

Algorithms and Datastructures

Graphs, Depth-/Breadth-first Search, Graph-Connectivity

Albert-Ludwigs-Universität Freiburg



**UNI
FREIBURG**

Prof. Dr. Rolf Backofen

Bioinformatics Group / Department of Computer Science
Algorithms and Datastructures, January 2017

Feedback

Exercises

Lecture

Graphs

Introduction

Implementation

Application example

Feedback

Exercises

Lecture

Graphs

Introduction

Implementation

Application example

The upcoming exercise sheet 12 and 13 will be merged together (finding largest connected component + Dijkstra)

Some people were asking for more solution sheets for the exercises

We are working on it.

Code in the lecture will be a little bit different from exercise sheet.

One person asked for additional explanations regarding proofs.

Feedback

Exercises

Lecture

Graphs

Introduction

Implementation

Application example

Feedback

Exercises

Lecture

Graphs

Introduction

Implementation

Application example



Graphs - Overview:

Graphs - Overview:

- Besides arrays, lists and trees the most common datastructure
(Trees are a special type of graph)

Graphs - Overview:

- Besides arrays, lists and trees the most common datastructure
(Trees are a special type of graph)
- Representation of graphs in the computer

Graphs - Overview:

- Besides arrays, lists and trees the most common datastructure
(Trees are a special type of graph)
- Representation of graphs in the computer
- Breadth first search (BFS)

Graphs - Overview:

- Besides arrays, lists and trees the most common datastructure
(Trees are a special type of graph)
- Representation of graphs in the computer
- Breadth first search (BFS)
- Depth first search (DFS)

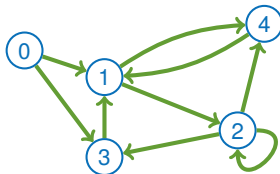
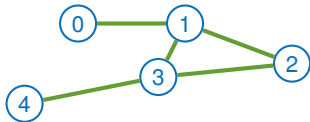
Graphs - Overview:

- Besides arrays, lists and trees the most common datastructure
(Trees are a special type of graph)
- Representation of graphs in the computer
- Breadth first search (BFS)
- Depth first search (DFS)
- Connected components of a graph

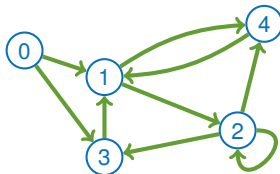
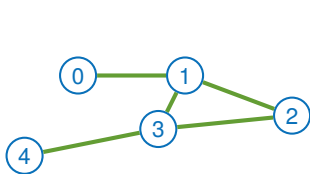


Terminology:

Terminology:



Terminology:



- Each Graph $G = (V, E)$ consists of:

Terminology:



- Each Graph $G = (V, E)$ consists of:
 - A set of vertices (nodes) $V = \{v_1, v_2, \dots\}$

Terminology:



- Each Graph $G = (V, E)$ consists of:
 - A set of vertices (nodes) $V = \{v_1, v_2, \dots\}$
 - A set of edges (arcs) $E = \{e_1, e_2, \dots\}$

Terminology:



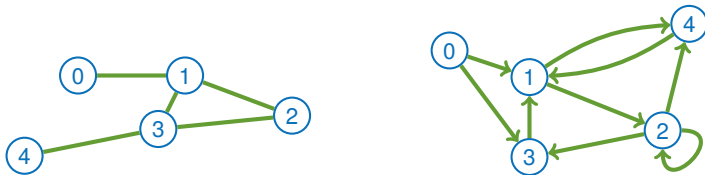
- Each Graph $G = (V, E)$ consists of:
 - A set of vertices (nodes) $V = \{v_1, v_2, \dots\}$
 - A set of edges (arcs) $E = \{e_1, e_2, \dots\}$
- Each edge connects two vertices ($u, v \in V$)

Terminology:



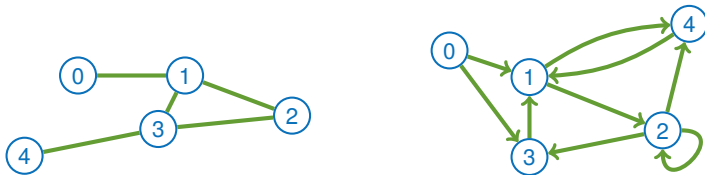
- Each Graph $G = (V, E)$ consists of:
 - A set of vertices (nodes) $V = \{v_1, v_2, \dots\}$
 - A set of edges (arcs) $E = \{e_1, e_2, \dots\}$
- Each edge connects two vertices ($u, v \in V$)
 - Undirected edge: $e = \{u, v\}$ (set)

Terminology:



- Each Graph $G = (V, E)$ consists of:
 - A set of vertices (nodes) $V = \{v_1, v_2, \dots\}$
 - A set of edges (arcs) $E = \{e_1, e_2, \dots\}$
- Each edge connects two vertices ($u, v \in V$)
 - Undirected edge: $e = \{u, v\}$ (set)
 - Directed edge: $e = (u, v)$ (tuple)

Terminology:

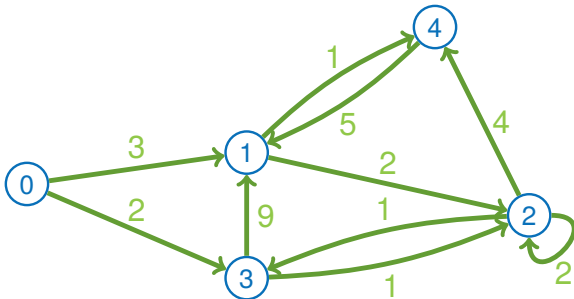


- Each Graph $G = (V, E)$ consists of:
 - A set of vertices (nodes) $V = \{v_1, v_2, \dots\}$
 - A set of edges (arcs) $E = \{e_1, e_2, \dots\}$
- Each edge connects two vertices ($u, v \in V$)
 - Undirected edge: $e = \{u, v\}$ (set)
 - Directed edge: $e = (u, v)$ (tuple)
- Self-loops are also possible: $e = (u, u)$ or $e = \{u, u\}$



Weighted graph:

Weighted graph:

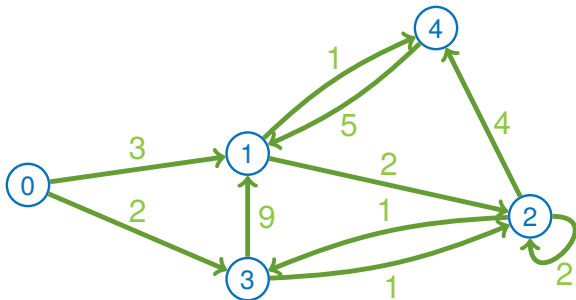


Weighted graph:



- Each edge is marked with a real number named **weight**

Weighted graph:



- Each edge is marked with a real number named **weight**
- The **weight** is also named **length** or **cost** of the edge depending on the application

Example: Road network

Example: Road network

- Intersections:
vertices

Example: Road network

- Intersections:
vertices
- Roads: edges

Example: Road network

- Intersections:
vertices
- Roads: edges
- Travel time:
costs of the edges

Example: Road network

- Intersections: **vertices**
- Roads: **edges**
- Travel time: **costs of the edges**



Figure: Map of Freiburg © OpenStreetMap

Feedback

Exercises

Lecture

Graphs

Introduction

Implementation

Application example



How to represent this graph computationally?



How to represent this graph computationally?

- Two classic variants

How to represent this graph computationally?

- Two classic variants

1 Adjacency matrix with space consumption $\Theta(|V|^2)$

How to represent this graph computationally?

- Two classic variants

1 Adjacency matrix with space consumption $\Theta(|V|^2)$



Figure: Weighted graph with
 $|V| = 4, |E| = 6$

How to represent this graph computationally?

- Two classic variants

1 **Adjacency matrix** with space consumption $\Theta(|V|^2)$

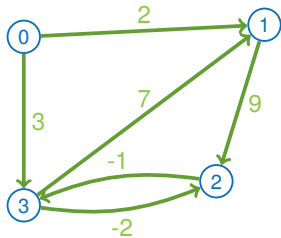


Figure: Weighted graph with
 $|V| = 4$, $|E| = 6$

		end-vertex			
		0	1	2	3
start-vertex	0		2		3
	1			9	
	2				-1
	3		7	-2	

Figure: Adjacency matrix



How to represent this graph computationally?



How to represent this graph computationally?

- Two classic variants

2 Adjacency list / fields with space consumption $\Theta(|V| + |E|)$



How to represent this graph computationally?

- Two classic variants
 - 2 Adjacency list / fields with space consumption $\Theta(|V| + |E|)$
 - Each list item stores the target vertex and the cost of the edge

How to represent this graph computationally?

- Two classic variants
 - 2 Adjacency list / fields with space consumption $\Theta(|V| + |E|)$
 - Each list item stores the target vertex and the cost of the edge

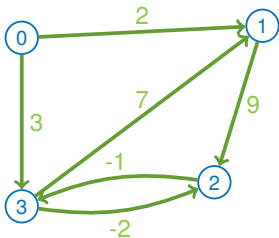


Figure: Weighted graph with
 $|V| = 4, |E| = 6$

How to represent this graph computationally?

- Two classic variants

2 Adjacency list / fields with space consumption $\Theta(|V| + |E|)$

- Each list item stores the **target vertex** and the **cost** of the edge



Figure: Weighted graph with
 $|V| = 4, |E| = 6$

start-vertex	0	1, 2	3, 3
	1	2, 9	
	2	3, -1	
	3	1, 7	2, -2

Figure: Adjacency list



Graph: Arrangement

Graph: Arrangement

- Graph is fully defined through the [adjacency matrix / list](#)

Graph: Arrangement

- Graph is fully defined through the [adjacency matrix / list](#)
- The arrangement is not relevant for visualisation of the graph

Graph: Arrangement

- Graph is fully defined through the **adjacency matrix / list**
- The arrangement is not relevant for visualisation of the graph



Figure: Weighted graph with
 $|V| = 4$, $|E| = 6$

Graph: Arrangement

- Graph is fully defined through the **adjacency matrix / list**
- The arrangement is not relevant for visualisation of the graph



Figure: Weighted graph with
 $|V| = 4$, $|E| = 6$



Figure: Same graph ordered by number - outer planar graph

```
class Graph:
    def __init__(self):
        self.vertices = []
        self.edges = []

    def addVertice(self, vert):
        self.vertices.append(vert)

    def addEdge(self, fromVert, toVert):
        self.edges.append((fromVert, toVert))

    ...
```


...

```
def toString(self):  
    return '{'  
        + ', '.join( \  
            [str(len(self.vertices)), \  
              str(len(self.edges))] \  
        + ["(%s, %s)" % tup \  
          for tup in self.edges]) \  
        + '}'
```



Degree of a vertex: Directed graph: $G = (V, E)$

Degree of a vertex: Directed graph: $G = (V, E)$

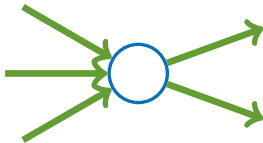


Figure: Vertex with in- / outdegree of 3 / 2

Degree of a vertex: Directed graph: $G = (V, E)$

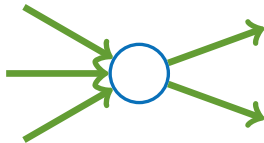


Figure: Vertex with in- / outdegree of 3 / 2

- **Indegree** of a vertex u is the number of **edge heads** adjacent to the vertex

$$\deg^+(u) = |\{(v, u) : (v, u) \in E\}|$$

Degree of a vertex: Directed graph: $G = (V, E)$

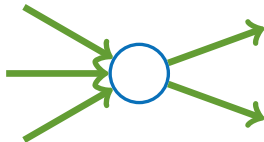


Figure: Vertex with in- / outdegree of 3 / 2

- **Indegree** of a vertex u is the number of **edge heads** adjacent to the vertex

$$\deg^+(u) = |\{(v, u) : (v, u) \in E\}|$$

- **Outdegree** of a vertex u is the number of **edge tails** adjacent to the vertex

$$\deg^-(u) = |\{(u, v) : (u, v) \in E\}|$$



Degree of a vertex: Undirected graph: $G = (V, E)$

Degree of a vertex: Undirected graph: $G = (V, E)$

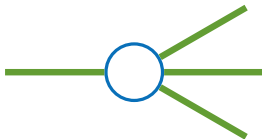


Figure: Vertex with degree of 4

Degree of a vertex: Undirected graph: $G = (V, E)$



Figure: Vertex with degree of 4

- **Degree** of a vertex u is the number of **vertices** adjacent to the vertex

$$\deg(u) = |\{\{v, u\} : \{v, u\} \in E\}|$$



Paths in a graph: $G = (V, E)$

Paths in a graph: $G = (V, E)$

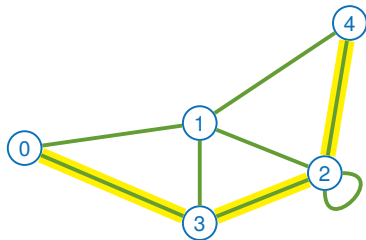


Figure: Undirected path of length 3
 $P = (0, 3, 2, 4)$

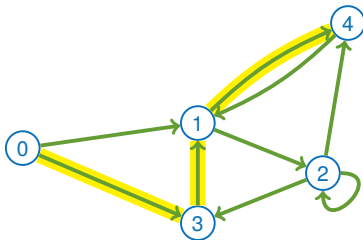


Figure: Directed path of length 3
 $P = (0, 3, 1, 4)$

Paths in a graph: $G = (V, E)$

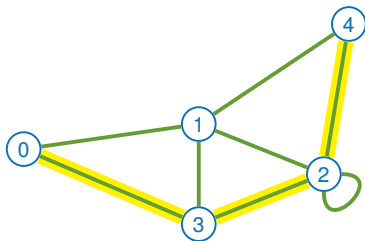


Figure: Undirected path of length 3
 $P = (0, 3, 2, 4)$



Figure: Directed path of length 3
 $P = (0, 3, 1, 4)$

- A path of G is a sequence of edges $u_1, u_2, \dots, u_i \in V$ with

Paths in a graph: $G = (V, E)$

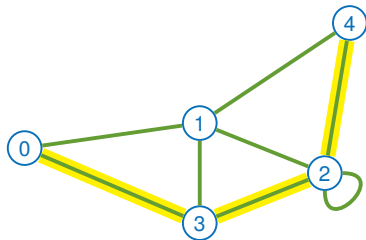


Figure: Undirected path of length 3
 $P = (0, 3, 2, 4)$



Figure: Directed path of length 3
 $P = (0, 3, 1, 4)$

- A path of G is a sequence of edges $u_1, u_2, \dots, u_i \in V$ with
 - Undirected graph: $\{u_1, u_2\}, \{u_2, u_3\}, \dots, \{u_{i-1}, u_i\} \in E$
 - Directed graph: $(u_1, u_2), (u_2, u_3), \dots, (u_{i-1}, u_i) \in E$

Paths in a graph: $G = (V, E)$

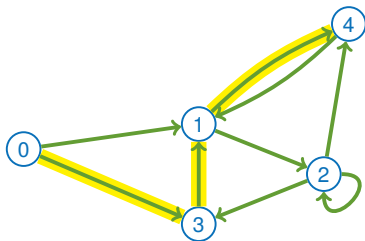


Figure: Directed path of length 3
 $P = (0, 3, 1, 4)$

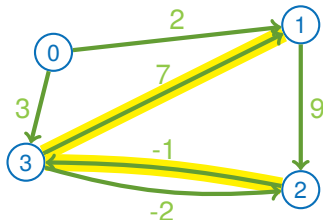


Figure: Weighted path with cost 6
 $P = (2, 3, 1)$

Paths in a graph: $G = (V, E)$



Figure: Directed path of length 3
 $P = (0, 3, 1, 4)$



Figure: Weighted path with cost 6
 $P = (2, 3, 1)$

Paths in a graph: $G = (V, E)$



Figure: Directed path of length 3
 $P = (0, 3, 1, 4)$

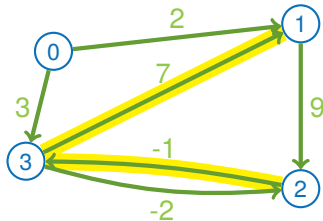


Figure: Weighted path with cost 6
 $P = (2, 3, 1)$

Paths in a graph: $G = (V, E)$

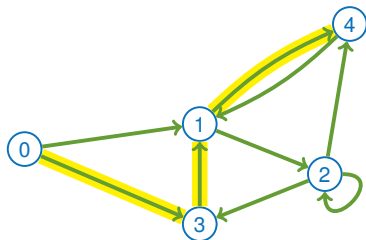


Figure: Directed path of length 3
 $P = (0, 3, 1, 4)$



Figure: Weighted path with cost 6
 $P = (2, 3, 1)$

- The length of a path is: (also costs of a path)

Paths in a graph: $G = (V, E)$



Figure: Directed path of length 3
 $P = (0, 3, 1, 4)$



Figure: Weighted path with cost 6
 $P = (2, 3, 1)$

- The **length of a path** is: (also costs of a path)
 - Without weights: **number of edges** taken

Paths in a graph: $G = (V, E)$



Figure: Directed path of length 3
 $P = (0, 3, 1, 4)$



Figure: Weighted path with cost 6
 $P = (2, 3, 1)$

- The **length of a path** is: (also costs of a path)
 - Without weights: **number of edges** taken
 - With weights: **sum of weights of edges** taken



Shortest path in a graph: $G = (V, E)$

Shortest path in a graph: $G = (V, E)$

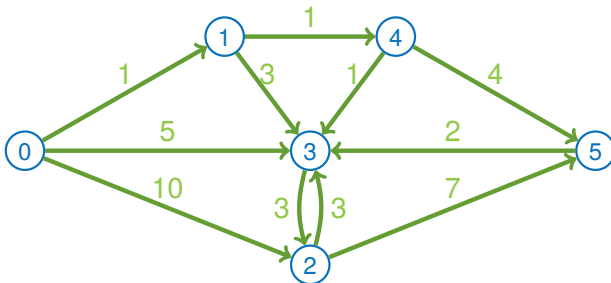


Figure: Shortest path from 0 to 2 with cost / distance $d(0,2) = ?$

Shortest path in a graph: $G = (V, E)$



Figure: Shortest path from 0 to 2 with cost / distance $d(0,2) = ?$

- The shortest path between two vertices u, v is the path $P = (u, \dots, v)$ with the shortest length $d(u, v)$ or lowest costs

Shortest path in a graph: $G = (V, E)$



Figure: Shortest path from 0 to 2 with cost / distance $d(0,2) = 6$
 $P = (0, 1, 4, 3, 2)$

- The shortest path between two vertices u, v is the path $P = (u, \dots, v)$ with the shortest length $d(u, v)$ or lowest costs



Diameter of a graph: $G = (V, E)$

Diameter of a graph: $G = (V, E)$

$$d = \max_{u,v \in V} d(u,v)$$



Figure: Diameter of graph is $d = ?$

Diameter of a graph: $G = (V, E)$

$$d = \max_{u,v \in V} d(u,v)$$



Figure: Diameter of graph is $d = ?$

- The **diameter** of a graph is the length / the costs of the longest shortest path

Diameter of a graph: $G = (V, E)$

$$d = \max_{u,v \in V} d(u,v)$$



Figure: Diameter of graph is $d = 10$, $P = (3, 2, 5)$

- The **diameter** of a graph is the length / the costs of the longest shortest path



Connected components: $G = (V, E)$

Connected components: $G = (V, E)$

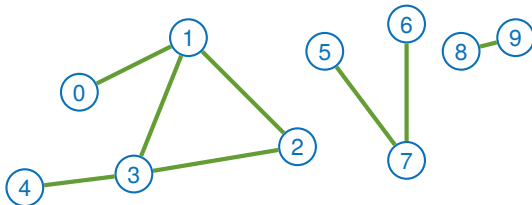


Figure: Three connected components

- Undirected graph:

Connected components: $G = (V, E)$



Figure: Three connected components

- Undirected graph:
 - All connected components are a partition of V

$$V = V_1 \cup \dots \cup V_k$$

Connected components: $G = (V, E)$



Figure: Three connected components

- Undirected graph:
 - All connected components are a partition of V

$$V = V_1 \cup \dots \cup V_k$$

- Two vertices u, v are in the same connected component if a path between u and v exists

Connected components: $G = (V, E)$

Connected components: $G = (V, E)$

- Directed graph:

Connected components: $G = (V, E)$

- Directed graph:
 - Named **strongly connected components**

Connected components: $G = (V, E)$

- Directed graph:
 - Named **strongly connected components**
 - Direction of edge has to be regarded

Connected components: $G = (V, E)$

- Directed graph:
 - Named **strongly connected components**
 - Direction of edge has to be regarded
 - Not part of this lecture

Graph Exploration: (Informal definition)

Graph Exploration: (Informal definition)

- Let $G = (V, E)$ be a graph and $s \in V$ a start vertex

Graph Exploration: (Informal definition)

- Let $G = (V, E)$ be a graph and $s \in V$ a start vertex
- We visit each reachable vertex connected to s

Graph Exploration: (Informal definition)

- Let $G = (V, E)$ be a graph and $s \in V$ a start vertex
- We visit each reachable vertex connected to s
- **Breadth-first search**: in sequence of the smallest distance to s

Graph Exploration: (Informal definition)

- Let $G = (V, E)$ be a graph and $s \in V$ a start vertex
- We visit each reachable vertex connected to s
- **Breadth-first search**: in sequence of the smallest distance to s
- **Depth-first search**: in sequence of the largest distance to s

Graph Exploration: (Informal definition)

- Let $G = (V, E)$ be a graph and $s \in V$ a start vertex
- We visit each reachable vertex connected to s
- **Breadth-first search**: in sequence of the smallest distance to s
- **Depth-first search**: in sequence of the largest distance to s
- Not a problem on its own but is often used as subroutine of other algorithms



Idea:



Idea:

- 1 We start with all vertices unmarked and **mark visited vertices**

Idea:

- 1 We start with all vertices unmarked and mark visited vertices
- 2 Mark the start vertex s (level 0)

Idea:

- 1 We start with all vertices unmarked and **mark visited vertices**
- 2 Mark the start vertex **s** (**level 0**)
- 3 Mark all unmarked **connected vertices** (**level 1**)

Idea:

- 1 We start with all vertices unmarked and **mark visited vertices**
- 2 Mark the start vertex **s** (**level 0**)
- 3 Mark all unmarked **connected vertices** (**level 1**)
- 4 Mark all unmarked **vertices connected** to a **level 1**-vertex (**level 2**)

Idea:

- 1 We start with all vertices unmarked and **mark visited vertices**
- 2 Mark the start vertex **s** (**level 0**)
- 3 Mark all unmarked **connected vertices** (**level 1**)
- 4 Mark all unmarked **vertices connected** to a **level 1**-vertex (**level 2**)
- 5 Iteratively mark reachable vertices for all levels

Idea:

- 1 We start with all vertices unmarked and **mark visited vertices**
- 2 Mark the start vertex **s (level 0)**
- 3 Mark all unmarked **connected vertices (level 1)**
- 4 Mark all unmarked **vertices connected** to a **level 1-vertex (level 2)**
- 5 Iteratively mark reachable vertices for all levels
- 6 All connected nodes are now marked and in the same **connected component** as the start vertex **s**

Graphs

Connected Components - Breadth-First Search



**UNI
FREIBURG**

- The marked vertices create a “spanning tree” containing all reachable nodes

- The marked vertices create a “spanning tree” containing all reachable nodes



Figure: spanning tree of a breadth-first search

- The marked vertices create a “spanning tree” containing all reachable nodes

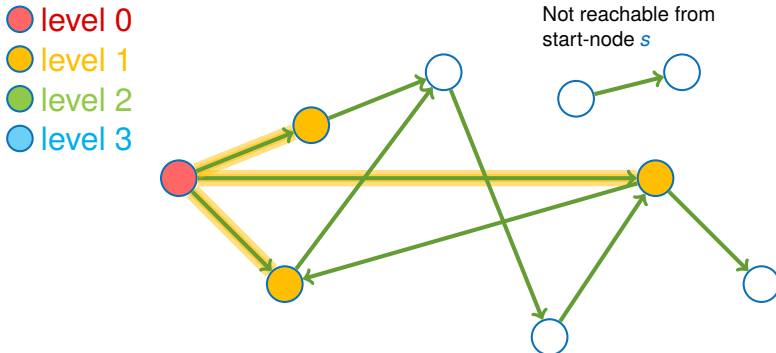


Figure: spanning tree of a breadth-first search

- The marked vertices create a “spanning tree” containing all reachable nodes

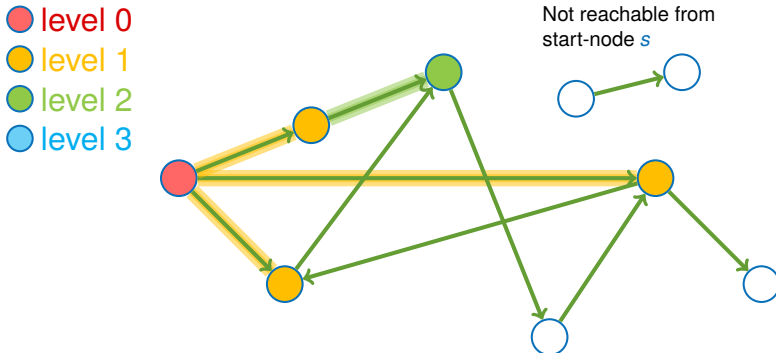


Figure: spanning tree of a breadth-first search

- The marked vertices create a “spanning tree” containing all reachable nodes

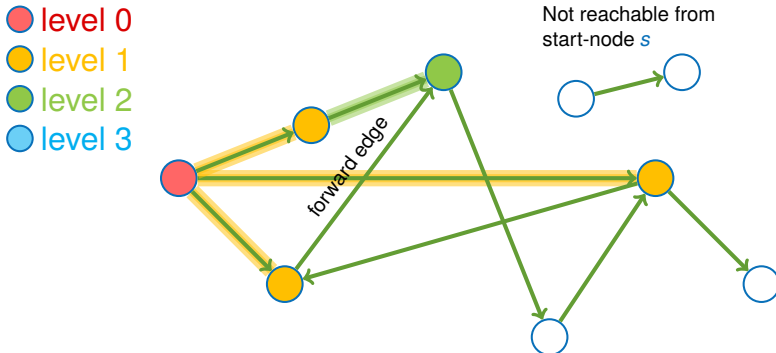


Figure: spanning tree of a breadth-first search

- 



- The marked vertices create a “spanning tree” containing all reachable nodes

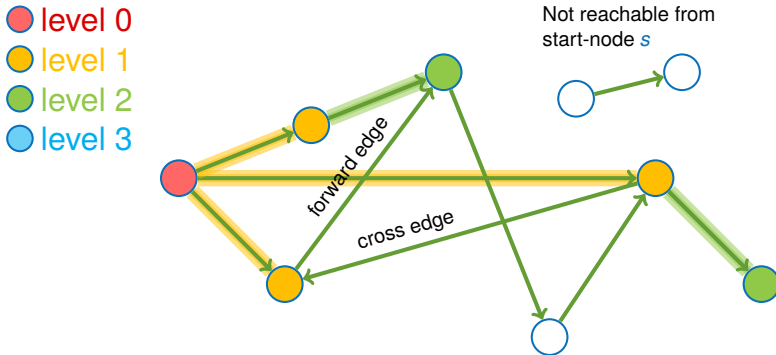


Figure: spanning tree of a breadth-first search

- The marked vertices create a “spanning tree” containing all reachable nodes



Figure: spanning tree of a breadth-first search

- The marked vertices create a “spanning tree” containing all reachable nodes



Figure: spanning tree of a breadth-first search



Idea:

Idea:

- 1 We start with all vertices unmarked and **mark visited vertices**

Idea:

- 1 We start with all vertices unmarked and **mark visited vertices**
- 2 Mark the start vertex **s**

Idea:

- 1 We start with all vertices unmarked and **mark visited vertices**
- 2 Mark the start vertex **s**
- 3 Pick an unmarked **connected vertex** and start a **recursive depth-first search** with the vertex as start vertex
(continue on step 2)

Idea:

- 1 We start with all vertices unmarked and **mark visited vertices**
- 2 Mark the start vertex **s**
- 3 Pick an unmarked **connected vertex** and start a **recursive depth-first search** with the vertex as start vertex (continue on step 2)
- 4 If no unmarked connected vertex exists go one vertex back (reduce the recursion level by one)



Depth-first search:



Depth-first search:

- Search starts with **long paths** (searching with depth)

Depth-first search:

- Search starts with **long paths** (searching with depth)
- Marks like **breadth-first search** all connected vertices

Depth-first search:

- Search starts with **long paths** (searching with depth)
- Marks like **breadth-first search** all connected vertices
- If the graph is acyclic we get a **topological sorting**

Depth-first search:

- Search starts with **long paths** (searching with depth)
- Marks like **breadth-first search** all connected vertices
- If the graph is acyclic we get a **topological sorting**
 - Each newly visited vertex gets marked by an increasing number

Depth-first search:

- Search starts with **long paths** (searching with depth)
- Marks like **breadth-first search** all connected vertices
- If the graph is acyclic we get a **topological sorting**
 - Each newly visited vertex gets marked by an increasing number
 - The numbers increase with path from the start vertex

Graphs

Connected Components - Depth-First Search



**UNI
FREIBURG**



- The marked vertices create a different spanning tree containing all reachable nodes

- The marked vertices create a different spanning tree containing all reachable nodes

● start-node

● path 1

● path 2

● path 3



Figure: spanning tree of a depth-first search

- The marked vertices create a different spanning tree containing all reachable nodes

● start-node

● path 1

● path 2

● path 3



Figure: spanning tree of a depth-first search

- The marked vertices create a different spanning tree containing all reachable nodes

● start-node

● path 1

● path 2

● path 3



Figure: spanning tree of a depth-first search

- The marked vertices create a different spanning tree containing all reachable nodes

● start-node

● path 1

● path 2

● path 3

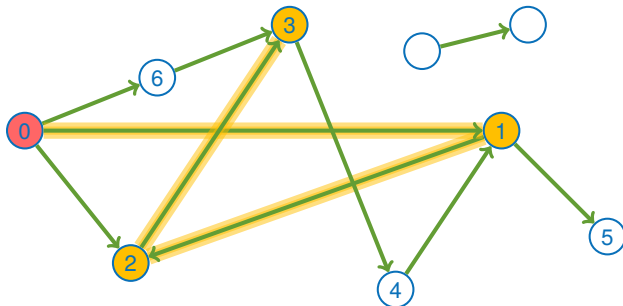


Figure: spanning tree of a depth-first search

- The marked vertices create a different spanning tree containing all reachable nodes

● start-node

● path 1

● path 2

● path 3

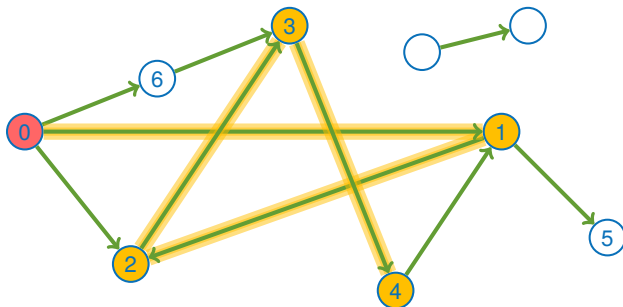


Figure: spanning tree of a depth-first search

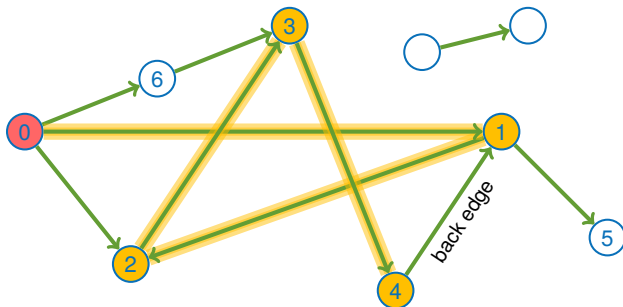
- The marked vertices create a different spanning tree containing all reachable nodes

● start-node

● path 1

● path 2

● path 3



- The marked vertices create a different spanning tree containing all reachable nodes

● start-node

● path 1

● path 2

● path 3

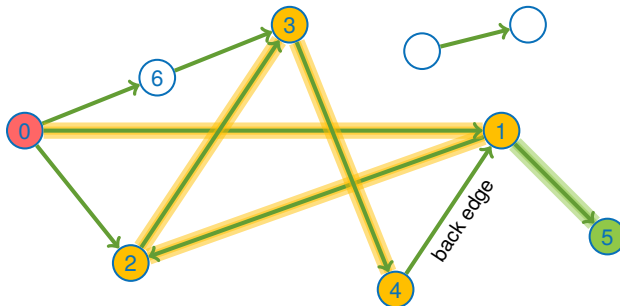


Figure: spanning tree of a depth-first search

Connected Components - Depth-First Search

- The marked vertices create a different spanning tree containing all reachable nodes

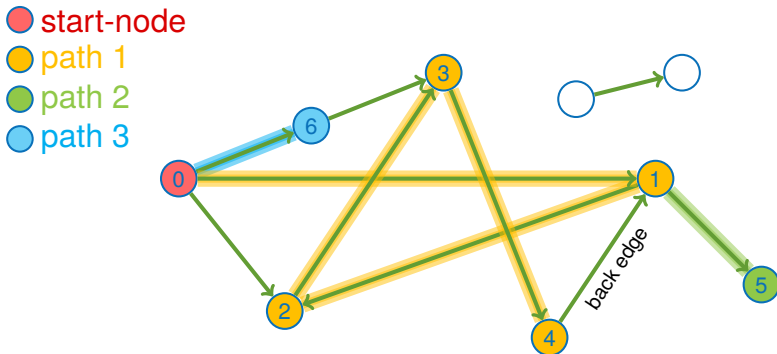


Figure: spanning tree of a depth-first search

- The marked vertices create a different spanning tree containing all reachable nodes

● start-node

● path 1

● path 2

● path 3

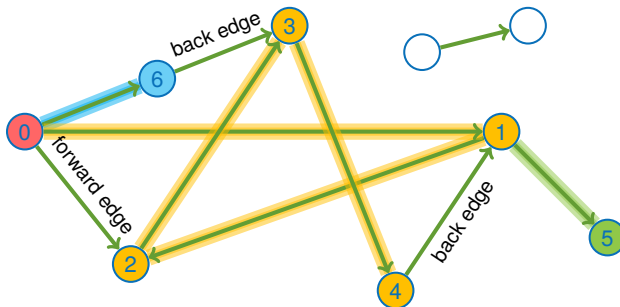


Figure: spanning tree of a depth-first search

- The marked vertices create a different spanning tree containing all reachable nodes

● start-node

● path 1

● path 2

● path 3



Figure: spanning tree of a depth-first search

Graphs

Why is this called Breadth - and Depth First Search?



**UNI
FREIBURG**



Runtime complexity:

Runtime complexity:

- Constant costs for each visited vertex and edge

Runtime complexity:

- Constant costs for each visited vertex and edge
- We get a runtime complexity of $\Theta(|V'| + |E'|)$

Runtime complexity:

- Constant costs for each visited vertex and edge
- We get a runtime complexity of $\Theta(|V'| + |E'|)$
- Let V' and E' be the reachable vertices and edges

Runtime complexity:

- Constant costs for each visited vertex and edge
- We get a runtime complexity of $\Theta(|V'| + |E'|)$
- Let V' and E' be the reachable vertices and edges
- All vertices of V' are in the same connected component as our start vertex s

Runtime complexity:

- Constant costs for each visited vertex and edge
- We get a runtime complexity of $\Theta(|V'| + |E'|)$
- Let V' and E' be the reachable vertices and edges
- All vertices of V' are in the same connected component as our start vertex s
- This can only be improved by a constant factor

Feedback

Exercises

Lecture

Graphs

Introduction

Implementation

Application example

Application example

Image processing



**UNI
FREIBURG**

Application example

Image processing



- Connected component labeling

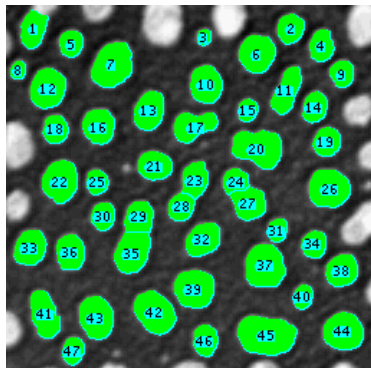
Application example

Image processing



- Connected component labeling
- Counting of objects in an image

- Connected component labeling
- Counting of objects in an image



What's object, what's background?



Convert to black white using threshold:

value = 255 if value > 100 else 0





Interpret image as graph:

Interpret image as graph:

- Each white pixel is a node

Interpret image as graph:

- Each white pixel is a node
- Edges between adjacent pixels (normally 4 or 8 neighbors)

Interpret image as graph:

- Each white pixel is a node
- Edges between adjacent pixels (normally 4 or 8 neighbors)
- Edges are not saved externally, algorithm works directly on array

Interpret image as graph:

- Each white pixel is a node
- Edges between adjacent pixels (normally 4 or 8 neighbors)
- Edges are not saved externally, algorithm works directly on array
- Breadth- / depth-first search find all connected components (particles)



Find connected components:

Find connected components:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
35	255	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
36	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
37	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
38	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
39	0	0	0	0	0	0	0	0	255	255	255	0	0	0	0	0	0	0	0
40	0	0	0	0	0	0	255	255	255	255	255	255	0	0	0	0	0	0	0
41	0	0	0	0	0	255	255	255	255	255	255	255	255	255	0	0	0	0	0
42	0	0	0	0	255	255	255	255	255	255	255	255	255	255	0	0	0	0	0
43	0	0	0	0	255	255	255	255	255	255	255	255	255	255	0	0	0	0	0
44	0	0	0	255	255	255	255	255	255	255	255	255	255	255	0	0	0	0	0
45	0	0	0	255	255	255	255	255	255	255	255	255	255	255	0	0	0	0	0
46	0	0	0	255	255	255	255	255	255	255	255	255	255	255	0	0	0	0	0
47	0	0	0	255	255	255	255	255	255	255	255	255	255	255	0	0	0	0	0
48	0	0	0	255	255	255	255	255	255	255	255	255	255	255	0	0	0	0	0
49	0	0	0	255	255	255	255	255	255	255	255	255	255	255	0	0	0	0	0
50	0	0	0	255	255	255	255	255	255	255	255	255	255	255	0	0	0	0	0
51	0	0	0	0	255	255	255	255	255	255	255	255	255	0	0	0	0	0	0
52	0	0	0	0	0	255	255	255	255	255	255	255	0	0	0	0	0	0	0
53	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	255
54	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	255
55	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	255	255

Find connected components:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
35	255	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
36	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
37	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
38	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
39	0	0	0	0	0	0	0	0	255	255	255	0	0	0	0	0	0	0	0
40	0	0	0	0	0	0	255	255	255	255	255	255	0	0	0	0	0	0	0
41	0	0	0	0	0	255	255	255	255	255	255	255	255	255	0	0	0	0	0
42	0	0	0	0	255	255	255	255	255	255	255	255	255	255	0	0	0	0	0
43	0	0	0	0	255	255	255	255	255	255	255	255	255	255	0	0	0	0	0
44	0	0	0	255	255	255	255	255	255	255	255	255	255	255	0	0	0	0	0
45	0	0	0	255	255	255	255	255	255	255	255	255	255	255	0	0	0	0	0
46	0	0	0	255	255	255	255	255	255	255	255	255	255	255	0	0	0	0	0
47	0	0	0	255	255	255	255	255	255	255	255	255	255	255	0	0	0	0	0
48	0	0	0	255	255	255	255	255	255	255	255	255	255	255	0	0	0	0	0
49	0	0	0	255	255	255	255	255	255	255	255	255	255	255	0	0	0	0	0
50	0	0	0	255	255	255	255	255	255	255	255	255	255	255	0	0	0	0	0
51	0	0	0	0	255	255	255	255	255	255	255	255	255	0	0	0	0	0	0
52	0	0	0	0	0	255	255	255	255	255	255	255	0	0	0	0	0	0	0
53	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	25
54	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	25
55	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	255	25

- Search pixel-by-pixel for non-zero intensity

Find connected components:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
35	255	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
36	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
37	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
38	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
39	0	0	0	0	0	0	0	0	255	255	255	0	0	0	0	0	0	0	0
40	0	0	0	0	0	0	255	255	255	255	255	255	0	0	0	0	0	0	0
41	0	0	0	0	0	255	255	255	255	255	255	255	255	255	0	0	0	0	0
42	0	0	0	0	255	255	255	255	255	255	255	255	255	255	0	0	0	0	0
43	0	0	0	0	255	255	255	255	255	255	255	255	255	255	0	0	0	0	0
44	0	0	0	255	255	255	255	255	255	255	255	255	255	255	0	0	0	0	0
45	0	0	0	255	255	255	255	255	255	255	255	255	255	255	0	0	0	0	0
46	0	0	0	255	255	255	255	255	255	255	255	255	255	255	0	0	0	0	0
47	0	0	0	255	255	255	255	255	255	255	255	255	255	255	0	0	0	0	0
48	0	0	0	255	255	255	255	255	255	255	255	255	255	255	0	0	0	0	0
49	0	0	0	255	255	255	255	255	255	255	255	255	255	255	0	0	0	0	0
50	0	0	0	255	255	255	255	255	255	255	255	255	255	255	0	0	0	0	0
51	0	0	0	0	255	255	255	255	255	255	255	255	255	0	0	0	0	0	0
52	0	0	0	0	0	255	255	255	255	255	255	255	0	0	0	0	0	0	0
53	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	25
54	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	25
55	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	255	25

- Search pixel-by-pixel for non-zero intensity
- Label found pixel as component 1

Find connected components:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
35	255	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
36	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
37	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
38	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
39	0	0	0	0	0	0	0	255	255	255	255	255	255	255	255	255	255	255	255
40	0	0	0	0	0	0	255	255	255	255	255	255	255	255	255	255	255	255	255
41	0	0	0	0	0	255	255	255	255	255	255	255	255	255	255	255	255	255	255
42	0	0	0	0	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
43	0	0	0	0	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
44	0	0	0	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
45	0	0	0	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
46	0	0	0	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
47	0	0	0	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
48	0	0	0	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
49	0	0	0	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
50	0	0	0	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
51	0	0	0	0	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
52	0	0	0	0	0	255	255	255	255	255	255	255	255	255	255	255	255	255	255
53	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	255
54	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	255
55	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	255	255

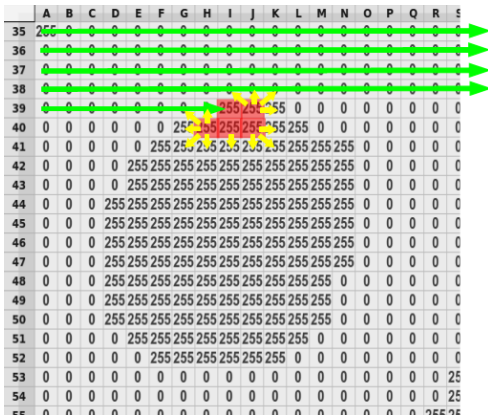
- Search pixel-by-pixel for non-zero intensity
- Label found pixel as component 1
- Check neighbors of all new labeled pixels

Find connected components:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
35	255	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
36	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
37	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
38	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
39	0	0	0	0	0	0	0	0	0	255	255	255	0	0	0	0	0	0	0
40	0	0	0	0	0	0	255	255	255	255	255	255	0	0	0	0	0	0	0
41	0	0	0	0	0	255	255	255	255	255	255	255	255	255	0	0	0	0	0
42	0	0	0	0	255	255	255	255	255	255	255	255	255	255	0	0	0	0	0
43	0	0	0	0	255	255	255	255	255	255	255	255	255	255	0	0	0	0	0
44	0	0	0	255	255	255	255	255	255	255	255	255	255	255	0	0	0	0	0
45	0	0	0	255	255	255	255	255	255	255	255	255	255	255	0	0	0	0	0
46	0	0	0	255	255	255	255	255	255	255	255	255	255	255	0	0	0	0	0
47	0	0	0	255	255	255	255	255	255	255	255	255	255	255	0	0	0	0	0
48	0	0	0	255	255	255	255	255	255	255	255	255	255	255	0	0	0	0	0
49	0	0	0	255	255	255	255	255	255	255	255	255	255	255	0	0	0	0	0
50	0	0	0	255	255	255	255	255	255	255	255	255	255	255	0	0	0	0	0
51	0	0	0	0	255	255	255	255	255	255	255	255	255	0	0	0	0	0	0
52	0	0	0	0	0	255	255	255	255	255	255	0	0	0	0	0	0	0	0
53	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	25
54	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	25
55	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	255	25

- Search pixel-by-pixel for non-zero intensity
- Label found pixel as component 1
- Check neighbors of all new labeled pixels
- Label non-zero pixels as component 1

Find connected components:



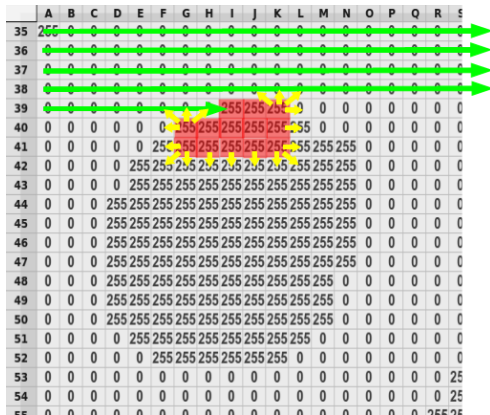
- Search pixel-by-pixel for non-zero intensity
- Label found pixel as component 1
- Check neighbors of all new labeled pixels
- Label non-zero pixels as component 1

Find connected components:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
35	255	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
36	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
37	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
38	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
39	0	0	0	0	0	0	0	0	255	255	255	0	0	0	0	0	0	0	0
40	0	0	0	0	0	0	255	255	255	255	255	255	255	0	0	0	0	0	0
41	0	0	0	0	0	255	255	255	255	255	255	255	255	255	0	0	0	0	0
42	0	0	0	0	255	255	255	255	255	255	255	255	255	255	0	0	0	0	0
43	0	0	0	0	255	255	255	255	255	255	255	255	255	255	0	0	0	0	0
44	0	0	0	255	255	255	255	255	255	255	255	255	255	255	0	0	0	0	0
45	0	0	0	255	255	255	255	255	255	255	255	255	255	255	0	0	0	0	0
46	0	0	0	255	255	255	255	255	255	255	255	255	255	255	0	0	0	0	0
47	0	0	0	255	255	255	255	255	255	255	255	255	255	255	0	0	0	0	0
48	0	0	0	255	255	255	255	255	255	255	255	255	255	255	0	0	0	0	0
49	0	0	0	255	255	255	255	255	255	255	255	255	255	255	0	0	0	0	0
50	0	0	0	255	255	255	255	255	255	255	255	255	255	255	0	0	0	0	0
51	0	0	0	0	255	255	255	255	255	255	255	255	255	0	0	0	0	0	0
52	0	0	0	0	0	255	255	255	255	255	255	0	0	0	0	0	0	0	0
53	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	25
54	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	25
55	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	255	25

- Search pixel-by-pixel for non-zero intensity
- Label found pixel as component 1
- Check neighbors of all new labeled pixels
- Label non-zero pixels as component 1

Find connected components:



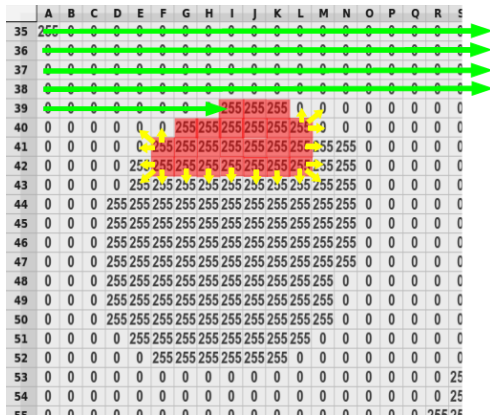
- Search pixel-by-pixel for non-zero intensity
- Label found pixel as component 1
- Check neighbors of all new labeled pixels
- Label non-zero pixels as component 1

Find connected components:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
35	255	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
36	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
37	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
38	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
39	0	0	0	0	0	0	0	0	255	255	255	0	0	0	0	0	0	0	0
40	0	0	0	0	0	0	255	255	255	255	255	255	0	0	0	0	0	0	0
41	0	0	0	0	0	255	255	255	255	255	255	255	255	255	0	0	0	0	0
42	0	0	0	0	255	255	255	255	255	255	255	255	255	255	255	0	0	0	0
43	0	0	0	0	255	255	255	255	255	255	255	255	255	255	255	0	0	0	0
44	0	0	0	255	255	255	255	255	255	255	255	255	255	255	255	0	0	0	0
45	0	0	0	255	255	255	255	255	255	255	255	255	255	255	255	0	0	0	0
46	0	0	0	255	255	255	255	255	255	255	255	255	255	255	255	0	0	0	0
47	0	0	0	255	255	255	255	255	255	255	255	255	255	255	255	0	0	0	0
48	0	0	0	255	255	255	255	255	255	255	255	255	255	255	0	0	0	0	0
49	0	0	0	255	255	255	255	255	255	255	255	255	255	255	0	0	0	0	0
50	0	0	0	255	255	255	255	255	255	255	255	255	255	255	0	0	0	0	0
51	0	0	0	0	255	255	255	255	255	255	255	255	255	0	0	0	0	0	0
52	0	0	0	0	0	255	255	255	255	255	255	0	0	0	0	0	0	0	0
53	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	25
54	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	25
55	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	255	25

- Search pixel-by-pixel for non-zero intensity
- Label found pixel as component 1
- Check neighbors of all new labeled pixels
- Label non-zero pixels as component 1

Find connected components:



- Search pixel-by-pixel for non-zero intensity
- Label found pixel as component 1
- Check neighbors of all new labeled pixels
- Label non-zero pixels as component 1

Find connected components:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
35	255	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
36	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
37	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
38	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
39	0	0	0	0	0	0	0	0	255	255	255	0	0	0	0	0	0	0	0
40	0	0	0	0	0	0	255	255	255	255	255	255	0	0	0	0	0	0	0
41	0	0	0	0	0	255	255	255	255	255	255	255	255	255	0	0	0	0	0
42	0	0	0	0	255	255	255	255	255	255	255	255	255	255	255	0	0	0	0
43	0	0	0	0	255	255	255	255	255	255	255	255	255	255	255	0	0	0	0
44	0	0	0	255	255	255	255	255	255	255	255	255	255	255	255	0	0	0	0
45	0	0	0	255	255	255	255	255	255	255	255	255	255	255	255	0	0	0	0
46	0	0	0	255	255	255	255	255	255	255	255	255	255	255	255	0	0	0	0
47	0	0	0	255	255	255	255	255	255	255	255	255	255	255	255	0	0	0	0
48	0	0	0	255	255	255	255	255	255	255	255	255	255	255	0	0	0	0	0
49	0	0	0	255	255	255	255	255	255	255	255	255	255	255	0	0	0	0	0
50	0	0	0	255	255	255	255	255	255	255	255	255	255	255	0	0	0	0	0
51	0	0	0	0	255	255	255	255	255	255	255	255	255	0	0	0	0	0	0
52	0	0	0	0	0	255	255	255	255	255	255	0	0	0	0	0	0	0	0
53	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	25
54	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	25
55	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	255	25

- Search pixel-by-pixel for non-zero intensity
- Label found pixel as component 1
- Check neighbors of all new labeled pixels
- Label non-zero pixels as component 1

Find connected components:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
35	255	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
36	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
37	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
38	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
39	0	0	0	0	0	0	0	0	255	255	255	0	0	0	0	0	0	0	0
40	0	0	0	0	0	0	255	255	255	255	255	255	0	0	0	0	0	0	0
41	0	0	0	0	0	255	255	255	255	255	255	255	255	255	0	0	0	0	0
42	0	0	0	0	255	255	255	255	255	255	255	255	255	255	255	0	0	0	0
43	0	0	0	0	255	255	255	255	255	255	255	255	255	255	255	0	0	0	0
44	0	0	0	255	255	255	255	255	255	255	255	255	255	255	255	0	0	0	0
45	0	0	0	255	255	255	255	255	255	255	255	255	255	255	255	0	0	0	0
46	0	0	0	255	255	255	255	255	255	255	255	255	255	255	255	0	0	0	0
47	0	0	0	255	255	255	255	255	255	255	255	255	255	255	255	0	0	0	0
48	0	0	0	255	255	255	255	255	255	255	255	255	255	255	255	0	0	0	0
49	0	0	0	255	255	255	255	255	255	255	255	255	255	255	255	0	0	0	0
50	0	0	0	255	255	255	255	255	255	255	255	255	255	255	255	0	0	0	0
51	0	0	0	0	255	255	255	255	255	255	255	255	255	0	0	0	0	0	0
52	0	0	0	0	0	255	255	255	255	255	255	0	0	0	0	0	0	0	0
53	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	25
54	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	25
55	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	255	25

- Search pixel-by-pixel for non-zero intensity
- Label found pixel as component 1
- Check neighbors of all new labeled pixels
- Label non-zero pixels as component 1

Find connected components:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
35	255	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
36	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
37	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
38	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
39	0	0	0	0	0	0	0	0	0	255	255	255	0	0	0	0	0	0	0
40	0	0	0	0	0	0	255	255	255	255	255	255	0	0	0	0	0	0	0
41	0	0	0	0	0	255	255	255	255	255	255	255	255	255	0	0	0	0	0
42	0	0	0	0	255	255	255	255	255	255	255	255	255	255	0	0	0	0	0
43	0	0	0	0	255	255	255	255	255	255	255	255	255	255	0	0	0	0	0
44	0	0	0	255	255	255	255	255	255	255	255	255	255	255	0	0	0	0	0
45	0	0	0	255	255	255	255	255	255	255	255	255	255	255	0	0	0	0	0
46	0	0	0	255	255	255	255	255	255	255	255	255	255	255	0	0	0	0	0
47	0	0	0	255	255	255	255	255	255	255	255	255	255	255	0	0	0	0	0
48	0	0	0	255	255	255	255	255	255	255	255	255	255	255	0	0	0	0	0
49	0	0	0	255	255	255	255	255	255	255	255	255	255	255	0	0	0	0	0
50	0	0	0	255	255	255	255	255	255	255	255	255	255	255	0	0	0	0	0
51	0	0	0	0	255	255	255	255	255	255	255	255	255	0	0	0	0	0	0
52	0	0	0	0	0	255	255	255	255	255	255	0	0	0	0	0	0	0	0
53	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	25
54	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	25
55	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	255	25

- Search pixel-by-pixel for non-zero intensity
- Label found pixel as component 1
- Check neighbors of all new labeled pixels
- Label non-zero pixels as component 1

Find connected components:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
35	255	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
36	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
37	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
38	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
39	0	0	0	0	0	0	0	0	0	25	0	0	0	0	0	0	0	0	0
40	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
41	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
42	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
43	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
44	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
45	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
46	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
47	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
48	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
49	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
50	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
51	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
52	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
53	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	25
54	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	25
55	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	255	25

- Search pixel-by-pixel for non-zero intensity
- Label found pixel as **component 2**
- ...

Result of connected component labeling:

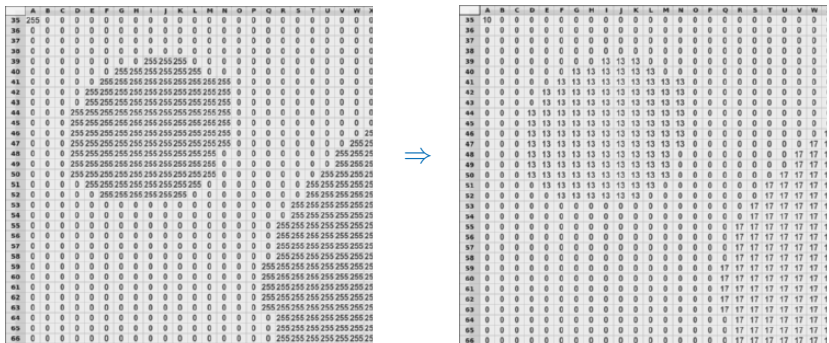


Figure: Result: particle indices instead of intensities

■ General

[CRL01] Thomas H. Cormen, Ronald L. Rivest, and Charles E. Leiserson.

Introduction to Algorithms.

MIT Press, Cambridge, Mass, 2001.

[MS08] Kurt Mehlhorn and Peter Sanders.

Algorithms and data structures, 2008.

<https://people.mpi-inf.mpg.de/~mehlhorn/ftp/Mehlhorn-Sanders-Toolbox.pdf>.

■ Graph-Search

[Wika] [Breadth-first search](#)

`https://en.wikipedia.org/wiki/
Breadth-first_search`

[Wikb] [Depth-first search](#)

`https:
//en.wikipedia.org/wiki/Depth-first_search`

■ Graph-Connectivity

[Wik] [Connectivity \(graph theory\)](#)

`https://en.wikipedia.org/wiki/Connectivity_
\(graph_theory\)`