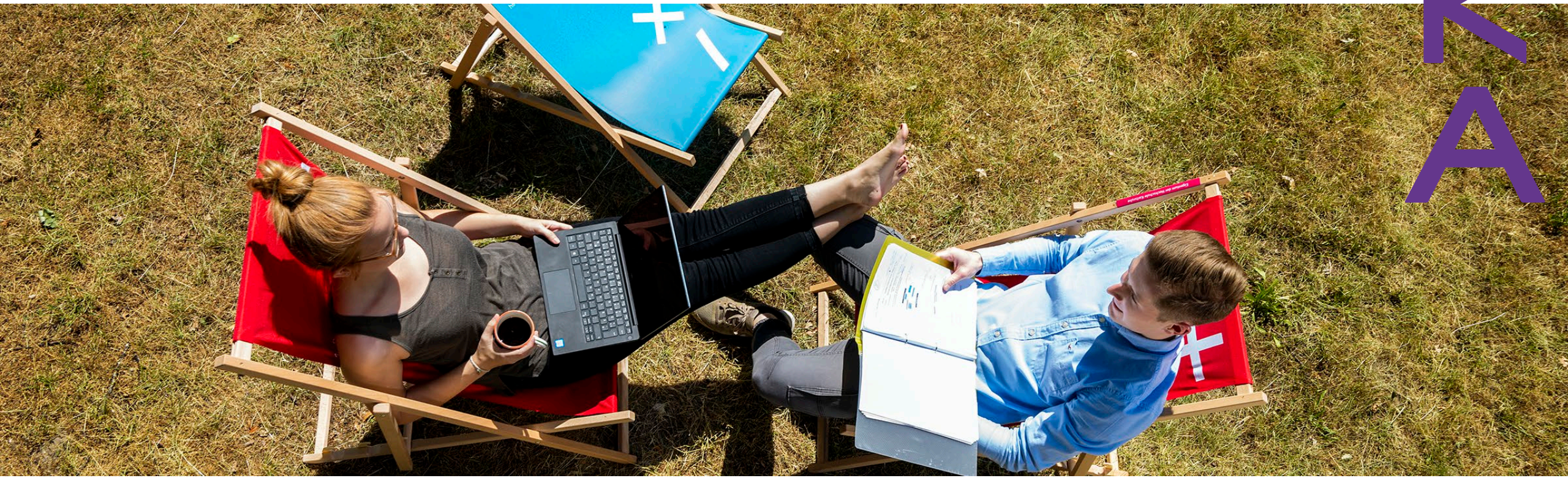
Hochschule Karlsruhe
University of
Applied Sciences

Fakultät für
Informatik und
Wirtschaftsinformatik

DSCB130 | 3.1 Algorithmen



Matthias Mruzek-Vering M.Sc.

Foto: Quirin Leppert

Themenblock 3 Programmieren

Lehrinhalte

3.1 Algorithmen

- Begriffsklärung und Historie
- Schema zur Beschreibung von Algorithmen
- Anforderungen und Algorithmen
- Rekursion und Backtracking
- Paradigmen
- Teile-und-Herrsche-Prinzip
- Versuch-und-Irrtum-Prinzip
- Heuristik
- Gierige Algorithmen

3.1 Algorithmen

Begriffsklärung und Historie

Algorithmus:

1. Eine präzise, d.h. in einer festgelegten Sprache abgefasste, endliche Beschreibung eines allgemeinen Verfahrens unter Verwendung elementarer Verarbeitungsschritte zur Lösung einer gegebenen Aufgabe.

2. Lösungsverfahren in Form einer Verfahrensanweisung, die in einer wohldefinierten Abfolge von Schritten zur Problemlösung führt.

Vgl. Gabler Wirtschaftslexikon

Algorithmus nach DIN 44300: Vorschrift, nach der Ergebnisse systematisch ausgehend von Eingabedaten erzeugt (umgewandelt) werden können. Eine Abfolge von Anweisungen wird als Programm bezeichnet.

Algorithmus Umgangssprachlich

+ Beschreibung einer systematischen Vorgehensweise zur Lösung eines Problems.
+ Was muss wann und wie getan werden, um das Problem zu lösen?

3.1 Algorithmen

Begriffsklärung und Historie

Historie

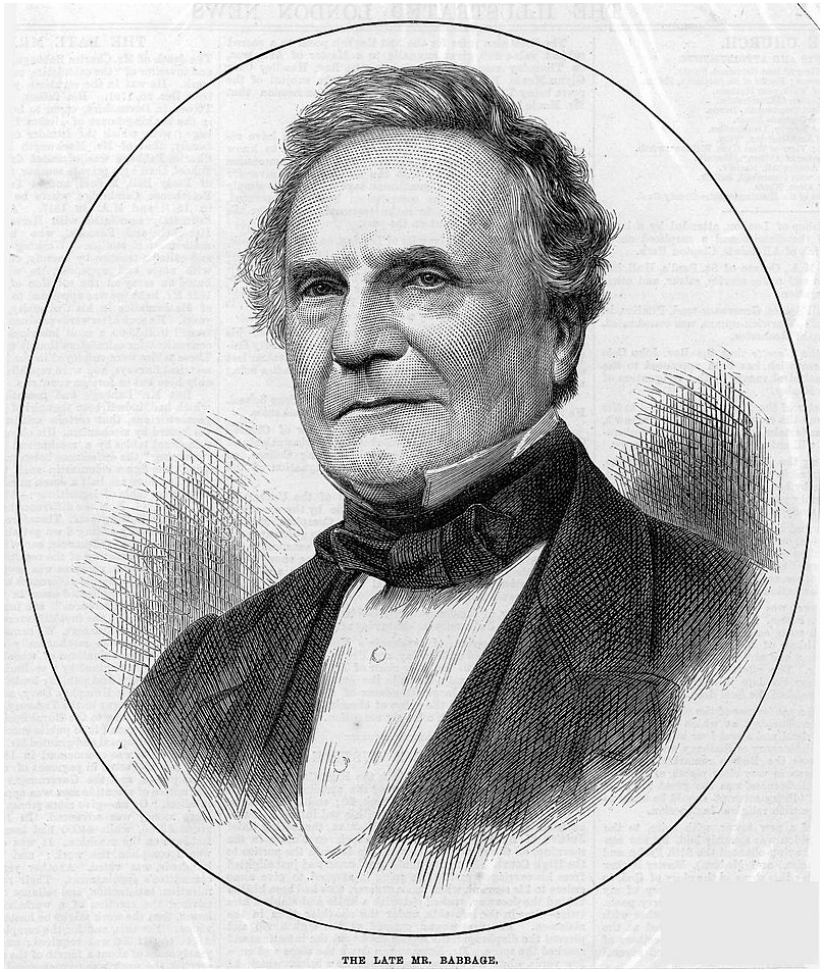
+ Abu Dscha'far Muhammad ibn Musa al-Chwārizmī
  بوجعفر محمد بن موسای خوارزمی (~ 783-850)
+ Lehrbuch: Über das Rechnen mit indischen Ziffern
+ Lateinische Übersetzung im Mittelalter beginnt mit:
  Dixit Algorismi (Algorismi hat gesagt...)
+ Weiterentwicklung im Mittelalter zu Algorithmus,
  Die Kunst des Rechnens mit indo-arabischen Ziffern
+ Vgl. Wikipedia

3.1 Algorithmen

Begriffsklärung und Historie

Historie

+ Charles Babbage veröffentlicht 1837 erste Beschreibungen seiner Analytical Engine, den ersten programmierbaren Computer.
+ Vgl. Wikipedia Charles Babagge, Wikipedia Analytical Engine



3.1 Algorithmen

Begriffsklärung und Historie

Historie

+ Ada Lovelace entwickelt 1843 den ersten, für einen Computer gedachten, Algorithmus zur Berechnung der Bernoulli-Zahlen für die *Analytical Engine*.

+ $B_n = -\frac{1}{n+1}\sum_{k=0}^{n-1}\left(\frac{n+1}{k}\right)B_k$

Weitere berühmte Frauen in der Informatik

+ Grace Hopper, Constanze Kurz, etc.

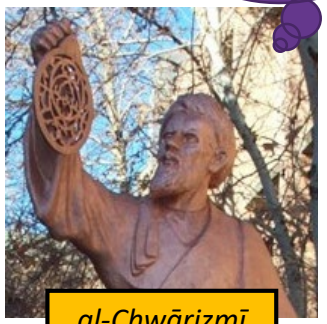
+ Vgl. Wikipedia Ada Lovelace, Frauen in der Informatik, IT-Girls

3.1 Algorithmen

Begriffsklärung und Historie

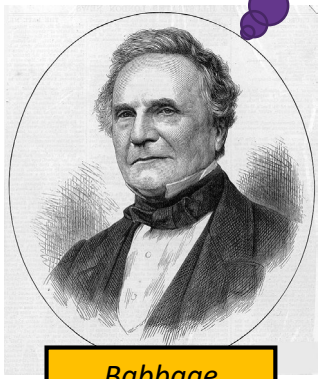
Historie kompakt

Wie löst man systematisch Rechenaufgaben?



al-Chwārizmī

Wie kann eine Rechenmaschine funktionieren?



Babbage

Wie kann man eine Rechenmaschine dazu nutzen, systematisch ein Problem zu lösen?

Lovelace

3.1 Algorithmen

Schema zur Beschreibung von Algorithmen

Was tut das Verfahren (geeigneter Name) **Womit** / **Welche** Informationen werden benötigt?

Wie sieht der Ablauf der Lösung aus?

Bestandteile einer Algorithmen-Beschreibung

+ „Leitgedanken des Autors“ als beigefügtem Kommentar
+ Ausgangssituation z. B. Eingaben
+ Aktionen z. B. Handlungsanweisungen
+ Verwendung anderer Algorithmen: Anweisungen
+ Bedingungen/Alternativen: wenn… dann… sonst…
+ Wiederholungen: wiederhole… bis… solange… tue…
+ Reihenfolgen: Anordnung der Aktionen, Bedingungen und Wiederholungen
+ Endsituationen: Ausgaben

3.1 Algorithmen

Anforderungen und Algorithmen

Grundlegende Anforderungen

+ **Eindeutigkeit:** Jeder Befehl muss zu einem eindeutigen Ergebnis führen.
+ **Korrektheit:** Ein Algorithmus muss zum richtigen Ergebnis führen. Die Korrektheit muss überprüfbar sein.
+ **Endlichkeit:** Nach einer endlichen Anzahl von Schritten muss der Lösungsalgorithmus
  a) vollständig beschrieben und b) technisch (maschinell, mechanisch, informationstechnisch) umsetzbar sein.
  Der Beschreibungscode muss begrenzt und die Ausführungszeit muss endlich sein.
+ **Universalität:** Algorithmus ist jederzeit auf alle möglichen (definierten) Daten anwendbar.

Aus praktischer Sicht meist zusätzlich gefordert

+ **Determinismus:** Bei gleichen Eingabewerten und unter gleichen Randwerten des Programms, muss jeweils das gleiche Ergebnis erzielt werden. Hierbei können die Rand-Bedingungen und -werte der verarbeitenden Maschine variieren.

3.1 Algorithmen

Anforderungen und Algorithmen

Merkmale

+ Algorithmen sind Anleitungen zur systematischen Problemlösung
+ Programmiersprachen sind „Fremdsprachen“ mit sehr geringem Wortschatz, die Problemlösungen formulieren lassen (sehr präzise)
+ Programme sind **Problemlösungen in Programmiersprachen formuliert**
+ Programmieren lernen bedeutet
  - Lernen, Probleme systematisch zu lösen
  - Eine einfache, wenngleich etwas seltsame, Fremdsprache mit weniger als 100 Vokabeln lernen

3.1 Algorithmen

Anforderungen und Algorithmen

Beispiele für Algorithmen aus dem Alltag

+ Finden eines Weges
+ Bezahlen an einer Kasse
+ Aufbauen von Möbeln
+ Suchen einer Telefonnummer zu einem Namen im Telefonbuch
+ Routenberechnung in Navigationsgeräten
+ Sortieren von Gegenständen oder Informationen
+ Lösen von Sudokus

3.1 Algorithmen

Anforderungen und Algorithmen

Beispiel aus der Mathematik

+ $n \cdot \frac{(n+1)}{2} = n + (n-1) + \cdots + 3 + 2 + 1$
+ $n^2 = 1 + 3 + 5 + 7 + \cdots (2n-1)$
+ $\ln(10) = 2{,}30258509299$
+ Euklidischer Algorithmus: Bestimmung des größten gemeinsamen Teilers (ggT) zweier Zahlen
+ Gaußscher Algorithmus: Verfahren zur Lösung linearer Gleichungen
+ pq-Formel: Verfahren zur Auflösung von quadratischen Gleichungen in Normalform
+ Zins- und Rentenberechnung

Sortier- und Suchverfahren

+ Bubblesort, Mergesort, Quicksort, Heapsort, Shellsort, etc.

3.1 Algorithmen

Anforderungen und Algorithmen

Beispiel Radwechsel am Kfz

Was Ein Rad wechseln	**Womit** Auto A, altes Rad Ra, neues Rad Rn

Wie

An Auto A tue:

1. Radschrauben bei Rad Ra mit Drehmomentwerkzeug anlösen
2. Wagenheber bei Rad Ra ansetzen und Auto damit anheben
3. Schrauben lösen
4. Rad Ra von der Nabe nehmen
5. Rad Rn auf die Nabe setzen
6. Schrauben anziehen
7. Auto wieder absenken
8. Radschrauben mit Drehmomentwerkzeug festziehen
9. Wagenheber entfernen

3.1 Algorithmen

Anforderungen und Algorithmen

Beispiel Radwechsel am Kfz aller Räder

Was Alle vier Räder wechseln	**Womit** Auto A, neue Räder Rn1..Rn4

Wie

An Auto A tue:

1. Ein Rad wechseln mit A, Altes Rad Ra1, neues Rad Rn1
2. Ein Rad wechseln mit A, Altes Rad Ra2, neues Rad Rn2
3. Ein Rad wechseln mit A, Altes Rad Ra3, neues Rad Rn3
4. Ein Rad wechseln mit A, Altes Rad Ra4, neues Rad Rn4

3.1 Algorithmen

Anforderungen und Algorithmen

Beispiel Radwechsel am Kfz, jedoch schlichter

Was Alle vier Räder wechseln	**Womit** Auto A, neue Räder Rn1..Rn4

Wie

für <n> in [1..4]:
  tue Rad wechseln mit A, Altes Rad Ra<n>, neues Rad Rn<n>

3.1 Algorithmen

Anforderungen und Algorithmen

Verständnis

+ Algorithmisches Denken und Programmieren kann man nicht aus einem Buch lernen
+ Typischerweise hohe initiale Hürde
+ „komische Sprache“
+ Technisches Zeug
+ Die eigentliche Schwierigkeit ist präzises Denken und formulieren!

3.1 Algorithmen

Rekursion und Backtracking

Rekursion

+ Formales Prinzip, demzufolge bei der Beschreibung eines Sachverhalts auf den zu beschreibenden Sachverhalt selbst Bezug genommen wird
+ Programmierung
  - Vorgehensweise, bei der rekursive Unterprogramme eingesetzt werden
  - Manche Algorithmen können mithilfe der Rekursion wesentlich kompakter und übersichtlicher dargestellt werden, insbesondere beim Sortieren, Suchen sowie bei der Benutzung von Bäumen.
  - Voraussetzung ist, dass die Programmiersprache rekursive Unterprogramme zulässt
+ Vgl. Gabler Wirtschaftslexikon

Beispiele

+ Sudokus lösen, Türme von Hanoi (TOH), Quicksort, Baumtraversierung
+ Weiterführendes Material, siehe [5] S. 545 ff.

3.1 Algorithmen

Rekursion und Backtracking

Backtracking

+ Suchmethode nach folgendem Prinzip:
  - An denjenigen Punkten des Suchvorgangs, an denen zur Fortsetzung der Suche eine Auswahlentscheidung zwischen mehreren Möglichkeiten getroffen werden muss, wird zunächst der aktuelle Zustand festgehalten, bevor man die verschiedenen Möglichkeiten verfolgt.
  - Durch das Festhalten des Zustands ist gewährleistet, dass ein Rücksprung möglich und von den richtigen Vorbedingungen ausgegangen werden kann.

Beispiele

+ Verwendung in der Künstlichen Intelligenz (KI) und bei rekursiver Programmierung
+ 8-Damen-Problem
+ Vgl. Gabler Wirtschaftslexikon
+ Weiterführendes Material: [5] S. 548 ff.

3.1 Algorithmen

Paradigmen

Allgemein

+ Anerkannte Orientierungsmodelle zur erfolgreichen Problemlösung

Merkmale im Kontext Informatik

+ Fundamentales Konzept
+ Prägt entscheidend die Sprachstruktur einer Programmiersprache oder eine Vorgehensweise zur Problemlösung
+ Auch als Muster bezeichnet

Vgl. Gabler Wirtschaftslexikon

Beispiele

+ Teile-und-Herrsche-Prinzip
+ Versuch-und-Irrtum-Prinzip
+ Objektorientiere Programmierung
+ Funktionale Programmierung

3.1 Algorithmen

Teile-und-Herrsche-Prinzip

Merkmale

+ Auch bekannt als *divide and conquer*
+ Lässt sich ein Problem sinnvoll in Teilprobleme zerlegen,
  dann löse zunächst die Teilprobleme und setze die Ergebnisse zusammen.
+ Schema für ein gegebenes Problem P
  - Zerlege das Problem in Teilprobleme $P_1 .. P_n$
  - Löse die Teilprobleme, ergibt Lösungen $L_1 .. L_n$
  - Setze die Einzellösungen zu einer Gesamtlösung zusammen

Beispiele

+ Sortieren
+ Effizientes Sortieren
+ Binäre Suche

3.1 Algorithmen

Teile-und-Herrsche-Prinzip

Beispiel Sortieren eines Kartenstapels

+ Gegeben sein ein Stapel mit unterschiedlichen Zahlenkarten.
+ Aufgabe: Sortieren Sie den Stapel so, dass anschließend alle Zahlen in einer aufsteigenden Reihenfolge hintereinander liegen.

Naiver Ansatz

```
Solange noch nicht alles sortiert ist:
    betrachte alle Karten der Reihe nach.
    Ist der aktuelle Wert kleiner als der darunterliegende:
      vertausche die beiden.
```

+ Verfahren **Bubblesort:** wandernde Blase, sortieren durch Austauschen, sehr langsam, Animation

3.1 Algorithmen

Teile-und-Herrsche-Prinzip

Beispiel Sortieren von $n = 10$ Zahlen aufsteigend mit Bubblesort in Pseudocode

```
tue n-1 mal # 10-1 = 9 mal
    tue für jedes Element # 9 mal
      falls aktuelles_Element > nächstes_Element
        vertausche aktuelles_Element mit nächstes_Element # 1 mal
```

+ Gesamtaufwand: $9\ x\ 9\ x\ 1\ =\ 9^2$ → $(n-1)^2$ → n^2

3.1 Algorithmen

Teile-und-Herrsche-Prinzip

Beispiel Sortieren von $n = 10$ Zahlen aufsteigend mit Bubblesort in Python

https://www.programiz.com/dsa/bubble-sort

```
# Bubble sort in Python

def bubbleSort(array):

  # loop to access each array element
  for i in range(len(array)):

        # loop to compare array elements
        for j in range(0, len(array) - i - 1):
            # compare two adjacent elements
            # change oeprator > to < to sort in descending order
            if array[j] > array[j + 1]:
                # swapping elements if elements
                # are not in the intended order
                temp = array[j]
                array[j] = array[j+1]
                array[j+1] = temp

bubbleSort(unsortiert_Liste)

print('Sorted Array in Ascending Order:')
print(unsortiert_Liste)
```

```
Sorted Array in Ascending Order:
[12, 31, 43, 44, 53, 53, 68, 72, 77, 80]
```

3.1 Algorithmen

Teile-und-Herrsche-Prinzip

Beispiel effizientes Sortieren eines Kartenstapels in aufsteigender Reihenfolge in Pseudocode

```
Halbiere das Problem indem man die Karten geeignet teilt
Sortiere die beiden kleineren Stapel
Führe die beiden Teillösungen zusammen
```

Die beiden wichtigsten Algorithmen hierfür

+ **Mergesort:** Sortiere beide Hälften und mische die Stapel dann zusammen
+ **Quicksort:** Teile den Stapel in eine Hälfte mit „kleinen“ Karten und eine Hälfte mit „großen“ Karten und sortiere jeweils

3.1 Algorithmen

Teile-und-Herrsche-Prinzip

Mergesort in Pseudocode

```
Wenn unser Stapel aus einer Karte besteht, sind wir fertig.
Wenn unser Stapel aus 2 Karten besteht, ordnen wir die beiden der Größe nach an und sind fertig.
Wenn unser Stapel größer ist:
  Teilen wir ihn in zwei etwa gleich große Hälften
    Sortieren mit diesem Verfahren jeden Teilstapel
  Und vermischen die sortierten Teilstapel zu einem sortierten kompletten Stapel.
```

\+ Mergesort Animation

3.1 Algorithmen

Teile-und-Herrsche-Prinzip

Quicksort in Pseudocode

```
Wenn unser Stapel aus einer Karte besteht, sind wir fertig.
Wenn unser Stapel aus 2 Karten besteht, ordnen wir die beiden der Größe nach an und sind fertig.
Wenn unser Stapel größer ist:
  Denken wir uns eine Zahl aus, die etwa den Mittelwert der vermuteten Zahlen wiedergibt
  Ordnen die Zahlen im Stapel so, dass auf der einen Seite nur Zahlen liegen, die kleiner oder gleich
  der erdachten Zahl sind und auf der anderen Seite nur solche, die größer sind
  Wenden wir das Verfahren auf jeden der beiden Teilstapel an.
```

\+ Quicksort Animation

3.1 Algorithmen

Teile-und-Herrsche-Prinzip

Grundidee

+ Aus einer Anzahl von möglichen Handlungen wähle eine (noch nicht untersuchte) aus.
+ ***Versuch***: Untersuche, ob mit dieser Auswahl eine Lösung gefunden werden kann.
+ ***Irrtum:*** Falls nicht, untersuche eine andere Möglichkeit.
+ Falls es keine weitere Möglichkeit gibt, dann gibt es für dieses (Teił) Problem keine Lösung.

Verfahren

+ Zerlegung des Problems in Teile
+ Anwendung von Rekursion
+ Zusätzlich: Zurücknehmen von Versuchen, bekannt als *Backtracking*

Beispiel

+ Damenproblem Animation, schachmathematische Aufgabe. Es sollen jeweils acht Damen auf einem Schachbrett so aufgestellt werden, dass keine zwei Damen einander, gemäß ihren in den Schachregeln definierten Zugmöglichkeiten, schlagen können.

3.1 Algorithmen

Heuristik

Heuristik: Vorgehensweise zur Lösung von allg. Problemen, für die keine eindeutigen Lösungsstrategien bekannt sind oder aufgrund des erforderlichen Aufwands nicht sinnvoll erscheinen; beinhaltet in erster Linie „Daumenregeln" auf der Grundlage subjektiver Erfahrungen und überlieferter Verhaltensweisen.

Heuristik wird v.a. in schlecht strukturierten und schwer überschaubaren Problembereichen angewendet.

Vgl. Gabler Wirtschaftslexikon

Merkmale allgemein

+ Mit begrenztem Wissen und wenig Zeit zu guten Lösungen kommen
+ Das Problem ist zu schwer, um es geschlossen/komplett durchzurechnen
+ Grundidee
  - Einen Kompromiss zwischen Rechenzeit und Qualität der Lösung finden
  - Verwendung von Faustregeln, Schätzungen, Raten
  - Es geht nicht um das Optimum, sondern um eine ausreichende Lösung!

3.1 Algorithmen

Gierige Algorithmen

Grundidee

+ Wähle zu jedem Zeitpunkt den Lösungsweg, der den aktuell größten Gewinn/Nutzen verspricht.

Merkmale

+ Auch bekannt als Greedy Algorithmen
+ Sehr schnelles finden von Lösungen
+ Setzt voraus, dass jede „legale“ Wahl zu einer Lösung führt
+ Keine Garantie für die beste Lösung
+ Verwendung für Näherungsfunktionen
+ Ist eine Heuristik

3.1 Algorithmen

Gierige Algorithmen

Beispiele

+ Activity selection problem: Eine Aktivitäten-Auswahl ist das Problem der Einplanung einer Ressource unter mehreren konkurrierenden Aktivitäten.

+ Dijkstra Algorithmus: Berechnung des kürzesten Pfades in einem Graphen von einem gegebenen Startpunkt aus.

+ Münzwechselgeld: Bestimmung der Mindestmenge an Münzen und/oder Scheinen für das Wechselgeld.

+ Rucksackproblem (auch *englisch knapsack problem*): Optimierungsproblem der Kombinatorik. Aus einer Menge von Objekten mit jeweils einem Gewicht und einem Nutzwert soll eine Teilmenge ausgewählt werden, deren Gesamtgewicht eine vorgegebene Gewichtsschranke nicht überschreitet. Unter dieser Bedingung soll der Nutzwert der ausgewählten Objekte maximiert werden.

3.1 Algorithmen

Gierige Algorithmen

Beispiel Münzwechselgeld

+ Es gibt folgende Cent Stücke: `50, 20, 10, 5, 2, 1`
+ Aufgabe:
  - Mit möglichst wenigen Münzen einen gegebenen Betrag darstellen.
  - Wie würde der Algorithmus zur Lösung des Problems aussehen?

Pseudocode

```
Setze den Anfangsbetrag als Restbetrag R
Initialisiere die Menge M der gewählten Münzen mit []
Solange der Restbetrag > 0 ist
    Wähle die größte Münze die kleiner als der Restbetrag ist und füge sie der Menge zu
Ziehe den Münzbetrag vom Restbetrag ab
```

3.1 Algorithmen

Gierige Algorithmen

Beispiel Münzwechselgeld in Python

Eigene Darstellung

```python
#Function to solve the coin change problem
def coin_change(change, coins):
    R = change
    M = []
    coins.sort()
    index = len(coins)-1

    while True:
        coin_val = coins[index]
        if R >= coin_val:
            M.append(coin_val)
            R = R - coin_val
        if R < coin_val:
            index -= 1
        if R == 0:
            break
    return(M)

coins = [50, 20, 10, 5, 2, 1]
coin_change(82, coins)

#Output: [50, 20, 10, 2]
```

```
[50, 20, 10, 2]
```

3.1 Algorithmen

Gierige Algorithmen

Beispiel Münzausgabe - Wann Greedy in Reinform nicht funktioniert

Beispiel 1

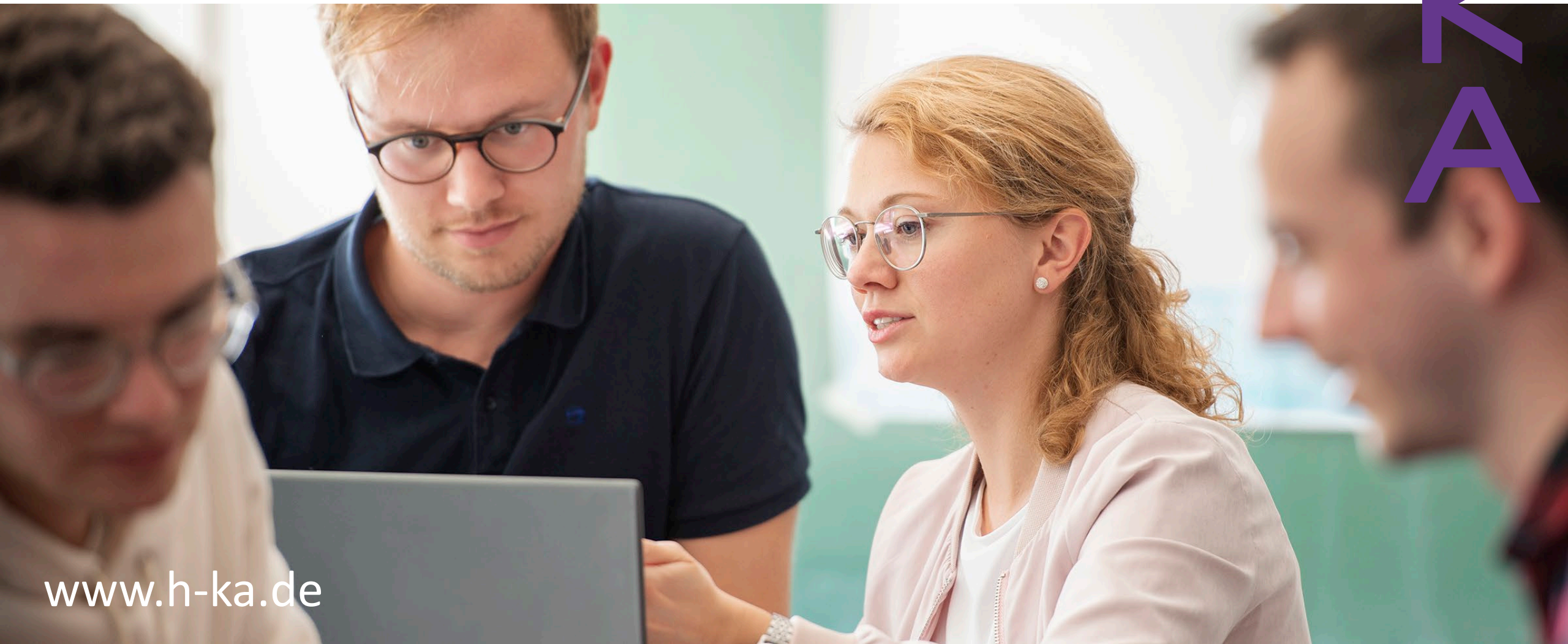
+ Münzen haben die Werte `11, 5, 1`
+ Gesucht ist eine Darstellung von `15`
  - Greedy: `15 = 11 + 1 + 1 + 1 + 1`
  - Optimal: `15 = 5 + 5 + 5`
  - Fazit: Greedy liefert suboptimale Lösung

Beispiel 2

+ Münzen haben die Werte `25, 10, 4`
+ Gesucht ist eine Darstellung von `41`
+ Greedy: `41 = 25 + 10 + 4` `Rest 2 → Keine Lösung!`
+ Optimal: `41 = 25 + 4 + 4 + 4 + 4`
+ Deshalb in der Praxis: Greedy als erste Lösungsnäherung + Kombination mit dem Versuch-und-Irrtum-Prinzip

Hochschule Karlsruhe
University of
Applied Sciences

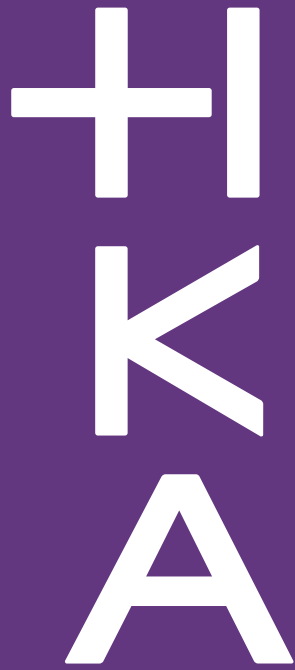
Fakultät für
Informatik und
Wirtschaftsinformatik



www.h-ka.de

Hochschule Karlsruhe
University of
Applied Sciences

Fakultät für
Informatik und
Wirtschaftsinformatik



www.h-ka.de