1 Exploration geordneter Wurzelbäume

Ein *geordneter Wurzelbaum* ist ein Baum, bei welchem ein Knoten als Wurzel ausgezeichnet wird und bei welchem die Kinder eines Knotens linear, sagen wir von links nach rechts, geordnet werden.[1] Die rekursive Definition eines geordneten Wurzelbaumes ist wie folgt:

1. Ein Baum mit einem einzigen Knoten (wie in Abb. 1(a) dargestellt) ist ein geordneter Wurzelbaum.

2. Wenn $T_1, \ldots, T_k$ geordnete Wurzelbäume sind, dann ist auch der Baum mit Wurzel r und „Unterbäumen“ $T_1, \ldots, T_k$ (wie in Abb. 1(b) dargestellt) ein geordneter Wurzelbaum.

Wenn jeder innere Knoten maximal zwei Kinder hat (ein linkes und/oder ein rechtes), dann spricht man auch von einem binären Baum. Ein binärer Baum T heißt „Suchbaum“, wenn der numerische Wert, der in einem Knoten v von T gespeichert ist, größer ist als alle im linken Teilbaum von v gespeicherten numerischen Werte und kleiner als alle im rechten Teilbaum von v gespeicherten numerischen Werte. Ein Beispiel für einen Suchbaum ist in Abb. 2 zu sehen. Die jeweils gespeicherten numerischen Werte werden auch als „Schlüssel“ oder als „Schlüsselwerte“ bezeichnet. Die Organisation eines Suchbaumes ermöglicht ein effizientes Aufspüren eines Knotens mit gegebenem Schlüssel.

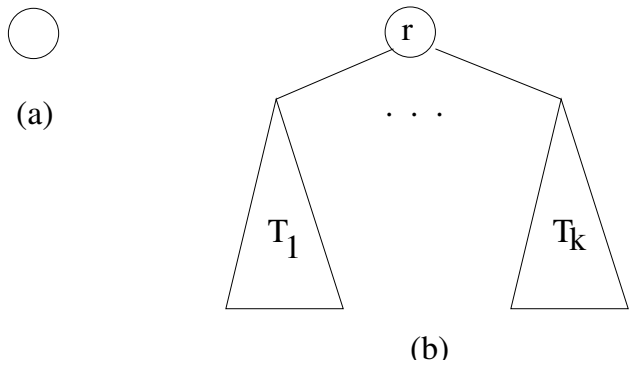


Abbildung 1: Illustration der rekursiven Definition eines geordneten Wurzelbaumes

In vielen Anwendungen müssen die Knoten eines geordneten Wurzelbaums systematisch, einer nach dem anderen, durchlaufen werden. Im Englischen spricht man von einem „traversal“ und unterscheidet die folgenden (rekursiv definierten) Durchlaufstrategien (vgl. mit Abb. 1):

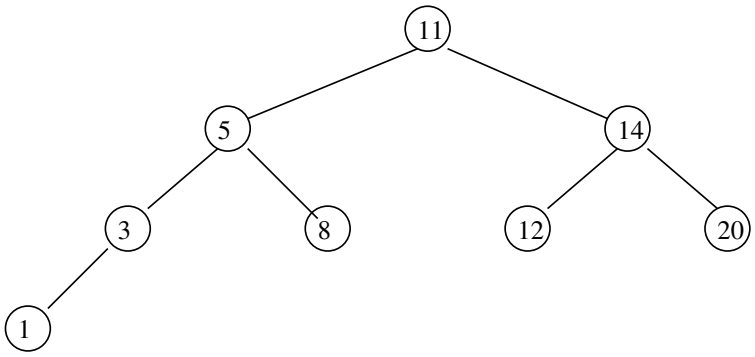
Durchlaufen in Präordnung (preorder traversal): Durchlaufe zuerst r und dann (jeweils in Präordnung) die Unterbäume $T_1, \ldots, T_k$ (in ebendieser Reihenfolge).

[1] Die Terminologie mit „Kindern“, „Vater“, „Großvater“, „Vorfahr“, „Nachkomme“ usw. ist wie bei Stammbäumen.

Durchlaufen in Postordnung (postorder traversal): Durchlaufe zuerst (jeweils in Postordnung) die Unterbäume $T_1, \ldots, T_k$ (in ebendieser Reihenfolge) und dann r.

Durchlaufen in Inordnung (inorder traversal): Diese Knotenreihenfolge ist nur für $k = 2$ erklärt: durchlaufe zuerst (in Inordnung) T_1, dann r und schließlich (in Inordnung) T_2.

Wenn zum Beispiel die Daten in einem geordneten Wurzelbaum „top down“ (von der Wurzel ausgehend blätterwärts) aktualisiert werden sollen, dann bietet sich ein „preorder traversal“ an. Werden die Daten „bottom-up“ aktualisiert (von den Blättern ausgehend wurzelwärts), so bietet sich ein „postorder traversal“ an.



Inorder Traversal:
1 , 3 , 5 , 8 , 11 , 12 , 14 , 20

Preorder Traversal:
11 , 5 , 3 , 1 , 8 , 14 ,12 , 20

Postorder Traversal:
1 , 3 , 8 , 5 , 12 , 20 , 14 , 11

Abbildung 2: Ein (Such-)Baum und seine „traversals“.

Beobachtung: Inorder Traversal angewendet auf einen Suchbaum liefert eine sortierte Reihenfolge der Schlüsselwerte.

2 Exploration von (ungerichteten) Graphen

Wir schildern zunächst ein generisches[2] Verfahren und beschreiben im Anschluss, wie die populären Explorationstechniken

- Tiefensuche (= Depth First Search = DFS)
- Breitensuche (= Breadth First Search = BFS)

daraus hervorgehen. Am Ende des Abschnittes diskutieren wir ein paar naheliegende Erweiterungen des simplen DFS- bzw. BFS-Verfahrens.

[2] allgemein gehaltenes, nicht detailliert ausimplementiertes

2.1 Exploration von einem Startknoten aus

Wie oben bereits angekündigt beginnen wir mit einem simplen generischen Verfahren.

Eingabe: Graph $G = (V, E)$, Startknoten $s \in V$

Ausgabe: Spannbaum der s enthaltenden Zusammenhangskomponente

Datenstrukturen:
- Eingabegraph in Adjazenzlistendarstellung
- Markierung der Knoten mit „alt“ (bereits besucht) oder „neu“ (noch unbesucht)
- Liste L bereits besuchter Knoten, deren Nachbarschaftsliste noch nicht vollständig durchforstet wurde
- Knotenzeiger zur Darstellung des auszugebenden Spannbaumes (Wurzel s, Zeiger zur Wurzel hin orientiert, **nil** bezeichnet den „Nullzeiger“)

Methode:
1. $L := \{s\}$; markiere s „alt“; Zeiger(s) := **nil**;
2. Markiere alle $v \in V \setminus \{s\}$ „neu“;
3. `--- evtl. weitere Instruktionen ---`
4. Wiederhole folgende Schritte bis L leer ist:
   (a) Sei v ein Knoten aus L.
   (b) **Fall 1:** v besitzt einen „neu“ markierten Nachbarn w
      i. Füge w in L ein und markiere w „alt“;
      ii. Zeiger(w) := v;
      iii. `--- evtl. weitere Instruktionen ---`

      Fall 2: v besitzt keinen „neu“ markierten Nachbarn w
      i. Entferne v aus L;
      ii. `--- evtl. weitere Instruktionen ---`

Die geschilderte Methode heißt *Tiefensuche*, *Depth First Search* oder kurz *DFS*, wenn wir die Liste L als STACK verwalten, d.h., L wird nach dem LIFO-Prinzip[3] organisiert.
Sie heißt *Breitensuche*, *Breadth First Search* oder kurz *BFS*, wenn wir die Liste L als QUEUE verwalten, d.h., L wird nach dem FIFO-Prinzip[4] organisiert.

Technische Vereinbarung: Wir nehmen im Folgenden an, dass im Rahmen einer Graphexploration die Nachbarn eines Knotens in der Reihenfolge aufsteigender Knotennummern inspiziert werden. Dies hat den Vorteil, dass die Beispielläufe, die wir betrachten werden, stets zu einem eindeutigen Ergebnis führen.

Wir betrachten als Beispiel den Graphen in Abb. 3 mit Startknoten 1. DFS führt zu folgender „Evolution“ des STACK L (wobei jede Spalte einen „Schnappschuss“ des STACK darstellt und Elemente stets oben eingefügt oder entnommen werden):

[3] LIFO = Last In First Out
[4] FIFO = First In First Out

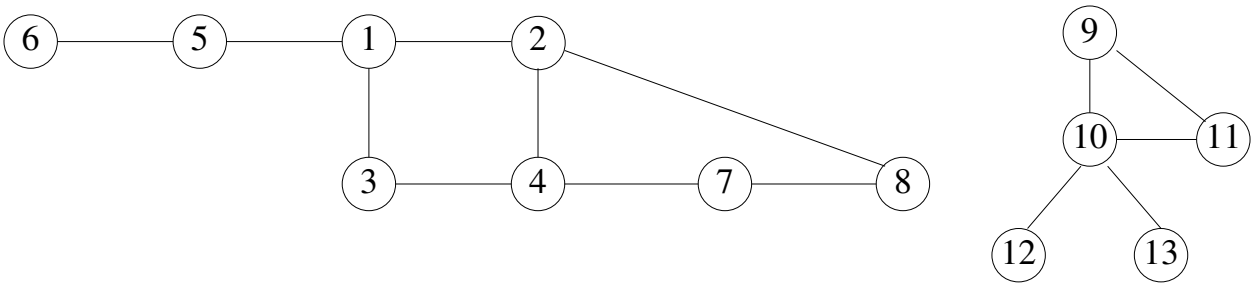


Abbildung 3: ein Graph mit zwei Zusammenhangskomponenten

						8									
			3		7	7	7								
		4	4	4	4	4	4	4				6			
	2	2	2	2	2	2	2	2	2		5	5	5		
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	STACK leer

Aus den dynamischen Veränderungen des STACK lassen sich die aktuellen Knotenmarkierungen und die aktuellen Knotenzeiger leicht ablesen:

- Zu jedem Zeitpunkt sind genau die Knoten „alt“ markiert, die bereits in den STACK aufgenommen wurden.
- Wann immer im STACK ein Knoten b auf einen Knoten a „gestapelt“ wird, muss ein Zeiger von b nach a gesetzt werden.

Die im Verlauf der DFS gesetzten Knotenzeiger sind in Abb. 4(a) zu besichtigen. Diese Zeiger repräsentieren den DFS-Spannbaum der Zusammenhangskomponente mit Startknoten 1. Derselbe Spannbaum ist in einem etwas ansprechenderen Layout nochmals in Abb. 4(b) zu sehen.

Wenn wir auf dem Eingabegraphen in Abb. 3 mit Startknoten 1 eine BFS durchführen, entwickelt sich die QUEUE wie folgt (wobei jede Spalte einen „Schnappschuss“ der QUEUE darstellt und Elemente stets oben eingefügt und unten entnommen werden)[5]:

						8									
			5		4	4	8		6		7				
		3	3	5	5	5	4	8	8	6	6	7			
	2	2	2	3	3	3	5	4	4	8	8	6	7		
1	1	1	1	2	2	2	3	5	5	4	4	8	6	6	QUEUE leer

Aus den dynamischen Veränderungen der QUEUE lassen sich die aktuellen Knotenmarkierungen und die aktuellen Knotenzeiger leicht ablesen:

[5] „oben“ entspricht also „hinten in der Warteschlange“; „unten“ entspricht „vorne in der Warteschlange“.

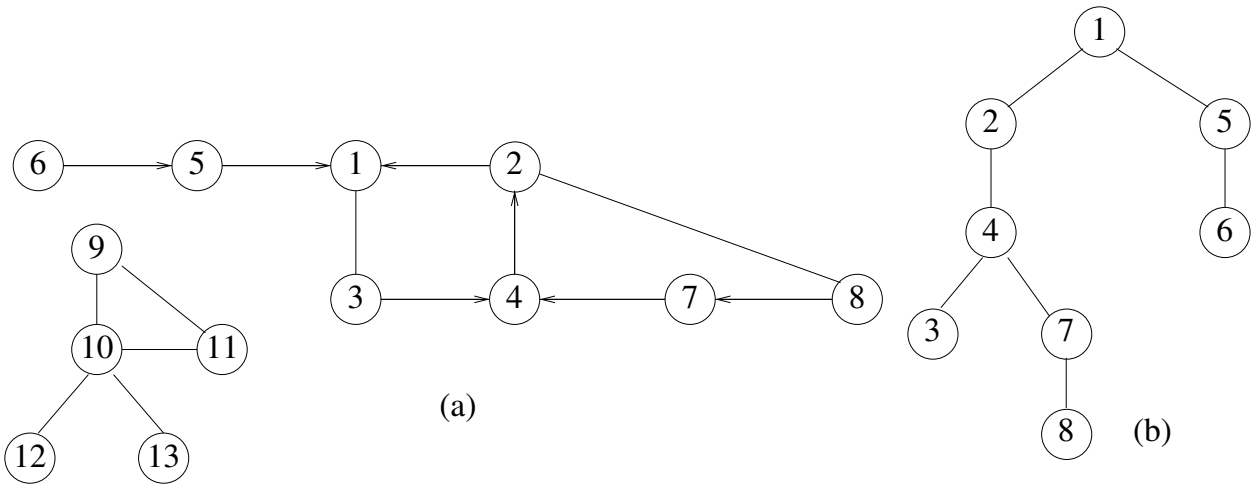


Abbildung 4: (a) der Eingabegraph ergänzt um die im Verlauf der DFS gesetzten Zeiger (b) anderes Layout des durch die Zeiger repräsentierten Spannbaums

- Zu jedem Zeitpunkt sind genau die Knoten „alt“ markiert, die bereits in die QUEUE aufgenommen wurden.
- Wann immer zuoberst ein Knoten b eingefügt wird während zuunterst ein Knoten a liegt, muss ein Zeiger von b nach a gesetzt werden.

Die im Verlauf der BFS gesetzten Knotenzeiger sind in Abb. 5(a) zu besichtigen. Diese Zeiger repräsentieren den BFS-Spannbaum der Zusammenhangskomponente mit Startknoten 1. Derselbe Spannbaum ist in einem etwas ansprechenderen Layout nochmals in Abb. 5(b) zu sehen.

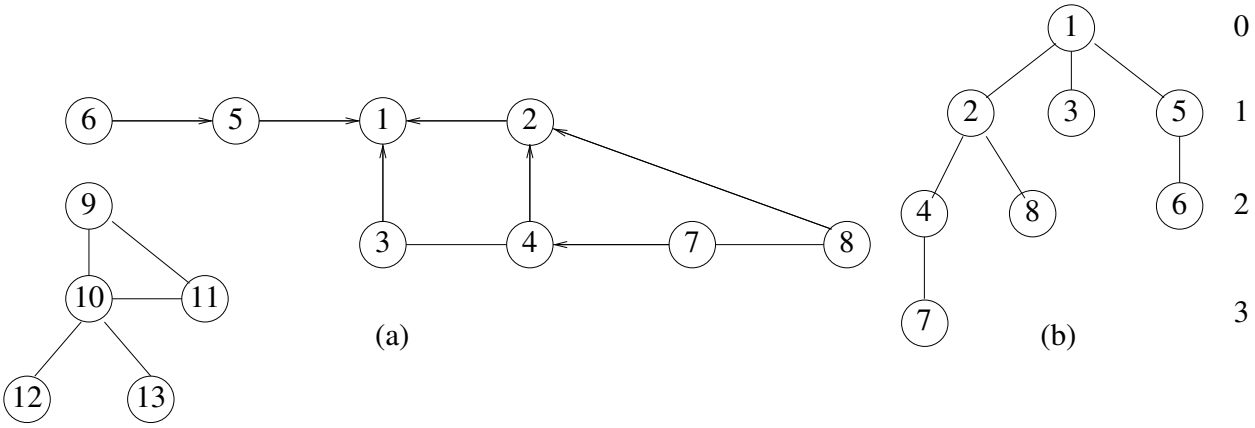


Abbildung 5: (a) der Eingabegraph ergänzt um die im Verlauf der BFS gesetzten Zeiger (b) anderes Layout des durch die Zeiger repräsentierten Spannbaums

2.2 Erweiterung 1: Vollständige Exploration eines Graphen

Bisher haben wir nur beschrieben, wie die Zusammenhangskomponente exploriert wird, welche den Startknoten s des Graphen enthält. Die Ausgabe bestand aus einem Spannbaum für diese Komponente. Falls der Graph nicht zusammenhängend ist, blieben unexplorierte Komponenten übrig. Es ist aber einfach, um das generische Verfahren herum ein Hauptprogramm zu „stricken“, welches die Knoten der Reihe nach inspiziert und auf einem „neu“ markierten Knoten u das generische Verfahren mit u als Startknoten neu startet. Auf diese Weise wird der Graph vollständig exploriert und die Ausgabe besteht aus einem „Spannwald“ mit einem Spannbaum für jede Komponente des Graphen. Wenden wir dabei DFS (BFS) an, so sprechen wir von einem DFS-Spannwald (BFS-Spannwald).

In dem Eingabegraphen aus Abb. 3 wären zwei Starts des generischen Verfahrens nötig: zum Beispiel einer mit Startknoten 1 und einer mit Startknoten 9. Der resultierende DFS-Spannwald (bzw. BFS-Spannwald) ist in Abb. 6 (bzw. in Abb. 7) zu sehen.

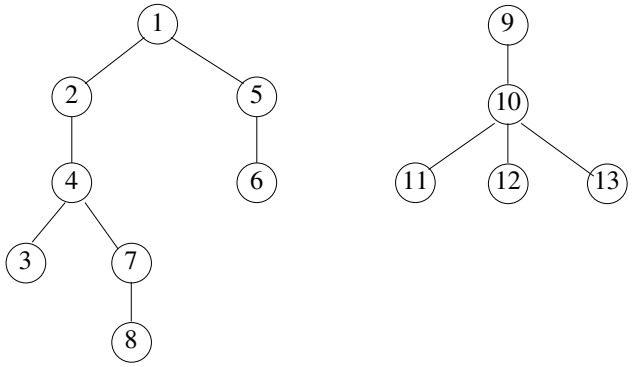


Abbildung 6: Der DFS-Spannwald zum Graphen aus Abb. 3

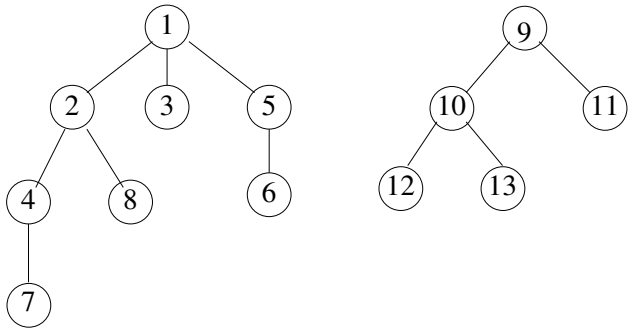


Abbildung 7: Der BFS-Spannwald zum Graphen aus Abb. 3

2.3 Erweiterung 2: Verteilung von DFS-Nummern

Wir sagen ein Knoten v erhält die *DFS-Eintrittsnummer* i, notiert als $N_{ein}(v) = i$, wenn er der i-te Knoten ist, der in den STACK aufgenommen wird. Analog sagen wir, v erhält die *DFS-Austrittsnummer* i, notiert als $N_{aus}(v) = i$, wenn er der i-te Knoten ist, der aus dem STACK entfernt wird. Anwendungen der DFS-Nummern werden wir sowohl in den Übungen als auch im Abschnitt über Exploration von Digraphen kennenlernen. An dieser Stelle wollen wir nur darauf hinweisen, dass die Berechnung der DFS-Nummern mit Hilfe zweier Zähler Z_{ein}, Z_{aus} leicht in das generische Verfahren integrierbar ist (und zwar an den Stellen, die wir vorsorglich mit „`evtl. weitere Instruktionen`“ gekennzeichnet hatten):

3. $N_{ein}(s) := 1$; $Z_{ein} := 1$; $Z_{aus} := 0$

4.(b), Fall 1, iii. $Z_{ein} := Z_{ein} + 1; N_{ein}(w) := Z_{ein}$

4.(b), Fall 2, ii. $Z_{aus} := Z_{aus} + 1; N_{aus}(v) := Z_{aus}$

Beide Nummern lassen sich alternativ auch absteigend vergeben. In diesem Fall würden die Zähler mit $n + 1$ (statt mit 0) initialisiert und vor jeder neu vergebenen Nummer um 1 runtergezählt (statt um 1 raufgezählt). Wenn wir nicht explizit auf absteigende Nummerierung hinweisen, wollen wir aber im Normalfall von einer aufsteigenden Nummerierung ausgehen.

Im Beipielgraphen aus Abb. 3 und einer DFS mit Startknoten 1 würden die Knoten den STACK in der Reihenfolge

$$1, 2, 4, 3, 7, 8, 5, 6$$

betreten und ihn in der Reihenfolge

$$3, 8, 7, 4, 2, 6, 5, 1,$$

wieder verlassen. Damit ergeben sich folgende DFS-Nummern:

v	1	2	3	4	5	6	7	8
$N_{ein}(v)$	1	2	4	3	7	8	5	6
$N_{aus}(v)$	8	5	1	4	7	6	3	2

Man überlegt sich leicht den folgenden

Satz:
1. Die Reihenfolge, in der die Knoten den STACK betreten, entspricht einem „preorder traversal“ des betreffenden DFS-Spannbaumes.
2. Die Reihenfolge, in der die Knoten den STACK verlassen, entspricht einem „postorder traversal“ des betreffenden DFS-Spannbaumes.

2.4 Erweiterung 3: BFS und die Distanz zum Startknoten

Die *Distanz* von Knoten u zum Knoten v in einem Graphen G ist die Länge eines kürzesten Pfades von u nach v (gemessen in der Anzahl der Kanten). Es ist leicht, eine BFS mit Startknoten s so zu erweitern, dass sie die Distanz $d(v)$ des Startknotens s zum Knoten v für jedes $v \in V$ berechnet:

3. $d(s) := 0$; $d(v) := \infty$ für alle $v \in V \setminus \{s\}$

4.(b), Fall 1, iii. $d(w) := d(v) + 1$

In Abb. 5(b) sind die Distanzwerte rechts am Spannbaum mit Wurzel 1 vermerkt. Die Distanz deckt sich jeweils mit der Tiefe des betreffenden Knotens im BFS-Spannbaum. Dies ist kein Zufall, denn es gilt der allgemeine

Satz: Der BFS-Spannbaum ist ein „Kürzeste-Pfade Baum“, d.h. der eindeutige Pfad im BFS-Spannbaum von der Wurzel s zu einem Knoten v ist auch in G ein kürzester Pfad von s nach v.

3 Exploration von Digraphen

DFS und BFS (und auch die im vorigen Abschnitt diskutierten Erweiterungen) können jeweils ohne Probleme auf Digraphen G (anstelle von ungerichteten Graphen) angewendet werden. Wenn wir von einem festen Startknoten s ausgehen, dann ist das Ergebnis ein gerichteter Baum mit Wurzel s, der alle Knoten enthält, welche in G von s aus erreichbar sind. Falls von s aus in G alle Knoten erreichbar sind, ergibt sich sogar ein Spannbaum von G. Falls das nicht der Fall ist, dann kann mit Hilfe von Neustarts (analog zur Vorgehensweise in Abschnitt 2.2) eine vollständige Graphexploration durchgeführt werden, deren Ergebnis ein gerichteter Spannwald von G ist.

Zur Illustration betrachten wir den Digraphen aus Abb. 8(a) mit Startknoten 1 (von dem aus alle Knoten des Digraphen erreichbar sind). Es sei an die Vereinbarung erinnert, dass Nachbarn eines Knotens in der Reihenfolge aufsteigender Knotennummern inspiziert werden. Der resultierende DFS-Spannbaum (BFS-Spannbaum) ist in Abb. 8(b) (bzw. in Abb. 8(c)) zu sehen. Wie schon bei ungerichteten Graphen, ist der BFS-Spannbaum wieder ein „Kürzeste-Pfade Baum“.

Die nicht zum jeweiligen Spannbaum T gehörenden Kanten sind in Abb. 8 gestrichelt dargestellt. Sie zerfallen auf natürliche Weise in folgende Typen:

F-Kanten Die „Vorwärtskanten“ (kurz: F-Kanten[6]) führen von einem Knoten v zu einem Nachfolger in T.

B-Kanten Die „Rückwärtskanten“ (kurz: B-Kanten[7]) führen von einem Knoten v zu einem Vorgänger in T.

[6] Englisch: Forward-edges
[7] Englisch: Backward-edges

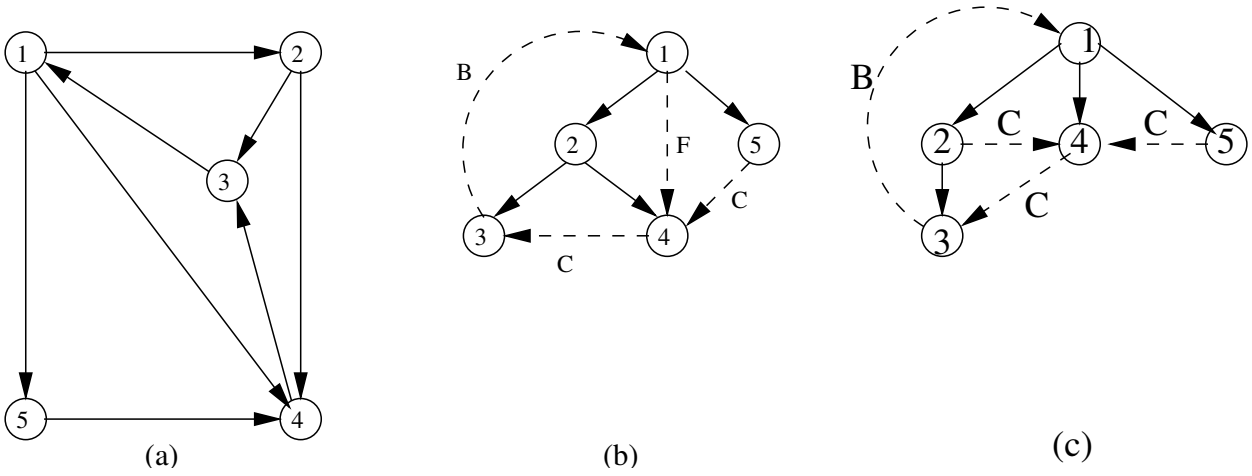


Abbildung 8: (a) ein Digraph (b) sein DFS-Spannbaum (c) sein BFS-Spannbaum (nicht zum Spannbaum gehörende Graphkanten gestrichelt)

C-Kanten Die „kreuzenden Kanten“ (kurz: C-Kanten[8]) verbinden zwei in T „unabhängige“ Knoten (d.h. zwei Knoten von denen keiner Nachfolger des anderen ist).

Man überlegt sich leicht:

Beobachtung 1: C-Kanten in einem DFS-Baum (oder DFS-Wald) führen stets von rechts nach links. (Warum?)

Beobachtung 2: In einem BFS-Baum (oder BFS-Wald) gibt es keine F-Kanten und C-Kanten verlaufen stets von einem Knoten einer Tiefe i zu einem Knoten einer Tiefe von maximal $i + 1$. (Warum?)

Aus Beobachtung 1 ergibt sich die

Folgerung: Ein Digraph ist genau dann azyklisch (also ein DAG = Directed Acyclic Graph), wenn sein DFS-Spannwald keine B-Kanten enthält.

Der Typ (F, B oder C) der nicht zum DFS-Wald gehörenden Kanten des Digraphen ist leicht an den Eintritts- und Austrittsnummern der Knoten ablesbar.[9] Somit ist auch ein „DAG-Test“ für einen Digraphen mit (geeignet erweiterter) DFS leicht durchzuführen.

Als letzte Anwendung von DFS betrachten wir die topologische Sortierung der Knoten eines DAGs $G = (V, E)$. Die Knoten heißen dabei *topologisch sortiert*, wenn jeder Knoten erst dann durchlaufen wird, nachdem alle seine Vorgänger bereits durchlaufen wurden. Wir wollen der Einfachheit halber annehmen, dass der DAG G einen Knoten s enthält, von dem aus alle Knoten erreichbar sind und den die DFS als Startknoten einsetzt.[10] Mit Hilfe von Beobachtung 1 (s. oben) ergibt sich leicht folgender

[8] Englisch: Crossing-edges

[9] Dies zu zeigen wäre eine gute Übungsaufgabe!

[10] Ein solcher Knoten kann notfalls hinzugefügt werden.

Satz: Die absteigenden DFS-Austrittsnummern (also ein „gespiegeltes postorder traversal“) liefern eine topologische Sortierung der Knoten von G.

Zur Illustration betrachte Abb. 9. Wie am DFS-Spannbaum erkennbar ist, gibt es keine B-Kanten. Somit handelt es sich wirklich um einen DAG. Weiterhin bringt ein „postorder traversal“ des DFS-Spannbaums die Knoten in die Reihenfolge $3, 4, 2, 5, 1$. Spiegelung liefert $1, 5, 2, 4, 3$: in der Tat eine topologische Sortierung der Knoten !

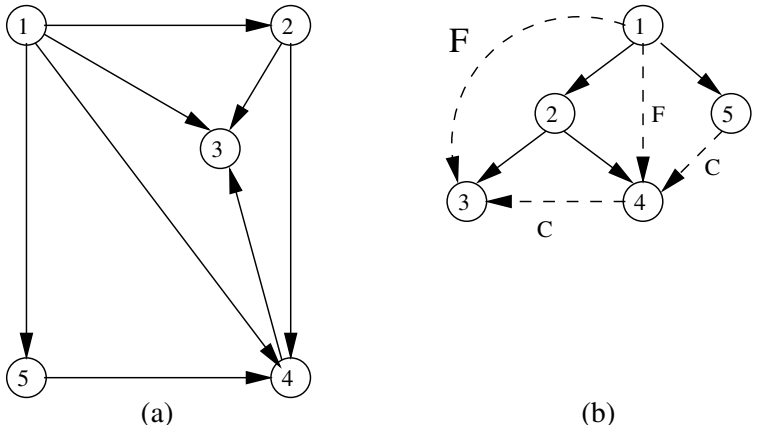


Abbildung 9: (a) ein DAG (b) sein DFS-Spannbaum (nicht zum Spannbaum gehörende Graphkanten gestrichelt)

4 Rechenzeit für DFS und BFS

Bei vernünftiger Implementierung von DFS bzw. BFS benötigen diese Verfahren auf einem (Di-)Graphen mit n Knoten und m Kanten nur $O(n + m)$ Rechenschritte (Linearzeit). Dies gilt auch für die vollständige Graphexploration (mit Neustarts bis alle Knoten exploriert sind) und für alle in diesem Manuskript geschilderten Erweiterungen. Insbesondere sind folgende Probleme in Linearzeit lösbar:

- Berechnung der Zusammenhangskomponenten eines Graphen
- Berechnung des DFS- oder BFS-Spannwaldes
- Test eines Digraphen auf Existenz von Zykeln (DAG oder nicht DAG)
- topologische Sortierung der Knoten eines DAG

Weitere Details zur Implementierung von DFS und BFS sowie weitere Anwendungen werden in der Vorlesung „Datenstrukturen“ vermittelt.