Tiefensuche, Breitensuche

Tiefesuche

Weiterführende Literatur:

- *Qualifizierungsmaßnahme Informatik: Algorithmen und Datenstrukturen 6*, Seite 39-52 (PDF 32-45)
- *Wikipedia-Artikel „Tiefensuche“*
- Schneider, *Taschenbuch der Informatik*, Kapitel 6.2.2.2 Graphalgorithmen, Seite 185

Die Tiefensuche (englisch *depth-first search*, *DFS*) ist in der Informatik ein Verfahren zum Suchen von Knoten in einem Graphen.

depth-first search

DFS

Der Tiefendurchlauf ist das Standardverfahren zum Durchlaufen eines Graphen bei dem *jeder Knoten mindestens einmal* und *jede Kante genau einmal* besucht wird. Man geht vom jeweiligen Knoten *erst zu einem nicht besuchten Nachbarknoten* und setzt den Algorithmus dort *rekursiv* fort. Bei schon besuchten Knoten wird abgebrochen. Als Hilfsstruktur wird ein *Stack* (Stapelspeicher, Keller) verwendet. Der konkrete Durchlauf hängt von der Reihenfolge der Knoten in den Adjazenzlisten bzw. in der Adjazenzmatrix ab.[1]

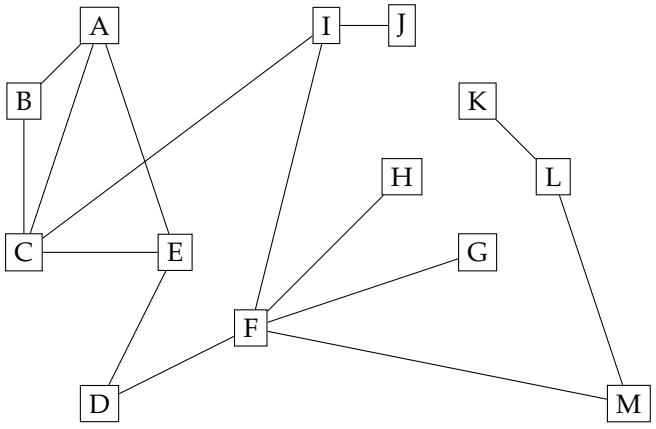
jeder Knoten mindestens einmal

jede Kante genau einmal

erst zu einem nicht besuchten Nachbarknoten

rekursiv

Stack




```
      add A [A]
del A       []
      add B [B]
      add C [C, B]
      add E [E, C, B]
del E       [C, B]
      add D [D, C, B]
del D       [C, B]
      add F [F, C, B]
del F       [C, B]
      add G [G, C, B]
```

[1] *Qualifizierungsmaßnahme Informatik: Algorithmen und Datenstrukturen 6*, Seite 40.

```
      add H [H, G, C, B]
      add I [I, H, G, C, B]
      add M [M, I, H, G, C, B]
del M       [I, H, G, C, B]
      add L [L, I, H, G, C, B]
del L       [I, H, G, C, B]
      add K [K, I, H, G, C, B]
del K       [I, H, G, C, B]
del I       [H, G, C, B]
      add J [J, H, G, C, B]
del J       [H, G, C, B]
del H       [G, C, B]
del G       [C, B]
del C       [B]
del B       []
```

Implementierung der Tiefensuche

- Eine Möglichkeit, abzuspeichern, welche Knoten bereits besucht wurden → Boolean-Array

- Eine Methode, die für uns diese Markierung der Knoten als besucht übernimmt (und somit die eigentliche Tiefensuche durchführt) → Knoten als besucht eintragen, existierende Nachbarknoten suchen und prüfen, ob diese bereits besucht wurden, falls nicht: diese durch rekursiven Aufruf besuchen

- Eine Methode, um die Tiefensuche zu starten → Wenn ein übergebnisener Startknoten existiert, dann müssen erst alle Knoten als nicht besucht markiert werden und dann vom Startknoten aus das Besuchen der Knoten gestartet werden

2

[2] *Qualifizierungsmaßnahme Informatik: Algorithmen und Datenstrukturen 6*, Seite 45.

Pseudocode: Tiefensuche mit explizitem Stack[3]

Algorithmus 1: Tiefensuche mit explizitem Stack

```
Data: G: Graph, k: Startknoten in G
S := leerer Stack;
lege k oben auf S;
markiere k;
while S nicht leer ist do
    a := entferne oberstes Element von S;
    bearbeite Knoten a;
    for alle Nachfolger n von a do
        if n noch nicht markiert then
            lege n oben auf S;
            markiere n;
        end
    end
end
```

```
3  import org.bschlangaul.graph.GraphAdjazenzMatrix;
4  import org.bschlangaul.helfer.Farbe;
5
6  import java.util.ArrayList;
7  import java.util.List;
8  import java.util.Stack;
9  import java.util.Vector;
10
11 /**
12  * nach Schulbuch: Informatik 1 Oberstufe Oldenbourg Verlag
13  */
14 public class TiefenSucheStapel extends GraphAdjazenzMatrix {
15
16   /**
17    * Der Schnappschuss wird entweder erstellt, nachdem ein Knoten besucht wurde,
18    * oder ein Knoten aus dem Stapel entfernt wurde.
19    */
20   class SchnappSchuss {
21     String besuchterKnoten;
22     String entnommenerKnoten;
23
24     public SchnappSchuss(Stack<String> stapel) {
25       this.kopiereStapel(stapel);
26     }
27
28     /**
29      * Eine Kopie des referenzierten Stapels als einfaches Feld.
30      */
31     Object[] stapel;
32
33     void kopiereStapel(Stack<String> stapel) {
34       this.stapel = stapel.toArray();
35     }
36
37     SchnappSchuss merkeBesuch(String knotenName) {
38       this.besuchterKnoten = knotenName;
```

[3] *Qualifizierungsmaßnahme Informatik: Algorithmen und Datenstrukturen 6*, Seite 51 (PDF 45).

```
39        return this;
40      }
41
42      SchnappSchuss merkeEntnahme(String knotenName) {
43        this.entnommenerKnoten = knotenName;
44        return this;
45      }
46    }
47
48    class Protokoll {
49      List<SchnappSchuss> schnappSchuesse;
50
51      /**
52       * Eine Referenze auf den vom Algorithmus verwendeten Stapel.
53       */
54      Stack<String> stapel;
55
56      public Protokoll(Stack<String> stapel) {
57        this.schnappSchuesse = new ArrayList<SchnappSchuss>();
58        this.stapel = stapel;
59      }
60
61      void merkeBesuch(String knotenName) {
62        schnappSchuesse.add(new SchnappSchuss(stapel).merkeBesuch(knotenName));
63      }
64
65      void merkeEntnahme(String knotenName) {
66        schnappSchuesse.add(new SchnappSchuss(stapel).merkeEntnahme(knotenName));
67      }
68    }
69
70    /**
71     * Liste der besuchten Knoten
72     */
73    private boolean[] besucht;
74
75    /**
76     * Stapel für die Tiefensuche
77     */
78    private Stack<String> stapel = new Stack<String>();
79    private Vector<String> route = new Vector<String>();
80
81    Protokoll protokoll = new Protokoll(stapel);
82
83    /**
84     * Die maximale Anzahl der Knoten wird dabei festgelegt.
85     *
86     * @param maximaleKnoten Anzahl der maximal möglichen Knoten
87     */
88    public TiefenSucheStapel(int maximaleKnoten) {
89      super(maximaleKnoten);
90      initialisiereTiefensuche(maximaleKnoten);
91    }
92
93    /**
94     * Die Adjazenzmatrix kann mit diesem Konstruktur im einfachen Graphenformat
95     * spezifiziert werden.
96     *
97     * @param einfachesGraphenFormat Ein String im einfachen Graphenformat.
98     */
99    public TiefenSucheStapel(String einfachesGraphenFormat) {
100     super(einfachesGraphenFormat);
```

```
101       initialisiereTiefensuche(gibKnotenAnzahl());
102     }
103
104     private void initialisiereTiefensuche(int maximaleKnoten) {
105       besucht = new boolean[maximaleKnoten];
106       route = new Stack<String>();
107     }
108
109     public void besuche(int knotenNummer) {
110       String name = gibKnotenName(knotenNummer);
111       besucht[knotenNummer] = true;
112       route.add(name);
113       stapel.push(name);
114       protokoll.merkeBesuch(name);
115       System.out.println(Farbe.rot("besucht: ") + name);
116       System.out.println(Farbe.grün("Stapel: ") + stapel.toString());
117     }
118
119     /**
120      * Durchlauf aller Knoten und Ausgabe auf der Konsole
121      *
122      * @param knotenNummer Nummer des Startknotens
123      */
124     public void besucheKnoten(int knotenNummer) {
125       besuche(knotenNummer);
126       // Stapel ausgeben
127       while (!stapel.isEmpty()) {
128         // oberstes Element des Stapels nehmen und in die Route einfügen
129         String knotenName = stapel.pop();
130         protokoll.merkeEntnahme(knotenName);
131         System.out.println(Farbe.gelb("Aus dem Stapel entfernen: ") + knotenName);
132
133         // alle nicht besuchten Nachbarn von w in den Stapel einfügen
134         for (int abzweigung = 0; abzweigung <= gibKnotenAnzahl() - 1; abzweigung++)
          ↪ {
135           if (matrix[gibKnotenNummer(knotenName)][abzweigung] != NICHT_ERREICHBAR
            ↪ && !besucht[abzweigung]) {
136             besuche(abzweigung);
137           }
138         }
139       }
140       // Route ausgeben
141       System.out.println("\n" + Farbe.gelb("Route: ") + route.toString());
142     }
143
144     /**
145      * Start der Tiefensuche
146      *
147      * @param startKnoten Bezeichnung des Startknotens
148      */
149     public void führeAus(String startKnoten) {
150       int startnummer;
151       startnummer = gibKnotenNummer(startKnoten);
152
153       if (startnummer != -1) {
154         for (int i = 0; i <= gibKnotenAnzahl() - 1; i++) {
155           besucht[i] = false;
156         }
157         besucheKnoten(startnummer);
158       }
159     }
160
```

```
161   public static void main(String[] args) {
162     TiefenSucheStapel ts = new TiefenSucheStapel(20);
163
164     ts.setzeKnoten("A");
165     ts.setzeKnoten("B");
166     ts.setzeKnoten("C");
167     ts.setzeKnoten("D");
168
169     ts.setzeKnoten("E");
170     ts.setzeKnoten("F");
171     ts.setzeKnoten("G");
172     ts.setzeKnoten("H");
173     ts.setzeKnoten("J");
174     ts.setzeKnoten("K");
175
176     ts.setzeUngerichteteKante("A", "B", 1);
177     ts.setzeUngerichteteKante("A", "C", 1);
178
179     ts.setzeUngerichteteKante("B", "A", 1);
180     ts.setzeUngerichteteKante("B", "D", 1);
181     ts.setzeUngerichteteKante("B", "E", 1);
182
183     ts.setzeUngerichteteKante("C", "A", 1);
184     ts.setzeUngerichteteKante("C", "F", 1);
185     ts.setzeUngerichteteKante("C", "G", 1);
186
187     ts.setzeUngerichteteKante("D", "B", 1);
188     ts.setzeUngerichteteKante("D", "H", 1);
189
190     ts.setzeUngerichteteKante("E", "B", 1);
191     ts.setzeUngerichteteKante("E", "F", 1);
192
193     ts.setzeUngerichteteKante("F", "C", 1);
194     ts.setzeUngerichteteKante("F", "E", 1);
195     ts.setzeUngerichteteKante("F", "G", 1);
196     ts.setzeUngerichteteKante("F", "J", 1);
197
198     ts.setzeUngerichteteKante("G", "C", 1);
199     ts.setzeUngerichteteKante("G", "F", 1);
200
201     ts.setzeUngerichteteKante("H", "D", 1);
202
203     ts.setzeUngerichteteKante("J", "F", 1);
204
205     ts.setzeUngerichteteKante("K", "F", 1);
206
207     ts.gibMatrixAus();
208
209     ts.führeAus("A");
210   }
211 }
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/graph/algorithmen/TiefenSucheStapel.java`

Breitensuche

Weiterführende Literatur:

- *Qualifizierungsmaßnahme Informatik: Algorithmen und Datenstrukturen 6*, Seite 53-64 (PDF 46-57)

- Schneider, *Taschenbuch der Informatik*, Kapitel 6.2.2.2 Graphalgorithmen, Seite 185
- *Wikipedia-Artikel „Breitensuche“*

Der Breitendurchlauf (englisch *breadth-first search*, *BFS*)[4] ist Verfahren zum Durchlaufen eines Graphen bei dem jeder Knoten genau einmal besucht wird. Man geht von einem Knoten *erst zu allen Nachbarknoten bevor deren Nachbarn* besucht werden. Bei schon besuchten Knoten wird abgebrochen. Als Hilfsstruktur wird eine *Queue* (*Warteschlange*) verwendet.

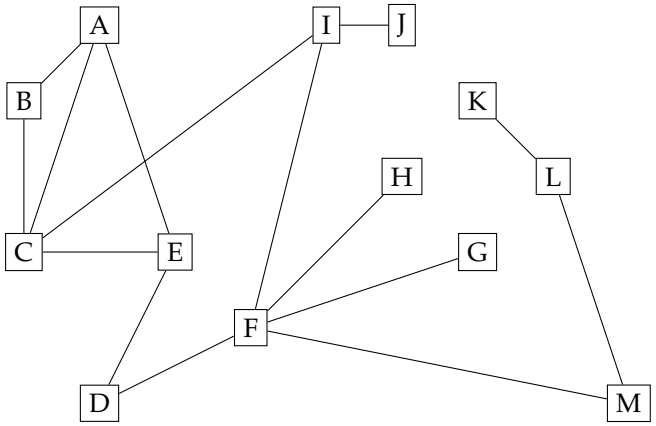
breadth-first search

BFS

erst zu allen Nachbarknoten bevor deren Nachbarn

Queue (Warteschlange)

Der konkrete Durchlauf hängt von der Reihenfolge der Knoten in den Adjazenzlisten ab.




```
      add A [A]
del A       []
      add B [B]
      add C [B, C]
      add E [B, C, E]
del B       [C, E]
del C       [E]
      add I [E, I]
del E       [I]
      add D [I, D]
del I       [D]
      add F [D, F]
      add J [D, F, J]
del D       [F, J]
del F       [J]
      add G [J, G]
      add H [J, G, H]
      add M [J, G, H, M]
del J       [G, H, M]
del G       [H, M]
del H       [M]
del M       []
      add L [L]
del L       []
```

[4] *Wikipedia-Artikel „Breitensuche“*.

```
      add K [K]
del K       []
```

Pseudocode Breitensuche mit Queue[5]

Algorithmus 2: Breitensuche mit Queue

```
Data: G: Graph, k: Startknoten in G
Q := leerer Queue;
füge k in Q;
markiere k;
while Q nicht leer do
    a := entferne vorderstes Element aus Q;
    bearbeite Knoten a;
    for alle Nachfolger n von a do
        if n noch nicht markiert then
            füge n hinten in Q;
            markiere n;
        end
    end
end
```

```
3  import org.bschlangaul.graph.GraphAdjazenzMatrix;
4  import org.bschlangaul.helfer.Farbe;
5
6  import java.util.ArrayList;
7  import java.util.List;
8  import java.util.Vector;
9
10 /**
11  * nach Schulbuch Informatik 1 Oberstufe Oldenbourg Verlag
12  */
13 public class BreitenSucheWarteschlange extends GraphAdjazenzMatrix {
14
15   /**
16    * Der Schnappschuss wird entweder erstellt, nachdem ein Knoten besucht wurde,
17    * oder ein Knoten aus dem Stapel entfernt wurde.
18    */
19   class SchnappSchuss {
20     String besuchterKnoten;
21     String entnommenerKnoten;
22
23     public SchnappSchuss(Vector<String> warteschlange) {
24       this.kopiereStapel(warteschlange);
25     }
26
27     /**
28      * Eine Kopie des referenzierten Stapels als einfaches Feld.
29      */
30     Object[] warteschlange;
31
32     void kopiereStapel(Vector<String> warteschlange) {
33       this.warteschlange = warteschlange.toArray();
```

[5] *Qualifizierungsmaßnahme Informatik: Algorithmen und Datenstrukturen 6*, Seite 64.

```
34      }
35
36      SchnappSchuss merkeBesuch(String knotenName) {
37        this.besuchterKnoten = knotenName;
38        return this;
39      }
40
41      SchnappSchuss merkeEntnahme(String knotenName) {
42        this.entnommenerKnoten = knotenName;
43        return this;
44      }
45    }
46
47    class Protokoll {
48      List<SchnappSchuss> schnappSchuesse;
49
50      /**
51       * Eine Referenze auf den vom Algorithmus verwendeten Stapel.
52       */
53      Vector<String> warteschlange;
54
55      public Protokoll(Vector<String> warteschlange) {
56        this.schnappSchuesse = new ArrayList<SchnappSchuss>();
57        this.warteschlange = warteschlange;
58      }
59
60      void merkeBesuch(String knotenName) {
61        schnappSchuesse.add(new
          ↪  SchnappSchuss(warteschlange).merkeBesuch(knotenName));
62      }
63
64      void merkeEntnahme(String knotenName) {
65        schnappSchuesse.add(new
          ↪  SchnappSchuss(warteschlange).merkeEntnahme(knotenName));
66      }
67    }
68
69    /**
70     * Liste der besuchten Knoten
71     */
72    private boolean[] besucht;
73
74    /**
75     * Eine Warteschlange für die Breitensuche
76     */
77    private Vector<String> warteschlange = new Vector<String>();
78    private Vector<String> route = new Vector<String>();
79
80    Protokoll protokoll = new Protokoll(warteschlange);
81
82    /**
83     * Die Adjazenzmatrix kann mit diesem Konstruktur im einfachen Graphenformat
84     * spezifiziert werden.
85     *
86     * @param einfachesGraphenFormat Ein String im einfachen Graphenformat.
87     */
88    public BreitenSucheWarteschlange(String einfachesGraphenFormat) {
89      super(einfachesGraphenFormat);
90      besucht = new boolean[gibKnotenAnzahl()];
91    }
92
93    public void besuche(int knotenNummer) {
```

```
94      String name = gibKnotenName(knotenNummer);
95      besucht[knotenNummer] = true;
96      route.add(name);
97      warteschlange.add(name);
98      protokoll.merkeBesuch(name);
99      System.out.println(Farbe.rot("besucht: ") + name);
100     System.out.println(Farbe.grün("Warteschlange: ") + warteschlange.toString());
101   }
102
103   /**
104    * Durchlauf aller Knoten und Ausgabe auf der Konsole
105    *
106    * @param knotenNummer Nummer des Startknotens
107    */
108   private void besucheKnoten(int knotenNummer) {
109     besuche(knotenNummer);
110
111     while (!warteschlange.isEmpty()) {
112       // oberstes Element der Liste nehmen
113       String knotenName = warteschlange.remove(0);
114       System.out.println(Farbe.gelb("Aus der Warteschlange entfernen: ") +
          ↪ knotenName);
115       protokoll.merkeEntnahme(knotenName);
116
117       // alle nicht besuchten Nachbarn von knotenName in die Liste einfügen
118       for (int abzweigung = 0; abzweigung <= gibKnotenAnzahl() - 1; abzweigung++)
          ↪ {
119         if (matrix[gibKnotenNummer(knotenName)][abzweigung] != NICHT_ERREICHBAR
            ↪ && !besucht[abzweigung]) {
120           besuche(abzweigung);
121         }
122       }
123     }
124     // Route ausgeben
125     System.out.println(Farbe.gelb("Route: ") + route.toString());
126   }
127
128   /**
129    * Start der Breitensuche
130    *
131    * @param startKnoten Bezeichnung des Startknotens
132    */
133   public void führeAus(String startKnoten) {
134     int startnummer;
135     startnummer = gibKnotenNummer(startKnoten);
136
137     if (startnummer != -1) {
138       for (int i = 0; i <= gibKnotenAnzahl() - 1; i++) {
139         besucht[i] = false;
140       }
141       besucheKnoten(startnummer);
142     }
143   }
144
145   public static void main(String[] args) {
146     BreitenSucheWarteschlange bs = new BreitenSucheWarteschlange(
147
          ↪ "a--e; a--f; a--s; b--c; b--d; b--h; c--d; c--h; c--s; d--h; e--f; f--s; g--s; h--s;");
148     bs.gibMatrixAus();
149     bs.führeAus("s");
150   }
151
```

```
152 }
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/graph/algorithmen/BreitenSucheWarteschlange.java`

Literatur

[1] *Qualifizierungsmaßnahme Informatik: Algorithmen und Datenstrukturen 6. Graphen*. `https://www.studon.fau.de/file2635324_download.html`.

[2] Uwe Schneider. *Taschenbuch der Informatik*. 7. Aufl. Hanser, 2012. ISBN: 9783446426382.

[3] *Wikipedia-Artikel „Breitensuche“*. `https://de.wikipedia.org/wiki/Breitensuche`.

[4] *Wikipedia-Artikel „Tiefensuche“*. `https://de.wikipedia.org/wiki/Tiefensuche`.