9. Quicksort

Wie bei vielen anderen Sortierverfahren (Bubblesort, Mergesort, usw.) ist auch bei Quicksort die Aufgabe, die Elemente eines Array $a[1..n]$ zu sortieren.

Quicksort ist ein Divide-and-Conquer-Verfahren.

Divide: Wähle ein Pivot-Element p (z.B. das letzte) und partitioniere $a[l..r]$ gemäß p in zwei Teile $a[l..i-1]$ und $a[i+1..r]$ (durch geeignetes Vertauschen der Elemente), wobei abschließend $a[i] = p$.

Conquer: Sortiere $a[l..i-1]$ und $a[i+1..r]$ rekursiv.

Algorithmus:

proc qs(a, l, r)
  if $l \geq r$ **then return fi**
  #ifdef Variante 2
    vertausche $a[$random$(l, r)]$ mit $a[r]$
  #endif
  $p := a[r]$
  $i := l$; $j := r$
  repeat
    while $i < j$ **and** $a[i] \leq p$ **do** $i++$ **od**
    while $i < j$ **and** $p \leq a[j]$ **do** $j--$ **od**
    if $i < j$ **then** vertausche $a[i]$ und $a[j]$ **fi**
  until $i = j$
  vertausche $a[i]$ und $a[r]$
  qs$(a, l, i-1)$
  qs$(a, i+1, r)$

Bemerkung:
Der oben formulierte Algorithmus benötigt pro Durchlauf für n zu sortierende Schlüssel n oder mehr Schlüsselvergleiche. Durch geschicktes Einstreuen von **if**-Abfragen kann man in jedem Fall mit $n-1$ Schlüsselvergleichen auskommen.

Komplexität von Quicksort:

- Best-case-Analyse: Quicksort läuft natürlich am schnellsten, falls die Partitionierung möglichst ausgewogen gelingt, im Idealfall also immer zwei gleich große Teilintervalle entstehen, das Pivot-Element ist dann stets der Median.

  Anzahl der Schlüsselvergleiche:

$$\leq \sum_{i=1}^{\log n} (n-1) = (n-1)\log n \approx n \log n$$

- Worst-case-Analyse: Z.B. bei einer aufsteigend sortierten Eingabe.

  Anzahl der Schlüsselvergleiche:

$$\Omega(n^2)$$

- Average-case: Da die Laufzeit von Quicksort sehr stark von den Eingabedaten abhängt, kann man die Frage stellen, wie lange der Algorithmus “im Mittel“ zum Sortieren von n Elementen braucht. Um jedoch überhaupt eine derartige Analyse durchführen zu können, muss man zuerst die genaue Bedeutung von “im Mittel“ festlegen. Eine naheliegende Annahme ist, dass alle möglichen Permutationen der Eingabedaten mit gleicher Wahrscheinlichkeit auftreten.

Satz 93

Die durchschnittliche Anzahl von Schlüsselvergleichen von Quicksort beträgt unter der Annahme, dass alle Permutationen für die Eingabe gleichwahrscheinlich sind, höchstens

$$C_n = 2(n+1)(H_{n+1}-1) \approx 2n\ln n - 0.846n + o(n) \approx 1.386n\log n$$

wobei $H_n := \sum_{i=1}^{n} i^{-1}$ *die* n*-te Harmonische Zahl ist.*

Beweis:

(Variante mit $n-1$ Vergleichen pro Durchlauf)
Sei C_n die Anzahl der Vergleiche bei einem Array der Länge n.
$C_0 = C_1 = 0$.

$$C_n = n - 1 + \frac{1}{n}\sum_{j=1}^{n}(C_{j-1} + C_{n-j})$$

Beweis (Forts.):

Da

i) (in beiden Varianten) das j-kleinste Element bestimmt wird und

ii) auch für die rekursiven Aufrufe wieder alle Eingabepermutationen gleichwahrscheinlich sind:

$$\Rightarrow \qquad C_n = n - 1 + \frac{2}{n}\sum_{j=0}^{n-1} C_j;$$

$$nC_n = n^2 - n + 2\sum_{j=0}^{n-1} C_j;$$

$$(n-1)C_{n-1} = (n-1)^2 - (n-1) + 2\sum_{j=0}^{n-2} C_j;$$

Beweis (Forts.):

$$
\begin{aligned}
nC_n - (n-1)C_{n-1} &= 2n - 1 - 1 + 2C_{n-1}; \\
nC_n &= 2n - 2 + (n+1)C_{n-1}; \qquad / \frac{1}{n(n+1)} \\
\frac{C_n}{n+1} &= 2\frac{n-1}{n(n+1)} + \frac{C_{n-1}}{n} = \\
&= 2\frac{n-1}{n(n+1)} + 2\frac{n-2}{(n-1)n} + \frac{C_{n-2}}{n-1} = \\
&= 2\frac{n-1}{n(n+1)} + \cdots + \frac{C_2}{3} = \\
&= 2\left[\sum_{j=2}^{n} \frac{j-1}{j(j+1)}\right] =
\end{aligned}
$$

Beweis (Forts.):

$$= 2\left[\sum_{j=2}^{n}\left(\frac{j}{j(j+1)} - \frac{1}{j(j+1)}\right)\right] =$$

$$= 2\left[H_{n+1} - \frac{3}{2} - \sum_{j=2}^{n}\left(\frac{1}{j} - \frac{1}{j+1}\right)\right] =$$

$$= 2\left[H_{n+1} - \frac{3}{2} - \left(\frac{1}{2} - \frac{1}{n+1}\right)\right] =$$

$$= 2\left[H_{n+1} - 2 + \frac{1}{n+1}\right]$$

$\Rightarrow C_n = 2(n+1)\left(H_{n+1} - 2 + \frac{1}{n+1}\right)$;
Mit $H_n \approx \ln n + 0.57721\ldots + o(1)$
$\Rightarrow C_n \approx 2n\ln n - 4n + o(n) \approx 1.386 n \log n$ □

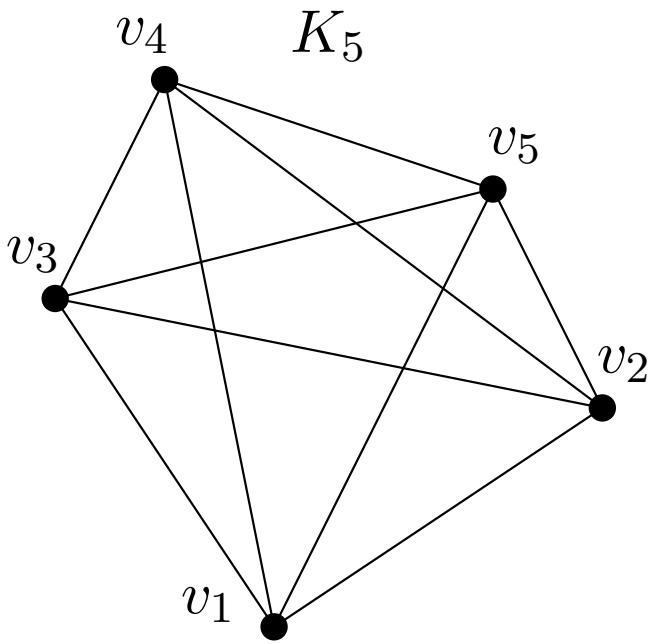
Kapitel IV Minimale Spannbäume

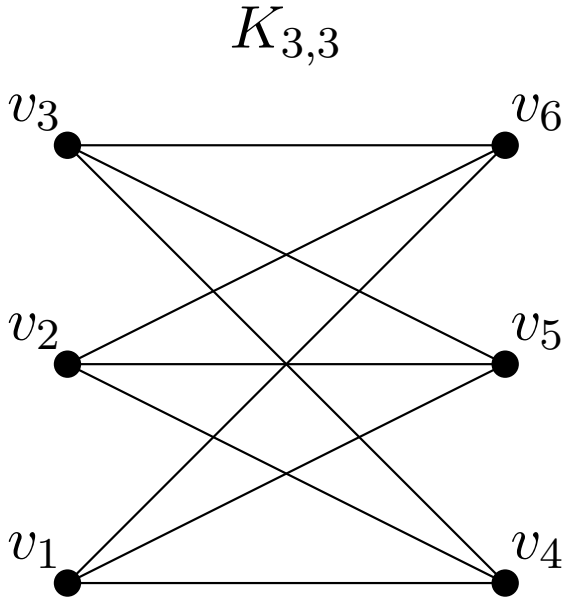
1. Grundlagen

Ein Graph $G = (V, E)$ besteht aus einer Menge V von Knoten und einer Menge E von Kanten. Wir werden nur endliche Knoten- (und damit auch Kanten-) Mengen betrachten. Die Anzahl der Knoten bezeichnen wir mit n ($|V| = n$), die Anzahl der Kanten mit m ($|E| = m$). Eine gerichtete Kante mit den Endknoten u und v wird mit (u, v), eine ungerichtete mit $\{u, v\}$ notiert. Eine Kante $(v, v) \in E$ (bzw. $\{v, v\}$) heißt Schlinge. Falls E eine Multimenge ist, spricht man von Mehrfachkanten. Kanten (u, v) und (v, u) heißen antiparallel. Graphen ohne Schlingen und Mehrfachkanten heißen einfache Graphen (engl. simple). Für einfache ungerichtete Graphen gilt daher:

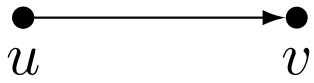
$$E \subseteq \binom{V}{2} := \{X \subseteq V, |X| = 2\}$$

Graphische Darstellung:





Ist $E \subseteq V \times V$, dann heißt G gerichteter Graph (engl. digraph).



Der zu G gehörige ungerichtete Graph ist $G' = (V, E')$. E' erhält man, indem man in E die Richtungen weglässt und Mehrfachkanten beseitigt.
Sei $v \in V$. Unter der Nachbarschaft
$N(v) := \{w; (v, w)$ oder $(w, v) \in E\}$ eines Knotens v versteht man die Menge der direkten Nachbarn von v. Der Grad eines Knotens ist definiert als:

$$\deg(v) = \begin{cases} |N(v)| & ; \text{falls } G \text{ ungerichtet und} \\ \text{indeg}(v) + \text{outdeg}(v) & ; \text{falls } G \text{ gerichtet.} \end{cases}$$

Dabei ist $\text{indeg}(v)$ die Anzahl aller Kanten, die v als Endknoten, und $\text{outdeg}(v)$ die Anzahl aller Kanten, die v als Anfangsknoten haben.

Beobachtung: Für einfache (gerichtete oder ungerichtete) Graphen gilt

$$\sum_{v \in V} \deg(v) = 2|E| \,.$$

Korollar 94

In jedem (einfachen) Graphen ist die Anzahl der Knoten mit ungeradem Grad eine gerade Zahl.

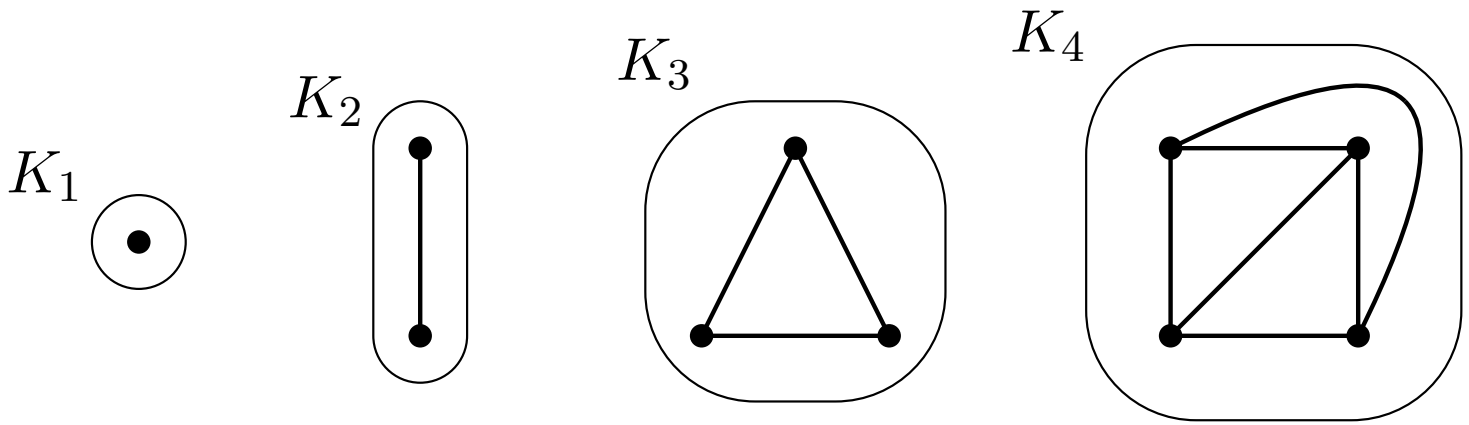
Das Komplement $\bar{G} = (V, \binom{V}{2} \setminus E)$ eines Graphen $G = (V, E)$ besitzt die gleiche Knotenmenge V und hat als Kantenmenge alle Kanten des vollständigen Graphen ohne die Kantenmenge E.

Ein Graph $H = (V', E')$ heißt Teilgraph (aka. Subgraph) von $G = (V, E)$, falls $V' \subseteq V$ und $E' \subseteq E$. H heißt (knoten-) induzierter Teilgraph, falls H Teilgraph von G ist und

$$E' = E \cap \binom{V'}{2}.$$

Ein Kantenzug (oder Pfad) ist eine Folge $e_1 := \{v_0, v_1\}, \ldots, e_l := \{v_{l-1}, v_l\}$. v_0 und v_l sind die Endknoten, l ist die Länge des Kantenzuges. Sind bei einem Pfad alle v_i (und damit erst recht alle e_i) verschieden, so sprechen wir von einem einfachen Pfad. Ein Kantenzug mit $v_l = v_0$ heißt Zykel oder Kreis. Ein Kreis, in dem alle v_i verschieden sind, heißt einfacher Kreis.

Ein (ungerichteter) Graph G heißt zusammenhängend, wenn es für alle $u, v \in V$ einen Pfad gibt, der u und v verbindet. Ein (knoten-)maximaler induzierter zusammenhängender Teilgraph heißt (Zusammenhangs-)Komponente.



Ein Graph G heißt azyklisch, wenn er keinen Kreis enthält. Wir bezeichnen einen solchen ungerichteten Graphen dann als Wald. Ist dieser auch zusammenhängend, so sprechen wir von einem Baum. Ist der Teilgraph $T = (V, E') \subseteq G = (V, E)$ ein Baum, dann heißt T ein Spannbaum von G.

Satz 95

Ist $T = (V, E)$ ein Baum, dann ist $|E| = |V| - 1$.

Beweis:

Induktion über die Anzahl n der Knoten:
$n = 0, 1$: klar.
$n \to n + 1$: Sei $|V| = n + 1$. Da T zusammenhängend ist, ist $\deg(v) \geq 1$ für alle $v \in V$. T muss einen Knoten v mit $\deg(v) = 1$ enthalten, denn ansonsten würde, wie wiederum eine einfache Induktion zeigt, T einen Kreis enthalten. Wende nun die Induktionsannahme auf den durch $V - \{v\}$ induzierten Teilgraphen an. □

Korollar 96

Ein (ungerichteter) Graph G ist zusammenhängend, gdw G einen Spannbaum hat.

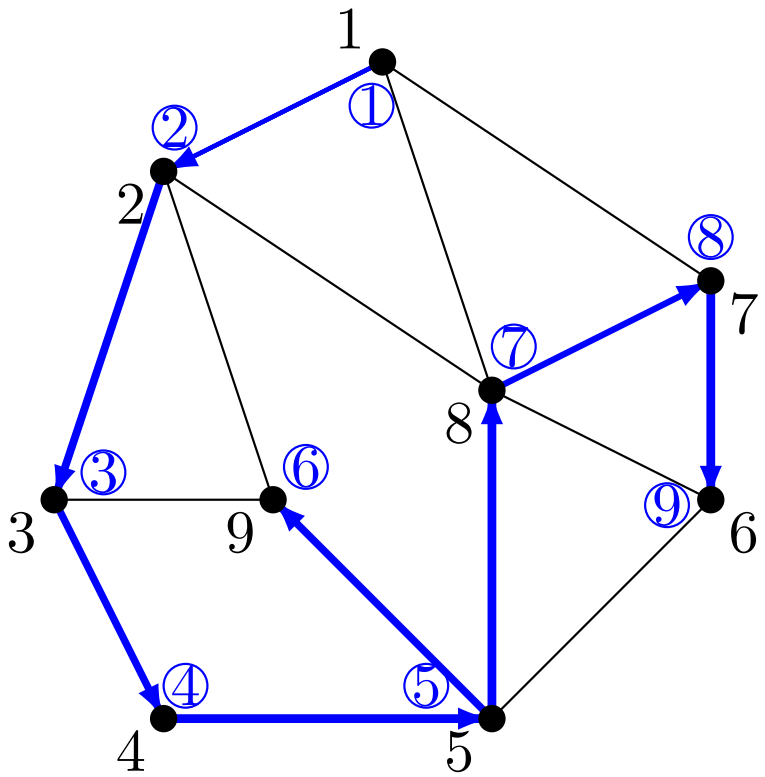
2. Traversierung von Graphen

Sei $G = (V, E)$ ein ungerichteter Graph. Anhand eines Beipiels betrachten wir die zwei Algorithmen DFS (Tiefensuche) und BFS (Breitensuche).

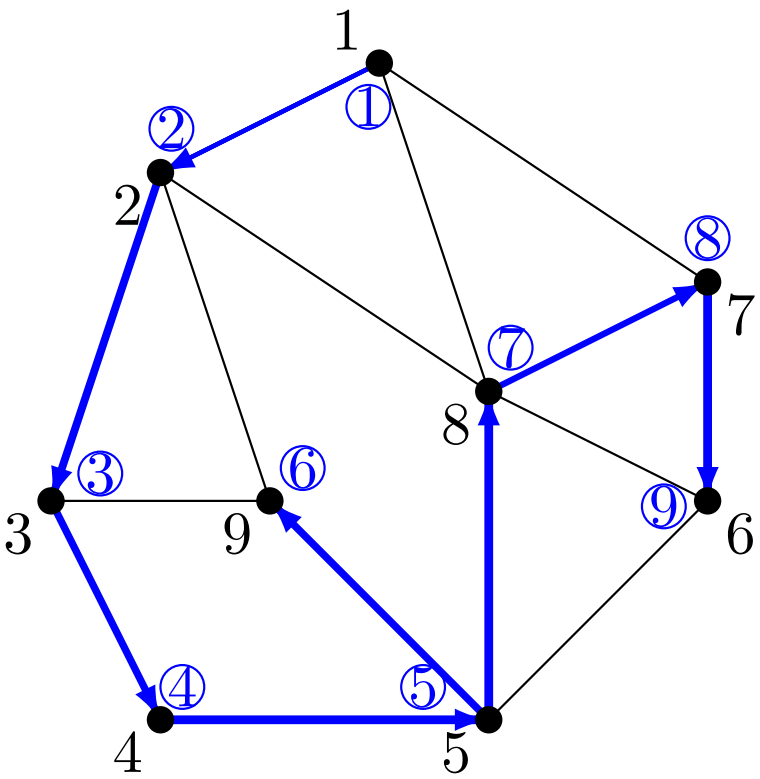
2.1 DFS-Algorithmus

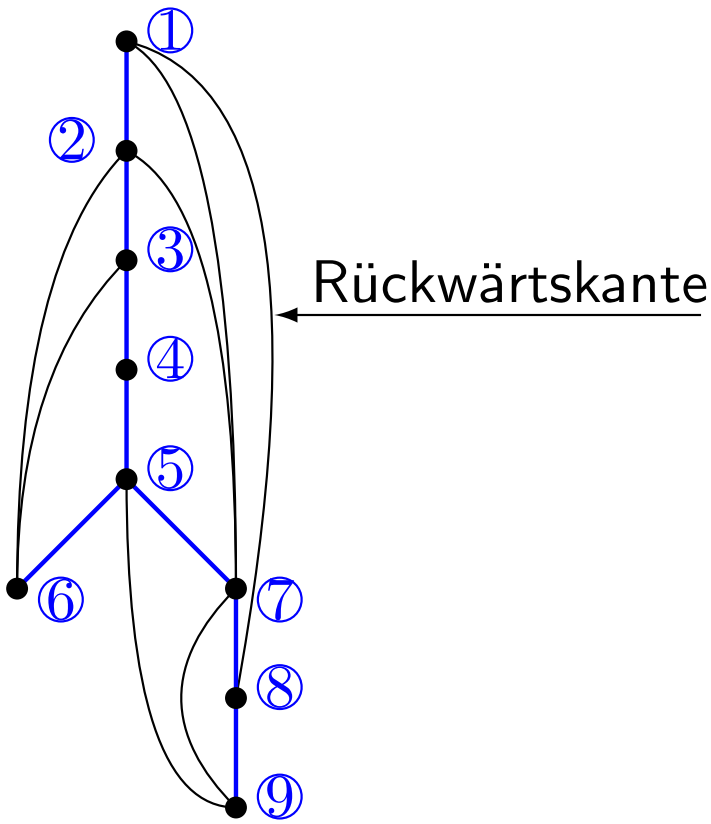
```
while ∃ unvisited v do
    r := pick (random) unvisited node
    push r onto stack
    while stack ≠ ∅ do
        v := pop top element
        if v unvisited then
            mark v visited
            push all neighbours of v onto stack
            perform operations DFS_Ops(v)
        fi
    od
od
```

Beispiel 97



Beobachtung: Die markierten Kanten bilden einen Spannbaum:

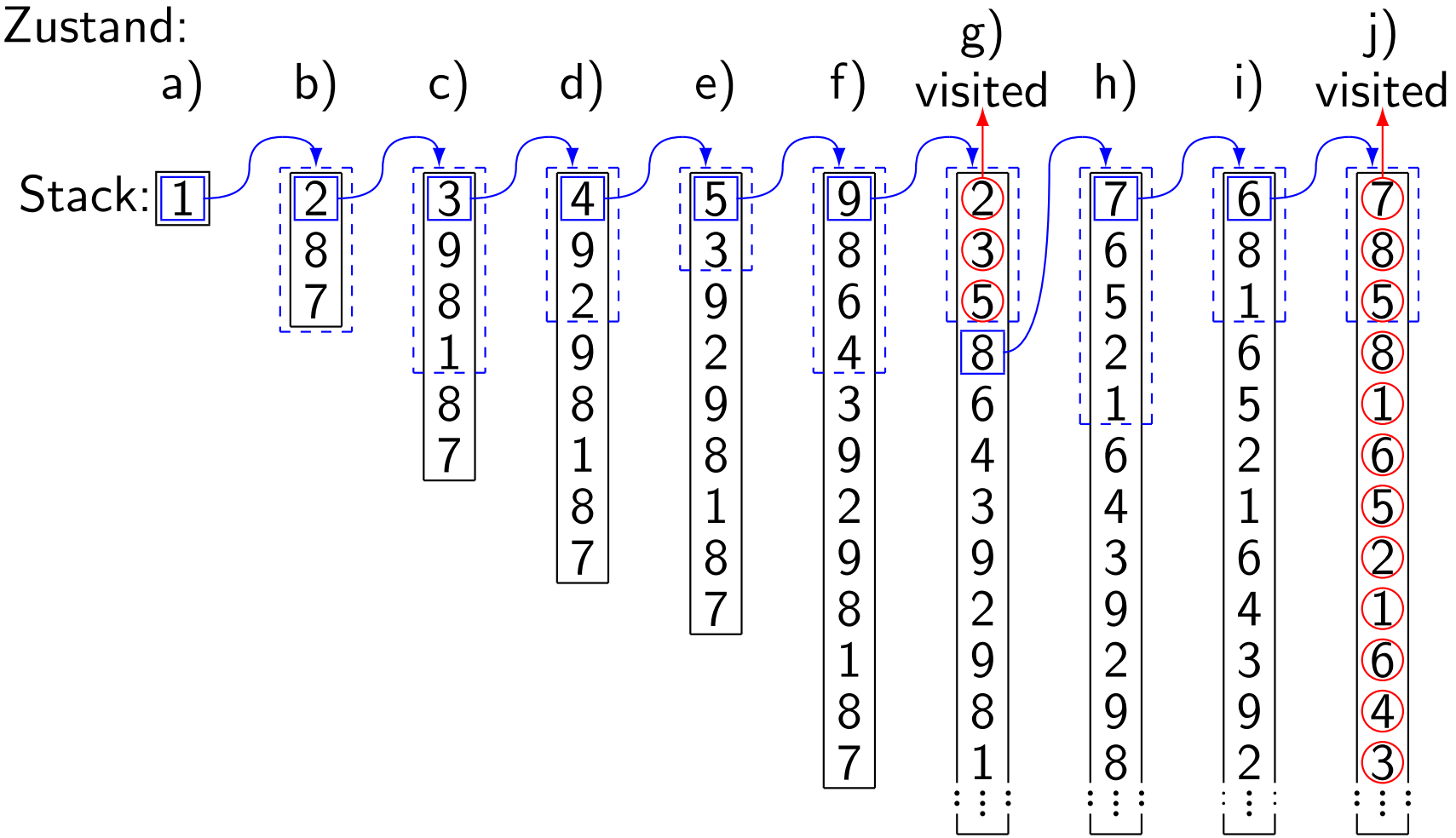




Folge der Stackzustände

: oberstes Stackelement : Nachbarn

: schon besuchte Knoten

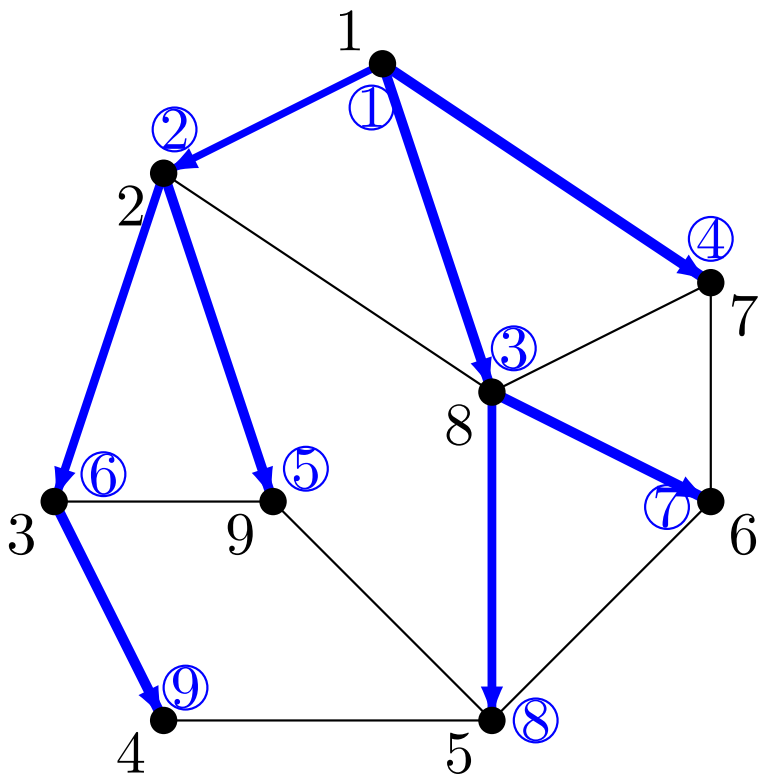


Wir betrachten den Stackzustand:
Im Zustand g) sind die Elemente 2, 3 und 5 als visited markiert (siehe Zustände b), c) und e)). Deswegen werden sie aus dem Stack entfernt, und das Element 8 wird zum obersten Stackelement. Im Zustand j) sind alle Elemente markiert, so dass eins nach dem anderen aus dem Stack entfernt wird.

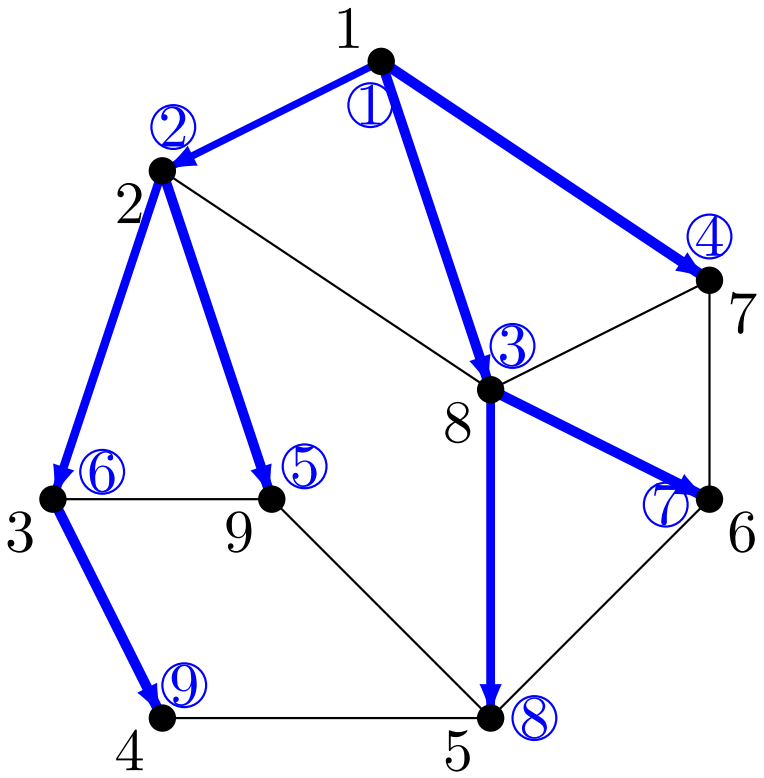
2.2 BFS-Algorithmus

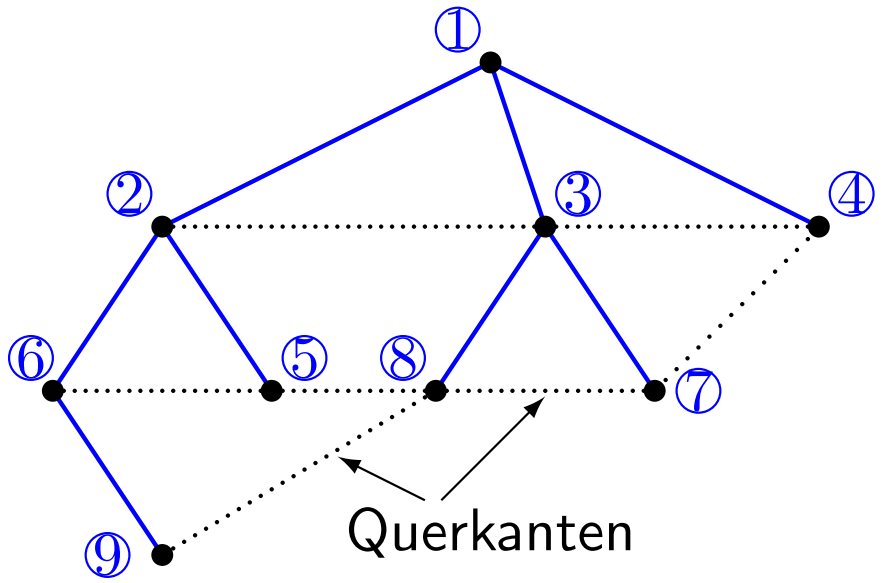
while $\exists$ unvisited v **do**
    $r :=$ pick (random) unvisited node
    push r into (end of) queue
    while queue $\neq \emptyset$ **do**
        $v :=$ remove front element of queue
        if v unvisited **then**
            mark v visited
            append all neighbours of v to end of queue
            perform operations BFS_Ops(v)
        fi
    od
od

Beispiel 98

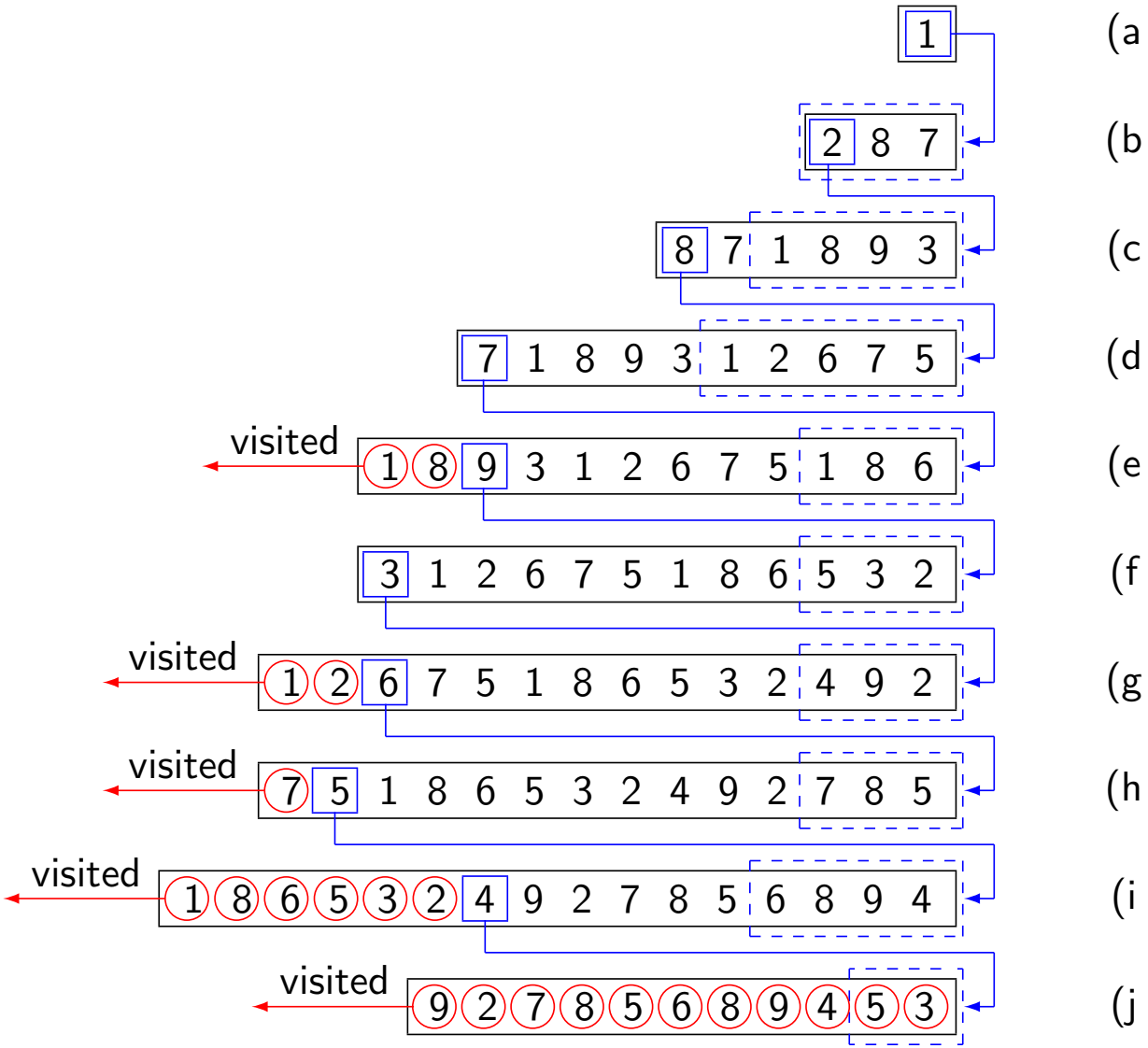


Beobachtung: Die markierten Kanten bilden einen Spannbaum:





Folge der Queuezustände



Wir betrachten die Folge der Queuezustände. Wiederum bedeutet die Notation:

 : vorderstes Queue-Element

 : schon besuchte Knoten

 : Nachbarn

Im Zustand e) sind die Elemente 1 und 8 als visited markiert (siehe Zustände a) und c)). Deswegen werden sie aus der Queue entfernt, und so wird das Element 9 das vorderste Queueelement. Das gleiche passiert in den Zuständen g), h) und i). Im Zustand j) sind alle Elemente markiert, so dass sie eins nach dem anderen aus der Queue entfernt werden.