Prof. Dr. Leif Kobbelt
Stefan Dollase, Ira Fesefeldt, Alexandra Heuschling, Gregor Kobsik

Lösung - Übung 7

Aufgabe 5 (Optimaler Suchbaum): 7 + 2 + 1 = 10 Punkte

Gegeben sind folgende Knoten mit dazugehörigen Zugriffswahrscheinlichkeiten:

Knoten	I_0	N_1	I_1	N_2	I_2	N_3	I_3	N_4	I_4	N_5	I_5
Werte	$(-\infty,1)$	1	(1,2)	2	(2,3)	3	(3,4)	4	(4,5)	5	$(5,\infty)$
Wahrscheinlichkeit	0	0.1	0	0.2	0	0.1	0	0.3	0	0.3	0

Konstruieren Sie einen optimalen Suchbaum wie folgt.

a) Füllen Sie untenstehende Tabellen für $W_{i,j}$ und $C_{i,j}$ nach dem Verfahren aus der Vorlesung aus. Geben Sie in $C_{i,j}$ ebenfalls **alle möglichen Wurzeln** des optimalen Suchbaums für $\{i,\dots,j\}$ an.

$W_{i,j}$	0	1	2	3	4	5
1						
2	–					
3	–	–				
4	–	–	–			
5	–	–	–	–		
6	–	–	–	–	–	

$C_{i,j}$ ($R_{i,j}$)	0	1	2	3	4	5
1		( )	( )	( )	( )	( )
2	–		( )	( )	( )	( )
3	–	–		( )	( )	( )
4	–	–	–		( )	( )
5	–	–	–	–		( )
6	–	–	–	–	–	

b) Geben Sie einen optimalen Suchbaum für die Knoten mit den gegebenen Zugriffswahrscheinlichkeiten und der gegebenen Reihenfolge der Knoten graphisch an.

c) Ist der optimale Suchbaum für die Knoten mit den gegebenen Zugriffswahrscheinlichkeiten und der gegebenen Reihenfolge der Knoten eindeutig? Geben Sie dazu eine kurze Begründung an.

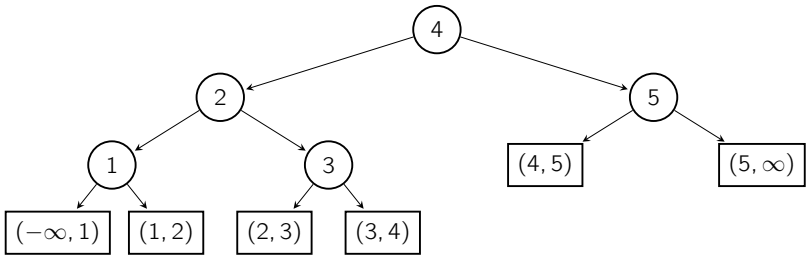
Lösung

a)

$W_{i,j}$	0	1	2	3	4	5
1	0	0.1	0.3	0.4	0.7	1.0
2	–	0	0.2	0.3	0.6	0.9
3	–	–	0	0.1	0.4	0.7
4	–	–	–	0	0.3	0.6
5	–	–	–	–	0	0.3
6	–	–	–	–	–	0

$C_{i,j}$ ($R_{i,j}$)	0	1	2	3	4	5
1	0	0.1 (1)	0.4 (2)	0.6 (2)	1.3 (2,4)	1.9 (4)
2	–	0	0.2 (2)	0.4 (2)	1.0 (4)	1.6 (4)
3	–	–	0	0.1 (3)	0.5 (4)	1.1 (4)
4	–	–	–	0	0.3 (4)	0.9 (4,5)
5	–	–	–	–	0	0.3 (5)
6	–	–	–	–	–	0

b) Nur die folgende Lösung ist ein korrekter optimaler Suchbaum.



c) Der Optimale Suchbaum ist eindeutig, denn bei der Konstruktion von $C_{i,j}$ war jedes relevante Minimum eindeutig.

Aufgabe 6 (Bucketsort): 10 Punkte

Sortieren Sie das folgende Array mithilfe von Bucketsort. Geben Sie dazu an, welche Buckets Sie verwenden, für welche Intervalle diese stehen und welche Elemente in welcher Reihenfolge in diese Buckets eingefügt werden. Geben Sie zusätzlich den Inhalt der Bucket an, nachdem diese sortiert worden sind. Zuletzt geben Sie das vollständig sortierte Array an.

Sie dürfen ein beliebiges Sortierverfahren nutzen um die einzelnen Buckets zu sortieren. Es ist nicht notwendig dazu Zwischenschritte anzugeben.

0.12	0.26	0.69	0.86	0.93	0.52	0.34	0.25	0.43	0.31

Lösung

Unsortierte Eingabe (n=10)

0.12	0.26	0.69	0.86	0.93	0.52	0.34	0.25	0.43	0.31

Eingeordnet in Buckets

0	$[0.0, 0.1)$			
1	$[0.1, 0.2)$	$\rightarrow$	0.12	
2	$[0.2, 0.3)$	$\rightarrow$	0.26	$\rightarrow$ 0.25
3	$[0.3, 0.4)$	$\rightarrow$	0.34	$\rightarrow$ 0.31
4	$[0.4, 0.5)$	$\rightarrow$	0.43	
5	$[0.5, 0.6)$	$\rightarrow$	0.52	
6	$[0.6, 0.7)$	$\rightarrow$	0.69	
7	$[0.7, 0.8)$			
8	$[0.8, 0.9)$	$\rightarrow$	0.86	
9	$[0.9, 1.0)$	$\rightarrow$	0.93	

Buckets wurden sortiert

0	$[0.0, 0.1)$			
1	$[0.1, 0.2)$	$\rightarrow$	0.12	
2	$[0.2, 0.3)$	$\rightarrow$	0.25	$\rightarrow$ 0.26
3	$[0.3, 0.4)$	$\rightarrow$	0.31	$\rightarrow$ 0.34
4	$[0.4, 0.5)$	$\rightarrow$	0.43	
5	$[0.5, 0.6)$	$\rightarrow$	0.52	
6	$[0.6, 0.7)$	$\rightarrow$	0.69	
7	$[0.7, 0.8)$			
8	$[0.8, 0.9)$	$\rightarrow$	0.86	
9	$[0.9, 1.0)$	$\rightarrow$	0.93	

Buckets wurden hintereinander gehängt

0.12	0.25	0.26	0.31	0.34	0.43	0.52	0.69	0.86	0.93

Aufgabe 7 (Mastertheorem - Wiederholung): 2 + 2 + 2 + 2 + 2 = 10 Punkte

Bestimmen Sie mithilfe des Mastertheorems die beste mögliche O-Klasse für folgende Rekursionsgleichungen. Geben Sie zusätzlich an, welches p Sie gewählt haben. Begründen Sie, warum weder durch eine andere Wahl von p, noch durch die Anwendung eines anderen Falls des Mastertheorems eine bessere O-Klasse erreicht werden kann.

a)

$$T(1) = 1$$
$$T(n) = 2 \cdot T(\frac{n}{4}) + n + 4 \qquad \text{für } n > 1$$

b)

$$T(1) = 1$$
$$T(n) = 9 \cdot T(\frac{n}{3}) + n \cdot \sqrt{n} \qquad \text{für } n > 1$$

c)

$$T(1) = 1$$
$$T(n) = 16 \cdot T(\frac{n}{2}) + 12 \cdot n^4 + 16 \cdot n^3 + 100 \qquad \text{für } n > 1$$

d)

$T(1) = 1$

$T(n) = 6 \cdot T(\frac{n}{6}) + \frac{n}{2}$ für $n > 1$

e)

$T(1) = 1$

$T(n) = 8 \cdot T(\frac{n}{4}) + \frac{n^2}{6} + n \cdot \sqrt{n}$ für $n > 1$

Lösung

a) Wir wählen $p = 1$, dann ist $n + 4 \in O(n)$ und $2 < 4$, damit ist nach dem Mastertheorem auch $T(n) \in O(n)$. Ein kleineres p kann nicht gewählt werden ohne das $n + 4 \in O(n) = O(n^p)$ gilt, damit ist dies der einzige zutreffende Fall.

b) Wir wählen $p = 1.5$, dann ist $n \cdot \sqrt{n} = n^{1.5} = O(n^{1.5})$ und $9 > 6 > 3^{1.5}$ und $\log_3(9) = 2$, somit ist nach dem Mastertheorem $T(n) \in O(n^2)$. Für den zweiten oder ersten Fall hätten wir $p \geq 2$ wählen müssen, womit die O-Klasse jedoch schlechter wäre.

c) Wir wählen $p = 4$, dann ist $12 \cdot n^4 + 16 \cdot n^3 + 100 = O(n^4)$ und $16 = 2^4$, somit ist nach dem Mastertheorem $T(n) \in O(n^4 \cdot \log(n))$. Die Wahl eines größeren p würde die O-Klasse verschlechtern. Mit einem kleineren p würde die Bedingung $12 \cdot n^4 + 16 \cdot n^3 + 100 = O(n^p)$ nicht mehr gelten.

d) Wir wählen $p = 1$, dann ist $\frac{n}{2} = O(n)$ und $6 = 6^1$, somit ist nach dem Mastertheorem $T(n) \in O(n \cdot \log(n))$. Die Wahl eines größeren p würde die O-Klasse verschlechtern. Mit einem kleineren p würde die Bedingung $\frac{n}{2} = O(n^p)$ nicht mehr gelten.

e) Wir wählen $p = 2$, dann ist $\frac{n^2}{6} + n \cdot \sqrt{n} < 2 \cdot n^2 = O(n^2)$ und $8 < 4^2$, somit ist nach dem Mastertheorem $T(n) \in O(n^2)$. Mit einem kleineren p würde die Bedingung $\frac{n^2}{6} + n \cdot \sqrt{n} < 2 \cdot n^2 = O(n^2)$ nicht mehr gelten.

Aufgabe 8 (Abstrakte Datentypen): 5 + 5 = 10 Punkte

Ein abstrakter Datentyp wird definiert durch die Spezifikationen der Methoden mit denen man auf diesen Datentyp zugreifen kann. Daher kann ein abstrakter Datentyp mehr als eine Implementierung haben, die unterschiedliche Vor- oder Nachteile, insbesondere unterschiedliche Laufzeitkomplexitäten haben können. Wir haben in dieser Aufgabe abstrakte Datentypen (mit möglicherweise etwas anderen Methodennamen) mit einer unüblichen und ineffizienten Implementierung angegeben. Welche abstrakten Datentypen haben wir hier implementiert?

a) Die Methoden `insert`, `delete` und `max` sind die Methoden für binäre Suchbäume wie in der Vorlesung und den Übungen vorgestellt.

```
class ADT_three:
    def __init__(self):
        self.root = None

def push(x,p,adt_three):
    adt_three.root = insert(adt_three.root,p,x)

def pop(adt_three):
    if adt_three.root is None:
        return None
    m = max(adt_three.root)
    adt_three.root = delete(m.key,adt_three.root)
    return m.value
```

Geben Sie an, welcher abstrakte Datentyp hier implementiert wurde. Begründen Sie auch ihre Antwort.

b)

```
class ADT_four:
    def __init__(self):
        self.dict_1 = {}
        self.dict_2 = {}

def all(adt_four):
    all_list = []
    for key in adt_four.dict_2.keys():
        if adt_four.dict_2[key]:
            all_list.append(key)
    return all_list

def all_n(v,adt_four):
    i = 0
    all_n_list = []
    while str(v)+str(i) in adt_four.dict_1:
        all_n_list.append(adt_four.dict_1[str(v)+str(i)])
        i += 1
    return all_n_list

def add(v,adt_four):
    adt_four.dict_2[v] = True

def connect(v1,v2,adt_four):
    i=0
    while str(v1)+str(i) in adt_four.dict_1 and adt_four.dict_1[str(v1)+str(i)] != v2:
        i += 1
    adt_four.dict_1[str(v1)+str(i)] = v2
```

Geben Sie an, welcher abstrakte Datentyp hier implementiert wurde. Begründen Sie auch ihre Antwort.

Lösung

a) Der abstrakte Datentyp dieser Implementierung ist eine Prioritätswarteschlange. Mittels `push` wird ein Objekt mit Priorität, bzw Schlüssel `x` in den binären Suchbaum hinzugefügt. Die Methode `pop` entfernt nun aber das Objekt mit der größten Priorität, bzw Schlüssel und gibt diesen zurück.

b) Der abstrakte Datentyp dieser Implementierung ist ein Graph in Form einer Adjazenzliste. Die Methode `all` gibt alle Knoten des Graphen zurück – diese werden in den Dictionary `dict_2` in Form eines Bitvektors gespeichert. Durch die Methode `add` können wir nun Knoten in diesem Dictionary hinzufügen. Die Methode `all_n` gibt die Adjanzentliste des Knoten `v` zurück. Die Adjanzentlisten speichern wir durchnummeriert in dem Dictionary `dict_1`. Durchnummeriert in dem Sinne, dass die benachbarten Knoten unter dem Schlüssel `v` konkateniert mit einer Zahl speichern. `connect` fügt nun eine Kante zwischen zwei Knoten hinzu, falls diese noch nicht existiert.

Dabei wird durch `connect` nicht überprüft, ob der Knoten bereits hinzugefügt worden ist. Das ist in der aktuellen Implementation nicht so drastisch, da viele Spezifikationen undefiniertes Verhalten in Fällen wie dieses erlaubt. Jedoch viel schlimmer ist, dass `connect` Kanten überschreibt, sollte die Knoten Zahlen als Namen haben: Gäbe es Knoten 1 und Knoten 11, wobei Knoten 11 eine Kante mit Index 1 und Knoten 1 eine Kante mit Index 11 haben, so wären beide Kante mit dem String 111 kodiert.