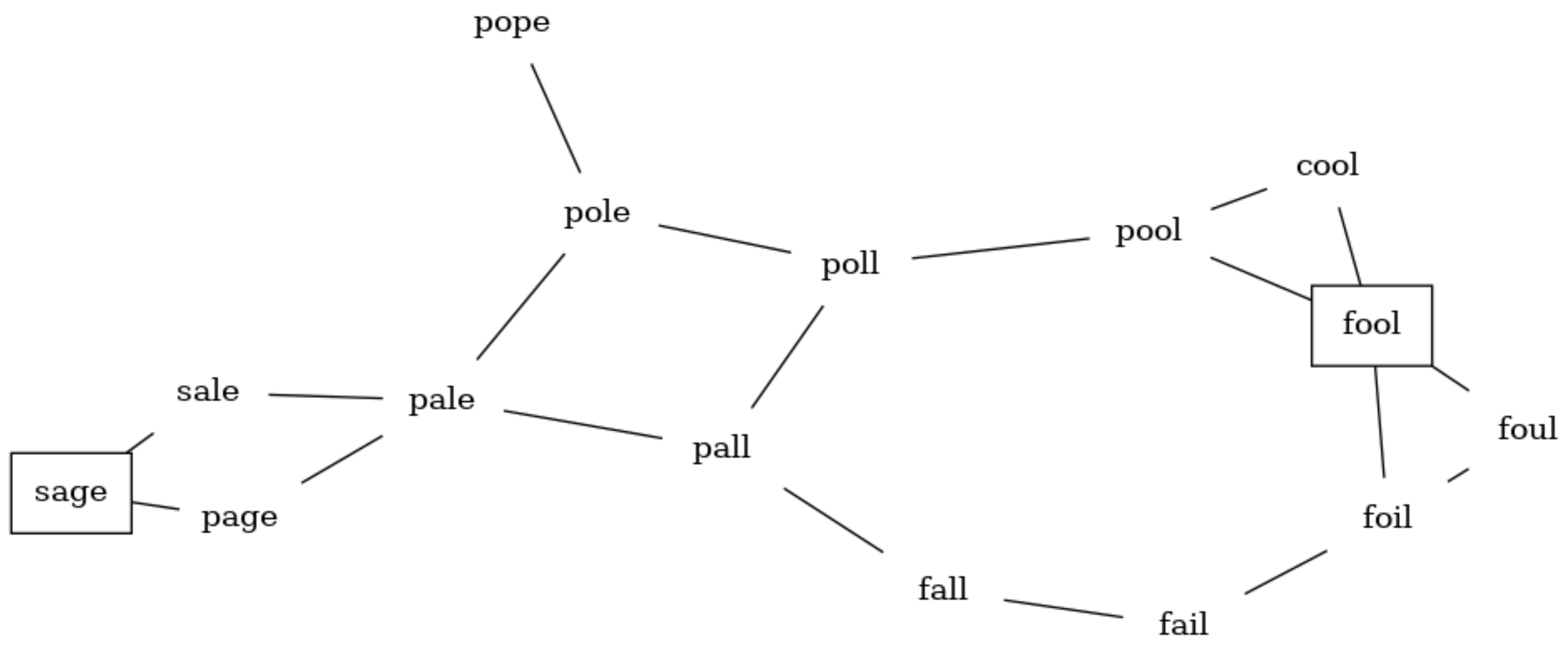
The Word Ladder Problem

BFS application

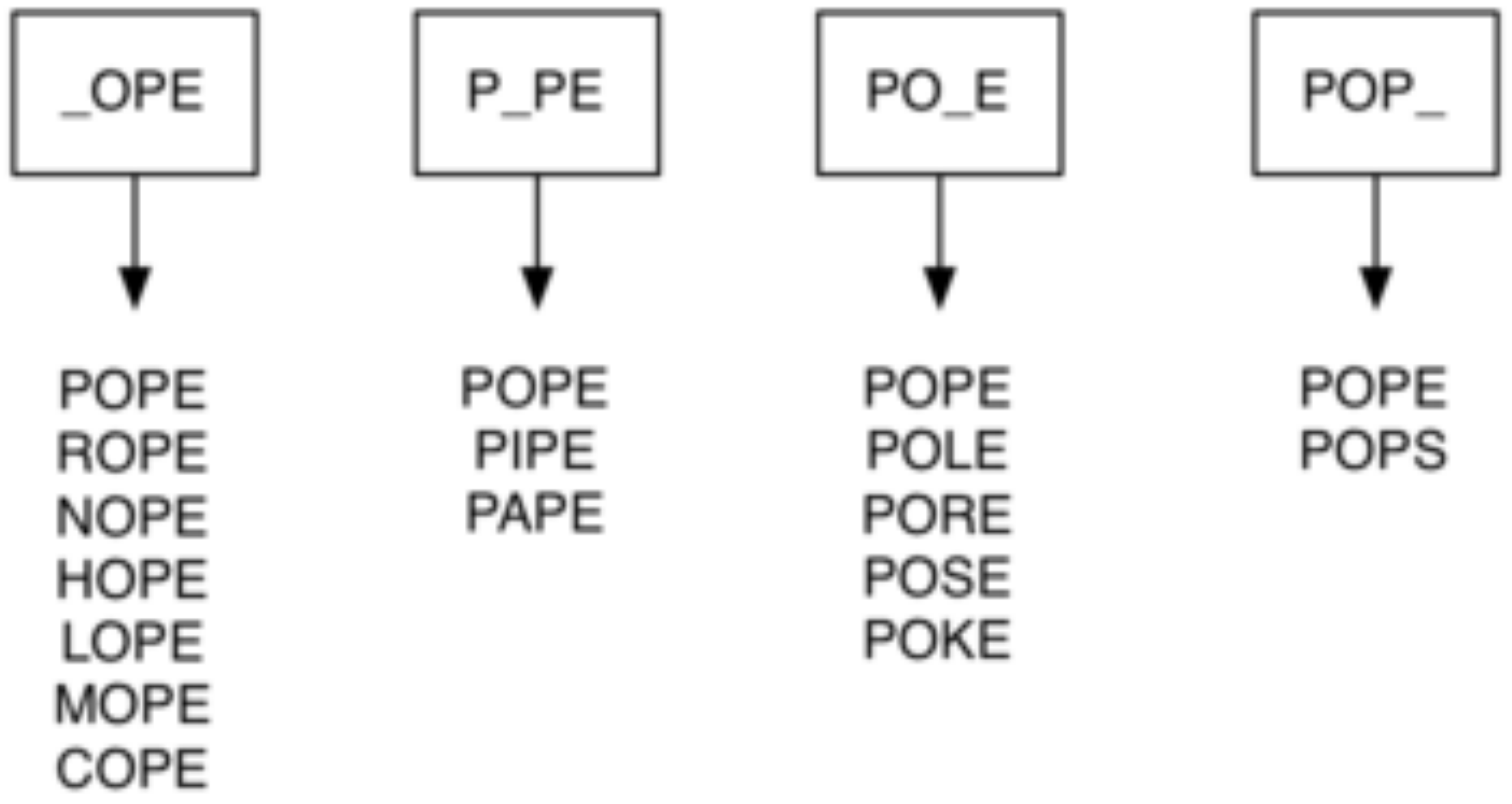
- invented in 1878 by Lewis Carroll, the author of *Alice in Wonderland*
- transform the word FOOL into the word SAGE
- gradually by changing one letter at a time:
  - FOOL- POOL-POLL-POLE-PALE-SALE-SAGE
  - Comparison ~ O(n^2)
  - Represent the relationships between the words as a graph.
  - Use the graph algorithm known as breadth-first search to find an efficient path from the starting word to the ending word

A Small Word Ladder Graph

-



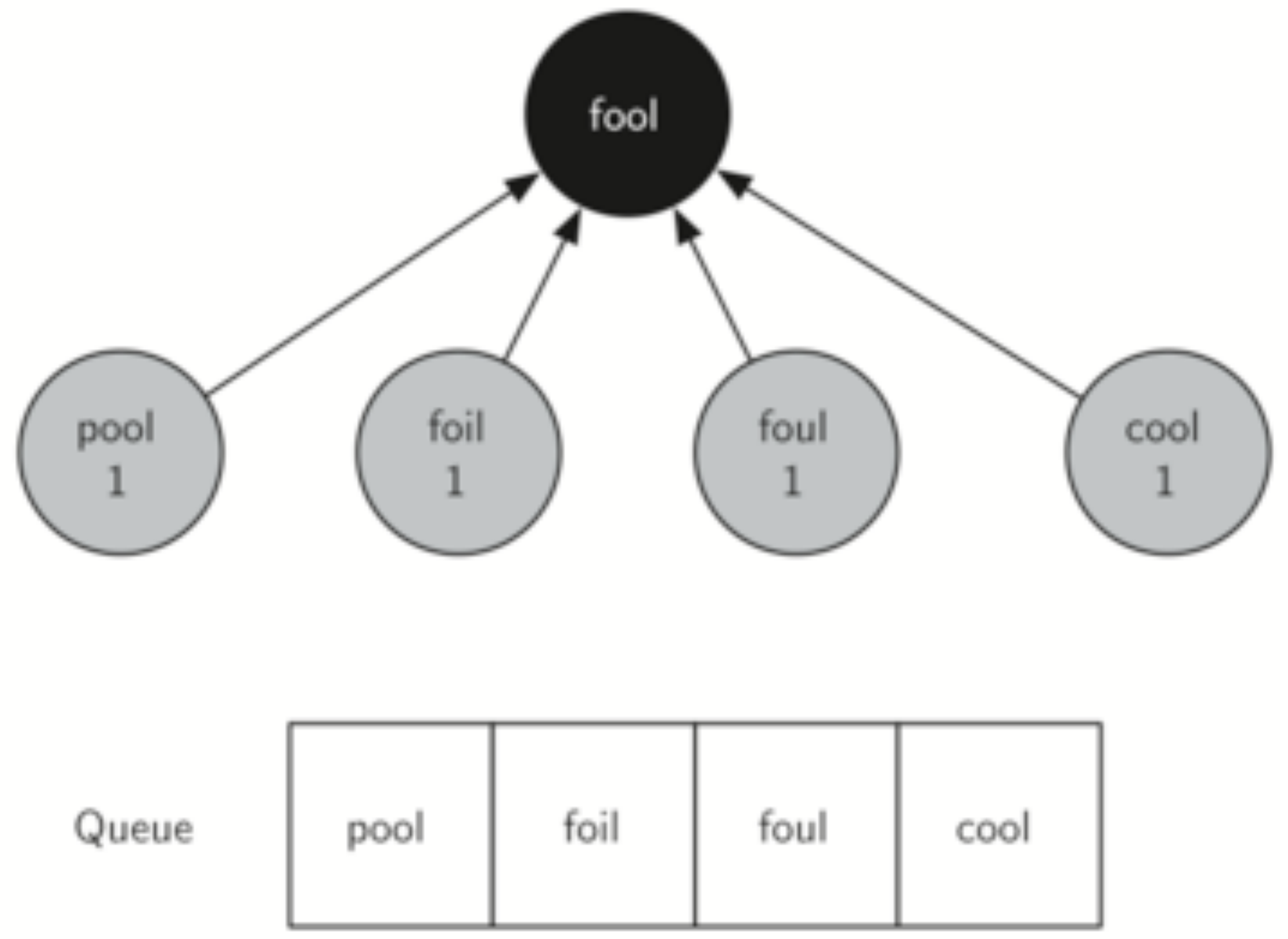
Word Buckets for Words That Differ by One Letter



BFS

Color

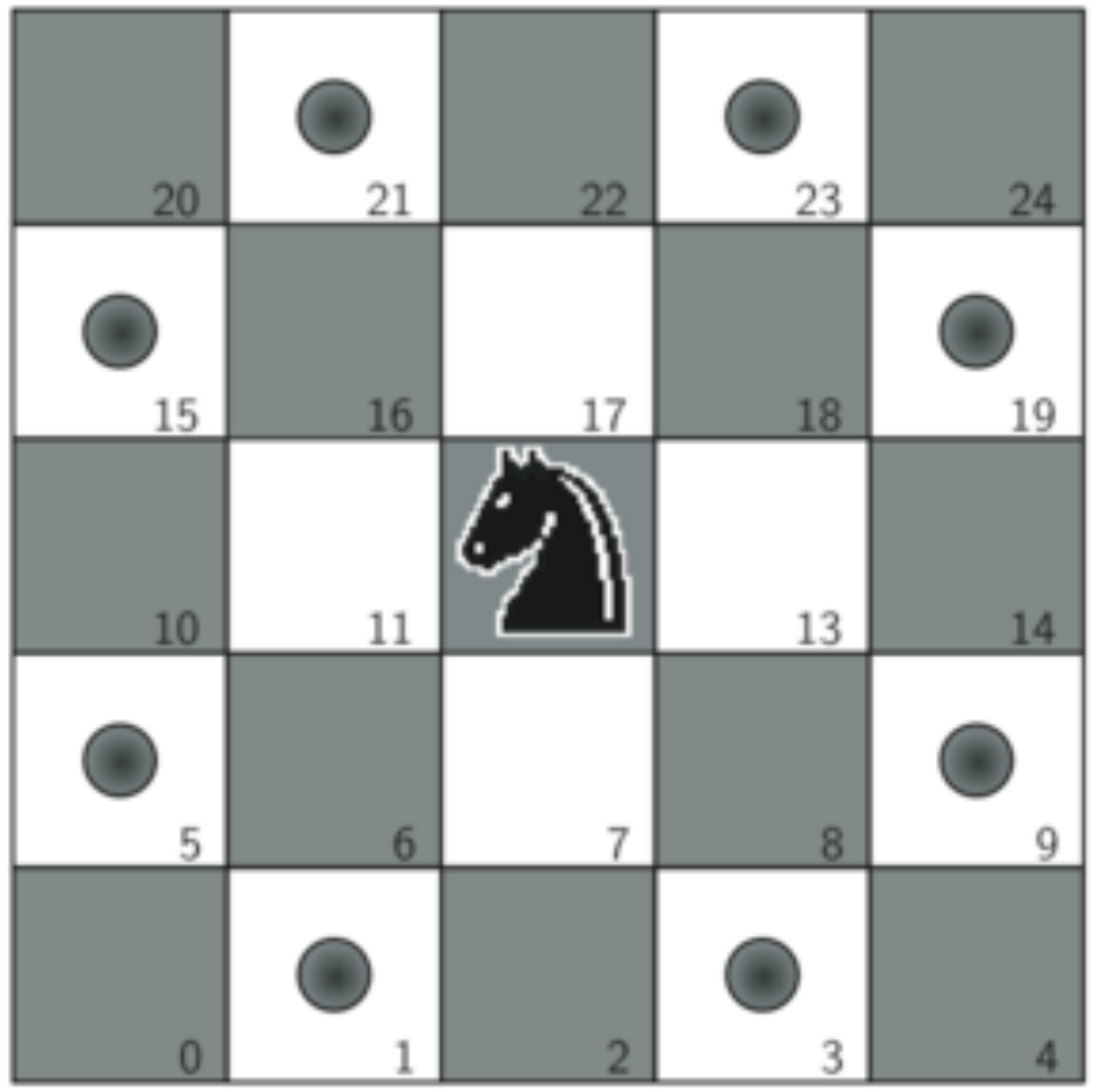
- To keep track of its progress, BFS colors each of the vertices
    - white, gray, or black
    - All the vertices are initialized to white when they are constructed. A white vertex is an undiscovered vertex.
    - When a vertex is initially discovered it is colored gray,
    - and when BFS has completely explored a vertex it is colored black.
    - This means that once a vertex is colored black, it has no white vertices adjacent to it. A gray node, on the other hand, may have some white vertices adjacent to it, indicating that there are still additional vertices to explore.

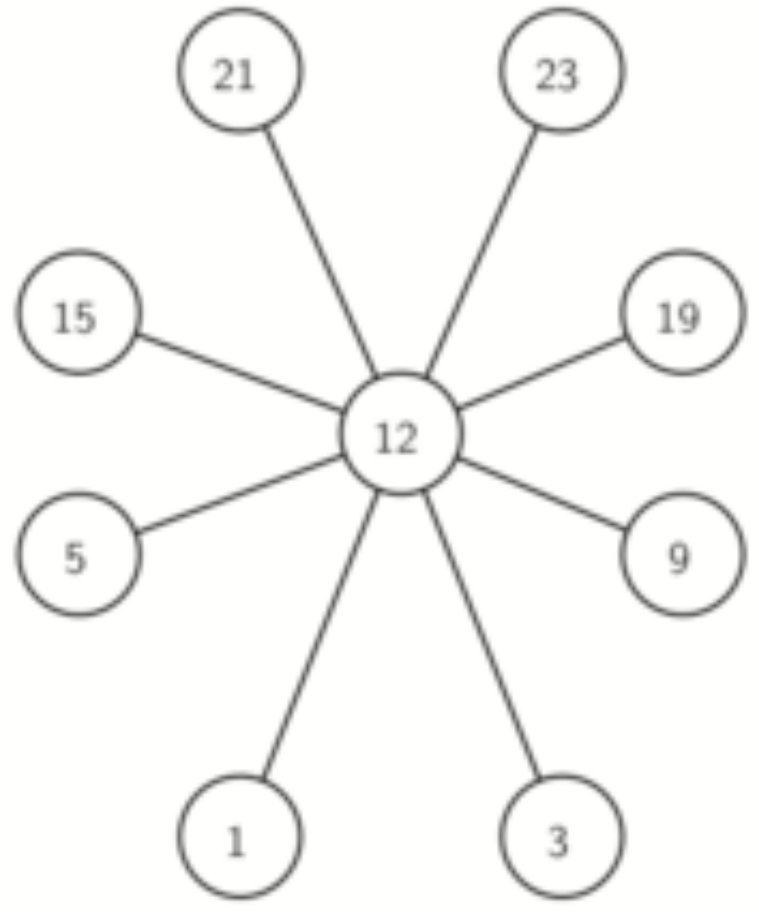


Building the Knight's Tour Graph

DFS application

- Each square on the chessboard can be represented as a node in the graph and each legal move by the knight can be represented as an edge in the graph.
- Legal moves for a knight on square 12 and the corresponding graph



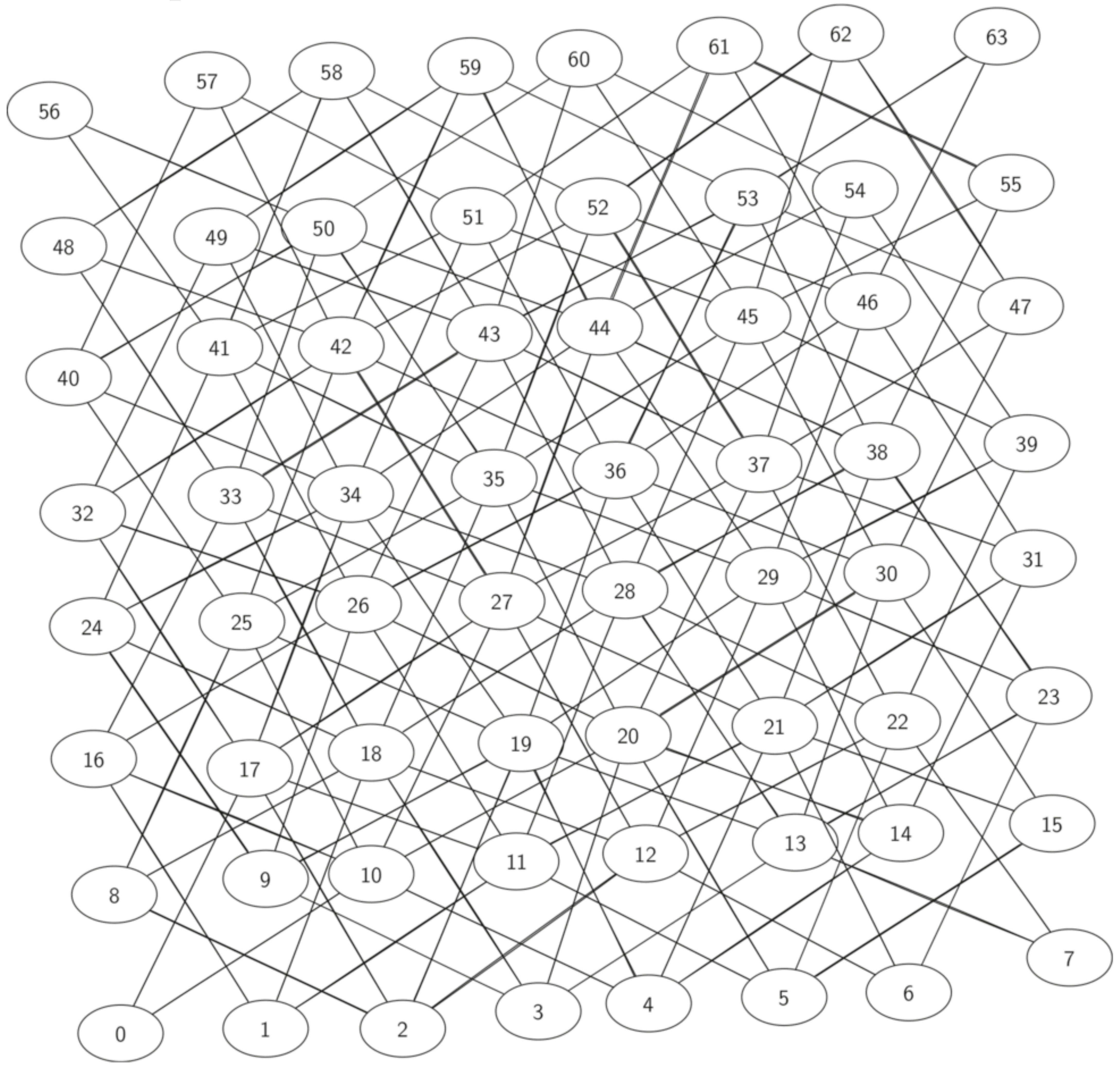


Generate Graph

- build the full graph for an *n*-by-*n* board
  - Vertices
  - Edges

All possible moves

The Knight's Tour Graph



Graph algorithms

BFS complexity

the run time performance

- # vertices = |V|
- # edges = |E|
- BFS: O(|V|+|E|)
    - The while loop is executed, at most, one time for each vertex in the graph. A vertex must be white before it can be examined and added to the queue.
        - O(|V|) for the *while* loop.
    - The for loop, which is nested inside the while is executed at most once for each edge in the graph. Every vertex is dequeued at most once, an edge from node to node is examined only when node is dequeued.
        - O(|E|) for the *for* loop

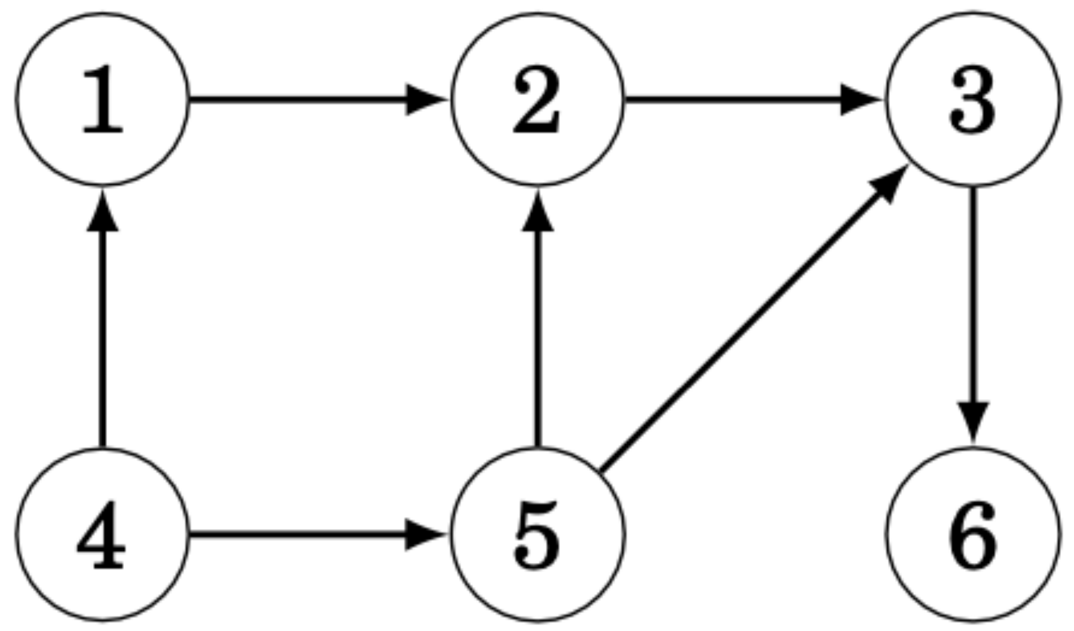
DFS performance

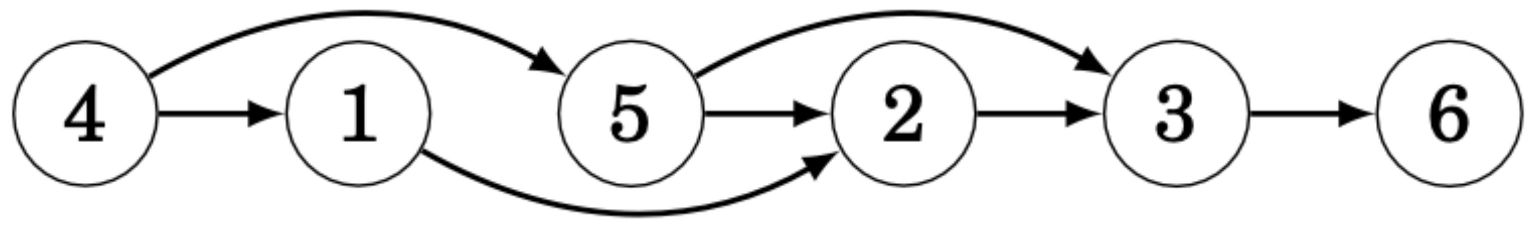
the run time performance

- The loops in *dfs* both run in O(|V|)
- *dfsvisit*: the loop is executed once for each edge in the adjacency list of the current vertex - if the vertex is ‘white’: O(|E|)
- DFS: O(|V|+|E|)

Topological sorting

- A **topological sort** is an ordering of the nodes of a directed acyclic graph such that if there is a path from node *a* to node *b*, then node *a* appears before node *b* in the ordering
- An acyclic graph always has a topological sort
- Applications to indicate the precedence of events
- Q1: *Which approach is more suitable for topological sorting: BFS or DFS?*





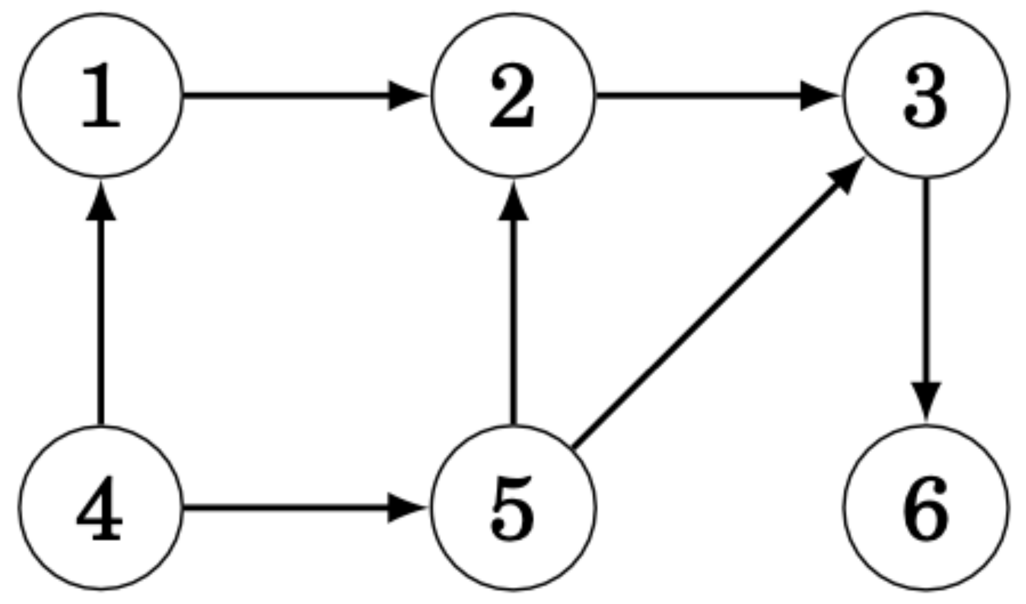
.

Topological Sorting

Implementation

- Depth-first search:
  - Initially, the state of each node is 0. When a search reaches a node for the first time, its state becomes 1. Finally, after all successors of the node have been processed, its state becomes 2.
  - add each node to a list when the state of the node becomes 2. This list in reverse order is a topological sort.

- Node states:
  - state 0: the node has not been processed (white)
  - state 1: the node is under processing (light gray)
  - state 2: the node has been processed (dark gray)

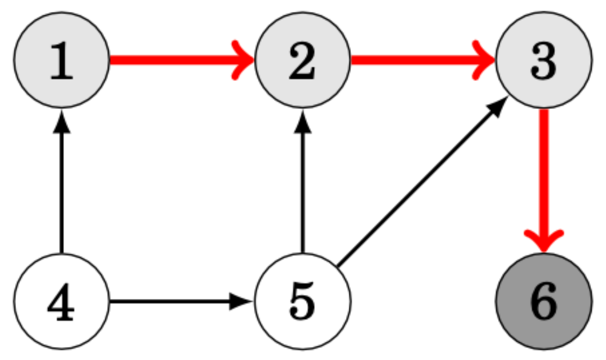


Topological sorting

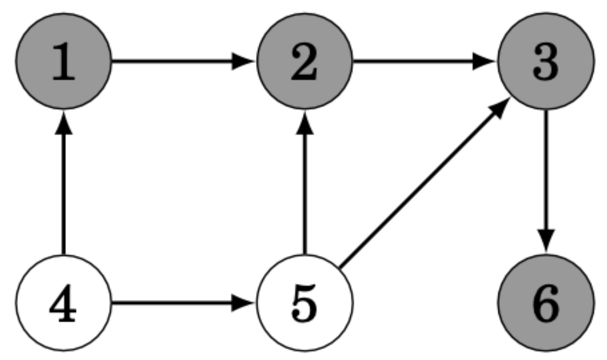
Example

1. Call `dfs(g)` for some graph `g`. The main reason we want to call depth first search is to compute the finish times for each of the vertices.
2. Store the vertices in a list in decreasing order of finish time.
3. Return the ordered list as the result of the topological sort.

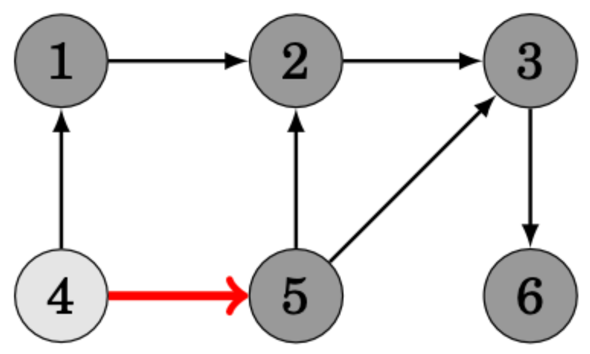
- [6]

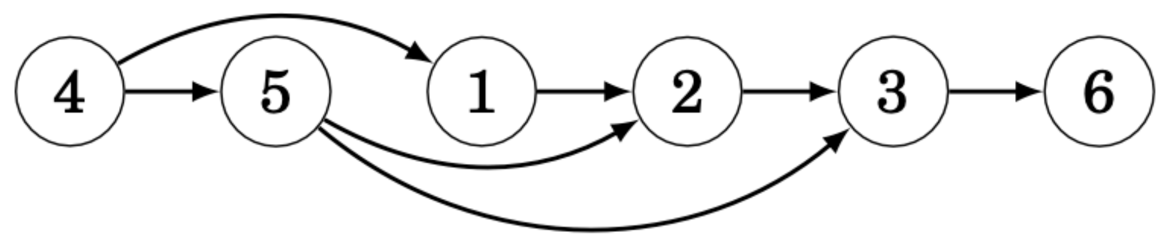


- [6,3,2,1]



- Search begins at 4





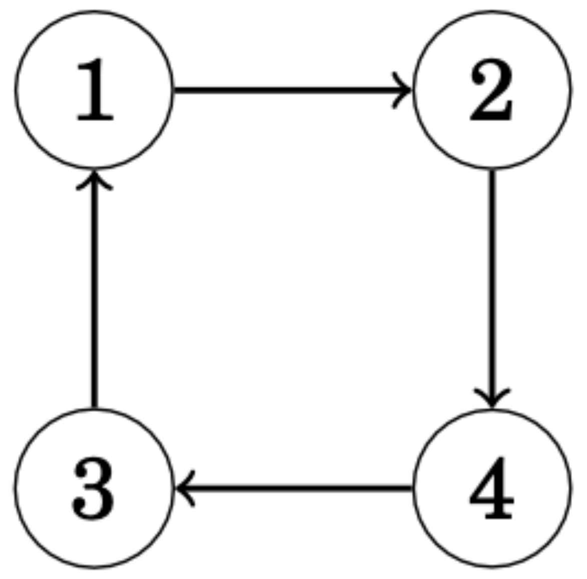
Strongly Connected Components

- Strongly connected component

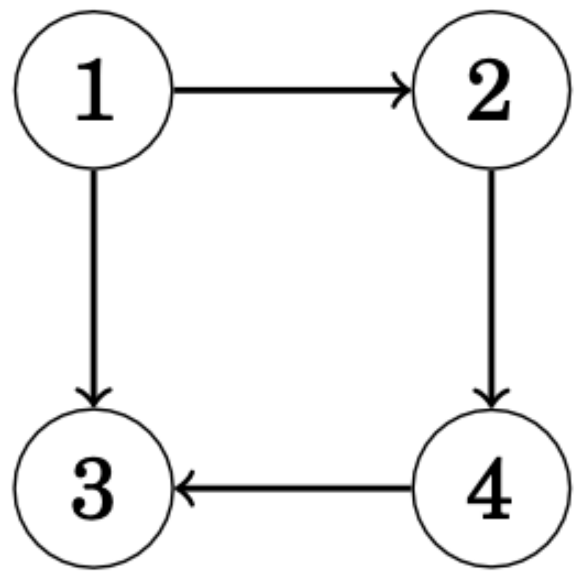
  C, of a graph G, as the largest subset of vertices $C \subset V$ such that for every pair of vertices $v, w \in C$ we have a path from v to w and a path from w to v

- A graph is **strongly connected** if there is a path from any node to all other nodes in the graph.

- *Q2: Is a graph strongly connected? Left graph? Right graph?*

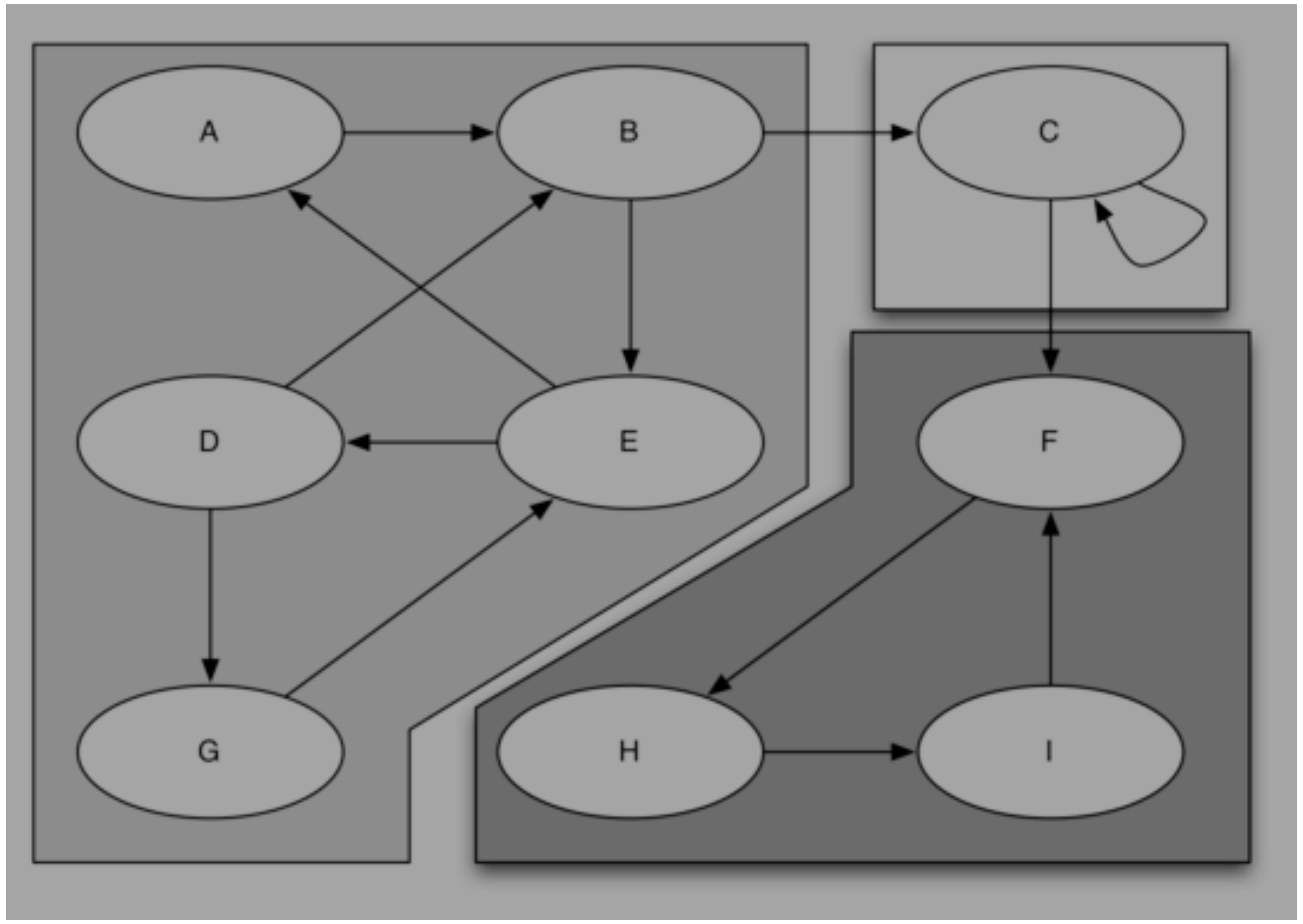


Left graph

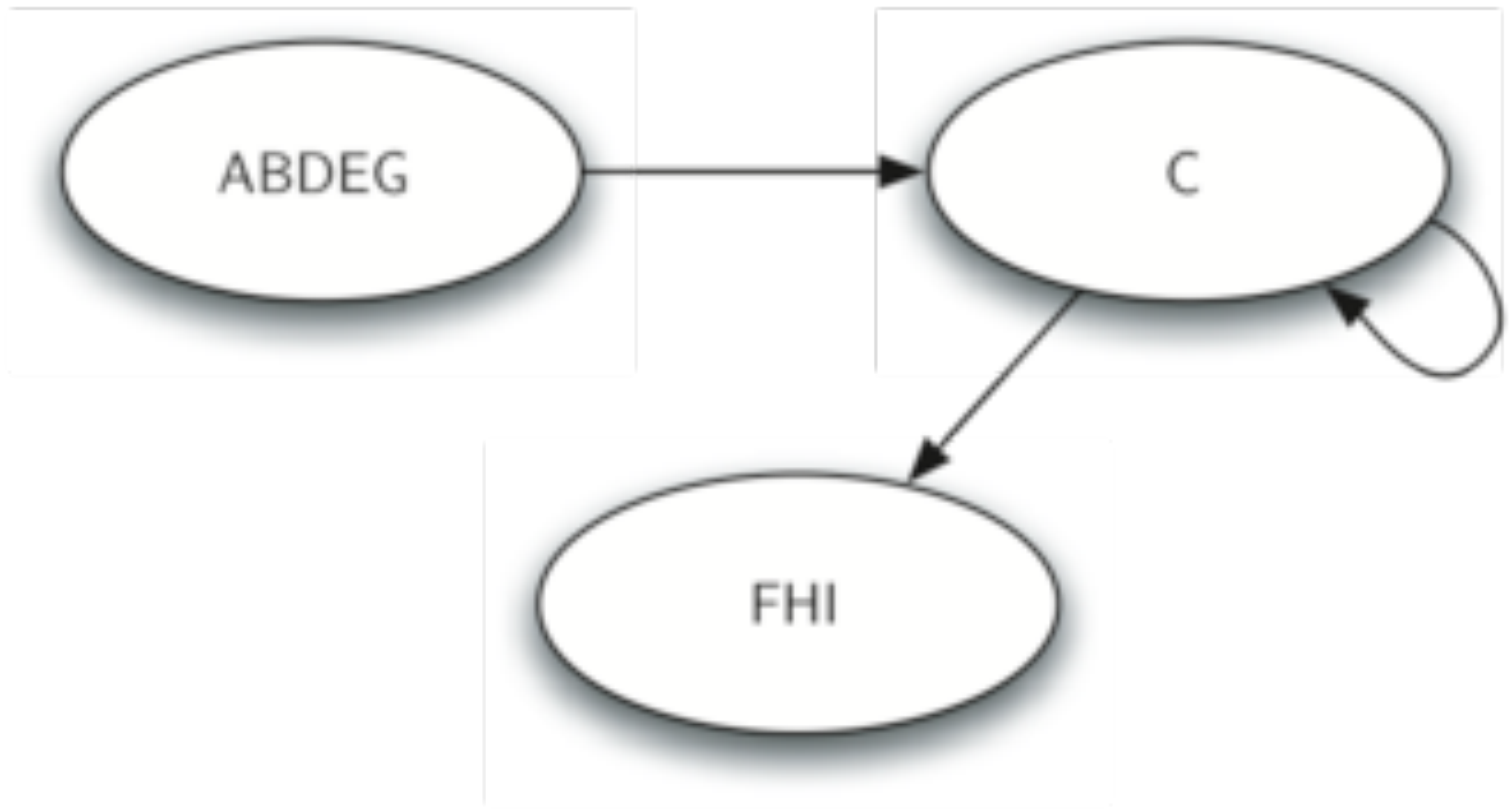


Right graph

SCC Example



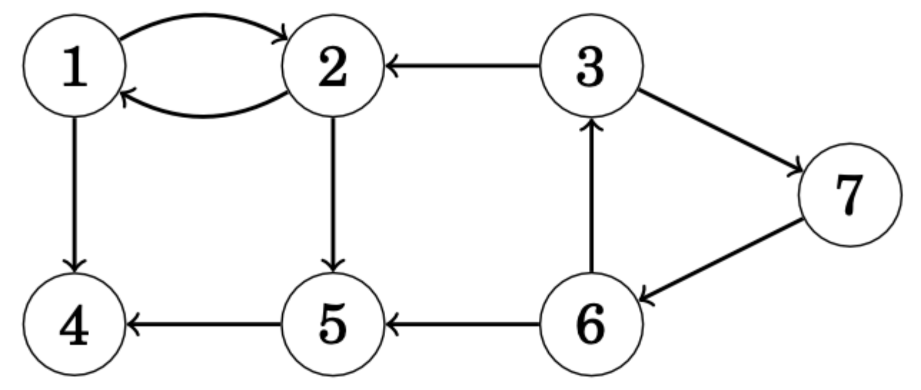
A Directed Graph with Three Strongly Connected Components

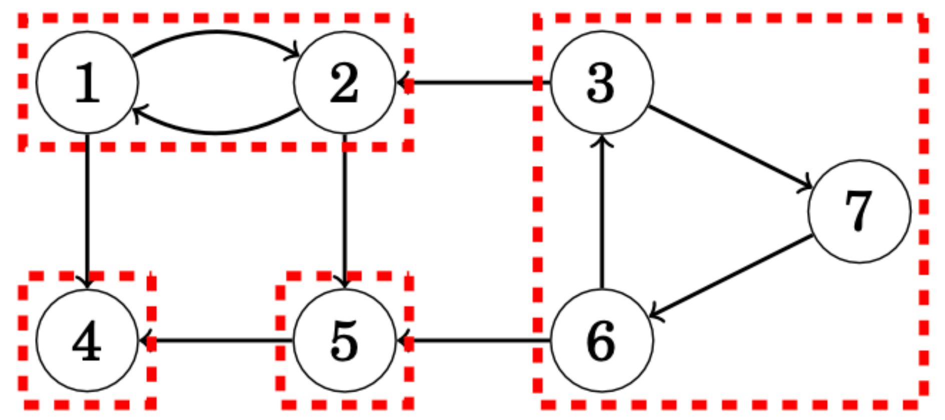


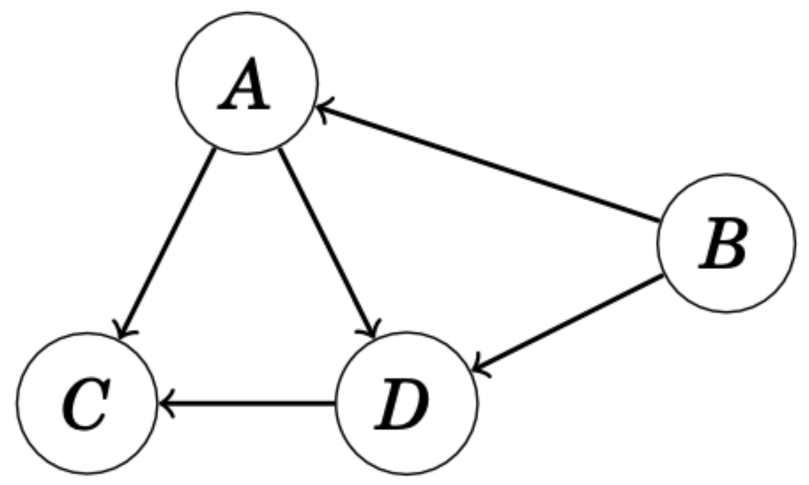
The Reduced Graph

Strongly connected components (SCC)

- The **strongly connected components** of a graph divide the graph into strongly connected parts that are as large as possible





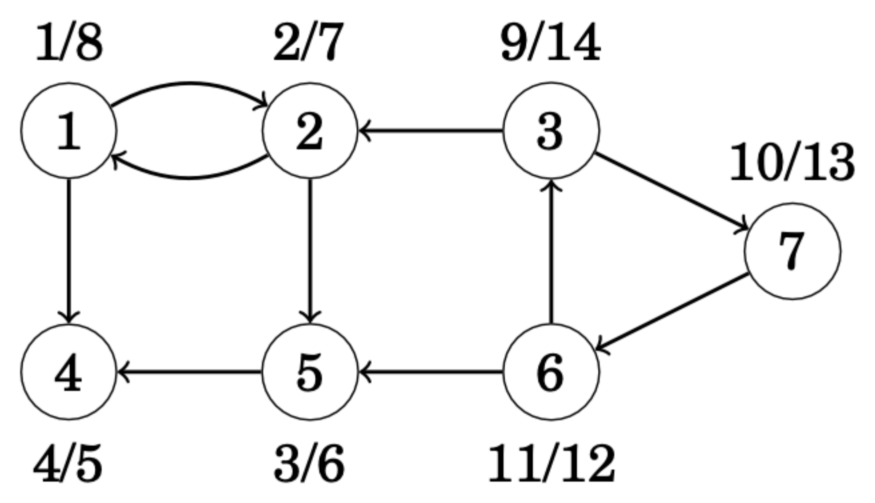


- *A* = {1,2}, *B* = {3,6,7}, *C* = {4} and *D* = {5}

SCC

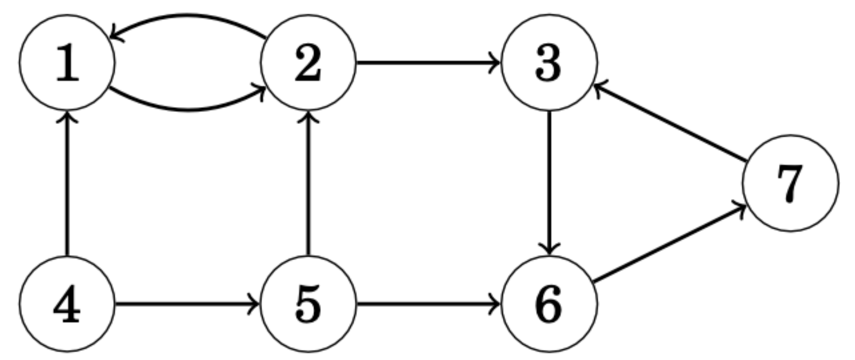
Kosaraju's algorithm

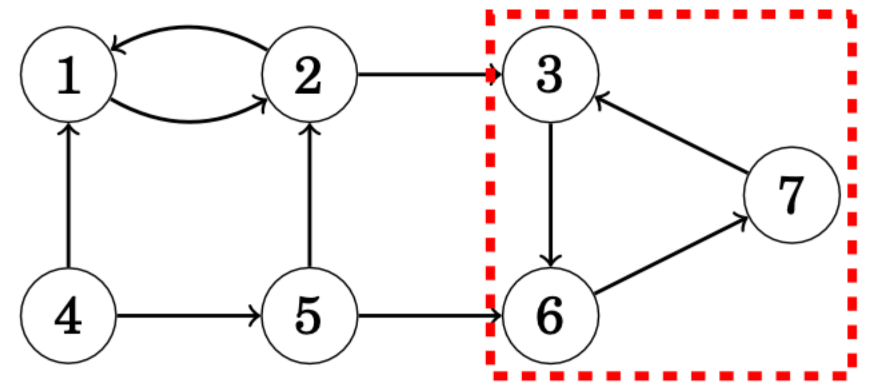
- DFS

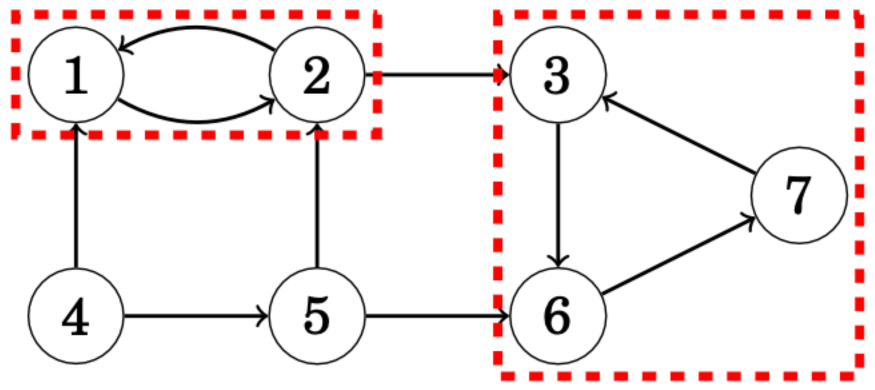


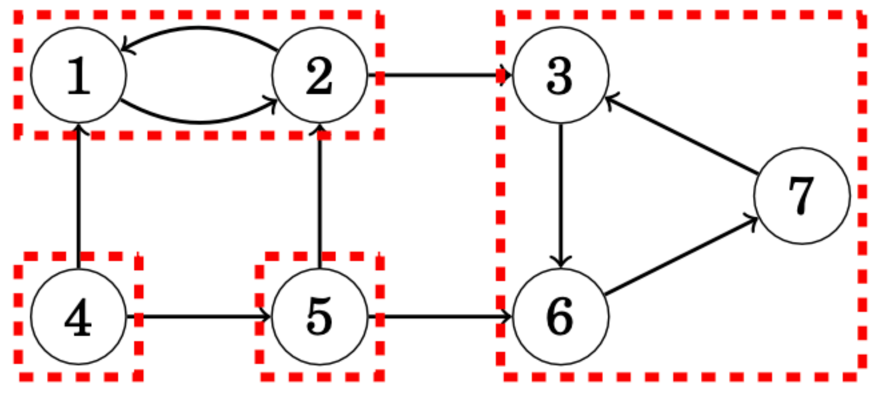
node	processing time
4	5
5	6
2	7
1	8
6	12
7	13
3	14

- Reverse edges and reverse DFS order based on the *end-processing time*



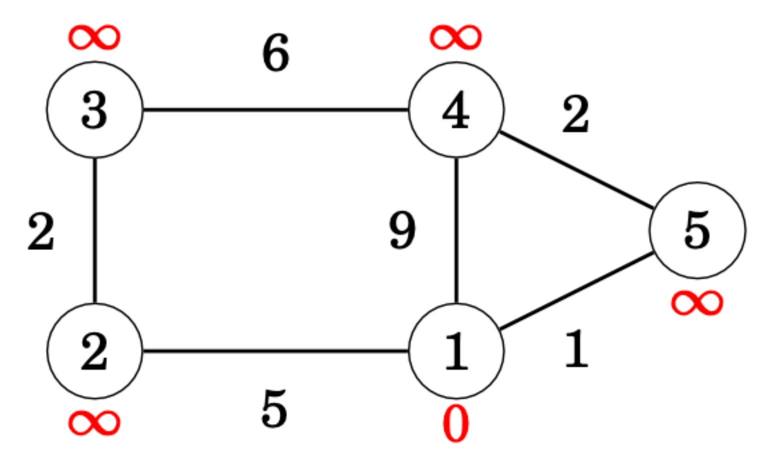






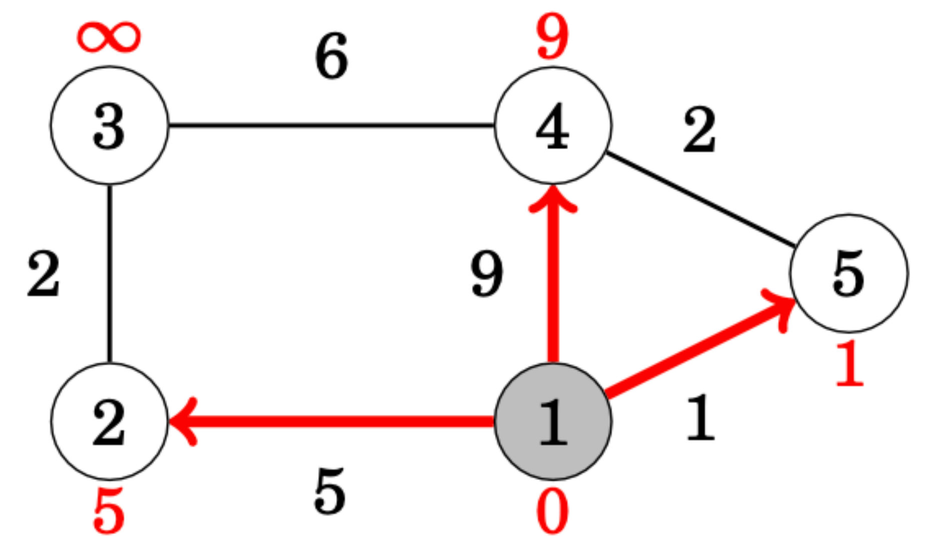
Dijkstra’s Algorithm (1)

- finds shortest paths from the starting node to all nodes of the graph
- requires that there are no negative weight edges in the graph
- Starting node 1 and distance to it 0



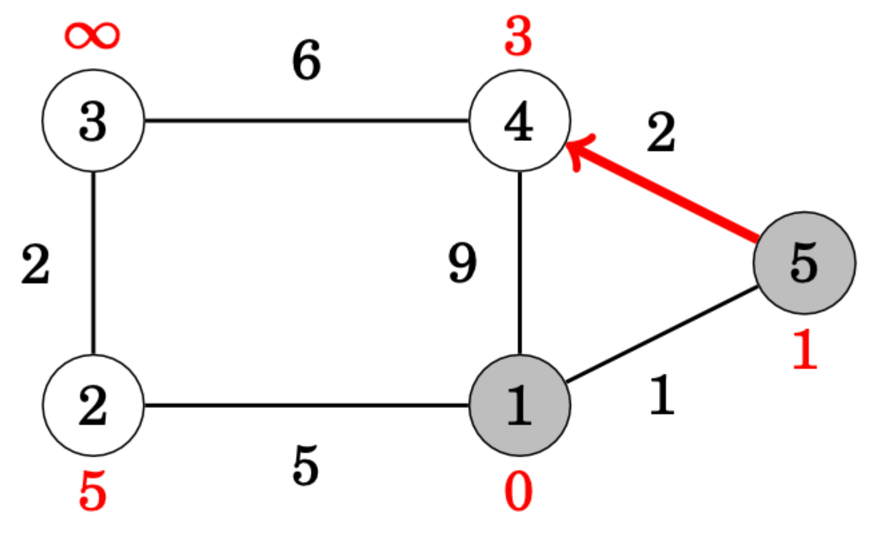
Dijkstra’s Algorithm (2)

- At each step, Dijkstra’s algorithm selects a node that has not been processed yet and whose distance is as small as possible. The first such node is node 1 with distance 0.
- When a node is selected, the algorithm goes through all edges that start at the node and reduces the distances using them
    - In this case, the edges from node 1 reduced the distances to nodes 2, 4 and 5, whose distances are now 5, 9 and 1.

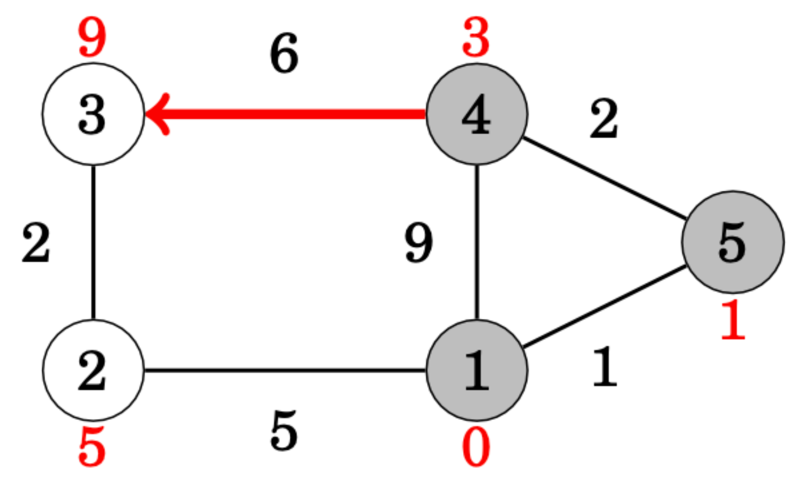


Dijkstra’s Alg (3)

The next node to be processed is node 5 with distance 1. This reduces the distance to node 4 from 9 to 3:

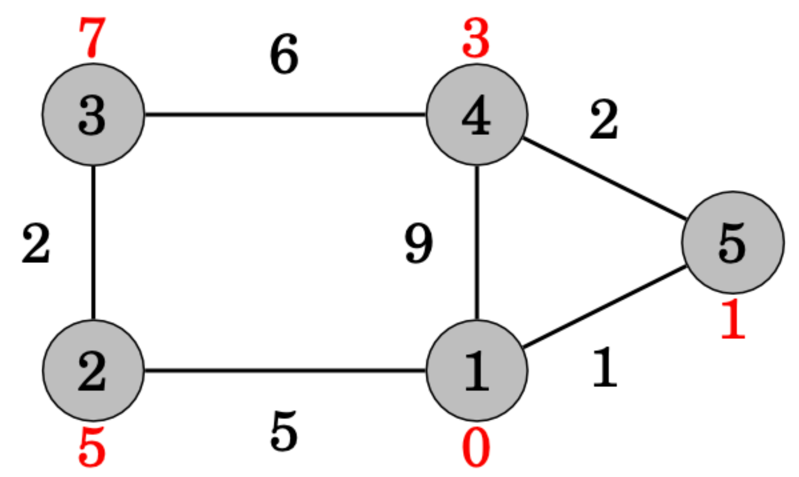


After this, the next node is node 4, which reduces the distance to node 3 to 9:



A remarkable property in Dijkstra’s algorithm is that whenever a node is selected, its distance is final. For example, at this point of the algorithm, the distances 0, 1 and 3 are the final distances to nodes 1, 5 and 4.

After this, the algorithm processes the two remaining nodes, and the final distances are as follows:



The Final Preparation

- Lecture slides: terms, definitions, in-class questions
- Labs: assignments
- Shared notebooks

The Final Preparation

Example Questions

- Which data structure and/or algorithm would you apply to find the K largest elements within an array?
- Write a function that prints the values stored in the nodes of a binary tree in **descending** order and define the node class:
  - The binary tree node class should include:
    - A data value for each node.
    - A reference (LPT) to the left subtree, containing nodes with values less than or equal to the node's value.
    - A reference (GPT) to the right subtree, containing nodes with values greater than the node's value.
- Binary tree: the execution time growth with increasing amount of data n?
- Write a recursive function for computing powers x with the signature:
  - def power(x,n):
  - X is integer and may be positive, negative or 0

The Final Preparation

Example Questions

- Write a function
  - To remove duplicates from linked list
  - To find a height of a tree
- Concepts:
  - Recognize terms: leaf node, internal node, level, height
  - What is a result of running the given code: such as a sorting algorithm, binary search, etc
  - Modify the given function to achieve the result: e.g. binary search to find the first occurrence of the target number on the array with repetitions

Examples

- P1: Binary search: Modify the binary search so that returns the index of the first occurrence of x in a non-decreasing array (or -1 if not present)
- P2: Graph nodes in-degrees (or outdegrees)
- P3 Top K Frequent Elements
- P4 Merge Sorted Array (in place)
- P5 Valid Parentheses
- P6 Minimum Remove to Make Valid Parentheses
- P7 Valid palindrome
- P8 Valid Word Abbreviation