https://matse.paddel.xyz/spicker

Algorithmen

Patrick Gustav Blaneck

Letzte Änderung: 13. Juni 2021

Inhaltsverzeichnis

1 Grundbegriffe

Definition: Eigenschaften eines Algorithmus

- *Terminierung:* Der Algorithmus bricht nach *endlichen vielen* Schritten ab.
- *Determiniertheit:* Bei vorgegebener Eingabe wird ein eindeutiges *Ergebnis* geliefert.
- *Determinismus:* Eindeutige Vorgabe der *Abfolge* der auszuführenden Schritte

1.1 Algorithmische Komplexität

Definition: Landau-Notation

Seien f, g reellwertige Funktionen der reellen Zahlen. Dann gilt: [**wiki:Landau-Symbole**]

Notation	Definition	Mathematische Definition
$f \in \mathcal{O}(g)$	obere Schranke	$\exists C > 0 \exists x_0 > 0 \forall x > x_0 : \lvert f(x)\rvert \leq C \cdot \lvert g(x)\rvert$
$f \in \Omega(g)$	untere Schranke	$\exists c > 0 \exists x_0 > 0 \forall x > x_0 : c \cdot \lvert g(x)\rvert \leq \lvert f(x)\rvert$
$f \in \Theta(g)$	scharfe Schranke	$\exists c > 0 \exists C > 0 \exists x_0 > 0 \forall x > x_0 : c \cdot \lvert g(x)\rvert \leq \lvert f(x)\rvert \leq C \cdot \lvert g(x)\rvert$

Anschaulicher gilt:

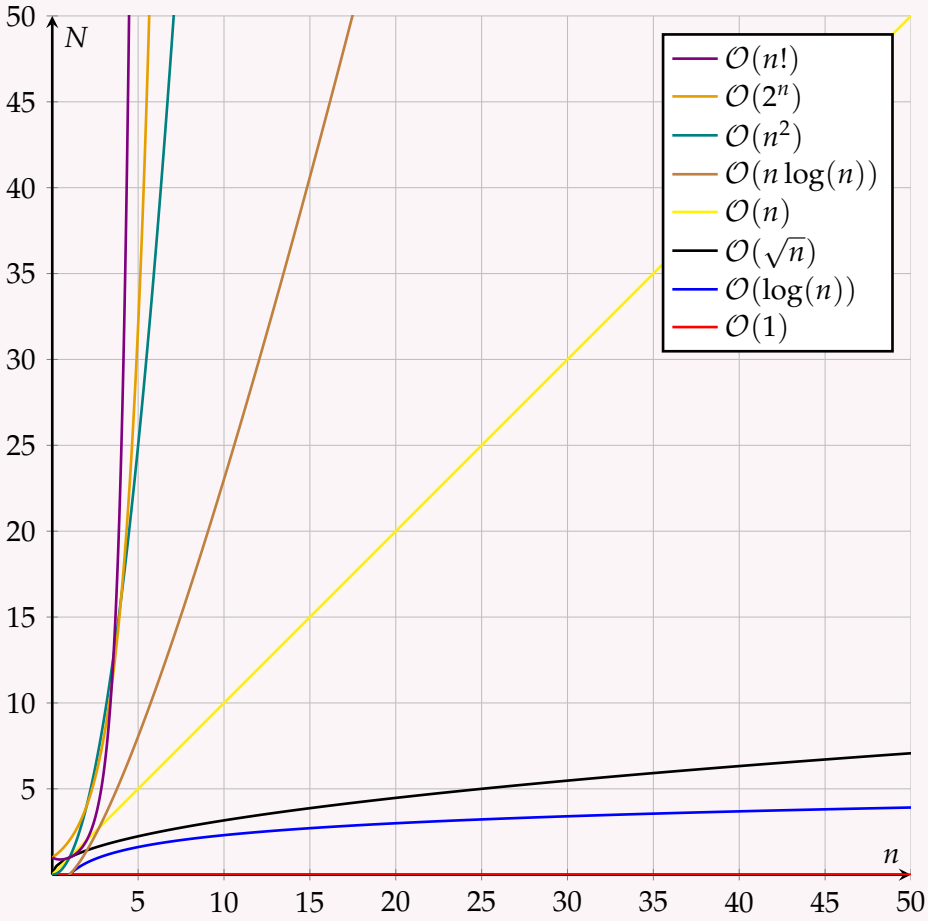
Notation	Anschauliche Bedeutung
$f \in \mathcal{O}(g)$	f wächst nicht wesentlich schneller als g
$f \in \Omega(g)$	f wächst nicht wesentlich langsamer als g
$f \in \Theta(g)$	f wächst genauso schnell wie g

Beispiel: Landau-Notation

Aus [**wiki:Landau-Symbole**] :

Notation	Beispiel
$f \in \mathcal{O}(1)$	Feststellen, ob eine Binärzahl gerade ist
$f \in \mathcal{O}(\log n)$	Binäre Suche im sortierten Feld mit n Einträgen
$f \in \mathcal{O}(\sqrt{n})$	Anzahl der Divisionen des naiven Primzahltests
$f \in \mathcal{O}(n)$	Suche im unsortierten Feld mit n Einträgen
$f \in \mathcal{O}(n \log n)$	Mergesort, Heapsort
$f \in \mathcal{O}(n^2)$	Selectionsort
$f \in \mathcal{O}(n^m)$	
$f \in \mathcal{O}(2^{cn})$	(Backtracking)
$f \in \mathcal{O}(n!)$	Traveling Salesman Problem

Bonus: Visualisierung Komplexitätsklassen



2 Elementare Datenstrukturen

Definition: Homogene Datenstruktur

In einer *homogenen Datenstruktur* haben alle Komponenten den *gleichen* Datentyp.

Definition: Heterogene Datenstruktur

In einer *heterogenen Datenstruktur* haben die Komponenten *unterschiedliche* Datentypen.

Definition: Abstrakte Datentupen (ADTs)

Anforderungen an die Definition eines Datentyps:

- *Spezifikation* eines Datentyps unabhängig von der Implementierung
- Reduzierung der von außen sichtbaren Aspekte auf die *Schnittstelle* des Datentyps

Daraus entstehen **zwei Prinzipien**:

- *Kapselung:*
  Zu einem ADT gehört eine Schnittstelle.
  Zugriffe auf den ADT erfolgen ausschließlich über die Schnittstelle.
- *Geheimnisprinzip:*
  Interne Realisierung eines ADT-Moduls bleibt verborgen.

Beispiel: ADT in Java

Viele wichtige abstrakte Datentypen werden in Java durch *Interfaces* beschrieben.

Es gibt ein oder mehrere Implementierungen dieser Interfaces mit unterschiedlichen dahinter stehenden Konzepten.

In Java: Package `java.util`

Wichtig in der Vorlesung:

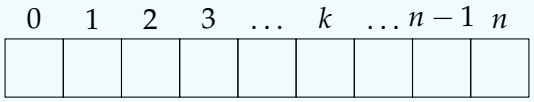
ADT	Grund-ADT/Interface	Java-Klassen
Feld		(Felder), HashMap
Liste	List	ArrayList, LinkedList
Menge	Set	HashSet, TreeSet
Prioritätswarteschlange		PriorityQueue
Stack	List	
Queue	List	
Deque	List	Deque (Interface), ArrayDeque
Map	Set	Map (Interface), HashMap, TreeMap
BidiMap	Map	BidiMap, BiMap (Interface)
MultiSet, Bag	Map	Bag, Multiset (Interface)

2.1 Felder und abstrakte Datenstrukturen

Definition: Array

Ein *Array* hat folgende spezielle Eigenschaften:

- Feste Anzahl an Datenobjekten
- Auf jedes Objekt kann direkt lesend oder schreibend zugegriffen werden



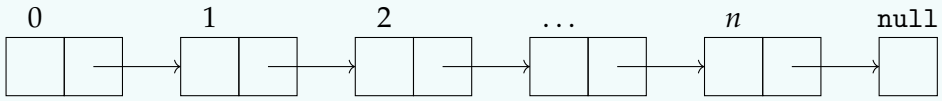
Performance:

Zugriff	Suche	Einf./Lösch. (Anfang)	Einf./Lösch. (Ende)	Einf./Lösch. (Mitte)
$\Theta(1)$	$\Theta(n)$	-	-	-

Definition: Liste (Einfach verkettete)

Für eine *Liste* kommt im Vergleich zu einem Array hinzu:

- Eine Liste kann wachsen und schrumpfen.



Performance:

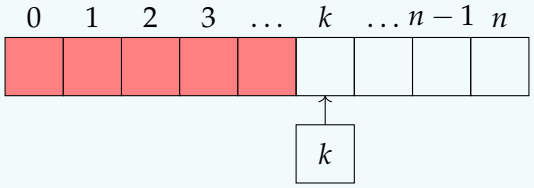
Zugriff	Suche	Einf./Lösch. (Anfang)	Einf./Lösch. (Ende)	Einf./Lösch. (Mitte)
$\Theta(n)$	$\Theta(n)$	$\Theta(1)$	$\Theta(1)/\Theta(n)$[a]	Suchzeit + $\Theta(1)$

[a] $\Theta(1)$, wenn das letzte Element bekannt ist, $\Theta(n)$ sonst

Definition: Dynamisches Feld

Ein *dynamisches Feld* besteht aus:

- Einem normalen Feld, das nicht vollständig gefüllt ist.
- Einem Zeiger, der anzeigt, welches das erste unbesetzte Element ist.



Performance:

Zugriff	Suche	Einf./Lösch. (Anfang)	Einf./Lösch. (Ende)	Einf./Lösch. (Mitte)
$\Theta(1)$	$\Theta(n)$	$\Theta(n)$	$\Theta(1)/\Theta(n)$[a]	$\Theta(n)$

Implementierung:

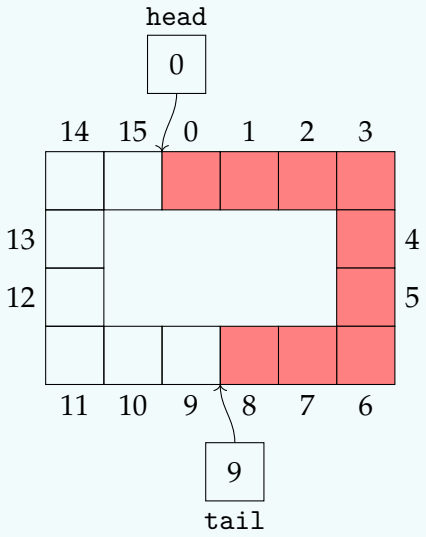
- Ein dynamisches Feld ist für einen Stack gut geeignet:
  Einfügen am Ende: $\Theta(1)$ (aber: Worst-Case $\Theta(n)$!)
  Auslesen am Ende: $\Theta(1)$

[a] Wenn das Feld schon voll ist, muss der komplette Inhalt kopiert werden.

Definition: Zirkuläres (dynamisches) Feld

Ein *zirkuläres Feld* besitzt einen Speicher fester Größe. Dabei speichern zwei Zeiger jeweils den Anfang (`head`) des Speichers, bzw. auf die nächste freie Speicheradresse (`tail`) im Speicher.

Wird ein Element am Anfang „abgearbeitet", bewegt sich `head` eine Position weiter. Wird ein Element am Ende eingefügt, bewegt sich `tail` eine Position weiter.



Performance: (dynamisch, bei unterliegender Datenstruktur Array)

Zugriff	Suche	Einf./Lösch. (Anfang)	Einf./Lösch. (Ende)	Einf./Lösch. (Mitte)
$\Theta(1)$	$\Theta(n)$	$\Theta(1)/\Theta(n)$[a]	$\Theta(1)/\Theta(n)$[b]	$\Theta(n)$

Implementierung:

- Ein zirkuläres (dynamisches) Feld ist für eine Queue/Deque gut geeignet.

[a] Wenn das Feld schon voll ist, muss der komplette Inhalt kopiert werden.
[b] Wenn das Feld schon voll ist, muss der komplette Inhalt kopiert werden.

Definition: Menge

Eine *Menge (Set)* ist eine Sammlung von Elementen des gleichen Datentyps. Innerhalb der Menge sind die Elemente ungeordnet. Jedes Element kann nur einmal in der Menge vorkommen.

Implementierung:

In Java ist *Set* ein Interface, das unter anderem folgende Klassen implementiert:

- `TreeSet`: Basiert auf der Datenstruktur Rot-Schwarz-Baum, implementiert Erweiterung `SortedMap`.
- `HashSet`: Basiert auf der Datenstruktur Hashtabelle.

Definition: Assoziatives Feld

Ein *assoziatives Feld* ist eine Sonderform des Feldes:

- Verwendet keinen numerischen Index zur Adressierung eines Elements.
- Verwendet zur Adressierung einen Schlüssel (z.B. `a["Meier"]`).

Assoziative Felder eignen sich dazu, Datenelemente in einer großen Datenmenge aufzufinden. Jedes Datenelement wird durch einen *eindeutigen Schlüssel* identifiziert.

Implementierung: In Java entspricht ein assoziatives Feld dem Interface `java.util.Map`, das folgende Klassen implementiert:

- `TreeMap`: Basiert auf der Datenstruktur Rot-Schwarz-Baum, implementiert Erweiterung `SortedMap`.
- `HashMap`: Basiert auf der Datenstruktur Hashtabelle.

Index

Beispiele