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

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

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
  - $b = new\ X[w]$: 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 $\in$ (reallocate ? $\mathcal{O}(reallocate)$: $\mathcal{O}(1)$)
  - popBack () : void (Laufzeiten wie pushBack)
  - reallocate (int) : void $\in \mathcal{O}(n)$, amortalisiert $\in \mathcal{O}(1)$

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 $\in \mathcal{O}(siftDown)$

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

2.5 Binomiale Heaps

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

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

2.6 Fibonacci Heaps

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

3 Hashing

c-universelles Hashing:
Eine Familie H von Hashfunktionen auf $\{0, ..., m-1\}$ 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 $h_a(x) = \langle a, x \rangle\ mod\ m$ ist $H = \{h_a : a \in \{0, ..., m-1\}^k\}$ 1-universell.
Mit m prim und $h_a(x) = \sum_{i=1}^{k} a^{i-1} x_i\ mod\ m$ ist $H = \{h_a : a \in \{1, ..., m-1\}\}$ 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 SSSP-Algorithmen

10.1.1 Topologische Sortierung

Löst SSSP auf DAGs mit beliebigen Kantengewichten.
Angenommen, unser Graph hat n Knoten und m Kanten.
Laufzeit: $\mathcal{O}(m+n)$

1. Wiederhole, bis alle Knoten abgearbeitet:
   - Wähle beliebigen Knoten, der keine oder nur bereits abgearbeitete Vorgänger hat.
   - Überprüfe, ob er eingetragene Distanzen "verbessern"kann und aktualisiere ggf. Distanz und Vorgänger des jeweiligen Nachfolgers.
   - Dieser Knoten ist nun abgearbeitet.

10.1.2 Dijkstra

Löst SSSP mit positiven Kantengewichten (Graph muss kein DAG sein). Ziel: Kürzeste Distanzen zu allen Knoten (von einem Startknoten aus gesehen).
Angenommen, unser Graph hat n Knoten und m Kanten.
Laufzeit: $\mathcal{O}(m + n\ log\ n)$, mit Radix Heaps über C-adische Darstellung bis zu $\mathcal{O}(m + n\ log\ C)$ zu verbessern

1. Setze alle Distanzen auf ∞ und füge $(s, 0)$ in die Queue ein
2. Wiederhole, solange die Queue nicht leer ist:

(a) Wähle den Knoten mit geringster Distanz (x) aus der Queue, der kürzeste Weg zu x ist nun gefunden und x ist jetzt abgearbeitet

(b) Berechne Distanzen zu allen noch nicht abgearbeiteten Nachfolgern von x und füge sie in die Queue ein bzw. aktualisiere ggf. den neuen kürzeren Weg

10.1.3 Bellman-Ford

Löst SSSP mit beliebigen Kantengewichten (Graph muss kein DAG sein).
Ziel: Kürzeste Distanzen zu allen Knoten (von einem Startknoten aus gesehen), Erkennung von negativen Kreisen.
Angenommen, unser Graph hat n Knoten und m Kanten.
Laufzeit: $\mathcal{O}(m \cdot n)$.
Parameter: Startknoten s.

1. Setze alle Distanzen auf ∞.
2. Setze Distanz des Knotens s auf 0.
3. Wiederhole $n - 1$ mal:
   (a) Gehe alle Kanten $e = (v, w)$ aus der Kantenmenge durch, dabei:
      i. Ist die Distanz des Knotens v plus die Kantenkosten weniger als die **aktuelle** Distanz des Knotens w? Falls ja, *aktualisiere* Distanz des Knotens w sowie seinen Vorgänger.
4. Gehe nun noch ein letztes Mal alle Kanten $e = (v, w)$ durch, dabei:
   (a) Ist die Distanz des Knotens v plus die Kantenkosten weniger als die **aktuelle** Distanz des Knotens w? Falls ja, infiziere w und setze seinen Vorgänger auf v.

Infizieren eines Knotens x: Setze Distanz des Knotens x auf $-\infty$, alle Nachfolger des Knotens werden sukzessive ebenfalls infiziert.
Anmerkung: Bis auf den Wert der Distanz sind Schritte 3.(a) und 4 *identisch*.