

1 O-Notation

$$O(c) < O(\log \log n) < O(\log n) < O((\log n)^c) < O(\sqrt[c]{n}) < O(n \log n) = O(\log n!) < O(n^c) < O(c^n) < O(n!)$$

mit $c > 1$ als positive Konstante.

Tipps:

- $\binom{n}{c} \in \Theta(n^c)$
- n^c ist polynomial
- $n! \approx n^n$

2 Hashing

Eine Hashingfunktion sollte

- möglichst schnell zu berechnen sein
- Datenstze möglichst gleichmig auf den Speicherbereich verteilen
- Hufungen (cluster) fast gleicher Schlssel aufbrechen

2.1 Sondierung

primary clustering: lang belegte Teilstcke haben also eine grssere Tendenz zu wachsen (linear)

secondary clustering: Synonyme durchlaufen die selbe Sondierungsfolge (linear und quadratisch)

2.1.1 Linear

$$h(k), h(k) - 1, h(k) - 2, \dots, 0, m - 1, \dots, h(k) + 1$$

2.1.2 Quadratisch

$$h(k), h(k) + 1, h(k) - 1, h(k) + 4, h(k) - 4, h(k) + 9, h(k) - 9$$

2.1.3 Double Hashing

$$h(k), h(k) - h'(k), h(k) - 2h'(k), \dots, h(k) - (m - 1)h'(k)$$

Genau so effizient wie uniformes Sondieren.

3 Datenstrukturen

3.1 Union-Find Struktur

Vereinigung nach Grsse/Hhe: Man macht die Wurzel des Baumes mit kleinerer Gre (bzw. geringerer Hhe) zum direkten weiteren Sohn des Baumes mit der greren Gre (bzw. Hhe).

Die Grsse/Hhe eines Teilbaums wird in der Wurzel gespeichert.

Pfadverkzung: Bei der Find-Operation werden die durchlaufenen Knoten direkt an die Wurzel gehngt. Die Operation wird doppelt so teuer.

3.2 Suchbäume

- preorder:** Hauptreihenfolge
Wurzel, linker Teilbaum, rechter Teilbaum
Rekonstruktion: In selber Reihenfolge in leeren Suchbaum einfügen
- postorder:** Nebenreihenfolge
linker Teilbaum, rechter Teilbaum, Wurzel
Rekonstruktion: In umgekehrter Reihenfolge in leeren Suchbaum einfügen
- inorder:** symmetrische Reihenfolge
linker Teilbaum, Wurzel, rechter Teilbaum

3.2.1 AVL-Baum

Für jeden Knoten p des Baumes gilt, dass sich die Höhe des linken Teilbaumes von der Höhe des rechten Teilbaumes von p höchstens um 1 unterscheidet.

Das Einfügen eines neuen Elements führt höchstens zu einer Rotation.

3.2.2 Splay-Baum

Suchbaum, der die Move-to-Root Operation unterstützt. Ein Knoten wird zur Wurzel hochgeschoben, in dem Rotationen wie beim AVL-Baum durchgeführt werden.

3.2.3 B-Baum

Für ein B-Baum der Ordnung m gilt:

- Alle Blätter haben die gleiche Tiefe
- Jeder Knoten mit Ausnahme der Wurzel und der Blätter hat wenigstens $\lceil \frac{m}{2} \rceil$ Söhne
- Die Wurzel hat wenigstens 2 Söhne
- Jeder Knoten hat höchstens m Söhne
- Jeder Knoten mit i Söhnen hat $i - 1$ Schlüssel
- Knoten dürfen höchstens $m - 1$ Schlüssel speichern

3.2.4 Segment-Baum

Aufspiessanfrage: Es werden alle Intervalle ausgegeben, die auf dem Suchpfad zum Elementarsegment gehören

3.2.5 Heap

Versickern: Max/Min mit letztem Element austauschen, Blatt entfernen, Wurzel so lang mit Max der Kinder austauschen, bis es grösser als diese ist (oder Blatt)

4 Sortieralgorithmen

Name	Best Case	Average Case	Worst Case	in-situ	stabil
Selectionsort	$O(n^2)$	$O(n^2)$	$O(n^2)$	ja	nein
Bubblesort / Insertionsort	$O(n)$	$O(n^2)$	$O(n^2)$	ja	ja
Heapsort	$O(n \log n)$	$O(n \log n)$	$O(n^2)$	ja	nein
Mergesort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(n)$	ja
Quicksort	$O(n \log n)$	$O(n \log n)$	$O(n^2)$	$O(\log n)$	nein

4.1 Selectionsort

1. Suche Minimum der letzten n Elemente
2. Tausche Minimum mit dem ersten Element
3. Fahre fort mit den $n - 1$ letzten Elementen

4.2 Insertionsort

1. Nächstes Element betrachten
2. Position für betrachtetes Element in sortierter Teilfolge (prefix) finden und einfügen

4.3 Quicksort

1. Pivot auswählen
2. Teilfolgen erstellen, von links und rechts in die Mitte iterieren, Elemente mit einander austauschen falls nötig
3. Pivot mit mittlerem Element tauschen

4.4 Mergesort

1. Array halbieren
2. Mergesort rekursiv auf beide Teilfolgen aufrufen
3. Sortierte Teilfolgen verschmelzen

5 Graphenalgorithmien

5.1 Tiefen- und Breitensuche

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

5.2 Matching

Zuordnung Teilmenge an Kanten, sodass keine zwei Kanten denselben Endknoten haben

Größe Anzahl Kanten in der Zuordnung, also $|Z|$

5.3 Minimaler Spannbaum

5.3.1 Kruskal

Laufzeit: $O(|E| \log |V|)$

Die billigste Kante wird betrachtet, verbindet sie zwei Teilmengen von Knoten, so wird sie hinzugefügt (Union-Find-Struktur)

5.3.2 Dijkstra

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

Tiefensuche, es wird immer die billigste Kante ausgewählt

5.4 Topologische Sortierung

Für G gibt es eine topologische Sortierung $\Leftrightarrow G$ ist zyklensfrei

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

1. Wähle ein Knoten aus
2. Besuch die Kinder, passe deren Ordnung an
3. Wähle nächsten nicht besuchten Knoten

5.5 Krzeste Wege

5.5.1 One-to-One (Dijkstra)

Bemerkung: Es sind nur positive Kanten erlaubt. Für die Priority Queue kann eine Fibonacci-Heap gebraucht werden.

Laufzeit: $|E| + |V| \log |V|$

1. Wähle Anfangsknoten aus
2. Wähle nächsten Knoten aus, der minimale Distanz zu Anfangsknoten hat
3. Passe neue Distanz aller Randknoten an
4. Wähle nächsten Knoten

5.5.2 One-to-All (Bellman/Ford)

Bemerkung: Funktioniert auch bei Digraphen mit negativen Pfaden.

Laufzeit: $O(|E| \cdot |V|)$

Funktioniert ähnlich wie nach Dijkstra. Knoten werden jedoch nicht endgültig gewählt, sondern mehrmals besucht. Beendet ist der Algorithmus, wenn sich kein Knoten mehr updaten lässt.

5.5.3 All-to-All

Laufzeit: $O(|V| \cdot (|E| + |V| \log |V|))$

1. Transformiere allgemeinen Graphen (mit möglichen negativen Kanten) zu Distanzgraphen
2. Wende Bellman/Ford auf Graphen an

5.6 Flussnetzwerke

5.7 Ford/Fulkerson

Laufzeit: $O(|E|f)$, mit dem minimalen Schnitt f

5.8 Edmonds/Karp

Angepasster Ford/Fulkerson Algorithmus.

The algorithm is identical to the FordFulkerson algorithm, except that the search order when finding the augmenting path is defined. The path found must be a shortest path that has available capacity. This can be found by a breadth-first search, as we let edges have unit length.

Laufzeit: $O(|V| \cdot |E|^2)$

1. Suche kürzeste Weg von Quelle zur Senke
2. Benutze diesen Pfad mit maximal möglicher Kapazität
3. Wiederhole so lang, bis es keinen Pfad mehr zur Senke gibt

5.9 Dinic

hnlich wie Edmonds/Karp

Laufzeit: $O(|V|^2 \cdot |E|)$

6 Geometrische Algorithmen

6.1 Scanline

Sichtbarkeit: AVL-Tree