

Wegfindung

Vergleich verschiedener
Algorithmen

Ablauf der Versuche





- ❖ Prozedurale Generierung
- ❖ *Seed*-Funktion von Pseudozufallsgenerator
- ❖ Wunscheinstellungen durch Nutzer möglich
- ❖ Beliebige Gerichtetheit, gemischt

$$e = \max\left(1, R\left(0, \min\left(\frac{n}{2} - 1, e_{\max}\right)\right)\right)$$



- ❖ Kraft-basierte, schrittweise Simulation
- ❖ Fruchterman-Reingold Algorithmus (1991)
- ❖ Gleichmäßige Verteilung der Knoten und Rahmen

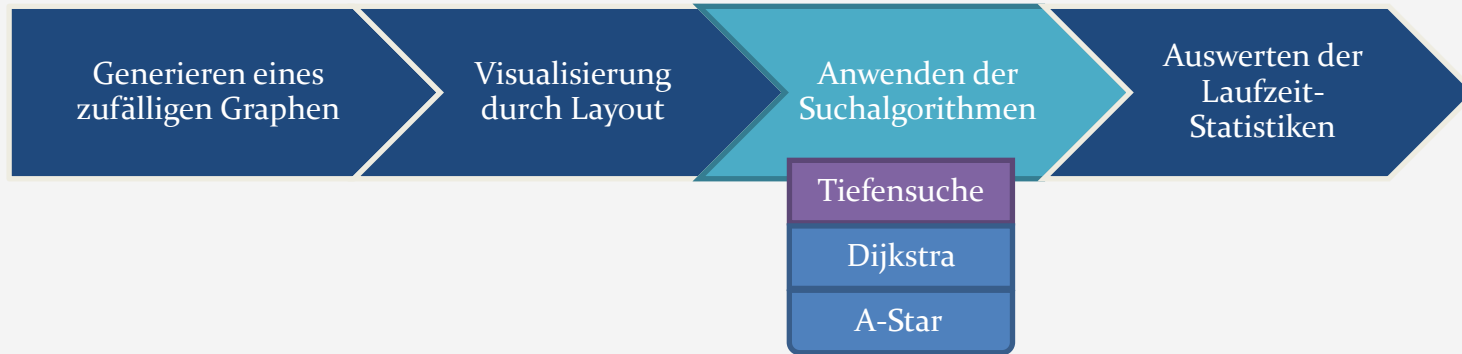
$$k = \sqrt{\frac{w \cdot h}{n}}$$

$$F_a(x) = \frac{x^2}{k} \qquad F_r(x) = \frac{k^2}{x}$$

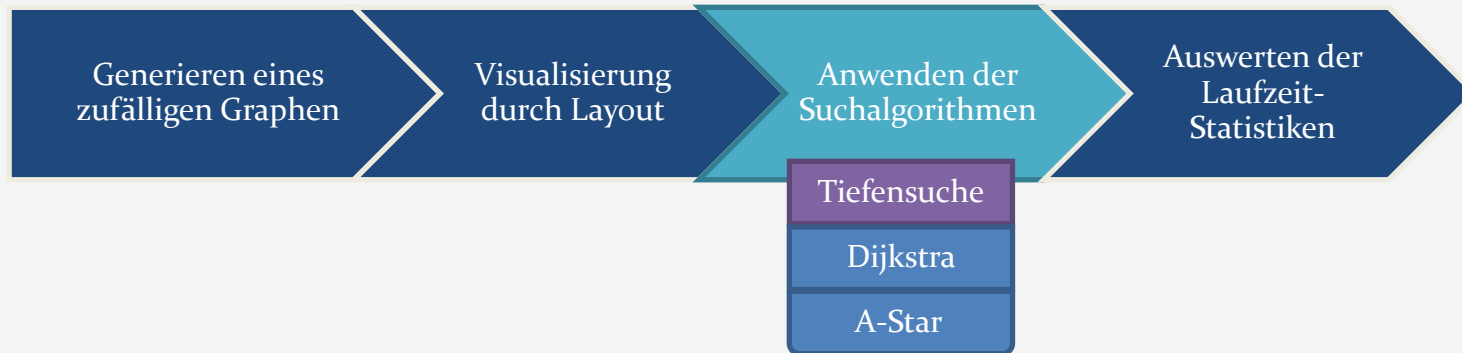
$$t(s) = \frac{w}{10} - \frac{w}{s_g \cdot 5000} \cdot s$$



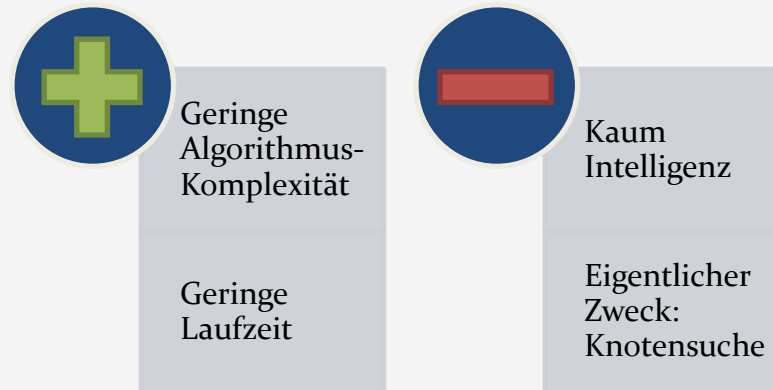
- ❖ Suche des kürzesten/günstigsten Wegs von A nach B
- ❖ Verschiedenste Ansätze für Sonderfälle & Bedürfnisse
- ❖ Vergleichbar durch Laufzeit-Ergebnis Relation
- ❖ Zwei Simulationsarten: Einzel- und Massensuche
- ❖ Animationsmodus zum Nachverfolgen der Abläufe

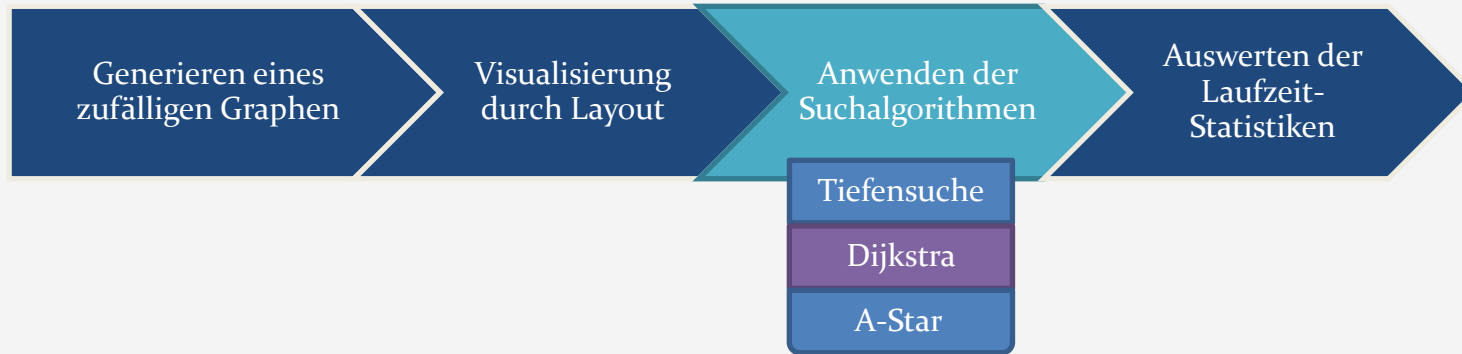


- ❖ „Gehen“ in eine durch Gewichte bestimmte Richtung
- ❖ Linearer Algorithmus-Verlauf
- ❖ Zeitkomplexität: $O(E + V)$

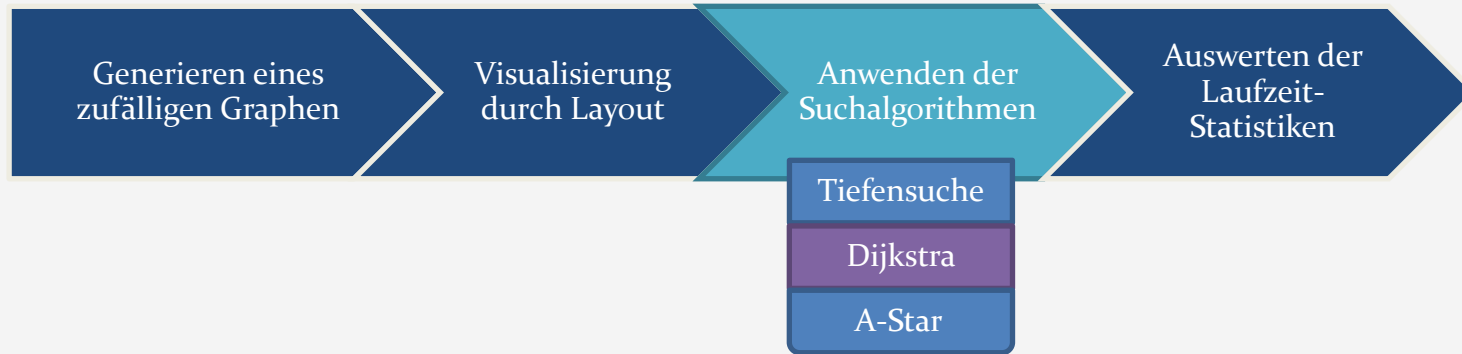


- ❖ „Gehen“ in eine durch Gewichte bestimmte Richtung
- ❖ Linearer Algorithmus-Verlauf
- ❖ Zeitkomplexität: $O(E + V)$



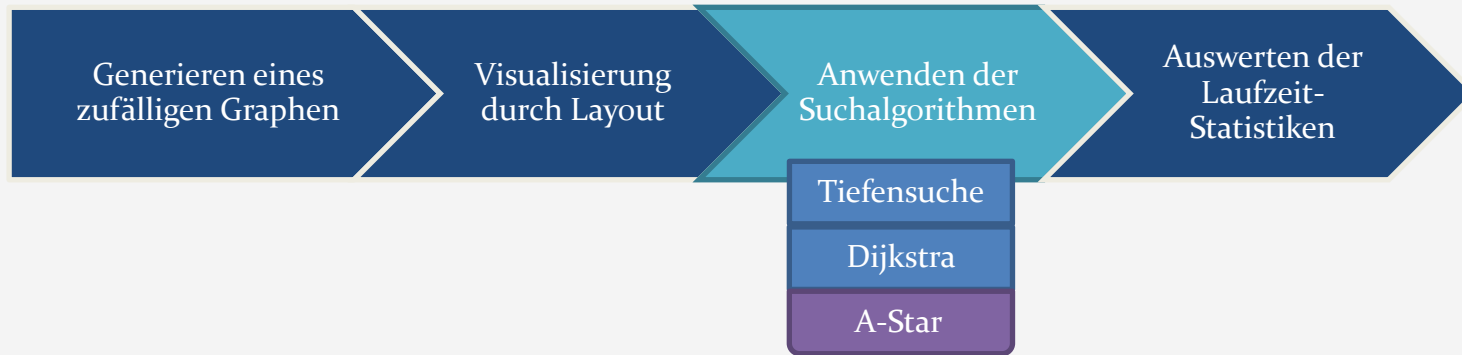


- ❖ Erfunden von E. W. Dijkstra, 1956
- ❖ Kategorisiert als *Greedy*-Algorithmus
- ❖ Zeitkomplexität: $O(V^2)$

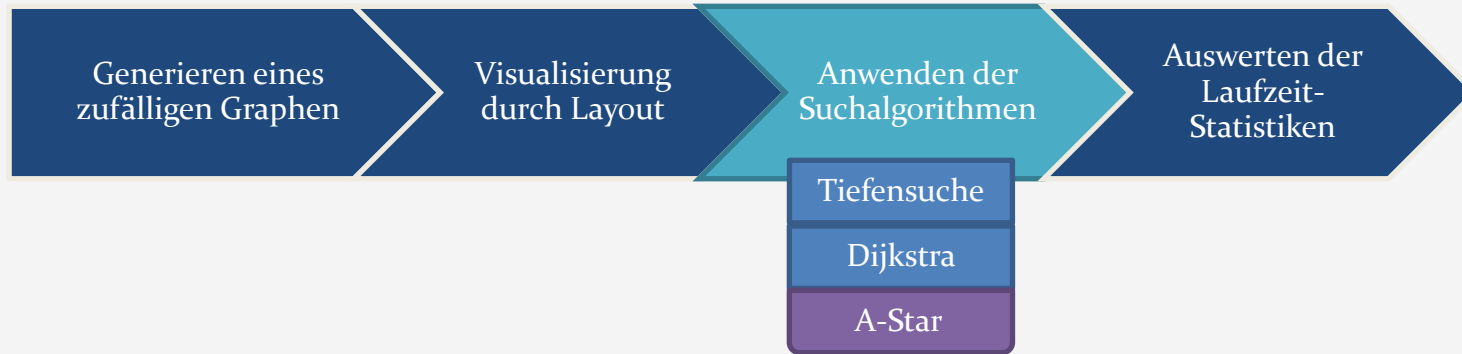


- ❖ Erfunden von E. W. Dijkstra, 1956
- ❖ Kategorisiert als *Greedy*-Algorithmus
- ❖ Zeitkomplexität: $O(V^2)$





- ❖ Peter Hart, Nils Nilsson und Bertram Raphael, 1968
- ❖ Gesehen als Allrounder und meistverbreitet
- ❖ Zeitkomplexität: abh. von Heuristik-Funktion
- ❖ Stets weiterentwickelt (Fast-Stacks etc.)



- ❖ Peter Hart, Nils Nilsson und Bertram Raphael, 1968
- ❖ Gesehen als Allrounder und meistverbreitet
- ❖ Zeitkomplexität: abh. von Heuristik-Funktion
- ❖ Stets weiterentwickelt (Fast-Stacks etc.)





	Laufzeit	Kosten	Kanten	versch.	Knoten	Sortierungen
DFS	72,61709	116	10	24	83	14
Dijkstra	223,76864	78	6	36	115	30
A-Star	109,71132	94	6	19	29	11



	Laufzeit	Kosten	Kanten	versch.	Knoten	Sortierungen
DFS	72,61709	116	10	24	83	14
Dijkstra	223,76864	78	<u>6</u>	36	115	30
<u>A-Star</u>	109,71132	94	<u>6</u>	<u>19</u>	<u>29</u>	<u>11</u>

- A-Star als klarer Favorit, Opfern von ein wenig Genauigkeit für Geschwindigkeit nachvollziehbar

```
public class Input {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter a number:");
        int number = scanner.nextInt();
        System.out.println("You entered: " + number);
    }
}

public class Output {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}

public class Arithmetic {
    public static void main(String[] args) {
        int a = 10;
        int b = 5;
        int sum = a + b;
        int difference = a - b;
        int product = a * b;
        int quotient = a / b;
        System.out.println("Sum: " + sum);
        System.out.println("Difference: " + difference);
        System.out.println("Product: " + product);
        System.out.println("Quotient: " + quotient);
    }
}

public class StringManipulation {
    public static void main(String[] args) {
        String name = "John Doe";
        String message = "Hello, " + name + "!";
        System.out.println(message);
    }
}

public class ConditionalLogic {
    public static void main(String[] args) {
        int age = 18;
        if (age > 18) {
            System.out.println("You are an adult.");
        } else {
            System.out.println("You are a minor.");
        }
    }
}

public class Loops {
    public static void main(String[] args) {
        for (int i = 0; i < 10; i++) {
            System.out.println("Iteration: " + i);
        }
    }
}

public class Arrays {
    public static void main(String[] args) {
        int[] numbers = {1, 2, 3, 4, 5};
        System.out.println("Array elements:");
        for (int number : numbers) {
            System.out.println(number);
        }
    }
}

public class Collections {
    public static void main(String[] args) {
        ArrayList<String> names = new ArrayList<>();
        names.add("Alice");
        names.add("Bob");
        names.add("Charlie");
        System.out.println("Names in the list:");
        for (String name : names) {
            System.out.println(name);
        }
    }
}

public class Exceptions {
    public static void main(String[] args) {
        try {
            int result = 10 / 0;
        } catch (ArithmeticException e) {
            System.out.println("Exception caught: " + e.getMessage());
        }
    }
}

public class FileOperations {
    public static void main(String[] args) {
        File file = new File("example.txt");
        if (file.exists()) {
            System.out.println("File exists.");
        } else {
            System.out.println("File does not exist.");
        }
    }
}

public class DateAndTime {
    public static void main(String[] args) {
        Date date = new Date();
        System.out.println("Current date and time: " + date);
    }
}

public class Networking {
    public static void main(String[] args) {
        InetAddress address = InetAddress.getByName("127.0.0.1");
        System.out.println("IP Address: " + address.getHostAddress());
    }
}

public class Thread {
    public static void main(String[] args) {
        Thread thread = new Thread() {
            @Override
            public void run() {
                System.out.println("Thread running.");
            }
        };
        thread.start();
    }
}

public class Reflection {
    public static void main(String[] args) {
        Class<String> stringClass = String.class;
        System.out.println("Class: " + stringClass.getName());
    }
}

public class Annotations {
    public static void main(String[] args) {
        @SuppressWarnings("unused")
        int unusedVariable = 1;
    }
}

public class Generics {
    public static void main(String[] args) {
        List<String> names = new ArrayList<>();
        names.add("Alice");
        names.add("Bob");
        System.out.println("Names: " + names);
    }
}

public class LambdaExpressions {
    public static void main(String[] args) {
        Runnable task = () -> {
            System.out.println("Task executed.");
        };
        task.run();
    }
}

public class StreamAPI {
    public static void main(String[] args) {
        List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5);
        numbers.stream().filter(n -> n % 2 == 0).forEach(System.out::println);
    }
}

public class CompletableFuture {
    public static void main(String[] args) {
        CompletableFuture<String> future = CompletableFuture.supplyAsync(() -> {
            return "Hello, Future!");
        });
        System.out.println(future.get());
    }
}

public class Project {
    public static void main(String[] args) {
        // Project code
    }
}
```