Karlsruher Institut für Technologie

Institut für Visualisierung und Datenanalyse
Lehrstuhl für Computergrafik
Prof. Dr.-Ing. Carsten Dachsbacher

Hauptklausur Algorithmen I
SS 2021

02. September 2021

Name:	
Matrikelnummer:	

Beachten Sie:

- Schreiben Sie Ihren vollständigen **Namen** und **Matrikelnummer** in Druckschrift in das Feld auf dem Deckblatt!
- Schreiben Sie Ihre Matrikelnummer oben auf jedes bearbeitete Aufgabenblatt.
- Schreiben Sie Ihre Lösungen auf die Aufgabenblätter. Bei Bedarf können Sie weiteres Papier anfordern.
- Halten Sie sich bei Pseudocode-Aufgaben an die Richtlinien auf der nächsten Seite.
- Wir akzeptieren auch englische Antworten.
- Sie dürfen **ein handbeschriebenes Din A4-Blatt** mitbringen, sonst sind **keine weiteren Hilfsmittel** zugelassen.
- Sie haben **120 Minuten** Bearbeitungszeit.
- Die Klausur umfasst 30 Seiten (14 Blätter) mit 10 Aufgaben.

Aufgabe	1	2	3	4	5	6	7	8	9	10	**Gesamt**
Erreichte Punkte											
Erreichbare Punkte	10	31	16	24	23	45	30	14	16	31	**240**

Note

Pseudocode-Richtlinien

Verwenden Sie in der Klausur folgende Konventionen für alle Aufgaben, bei denen Sie Pseudocode verwenden! In den Fällen, die hiervon nicht abgedeckt sind, halten Sie sich an die Notation aus der Übung.

`for`-Schleifen: Indizes iterieren

Verwenden Sie eine der folgenden Schreibweisen für `for`-Schleifenköpfe! In allen Beispielen nimmt `i` alle ganzzahligen Werte von 0 bis inklusive 99 an, aber nicht 100:

```
for i ← 0; i < 100; i++
for i in [0, 99]                for i ∈ [0, 99]
for i in [0,...,99]             for i ∈ [0,...,99]
for i from 0 to 99              for i ← 0 to 99
```

`foreach`-Schleifen: Datenstrukturen iterieren

Über Elemente einer Datenstruktur können Sie in undefinierter Reihenfolge wie folgt iterieren:

```
foreach e in E
foreach (u,v) in E
foreach e = (u,v) in E
```

Array-Indizierung

Arrays und Felder werden mit 0 indiziert. Das heißt, für ein Array `a: [int; 5]` mit 5 Einträgen wird mit `a[0]` auf das erste Element von `a` zugegriffen und mit `a[4]` auf das Letzte.

2D-Arrays

Ein $M \times N$-Array `a` (z.B. zur Repräsentation einer $M \times N$-Matrix $A = (a_{ij})$) mit M Zeilen und N Spalten und Einträgen vom Typ `typ` wird folgendermaßen deklariert:

```
a: [[typ; N]; M]
```

Das heißt, jede Zeile ist ein Feld `[typ; N]`.
Die Zeilen werden nacheinander in `a` abgelegt.

Auf den Eintrag in der i-ten Zeile und j-ten Spalte (bzw. den Matrixeintrag a_{ij}) wird zugegriffen mit

```
a[i][j]
```

Name: MUSTERLÖSUNG Matrikelnummer: _____

Aufgabe 1: Laufzeit (10 Punkte)

a) Zeigen Sie, dass $\mathcal{O}(n^2) \subseteq \mathcal{O}(n^3)$ gilt! **(6 Punkte)**

Musterlösung

Sei $f(n) \in \mathcal{O}(n^2)$, dann ex. C, n_0 sodass $\forall n \geq n_0 : f(n) \leq Cn^2$

Da $\lim n^2/n^3 = \lim 1/n \to 0$, ex. n_1 sodass $\forall n \geq n_1 : n^2 \leq n^3$
Folglich gilt auch $\forall n \geq \max\{n_0, n_1\} : f(n) \leq Cn^3$. Damit ist $f(n) \in \mathcal{O}(n^3)$.

b) Zeigen Sie, dass $\sum_{i=1}^{n} i \in \mathcal{O}(n^2)$ gilt! **(4 Punkte)**

Musterlösung

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2} = \frac{n^2+n}{2} \in \mathcal{O}(n^2) \tag{1}$$

Aufgabe 2: Algorithmenanalyse (31 Punkte)

In dieser Aufgabe sollen Sie die Laufzeit eines Algorithmus analysieren. Der Algorithmus überprüft, ob eine gegebene Texteingabe `text` $\in [a, \ldots, z]^*$ zu der gegebenen Schablone `pattern` $\in [a, \ldots, z, *]^*$ passt. In der Schablone stellt der Wildcard-Buchstabe '$*$' beliebig viele oder keinen Textbuchstaben in `text` dar. Die Tabelle zeigt einige Beispiele:

Pattern / Text	ab	aab	axb	aaxazbbb	ac
ab	✓	·	·	·	·
a∗b	✓	✓	✓	✓	·
aa∗z∗b	·	·	·	✓	·

```
Function match(text: string, textpos: int, pattern: string,
patternpos: int) : bool {
  endOfPattern : bool ← patternpos == length(pattern)
  endOfText : bool ← textpos == length(text)

  if endOfPattern then
|    return endOfText

   else if pattern[patternpos] == '*' then
|     if not endOfText and
|     match(text, textpos+1, pattern, patternpos) then
|  |    return True
|     return match(text, textpos, pattern, patternpos+1)
   else if not endOfText and
        text[textpos] == pattern[patternpos]) then
|    return match(text, textpos+1, pattern, patternpos+1)

   return False
}
```

Beispielaufruf: `match("aab", 0, "a*b", 0)`

a) Stellen Sie eine Rekurrenzformel für die asymptotische Laufzeit von `match` auf! Die Eingabegröße für die Rekurrenz sei $n = N + M$, wobei noch N Buchstaben im Eingabetext und M Buchstaben im Eingabe-Pattern verbleiben:
$N =$ `length(text)-textpos`,
$M =$ `length(pattern)-patternpos`.
Es genügt, wenn Sie sich bei Zweigen in Abhängigkeit der Eingabebuchstaben auf die maximal mögliche Laufzeit beschränken. **(6 Punkte)**

Hinweis: Beachten Sie, dass der Algorithmus bzgl. n in jeder Rekursionsebene Fortschritte macht.

Musterlösung

$$T(N+M) = 2T(N+M-1) \quad (2)$$
$$T(0) = C \quad (3)$$

b) Ist das Master-Theorem anwendbar? Begründen Sie! **(4 Punkte)**

Name: MUSTERLÖSUNG Matrikelnummer: _____

Musterlösung

Nein , da kein konstanter Reduktionsfaktor in jeder Rekursionsebene

c) Bestimmen Sie eine möglichst enge obere Schranke in $\mathcal{O}$-Notation für die Rekurrenz aus a) und begründen Sie Ihre Antwort mit dem Master-Theorem oder vollständiger Induktion, je nach Anwendbarkeit! **(7 Punkte)**

Musterlösung

Laufzeit: $\mathcal{O}(2^{N+M})$ (Punkte: 1)

Zeige $T(N+M) \leq 2^{N+M}$: (Punkte: 2)
I.A.: $T(0) = 1 = 2^0$. (Punkte: 2)
I.S.: $T(N+M+1) = 2T(N+M) = 2 \cdot 2^{N+M} = 2^{N+M+1}$ (Punkte: 2).

Falls Rekkurenz aus a) falsch und damit einfacherer aber korrekter Beweis (z.B. mit Master-Theorem): -4 Punkte

d) Nehmen Sie an, `pattern` enthält mindestens ein '$*$' und `match` verarbeitet zum ersten Mal den letzten '$*$'-Buchstaben in `pattern`.

Dann setzt sich die Rekurrenz T_R für die Restlaufzeit aus zwei Komponenten T_T und T_P zusammen:

$T_R(N+M) = T_T(N,M) + T_P(N,M)$.

- $T_T(N,M)$ ist die Restlaufzeit zur Verarbeitung des nächsten Text-Buchstabens
- $T_P(N,M)$ ist die Restlaufzeit zur Verarbeitung des nächsten Pattern-Buchstabens
- N ist die Anzahl verbleibender Text-Buchstaben
- M ist die Anzahl verbleibender Pattern-Buchstaben

☐ Geben Sie die Rekurrenzen T_T und T_P an! **(10 Punkte)**

Musterlösung

$$T_T(N,M) = T_T(N-1,M-1) \tag{4}$$
$$T_T(\cdot,0) = 1 \tag{5}$$
$$T_T(0,\cdot) = 1 \tag{6}$$

$$T_P(N,M) = T_P(N-1,M) + T_T(N,M-1) \tag{7}$$
$$T_P(0,M) = T_T(0,M-1) \tag{8}$$

☐ e) Geben Sie für $T_T(N,M)$ und $T_P(N,M)$ möglichst enge obere Schranken in $\mathcal{O}$-Notation abhängig von $n = N + M$ an! **(4 Punkte)**

Musterlösung

$$T_T \in O(n) \tag{9}$$
$$T_P \in O(n^2) \tag{10}$$

Name: MUSTERLÖSUNG Matrikelnummer: _____

Aufgabe 3: Hashing (16 Punkte)

a) Wann ist $\mathcal{H} \subseteq \{0 \dots m-1\}^{key}$ eine Familie universeller Hashfunktionen? **(4 Punkte)**

Musterlösung

Falls für alle x, y in key mit $x \neq y$ und **zufälligem** $h \in \mathcal{H}$ gilt:
$\mathbb{P}(h(x) = h(y)) = 1/m$
Punkte: Wichtig dass für alle x,y (nicht nur zufällige), quantoren nicht verdreht

b) Sie erhalten einen Datensatz `D` bestehend aus n Paaren (i, d), die jeweils eine durchgeführte Corona-Impfung dokumentieren. Dabei ist $i \in \mathbb{N}$ eine eindeutige ID der Person, welche eine Impfung bekommen hat, und $d \in \{1, \dots, 366\}$ der Tag der Impfung. Beachten Sie, dass pro Person bis zu zwei Impfungen an unterschiedlichen Tagen durchgeführt werden. Für jede Impfung gibt es genau ein Paar in `D`.

- Geben Sie einen Algorithmus in Pseudocode an, der in erwarteter Zeit $\mathcal{O}(n)$ und mit Speicherverbrauch $\mathcal{O}(n)$ die IDs aller Personen mit `print(id)` ausgibt, die bereits zwei Mal geimpft wurden!
- Begründen Sie kurz, warum Ihr Algorithmus das gewünschte Laufzeitverhalten aufweist!

(12 Punkte)

```
Function printVaccinated(D: [(int, int); n]) {
}
```

Musterlösung

Einfache Lösung mit einer Schleife:
$n \leftarrow |D|$ // $\mathcal{O}(n)$
Initialisiere Hashtabelle $H : N \to N \times N$ mit $k \cdot n$ Slots mit $k \geq 1$ // $\mathcal{O}(n)$
oder Initialisiere Hashtabelle $H : N \to N \times N$ mit $\Theta(n)$ Slots *und verketteten Listen zur Kollisionsauflösung* // $\mathcal{O}(n)$
for $(i, d) \in D$ **do**
  if $i \notin H$ **then**
    H.insert $(i, 1)$
  else
    `print(`i`)`
  end
end
// $\mathcal{O}(n)$

Längere Lösung mit zwei Schleifen:
$n \leftarrow |D|$ // $\mathcal{O}(n)$
Initialisiere Hashtabelle $H : N \to N \times N$ mit $k \cdot n$ Slots mit $k \geq 1$ // $\mathcal{O}(n)$
oder Initialisiere Hashtabelle $H : N \to N \times N$ mit $\Theta(n)$ Slots *und verketteten Listen zur Kollisionsauflösung* // $\mathcal{O}(n)$
for $(i, d) \in D$ **do**
  if $i \notin H$ **then**
    H.insert $(i, 1)$
  else
    $H[i] \leftarrow H[i] + 1$
  end
end
// $\mathcal{O}(n)$
for $(i, d) \in H$ **do**
  if $H[i] = 2$ **then**
    `print(`i`)`
  end
end

Name: MUSTERLÖSUNG Matrikelnummer: _____

Zusätzlicher Platz:

Aufgabe 4: Sortieren (24 Punkte)

a) Geben Sie für die gegebenen Sortieralgorithmen in $\mathcal{O}$-Notation abhängig von der Anzahl n zu sortierender Elemente an:

i) die asymptotische Worst Case-Laufzeit! **(3 Punkte)**

2-Wege-Quicksort	Mergesort	Insertionsort

Musterlösung

Mergesort ($\mathcal{O}(n \log n)$), 2-Wege-Quicksort ($\mathcal{O}(n^2)$),
Halbrekursiver 2-Wege-Quicksort ($\mathcal{O}(n^2)$), Insertionsort ($\mathcal{O}(n^2)$)

ii) den zusätzlichen Speicherbedarf (inklusive Rekursionsstack) im Worst Case! **(3 Punkte)**

2-Wege-Quicksort	Mergesort	Insertionsort

Musterlösung

Insertionsort ($\mathcal{O}(1)$), Halbrekursiver 2-Wege-Quicksort $\mathcal{O}(\log n)$,
2-Wege-Quicksort ($\mathcal{O}(n)$, Stack), Mergesort ($\mathcal{O}(n)$), sekundäres Array)

b) Gegeben sei der folgende Algorithmus *stableSort*, der Arrays von Elementen vom Typ `T` stabil sortieren soll. Intern verwendet *stableSort* dabei einen nicht stabilen Sortieralgorithmus *unstableSort*. In dieser Aufgabe sollen Sie die Eingabe für *unstableSort* in einen neuen Datentyp mit neuer Ordnungsrelation transformieren, sodass das Ergebnis von *unstableSort* in eine stabile Sortierung der ursprünglichen Eingabe zurücktransformiert werden kann.

```
Function stableSort(array: [T; n]) {
    transformed: [U; n] ← transform(array)
    unstableSort(transformed)
    array ← reverseTransform(transformed)
}
```

i) Definieren Sie den Typ `U` der Elemente des transformierten Arrays! **(4 Punkte)**

Musterlösung

$\texttt{U} := (\texttt{T}, \mathbb{N}_0)$

ii) Definieren Sie eine passende Ordnungsrelation $<_{\texttt{U}}$ auf `U`, welche für die anschließende Sortierung verwendet wird! Sie können dabei die auf `T` definierte Ordnungsrelation $<_{\texttt{T}}$ verwenden. **(6 Punkte)**

Name: MUSTERLÖSUNG Matrikelnummer: _____

Musterlösung

$(e_a, r_a) := a; (e_b, r_b) := b$
Mögliche Lösungen:
$e_a <_\mathrm{T} e_b \vee (\neg (e_a <_\mathrm{T} e_b \vee e_b <_\mathrm{T} e_a) \wedge i_a < i_b)$
$e_a <_\mathrm{T} e_b \vee (e_a =_\mathrm{T} e_b \wedge i_a < i_b)$

$(a, x) <_U (b, y) :\Leftrightarrow (a <_T b) \vee (a == b \wedge x < y)$

iii) Führen Sie die Transformation der Eingabe in der Funktion *transform* so in Pseudocode durch, dass *stableSort* dann eine stabile Sortierung der Eingabe durchführt! Die Transformation muss dabei reversibel und die Laufzeit in $\mathcal{O}(n)$ sein. **(8 Punkte)**

Musterlösung

```
Function transform(array: [T; n]) : [U; n]
{
newArray : [U;ℕ0]
for int i ← 0; i < n; i++ do
 | newArray[i] ← (array[i],i)
end
return newArray
}
```

Laufzeit und Speicherbedarf für *transform* sind offensichtlich in $\mathcal{O}(n)$.

Aufgabe 5: Prioritätslisten (23 Punkte)

a) Sei v ein Knoten in einem binären Min-Heap mit dem linken Kind l und dem rechten Kind r. Was gilt aufgrund der Heap-Eigenschaft für diese Knoten? **(2 Punkte)**

Musterlösung

$v \leq l$ und $v \leq r$

b) Führen Sie die Binär-Heap-Konstruktion `buildHeapBackwards` aus der Vorlesung auf dem gegebenen 1-indizierten Array aus! Geben Sie dazu nach jedem Aufruf von `siftDown(i)` aus `buildHeapBackwards`, welcher das Array ändert, den neuen Zustand des Arrays sowie den Wert des Parameters `i` an! **(9 Punkte)**

Index	**Array**								
i	50	30	20	70	90	10	80	60	40

Musterlösung

Index	50	30	20	70	90	10	80	60	40
4				40					70
3			10			20			
1	10		20			50			

Lösung für 0-indiziert: 2 Punkte Abzug

Index	50	30	20	70	90	10	80	60	40
3				40					70
2			10			20			
0	10		20			50			

c) Die Operation `delete` soll das Element `h[i]` so aus einem binären Min-Heap `heap` mit n Elementen vom Typ `float` löschen, dass `heap` nach Ausführung wieder ein

Name: MUSTERLÖSUNG Matrikelnummer: _____

gültiger Min-Heap ist! Implementieren Sie die Funktion `delete` in Pseudocode mit Laufzeit $\mathcal{O}(\log n)$! Sie dürfen die angegebenen Funktionen nutzen. **(12 Punkte)**

```
Struct Heap {
  h: [float; N_MAX],
  n: int,
}
```

```
Function parent(i: int) : int
Function leftChild(i: int) : int
Function rightChild(i: int) : int
```

```
Function insert(heap: Heap, e: float)
Function deleteMin(heap: Heap) : float
Function siftDown(heap: Heap, i: int)
Function siftUp(heap: Heap, i: int)
```

```
Function delete(heap: Heap, i: int) {
   assert 0 ≤ i <heap.n
}
```

Musterlösung

Einfache Variante:

```
Function delete(heap: Heap, i: int) {
    heap.h[i] ← heap.h[heap.n - 1]
    heap.n ← heap.n - 1
    siftDown(heap, i) siftUp(heap, i)
}
```

Variante mit IF-Bedingung:

```
Function delete(heap: Heap, i: int) {
    heap.h[i] ← heap.h[heap.n - 1]
    heap.n ← heap.n - 1
    if i == 0 || heap.h[parent(i)] ≤ heap.h[i] then
 |     siftDown(heap, i)
    else
 |     siftUp(heap, i)
    end
}
```

Aufgabe 6: Graphen (45 Punkte)

a) Wie können Sie anhand einer Adjazenzmatrix erkennen, ob der zugehörige Graph schleifenfrei ist? **(2 Punkte)**

Musterlösung

Die Diagonale ist überall 0

b) Bei der Breitensuche oder Tiefensuche in einem Graphen ergibt sich ein Suchbaum. Die Kanten des Graphen können bezüglich eines solchen Suchbaums in 4 Klassen klassifiziert werden: Baumkanten, Cross-Kanten, und zwei Weitere. Nennen und beschreiben Sie die beiden weiteren Klassen! **(6 Punkte)**

1.

2.

Musterlösung

- Vorwärtskanten: Verlaufen parallel zu Wegen aus Baumkanten
- Rückwärtskanten: Verlaufen antiparallel zu Wegen aus Baumkanten

c) Wie können Sie mithilfe der Tiefensuche bestimmen, ob ein gerichteter Graph kreisfrei ist? **(2 Punkte)**

Musterlösung

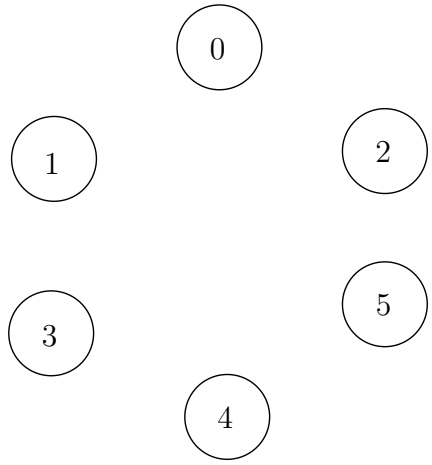
Die Tiefensuche findet keine Rückwärtskante bei kreisfreien Graphen.

Name: MUSTERLÖSUNG Matrikelnummer: _____

d) Die folgende Adjazenzmatrix M beschreibt einen gerichteten Graphen $G = (V, E)$ mit $V = \{0, 1, 2, 3, 4, 5\}$:

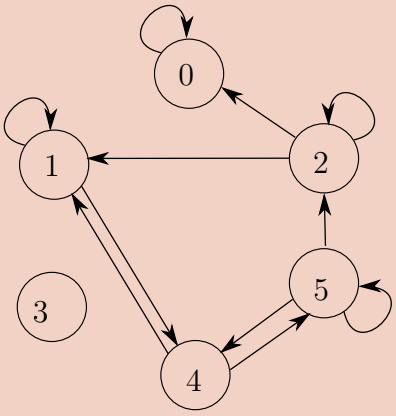
$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}$$

i) Zeichnen Sie den Graphen! **(4 Punkte)**



Musterlösung

Solution:



ii) Stellen Sie den Graphen durch die Adjazenzfeld-Repräsentation dar, indem Sie die angegebene Tabelle ausfüllen! **(8 Punkte)**

V:													
E:													

Zusätzlicher Platz zur Fehlerkorrektur. Kennzeichnen Sie eindeutig, welche Lösung gewertet werden soll!

V:													

E:													

Musterlösung

V:	0	1	3	6	6	8	11						

E:	0	1	4	0	1	2	1	5	2	4	5		

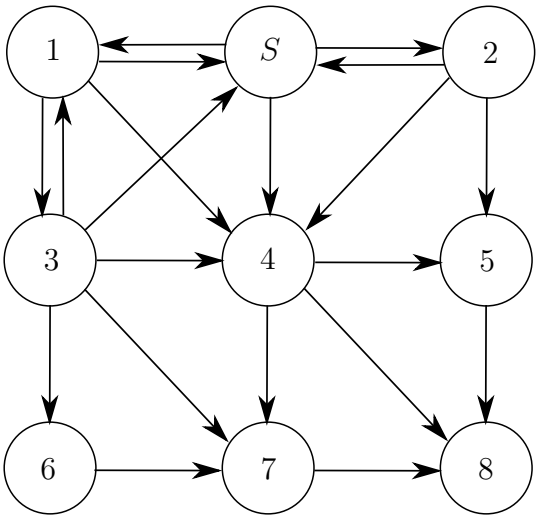
Name: MUSTERLÖSUNG Matrikelnummer: _____

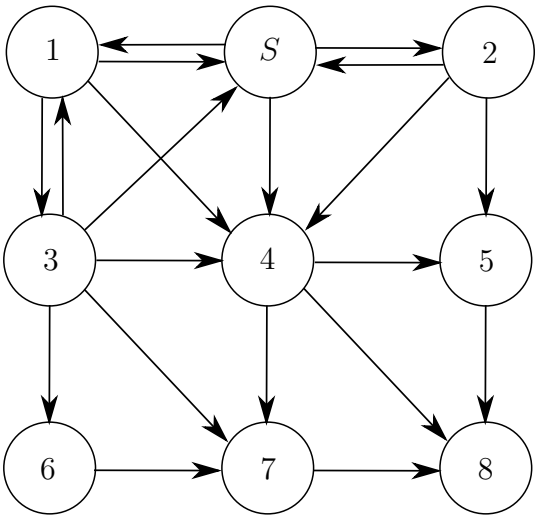
e) Führen Sie auf dem folgenden Graphen eine Tiefensuche vom Startknoten S aus durch! Falls mehrere Knoten als Nächstes traversiert werden können, wählen Sie den Knoten mit geringstem Index!

- Markieren Sie alle Baumkanten der Tiefensuche mit b und alle Cross-Kanten mit x!
- Welche Kante wird den Baumkanten als Letztes hinzugefügt?

Falls Sie den zweiten Graphen zur Fehlerkorrektur nutzen, kennzeichnen Sie eindeutig, welche Lösung gewertet werden soll! **(8 Punkte)**

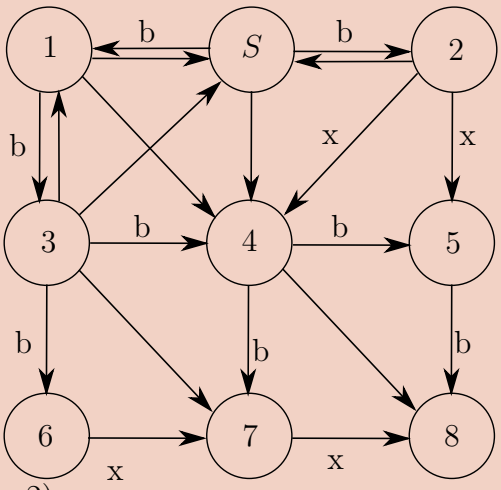
Letzte Baumkante:





Musterlösung

Musterlösung:



Letzte Kante: $(s, 2)$.

f) Gegeben sei ein gerichteter Graph $G = (V, E)$ mit $V = \{0, ..., N-1\}$ und $|E| = k$. Die Funktion `matrixToField` soll als Eingabe die Adjazenzmatrix `M` von G sowie ein leeres Adjazenzfeld bestehend aus `V` und `E` erhalten, und das Adjazenzfeld möglichst laufzeiteffizient mit der Repräsentation für den Graphen G ausfüllen.

- Implementieren Sie die Funktion in Pseudocode!
- Bestimmen und begründen Sie möglichst enge Laufzeitschranken für Ihren Algorithmus in $\mathcal{O}$-Notation!

☐ *Hinweis: Mit `M[i][j]` können Sie auf den Eintrag m_{ij} der Matrix M zugreifen.* **(15 Punkte)**

```
Function matrixToField(M: [[int; N]; N],
                       V: [int; N+1],
                       E: [int; k])
{
    V[0] ← 0
}
```

Musterlösung

```
e ← 0
for i = 0; i < N; i++ do
    V[i] ← e
    for j = 0; j < N; j++ do
        if M[i][j] == 1 then
            E[e] ← j
            e ← e+1
        end
    end
end
V[N] ← e
```

Name: MUSTERLÖSUNG Matrikelnummer: _____

Aufgabe 7: Kürzeste Wege (30 Punkte)

a) Gegeben sei die folgende Implementierung von Dijkstras Algorithmus für gerichtete Graphen $G = (V, E)$ mit Knoten $V = \{0, \ldots, N-1\}$, Startknoten `s` und Kantengewichtungsfunktion `w(e: Edge):float`.

```
Function Dijkstra(s: int) : ([float; |V|], [int; |V|])
parent ← [int; |V|]
parent[s] ← s
d← [float; |V|]
d ← {∞,...,∞}
d[s] ← 0
Q ← NodePQ
Q.insert(s, d[s])
while Q ≠ ∅ do
    u ← Q.deleteMin()
    for Edge e = (u,v) in E do
        if d[u] + w(e) < d[v] then
            d[v] ← d[u] + w(e)
            parent[v] ← u
            if v in Q then
                Q.decreaseKey(v, d[v])
            else
                Q.insert(v, d[v])
return (d, parent)
```

Wir betrachten die Operationen `deleteMin`, `decreaseKey` und `insert`.

i) Geben Sie in $\mathcal{O}$-Notation möglichst enge Schranken an für die Anzahl der Aufrufe der Operationen! Begründen Sie kurz Ihre Antworten! **(7 Punkte)**

deleteMin	decreaseKey	insert

Begründung:

Musterlösung

deleteMin	decreaseKey	insert
$\mathcal{O}(\|V\|)$	$\mathcal{O}(\|E\|)$ oder $\mathcal{O}(\|E\| - \|V\|)$	$\mathcal{O}(\|V\|)$

`insert` und `deleteMin`: Jeder Knoten wird höchstens einmal in Q eingefügt und höchstens einmal gelöscht.
`decreaseKey`: wird $|E| - |V|$ mal ausgeführt, bzw. pro Knoten für alle eingehenden Kanten bis auf eine

ii) Geben Sie in $\mathcal{O}$-Notation möglichst enge Schranken an für die Worst-Case-Laufzeit je einer Ausführung der Operation, falls `Q` als addressierbare Binärheap-Prioritätsliste implementiert ist! *(ohne Begründung)* **(3 Punkte)**

deleteMin	decreaseKey	insert

Musterlösung

alles in $\mathcal{O}(\log |V|)$

iii) Geben Sie in $\mathcal{O}$-Notation möglichst enge Schranken an für die Worst-Case-Laufzeit je einer Ausführung der Operation, falls `Q` basierend auf einer doppelt verketteten Liste implementiert ist! Begründen Sie Ihre Antworten! **(9 Punkte)**

deleteMin	decreaseKey	insert

Begründung:

Musterlösung

Alternative 1: `Q` wird sortiert gehalten.

- Insert $\mathcal{O}(|V|)$ - Liste durchlaufen um die Stelle zu suchen, an der der Knoten mit seiner aktuellen Priorität eingefügt werden soll, im Worst Case ist diese Stelle am Ende der Liste.
- decreaseKey: $\mathcal{O}(|V|)$ weil man zuerst in $\mathcal{O}(|V|)$ den Eintrag sucht, an dem der Knoten steht, und dann in $\mathcal{O}(|V|)$ die Stelle sucht, wo er hin soll, und ihn dann in $\mathcal{O}(1)$ an diese Stelle einfügt.
- deleteMin: $\mathcal{O}(1)$ - Ersten bzw. Letzen (je nach Sortierreihenfolge) Eintrag einer verketteten Liste zu löschen benötigt konstante Zeit.

Alternative 2: `Q` speichert die Knoten in beliebiger Reihenfolge.

- insert $\mathcal{O}(1)$ um Element vorne / hinten an die Liste anzuhängen
- decreaseKey $\mathcal{O}(|V|)$ weil wie oben in $\mathcal{O}(|V|)$ der Eintrag gesucht werden muss, da man keinen freien Zugriff in verketteten Listen hat.
- deleteMin $\mathcal{O}(|V|)$ weil zunächst das Minimum in $\mathcal{O}(|V|)$ gesucht werden muss, und dann in $\mathcal{O}(1)$ entfernt.

b) Zeichnen Sie einen gerichteten, gewichteten Graphen $G = (V, E)$ mit $V = \{A, B, C\}$, der keinen eindeutigen kürzesten Weg von Knoten A zu Knoten C hat! **(3 Punkte)**

Name: MUSTERLÖSUNG Matrikelnummer: _____

Musterlösung

z.B. $A \to C$ mit negativ gewichteter Kante (A, A), beliebige Kante nach Knoten B
oder: Ein Graph mit mehreren gleichgewichteten Wegen von A zu C

c) Betrachten Sie folgende Distanzmatrix eines gerichteten Graphen $G = (V, E)$ mit den Knoten $v_0, v_1, ..v_4 \in V$. Ein Eintrag a_{ij} mit $i, j \in [0; 4]$ in der Matrix gibt das Kantengewicht der Kante (v_i, v_j) an. Ein Wert von ∞ bedeutet, dass keine Kante von v_i nach v_j existiert:

$$A = \begin{pmatrix} 5 & \infty & \infty & 8 & \frac{1}{3} \\ 1 & \infty & 10 & \infty & 3 \\ \infty & 11 & \infty & \frac{2}{3} & 1 \\ \frac{8}{7} & \infty & \infty & -1 & 5 \\ \infty & \infty & \infty & -2 & \infty \end{pmatrix}$$

i) Welcher Algorithmus aus der Vorlesung ist geeignet, effizient kürzeste Wege zwischen beliebigen Knoten in diesem Graphen zu finden? Begründen Sie ihre Antwort! **(2 Punkte)**

Musterlösung

Bellman-Ford wegen negativen Kantengewichten

ii) Gibt es einen kürzesten Weg von Knoten v_0 zu Knoten v_2? Begründen Sie Ihre Antwort! **(6 Punkte)**

Musterlösung

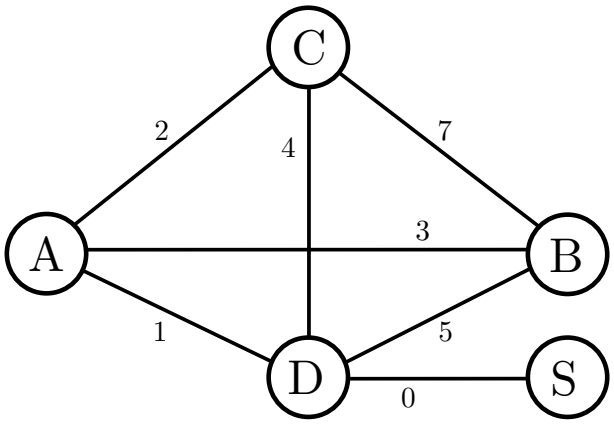
Nein. Da v_2 nur eingehende Kanten von v_1 hat und v_1 nur eingehende Kanten von v_2, ist v_1 von keinem anderen Knoten erreichbar.
Punkte:

- Keine Punkte ohne Begründung!!!!!!
- Zeichnung als Begründung okay

Name: MUSTERLÖSUNG Matrikelnummer: _____

Aufgabe 8: Minimale Spannbäume (MST) (14 Punkte)

a) Bestimmen Sie den minimalen Spannbaum des folgenden Graphen mit Hilfe des Jarník-Prim-Algorithmus! Setzen Sie die Tabelle fort, indem Sie pro Schritt die zum MST hinzugefügte Kante sowie den aktuellen Inhalt der Prioritätsliste in der Form (Knoten, Gewicht) angeben! **(6 Punkte)**



Musterlösung

	Hinzugefügte Kante	PQ $\{(X, 42), \dots\}$
1. Schritt	{S,D}	{(A,1), (C,4), (B,5)}
2. Schritt	{D,A}	{(C,2), (B,3)}
3. Schritt	{A,C}	{(B,3)}
4. Schritt	{A,B}	

	Hinzugefügte Kante	PQ $\{(X, 42), \dots\}$
1. Schritt	{S,D}	{(A,1), (C,4), (B,5)}
2. Schritt	{ , }	
3. Schritt	{ , }	
4. Schritt	{ , }	

Name: MUSTERLÖSUNG Matrikelnummer: _____

b) Erklären Sie die folgenden zwei Eigenschaften minimaler Spannbäume (MST)! **(4 Punkte)**

i) Schnitt-Eigenschaft:

Musterlösung

leichteste (Kante zwischen zwei Teilmengen / Schnittkante) kann in MST verwendet werden

ii) Kreis-Eigenschaft:

Musterlösung

schwerste Kante eines Kreises wird nicht für MST benötigt / nicht Teil des MST

c) Führen Sie auf der im Folgenden tabellarisch dargestellten Union-Find-Datenstruktur (ohne Pfadkompression und ohne Union by rank) die angegebenen Operationen aus! **(4 Punkte)**

Node	1	2	3	4	5	6
Parent	1	1	4	4	4	5

i) find(6) =

Musterlösung

4

ii) find(2) =

Musterlösung

1

iii) union(2, 6)

Musterlösung

Node	1	2	3	4	5	6
Parent	4 (1)	1	4	4 (1)	4	5

Node	1	2	3	4	5	6
Parent						

Name: MUSTERLÖSUNG Matrikelnummer: _____

Aufgabe 9: Suchbäume (16 Punkte)

a) Geben Sie die minimale und maximale Höhe eines binären Suchbaums mit n Elementen in Θ-Notation an! **(4 Punkte)**

minimale Höhe $h_{\min} \in$

maximale Höhe $h_{\max} \in$

Musterlösung

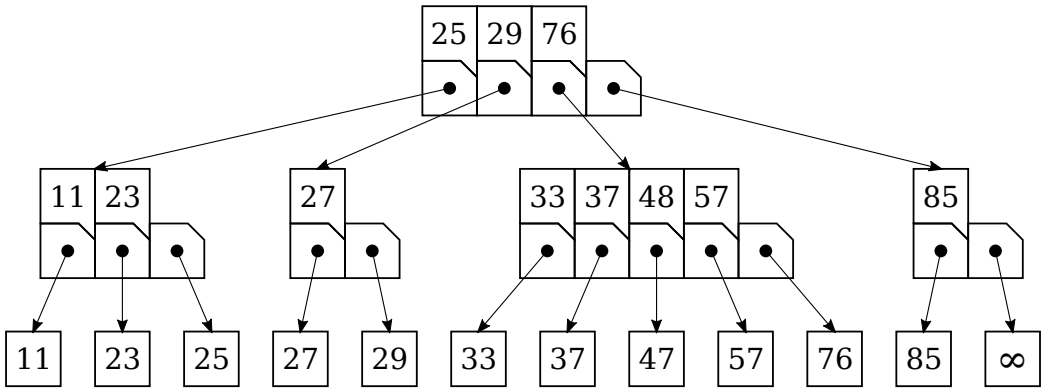
$h_{\min} \in \Theta(\log n)$, $h_{\max} \in \Theta(n)$

b) Ist die Konstruktion eines binären Suchbaums im Allgemeinen aus einer Sequenz von n Elementen in Laufzeit $\mathcal{O}(n)$ möglich? Begründen Sie! **(6 Punkte)**

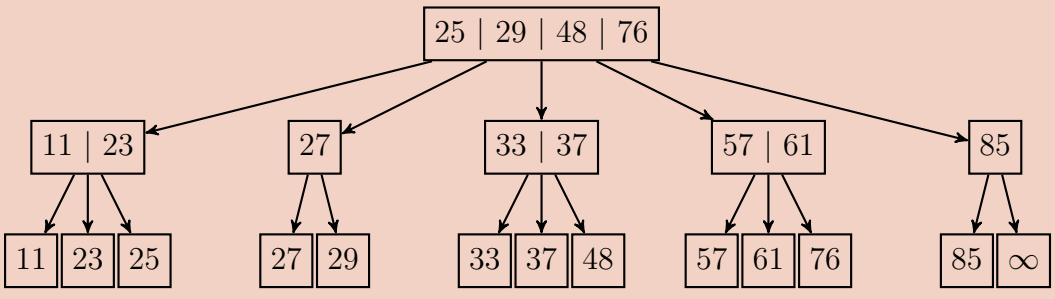
Musterlösung

Nicht möglich. Mit dem Suchbaum können die Elemente sortiert in $\mathcal{O}(n)$ ausgegeben werden , aber Sortieren ist im Allgemeinen in $\Omega(n \log n)$.

c) Führen Sie auf dem folgenden $(2, 5)$-Baum die Operation `insert(61)` aus und geben Sie den resultierenden Baum an! **(6 Punkte)**



Musterlösung



Aufgabe 10: Greedy-Münzwechselproblem (31 Punkte)

Beim Münzwechselproblem muss ein gegebener Betrag aus einer minimalen Anzahl Münzen verschiedener Wertigkeit zusammengestellt werden.

Gegeben seien n Münzwerte $c_0, \ldots, c_{n-1}$, wobei $\forall i \in \{0, \ldots, n-2\}$ gilt:

- $c_0 = 1$
- $c_i < c_{i+1}$
- $\frac{c_{i+1}}{c_i} \in \mathbb{N}_+$

Ein Vektor $\boldsymbol{v} \in \mathbb{N}^n$ beschreibt die Anzahlen v_i der gewählten Münzen und der Vektor $\boldsymbol{c} \in \mathbb{N}^n$ beschreibt die Wertigkeit der Münzen c_i. Er ist eine Lösung des Münzwechselproblems für den Betrag $A \in \mathbb{N}_+$, wenn gilt:

$$\boldsymbol{v} \cdot \boldsymbol{c} = \sum_{i=0}^{n-1} v_i \, c_i = A.$$

Eine Lösung $\boldsymbol{v}$ sei optimal, falls $\sum_{i=0}^{n-1} v_i$ minimal ist.

a) Geben Sie für den Betrag $A = 345$ eine optimale Lösung $\boldsymbol{v}$ an, wenn folgende Münzwerte zur Verfügung stehen: **(5 Punkte)**

c_0	c_1	c_2	c_3	c_4	c_5
1	5	10	50	100	200

$\boldsymbol{v} =$

Musterlösung

$\boldsymbol{v} = (0, 1, 4, 0, 1, 1)$

b) Zeigen Sie, dass für eine optimale Lösung $\boldsymbol{v}$ folgende Aussage gilt: **(10 Punkte)**

$$\forall i \in \{0, \ldots, n-2\} : v_i \leq \frac{c_{i+1}}{c_i} - 1$$

Hinweis: Sie können den Beweis zum Beispiel als Widerspruchsbeweis führen.

Name: MUSTERLÖSUNG Matrikelnummer: _____

Musterlösung

Mögliche Lösung: Beweis durch Widerspruch.
Annahme: Es gibt ein $v_i > \frac{c_{i+1}}{c_i} - 1$.
Wir finden eine bessere Lösung, indem wir $\frac{c_{i+1}}{c_i}$ Münzen vom Wert c_i ersetzen durch eine Münze vom Wert c_{i+1}.
Da $\frac{c_{i+1}}{c_i} > 1$ ist, verringern wir dadurch die Gesamtzahl.
Dabei ist auch wichtig zu zeigen, dass es bei so einem v_i auch tatsächlich genug Münzen vom Wert c_i gibt, die man entfernen kann.

Beweis
Sei i sodass $v_i > \frac{c_{i+1}}{c_i} - 1$.
Wir definieren $\boldsymbol{v}'$ sodass

$$v'_j = \begin{cases} v_j + 1 & j = i+1 \\ v_j - \frac{c_{i+1}}{c_i} & j = i \\ v_j & \text{sonst} \end{cases}$$

$\boldsymbol{v}'$ ist eine Lösung, weil:

1. Alle Einträge sind in $\mathbb{N}$:
   - für $j \neq i$ und $j \neq i+1$ ändert sich nichts
   - für $j = i+1$ ist $v'_j = v_j + 1 \in \mathbb{N}$
   - Da $v_i > \frac{c_{i+1}}{c_i} - 1$ ist $v'_i = v_i - \frac{c_{i+1}}{c_i} > -1$, und da der Bruch und v_i ganze Zahlen sind, ist damit $v'_i \geq 0$ und $v'_i \in \mathbb{N}$
2. Entsprechend viele Münzen ergeben den selben Betrag:

$$\boldsymbol{v'} \cdot \boldsymbol{c} = \boldsymbol{v} \cdot \boldsymbol{c} + c_{i+1} - \frac{c_{i+1}}{c_i} c_i = \boldsymbol{v} \cdot \boldsymbol{c}.$$

Außerdem ist $\boldsymbol{v}'$ eine bessere Lösung als $\boldsymbol{v}$, weil:

$$\sum_{j=0}^{n-1} v'_j = 1 - \frac{c_{i+1}}{c_i} + \sum_{j=0}^{n-1} v_j \leq \sum_{j=0}^{n-1} v_j.$$

Damit kann $\boldsymbol{v}$ keine optimale Lösung sein.

c) Diese Aufgabe soll zeigen, dass ein Greedy-Algorithmus, der stets die größtmögliche Münze wählt, immer die optimale Lösung findet. Dafür sollen Sie die folgenden Eigenschaften zeigen.

i) Zeigen Sie: Für jeden Betrag $A \in \mathbb{N}_+$ gibt es eine Lösung. **(2 Punkte)**

Musterlösung

Da $c_0 = 1$ ist $(|A|, 0, ..., 0)$ eine Lösung für jeden Betrag A.

ii) Sei k maximal sodass $A \geq c_k$. Sei $A' = A - c_k$ und $\boldsymbol{v}'$ eine optimale Lösung für A'.

Sei $\boldsymbol{v}$ mit $v_k = v'_k + 1$ und $v_{j \neq k} = v'_j$.

Zeigen Sie: $\boldsymbol{v}$ ist eine optimale Lösung für A. **(14 Punkte)**

Hinweise:

- Sie können den folgenden Satz verwenden: Sei $\boldsymbol{v}$ eine optimale Lösung für A und sei k maximal sodass $A \geq c_k$. Dann gilt $v_k > 0$.
- Zeigen Sie erst, dass $\boldsymbol{v}$ eine Lösung für A ist, und dann, dass es eine optimale Lösung für A ist, z.B. durch Widerspruch.

Musterlösung

1. Zeige $\boldsymbol{v}$ ist überhaupt eine Lösung:

$$\sum_{i=0}^{n-1} v_i c_i = \sum_{i \neq k} v'_i c_i + (v'_k + 1)c_k = \sum_{i=0}^{n-1} v'_i c_i + c_k = A' + c_k = A$$

Musterlösung

2. Zeige: $\boldsymbol{v}$ ist eine optimale Lösung:
Annahme: $\boldsymbol{v}$ ist keine optimale Lösung für A.
Dh die $\sum v_i$ ist nicht minimal.
Sei $\boldsymbol{w}$ eine optimale Lösung für A.
also $\sum w_i < \sum v_i$ [1]
Wir wissen aus Hinweis 1: $w_k > 0$.
Damit ist $\boldsymbol{w'}$ mit $w'_k = w_k - 1, w'_{j \neq k} = w_j$ eine Lösung für A'.
Es gilt $\sum w'_i = \sum w_i - 1 < \sum v_i - 1 = \sum v'_i$ wegen [1]
Damit ist v' keine optimale Lösung.
Widerspruch.