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

Lösung - Übung 9

Tutoraufgabe 1 (Optimaler Suchbaum):

Gegeben sind folgende Knoten mit dazugehörigen Zugriffswahrscheinlichkeiten:

Knoten	l_0	N_1	l_1	N_2	l_2	N_3	l_3	N_4	l_4
Wert	$(-\infty,1)$	1	(1,2)	2	(2,3)	3	(3,4)	4	$(4,\infty)$
Wahrscheinlichkeiten	0.1	0.1	0.1	0.2	0.2	0.1	0.1	0.1	0.0

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
1					
2	–				
3	–	–			
4	–	–	–		
5	–	–	–	–	

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

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.

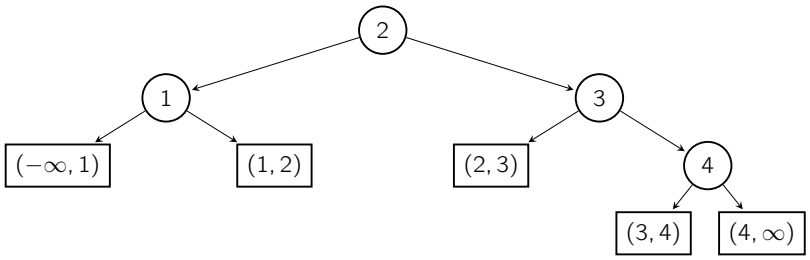
Lösung

a)

$W_{i,j}$	0	1	2	3	4
1	0.10	0.30	0.70	0.90	1.00
2	–	0.10	0.50	0.70	0.80
3	–	–	0.20	0.40	0.50
4	–	–	–	0.10	0.20
5	–	–	–	–	0.00

$C_{i,j}(R_{i,j})$	0	1	2	3	4
1	0.10	0.50 (1)	1.40 (2)	2.10 (2)	2.50 (2)
2	–	0.10	0.80 (2)	1.50 (2)	1.90 (2, 3)
3	–	–	0.20	0.70 (3)	1.00 (3)
4	–	–	–	0.10	0.30 (4)
5	–	–	–	–	0.00

b) Die folgenden Lösungen sind korrekte optimale Suchbäume.



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

Tutoraufgabe 2 (Union Find):

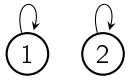
Führen Sie die folgenden Operationen beginnend mit einer anfangs leeren *Union-Find-Struktur* aus und geben Sie die entstehende Union-Find-Struktur nach jeder *MakeSet*, *Union* und *Find* Operation an. Nutzen Sie dabei die beiden Laufzeitverbesserungen: Höhenbalencierung und Pfadkompression. Dabei soll die Union-Operation bei **gleicher Höhe der Wurzeln immer die Wurzel des zweiten Parameters** als neue Wurzel wählen. Es ist nicht notwendig die Höhe der Bäume zu notieren.

1. MakeSet(1)
2. MakeSet(2)
3. MakeSet(3)
4. MakeSet(4)
5. MakeSet(5)
6. MakeSet(6)
7. MakeSet(7)
8. MakeSet(8)
9. MakeSet(9)
10. Union(1,2)
11. Union(3,4)
12. Union(3,1)
13. Union(5,6)
14. Union(7,8)
15. Union(7,9)
16. Union(9,5)
17. Union(9,2)
18. MakeSet(10)
19. Union(7,10)
20. Find(3)

Lösung

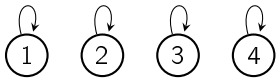
1. MakeSet(1)

2. MakeSet(2)



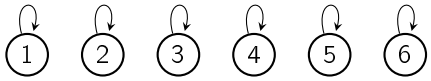
3. MakeSet(3)

4. MakeSet(4)

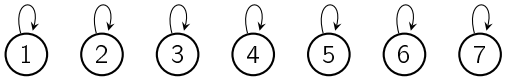


5. MakeSet(5)

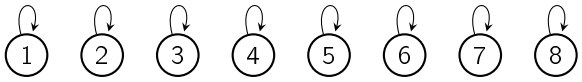
6. MakeSet(6)



7. MakeSet(7)

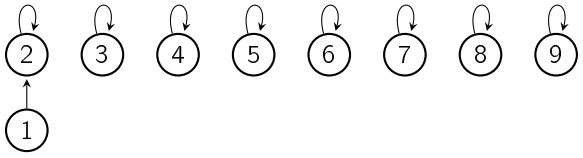


8. MakeSet(8)

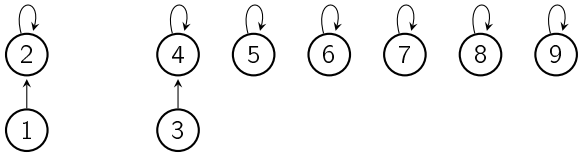


9. MakeSet(9)

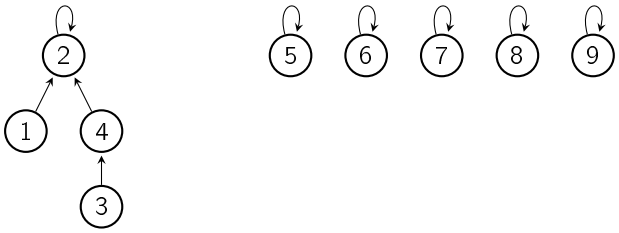
10. Union(1,2)
    - Find(1): unverändert
    - Find(2): unverändert
    - Union(1,2):



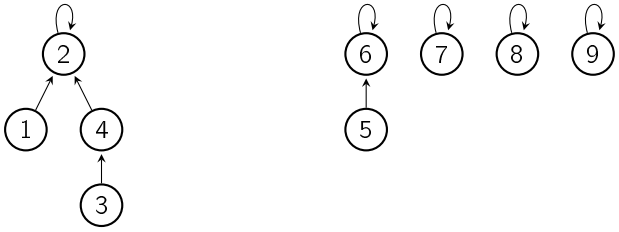
11. Union(3,4)
    - Find(3): unverändert
    - Find(4): unverändert
    - Union(3,4):



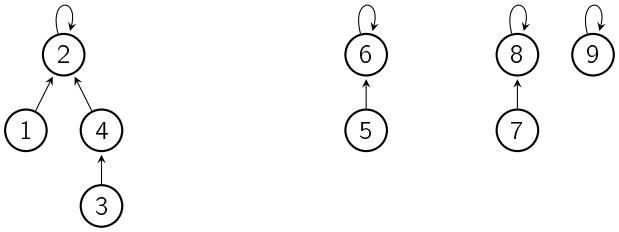
12. Union(3,1)
    - Find(3): unverändert
    - Find(1): unverändert
    - Union(3,1):



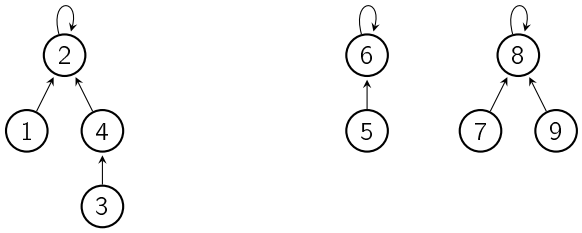
13. Union(5,6)
    - Find(5): unverändert
    - Find(6): unverändert
    - Union(5,6):



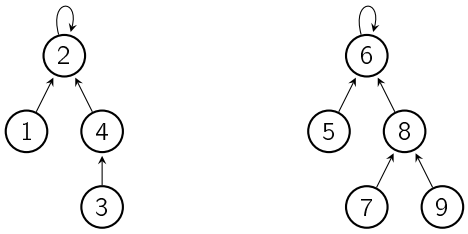
14. Union(7,8)
    - Find(7): unverändert
    - Find(8): unverändert
    - Union(7,8):



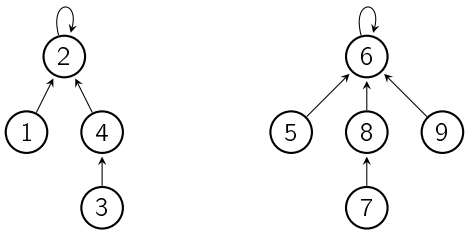
15. Union(7,9)
    - Find(7): unverändert
    - Find(9): unverändert
    - Union(7,9):



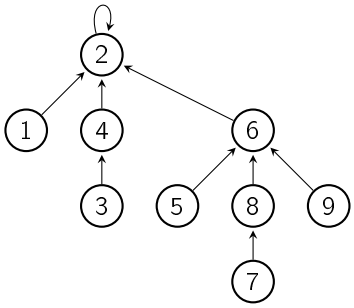
16. Union(9,5)
    - Find(9): unverändert
    - Find(5): unverändert
    - Union(9,5):



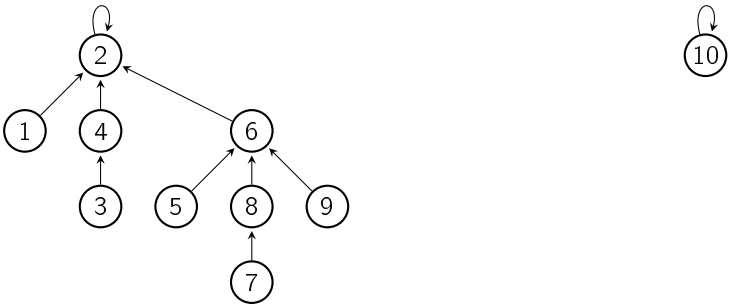
17. Union(9,2)
    - Find(9):



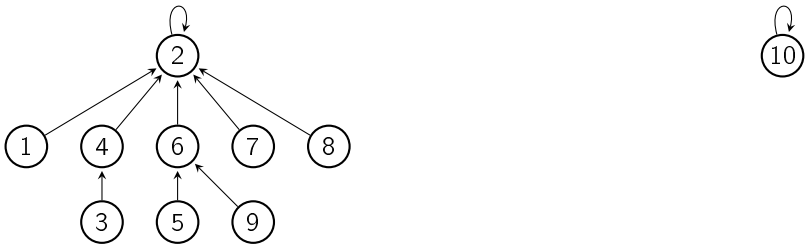
    - Find(2): unverändert
    - Union(9,2):



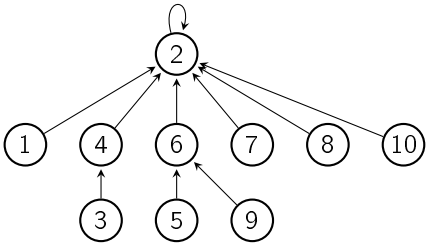
18. MakeSet(10)



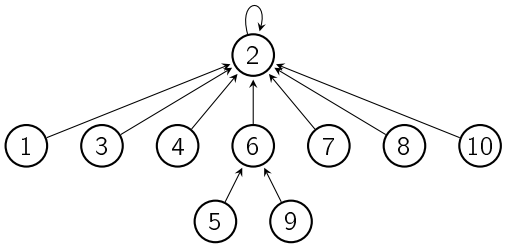
19. Union(7,10)
    - Find(7):



    - Find(10): unverändert
    - Union(7,10):



20. Find(3)



Tutoraufgabe 3 (Prominenz suchen):

Sei ein gerichteter Graph $G = (V, E)$ als Adjazenzmatrix gegeben. Wir nennen einen Knoten $v \in V$ prominent, wenn von allen Knoten $v' \in V \setminus \{v\}$ eine Kante $(v', v) \in E$ nach v existiert, aber es von keinem Knoten $v' \in V$ eine Kante $(v, v') \notin E$ zurück gibt.
Geben Sie einen Algorithmus an, der in $O(|V|)$ Worst-case Laufzeit herausfindet, ob G einen prominenten Knoten besitzt. Begründen Sie die Korrektheit und die Laufzeit Ihres Algorithmus.

Lösung

Da G als Adjazenzmatrix gegeben ist, können wir die Senke durch geschicktes Erkunden der Adjazenzmatrix finden. Angenommen die Adjazenzmatrix ist eine $n \times n$ Matrix und von 1 bis n indiziert.
Wir gehen wie folgt vor: Wir starten bei der Position $(1, n)$ in der Adjazenzmatrix. Ist der Wert an der aktuellen Position (i, j) eine 1 (also ist $(i, j) \in E$), dann erhöhen wir die Zeile um eins. Ist dagegen der Wert an der aktuellen Position (i, j) eine 0 (also ist $(i, j) \notin E$), dann verringern wir die Spalte um eins. Sind wir irgendwann bei einer Position $(k, 1)$ mit Wert 0 angekommen, dann ist entweder Knoten k eine Senke, oder es existiert kein prominenter Knoten. Welcher dieser beiden Fälle zutrifft überprüfen wir anschließend, indem wir die Zeile k auf nicht-null Einträge und die Spalte k auf nicht-eins Einträge prüfen.
In Code sähe dies wie folgt aus:

```
def find_sink(E,n):
    (i,j) = (1,n)
    #bilinear search for sink
    while not (E[i][j] == 0 and j == 1):
        if E[i][j] == 1:
            i += 1
        else:
            j -= 1
    k = i

    #check if k has no out edges
    for i in range(1,n):
        if E[k][i] != 0:
            return False

    #check if k has all in edges
    for i in range(1,n):
        if E[i][k] != 1 and i != k:
            return False

    return k
```

Die Idee hier ist, dass ein prominenter Knoten eine 0 Zeile und eine 1 Spalte besitzt, das heißt eine Matrix dieser Form:

$$\begin{pmatrix} \ddots & \ddots & 1 & \ddots & \ddots \\ \ddots & \ddots & \vdots & \ddots & \ddots \\ \ddots & \ddots & 1 & \ddots & \ddots \\ 0 & \cdots & 0 & \cdots & 0 \\ \ddots & \ddots & 1 & \ddots & \ddots \\ \ddots & \ddots & \vdots & \ddots & \ddots \\ \ddots & \ddots & 1 & \ddots & \ddots \end{pmatrix}$$

Dieses Kreuz lässt sich finden, in dem wir die horizontale 0 Reihe suchen. Daher gehen wir stets nach links wenn wir auf eine null treffen. Damit werden wir auch irgendwann auf die 0 Reihe treffen. Treffen wir dagegen auf eine 1 gehen wir nach unten, da die derzeitige Spalte nicht die 0 Reihe sein kann.
Zwecks Korrektheit müssen wir zuerst einsehen, dass es stets maximal einen prominenten Knoten geben kann:

Gäbe es zwei, seien diese beide i und j müsste es auch eine Kante $(i, j) \in E$ geben, damit j prominent ist. Dann kann aber i nicht mehr prominent sein, da es eine ausgehende Kante gibt.
Nun nehmen wir zuerst an, dass es keine prominente Knoten gibt. Dann wird der Algorithmus bei welchem Knoten auch immer er glaubt einen prominenten Knoten gefunden zu haben, feststellen, dass es nicht dem 0-1-Kreuz entspricht und daher auch kein prominenter Knoten ist.
Nehmen wir nun an, dass es genau einen prominenten Knoten k gibt. Dann muss der Algorithmus irgendwann entweder auf einen Eintrag (i, k) mit $i < k$ oder einen Eintrag (k, j) mit $j > k$ treffen. Dies gilt, da wir in jeder Iteration nur einen Eintrag um eines ändern. Landen wir auf einem Eintrag (i, k) erhöhen wir das i solange bis wir bei (k, k) angekommen sind. Von da werden wir solange den zweiten Eintrag senken bis wir bei $(k, 1)$ angekommen sind. Landen wir bei auf einem Eintrag (k, j) senken wir das j solange bis wir bei $(k, 1)$ angekommen sind. In allen Fällen landen wir bei $(k, 1)$ und geben k als korrekten prominenten Knoten zurück. Für das Suchen des k haben wir maximal $2n$ viele Inkrementierungs-, bzw Dekrementierungsoperationen. Zum überprüfen ob k auch wirklich ein prominenter Knoten ist, müssen wir sowohl einmal die k Zeile, als auch die k Reihe mit jeweils n vielen Vergleichen überprüfen. Insgesamt haben wir damit also $4n$ viele Iterationen, damit also auch eine Laufzeit von $O(n)$.