BFH-TI: Abteilung Informatik

Algorithmen und Datenstrukturen (AnD)

Modulabschlussprüfung

Simon KRAMER

9. Juli 2021

Total: 53.5 von 118 Punkten

RZM: 0

CC: ......

Modalitäten

1. Wie angekündigt dürfen Sie beidseitig ausgedruckt genau die folgenden Hilfsmittel für diese Prüfung verwenden:
   - entweder alle meine Folien (vier auf einer Seite) mit Ihren eigenen Annotationen darauf vom Unterricht und von Ihren Hausaufgaben dieses Kurses
   - oder Ihre eigene Zusammenfassung nicht grösseren Umfangs davon
2. Bitte
   (a) sehen Sie zuerst alles durch
   (b) fragen Sie, wenn etwas für Sie nicht klar ist
   (c) beginnen Sie mit den für Sie am einfachsten Aufgaben
   (d) wenden Sie alle Algorithmen wie im Unterricht präsentiert an
   (e) schreiben Sie in ***Blockschrift*** und mit (nicht-rotem) Kugelschreiber
   (f) händigen Sie alle Aufgabenblätter mit Ihren Vervollständigungen darauf innerhalb von 120 Minuten ein und versehen Sie dieses Deckblatt gleich jetzt mit Ihrem Namen

KandidatIn: Miriam Streit

Feedback vom Lehrer: ....

Aufgabe 1 (....5.. von 8 Punkten)

log Π i = log n! → FOLIEN!

Aussage				wahr/falsch?	Begründung (⇐ Punkt)
$\sum_{i=1}^{n}$	c	ist	$\Theta(n)$	wahr	$\Theta(n \cdot c) = \Theta(n)$
	$\log i$		$\Theta(n \log n)$	wahr	$\log(i) + \log(i) + \log(i) \ldots = n \log(n) = \Theta(n \log n)$
	$\sqrt{i}$ ($i^{\frac{1}{2}}$)		$\Theta(n\sqrt{n})$	wahr	$\sqrt{i} + \sqrt{i} + \sqrt{i} \ldots = n \cdot \sqrt{n} = \Theta(n\sqrt{n})$
	i (i^1)		$\Theta(n^2)$	wahr	$i + i + i + i = n \cdot i = \Theta(n \cdot n) = \Theta(n^2)$ ARITHMETISCHE Σ
	$i \log i$		$\Theta(n^2 \log n)$	wahr	$i\log i + i\log i + i\log i = n i \log i = \Theta(n^2 \log n)$ (✓)
	$i\sqrt{i}$ ($i^{\frac{3}{2}}$)		$\Theta(n^2\sqrt{n})$	wahr	$i\sqrt{i} + i\sqrt{i} + i\sqrt{i} = n \cdot i \cdot \sqrt{i} = \Theta(n^2\sqrt{n})$
	i^c		$\Theta(n^{c+1})$	wahr	$n \cdot n^c = \Theta(n^{c+1})$ FOLIEN!
	c^i		$\Theta(c^n)$	falsch	$c^i + c^i + c^i = n \cdot c^i = \Theta(n \cdot c^n)$

~~Begründungen basieren auf den Rechenregeln der~~ Big-O Notation, Θ wurde jeweils weggelassen

GEOMETRISCHE SUMME!

Aufgabe 2 (....5. von 18 Punkten)

Das asymmetrische Verschlüsselungs- und Signaturverfahren RSA von Rivest-Shamir-Adleman interpretiert zu verschlüsselnde respektive zu signierende Texte oder Dokumente (Bit-Strings, meistens nur symmetrische Session-Schlüssel respektive gehashte zu signierende Dokumente, beide mit einer typischen Länge von 256–512 bit) und die dazu verwendeten asymmetrischen Schlüssel (Bit-Strings, mindestens 4096 bit) als natürliche Zahlen (in binärer Repräsentation). Die Verschlüsselungsoperation (*encr*) ist (direkt) als Potenzieren in der modularen Arithmetik definiert:

$$encr(M,(e\,,N)) \;=\; M^e \bmod N\,,$$

wobei M die zu verschlüsselnde oder signierende Nachricht (*plaintext message*) und $(e\,,N)$ der aus den zwei Teilen e (*encryption exponent*) und N (*modulus*) bestehende (öffentliche) Schlüssel ist.

Weil jedoch die involvierten Zahlen grösser als primitiv-typisierte Zahlen (fixer Länge, typischerweise 32 oder 64 bit, z.B. Java-`int` respektive Java-`long`) und sogar nach oben unbegrenzt sind (vor allem die Textlänge aber mit wachsenden Sicherheitsanforderungen auch die Schlüssellänge), müssen Zahlen beliebiger Länge (z.B. Java-`BigInteger`) verwendet und die Operation effizienzhalber schrittweise indirekt berechnet werden und zwar wie folgt (recursive *successive squaring*):

$$encr(M,(e\,,N)) = \begin{cases} 1 & \text{wenn } e = 0, \\ C & \text{wenn } e \text{ geradzahlig ist und} \\ (C\cdot M) \bmod N & \text{andernfalls} \end{cases}$$

wobei $C = (encr(M,(\lfloor e/2 \rfloor\,,N)))^2 \bmod N$. Bitte (..5.. von 9 respektive ...0 von 18 Punkten):

1. übersetzen Sie diese rekursive Definition in ein (iteratives) Java `while`-Schlaufen-Programm

```
public BigInteger encrIterative(String M, long e, long N) {
    while(e!=0) {
        if(e % 2==0) {
            e=Math.floor(e/2); // hier wird nur e kleiner, Rest sollte bleiben
            return;
        } //if
        else {
            e=Math.floor(e/2);
M=
        } //else } //while
    return M;
} // Funktion
```

im Else-Fall müsste eine weitere Schlaufe laufen, die M=(C·M) mod N → berechnet

im if-Fall sollte nur e reduziert werden

Problem: wie bringt man iterativ das Quadrat (hoch 2) hinein?

? ? ?

2. bestimmen Sie die Θ-Zeitkomplexität Ihres Programms (und damit der Definition)

Hinweis: Gehen Sie analog zu Ihren benoteten Hausaufgaben vor, und nehmen Sie also vereinfachend an, dass Multiplikation und Division (auch die ganzzahlige) Θ(1) sind ;-)

$$\sum_{i=e}^{n=e/2}\left(\frac{i}{2}+c_1\right)^2 = \sum_{i=e}^{n=e/2}\left(\frac{i^2}{4}+c_1^2\right) \;\dots\; ?$$

Aufgabe 3 (siehe Abbildung 1, ..3.. von 25 Punkten)

In einem *Divide-and-Conquer* und fächerübergreifenden (Matrizenmultiplikation $*$) *Esprit* seien

$$A = B * C \qquad B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} \qquad C = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

wobei B und C rekursiv quadratische Matrizen mit Zweierpotenzen-Seitenlängen n sind.

Hypothese: Addition und Multiplikation der (atomaren) Skalaren (zum Beispiel Zahlen mit endlicher—z.B. 64 bit—Präzision) der Matrizen ist $\Theta(1)$. Warum ist diese Hypothese realistisch?

Antwort (....1 von 1 Punkt):

keine Iteration, nur elementare Instruktionen (vgl. Multiplikation + Addition + Subtraktion)

Gedankenblitz aus heiterem Himmel (....0. von 12 Punkten): Bitte bestimmen Sie die folgenden Zeitkomplexitäten unter Verwendung von (abstrakten) indexierten Konstanten

Berechnung (Matrizenadditionen und -multiplikationen)	Zeitkomplexität	Punkt
$B_2 = B_{21} + B_{22}$	c_1 $\mathcal{O}(1)$	
$B_3 = B_2 - B_{11}$	$c_1 + c_2$ $\mathcal{O}(1)$	
$C_2 = C_{22} - C_{12}$	c_3 $\mathcal{O}(1)$	
$C_3 = C_2 + C_{11}$	$c_3 + c_4$ $\mathcal{O}(1)$	
$M_1 = B_3 * C_3$	$T_*(n/b) + c_5$	
$M_2 = B_{11} * C_{11}$	$T_*(n/b) + c_6$	
$M_3 = B_{12} * C_{21}$	$T_*(n/b) + c_7$	
$M_4 = (B_{11} - B_{21}) * C_2$	$T_*(n/b) + c_3 + c_8$	
$M_5 = B_2 * (C_{12} - C_{11})$	$T_*(n/b) + c_1 + c_9$	
$M_6 = (B_{12} - B_3) * C_{22}$	$T_*(n/b) + c_1 + c_2 + c_{10}$	
$M_7 = B_{22} * (C_3 - C_{21})$	$T_*(n/b) + c_3 + c_4 + c_{11}$	
$A = \begin{bmatrix} M_2 + M_3 & (M_1 + M_2) + M_5 + M_6 \\ ((M_1 + M_2) + M_4) - M_7 & ((M_1 + M_2) + M_4) + M_5 \end{bmatrix}$	$3c_1 + c_2 + 3c_3 + c_4$ $3c_5 + 4c_6 + c_7 + 2c_8 + 2c_9 + c_{10} + c_{11}$	

Matrizenadditionen: ist $\Theta(n^2)$

$c_1 + c_2 + c_3 + c_4 = c_5$

und zwar zur Berechnung (*closed-form solution*) der Θ-Zeitkomplexität

$$T_*(n) = \begin{cases} \Theta(1) & \text{wenn } n = 2^1 \text{ und} \\ a\,T_*(n/b) + f(n) & \text{andernfalls} \end{cases}$$

mit Hilfe unseres Grundlagen-Tools (*Master Theorem*) durch Bestimmung der Knotenarbeit

$f(n) =$ $3c_1 + c_2 + 3c_3 + c_4 + 3c_5 +$ [illegible] $+ c_7 + 2c_8 + 2c_9 + c_{10} + c_{11}$

(...0... von 7 Punkten) und seiner Parameter a, b, c, l und k (...2... von 5 Punkten):

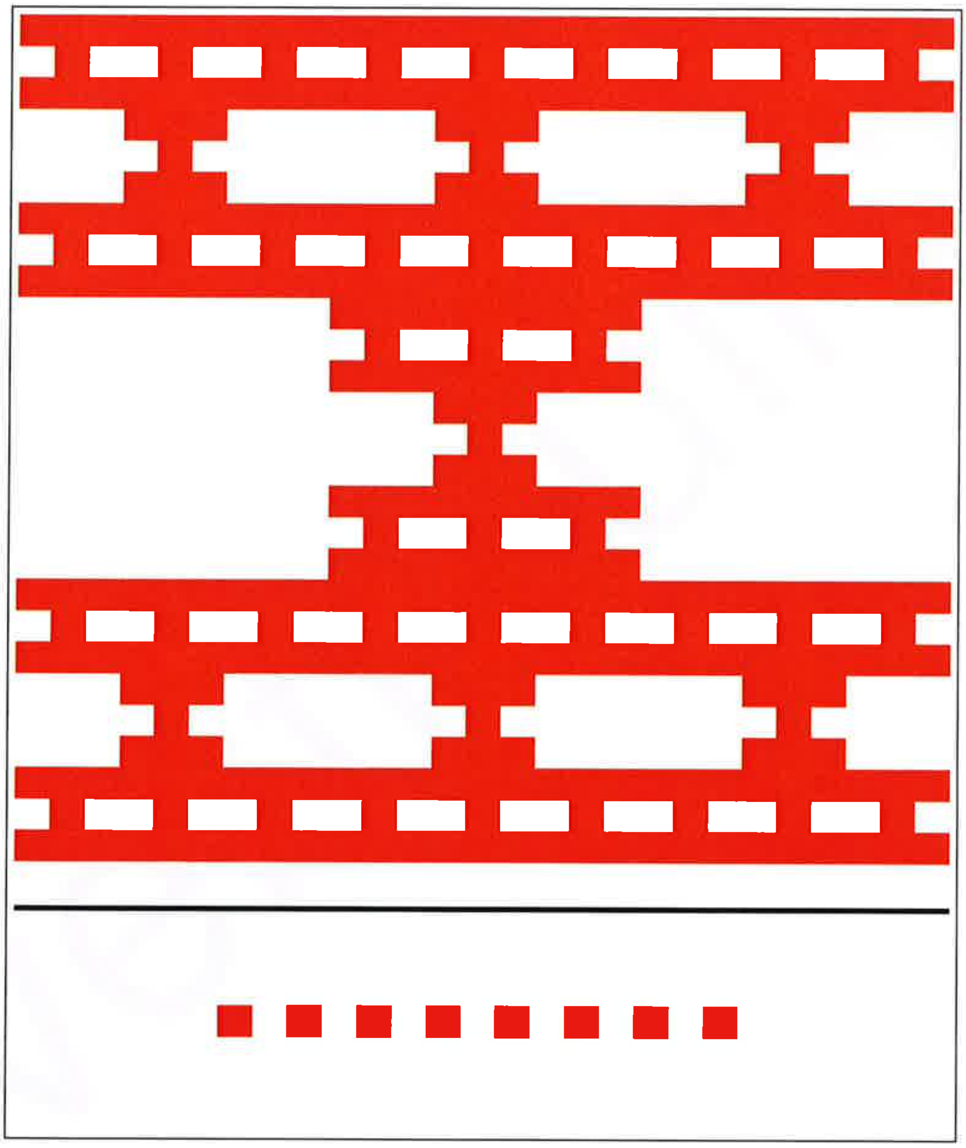
Lösung	Punkt	Lösung	Punkt
$a = \ldots$	~~2~~ 7	$l =$ ~~1~~ 2 $f(n) = \Theta(n)$	
$b = \ldots$	2	$k = 0$	
$c = 1$		$T_*(n) = \Theta($~~$n \log n$~~$)$ $n^{\log_2 7}$	

Bei der 3x3 Matrix multiplikation besteht die Multipl. aus einer Subtraktion von 2 Skalarprodukten.
→ Ein Problem wird in 2 Probleme von Grösse n/2 unterteilt

$l = \log_b a \Rightarrow 1 = 1$
$\rightarrow \Theta(f(n) \log(n))$

!

Abbildung 1: Graphischer Hinweis zu Aufgabe 3 («Fraktalbruch» zu $n^{\log_b(a)} = n^{\frac{\log_3(a)}{\log_3(b)}}$;-)

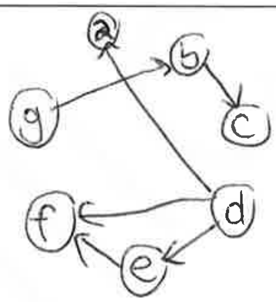


Aufgabe 4 (. . .5. . von 15 Punkten)

1. (. . .2. von 7 Punkten) Bitte
   (a) betrachten Sie die nachfolgend links aufgezählten, sieben Konstruktionen
   (b) ordnen Sie deren Bezeichner daneben rechts auf einem imaginären Kreis an
   (c) verbinden Sie diese so angeordneten Bezeichner mit gerichteten Verwendungspfeilen:

   $\boxed{\text{Konstruktion } (x) \text{ dient als } \textit{Plug-in} \text{ für Konstruktion } (y)\text{, symbolisch: } x \mapsto y}$

   (also sind x und y Bezeichnervariablen) und zwar als Motivation für Aufgabe 5 ;-)

existierende Konstruktionen	**gerichteter Konstruktionsgraph**
(a) injective binary-string encoding (serialisation) (b) binary-digital search tree (binary trie, fast fail) (c) string-prefix *partial* ordering (properties?) (d) hash function (`int` hash code) (e) hash table (expected $O(1)$-time, slow fail) (f) *linear/total* ordering (properties?) (g) (ordinary) search tree ($O(\log n)$-time)	

$x \mapsto y$	**Begründung** ($\Leftarrow$ **Punkt**)
d. ↦ e. ✓	HashTable benutzt Hashwerte zur Anordnung
b. ↦ E ?	DST wird benutzt, um Strings nach Präfix zu sortieren
d ↦ a. ?	Hashfunktion kann auch zum verschlüsseln dienen
g. ↦ b. ✓	DST ist ein Subtyp von search trees
e. ↦ f. ?	HashTable implementiert eine totale Sortierung
d. ↦ f. ?	Mit Hashcode kann eine totale Sortierung implementiert werden
. . . ↦ . . .	

2. Bitte ordnen Sie nun freihändig Ihren resultierenden (nicht linear geordneten, azyklischen) Graphen (*directed acyclic graph, DAG*) topologisch—also linear (. . .2. von 7 Punkten): FOLIE!

Hinweis: post-order traversal ;-)	
~~c → b → g~~	g → b → c
f → e → d	d → e → f
a → d	d → a
e → d	d → e
→ = 1. basiert auf 2.	→ = 1. hilft bei Implementierung von 2.

3. Welches ist also das nützlichste *Plug-in* (. . . .1. von 1 Punkt)?

Die Hashfunktion, (d) KONSISTENT MIT IHREM 2.

(a)

Aufgabe 5 (......2 von 16 Punkten)

Bitte bestimmen Sie die Θ-Zeitkomplexität T_{enc} der folgenden, rekursiv definierten Encodierungsfunktion *enc* der (Turing-vollständigen) sogenannten Kombinatoren-$\mathcal{T}$erme S und K mit der sich verzweigenden, jedoch ziemlich primitiven Datenstruktur des geordneten $\mathcal{T}$erm-Paares $(\mathcal{T}, \mathcal{T}')$

$$
\begin{aligned}
enc(\mathtt{S}) &= \mathtt{00} \\
enc(\mathtt{K}) &= \mathtt{01} \\
enc((\mathcal{T}, \mathcal{T}')) &= \mathtt{1}\,enc(\mathcal{T})\,enc(\mathcal{T}')
\end{aligned}
$$

und zwar in Analogie zum Quicksort und unter der Annahme, dass die Knotenarbeit (Arbeit vor dem Hinuntersteigen in die—plus Arbeit nach dem Hochsteigen aus der—Rekursion) konstant ist:

$$
\boxed{T_{enc}(n) = \begin{cases} \Theta(1) & \text{beide Basisfälle } (n = 1) \\ T_{enc}(|\mathcal{T}|) + T_{enc}(|\mathcal{T}'|) + \Theta(1) & \text{rekursiver Fall } (n = |\mathcal{T}| + |\mathcal{T}'| + 3) \end{cases}}
$$

Hinweis: $enc(((\mathtt{S}, \mathtt{K}), \mathtt{K})) = \mathtt{11000101}$ (Objekt-Serialisierung in eine binäre Zeichenkette)

- **Best case** (bitte mit sorgfältiger Herleitung) (**...... von 7 Punkten**).

- **Worst case** (bitte mit sorgfältiger Herleitung) (**...... von 7 Punkten**).

Aus was besteht die Knotenarbeit (...2 von 2 Punkten)?

- *decompose* (...... von 1 Punkt): aufteilen in Paare (die keine Unterpaare enthalten) ✓
- *recombine* (...... von 1 Punkt): wieder in Paare mit Unterpaaren zusammenfügen ✓

  ↳ CONCATENATION

Aufgabe 6 (...18. von 20 Punkten)

1. Bitte erklären Sie die Funktionsweise des folgenden, sogenannten *Selection Sort*-Verfahrens:

```
1 procedure select(T[1..n])
2     for i ← 1 to n − 1 do
3         minj ← i; minx ← T[i]
4         for j ← i + 1 to n do
5             if T[j] < minx then minj ← j
6                                 minx ← T[j]
7         T[minj] ← T[i]
8         T[i] ← minx
```

Kurze Erklärung (bitte keine triviale Code-Paraphrase) (..3. **von** 3 **Punkten**):

In der äusseren Schleife wird ein Minimum selektiert, in des inneren Schleife wird dieses Minimum gegen jeden Wert geprüft. Ist des aktuelle Wert kleiner als das Minimum, wird er als Minimum gewählt und es wird weiter geprüft. Das in der inneren Schleife gefundene Minimum wird an das sortierte Präfix angehängt und es beginnt von vorne (nur beim unsortierten Postfix)

2. Bitte bestimmen Sie die Θ-Zeitkomplexität des Algorithmus.

Lösung (bitte mit sorgfältiger Herleitung) (...11.. **von** 13 **Punkten**):

~~Elementare operationen: Zeilen~~ 3, 5, 6, 7, 8, wobei 3, 5, 6 in 1-2 Loops sind

Ein 2facher for-loop sollte zu $O(n^2)$ führen

$$= \sum_{i=1}^{n-1} (i + c_1) + \sum_{i=2}^{n} (i + c_2) = \sum_{i=1}^{n-1} (i + c_1) + \sum_{i=1}^{n-1} (i + 1 + c_2)$$

$$= \sum_{i=1}^{n-1} \big((i + c_1)((i+1) + c_2)\big) = \sum_{i=0}^{n} \big((i+1+c_1)(i+2+c_2)\big) = (n+1+c_1)(n+2 \cdot c_2)$$

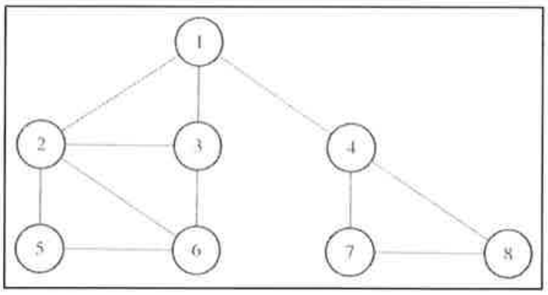
$$= \Theta(n^2)$$

3. Bitte beantworten Sie die folgenden zwei Fragen zum Algorithmus: +

Frage	Ist er *in-place?*
Antwort	Ja
Begründung	Es wird maximal der Speicher für den Input (T) plus einen konstanten Zusatz verwendet
Frage	Ist er auch *on-line* tauglich?
Antwort	nein
Begründung	kommt ein neues Minimum hinzu, das kleiner ist als die letzte Stelle des sortierten Präfixes, so kommt die Sortierung falsch heraus, da das Minimum ohne weiteren Check dahinter gehängt wird

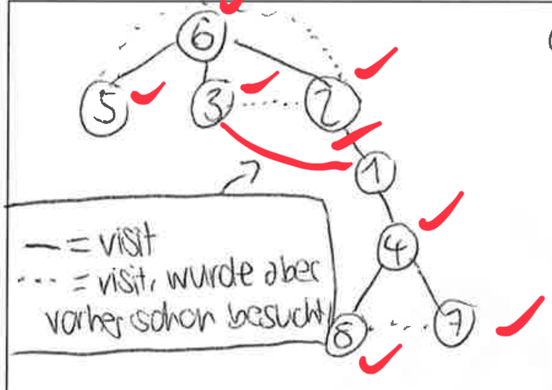
Aufgabe 7 (7.5 von 8 Punkten)

Betrachten Sie den folgenden *ungerichteten* Graphen und setzen Sie sich gedanklich auf Knoten 6:



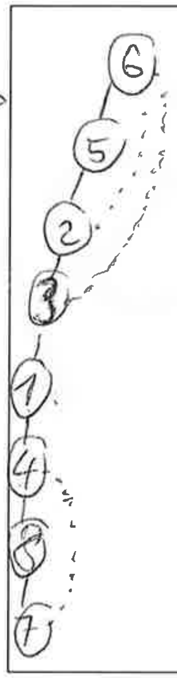
Bitte besuchen Sie nun den Graphen in absteigender Reihenfolge der Nachkommen und zwar als

1. *breadth-first* Baum (3.5 von 4 Punkten):



Queue: | 6 | 5 | 3 | 2 | 1 | 4 | 8 | 7 |

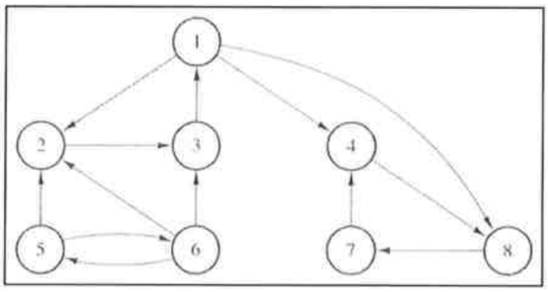
2. *depth-first* Baum (4 von 4 Punkten): :-)



(Arbeit mit Stack)

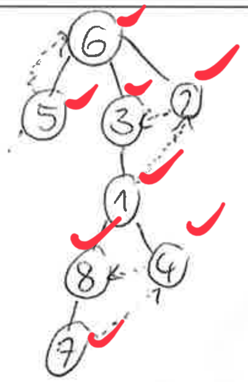
Aufgabe 8 (..8.. von 8 Punkten) :-)

Betrachten Sie den folgenden *gerichteten* Graphen und setzen Sie sich gedanklich auf Knoten 6:



Bitte besuchen Sie nun diesen Graphen in absteigender Reihenfolge der Nachkommen und zwar als

1. *breadth-first* Baum (..4.. von 4 Punkten): :-)



Queue: | 6 | 5 | 3 | 2 | 1 | 8 | 4 | 7 |

— = direkter Besuch

--> = Besuch, ist aber schon besucht worden, deshalb wird der Knoten nicht erneut besucht

2. *depth-first* Baum (..4.. von 4 Punkten): :-)