Algorithmen und Datenstrukturen

Vorlesung #13 – Fragestunde

Benjamin Blankertz

Lehrstuhl für Neurotechnologie, TU Berlin

benjamin.blankertz@tu-berlin.de

15 · Jul · 2020

Inhalt der heutigen Vorlesung

- Rekapitulation der Themen
- Organisatorische Punkte zu den schriftlichen Tests
- Hinweise zu Aufgaben
- Fragen
- Ergebnis der Lehrevaluation

Überblick über die Themen der AlgoDat Vorlesung

1 – 3 Java, Laufzeit, (indizierte) Priority Queue

4 Backtacking, Greedy-Algorithmen

5 Minimax-Verfahren, Alpha-Beta-Suche, Branch-and-Bound

6 Dynamisches Programmieren

7 Tiefensuche (DFS), Breitensuche (BFS), Topologisches Sortieren

8 Minimale Spannbäume: Prim, Kruskal (Union-Find)

9 Kürzeste Wege: Dijkstra, Bellman-Ford, DAG mit Topo. Sortierung

10 Heuristische Algorithmen, [Randomisierte Algorithmen]

11 Hashtabellen

12 [Flussgraphen, Approximative Algorithmen]

[grau] = nicht relevant für Tests im Sommersemester 2020

Algorithmenentwicklung basierend auf Baum der Teillösungen

Backtracking

- vollständige Suche in statischem BdT; Verbesserungen durch:
- dynamische Entscheidungsstruktur
- informierte Reihenfolge des Expandierens
- aussichtslose Äste abschneiden

Branch-and-Bound

- Äste abschneiden, die laut *Bound* zu keiner besseren Lösungen führen können

Greedy

- Suchbaum so beschneiden, dass nur ein Pfad übrig bleibt

Dyn. Programmieren

- Gleiche Teillösungen identifizieren, so dass eine exponentielle Baumstruktur auf eine Matrix (k-dim Array) reduziert werden kann

Algorithmenentwicklung als Varianten anderer Algorithmenarten

Heuristische Algorithmen

- Heuristik, die Problem-spezifische Information liefert
- wird im Rahmen anderer Algorithmenarten genutzt,
- z.B. um zu entscheiden, welcher Knoten als nächstes expandiert wird

Approximative Algorithmen

- sind oft Abwandlungen anderer Algorithmenarten, z.B.
- Dynamisches Programmieren mit einer skalierten Variable in der OPT-Funktion
- Lösung eines ähnlichen Problems (wie MST für TSP) nehmen und 'umbauen'

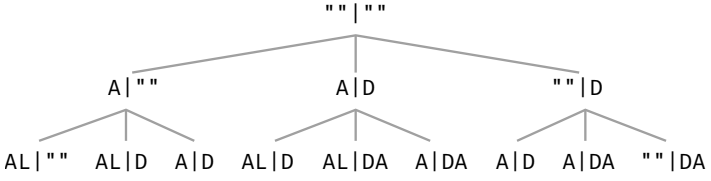
Randomisierte Algorithmen

- sind meist eigene Algorthmenarten
- Manchmal die einzige effiziente Lösung für ein Problem
- Oft einfache Algorithmen; wenig Speicherbedarf; lassen sich gut parallelisieren

Dynamisches Programmieren

- Dies Konzept greift, wenn in dem Suchbaum viele gleiche Teillösungen vorkommen.
- Dann versucht man die (exponentiell große) Baumstruktur auf ein k-dimensionales Feld bzw. ($k = 2$) eine Matrix zu reduzieren.
- Dafür werden die Lösungswerte als eine rekursive Funktion $\textsc{Opt}$ definiert.
- Die Matrix der Lösungswerte kann z.B. *bottom-up* von den Randwerten bestimmt werden.

Baum der Teillösungen



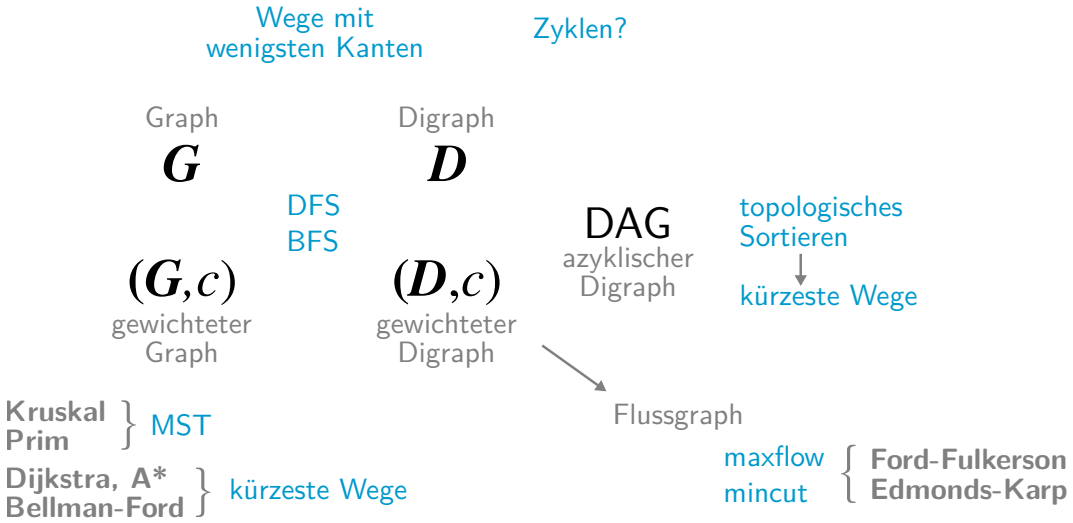
wächst exponentiell

Teillösungen in $\textsc{Opt}$ Matrix

""|""	""|D	""|DA
A|""	A|D	A|DA
AL|""	AL|D	AL|DA

wächst quadratisch

Graphen und ihre Algorithmen im Überblick



Graphenalgorithmen: Konzept für Kantenauswahl

- Viele Graphenalgorithmen explorieren die Knoten ausgehend vom einem Startknoten indem ein Baum über schrittweise ausgewählte Kanten aufgebaut wird.
  - Markierter Bereich: bisher besuchte Knoten
  - Kreuzende Kanten führen aus markiertem Bereich heraus
- Eine der kreuzenden Kanten wird gewählt.
- Wenn die Auswahl endgültig ist → Greedy-Algorithmus

Graphenalgorithmen

- Viele der besprochenen Graphenalgorithmen lassen sich als Auswahl einer der kreuzenden Kanten formulieren:
  - Kante vom Knoten, dessen Entdeckung am längsten her ist: **Breitensuche**
  - Kante vom Knoten, der als letztes entdeckt wurde: **Tiefensuche**
  - Kante, deren Kosten am geringsten sind: **Prim**
  - Kante zum Knoten mit kleinstem Abstand (gemäß Kantengewichten/ Kosten) vom Startknoten: **Dijkstra**
  - Kante zum Knoten mit kürzestem Weg in Kantendistanz (Anzahl der Kanten) vom Startknoten: **Bellman-Ford**
- Die ersten vier Algorithmen sind Greedy-Algorithmen. Bei Bellman-Ford kann die Kantenauswahl revidiert werden: kein Greedy-Algorithmus
- **Kruskals** Kantenauswahl ist *greedy*, es wird aber kein wachsenden Baum über kreuzende Kanten aufgebaut.

Organisatorische Punkte zu den schriftlichen Tests

- 1. Termin am Do 27.07.2023, Einlass um 12:15 Uhr, Beginn 12:30 Uhr
  - Räume H 0105, H 0104, HE 0101
- 2. Termin am Di 10.10.2023, Einlass um 08:15 Uhr, Beginn 08:30 Uhr
  - Räume: H 0105, HE 101

- Zuteilung auf die Räume und die Platznummer werden über ISIS bekannt gegeben.
- Voraussetzung: Anmeldung zur Prüfung. **Termine** werden gemäß **ISIS-Umfrage** zugeteilt. Wer nicht daran teilgenommen hat, wird zum Zweittermin erwartet.
- Bearbeitungszeit: 60 Minuten
- Hilfsmittel: ein handbeschriebenes DIN A4 Blatt
- Nach Mitteilung bis zum 24.07. an ‹`algodat@neuro.tu-berlin.de`› kann ein nicht-elektronisches Wörterbuch benutzt werden.
- **Auf Mitteilungen über ISIS zu den Tests achten!**

Organisatorische Punkte zu den schriftlichen Tests

Bitte tragen Sie sich auf ISIS für einen der beiden Termine ein!

Die Umfrage schließt am Mi, 12.07. um 23:59 Uhr.

Typische Aufgabenstellungen

1 – 3 Java, Laufzeit, (indizierte) Priority Queue

- **Java** kommt zu einem kleinen Teil vor.
- Mögliche Aufgabentypen (Beispiele):
  - Java Code gegeben, was ist die Ausgabe (auch Laufzeitfehler)?
  - Korrektur und/oder Vervollständigung von gegebenem Java Code
  - gegebenen Pseudocode als Java Code implementieren

- Wachstumsordnungen von **Laufzeit** bestimmen:
  - von kurzem Java- und Pseudocode, siehe auch Abschnitt 3.4 des Skriptes
  - von einfachen, als Text beschriebenen Algorithmen

Typische Aufgabenstellungen

4 Backtracking

- Elemente des Backtracking nennen und identifizieren
- Backtracking Algorithmus zu gegebenem Problem als Pseudocode entwerfen
  (ähnlich dem Permutationsbeispiel)

5 Minimax Verfahren, Alpha-Beta Suche

- Handsimulationen
- Alpha- und Beta Schnitte in einem Minimax Baum markieren

6 Dynamische Programmierung

- wesentliche Elemente der dynamischen Programmierung beschreiben
- *bottom-up* und *top-down* Ansatz beschreiben
- optimale Substruktur und überlappende Teilprobleme identifizieren
- rekursive $\textsc{Opt}$ Funktion zu gegebenem Problem aufstellen
- Werte *bottom-up* in Matrix gemäß einer Rekursionsgleichung eintragen
- beschreiben, wie Lösung aus gespeicherten Lösungswerten rekonstruiert wird

7 - 9 Graphenalgorithmen

(z. B. DFS, BFS, Topologische Sortierung, Prim, Kruskal, Dijkstra, Bellman-Ford)

- Handsimulationen aller Algorithmen
- Gegeben Graph mit markierten Kanten oder Knoten: Von welchem Algorithmus kann dies ein Zwischenstand des Ablaufes sein?
- Graph mit markiertem Bereich gegeben: welcher Knoten / welche Kante wird als nächstes von Algorithmus X gewählt?
- Laufzeiten von den besprochenen Algorithmen

  Dabei geht es um die Laufzeiten der in dem Skript besprochenen Varianten, nicht um ggf. effizientere Laufzeiten, die nur erwähnt wurden (also die Laufzeit von Dijkstra mit Fibonacci-Heap muss z. B. nicht bekannt sein).

Typische Aufgabenstellungen

12 Hashtabellen

- Gegebene Werte mit Hashfunktionen auf Hashadressen abbilden
- Schlüssel in Hashtabelle mit Kollisionsauflösung eintragen
- Löschen (*eager deletion*) eines Eintrages aus gegebener Hashtabelle (mit Sondierung)

Typische Aufgabenstellungen

- Grundlegende Begriffe definieren (z. B. Zusammenhang, Zyklus, ...)
- gegeben neue Definition, zeichne Beispielgraph
- Skizzierung eines Anwendungsproblems: Mit welchen Algorithmus aus der Vorlesung lässt es sich effizient lösen?
- Problemstellung (und Lösungsansatz) gegeben: Algorithmus entwerfen und analysieren

Fragen

- Zunächst werden die Fragen aus den Slido Q&A beantwortet.
- Thematisch passende Fragen können direkt live dazu gestellt werden.
- Fragen zu anderen Themen werden anschließend besprochen.