Informatik II

Ein Aufschrieb der Vorlesung *Informatik II* an der Uni Karlsruhe im Sommersemester 1999, gelesen von Prof. Dr. A. Waibel.

GeTEXt von Andreas Klöckner (ak@ixion.net). Für Kommentare und Berichtigungen bin ich jederzeit dankbar. Neue Versionen gibt es unter http://www.ixion.net/ak/aufschrieb.

LATEX-Lauf am 23. Februar 2006.

Inhaltsverzeichnis

1 Algorithmen für Graphen

Definition 1.1 Graph

Ein *Graph* $G = (V, E)$ besteht aus einer Menge V von Knoten (vertices) und einer Menge E von Kanten (edges). Es gibt *gerichtete* und *ungerichtete* Graphen. Bei ersteren besteht die Menge V aus geordneten, bei letzterem aus ungeordneten Paaren von Knoten.

Ein Graph heißt *gewichtet*, falls seine Kanten mit skalaren Gewichten versehen sind.

Der *Grad* eines Knotens $v \in V$ ist die Anzahl an Kanten, die an v enden. (analog: Eingangsgrad/ Ausgangsgrad)

Ein *Pfad* ist eine Folge von Knoten $v_1, v_2, ..., v_n$, für die eine Folge von Kanten $(v_1, v_2), (v_2, v_3), ..., (v_{n-1}, v_n)$ existiert. Ein Pfad heißt *einfach* oder *elementar*, falls kein Knoten in ihm zweimal vorkommt.

Ein Graph heißt *zusammenhängend*, wenn in seiner ungerichteten Form zwischen allen Knoten Pfade existieren.Ein Graph heißt *zweifach/n-fach zusammenhängend* (biconnected/n-connected), wenn zwischen je zwei Knoten mindestens zwei/n knotendisjunkte Pfade existieren. Ein Graph heißt stark zusammenhängend, wenn zwischen je zwei Knoten $v_1, v_2 \in V$ Pfade von v_1 nach v_2 und umgekehrt existieren. Ein Graph heißt *vollständig*, wenn zwischen je zwei verschiedenen Knoten in G eine Kante existiert.

Ein Knoten v_2 heißt *erreichbar* von v_1 genau dann, wenn ein Pfad zwischen v_1 und v_2 existiert.

Gilt für einen Graphen $G = (V, E)$ und einen zweiten Graphen $G' = (V', E')$ $V' \subseteq V$ und $E' \subseteq E$, so heißt G' Unter- oder Teilgraph von G.

Sei $U \subseteq V$ eine Menge von Knoten. Dann heißt $H = (U, F)$ der *von U induzierte Subgraph*, wenn F alle zu U gehörigen Kanten beinhaltet.

Eine *unabhängige Menge S in G* ist eine Menge von Knoten, die paarweise nicht durch Kanten verbunden werden.

Ein *Zyklus* in G ist ein Pfad P mit dem gleichen Anfangs- und Endpunkt. Von einem *Kreis* spricht man, falls der Zyklus elementar ist. Ein Kreis mit $|P| = 1$ heißt auch *Schlinge*. (circuit: Zyklus, cycle: Kreis)

Ein Graph heißt *Wald*, falls er keine Kreise enthält. Ein zusammenhängender Wald heißt *Baum*. Ist ein Knoten in ihm besonders ausgezeichnet, so heißt der Graph ein *Baum mit Wurzel*.

Ein *aufspannender Wald* eines Graphen ist ein Wald, der alle Knoten des Graphen enthält. Analog: aufspannender Baum ("spanning tree")

Ein ungerichteter Graph heißt bipartit genau dann, wenn seine Knoten in zwei Mengen aufgeteilt werden können, so dass jeder Knoten nur mit Knoten aus der jeweils anderen Menge verbunden ist.

Ein Graph heißt *eulersch*, wenn er zusammenhängend ist und seine Knoten von geradem Grad sind.

Ein Graph ist ein *einfacher Graph*, falls keine zwei Knoten in ihm von mehr als einer Kante verbunden werden. Ein Graph, der diese Eigenschaft nicht hat, heißt Multigraph.

Eine *Clique* in G ist ein vollständiger Untergraph.

Fläche: Keine Definition gefunden.

Problem Darstellung von Graphen im Rechner

- Graphisch: naja
- Mengen: Zwei Felder, eins mit Knoten (die wiederum mit zugehörigen Informationen) und Kanten (von Knoten-Index a nach Knotenindex b).
- Adjazenzliste: Jeder Knoten schleppt eine Liste von verbundenen Knoten mit sich.
- Adjazenzliste in einem Feld: Erste n Felder repräsentieren Knoten, tragen jeweils Beginn Ihrer Adjazenzliste in diesem Feld in sich.
- Adjazenzmatrix: Tabelle mit Knoten-Indizes entlang Zeile ("von") und Spalte ("nach"), Gewichte (bzw. 1 für verbunden, 0 für nicht verbunden) werden bei der entsprechenden "von"–"nach"-Zelle eingetragen.

Algorithmus Tiefensuche

(auch DFS: Depth first search) Gehe zu einem Knoten, markiere ihn, klappere rekursiv alle benachbarten Knoten ab, vermeide dabei bereits markierte Knoten. DFS kann gewisse Arbeiten "vor", "nach" und "während" des Abstiegs vornehmen, z.B.

- Versieht man jeden Knoten in der Reihenfolge der Abarbeitung bei der DFS mit einer Nummer, so nennt man diese Nummer die *DFS-Nummer*.
- Mittels DFS lässt sich aus einem ungerichteten Graphen in einem Anlauf ein DFS-Baum erzeugen, der folgenden Eigenschaft hat: Entweder ist eine Kante Teil des Baumes, oder sie ist eine Querkante ("cross edge"), d.h. sie verbindet im DFS-Baum einen Knoten mit einem seiner Vorgänger.
- Ebenso lassen sich mittels DFS aus gerichteten Graphen DFS-Bäume erzeugen. (dafür sind gegebenenfalls im Gegensatz zu ungerichteten Graphen mehrere Anläufe nötig. Es kann ja schließlich mehrere "Hügel" im Graphen geben, von denen man hinunter, aber nicht mehr hinauf kommt.) Hier gibt es für eine Kante die folgenden Möglichkeiten:
  - im Baum enthalten
  - Querkante (cross edge)
  - Vorwärtskante (forward edge)

  - Rückwärtskante (back edge)

  Haupteigenschaft von gerichteten DFS-Bäumen ist, dass der DFS-Baum bezüglich der DFS-Numerierung ein Heap ist. (Eltern-Knoten hat kleinere Zahl als Kind-Knoten)

Bei unzusammenhängenden Graphen muss evtl. noch geprüft werden, ob alle Knoten besucht worden sind.

Aufwand: zusammenhängend, ungerichtet $O(|V|)$, unzusammenhängend oder gerichtet $O(|V| + |E|)$.

Algorithmus Breitensuche

(auch BFS: Breadth first search) Beginne bei einem Knoten. Markiere ihn, markiere alle seine Kinder und setze Verweise auf sie in einen FIFO. Nimm den nächsten so zu bearbeitenden Knoten aus dem FIFO.

Versieht man jeden Knoten in der Reihenfolge der Abarbeitung bei der BFS mit einer Nummer, so nennt man diese Nummer die *BFS-Nummer*.

Aufwand: $O(|V| + |E|)$.

Algorithmus Topologisches Sortieren

Gegeben: endlicher Baum $G = (V, E)$

Gesucht: "Reihenfolge"

Ablauf:

- entferne sukzessive alle Knoten mit Eingangsgrad 0, merke die Reihenfolge

Algorithmus Alle kürzesten Pfade von einem Ausgangspunkt

Gegeben: Graph $G = (V, E)$, Gewicht $Wt(x, y)$ der Kante (x, y), Startknoten v

Gesucht: Gewichtesumme des jeweils kürzesten Pfades von einem Ausgangspunkt zu jedem Knoten eines Graphen.

Ablauf:

- Ordne jedem Knoten die gegenwärtige Pfadlänge $L(v)$ zu (anfangs ∞)
- $L(v) := 0$
- Markiere alle Knoten (inklusive v) als unbesucht
- Solange unbesuchte Knoten existieren:
  - Wähle unbesuchten Knoten w mit dem geringsten $L(w)$
  - Markiere w als besucht

  - Für jeden Nachbarn x von w: $L(x) := \min\{L(w) + Wt(w,x), L(x)\}$

Aufwand: $O((|V| + |E|) \log |V|)$

Um den kürzesten Pfad jeweils herauszufinden, vermerke die Herkunft des Minimums am Knoten.

Algorithmus Algorithmus von Prim

(Minimaler Spannbaum, auch: minimum cost spanning tree (MCST))

Gegeben: Graph $G = (V, E)$, Gewicht $Wt(x, y)$ der Kante (x, y)

Gesucht: Menge von Kanten, so dass diese einen Spannbaum bilden und ihre Gewichtesumme minimal ist.

Ablauf:

- Lege Knoteneigenschaften fest: $C(x), x \in V$ gibt die minimalen Kosten wieder, die gebraucht werden, um diesen Knoten vom Restbaum aus zu erreichen, $E_m(x), x \in V$ gibt an, über welche Kante diese minimalen Kosten erreicht werden, jeder Knoten erhält eine Besuchsmarkierung.
- $\forall x \in V : C(x) := \infty$, alle Knoten sind anfangs unbesucht
- Suche die Kante mit minimalen Gewicht. Sei diese (m_1, m_2)
- Markiere m_1 als besucht
- $\forall (m_1, x) \in E : C(x) := Wt(m_1, x), E_m(x) := (m_1, x)$
- Solange unbesuchte Knoten existieren:
  - Wähle Knoten w mit dem geringsten $C(w)$
  - Markiere w als besucht, füge $E_m(w)$ zu Spannbaum hinzu
  - $\forall (w, x) \in E$: Ist $C(x) > Wt(w, x)$, so setze $C(x) := Wt(w, x)$ und $E_m(x) := (w, x)$

Aufwand: $O((|V| + |E|) \log |V|)$

Algorithmus Algorithmus von Kruskal

Gegeben: ungerichteter Graph $G = (V, E)$, Gewicht $Wt(x, y)$ der Kante (x, y)

Gesucht: Menge von Kanten, so dass diese einen Spannbaum bilden und ihre Gewichtesumme minimal ist.

Ablauf:

- Füge $|V| - 1$ mal die kürzeste noch nicht hinzugefügte Kante zum Spannbaum hinzu, die keinen Zyklus entstehen lässt.

Algorithmus Transitive Hülle

Gegeben: Graph $G = (V, E)$

Gesucht: Menge von Kanten, so dass zwischen je zwei voneinander erreichbaren Knoten eine Kante existiert.

Zwei verschiedene Möglichkeiten:

- $|V| - 1$ mal für jede Kante alle angrenzenden Kanten dazufügen
- (Voraussetzung: Graph liegt in Form einer Adjazenzmatrix A vor) Bestimme $H = \sum_{i=0}^{n-1} A^i$. Ist $H_{ij} \neq 0$, so existiert eine Abkürzung von i nach j.

Definition 1.2 Matching

Zwei Kanten (u, v) und (x, y) heißen *unabhängig*, wenn u, v, x, y paarweise verschieden sind. Ansonsten heißen die beiden Kanten benachbart.

Eine Kantenmenge heißt *Matching/unabhängig* :⇔ alle ihre Elemente sind paarweise unabhängig.

Ein Knoten heißt *frei* bezüglich eines Matchings, wenn er von keiner Kante berührt wird. Andernfalls heißt er gematcht.

Ein Matching heißt *(fast) perfekt* :⇔ alle (bis auf einen) Knoten des Graphen werden gematcht.

Ein um keine Kante erweiterbares Matching heißt *maximal*.

Ein Matching heißt maximum :⇔ jedes andere Matching hat weniger oder gleich viele Kanten.

Satz 1.1

Sei $G = (V, E)$ ein Graph. Eine Menge $M \subseteq E$ ist genau dann ein Matching, wenn der Grad jedes Knotens bezüglich M 0 oder 1 ist.

Definition 1.3 Alternierender Pfad

Sei $G = (V, E)$ ein Graph, $M \subseteq E$ ein Matching, $v, w \in V$ zwei ungematchte Knoten. Dann ist ein alternierender Pfad $P \subset E$ von v nach w ein Pfad in G, so dass genau jede zweite Kante von P in M liegt.

Satz 1.2 Satz von Berge

Voraussetzung: $G = (V, E)$ ein Graph, $M \subseteq E$ ein Matching

Es gibt in G keinen alternierenden Pfad bezüglich $M \Leftrightarrow M$ ist Maximum-Matching

Algorithmus Bipartites Matching

Gegeben: bipartiter Graph $G = (V \cup U, E)$

Gesucht: Ein Maximum-Matching $M \subset E$

Ablauf:

- Markiere alle Kanten als unbesucht
- Solange es eine unbesuchte Kante (u, v) gibt:
  - Nimm Kante in M auf
  - Markiere (u, v) und alle zu (u, v) benachbarten Kanten
- Betrachte G nun als gerichtet: Ist eine Kante $e \in M$, so richte sie von V nach U, ansonsten umgekehrt.
- Suche für jeden ungematchten Knoten v einen gerichteten Pfad (der jetzt genau einem alternierenden Pfad entspricht) zu einem ebenfalls ungematchten Knoten u.

  Existiert ein solcher alternierender Pfad P, vertausche entlang P die Matchings-Zugehörigkeit. Dann ist M immer noch Matching, aber größer als vorher.

Aufwand: $O(|V|(|V| + |E|))$

Satz 1.3 Satz von König

Voraussetzung: $G = (V, E)$ Graph

G bipartit $\Leftrightarrow$ Es existieren keine ungeraden Zyklen in G.

Bemerkung

Aus dem Satz von König folgt u.a., dass alle Bäume und alle Hyperwürfel bipartit sind.

Definition 1.4 Netzwerk

Ein gerichteter Graph $G = (V, E, s, t)$ mit Gewichten ("Kapazitäten") $Cap((x, y)), (x, y) \in E$ und zwei Knoten s (Quelle) und t (Senke) heißt ein *Netzwerk*. Dabei muss die Quelle Eingangsgrad 0 und die Senke Ausgangsgrad 0 haben.

Ein *Fluss* ist eine Funktion $f : E \to \mathbb{R}$, die den folgenden Forderungen genügt:

- $0 \leq f(e) \leq Cap(e)$
- $\forall v \in V \setminus \{s, t\} : \sum_u f((u, v)) = \sum_u f((v, u))$

Mit dem Wert eines Flusses bezeichnet man die Summe aller von s ausgehenden/bei t eintreffenden Ströme. Der Fluss mit maximalem Wert heißt maximaler Fluss.

Bemerkung Aufstockbarkeit von Matching- zu Netzwerkproblemen

Gegeben sei ein bipartiter Graph $G = (V \cup U, E)$. Dann konstruiert man das zugehörige Netzwerkproblem durch hinzufügen einer Quelle, einer Senke, Verbinden der Quelle mit allen Knoten aus V, bzw. aller Knoten aus U mit der Senke und Zuweisen der Kapazität 1 an alle Kanten.

Definition 1.5 Augmentierender Pfad

Sei $G = (V, E, s, t)$ mit $Cap((x, y))$ ein Netzwerk. Dann ist ein *augmentierender Pfad* ein Pfad von s nach t aus Kanten (v, u), die je genau eine der folgenden zwei Bedingungen erfüllen:

- $(v, u) \in E,\ f((v, u)) < Cap((v, u))$ ("Vorwärtskante")
  Die Differenz $Cap((v, u)) - f((v, u)) > 0$ nennt sich "Flaute" (slack) der Kante.
- $(u, v) \in E,\ f((u, v)) > 0$ ("Rückwärtskante")

Bemerkung Wie man mit Hilfe eines augmentierenden Pfades den Fluss vergrößert

Sei P augmentierender Pfad in $G = (V, E, s, t)$. Sei $F \subseteq P$ die Menge der Vorwärtskanten, $B \subseteq P$ die Menge der Rückwärtskanten. Dann nimm

$$m_F := \min\{Cap((v, u)) - f((v, u)) \mid (v, u) \in F\}$$

und

$$m_B := \min\{f((u, v)) \mid (u, v) \in B\}$$

Dann kann der Fluss um $m := \min\{m_F, m_B\} > 0$ vergrößert werden, indem man:

- Bei Vorwärtskanten m zum Fluss dazuaddiert
- Bei Rückwärtskanten m vom Fluss abzieht

Die Anforderungen an Erhaltung und Kapazität werden erfüllt, und der gesamte Fluss wird vergrößert, weil nur Vorwärtskanten s verlassen und nur Vorwärtskanten bei t eintreffen (und bei beiden der Fluss vergrössert wurde).

Definition 1.6 Schnitt

Sei $G = (V, E, s, t)$ mit $Cap((x, y))$ ein Netzwerk. Sei $A \subset E$ eine Teilmenge mit $(s, x) \in A, (x, t) \notin A$ $(x \in V)$. Dann ist der Schnitt von $A := \{(u, v) \mid$

$(u, v) \in E, u \in A, v \in E \setminus A\}$. Die Kapazität eines Schnittes ist die Summe der Kapazitäten der Kanten im Schnitt.

Satz 1.4 Satz vom augmentierenden Pfad

Voraussetzung: $G = (V, E, s, t)$ mit $Cap((x, y))$ Netzwerk, f ein Fluss darin

f maximal $\Leftrightarrow$ Es existiert kein augmentierender Pfad bezgl. f in G.

Satz 1.5 Max-Flow-Min-Cut

Voraussetzung: $G = (V, E, s, t)$ mit $Cap((x, y))$ Netzwerk

Der Wert des maximalen Flusses in G entspricht der geringsten Kapazität eines Schnittes in G.

Satz 1.6 Satz vom ganzzahligen Fluss

Voraussetzung: $G = (V, E, s, t)$ mit $Cap((x, y)) \in \mathbb{N}$ Netzwerk, f_{max} maximaler Fluss in G

Für alle Kanten $e \in E$ gilt: $f_{max}(e) \in \mathbb{N}$.

Algorithmus Suche eines augmentierenden Pfades

Gegeben: $G = (V, E, s, t)$ mit $Cap((x, y))$ Netzwerk, f ein Fluss darin

Gesucht: Ein augmentierender Pfad P.

Ablauf:

- Erstelle aus dem "alten" Netzwerk ein neues ("Residualnetzwerk") nach den folgenden Regeln: Betrachte jede Kante $e = (v, w) \in E$.
  - Ist $f(v) = Cap(v)$, so übernimm v in umgekehrter Richtung mit gleicher Kapazität.
  - Ist $0 \neq f(v) < Cap(v)$, so übernimm v in umgekehrter Richtung mit gleicher und in gleicher Richtung mit umgekehrter $(Cap(v) - f(v))$ Kapazität.
  - Ist $f(v) = 0$, so übernimm v in gleicher Richtung mit umgekehrter $(Cap(v) - f(v))$ Kapazität.
- Suche nun mittels des bereits erwähnten Algorithmus einen (am besten den kürzesten) Pfad zwischen s und t.

Aufwand: $O(|V| + |E|)$

Definition 1.7 Hamilton'scher Pfad

Sei $G = (V, E)$ ein Graph. Dann ist ein *Hamilton'scher Pfad* ein Kreis, der alle Knoten von G enthält.

Zu finden ist ein/der kürzeste Hamilton'sche(r) Pfad mit derzeit bekannten Algorithmen nicht in polynomialer Zeit.

Algorithmus Minimale Editierdistanz

Gegeben: Zwei Zeichenketten $a_1, \ldots, a_n / b_1, \ldots, b_m$, c_i, c_s, c_d Kosten der Editierschritte *insert*, *substitute*, *delete*.

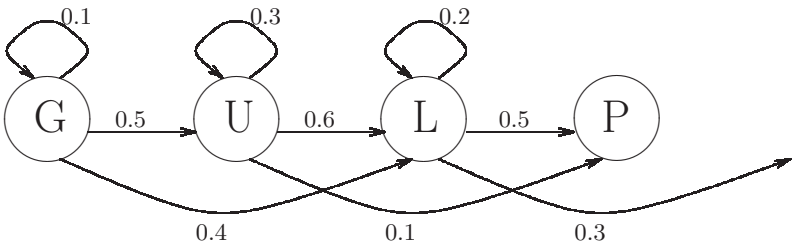
Gesucht: Editiersequenz mit minimaler Kostensumme, die (a_i) in (b_i) umformt

Ablauf:

- Erstelle Matrix $C(i,j)$, die die minimale Zahl an Editierschritten angibt, um $a_1, \ldots, a_i$ in $b_1, \ldots, b_j$ umzuformen. Betrachte hilfsweise $a_{n+1} = b_{m+1} = \emptyset$
- Initialisiere $C(0 \ldots n+1, 0) = 0 \ldots (n+1) \cdot c_d$, $C(0, 0 \ldots m+1) = 0 \ldots (m+1) \cdot c_i$
- Berechne nach folgender Vorschrift von links oben nach rechts unten jede Zelle $C(i,j)$ der Matrix als Minimum von:
  - Wenn $a_i = b_j$: $C(i-1, j-1)$
  - Wenn $a_i \neq b_j$: $C(i-1, j-1) + c_s$
  - $C(i-1, j) + c_d$
  - $C(j, i-1) + c_i$
- Die minimale Editierdistanz findet sich in $C(n+1, m+1)$

Indem man sich in jedem Schritt merkt, welche Art Schritt das Minimum erbracht hat, kann man den minimalen Editierpfad rekonstruieren.

Definition 1.8 Markov-Modell



Eine Hypothese/ein Modell für ein zu erkennendes Muster wird dargestellt in einem Graphen wie dem obigen. Jeder der n Knoten im Graphen heißt ein Zustand. In jedem Zustand gibt es Übergangswahrscheinlichkeiten $p_0(i)$ (Wiederholung/Übergang zum gleichen Knoten), $p_1(i)$ (Übergang zum nächstfolgenden Knoten), $p_2(i)$ (Übergang zum übernächsten Knoten). Man definiert: $p(i,j) :=$ Wahrscheinlichkeit für i im Zustand j

$$p(i,j) = \begin{cases} p_0(j) & i = j \\ p_1(j) & i = j+1 \\ p_2(j) & i = j+2 \\ 0 & \text{sonst} \end{cases}$$

Zusätzlich gibt es eine Tabelle von "Emissionswahrscheinlichkeiten"

$$p_E(< Zeichen >, < Zustand >)$$

zu jedem Zustand im Graphen, die angibt, welches Zeichen mit welcher Wahrscheinlichkeit statt dem im Graph (ohnehin nur symbolisch) vermerkten erkannt wird.

Mit Hilfe eines solchen Modells kann unter Zuhilfenahme eines geeigneten Algorithmus die (maximale) Wahrscheinlichkeit des Zutreffens der Hypothese auf eine Beobachtung ermittelt werden.

Algorithmus Forward-Algorithmus

Gegeben: Markov-Modell (p, p_E), Beobachtung $a_1, \ldots, a_m$. O.B.d.A sei Zustand 1 der Startzustand, Zustand n der Endzustand.

Gesucht: Bestimmung der Wahrscheinlichkeit des Zutreffens der von (p, p_E) dargestellten Hypothese auf die Beobachtung.

Ablauf:

- Erstelle Matrix $\alpha(1 \ldots n, 0 \ldots m)$, wobei $\alpha(i, j)$ die Wahrscheinlichkeit angibt, nach Beobachtung von $a_1, \ldots, a_j$ im Zustand i zu sein.
- Initialisiere Matrix $\alpha(2 \ldots n, 0) := 0$ und $\alpha(1, 0) := 1$
- Berechne Matrix spaltenweise (durch die Beobachtung fortschreitend). Dabei ist für festes $j = 1 \ldots m$ und $i = 1 \ldots n$

$$\alpha(i, j) = \sum_{k=1}^{n} \alpha(k, j-1) \cdot p(i, k) \cdot p_E(a_j, i)$$

- Dann findet sich die Wahrscheinlichkeit, dass diese Hypothese auf die Beobachtung passt, in $\alpha(n, m)$

Algorithmus Viterbi-Algorithmus

Gegeben: Markov-Modell (p, p_E), Beobachtung $a_1, \ldots, a_m$. O.B.d.A sei Zustand 1 der Startzustand, Zustand n der Endzustand.

Gesucht: Bestimmung der wahrscheinlichsten Zuordnung von Symbolen der Beobachtung zu den Zuständen

Ablauf:

- Erstelle Matrix $\alpha(1 \ldots n, 0 \ldots m)$, wobei $\alpha(i, j)$ die maximale Wahrscheinlichkeit eines Pfades zum Zustand i nach Beobachtung von $a_1, \ldots, a_j$ angibt.
- Initialisiere Matrix $\alpha(2 \ldots n, 0) := 0$ und $\alpha(1, 0) := 1$

- Berechne Matrix spaltenweise (durch die Beobachtung fortschreitend). Dabei ist für festes $j = 1 \dots m$ und $i = 1 \dots n$

$$\alpha(i, j) = \max\{\alpha(k, j-1) \cdot p(i, k) \cdot p_E(a_j, i) \mid k \in \{1, \dots, n\}\}$$

- Den wahrscheinlichsten Pfad durch die Matrix findet man dann durch Zurückverfolgen der Maxima ausgehend von $\alpha(n, m)$

2 Geometrische Algorithmen

Definition 2.1 Punkt

Ein Punkt p im n-dimensionalen Raum wird dargestellt durch ein n-Tupel: $p = (x_1, \dots, x_n)$

Definition 2.2 Gerade

Eine *Gerade* wird durch zwei Punkte auf ihr festgelegt.

Definition 2.3 Strecke

Eine *Strecke* wird durch zwei Punkte, ihre *Endpunkte* festgelegt.

Definition 2.4 Streckenzug

Ein *Streckenzug* ist eine Folge von Strecken (*Kanten*), wobei der Endpunkt (*Knoten*) einer Strecke der Anfangspunkt der nächsten ist.

Definition 2.5 Polygon

Ein *Polygon* ist ein Streckenzug, dessen erster und letzter Punkt identisch sind.

Definition 2.6 Inneres

Ein Polygon, dessen Kanten sich nicht schneiden, umrahmt eine Region, deren Punkte *innen* sind.

Definition 2.7 Konvexität

Ein Polygon ist *konvex*, wenn jede Strecke zwischen zwei Punkten in seinem Inneren komplett im Inneren liegt.

Algorithmus Punkt in Polygon

Gegeben: Punkt p, $P = \{u_1 - u_2, \ldots u_n - u_1\}$ Polygon

Gesucht: Liegt p in P?

Ablauf:

- Zähle Kanten, die sich mit einer Halbgerade von p nach ∞ in beliebiger Richtung schneiden (Findet der Schnitt in einem Endpunkt einer Kante statt, hängt das Verfahren vom weiteren Verlauf des Streckenzuges ab)
- Ist diese Zahl gerade, liegt p außen, ist sie ungerade, innen.

Aufwand: $O(n)$

Algorithmus Konstruktion eines kreuzungsfreien Polygons

Gegeben: Punkte $p_1, \ldots, p_n$ im 2-dimensionalen Raum

Gesucht: Kreuzungsfreies Polygon, das alle Punkte als Knoten enthält

Ablauf:

- Wähle Punkt p, der garantiert später im Inneren des Polygons liegt
- Sortiere Punkte nach dem Winkel, den sie mit dem gewählten Punkt und der x-Achsen-Parallele durch den gewählten Punkt einschließen
- Sortiere Punkte, die den gleichen Winkel einschließen, nach Ihrem Abstand zu p
- Verbinde Punkte in dieser Reihenfolge

Aufwand: $O(n \log n)$

Algorithmus Stretching algorithm

Gegeben: Punkte $p_1, \ldots, p_n$ im 2-dimensionalen Raum

Gesucht: Konvexes Polygon, das alle Punkte enthält

Ablauf:

- Beginne mit einem Dreieck aus drei beliebigen Punkten
- Solange noch ein nicht bearbeiteter p Punkt außerhalb der bisher erzeugten Hülle existiert
  - Suche Punkt o in bisheriger Hülle, der mit x-Achsen-Parallele durch p größtmöglichen Winkel einschließt
  - Suche Punkt u in bisheriger Hülle, der mit x-Achsen-Parallele durch p größtmöglichen Winkel einschließt

  - Wirf alle Punkte zwischen o und u aus der Hülle
  - Füge p statt ihrer ein

Aufwand: $O(n^2)$

Algorithmus Gift wrapping algorithm

Gegeben: Punkte $p_1, \ldots, p_n$ im 2-dimensionalen Raum

Gesucht: Konvexes Polygon, das alle Punkte enthält

Ablauf:

- Erzeuge nacheinander Strecken nach dem Schema des Einpackens eines kantigen Geschenks (maximiere Innenwinkel)

Aufwand: $O(n^2)$

Algorithmus Graham's scan

Gegeben: Punkte $p_1, \ldots, p_n$ im 2-dimensionalen Raum

Gesucht: Konvexes Polygon $q_1, \ldots q_m$, das alle Punkte enthält

Ablauf:

- Bestimme den äußerst rechten Punkt p, falls mehrere äußerst rechte existieren, nimm den untersten.
- Bestimme für alle Punkte q den Winkel von pq mit der x-Achse. (es treten Winkel zwischen $\pi/2$ und $3\pi/2$ auf)
- Sortiere diese Liste, sei diese sortierte Liste nun $r_1, \ldots, r_n$
- Länge der bisher gebauten Hülle $l := 2$, $q_i := r_i$ $(i = 1 \ldots l)$
- Gehe die Liste der Reihe nach durch, für jeden Punkt r_i $(i = 3 \ldots n)$ führe folgende Schritte durch:
  - Bestimme Innenwinkel $\alpha := \angle r_i q_l q_{l-1}$
  - Ist $\alpha > \pi$, dekrementiere l solange, bis $\angle r_i q_l q_{l-1} \leq \pi$
  - Inkrementiere l, $q_l := r_i$

Aufwand: $O(n \log n)$

Definition 2.8 Ballung

Das Zusammenfassen einer Mengen von Punkten $P = \{P_1, \ldots, P_n\}$ zu paarweise disjunkten Teilmengen (*Klassen*) $Q_1, \ldots, Q_m \subseteq P$ nennt man *Ballen*, das Ergebnis eine *Ballung*.

Man unterscheidet *überwachtes* und *unüberwachtes* Ballen. Beim überwachten Ballen sind von vornherein Kriterien für jede Klasse Q_i bekannt, die die Zugehörigkeit eines Elementes bestimmen. Beim unüberwachten Ballen ist dies nicht der Fall.

Außerdem unterscheidet man zwischen *agglomerativem* und *divisivem* Vorgehen beim Ballen. Agglomeratives Ballen arbeitet in einer Weise, bei der jeweils mehrere existierende Klassen zu einer einzigen zusammengefasst werden, während divisives Ballen mit nur einer Klasse beginnt und diese dann jeweils in mehrere Klassen zerteilt.

Die (agglomerative) Klassifizierung kann (u.a.) durch die folgenden zwei Kriterien geschehen:

- nach dem nächsten Nachbarn (Punkt wird der Klasse zugeordnet, in der sich auch sein nächster Nachbar befindet).
- nach den k nächsten Nachbarn (Punkt wird der Klasse zugeordnet, in der sich die meisten der k ihm nächstliegenden Punkte befinden)

Algorithmus Nächstes Paar

Gegeben: Eine Menge $P = \{P_1, \dots P_n\}$ von Punkten $P_i = \{x_i, y_i\}$ im 2-dimensionalen Raum

Gesucht: Bestimmen des kleinsten Abstands zweier Punkte

Ablauf:

- Besteht Raum aus zwei Punkten? Dann gib als Ergebnis den Abstand dieser Punkte voneinander aus und beende (Basisfall)
- Trenne Raum entlang einer y-Achsen-Parallele mit der Koordinate x_h in zwei Hälften Q_1 und Q_2 mit $|Q_1| = |Q_2| \pm 1$ (z.B. anhand einer Sortierung nach x-Koordinaten)
- Rekursion über Q_1 und Q_2, Ergebnis: Sei d_1 geringster Abstand in Q_1, d_2 geringster Abstand in Q_2
- Setze $d := \min\{d_1, d_2\}$
- Eliminiere alle Punkte P_i in Q_1 und Q_2, für die gilt $|x_h - x_i| \geq d$
- Sortiere verbleibende Punkte nach y-Koordinate
- Prüfe für jeden Punkt drei Punkte "aufwärts" und drei Punkte abwärts in der Liste auf geringere Abstände, falls dem so ist, setze d neu
- Ergebnis: d

Aufwand: $O(n \log^2 n)$. Es existiert auch ein Algorithmus mit $O(n \log n)$

Algorithmus Linienschnitt

Gegeben: Eine Menge $H = \{h_1, \ldots, h_m\}$ von horizontalen und eine Menge $V = \{v_1, \ldots, v_n\}$ von vertikalen Linien

Gesucht: Menge aller Schnitte $C \subseteq H \times V$ von horizontalen und vertikalen Linien

Ablauf:

- Erstelle nach x-Koordinaten sortierte Liste Q aller Endpunkte der Linien in V und H
- Erstelle eine nach y-Koordinaten sortierte Liste $H' \subseteq H$ der gerade betrachteten horizontalen Linien
- Für jeden Punkt P aus Q
  - Ist P linker Punkt einer horizontalen Linie h? Füge h zu H' hinzu.
  - Ist P rechter Punkt einer horizontalen Linie h? Entferne h aus H'.
  - Ist $x(P)$ Koordinate einer vertikalen Linie v? Prüfe auf Schnitt mit jeder Linie h aus H', falls gegeben, füge (h, v) zu C hinzu.

Aufwand: $O((m+n)\log(m+n) + R)$, wobei R die Anzahl der Schnitte angibt

3 Probabilistische Algorithmen

Definition 3.1 Monte-Carlo-Algorithmus

Einen *Monte-Carlo-Algorithmus* nennt man einen Algorithmus, der mit einer gewissen (hohen) Wahrscheinlichkeit ein richtiges Ergebnis liefert. (Beispiel: Element in der oberen Hälfte)

Definition 3.2 Las-Vegas-Algorithmus

Einen *Las-Vegas-Algorithmus* nennt man einen Algorithmus, der zwar mit Sicherheit ein richtiges Ergebnis liefert, dessen Laufzeit aber unbestimmt ist. (Beispiel: Erstelle solange zufällige Dinge, bis ein gesuchtes dabei herauskommt)

Algorithmus Pseudozufallszahlengenerator

Gegeben: $r_1 \in \mathbb{N}_0$ seed, $b, t \in \mathbb{N}$ Konstanten

Gesucht: $(r_n) \subseteq \mathbb{N}_0$ Zufallszahlen

Ablauf:

- $r_n := (r_{n-1}b + 1) \bmod t$

4 Algebraische Algorithmen

Algorithmus Verfahren des fortgesetztes Quadrierens

Gegeben: $n, k \in \mathbb{N}$

Gesucht: n^k

Ablauf:

- Ist $k = 2$? Ergebnis: $n \cdot n$
- Bestimme (rekursiv) $h := n^{\lfloor k/2 \rfloor}$
- Ist k gerade? Ergebnis: $h \cdot h$
- Ist k ungerade? Ergebnis: $h \cdot h \cdot n$

Aufwand: $O(\log k)$

Algorithmus Euklidischer Algorithmus

Gegeben: $m, n \in \mathbb{N}$

Gesucht: Größter gemeinsamer Teiler von m und n

Ablauf:

- Setze $d := m$, $v := n$
- Solange $v \neq 0$
  - Bestimme $r := d \bmod v$ (Es gilt $d = kv + r$, $k \in \mathbb{N}$)
  - Setze $d := v$, $v := r$

Aufwand: $O(\log(n + m))$

Algorithmus Polynommultiplikation

Gegeben: Zwei Polynome $p, q \in \mathbb{R}[x]$ mit $\operatorname{grad} p = \operatorname{grad} q = n$, $n \in \mathbb{N}$

Gesucht: $p \cdot q$

Ablauf:

- Zerteile p in p_1, p_2, so dass $p = p_1 \cdot x^{\lfloor n/2 \rfloor} + p_2$
- Zerteile q in q_1, q_2, so dass $p = q_1 \cdot x^{\lfloor n/2 \rfloor} + q_2$
- Basisfall: $\operatorname{grad} p, q = 1$. Ergebnis: $p_1 q_1 x^2 + (p_1 q_2 + p_2 q_1)x + p_2 q_2$
- Bestimme rekursiv: $a := p_1 q_1$, $c := p_2 q_2$, $b := (p_1 + p_2)(q_1 + q_2) - a - b$

- Ergebnis: $ax^n + bx^{\lfloor n/2 \rfloor} + c$

Aufwand: $O(n^{1,59})$

Algorithmus Matrizenmultiplikation nach Winograd

Gegeben: $A, B \in \mathbb{R}^{n \times n}$, $A = ((a_{ij}))$, $B = ((b_{ij}))$, $n = 2m$, $m \in \mathbb{N}$

Gesucht: $C := A \cdot B = ((c_{ij}))$

Ablauf:

- Berechne für jede Zeile $A_i := \sum_{k=1}^{n/2} a_{i,2k-1} a_{i,2k}$
- Berechne für jede Spalte $B_i := \sum_{k=1}^{n/2} b_{2k-1,i} b_{2k,i}$
- Berechne für $i, j = 1 \ldots n$

$$c_{ij} = \sum_{k=1}^{n/2} (a_{i,2k\_1} + b_{2k-1,i}) \cdot (a_{i,2k} + b_{2k,i}) - A_i - B_j$$

Aufwand: $\frac{n^3}{2} + n^2$

Algorithmus Matrizenmultiplikation nach Strassen

... (kann sich kein Schwein merken, kommt daher auch nicht in der Klausur dran)

Aufwand: $O(n^{2,81})$

Algorithmus RSA-Verschlüsselung

Schlüsselerzeugung:

Gesucht: $n, e, d \in \mathbb{N}$, wobei $e \cdot d = k \cdot \varphi(n) + 1$

$\varphi(n)$ ist die Anzahl der zu n teilerfremden Zahlen $< n$

Erläuterung: Unter der Voraussetzung, dass a und n teilerfremd sind, gilt $a^{\varphi(n)} \bmod n = 1$

Ablauf:

- Wähle zwei große Primzahlen p, q, $n := pq$
- Bestimme $\varphi(n) = n - (p - 1 + q - 1 + 1) = (p - 1)(q - 1)$
- Zerlege $k\varphi(n) + 1$ in zwei Teiler e und d

Verschlüsselung:

Gegeben: $n, e, m \in \mathbb{N}$ (m ist zu verschlüsselnde Nachricht)

Gesucht: $c \in \mathbb{N}$ (Code)

Ablauf:

- $c = m^e \bmod n$

Entschlüsselung:

Gegeben: $n, d, c \in \mathbb{N}$ (c ist der Code)

Gesucht: $m \in \mathbb{N}$ (entschlüsselte Nachricht)

Ablauf:

- $m = c^d \bmod n$

Algorithmus Algorithmus der vier Russen

Gegeben: $A, B \in \{0,1\}^{n \times n}$

Gesucht: $C := A \cdot B$

Schreibweise:

- $A_C(i \dots j)$ $n \times (j - i + 1)$-Matrix, bestehend aus den Spalten $i \dots j$ von A.
- $B_R(i \dots j)$ $(j - i + 1) \times n$-Matrix, bestehend aus den Zeilen $i \dots j$ von A.

Ablauf:

- Bestimme ein $k < n$ mit $n = mk, m \in \mathbb{N}$
- Lege Matrix $C \in \{0,1\}^{n \times n}$, gefüllt mit Nullen, an
- Für $i = 1$ bis m
  - Berechne eine Tabelle für alle möglichen Ergebnisse von $\vec{a} B_R((i-1) \cdot k + 1 \dots ik)$ mit $a \in \{0,1\}^k$
  - Bestimme mit Hilfe dieser Tabelle $D := A_C((i-1) \cdot k + 1 \dots ik) \cdot B_R((i-1) \cdot k + 1 \dots ik) \in \{0,1\}^{n \times n}$
  - $C := C + D$

Definition 4.1 Einheitswurzel

Der Ausdruck

$$\sqrt[n]{1} := e^{i\frac{2\pi}{n}}$$

heißt komplexe n-te Einheitswurzel.

Definition 4.2 DFT-Matrix

Sei $n \in \mathbb{N}$ und $\eta := \sqrt[n]{1}$. Dann nennt man

$$\mathrm{DFT} := ((d_{ij})) \in \mathbb{C}^{n\times n} = ((\eta^{(i-1)\cdot(j-1)}))$$

die Matrix der diskreten Fouriertransformation (DFT).

Bemerkung Inverse der DFT

Die DFT-Matrix ist regulär und es ist

$$\mathrm{DFT}^{-1} = ((t_{ij})) \in \mathbb{C}^{n\times n} = \frac{1}{n}((\eta^{-(i-1)\cdot(j-1)}))$$

Algorithmus FFT

Gegeben: $m, n \in \mathbb{N}, n = 2^m$,$x \in \mathbb{C}^n = (x_0, \ldots, x_{n-1})$

Gesucht: $\mathrm{DFT}\cdot x$

Ablauf:

- Basisfall: Ist $n = 1$, so gib x_0 zurück
- Setze $x_g := (x_0, x_2, x_4, \ldots, x_{n-2})$
- Setze $x_u := (x_1, x_3, x_5, \ldots, x_{n-1})$
- Berechne $y_g := \mathrm{DFT}\cdot x_g$ und $y_u := \mathrm{DFT}\cdot x_u$ (Rekursion mit Problemgröße $n/2$)
- Setze $\eta := \sqrt[n]{1}$
- Ergebnisvektor:

$$\begin{pmatrix} r_0 \\ r_1 \\ r_2 \\ \vdots \\ r_{n/2} \\ r_{n/2+1} \\ \vdots \\ r_{n-1} \end{pmatrix} = \begin{pmatrix} y_g^{(0)} + \eta^0 y_u^{(0)} \\ y_g^{(1)} + \eta^1 y_u^{(1)} \\ y_g^{(2)} + \eta^2 y_u^{(2)} \\ \vdots \\ y_g^{(0)} - \eta^0 y_u^{(0)} \\ y_g^{(1)} - \eta^1 y_u^{(1)} \\ \vdots \\ y_g^{(n/2-1)} - \eta^{n/2-1} y_u^{(n/2-1)} \end{pmatrix}$$

Aufwand: $O(n \log n)$

Rechtfertigung:

Sei

$$\begin{aligned} P(x) &:= a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \cdots + a_1 x + a_0 \\ P_u(x) &:= a_{n-1}x^{(n-2)/2} + a_{n-3}x^{(n-4)/2} + \cdots + a_3 x + a_1 \\ P_g(x) &:= a_{n-2}x^{(n-2)/2} + a_{n-4}x^{(n-4)/2} + \cdots + a_2 x + a_0 \end{aligned}$$

Dann gilt

$$\begin{pmatrix} 1 & 1 & \dots & 1 \\ 1 & \eta & \dots & \eta^{n-1} \\ 1 & \eta^2 & \dots & \eta^{2(n-1)} \\ 1 & \vdots & \ddots & \vdots \\ 1 & \eta^{n/2} & \dots & \\ 1 & \eta^{n/2+1} & \dots & \vdots \\ 1 & \vdots & \ddots & \vdots \\ 1 & \eta^{n-1} & \dots & \eta^{(n-1)(n-1)} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n/2} \\ a_{n/2+1} \\ \vdots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} P(1) = P_g(1^2) + 1 \cdot P_u(1^2) \\ P(\eta) = P_g(\eta^2) + \eta \cdot P_u(\eta^2) \\ P(\eta^2) = P_g(\eta^4) + \eta^2 \cdot P_u(\eta^4) \\ \vdots \\ P(\eta^{n/2}) = P(-1) = P_g(1^2) - 1 \cdot P_u(1^2) \\ P(\eta^{n/2+1}) = P(-\eta) = P_g(\eta^2) - \eta \cdot P_u(\eta^2) \\ \vdots \\ \otimes \end{pmatrix}$$

Algorithmus Polynommultiplikation mit FFT

Gegeben: $p, q \in \mathbb{R}[x]$,$\operatorname{grad} p = \operatorname{grad} q \leq n/2$, $p = p_{n-1}x^{n-1} + \cdots + p_0$, $q = q_{n-1}x^{n-1} + \cdots + q_0$, $n = 2^k$, $k \in \mathbb{N}$

Gesucht: $p \cdot q$

Ablauf:

- Werte p und q mittels FFT an den gleichen n Stellen aus, also $\vec{p} := (p_0, \dots, p_{n-1})$, $\vec{q} := (q_0, \dots, q_{n-1})$, $\vec{s} := \operatorname{DFT} \vec{p}$, $\vec{t} := \operatorname{DFT} \vec{q}$
- $\vec{r} := (s_0 \cdot t_0, \dots, s_{n-1} \cdot t_{n-1})$
- Wende inverse FFT auf r an: $\operatorname{DFT}^{-1} \vec{r}$
- Interpretiere $\vec{r}$ wie oben

5 Fourier-Transformation

5.1 Fourier-Reihe für periodische kontinuierliche Signale

Definition 5.1 Kontinuierliches Signal

Ein kontinuierliches Signal sei hier eine Funktion $f : \mathbb{R} \to \mathbb{R}$, dargestellt als $f(x) := y$, wobei $x, y \in \mathbb{R}$.

Definition 5.2 Periodisches Signal

Gilt für alle $x \in \mathbb{R}$ und ein $T \in \mathbb{R}^+$

$$f(x) = f(x+T)$$

so heißt f periodisch mit der Periode T. (kurz: T-periodisch) Existiert ein kleinstes $T_0 \in \mathbb{R}^+$, so dass f T_0-periodisch ist, so nennt man T_0 die Grundperiode von f.

Definition 5.3 Kreisfrequenz

Den einer Periode T zugeordneten Wert

$$\omega = \frac{2\pi}{T}$$

nennt man die Kreisfrequenz. Die der Grundperiode T_0 eines Signals zugeordnete Kreisfrequenz heißt Grundkreisfrequenz ω_0.

Definition 5.4 Trigonometrische Reihe

Seien $a_k, b_k \in \mathbb{R}$ ($k \in \mathbb{N}_0$). Dann heißt eine Funktionenreihe der Gestalt

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n \cos(\omega_0 n x) + b_n \sin(\omega_0 n x)$$

eine trigonometrische Reihe oder Fourierreihe mit Periode T_0.

Satz 5.1

Voraussetzung: f trigonometrische Reihe

Sei

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n \cos(\omega_0 n x) + b_n \sin(\omega_0 n x)$$

mit $a_n, b_n \in \mathbb{R}$. Dann legt man für $k = 1 \ldots n$ fest

$$c_k := \frac{a_k - i b_k}{2}$$
$$c_0 := \frac{a_0}{2}$$
$$c_{-k} := \frac{a_k + i b_k}{2}$$

Dann gilt

$$\begin{aligned}
f(x) &= \sum_{n=-\infty}^{\infty} c_n e^{i\omega_0 nx} \\
&= \underbrace{\sum_{n=1}^{\infty} \frac{a_n - ib_n}{2}(\cos(-\omega_0 nx) + i\sin(-\omega_0 nx))}_{\text{“negative Hälfte”}} + \frac{a_0}{2} + \underbrace{\sum_{n=1}^{\infty} \frac{a_n + ib_n}{2}(\cos(\omega_0 nx) + i\sin(\omega_0 nx))}_{\text{“positive Hälfte”}} \\
&= \frac{a_0}{2} + \sum_{n=1}^{\infty} \frac{a_n - ib_n}{2}(\cos(\omega_0 nx) - i\sin(\omega_0 nx)) + \underbrace{\frac{a_n + ib_n}{2}(\cos(\omega_0 nx) + i\sin(\omega_0 nx))}_{\text{konjugiert komplex zum 1. Summanden}} \\
&= \frac{a_0}{2} + \sum_{n=1}^{\infty} 2\Re(\frac{a_n + ib_n}{2}(\cos(\omega_0 nx) - i\sin(\omega_0 nx)) \\
&= \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n \cos(\omega_0 nx) + b_n \sin(\omega_0 nx)
\end{aligned}$$

Definition 5.5 Integral über beliebiges Intervall

Der Ausdruck

$$\int_T f(x)dx = a$$

mit $f : \mathbb{R} \to \mathbb{R}$, $T, a \in \mathbb{R}$ bedeutet

$$(\forall s \in \mathbb{R})(\int_s^{s+T} f(x)dx = a)$$

Satz 5.2 Orthogonalitätsrelationen

Voraussetzung: $m, n \in \mathbb{N}$

Dann gilt:

(1)

$$\int_{2\pi} \sin(mx) \cdot \sin(nx)dx = \begin{cases} 0 & \text{für } m \neq n \\ \pi & \text{für } m = n > 0 \\ 0 & \text{für } m = n = 0 \end{cases}$$

(2)

$$\int_{2\pi} \sin(mx) \cdot \cos(nx)dx = 0$$

(3)

$$\int_{2\pi} \cos(mx) \cdot \cos(nx) dx = \begin{cases} 0 & \text{für } m \neq n \\ \pi & \text{für } m = n > 0 \\ 2\pi & \text{für } m = n = 0 \end{cases}$$

(4) Oder, deutlich kürzer und eleganter:

$$\int_{2\pi} e^{inx} \cdot e^{-imx} dx = \begin{cases} 0 & \text{für } m \neq n \\ 2\pi & \text{für } m = n \end{cases}$$

(5) Und noch allgemeiner:

$$\int_{T_0} e^{i\omega_0(n-m)x} dx = \begin{cases} 0 & \text{für } m \neq n \\ T_0 & \text{für } m = n \end{cases}$$

Satz 5.3

Voraussetzung: $f : (-T_0/2, T_0/2] \to \mathbb{R}$ integrierbar und durch eine konvergente trigonometrische Reihe darstellbar

Sei

$$g(x) := \sum_{n=-\infty}^{\infty} c_n e^{i\omega_0 nx}$$

mit

$$c_n := \frac{1}{T_0} \int_{T_0} f(x) e^{-i\omega_0 nx} dx$$

Dann gilt $f(x) = g(x)$.

Satz 5.4 Dirichletsche Bedingungen

Voraussetzung: $f : \mathbb{R} \to \mathbb{R}$ T_0-periodisch

Gelten für f die folgenden drei Bedinungen

(1) f ist absolut integrierbar:

$$\int_{T_0} |f(x)| dx < \infty$$

(2) f ist auf ganz $\mathbb{R}$ von beschränkter Variation

(3) f hat auf jedem endlichen Intervall höchstens endlich viele Unstetigkeitsstellen.

so ist f als Fourier-Reihe darstellbar.

Bemerkung

Insbesondere erfüllen stückweise stetig differenzierbare Funktionen die Dirichletschen Bedinungen.

5.2 Fourier-Transformation für aperiodische kontinuierliche Signale

Definition 5.6 Fourier-Transformation

Sei f ein aperiodisches Signal, für das gilt $(\forall |x| > T_1)(f(x) = 0)$. Sei $\tilde{f}(x)$ die periodische Fortsetzung von f mit Periode $T_0 > T_1$. Betrachte Satz 5.3. Setzt man

$$\mathcal{F}(\tilde{f})(n\omega_0) = T_0 c_n = \int\limits_{T_0} \tilde{f}(x) e^{-i\omega_0 n x} dx$$

so gilt

$$\tilde{f}(x) = \frac{1}{2\pi} \sum_{n=-\infty}^{\infty} \mathcal{F}(\tilde{f})(n\omega_0) e^{i\omega_0 n x} \omega_0$$

Lässt man nun $T_0 \to \infty$, also $\omega_0 \to 0$ (und somit auch "$\tilde{f} \to f$") gehen und verallgemeinert man $\mathcal{F}(f)(n\omega_0)$ für beliebige ω ("$\omega := n\omega_0$"), erhält man die Analysegleichung

$$\mathcal{F}(f)(\omega) = \int\limits_{-\infty}^{\infty} f(x) e^{-i\omega x} dx$$

und die Synthesegleichung

$$\mathcal{F}^{-1}(\mathcal{F}(f))(x) = f(x) = \frac{1}{2\pi} \int\limits_{-\infty}^{\infty} \mathcal{F}(f)(\omega) e^{i\omega x} d\omega$$

der Fouriertransformation. Hierbei nennt sich $\mathcal{F}(f) : \mathbb{R} \to \mathbb{R}$ die Fourier-Transfomierte bzw. das Spektrum von f.

Bemerkung Weitere Spektren

Weiterhin unterscheidet man folgende Spektren:

- Amplitudenspektrum $|\mathcal{F}(f)|$
- Phasenspektrum $\varphi(\omega) := \arctan \frac{\Im(\mathcal{F}(f))}{\Re(\mathcal{F}(f))}$

- Leistungsspektrum $|\mathcal{F}(f)|^2$

Satz 5.5 Dirichletsche Bedingungen

Voraussetzung: $f : \mathbb{R} \to \mathbb{R}$

Gelten für f die folgenden drei Bedinungen

(1) f ist absolut integrierbar:

$$\int_{-\infty}^{\infty} |f(x)|dx < \infty$$

(2) f ist auf ganz $\mathbb{R}$ von beschränkter Variation

(3) f hat auf jedem endlichen Intervall höchstens endlich viele Unstetigkeitsstellen.

so hat f eine Fourier-Transformierte und es gilt $\mathcal{F}^{-1} \circ \mathcal{F}(f) = f$.

Definition 5.7 Dirac-Distribution

Die Zuordnung $\delta : \mathbb{R} \to \mathbb{R}$ mit

$$\delta(x) := \begin{cases} 0 & \text{für } |x| > \varepsilon \\ \frac{1}{2\varepsilon} & \text{für } |x| \leq 2\varepsilon \end{cases}$$

und $\varepsilon \to 0+$ ist eine mögliche Annäherung der Dirac-Distribution.

Definition 5.8 Einheitssprung

Das kontinuierliche Signal

$$\sigma(x) := \begin{cases} 0 & \text{für } x < 0 \\ 1 & \text{für } x \geq 0 \end{cases}$$

nennt man das Einheitssprung-Signal.

Bemerkung

Näherungsweise gelten:

(1)

$$\frac{d\sigma}{dx} = \delta(x)$$

(2)

$$\int_{-\infty}^{\infty} \delta(x)dx = 1$$

(3)

$$\int_{-\infty}^{\infty} f(x)\delta(x)dx = f(0)$$

Aus (3) folgt, dass die Fouriertransformierte des Impulses konstant 1 ist.

Beispiel

Es gelten:

$$\begin{aligned}
\mathcal{F}(1)(\omega) &= 2\pi\delta(\omega) \\
\mathcal{F}(\delta(x))(\omega) &= 1 \\
\mathcal{F}(\sigma(x))(\omega) &= \pi\delta(\omega) - i\frac{1}{\omega} \\
\mathcal{F}(e^{i\omega_0 x})(\omega) &= 2\pi\delta(\omega - \omega_0) \\
\mathcal{F}(\sin(\omega_0 x))(\omega) &= -i\pi[\delta(\omega - \omega_0) + \delta(\omega + \omega_0)] \\
\mathcal{F}(\cos(\omega_0 x))(\omega) &= \pi[\delta(\omega - \omega_0) - \delta(\omega + \omega_0)] \\
\mathcal{F}(\sum_{k=-\infty}^{\infty} \delta(x - kT_0))(\omega) &= \omega_0 \sum_{k=-\infty}^{\infty} \delta(\omega - k\omega_0)
\end{aligned}$$

Bemerkung

Sei f durch eine trigonometrische Reihe darstellbar. Dann folgt mit Hilfe des Beispiels, dass $\mathcal{F}(f)$ aus einer Summe von Impulsen besteht, deren Intensitäten den Fourier-Koeffizienten entsprechen.

Definition 5.9 Faltung

Seien $f, g : \mathbb{R} \to \mathbb{R}$ Funktionen. Dann nennt man

$$(f * g)(t) := \int_{-\infty}^{\infty} f(t - \tau)g(\tau)d\tau$$

die Faltung von f und g.

Satz 5.6 Eigenschaften der Fourier-Transformation

Voraussetzung: f, g kontinuierliche Signale, $a, b \in \mathbb{C}$, $t_0 \in \mathbb{R}$

Es gelten

(1) Linearität:

$$\mathcal{F}(af + bg) = a\mathcal{F}(f) + b\mathcal{F}(g)$$

(2) Differentiation:

$$\mathcal{F}(f^{(n)}) = (i\omega)^n \mathcal{F}(f)$$

(3) Originalverschiebung:

$$\mathcal{F}(f(x - x_0))(\omega) = e^{-i\omega x_0}\mathcal{F}(f)(\omega)$$

(4) Dehnung:

$$\mathcal{F}(f(ax))(\omega) = \frac{1}{|a|}\mathcal{F}(f)(\frac{\omega}{a})$$

(5) Faltung:

$$\mathcal{F}(f * g) = \mathcal{F}(f) \cdot \mathcal{F}(g)$$

(6) Konjugierte Symmetrie:

$$\mathcal{F}(f)(-\omega) = \mathcal{F}^*(f)(\omega)$$

wobei x^* die zu x konjugiert komplexe Zahl darstellt.

Bemerkung

Aus der konjugierten Symmetrie ergibt sich (durch Substitution im Integral), dass gerade Signale ($f(x) = f(-x)$) eine rein reelle und gerade Fouriertransformierte haben, während reelle ungerade Signale eine rein imaginäre und ungerade Fouriertransformierte haben.

Satz 5.7 Reziprozität der Fouriertransformation

Voraussetzung: f kontinuierliches Signal, $g := \mathcal{F}(f)$

Falls g existiert, gilt

$$\mathcal{F}(g)(\omega) = 2\pi f(-\omega)$$

5.3 Fourier-Reihe für periodische diskrete Signale

Definition 5.10 Diskretes Signal

Ein diskretes Signal sei hier eine Funktion $f : \mathbb{Z} \to \mathbb{R}$, dargestellt als $f[x] := y$, wobei $x \in \mathbb{Z}$ und $y \in \mathbb{R}$.

Definition 5.11 Periodisches Signal

Gilt für alle $x \in \mathbb{Z}$ und ein $N \in \mathbb{N}$

$$f[x] = f[x + N]$$

so heißt f periodisch mit der Periode N. (kurz: N-periodisch) Existiert ein kleinstes $N_0 \in \mathbb{N}$, so dass f N_0-periodisch ist, so nennt man N_0 die Grundperiode von f.

Definition 5.12 Kreisfrequenz

Den einer Periode N zugeordneten Wert

$$\omega = \frac{2\pi}{N}$$

nennt man die Kreisfrequenz. Die der Grundperiode N_0 eines Signals zugeordnete Kreisfrequenz heißt Grundkreisfrequenz ω_0.

Satz 5.8 Periodizität von diskreten komplex exponentiellen Signalen

Voraussetzung: f diskretes komplex exponentielles Signal

Sei $f[x] := e^{i\omega x}$. Dann ist f genau dann periodisch, wenn es $m, N \in \mathbb{N}$ gibt mit $\frac{\omega}{2\pi} = \frac{m}{N}$. Die Grundperiode ist dann $N_0 = N = m\frac{2\pi}{\omega}$. Dass solche m, N existieren sei ab jetzt immer vorausgesetzt, wenn ω_0 erscheint.

Definition 5.13 Summe über beliebigen Bereich

Der Ausdruck

$$\sum_{n=<N>} f[n] = a$$

mit $f : \mathbb{Z} \to \mathbb{R}$, $N \in \mathbb{N}, a \in \mathbb{R}$ bedeutet

$$(\forall s \in \mathbb{Z})(\sum_{n=s}^{s+N} f[n] = a)$$

Definition 5.14 Trigonometrische Reihe

Seien $c_n \in \mathbb{R}$ ($n \in \mathbb{Z}$). Dann heißt eine Funktionenreihe der Gestalt

$$f[x] = \sum_{n=<N_0>} c_n e^{i\omega_0 nx}$$

eine trigonometrische Reihe oder Fourierreihe mit Periode N_0.

Bemerkung

Da gilt

$$e^{i\omega_0 n} = e^{i\omega_0 (n+N_0)}$$

genügen für eine N_0-periodische Fourierreihe N_0 Koeffizienten.

Satz 5.9

Voraussetzung: $k \in \mathbb{N}$

Dann gilt:

$$\sum_{n=<N_0>} e^{i\omega_0 nk} = \begin{cases} N_0 & \text{für } k = zN_0 \ (z \in \mathbb{Z}) \\ 0 & \text{sonst} \end{cases}$$

Satz 5.10

Voraussetzung: $f : \mathbb{Z} \to \mathbb{R}$ diskretes N_0-periodisches Signal

Sei

$$g[x] := \sum_{n=<N_0>} c_n e^{i\omega_0 nx}$$

mit

$$c_n := \frac{1}{N_0} \sum_{x=<N_0>} f[x] e^{-i\omega_0 nx}$$

Dann gilt $f[x] = g[x]$.

Bemerkung

Diese Zerlegung existiert für jedes N_0-periodische Signal.

5.4 Fourier-Transformation für aperiodische diskrete Signale

Definition 5.15 Fourier-Transformation

Sei f ein aperiodisches Signal, für das gilt $(\forall |x| > N_1)(f[x] = 0)$. Sei $\tilde{f}$ die periodische Fortsetzung von f mit Periode $N_0 > N_1$. Betrachte Satz 5.10. Setzt man

$$\mathcal{F}(\tilde{f})(n\omega_0) = N_0 c_n = \sum_{x=<N_0>} \tilde{f}[x] e^{-i\omega_0 n x}$$

so gilt

$$\tilde{f}[x] = \frac{1}{2\pi} \sum_{n=<N_0>} \mathcal{F}(\tilde{f})(n\omega_0) e^{i\omega_0 n x} \omega_0$$

Lässt man nun $N_0 \to \infty$, also $\omega_0 \to 0$ (und somit auch "$\tilde{f} \to f$") gehen und verallgemeinert man $\mathcal{F}(f)(n\omega_0)$ für beliebige ω ("$\omega := n\omega_0$"), erhält man die Analysegleichung

$$\mathcal{F}(f)(\omega) = \sum_{x=-\infty}^{\infty} f[x] e^{-i\omega x} dx$$

und die Synthesegleichung

$$\mathcal{F}^{-1}(\mathcal{F}(f))[x] = f[x] = \frac{1}{2\pi} \int_{2\pi} \mathcal{F}(f)(\omega) e^{i\omega x} d\omega$$

der Fouriertransformation. Hierbei nennt sich $\mathcal{F}(f) : \mathbb{R} \to \mathbb{R}$ die Fourier-Transfomierte bzw. das Spektrum von f.

Bemerkung

Hier wirkt sich die Reziprozität deutlich aus: Bei der kontinuierlichen Fourierreihe war das Spektrum diskret, jedoch das Signal kontinuierlich, hier ist das Spektrum kontinuierlich, jedoch das Signal diskret.

Bemerkung

Die Eigenschaften der diskreten Fouriertransformation ergeben sich analog zur kontinuierlichen.

5.5 Ergänzungen

Definition 5.16 DFT

Sei f ein aperiodisches diskretes Signal, für das gilt $(\forall x \notin \{0, \ldots, N_1 - 1\})(f[x] = 0)$. Sei $\tilde{f}$ die periodische Fortsetzung von f mit Periode $N_0 > N_1$ so, dass

$f[x] = \tilde{f}[x]$ für alle $x \in \{0, \ldots, N_0 - 1\}$. Betrachte Satz 5.10. Setzt man

$$\mathrm{DFT}(f)[n] = c_n = \frac{1}{N_0} \sum_{x=0}^{N_0-1} \tilde{f}[x] e^{-i\omega_0 nx}$$

so gilt

$$f[x] \approxeq \tilde{f}[x] = \sum_{x=0}^{N_0-1} \mathrm{DFT}(f)[n] e^{i\omega_0 nx}$$

Dann nennt man $\mathrm{DFT}(f)$ die diskrete Fouriertransformierte von f.

Bemerkung

Die diskrete Fouriertransformation ist *nicht* gleich der Fouriertransformation eines diskreten Signals. (dieser aber in gewisser Weise ähnlich)

Bemerkung

Da gilt

$$e^{i\omega_0 nx} = (\underbrace{e^{i2\pi/N_0}}_{N_0\text{-te Einheitswurzel}})^{xn}$$

lässt sich die DFT mit Hilfe der DFT-Matrix auf einen Schlag berechnen. (Beachte formale Übereinstimmung der DFT-Synthesegleichung mit der Matrixmultiplikation mit der DFT, ebenso der DFT-Analyse mit der Inversen der DFT-Matrix)

Definition 5.17 Bandbegrenztheit

Sei $f : \mathbb{R} \to \mathbb{R}$, eine Funktion, $F : \mathbb{R} \to \mathbb{C}$ ihre Fouriertransformierte. Dann heißt f bandbegrenzt mit der Grenzkreisfrequenz ω_g genau dann, wenn gilt

$$f(t) = \frac{1}{2\pi} \int_{-\omega_g}^{\omega_g} F(\omega) d\omega$$

Bemerkung

Für eine mit ω_g bandbegrenzte Funktion folgt $\omega > \omega_g \implies F(\omega) = 0$.

Satz 5.11 Abtasttheorem

Voraussetzung: $f : \mathbb{R} \to \mathbb{R}$ ω-periodisch, bandbegrenzt mit ω_g

Sei $n := \frac{\omega_g}{\omega}$. Wähle ein t_s mit $|t_s| < \frac{\pi}{\omega_g}$ und ein $t_0 \in \mathbb{R}$ beliebig. Dann ist f durch die Funktionswerte $f(t_0 - nt_s), \ldots, f(t_0 + nt_s)$ eindeutig bestimmt.

Bemerkung

Betrachtet man das Abtasten/Samplen als Faltung mit einer Impulssequenz der Frequenz ω_0, so ergibt sich als Fouriertransformierte eine Reihe von Spektren im Abstand ω_0. Rücken diese nun so dicht aneinander, dass sie sich überlappen (ist also der Abstand der Spektren ω_0 kleiner als deren jeweilige Ausdehnung $2\omega_g$),so überlappen sich diese (Aliasing) und das ursprüngliche Signal ist nicht mehr rekonstruierbar.

6 Information und Kodierung

Definition 6.1 Alphabet

Ein Alphabet A ist eine geordnete endliche Menge von Symbolen.

$$A = (a_1, a_2, \ldots, a_n)$$

Definition 6.2 Verteilung eines Alphabets

Als Verteilung p_A eines Alphabets A bezeichnet man einen Vektor von normierten Wahrscheinlichkeiten, der für jedes Zeichen angibt, mit welcher Wahrscheinlichkeit es auftritt.

$$p_A = (p(a_1), p(a_2), \ldots, p(a_n))$$

"Normiert" bedeutet, dass gelten muss

$$1 = \sum_{k=1}^{n} p(a_i)$$

Definition 6.3 Übertragungseinrichtung

Zu einer Übertragungseinrichtung gehören ein Sender mit einem Senderalphabet $S = (s_1, s_2, \ldots, s_n)$ und einer Verteilung $p_S = (p(s_1), p(s_2), \ldots, p(s_n))$, ein Empfänger mit einem mit einem Empfängeralphabet $E = (e_1, e_2, \ldots, e_m)$ und einer Verteilung $p_E = (p(e_1), p(e_2), \ldots, p(e_m))$ und einem Übertragungskanal, der definiert wird durch eine Matrix von bedingten Wahrscheinlichkeiten $p_K = ((p(e_i|s_j)))$.

Bemerkung

Bei dieser Definition einer Übertragungseinrichtung ist die Verteilung des Empfängers durch Senderverteilung und Kanalmatrix in folgender Weise vollständig bestimt:

$$p(e_i) = \sum_{j=1}^{n} p(e_i|s_j) \cdot p(s_j)$$

Definition 6.4 Information

Gesucht ist als $I(p)$ der mittlere Informationswert eines Zeichens aus einem Alphabet A mit der Verteilung p_A, bzw. als $I(p_S, p_E)$ der mittlere Informationswert eines Zeichens aus dem Empfängeralphabet E mit der möglicherweise über die Kanalmatrix p_K von der Senderverteilung p_S abhängigen Verteilung p_E. An beide stellt man folgende Forderungen:

(1) **Stetigkeit:** $I(p_A)$ und $I(p_S, p_E)$ sind für die Wahrscheinlichkeitswerte $\in [0, 1]$ stetig.

(2) **Positivität:** $I(p_A) \geq 0$, $I(p_S, p_E) \geq 0$

(3) **Sicherheit:** Man nennt eine Verteilung degeneriert $:\Leftrightarrow$ sie ist eine Permutation von $(1, 0, \ldots, 0)$. Ist p degeneriert, gilt $I(p_A) = 0$, sind p_S und p_E beide degeneriert, so gilt $I(p_S, p_E) = 0$.

(4) **Unsicherheit:** Maximum der Information genau dann, wenn p bzw. p_S, p_E Gleichverteilungen sind.

(5) **Additivität:** Sind p_S und p_E unabhängige Verteilungen, dann gilt $I(p_S, p_E) = I(p_S) + I(p_E)$. (bezogen auf obiges Modell müsste gelten $p(e_i|s_j) = p(e_i)$)

Die beiden folgenden Terme erfüllen die obigen Forderungen:

$$I(p) = \sum_{i=1}^{n} p(a_i) \log_2 \frac{1}{p(a_i)}$$

$$I(p_S, p_E) = \sum_{i=1}^{n} \sum_{j=1}^{n} p(e_i|s_j) \cdot p(s_j) \log_2 \frac{1}{p(e_i|s_j) \cdot p(s_j)}$$

Hierbei muss man zur Wahrung der Stetigkeit $0 \log_2 0 := 0$ festlegen. Der Zahlenwert $I(p)$ wird als Information, Unsicherheit oder Entropie bezeichnet. Die Information $I(p)$ wird in bit angegeben.

Definition 6.5 bit

Die Information $I(p)$ zweier sich gegenseitig ausschliessender Ereignisse, die beide gleich wahrscheinlich sind, ist 1 bit. (Ein bit darf nicht mit dem technischen Ausdruck Bit verwechselt. Ersteres ist das Zählmaß der Information, während letzteres einer Binärziffer entspricht.)

Definition 6.6 Mittlere Wortlänge

Sei N_i die Anzahl Bits, die zur Kodierung des Zeichens a_i aus dem Alphabet A mit der Verteilung p_A verwendet wird. Dann heißt

$$L(p_A) := \sum_{i=1}^{n} p_A(a_i) N_i$$

die mittlere Wortlänge von A bezüglich p_A und den Zeichenlängen N_i

Satz 6.1 Kodierungstheorem von Shannon

Voraussetzung: A Alphabet mit Verteilung p_A, N_i Zeichenlängen

Dann gilt

$$I(p_A)\,\text{Bit} \leq L(p_A)$$

Definition 6.7 Datei

Sei A ein Alphabet. Dann ist eine Datei $(f_i) : \{0, \ldots, m\} \to A$ eine endliche Folge f_i von Elementen von A.

Definition 6.8 Arten von Codes

Ein Code heißt

- *präfixfrei* bzw. *Präfixcode*, wenn keines seiner Symbole Präfix eines anderen ist
- *eindeutig* bzw. nicht *ambig*, wenn es zu seiner Dekodierung nur eine Möglichkeit gibt
- *sofort dekodierbar* wenn zu seiner Dekodierung kein Lookahead erforderlich ist

Algorithmus Erzeugung des Huffman-Codes

Gegeben: Alphabet $A = (a_1, \ldots, a_n)$, Datei $(f_i)_{i=0}^m$

Gesucht: Huffman-Baum T, der einen optimalen präfixfreien binären Code für die gegebene Datei repräsentiert

Ablauf:

- Berechne für jedes Zeichen a_i die absolute Häufigkeit $n(a_i)$ in (f_i)
- Baue einen Heap H auf, der die Zeichen von A nach ihrer absoluten Häufigkeit $n(a_i)$ ordnet (geringste zuoberst)
- Lege einen leeren Baum T an
- Solange H nicht leer ist
  - Ist nur noch ein Zeichen c in H enthalten, so nimm den c entsprechenden Knoten in T als Wurzel und beende
  - Entferne die zwei Zeichen c und d mit geringster Häufigkeit aus dem Heap

– Erzeuge ein Pseudozeichen z, das in T Elternknoten von c und d wird. Setze $n(z) := n(c) + n(d)$. Füge z mit $n(z)$ in H ein.

Definition 6.9 Kolmogorov-Komplexität

Ordnet einer Datenmenge die Länge des kürzesten Programms, das diese Datenmenge generiert, als Komplexität zu.

Algorithmus RLE-Kompression

Kodiert Wiederholungen durch Blöcke der Art (Anzahl,Datenelement).

Definition 6.10 Parität

Das Paritätsverfahren besteht im Anfügen eines zusätzlichen Bits an eine Bitsequenz, so dass eine bestimmte Eigenschaft der Gesamtsequenz sichergestellt ist. Man unterscheidet

- Das Anfügen eines Bits so, dass die Anzahl der Einsen stets gerade ist (even parity)
- Das Anfügen eines Bits so, dass die Anzahl der Einsen stets ungerade ist (odd parity)

Definition 6.11 Rechteck-Code

Man denke sich eine Sequenz von $m \cdot n$ Bits ($m, n \in \mathbb{N}$) in einer $m \times n$-Matrix angeordnet. Bildet man nun über jede Spalte und jede Zeile eine Parität, so kann man bei einem fehlerhaften Bit anhand der beiden falschen Paritäten das fehlerhafte Bit erkennen und "umdrehen".

Analog: Dreieck-Code, kubische Codes

Bemerkung

Hat man für zwei Bitsequenzen die Paritäten p_1 und p_2 ermittelt, so lässt sich mit Hilfe der Operation p_1 xor p_2 die Parität der gesamten Sequenz ermitteln.

Definition 6.12 Prüfsumme

Für eine Datei $(f_i)_{i=1}^n$ gibt es verschiedene Varianten, Prüfsummen zu ermitteln:

- Einfach:

$$\sum_{i=1}^{n} f_i$$

- Gewichtet:

$$\sum_{i=1}^{n} i \cdot f_i$$

Definition 6.13 Verlustbehaftete Kompression

Man spricht von verlustbehafteter Kompression, wenn aus den komprimierten Daten die Originaldaten nicht vollständig identisch rekonstruiert werden können.

Algorithmus k-Mittelwerte Algorithmus

Gegeben: Ein Vektorraum V mit Abstandsfunktion $d(x, y) : V \times V \to \mathbb{R}$, $v_1, \ldots, v_n \in V$, $k \in \mathbb{N}$

Gesucht: Ein Codebuch $C = \{\mu_1, \ldots, \mu_k\}$, so dass eine Abbildung $\varphi : V \to \{0, \ldots, k\}$ gefunden werden kann, für die

$$\sum_{i=1}^{n} d(\mu_{\varphi(v_i)}, v_i)$$

einem lokalen Minimum nahekommt.

Ablauf:

- Initialisiere μ_i mit beliebigen v_j. ($i \in \{1, \ldots k\}$,$j \in \{1, \ldots n\}$)
- Solange Quantisierungsfehler zu groß
  - Zähle je Codebuchvektor μ_i die auf diesen Vektor abgebildeten Mustervektoren, setze deren Anzahl

$$n[i] := |\{j \mid \varphi(v_j) = i\}|$$

  - Lege einen neuen Satz von Codebuchvektoren $\mu'_1, \ldots, \mu'_k$ nach folgendem Verfahren fest:

$$\mu'_i := \frac{1}{n[i]} \sum_j \begin{cases} v_j & \varphi(j) = i \\ 0 & \text{sonst} \end{cases}$$

  - Nimm das Codebuch (μ'_i) als neues Codebuch (μ_i)

Bemerkung

Die Menge aller jeweils einem Vektor aus dem Codebuch zugeordneten Vektoren nennt man Voronoi-Region.

7 Problemklassen

Definition 7.1 Entscheidungsproblem

Ein Entscheidungsproblem ist ein Problem, dessen Lösung für alle möglichen Eingaben in einer Ja-Nein-Antwort besteht.

Bemerkung

Sei U die Menge aller möglichen Eingaben eines Entscheidungsproblems, und sei $L \subseteq U$ die Menge aller Eingaben, für die die Antwort "ja" lautet. Dann nennt man L die dem Problem entsprechende Sprache. Folglich kann jedes Entscheidungsproblem als ein Spracherkennungsproblem aufgefasst werden. Die Begriffe "Problem" und "Sprache" können weiterhin als gleichwertig aufgefasst werden.

Definition 7.2 Polynomialer Algorithmus

Ein Algorithmus heißt polynomial genau dann, wenn seine asymptotische Laufzeit $O(T(n))$ durch ein Polynom dargestellt werden kann.

U.a. erfüllen polynomiale Algorithmen die Beziehung

$$T(cn) = O(T(n)) \quad c \in \mathbb{R}^+$$

Die Begriffe effizient bzw. machbar (tractable) sind in Bezug auf Algorithmen synonym zu polynomial.

Bemerkung

Man spricht auch von einem "polynomialen Problem" und meint damit die Polynomialität des Algorithmus, der das Problem in den asymptotisch wenigsten Schritten löst.

Bemerkung

Man kann zeigen, dass die meisten "sinnvollen" Rechnerarchitekturen einander in polynomialer Zeit emulieren können. Damit ist jeder polynomiale Algorithmus auf jeder "sinnvollen" Maschine polynomial.

Definition 7.3 Reduzierbarkeit

Seien $L_1 \subseteq U_1$ und $L_2 \subseteq U_2$ zwei Sprachen. Dann nennt man L_1 reduzierbar auf L_2, falls es einen Algorithmus $f : U_1 \to U_2$ gibt, so dass gilt $f(L_1) = L_2$.

Definition 7.4 Polynomiale Reduzierbarkeit

Seien $L_1 \subseteq U_1$ und $L_2 \subseteq U_2$ zwei Sprachen. Dann nennt man L_1 polynomial reduzierbar auf L_2, falls es einen polynomialen Algorithmus $f : U_1 \to U_2$ gibt, so dass gilt $f(L_1) = L_2$.

Definition 7.5 Polynomiale Äquivalenz

Seien $L_1 \subseteq U_1$ und $L_2 \subseteq U_2$ zwei Sprachen. Dann nennt man L_1 polynomial äquivalent zu L_2, falls L_1 und L_2 gegenseitig aufeinander polynomial reduzierbar sind.

Satz 7.1

Voraussetzung: L_1, L_2 Sprachen, L_2 polynomial erkennbar, L_1 auf L_2 polynomial reduzierbar

Dann existiert auch ein polynomialer Algorithmus für die Erkennung von L_1.

Bemerkung

Polynomiale Reduzierbarkeit und damit auch polynomiale Äquivalenz sind transitive Relationen.

Hat man für ein Problem ein Laufzeitminimum bewiesen, und kann dieses Problem auf ein anderes reduziert werden, so hat auch das letztere das für das erste bewiesene Laufzeitminimum. (Voraussetzung: Aufwand des Algorithmus asymptotisch größer als Aufwand der Reduktion)

Problem Lineare Optimierung

Gegeben ist eine Zielfunktion $f : \mathbb{R}^n \to \mathbb{R}$ mit folgender Gestalt

$$f(x) := \vec{a} \cdot \vec{x}$$

wobei $\vec{a} = (a_1, \ldots, a_n) \in \mathbb{R}^n$ und $\vec{x} = (x_1, \ldots, x_n) \in U$. Der Wertebereich U, in dem sich die Werte von x bewegen dürfen, wird eingeschränkt durch Bedingungen der folgenden drei Arten

(1) $\vec{\alpha}_i \cdot \vec{x} \leq \beta_i$, wobei $\vec{\alpha}_i \in \mathbb{R}^n, \beta_i \in \mathbb{R}$

(2) $\vec{\gamma}_i \cdot \vec{x} = \delta_i$, wobei $\vec{\gamma}_i \in \mathbb{R}^n, \delta_i \in \mathbb{R}$

(3) $x_i \geq 0$

Für die lineare Optimierung gibt es polynomiale Algorithmen. Weiterhin ist die ganzzahlige lineare Optimierung ($\vec{x} \in \mathbb{N}^n$) ziemlich schwierig.

Problem Problem des Philantropen

n_S Spender wollen an n_E Empfänger spenden, es existieren jedoch Beschränkungen der folgenden Arten:

- Ein Spender hat eine Spendehöchstsumme
- Ein Empfänger hat eine Empfangshöchstsumme
- Ein Spender will nur einen Höchstbetrag an einen bestimmten Spender richten

Welches ist die Verteilung, die die meisten Spendengelder fließen lässt?

Problem Zuweisungsproblem

n_S Spender wollen an n_E Empfänger spenden, es existieren jedoch Beschränkungen der folgenden Arten:

- Ein Spender spendet an höchstens einen Empfänger
- Ein Spender hat eine Spendehöchstsumme
- Ein Empfänger hat eine Empfangshöchstsumme
- Ein Spender will nur einen Höchstbetrag an einen bestimmten Spender richten

Welches ist die Verteilung, die die meisten Spendengelder fließen lässt?

Beispiel

Die folgenden Probleme sind polynomial aufeinander reduzierbar ("→"):

- System verschiedener Repräsentanten → bipartites Matching (Jede Menge und jedes Element in der Gesamtvereinigung bekommt einen Knoten)
- Editierdistanz → Kürzester-Pfad-Problem
- Suche nach Dreiecken in Graphen → Multiplikation boolescher Matrizen
- Netzwerkflussproblem → Lineare Optimierung
- Statisches Routing → Lineare Optimierung
- Problem des Philantropen → Lineare Optimierung
- Zuweisungsproblem → Lineare Optimierung
- Sortieren ↔ Erzeugung eines kreuzungsfreien Polygons
- Multiplikation symmetrischer Matrizen ↔ Multiplikation zweier beliebiger Matrizen
- Quadrieren einer Matrix ↔ Multiplikation zweier beliebiger Matrizen

Diese Aufzählung umfasst alle Beispiele der Vorlesung.

Definition 7.6 Nichtdeterministischer Algorithmus

Ein nichtdeterministischer Algorithmus, der zusätzlich zu den verschiedenen "üblichen" Operationen eine zusätzliche Verzweigungsoperation gestattet, deren beide Zweige gewissermaßen "gleichzeitig" ausgeführt werden. Die Ausführung eines nichtdeterministischen Problems resultiert also in einer Masse von Einzelergebnissen, entsprechend den für verschiedene Verzweigungen ausgewählten Wegen.

Als Ergebnis eines nichtdeterministischen Spracherkennungsalgorithmus gilt die Disjunktion aller Ergebnisse. (mindestens einmal wahr $\Longrightarrow$ wahr, alle falsch $\Longrightarrow$ falsch)

Definition 7.7 Problemklasse P

Man sagt, ein Problem sei in P :$\Leftrightarrow$ es gibt für das Problem einen deterministischen Algorithmus mit polynomialer Laufzeit.

Definition 7.8 Problemklasse NP

Man sagt, ein Problem sei in NP :$\Leftrightarrow$ es gibt für das Problem einen nichtdeterministischen Algorithmus mit polynomialer Laufzeit.

Bemerkung

Offensichtlich gilt P $\subseteq$ NP.

Definition 7.9 K-hart

Sei K eine Problemklasse und P ein Problem. Dann heißt P K-hart, falls jedes Problem aus K polynomial auf P reduzierbar ist.

Definition 7.10 K-vollständig

Sei K eine Problemklasse und P ein K-hartes Problem. Dann heißt P K-vollständig, falls P selbst in K liegt.

Satz 7.2

Voraussetzung: L_1, L_2 Sprachen, L_2 NP-vollständig, L_2 auf L_1 reduzierbar, L_1 in NP

Dann ist auch L_1 NP-vollständig.

Problem SAT

Finde zu einer gegebenen aussagenlogischen Formel in konjunktiver Normalform eine Variablenbelegung, so dass die Auswertung der Formel den Wert "Wahr" ergibt.

Satz 7.3 Satz von Cook

SAT ist NP-vollständig, denn (a) jeder von einer Turingmaschine entscheidbare Algorithmus lässt sich auf SAT zurückführen (b) SAT ist in NP.

Problem Cliquenproblem

Gegeben ist ein Graph G, gesucht ist eine Clique mit k Elementen in G.

Das Cliquenproblem ist NP-vollständig, da SAT darauf reduzierbar ist und es selbst in NP liegt. (Man erzeugt einen Graphen mit einer "Spalte" für jede Klausel mit allen Literalen dieser Klausel, erzeugt dann zwischen je zwei Variablen, die nicht in der gleichen Spalte stehen, und nicht zueinander negiert sind, eine Kante. Dann ist jede Clique eine mögliche Belegung und jede mögliche Belegung eine Clique.)

Problem 3SAT

Gegeben: Eine aussagenlogische Formel in konjunktiver Normalform, bei der pro konjunktiv verbundenem Einzelterm nur drei Variablen vorkommen

Gesucht: Eine Belegung der vorkommenden Variablen, so dass die Auswertung der Formel den Wert "Wahr" ergibt.

SAT ist auf 3SAT reduzierbar, indem man nach folgendem Schema neue Variablen einführt ($k > 3$):

$$(x_1 \vee x_2 \vee \ldots x_k) \Leftrightarrow$$
$$(x_1 \vee x_2 \vee y_1) \wedge (x_3 \vee \neg y_1 \vee y_2) \wedge \ldots \wedge (x_{k-1} \vee x_k \vee \neg y_{k-3})$$

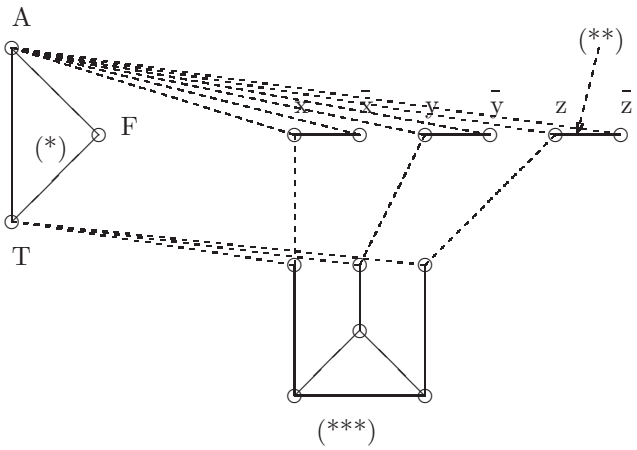
Da 3SAT selbst in NP liegt und SAT darauf reduzierbar ist, ist 3SAT NP-vollständig.

Problem 3COL

Gegeben: Ein Graph $G = (V, E)$

Gesucht: Eine Abbildung $f : V \to \{A, B, C\}$, so dass keine zwei benachbarten Knoten die gleiche "Farbe" (A, B, C) haben.

3SAT ist auf 3COL reduzierbar, indem man nach folgendem Bild die Formel in einen Graphen verwandelt:



Zunächst baut man eine Konstruktion der Art von (*), um die drei verfügbaren Farben an den drei Knoten benennen zu können. Nennen wir sie hier F,T und A (wie False, True und A). Dann stellen Konstruktionen wie (**) sicher, dass jeweils x und $\overline{x}$ mit F oder T bzw. umgekehrt gefärbt werden, indem x und $\overline{x}$ mit A aus (*) verbunden werden.

Zuletzt verunmöglicht die Konstruktion (***) für jede Klausel die Belegung aller drei Variablen einer Klausel mit F. Wäre dies der Fall, wäre die einzige mögliche Belegung für die mit T verbundenen Knoten in (***) die Belegung A, womit die Färbung des darunterliegenden Dreiecks unmöglich würde.

3COL liegt offensichtlich in NP, und da 3SAT polynomial darauf reduziert werden kann, ist 3COL NP-vollständig.

Beispiel Weitere NP-vollständige Probleme

Als weitere NP-vollständige Probleme sind bekannt:

- Das Problem des Handlungsreisenden

Bemerkung

Zum tatsächlichen Lösung von NP-vollständigen Problemen empfiehlt sich eine der folgenden Vorgehensweisen:

- Algorithmus, der nur für bestimmte Eingaben schnell/gut/überhaupt funktioniert
- **Backtracking** Arbeite solange vorwärts, bis es nicht mehr weitergeht, geh' dann nur einen Schritt zurück (verwirf nicht gleich alles), versuche dort anders zu gehen usw.

  Eigenschaften: In der Regel drastische Aufwandsreduktion, aber nicht garantiert. Es wird immer eine exakte Lösung gefunden.

- **Branch and Bound** Backtracking, das schon aufhört, eine Variante zu verfolgen, sobald eine bestimmte Grenze/Bedingung ("boundary") erreicht ist.

  Eigenschaften wie Backtracking.

- Näherungslösungen (Beispiel: Handlungsreisender, zunächst Spannbaum aufstellen, dann mittels DFS Abkürzungen suchen)
- Probabilistische Algorithmen (z.B. genetische Algorithmen)
- Speziallösungen für Einzelfälle

8 Parallelität

Definition 8.1 Zeitaufwand mit mehreren Prozessoren

Mit $T(n,p)$ $(p,n \in \mathbb{N})$ bezeichnet man den Aufwand zur Lösung eines gegebenen Problems mit der Eingabegröße n, wenn p Prozessoren zur Verfügung stehen. (Für unterschiedliche p kommen gegebenfalls unterschiedliche Algorithmen zum Einsatz.)

Definition 8.2 Speedup

Den Speedup $S(n,p)$ (die Beschleunigung) durch die Verwendung von p gegenüber einem Prozessor definiert man durch folgende Gleichung:

$$S(n,p) := \frac{T(n,1)}{T(n,p)}$$

Den Speedup nennt man optimal $:\Leftrightarrow S(n,p) = p$.

Definition 8.3 Effizienz

Die Effizienz $E(n,p)$ eines parallelen Algorithmus gibt die durchschnittliche Auslastung eines Prozessors an:

$$E(n,p) := \frac{S(n,p)}{p} = \frac{T(n,1)}{pT(n,p)}$$

Die Effizienz nennt man optimal $:\Leftrightarrow E(n,p) = 1$.

Bemerkung

In der Regel gilt

$$T(n,p) = X \Leftrightarrow T(n,p/k) \approx kX$$

Definition 8.4 Parallelrechnerarchitekturen

Man unterteilt Parallelrechner nach verschieden Kriterien in Klassen, darunter:

SISD Eine Anweisung wird auf ein Datenwort angewendet.

MISD Mehrere Anweisungen werden parallel auf ein Datenwort angewendet.

SIMD Eine Anweisung wird auf parallel auf mehrere Datenworte angewendet.

MIMD Mehrere Anweisungen werden parallel auf mehrere Datenworte angewendet.

Die bekanntesten Parallelarchitekturen sind:

- Mehrkopf-Turing-Maschine
- Schaltkreis
- systolischer/zellulärer Automat (Transputer, Connection Machine)
- Rechnernetzwerk
- PRAM (parallel random access machine), shared memory machine

Hierunter bilden die PRAMs die wichtigste/häufigste Klasse. Hier unterscheidet man noch nach den Fähigkeiten der Speicherzugriffsarchitektur:

EREW exclusive read exclusive write

CREW concurrent read exclusive write

ERCW exclusive read concurrent write

CRCW concurrent read concurrent write

Im Falle von concurrent write wird ein Vereinheitlichung bezüglich des Schreibergebnisses benötigt, z.B.:

- Es zählt das erste geschriebene Wort
- Es zählt das letzte geschriebene Wort
- Es zählt ein zufälliges geschriebenes Wort
- Es zählt das Wort vom Prozessor mit der kleinsten/größten Nummer
- Alle geschriebenen Worte müssen gleich sein

8.1 Algorithmen für shared memory machines

Algorithmus Parallele Addition

Gegeben: Zwei binäre Zahlen mit n Ziffern, $a = a_{n-1} \dots a_0$, $b = b_{n-1} \dots b_0$,$u \in \{0, 1\}$, $n = 2^m$, $m \in \mathbb{N}$, eine EREW-Maschine mit $O(n)$ Prozessoren

Gesucht: $c = a + b + u = c_n \dots c_0$

Ablauf:

- Basisfall: $n \leq$ Maschinenwortlänge Berechne Ergebnis unmittelbar.
- Sei $k := n/2$. Berechne parallel rekursiv

$$\begin{aligned} l &= a_{n-1} \dots a_k + b_{n-1} \dots b_k \\ l' &= a_{n-1} \dots a_k + b_{n-1} \dots b_k + 1 \\ r &= a_{k-1} \dots a_0 + b_{k-1} \dots b_0 + u \end{aligned}$$

- Setze $v :=$ Übertrag bei r.
- Wähle Ergebnis r

$$r := \begin{cases} lr & v = 0 \\ l'r & v = 1 \end{cases}$$

Aufwand: $O(\log n)$

Definition 8.5 Statischer Algorithmus

Ein paralleler Algorithmus heißt statisch genau dann, wenn die Verteilung der Rechenschritte auf die Prozessoren im Voraus festgelegt ist.

Satz 8.1 Satz von Brent

Voraussetzung: A statischer EREW-Algorithmus, $T(n, p) = O(t)$, s gesamte Anzahl der Rechenschritte

Dann existiert auch ein EREW-Algorithmus B mit einem Aufwand $T(n, s/t) = O(t)$.

Bemerkung

Satz 8.1 lässt sich folgendermaßen interpretieren: Wenn der parallele Algorithmus nicht mehr Schritte als der sequentielle macht, dann ist die optimale Effizienz erreichbar.

Algorithmus Maximumsuche I

Gegeben: : $a_1, \ldots, a_n$ aus einer angeordneten Menge, $n \in \mathbb{N}$, CRCW-Maschine mit $n(n-1)/2$ Prozessoren, $n = 2^m$, $m \in \mathbb{N}$

Gesucht: $\max\{a_1, \ldots, a_n\}$

Ablauf:

- Belege Variablen $v_1, \ldots, v_n := 1$, m
- Prozessor (i, j):
  - Sei o.B.d.A $a_i < a_j \rightarrow v_i := 0$.
  - Ist $v_j = 1$, so setze $\max := a_j$.
- Ist $v_j = 1$, so schreibe $m := j$
- Index des Maximums findet sich in m

Aufwand: $O(1)$, $E(n, n(n-1)/2) = O(1/n)$

Algorithmus Maximumsuche II

Gegeben: : $a_1, \ldots, a_n$, $n \in \mathbb{N}$, CRCW-Maschine mit n Prozessoren, $n = 2^m$, $m \in \mathbb{N}$

Gesucht: $\max\{a_1, \ldots, a_n\}$

Ablauf:

- Setze $g := 2$ als Größe der zu bearbeitenden Gruppen, setze die Zahl der Kandidaten $k := n$
- Solange $k \neq 1$
  - Finde mit Maximumsuche I das Maximum in jeder Gruppe der Größe g.
  - Nun sind noch $k/2$ Kandidaten übrig. Setze also $k := k/2$. Setze $g := g^2$. Dann ist

    $$g \cdot \left(\frac{k}{g}\right)^2 < n$$

    Also genügt die gegebene Anzahl der Prozessoren noch immer, um in $O(1)$ die Maxima jeder Gruppe zu finden.

Aufwand: $O(\log\log n)$, $E(n, n) = O(1/\log\log n)$

Algorithmus Parallel Prefix

Gegeben: $a_1, \ldots, a_n$, $\circ$ ein beliebiger assoziativer Operator, eine EREW-Maschine mit n Prozessoren, $n = 2^m$, $m \in \mathbb{N}$

Gesucht: a_1,$a_1 \circ a_2$,...,$a_1 \circ \cdots \circ a_n$

Ablauf:

- $n = 2 \rightarrow$ trivial, ansonsten
- Setze $a_{2i} := a_{2i-1} \circ a_{2i}$, $i = 1 \ldots n/2$
- Benutze Parallel Prefix mit $\{a_{2i} \mid i = 1 \ldots n/2\}$
- Setze $a_{2i-1} := a_{2i-2} \circ a_{2i-1}$, $i = 2 \ldots n/2$

Aufwand: $O(\log n)$, $E(n,n) = O(1/\log n)$. Gesamtzahl der Schritte ist $O(n)$. Nach dem Satz von Brent kann nun ein Algorithmus gefunden werden, der ebenfalls in $O(\log n)$ Schritten und mit nur $O(n/\log n)$ Prozessoren zum Ziel kommt.

Algorithmus Rang von Listenelementen

Gegeben: $a_1, \ldots, a_n$ verkettete Liste (a_i sei Index des nächsten Elements), eine EREW-Maschine mit n Prozessoren

Gesucht: $r_1, \ldots, r_n$, wobei r_i =Abstand vom Listenende

Ablauf:

- Belege $n_1, \ldots, n_n$ als Index des am weitesten entfernten bekannten Nachfolgers
- Prozessor i mit $i = 1, \ldots, n$:
  - $n_i := a_i$, $r_i := 0$
  - Ist $n_i = NIL$, setze $r_i := 1$.
  - Setze $d := 1$ als Verdopplungszähler
  - Solange $r_i = 0$
    - Ist $r_{n_i} = 0$, so setze $n_i := n_{n_i}$, $d := 2 \cdot d$. Ansonsten setze $r_i := r_{n_i} + d$.

Aufwand: $O(\log n)$, $E(n,n) = O(1/\log n)$. Die von diesem Algorithmus verwandte Technik nennt sich die Verdoppelungsmethode.

Trick Verfahren des Euler'schen Weges

Man kann den obigen Rang-Algorithmus leicht für Bäume modifizieren, indem man eine DFS-artig verzeigerte Liste von Knoten bereithält.

8.2 Algorithmen für Netzwerke

Definition 8.6 Netzwerk

Ein Rechnernetzwerk wird dargestellt als ein Graph $G = (V, E)$, wobei die Knoten Prozessoren/Recheneinheiten und die Kanten Kommunikationsverbindungen darstellen.

Definition 8.7 Durchmesser eines Netzwerks

Der Durchmesser eines Netzwerks ist das Maximum über die Länge aller kürzesten Verbindungen zwischen zwei Knoten.

Bemerkung Beliebte Netzwerk-Topologien

Folgende Topologien finden breitere Anwendung:

- Kette, Kreis
- Raster (großer Durchmesser), Torus
- Baum (geringer Durchmesser, dafür mit Flaschenhals)
- Hyperwürfel

Algorithmus Odd-even transposition sort

Gegeben: $a_1, \ldots, a_n$ aus einer geordneten Menge, Kette von n Prozessoren

Gesucht: Sortierte Folge

Ablauf:

- Verteile a_i an Prozessor i als p_i $(i = 1, \ldots, n)$
- Mache n-mal mit Prozessor i $(i = 1, \ldots, n)$
  - Vergleiche p_i mit p_{i-1}, wenn falsche Reihenfolge, tausche
  - Vergleiche p_i mit p_{i+1}, wenn falsche Reihenfolge, tausche

Aufwand: $O(n)$

Algorithmus Paralleler Mergesort

Zu kompliziert, um nur verbal beschrieben zu werden. Grundidee: Wenn man zwei aufsteigend sortierte Felder a, b in gerade e_a, e_b und ungerade indizierte Hälfte o_a, o_b zerlegt, dann e_a und e_b zu e sowie o_a o_b zu o "merget", erhält man einen Gesamt-"Merge" dadurch, dass man jeweils indizierte Elemente aus o_{i+1} und e_i miteinander vergleicht.

Aufwand: $O(\log^2 n)$ bei $O(n \log^2 n)$ Prozessoren $\implies$ $E(\log^2 n, n \log^2 n) = O(1/\log^3 n)$

Algorithmus k-kleinstes Element

Gegeben: $a_1, \ldots, a_n$ aus einer angeordneten Menge, n Prozessoren mit allen Verbindungen, die für einen vollständigen binären Baum erforderlich sind, $k \in \{1, \ldots, n\}$

Gesucht: k-kleinstes Element

Ablauf:

- Ordne jedem Prozessor p_i ein Datenelement a_i fest zu $(i = 1, \ldots, n)$
- Wähle durch zufällige Aufwärtspropagation ein Pivotelement p, berücksichtige dabei Chancenverschiebung durch eliminierte Elemente
- Propagiere das gewählte Pivotelement abwärts
- Propagiere die Anzahl der Elemente, die kleiner sind, aufwärts
- Propagiere den Rang r des Pivot abwärts
- Ist $r < k$, eliminiere alle Elemente $a_i \leq p$, ansonsten alle $a_i > p$.

Aufwand: $O(\log n)$. Indem man in jedem Schritt eine neue Pivotwahl initiiert, kann man die Laufzeit auf $O(\log^2 n / \log \log n)$ senken.

Algorithmus Matrixmultiplikation

Gegeben: $A = ((a_{ij}))$, $B = ((b_{ij}))$, $A, B \in \mathbb{R}^{n \times n}$, ein Gitter-Netzwerk mit Wrap-Around (Torus)

Gesucht: $C = AB = ((c_{ij}))$

Ablauf:

- Belege für jeden Prozessor P_{ij} ein $a'_{ij} := a_{i,i+j}$ und ein $b'_{ij} := b_{i+j,j}$ sowie $c_{ij} = 0$. $(i = 1, \ldots, n)$
- Mache n-mal:
  - Setze für jeden Prozessor P_{ij}: $c_{ij} := c_{ij} + a'_{i,j} \cdot b'_{ij}$ $(i = 1, \ldots, n)$
  - Schiebe die $((a'_{ij}))$ eine Spalte nach links
  - Schiebe die $((b'_{ij}))$ eine Zeile nach oben

Aufwand: $O(n)$, $E(n, n^2) = O(1)$

Bemerkung Matrix-Vektor-Produkt

Analog kann mit einer Zeile von n-Prozessoren ein Matrix-Vektor-Produkt implementiert werden.

Algorithmus Minimale Editierdistanz

Gegeben: Zwei Zeichenketten $a_1, \ldots, a_n / b_1, \ldots, b_n$, c_i, c_s, c_d Kosten der Editierschritte *insert*, *substitute*, *delete*, eine Prozessorenkette mit $2n-1$ Prozessoren

Gesucht: Minimale Kostensumme einer Editiersequenz, die (a_i) in (b_i) umformt

Ablauf:

- Bereite zum rechtsseitigen Einschieben vor: $\leftarrow a_1, c_d, a_2, 2c_d, \ldots, a_n$
- Bereite zum linksseitigen Einschieben vor: $b_n, \ldots, 2c_i, b_2, c_i, b_1 \rightarrow$
- Setze $c_i := 0$ und $d_i := 0$ für jeden Prozessor $(i = 1, \ldots, 2n-1)$
- Für jeden Prozessor P_i führe $4n$-mal folgenden Schritt durch:
  - Kommen von links und rechts a_j,b_{n+i-j}, vergleiche diese, merke $d_i = 0$ (gleich) bzw. $d_i = c_s$ (ungleich) und gib a_j nach links, b_{n+i-j} nach rechts weiter
  - Kommen von links Editierkosten f, g , so setze $c_i := \min\{d_i, f, g\}$ und gib c_i nach links und rechts weiter
- Die Zahl, die auf beiden Seiten nach dem letzten Zeichen die Kette verlässt, gibt die minimalen Editierkosten an.

A GNU Free Documentation License

A “Transparent” copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not “Transparent” is called “Opaque”.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The “Title Page” means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, “Title Page” means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies of the Document numbering more than 100, and the Document’s license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission. B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five). C. State on the Title page the name of the publisher of the Modified Version, as the publisher. D. Preserve all the copyright notices of the Document. E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices. F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below. G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document’s license notice. H. Include an unaltered copy of this License. I. Preserve the section entitled ”History”, and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled ”History” in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence. J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the ”History” section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission. K. In any section entitled ”Acknowledgements” or ”Dedications”, preserve the section’s title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein. L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles. M. Delete any section entitled ”Endorsements”. Such a section may not be included in the Modified Version. N. Do not retitle any existing section as ”Endorsements” or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version’s license notice. These titles must be distinct from any other section titles.

You may add a section entitled ”Endorsements”, provided it contains nothing but endorsements of your Modified Version by various parties–for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already

includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled "History" in the various original documents, forming one section entitled "History"; likewise combine any sections entitled "Acknowledgements", and any sections entitled "Dedications". You must delete all sections entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an "aggregate", and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document's Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

Index

Sätze und Definitionen

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

Z