Graph Traversal

Data Structures and Algorithms for Computational Linguistics III (ISCL-BA-07)

Çağrı Çöltekin
ccoltekin@sfs.uni-tuebingen.de

University of Tübingen
Seminar für Sprachwissenschaft

Winter Semester 2024/25

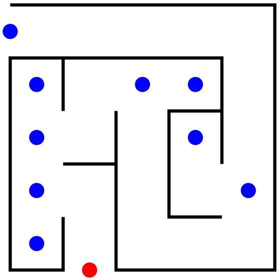
version: 151cdd0+ @2024-11-13

Graph traversal

- A graph traversal is a systematic way to visit all nodes in a graph
- Graph traversal is one of the basic tasks on a graph, answering many interesting questions
  - Is there a path from one node to another?
  - What is the shortest path (with minimum number of edges) between two nodes?
  - Is the graph connected?
  - Is the graph cyclic?
  - ...
- Two main methods of traversals are *breadth-first* and *depth-first*

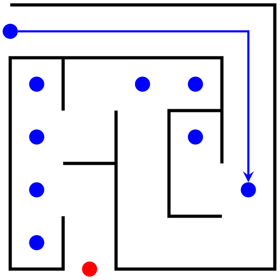
DFS - intuition

- Depth first search follows the same idea as exploring a labyrinth with a string and a chalk
- Visit each intersection (node), while marking the path you took with the string
- Mark each visited node, backtrack (following the string) when hit a dead end



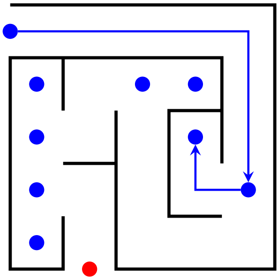
DFS - intuition

- Depth first search follows the same idea as exploring a labyrinth with a string and a chalk
- Visit each intersection (node), while marking the path you took with the string
- Mark each visited node, backtrack (following the string) when hit a dead end



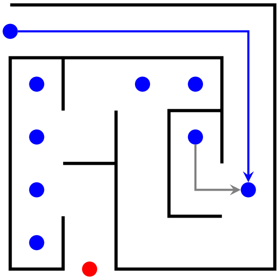
DFS - intuition

- Depth first search follows the same idea as exploring a labyrinth with a string and a chalk
- Visit each intersection (node), while marking the path you took with the string
- Mark each visited node, backtrack (following the string) when hit a dead end



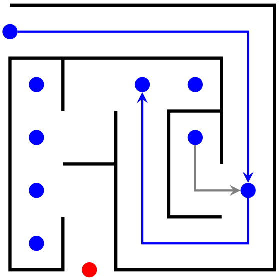
DFS - intuition

- Depth first search follows the same idea as exploring a labyrinth with a string and a chalk
- Visit each intersection (node), while marking the path you took with the string
- Mark each visited node, backtrack (following the string) when hit a dead end



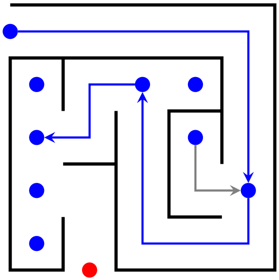
DFS - intuition

- Depth first search follows the same idea as exploring a labyrinth with a string and a chalk
- Visit each intersection (node), while marking the path you took with the string
- Mark each visited node, backtrack (following the string) when hit a dead end



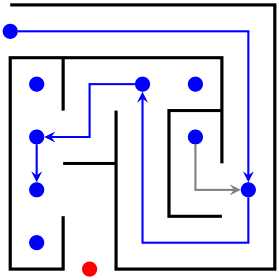
DFS - intuition

- Depth first search follows the same idea as exploring a labyrinth with a string and a chalk
- Visit each intersection (node), while marking the path you took with the string
- Mark each visited node, backtrack (following the string) when hit a dead end



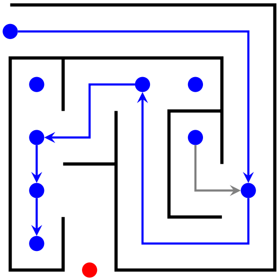
DFS - intuition

- Depth first search follows the same idea as exploring a labyrinth with a string and a chalk
- Visit each intersection (node), while marking the path you took with the string
- Mark each visited node, backtrack (following the string) when hit a dead end



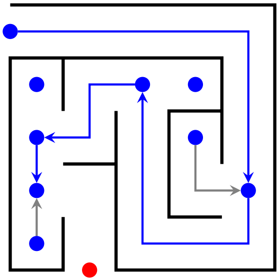
DFS - intuition

- Depth first search follows the same idea as exploring a labyrinth with a string and a chalk
- Visit each intersection (node), while marking the path you took with the string
- Mark each visited node, backtrack (following the string) when hit a dead end



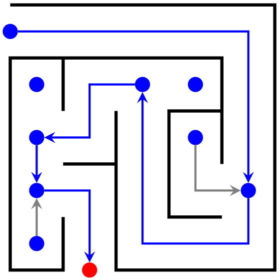
DFS - intuition

- Depth first search follows the same idea as exploring a labyrinth with a string and a chalk
- Visit each intersection (node), while marking the path you took with the string
- Mark each visited node, backtrack (following the string) when hit a dead end



DFS - intuition

- Depth first search follows the same idea as exploring a labyrinth with a string and a chalk
- Visit each intersection (node), while marking the path you took with the string
- Mark each visited node, backtrack (following the string) when hit a dead end

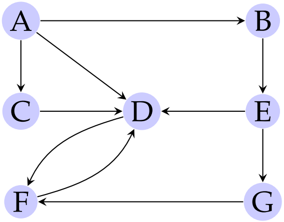


DFS - algorithm

```
def dfs(start, visited=None):
  if visited is None:
    visited = {start: None}
  for node in start.neighbors():
    if node not in visited:
      visited[node] = start
      dfs(node, visited)
```

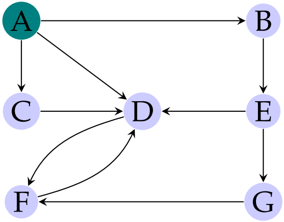
- Depth-first search (DFS) is easy with recursion
- DFS starts from a start node
- Marks each node it visits as *visited* (typically put it in a set data structure)
- Then, take an arbitrary *unvisited* neighbor, and continue visiting the nodes recursively
- Algorithm terminates when backtracking leads to the start node with no unvisited nodes left

DFS - demonstration, definitions



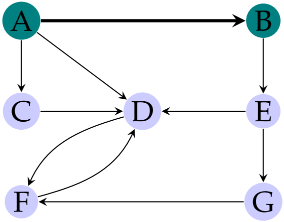
- The edges that we take to discover a new node are called the **discovery edges**
- The discovery edges form the DFS tree
- The other edges are called non-tree edges
- The edges to an ancestor in the DFS tree are called back edges
- The edges to a descendant node in the DFS tree are called forward edges
- The edges to a non-ancestor/non-descendant node in the BFS tree are called cross edges

DFS - demonstration, definitions



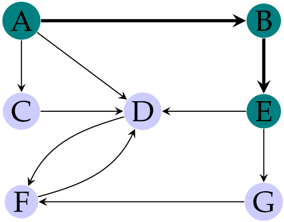
- The edges that we take to discover a new node are called the **discovery edges**
- The discovery edges form the DFS tree
- The other edges are called non-tree edges
- The edges to an ancestor in the DFS tree are called back edges
- The edges to a descendant node in the DFS tree are called forward edges
- The edges to a non-ancestor/non-descendant node in the BFS tree are called cross edges

DFS - demonstration, definitions



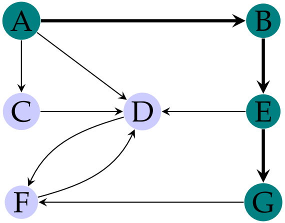
- The edges that we take to discover a new node are called the **discovery edges**
- The discovery edges form the DFS tree
- The other edges are called non-tree edges
- The edges to an ancestor in the DFS tree are called back edges
- The edges to a descendant node in the DFS tree are called forward edges
- The edges to a non-ancestor/non-descendant node in the BFS tree are called cross edges

DFS - demonstration, definitions



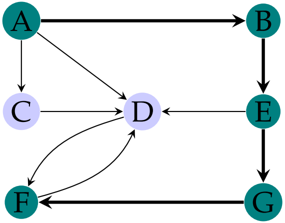
- The edges that we take to discover a new node are called the **discovery edges**
- The discovery edges form the DFS tree
- The other edges are called non-tree edges
- The edges to an ancestor in the DFS tree are called back edges
- The edges to a descendant node in the DFS tree are called forward edges
- The edges to a non-ancestor/non-descendant node in the BFS tree are called cross edges

DFS - demonstration, definitions



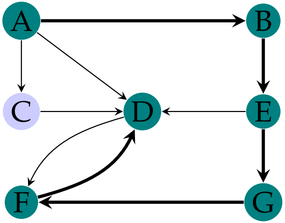
- The edges that we take to discover a new node are called the **discovery edges**
- The discovery edges form the DFS tree
- The other edges are called non-tree edges
- The edges to an ancestor in the DFS tree are called back edges
- The edges to a descendant node in the DFS tree are called forward edges
- The edges to a non-ancestor/non-descendant node in the BFS tree are called cross edges

DFS - demonstration, definitions



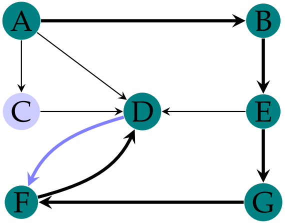
- The edges that we take to discover a new node are called the **discovery edges**
- The discovery edges form the DFS tree
- The other edges are called non-tree edges
- The edges to an ancestor in the DFS tree are called back edges
- The edges to a descendant node in the DFS tree are called forward edges
- The edges to a non-ancestor/non-descendant node in the BFS tree are called cross edges

DFS - demonstration, definitions



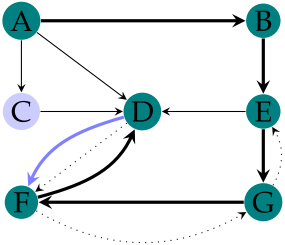
- The edges that we take to discover a new node are called the **discovery edges**
- The discovery edges form the DFS tree
- The other edges are called non-tree edges
- The edges to an ancestor in the DFS tree are called back edges
- The edges to a descendant node in the DFS tree are called forward edges
- The edges to a non-ancestor/non-descendant node in the BFS tree are called cross edges

DFS - demonstration, definitions



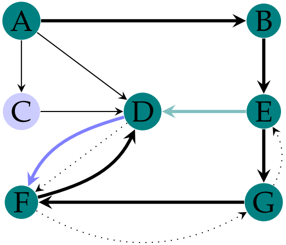
- The edges that we take to discover a new node are called the **discovery edges**
- The discovery edges form the DFS tree
- The other edges are called non-tree edges
- The edges to an ancestor in the DFS tree are called back edges
- The edges to a descendant node in the DFS tree are called forward edges
- The edges to a non-ancestor/non-descendant node in the BFS tree are called cross edges

DFS - demonstration, definitions



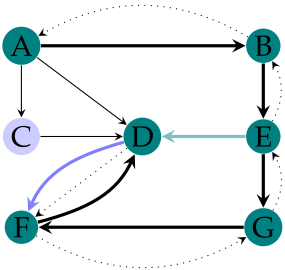
- The edges that we take to discover a new node are called the **discovery edges**
- The discovery edges form the DFS tree
- The other edges are called non-tree edges
- The edges to an ancestor in the DFS tree are called back edges
- The edges to a descendant node in the DFS tree are called forward edges
- The edges to a non-ancestor/non-descendant node in the BFS tree are called cross edges

DFS - demonstration, definitions



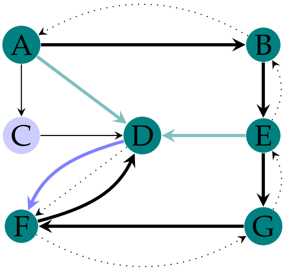
- The edges that we take to discover a new node are called the **discovery edges**
- The discovery edges form the DFS tree
- The other edges are called non-tree edges
- The edges to an ancestor in the DFS tree are called back edges
- The edges to a descendant node in the DFS tree are called forward edges
- The edges to a non-ancestor/non-descendant node in the BFS tree are called cross edges

DFS - demonstration, definitions



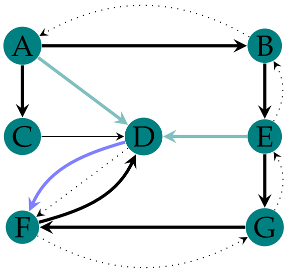
- The edges that we take to discover a new node are called the **discovery edges**
- The discovery edges form the DFS tree
- The other edges are called non-tree edges
- The edges to an ancestor in the DFS tree are called back edges
- The edges to a descendant node in the DFS tree are called forward edges
- The edges to a non-ancestor/non-descendant node in the BFS tree are called cross edges

DFS - demonstration, definitions



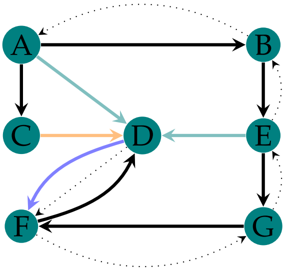
- The edges that we take to discover a new node are called the **discovery edges**
- The discovery edges form the DFS tree
- The other edges are called non-tree edges
- The edges to an ancestor in the DFS tree are called back edges
- The edges to a descendant node in the DFS tree are called forward edges
- The edges to a non-ancestor/non-descendant node in the BFS tree are called cross edges

DFS - demonstration, definitions



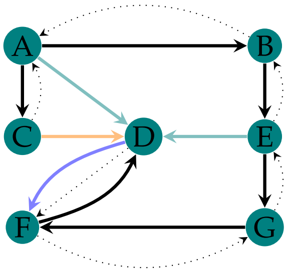
- The edges that we take to discover a new node are called the **discovery edges**
- The discovery edges form the DFS tree
- The other edges are called non-tree edges
- The edges to an ancestor in the DFS tree are called back edges
- The edges to a descendant node in the DFS tree are called forward edges
- The edges to a non-ancestor/non-descendant node in the BFS tree are called cross edges

DFS - demonstration, definitions



- The edges that we take to discover a new node are called the **discovery edges**
- The discovery edges form the DFS tree
- The other edges are called non-tree edges
- The edges to an ancestor in the DFS tree are called back edges
- The edges to a descendant node in the DFS tree are called forward edges
- The edges to a non-ancestor/non-descendant node in the BFS tree are called cross edges

DFS - demonstration, definitions

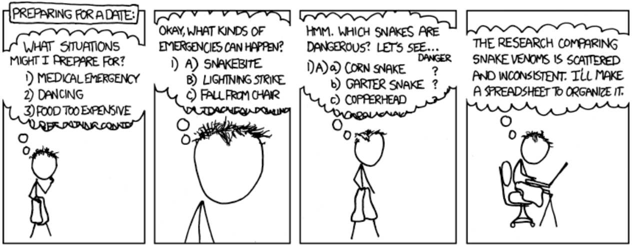


- The edges that we take to discover a new node are called the **discovery edges**
- The discovery edges form the DFS tree
- The other edges are called non-tree edges
- The edges to an ancestor in the DFS tree are called back edges
- The edges to a descendant node in the DFS tree are called forward edges
- The edges to a non-ancestor/non-descendant node in the BFS tree are called cross edges

Properties of DFS

- DFS visits all nodes in the connected component from the start node
- Discovery edges form a spanning tree of the connected component
- If a node v is connected to the start node, there is a path from the start node v in the DFS tree
- The DFS algorithm visits each node and checks each edge once (twice for undirected graphs)
- The complexity of the algorithm is $O(n + m)$ for n nodes and m edges

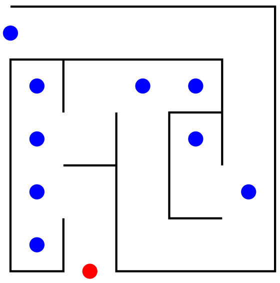
Dangers of DFS



https://xkcd.com/761/

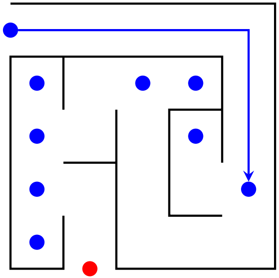
BFS - intuition

- A way to think about breadth-first search (BFS) is to explore all options in parallel
- In the maze, at every intersection send out people in all directions
- BFS divides the nodes into levels:
  - starting node at level 0
  - nodes directly accessible from start at level 1
  - ...



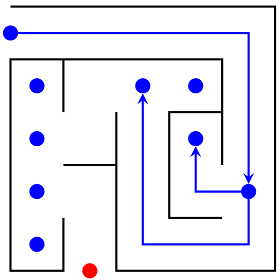
BFS - intuition

- A way to think about breadth-first search (BFS) is to explore all options in parallel
- In the maze, at every intersection send out people in all directions
- BFS divides the nodes into levels:
  - starting node at level 0
  - nodes directly accessible from start at level 1
  - ...



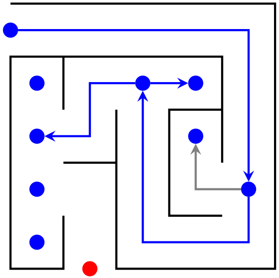
BFS - intuition

- A way to think about breadth-first search (BFS) is to explore all options in parallel
- In the maze, at every intersection send out people in all directions
- BFS divides the nodes into levels:
  - starting node at level 0
  - nodes directly accessible from start at level 1
  - ...



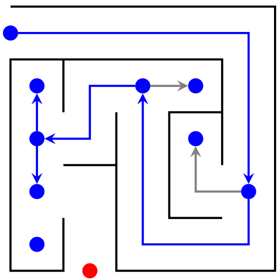
BFS - intuition

- A way to think about breadth-first search (BFS) is to explore all options in parallel
- In the maze, at every intersection send out people in all directions
- BFS divides the nodes into levels:
  - starting node at level 0
  - nodes directly accessible from start at level 1
  - ...



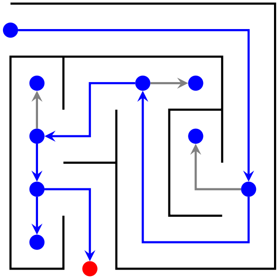
BFS - intuition

- A way to think about breadth-first search (BFS) is to explore all options in parallel
- In the maze, at every intersection send out people in all directions
- BFS divides the nodes into levels:
  - starting node at level 0
  - nodes directly accessible from start at level 1
  - ...



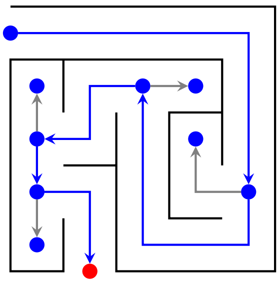
BFS - intuition

- A way to think about breadth-first search (BFS) is to explore all options in parallel
- In the maze, at every intersection send out people in all directions
- BFS divides the nodes into levels:
  - starting node at level 0
  - nodes directly accessible from start at level 1
  - ...



BFS - intuition

- A way to think about breadth-first search (BFS) is to explore all options in parallel
- In the maze, at every intersection send out people in all directions
- BFS divides the nodes into levels:
  - starting node at level 0
  - nodes directly accessible from start at level 1
  - ...

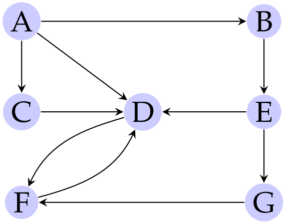


BFS - algorithm

```
def bfs(start):
  queue = [start]
  visited = {start: None}
  while queue:
    current = queue.pop(0)
    for node in current.neighbors():
      if node not in visited:
        visited[node] = current
        queue.append(node)
```

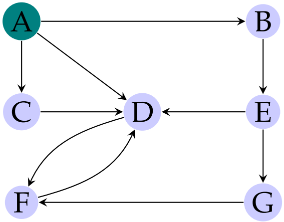
- Typically BFS is implemented with a queue
- The algorithm visits nodes closest to the start node first
- If you replace the queue with a stack, you get an iterative version of the DFS

BFS - demonstration



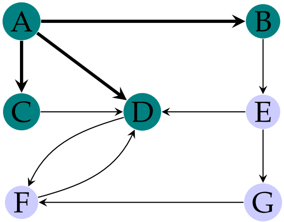
- Similar to DFS, the edges that we take to discover a new node are called the **discovery edges**
- The discovery edges form the BFS tree
- The other edges are called non-tree edges
- The edges to an ancestor in the BFS tree are called back edges
- The edges to a non-ancestor/non-descendant node in the BFS tree are called cross edges

BFS - demonstration



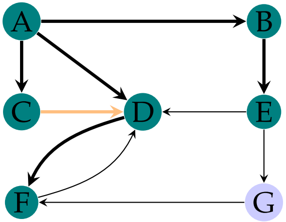
- Similar to DFS, the edges that we take to discover a new node are called the **discovery edges**
- The discovery edges form the BFS tree
- The other edges are called non-tree edges
- The edges to an ancestor in the BFS tree are called back edges
- The edges to a non-ancestor/non-descendant node in the BFS tree are called cross edges

BFS - demonstration



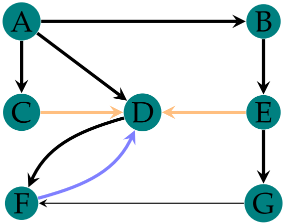
- Similar to DFS, the edges that we take to discover a new node are called the **discovery edges**
- The discovery edges form the BFS tree
- The other edges are called non-tree edges
- The edges to an ancestor in the BFS tree are called back edges
- The edges to a non-ancestor/non-descendant node in the BFS tree are called cross edges

BFS - demonstration



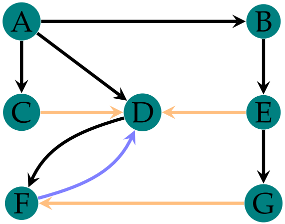
- Similar to DFS, the edges that we take to discover a new node are called the **discovery edges**
- The discovery edges form the BFS tree
- The other edges are called non-tree edges
- The edges to an ancestor in the BFS tree are called back edges
- The edges to a non-ancestor/non-descendant node in the BFS tree are called cross edges

BFS - demonstration



- Similar to DFS, the edges that we take to discover a new node are called the **discovery edges**
- The discovery edges form the BFS tree
- The other edges are called non-tree edges
- The edges to an ancestor in the BFS tree are called back edges
- The edges to a non-ancestor/non-descendant node in the BFS tree are called cross edges

BFS - demonstration



- Similar to DFS, the edges that we take to discover a new node are called the **discovery edges**
- The discovery edges form the BFS tree
- The other edges are called non-tree edges
- The edges to an ancestor in the BFS tree are called back edges
- The edges to a non-ancestor/non-descendant node in the BFS tree are called cross edges

Properties of BFS

- DFS visits all nodes in the connected component from the start node
- Discovery edges form a spanning tree of the connected component
- If a node v is reachable from the start node, the BFS finds the *shortest path* from the start node to v
- The BFS algorithm visits each node and checks each edge once
- The complexity of the algorithm is $O(n + m)$ for n nodes and m edges

Problems solved by graph traversals

- Finding a path between two nodes (if one exists)
- Testing whether G is connected
- Computing connected components of G
- Detecting cycles

Finding a path between two nodes

- Traverse the graph from the source node, record the *discovery edges*
- Start from the target node, trace the path back to the source
- With BFS, we get the shortest path
- Running time is the length of the path:

```
def find_path(source, target, visited):
  path = []
  if target in visited:
    path.append(target)
    current = target
    while current is not source:
      parent = visited[current]
      path.append(parent)
      current = parent
  return path.reverse()
```

Finding a path between two nodes

- Traverse the graph from the source node, record the *discovery edges*
- Start from the target node, trace the path back to the source
- With BFS, we get the shortest path
- Running time is the length of the path: $O(n)$

```
def find_path(source, target, visited):
  path = []
  if target in visited:
    path.append(target)
    current = target
    while current is not source:
      parent = visited[current]
      path.append(parent)
      current = parent
  return path.reverse()
```

Some other problems solved by graph traversal

- Is the graph connected?
  - Yes if the 'visited' nodes have the same length as the nodes of the graph
- Find the connected components
  - Run traversal multiple times, until all nodes are visited
- Is the graph cyclic?
  - A directed graph is cyclic if there is a back edge during graph traversal
  - A undirected graph is cyclic if a traversal finds any visited nodes (if there are back, forward or cross edges)

Summary

- Traversal is one of the basic operations in graphs
- Graph traversals already solve some interesting problems:
  - Find a path (shortest with BFS)
  - Test connectivity, find connected components
  - Find cycles
- Reading on graphs: Goodrich, Tamassia, and Goldwasser (2013, chapter 14)

Next:

- More graph algorithms: special problems on directed graphs, shortest paths

Acknowledgments, credits, references

Goodrich, Michael T., Roberto Tamassia, and Michael H. Goldwasser (2013). *Data Structures and Algorithms in Python*. John Wiley & Sons, Incorporated. ISBN: 9781118476734.

blank

blank

blank

blank