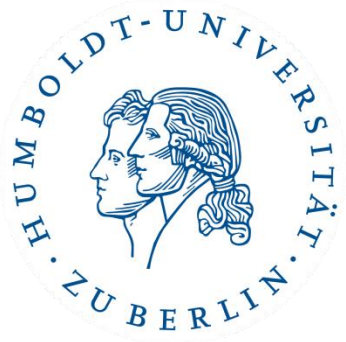
Übung Algorithmen und Datenstrukturen



Sommersemester 2017

Marc Bux, Humboldt-Universität zu Berlin

Agenda

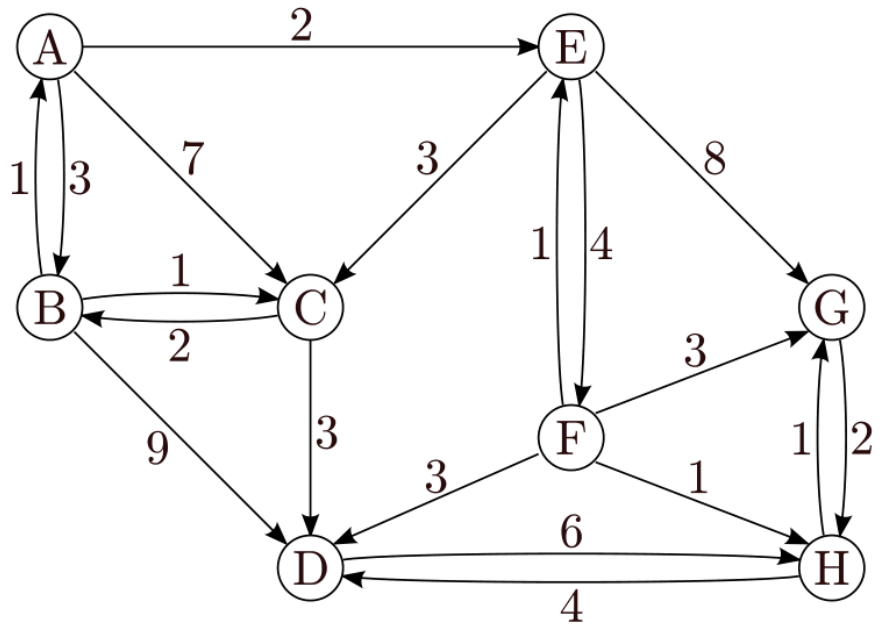
1. Dijkstra-Algorithmus
2. Heaps
3. Hashtabellen
4. Lehrevaluation

Agenda

1. Dijkstra-Algorithmus
2. Heaps
3. Hashtabellen
4. Lehrevaluation

Dijkstra-Algorithmus

Gegeben sei ein gerichteter Graph mit 8 Knoten:



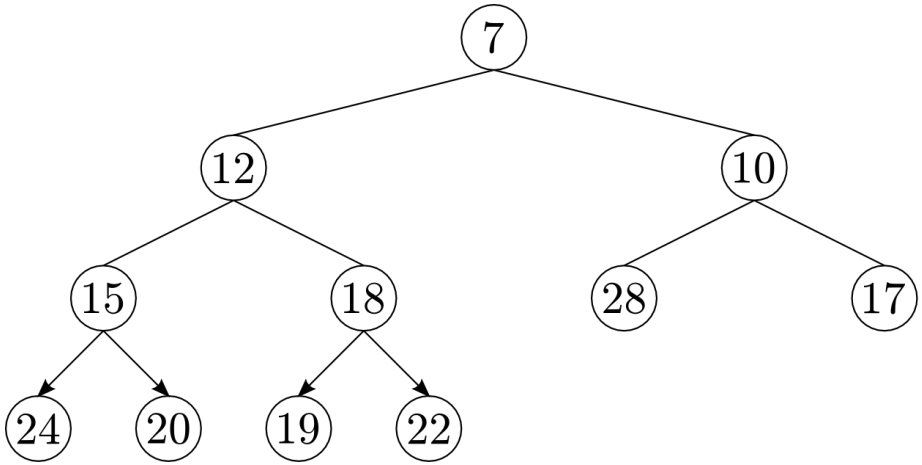
Bestimmen Sie den Abstand aller Knoten zum Knoten A. Führen Sie dafür den Dijkstra-Algorithmus aus der Vorlesung mit dem Startknoten A aus. Geben Sie die Zwischenergebnisse (d.h., die derzeit ermittelten Distanzen aller Knoten zu A) nach jedem Schritt an. Markieren Sie den aktuell betrachteten Knoten in jedem Schritt durch Umkreisen. Geben Sie außerdem den Inhalt der Priority Queue an, in der die als nächstes zu betrachtenden Knoten gespeichert werden.

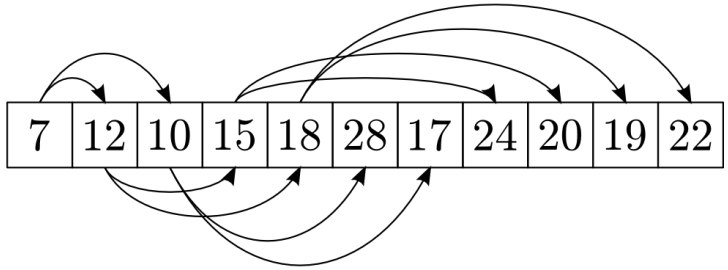
Agenda

1. Dijkstra-Algorithmus
2. Heaps
3. Hashtabellen
4. Lehrevaluation

Heaps

- Ein Heap ist eine auf Bäumen basierende Datenstruktur zum Speichern von Elementen, über deren Schlüssel eine totale Ordnung definiert ist
  - Form Constraint: Der Baum ist fast vollständig
    1. Alle Ebenen außer der untersten müssen vollständig gefüllt sein
    2. In der letzten Ebene werden Elemente von links nach rechts aufgefüllt
  - Heap Constraint: Bei einem Min-Heap (Max-Heap) sind die Schlüssel jedes Knotens kleiner (größer) als die Schlüssel seiner Kinder
- Heaps lassen sich als heapgeordnete Arrays repräsentieren





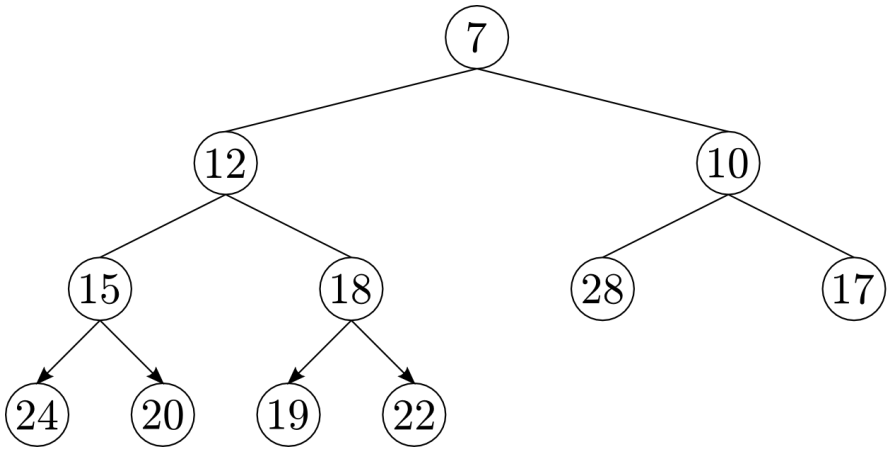
Gängige Operationen, Komplexität

- Element einfügen:
  - neues Element als „letzten“ Knoten einfügen
  - neues Element „hochsickern“ lassen (”sift up”)
- Minimum löschen:
  - Wurzel durch „letzten“ Knoten ersetzen
  - Wurzel „heruntersickern“ lassen (”sift down”)

Datenstruktur	Einfügen	Suchen	Besonderheit	Java-Interface	Java-Klassen (Auswahl)
Array / String	N/A	$O(n)$	Indexbasierter Zugriff in $O(1)$		[], String
Liste	$O(1)$	$O(n)$		List	LinkedList, ArrayList
Stack	$O(1)$	$O(n)$	Zugriff auf zuletzt eingefügtes Element in O(1)		Stack
Queue	$O(1)$	$O(n)$	Zugriff auf zuerst eingefügtes Element in O(1)	Queue	LinkedList
Binärer Heap	$O(\log n)$	$O(n)$	Zugriff auf Min / Max in O(1)		PriorityQueue

Aufgaben zu Min-Heaps

1. Geben Sie alle möglichen Min-Heaps zur Speicherung der Zahlen 1, 2, 3, 4 und 5 an.
2. Es sei der folgende Min-Heap gegeben:



Wie sieht der Heap nach Anwendung der folgenden Operationen (in der gegebenen Reihenfolge) aus?

- `deleteMin()`
- `deleteMin()`
- `add(14)`
- `add(8)`

Aufgaben zu Max-Heaps

In der Vorlesung haben Sie bereits *binäre Min-Heaps* kennengelernt. In dieser Aufgabe betrachten wir außerdem *binäre Max-Heaps*. Im Gegensatz zu einem Min-Heap hat ein Max-Heap die Eigenschaft, dass der Wert jedes Knotens größer als der Wert seiner Kinder ist.

1. Wie kann eine Queue mit Hilfe eines Min-Heaps realisiert werden? Beschreiben Sie dafür, wie die Methoden `Queue.enqueue(element)` und `Queue.dequeue()` nur unter Verwendung der in Aufgabe 3 auf Min-Heaps eingeführten Methoden realisiert werden können. Beide Methoden `Queue.enqueue(element)` und `Queue.dequeue()` sollen die Komplexität $\mathcal{O}(\log n)$ haben.

2. Wie kann ein Max-Heap für das Speichern von ganzen Zahlen nur mit Hilfe eines Min-Heaps realisiert werden? Beschreiben Sie dafür, wie die Methoden `MaxHeap.deleteMax()` und `MaxHeap.add(element)` nur unter Verwendung der in Aufgabe 3 auf Min-Heaps eingeführten Methoden realisiert werden können. Der Max-Heap soll das Maximum in $\mathcal{O}(\log n)$-Operationen löschen und extrahieren können und $\mathcal{O}(\log n)$-Operationen für das Einfügen eines neuen Elements brauchen.

3. Wie kann eine Datenstruktur unter Verwendung zweier Heaps (also zweier Min-Heaps, zweier Max-Heaps oder jeweils eines Min- und Max-Heaps) realisiert werden, so dass der Median der eingefügten Elemente stets in Laufzeit $\mathcal{O}(1)$ ermittelt werden kann? Beschreiben Sie dafür die beiden Methoden `printMedian()`, zum Ausgeben des Medians in $\mathcal{O}(1)$ Operationen, und `add(element)`, zum Einfügen eines neuen Elements in $\mathcal{O}(\log n)$ Operationen.

Agenda

1. Dijkstra-Algorithmus
2. Heaps
3. Hashtabellen
4. Lehrevaluation

Hashtabellen

Beim Hashing bestimmt die Sondierungsreihenfolge für jeden Schlüssel, in welcher Reihenfolge die Einträge der Hashtabelle auf einen freien Platz hin durchsucht werden.
Gegeben sei eine Hashtabelle mit 11 Feldern für Einträge, die durch $0, \ldots, 10$ indiziert sind, und die Hashfunktion $h(k) = k \bmod 11$.
Führen Sie für die folgenden Hashverfahren einen Schreibtischtest durch, indem Sie jeweils die Elemente 54, 15, 27, 76, 22, 5 und 14 in dieser Reihenfolge in eine leere Tabelle einfügen und diese nach jeder Einfügeoperation ausgeben.

a) **Offenes Hashing mit linearem Sondieren**
Bei Kollisionen wird hier das einzufügende Element an der nächsten freien Stelle links vom berechneten Hashwert eingefügt. Das heißt, die Sondierungsreihenfolge (die Reihenfolge, in der die Plätze im Array durchgegangen werden, bis erstmals ein freier Platz angetroffen wird) ist gegeben durch $s(k, i) = (h(k) - i) \bmod 11$ für $i = 0, \ldots, 10$.

b) **Doppeltes Hashing**
Das doppelte Hashing ist ein offenes Hashing, bei dem die Sondierungsreihenfolge von einer zweiten Hashfunktion $h'(k) = 1 + (k \bmod 7)$ abhängt. Die Position für das i-te Sondieren ist bestimmt durch die Funktion $s(k, i) = (h(k) - i \cdot h'(k)) \bmod 11$ für $i = 0, \ldots, 10$.

Datenstrukturen

Datenstruktur	Einfügen	Suchen	Besonderheit	Java-Interface	Java-Klassen (Auswahl)
Array / String	N/A	$O(n)$	Indexbasierter Zugriff in $O(1)$		[], String
Liste	$O(1)$	$O(n)$		List	LinkedList, ArrayList
Stack	$O(1)$	$O(n)$	Zugriff auf zuletzt eingefügtes Element in O(1)		Stack
Queue	$O(1)$	$O(n)$	Zugriff auf zuerst eingefügtes Element in O(1)	Queue	LinkedList
Binärer Heap	$O(\log n)$	$O(n)$	Zugriff auf Min / Max in O(1)		PriorityQueue
Hash-Tabelle	$O(1)$*	$O(1)$*	Konstante Laufzeit nur im Average Case		HashMap, HashSet

Agenda

1. Dijkstra-Algorithmus
2. Heaps
3. Hashtabellen
4. Lehrevaluation

Ausblick

- nächste Woche:
  - Aufgabenblatt 4 durchrechnen
- zu nächster Woche:
  - mit Aufgabenblatt 5 auseinandersetzen
  - falls noch nicht vorgerechnet: zum Vorrechnen eintragen
    Montag: https://dudle.inf.tu-dresden.de/algodat24/
    Dienstag: https://dudle.inf.tu-dresden.de/algodat34/
    (first-come-first-served)