Algorithmen I

Tutorium 1 - 3. Sitzung

Dennis Felsing

dennis.felsing@student.kit.edu
www.stud.uni-karlsruhe.de/~ubcqr/algo

2011-05-02

Überblick

Sortieren und Suchen

Übungsblatt 1 - Aufgabe 1

Aufgabe

(a) Zeigen Sie, dass $f(n) = n!$ in $O(n^n)$ liegt.
(b) Zeigen Sie, dass $f(n)$ sogar in $o(n^n)$ liegt.

Eigenschaften von Sortieralgorithmen

Laufzeiten für Worst Case, Average Case und Best Case

in-place Kein zusätzlicher Speicherplatz notwendig

stabil Erhält Reihenfolge der Sortierschlüssel

Bubblesort

Grundlegender Ablauf

Vergleiche nebeneinander liegende Elemente und vertausche diese, wenn notwendig. Wiederholen bis bei einem Durchlauf keine Vertauschungen mehr auftreten.

Eigenschaften

Worst Case:	$O(n^2)$
Average Case:	$O(n^2)$
Best Case:	$O(n)$
inplace:	Ja
stabil:	Ja

Insertionsort

Grundlegender Ablauf

Schrittweise nächsten Wert an passender Stelle in sortierte Sequenz einfügen.

Eigenschaften

Worst Case:	$O(n^2)$
Average Case:	$O(n^2)$
Best Case:	$O(n)$
inplace:	Ja
stabil:	Ja

Selectionsort

Grundlegender Ablauf

Wähle Minimum aus unsortiertem Rest aus und vertausche mit erstem unsortiertem Element.

Eigenschaften

Worst Case:	$O(n^2)$
Average Case:	$O(n^2)$
Best Case:	$O(n^2)$
inplace:	Ja
stabil:	Nein

Mergesort

Grundlegender Ablauf

1. Teile die Zahlenfolge in zwei Hälften
2. Sortiere die beiden Hälften rekursiv
3. Füge die beiden Hälften zusammen

Eigenschaften

Worst Case:	$O(n \cdot \log n)$
Average Case:	$O(n \cdot \log n)$
Best Case:	$O(n \cdot \log n)$
inplace:	Nein
stabil:	Ja

Quicksort

Grundlegender Ablauf

1. Wähle ein Pivot-Element p
2. Partitioniere die Folge in zwei Teile $a = \langle x : x < p \rangle$, $b = \langle x : x > p \rangle$
3. Sortiere a und b rekursiv
4. Füge die sortierten Felder zusammen

Eigenschaften

Worst Case:	$O(n^2)$
Average Case:	$O(n \cdot \log n)$
Best Case:	$O(n \cdot \log n)$
inplace:	Ja
stabil:	Nein

Radixsort

Grundlegender Ablauf

Sortiere Zahlen nach d Stellen, die jeweils k Werte annehmen können, beginnend bei geringwertigster Stelle.

Eigenschaften

Worst Case:	$O(d(n+k))$
Average Case:	$O(d(n+k))$
Best Case:	$O(d(n+k))$
inplace:	Ja
stabil:	Unbedingt!

Mastertheorem

Anwendung

Auflösung rekursiver Formeln der Form

$$T(n) = aT(\frac{n}{b}) + f(n)$$

Wird häufig zur Laufzeitbestimmung bei rekursiven Algorithmen verwendet.

Fälle

1. $f(n) \in O(n^{\log_b a - \varepsilon}) \Rightarrow T(n) \in \Theta(n^{\log_b a})$
2. $f(n) \in \Theta(n^{\log_b a}) \Rightarrow T(n) \in \Theta(n^{\log_b a} \cdot \log n)$
3. $f(n) \in \Omega(n^{\log_b a + \varepsilon})$ und $a \cdot f(\frac{n}{b}) \leq c \cdot f(n)$ für ein $c < 1 \Rightarrow T(n) \in \Theta(f(n))$

Es kann auch keiner der Fälle auftreten.

Datenstrukturen

Datenstrukturen

Dynamische Mengen können in unterschiedlichen Datenstrukturen verwaltet werden.

Operationen auf Dynamischen Mengen

- Search(S,k)
- Insert(S,x)
- Delete(S,x)
- Minimum(S)
- Maximum(S)
- Successor(S,x)
- Predecessor(S,x)

Uns interessiert der Zeitaufwand für die Operationen.

Stacks

Funktionsweise

Ein **Stack** (Stapel) implementiert einen **last-in, first-out**-Speicher.

Operationen

- PUSH(S,k): Füge neues Element hinzu
- POP(S): Entferne zuletzt hinzugefügtes Element

Stacks

Implementierung auf Array (Feld)

STACK-EMPTY(S)

```
1  return S.top == 0
```

PUSH(S, x)

```
1  S.top = S.top + 1
2  S[S.top] = x
```

POP(S)

```
1  if STACK-EMPTY(S)
2        error "underflow"
3  else S.top = S.top − 1
4        return S[S.top + 1]
```

$\Rightarrow$ Alle Operationen in $O(1)$.

Stacks

Beispiel

Sei S ein Stack. Führe folgende Operationen aus:

- Pop(S)
- Push(S,1)
- Push(S,2)
- Push(S,3)
- Push(S,4)
- Pop(S)
- Pop(S)
- Push(S,5)
- Pop(S)

Welche Informationen fehlen? Array-Größe 3, Stack leer

Stacks

```
STACK-EMPTY(S)
1  return S.top == 0

PUSH(S, x)
1  S.top = S.top + 1
2  S[S.top] = x

POP(S)
1  if STACK-EMPTY(S)
2        error "underflow"
3  else S.top = S.top − 1
4        return S[S.top + 1]
```

Aufgabe

Wandle die Operationen auf Stacks ab, so dass auch Overflows erkannt werden.

Queues

Funktionsweise

Eine **Queue** (Warteschlange) implementiert einen **first-in, first-out**-Speicher.

Operationen

- ENQUEUE(S,k): Füge neues Element hinzu
- DEQUEUE(S): Entferne zuerst hinzugefügtes Element

Queues

Implementierung auf Array (Feld)

ENQUEUE(Q, x)

```
1  Q[Q.tail] = x
2  if Q.tail == Q.length
3        Q.tail = 1
4  else Q.tail = Q.tail + 1
```

DEQUEUE(Q)

```
1  x = Q[Q.head]
2  if Q.head == Q.length
3        Q.head = 1
4  else Q.head = Q.head + 1
5  return x
```

$\Rightarrow$ Alle Operationen in $O(1)$.

Queues

Beispiel

Sei Q eine Queue. Führe folgende Operationen aus:

- Dequeue(Q)
- Enqueue(Q,1)
- Enqueue(Q,2)
- Enqueue(Q,3)
- Enqueue(Q,4)
- Dequeue(Q)
- Dequeue(Q)
- Enqueue(Q,5)
- Dequeue(Q)

Welche Informationen fehlen? Array-Größe 3, Queue leer

Queues

ENQUEUE(Q, x)

```
1  Q[Q.tail] = x
2  if Q.tail == Q.length
3        Q.tail = 1
4  else Q.tail = Q.tail + 1
```

DEQUEUE(Q)

```
1  x = Q[Q.head]
2  if Q.head == Q.length
3        Q.head = 1
4  else Q.head = Q.head + 1
5  return x
```

Aufgabe

Schreibe ENQUEUE- und DEQUEUE-Operationen, die Über- und Unterläufe erkennen.

Arrays

Operationen

- SEARCH: Suche ein Element ($O(n)$)
- GET: Lese Element mit Index ($O(1)$)
- DELETE: Entferne Element mit Index ($O(n)$)
- INSERT: Füge Element an Stelle ein ($O(n)$)

Verkettete Listen

Funktionsweise

Objekte linear angeordnet, Zugriff mit Zeigern

Varianten

- Einfach und doppelt verkettet
- Unsortiert und sortiert
- Nichtzyklisch und zyklisch

Wir betrachten Doppelt verkettete, unsortierte, nichtzyklische Listen.

Operationen

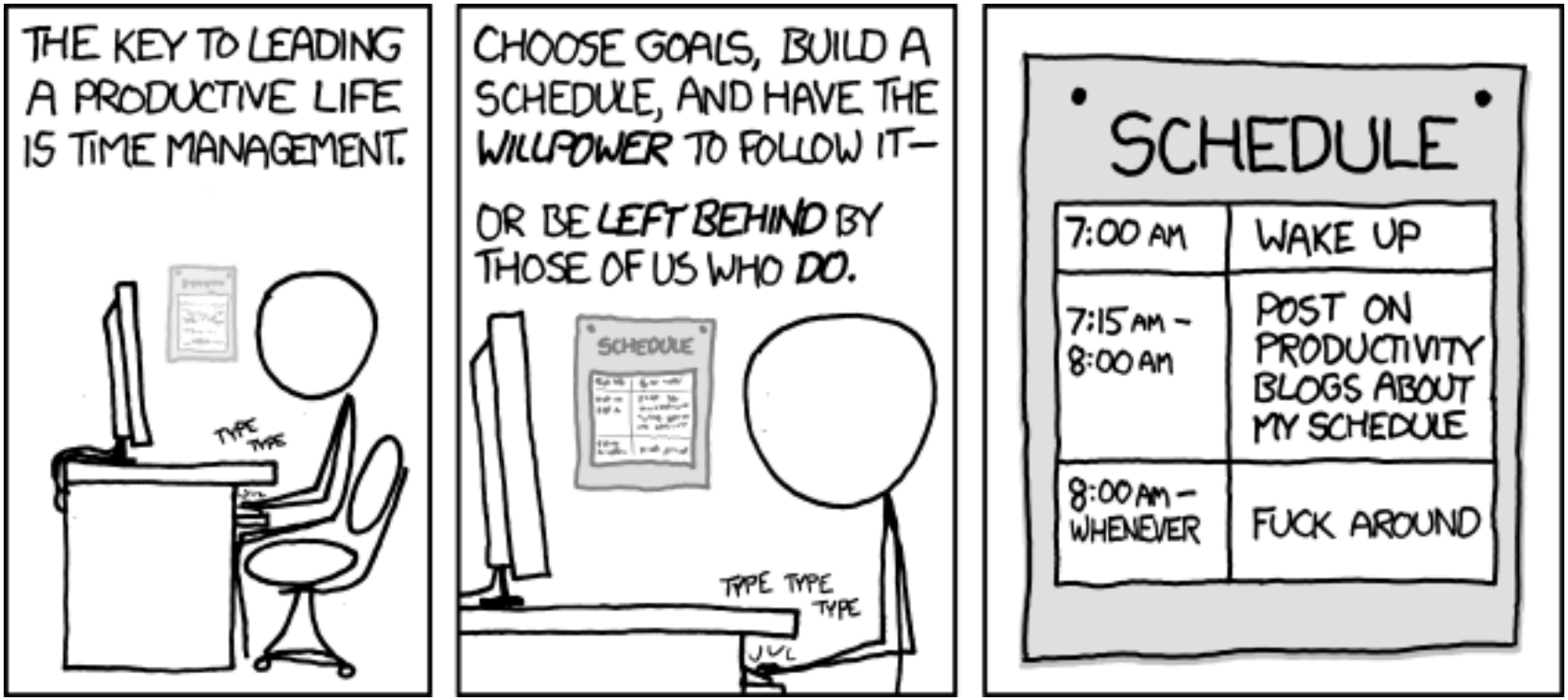
- SEARCH: Suche ein Element ($O(n)$)
- INSERT: Füge Element ein ($O(1)$)
- DELETE: Entferne Element ($O(1)$)

Entwurf Sortieralgorithmus

Aufgabe

Entwerfe einen Algorithmus der eine Folge von n Zahlen sortiert. Dabei liegen alle Zahlen zwischen 1 und n. Der Algorithmus soll im Worst-Case in $O(n)$ laufen.

Übersicht


THE KEY TO LEADING A PRODUCTIVE LIFE IS TIME MANAGEMENT.
TYPE TYPE
CHOOSE GOALS, BUILD A SCHEDULE, AND HAVE THE WILLPOWER TO FOLLOW IT—
OR BE LEFT BEHIND BY THOSE OF US WHO DO.
SCHEDULE
TYPE TYPE TYPE
SCHEDULE
7:00 AM
WAKE UP
7:15 AM – 8:00 AM
POST ON PRODUCTIVITY BLOGS ABOUT MY SCHEDULE
8:00 AM – WHENEVER
FUCK AROUND