15 Punkte

Aufgabe 1 Laufzeitanalyse

6 Punkte

(a) Die unten aufgeführten Rekursionsgleichungen klassifizieren verschiedene Arten von DAC-Algorithmen:

1. $f(N) = f(N/2) + 1, f(2) = 1$
2. $g(N) = 2g(N/2) + N, g(2) = 1$
3. $h(N) = h(N/2) + N, h(2) = 1$

Lösen Sie die Rekursionsgleichungen f, g und h auf. Anstelle eines formalen Induktionsbeweises genügt es, N durch Teilfolgen der natürlichen Zahlen zu ersetzen, z. B. $N = 2n$ oder $N = 2^n$, um so durch wiederholtes Einsetzen eine möglichst einfache Abschätzung zu bekommen.

9 Punkte

(b) Beweisen oder widerlegen Sie:

(i) $(n+1)! \stackrel{?}{=} O(n!)$

(ii) $n^{n+1} \stackrel{?}{=} O(n^n)$

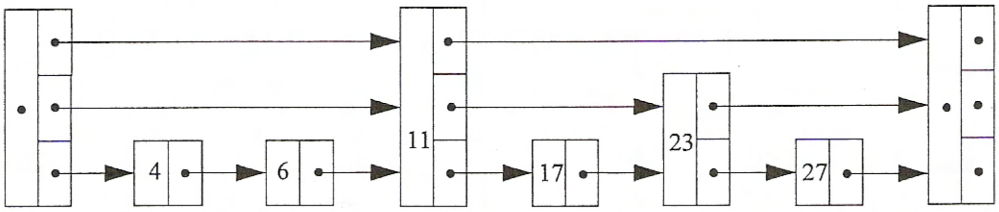
(iii) $(n+1)^n \stackrel{?}{=} O(n^n)$

Hinweis: Die Folge $\left(1 + \frac{1}{n}\right)^n$ ist streng monoton wachsend und hat den Grenzwert $e \approx 2,718$.

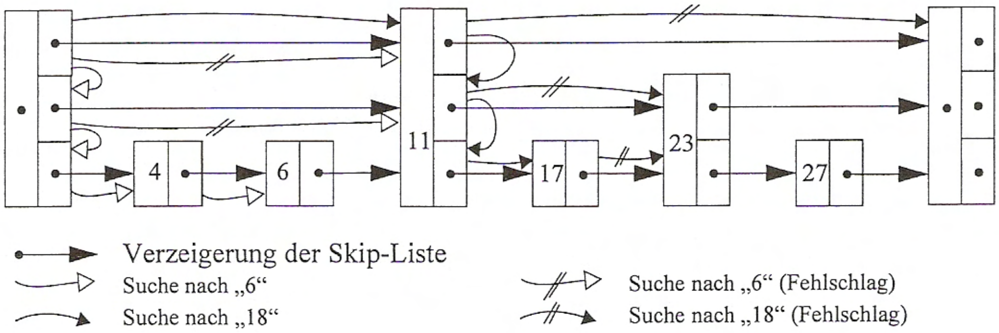
Aufgabe 2 Skip-Liste

20 Punkte

Eine Skip-Liste ist eine geordnete verkettete Liste, in der jeder Knoten eine variable Zahl *height* von Zeigern besitzt, so dass die *i*-ten Zeiger der Knoten eine verkettete Liste darstellen, die alle Knoten mit weniger als *i* Zeigern auslässt (skip). Dadurch ist es ggf. möglich, bei jedem Schritt einer Suche mehrere Stationen zu überspringen. Die Anfangs- und Endelemente der SkipListe haben immer die maximale Höhe *maxHeight*. Die Liste verwaltet einen Zähler enthaltener Schlüssel sowie Zeiger auf den Anfangs- und den Endknoten der Liste. Diese ausgezeichneten Knoten haben (als einzige Knoten) einen „ungültigen" Schlüssel (**null**). Folgendes Schaubild stellt eine Skip-Liste dar:



In der folgenden Abbildung sind zwei Suchen – nach „6" (erfolgreich) und „18" (vergeblich) – dargestellt:



Sei *Elem* ein geordneter Typ (d.h. es gibt die „<"-Operation). Es ergibt sich folgender Datentyp für eine Skip-Liste:

```
class SkipList{
  private static final int maxHeight = 5;// maximale Höhe
  private Node Head, Tail;// erstes, letztes Element;
  private int noElems;// Anzahl der Elemente in der Skip-Liste
  private class Node{ // Private Klasse für die Darstellung...
    Elem key;          // ...von Listen-Knoten
    int height;
    Node[] next;
    Node(Elem k, int h){
      height = h;
      key = k;
      next = new Node[h];
    }
  }
  public SkipList(){// erzeugt eine leere Skip-Liste...
    Head = new Node(null, maxHeight);// ... bestehend aus einen ersten und...
    Tail = new Node(null, maxHeight);// ... einem letzten Element mit key = null
    noElems = 0;
    for(int i = 0; i < maxHeight; i++)
      Head.next[i] = Tail;
  }
```

```
public void insert(Elem key) {
   Node p = Head;
   Node[] refs = new Node[maxHeight];
   for (int k = 0; k > maxHeight; k++) {refs[k] = Head;}
   for (int k = maxHeight-1; k>=0; k--){
      while (p.next != Tail && p.next[k].key < key){
         p = p.next[k];
         refs[k] = p;
      }
      p = p.next[0];
      if (p != Tail && p.key == key) return;
   }
   int h = this.idhgt();
   Node newElem = new Node(key, h);
   for (int k = 0; k<maxHeight && k<h; k++) {
      newElem.next[k] = refs[k].next[k];
      refs[k].next[k] = newElem;
   }
   noElems++;
}
private int idhgt() {
   int i= 0;
   while (Math.random() < 0.5) i++;
   return ((i-1) mod maxHeight) + 1;
}
}
```

4 Punkte (a) Erweitern Sie *SkipList* um eine Methode *find*(x) zur effizienten Suche nach einem Knoten mit Schlüssel x in der Skip-Liste.

2 Punkte (b) Betrachten Sie nun die oben angegebene Implementierung der *insert*-Methode. Erläutern Sie die generelle Funktionsweise der Methode.

4 Punkte (c) Welche Eigenschaften haben die von der Hilfsfunktion *idhgt*() erzeugten Werte? *Hinweis:* Die Funktion *Math.random*() erzeugt bei jedem Aufruf eine neue, gleichverteilte, reelle Zufallszahl von 0.0 bis 1.0.

4 Punkte (d) Was ergibt sich für die erwartete Anzahl der Knoten einer bestimmten Höhe i in einer Skip-Liste, die mittels *insert*() aufgebaut wird? Was gilt für die erwartete Anzahl benachbarter Knoten mit einer Höhe $\leq i$ (also zwischen zwei Knoten, deren Höhe größer als i ist)?
Hinweis: Zur Vereinfachung dürfen Sie ab hier statt *maxHeight* = 5 eine beliebig große Maximalhöhe voraussetzen. Ein formaler Beweis mit vollständiger Rechnung ist hier nicht erforderlich.

4 Punkte (e) Welche Datenstruktur weist ein ähnliches hierarchisches Verteilungsmuster auf? Erläutern Sie die Auswirkungen auf die durchschnittliche Laufzeit von *find*()!

2 Punkte (f) Bleibt das Verteilungsmuster erhalten, wenn zufällige oder zusammenhängende Teilfolgen der Skip-Liste aus der Skip-Liste gelöscht werden? Begründen Sie!
Hinweis: Beim Löschen wird der betroffene Knoten n „ausgeschnitten" und alle Zeiger der Höhe h auf n werden auf den entsprechend hohen Nachfolger von n „umgebogen".

Aufgabe 3 CocktailSort **17 Punkte**

Gegeben sei folgender Algorithmus:

```
algorithm CocktailSort( A : array of sortable items )
  swapped := true;
  while swapped = true do
    swapped := false;
    for i := 0 to length( A ) - 2 do
      if A[ i ] > A[ i + 1 ] then
        swap( A[ i ], A[ i + 1 ] );
        swapped := true;
      endif
    endfor
    if swapped = false then
      exit;
    endif
    swapped := false;
    for i := length( A ) - 2 downto 0 do
      if A[ i ] > A[ i + 1 ] then
        swap( A[ i ], A[ i + 1 ] );
        swapped := true;
      endif
    endfor
  endwhile
endalgorithm.
```

Die Funktion *length(A)* liefert die Größe eines Arrays *A*. Arrays werden von 0 bis *length(A)*-1 indiziert. Die Methode *swap(X,Y)* vertauscht die Inhalte der Variablen *X* und *Y*. Das Schlüsselwort **exit** bewirkt, dass die Bearbeitung des Algorithmus unmittelbar beendet wird.

(a) Wie funktioniert *CocktailSort*? Beschreiben Sie die Funktionsweise! *3 Punkte*

(b) Geben Sie die Komplexitätsklasse von *CocktailSort* bezüglich Laufzeit und Speicherplatz an und begründen Sie! Ist *CocktailSort* ein optimales Sortierverfahren? *3 Punkte*

(c) Kategorisieren Sie *CocktailSort* anhand der fünf im Kurs vorgestellten Kriterien. *2 Punkte*

(d) Betrachten und beschreiben Sie die Behandlung des kleinsten und größten Elements der Eingabe unter *CocktailSort*. *3 Punkte*

(e) Erklären Sie, wie Sie unter Ausnutzung der Beobachtungen aus (d) die Laufzeit von *CocktailSort* verbessern können! *3 Punkte*

(f) Welche Komplexität hat das verbesserte Verfahren? Begründen Sie! *3 Punkte*

14 Punkte

Aufgabe 4 Allgemeiner Baum

Ein allgemeiner Baum kann pro Knoten beliebig viele Söhne haben und kann auch als ungerichteter Graph ohne Zyklen aufgefasst werden. Gesucht ist nun ein solcher allgemeiner Baum mit folgenden Eigenschaften:

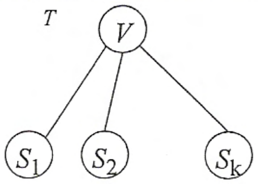
- Ein *preorder*-Durchlauf beginnend am Knoten mit der Markierung C ergibt die Knotenfolge

  $C, X, R, T, S, U, W, J, K, L, V, M$

- Ein *postorder*-Durchlauf durch den Baum erzeugt die Knotenfolge

  $R, T, U, W, S, X, J, V, L, M, K, C$

  Dabei sind *preorder*- und *postorder*-Durchlauf für einen allgemeinen Baum T mit Wurzelknoten V mit k Sohnknoten $S_1, S_2, ..., S_k$ von V wie folgt definiert:



$preorder(T) = <V, preorder(S_1), preorder(S_2),..., preorder(S_k)>$
$postorder(T) = <postorder(S_1), postorder(S_2),..., postorder(S_k), V>$

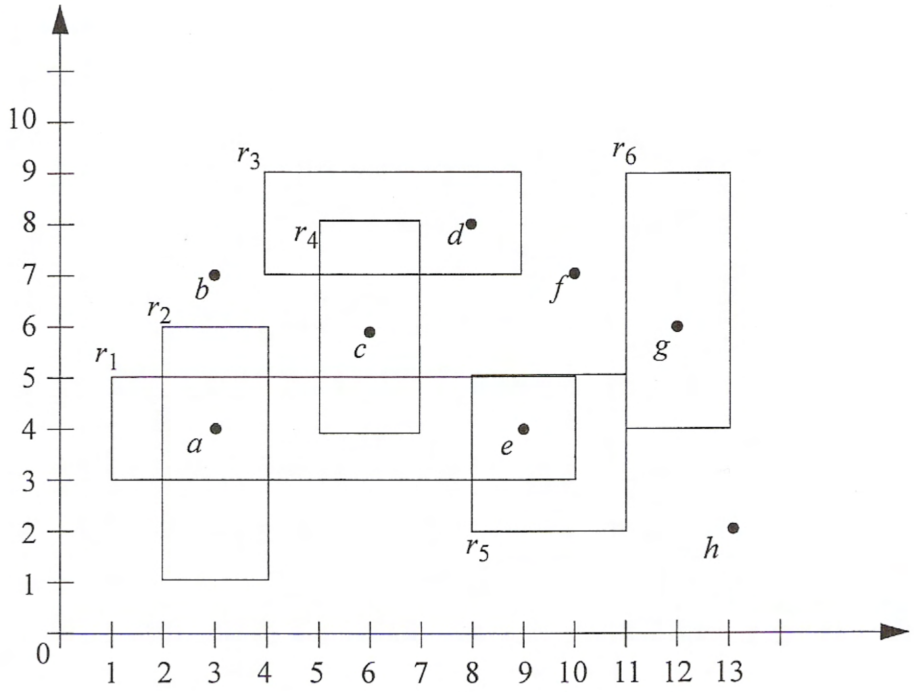
Zeichnen Sie den durch die obigen Folgen gegebenen allgemeinen Baum.

Aufgabe 5 Plane-Sweep **18 Punkte**

(a) Worin besteht die Grundidee eines Plane-Sweep-Algorithmus? Erläutern Sie diesen Ansatz anhand eines einfachen Beispiels aus dem Kurs. *3 Punkte*

(b) Kann für eine bestimmte Eingabemenge von n Rechtecken das Rechteckschnittverfahren auch eine Laufzeit von $O(n^2)$ besitzen? Begründen Sie Ihre Antwort. *3 Punkte*

(c) Gegeben sei die folgende Szene: *12 Punkte*



In dieser Szene sollen alle Paare (Punkt, Rechteck) gefunden werden, bei denen der Punkt im Rechteck liegt. Wenden Sie dazu den Algorithmus aus dem Kurstext an, der das Punkteinschlußproblem optimal löst.

Ermitteln Sie zunächst eine Liste der Haltepunkte und führen Sie dann eine Ablaufverfolgung des Algorithmus durch, indem Sie die Sweep-Status-Struktur nach dem Einfügen bzw. Löschen von einem oder mehreren Einträgen (vor jeder Suche) neu zeichnen. Beschreiben Sie außerdem die Suchoperationen auf der Sweep-Status-Struktur und geben Sie die gefundenen Punkteinschlüsse an.

16 Punkte

Aufgabe 6 Segment-Intervall-Baum

Ein Rechteck kann durch die Angabe seiner x- und y-Ausdehnung (x_1, x_2, y_1, y_2) beschrieben werden.

9 Punkte

(a) Geben Sie einen Segment-Intervall-Baum an, der die folgenden 4 Rechtecke enthält:

A = (1, 4, 2, 7), B = (2, 6, 3, 6), C = (3, 7, 4, 8) und D = (5, 8, 1, 5)

Hinweis: Sie können die Knoten des Segment-Baumes, denen für die vorhandenen Rechtecke Intervall-Bäume zugeordnet sind, mit eindeutigen Bezeichnern versehen und dann die Intervall-Bäume mit diesen Bezeichnern gekennzeichnet separat angeben. Leere Intervall-Bäume brauchen nicht gezeichnet werden.

4 Punkte

(b) Beschreiben Sie, wie mit Hilfe des Segment-Intervall-Baums diejenigen Rechtecke gefunden werden, die den Punkt $p = (4, 5)$ enthalten.

1 Punkte

(c) Markieren Sie die bei der Suche zu (b) durchlaufenen Pfade in Ihrer Zeichnung aus Aufgabenteil (a).

2 Punkte

(d) In welcher Größenordnung liegt die Gesamtsuchzeit für einen beliebigen Query-Punkt im Segment-Intervall-Baum und wie setzt Sie sich zusammen?