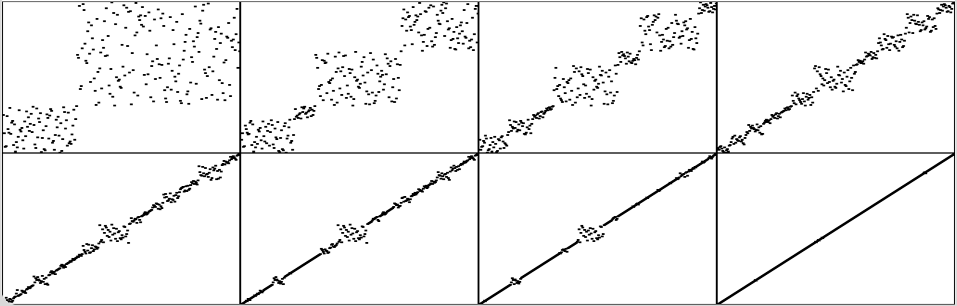
Karlsruher Institut für Technologie

Tutorium 6: Heaps und Sortieren

Holger Ebhart | 27. Mai 2015

TUTORIUM ZUR VORLESUNG ALGORITHMEN I IM SS15



Gliederung

Karlsruher Institut für Technologie

5. Übungsblatt

5. Übungsblatt

Duplikaterkennung:

procedure *duplicates*(A : Array $[0..n-1]$ **of** $\mathbb{N}$, $x \in \mathbb{N}$)
    for $i = 0$ **to** $n - 1$ **do**
        while $A[i] \neq i$
            if $A[A[i]] == A[i]$ **then** print “Duplikat gefunden:”, A[i]; **return**
            else swap(A, i, $A[i]$)
    print “keine Duplikate”
return

Heaps

Speichere Werte in sortierter Reihenfolge in einem (binären) Baum

- Min- und Max-Heaps möglich
- Baum hat Höhe $\lfloor \log n \rfloor$
- insert und delete in $\mathcal{O}(logn)$
- min in $\mathcal{O}(1)$
- Invariante: $\forall e : parent(e) \leq e$
- speichern als Array (unbounded?)
- als binärer Baum:
  - parent(i) $\rightarrow \lfloor \frac{i}{2} \rfloor$
  - child(i) $\rightarrow 2i$ bzw. $2i + 1$

Heaps

Speichere Werte in sortierter Reihenfolge in einem (binären) Baum

- Min- und Max-Heaps möglich
- Baum hat Höhe $\lfloor \log n \rfloor$
- insert und delete in $\mathcal{O}(logn)$
- min in $\mathcal{O}(1)$
- Invariante: $\forall e : parent(e) \leq e$
- speichern als Array (unbounded?)
- als binärer Baum:
  - parent(i) $\rightarrow \lfloor \frac{i}{2} \rfloor$
  - child(i) $\rightarrow 2i$ bzw. $2i + 1$

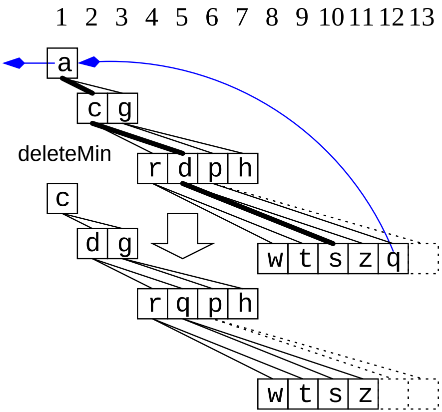
Implementierung

- insert(e)
  1. füge e unten rechts / hinten ein
  2. schiebe e durch Vertauschen maximal hoch (siftUp)
- deleteMin()
  1. lösche Wurzel
  2. ziehe letztes Element des Heaps nach oben
  3. Vertausche nun evtl. den parent mit dem kleinsten child
  4. führe dies für den jeweiligen Teilbaum aus, bis sich nichts mehr ändert (siftDown)

Beispiel

Sanders: Algorithmen I June 30, 2014 232

deleteMin: Beispiel



Aufgaben

Baue aus folgenden Elementen einen Heap:
4, 5, 8, 1, 2, 9, 6, 3, 7, 12, 13, 11
Stelle den Heap sowohl in Baum, als auch in Array-Form dar.
Wie sieht der Heap als Baum und im Array aus?

füge 0 ein

führe nun 2 mal deleteMin aus

Aufgaben

Baue aus folgenden Elementen einen Heap:
4, 5, 8, 1, 2, 9, 6, 3, 7, 12, 13, 11
Stelle den Heap sowohl in Baum, als auch in Array-Form dar.
Wie sieht der Heap als Baum und im Array aus?

füge 0 ein

führe nun 2 mal deleteMin aus

Aufgaben

Baue aus folgenden Elementen einen Heap:
4, 5, 8, 1, 2, 9, 6, 3, 7, 12, 13, 11
Stelle den Heap sowohl in Baum, als auch in Array-Form dar.
Wie sieht der Heap als Baum und im Array aus?

füge 0 ein

führe nun 2 mal deleteMin aus

Heapsort

Idee:
Füge alle Elemente in einen Heap ein
entnehme stets das kleinste Element

Laufzeit:
Average-Case: $\mathcal{O}(n \cdot \log n)$
Worst-Case: $\mathcal{O}(n \cdot \log n)$
Best-Case: $\mathcal{O}(n \cdot \log n)$
Algorithmus:

```
procedure heapSort(A:Array[1...n] of Digit)
        H := buildHeap(A):Heap
        for i := 1 to n do
                A[i] := H.deleteMin()
        end
        return A;
```

Heapsort

Idee:
Füge alle Elemente in einen Heap ein
entnehme stets das kleinste Element

Laufzeit:
Average-Case: $\mathcal{O}(n \cdot \log n)$
Worst-Case: $\mathcal{O}(n \cdot \log n)$
Best-Case: $\mathcal{O}(n \cdot \log n)$
Algorithmus:

```
procedure heapSort(A:Array[1...n] of Digit)
        H := buildHeap(A):Heap
        for i := 1 to n do
                A[i] := H.deleteMin()
        end
        return A;
```

Ganzzahliges Sortieren - Voraussetzungen

- spezielle Schlüsselfunktion:
- $key : Element \rightarrow \mathbb{N}_0$
- **UND** $\forall e \in Elements : \exists M \in \mathbb{N}_0 : key(e) \leq M$

Ganzzahliges Sortieren - Voraussetzungen

- spezielle Schlüsselfunktion:
- $key : Element \rightarrow \mathbb{N}_0$
- **UND** $\forall e \in Elements : \exists M \in \mathbb{N}_0 : key(e) \leq M$

Ganzzahliges Sortieren - Voraussetzungen

- spezielle Schlüsselfunktion:
- $key : Element \rightarrow \mathbb{N}_0$
- **UND** $\forall e \in Elements : \exists M \in \mathbb{N}_0 : key(e) \leq M$

Bucketsort / K-Sort

- Voraussetzung: $\max_{e \in Elements}(key(e)) = M \in \mathbb{N}_0$
- allokiere Array A mit M+1 Zeigern auf leere Liste
- füge jedes Element e an Stelle $A[key(e)]$ mit $pushFront(e)$ ein
- Laufzeit: $\mathcal{O}(n + M)$
- Ist diese Implementierung stabil? Falls ja, begründen Sie warum, andernfalls geben Sie eine stabile Implementierung an.

Bucketsort / K-Sort

- Voraussetzung: $\max_{e \in Elements}(key(e)) = M \in \mathbb{N}_0$
- allokiere Array A mit M+1 Zeigern auf leere Liste
- füge jedes Element e an Stelle $A[key(e)]$ mit $pushFront(e)$ ein
- Laufzeit: $\mathcal{O}(n + M)$
- Ist diese Implementierung stabil? Falls ja, begründen Sie warum, andernfalls geben Sie eine stabile Implementierung an.

Bucketsort / K-Sort

Karlsruher Institut für Technologie

- Voraussetzung: $\max\limits_{e \in Elements}(key(e)) = M \in \mathbb{N}_0$
- allokiere Array A mit M+1 Zeigern auf leere Liste
- füge jedes Element e an Stelle $A[key(e)]$ mit $pushFront(e)$ ein
- Laufzeit: $\mathcal{O}(n + M)$
- Ist diese Implementierung stabil? Falls ja, begründen Sie warum, andernfalls geben Sie eine stabile Implementierung an.

Radix-Sort

Voraussetzung: $\forall e \in \mathit{Elements} : \mathit{key}(e) \in \mathbb{N}_0 \wedge \mathit{key}(e) \leq M \in \mathbb{N}_0$
Sei d die Anzahl der Stellen von M in der k-ären Darstellung ($k \geq 2$).

Idee:
Sortiere d mal alle Elemente mit einem stabilen Bucketsort um nach der jeweils i-ten Stelle in key(e) zu sortieren. Beginne mit der kleinsten Stelle (LSB).
Algorithmus:

```
procedure radixSort(A:Array[1...n] of Digit)
        for i: = 0 to d-1 do
                define key(e) as (e div k^i) mod k
                bucketSort(A, k)
```

Laufzeit:
Best-Case / Average-Case / Worst-Case: $\mathcal{O}(d \cdot (n + k))$

Aufgabe 1

Es sei die folgende Zahlenfolge gegeben:
$Z_1 = (6, 88, 7, 33, 56, 1, 14, 16, 29)$
Sortieren sie Z_1 mit

- Quicksort ($Dual - Pivot : p_1 = Z[1], p_2 = Z[2]$)
- Heapsort
- Radixsort

Aufgabe 2

Sortiere folgende Mengen Z_i mit dem angegebenen Algorithmus:

a) $Z_1 = (7, 6, 5, 4, 3, 2, 1)$ Heapsort

b) $Z_2 = (7, 4, 9, 1, 5, 9, 3, 0, 5, 2, 7, 3, 8, 2, 1, 3, 4, 9, 6, 3, 7, 9, 1, 0)$ Bucketsort

b) $Z_3 = (111, 76, 223, 567, 349, 496, 201, 872, 3)$ Radixsort

Aufgabe 3

Gegeben seien k doppelt-verkettete sortierte Listen $L_1, \cdots, L_k$ jeweils der Länge $\frac{n}{k}$

a) Geben Sie einen Algorithmus an, der in Zeit $\Theta(nk)$ eine sortierte Liste L erzeugt, die genau die Elemente der k sortierten Listen $L_1, \cdots, L_k$ enthält. Begründen Sie die Laufzeit.

b) Geben Sie einen Algorithmus an, der in Zeit $\mathcal{O}(n \log k)$ eine sortierte Liste L erzeugt, die genau die Elemente der k sortierten Listen $L_1, \cdots, L_k$ enthält. Begründen Sie die Laufzeit.

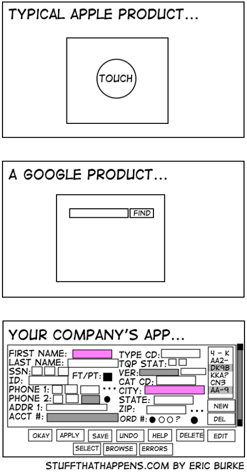
Aufgabe 3

Gegeben seien k doppelt-verkettete sortierte Listen $L_1, \cdots, L_k$ jeweils der Länge $\frac{n}{k}$

a) Geben Sie einen Algorithmus an, der in Zeit $\Theta(nk)$ eine sortierte Liste L erzeugt, die genau die Elemente der k sortierten Listen $L_1, \cdots, L_k$ enthält. Begründen Sie die Laufzeit.

b) Geben Sie einen Algorithmus an, der in Zeit $\mathcal{O}(n \log k)$ eine sortierte Liste L erzeugt, die genau die Elemente der k sortierten Listen $L_1, \cdots, L_k$ enthält. Begründen Sie die Laufzeit.

Vielen Dank für eure Aufmerksamkeit!
Bis zum nächsten Mal.



stackoverflow.com