Prüfung aus

Diskrete und geometrische Algorithmen (Hetzl)

Zuname: ______________________________ Matrikelnummer: ____________________

Vorname: ______________________________

Titelseite ab hier bitte freilassen!

TU Wien, 29.1.2019 Arbeitszeit: 100 Minuten

1)

2)

3)

4)

1) (7 P.) Lösen Sie die Rekursionsgleichung $a_n = 5a_{n-1} - 6a_{n-2}$ (für $n \geq 2$) mit den Anfangswerten $a_0 = 1$ und $a_1 = 5$.

$z^2 - 5z + 6 = 0 \Leftrightarrow z = \frac{5 \pm \sqrt{25-24}}{2} \Leftrightarrow z = 3 \vee z = 2$

also $a_n = c_1 3^n + c_2 2^n$

$\left.\begin{array}{l} 1 = a_0 = c_1 + c_2 \Leftrightarrow c_1 = 1 - c_2 \\ 5 = a_1 = 3c_1 + 2c_2 \end{array}\right\} \Rightarrow 5 = 3(1-c_2) + 2c_2 = 3 - c_2 \Leftrightarrow c_2 = -2$

$\Rightarrow c_1 = 1 - c_2 = 1 + 2 = 3$

$a_n = 3^{n+1} - 2^{n+1} = 5 \cdot 3^n - 5 \cdot 2^n - 2 \cdot 3^n + 3 \cdot 2^n = 5(3^n - 2^n) - 6(3^{n-1} - 2^{n-1}) = 5a_{n-1} - 6a_{n-2}$

2) (7 P.) Wir definieren die Menge von Zeichen

$$A = \{a, b, c\} \cup \{\ (_i \mid i \in \mathbb{N}\} \cup \{\ )_i \mid i \in \mathbb{N}\}$$

wobei wir sowohl eine indizierte öffnende Klammer $(_i$ als auch eine indizierte schließende Klammer $)_i$ als jeweils ein einzelnes Zeichen betrachten. Sei A^* die Menge aller endlich langen Zeichenketten die nur aus Zeichen aus A bestehen. Die Menge der *indiziert wohlgeklammerten Ausdrücke* ist die kleinste Menge $W \subseteq A^*$ für die gilt:

1. Die leere Zeichenkette $\varepsilon \in W$.
2. Die Zeichen $a, b, c \in W$.
3. Falls $w_1, w_2 \in W$, dann ist auch $w_1 w_2 \in W$.
4. Falls $w \in W$ und $i \in \mathbb{N}$, dann ist auch $(_i w)_i \in W$.

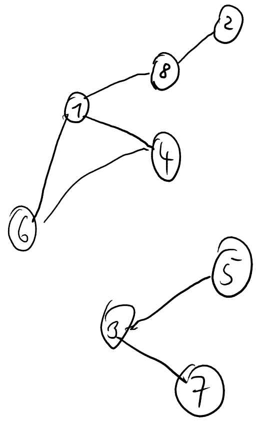
So ist zum Beispiel $(_3c)_3ab(_3ab(_1c)_1)_3 \in W$ aber $bc(_1b(_2a)_1cc)_2 \notin W$.

Geben Sie einen Algorithmus in Pseudocode an, der eine Zeichenkette $v \in A^*$ als Eingabe erhält und entscheidet, ob $v \in W$ ist. Die Eingabe wird in Form eines Datenfelds übergeben. Die Laufzeit des Algorithmus muss $O(|v|)$ sein wobei $|v|$ die Anzahl der Zeichen in v ist.

```
Prozedur InW(v)
    If v.Länge = 0
        return 1                          ▷ ε ∈ W
    End If
    If v[0] ∈ {a, b, c}
        return (InW(v[1,..., v.Länge-1]))
    else if v[0] und v[v.Länge-1] sind passende Klammern
        return (InW(v[1,..., v.Länge-2]))
    else
        return 0                          ▷ v ∉ W, falsch geklammert
    End If
```

Die Laufzeit des Programms ist in $\mathcal{O}(|v|)$, da bei jedem rekursiven Aufruf das Wort mindestens um ein Zeichen kürzer wird und sonst keine weiteren Kosten anfallen.

3) (7 P.) Sei $G = (V, E)$ ein nicht-leerer endlicher ungerichteter Graph, dann definieren wir $m(G) = \max\{|V_i| \mid 1 \leq i \leq k\}$ wobei $(V_1, E_1), \ldots, (V_k, E_k)$ die Zusammenhangskomponenten von G sind. Geben Sie einen Algorithmus in Pseudocode an, der einen nicht-leeren endlichen ungerichteten Graph $G = (V, E)$ als Eingabe erhält und $m(G)$ als Ausgabe liefert. Der Eingabegraph wird in Form einer Adjazenzliste übergeben wobei $V = \{1, \ldots, n\}$. Die Laufzeit des Algorithmus muss $O(|V| + |E|)$ sein.




```
Prozedur ZSKP(G]
    Sei Z neues Datenfeld der Länge |V| überall mit 0 initialisiert
    Sei Q Warteschlange
    l:=0
    While SucheNull(Z)<n+1
        i:= SucheNull(Z)
        l:=l+1
        Z[i]:=l
        Hinzufügen(Q, i)
        While Q≠∅
            k:= Entfernen(Q)
            for alle {k,j} ∈ E
                if Z[j] = 0
                    Z[j]:= Z[k]
                    Hinzufügen(Q,j)
                End if
            End for
        End While
        return FINDEMAX(Z,l)
```

```
SucheNull(Z)
    i:=1
    while i<n+1 ∧ Z(i) ≠ 0
        i:= i+1
    end while
    return i
```

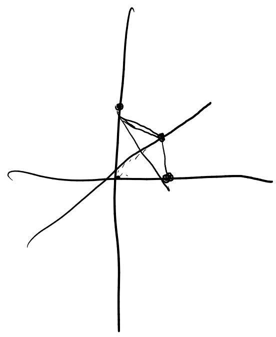
```
FINDEMAX(Z,l)
    Sei M neues Datenfeld der Länge l mit 0 initialisiert
    for i=1,..., Z.Länge
        M[Z[i]] := M[Z[i]]+1
    End for
    m:=0
    for j=1,...,l
        if M[j] > m
            m:=M[j]
        End if
    End for
    return m
```

4) (3 P.) Geben Sie ein lineares Programm in Standardform an, das drei optimale Lösungen besitzt die nicht auf einer Gerade liegen.

$A := (1,1,1)$

$c := (1,1,1)$

$b := 1$



$Ax \leq b$

cx maximal

$\forall i \in \{1,2,3\}: x_i \geq 0$

e_1, e_2, e_3 sind optimale Lsg. die nicht auf einer Geraden liegen

Prüfung aus

Diskrete und geometrische Algorithmen (Hetzl)

Zuname: ______________________________ Matrikelnummer: ____________________

Vorname: ______________________________

Titelseite ab hier bitte freilassen!

TU Wien, 1.3.2019 Arbeitszeit: 100 Minuten

1)

2)

3)

4)

1) (4 P.) Geben Sie asymptotische Lösungen der folgenden Rekursionsgleichungen an:

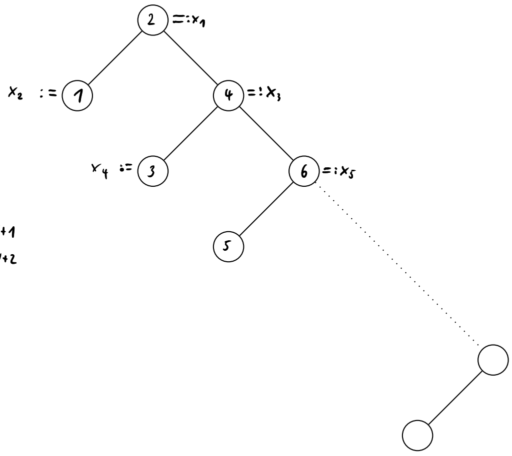
a) $T(n) = 9T(\frac{n}{3}) + n(n+1)$

b) $T(n) = 3T(\frac{n}{4}) + n \log n$

a) $n(n+1) \in \mathcal{O}(n^{\log_3(9)-\varepsilon}) = \mathcal{O}(n^{3-\varepsilon})$ für $\varepsilon < 1$
daher nach dem Mastertheorem $T(n) \in \Theta(n^{\log_b(a)})$

b) $3 \frac{n}{4} \log(\frac{n}{4}) \leq c\, n \log(n) \Leftrightarrow 3\log(n) - 3\log(4) \leq 4c \log(n) \Leftrightarrow -3\log(4) \leq (4c-3)\log(n)$
für $c > \frac{3}{4}$ ab hinr. großem $n_0 \in \mathbb{N}$ erfüllt, also $T(n) \in \Theta(n \log(n))$

2) (4 P.) Geben Sie, für jedes gerade $n \in \mathbb{N}$ paarweise unterschiedliche Schlüssel $x_1, \ldots, x_n$ an, so dass ein Suchbaum in Kammform (sh. Abbildung) entsteht wenn, beginnend mit dem leeren Suchbaum, die Schlüssel $x_1, \ldots, x_n$ in dieser Reihenfolge eingefügt werden.



$$x_n := \begin{cases} n+1, & n \in 2\mathbb{N}+1 \\ n-1, & n \in 2\mathbb{N}+2 \end{cases}$$

3) (7 P.) Sei

$$A = (a_{i,j})_{\substack{1 \le i \le m \\ 1 \le j \le n}}$$

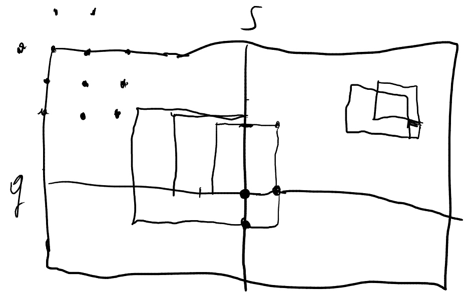
eine Matrix. Als *Untermatrix von A* bezeichnen wir eine Matrix der Form

$$(a_{i,j})_{\substack{p \le i \le q \\ r \le j \le s}}$$

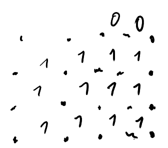
wobei $p, r \geq 1$, $q \leq m$ und $s \leq n$.

Geben Sie einen Algorithmus in Pseudocode an der eine Matrix $A \in \{0,1\}^{m \times n}$ als Eingabe erhält und der als Ausgabe das größte $k \in \mathbb{N}$ zurückliefert so dass A eine $k \times k$ Untermatrix enthält die nur aus Einsern besteht. Die Laufzeit des Algorithmus muss $O(m \cdot n)$ sein.

Hinweis: Dynamische Programmierung, betrachten Sie $m_{q,s}$, die maximale Größe einer quadratischen Einser-Matrix die an der Stelle (q, s) endet.



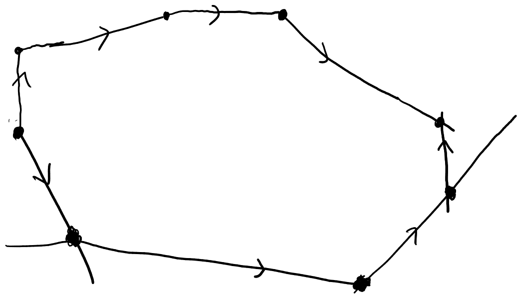

```
Prozedur Untermatrix
    m_{q,s} := 0, falls q=0 oder s=0
    for i = 1, ..., m
        for j = 1, ..., n
            if a_ij = 0
                m_ij := 0
            else if m_{i-1,j} = m_{i,j-1}
                if m_{i,j-1} ≤ m_{i-1,j-1}
                    m_{i,j} := m_{i-1,j} + 1
                else
                    m_{i,j} := m_{i-1,j}
                end if
            else
                m_{i,j} := min{ m_{i,j-1}, m_{i-1,j} } + 1
            end if
        End for
    End for
    return max(m_{i,j})
```



4) (9 P.) Eine *Strecke* ist ein ungeordnetes Paar $\{p, q\}$ wobei $p, q \in \mathbb{R}^2$ und $p \neq q$. Geben Sie einen Algorithmus in Pseudocode an, der als Eingabe eine endliche Menge von Strecken erhält, bei denen es sich um die Kanten eines konvexen Polygons handelt und der als Ausgabe eine im Uhrzeigersinn sortierte Liste der Knoten des Polygons zurückgibt.

Nehmen Sie der Einfachheit halber an, dass die Eingabe keine zwei Strecken $\{p, q\}, \{q, r\}$ enthält so dass p, q und r auf einer Gerade liegen.

Die Laufzeit des Algorithmus muss bei Eingabe von n Strecken $O(n \log n)$ sein.



```
Sei S ein Array mit Strecken der Länge n
for i = 1,...,n
    S[i] := Lexsort-x-y (S[i])    ◁ sortiert aufsteigend erst nach x, dann nach y
End for
S := Lexsort2(S)    ◁ Sortiert lexikographisch nach dem ersten Punkt der Strecke
Sei P neues Datenfeld der Länge S.Länge
P[1] := S[1][1]
if S[1][2].y < S[2][2].y
    P[2] := S[1][2]
else
    P[2] := S[2][2]
end if
i := 2
j := 3
while P[i] <_lex S[S.Länge][1]
    while P[i] ≠ S[j][1]
        j := j+1
    End while
    i := i+1
    P[i] := S[j][2]
End while
j := S.Länge
while P[i] >_lex S[1][1]
    while P[i] ≠ S[j][1]
        j := j-1
    End while
    i := i+1
    P[i] := S[j][2]
End while
return P
```

Prüfung aus

Diskrete und geometrische Algorithmen (Hetzl)

Zuname: ______________________________ Matrikelnummer: ____________________

Vorname: ______________________________

Titelseite ab hier bitte freilassen!

TU Wien, 26.4.2019 Arbeitszeit: 100 Minuten

1)

2)

3)

4)

1) (5 P.) Geben Sie für unendlich viele $n \in \mathbb{N}$ ein Datenfeld A_n der Länge n an, auf dem Einfügesortieren Laufzeit $\Theta(n\sqrt{n})$ hat und bewiesen Sie dass dem so ist.

$A_1 = (1)$

$A_2 = (2, 1)$

$A_3 = (3, 1, 2)$

$A_4 = (4, 1, 3, 2)$

$$A_n[i] = \begin{cases} A_{n-1}[i] + 1, & \text{falls } i < n \text{ und } A_{n-1}[i] \geq n - \lceil\sqrt{n}\rceil \\ A_{n-1}[i], & \text{falls } i < n \text{ und } A_{n-1}[i] < n - \lceil\sqrt{n}\rceil \\ \lceil n - \sqrt{n} \rceil, & \text{falls } i = n \end{cases}$$

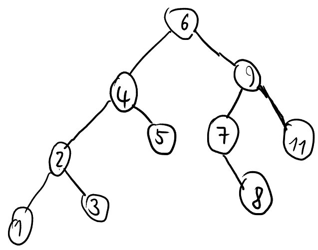
gibt es für A_n eine Anzahl von $n\lceil\sqrt{n}\rceil$ Schritten dann kommen bei A_{n+1} im letzten Schritt noch ungefähr $\lceil\sqrt{n}\rceil$ dazu um das letzte Element an die richtige Stelle zu bringen

2) (7 P.)

a) Geben Sie einen Algorithmus in Pseudocode an, der als Eingabe einen Suchbaum B mit n Elementen erhält und als Ausgabe ein Datenfeld liefert, da~~ss~~ die Elemente von B in aufsteigender Reihenfolge enthält. Die Laufzeit des Algorithmus muss $O(n)$ sein.

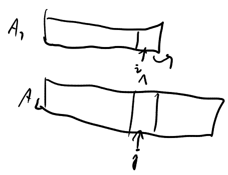
b) Geben Sie einen Algorithmus in Pseudocode an, der als Eingabe zwei Suchbäume B_1 und B_2 mit n_1 bzw. n_2 Elementen enthält und festellt ob deren Schlüsselmengen disjunkt sind. Die Laufzeit des Algorithmus muss $O(n_1 + n_2)$ sein.

a) Prozedur SORT (B)



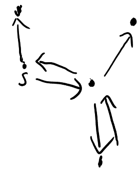
b) Prozedur DISJUNKT (B_1, B_2)

```
A_1 := SORT (B_1)
A_2 := SORT (B_2)
i := 1
j := 1
while A_1[i] ≠ A_2[j] ∧ j ≤ A_2.Länge ∧ i ≤ A_1.Länge
    if A_1[i] < A_2[j]
        i := i+1
    else if A_2[j] < A_1[i]
        j := j+1
    end if
end while
```

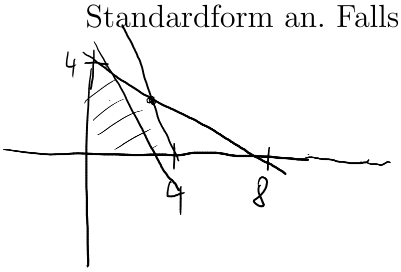


3) (7 P.) Ein *gerichteter Zyklus* in einem gerichteten Graphen $G = (V, E)$ ist ein Pfad der Form $v_1, v_2, \ldots, v_{k-1}, v_k = v_1$ mit $k \geq 2$ so dass für alle $i, j \in \{1, \ldots, k-1\}$ mit $i \neq j$ auch $v_i \neq v_j$ ist. Geben Sie einen Algorithmus in Pseudocode an, der als Eingabe einen endlichen gerichteten Graphen $G = (V, E)$ sowie ein $s \in V$ erhält und feststellt ob s in G auf einem gerichteten Zyklus liegt. Der Eingabgraph wird als Adjazenzliste übergeben. Die Laufzeit des Algorithmus muss $O(|V| + |E|)$ sein.

```
Prozedur ZYKLUS(V,E,s)
Sei A neues Datenfeld der Länge |V| überall mit falsch initialisiert
Sei Q leere Warteschlange
HINZUFÜGEN(Q,s)
A(s) := wahr
while Q ≠ ∅
    v := ENTFERNEN(Q)
      for alle (v,w) ∈ E
        if w = s und v.Vorgänger ≠ s
            return „Zyklus"
        end if
        if A(w) ≠ wahr
            w.Vorgänger := v
            A(w) := wahr
            HINZUFÜGEN(Q,w)
        End if
      End for
  end while
  return „kein Zyklus"
```



4) (5 P.) Wenden Sie den Simplex-Algorithmus auf das folgende lineare Programm in Standardform an. Falls das Programm eine optimale Lösung hat geben Sie diese an.



$$z = 2x_1 + x_2 \,\text{max!}$$

$x_1 + 2x_2 \leq 8 \rightsquigarrow 8 - x_1 - 2x_2 = x_3 \qquad \wedge\ x_3 \geq 0$

$3x_1 + x_2 \leq 12 \rightsquigarrow 12 - 3x_1 - x_2 = x_4 \qquad \wedge\ x_4 \geq 0$

Basislösung: $(0,0,8,12)$, also $z = 0$

aus x_3-Gleichung $x_1 \leq 8$ und aus x_4-Glg.: $x_1 \leq 4$

also tausche x_1 und x_4:

$x_1 = (12 - x_2 - x_4)\frac{1}{3} = 4 - \frac{1}{3}x_2 - \frac{1}{3}x_4$

$x_3 = 8 - (4 - \frac{1}{3}x_2 - \frac{1}{3}x_4) - 2x_2 = 4 - \frac{5}{3}x_2 + \frac{1}{3}x_4$

$z = 2(4 - \frac{1}{3}x_2 - \frac{1}{3}x_4) + x_2 = 8 + \frac{1}{3}x_2 - \frac{2}{3}x_4$

Basislösung $(4,0,4,0)$
$z = 8$

aus x_1-Glg. $x_2 \leq 12$, aus x_3-Glg. $x_2 \leq \frac{12}{5}$

also tausche x_3 und x_2

$x_2 = \frac{3}{5}(4 + \frac{1}{3}x_4 - x_3) = \frac{12}{5} + \frac{1}{5}x_4 - \frac{3}{5}x_3$

$x_1 = 4 - \frac{1}{3}(\frac{12}{5} + \frac{1}{5}x_4 - \frac{3}{5}x_3) - \frac{1}{3}x_4 = \frac{16}{5} - \frac{6}{15}x_4 + \frac{1}{5}x_3$

$z = 8 + \frac{1}{3}(\frac{12}{5} + \frac{1}{5}x_4 - \frac{3}{5}x_3) - \frac{2}{3}x_4 = \frac{44}{5} - \frac{9}{15}x_4 - \frac{1}{5}x_3$

Basislösung $(\frac{16}{5}, \frac{12}{5}, 0, 0)$
$z = \frac{44}{5}$: optimale Lsg.!

Prüfung aus

Diskrete und geometrische Algorithmen (Hetzl)

Zuname: ______________________________ Matrikelnummer: ____________________

Vorname: ______________________________

Titelseite ab hier bitte freilassen!

TU Wien, 2.7.2019 Arbeitszeit: 100 Minuten

1)

2)

3)

4)

1) (6 P.) Lösen Sie die Rekursionsgleichung $a_n = 4a_{n-1} - 3a_{n-2}$ (für $n \geq 2$) mit den Anfangswerten $a_0 = 1$ und $a_1 = 2$.

2) (6 P.) Geben Sie einen Algorithmus in Pseudocode an, der als Eingabe ein aufsteigend sortiertes Datenfeld A paarweise verschiedener ganzer Zahlen erhält und feststellt ob ein $i \in \{1, \ldots, n\}$ existiert mit $A[i] = i$. Die Laufzeit des Algorithmus muss $O(\log n)$ sein wobei n die Anzahl der Elemente in A ist.

3) (6 P.) Seien B_1 und B_2 Suchbäume die die selbe Menge von Einträgen enthalten. Gibt es eine Folge von Links- und Rechtsrotationen mit denen B_1 in B_2 transformiert werden kann? Beweisen Sie die Korrektheit Ihrer Antwort.

4) (6 P.) Geben Sie einen Algorithmus in Pseudocode an, der als Eingabe ein Datenfeld A paarweise verschiedener Elemente sowie ein $k \in \{1, \dots, A.\mathit{Länge}\}$ erhält und als Ausgabe eine zufällige k-elementige Teilmenge von A als Datenfeld liefert. Die Laufzeit des Algorithmus muss $O(k)$ sein.

Sie dürfen dazu die Existenz einer Prozedur $\textsc{Zufall}(i, j)$ voraussetzen, die für $i, j \in \mathbb{N}$ mit $i \leq j$ in Zeit $O(1)$ eine im Intervall $[i, j] \subseteq \mathbb{N}$ uniform verteilte Zufallszahl liefert.

Prüfung aus

Diskrete und geometrische Algorithmen (Hetzl)

Zuname: ______________________________ Matrikelnummer: ____________________

Vorname: ______________________________

Titelseite ab hier bitte freilassen!

TU Wien, 4.10.2019 Arbeitszeit: 100 Minuten

1)

2)

3)

4)

1) (6 P.) Lösen Sie die Rekursionsgleichung $a_n = 6a_{n-1} - 5a_{n-2}$ (für $n \geq 2$) mit den Anfangswerten $a_0 = 0$ und $a_1 = 1$.

2) (6 P.) Geben Sie einen Algorithmus in Pseudocode an, der als Eingabe ein aufsteigend sortiertes Datenfeld A paarweise verschiedener ganzer Zahlen erhält und feststellt ob ein $i \in \{1, \ldots, n\}$ existiert mit $A[i] = i$. Die Laufzeit des Algorithmus muss $O(\log n)$ sein wobei n die Anzahl der Elemente in A ist.

3) (7 P.) Sei

$$A = (a_{i,j})_{\substack{1 \leq i \leq m \\ 1 \leq j \leq n}}$$

eine Matrix. Als *Untermatrix von A* bezeichnen wir eine Matrix der Form

$$(a_{i,j})_{\substack{p \leq i \leq q \\ r \leq j \leq s}}$$

wobei $p, r \geq 1$, $q \leq m$ und $s \leq n$.

Geben Sie einen Algorithmus in Pseudocode an der eine Matrix $A \in \{0,1\}^{m \times n}$ als Eingabe erhält und der als Ausgabe das größte $k \in \mathbb{N}$ zurückliefert so dass A eine $k \times k$ Untermatrix enthält die nur aus Einsern besteht. Die Laufzeit des Algorithmus muss $O(m \cdot n)$ sein.

Hinweis: Dynamische Programmierung, betrachten Sie $m_{q,s}$, die maximale Größe einer quadratischen Einser-Matrix die an der Stelle (q, s) endet.

4) (5 P.) Wenden Sie den Simplex-Algorithmus auf das folgende lineare Programm in Standardform an. Falls das Programm eine optimale Lösung hat geben Sie diese an.

$$z = x_1 + 2x_2 \,\mathrm{max!}$$

$$2x_1 + x_2 \leq 8$$

$$x_1 + 3x_2 \leq 12$$