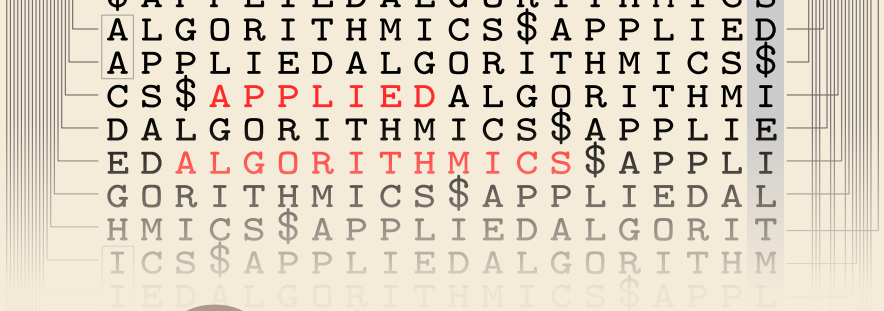


2 Fundamental Data Structures

04 February 2020

Sebastian Wild

version 2020-02-04 14:50

Outline

2 Fundamental Data Structures

2.1 Stacks & Queues

2.2 Resizable Arrays

2.3 Priority Queues

2.4 Binary Search Trees

2.1 Stacks & Queues

ADT = abstract data type

- list of supported operations
  - what should happen
  - not how to do this

not how data is stored

VS

data structures

- how data is stored in memory
- algorithms to work on data

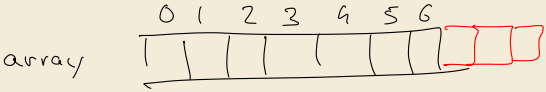
ex: stack

pop() → removes topmost element

push(v) → adds v to top of stack

pop pop push(1)

2.2 Resizable Arrays



- arrays have fixed size
  ↳ if we need more space, allocate new array & copy old data
- doubling arrays: when array is full double its size
- if array becomes too empty (deletions)
  ↳ if $\leq \frac{1}{4}$ full ⇝ halve size

⇝ space $\Theta(n)$ n = # elements stored

Java Generics

Stack<String>

Stack<Integers>

implement stack with type parameter — ONCE

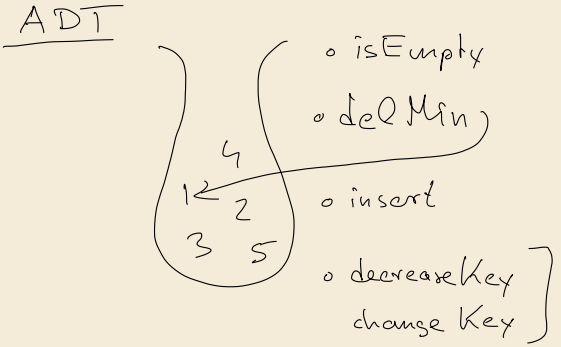
use stack with many different types

Iterators

ADT abstracts linear scan over collection of items

- hasNext()
- next move ahead & return element

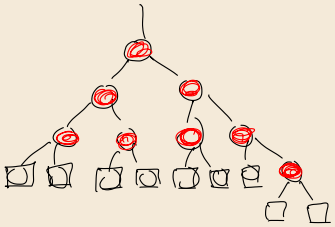
2.3 Priority Queues



Heaps

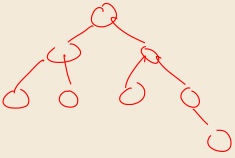
- binary tree

extended binary trees



⟷

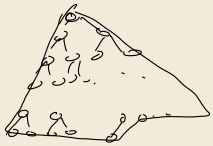
binary trees



every node has
0 or 2 children

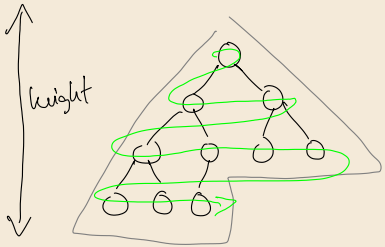
nodes have
left / right children
(both can be missing)

- complete binary tree + flush-left last level — lowest level as far to the left as possible



——— "heap-shaped trees"

Why heap-shaped trees?



- minimal height among binary trees with n nodes

n nodes → 1 heap-shape

⇒ easy

can use <u>array</u>: store nodes in level-order starting at 1

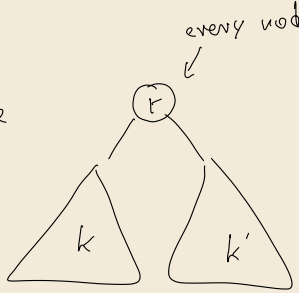
→ find parent of node k at $\lfloor \frac{k}{2} \rfloor$

left child at $2k$

right " $2k+1$

heap-order

entire subtree version



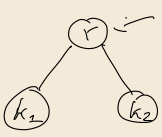
$k \geq r$

$k' \geq r$

all keys in subtrees are $\leq$ key in root

$\Rightarrow$ root stores minimum

children-version

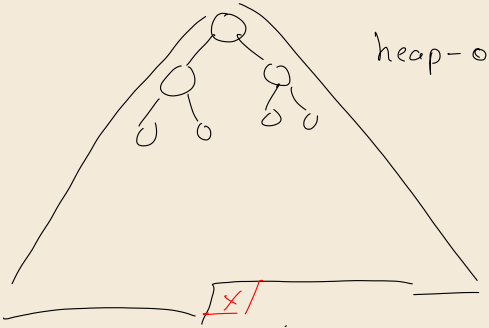


$k_1 \geq r$

$k_2 \geq r$

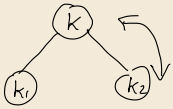
Insertion

insert new key x



heap-ordered

1. store x in lowest level, next free position
2. repair heap-order

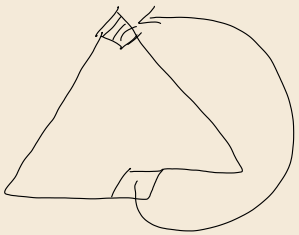


$k_2 < k$ violation

→ swap them!

repeat up the tree

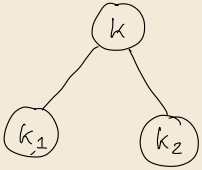
delete Min



(1) delete root (and return its key)

(2) move "last" element to root
└ rightmost on lowest level

(3) repair heap-order



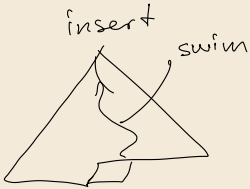
$k > k_1$ and/or $k > k_2$

(a) find $\min\{k_1, k_2\}$

(b) swap k with smaller child

continue along the changed child link

Analysis



worst case in both cases: follow one path

$\Rightarrow$ cost = # levels = height of tree

$\sim \lg n$