Script **generated by TTT**

Title: Einf_HF (25.06.2012)

Date: Mon Jun 25 14:16:54 CEST 2012

Duration: 91:39 min

Pages: 31

Datenstrukturen und Algorithmen

In diesem Kapitel werden einige Klassen von Algorithmen vorgestellt, insbesondere Suchverfahren und Sortierverfahren.

- Fragestellungen des Abschnitts:
  - Welche Möglichkeiten gibt es, Datenmengen im System darzustellen?
  - Welche Möglichkeiten gibt es, in Datenmengen zu suchen?
  - Welche Möglichkeiten gibt es, Datenmengen zu sortieren?
  - Was versteht man unter der Komplexität eines Algorithmus?

Voraussetzung: Ordnung auf Datenelementen. Entsprechend der Ordnung in der Reihung gespeichert (aufsteigend oder absteigend).

Sei int A[n] eine Reihung mit n Elementen, das aufsteigend sortiert ist.

Gesucht wird ein Element mit dem Wert x; gesucht wird x im Bereich A[0] bis A[n-1].

1. wähle m zwischen 0 und n-1; man wird m ungefähr in der Mitte zwischen 0 und n-1 wählen.
2. wenn A[m] == x, dann sind wir fertig, gib m als Ergebnis aus.
3. wenn x < A[m], dann suche weiter im Bereich A[0] bis A[m-1];
4. wenn x > A[m], dann suche weiter im Bereich A[m+1] bis A[n-1]

Doppelte Zahl von Elementen: ein zusätzlicher Vergleich. Bei linearer Suche: Verdopplung des Aufwandes.

Suchverfahren

Gegeben: Menge von Datensätzen. Gesucht: Datensatz mit bestimmter Eigenschaft.

Mengen von Datensätzen

Üblicherweise in Reihung oder Liste gespeichert.

Annahme für die folgenden Verfahren: Daten in einer Reihung gespeichert.

Voraussetzung: Ordnung auf Datenelementen. Entsprechend der Ordnung in der Reihung gespeichert (aufsteigend oder absteigend).

Sei int A[n] eine Reihung mit n Elementen, das aufsteigend sortiert ist.

Gesucht wird ein Element mit dem Wert x; gesucht wird x im Bereich A[0] bis A[n-1].

1. wähle m zwischen 0 und n-1; man wird m ungefähr in der Mitte zwischen 0 und n-1 wählen.
2. wenn A[m] == x, dann sind wir fertig, gib m als Ergebnis aus.
3. wenn x < A[m], dann suche weiter im Bereich A[0] bis A[m-1];
4. wenn x > A[m], dann suche weiter im Bereich A[m+1] bis A[n-1]

Doppelte Zahl von Elementen: ein zusätzlicher Vergleich. Bei linearer Suche: Verdopplung des Aufwandes.

Gegeben: Menge von Datensätzen. Gesucht: Datensatz mit bestimmter Eigenschaft.

Mengen von Datensätzen

Üblicherweise in Reihung oder Liste gespeichert.

Annahme für die folgenden Verfahren: Daten in einer Reihung gespeichert.

Lineare Suche

Binäre Suche

Suchverfahren Animation

Sortierverfahren

Ziel: Datenmenge entsprechend vorgegebenen Ordnungskriterium sortieren.

Sortierverfahren - Beispiel Bubble Sort

Rekursives Sortierverfahren - Beispiel Quicksort

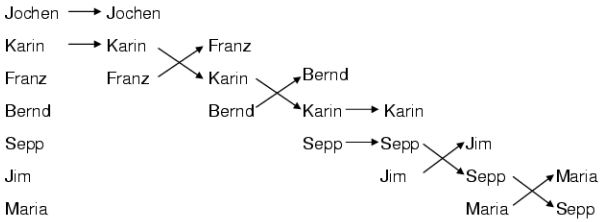
Sortierverfahren Animation

Sortieren durch Vertauschen von benachbarten Elementpaaren.

Mehrere Durchgänge. Bei n Elementen maximal n Durchgänge.

Beispiel: alphabetisches Sortieren einer Menge von Namen

1. Sortierdurchgang mit jeweils paarweisen Vergleich



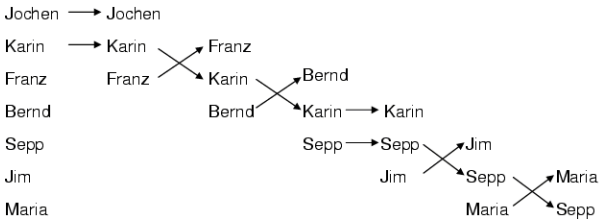
Original	nach 1	nach 2	nach 3	nach 4 Schritten
Jochen	Jochen	Franz	Bernd	Bernd
Karin	Franz	Bernd	Franz	Franz
Franz	Bernd	Jochen	Jim	Jim
Bernd	Karin	Jim	Jochen	Jochen
Sepp	Jim	Karin	Karin	Karin
Jim	Maria	Maria	Maria	Maria
Maria	Sepp	Sepp	Sepp	Sepp

Sortierverfahren - Beispiel Bubble Sort

Mehrere Durchgänge. Bei n Elementen maximal n Durchgänge.

Beispiel: alphabetisches Sortieren einer Menge von Namen

1. Sortierdurchgang mit jeweils paarweisen Vergleich



Original	nach 1	nach 2	nach 3	nach 4 Schritten
Jochen	Jochen	Franz	Bernd	Bernd
Karin	Franz	Bernd	Franz	Franz
Franz	Bernd	Jochen	Jim	Jim
Bernd	Karin	Jim	Jochen	Jochen
Sepp	Jim	Karin	Karin	Karin
Jim	Maria	Maria	Maria	Maria
Maria	Sepp	Sepp	Sepp	Sepp

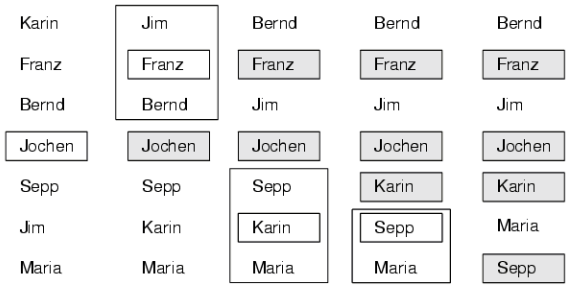
Sortierverfahren

Ziel: Datenmenge entsprechend vorgegebenen Ordnungskriterium sortieren.

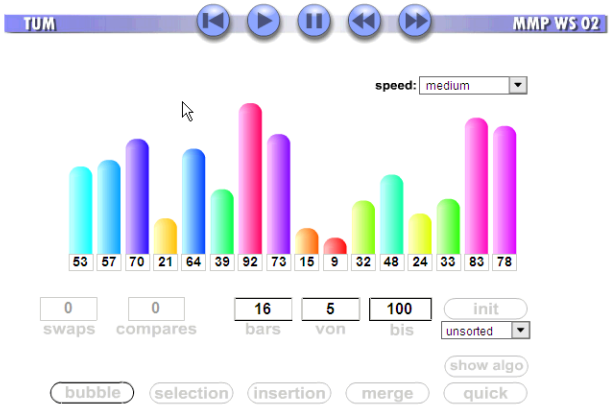
Sortierverfahren - Beispiel Bubble Sort

Rekursives Sortierverfahren - Beispiel Quicksort

Sortierverfahren Animation



Sortierverfahren Animation



Ziel: Datenmenge entsprechend vorgegebenen Ordnungskriterium sortieren.

Sortierverfahren - Beispiel Bubble Sort

Rekursives Sortierverfahren - Beispiel Quicksort

Sortierverfahren Animation

Sortierverfahren

Ziel: Datenmenge entsprechend vorgegebenen Ordnungskriterium sortieren.

Sortierverfahren - Beispiel Bubble Sort

Rekursives Sortierverfahren - Beispiel Quicksort

Sortierverfahren Animation

Generated by Targeteam

Komplexität

Algorithmus kann Aufwand erfordern, der"nicht vertretbaren" ist. Bei algorithmischer Lösung eines Problems ist daher auch Effizienz wesentlich.

Komplexität von Algorithmen

Ein Algorithmus ist umso effizienter, je geringer der Aufwand zu seiner Abarbeitung ist.

Aufwand bezieht sich auf bestimmte Ressourcen, z.B. Rechenzeit, Speicherplatz, Anzahl der Geräte

Je nach Ressource verschiedene Komplexitätsmaße. Wichtigste: Zeitkomplexität, Speicherplatzkomplexität.

Zu unterscheiden: Komplexität eines Algorithmus / eines Problems. Problem: zu erreichendes Ziel. Algorithmus: Vorgehen. Komplexität Problem = Komplexität effizientester bekannter Algorithmus.

Komplexitätsmaß für Algorithmus: Funktion abhängig von Größe der Eingabe. Misst Aufwand der Verarbeitung relativ zur zu verarbeitenden Information.

Beispiel

Liste Sortieren: Komplexitätsmaß abhängig von Anzahl der zu sortierenden Elemente; Brechen eines Schlüssels: Komplexitätsmaß meist abhängig von Schlüssellänge.

Algorithmen bewertet man relativ zu ihrer Komplexität.

Komplexitätsklassen

Relative Größenordnung (abzüglich konstanter Faktor) in Abhängigkeit von Zahl n der Wertelemente (bestimmt durch Kontext, z.B. Anzahl der Eingabeelemente, Anzahl der zu untersuchenden Datenelemente).

Definition

Sei f: $N \to N$ eine Funktion. Die Komplexitätsklasse O(f) ist definiert durch:

$O(f) := \{ g: N \to N \mid \exists\ c > 0, m \geq 0: \forall\ n > m: 0 \leq g(n) \leq cf(n) \}$

O-Notation dient dazu, das asymptotische Wachstum einer Funktion abzuschätzen.

- Sei t(n) die Funktion zur Bestimmung der Laufzeit des Programms.
- mit wachsendem n gewinnt die höchste Potenz von n in t(n) an Bedeutung.

Übliche Komplexitätsklassen

Nicht-polynomiale Probleme gelten als "hart".

Hauptsächlich verwendete Komplexitätsklassen

logarithmisch O(log n)

linear O(n)

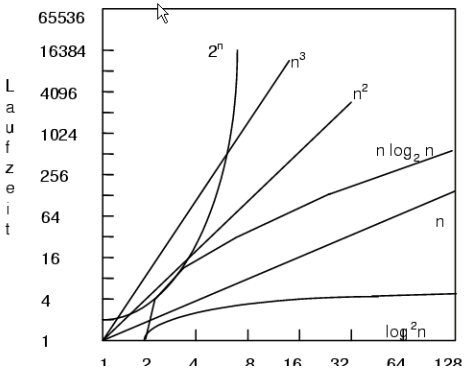
O(n) ⇒ Komplexität ist z.B. 3n + 10;

überlinear O(n log n)

quadratisch $O(n^2)$

polynomial $O(n^k)$, k > 2

exponentiell $O(k^n)$, k >= 2



Übliche Komplexitätsklassen

logarithmisch O(log n)

linear O(n)

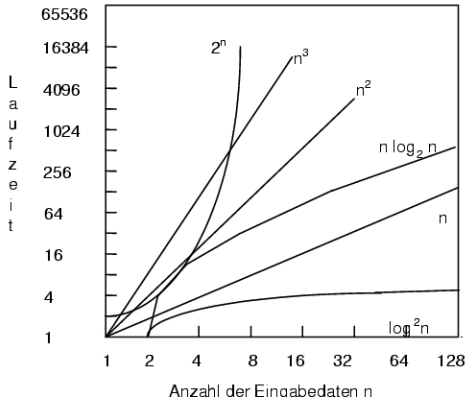
O(n) ⇒ Komplexität ist z.B. 3n + 10;

überlinear O(n log n)

quadratisch $O(n^2)$

polynomial $O(n^k)$, k > 2

exponentiell $O(k^n)$, k >= 2



Komplexitätsklassen

Relative Größenordnung (abzüglich konstanter Faktor) in Abhängigkeit von Zahl n der Wertelemente (bestimmt durch Kontext, z.B. Anzahl der Eingabeelemente, Anzahl der zu untersuchenden Datenelemente).

Definition

Sei f: N → N eine Funktion. Die Komplexitätsklasse O(f) ist definiert durch:

$O(f) := \{ g: N \to N \mid \exists\ c > 0, m \geq 0: \forall\ n > m: 0 \leq g(n) \leq cf(n) \}$

O-Notation dient dazu, das asymptotische Wachstum einer Funktion abzuschätzen.

- Sei t(n) die Funktion zur Bestimmung der Laufzeit des Programms.
- mit wachsendem n gewinnt die höchste Potenz von n in t(n) an Bedeutung.

Übliche Komplexitätsklassen

Nicht-polynomiale Probleme gelten als "hart".

Algorithmus kann Aufwand erfordern, der"nicht vertretbaren" ist. Bei algorithmischer Lösung eines Problems ist daher auch Effizienz wesentlich.

Komplexität von Algorithmen

Ein Algorithmus ist umso effizienter, je geringer der Aufwand zu seiner Abarbeitung ist.

Aufwand bezieht sich auf bestimmte Ressourcen, z.B. Rechenzeit, Speicherplatz, Anzahl der Geräte

Je nach Ressource verschiedene Komplexitätsmaße. Wichtigste: Zeitkomplexität, Speicherplatzkomplexität.

Zu unterscheiden: Komplexität eines Algorithmus / eines Problems. Problem: zu erreichendes Ziel. Algorithmus: Vorgehen. Komplexität Problem = Komplexität effizientester bekannter Algorithmus.

Komplexitätsmaß für Algorithmus: Funktion abhängig von Größe der Eingabe. Misst Aufwand der Verarbeitung relativ zur zu verarbeitenden Information.

Beispiel

Liste Sortieren: Komplexitätsmaß abhängig von Anzahl der zu sortierenden Elemente; Brechen eines Schlüssels: Komplexitätsmaß meist abhängig von Schlüssellänge.

Algorithmen bewertet man relativ zu ihrer Komplexität.

Komplexitätsklassen

"Vorgehen bei der Entwicklung von Softwaresystemen". Vorgehensmodelle für die Entwicklung von Programmen, Modelle für Analyse und Entwurf.

- Fragestellungen des Abschnitts:
  - Welche Kategorien von Programmiersprachen gibt es ?

    interpretierte und übersetzte Sprachen
  - Wie kann man bei der Konzeption und der Realisierung eines Software-Programms geeignet vorgehen?

    Modellierung der verschiedenen Aspekte, z.B. Daten, Abläufe und Interaktion mit dem Benutzer.

Programmiersprachen

Software-Engineering

Bereitstellung von höheren Programmiersprachen, wie z.B. C, Java, die die Erstellung von Programmen erleichtern.

Programme in höherer Programmiersprache müssen in Maschinensprache transformiert werden.

Interpretierer und Übersetzer

Scriptsprachen

Auswahl einer Programmiersprache

Interpretierer und Übersetzer

"Interpretierer "(Interpreter): Programmiersystem, das Anweisungen schrittweise zergliedert und Schritte sofort ausführt. Jede Programmanweisung wird separat betrachtet und Maschinensprache-Unterprogramm für Realisierung der Anweisung aufgerufen.

Alternative: "Übersetzer"(Compiler): wandeln komplettes Programm in Maschinensprache um.

Vorteile von Interpretierern

- Einfacher zu realisieren
- Quellprogramm-bezogenes Testen

Nachteile von Interpretierern

- Ineffizient
- Fehler erst zur Laufzeit entdeckt

Beispiele für Interpretierer(sprachen)

- Basic
- Kommandosprachen von Betriebssystemen, z.B. Shell in DOS Eingabefenster
- Skriptsprachen (z.B. JavaScript)

Scriptsprachen

Für Endanwender: programmieren innerhalb einer Applikation.

Ziele von Scriptprogrammen

- Anpassen der Applikation an eigene Umgebung
- Definition von Macros zur Erleichterung der Eingabe.
- zur automatische Aktualisierung, z.B. von Feldern in einer Tabellenkalkulation
- zur Einbettung von Funktionen in Web-Seiten

Funktionen/Programme nicht übersetzt, sondern bei Aufruf direkt ausgeführt. Verwendung von Interpretierer.

Tabellenkalkulation

Rechenblätter mit Zellen, die in Zeilen und Spalten organisiert sind. Zellen enthalten

Daten verschiedener Sorten (Zahl, Währung, Datum, Text, ...)

Formeln, die aus vordefinierten Funktionen zusammengesetzt sind (Summe, Mittelwert, Runden, ...)

variable Daten werden mit Hilfe von Verweisen auf andere Zellen in Formeln eingegeben, z.B. Summe (A1:A4), Summe(A1;B1;C1)

nur Funktionen mit einer Datenausgabeleitung, d.h. es gibt keinen Datenspeicher

⇒ nicht jeder Algorithmus kann in einer Tabellenkalkulation dargestellt werden.

Auswertung von Formeln bei Zellenänderung.

Programmierung von Kommandobuttons, Textfelder und Dialogboxen.

Technische Kriterien

Angemessenheit für Problem

Übersichtlichkeit, Strukturiertheit, Selbst-Dokumentation

Unterstützung fortschrittlicher Softwareentwicklungskonzepte

Unterstützung/Einbindung von Systemdiensten und Systemstrukturen (des Betriebssystems)

Qualität und Leistungsfähigkeit des Übersetzers

Vorhandene Entwicklungs- und Testhilfen; vorgefertigte Routinen

Übertragbarkeit, Wiederverwendbarkeit von Programmteilen (Module)

Nicht-technische Kriterien

Verträglichkeit mit vorhandenen Daten und Programmen

Erfahrung der Benutzer

Wünsche des Auftraggebers

Lizenzgebühren und -bedingungen

Güte der Lieferanten-Unterstützung

Standardisierung

Zukünftige Bedeutung der Sprache

Auswahl einer Programmiersprache

Prinzipiell Programmiersprachen gleichwertig: alle Algorithmen formulierbar. Praktische Unterschiede helfen bei Auswahl für Entwicklungsprojekt.

Kriterien für die Auswahl einer Programmiersprache

Empfehlungen

wenn ...	... dann
einfache Programme, Anwendungserweiterungen	Basic, Visual Basic, Python, Perl
Datenbank-Anwendung und komplexe Programmlogik	C, C++ , Java
Datenbank-Anwendung und einfache Programmlogik	SQL, Reportgenerator
Technisch-wissenschaftliche Anwendung und (Datenbank oder komplexe E/A-Strukturen) und Portabilität	C, C++, Java
(System-Software oder PC-Anwendung) und Portabilität	C, C++, Java
Künstliche Intelligenz-Anwendung	Prolog, LISP
Internet-Anwendung und Portabilität	Java, PHP, Python

Softwareerstellung als Ingenieurdisziplin.

Software/Engineering - Definition des Ideals

Die Aufstellung und Befolgung guter Ingenieur-Grundsätze und Management-Praktiken, sowie die Entwicklung und Anwendung zweckdienlicher Methoden und Werkzeuge, mit dem Ziel, mit vorhersagbaren Mitteln, System- und Software-Produkte zu erstellen, die hohe, explizit vorgegebene Qualitätsansprüche erfüllen (nach A. Marco & J. Buxton, 1987)

Komplexität von Software-Projekten

Vorgehensmodelle

Strukturierte Programmierung

Modellierung

Modelle für Analyse und Entwurf

Software-Qualitätssicherung

Maße für den Umfang von Software

Zahl der Quelltextzeilen der Programme, aus denen das Softwareprodukt besteht (LOC = Lines of Code).

Zeit, die benötigt wird, um eine Programm zu erstellen (Messung in Bearbeiter-Jahre (BJ)).

Klassifikation von Software-Projekten

Projektklasse	Quelltext-Zeilen (LOC)	Bearbeitungsaufwand (BJ)
sehr klein	1 - 1.000	0 - 0,2
klein	1.000 - 10.000	0,2 - 2
mittel	10.000 - 100.000	2 - 20
groß	100.000 - 1 Mio.	20 - 200
sehr groß	1 Mio - ...	200 - ...

Beispiele

Projektklasse	Quelltext-Zeilen (LOC)
Windows XP	ca. 50 Millionen
Windows Vista	ca. 70 Millionen
Linux Kernel 2.6	ca 5.2 Millionen
Mac OS X 10	ca. 85 Millionen

Nutzung von LOC, um Programmierfortschritt zu messen, ist umstritten:" Measuring programming

Komplexität von Software-Projekten

Projektklasse	Quelltext-Zeilen (LOC)	Bearbeitungsaufwand (BJ)
sehr klein	1 - 1.000	0 - 0,2
klein	1.000 - 10.000	0,2 - 2
mittel	10.000 - 100.000	2 - 20
groß	100.000 - 1 Mio.	20 - 200
sehr groß	1 Mio - ...	200 - ...

Beispiele

Projektklasse	Quelltext-Zeilen (LOC)
Windows XP	ca. 50 Millionen
Windows Vista	ca. 70 Millionen
Linux Kernel 2.6	ca 5.2 Millionen
Mac OS X 10	ca. 85 Millionen

Nutzung von LOC, um Programmierfortschritt zu messen, ist umstritten:" Measuring programming progress by lines of code is like measuring aircraft building progress by weight (Bill Gates)."

Hauptanforderungen bei der Softwareentwicklung

Ergebnis ist zuverlässig (fehlerfrei), verhält sich gemäß Anforderungen.

Kann später problemlos geändert werden.

Softwareerstellung als Ingenieurdisziplin.

Software/Engineering - Definition des Ideals

Die Aufstellung und Befolgung guter Ingenieur-Grundsätze und Management-Praktiken, sowie die Entwicklung und Anwendung zweckdienlicher Methoden und Werkzeuge, mit dem Ziel, mit vorhersagbaren Mitteln, System- und Software-Produkte zu erstellen, die hohe, explizit vorgegebene Qualitätsansprüche erfüllen (nach A. Marco & J. Buxton, 1987)