Willkommen

Algorithmen I Tutorium 19

Wer? Florian Tobias Schandinat

Wo? 50.34, Raum -118

Wann? jeden Donnerstag 15:45-17:15

Material online

http://github.com/schandinat/algorithmen1_ss11

Nachtrag: TSP

Brute-force Lösung

Nachtrag: TSP

Brute-force Lösung

$\Rightarrow O(n!)$

$$O(2^n) \subset O(n!) \subset O(n^n)$$

Nachtrag: TSP

Brute-force Lösung

$\Rightarrow O(n!)$

$$O(2^n) \subset O(n!) \subset O(n^n)$$

Beispiel

Ein Logistikunternehmen beliefert täglich auf einer Tour 99 Adressen. Die Route dafür berechnet es mittels diesem Algorithmus innerhalb einer Stunde. Wie lange dauert die Berechnung ungefähr wenn es stattdessen 100 Adressen beliefert? (Annahme: nur asymptotischer Faktor relevant)

Nachtrag: TSP

Brute-force Lösung

$\Rightarrow O(n!)$

$$O(2^n) \subset O(n!) \subset O(n^n)$$

Beispiel

Ein Logistikunternehmen beliefert täglich auf einer Tour 99 Adressen. Die Route dafür berechnet es mittels diesem Algorithmus innerhalb einer Stunde. Wie lange dauert die Berechnung ungefähr wenn es stattdessen 100 Adressen beliefert? (Annahme: nur asymptotischer Faktor relevant)

$\Rightarrow$ ca 100 Stunden

$\Longrightarrow$ häufig nur näherungsweise Lösung

Rekursive Algorithmen

Mastertheorem

$$T(n) = 1, n = 1$$

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n), n > 1$$

$$\begin{aligned} \theta(n^{log_b a}) &, \quad f(n) = O(n^{log_b(a)-\epsilon}), \epsilon > 0 \\ \theta(n^{log_b a} \cdot log(n)) &, \quad f(n) = \theta(n^{log_b a}) \\ \theta(f(n)) &, \quad f(n) = \Omega(n^{log_b(a)+\epsilon}), \epsilon > 0 \end{aligned}$$

Rekursive Algorithmen

Mastertheorem

$$T(n) = 1, n = 1$$

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n), n > 1$$

$$\begin{aligned} \theta(n^{log_b a}) &, \quad f(n) = O(n^{log_b(a)-\epsilon}), \epsilon > 0 \\ \theta(n^{log_b a} \cdot log(n)) &, \quad f(n) = \theta(n^{log_b a}) \\ \theta(f(n)) &, \quad f(n) = \Omega(n^{log_b(a)+\epsilon}), \epsilon > 0 \end{aligned}$$

Für die Klausur auswendig lernen!

Suchen

Lineare Suche

- geht immer
- O(n)

Suchen

Lineare Suche

- geht immer
- O(n)

Binäre Suche

- Sortierung erforderlich
- random access
- O(log(n))

Arrays (Felder)

Eigenschaften

- bounded $\leftrightarrow$ unbounded
- Zugriff auf beliebiges Element: $O(1)$
- Löschen/Einfügen an beliebiger Stelle:

Arrays (Felder)

Eigenschaften

- bounded $\leftrightarrow$ unbounded
- Zugriff auf beliebiges Element: $O(1)$
- Löschen/Einfügen an beliebiger Stelle: $O(n)$

Stacks and Queues (Stapel und Schlangen)

Stack

- push, pop

Stacks and Queues (Stapel und Schlangen)

Stack

- push, pop
- LIFO

Stacks and Queues (Stapel und Schlangen)

Stack

- push, pop
- LIFO

Queue

- enqueue, dequeue

Stacks and Queues (Stapel und Schlangen)

Stack

- push, pop
- LIFO

Queue

- enqueue, dequeue
- FIFO

Konzept des Ringpuffers wird häufig verwendet
(Multithreading/Hardwaretreiber)

Stacks and Queues (Stapel und Schlangen)

Stack

- push, pop
- LIFO

Queue

- enqueue, dequeue
- FIFO

Konzept des Ringpuffers wird häufig verwendet
(Multithreading/Hardwaretreiber)

Implementieung

Alternativ: Implementierung als (einfach) verkettete Liste

Listen

Eigenschaften

- einfach verkettet $\leftrightarrow$ doppelt verkettet
- unsortiert $\leftrightarrow$ sortiert
- nicht-zyklisch $\leftrightarrow$ zyklisch
- Wächterelement

Praxisbeispiel – Eigenschaften

Linux-2.6.38-rc7: include/linux/list.h

```
static inline void list_replace(struct list_head *old,
                struct list_head *new)
{
    new->next = old->next;
    new->next->prev = new;
    new->prev = old->prev;
    new->prev->next = new;
}

static inline int list_empty(const struct list_head
                *head)
{
    return head->next == head;
}
```

Ende

Vielen Dank für die Aufmerksamkeit!