RWTH Aachen **SS 2011**
Lehrgebiet Theoretische Informatik Übungsblatt 6
Emmes—Langer—Reidl—Rossmanith 16.05.2011

Übung zur Vorlesung Datenstrukturen und Algorithmen

Aufgabe T19

Überlegen Sie, wie sich mit Hilfe eines einzigen 32-Bit Integers Teilmengen von $\{0, \ldots, 30\}$ darstellen lassen und geben Sie an, wie die folgenden Operationen implementiert werden können: (1) Einfügen, Löschen und Enthaltensein eines Elements, und (2) Vereinigung und Schnitt zweier Mengen.

Aufgabe T20

Eine Skip-List hat einen Head-Knoten, der eine bestimmte Höhe besitzt. Soll ein Element mit einer größeren Höhe in die Liste eingefügt werden, so muss der Head-Knoten vergrößert werden—der in der Vorlesung präsentierte Algorithmus zum Löschen von Elementen verändert die Höhe des Head-Knoten allerdings nie. Kann dies zu einem schlechteren Laufzeitverhalten führen, als eine Implementierung die beim Löschen den Head-Knoten verkleinert?

Aufgabe T21

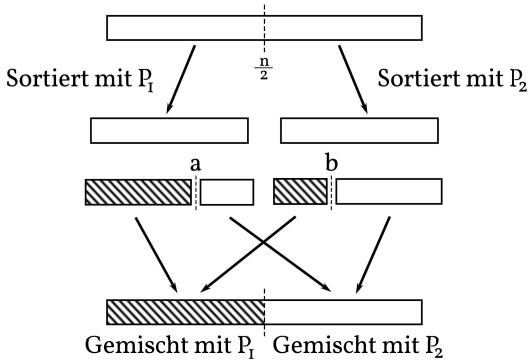
In der Vorlesung wurde Quicksort auch iterativ mit Hilfe eines expliziten Stacks implementiert. Da Speicherverbrauch immer ein wichtiger Faktor ist, sind wir an der maximalen Höhe dieses Stacks interessiert: Finden Sie ein Beispiel, in welchem die gegebene Quicksort-Implementation $\Omega(n)$ Paare gleichzeitig im Stack speichert. Überlegen Sie dann, wie der Algorithmus abgeändert werden kann, um diesen schmerzhaften Speicherverbrauch deutlich zu senken.

Aus Effizienzgründen bestimmt die vorliegende Implementation zunächst das minimale Element des Eingabearrays und vertauscht es mit dem ersten Element desselben, die verbleibenden Elemente werden dann wie gehabt sortiert.

```
public void quicksort() {
  Stack⟨Pair⟩ stack = new Stack⟨Pair⟩();
  stack.push(new Pair(1, size − 1));
  int min = 0;
  for(int i = 1; i < size; i++)
    if(less(i, min)) min = i;
  D t = get(0); set(0, min); set(min, t);
  while(!stack.isempty()) {
    Pair p = stack.pop();
    int l = p.first(), r = p.second();
    int i = l − 1, j = r, pivot = j; D t;
    do {
      do {i++;} while(less(i, pivot));
      do {j−−;} while(less(pivot, j));
      t = get(i); set(i, get(j)); set(j, t);
    } while(i < j);
    set(j, get(i));
    if(r − i > 1) stack.push(new Pair(i + 1, r));
    if(i − l > 1) stack.push(new Pair(l, i − 1));
  }
}
```

Die Analyse von Quicksort setzt jedoch voraus, daß jede mögliche Permutation der zu sortierenden Schlüssel gleich wahrscheinlich ist. Sind nach der obigen Veränderung der Eingabe alle Permutationen der verbleibenden Elemente immer noch gleich wahrscheinlich?

Aufgabe H15 (15 Punkte)



Betrachten wir die unten beschriebene, parallelisierte Variante von Mergesort. Wir setzen zwei Prozessoren P_1, P_2 ein, um jeweils eine Hälfte der Eingabe (seriell) zu sortieren. Sei zu diesem Zweck n die Länge des Eingabearrays; P_1 sortiere dann den Teil $0 \dots m-1$ und P_2 den Teil $m \dots n-1$ mit $m = \lceil n/2 \rceil$.
Jetzt soll auch das anschließende Mischen dieser nun geordneten Teilarrays parallel stattfinden: dazu bestimme man zwei Zahlen a, b mit $0 \leq a \leq m-1$ und $m \leq b \leq n-1$ so, daß der angegebenen Algorithmus korrekt sortiert und $a+b = n \pm 1$ gilt. Die von Ihnen vorgeschlagene Lösung sollte höchstens $O\big((\log n)^2\big)$ Schritte benötigen.

1: **sort**$(A[0 \dots m-1])$ *on* P_1 || **sort**$(A[0 \dots m-1])$ *on* P_2
2: *Bestimme a und b*
3: $c = a + b - m + 1$
4: $T[0 \dots n-1] := A[0 \dots n-1]$
5: **merge**$(T[0 \dots a], T[m \dots b-1])$ *into* $A[0 \dots c]$ *on* P_1 ||
merge$(T[a+1 \dots m-1], T[b, n-1])$ *into* $A[c \dots m-1]$ *on* P_2

Ergeizige können versuchen, es in nur $O(\log n)$ Schritten zu schaffen.

Aufgabe H16 (5 Punkte)

Ändern Sie den Algorithmus der Vorlesung zum Löschen von Elementen aus einer Skip-Liste so ab, daß er den Head-Knoten, wenn möglich, verkleinert.

Aufgabe H17 (10 Punkte)

Die Funktionen *puzzle* und *riddle* bekommen einen Integer-Wert als Argument, der wie in Aufgabe T19 eine Teilmenge von $\{0, \dots, 30\}$ beschreibt. Suchen Sie sich eine der beiden Funktionen aus! Welche Operation auf Bitsets implementiert diese Funktion? Beschreiben Sie, wie sie funktioniert.

Bedenken Sie bei der Funktion *puzzle*, daß $(5)_{16} = (0101)_2$, $(A)_{16} = (1010)_2$, $(3)_{16} = (0011)_2$, $(C)_{16} = (1100)_2$, $(F)_{16} = (1111)_2$ und $(0)_{16} = (0000)_2$ gilt.

```
int puzzle(int m) {
    m = (m & 0x55555555)
     +((m & 0xAAAAAAAA) >> 1);
    m = (m & 0x33333333)
     +((m & 0xCCCCCCCC) >> 2);
    m = (m & 0x0F0F0F0F)
     +((m & 0xF0F0F0F0) >> 4);
    m = m + (m >> 8);
    m = (m + (m >> 16)) & 31;
    return m;
}
```

```
void riddle(int m) {
    int z = 0;
    int cp =  m + 1;
    do {
        print(z);
        z = (z + cp) & m;
    } while (z ≠ 0);
}

void print(int c) {
    String s = "";
    for (int i = 0; i ≤ 31; ++i) {
        s = (c & 1) + s;
        c = c >> 1;
    }
    System.out.println(s);
}
```