9. Rekursive Algorithmen

Algorithmen, die darauf basieren, dass zur Lösung eines Problems der Algorithmus auf ein oder mehrere Teilprobleme kleinerer Größenordnung angewandt wird.

Wesentlich ist, dass man bei der Reduktion des Problems schließlich bei trivialen Problemen ankommt.

Beispiele:

Berechnung der Fakultät von n:

1. **Eingabe n (n>=0)**
2. **Falls n = 0 gilt (Trivialfall), führe Schritt 3 durch:**
3. **fak = 1**
4. **Ansonsten führe Schritt 5 durch:**
5. **fak = n * fak(n-1)**
6. **Ausgabe fak**

Das Problem der Größenordnung n wird auf ein Problem der Größenordnung n-1 reduziert. Daher läßt sich der Algorithmus einfach auch iterativ formulieren.

Berechnung der Binomialkoeffizienten binom(n,k):

1. **Eingabe n,k (n>=k>=0)**
2. **Falls k=n oder k=0 gilt (Trivialfall), führe Schritt 3 durch:**
3. **binom = 1**
4. **Ansonsten führe Schritt 5 durch:**
5. **binom = binom(n-1,k)+binom(n-1,k-1)**
6. **Ausgabe binom**

Die rekursive Implementierung führt zu Wiederholungen der gleichen Berechnung! Um dies zu vermeiden, kann man bottom up vorgehen und Zwischenergebnisse speichern. Man speichert binom(i,k) für alle n>=i>=k>=0. (Dynamic Programming)

1. **Für i = 0,...,n führe die folgenden Schritte durch:**
2. **Für k=0,...,i führe die folgenden Schritte durch:**
3. **Falls k=0 oder k=i, führe Schritt 4 durch:**
4. **Setze binom (i,k) auf 1**
5. **Ansonsten führe Schritt 6 durch:**
6. **Setze binom (i,k) = binom (i-1,k-1) + binom (i-1,k)**

Eine noch einfachere Lösung erhält man, wenn man beachtet, daß

Binom(n,k) = fak(n) / fak(n-k) * fak(k) gilt,

also binom (n,k) das Produkt n/1 * (n-1)/2 * (n-2)/3 * ... ist.

Türme von Hanoi:

Es gibt drei Stapel, auf denen Scheiben unterschiedlicher Größe abgelegt werden. Es gelten folgende Aussagen:

- **Die Scheiben haben unterschiedliche Größe**
- **Es darf keine größere Scheibe auf einer kleineren Scheibe liegen.**

Die Aufgabe besteht darin, dass n Scheiben von Stapel i nach Stapel j bewegt werden, wobei die o.g. Regeln eingehalten werden.

1. **Falls n=1 gilt (Trivialfall), führe Schritt 2 durch:**
2. **Bewege die Scheibe von Stapel i nach Stapel j.**
3. **Ansonsten führe die folgenden Schritte durch:**
4. **Bewege n-1 Scheiben von Stapel i nach Stapel k.**
5. **Bewege 1 Scheibe von Stapel i nach Stapel j.**
6. **Bewege n-1 Scheiben von Stapel k nach Stapel j.**

Hier ist die Formulierung ohne Rekursion weniger einfach. Man muß explizit den Stack, der zur Verwaltung der Funktionsaufrufe verwendet wird, nachbilden. Dies ist eine gute Übung zur Verwendung des abstrakten Datentyps Stack.

Generell können wir feststellen:

Jeder rekursive Algorithmus lässt sich unter Verwendung der Ablaufstrukturen Sequenz, Selektion und Iteration formulieren.

10. Lösungsverfahren

i. Einfache Berechnung (Formel)

ii. Schrittweise Annäherung

Beispiel: Iterationen in der numerischen Mathematik

iii. Komponentenweiser Aufbau

Beispiel: Gleichungssystem mit Dreiecksmatrix

iv. Reduktion auf ein Problem geringerer Größenordnung

Beispiel: Fakultät, Sequentielle Suche, Binäre Suche

ALGO (Problem):

1. **Falls das Problem trivial ist, führe Schritt 2 durch:**
2. **Löse das Trivialproblem.**
3. **Ansonsten führe die folgenden Schritte durch:**
4. **Reduziere das Problem auf ein Problem P geringerer Größenordnung.**
5. **Führe ALGO (P) aus.**

Komplexitätsbetrachtung:

Reduktion von n auf n-1:

Z(n)=C1+Z(n-1)

Z(1)=C0

=> Z(n)= Theta(n)

Reduktion von n auf n/a:

Z(n) = C1 + Z(n/a)

Z(1)=C0

=> Z(n) = Theta(log(n))

v. Teilen und Beherrschen (divide and conquer, divide et impera)

Beispiel: Türme von Hanoi, Quicksort, Mergesort

ALGO (Problem):

1. **Falls das Problem trivial ist, führe Schritt 2 durch:**
2. **Löse das Trivialproblem.**
3. **Ansonsten führe die folgenden Schritte durch:**
4. **Teile das Problem in zwei Teilprobleme P1 und P2 auf.**
5. **Führe ALGO(P1) und ALGO(P2) aus.**
6. **Fasse die Lösungen der Teilprobleme zusammen.**

Komplexitätsbetrachtung:

Falls Reduktion von n auf n-1, gilt:

Z(n)=C*n + Z(1) + Z(n-1),

Z(1)=C0

=> Z(n) = Theta(n^2)

Falls Reduktion von n auf n/a, gilt:

Z(n)=C*n + a*Z(n/a)

Z(1)=C0

=> Z(n) = Theta(n*log(n))

vi. Backtracking, Branch and Bound

Beispiel: Damenproblem, Rucksackproblem (Optimierung)

ALGO:

1. **Setze i auf 1.**

2. **Solange i<=n gilt, führe die folgenden Schritte durch:**

3. **Falls die i-te Komponente der Lösung leer ist, setze die i-te Komponente der Lösung auf den ersten möglichen Wert, ansonsten setzte die i-te Komponente der Lösung auf den nächsten möglichen Wert.**

4. **Solange noch nicht alle Werte ausgeschöpft sind und die i-te Komponente der Lösung nicht brauchbar ist, setze die i-te Komponente der Lösung auf den nächsten möglichen Wert.**

5. **Falls die Werte für die i-te Komponente der Lösung noch nicht ausgeschöpft sind, erhöhe i, ansonsten setze i zurück (jeweils um 1).**

Klassifikation von Algorithmen (erweitert)

Deterministische Algorithmen:

Algorithmen, für die in jeder Situation der nächste Schritt bestimmt ist.

Nichtdeterministische Algorithmen:

Algorithmen, für die der nächste Schritt von der jeweiligen Situation abhängig ist.

Nichtdeterministische Algorithmen weisen exponentielle Komplexität auf, da der jeweils nächste Schritt durch Trial und Error zu bestimmen ist. Sie sind deshalb mit Vorsicht einzusetzen.

Probabilistische Algorithmen:

Algorithmen, für die es eine Situation gibt, in der der nächste Schritt von einer Zufallsauswahl abhängt.

Heuristische Algorithmen:

Algorithmen, bei denen es darum geht, für Probleme, deren Lösung zu komplex ist, eine Näherungslösung zu finden, die man mit vertretbarem Aufwand bestimmen kann (und für die man die Qualität der Näherung bestimmen kann).

Determinierte Algorithmen:

Algorithmen, die bei gegebener Eingabe zu einer eindeutigen Lösung führen.

11. Parallele Algorithmen

Betrachtet man die Lösungsverfahren in Kapitel 10, so gibt es ein Verfahren, bei dem die bisher betrachtete sequentielle Programmierung eher hinderlich erscheint: Das Verfahren „Teile und Herrsche“

Der Ablauf sieht wie folgt aus:

1. **Teile das Problem der Größenordnung n in Probleme geringerer Größenordnung.**

2. **Löse die Teilprobleme.**

3. **Fasse die Lösung der Teilprobleme zur Lösung des Gesamtproblems zusammen.**

Falls die Teilprobleme in Schritt 2 unabhängig voneinander gelöst werden können, können sie auch parallel ausgeführt werden, sofern die Ressourcen für die parallele Ausführung vorhanden sind (SMP Rechner, MPP Systeme).

Neben der parallelen Ausführung kann es auch sinnvoll sein, Pipelining durchzuführen:

Für den Start der Lösung der Teilprobleme muß die Aufteilung des Gesamtproblems in Teilprobleme nicht notwendig abgeschlossen sein.

Für den Start der Zusammenfassung der Lösungen der Teilprobleme zur Lösung des Gesamtproblems muß die Lösung der Teilprobleme nicht notwendig abgeschlossen sein.

12. Programmierparadigmen

Imperative Programmierung (Strukturierte Programmierung)

Programmierung mit den Ablaufstrukturen

- **Sequenz**
- **Selektion**
- **Iteration**

Jeder Algorithmus kann mittels imperativer Programmierung umgesetzt werden.

Programmiersprachen: C, C++

Logische Programmierung:

Charakteristka:

- **Fakten und Regeln**
- **Rekursion**

Jeder Algorithmus kann mittels logischer Programmierung umgesetzt werden.

Programmiersprache: Prolog

Funktionale Programmierung:

Charakteristka:

- **Listenverarbeitung**
- **Rekursion**

Jeder Algorithmus kann mittels funktionaler Programmierung umgesetzt werden.

Programmiersprache: Lisp

Objektorientierte Programmierung

Betrachtung von Algorithmen und Datenstrukturen als Einheit

Programmiersprachen: C++, Java

13. Elementare Datentypen

Zahlen

<u>Natürliche Zahlen, dargestellt in Stellenwertsystemen</u>

Berechnung mit Horner – Schema

Sei b Basis und a(n),...,a(0) eine natürliche Zahl zur Basis b.

Dann wird der Wert dieser Zahl im Dezimalsystem bestimmt durch den Folgenden Algorithmus:

1. **Setze Wert auf a(n)**
2. **Setze i auf n-1**
3. **Solange i größer oder gleich 0 ist, führe folgende Schritte durch:**
4. **Setze Wert auf Wert*b + a(i)**
5. **Setze i auf i-1**

Umrechnung von Dezimal n in einen Wert zur Basis b sieht so aus:

1. **Setze i auf 0.**
2. **Solange n größer als 0 ist, führe folgende Schritte durch:**
3. **Setze a(i) auf n mod b.**
4. **Setze n auf n div b.**

Darstellung ganzer Zahlen

Vorzeichen-Absolutbetrag Darstellung

- **Null nicht eindeutig dargestellt**
- **Subtraktion führt zu Fallunterscheidungen**

Zweierkomplement

- **Eindeutigkeit der Null**
- **Es gilt immer: -x =inv(x)+1**
- **Einfachere Addition und Subtraktion**
- **Überlaufbit einfach zu interpretieren**

Gleitkommazahlen

ANSI-IEEE Standard 754-1985

1. **Finde Zahlen m,e mit abs(x) = (1+m) * 2^e mit 0<=m<1 (genormte Darstellung).**
2. **Codiere das Vorzeichen mit 0 für positiv, 1 für negativ.**
3. **Codiere die Mantisse in 23 bit.**
4. **Codiere den Exponenten in 8 bit als e+127**
5. **Ordne die Bits in der Reihenfolge Vorzeichen –Mantisse – Exponent an.**

Sonderfälle:

Exponent –127, Mantisse 0: 0

Exponent 128, Mantisse 0: „unendlich“

Exponent 128, Mantisse <>0: Fehler

Gleitkommazahlen sind nicht exakt!

Vergleiche sollten nicht mit „=“ durchgeführt werden!

Rechenfehler sind einzukalkulieren!

Absoluter Fehler vs. Relativer Fehler

Darstellungsfehler vs. Rundungsfehler

Relativer Fehler bei Addition und Subtraktion: Verdopplung möglich

Relativer Fehler bei Multiplikation, Division: Verdreifachung möglich

Strings (Zeichenketten)

Grundlage ist ein Zeichensatz. Der Zeichensatz bestimmt die zulässigen Zeichen und die Ordnung auf diesen Zeichen. Häufig ist die Ordnung durch die Nummerierung im Zeichensatz gegeben.

Strings werden häufig (je nach Zeichensatz) mit 1 bzw. 2 byte pro Zeichen gespeichert. Dabei stellt sich die Frage, ob Strings mit fester oder variabler Länge gespeichert werden oder welche sonstigen Verfahren zur Kompression sinnvoll eingesetzt werden können. Mit derartigen Verfahren werden wir uns noch beschäftigen.

14. Komplexe Datentypen

Strukturierung von Daten analog zu Ablaufstrukturen

Sequenz => Zusammenfassung von Daten i.a. verschiedener Datentypen zu einem Datentyp

Records, Strukturen

Beispiel:

Datentyp Adresse ist die Zusammenfassung von Strasse (String), Hausnummer (Integer), Postleitzahl (Integer), Ort (String)

Selektion => Überlagerung von Datentypen; Zugriff erfolgt via Bedingung

Variante Records, Unions

Beispiel:

Überlagerung von kartesischen Koordianten und Polarkoorsinaten für Vektoren in der Ebene

Ziel: Speicherplatz sparen

Iteration => Zusammenfassung von Daten gleichen Datentyps zu einem Datentyp (Folge)

Arrays, Listen, Dateien

(Statische) Arrays sind i.a. Bestandteile von Programmiersprachen, Listen werden über Zeiger implementiert (z.B.: Jedes Listenelement enthält Daten und einen Zeiger auf den Nachfolger, die Liste wird durch einen Zeiger auf das erste Listenelement repräsentiert), für den Dateizugriff gibt es i.a. Standardbibliotheken.

Unterschiede im Zugriff: Direktzugriff (Array) vs. sequentieller Zugriff (Liste)

Beispiel:

Folge von Personendaten