

1 O-Notation

$$O(f) := f \mid \exists c, n_0 \in \mathbb{N}, c > 0, n_0 > 0, \forall n \in \mathbb{N}, n > n_0 : g(n) \leq c * f(n)$$

g wächst nicht schneller als f

$$\Omega(f) := g \mid \exists c, n_0 \in \mathbb{N}, c > 0, n_0 > 0, \forall n \in \mathbb{N}, n > n_0 : g(n) \geq c * f(n)$$

g wächst mindestens so schnell wie f

$$\Theta(f) = O(f) \cap \Omega(f)$$

f und g von gleicher Größenordnung

$g(n)$: Funktion in Langform c : Anzahl Vergleiche n_0 : ab welchem n gilt es

Master-Theorem

$$t(n) = a * t\left(\frac{n}{b}\right) + f(n)$$

$t(n) \in \Theta(f(n))$ falls $f(n) \in \Omega(n^{\log_b a + \epsilon})$ und $\epsilon > 0$ und $a * f\left(\frac{n}{b}\right) \leq c * f(n)$

$$t(n) \in \Theta(n^{\log_b a}) \text{ falls } f(n) \in O(n^{\log_b a - \epsilon})$$

$$f(n) \in \Theta(n^{\log_b a} \log n) \text{ falls } f(n) \in \Theta(n^{\log_b a})$$

a: Anzahl rekursiver Aufrufe der Funktion
n: Anzahl Element b: Um wie viel wird reduziert
f(n): Komplexität vom Rest

2 Bäume

Preorder: Wurzel $\rightarrow$ Preorder von linken Knoten

Postorder: Postorder der Knoten $\rightarrow$ Wurzel

Inorder: Inorder vom linken Knoten $\rightarrow$ Wurzel $\rightarrow$ Inorder Rechter Knoten

2.1 Huffman Codes

Tabelle mit Symbolen und Häufigkeiten while(2 kleinste Wahrscheinlichkeit markieren; neuen Knoten mit Summenwahrscheinlichkeit; bis nur einer unmarkiert ist)

AVL-Baum: → sortierte Binärbäume; **2-3 Bäume:** innerer Knoten enthält minimum der Kinder; **B-Baum:** 3 in Wurzel mit jeweils einem Kind

3 Sortieren

BubbleSort: Aufsteigen

Shakersort: gemischer Bubblesort

Shellsort: Einfügen mit abnehmender Schrittweite

Heapsort: Heap geblockt aufbauen...

Quicksort: rekursiv, < kleiner, > größer, vom Pivot aus

Mergesort: rekursiv auseinander nehmen, und passend zusammensetzen

4 Indexieren

Hashing, Schlüssel mit Referenz, Schlüssel mit Block Referenz

5 Graphen

5.1 Dijkstra Algorithmus

Adjazenztablette aufstellen, kürzeren Umweg suchen und Länge in Tabelle ersetzen.

5.2 Suche

Tiefensuche

bei Besuch markieren

Vorgänger nie besucht: **T**; Vorgänger früher besucht: **F**; Vorgänger später besucht: **B**;
Vorgänger gefinished: **C**; Tree, Forward, Back, Cross

Breitensuche Erst direkte Nachfolger, dann deren Nachfolger

Hat ein Graph den Zusammenhang m , hat jeder Knoten min m Kanten.

6 Algorithmenentwurf

divide and conquer in kleinere Teilprobleme aufteilen

Dynamische Programmierung Lösung von Teilproblemen in Tabelle notieren

Greedy Verfahren gierig: beste Lösung als nächsten Schritt, "größtmögliche Münze"

Backtracking ausprobieren und zurück wenn Ergebnis nicht passt.