Algorithmen und Datenstrukturen 1 (II)

SoSe 2020 · Prof. Dr. U. Meyer
05. Oktober 2020
Klausurnummer: **0001**

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Nachname (Druckschrift)**:** _____

Vorname (Druckschrift)**:** _____

Matrikelnummer: _____

Studiengang: _____

Bitte Hinweise beachten:

- Schreiben Sie Ihren **Namen nur auf dieses Titelblatt**. Sollten Sie Zusatzpapier bekommen, vermerken Sie darauf unbedingt die Klausurnummer **0001**.
- Diese Klausur ist für die Variante **Algo1 (vollständig)**.
- Die Klausur besteht aus den Aufgaben **1 − 9**; Rückseiten werden verwendet.
- Merken oder notieren Sie sich Ihre Klausurnummer **0001**, da nur unter dieser Nummer die Ergebnisse veröffentlicht werden.
- Es dürfen nur **dokumentenechte Stifte** in den Farben blau und schwarz verwendet werden. Insb. ist die Nutzung von **Tintenlöschern und Tipp-Ex** untersagt. Zugelassene Hilfsmittel:
  1 Blatt DIN A4 mit handschriftlichen Notizen (zweiseitig).
- Das Mitbringen nicht zugelassener Hilfsmittel stellt einen Täuschungsversuch dar und führt zum Nichtbestehen der Klausur. **Schalten Sie bitte deshalb alle elektronischen Geräte, insbesondere Handys und Smartwatches, vor Beginn der Klausur aus und packen Sie diese weg.**
- Werden zu einer Aufgabe zwei oder mehr Lösungen angegeben, so gilt die Aufgabe als nicht gelöst. Entscheiden Sie sich also immer für **eine** Lösung. Begründungen sind nur dann notwendig, wenn die Aufgabenformulierung dies verlangt.
- Die Klausur ist mit Sicherheit bestanden, wenn (ohne Bonifikation aus den Übungspunkten) mindestens **50%** der Höchstpunktzahl erreicht wird.
- Die Klausur dauert **180 Minuten**.

2 0 1 2 0 0 0 1 0 0 0 1 5

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Diese Seite ist nur für den internen Gebrauch bestimmt.
Bitte nicht beschreiben.

Aufgabe	1	2	3	4	5	6	7	8	9
Erreichbar	23	22	16	20	19	24	12	10	14
Erreicht									

Klausur	Bonfikation

2 0 1 2 0 0 0 1 0 0 0 1 5

Aufgabe 1: Union-Find — $8 + 8 + 7 = 23$ Punkt(e)

In der Vorlesung präsentierten wir die Union-Find Datenstruktur sinngemäß wie folgt:

```
struct UnionFind {
  int[] parent; // Elternarray
  int[] helper; // siehe Aufgabenteil b
};
```

a) Betrachten Sie folgende langsame Implementierung:

```
// Initialisiere UnionFind
function create(int n) {
  UnionFind uf;
  // allokiere Array[0, ..., n-1]:
  uf.parent = new int[n];
  for(int i = 0; i < n; ++i) {
    uf.parent[i] = i;
  }

  return uf;
}
```

```
// Verschmelze Bäume von u und v.
// uf wird als Referenz übergeben
// und kann verändert werden
function union(UnionFind uf,
               int u, int v) {
  while (uf.parent[u] != u) {
    u = uf.parent[u]; }
  while (uf.parent[v] != v) {
    v = uf.parent[v]; }
  uf.parent[v] = u;
}
```

Wir suchen eine Sequenz von $\Theta(n)$ sukzessiven `union` Aufrufen, deren **Gesamtlaufzeit möglichst groß** ist. Die Sequenz soll unmittelbar nach einem `create(n)` starten.

Beschreiben Sie eine solche Sequenz für alle $n>1$, nennen Sie ihre Gesamtlaufzeit asymptotisch exakt[1], und begründen Sie wie sich diese Laufzeit ergibt.

Lösungsskizze: Wir nutzen die folgende Sequenz:
`union`$(1, 0)$ // 0 hängt unter 1
`union`$(2, 0)$ // 1 hängt unter 2
$\ldots$`union`$(i, 0)$ $\ldots$
`union`$(n-1, 0)$
Die Datenstruktur kodiert also einen Pfad $0 \to 1 \to \ldots \to n-1$. Da wir jedes Mal im Blatt 0 starten, muss die zweite `while`-Schleife im $i$ten Aufruf $\Theta(i)$ Iterationen durchlaufen. Die Gesamtlaufzeit folgt also als $\sum_{i=0}^{n-1} i = \Theta(n^2)$.

Gesamtlaufzeit: $\Theta(n^2)$

⇑⇑⇑ _____ / 8 Punkt(e) ⇑⇑⇑

[1] Übereinstimmende obere und untere Schranken.

b) In der Vorlesung und Übung diskutierten wir unterschiedliche Ansätze, die Laufzeit mit Hilfe eines weiteren Arrays (im Folgenden `helper` genannt) zu verbessern. Wählen und beschreiben Sie eine dieser Möglichkeiten natürlichsprachlich. Nennen Sie die verbesserte Laufzeit von `union` und `find` und skizzieren Sie kurz die Beweisidee.

Lösungsskizze: Wir müssen die Tiefe der Union-Find Bäume beschränken. In der Vorlesung stellten wir sicher, dass sich die Tiefe eines Baums höchstens dann erhöht, wenn er mit einem größeren verschmolzen wird. Dies führt induktiv zu einer Tiefe von $\mathcal{O}(\log n)$.

Wir erzielten dies mit Hilfe folgender Invariante: Wenn i eine Wurzel ist, speichert `helper`$[i]$ die Anzahl der Knoten im Baum, der in i gewurzelt ist; d.h. initial haben alle Einträge von `helper` den Wert 1. Seien u und v zwei Wurzeln, wobei wir dann v unter u hängen, dann gilt `helper`$[u] \leftarrow$ `helper`$[u] +$ `helper`$[v]$; der Wert von v muss nicht angepasst werden, da v keine Wurzel mehr ist (und es auch nie wieder sein wird). Bei jedem Verschmelzen ermitteln wir also eine Wurzel, deren Gewicht nicht kleiner als das der anderen ist, und nutzen diese als Wurzel des verschmolzenen Baums.

Worst-Case Laufzeit von `union`: $\mathcal{O}(\log n)$ oder $\mathcal{O}(1)$

Worst-Case Laufzeit von `find`: $\mathcal{O}(\log n)$

⇑⇑⇑ _____ / 8 Punkt(e) ⇑⇑⇑

c) Implementieren Sie `create` und `union` (Semantik siehe a)) einer möglichst effizienten Union-Find Datenstruktur indem Sie den folgenden **Pseudocode vervollständigen**.

```
function create(int n) {
  UnionFind uf;
  uf.parent = new int[n]; // allokiere Array[0, ..., n-1]
  for(int i = 0; i < n; ++i) { uf.parent[i] = i; }
```

Lösungsskizze:

```
  uf.helper = new int[n]; // allokiere Array[0, ..., n-1]
  for(int i = 0; i < n; ++i) { uf.parent[i] = 1; }
```

```
  return uf;
}

function union(UnionFind uf, int u, int v) {
  while(uf.parent[u] != u) { u = uf.parent[u]; }
  while(uf.parent[v] != v) { v = uf.parent[v]; }
```

Lösungsskizze:

```
  if (uf.helper[v] > uf.helper[u]) swap(u, v);
  uf.helper[u] += uf.helper[v];
  uf.parent[v] = u;
```

```
}
```

⇑⇑⇑ _____ / 7 Punkt(e) ⇑⇑⇑

Aufgabe 2: O-Notation und Codeanalyse — $8 + 8 + 6 = 22$ Punkt(e)

a) Geben Sie jeweils eine Funktion f an, welche die angegebenen Wachstumseinschränkungen einhält.

$f \in \omega(\frac{1}{n})$	$f \in o(\log n)$	$f(n) =$ 1
$f \in \omega(\log n)$	$f \in o(n)$	$f(n) =$ $\sqrt{n}$
$\log(f) \in \omega(n)$		$f(n) =$ 2^{n^2}
$f \in \Theta(n)$	$2^f \in \omega(2^n)$	$f(n) =$ $2n$

⇑⇑⇑ _____ / 8 Punkt(e) ⇑⇑⇑

b) Lösen Sie die folgenden Rekursionsgleichungen auf. Für alle Rekursionsgleichungen gilt $T(n) = 1$ für alle $n \leq 1$. Nehmen Sie vereinfachend an, dass n so gewählt wurde, so dass sämtliche Divisionen restfrei aufgehen.

$T(n) = 4 \cdot T(n - \frac{n}{2}) + 3n$	$T(n) = \Theta($ n^2 $)$
$T(n) = 2 \cdot T(n - 1) + 3$	$T(n) = \Theta($ $3 \cdot 2^n - 3$ $)$
$T(n) = 2 \cdot T(n - \frac{3n}{4}) + 3$	$T(n) = \Theta($ $\sqrt{n}$ $)$
$T(n) = 3 \cdot T(n - \frac{2n}{3}) + 2n$	$T(n) = \Theta($ $n \log n$ $)$

⇑⇑⇑ _____ / 8 Punkt(e) ⇑⇑⇑

c) Geben Sie für die folgenden Algorithmen jeweils die Laufzeit in Θ-Notation an.

Algorithmus loop1(n)
```
s = 0
i = 1
while i * i ≤ n
    for j = 1 to i
        s = s + 1
    i = i + 1
```

Algorithmus loop2(n)
```
i = 1
while i ≤ n * n
    i = 3 * i
```

Algorithmus loop3(n)
```
i = 1
while i ≤ n
    j = 0
    while j ≤ i
        j = j + 1
    i = 2 * i
```

loop1: $\Theta($ n $)$

loop2: $\Theta($ $\log n$ $)$

loop3: $\Theta($ n $)$

⇑⇑⇑ _____ / 6 Punkt(e) ⇑⇑⇑

a) Fügen Sie die Elemente

$$2, 10, 1, 6, 4, 7, 5, 9, 3, 8$$

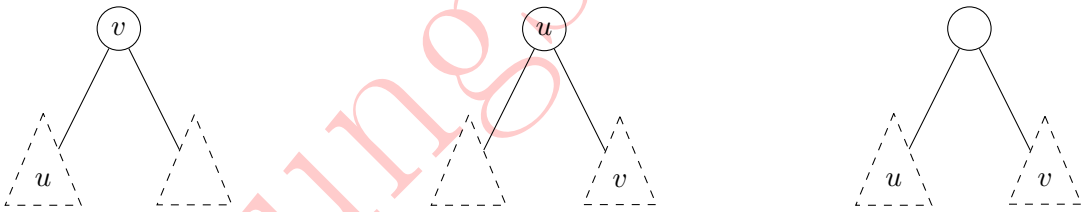
in der gegebenen Reihenfolge in einen zu Beginn leeren Min-Heap ein und stellen Sie den resultierenden Heap als Array dar.

1	2	3	4	5	6	7	8	9	10
1	3	2	4	6	7	5	10	9	8

⇑⇑⇑ _____ / 8 Punkt(e) ⇑⇑⇑

b) Zeigen oder widerlegen Sie: Eine Inorder-Traversierung eines nicht leeren AVL-Baums liefert eine sortierte Sequenz.

Lösungsskizze: Beweis durch Widerspruch:
Angenommen, die Inorder-Traversierung liefert eine unsortiere Sequenz. Dann existieren zwei Knoten u, v mit Schlüsseln $s(u)$ und $s(v)$, sodass $s(u) > s(v)$ gilt, aber u vor v in der Inorder-Traversierung besucht wird.
Knoten u vor v in der Inorder-Traversierung bedeutet, dass einer der folgenden Fälle vorliegt:



In allen diesen Fällen ist jedoch $s(u) > s(v)$ ein Widerspruch zur Suchbaum-Struktur. Diese garantiert ...

- ... alle Schlüssel im linken Teilbaum eines Knotens u nicht größer als $s(u)$ sind
- ... alle Schlüssel im rechten Teilbaum eines Knotens u nicht kleiner als $s(u)$ sind
- $\Rightarrow$ alle Schlüssel im linkten Teilbaum eines Knotens u nicht größer als ein Schlüssel in us rechten Teilbaum sind

⇑⇑⇑ _____ / 8 Punkt(e) ⇑⇑⇑

a) Gegeben ist folgende Hashtabelle der Größe 13. Es wird doppeltes Hashing mit den folgenden Funktionen verwendet:

$$f_i(k) = (h_1(k) + i \cdot h_2(k)) \bmod 13$$
$$h_1(k) = 5k \bmod 13$$
$$h_2(k) = 6 - (k \bmod 6)$$

0	1	2	3	4	5	6	7	8	9	10	11	12
30		11		41	1	4	20	3	57	2	8	5

Fügen Sie die folgenden Elemente in der gegebenen Reihenfolge in die Hashtabelle ein.

$$1, 2, 3, 4, 5$$

Rechenhilfe:

k	1	2	3	4	5
$h_1(k)$	5	10	2	7	12

⇑⇑⇑ _____ / 10 Punkt(e) ⇑⇑⇑

b) Gegeben ist folgende Hashtabelle der Größe 13. Es wird Hashing mit linearem Austesten mit den Hashfunktionen $h_i(k) = (8k + i) \bmod 13$ verwendet.

0	1	2	3	4	5	6	7	8	9	10	11	12
34	41	3	57	8	2	22	4	11	1	5	63	78

Fügen Sie die folgenden Elemente in der gegebenen Reihenfolge in die Hashtabelle ein.

$$1, 2, 3, 4, 5$$

Rechenhilfe:

k	1	2	3	4	5
$h_0(k)$	8	3	11	6	1

⇑⇑⇑ _____ / 10 Punkt(e) ⇑⇑⇑

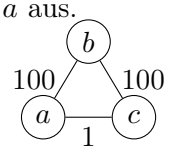
Aufgabe 5: Baum kürzester Wege — $4 + 7 + 8 = 19$ Punkt(e)

Gegeben sei ein ungerichteter und zusammenhängender Graph $G = (V, E)$ mit $V = \{1, ..., n\}$ und nicht-negativen Kantengewichten $w\colon E \to \mathbb{R}_{\geq 0}$ sowie ein fixierter Startknoten $s \in V$. Gegeben ist zudem ein Distanzarray D, das an i-ter Stelle die **Länge** eines kürzsten Weges von s nach Knoten i speichert (z.B. mittels Dijkstra berechnet).

Wir möchten nun einen gerichteten Kürzeste-Wege-Baum (KWB) $T = (V, E_T)$ berechnen, in dem s die Wurzel ist und alle Kanten in Richtung der Wurzel zeigen. Der KWB erlaubt es von jedem Knoten aus den kürzesten Weg zu s zu finden: wir müssen nur den gerichteten Kanten bis zur Wurzel folgen. Zur Berechnung des KWB modifizieren wir im Folgenden die Tiefensuche.

a) Zeigen Sie anhand eines Beispiels, dass eine in s startende unmodifizierte Tiefensuche (DFS) Kanten wählen kann, die nicht im KWB T enthalten sein dürfen.

Lösungsskizze: Betrachte folgenden Graphen und die kürzesten Wege vom Knoten a aus.



Wenn wir von einer sortierten Adjazenzliste ausgehen, wird DFS die zwei schweren Kanten nutzen, wobei im KWB die Kante $\{a, c\}$ statt $\{b, c\}$ enthalten muss.

⇑⇑⇑ _____ / 4 Punkt(e) ⇑⇑⇑

b) Beschreiben Sie **natürlich-sprachlich**, wie wir DFS durch eine Überprüfung von D modifizieren können, damit diese nur Kanten wählt, die im KWB vorhanden sein müssen. Die asymptotische Laufzeit der DFS soll hierbei unverändert bleiben.

Lösungsskizze: Angenommen DFS befindet sich gerade in Knoten v_i und entdeckt einen unbesuchten Nachbarn v_j. Dann wird dieser nur dann als besucht markiert und den Stack gelegt, falls folgende Bedingung gilt: $D[j] - D[i] = w(v_i, v_j)$.
Hierbei entstehen nur konstante Kosten pro Kante – die asymptotische Laufzeit bleibt folglich unbenommen.

⇑⇑⇑ _____ / 7 Punkt(e) ⇑⇑⇑

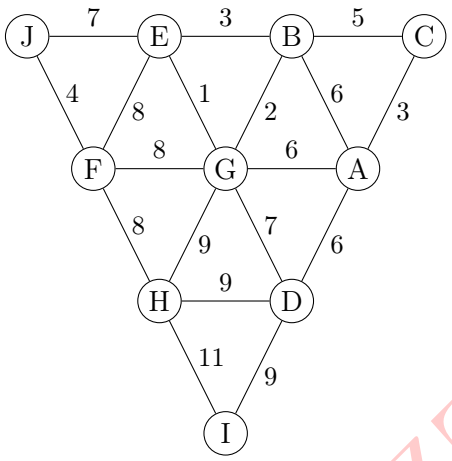
c) Implementieren Sie die modifizierte Tiefensuche **in Pseudocode**; diese soll alle Anforderungen aus b) erfüllen. Das Ergebnis soll als **Elternarray** ausgeben werden. Reproduzieren Sie dabei die relevanten Teile des Algorithmus (d.h. **DFS darf nicht als Blackbox** verwendet werden), **definieren Sie alle verwendeten Symbole**, nennen Sie Datenstrukturen (inklusive der Eingabe!), und geben Sie die Rückgabe mittels `return` aus.

Lösungsskizze: Erwarte G als Adjazenzliste. Somit können wir alle Nachbarn eines Knotens effizient enumerieren. Wir können das Besucht-Array über das berechnete Parentarray abbilden, und verwalten es daher nicht dediziert.

- Sei Q ein leerer Stack, und füge s in Q ein
- Sei $P[1 \dots n]$ das Elternarray, das mit 0 initialisiert wurde
- Setzte $P[s] \leftarrow s$
- Solange Q nicht leer ist:
  - Entferne das letzte Element aus Q und nenne es u
  - Für alle Nachbarn v von u
    - Wenn $P[v] > 0$ (bereits besucht): überspringe v
    - Wenn $D[v] - D[u] \neq w(u, v)$ (nicht kürzester Weg): überspringe v
    - $P[v] \leftarrow u$
    - Pushe v nach Q
- Return P

⇑⇑⇑ _____ / 8 Punkt(e) ⇑⇑⇑

a) Auf folgendem Graph soll ein minimaler Spannbaum sowohl mit Prims als auch Kruskals Algorithmus berechnet werden.



Geben Sie die Kanten in der Reihenfolge an, in der sie vom jeweiligen Algorithmus in den Spannbaum eingefügt werden.
Hinweis: Sie können Kanten $\{A, B\}$ vereinfachend als AB schreiben..

Reihenfolge für Prims Algorithmus bei Start in Knoten A:

$\{A, C\}, \{B, C\}, \{B, G\}, \{E, G\}, \{A, D\}, \{E, J\}, \{F, J\}, \{F, H\}, \{D, I\}$

Reihenfolge für Kruskals Algorithmus:

$\{E, G\}, \{B, G\}, \{A, C\}, \{F, J\}, \{B, C\}, \{A, D\}, \{E, J\}, \{F, H\}, \{D, I\}$

⇑⇑⇑ _____ / 16 Punkt(e) ⇑⇑⇑

b) Sei $G = (V, E)$ ein gewichteter, ungerichteter, zusammenhängender Graph und eine Kantengewichtsfunktion $w\colon E \to \mathbb{R}_{\geq 1}$. Wir betrachten die folgende Transformation der Kantengewichte:

$$w'_a(e) = 2^{\log_a(w(e))}$$

Zeigen oder widerlegen Sie: Minimale Spannbäume bleiben unter der Transformation w'_a für alle $a \in \mathbb{N}_{>1}$ erhalten. *Hinweis:* Sie können annehmen, dass jedes Kantengewicht nur einmal vorkommt.

Lösungsskizze: Da die Kantengewichte jeweils eindeutig sind, gibt es genau einen minimalen Spannbaum. Nun können wir über einen beliebigen MST-Algorithmus argumentieren.
Für alle $a \geq 2$ ist $w'_a(e) = \sqrt[\log_2(a)]{w(e)}$ eine streng monoton wachsende Funktion, d.h. für alle Kanten $e, f \in E$ mit $w(e) < w(f)$ gilt auch $w'_a(e) < w'_a(f)$. Die relative Ordnung der Kanten bzgl. ihres Gewichts ändert sich also durch die Transformation nicht. Kruskals Algorithmus betrachtet demnach die Kanten für beide Kantengewichtsfunktionen in der gleichen Reihenfolge und es entstehen somit die gleichen minimalen Spannbäume.

⇑⇑⇑ _____ / 8 Punkt(e) ⇑⇑⇑

Aufgabe 7: Huffman-Codes — 12 Punkt(e)

Geben Sie zu jedem Zeichen das Codewort eines Huffman-Codes an, der zu folgender Häufigkeitstabelle gehört.

a	b	c	d	e	f	g
1	21	36	10	12	14	45

Immer wenn zwei Teilbäume vereint werden, können Sie diese beliebig anordnen. Sollten Sie zwischen zwei gleichwertigen Knoten auswählen müssen, können Sie beliebig entscheiden.

Der Huffman-Baum muss nicht angegeben werden und wird auch nicht bewertet.

a ≙ 0000

b ≙ 011

c ≙ 10

d ≙ 0001

e ≙ 001

f ≙ 010

g ≙ 11

⇑⇑⇑ _____ / 12 Punkt(e) ⇑⇑⇑

Für $n, k \in \mathbb{N}_{>0}$ sei $Z_{k,n}$ die Menge aller Wörter in $\{0, \ldots, k-1\}^n$, deren Ziffern von links nach rechts aufsteigend sortiert sind:

$$Z_{k,n} := \{\ d_1 d_2 \cdots d_n \quad | \quad d_1, \ldots, d_n \in \{0, 1, \ldots, k-1\}, \text{ mit } d_i \leq d_{i+1}\ \forall 1 \leq i < n\ \}$$

Beispiele:

$$\{\texttt{000}, \texttt{001}, \texttt{011}, \texttt{111}\} = Z_{2,3} \subseteq Z_{3,3} \subseteq Z_{4,3}, \quad \texttt{0112} \in Z_{3,4}$$

Gegenbeispiele:

`100`	$\notin$	$Z_{2,3}$	*da nicht aufsteigend sortiert (`0` kommt nach `1`)*
`002`	$\notin$	$Z_{2,3}$	*da Ziffer `2` nicht zur Sprache gehört*
`0000`	$\notin$	$Z_{2,3}$	*da das Wort Länge 4 statt 3 hat*

Im Folgenden möchten wir die **Anzahl solcher Worte** $N_{k,n} = |Z_{k,n}|$ mit Hilfe eines **dynamischen Programms** berechnen. Modellieren Sie hierfür zunächst geeignete Teilprobleme $T[\cdots]$, die sich effizient rekursiv berechnen lassen. Nennen Sie dann eine geeignete Rekursionsgleichung und minimal notwendige Basisfälle. Nennen Sie wie sich das Endergebnis $N_{k,n}$ aus Ihrer Rekursion ergibt.

Natürlichsprachliche Beschreibung der Teilprobleme $T[\cdots]$:

Lösungsskizze: Das Teilproblem $T[i, m]$ beschreibt die Anzahl legaler Suffixe, unter der Annahme, dass der Präfix $d_1 \cdots d_{i-1}$ bereits korrekt gesetzt wurde, und $d_i \geq m$ annehmen darf.

Rekursion: $T[\ i, m\ ] = \sum_{j=m}^{k-1} T[i+1, j]$

Rekursionsbasis: $T[n, m] = k - m$

Endergebnis: $N_{k,n} = T[1, 0]$

Aufgabe 9: Dynamische Programmierung — $4 + 6 + 4 = 14$ Punkt(e)

Bei der Modellierung eines dynamisches Programms ergab sich folgende Rekursion:

$$X[a,b] \quad = \quad \begin{cases} 1 & \text{falls } b = 0 \ \vee \ a = 0 \\ a \cdot X[a-1,b] + b \cdot X[a-1,b-1] & \text{falls } a > 0 \ \wedge \ b > 0 \end{cases}$$

a) Betrachten Sie die folgende naive Implementierung der Gleichung:

```
def f(a,b):
  if a == 0 or b == 0:
    return 1
  return a*f(a-1, b) + b*f(a-1,b-1)
```

Ermitteln Sie die Laufzeit dieses Algorithmus für den Fall $a = b$ asymptotisch exakt. Begründen Sie Ihre Antwort kurz.

Laufzeit: $\Theta(2^a)$

Lösungsskizze: In jedem Schritt werden zwei Teilprobleme benötigt, die jeweils rekursiv weitere Teilprobleme benötigen. Beobachte, dass aufgrund der Annahme $a = b$ alle Blätter im Rekursionsbaum auf derselben Tiefe liegen. Der Rekursionsbaum ist also ein vollständiger Binärbaum der Tiefe a.
Jeder Schritt verursacht dabei konstante Kosten. Es ergibt sich also folgende Rekursionsgleichung $T(a) = 2 \cdot T(a-1) + \Theta(1) = \Theta(2^a)$.

⇑⇑⇑ _____ / 4 Punkt(e) ⇑⇑⇑

b) Beschreiben Sie einen **iterativen** Algorithmus in **Pseudo-Code**, der für die Eingabe $a, b \in \mathbb{N}$ den Wert $X[a,b]$ in Zeit $\mathcal{O}(a \cdot b)$ berechnet. Nutzen Sie `return` um das Endergebnis $X[a,b]$ zurückzugeben.

Lösungsskizze:

- If $b == 0$ or $a == 0$: return 1
- Allokiere Matrix $X[1..a, 1..b]$
- For i in $1 \ldots a$:
  For j in $1 \ldots b$:
  $A \leftarrow 1$ if $i = 1$ else $X[i-1, j]$
  $B \leftarrow 1$ if $i = 1 \vee j = 1$ else $X[i-1, j-1]$
  $X[i,j] \leftarrow i \cdot A + j \cdot B$
- Return $X[a,b]$

`return` $X[a,b]$

⇑⇑⇑ _____ / 6 Punkt(e) ⇑⇑⇑

c) Beschreiben Sie einen Implementierungsansatz **natürlichsprachlich**, der den Speicherverbrauch auf $\mathcal{O}(b)$ Speicherzellen reduziert und in Zeit $\mathcal{O}(a \cdot b)$ arbeitet. **Begründen Sie kurz die Korrektheit** Ihrer Antwort.

Hinweis: Sollte Ihre Lösung in b) nur $\mathcal{O}(b)$ Speicherzellen benötigen, zeigen Sie dies.

Lösungsskizze: In der Rekursion wird nur die jeweils vorherige Zeile benötigt. Somit müssen nur die jeweils vorherige und die aktuelle Zeile im Speicher gehalten werden; die zwei Arrays können mit jeder neuen Zeile ihre Rolle wechseln. Jede Zeile besteht aus $\mathcal{O}(b)$ Einträgen.

⇑⇑⇑ _____ / 4 Punkt(e) ⇑⇑⇑

Wichtig: Lösungen auf dieser Seite werden nur dann berücksichtigt, wenn bei der entsprechenden Aufgabe ein Hinweis auf Seite 16 platziert wurde.

2012000100015