Informatik II

Übung 2

Andreas Bärtschi, Andreea Ciuprina, Felix Friedrich, Patrick Gruntz,
Hermann Lehner, Max Rossmannek, Chris Wendler

FS 2018

Heutiges Programm

1. Wiederholung Theorie
   - Induktion
   - Analyse von Programmen
   - Divide & Conquer
2. Programmieraufgabe
   - Quick Select
   - Collections in Java

Induktion: Was wird gebraucht?

- Beweise Aussagen, z.B. $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$.

Induktion: Was wird gebraucht?

- Beweise Aussagen, z.B. $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$.
- Induktionsanfang:
  - Die gegebene Gleichung bzw. Ungleichung stimmt für einen oder mehrere Basisfälle.
  - z.B.: $\sum_{i=1}^{1} i = 1 = \frac{1(1+1)}{2}$.

Induktion: Was wird gebraucht?

- Beweise Aussagen, z.B. $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$.
- Induktionsanfang:
  - Die gegebene Gleichung bzw. Ungleichung stimmt für einen oder mehrere Basisfälle.
  - z.B.: $\sum_{i=1}^{1} i = 1 = \frac{1(1+1)}{2}$.
- Induktionshypothese: Wir nehmen an, die Aussage stimmt für ein allgemeines n.

Induktion: Was wird gebraucht?

- Beweise Aussagen, z.B. $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$.
- Induktionsanfang:
  - Die gegebene Gleichung bzw. Ungleichung stimmt für einen oder mehrere Basisfälle.
  - z.B.: $\sum_{i=1}^{1} i = 1 = \frac{1(1+1)}{2}$.
- Induktionshypothese: Wir nehmen an, die Aussage stimmt für ein allgemeines n.
- Induktionsschritt ($n \to n+1$):
  - Aus der Gültigkeit der Aussage für n (Induktionshypothese) folgt die Gültigkeit für $n+1$.
  - z.B.: $\sum_{i=1}^{n+1} i = n+1+\sum_{i=1}^{n} i = n+1+\frac{n(n+1)}{2} = \frac{(n+2)(n+1)}{2}$.

Analyse

Wie oft wird `f()` aufgerufen?

```
for(unsigned i = 1; i <= n/3; i += 3)
  for(unsigned j = 1; j <= i; ++j)
    f();
```

Analyse

Wie oft wird `f()` aufgerufen?

```
for(unsigned i = 1; i <= n/3; i += 3)
  for(unsigned j = 1; j <= i; ++j)
    f();
```

Das Code-Fragment ruft `f()` $\Theta(n^2)$ mal auf: die äußere Schleife wird $n/9$ mal durchlaufen, und die innere Schleife ruft `f()` i mal auf.

Analyse

Wie oft wird `f()` aufgerufen?

```
for(unsigned i = 0; i < n; ++i) {
  for(unsigned j = 100; j*j >= 1; --j)
    f();
  for(unsigned k = 1; k <= n; k *= 2)
    f();
}
```

Analyse

Wie oft wird `f()` aufgerufen?

```
for(unsigned i = 0; i < n; ++i) {
  for(unsigned j = 100; j*j >= 1; --j)
    f();
  for(unsigned k = 1; k <= n; k *= 2)
    f();
}
```

Wir können die erste innere Schleife ignorieren, weil sie `f()` nur konstant oft aufruft.

Analyse

Wie oft wird `f()` aufgerufen?

```
for(unsigned i = 0; i < n; ++i) {
  for(unsigned j = 100; j*j >= 1; --j)
    f();
  for(unsigned k = 1; k <= n; k *= 2)
    f();
}
```

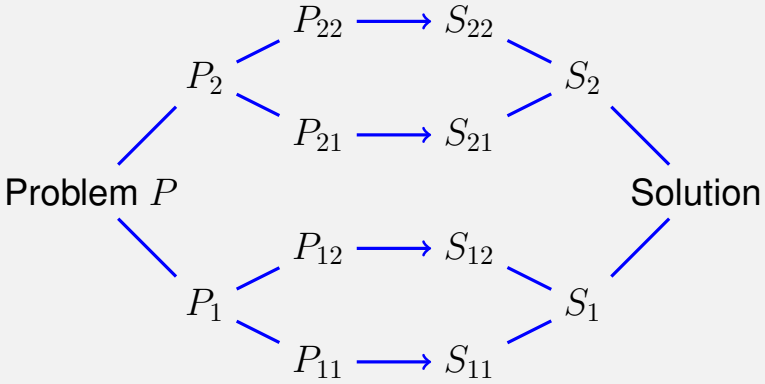
Wir können die erste innere Schleife ignorieren, weil sie `f()` nur konstant oft aufruft.
Die zweite innere Schleife ruft `f()` $\lfloor \log_2(n) \rfloor + 1$ mal auf, in Summe haben wir $\Theta(n \log(n))$ Aufrufe.

divide et impera

Teile und (be)herrsche (engl. divide and conquer)

Zerlege das Problem in Teilprobleme, deren Lösung zur vereinfachten Lösung des Gesamtproblems beitragen.

divide et impera



Binärer Suchalgorithmus BSearch$(A[l..r], b)$

Input : Sortiertes Array A von n Schlüsseln. Schlüssel b. Bereichsgrenzen $1 \leq l \leq r \leq n$ oder $l > r$ beliebig.
Output : Index des gefundenen Elements. 0, wenn erfolglos.

$m \leftarrow \lfloor (l + r)/2 \rfloor$
if $l > r$ **then** // erfolglose Suche
    return *NotFound*
else if $b = A[m]$ **then** // gefunden
    return m
else if $b < A[m]$ **then** // Element liegt links
    return BSearch$(A[l..m-1], b)$
else // $b > A[m]$: Element liegt rechts
    return BSearch$(A[m+1..r], b)$

2. Programmieraufgabe

Das Auswahlproblem

Eingabe

- Unsortiertes Array $A = (A_1, \ldots, A_n)$ paarweise verschiedener Werte
- Zahl $1 \leq k \leq n$.

Ausgabe: $A[i]$ mit $|\{j : A[j] < A[i]\}| = k - 1$

Spezialfälle

$k = 1$: Minimum: Algorithmus mit n Vergleichsoperationen trivial.
$k = n$: Maximum: Algorithmus mit n Vergleichsoperationen trivial.
$k = \lfloor n/2 \rfloor$: Median.

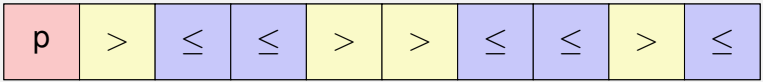
Pivotieren

Pivotieren

1. Wähle ein Element p als Pivotelement

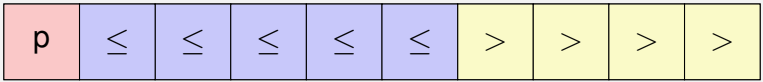
Pivotieren

1. Wähle ein Element p als Pivotelement
2. Teile A in zwei Teile auf, den Rang von p bestimmend.



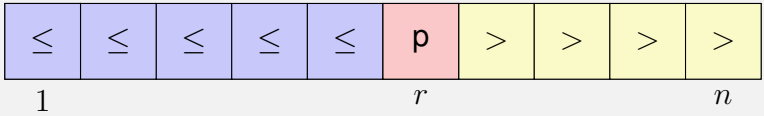
Pivotieren

1. Wähle ein Element p als Pivotelement
2. Teile A in zwei Teile auf, den Rang von p bestimmend.



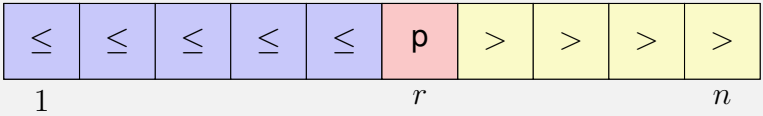
Pivotieren

1. Wähle ein Element p als Pivotelement
2. Teile A in zwei Teile auf, den Rang von p bestimmend.



Pivotieren

1. Wähle ein Element p als Pivotelement
2. Teile A in zwei Teile auf, den Rang von p bestimmend.
3. Rekursion auf dem relevanten Teil. Falls $k = r$, dann gefunden.



Algorithmus Partition($A[l..r], p$)

Input : Array A, welches den Pivot p im Intervall $[l, r]$ mindestens einmal enthält.
Output : Array A partitioniert um p. Rückgabe der Position von p.

```
while l ≤ r do
    while A[l] < p do
        l ← l + 1
    while A[r] > p do
        r ← r − 1
    swap(A[l], A[r])
    if A[l] = A[r] then
        l ← l + 1
return l-1
```

Algorithmus Quickselect ($A[l..r], k$)

Input : Array A der Länge n. Indizes $1 \leq l \leq k \leq r \leq n$, so dass für alle $x \in A[l..r] : |\{j|A[j] \leq x\}| \geq l$ und $|\{j|A[j] \leq x\}| \leq r$.
Output : Wert $x \in A[l..r]$ mit $|\{j|A[j] \leq x\}| \geq k$ und $|\{j|x \leq A[j]\}| \geq n - k + 1$

if l=r **then**
    return $A[l]$;
$x \leftarrow$ RandomPivot$(A[l..r])$
$m \leftarrow$ Partition$(A[l..r], x)$
if $k < m$ **then**
    return QuickSelect$(A[l..m-1], k)$
else if $k > m$ **then**
    return QuickSelect$(A[m+1..r], k)$
else
    return $A[k]$

Daten Organisieren

- Datenstrukturen, die wir kennen
  - Arrays – Sequenzen fixer Grösse
  - Strings – Buchstabensequenzen
  - Verkettete Listen (bisher: für festen Elementtyp selbstgemacht)
- Allgemeines Container Konzept in Java
  - ArrayList auf generischem Elementtyp – dynamischer als Arrays
  - LinkedList, HashMaps, Sets, Maps, ...

Generische Liste in Java: `java.util.List`

```
import java.util.ArrayList;
import java.util.List;
...

// Liste von Strings
List<String> list = new ArrayList<String>();

list.add("abc");
list.add("xyz");
list.add(1,"123"); // Fuege 123 an Position 1 ein
System.out.println(list.get(0)); // abc

for (String s: list) // For auf Iterator der Liste
       System.out.println(s); // abc 123 xyz
```

Liste von Integers

- Java Generics (z.B. Container) können nur auf Objekten operieren
- Fundamentaltypen `int`, `float` (etc.) sind keine Objekte
- Java bietet Wrapperklassen für Fundamentaltypen an, z.B. typ `Integer`
- Java macht *autoboxing* und packt einen Fundamentaltyp automatisch in eine Wrapperklasse, wo nötig.

Liste von Integers

```
import java.util.ArrayList;
import java.util.List;
...

// Lists of integers
List<Integer> list = new ArrayList<Integer>();

list.add(3);
list.add(4);
list.add(1,5); // Fuege 5 an Position 1 ein
System.out.println(list.get(0)); // 3

for (int i: list){ // For auf Iterator der Liste
       System.out.println(i); // 3 5 4
}
```

Liste von Integers

```
import java.util.ArrayList;
import java.util.List;
...

// Lists of integers
List<Integer> list = new ArrayList<Integer>();

list.add(3);
list.add(4);
list.add(1,5); // Fuege 5 an Position 1 ein
System.out.println(list.get(0)); // 3

for (int i: list){ // For auf Iterator der Liste
       System.out.println(i); // 3 5 4
}
```

Fragen oder Anregungen?