Uni Algodat Notes

Felix Pojtinger

July 4, 2021

Uni Algodat Notes

Uni Algodat Notes

Themen

Themen

- Einleitung
  - Zyklus der Informationsverarbeitung
  - Grundbegriffe zu Algodat
  - Eigenschaften von Algorithmen
- Zeitkomplexität
  - Problemgröße des Inputs
  - Schrittzahlfunktionen
  - Testfunktionen
  - Best Case & Worst Case
  - Beispiele
    - Suche im Binärbaum
    - Fibonacci-Sequenz
    - Bubble-Sort
- Applikative Algorithmen
  - Definition Algorithmus
  - Schritte des Auswertung
  - Rekursion
- Sortieralgorithmen
  - Merge-Sort
    - Zeitkomplexität
  - Quicksort

Paradigm Conversion

Paradigm Conversion

I'm too stupid to write proper functional code, so most of the times I "convert" my imperative solutions to functional ones using the following schema I found on the web:

1. Isolate the loop in its own function. Make sure that all captured variables (i.e., variables that are used in the loop, but not declared in the loop) are passed in as parameters.
2. If the loop declares its own counter (e.g., in a for-loop), remove that declaration and make the loop counter another parameter.
3. Replace the loop construct itself:
   3.1 Replace the loop condition check with an if statement (or if expression, if that's what your language has).
   3.2 In the failure branch, return.
   3.3 Move the rest of the loop code into the success branch.
   3.4 At the end of the success branch, add a recursive call which passes the modified values of the loop counter and all captured variables; this is equivalent to the jump back to the top of the original loop.
4. At the original site of the loop, put in a call to your new

Begriffe

Begriffe

- **Algorithmus**: Präzise, endliche Beschreibung eines allgemeinen Verfahrens unter Verwendung ausführbarer, elementarer Schritte
- **Daten**: Von Maschinen verarbeitbare Form von Infos
- **Datenstruktur**: Von Maschinen verarbeitbarer struktureller Rahmen für die Darstellung von Daten
- **Programm**: Formulierung eines oder mehrerer Algorithmen und Datenstrukturen durch Programmiersprache

Eigenschaften von Algorithmen

Eigenschaften von Algorithmen

- **Terminierend**: Bricht bei jeder (erlaubten) Eingabe nach endlich vielen Schritten ab
- **Determiniertes Ergebnis**: Liefert bei selber (erlaubter) Eingabe immer dasselbe Ergebnis
- **Determinierter Ablauf**: Führt bei selber (erlaubter) Eingabe immer die Schritte in derselben Reihenfolge aus
- **Deterministisch**: Determiniertes Ergebnis & determinierter Ablauf
- **Zeitkomplexität**: Wachstumsverhalten der benötigten Anzahl von Schritten, wenn Problemgröße gegen unendlich strebt

Zeitkomplexität

Zeitkomplexität

1. Ausgangspunkt: Algorithmus A
2. Finde ein Maß für die Problemgröße des Inputs
   - **Sortieren einer Liste**: $N = Anzahl\ Elemente\ der\ Liste$
   - **Bäume**: $N = Anzahl\ Knoten$
   - **Textsuche**: $N = Anzahl\ Zeichen\ im\ Text$
3. Finde die Anzahl der signifikanten Schritte von A bei Problemgröße N.
4. Bestimme das Wachstumsverhalten der Funktion für $N \to \infty$
5. Vergleiche Funktion mit $t(N)$ und finde *richtiges* Wachstumsverhalten.

Wichtige Wachstumsfunktionen

- **Bäume**: $O(log(n))$
- **Binary Search**: $O(log(n))$
- **Merge Sort**: $O(log(n))$ (Out-of-Place)
- **Quicksort**: $O(log(n))$ (In-Place)
- **Bubble Sort**: $O(n^2)$

Sortieralgorithmen

- **In-Place**: Es wird (nahezu) kein weiterer Speicher bei der Ausführung gebraucht (i.e. Quicksort)
- **Out-of-Place**: Es wird zusätzlicher Speicher bei der Ausführung gebraucht (i.e. Merge Sort)

ADTs

Übersicht

- Abstrakte Datentypen als Vorläufer der Klassen in OOP
- ADT: Mehrfach instanziierbares Software-Modul
- Jede Instanz hat ein Interface
- Syntax für Operatoren: <op_name>: <arg1> [x arg2...] → <return_type>

Stack

- **type** $Stack(T)$
- **sorts**: $T, bool$
- **operations**
  - **new**: $\rightarrow Stack(T)$
  - **push**: $Stack(T)xT \rightarrow Stack(T)$
  - **pop**: $Stack(T) \rightarrow Stack(T)$
  - **top**: $Stack(T) \rightarrow T$
  - **empty**: $Stack(T) \rightarrow bool$
- **axioms**
  - $pop(push(s, t)) = s$
  - $top(push(s, t)) = t$
  - $empty(new(())) = true$
  - $empty(push(s, t)) = false$

Queue

- **type** $Queue(T)$
- **sorts** $T, bool$
- **operations**
  - **new**: $\rightarrow Queue(T)$
  - **enter**: $Queue(T)xT \rightarrow Queue(T)$
  - **leave**: $Queue(T) \rightarrow Queue(T)$
  - **head**: $Queue(T) \rightarrow T$
  - **empty**: $Stack(T) \rightarrow bool$
- **axioms**
  - $empty(new()) = true$
  - $head(enter(new(), t)) = t$
  - $leave(enter(new(), t)) = new()$
  - $leave(new()) = new()$
  - $head(enter(enter(q, s), t)) = head(enter(q, s))$
  - $leave(enter(enter(q, s), t)) = enter(leave(enter(q, s)), t)$

Binärbaum

- **type** $BinTree(T)$
- **sort** $T, bool$
- **operations**
  - **new**: $\rightarrow BinTree(T)$
  - **bin**: $BinTree(T)xTxBinTree(T) \rightarrow BinTree(T)$
  - **left**: $BinTree(T) \rightarrow BinTree(T)$
  - **right**: $BinTree(T) \rightarrow BinTree(T)$
  - **root**: $BinTree(T) \rightarrow T$
  - **empty**: $BinTree(T) \rightarrow bool$
- **axioms**
  - $left(bin(x, t, y)) = x$
  - $right(bin(x, t, y)) = y$
  - $root(bin(x, t, y)) = t$
  - $empty(new()) = true$
  - $empty(bin(x, t, y))) = false$