Graph Traversal nal Linguistics III

Çağrı Çöltekin ccoltekin@sfs.uni-tuebingen.de

Winter Semester 2023/24

DFS - intuition

Graph traversal

· Depth first search follows the same idea as exploring a labyrinth with a string and a chalk

* 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 the path connected?

Is the graph connected?

Is the graph cyclic?

Two main methods of traversals are breadth-first and depth-first

- 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

string Mark each visited node, backtrack

DFS - intuition

· Depth first search follows the s idea as exploring a labyrinth with a string and a chalk

· 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

(following the string) when hit a dead end

- 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 s 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



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



DPS - intuition

dead end

- 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





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
- Visit each intersection (node), while marking the path you took with the string



DFS - algorithm

· Depth-first search (DFS) is easy with def dfs(start, visited=Nome):
 if visited is Nome:
 visited = {start: Nome}
 for node in start.neighbors():
 if node not in visited:
 visited[node] = start

· DFS starts from a start node

· Marks each node it visits as visited (typically nut it in a set data structure) . Then take an arbitrary sumisited neighbor

and continue visiting the nodes reco · Algorithm terminates when backtracking leads to the start node with no unvisited nodes left

. 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

. The edges to a descendant node in the DFS tree are called forward edge

node in the BFS tree are called cross edge

DFS - demonstration, definitions



. The edges that we take to disc 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 . The edges to a descendant node in the DFS

tree are called forward edges

. The edges to a non-ancestor/non-descenda node in the BFS tree are called cross ed

DFS - demonstration, definitions

DFS - demonstration, definitions



. The edges that we take to dis 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

. 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 ede

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 an called

· 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 b

. The edges to a descendant node in the DFS tree are called forward edge The edges to a non-ancestor/non-descendary node in the BPS 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 DPS tree

· The other edges are called non-tree edges . The edges to an ancestor in the DFS tree an called

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 ed

. 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

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

DPS - demonstration, definitions

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 BPS tree are called con-



. 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 edge

. The edges to a non-ancestor/non-descendant

node in the BPS tree are called con-

DFS - demonstration, definitions



 The edges that we take to disco 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 . The edges to a descendant node in the DFS tree are called forward edges

The edges to a non-ancestor/non-descend node in the BFS tree are called cross edges



DFS - demonstration, definitions

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 ed

. The edges to a descendant node in the DFS tree are called forward edge.

. The edges that we take to disc

node in the BPS tree are called cross



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
- . The edges to a descendant node in the DFS tree are called forward edges
- The edges to a non-ancestor/non-descenda node in the BFS tree are called cross edges.

- . 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 ed
- . The edges to a descendant node in the DFS tree are called forward edges
- The edges to a non-ancestor/r
- node in the BFS tree are called cross edge

DFS - demonstration, definitions



- . The edges that we take to disco 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
- . The edges to a descendant node in the DFS
- tree are called forward edges . The edges to a non-ancestor/non-descenda
- node in the BPS tree are called cross eds

DFS - demonstration, definitions

DFS - demonstration, definitions



- . The edges that we take to dis 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-descenda node in the BFS tree are called cross edge

Properties of DFS

- · DFS visits all nodes in the concted comp
- 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 check each edge once (twice for
- undirected graphs
- \bullet The complexity of the algorithm is O(n+m) for n nodes and m edges

Dangers of DFS

Part of

Ā 1. 90% or 1

Smusel on the control of the control VAN SOURCE PART SO

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
 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-firs 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 at level 1





BFS - intuition

- 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
 at level 1



BFS - intuition

- . A way to think about breadth-first search (BPS) is to explore all options in parallel In the maze, at every inters
- 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 intersec
- send out people in all directions

 BFS divides the nodes into levels:
 - starting node at level 0
 nodes directly accessible
 at level 1



BFS - algorithm

- def bfs(start)
- of 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 firs
 - If you replace the queue with a stack, you get an iterative version of the DFS

BFS - demonstration



- · Similar to DPS, the edges that we take to discover a new node are called the discov edges
- The discovery edges form the BFS tree · The other edges are called non-tree edges
- . The edges to an ancestor in the BPS tree are
- called back edges . The edges to a non-ancestor/non-descendant
- node in the BFS tree are called cross edg

BPS - demonstration



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

BFS - demonstration



- · Similar to DPS, 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 BPS tree are
- called back edges
- . The edges to a non-ancestor/non-descendant ode in the BFS tree are called

BFS - demonstration



- . Similar to DPS, 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 BPS tree are called back edges
- . The edges to a non-ancestor/non-descendant ode in the BPS tree are called

BFS - demonstration



- Similar to DPS, the edges that we take to discover a new node are called the discovery edges
 - . The discovery edges form the BPS 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-descendan node in the BFS tree are called cross ex

BFS - demonstration



- Similar to DPS, 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 BPS tree are
- called back edges . The edges to a non-ancestor/non-descendant node in the BFS tree are called cross

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 check each edge once

* The complexity of the algorithm is O(n + m) for n nodes and m edges

- · Finding a path bet veen two n des (if one exi Testing whether G is connected

Some other problems solved by graph traversal

 Computing connected components of G Detecting cycles

Problems solved by graph traversals

. Is the graph connected?

. Find the connected components

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
- · With BFS, we get the
 - shortest path
- * Running time is the length of the path: O(n)
- def find_path(source, target, visited):
 path = path = []
 if target in visited:
 path.append(target)
 current = target
 - while current is not source parent = visited[current]
 - path.append(parent) current = parent urn path.reverse()
- Run traversal multiple times, until all nodes are visited - A graph is cyclic if there is a back edge during graph traversal
- Is the graph cyclic?

Yes if the 'visited' nodes have the same length as the nodes of the graph

Summary	Acknowledgments, credits, references
Traversal is one of the basic operations in graphs Graph traversals already solve some interesting problems: First a yeal to identify a solve the contraction of the problems: First cycles First cycles Reading on graphs: Coodrich, Tamassia, and Goldwasser (2013, chapter 14) Nest: More graph algorithms: special problems on directed graphs, shortest paths	☐ Goodrich, Michael T., Roberto Tamassia, and Michael H. Goldwasser (2013). Data Structures and Algorithms in Python. John Wiley & Sons, Incorporated. sase: 9781185(76734.
_C.C. (Clirkin, 167 / Dimensity of Tallenges State of State (State of State	C Caldian SB / Decemby of Stringers Water formels SED/SL A1
C Cilibido, 187 (Decreoly of Rillinger Windowshort Stringer Windowshort Stringer	C. Cilolinia, 180 Discountry of Difference 2021/20 A.1
C Cilidas Mr J Monady of Tillingus State S	C Cilibin. 30 Dannah of Singer Visia Income 2013 Ad.