Name: Matrikelnummer:

Klausur Algorithmen I, 03.08.2009 Blatt 1 von 6

Lösungsvorschlag

Aufgabe 1. Kleinaufgaben [19 Punkte]

Beantworten Sie die folgenden Fragen. Antworten Sie jeweils kurz!

a. Geben Sie ein Beispiel für die Verwendung eines Wächterobjektes an. [2 Punkte]

Lösung

Z. B. bei find(x) in Liste: Speichere x in Wächter.

Lösungsende

b. Geben Sie den Namen eines Sortieralgorithmus an, der eine Laufzeit von $O(n \log n)$ besitzt und nur $O(1)$ zusätzlichen Platz benötigt. [2 Punkte]

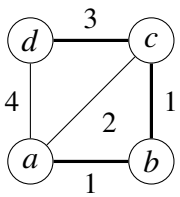
Lösung

Heapsort

Lösungsende

c. Geben Sie einen Graphen an, bei dem die eindeutige LEICHTESTE Kante auf einem Kreis NICHT im minimalen Spannbaum ist. [4 Punkte]

Lösung



Die dicken Kanten stellen den MST dar. Die leichteste Kante des Kreises $\langle a, c, d, a \rangle$ ist nicht im MST.

Lösungsende

d. Wie lautet das Optimalitätsprinzip, welches für die dynamische Programmierung wichtig ist? [2 Punkte]

Lösung

- Optimale Lösungen bestehen aus optimalen Lösungen für Teilprobleme.
- Mehrere optimale Lösungen $\Rightarrow$ es ist egal welche benutzt wird.

Lösungsende

Lösungsvorschlag

Fortsetzung von Aufgabe 1

e. Wie ist eine Familie von universellen Hashfunktionen definiert? [2 Punkte]

Lösung

Für eine Familie von universellen Hashfunktionen $\mathscr{H} \subseteq \{0..m-1\}^{\mathsf{Key}}$ gilt für alle $x, y \in \mathsf{Key}$ mit $x \neq y$ und zufälligem $h \in \mathscr{H}$,

$$\mathbb{P}[h(x) = h(y)] = \frac{1}{m} \ .$$

Lösungsende

f. Wie viele Kanten kann ein Graph mit n Knoten ohne Parallelkanten und Schleifen höchstens besitzen? Geben Sie die Anzahl für einen gerichteten und einen ungerichteten Graphen an. [2 Punkte]

Lösung

- gerichtet: $m_{\text{gerichtet}} = n \cdot (n-1) = n^2 - n$, da sich jeder der n Knoten maximal einmal mit jedem der anderen $n-1$ Knoten verbinden kann.
- ungerichtet: $m_{\text{ungerichtet}} = m_{\text{gerichtet}}/2 = (n^2-n)/2$, da aus einer vorwärts- und einer rückwärtsgerichteten Kante eine ungerichtete wird.

Lösungsende

g. Zeigen Sie: $\log(n!) = \mathrm{O}(n \log n)$. [2 Punkte]

Lösung

Es gilt $n! \leq n^n \Rightarrow \log(n!) \leq \log(n^n) = n \log n \Rightarrow \log(n!) = \mathrm{O}(n \log n)$.

Lösungsende

h. Zeigen Sie: $\log(n!) = \Omega(n \log n)$. [3 Punkte]

Lösung

$n! \geq \lceil \frac{n}{2} \rceil \cdot (\lceil \frac{n}{2} \rceil + 1) \cdot \ldots \cdot n \geq \lceil \frac{n}{2} \rceil^{\lceil n/2 \rceil} \geq \left(\frac{n}{2}\right)^{n/2}$
$\Rightarrow \log(n!) \geq \frac{n}{2} \log(\frac{n}{2}) = \frac{n}{2}(\log(n) - \log(2))$
$\Rightarrow \forall n \geq 2^2 : \log(n!) \geq \left(\log(n) - \frac{1}{2}\log(n)\right) = \frac{n}{4} \log n \Rightarrow \log(n!) = \Omega(n \log n)$

Lösungsende

(weitere Teilaufgaben zu dieser Aufgabe auf dem nächsten Blatt)

Lösungsvorschlag

Aufgabe 2. Algorithmen-Entwurf: Häufigster Wert in einer Folge [10 Punkte]

Gegeben sei ein Feld $F[1..n]$ von $n \geq 1$ Zahlen aus $\{1,\ldots,n^3\}$. Gesucht ist die Zahl, die in F am häufigsten vorkommt. Wenn es mehrere Zahlen gibt, die gleich häufig sind, kann eine beliebige gewählt werden.
Geben Sie einen möglichst "guten" Algorithmus an, der dieses Problem löst. Die Beschreibung soll textuell oder in kommentiertem Pseudocode erfolgen.
Begründen Sie kurz die Laufzeit Ihres Algorithmus.

Bewertung:

- 6 Punkte für einen Algorithmus mit erwarteter Laufzeit $O(n \log n)$
- 7 Punkte für einen deterministischen Algorithmus mit Laufzeit $O(n \log n)$
- 8 Punkte für einen Algorithmus mit erwarteter Laufzeit $O(n)$
- 10 Punkte für einen deterministischen Algorithmus mit Laufzeit $O(n)$

Lösung

```
radixSort(F)                                              // sortiere Feld
F.pushBack(⊥)                                             // Wächter
(maxKey, maxCount) := (⊥, 0)                              // häufigster bisheriger Schlüssel
(lastKey, lastCount) := (⊥, 0)                            // letzter Schlüssel mit Häufigkeit
for i := 1 to n+1 do          // Zähle jeweils die Anzahl gleicher konsekutiver Schlüssel.
    if F[i] = lastKey then lastCount++
    else
        if lastCount > maxCount then                      // Den häufigsten Schlüssel merken.
            (maxCount, maxKey) := (lastCount, lastKey)
            (lastKey, lastCount) := (F[i], 1)
return maxKey
```

Radix-Sortieren mit K^d Schlüsseln hat deterministische Laufzeit $O(d(n+K))$. Für $K^d = n^3$ ergibt sich also für das Sortieren eine deterministische Laufzeit von $O(n)$. Der Rest des Algorithmus ist auch in $O(n)$, da die for-Schleife genau $n+1$ Mal durchlaufen wird und der Schleifenkörper in $O(1)$ ist.
Eine erwartete Laufzeit von $O(n \log n)$ erhält man beispielsweise mit Quick-Sort statt Radix-Sort.
Mit Merge-Sort oder Heap-Sort ergibt sich deterministisch die Laufzeit $O(n \log n)$.
Eine weitere mögliche Lösung arbeitet mit einer Hash-Tabelle, die für jede Zahl die Häufigkeit speichert und eine erwartete Laufzeit von $O(n)$ erreicht.
Lösungen, die die Anzahlen in einem direkt adressierten Feld speichern, benötigen kubische Zeit für die Initialisierung desselben (z. B. mit Nullen) und sind somit nicht ausreichend. Initialisiert man allerdings nur die wirklich notwendigen Einträge, z. B. in einem Vorab-Lauf über die Eingabe, so erhält man einen deterministischen Algorithmus mit Laufzeit $O(n)$. Der kubische Speicherplatzverbrauch fällt nicht ins Gewicht.

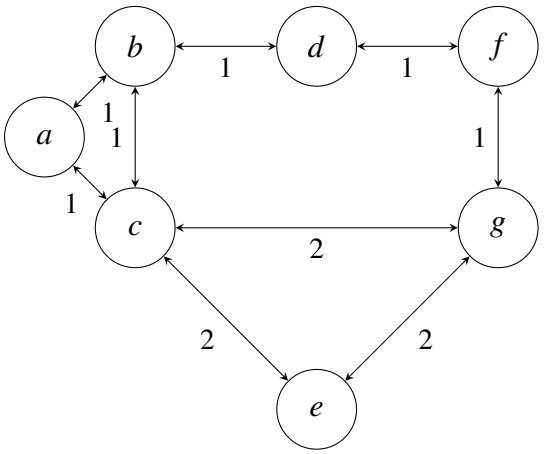
Lösungsende

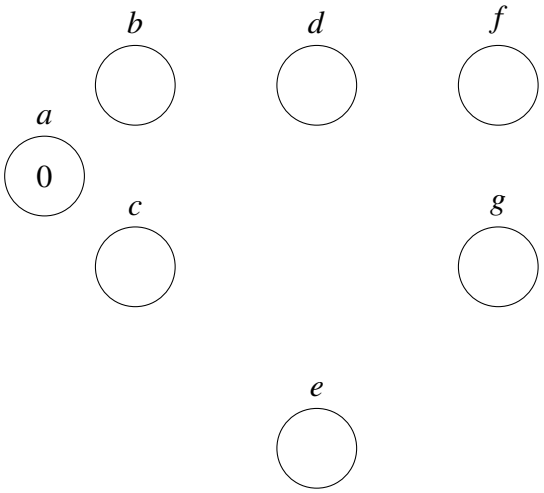
Lösungsvorschlag

Aufgabe 3. Algorithmen-Entwurf: Kürzeste Wege [14 Punkte]

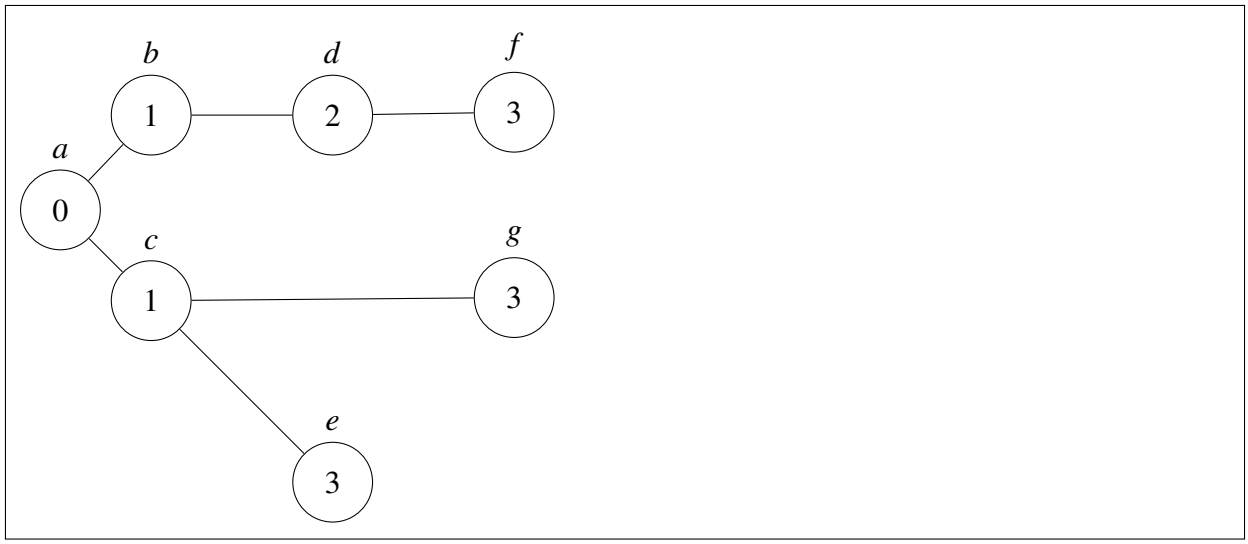
Ein *1-2-Graph* sei ein gerichteter Graph mit Kantengewichten aus $\{1,2\}$.

a. Berechnen Sie die kürzesten Wege und die Distanzen von Startknoten a zu allen Knoten im unten gegebenen 1-2-Graphen. Zeichnen Sie die Distanzen in das Knotengerüst rechts ein und verbinden Sie die Knoten über die kürzesten Wege. [4 Punkte]





Lösung



Lösungsende

(weitere Teilaufgaben zu dieser Aufgabe auf dem nächsten Blatt)

Lösungsvorschlag

Fortsetzung von Aufgabe 3

b. Beschreiben Sie einen Algorithmus zur Berechnung von kürzesten Wegen in 1-2-Graphen. Kürzeste Wege von einem Startpunkt s zu allen von s aus erreichbaren Knoten sollen in Zeit $O(m+n)$ berechnet werden, n steht dabei für die Anzahl Knoten und m für die Anzahl Kanten. Begründen Sie, warum diese Laufzeit erreicht wird. Die Beschreibung des Algorithmus soll textuell oder in kommentiertem Pseudocode erfolgen. [10 Punkte]

Lösung

Es gibt zwei verschiedene Lösungsvarianten:

1. Baue einen neuen Graphen auf, bei dem jede Kante mit Gewicht 2 durch einen Pfad aus zwei Kanten mit Gewicht 1 ersetzt wird. Die neuen Knoten erhalten fortlaufende Nummern ab $n+1$. Die erzeugte Kantenliste wird dann in Linearzeit in eine Adjazenzfeldrepräsentation umgewandelt. Dann wird Breitensuche von s aus durchgeführt. Die Distanzen sind dann bereits korrekt. Als Nachverarbeitung werden parent-Knoten $> n$ durch deren parent-Knoten ersetzt.

   Analyse: Der neue Graph hat $n' \leq n+m$ Knoten und $m' \leq 2m$ Kanten. Erzeugen der Kanten geht in Zeit $O(m') = O(m+n)$. Aufbauen einer Adjazenzfeldrepräsentation geht in Zeit $O(m'+n') = O(m+n)$. Die Breitensuche dauert Zeit $O(m'+n') = O(m+n)$. Die Gesamtzeit ist also $O(m+n)$.

2. (Skizze) Benutze Dijkstra's Algorithmus mit einer beschränkten Prioritätswarteschlange. Die Prioritätswarteschlange verwaltet drei Mengen: die erste Menge enthält alle Elemente mit der aktuellen minimalen Distanz ℓ, die zweite mit $\ell+1$ und die dritte mit $\ell+2$. Mit einer Listenimplementierung sind die Operationen *insert*, *deleteMin* und *decreaseKey* in $O(1)$ zu implementieren. Die Gesamtlaufzeit des Algorithmus ist damit also $O(m+n)$. Variante 2. a): modifizierte Breitensuche mit drei Queues.

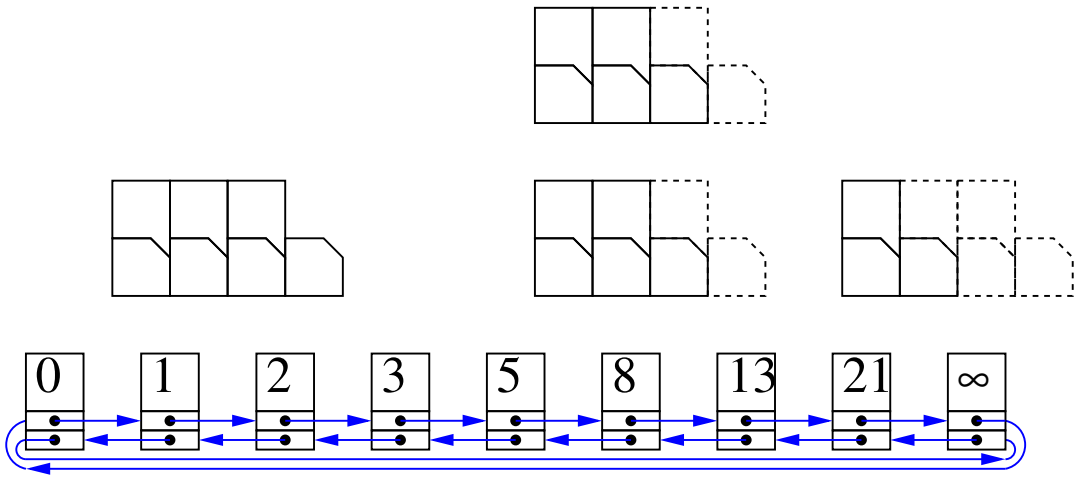
Häufige Fehler: Breitensuche ist nicht korrekt, Dijkstra zu langsam. Bei Lösungen mit drei Queues gab es mehrere subtile Fehlermöglichkeiten (decreaseKey, Abbruchbedingung, ...)

Lösungsende

Lösungsvorschlag

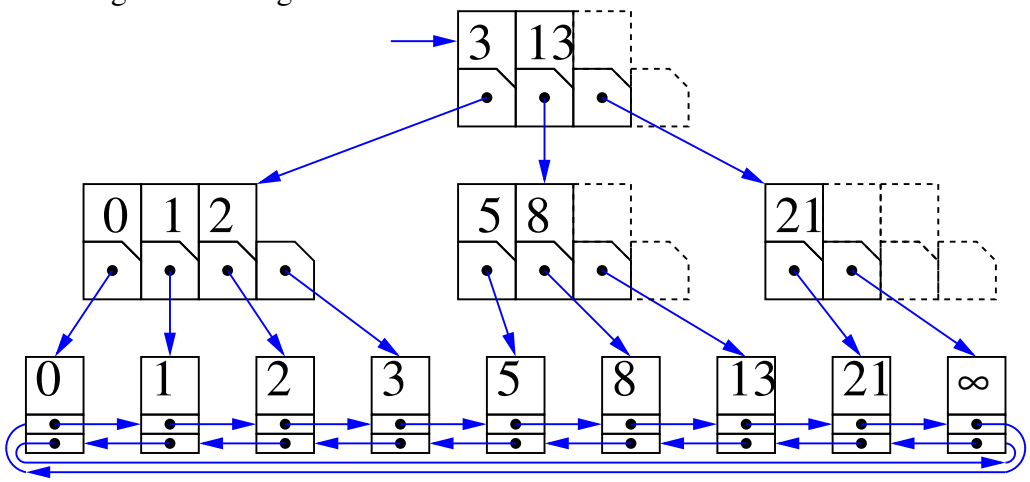
Aufgabe 4. (a,b)-Bäume [8 Punkte]

a. Geben Sie einen gültigen $(2,4)$-Baum mit den Elementen $\{0,1,2,3,5,8,13,21\}$ an. Ergänzen Sie dazu geeignet das folgende Gerüst. [4 Punkte]



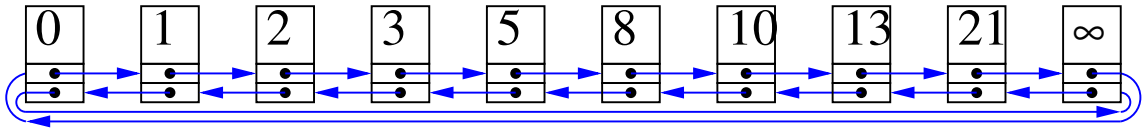
Lösung

Eine mögliche Lösung ist:



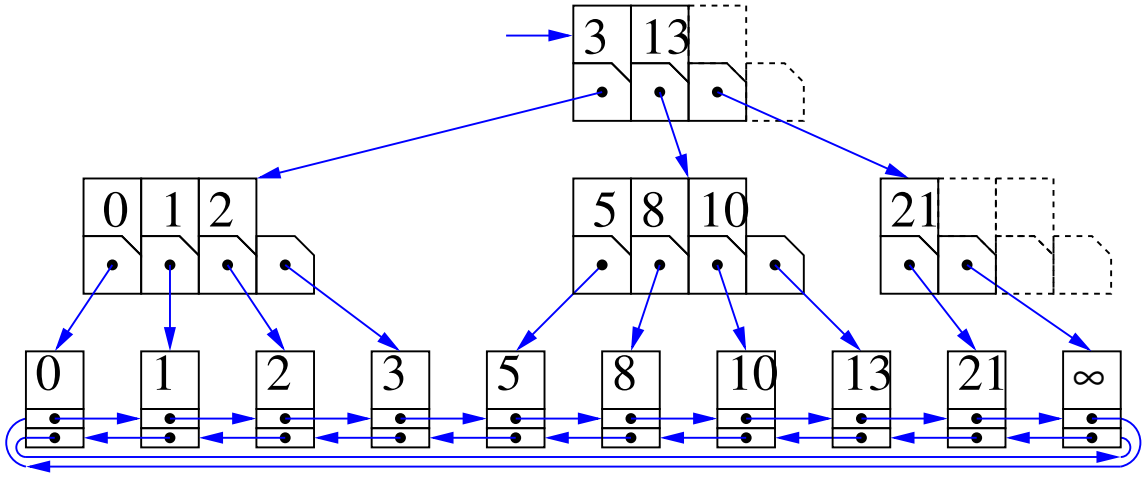
Lösungsende

b. Fügen Sie in den in **a.** konstruierten Baum das Element 10 mit dem Algorithmus aus der Vorlesung ein. Ergänzen Sie dazu die gegebene Liste um die Navigationsdatenstruktur. [4 Punkte]



Lösung

Eine mögliche Lösung ist:

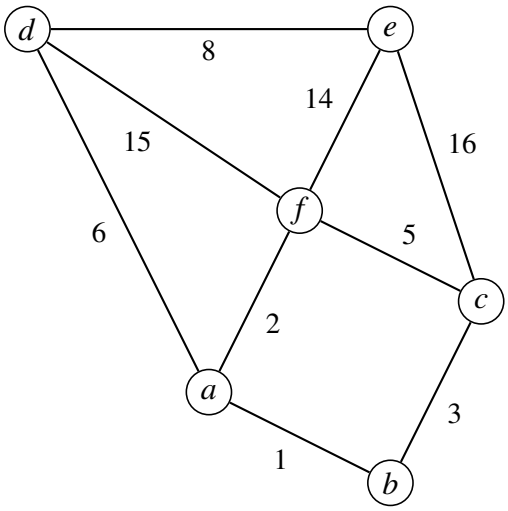


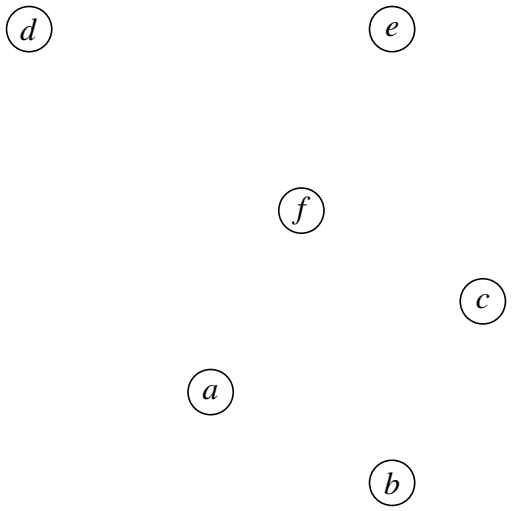
Lösungsende

Lösungsvorschlag

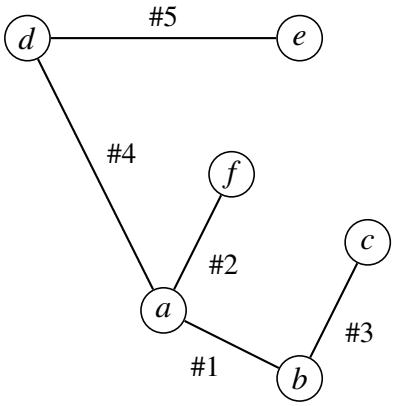
Aufgabe 5. Minimale Spannbäume [5 Punkte]

Benutzen Sie den Algorithmus von Jarník-Prim aus der Vorlesung, um für den links gegebenen Graphen einen minimalen Spannbaum (MST) auszurechnen. Verwenden Sie Knoten a als Startknoten. Zeichnen Sie den MST in das rechte Knotengerüst ein und NUMMERIEREN Sie die Kanten in der Reihenfolge, in der Sie hinzugefügt werden.





Lösung



Lösungsende

Lösungsvorschlag

Aufgabe 6. Hashing [4 Punkte]

Betrachten Sie die folgende Hashtabelle:

0	1	2	3	4	5	6	7	8	9
30	10	22	83	42	55		17	37	39

Zur Kollisionsauflösung wird offenes Hashing mit linearer Suche verwendet. Die Hashfunktion, $h(x) = x \bmod 10$, bildet eine Zahl auf ihre niederwertigste Ziffer ab. Geben Sie den Zustand der Tabelle nach dem Entfernen von 22 an.

0	1	2	3	4	5	6	7	8	9

Lösung

0	1	2	3	4	5	6	7	8	9
30	10	42	83		55		17	37	39

Lösungsende