



*nog toevoegen: uitwerkingen van 5,6, 11,12*

## Uitwerkingen

1. We gebruiken alleen dat een heap een complete binaire boom is; de (max- of min-)heap eigenschap is hier niet relevant.

Bij preorder: 1, 2, 4, 8, 9, 5, 10, 11, 3, 6, 12, 13, 7, 14, 15.

Bij inorder: 8, 4, 9, 2, 10, 5, 11, 1, 12, 6, 13, 3, 14, 7, 15.

Bij postorder: 8, 9, 4, 10, 11, 5, 2, 12, 13, 6, 14, 15, 7, 3, 1.

2. Een voorbeeld waaruit blijkt dat een preorder traversal van een min-heap niet perse de keys in niet-dalende volgorde geeft:



3. Een voorbeeld waaruit blijkt dat een postorder traversal van een min-heap niet perse de keys in niet-stijgende volgorde geeft:



4. Een voorbeeld waaruit blijkt dat de bottum-up heap constructie een andere min-heap levert dan wanneer we een-voor-een toevoegen:

[1, 3, 2] geeft met de bottum-up heap constructie de min-heap



en met een-voor-een toevoegen de min-heap



5. We definiëren en bekijken de 3-min-heaps. (Voor de terminologie: een binaire boom is een 2-boom.)

- (a) Een 3-boom is een boom waarbij elke interne knoop 0, 1, 2, of 3 kinderen heeft.
  - (b) Een complete 3-boom is een 3-boom waarin de bovenste lagen compleet gevuld zijn met interne knopen, en de onderste laag is van links af vol tot een zeker punt, en daarna leeg.
  - (c) De hoogte van een complete 3-boom is in  $\mathcal{O}(\log 3(n))$ .
  - (d) De 3-min-heap eigenschap: voor elke (van de maximaal drie interne) opvolger  $w$  van een interne knoop  $v$  geldt:  $\text{key}(v) \leq \text{key}(w)$ .
  - (e) Beschrijf de procedure `3downMinHeap`.  
Nog toevoegen.
  - (f) Beschrijf de procedure `3upMinHeap`.  
Nog toevoegen.
  - (g) Extra.
6. Een worst-case input rijtje voor  $n$  toevoegingen aan een min-heap is van de vorm  $n, n - 1, n - 2, \dots 1$ .
7. De priority queue weergegeven als gesorteerde lijst, met rechts vooraan:

(5, 1)  
 (5, a), (4, b)  
 (7, i), (5, a), (4, b)  
 (7, i), (5, a), (4, b), (1, d)  
 (7, i), (5, a), (4, b)  
 (7, i), (5, a), (4, b), (3, j)  
 (7, i), (6, l), (5, a), (4, b), (3, j)  
 (7, i), (6, l), (5, a), (4, b)  
 (7, i), (6, l), (5, a)  
 (8, g), (7, i), (6, l), (5, a)  
 (8, g), (7, i), (6, l)

8. We hebben een priority queue  $Q$  met de operaties uit het ADT voor priority queues: `insertItem(k, e)`, `removeMin`, `minElement`, `minKey`, `size()`, `isEmpty()`. Daarnaast hebben we een (globale) integer variabele  $i$ .
- We implementeren hiermee het ADT van queues. Het idee is dat we  $i$  gaan gebruiken om een key aan een element te geven; die key geeft aan wat de rangorde in 'toevoegen ooit' van dat element is.

**Algorithm enqueue( $e$ ):**

  insertItem( $i, e$ )

$i := i + 1$

**Algorithm dequeue():**

$e := \text{removeMin}()$

**return**  $e$

**Algorithm front():**

**return** minElement()

De methoden `size()` en `isEmpty()` worden gewoon overgenomen uit het ADT voor priority queues.

9. We passen selection-sort toe:

```
[22, 15, 36, 44, 10, 3, 9, 13, 29, 25]
[3, 15, 36, 44, 10, 22, 9, 13, 29, 25]
[3, 9, 36, 44, 10, 22, 15, 13, 29, 25]
[3, 9, 10, 44, 26, 22, 15, 13, 29, 25]
[3, 9, 10, 13, 26, 22, 15, 44, 29, 25]
[3, 9, 10, 13, 15, 22, 26, 44, 29, 25]
[3, 9, 10, 13, 15, 22, 26, 44, 29, 25]
[3, 9, 10, 13, 15, 22, 25, 44, 29, 26]
[3, 9, 10, 13, 15, 22, 25, 26, 29, 44]
[3, 9, 10, 13, 15, 22, 25, 26, 29, 44]
[3, 9, 10, 13, 15, 22, 25, 26, 29, 44]
```

10. We passen insertion-sort toe:

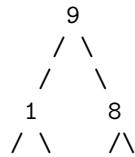
```
[22, 15, 36, 44, 10, 3, 9, 13, 29, 25]
[22, 15, 36, 44, 10, 3, 9, 13, 29, 25]
[15, 22, 36, 44, 10, 3, 9, 13, 29, 25]
[15, 22, 36, 44, 10, 3, 9, 13, 29, 25]
[15, 22, 36, 44, 10, 3, 9, 13, 29, 25]
[10, 15, 22, 36, 44, 3, 9, 13, 29, 25]
[3, 10, 15, 22, 36, 44, 9, 13, 29, 25]
[3, 9, 10, 15, 22, 36, 44, 13, 29, 25]
[3, 9, 10, 13, 15, 22, 36, 44, 29, 25]
[3, 9, 10, 13, 15, 22, 29, 36, 44, 25]
[3, 9, 10, 13, 15, 22, 25, 29, 36, 44]
```

Pas stap voor stap insertion-sort toe op dezelfde input-rij.

11. Een voorbeeld van een worst-case rijtje met  $n$  elementen voor insertion sort:  $n, n - 1, n - 2, \dots, 1$ .
12. We bekijken de input-rij

[3, 9, 8, 6, 2, 4, 1]

- (a) Eerst bouwen we een min-heap:



met max-heap

- (b) Eerst bouwen we een min-heap met de bottum-up constructie. Dat levert:
13. De tijdscomplexiteit is in  $\mathcal{O}(n)$  omdat we voor elk van de  $n$  elementen de methode ‘toevoegen’ van constante tijdscomplexiteit moeten uitvoeren.
14. We beginnen met een initieel lege hash-table van grootte 11, en we gebruiken de hash functie  $h(i) = (3i + 5) \bmod 11$ . Voeg (in deze volgorde) de volgende keys toe:

12, 44, 13, 88, 23, 94, 11, 39, 20, 16, 5

waarbij collision wordt opgelost met chaining.

$k$	12	44	13	88	23	94	11	39	20	16	5
$h(k)$	8	5	0	5	8	1	5	1	10	9	9

13	[94,39]					[44,88,11]				[12,23]		[16,5]		20
----	---------	--	--	--	--	------------	--	--	--	---------	--	--------	--	----

15. Wat krijg je als resultaat van de vorige opgave, met als enige verschil dat nu collision wordt opgelost met linear probing?

13	94	39	16	5	44	88	11	12	23	20
----	----	----	----	---	----	----	----	----	----	----

16. We bekijken hashing met linear probing. Een mogelijkheid (zie de slides) is om verwijderde elementen te representeren met een speciale marker (bijvoorbeeld ‘available’). Het kan ook anders: je kan ook de array herarrangeren, zo dat het lijkt alsof het verwijderde element er nooit in heeft bestaan. Hoe zie de operatie voor verwijderen er dan uit?

17. Het is niet handig omdat je dan je volgorde kwijtraakt.

18. (Dit zijn ongeveer opgaven R-2.19 – R-2.22.)

We beginnen met een lege hash table ter grootte 11 en voegen toe:

12, 44, 13, 88, 23, 94, 11, 39, 20, 16, 5

(in deze volgorde), met hash functie  $h(k) = (2 \cdot k + 5) \bmod 11$ . Om te beginnen de hash functie:

$k$	12	44	13	88	23	94	11	39	20	16	5
$h(k)$	7	5	9	5	7	6	5	6	1	4	4

Nu de hashing, met verschillende manier om met collision om te gaan.

(a) Met chaining:

	20			[16,5]	[44,88,11]	[94,39]	[12,23]		13	
--	----	--	--	--------	------------	---------	---------	--	----	--

(b) Met linear probing:

11	39	20	5	16	44	88	12	23	13	94
----	----	----	---	----	----	----	----	----	----	----

(c) Met quadratic probing:

	20	16	11	39	44	88	12	23	13	94
--	----	----	----	----	----	----	----	----	----	----

Het lukt niet om 5 te plaatsen. Daarvoor zou nodig zijn:  $4 + j^2 = 0 \bmod 11$ .

(d) Met double hashing, met als tweede hash functie  $h'(k) = 7 - (k \bmod 7)$ :

11	23	20	16	39	44	94	12	88	13	5
----	----	----	----	----	----	----	----	----	----	---

19. (Dit is ongeveer opgave R-2.23.)

Geef de pseudo-code voor het toevoegen van een item aan een hash table waarbij we quadratic probing gebruiken om collisions op te lossen. We gaan ervan uit dat een verwijderde items vervangen zijn door een speciaal object  $A$  (van het juiste type).

De grootte van de hash-table is  $N$ . Hier wordt gebruikt 'beschikbaar' voor of leeg, of available.

Met quadratic probing:

**Algorithm** insertItem( $k, e$ ):

$j := 0$

**while not**  $A[h(k) + j^2 \bmod N]$  beschikbaar **do**

$j := j + 1$

$A[h(k) + j^2] := (k, e)$

Met double hashing, gebruikmakend van een tweede hash functie  $h'$ :

**Algorithm** insertItem( $k, e$ ):

$j := 0$

**while not**  $A[h(k) + j \cdot h'(k) \bmod N]$  beschikbaar **do**

$j := j + 1$

$A[h(k) + j \cdot h'(k)] := (k, e)$