Algorithmen

Adrian Schüpbach
adrian_laurent.schuepbach@alumni.ethz.ch

Worum geht es?

- Algorithmen

Algorithmen

- Sind Lösungsanleitungen für gegebenes Problem
  - → “Kochbuchrezept”
- Müssen für gegebenes Problem zuerst entwickelt werden
  - Idealerweise konkrete Problemstellung zuerst allgemein formulieren
  - Allgemeingültiger Algorithmus für allgemeine Lösung entwickeln

Algorithmen

- Es gibt bereits viele Algorithmen
  - Für die Datenverwaltung
  - Für Netzwerke
  - Für Grafikanwendungen
  - Für Berechnungen
  - ...

Algorithmen

- Wichtige Eigenschaften der Algorithmen
  - Sollen korrekt sein
  - Sollen möglichst allgemeingültig sein
  - Sollten kleine Codekomplexität haben
  - Sollen effizient sein (“asymptotische Laufzeit”)

Algorithmen

- Wichtige Eigenschaften der Algorithmen
  - Sollen korrekt sein
  - Sollen möglichst allgemeingültig sein
  - Sollten kleine Codekomplexität haben
  - Sollen effizient sein (“asymptotische Laufzeit”)
- Entwicklungsschritte von Algorithmen
  - Korrektheit ist erstes Ziel
  - Codekomplexität verringern
  - Effizienz des Algorithmus’ steigern

Algorithmen

Beispiel: Suchen

- Beispiel:
  - Gegeben: Array mit vielen Zahlen
  - Gesucht: Kommt Zahl ’6’ vor?

Algorithmen

Beispiel: Suchen

- Beispiel:
  - Gegeben: Array mit vielen Zahlen
  - Gesucht: Kommt Zahl ’6’ vor?
- Lösung:
  - Schleife über das ganze Array
  - Wenn Zahl gefunden, dann `true` zurückgeben
  - Wenn Schleife beendet && Zahl nicht gefunden, dann `false` zurückgeben

Algorithmen

Beispiel: Suchen

Suchen im Array

```
public class Programm70 {
  public static boolean vorhanden(int zahl, int[] a) {

    for (int i = 0; i < a.length; i++) {
      if (a[i] == zahl) {
        return (true);
      }
    }
    return (false);
  }
}
```

Algorithmen

Beispiel: Suchen

- Analyse:
  - korrekt?

Algorithmen

Beispiel: Suchen

- Analyse:
  - korrekt? $\rightarrow$ ja

Algorithmen

Beispiel: Suchen

- Analyse:
  - korrekt? $\rightarrow$ ja
  - Codekomplexität klein?

Algorithmen

Beispiel: Suchen

- Analyse:
  - korrekt? $\rightarrow$ ja
  - Codekomplexität klein? $\rightarrow$ ja

Algorithmen

Beispiel: Suchen

- Analyse:
  - korrekt? $\rightarrow$ ja
  - Codekomplexität klein? $\rightarrow$ ja
  - Effizienz?

Algorithmen

Beispiel: Suchen

- Analyse:
  - korrekt? → ja
  - Codekomplexität klein? → ja
  - Effizienz?
    - Wenn Arraygrösse verdoppelt wird, dann doppelt soviele Schritte nötig
    - ⇒ Lineare Komplexität

Algorithmen

Beispiel: Vergleichen

- Beispiel:
  - Gegeben: Zwei Arrays mit vielen Zahlen
  - Gesucht: Welche Zahlen kommen in beiden Arrays vor?

Algorithmen

Beispiel: Vergleichen

- Beispiel:
  - Gegeben: Zwei Arrays mit vielen Zahlen
  - Gesucht: Welche Zahlen kommen in beiden Arrays vor?
- Lösung:
  - Schleife über das erste Array
  - Wenn Zahl im anderen Array gefunden, dann Zahl merken
  - Wenn Schleife beendet, alle gemerkten Zahlen zurückgeben

Algorithmen

Vergleichen

```
public class Programm71 {
  public static int[] glZahl(int[] z1, int[] z2) {
    int[] res = new int[MAX]; int idx = 0; boolean gefun

    for (int i = 0; i < z1.length; i++) {
      gefunden = false;
      for (int j = 0; j < z2.length; j++) {
        if (z1[i] == z2[j]) {
          gefunden = true; break;
        }
      }
      if (gefunden) {
        res[idx++] = z1[i];
      }
    }
    return (res);
```

Algorithmen

Beispiel: Vergleichen

- Analyse:
  - korrekt?

Algorithmen

Beispiel: Vergleichen

- Analyse:
  - korrekt? $\rightarrow$ ja

Algorithmen

Beispiel: Vergleichen

- Analyse:
  - korrekt? $\rightarrow$ ja
  - Codekomplexität klein?

Algorithmen

Beispiel: Vergleichen

- Analyse:
  - korrekt? $\rightarrow$ ja
  - Codekomplexität klein? $\rightarrow$ ja

Algorithmen

Beispiel: Vergleichen

- Analyse:
  - korrekt? $\rightarrow$ ja
  - Codekomplexität klein? $\rightarrow$ ja
  - Effizienz?

Algorithmen

Beispiel: Vergleichen

- Analyse:
  - korrekt? $\rightarrow$ ja
  - Codekomplexität klein? $\rightarrow$ ja
  - Effizienz?
    - Im Durchschnitt $n * m$ Schritte nötig
    - Wenn Arraygrösse verdoppelt wird, dann vierfache Anzahl Schritte nötig
    - $\Rightarrow$ quadratische Komplexität

Algorithmen

Beispiel: Vergleichen

- Beispiel noch schlimmer:
  - Gegeben: Zwei Arrays mit vielen Zahlen
  - Gesucht: Welche Zahlen kommen in *einem* der beiden Arrays vor?

Algorithmen

Beispiel: Vergleichen

- Beispiel noch schlimmer:
  - Gegeben: Zwei Arrays mit vielen Zahlen
  - Gesucht: Welche Zahlen kommen in *einem* der beiden Arrays vor?
- Lösung:
  - Schleife über das erste Array
  - Wenn Zahl im anderen Array *nicht* gefunden, dann Zahl merken
  - Schleife über das zweite Array
  - Wenn Zahl im anderen Array *nicht* gefunden, dann Zahl merken
  - Wenn Schleife beendet, alle gemerkten Zahlen zurückgeben

Effizienz steigern

- Kann man die drei Beispiele besser lösen?

Effizienz steigern

- Kann man die drei Beispiele besser lösen?
- Würde es sich lohnen, sie besser zu lösen?

Effizienz steigern

- Annahme: Arrays sind sortiert

Effizienz steigern

- Annahme: Arrays sind sortiert
- Beispiele nochmals lösen

Algorithmen

Beispiel: Suchen V2

- Beispiel:
  - Gegeben: Array mit vielen Zahlen
  - Gesucht: Kommt Zahl ’6’ vor?

Algorithmen

Beispiel: Suchen V2

- Beispiel:
  - Gegeben: Array mit vielen Zahlen
  - Gesucht: Kommt Zahl ’6’ vor?
- Lösung:
  - Mittlere Zahl anschauen
  - Wenn mittlere Zahl == 6, dann `true` zurückgeben
  - Wenn mittlere Zahl > 6, dann linke Hälfte wählen
  - Wenn mittlere Zahl < 6, dann rechte Hälfte wählen
  - Wenn Hälfte nur noch ein Element hat, dann `false` zurückgeben
  - Mittlere Zahl der gewählten Hälfte anschauen und wieder beginnen

Algorithmen

Beispiel: Suchen V2

Suchen im Array

```
public class Programm73  {
  public boolean vorhanden_bin(int zahl, int[] a,
                               int l, int r) {
    boolean gefunden = false;

    System.out.println("l = " + l + ", r = " + r);
    int m = (r - l) / 2 + l;
    if (a[m] == zahl) { return (true); }
    if (l == r) { return (false); }
    if (zahl < a[m]) {
      return (vorhanden_bin(zahl, a, l, m - 1));
    } else {
      return (vorhanden_bin(zahl, a, m + 1, r));
    }
  }
```

Algorithmen

Beispiel: Suchen V2

- Analyse:
  - korrekt?

Algorithmen

Beispiel: Suchen V2

- Analyse:
  - korrekt? $\rightarrow$ ja

Algorithmen

Beispiel: Suchen V2

- Analyse:
  - korrekt? $\rightarrow$ ja
  - Codekomplexität klein?

Algorithmen

Beispiel: Suchen V2

- Analyse:
  - korrekt? → ja
  - Codekomplexität klein? → ja

Algorithmen

Beispiel: Suchen V2

- Analyse:
  - korrekt? → ja
  - Codekomplexität klein? → ja
  - Effizienz?

Algorithmen

Beispiel: Suchen V2

- Analyse:
  - korrekt? $\rightarrow$ ja
  - Codekomplexität klein? $\rightarrow$ ja
  - Effizienz?
    - Braucht $\log_2(n)$ Schritte

Algorithmen

Beispiel: Vergleichen V2

- Beispiel:
  - Gegeben: Zwei Arrays mit vielen sortierten Zahlen
  - Gesucht: Welche Zahlen kommen in beiden Arrays vor?

Algorithmen

Beispiel: Vergleichen V2

- Beispiel:
  - Gegeben: Zwei Arrays mit vielen sortierten Zahlen
  - Gesucht: Welche Zahlen kommen in beiden Arrays vor?
- Lösung:
  - Index pro Array
  - Wenn Zahlen gleich, dann Zahl merken
  - Kleineren Index erhöhen
  - Wenn ein Index das Arrayende erreicht, fertig

Algorithmen

Vergleichen V2

```
public static int[] glZahl_sort(int[] z1, int[] z2) {
  int idx1 = 0, idx2 = 0; int idx = 0;
  int max = (z1.length > z2.length ? z1.length : z2.leng
  int[] res = new int[max];

  while ((idx1 < z1.length) && (idx2 < z2.length)) {
    if (z1[idx1] == z2[idx2]) {
      res[idx++] = z1[idx1];
    }
    if(z1[idx1] < z2[idx2]) { idx1++; }
    else { idx2++; }
  }
  return (res);
}
```

Algorithmen

Beispiel: Vergleichen V2

- Analyse:
  - korrekt?

Algorithmen

Beispiel: Vergleichen V2

- Analyse:
  - korrekt? $\rightarrow$ ja

Algorithmen

Beispiel: Vergleichen V2

- Analyse:
  - korrekt? $\rightarrow$ ja
  - Codekomplexität klein?

Algorithmen

Beispiel: Vergleichen V2

- Analyse:
  - korrekt? $\rightarrow$ ja
  - Codekomplexität klein? $\rightarrow$ ja

Algorithmen

Beispiel: Vergleichen V2

- Analyse:
  - korrekt? $\rightarrow$ ja
  - Codekomplexität klein? $\rightarrow$ ja
  - Effizienz?

Algorithmen

Beispiel: Vergleichen V2

- Analyse:
  - korrekt? $\rightarrow$ ja
  - Codekomplexität klein? $\rightarrow$ ja
  - Effizienz?
    - Linear gleichzeitig durch beide Arrays
    - $n + m$ Schritte

Algorithmen

Beispiel: Vergleichen V2

- Vergleich zweier sortierter Array kann benutzt werden, um
  - Schnittmenge zu berechnen
  - Mengendifferenz zu berechnen

Algorithmen

Beispiel: Vergleichen V2

- Vergleich zweier sortierter Array kann benutzt werden, um
  - Schnittmenge zu berechnen
  - Mengendifferenz zu berechnen
- Schlüsselwort ist sortiert

Sortieren

- Es gibt viele Sortieralgorithmen

Sortieren

- Es gibt viele Sortieralgorithmen
- Verschiedene Eigenschaften
- Überlegungen, wie bei Suchalgorithmen
  - Laufzeit (n, n^2, $log_2(n)$)
  - Speicherverbrauch

Sortieren

- Beispiel: Mergesort

Sortieren

- Beispiel: Mergesort
- Idee:
  - Rekursiver Algorithmus
  - Rekursionsende: Kann ein oder zwei Elemente sortieren
  - Mehr Elemente: In linke und rechte Hälfte aufteilen
  - Beide Hälften mit mergesort sortieren
  - Beide sortierte Hälften mergen (zusammenfügen)

Sortieren

- Beispiel: Mergesort
- Idee:
  - Rekursiver Algorithmus
  - Rekursionsende: Kann ein oder zwei Elemente sortieren
  - Mehr Elemente: In linke und rechte Hälfte aufteilen
  - Beide Hälften mit mergesort sortieren
  - Beide sortierte Hälften mergen (zusammenfügen)
- Drei Phasen
  - Aufteilen
  - Sortieren
  - Mergen

Sortieren

- Beispiel: Mergesort
- Idee:
  - Rekursiver Algorithmus
  - Rekursionsende: Kann ein oder zwei Elemente sortieren
  - Mehr Elemente: In linke und rechte Hälfte aufteilen
  - Beide Hälften mit mergesort sortieren
  - Beide sortierte Hälften mergen (zusammenfügen)
- Drei Phasen
  - Aufteilen
  - Sortieren
  - Mergen
- Laufzeit: $n * log_2(n)$

Sortieren

- “Merge”
- Gegeben:
  - Zwei sortierte Arrays
  - Index pro Array
- Algorithmus
  - Kleinere Zahl in Ausgabearray speichern
  - Index der kleineren Zahl erhöhen
  - Solange weitermachen, bis Index beider Arrays == Arraylänge ist

Sortieren

Merge

```
public static int[] merge(int[] z1, int[] z2) {
  int idx1 = 0, idx2 = 0, idx = 0;
  int[] res = new int[z1.length + z2.length];

  while ((idx1 < z1.length) && (idx2 < z2.length)) {
    if (z1[idx1] < z2[idx2]) {
      res[idx++] = z1[idx1++];
    } else {
      res[idx++] = z2[idx2++];
    }
  }
  while(idx1 < z1.length) {
    res[idx++] = z1[idx1++];
  }
  while(idx2 < z2.length) {
    res[idx++] = z2[idx2++];
```

Sortieren

- Mergesort
- Gegeben: Unsortiertes Array
- Ausgabe: Sortiertes Array

Sortieren

- Mergesort
- Gegeben: Unsortiertes Array
- Ausgabe: Sortiertes Array
- Algorithmus:
  - Falls Array 1 Element hat:
    - Element zurückgeben
  - Falls Array 2 Elemente hat:
    - Zuerst kleineres Element speichern
    - Dann grösseres Element speichern
    - Dann Resultatarray zurückgeben
  - Sonst:
    - Mittelpunkt berechnen
    - Array aufteilen
    - Mergesort links aufrufen
    - Mergesort rehts aufrufen
    - “Merge” von links und rechts aufrufen

Sortieren

Mergesort

```
public static int[] mergesort(int[] z, int l, int r) {
  int anzahl = r - l + 1;          int res[];
  if (anzahl == 1) {
    res =  new int[anzahl];        res[0] = z[l];
  } else if (anzahl == 2) {
    res = new int[anzahl];
    if (z[l] < z[r]) { res[0] = z[l]; res[1] = z[r];
    } else {                  res[0] = z[r]; res[1] = z[l];
    }
  } else {
    int m = (r - l) / 2 + l;
    int[] r1 = mergesort(z, l, m - 1);
    int[] r2 = mergesort(z, m, r);
    res = merge(r1, r2);
  }
  return(res);
```

Sortieren in Java

- Java hat Sortier-Methoden
- Arrays kann man mit `Arrays.sort()` sortieren
  - Ist eine Variante von Quicksort
  - Bietet (meistens) Laufzeit von $n * log_2(n)$
  - Muss `import java.util.Arrays;` hinschreiben

Sortieren in Java

Sortieren in Java

```
import java.util.Arrays;

public class Programm76 {
  public static void main(String[] args) {
    int[] zahlen = {6,3,4,1,2,1};

    Arrays.sort(zahlen);

    for (int i = 0; i < zahlen.length; i++) {
      System.out.print(zahlen[i] + ", ");
    }
  }
}
```

In sortiertes Array einfügen

Insertion-Sort

- Wenn Array schon sortiert ist...
  - ...neues Element an richtiger Stelle einfügen
  - ...nicht ganzes Array neu sortieren
- Einfüge-Algorithmus
  - Neues Element hinten einfügen
  - Mit vorhergehendem Element vergleichen
  - Falls neues Element kleiner isst, beide Elemente tauschen
  - Solange weitermachen, bis eingefügtes Element an richtiger Stelle ist
- Löschen: Nach hinten austauschen

In sortiertes Array einfügen

Insertion-Sort

```
public class Programm77 {
  public int einfuegen(int[] z, int anz, int neu) {
    int tmp;    int i = anz;
    z[anz] = neu;

    if (anz == 0) {
      return (anz + 1);
    }
    while ((i > 0) && z[i] < z[i - 1]) {
      tmp = z[i - 1];
      z[i - 1] = z[i];
      z[i] = tmp;
      i--;
    }
    return (anz + 1);
  }
```

Meta-Daten

- Array beinhaltet unsere Daten

Meta-Daten

- Array beinhaltet unsere Daten
- Zusätzlich braucht man oft “Verwaltungsdaten”
  - Wieviele Elemente sind im Array enthalten/gültig?
  - Welches ist das letzte gültige Element?

Meta-Daten

- Array beinhaltet unsere Daten
- Zusätzlich braucht man oft “Verwaltungsdaten”
  - Wieviele Elemente sind im Array enthalten/gültig?
  - Welches ist das letzte gültige Element?
  - Ist das Array aufwärts oder abwärts sortiert?
  - ...

Meta-Daten

- Array beinhaltet unsere Daten
- Zusätzlich braucht man oft “Verwaltungsdaten”
  - Wieviele Elemente sind im Array enthalten/gültig?
  - Welches ist das letzte gültige Element?
  - Ist das Array aufwärts oder abwärts sortiert?
  - ...
- Solche zusätzlichen Daten nennt man Meta-Daten

Meta-Daten

- In Beispielen bisher immer zusätzlichen Parameter verwendet
- Alternative: Ersten Teil des Arrays für Meta-Daten verwenden
  - Anzahl Elemente in `array[0]` speichern
  - Eigentliche Daten beginnen erst bei `array[1]`

Zusätzliche Daten

- Normalerweise hat man nicht nur eine Zahl
- Zusätzliche Daten, die zur Zahl gehören

Zusätzliche Daten

- Normalerweise hat man nicht nur eine Zahl
- Zusätzliche Daten, die zur Zahl gehören
- Beispiel: Rangliste
  - “Sekunden” für eine Aktivität
  - Kürzeste Zeit $\Rightarrow$ Erster Platz
  - Daten zur Zeit: Index in Namens-Array

Zusätzliche Daten

- Normalerweise hat man nicht nur eine Zahl
- Zusätzliche Daten, die zur Zahl gehören
- Beispiel: Rangliste
  - “Sekunden” für eine Aktivität
  - Kürzeste Zeit $\Rightarrow$ Erster Platz
  - Daten zur Zeit: Index in Namens-Array
- Ziel: Für jede Person Resultat in Sekunden eintragen
  - Personen nach Sekunden sortieren
  - Rangliste ausgeben

Zusätzliche Daten

- Ein Element = <Sekunden, Index in Namens-Array>
- Array-Aufbau:
  - Erster Index: Anzahl Elemente
  - Zweiter Index: Tatsächlich eingefügte Anzahl Elemente

Zusätzliche Daten

- Ein Element = <Sekunden, Index in Namens-Array>
- Array-Aufbau:
  - Erster Index: Anzahl Elemente
  - Zweiter Index: Tatsächlich eingefügte Anzahl Elemente
- → Demo

Fazit Algorithmen und Arrays

- Arrays sind schnell
- Algorithmen mit Arrays relativ einfach umsetzbar
- Ein bisschen unflexibel
- Unbequem, wenn Element aus mehreren Daten besteht