Übungen zur Vorlesung

Algorithmen und Datenstrukturen

Übungsblatt 9

Arbeitsgruppe Kryptographie und Komplexitätstheorie
Prof. Dr. Marc Fischlin
Dr. Christian Janson
Patrick Harasser
Felix Rohrbach

Sommersemester 2019
Veröffentlicht am: 14.06.2019
Abgabe am: 28.06.2019, 12:00 Uhr

P1 (Gruppendiskussion)

Nehmen Sie sich etwas Zeit, um die folgenden Fachbegriffe in einer Kleingruppe zu besprechen, sodass sie anschließend in der Lage sind, die Begriffe dem Rest der Übungsgruppe zu erklären:

(a) Backtracking

(b) Meta-Heuristiken

P2 (Hash-Funktionen)

(a) Welche der folgenden Hash-Funktionen sind für Hashtabellen geeignet? Begründen Sie Ihre Entscheidung.

i) $H(a) = \lceil \log_2 a^{32} \rceil$

ii) $H(b) = H(b-1) + H(b-2), \quad H(0) = 1, \quad H(1) = 1$

iii) $H(c) = \left\lfloor \frac{n}{2}(1 + \cos c) \right\rfloor$

iv) $H(d) = d \mod n - (d \mod 7)$

(b) Angenommen, wir verwenden eine Applikation auf einem Server, die Hashtabellen mit Linked Lists bei Kollisionen und die universelle Hashfunktion verwendet. Wir können Elemente einfügen sowie nach Elementen in der Hashtabelle suchen. Weiterhin kennen wir a, b und p aus der Hashtabelle. Was ist die schlechteste asymptotische Laufzeit auf dem Server, die wir durch die Wahl der Einfüge- und Suchoperationen erreichen können? Zu welchen Problemen kann das in der Realität führen?

P3 (Backtracking)

Sudoku ist ein Logik-Rätsel, welches aus 9x9 Feldern, die in 9 3x3-Blöcke aufgeteilt sind, besteht. Jedem Feld soll eine Zahl von 1 bis 9 zugewiesen werden. Dabei darf jedoch jede Zahl in jeder Zeile, jeder Spalte und jedem Block nur genau einmal auftauchen. Das Ziel ist, bei einer vorgegebenen Belegung von einigen Feldern eine vollständige Belegung aller Felder zu finden, die die obrigen Regeln einhält.

(a) Geben Sie einen Algorithmus an, der Sudoku mittels Backtracking löst. Dabei bekommt der Algorithmus ein zweidimensionales Array A[x][y] übergeben, in dem einige Felder bereits mit einer Zahl zwischen 1 und 9 ausgefüllt ist und alle leeren Fehler den Wert 0 haben, und soll ein fertig ausgefülltes Array oder eine Fehlermeldung zurückgeben. Benutzen Sie die Funktion CheckSudoku(A), um zu überprüfen, ob ein teilweise ausgefülltes A alle Sudoku-Regeln erfüllt. Sie dürfen zur Übersichtlichkeit gerne Unterfunktionen anlegen.

(b*) Geben Sie einen Algorithmus für CheckSudoku(A) an.

P4 (Meta-Heuristiken)

Sie sollen das n-Damen-Problem, welches in der Vorlesung vorgestellt wurde, mittels iterativer lokaler Suche lösen. Dabei soll die lokale Suche (wie in der Vorlesung gezeigt) die Dame mit der höchsten Konfliktzahl innerhalb der Zeile auf eines der Felder mit minimaler Konfliktanzahl verschoben werden.

(a) Schreiben Sie zunächst die lokale Suche. Hierbei bekommen Sie eine Verteilung der Damen $S[x][y]$ gegeben, wobei $S[x][y] = 1$ ist, wenn an der Position (x, y) eine Dame ist und ansonsten $S[x][y] = 0$. Weiterhin bekommen Sie die Größe des Schachbrettes n und einen Zeitparameter t übergeben und Sie haben Zugriff auf eine Methode $\textsc{Conflicts}(S, n, x, y)$, die die Anzahl der Damen zurückgibt, die eine Figur auf Feld (x, y) schlagen könnten. Zudem ist vorausgesetzt, dass in jeder Zeile von S nur eine Dame steht. Sie sollen nun so lange die Dame mit der höchsten Konfliktzahl auf einen optimalen Platz in der gleichen Reihe verschieben, bis es keine Konflikte mehr gibt oder die Abbruchbedingung `runningTime`$() > t$ erfüllt ist.

(b) Schreiben Sie nun die iterative lokale Suche. Die Qualität einer Lösung wird durch das Negative der Summe der Konflikte aller Damen berechnet. Die Methode `NewHomeBase` soll die aktuell beste Lösung übernehmen. Die Methode `Pertub` soll die Hälfte aller Damen auf eine zufällige neue Position in der gleichen Zeile stellen. Der Algorithmus bekommt die Zeitdistribution T sowie eine initiale Lösung, bei der auf jeder Zeile eine Dame zufällig gesetzt wurde, als zweidimensionales Array $S[x][y]$ übergeben und soll eine möglichst gute (also möglichst konfliktfreie) Lösung $S'[x][y]$ zurückgeben.

Sie dürfen Unterfunktionen benutzten.

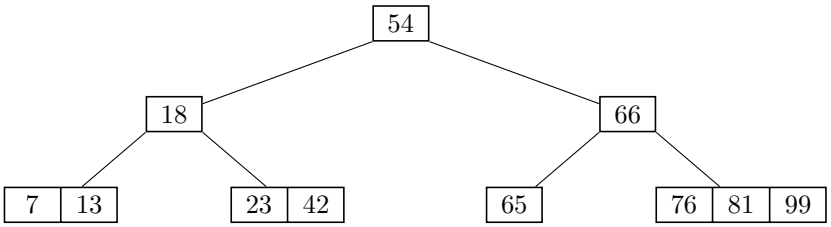
Hausübungen

In diesem Bereich finden Sie die theoretischen Hausübungen von Blatt 9. Bitte beachten Sie die allgemeinen Hinweise zu den Hausübungen und deren Abgabe im Moodle-Kurs! **Denken Sie bitte daran, dass Ihre Lösungen nachvollziehbar und entsprechend ausführlich dargestellt und begründet werden sollen.**
Bitte reichen Sie Ihre Abgabe bis spätestens **Freitag, 28.06.2019, um 12:00 Uhr** ein. Verspätete Abgaben können **nicht** berücksichtigt werden.

H1 (B-Bäume) (2+1+1+1 Punkte)

(a) Fügen Sie der Reihe nach Knoten mit den Schlüsseln $3, 5, 10, 11, 2, 4, 8, 7, 1, 6, 9$ in einen leeren B-Baum mit Grad $t = 2$ ein. Verwenden Sie dabei die Algorithmen aus der Vorlesung. Skizzieren Sie Ihr Zwischenergebnis nach jeder Einfügeoperation.

(b) Löschen Sie die Schlüssel 99 und 18 in gegebener Reihenfolge aus dem untenstehenden B-Baum mit Grad $t = 2$, und zeichnen Sie den daraus resultierenden B-Baum nach jeder Löschoperation. Verwenden Sie dabei die Algorithmen aus der Vorlesung.



(c) Wie lauten die Invarianten für das Einfügen und Löschen eines Schlüssels in einem B-Baum mit Grad t? Und wie wird sichergestellt, dass die Invarianten eingehalten werden?

(d) Nehmen Sie für einen Moment an, dass die Knoten eines B-Baumes mit Grad t mindestens t Schlüssel statt nur $t-1$ Werte enthalten müssen. Welche Auswirkungen hat dies auf das Einfügen und Löschen von Schlüsseln in einem B-Baum?

H2 (Backtracking) (5 Punkte)

Betrachten Sie folgendes kombinatorisches Problem: Es seien $q \geq 1$ eine ganze Zahl, U, V, W paarweise disjunkte Mengen mit $|U| = |V| = |W| = q$, und $M \subseteq U \times V \times W$ eine Teilmenge mit $|M| \geq q$. Es soll bestimmt werden, ob eine Teilmenge $N \subseteq M$ mit $|N| = q$ existiert, deren Elemente paarweise in allen Koordinaten verschieden sind:

$$(\forall (x_1, y_1, z_1) \in N)(\forall (x_2, y_2, z_2) \in N)(x_1 \neq x_2 \wedge y_1 \neq y_2 \wedge z_1 \neq z_2).$$

Verwenden Sie das Backtrack-Prinzip um einen Algorithmus anzugeben, der obiges Problem löst. Der Algorithmus soll die Menge M und die Zahl q als Eingabe bekommen, und eine Menge N wie oben ausgeben, falls diese existiert, und eine Fehlermeldung sonst.

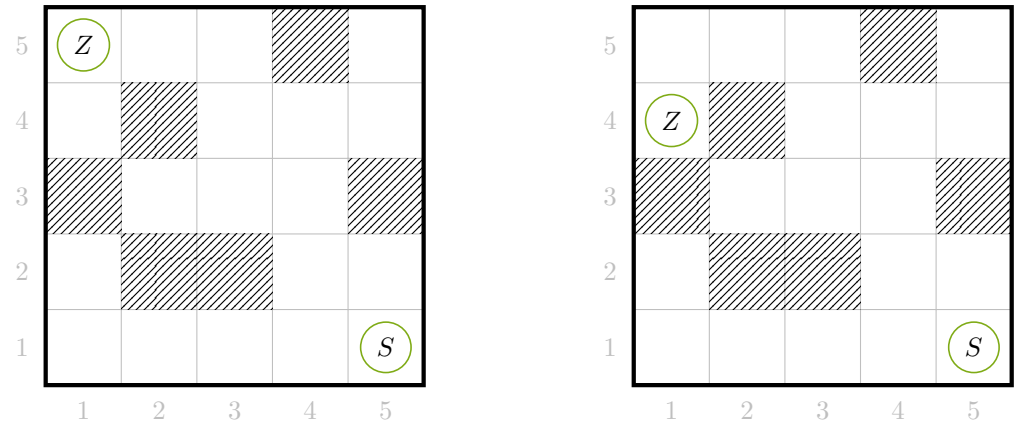
H3 (Bergsteigeralgorithmus) (2.5+2.5 Punkte)

Ziel dieser Aufgabe ist es, die "Bergsteiger-Metaheuristik" anzuwenden, um zwei diskrete Optimiergunsprobleme zu lösen. Wir erklären hier die Heuristik anhand eines Maximierungsproblems – Der Fall eines Minimierungsproblems ergibt sich mit den entsprechenden Änderungen.
Gegeben sei ein diskretes Maximierungsproblem mit Lösungsmenge L und Zielfunktion $f\colon L \to \mathbb{R}$, und sei $d\colon L \times L \to \mathbb{R}$ eine Distanzfunktion auf L. Die Grundidee dieser Metaheuristik lautet wie folgt: Man wählt zunächst einen Startwert $x_0 \in L$ (entweder durch das Problem vorgegeben oder nach einer Verteilung gewählt). Man untersucht dann eine Nachbarschaft von x_0 vom Radius ε (wieder durch das Problem vorgegeben, oder heuristisch gewählt) bezüglich d, und sucht die beste Lösung $(x_1, f(x_1))$ des Problems in dieser Nachbarschaft. Wenn keine Verbesserung möglich war, dann terminiert

der Algorithmus und gibt $(x_0, f(x_0))$ als Lösungskandidat für das Maximierungsproblem aus. Anderenfalls wird diese Vorgehensweise mit Startpunkt x_1 wiederholt. Das Verfahren endet, wenn vom aktuellen Punkt aus keine Verbesserung mehr möglich ist, oder eine maximale Zeit $T_{\max}$ erreicht wurde.

Die gefundene Lösung des Maximierungsproblems ist im besten Fall ein globales Maximum, könnte aber auch nur ein lokales Maximum sein. Damit die Wahrscheinlichkeit ein globales Maximum gefunden zu haben steigt, kann man (sofern das Problem es erlaubt) den Bergsteigeralgorithmus mehrmals mit unterschiedlichen Startwerten ausführen, und dann das beste Ergebnis ausgeben.

(a) Betrachten Sie das Labyrinth im Bild unten links.



In dieser Aufgabe sollen Sie einen Algorithmus erarbeiten, der eine Figur F von der angegebenen Startposition $S = (5, 1)$ zum Ziel $Z = (1, 5)$ führt, ohne dabei die markierten Felder zu betreten. Die Figur F bewegt sich schrittweise, und kann in jedem Schritt nur innerhalb der Nachbarschaft vom Radius $\varepsilon = 1$ der aktuellen Position fortbewegt werden. Hier verwenden wir die Distanzfunktion

$$d\colon L^2 \times L^2 \to \mathbb{R}, \qquad ((x_1, y_1), (x_2, y_2)) \mapsto |x_1 - x_2| + |y_1 - y_2|,$$

wobei $L = \{1, \ldots, 5\}$. Die Entfernung der Figur F zum Ziel soll ebenfalls über die Distanzfunktion d gemessen werden; die zu behandelnde Funktion ist also

$$f\colon L^2 \to \mathbb{R}, \qquad (x, y) \mapsto |x - 1| + |y - 5|.$$

i) Geben Sie einen Algorithmus an, der dieses Problem löst. Ihr Algorithmus soll die "Bergsteiger-Metaheuristik" (ohne zufällige (Neu)Starts, also nur vom angegebenen Startpunkt aus) verwenden, um eine Näherung an das Ziel zu berechnen. Erkunden Sie die erlaubten Felder in der Nachbarschaft eines Punktes immer von oben, gegen den Uhrzeigersinn. Es ist nicht nötig, zeitliche Obergrenzen in Ihrem Algorithmus festzulegen oder zu kontrollieren.

ii) Führen Sie nun den Algorithmus aus, und geben Sie alle Zwischenschritte der Figur F an. Erreicht F das Ziel?

iii) Betrachten Sie nun dasselbe Labyrinth, aber mit geänderten Zielkoordinaten, wie im Bild oben rechts. Beschreiben Sie, wie Sie Ihren Algorithmus diesem Fall anpassen können. Erreicht F das Ziel in diesem Fall? Begründen Sie Ihre Antwort.

(b) Sei $L = \{-4, \ldots, 4\}$, und betrachten Sie die Funktion

$$f\colon L \times L \to \mathbb{R}, \qquad (x, y) \mapsto 2^{-x^2-y^2+1} + 3 \cdot 2^{-x^2-y^2+2x+4y-5}.$$

i) Benutzen Sie die "Bergsteiger-Metaheuristik" um einen Algorithmus anzugeben, der lokale Maxima dieser Funktion findet. Verwenden Sie dabei die Distanzfunktion

$$d\colon L^2 \times L^2 \to \mathbb{R}, \qquad ((x_1, y_1), (x_2, y_2)) \mapsto \max\{|x_1 - x_2|, |y_1 - y_2|\}.$$

Der Radius der betrachteten Umgebungen soll $\varepsilon = 1$ sein. Erkunden Sie die erlaubten Felder in der Nachbarschaft eines Punktes immer von links nach rechts, und von oben nach unten. Benutzen Sie als Startpunkte für Ihr Verfahren die Punkte $S_0 = (-2, -4)$ und $S_1 = (4, 4)$. Es ist nicht nötig, zeitliche Obergrenzen in Ihrem Algorithmus festzulegen oder zu kontrollieren.

ii) Führen Sie jetzt Ihren Algorithmus aus, und geben Sie dessen Zwischenschritte an. Welche Extremwerte findet Ihr Algorithmus?