Fortgeschrittene Algorithmen

Begrüßung

- Vorstellung
- Themen Semester
- Einstieg + Einführung

Vorstellung Dozent – Uli Siebold

- Studium Informatik
  Karlsruhe & Freiburg
- 10 Jahre bei Fraunhofer (D)
  Dr.-Ing.: Systemmodellierung
  Forschungsgruppe zu:
  Sicherheits- und
  Zuverlässigkeitsanalysen
- 5 Jahre bei CuriX (CH)
  Head of Dev, Head of Res:
  Monitoring-Daten auswerten,
  vorhersagen und Systeme
  resilienter machen

- 44 Jahre
- 5 Kinder
- Hobbies:
  Taekwon-Do, Ausflüge
- Was mir wichtig ist:

  Zuhören,
  Offener Austausch,
  Rückfragen

Vorstellung Dozent – Uli Siebold

- Studium Informatik
  Karlsruhe & Freiburg
- 10 Jahre bei Fraunhofer (D)
  Dr.-Ing.: Systemmodellierung
  Forschungsgruppe zu:
  Sicherheits- und
  Zuverlässigkeitsanalysen
- 5 Jahre bei CuriX (CH)
  Head of Dev, Head of Res:
  Monitoring-Daten auswerten,
  vorhersagen und Systeme
  resilienter machen

- 44 Jahre
- 5 Kinder
- Hobbies:
  Taekwon-Do, Ausflüge
- Was mir wichtig ist:

  Zuhören,
  Offener Austausch,
  Rückfragen

Themenvorstellung

- Datenstrukturen
- (Ausgewählte) Algorithmen
- Entwurfsmethoden
- Analysemethoden

<u>Vorwiegend genutzte Quelle:</u>

Thomas Cormen, Charles Leiserson, Ronald Rivest, Clifford Stein: Algorithmen – eine Einführung

+ vereinzelt andere Quellen, auf den Folien angegeben

Strutur / Generelles

- Vorstellung an der Tafel und/oder Folien
- Life-Coding in Java
- Gruppenarbeit
- Einzelarbeit (Programmieren oder Papier)
- Tafelrechnen
- Hausaufgaben → Bonuspunkte 10 % für Klausur möglich

(vermutlich allgemein) bekannte Datentypen

Typname
boolean
char
byte
short
int
long
float
double

https://de.wikibooks.org/wiki/Java_Standard:_Primitive_Datentypen

Datenwort – vereinfacht Wort

Grundsätzlich:

- Ein Datenwort oder einfach nur Wort ist eine bestimmte Datenmenge, die ein Computer in der arithmetisch-logischen Einheit des Prozessors in einem Schritt verarbeiten kann. Ist eine maximale Datenmenge gemeint, so wird deren Größe Wortbreite, Verarbeitungsbreite oder Busbreite genannt. (https://de.wikipedia.org/wiki/Datenwort)

- In Programmiersprachen ist das Datenwort die Bezeichnung für Datentypen

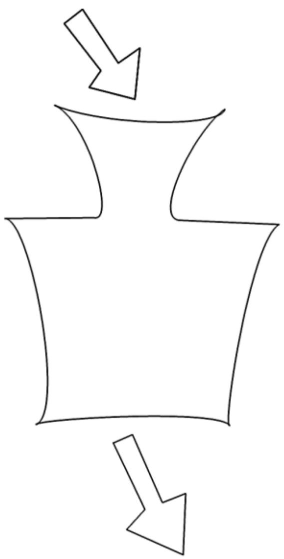
(vermutlich allgemein) bekannte Datentypen

Typname	Größe	Wertebereich
boolean	1 bit (nicht präzise festgelegt)	true / false
char	16 bit	0 ... 65.535 (z. B. 'A')
byte	8 bit	-128 ... 127
short	16 bit	-32.768 ... 32.767
int	32 bit	-2.147.483.648 ... 2.147.483.647
long	64 bit	-2^{63} bis 2^{63}-1, 0 bis 2^{64} -1[
float	32 bit	+/-1,4E-45 ... +/-3,4E+38
double	64 bit	+/-4,9E-324 ... +/-1,7E+308

https://de.wikibooks.org/wiki/Java_Standard:_Primitive_Datentypen

Algorithmus

- Ein Algorithmus ist eine wohldefinierte Rechenvorschrift, die eine Größe (=Entität, Objekt) oder Menge von Größen verwendet und eine Größe oder Menge von Größen als Ausgabe erzeugt.



Verwendete Größe

Wohldefinierte (Rechen)vorschrift

Ausgegebene Größe

Sortierspiel

Sortierspiel

➔ Geht das irgendwie „optimal“?

➔ Gibt es ganz allgemein ein Rezept, Aufgaben optimal zu lösen

Was man nicht messen kann, kann man auch nicht verbessern.

Aufwandsmessung

Wir wollen wissen wie lange ein Algorithmus für eine Aufgabe benötigt (Zeit, Rechenschritte). Wir wollen vermutlich auch wissen, wie viel Platz wir im Rechner benötigen (Memory, Storage)

~~Aufwandsmessung~~ → Aufwandschätzungen

Wir wollen wissen wie lange ein Algorithmus für eine Aufgabe benötigt (Zeit, Rechenschritte). Wir wollen vermutlich auch wissen, wie viel Platz wir im Rechner benötigen (Memory, Storage)

Hierzu müssen wir

- Zählen und Rechnen
- mathematische Aufgaben praktisch lösen: approximativ

Wir werden uns also mit Algorithmik und Numerik beschäftigen

Datenstrukturen

Geeignete Datenstrukturen helfen, Aufgabenstellungen (effizient) zu lösen.

- Graph

Graph

- Gerichteter Graph

Ein gerichteter Graph (Digraph) ***G*** ist ein Paar ***(V, E)***, wobei ***V*** eine endliche Menge und ***E*** eine binäre Relation auf V ist …

Graph

- Gerichteter Graph

Ein gerichteter Graph (Digraph) ***G*** ist ein Paar ***(V, E)***, wobei ***V*** eine endliche Menge und ***E*** eine **binäre Relation** auf V ist …

➔ Wir sollten uns wirklich erst einmal mit Grundlagen beschäftigen!

Grundlagen

- Mengen

Mengen

- Eine Menge ist ein abstraktes Objekt, das aus der Zusammenfassung einer Anzahl einzelner Objekte hervorgeht. (https://de.wikipedia.org/wiki/Menge_(Mathematik))
- Schreibweisen:

$M = \{blau, gelb\}$ abgekürzt für $M = \{x \mid x = blau\ oder\ x = gelb\}$

$M = \{3, 6, 9, 12, \ldots 96, 99\}$
$M = \{x \mid x\ ist\ eine\ durch\ 3\ teilbare\ Zahl\ zwischen\ 1\ und\ 100\}$

$M = \{1, 2, 3, 5, \ldots\}$ ist die Darstellung einer unendlichen Menge

Mengen - Beziehungen

- Gleichheit: Zwei Mengen heißen gleich, wenn sie dieselben Elemente enthalten:

$$A = B :\Leftrightarrow \forall x (x \in A \Leftrightarrow x \in B)$$

- Teilmenge: Eine Menge heißt Teilmenge einer Menge B, wenn jedes Element von A auch in ein Element von B ist.

$$A \subseteq B :\Leftrightarrow \forall x (x \in A \rightarrow x \in B)$$

- Differenz:

$$A \backslash \mathrm{B} \ := \{x \mid (x \in A) \land (x \notin B)\}$$

Mengen – Kartesisches Produkt

- Das kartesische Produkt oder auch Produktmenge enthält komplexe Elemente, die nicht Elemente der Ausgangsmengen sind.
- $A \times B \coloneqq \{(a, b) \mid a \in A, b \in B\}$

Binäre Relation

- Formal ist eine binäre Relation R die Untermenge eines Kartesischen Produkts einer Menge:

$$A \times A := \{(a, b) \mid a \in A, b \ \in A\} = A^2$$
$$R \subseteq A^2$$

- Ist $(a, b) \in R$ so sagt man, a und b stehen in Relation.

- Wichtige Eigenschaften (die jeweils nicht immer gelten **müssen**)

  – Reflexivität
  – Transitivität
  – Symmetrie
  – Antisymmetrie
  – Vollständigkeit

Datenstrukturen

Geeignete Datenstrukturen helfen, Aufgabenstellungen (effizient) zu lösen.

- Graph

Graph: gerichteter Graph

- Ein gerichteter Graph (Digraph) ***G*** ist ein Paar ***(V, E)***, wobei ***V*** eine endliche Menge und ***E*** eine binäre Relation auf V ist.

- ***V*** = Knotenmenge von ***G***, Elemente: Knoten.

- ***E*** = Kantenmenge von ***G***, Elemente: Kanten
  Kantenmenge: geordnet!

  → (u, v) und (v, u) sind **nicht** dieselben Kanten

- Schlingen (Kante von Knoten auf sich selbst) sind möglich

- Darstellung Knoten als Kreis, Kante als Pfeil

Graph: ungerichteter Graph

- Ein ungerichteter Graph ***G*** ist ein Paar ***(V, E)***, wobei ***V*** eine endliche Menge und ***E*** eine binäre Relation auf V ist.

- ***V*** = Knotenmenge von ***G***, Elemente: Knoten.

- ***E*** = Kantenmenge von ***G***, Elemente: Kanten
  Kantenmenge: ungeordnet!

  → (u, v) und (v, u) ist die selbe Kante

- Schlingen (Kante von Knoten auf sich selbst) sind nicht möglich

- Darstellung Knoten als Kreis, Kante als Linie (ohne Pfeilspitze)

Wo finden wir Graphen / Beispiele

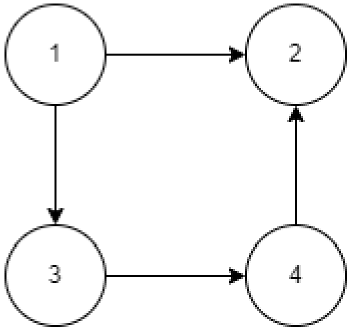
- Tafel ☺

Algorithmen

- Graphenalgorithmen

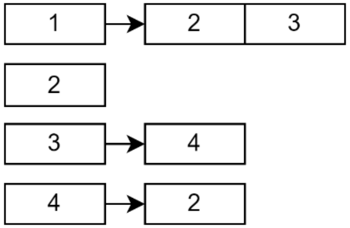
Graphenalgorithmen

- Darstellung als Grafik
  (Kreise und Linien)



- Darstellung im Rechner

  – Adjazenzlisten
    Für jeden Knoten gibt es eine Liste, die damit verbundene Knoten enthält.



  – Adjazenzmatrix (Knoten durchnummeriert)
    $|A| \times |A|$- Matrix $A = (a_{ij})$

$$a_{ij} \begin{cases} 1\ falls\ (i,j) \epsilon\ E \\ 0\ sonst. \end{cases}$$

	1	2	3	4
1	0	1	1	0
2	0	0	0	0
3	0	0	0	1
4	0	1	0	0

Graphenalgorithmen

- Übung auf dem Blatt und Tafel:

  Weg der Nahrung zu Ihnen

- Auswertung / Diskussion:

  Vor- und Nachteile der Darstellungsarten

Graphenalgorithmen

- Übung auf dem Blatt und Tafel:

  Weg der Nahrung zu Ihnen (persönliche Nahrungskette)

- Auswertung / Diskussion:

  Vor- und Nachteile der Darstellungsarten

Finden wir die Ende der Nahrungskette

- Wie könnten wir im Nahrungsketten-Graph das Ende der Nahrungskette finden?

mit einem (oder mehreren) Algorithmen

Transitive Hülle

- Die transitive Hülle ist:
  die Erweiterung der Relation, die zusätzlich alle indirekt erreichbaren Paare erhält (transitiv)

- Der Algorithmus von Warshall kann die transitive Hülle erzeugen:

```
Für k=1 bis n
    Für i=1 bis n
        Falls d[i, k] = 1
            Für j=1 bis n
                Falls d[k, j] = 1
                    d[i, j] = 1
```

- Laufzeit-Komplexität: $O(n^3)$

Universelle Senke

- Stille arbeit

Universelle Senke

- Stille arbeit

	V1	V2	V3	V4	V5	V6
V1	0	1	0	0	0	0
V2	0	0	0	0	0	0
V3	0	1	0	0	0	0
V4	0	1	0	0	0	0
V5	0	1	0	0	0	0
V6	0	1	0	0	0	0

Hausaufgaben / Übung / Selbststudium

- Multigraph → zu ungerichtetem Graph:
  (Finden Sie heraus, was ein Multigraph ist und lösen Sie dann folgende Aufgabe)
  Geben Sie je einen Algorithmus an, der einen Multigraph in einen „äquivalenten“ ungerichteten Graph transformiert.
  - Variante A: Ausgehend von einer Adjazenzlistendarstellung
  - Variante B: Ausgehend von einer Adjazenzmatrixdarstellung

- Geben Sie je einen Algorithmus an, der einen gerichteten Graphen transponiert.
  - Variante A: Ausgehend von einer Adjazenzlistendarstellung
  - Variante B: Ausgehend von einer Adjazenzmatrixdarstellung

Exkurs / Voraussetzung: LIFO + FIFO

- Dynamische Mengen

- Stapel
  - **L**ast-**I**n → **F**irst-**O**ut

  ➔ LIFO

- Warteschlange
  - **F**irt-**I**n → **F**irst-**O**ut

  ➔ FIFO

Suchen: Breitensuche

- Gegeben: Graph ***G*** = ***(V, E)***,
- Ziel:
  Alle (erreichbaren) Knoten entdecken, systematisch von einem Startknoten **s** ausgehend

- Beispiele in „realer Welt“ finden → Tafel

Suchen: Breitensuche - Algorithmus

Setup / Initializing

```
Für jeden Knoten u aus G.V-{s}
     u.farbe = weiß
     u.d = infinity
     u.pre = null
s.farbe = grau
s.d = 0
s.pre = null
Queue Q = {}
Q.Enqueue(s)
```

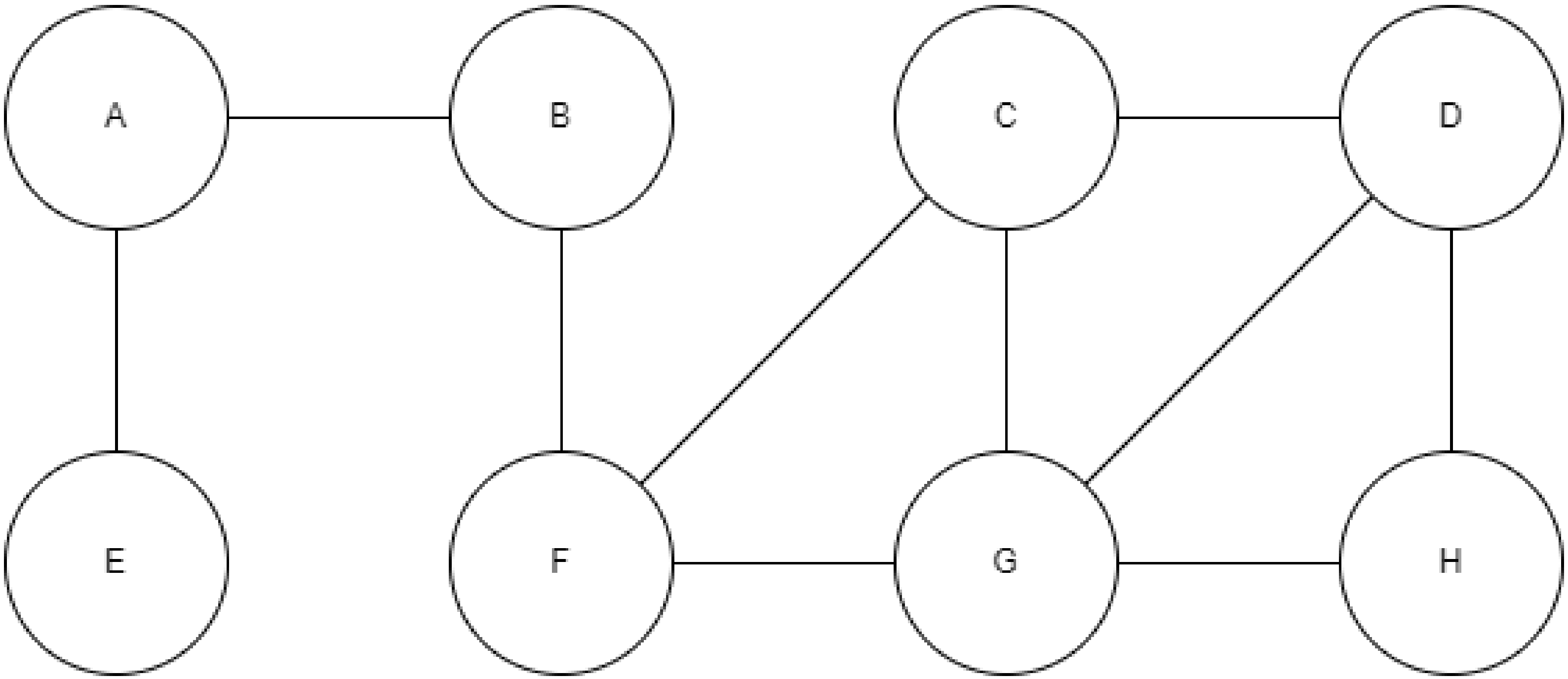
Suchen: Breitensuche - Algorithmus

Breadth First Search (BFS)

```
While Q != {}
      u = Q.dequeue()
      für jeden Knoten v aus G.Adj[u]
            if v.farbe == weiss
                  v.farbe = grau
                  v.d = u.d + 1
                  v.pre = u
                  Q.enqueue(v)
      u.farbe = schwarz
```

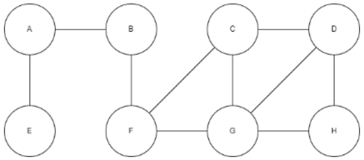
Suchen: Breitensuche - Algorithmus

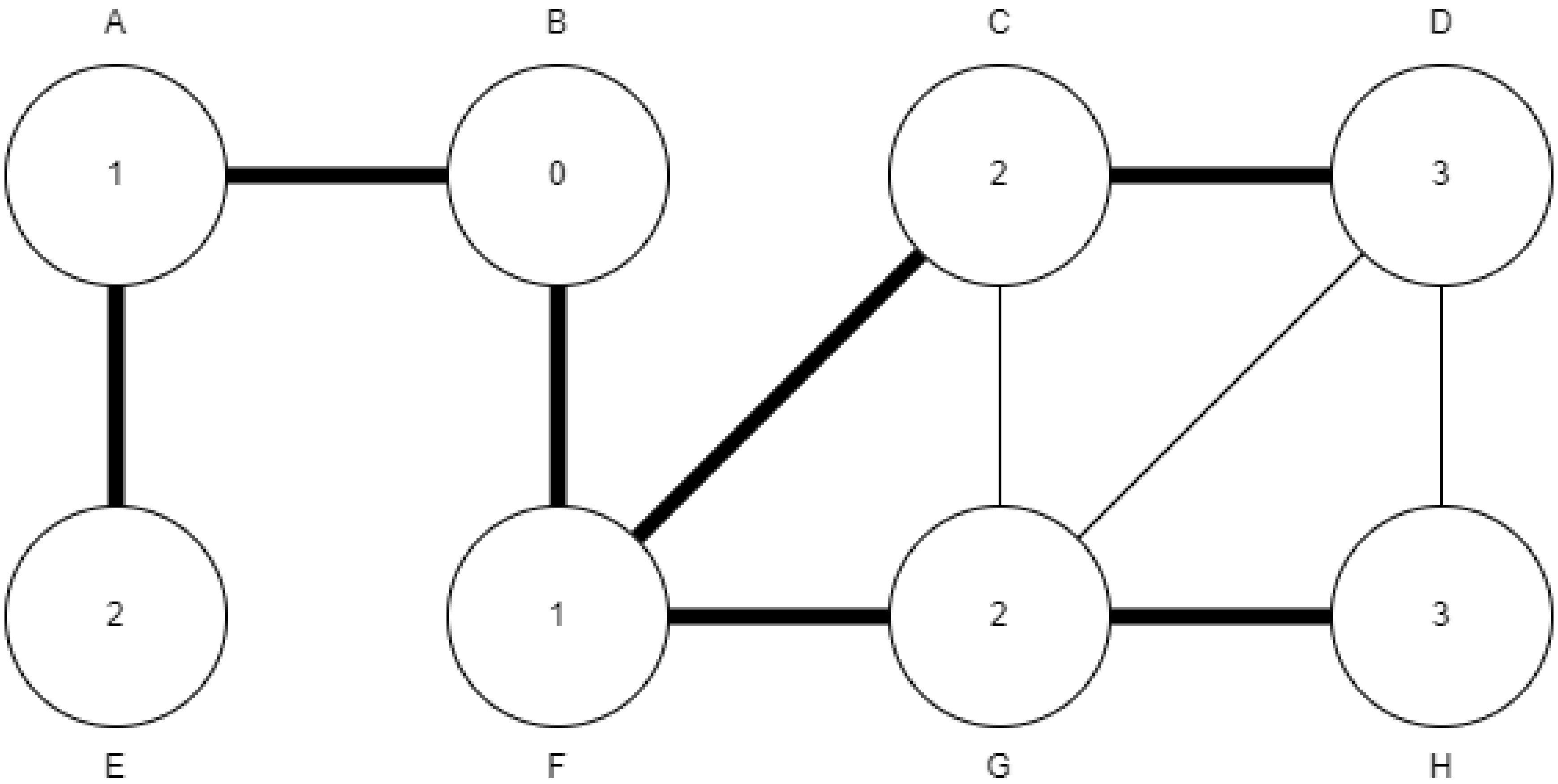
Übung: BFS auf Graph: Startknoten **B**



Suchen: Breitensuche - Algorithmus

Übung: BFS auf Graph: Startknoten **B**





Suchen: Breitensuche

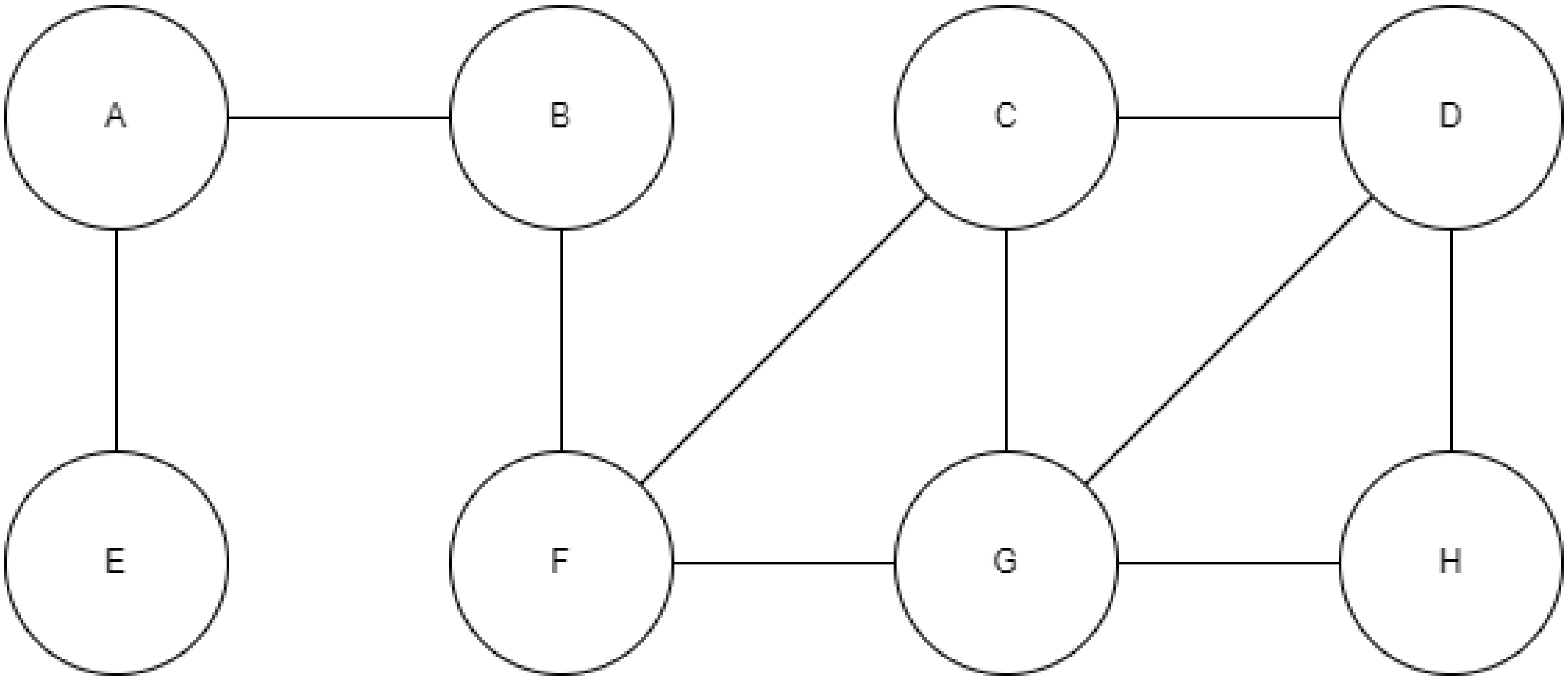
- Erkundet alle erreichbaren Knoten von Startknoten aus
- Findet dabei kürzeste Pfade zwischen S und erkundeten Knoten

- Erzeugt:

  Breitensuchbaum !

Stille Arbeit

- Tiefensuche durchführen auf Graph mit Startknoten B



Suchen: Tiefensuche - Algorithmus

Setup / Initializing

```
Für jeden Knoten u aus G.V-{s}
     u.farbe = weiß
     u.pre = null
zeit = 0

DFS-Visit(G, s)
```

Suchen: Breitensuche - Algorithmus

Depth First Search (DFS)

```
DFS-Visit(G, u)
      zeit = zeit + 1
      u.d = zeit
      u.farbe = grau
      für jeden Knoten v aus G.Adj[u]
            if v.farbe == weiss
                  v.pre = u
                  DFS-Visit(G, v)
      u.farbe = schwarz
      zeit = zeit + 1
      u.f = zeit
```