Name:

Vorname:

Matrikelnummer:

Klausur-ID:

Lösungsvorschlag

Karlsruher Institut für Technologie
Institut für Theoretische Informatik

Prof. Dr. P. Sanders 21.02.2018

Klausur Algorithmen II

Aufgabe 1.	Kleinaufgaben	17 Punkte
Aufgabe 2.	Starke Zusammenhangskomponenten	10 Punkte
Aufgabe 3.	Flussalgorithmen	10 Punkte
Aufgabe 4.	RangeMode Queries	8 Punkte
Aufgabe 5.	Onlinealgorithmen und randomisierte Algorithmen	9 Punkte
Aufgabe 6.	Geometrische Algorithmen	6 Punkte

Bitte beachten Sie:

- Als Hilfsmittel ist nur **ein** DIN-A4 Blatt mit Ihren **handschriftlichen** Notizen zugelassen.
- **Schreiben** Sie auf **alle** Blätter der Klausur und Zusatzblätter Ihre **Klausur-ID**.
- Merken Sie sich Ihre **Klausur-ID** auf dem Aufkleber für den Notenaushang.
- Die Klausur enthält **14 Blätter**.
- Zum Bestehen der Klausur sind 20 Punkte hinreichend.

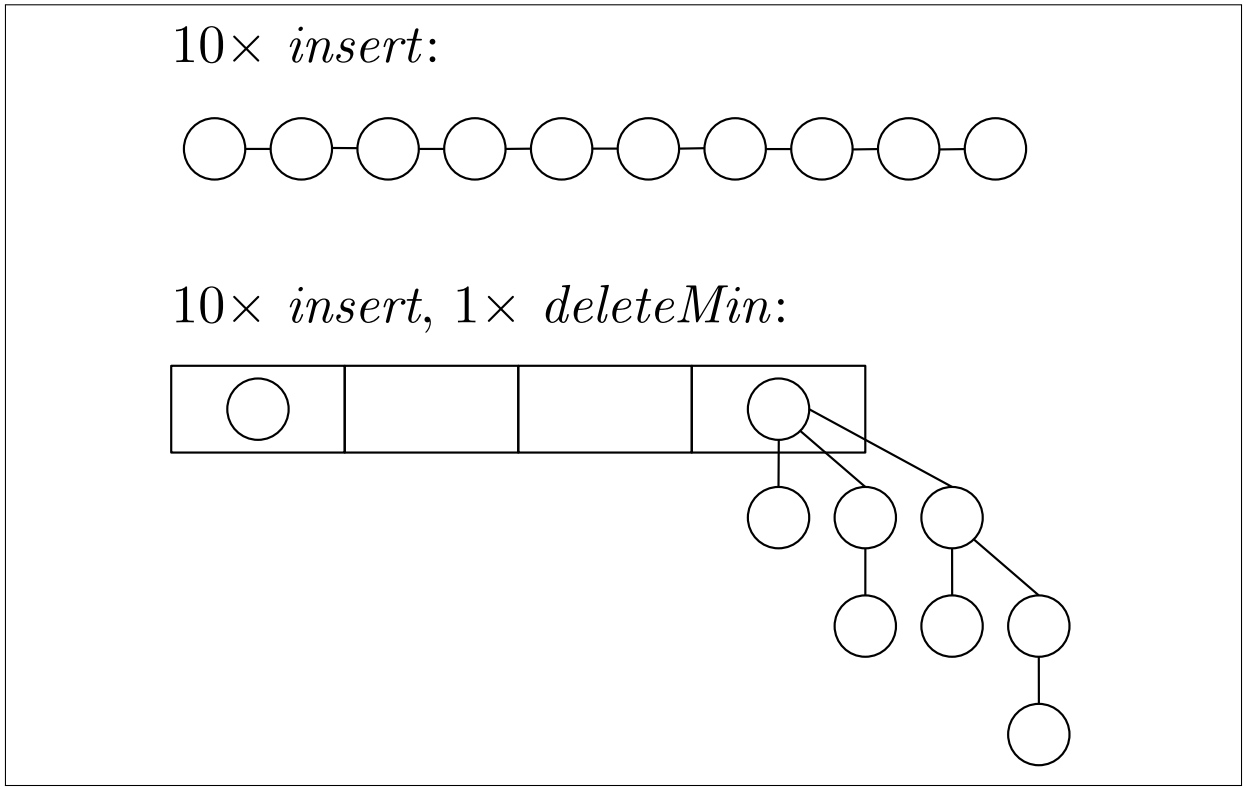
Lösungsvorschlag

Aufgabe 1. Kleinaufgaben [17 Punkte]

a. Betrachten Sie die folgenden Sequenzen von Operationen auf einem Fibonacci Heap. [2 Punkte]

1. Zeichnen Sie die Struktur eines Fibonacci Heaps nach zehn *insert*-Operationen.
2. Zeichnen Sie die Struktur eines Fibonacci Heaps nach zehn *insert*-Operationen gefolgt von einer *deleteMin*-Operation.

Lösung



b. Sie führen auf einem Fibonacci Heap folgende Operationen aus: $n+1$ *insert*-Operationen gefolgt von einer *deleteMin*-Operation. Nennen Sie alle vorkommenden Ränge im Fibonacci Heap. Beweisen Sie Ihre Aussage.
Dabei sei $n = \sum_{i=0}^{\lfloor \log n \rfloor} b_n[i] \cdot 2^i, b_n[i] \in \{0,1\}$.
Sie können dabei verwenden, dass sich die Schritte n und $n+1$ lediglich durch das Hinzufügen eines einzelnen Knotens unterscheiden.
Hinweis: Verwenden Sie einen induktiven Beweis.

Lösung

Sei b_n die Binärrepräsentation von n. Der Fibonacci Heap besteht aus den Bäumen der Ränge i mit $b_n[i] = 1$.
Beweis: Nach den angegebenen Operationen besteht der Heap aus n einzelnen Knoten, die nun entsprechend des Algorithmus vereinigt werden.
Wir verwenden hierbei, dass sich die Schritte n und $n+1$ lediglich dadurch unterscheiden, dass im union-Schritt ein Element mehr vereinigt wird.

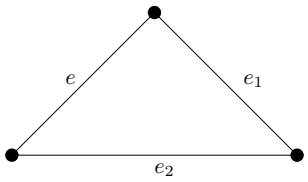
- IA: $n = 1$: Der Heap besteht aus einem einzelnen Knoten (der offensichtlich Rang 0 hat).
- IS: $n \to n+1$: Die Aussage gelte für n Elemente. Jetzt wird ein weiteres Element hinzugefügt. Betrachte die Folge von Rängen angefangen bei 0 bis l, so dass Bäume mit Rang 0 bis $l-1$ im Heap vorhanden sind und kein Baum mit Rang l im Heap ist. Nach Induktionsvoraussetzung sind die Bits $b_n[0] \ldots b_n[l-1]$ alle gleich 1 und $b_n[l] = 0$. Durch das Hinzufügen eines weiteren Elements werden im Fibonacci Heap sukzessiv alle Bäume mit Rang 0 bis $l-1$ vereinigt, bis ein Baum der Größe l eingefügt wird. Dies entspricht dem Addieren von 1 auf eine Binärzahl.

[4 Punkte]

c. Gegeben sei ein vollständiger, schlingenfreier, ungerichteter Graph $G = (V,E)$ mit $|V| \geq 3$ und eine Kantengewichtsfunktion c, die die Dreiecksungleichung erfüllt. Zeigen Sie, dass für alle $e \in E$ gilt: $c(e) \geq 0$.

Lösung

Beweis per Widerspruch: Wir betrachten den folgenden Teilgraphen:



Angenommen $c(e) < 0$, dann

$$c(e_1) + c(e) \geq c(e_2) \Rightarrow c(e_1) > c(e_2)$$

und

$$c(e_2) + c(e) \geq c(e_1) \Rightarrow c(e_2) > c(e_1),$$

was zum Widerspruch führt.

[3 Punkte]

d. Gegeben sei ein Suffixarray (SA). Geben Sie einen möglichst I/O-effizienten Algorithmus im externen Speichermodell an, der das inverse Suffixarray (SA^{-1}) berechnet. Die Größe des schnellen Speichers sei dabei M und die Blockgröße B. Geben Sie außerdem asymptotisch die Anzahl I/Os an, die Ihr Algorithmus braucht.

Lösung

Der Algorithmus funktioniert wie folgt: Gegeben ein Suffixarray $SA = \pi_1, \pi_2, \ldots, \pi_n$, scanne über SA und erzeuge ein neues Array $SA' = (\pi_1, 1), (\pi_2, 2), \ldots, (\pi_n, n)$. Sortiere dann SA' nach der ersten Komponente. Scanne dann über SA' und gebe die hinteren Elemente aus. Die Anzahl I/Os wird durch das Sortieren dominiert. Die Anzahl I/Os ist damit in $\mathcal{O}\left(\frac{n}{B}(1 + \lceil \log_{M/B} \frac{n}{M} \rceil)\right)$.

[3 Punkte]

e. Zu einem Problem P habe der *bisher* beste sequenzielle Algorithmus eine Laufzeit in $\mathcal{O}(A)$. Sie finden einen neuen, parallelen Algorithmus mit Laufzeit $\mathcal{O}(B)$. Geben Sie den Speedup zum besten sequenziellen Algorithmus in Abhängigkeit von p mit folgenden Werten für A und B an.

Lösung

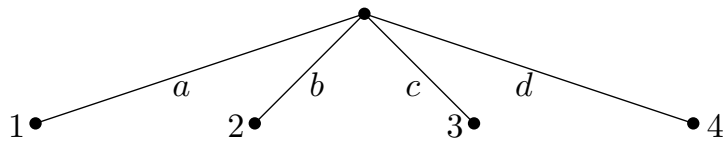
1. $A = n + m \log n, \quad B = \frac{n + m \log \log n}{\sqrt{p}}$
   - Die Laufzeit von B ist für $p = 1$ besser als die Laufzeit A. Für den Speedup muss also die sequenzielle Laufzeit von B verwendet werden.
     $S = \sqrt{p}$
2. $A = m \log n, \quad B = \frac{m \cdot n^{0{,}001}}{p}$. Was können Sie über den Speedup für große n sagen?
   - $S = \frac{m \log n}{\frac{m \cdot n^{0{,}001}}{p}} = p \cdot \frac{\log n}{n^{0{,}001}}$. Für $n \to \infty$ geht der Speedup also gegen 0.

[2 Punkte]

f. Dekomprimieren Sie den unten stehenden mittels Lempel-Ziv Kompression komprimierten Text. Vervollständigen Sie dazu den unten stehenden Trie und geben Sie den dekomprimierten Text an.

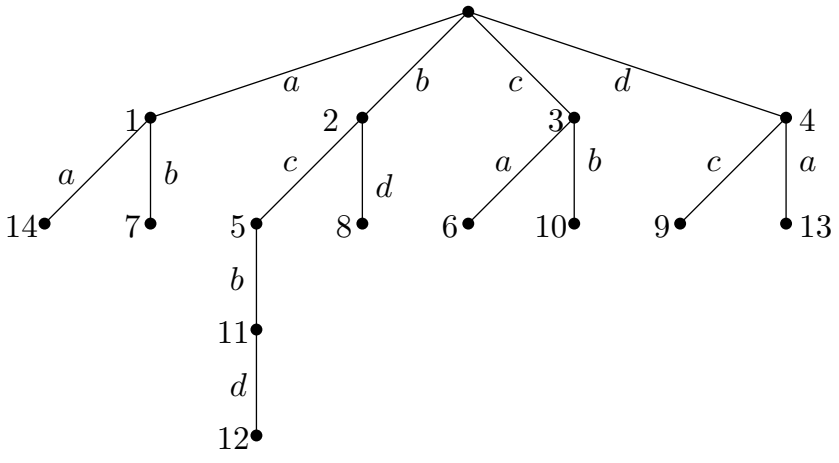
Komprimierter Text:

2	3	1	2	4	3	5	11	4	1	14



Lösung

Komprimierter Text:	2	3	1	2	4	3	5		11			4	1	14	
Dekomprimierter Text:	b	c	a	b	d	c	b	c	b	c	b	d	a	a	a



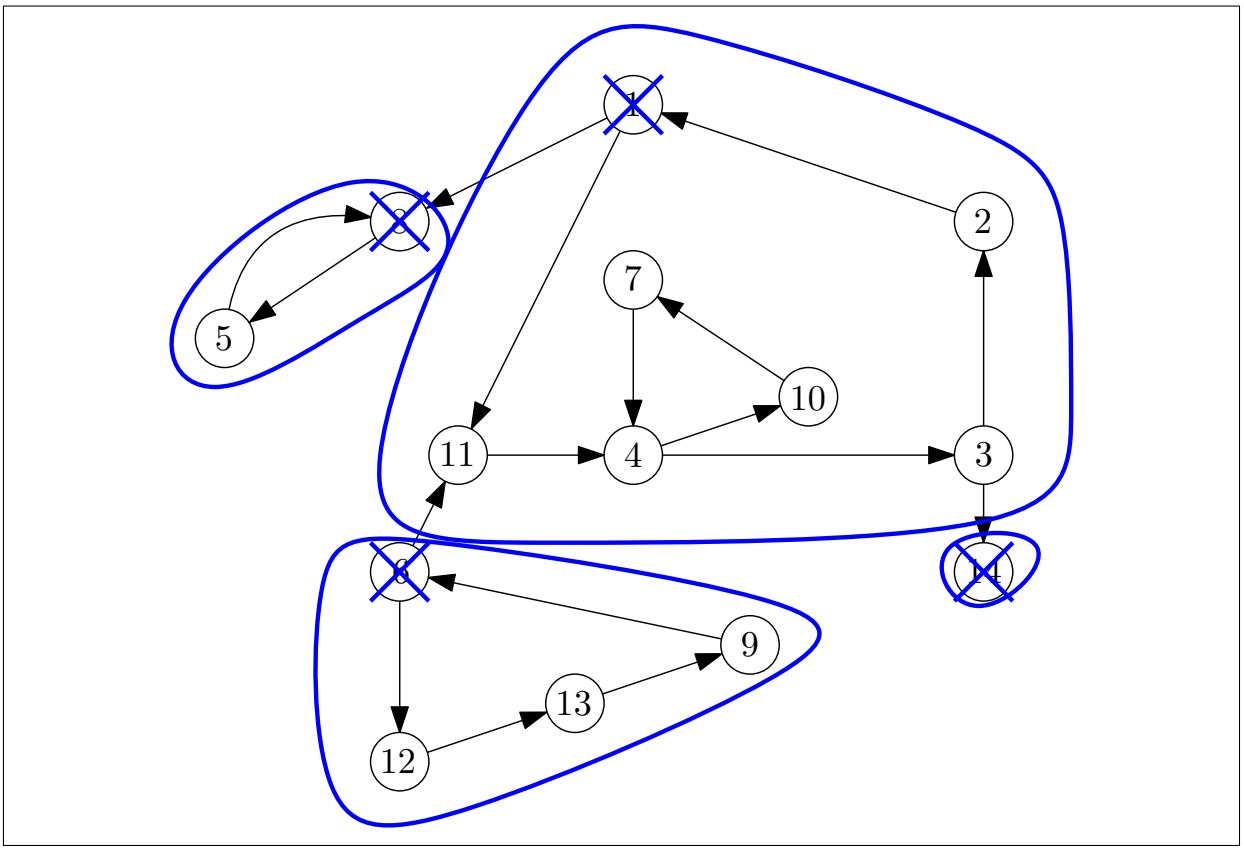
[3 Punkte]

Lösungsvorschlag

Aufgabe 2. Starke Zusammenhangskomponenten [10 Punkte]

a. Zeichnen Sie in den gegebenen Graphen alle starken Zusammenhangskomponenten ein. Markieren Sie außerdem für jede starke Zusammenhangskomponente den Repräsentanten, der durch den Algorithmus aus der Vorlesung gefunden wird, durch Ankreuzen.
Beginnen Sie Ihre Tiefensuchen dabei bei Knoten 1. Falls Sie mehrere Tiefensuchen verwenden, starten Sie diese immer bei dem Knoten mit der kleinsten Nummer, der noch nicht besucht wurde.

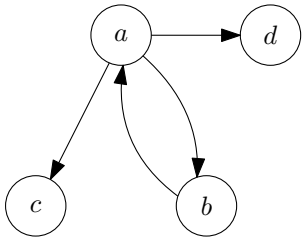
Lösung



[3 Punkte]

b. Gegeben Sei ein gerichteter Graph $G = (V,E)$, sowie Beschriftungen $L : V \to \mathbb{N}$ der Knoten mit eindeutigen natürlichen Zahlen. Sei $R(v)$ die Menge der Knoten, die in G von v aus erreichbar sind. Geben Sie einen Algorithmus in Pseudocode an, der in $\mathcal{O}(n+m)$ Zeit $M(v) = \min\{L(w) \mid w \in R(v)\}$ für alle $v \in V$ berechnet. Begründen Sie ihre Laufzeit. Betrachten Sie auch untenstehendes Beispiel.
Hinweis: Überlegen Sie sich, wie Sie das Problem auf einem azyklischen Graphen lösen würden.

v	$L(v)$	$M(v)$
a	2	1
b	3	1
c	1	1
d	4	4



Lösung

Algorithmus 1 `MinLabels(G) mit G=(V,E): Ein gerichteter Graph`

$S = (V^S, E^S) \leftarrow$ Schrumpfgraph von G
$M \leftarrow$ Knotenarray für Knoten aus V^S
for $v \in V^S$ **do**
    $M[v] \leftarrow \min\{L(w) | w \in v\}$
for $v \in V^S$ in umgekehrter topologischer Reihenfolge **do**
    for $(w,v) \in E^S$ **do**
        $M[v] \leftarrow \min\{M[v], M[w]\}$
result $\leftarrow$ Knotenarray für Knoten aus V
for $v_S \in V^S$ **do**
    for $v \in v_S$ **do**
        result$[v] \leftarrow \min[v_S]$
return *result*

Laufzeit: Der Schrumpfgraph und die topologische Sortierung können mittels Tiefensuche in $\mathcal{O}(n+m)$ gefunden werden. Die restlichen Schleifen laufen alle in $\mathcal{O}(n+m)$.

[7 Punkte]

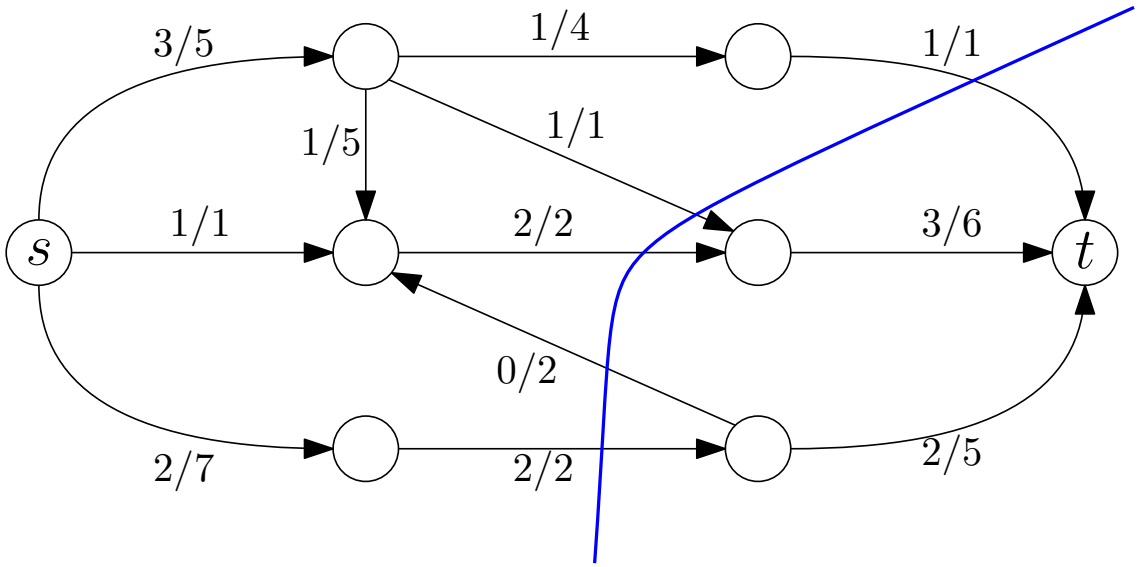
Lösungsvorschlag

Aufgabe 3. Flussalgorithmen [10 Punkte]

a. Betrachten Sie das unten stehende Flussnetzwerk mit dem eingetragenen Fluss und Kapazitäten. Zeigen Sie, dass der eingetragene Fluss maximal ist.

[2 Punkte]

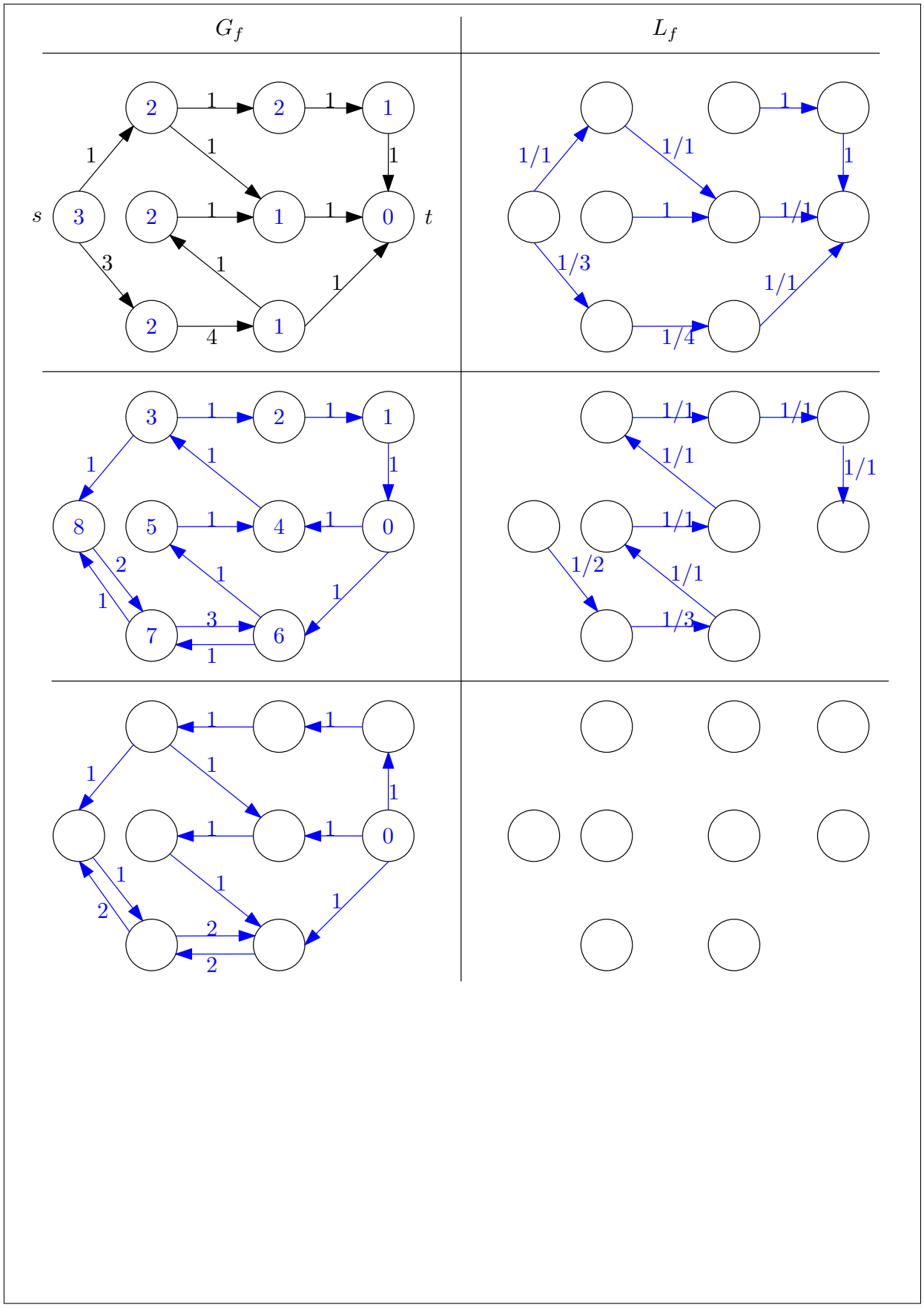
Lösung



Der Fluss hat Wert 6, da s einen ausgehenden Fluss von 6 hat (bzw. t einen eingehenden Fluss von 6). Der eingezeichnete Schnitt hat ebenfalls Wert 6. Aufgrund des *min-cut-max-flow*-Theorems, ist der Fluss damit maximal.

b. Gegeben sei das Flussnetzwerk auf der nächsten Seite. Führen Sie den Algorithmus von Dinitz darauf aus um einen maximalen Fluss zu berechnen. Tragen Sie dazu in die linke Spalte den Residualgraphen ein und schreiben Sie die Distanzlabels in die Knoten. Tragen Sie in die rechte Spalte den Schichtgraphen ein und markieren Sie den gefundenen Fluss, indem Sie die Kanten mit ihrer Kapazität und dem darüber laufenden Fluss beschriften. Nutzen Sie pro Phase eine Zeile.
Kennzeichnen Sie deutlich, wenn eine der Kopien des Graphen nicht gewertet werden soll. [8 Punkte]

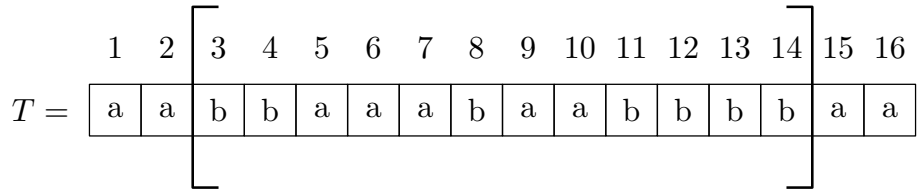
Lösung



Lösungsvorschlag

Aufgabe 4. RangeMode Queries [8 Punkte]

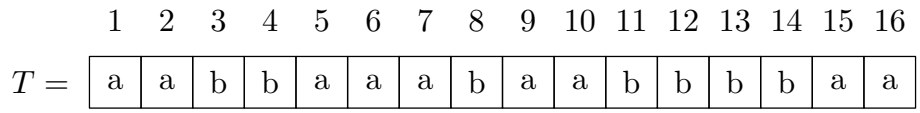
Gegeben sei ein String T der Länge n mit unbekannter (nicht konstanter) Alphabetgröße. Die Anfrage $RangeMode(T,l,r)$ berechnet eines der am häufigsten vorkommende Zeichen in $T[l]\dots T[r]$. Betrachten Sie dazu das folgende Beispiel:



$$RangeMode(T, 3, 14) = b$$

a. Berechnen Sie $RangeMode(T,5,13)$ für folgenden Text:

Lösung



$$RangeMode(T, 5, 13) = a$$

[1 Punkt]

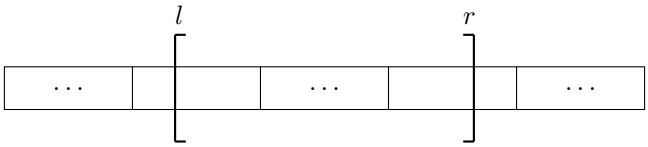
b. Seien nun l und r Vielfache von $\lceil\sqrt{n}\rceil$. Entwerfen Sie eine Datenstruktur mit $\mathcal{O}(n)$ Speicherverbrauch, die für einen gegebenen Text T vorberechnet werden kann, mit der Sie Anfragen $\mathit{RangeMode}(T,l,r)$ in $\mathcal{O}(1)$ Zeit beantworten können.

Lösung

Es gibt $\mathcal{O}(\sqrt{n}\cdot\sqrt{n}) = \mathcal{O}(n)$ mögliche Anfragen, bei denen l und r Vielfache von $\lceil\sqrt{n}\rceil$ sind. Die Antworten auf diese Anfragen können wir also alle vorberechnen und zum Beispiel in einem 2-D-Array der Größe $\sqrt{n}\times\sqrt{n}$ abspeichern.

[2 Punkte]

c. Seien nun l und r beliebige Indizes im Text. Entwerfen Sie eine Datenstruktur, die für einen gegebenen Text T vorberechnet werden kann, mit der Sie Anfragen $\mathit{RangeMode}(T,l,r)$ beantworten können. Ihre Datenstruktur darf $\mathcal{O}(n\log n)$ Speicher verbrauchen und Zeit $\mathcal{O}(\sqrt{n}\log n)$ für Anfragen in Anspruch nehmen. Die Datenstruktur aus Teilaufgabe **b** sei dabei gegeben. *Hinweis:* Überlegen Sie sich, wie Sie die Zahl der Kandidaten auf $\mathcal{O}(\sqrt{n})$ einschränken können.



[5 Punkte]

Lösung

Sollten l und r keine Vielfachen von $\lceil\sqrt{n}\rceil$ sein, bezeichnen wir das erste Vielfache von $\lceil\sqrt{n}\rceil$ innerhalb des Anfragebereiches mit l_b und das letzte mit r_b. Jetzt gibt es zwei Möglichkeiten dafür, welches Zeichen am häufigsten ist:

1. Entweder es ist das Zeichen, welches von l_b bis r_b am häufigsten ist (das kann mit der Datenstruktur aus Teilaufgabe **b** in $\mathcal{O}(1)$ berechnet werden),
2. Oder es ist eines der Zeichen zwischen l und l_B oder zwischen r_b und r. Dies sind maximal $2\cdot\lceil\sqrt{n}\rceil$ mögliche Kandidaten.

Mithilfe eines Wavelet Trees können *rank*-Anfragen in $\mathcal{O}(\log n)$ beantwortet werden. Die Größe des Wavelet Trees ist dabei in $\mathcal{O}(n\log n)$. Wir finden also die Häufigkeit aller Kandidaten x in je $\mathcal{O}(\log n)$ Zeit mittels $\mathit{rank}(r+1,x,T) - \mathit{rank}(l,x,T)$. Damit ergibt sich eine Gesamtlaufzeit von $\mathcal{O}(\sqrt{n}\log n)$.

Lösungsvorschlag

Aufgabe 5. Onlinealgorithmen und randomisierte Algorithmen [9 Punkte]

a. Sie möchten in einem Haus in eines der oberen Stockwerke gelangen. Der Fahrstuhl ist allerdings sehr unzuverlässig. Wenn Sie den Knopf betätigen um den Fahrstuhl zu rufen, wissen Sie nicht, wie lange es dauert, bis er bei Ihnen ankommt. Das kann beliebig lange dauern.
Angenommen, der Fahrstuhl braucht Zeit E, um das gewünschte Stockwerk zu erreichen (nachdem Sie den Fahrstuhl betreten haben) und es dauert Zeit S die Treppe zu nehmen. Die Werte $0 < E < S$ seien bekannt. Es gilt also zu entscheiden, wie lange man auf den Fahrstuhl warten sollte, bevor man die Treppe nimmt.
Entwerfen Sie eine (deterministische) Strategie mit Competitive Ratio < 2 (nicht gleich 2). Geben Sie Ihren Competitive Ratio an.

Lösung

Wir warten maximal Zeit $S - E$ auf den Fahrstuhl. Wenn er nach dieser Zeit nicht gekommen ist, nehmen wir die Treppe.

- Wenn der Fahrstuhl kommt, während wir noch warten, sind wir optimal, da wir schneller in unserem Zielstockwerk ankommen als wenn wir die Treppe genommen hätten.
- Anderenfalls wäre es optimal die Treppe zu nehmen (also Zeit S). Wir haben aber $S - E + S$ Zeit gebraucht. Der Competitive Ratio ist damit $\frac{S-E+S}{S} = 2 - \frac{E}{S}$.

[3 Punkte]

b. Betrachten Sie folgenden Algorithmus, wobei $randInt(n)$ in konstanter Zeit unabhängig gleichverteilt eine zufällige Ganzzahl aus $[0, n-1]$ zurückgibt:

Algorithmus 2 n: `eine ganze Zahl` > 0

```
A ← [0,0,...,0] (n Bits)
i ← 0
while i < n do
    repeat
        x ← randInt(n)
    until A[x] = 0
    A[x] ← 1
    i ← i + 1
```

Hinweis: Für $0 < x < 1$ gilt $\sum_{k=0}^{\infty} k \cdot x^{k-1} = \frac{1}{(1-x)^2}$.

Lösung

1. Bestimmen Sie für $0 \leq i < n$ die Wahrscheinlichkeit p_i, dass im i-ten Schleifendurchlauf der äußeren Schleife $A[x] = 0$ ist.

   $p_i = 1 - \frac{i}{n} = \frac{n-i}{n}$

2. Für $0 \leq i < n$ sei X_i die Anzahl der Durchläufe der inneren Schleife im i-ten Durchlauf der äußeren Schleife ist. Bestimmen Sie den Erwartungswert von X_i.

   Für $i > 0$ gilt: $\mathbb{E}(X_i) = \sum_{k=0}^{\infty} k \cdot (1-p_i)^{k-1} \cdot p_i = p_i \cdot \sum_{k=0}^{\infty} k \cdot (1-p_i)^{k-1} = p_i \cdot \frac{1}{p_i^2} = \frac{1}{p_i}$

   Für $i = 0$ ist $p_i = 1$ und damit $\mathbb{E}(X_0) = 1 = \frac{1}{p_0}$

   (Alternativ kann auch mit dem Erwartungswert geometrisch verteilter Zufallsvariablen argumentiert werden)

3. Berechnen Sie asymptotisch die erwartete Laufzeit des Algorithmus.

   $\mathbb{E}(\sum_{i=0}^{n-1} X_i) = \sum_{i=0}^{n-1} \mathbb{E}(X_i) = \sum_{i=0}^{n-1} \frac{1}{p_i} = \sum_{i=0}^{n-1} \frac{n}{n-i} = n \cdot \sum_{i=0}^{n-1} \frac{1}{n-i} = n \cdot \sum_{i=1}^{n} \frac{1}{i} = n \cdot H_n$. Wobei H_n die n-te harmonische Zahl ist. Die Laufzeit des Algorithmus ist daher in $\mathcal{O}(n \log n)$.

[6 Punkte]

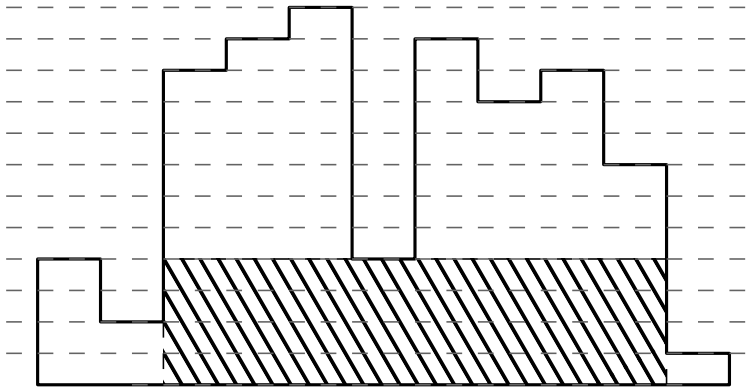
Lösungsvorschlag

Aufgabe 6. Geometrische Algorithmen [6 Punkte]

a. Gegeben sei ein Histogramm, also eine Folge von aneinandergrenzenden Balken gleicher Breite aber unterschiedlicher Höhen (siehe Abbildung unten). Sie erhalten die Höhen der Balken von links nach rechts. Entwerfen Sie einen möglichst zeiteffizienten Algorithmus in Pseudocode, der die Größe des größten Rechtecks unterhalb des Histogramms findet (siehe schraffierte Fläche in der Abbildung unten).
Falls nötig, dürfen sie am Anfang und Ende der Eingabe Pseudoelemente einfügen.

- Lösungen in $\mathcal{O}(n^3)$ erhalten bis zu **1** Punkt,
- Lösungen in $\mathcal{O}(n^2)$ erhalten bis zu **3** Punkte,
- Lösungen in $\mathcal{O}(n)$ erhalten bis zu **6** Punkte. *Hinweis:* Verwenden Sie einen Stack.

Begründen Sie die Laufzeit Ihres Algorithmus.



Lösung

Wir fügen an das Ende der Eingabe ein Pseudoelement mit Höhe 0 ein.

Algorithmus 3 H: `Höhen der Balken,` n: `Anzahl Balken`

```
maxSize ← 0
S ← leerer Stack (aus Elementen der Form (x,h))
S.push(0,0)
for i from 1 to n do
    x ← i
    while S.top.h ≥ H[i] do
        t ← S.pop()
        x ← t.x                                  // Speichere x, mit H[x] ≥ H[i]
        maxSize ← max{maxSize, t.h · (i − t.x)}  // Rechteck mit Höhe t.h endet bei i
    S.push(x, H[i])                              // Benutze kleinstmögliches x für H[i]
return maxSize
```

Die *for*-Schleife wird für jedes Element genau ein mal durchlaufen. Weiterhin wird jedes Element genau ein mal vom Stack genommen, womit auch die *while*-Schleife n mal durchlaufen wird.

[6 Punkte]