Informeller Überblick über die Vorlesung „Algorithmen und Datenstrukturen"

Kapitel 1 Einführung

- Was ist ein Algorithmus
- Laufzeitanalyse
- O-Notation, Rechnen mit O-Notation
- Datenstrukturen
  - Arrays (Direkter Zugriff auf Elemente, aber statisch)
  - Lineare Listen (verkettete Datenstruktur /dynamisch)
  - Baumstrukturen (hierarchisch, Grad, Knoten, Level, Höhe, Vater, Söhne, Brüder)
  - Implementierung von Datenstrukturen

Kapitel 2 Suchverfahren

Ziel: Suchen, Einfügen und Entfernen von Schlüsseln in O (log n)

- Allgemeine binäre Suchbäume
  - (-) Können zu linearer Liste entarten
- Vollständig ausgeglichene binäre Suchbäume
  - (+) garantieren Suchzeit von log n
  - (-) z.T. vollständige Reorganisation beim Einfügen/Entfernen nötig
- AVL-Bäume
  - Kompromisslösung $|h(T_r) - h(T_l)| \leq 1$ gilt für alle Knoten => Höhe O (log n)
  - Strukturverletzungen beim Einfügen/Entfernen werden durch (Doppel-) Rotationen behoben
- B-Bäume
  - Für große Datenmengen => Minimierung von Seitenzugriffen auf den Plattenspeicher
  - 1 Knoten des Baumes entspricht 1 Seite (bzw. mehreren Seiten) des Plattenspeichers
  - Besitzen Ordnung m
  - Behandlung von Overflow (Split) und Underflow (Ausgleichen/Verschmelzen mit dem rechten Bruder) zur Erhaltung der Baumeigenschaften.
- B*-Bäume
  - Mindestens 66% Speicherplatzausnutzung (im Gegensatz zu 50% beim B-Baum)
  - Beim Overflow Ausgleichen bzw. Verschmelzen mit dem rechten Bruder
  - Beim Underflow evtl. Betrachtung von 2 Brüdern (wenn beide zu 4/3 gefüllt sind)
- Optimale binäre Suchbäume
  - Level eines Schlüssels soll abhängen von dessen Suchhäufigkeit
  - Aufbau mittels Optimalitätskriterium u. Monotonieeigenschaft
  - Statische Datenstruktur, aber Grundlage für dynamische DS
- Allgemeine Hashverfahren
  - Hashverfahren ermitteln mithilfe von Hashfunktion die Adresse eines Schlüssels
  - Offene Hashverfahren (Direkte Verkettung), Geschlossene Hashverfahren (Direkte Verkettung mit Verschmelzen)

- Lineares Hashing
    - Anpassung der Größe der Hashtabelle an die Anzahl der gespeicherten Schlüssel
    - Dazu benötigt wird eine Folge von Hashfunktionen (Bereichs-/Splitbedingung)
    - Erweiterung wenn Belegungsfaktor überschritten wird.
- Lineares Hashing mit partieller Erweiterung (nicht in der Klausur am 14.07.12)
    - Höherer Belegungsfaktor der einzelnen Seiten durch Verdoppelung in mehreren Phasen (statt nur in einer wie bei einfachen linearem Hashing)
    - no = 2 bester Kompromiss

Kapitel 3 Sortierverfahren

- Sortiervervahren (allgemein)
    - Herstellen einer Sortierung => Existenz einer Ordnung
    - Unterscheidung interne / externe Verfahren
    - Stabilität manchmal Wünschenswert
    - Interessant v.a. Laufzeitverhalten (Anzahl Vergleiche / Zuweisungsoperationen)
- Einfache Sortierverfahren (O (n^2))
    - Selectionsort („jeweils auswählen des kleinsten verbleibenden Elements")
    - Bubblesort („Vertauschen von benachbarten Elementen wenn linkes>rechtes")
    - Insertionsort („ Einfügen des momentan betrachteten Elements in das sortierte Teilarray")
    - ~~Shellsort (Insertionsort auf Unterarrays) (O(n1·25))~~
- Divide-and-Conquer-Verfahren
    - Mergesort („Zusammenmischen von jeweils 2 Stapeln") (O(n*log n))
    - Quicksort („kleine Schlüssel nach links, große Schlüssel nach rechts dann divide") $O(n^2)$
- Sortieren mit Bäumen (Heapsort) (O(n*log n))
    - 1. Phase: Aufbau des Heaps Bottom-Up. Heapeigenschaft: Der Schlüssel in einem Knoten muss größer gleich den Schlüsseln in den Söhnen sein (O(n))
    - 2 Phase: Sortierschritt sukzessive Entnahme des Maximums —> Ersetzung durch letztes Element —> Wiederherstellen der Heapeigenschaft (O(n*log n))
- ~~Sortieren in linearer Zeit (O(n))~~
  ~~Voraussetzungen: Wertebereich der Schlüssel bekannt und relativ klein und auch andere Operationen als Vergleiche möglich~~
    - ~~Bucketsort („Schlüssel in Buckets mit der gleichen Adresse —> Sortierung der Schlüssel durch durchlaufen der Buckets")~~
    - ~~Radixsort (Schlüssel werden durch eine Zeichenfolge über einem Alphabet mit gleicher Länge dargestellt -> fürjeden „Buchstaben" wird vom letzten Buchstaben ausgehend im Prinzip Bucketsort durchgeführt. Jede Phase benutzt das Ergebnis der vorherigen Phase als Eingabe)~~

Kapitel 4 Graphalgorithmen

Graph (G(V,E)) = Menge von Knoten(V) und Kanten(E)

- Speicherdarstellung von Graphen
  - Adjazenzmatrix (Eintrag für jede mögliche Kante)
  - Adjazenzliste (Enthält alle tatsächlich verbundenen Knoten zu jedem Knoten)
- Graphendurchläufe
  - Tiefendurchlauf (,,Durchlaufen bis zum Ende, dann zurück zur letzten Verzweigung")
  - Breitendurchlauf (,,AIIe adjazenten Knoten in Schlange, danach Abarbeitung ebenso")
- Algorithmen
  - Dijkstra: Single Source Shortest Path (O(n2))
  - Floyd bzw. Warshall: All Pairs Shortest Path (O(n3))
  - Kruskal: Minimaler Spannbaum (O(| E|*log|E|))

Kapitel 5 Algorithmische Methoden und Techniken

- Erschöpfende Suche (,,Berechne alle Möglichkeiten und gib die beste als Ergebnis zurück")
- Lokal optimierende Berechnung (,,Heuristischer Ansatz soll eine gute Lösung liefern")
- Backtracking (entspricht Tiefendurchlauf durch den Lösungsbaum): ,,Breche die Berechnung ab, wenn bereits eine bessere Lösung existiert"
- Branch-and—Bound (entspricht Breitendurchlauf): ,,Versuche durch aufwendige Berechnungen Kosten abzuschätzen und Durchlauf von Teilbäumen zu vermeiden")
- Divide-and-Conquer: Zerlegung von großen Problemen in Teilprobleme (rekursive Behandlung der Teilprobleme). Wichtig: gleichmäßige Zerlegung, so dass die Problemgröße effektiv kleiner wird.
- Dynamische Programmierung: Lösung von Basisproblemen -> Zusammensetzen der Lösungen (siehe optimaler Binärer Suchbaum)