Vorkurs **Datenstrukturen**

Sommersemester 2011

Herzlich willkommen!

Abstrakte Datentypen und Datenstrukturen

- Ein abstrakter Datentyp besteht aus einer Sammlung von Operationen auf einer Menge von Objekten.
- Eine Datenstruktur implementiert einen abstrakten Datentyps.

- Warum abstrakte Datentypen?
  - In Anwendungen treten dieselben Operationen für unterschiedlichste Datenmengen auf.
  - Gib eine Implementierung an!
- Zum Beispiel, füge Schlüssel ein und entferne Schlüssel geordnet nach ihrem Alter:
  - Wenn der jüngste Schlüssel zu entfernen ist: Wähle die Datenstruktur „Stack“.
  - Wenn der älteste Schlüssel zu entfernen ist: Wähle die Datenstruktur „Schlange“.
  - Wenn Schlüssel mit zugeordneter Prioritäten eingefügt werden und der Schlüssel mit jeweils höchster Priorität zu entfernen ist: Benutze die Datenstruktur „Heap“.

Effizienz I

Das Ziel der Vorlesung „**Datenstrukturen**“:

Entwerfe eine möglichst effiziente Datenstruktur für einen gegebenen abstrakten Datentyp.

- Wie skaliert die Laufzeit einer Implementierung mit wachsender Eingabelänge?
  - Das Verhalten für große Eingabelängen ist kritisch.
  - Für **Eingabelänge n** sei $T(n)$ die maximale Laufzeit für eine Eingabe der Länge n.
- Führe eine asymptotische Analyse von Laufzeit und Speicherplatzverbrauch durch.
  - Die Laufzeit ist linear, falls $0 < \lim_{n\to\infty} \frac{T(n)}{n} < \infty$
  - und quadratisch, falls $0 < \lim_{n\to\infty} \frac{T(n)}{n^2} < \infty$.

Wir müssen uns mit Grenzwerten beschäftigen.

Effizienz II

Angenommen, wir möchten n Zahlen sortieren.

- Wir definieren die Eingabelänge als die Anzahl n der Zahlen.
- Ein Sortierverfahren wie Bubble Sort benötigt (ungefähr) quadratisch viele Vergleiche im schlimmsten Fall.
- Merge Sort hingegen kommt für jede Eingabe mit (ungefähr) $n \cdot \log_2 n$ Vergleichen aus.

Die asymptotische Analyse zeigt:

$$\lim_{n \to \infty} \frac{n \cdot \log_2 n}{n^2} = 0$$

Merge Sort ist wesentlich schneller als Bubble Sort.

Wie schnell dominiert die Asymptotik?

Annahme: Ein einfacher Befehl benötigt 10^{-9} Sekunden.

n	n^2	n^3	n^{10}	2^n	$n!$
16	256	4.096	$\geq 10^{12}$	65536	$\geq 10^{13}$
32	1.024	32.768	$\geq 10^{15}$	$\geq 4 \cdot 10^9$	$\geq 10^{31}$
64	4.096	262.144	$\geq 10^{18}$	$\geq 6 \cdot 10^{19}$	
					mehr als
128	16.384	2.097.152	mehr als	mehr als	10^{14} Jahre
256	65.536	16.777.216	10 Jahre	600 Jahre	
512	262.144	134.217.728			
1024	1.048.576	$\geq 10^9$			
Million	$\geq 10^{12}$	$\geq 10^{18}$			
	mehr als	mehr als			
	15 Minuten	10 Jahre			

Effizienz III

Angenommen, wir möchten feststellen, ob eine Zahl x in einem sortierten Array von n Zahlen vorkommt.

- Wir definieren die Eingabelänge wieder als die Anzahl n der Zahlen.
- Lineare Suche: Wir durchlaufen das Array von „links nach rechts“ und überprüfen jeweils ob wir x gefunden haben.

  Mindestens n Vergleiche sind notwendig, wenn x nicht vorkommt.
- Binärsuche vergleicht x mit $A[n/2]$.
  - Wenn $x = A[n/2]$, dann haben wir x gefunden!
  - Wenn $x < A[n/2]$, dann suche in der linken Hälfte $(A[1], \ldots, A[n/2] - 1)$,
  - und sonst in rechten Hälfte $(A[n/2 + 1], \ldots, A[n])$.

Wer ist schneller: Lineare Suche oder Binärsuche?

Binärsuche

```
void Suche( int unten, int oben){
  if (oben < unten)
    std::cout « x « " wurde nicht gefunden."
     « std::endl;
  int mitte = (unten+oben)/2;
  if (A[mitte] == x)
    std::cout « x « " wurde in Position "
     « mitte « " gefunden." « std::endl;
    else {
      if (x < A[mitte])
        Suche(unten, mitte-1);
      else
        Suche(mitte+1,oben);
    }
}
```

Rekursion: Eine fundamentale Programmiertechnik

- Wie kann man ein rekursives Programm verifizieren?
  Wie lassen sich Aussagen über die Laufzeit machen?
- Mit Hilfe der vollständigen Induktion.

Warum ist `Suche(a,b)` korrekt?

Wir beweisen Korrektheit durch Induktion nach der Anzahl $n = \mathit{oben} - \mathit{unten} + 1$ der Zahlen.

- Der **Basisschritt** für $n \leq 0$:
  - Wenn $n \leq 0$, dann ist das Array leer und x kann nicht vorkommen.
  - `Suche(unten,oben)` ist für $n \leq 0$ korrekt.
- Der **Induktionsschritt**:
  - Wir können die **Induktionsvoraussetzung** annehmen: `Suche(unten',oben')` ist korrekt, wenn $\mathit{oben}' - \mathit{unten}' \leq n$.
  - Wir müssen zeigen, dass `Suche(unten,oben)` für $\mathit{oben} - \mathit{unten} = n + 1$ korrekt ist.

Der Induktionsschritt

- Wenn an der Position `mitte` die Zahl x steht, wird sie ordnungsgemäß gefunden.
- Andernfalls ruft sich der Algorithmus rekursiv auf einem Teilproblem auf.
  - Das Teilproblem hat weniger als $\texttt{oben} - \texttt{unten} - 1 = n$ Zahlen.
  - Auf einem Problem dieser Größe arbeitet der Algorithmus jedoch nach Induktionsannahme korrekt!

Wieviele Vergleiche maximal führt `suche(unten,oben)` aus?

Die Anzahl der Vergleiche hängt nur von der Zahl $n = \mathit{oben} - \mathit{unten} + 1$ der Elemente des Arrays ab.

- Es gelte $n = 2^k - 1$.
  - Sei $T(n)$ die maximale Vergleichszahl für Arrays der Länge n.
  - Dann ist $T(1) = 1$.
  - `Suche(unten,b oben)` arbeitet entweder in der linken Hälfte oder in der rechten Hälfte rekursiv weiter:
    - ⋆ Beide Hälften bestehen aus genau $2^{k-1} - 1$ Zahlen.
  - Es ist $T(2^k - 1) = T(2^{k-1} - 1) + 1$, falls $k \geq 1$.
  - Es ist $T(2^k - 1) = k$. Warum?
- Und wenn n eine beliebige natürliche Zahl ist? Dann ist

$$T(n) = \lceil \log_2(n+1) \rceil.$$

Binärsuche ist viel schneller als lineare Suche, denn $\lim_{n\to\infty} \frac{\log_2 n}{n} = 0$.

Der Fahrplan

Wir machen mit

rekursiver Programmierung / vollständiger Induktion

weiter und kehren dann zur

asymptotischen Analyse

zurück.

Die Türme von Hanoi

- Die **Anfangskonfiguration**:
  - Wir haben drei Stäbe 1, 2 und 3.
  - Ursprünglich besitzt Stab 1 N Ringe, wobei die Ringe in absteigender Größe auf dem Stab aufgereiht sind:
    - ⋆ Der größte Ring von Stab 1 ist also der unterste Ring.
  - Die Stäbe 2 und 3 sind zu Anfang leer.
- Die **Züge**
  - Bewege einen zuoberst liegenden Ring von einem Stab zu einem anderen.
  - Der Zug ist nur dann erlaubt, wenn der Ring auf einen größeren Ring gelegt wird oder wenn der Stab leer ist.
- Die **Zielkonfiguration**:
  - Alle Ringe müssen nach Stab 2 bewegt werden.

Ein rekursives Programm

```
void Hanoi( int N, int stab1, int stab2, int stab3)
{if (N==1)
    bewege einen Ring von Stab stab1 nach Stab stab2;
else {
    Hanoi(N-1,stab1,stab3,stab2);
    bewege einen Ring von Stab stab1 nach Stab stab2;
    Hanoi(N-1,stab3,stab2,stab1); }}
```

- Wie zeigt man, dass `Hanoi` korrekt ist?
- Sei $T(N)$ die Anzahl der Ringbewegungen nach Aufruf des Programms Hanoi(N,*,*,*): Wie bestimmt man $T(N)$?

Die Summe der ersten n Zahlen

$$\sum_{i=1}^{n} i = 1 + 2 + \ldots + n = \frac{n \cdot (n+1)}{2}.$$

- Vollständige Induktion nach n:
  - Induktionsbasis für $n = 0$: $\sum_{i=1}^{0} i = 0$ und $\frac{0 \cdot (0+1)}{2} = 0$. Stimmt!
  - Induktionsschritt von n auf $n+1$:
    - Wir können annnehmen, dass $\sum_{i=1}^{n} i = \frac{n \cdot (n+1)}{2}$ gilt.
    - $\sum_{i=1}^{n+1} i = \sum_{i=1}^{n} i + (n+1) = \frac{n \cdot (n+1)}{2} + (n+1) = \frac{(n+1) \cdot (n+2)}{2}$. Stimmt!
- Ein direktes Argument:
  - Betrachte ein Gitter mit n Zeilen und n Spalten: n^2 Gitterpunkte.
  - Wir müssen die Gitterpunkte unterhalb der Hauptdiagonale und auf der Hauptdiagonale zählen.
    - Die Hauptdiagonale besitzt n Gitterpunkte.
    - Unterhalb der Hauptdiagonale befindet sich die Hälfte der verbleibenden $n^2 - n$ Gitterpunkte.
  - Also folgt $\sum_{i=1}^{n} i = n + \frac{n^2 - n}{2} = \frac{n \cdot (n+1)}{2}$.

Die geometrische Reihe

$$\sum_{i=0}^{n} a^i = \frac{a^{n+1}-1}{a-1} \text{ falls } a \neq 1.$$

- Vollständige Induktion nach n:
  - Induktionsbasis für $n = 0$: $\sum_{i=1}^{0} a^i = 1$ und $\frac{a^{0+1}-1}{a-1} = 1$. Stimmt!
  - Induktionsschritt von n auf $n+1$:
    - ★ Wir können annnehmen, dass $\sum_{i=1}^{n} a^i = \frac{a^{n+1}-1}{a-1}$ gilt. Dann ist
    - ★ $\sum_{i=1}^{n+1} a^i = \sum_{i=1}^{n} a^i + a^{n+1} = \frac{a^{n+1}-1}{a-1} + a^{n+1} = \frac{a^{n+2}-1}{a-1}$. Stimmt!
- Ein direktes Argument:

$$\begin{aligned}(a-1)\cdot\sum_{i=0}^{n} a^i &= a\cdot\sum_{i=0}^{n} a^i - \sum_{i=0}^{n} a^i \\ &= \sum_{i=1}^{n+1} a^i - \sum_{i=0}^{n} a^i = a^{n+1} - a^0 = a^{n+1} - 1\end{aligned}$$

Kombinatorische Fragestellungen

Die Menge U bestehe aus n Objekten.

(a) Zeige, dass U genau 2^n Teilmengen besitzt, wenn wir die leere Menge als eine Teilmenge von U zulassen.

(b) Zeige, dass U genau $\binom{n}{k} = \frac{n!}{k!(n-k)!}$ Teilmengen mit k Elementen besitzt.

(c) Zeige: Es gibt genau $n! = n \cdot (n-1) \cdot (n-2) \cdots 2 \cdot 1$ verschiedene Permutationen von n Objekten.

Binomischer Lehrsatz

Zeige: $(a+b)^n = \sum_{k=0}^{n} \binom{n}{k} a^k b^{n-k}$. (Zeige $\binom{n+1}{k} = \binom{n}{k} + \binom{n}{k-1}$ zuerst.)

Geometrische Fragestellungen

- Teile ein Rechteck durch Geraden in Teilflächen.
- Kann man die Teilflächen immer so mit den Farben Schwarz und Weiß färben, dass Teilflächen, die an einer Kante zusammenstoßen, verschiedene Farben besitzen?

Wo ist der Fehler?

Wir zeigen, dass je zwei natürliche Zahlen a und b gleich sind.

- Wir setzen $k = \max\{a, b\}$ und führen eine Induktion nach k.
  - Im Basisschritt haben wir $k = 0$ und deshalb ist $a = 0 = b$ und das war zu zeigen.
  - Im Induktionsschritt ist $\max\{a, b\} = k + 1$.
    - Wir können die Induktionsbehauptung auf $a - 1$ und $b - 1$ anwenden, denn $\max\{a - 1, b - 1\} = k$.
    - Also ist $a - 1 = b - 1$ und die Behauptung $a = b$ folgt.
- Die Behauptung ist richtig für $k = 0$, aber falsch schon für $k = 1$.