Ein deterministischer Algorithmus

1. Falls $n < 30$: Sortiere und finde so das Ergebnis.
2. Bilde $m = \lfloor n/5 \rfloor$ Gruppen mit je fünf Schlüsseln. Bis zu vier Schlüssel bleiben übrig.
3. Berechne den Median jeder Gruppe.
4. Berechne rekursiv den Median p dieser $\lfloor n/5 \rfloor$ Mediane.
5. Verwende p als Pivotelement und führe damit Quickselect aus.

Welchen Rang r hat p?

p ist der Median von $m = \lfloor n/5 \rfloor$ vielen kleinen Medianen
$\rightarrow$ mindestens $m/2 - 1$ kleine Mediane sind kleiner als p
Jeder kleine Median hat zwei Schlüssel die kleiner sind
$\rightarrow$ mindestens $3m/2 - 1$ kleinere Schlüssel als p

Ebenso: Mindestens $3m/2 - 1$ größere Schlüssel als p
Das sind jeweils $3\lfloor n/5 \rfloor/2 - 1 \geq 3n/10 - 3/2 \geq n/4$

Deterministisches Selektieren – Analyse

Die Anzahl der Vergleiche ist jetzt

$$C_n \leq n + 1 + C_{\lfloor n/5 \rfloor} + C_{\lfloor 3n/4 \rfloor}$$

falls $n > 30$ und $O(1)$ falls $n \leq 30$:

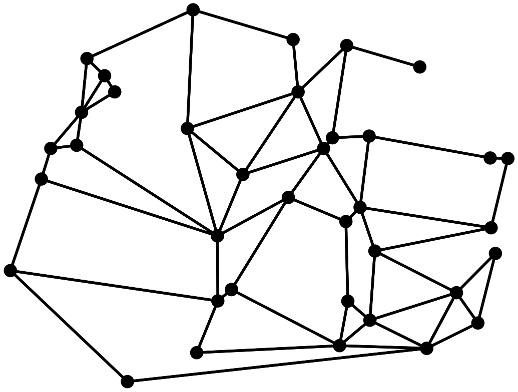
- $C_{\lfloor n/5 \rfloor}$ für das rekursive Finden der Mediane
- $C_{\lfloor 3n/4 \rfloor}$ für die nächste Suche

Es folgt $C_n = O(n)$, da $1/5 + 3/4 < 1$.

Wir können also den Schlüssel mit Rang k in linearer Zeit finden.

Übersicht

Graphen



Graphen

Definition

Ein ungerichteter Graph ist ein Paar (V, E), wobei V die Menge der Knoten und $E \subseteq \binom{V}{2}$ die Menge der Kanten ist.

Definition

Ein gerichteter Graph ist ein Paar (V, E), wobei V die Menge der Knoten und $E \subseteq V \times V$ die Menge der Kanten ist.

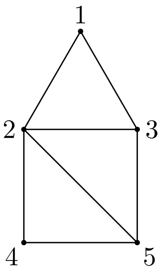
Oft betrachten wir Graphen mit Knoten- oder Kantengewichten.

Dann gibt es zusätzlich Funktionen $V \to \mathbf{R}$ oder $E \to \mathbf{R}$.

Übersicht

Darstellung von Graphen

Adjazenzmatrix



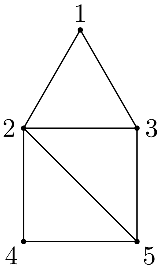
$$\begin{pmatrix} \cdot & 1 & 1 & 0 & 0 \\ \cdot & \cdot & 1 & 1 & 1 \\ \cdot & \cdot & \cdot & 0 & 1 \\ \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot \end{pmatrix}$$

Speicherbedarf: $\Theta(|V|^2)$

Für gerichtete Graphen wird die ganze Matrix verwendet.

Darstellung von Graphen

Adjazenzliste



1	2, 3
2	1, 3, 4, 5
3	1, 2, 5
4	2, 5
5	2, 3, 4

Speicherbedarf: $\Theta(|V| + |E|)$.

In $O(n^2)$ Schritten kann zwischen beiden Darstellungen konvertiert werden.

Darstellung von Graphen

Java

```
public class SimpleGraph<V> implements Graph<V> {
  protected Set<V> nodes;
  protected Map<V, List<V>> edges;
  protected boolean directed = false;
  protected Map<String, Map<V, Object>> nodeAttributes;
  protected Map<String, Map<Edge<V>, Object>> edgeAttributes;
```

Wir wählen die Darstellung durch eine Adjazenzliste.

Java

```
public class Edge⟨V⟩ implements Serializable {
  private static final long serialVersionUID = −262552244439018519 L;
  public V s, t;
  private boolean directed;
  protected Edge(V s, V t, boolean directed) {
    this.s = s;
    this.t = t;
    this.directed = directed;
  }
```

Java

```
public void addNode(V u) {
  nodes.add(u);
  edges.put(u, new LinkedList<V>());
}
```

Java

```
public void addEdge(V s, V t) {
  List<V> adjlist = edges.get(s);
  adjlist.add(t);
  if(!directed) {
    adjlist = edges.get(t);
    adjlist.add(s);
  }
}
```

Java

```
public void delEdge(V s, V t) {
  List⟨V⟩ adjlist = edges.get(s);
  adjlist.remove(t);
  if(!directed) {
    adjlist = edges.get(t);
    adjlist.remove(s);
  }
}
```

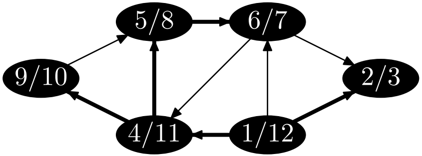
Übersicht

Tiefensuche

Tiefensuche ist ein sehr mächtiges Verfahren, das iterativ alle Knoten eines gerichteten oder ungerichteten Graphen besucht.

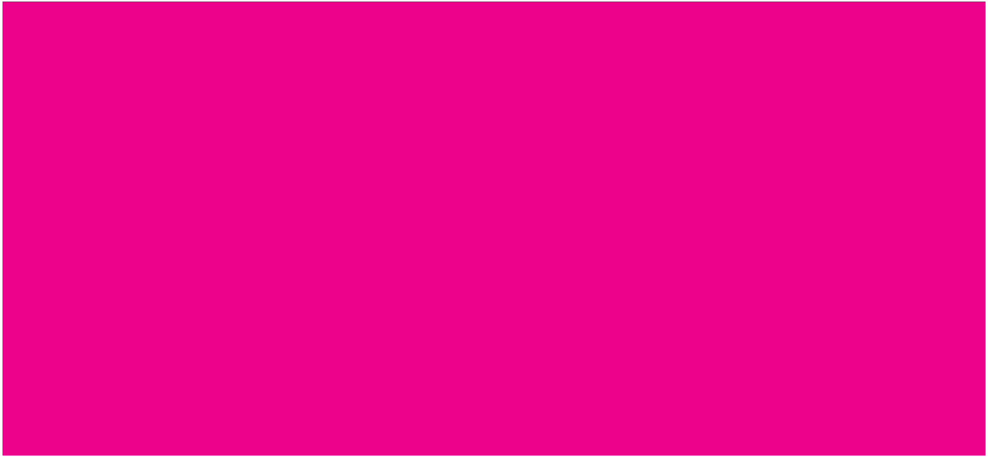
- Sie startet bei einem gegebenen Knoten und färbt die Knoten mit den Farben weiß, grau und schwarz.
- Sie berechnet einen gerichteten Tiefensuchwald, der bei einem ungerichteten Graph ein Baum ist.
- Sie ordnet jedem Knoten eine Anfangs- und eine Endzeit zu.
- Alle Zeiten sind verschieden.
- Die Kanten des Graphen werden als Baum-, Vorwärts-, Rückwärts- oder Querkanten klassifiziert.

Tiefensuche – Beispiel



- Die Kanten des Tiefensuchwaldes sind dick dargestellt.
- Ein Knoten ist anfangs weiß.
- Ein Knoten ist grau, während er aktiv ist.
- Danach wird er schwarz.

Noch ein Beispiel



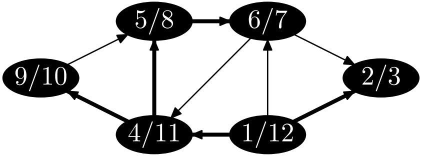
Java

```
public static⟨V⟩ void DFS(Graph⟨V⟩ G, Map⟨V, Integer⟩ d,
    Map⟨V, Integer⟩ f, Map⟨V, V⟩ p) {
  Map⟨V, Integer⟩ color = new HashMap⟨V, Integer⟩();
  for(V u : G.allNodes())
    color.put(u, WHITE);
  int time = 0;
  for(V u : G.allNodes())
    if(color.get(u) == WHITE)
      time = DFS(G, u, time, color, d, f, p);
}
```

Java

```
public static⟨V⟩ int DFS(Graph⟨V⟩ G, V u, int t, Map⟨V, Integer⟩ c,
        Map⟨V, Integer⟩ d, Map⟨V, Integer⟩ f, Map⟨V, V⟩ p) {
  d.put(u, ++t);
  c.put(u, GRAY);
  for(V v : G.neighbors(u))
    if(c.get(v) == WHITE) {
      p.put(v, u);
      t = DFS(G, v, t, c, d, f, p);
    }
  f.put(u, ++t);
  c.put(u, BLACK);
  return t;
}
```

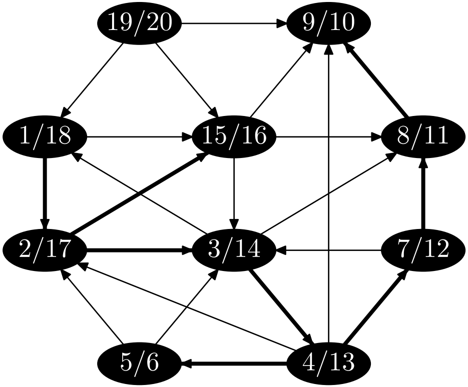
Taxonomie der Kanten



1. Eine Baumkante ist im DFS-Wald (geht von einem Knoten zu einem seiner Kinder im DFS-Wald).
2. Eine Vorwärtskante geht von einem Knoten zu einem seiner Nachfahren im DFS-Wald (aber nicht Kind).
3. Eine Rückwärtskante geht von einem Knoten zu einem seiner Vorfahren im DFS-Wald.
4. Eine Querkante verbindet zwei im DFS-Wald unvergleichbare Knoten.

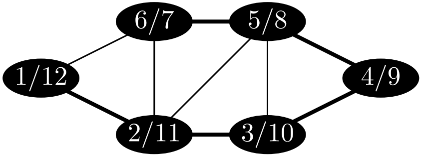
Frage: Welchen Typ hat jede Kante in diesem Beispiel?

Taxonomie der Kanten



Frage: Welchen Typ hat jede Kante in diesem Beispiel?

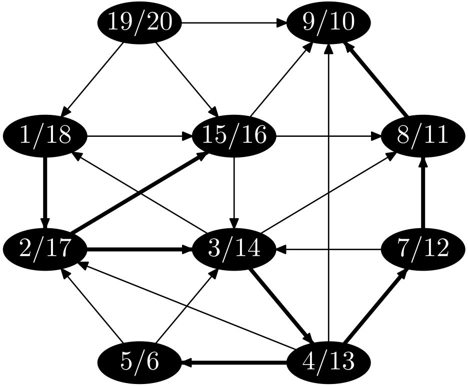
DFS – Ungerichtete Graphen



Wir erhalten immer einen Baum, wenn der Graph zusammenhängend ist.

Implementierung: Eine ungerichtete Kante wird durch Kanten in beide Richtungen dargestellt.

Taxonomie der Kanten



Betrachte Kante (u, v):

1. $d(u) < d(v)$ und $f(v) < f(u)$
   $\Longleftrightarrow$
   Baum- oder Vorwärtskante
2. $d(v) < d(u)$ und $f(u) < f(v)$
   $\Longleftrightarrow$
   Rückwärtskante
3. sonst Querkante

Tiefensuche

Theorem

Gegeben sei ein gerichteter Graph $G = (V, E)$ (in Adjazenzlistendarstellung). Durch Tiefensuche kann ein DFS-Wald inklusive der Funktionen $d\colon V \to \mathbf{N}$, $f\colon V \to \mathbf{N}$ in $O(|V| + |E|)$ Schritten (also in linearer Zeit) berechnet werden.

Beweis.

(Skizze) Solange ein Knoten grau ist, wird jede inzidente Kante einmal besucht. Jeder Knoten wechselt seine Farbe nur zweimal, jedesmal mit konstantem Aufwand. Jede Kante wird daher ebenfalls nur einmal besucht. $\square$

Zusammenhangskomponenten

Definition

Es sei $G = (V, E)$ ein ungerichteter Graph. Ein Pfad der Länge k von u_1 nach u_{k+1} ist eine Folge $(u_1, u_2), (u_2, u_3), \ldots, (u_k, u_{k+1})$ von Kanten aus E wobei $u_1, \ldots, u_{k+1}$ paarweise verschieden sind.

Wir sagen u und v sind zusammenhängend, wenn es einen Pfad von u nach v gibt.

Eine Menge $C \subseteq V$ ist eine Zusammenhangskomponente, wenn alle Knoten in C zusammenhängend sind und es keine echte Obermenge von C mit dieser Eigenschaft gibt.
(Alternativ: Die Zusammenhangskomponenten sind die Äquivalenzklassen der Relation „zusammenhängend“.)

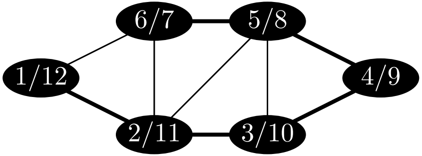
Zusammenhangskomponenten

Theorem

Die Zusammenhangskomponenten eines ungerichteten Graphen können in linearer Zeit gefunden werden.

Beweis.

Die Knoten, die jeweils in einem Aufruf der Tiefensuche schwarz werden, gehören zu einer Komponente. □



Finden von Kreisen

Theorem

Gegeben sei ein gerichteter Graph G. Dann können wir in linearer Zeit feststellen, ob G azyklisch ist (keine Kreise enthält).

Beweis.

Führe eine Tiefensuche auf G aus.

Behauptung: G ist genau dann azyklisch, wenn es keine Rückwärtskanten gibt.

$\Rightarrow$ Wenn es eine Rückwärtskante von u nach v gibt, dann gibt es auch einen Pfad von v nach u im DFS-Wald. Dies ist ein Kreis.

$\Leftarrow$ Angenommen es gibt einen Kreis. Sei u der Knoten auf dem Kreis mit minimalem $d(u)$ und v der Knoten auf dem Kreis vor u. Dann gilt $d(u) < d(v)$ und $f(v) < d(u)$. Also ist (v, u) eine Rückwärtskante. $\square$