



## Übungsblatt 4

### Komplexität

Abgabe bis: 17.05.2002, 12:00

**Bonussystem:** (\*) Theorieaufgabe zum Korrigieren, (\*\*) Programmieraufgabe zum Korrigieren

#### \*Aufgabe 1: Komplexitätsklassen $P$ und $NP$ (2 Punkte)

Geben Sie alle gültigen Implikationen zwischen den folgenden Aussagen an.

- $\pi$  ist NP-vollständig.
- $\pi$  ist NP-hart.
- $\pi$  ist polynomial reduzierbar auf SAT.
- SAT ist polynomial reduzierbar auf  $\pi$ .

#### Aufgabe 2: Reduktion, NP-Vollständigkeit

Beweisen Sie die folgenden Aussagen:

- Wenn  $A$  polynomial auf  $B$  reduzierbar und  $B \in P$  (bzw.  $NP$ ), dann ist  $A \in P$  (bzw.  $NP$ ).
- $A$  sei NP-vollständig. Dann gilt  $A \in P \Leftrightarrow P = NP$
- “Polynomial reduzierbar” ist eine Quasiordnung, d.h. reflexiv und transitiv auf der Menge der Probleme.
- “Polynomial reduzierbar” ist eine Ordnung auf den Klassen äquivalenter Probleme  $X, Y$  mit  $X$  auf  $Y$  und  $Y$  auf  $X$  polynomial reduzierbar.

#### \*Aufgabe 3: Reduktion des MON-3SAT-Problems (4 Punkte)

Das Monotone-3SAT-Problem (MON-3SAT) unterscheidet sich vom 3SAT-Problem, indem jede Klausel entweder nur positive oder nur negative Atome enthält. Beweisen Sie die folgende Aussage:

MON - 3SAT ist NP-vollständig.

#### \*Aufgabe 4: SAT / 3SAT Reduktion (3 Punkte)

- Bringen Sie die Formel  $F = (a \wedge (c \vee \neg b \vee d)) \vee (e \vee \neg f)$  in konjunktive Normalform mit genau 3 Variablen pro Klausel. Verwenden Sie die Umformungen, die bei der Reduktion von SAT auf 3SAT in der Vorlesung verwendet wurden.
- Geben Sie einen nichtdeterministischen und einen deterministischen Algorithmus (in Pseudocode) an, der überprüft, ob  $F = a \wedge (\neg b \vee c) \wedge (\neg a \vee \neg c)$  eine erfüllbare Formel ist. Wie groß ist jeweils der asymptotische Aufwand für  $n$  Variablen?

#### Aufgabe 5: Das Acht-Damen-Problem – Theorie

Die Dame ist eine Figur des Schachspiels. Sie gilt – nach dem König – als die wertvollste Figur, weil sie auf dem Brett sowohl in horizontaler und vertikaler als auch in beiden diagonalen Richtungen beliebig weit ziehen kann. Eine Dame kann also – wenn sie nicht gerade auf einem Randfeld steht – sich in acht unterschiedliche Richtungen fortbewegen. Sie kann dabei bis zu sieben Felder weit kommen, außer eines der dazwischenliegenden Felder ist von einer anderen Figur besetzt (sie darf also nicht über andere Figuren hinwegspringen).

Man sagt, dass eine Figur eine andere *schlagen* kann, wenn sie das Feld der anderen Figur in einem Zug erreichen kann.

- Betrachten Sie eine beliebige Diagonale auf dem Schachfeld (am besten aber keine der beiden Hauptdiagonalen). Notieren Sie untereinander die Koordinatenpaare aller Felder auf dieser Diagonalen. Was fällt Ihnen auf? Finden Sie eine Funktion, die für alle Felder auf dieser Diagonalen denselben Wert liefert. – Verfahren Sie genauso mit der anderen Diagonalenrichtung.

- b) Geben Sie einen booleschen Ausdruck an, der genau dann den Wert *wahr* liefert, wenn zwei Damen – gegeben durch Koordinatenpaare  $(x_1, y_1)$  und  $(x_2, y_2)$  – sich gegenseitig schlagen können. [*Hinweis*: Das Schachbrett enthält keine anderen Figuren, die die zwei Damen hindern.]
- c) Das *Acht-Damen-Problem* lautet: Positionieren Sie acht Damen so auf einem Schachbrett (8 auf 8 Felder), dass keine der Damen im nächsten Zug eine andere Dame schlagen kann. Skizzieren Sie 2 mögliche Lösungen.

**\*\*Aufgabe 6: Das Acht-Damen-Problem – Programmierung (6 Punkte)**

- a) Implementieren Sie ein (nichtdeterministisches) Programm nach dem *Las Vegas-Prinzip* zur Lösung des Acht-Damen-Problems von Aufgabe 5. Platzieren Sie zunächst eine Dame beliebig auf dem Schachbrett. Versuchen Sie dann, die zweite Dame so zu platzieren, dass sie die erste nicht schlagen kann. Platzieren Sie genauso die anderen Damen eine nach der anderen so, dass sie die zuvor platzierten nicht schlagen können.
- b) Schreiben Sie ein (deterministisches!) Programm, das alle Lösungen des Acht-Damen-Problems findet. Gehen Sie dazu wie folgt vor:
- Wenn die Damen sich nicht gegenseitig schlagen können sollen, dann dürfen keine zwei Damen auf derselben Zeile des Schachbretts stehen. Platzieren Sie also die erste Dame auf der ersten Zeile des Schachbrettes, die zweite Dame auf der zweiten Zeile usw. Der Index (die Nummer) einer Dame ist also immer gleich dem Index der Zeile, auf der sie steht!
  - Die *i*-te Dame steht immer auf der *i*-ten Zeile des Schachbretts: Sie müssen sich die Zeilenkoordinaten also gar nicht merken. Sie müssen nur die Spaltenkoordinaten speichern – dafür genügt eine eindimensionale Reihung, deren *i*-tes Element angibt, in welcher Spalte die *i*-te Dame steht. Deklarieren Sie eine solche Reihung und geben Sie ihr den Namen **position**.
  - Implementieren Sie eine Methode **schlaegt(zeile, spalte)**, die prüft, ob eine Dame, die an den als Parameter übergebenen Koordinaten steht, eine der zuvor platzierten Damen schlagen kann. Gehen Sie davon aus, dass in der Reihung **position** die Spaltenkoordinaten aller zuvor platzierten Damen gespeichert sind. Verwenden Sie die Funktion aus Aufgabe 5, um zu prüfen, ob zwei Damen sich schlagen können.
  - Implementieren Sie eine *rekursive* Methode **platziere(zeile)**. Der Parameter dieser Methode soll angeben, auf welcher Zeile eine Dame platziert werden soll.
    - Versuchen Sie, eine Dame auf der angegebenen Zeile zu platzieren. Prüfen Sie dazu für jede mögliche Spaltenkoordinate, ob es von diesem Feld aus möglich ist, eine der bereits platzierten Damen zu schlagen. Rufen Sie dazu die Methode **schlaegt** auf!
    - Wenn Sie auf der angegebenen Zeile eine Dame platzieren konnten, dann speichern Sie die gefundene Position in der Reihung **position** und rufen Sie die Methode **platziere** rekursiv auf, um die nächste Dame zu platzieren.

Wie müssen Sie **platziere** aufrufen, um die erste Dame zu platzieren?

- Verwenden Sie die folgende Methode **drucke()**, um eine Schachbrettbelegung auszugeben. Eine statische Variable **anzahl** zählt die Anzahl der ausgegebenen Schachbrettbelegungen. [*Hinweis*: Es gibt 92 gültige Belegungen.]

```
static int anzahl = 0;
static boolean istLV_Loesung = false;

static void drucke() {
    System.out.println();

    if (istLV_Loesung) System.out.println("Las-Vegas-Loesung\n");
    else { anzahl++; System.out.println("Loesung Nr. "+anzahl); }

    for (int i = 0; i < 8; i++) {
        for (int j = 0; j < position[i]; j++) System.out.print("-");
        System.out.print("X");
        for (int j = position[i] + 1; j < 8; j++) System.out.print("-");
        System.out.println();
    }
}
```