

Infos

Telegram-Gruppe

(<https://t.me/joinchat/BOnvBRAj4DSmBVteVVR5cA>)

1 Komplexitätsanalyse

1.1 Effizienzmaße

- Worst Case: $t(n) = \max\{T(i) : i \in I_n\}$
- Average Case:
 - Gleichverteilt:
$$t(n) = \frac{1}{|I_n|} \sum_{i \in I_n} T(i)$$
 - Nicht gleichverteilt:
$$t(n) = \sum_{i \in I_n} p_i \cdot T(i)$$
- Best Case: $t(n) = \min\{T(i) : i \in I_n\}$

1.2 Formalisierung asymptotischen Verhaltens

- $\mathcal{O}(f(n)) = \{g(n) : \exists c > 0 : \exists n_0 > 0 : \forall n \geq n_0 : g(n) \leq c \cdot f(n)\}$
- $\Omega(f(n)) = \{g(n) : \exists c > 0 : \exists n_0 > 0 : \forall n \geq n_0 : g(n) \geq c \cdot f(n)\}$
- $\Theta(f(n)) = \mathcal{O}(f(n)) \cap \Omega(f(n))$
- $\mathcal{o}(f(n)) = \{g(n) : \forall c > 0 : \exists n_0 > 0 : \forall n \geq n_0 : g(n) \leq c \cdot f(n)\}$
- $\omega(f(n)) = \{g(n) : \forall c > 0 : \exists n_0 > 0 : \forall n \geq n_0 : g(n) \geq c \cdot f(n)\}$

1.3 Rechenregeln für \mathcal{O} -Notation

- Polynome k -ten Grades $\in \Theta(n^k)$
- Für Funktionen $f(n)$ bzw. $g(n)$ mit $\exists n_0 \forall n \geq n_0 : f(n) > 0$:
 - $c \cdot f(n) \in \Theta(f(n))$
 - $\mathcal{O}(f(n)) + \mathcal{O}(g(n)) = \mathcal{O}(f(n) + g(n))$
 - $\mathcal{O}(f(n)) \cdot \mathcal{O}(g(n)) = \mathcal{O}(f(n) \cdot g(n))$
 - $\mathcal{O}(f(n) + g(n)) = \mathcal{O}(f(n))$, falls $g(n) \in \mathcal{O}(f(n))$

- $\Omega(f(n)) + \Omega(g(n)) = \Omega(f(n) + g(n))$
- $\Omega(f(n)) \cdot \Omega(g(n)) = \Omega(f(n) \cdot g(n))$
- $\Omega(f(n) + g(n)) = \Omega(f(n))$, falls $g(n) \in \mathcal{O}(f(n))$
- Falls f, g differenzierbar, X eines der fünf Landau-Symbole:
 $f'(n) \in X(g'(n)) \Rightarrow f(n) \in X(g(n))$

2 Datenstrukturen

2.1 Dynamische Felder

- Parameter:
 - $\beta = 2$: Wachstumsfaktor
 - $\alpha = 4$: max. Speicheroverhead
 - $w = 1$: momentane Feldgröße
 - $n = 0$: momentane Elementanzahl
 - $b = \text{new } X[w]$: statisches Feld
- Methoden:
 - `get (int) : X` $\in \mathcal{O}(1)$
 - `set (int, X) : void` $\in \mathcal{O}(1)$
 - `size () : int` $\in \mathcal{O}(1)$
 - `pushBack (X x) : void` $\in (\text{reallocate} ? \mathcal{O}(\text{reallocate}) : \mathcal{O}(1))$
 - `popBack () : void` (Laufzeiten wie `pushBack`)
 - `reallocate (int) : void` $\in \mathcal{O}(n)$, amortisiert $\in \mathcal{O}(1)$

2.2 Doppelt verkettete Listen

2.3 Stacks und Queues

2.4 Binäre Heaps

- `-siftDown(int) : void` $\in \mathcal{O}(\log n)$
- `-siftUp(int) : void` $\in \mathcal{O}(\log n)$
- `+deleteMin() : X` $\in \mathcal{O}(\text{siftDown})$

- $+\text{min}() : X \in \mathcal{O}(1)$
- $+\text{insert}(X) : \text{void} \in \mathcal{O}(\text{siftUp})$
- $+\text{build}(x_1, \dots, x_n) : \text{void} \in \mathcal{O}(n)$
- $+\text{merge}(B_2) : \text{void} \in \Theta(n)$

2.5 Binomiale Heaps

Binomialbäume: gleiche Operationen und Laufzeiten wie Binäre Heaps.
 Weitere Operationen und Ausnahmen:

- $\text{decreaseKey} \in \mathcal{O}(\log n)$
- $\text{remove} \in \mathcal{O}(\log n)$
- $\text{merge} \in \mathcal{O}(\log n)$

2.6 Fibonacci Heaps

- $\text{min}, \text{insert}, \text{merge} : \mathcal{O}(1)$
- $\text{decreaseKey} : \mathcal{O}(1)$ (amortisiert)
- $\text{deleteMin}, \text{remove} : \mathcal{O}(\log n)$ (amortisiert)

3 Hashing

c -universelles Hashing:

Eine Familie H von Hashfunktionen auf $\{0, \dots, m-1\}$ heißt c -universell, falls für jedes Paar $x \neq y$ von Schlüsseln gilt:

$$|\{h \in H : h(x) = h(y)\}| \leq \frac{c}{m} |H|.$$

- c : Kollisionsmaß
- m : Größe der Hashtabelle
- n : Anzahl Elemente in der HT
- Erwartete Laufzeit remove/find dann in $\mathcal{O}(1 + c \cdot \frac{n}{m})$.

Mit m prim und $h_a(x) = \langle a, x \rangle \bmod m$ ist $H = \{h_a : a \in \{0, \dots, m-1\}^k\}$ 1-universell.

Mit m prim und $h_a(x) = \sum_{i=1}^k a^{i-1} x_i \bmod m$ ist $H = \{h_a : a \in \{1, \dots, m-1\}\}$ k -universell.

4 Sortieren

4.1 Master-Theorem

Seien a, b, c, d positive Konstanten und $n = b^k$ mit $k \in \mathbb{N}$.

Sei $r(n) =$

- a , falls $n = 1$,
- $cn + d \cdot r(\frac{n}{b})$, falls $n > 1$.

Dann gilt: $r(n) \in$

- $\Theta(n)$, falls $d < b$,
- $\Theta(n \log n)$, falls $d = b$,
- $\Theta(n^{\log_b d})$, falls $d > b$.

4.2 Verfahren

Algorithmus	Laufzeit			Eigenschaften
	Best Case	Average Case	Worst Case	
Selection Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	Einfach zu implementieren
Bubble Sort	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	Einfach zu implementieren
Heap Sort	$\mathcal{O}(n)$	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$	Var. d. Selection Sort, bessere Minimumstrategie
Insertion Sort	$\mathcal{O}(n)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	Einfach zu implementieren
Bogo Sort	$\mathcal{O}(n)$	$\mathcal{O}(n \cdot n!)$	$\mathcal{O}(\infty)$	Einfach zu implementieren
Shell Sort	$\mathcal{O}(n)$	$\mathcal{O}(n \log^2 n)$	$\mathcal{O}(n \log^2 n)$	Variante d. Insertion Sort, bessere Einfügestrategie
Merge Sort	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$	Sehr gut für Multi-Threading geeignet
Quick Sort	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$	$\mathcal{O}(n^2)$	Schnellstes Verfahren in der Praxis
Radix Sort	$\mathcal{O}(d \cdot (n + k))$	$\mathcal{O}(d \cdot (n + k))$	$\mathcal{O}(d \cdot (n + k))$	d Stellen, k mögl. 'Ziffern' pro Stelle

Stabil

In-Place

Stabil und In-Place

- 5 Priority Queues
- 6 Suchstrukturen
- 7 Graphen
- 8 Pattern Matching
- 9 Datenkompression
- 10 Algorithmen

- 10.1 Bellman-Ford

Löst SSSP mit beliebigen Kantengewichten. Ziel: Kürzeste Distanzen zu allen Knoten (von einem Startknoten aus gesehen), Erkennung von negativen Kreisen.

- S