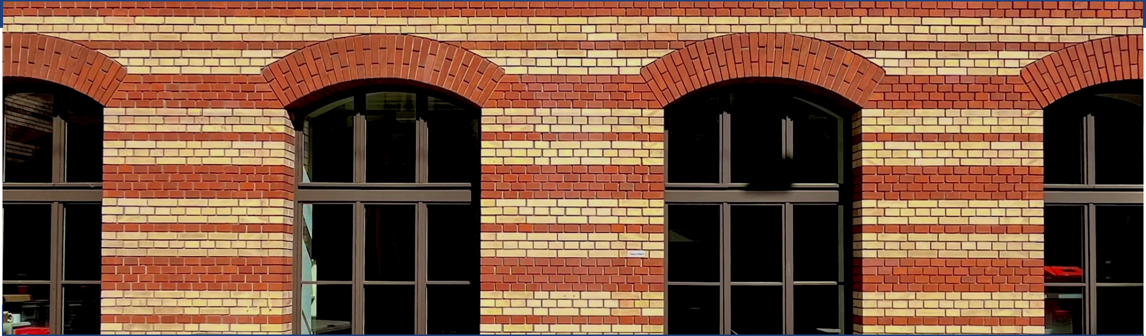


Datastructures and Algorithms

Binary Trees, Heaps, Hashing

Adel Gavranović — ETH Zürich — 2025

Overview

Learning Objectives
Binary Trees and Heaps
Hashing
Binary Tree: Simple Tasks
Code-Example: Hashtables, Hash-functions and Collisions
Past Exam Questions
Tips for **code** expert

n.ethz.ch/~agavranovic

Material

Webpage

Mail

1. Follow-up

Follow-up from last session

Follow-up from last session

- Regarding last week’s in-class coding exercise

Follow-up from last session

redo matrices in class

- Regarding last week's in-class coding exercise
  - No worries if you were not able to solve the example exercise during the session
  - It was a rather hard task to get into (no matter how "easy" it was to solve)

Follow-up from last session

- Regarding last week's in-class coding exercise
  - No worries if you were not able to solve the example exercise during the session
  - It was a rather hard task to get into (no matter how "easy" it was to solve)
- In general: the reference solutions (for the in-class code examples) will now be published sooner

2. Feedback regarding **code** expert

General things regarding code expert

General things regarding **code** expert

- Re Corrections: I’m on it

General things regarding code expert

- Re Corrections: I'm on it
- If you **need** the XP: email me

Any questions regarding **code** expert on your part?

3. Learning Objectives

←

Objectives

- ☐ Understand *Search Trees* and *Heaps*, and operations on them as well as their drawbacks and benefits
- ☐ Be able to perform operations on *Search Trees* and *Heaps* by hand
- ☐ Understand *Hashing*, its components, and related concepts:
  - ☐ Prehashing
  - ☐ Collision
  - ☐ Simple Uniform Hashing
  - ☐ Uniform Hashing
  - ☐ Open/Closed Addressing & Closed/Open Hashing
  - ☐ Chaining
- ☐ Be able to apply simple *hashing methods* by hand

4. Summary

←

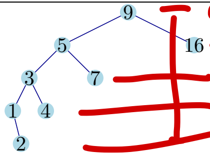
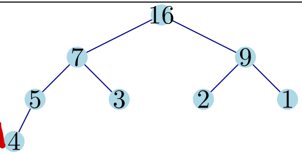
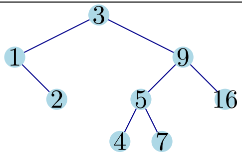
Getting on the same page

←

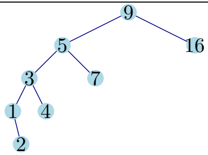
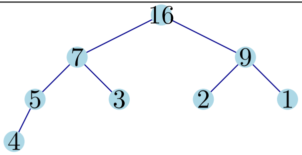
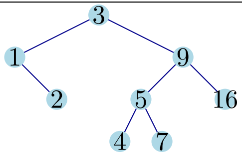
5. Binary Trees and Heaps

←

Comparison of binary Trees

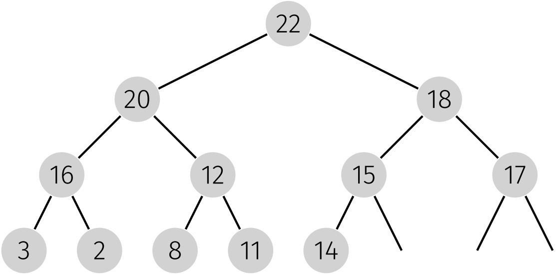
	Search trees	Heaps Min- / Max- Heap	Balanced trees AVL, red-black tree
in C++:		`std::make_heap`	`std::map`
			
Insertion	$\Theta(h(T))$	$\Theta(\log n)$	$\Theta(\log n)$
Search	$\Theta(h(T))$	$\Theta(n)$ (!!)	$\Theta(\log n)$
Deletion	$\Theta(h(T))$	Search + $\Theta(\log n)$	$\Theta(\log n)$
Min/Max	$\Theta(h(T))$	$\Theta(1)$/search	$\Theta(\log n)$

Comparison of binary Trees

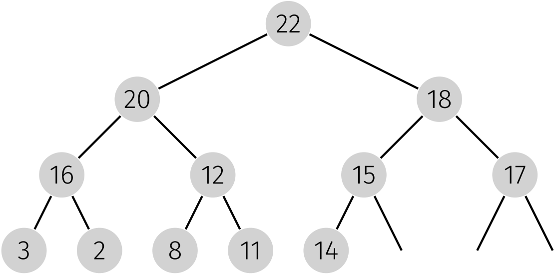
	Search trees	**Heaps** Min- / Max- Heap	**Balanced trees** AVL, red-black tree
in C++:		`std::make_heap`	`std::map`
			
Insertion	$\Theta(h(T))$	$\Theta(\log n)$	$\Theta(\log n)$
Search	$\Theta(h(T))$	$\Theta(n)$ (!!)	$\Theta(\log n)$
Deletion	$\Theta(h(T))$	Search + $\Theta(\log n)$	$\Theta(\log n)$
Min/Max	$\Theta(h(T))$	$\Theta(1)$/search	$\Theta(\log n)$

Remark: $\Theta(\log n) \leq \Theta(h(T)) \leq \Theta(n)$

Recall: Binary Tree as Array

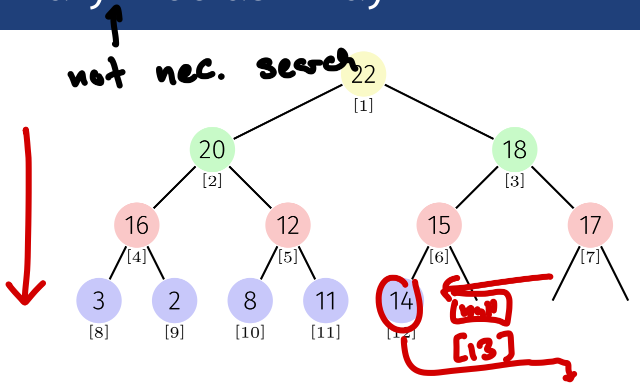


Recall: Binary Tree as Array



22	20	18	16	12	15	17	3	2	8	11	14
1	2	3	4	5	6	7	8	9	10	11	12

Recall: Binary Tree as Array



22	20	18	16	12	15	17	3	2	8	11	14
1	2	3	4	5	6	7	8	9	10	11	12

Repetition: Binary Trees, Inserting a Key

Binary Search Trees

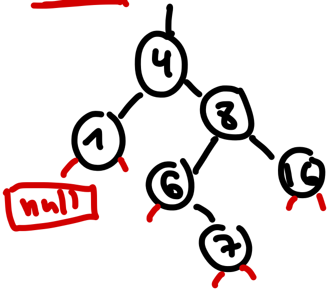
- Search for Key.
- Insert at the reached empty leaf (`null`).

MinHeap

- Insert at the very next free spot (back of the array).
- Restore Heap-Condition: `siftUp` (climb successively).

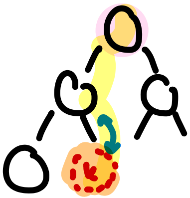
Binary Search Trees

- Search for Key.
- Insert at the reached empty leaf (`null`).



MinHeap

- Insert at the very next free spot (back of the array).
- Restore Heap-Condition: `siftUp` (climb successively).

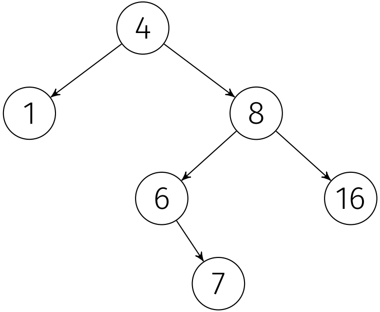


Exercise: Insert $4, 8, 16, 1, 6, 7$ into empty Search Tree/Min-Heap.

Repetition: Binary Trees, Inserting a Key

Binary Search Trees

- Search for Key.
- Insert at the reached empty leaf (`null`).



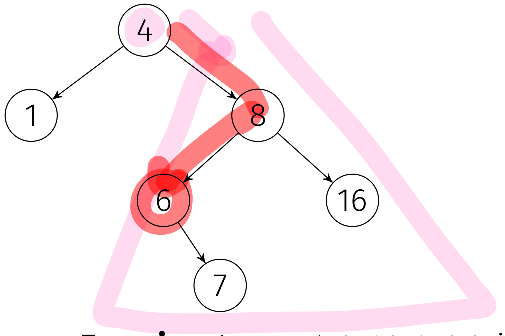
MinHeap

- Insert at the very next free spot (back of the array).
- Restore Heap-Condition: `siftUp` (climb successively).

Exercise: Insert $4, 8, 16, 1, 6, 7$ into empty Search Tree/Min-Heap.

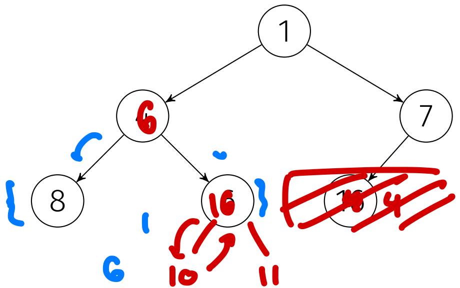
Binary Search Trees

- Search for Key.
- Insert at the reached empty leaf (`null`).



MinHeap

- Insert at the very next free spot (back of the array).
- Restore Heap-Condition: `siftUp` (climb successively).



Exercise: Insert $4, 8, 16, 1, 6, 7$ into empty Search Tree/Min-Heap.

Repetition: Binary Trees, Deleting a Key

Binary Search Trees

- Replace key k by symmetric successor n.
- Careful: What about right child of n?

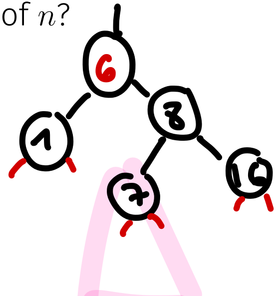
MinHeap

- Replace key by last element of the array.
- Restore Heap-Condition: `siftDown` *or* `siftUp`.

Repetition: Binary Trees, Deleting a Key

Binary Search Trees

- Replace key k by symmetric successor n.
- Careful: What about right child of n?



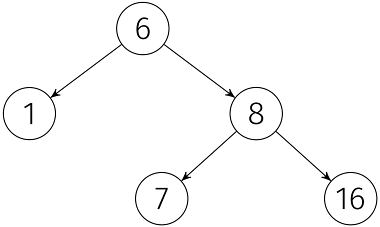
MinHeap

- Replace key by last element of the array.
- Restore Heap-Condition: `siftDown` or `siftUp`.

Exercise: Delete 4 from Search Tree/Min-Heap.

Binary Search Trees

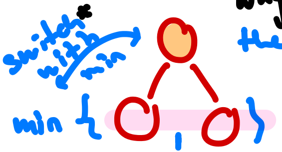
- Replace key k by symmetric successor n.
- Careful: What about right child of n?



MinHeap

- Replace key by last element of the array.
- Restore Heap-Condition: `siftDown` *or* `siftUp`.

why?: because that's the minHeap condit.!



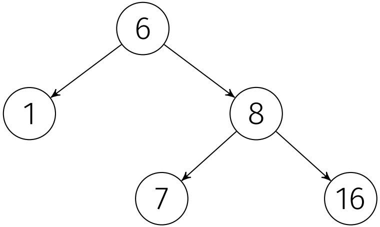
* if parent is not smaller than child.

Exercise: Delete 4 from Search Tree/Min-Heap.

Repetition: Binary Trees, Deleting a Key

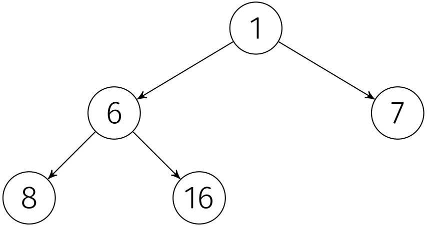
Binary Search Trees

- Replace key k by symmetric successor n.
- Careful: What about right child of n?



MinHeap

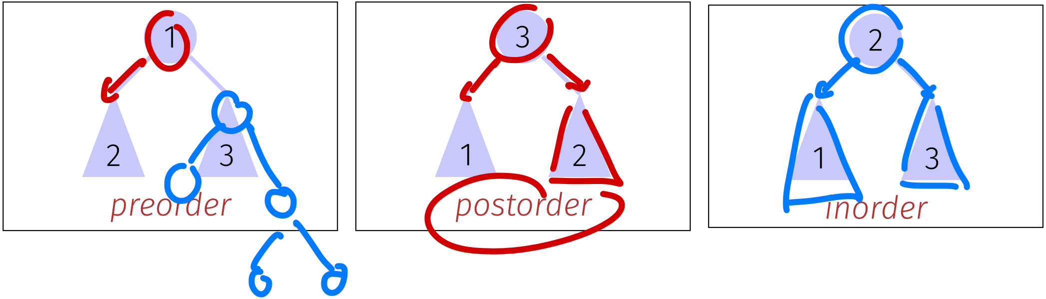
- Replace key by last element of the array.
- Restore Heap-Condition: `siftDown` or `siftUp`.



Exercise: Delete 4 from Search Tree/Min-Heap.

Traversal possibilities

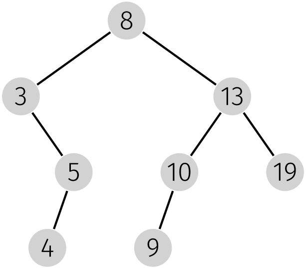
From the Lecture



←

Traversal possibilities

From the Lecture

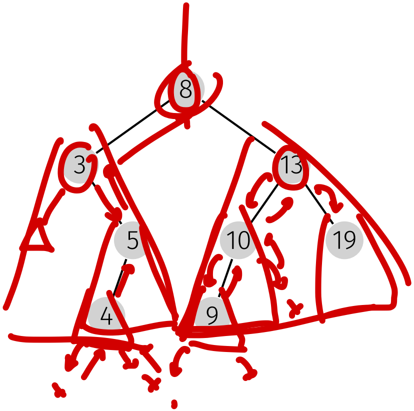


Traversal possibilities

From the Lecture

- *preorder*:
  v, then $T_{\text{left}}(v)$, then $T_{\text{right}}(v)$.

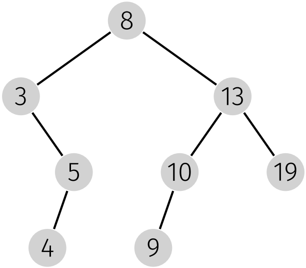
8, 3, 5, 4, 13, 10, 9, 19



Traversal possibilities

From the Lecture

- *preorder*:
  v, then $T_{\text{left}}(v)$, then $T_{\text{right}}(v)$.
  8, 3, 5, 4, 13, 10, 9, 19



Traversal possibilities

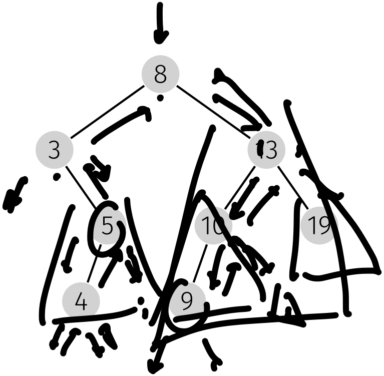
From the Lecture

- *preorder*:
  v, then $T_{\text{left}}(v)$, then $T_{\text{right}}(v)$.
  8, 3, 5, 4, 13, 10, 9, 19

postord(13)

- *postorder*:
  $T_{\text{left}}(v)$, then $T_{\text{right}}(v)$, then v.

4, 5, 3, 9, 10, 19, 13, 8



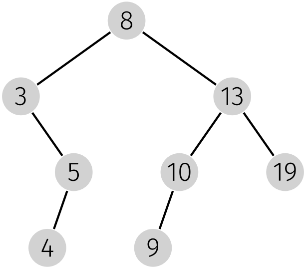
←

Traversal possibilities

From the Lecture

- *preorder*:
  v, then $T_{\text{left}}(v)$, then $T_{\text{right}}(v)$.
  8, 3, 5, 4, 13, 10, 9, 19

- *postorder*:
  $T_{\text{left}}(v)$, then $T_{\text{right}}(v)$, then v.
  4, 5, 3, 9, 10, 19, 13, 8

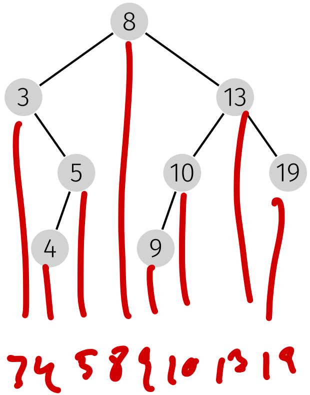


Traversal possibilities

- *preorder*:
  v, then $T_{\text{left}}(v)$, then $T_{\text{right}}(v)$.
  8, 3, 5, 4, 13, 10, 9, 19

- *postorder*:
  $T_{\text{left}}(v)$, then $T_{\text{right}}(v)$, then v.
  4, 5, 3, 9, 10, 19, 13, 8

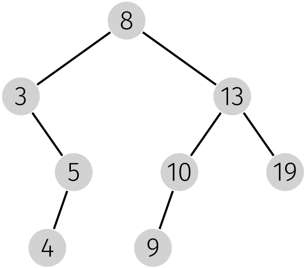
- *inorder*:
  $T_{\text{left}}(v)$, then v, then $T_{\text{right}}(v)$.



Traversal possibilities

From the Lecture

- *preorder*:
  v, then $T_{\text{left}}(v)$, then $T_{\text{right}}(v)$.
  8, 3, 5, 4, 13, 10, 9, 19
- *postorder*:
  $T_{\text{left}}(v)$, then $T_{\text{right}}(v)$, then v.
  4, 5, 3, 9, 10, 19, 13, 8
- *inorder*:
  $T_{\text{left}}(v)$, then v, then $T_{\text{right}}(v)$.
  3, 4, 5, 8, 9, 10, 13, 19



←

Quiz

For each of the following traversals, draw a binary search tree that could have produced such a traversal. Is the tree unique, or could different trees have produced this traversal?

inorder	1 2 3 4 5 6 7 8
preorder	4 3 1 2 8 6 5 7
postorder	1 3 2 5 6 8 7 4

Provide for each order a sequence of numbers from $\{1, \ldots, 4\}$ such that it cannot result from a valid binary search tree

Answers

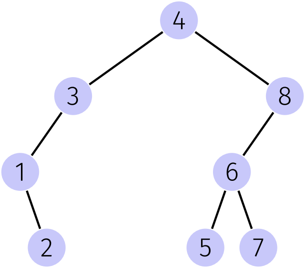
inorder: any binary search tree with numbers $\{1, \ldots, 8\}$ is valid.

The tree is not unique

There is no search tree for any non-sorted sequence. Counterexample 1 2 4 3

Answers

preorder 4 3 1 2 8 6 5 7

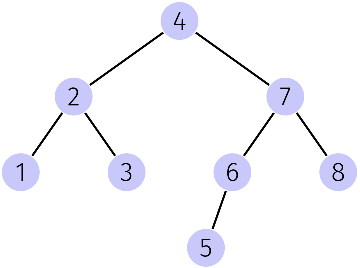


Tree is unique

It must hold recursively that first there is a group of numbers with lower and then with higher number than the first value. Counterexample: 3 1 4 2

Answers

postorder 1 3 2 5 6 8 7 4



Tree is unique

Construction here: https://www.techiedelight.com/build-binary-search-tree-from-postorder-sequence/, similar argument as before, but backwards. Counterexample 4 2 1 3

Quiz

True or false:

1. The preorder is the reversed postorder.
2. The first node in the preorder is always the root.
3. The first node in the inorder is never the root.
4. Inserting the nodes in preorder into an empty tree leads to the same tree.
5. Inserting the nodes in postorder into an empty tree leads to the same tree.
6. Inserting the nodes in inorder into an empty tree leads to the same tree.

Quiz: Solution

True or false:

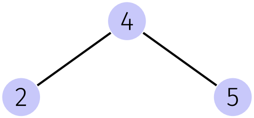
1. The preorder is the reversed postorder.

←

Quiz: Solution

True or false:

1. The preorder is the reversed postorder.
   False Preorder: $4, 2, 5$. Postorder: $2, 5, 4$.

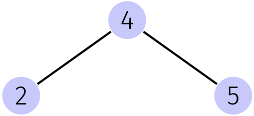


←

Quiz: Solution

True or false:

1. The preorder is the reversed postorder.
   False Preorder: $4, 2, 5$. Postorder: $2, 5, 4$.

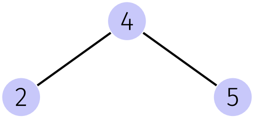


2. The first node in the preorder is always the root.

Quiz: Solution

True or false:

1. The preorder is the reversed postorder.
   False Preorder: $4, 2, 5$. Postorder: $2, 5, 4$.



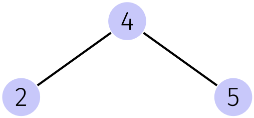
2. The first node in the preorder is always the root.
   true (by definition!)

←

Quiz: Solution

True or false:

1. The preorder is the reversed postorder.
   False Preorder: $4, 2, 5$. Postorder: $2, 5, 4$.

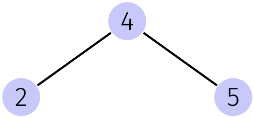


2. The first node in the preorder is always the root.
   true (by definition!)
3. The first node in the inorder is never the root.

Quiz: Solution

True or false:

1. The preorder is the reversed postorder.
   False Preorder: $4, 2, 5$. Postorder: $2, 5, 4$.



2. The first node in the preorder is always the root.
   true (by definition!)
3. The first node in the inorder is never the root.
   False. When the left subtree is empty, the root is the first node inorder.

Quiz: Solution

True or false:

4. Inserting the nodes of a tree in preorder into a new empty tree leads to the same tree.

Quiz: Solution

True or false:

4. Inserting the nodes of a tree in preorder into a new empty tree leads to the same tree.
   True. Since first the root is inserted and then its children, we will get the same tree.

Quiz: Solution

True or false:

4. Inserting the nodes of a tree in preorder into a new empty tree leads to the same tree.
   True. Since first the root is inserted and then its children, we will get the same tree.
5. Inserting the nodes in postorder into an empty tree leads to the same tree.

Quiz: Solution

True or false:

4. Inserting the nodes of a tree in preorder into a new empty tree leads to the same tree.
   True. Since first the root is inserted and then its children, we will get the same tree.
5. Inserting the nodes in postorder into an empty tree leads to the same tree.
   False. But it is true for the reversed postorder!

Quiz: Solution

True or false:

4. Inserting the nodes of a tree in preorder into a new empty tree leads to the same tree.
   True. Since first the root is inserted and then its children, we will get the same tree.
5. Inserting the nodes in postorder into an empty tree leads to the same tree.
   False. But it is true for the reversed postorder!
6. Inserting the nodes in inorder into an empty tree leads to the same tree.

←

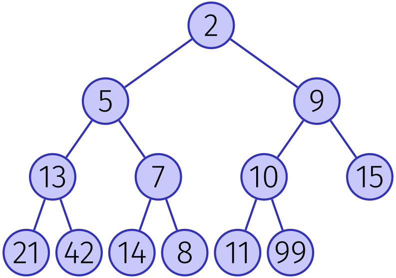
Quiz: Solution

True or false:

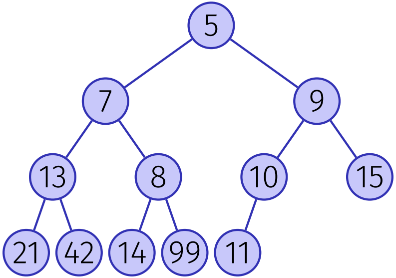
4. Inserting the nodes of a tree in preorder into a new empty tree leads to the same tree.
   True. Since first the root is inserted and then its children, we will get the same tree.
5. Inserting the nodes in postorder into an empty tree leads to the same tree.
   False. But it is true for the reversed postorder!
6. Inserting the nodes in inorder into an empty tree leads to the same tree.
   False. There are many different trees with the same inorder!

Heap

On the following Min-Heap, perform an extract-min operation, including re-establishing the heap-condition, as shown in class. What does the heap look like after the operation?



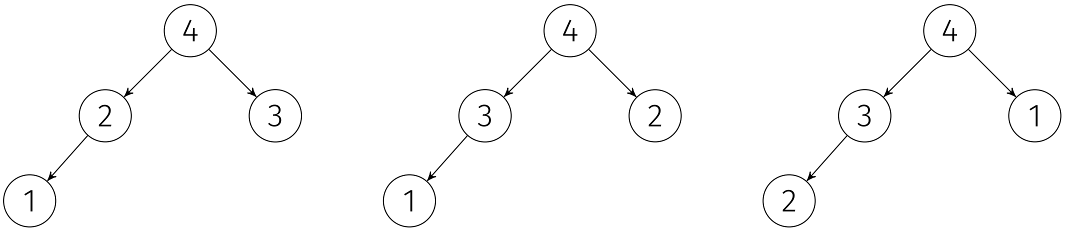
Solution



←

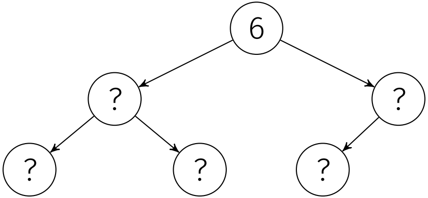
Quiz: Number of MaxHeaps on n keys

Let $N(n)$ denote the number of distinct Max-Heaps which can be built from all the keys $1, 2, \ldots, n$. For example we have
$N(1) = 1,\ N(2) = 1,\ N(3) = 2,\ N(4) = 3$ and $N(5) = 8$.
Find the values $N(6)$ and $N(7)$.



Number of MaxHeaps on n distinct keys

A MaxHeap containing the elements $1, 2, 3, 4, 5, 6$ has the structure:



Number of combinations to choose elements for the left subtree: $\binom{5}{3}$.

$$\Rightarrow N(6) = \binom{5}{3} \cdot N(3) \cdot N(2) = 10 \cdot 2 \cdot 1 = 20.$$

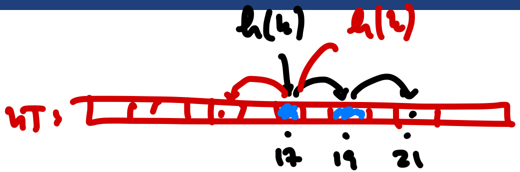
$$\text{and } N(7) = \binom{6}{3} \cdot N(3) \cdot N(3) = 20 \cdot 2 \cdot 2 = 80.$$

$\leftarrow$

6. Hashing

←

Hashing well-done



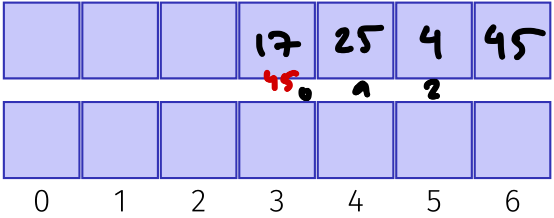
Useful Hashing...

- distributes the keys as uniformly as possible in the hash table.
- avoids probing over long areas of used entries (e.g. primary clustering).
- avoids using the same probing sequence for keys with the same hash value (e.g. secondary clustering).

Hashing Examples

Insert the keys 25, 4, 17, 45 into the hash table, using the function $h(k) = k \bmod 7$ and probing to the right, $h(k) + \mathit{offset}(j, k)$:

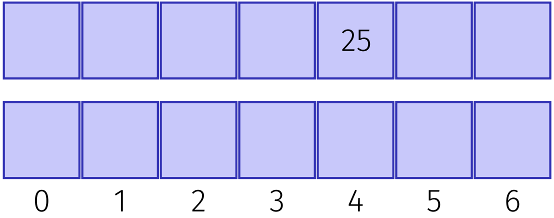
- linear probing,
  $\mathit{offset}(j, k) = j$;
- Double Hashing,
  $\mathit{offset}(j, k) = j \cdot (1 + (k \bmod 5))$.



Hashing Examples

Insert the keys $25, 4, 17, 45$ into the hash table, using the function $h(k) = k \bmod 7$ and probing to the right, $h(k) + \mathit{offset}(j, k)$:

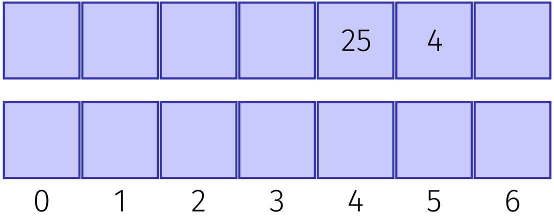
- linear probing,
  $\mathit{offset}(j, k) = j$.
- Double Hashing,
  $\mathit{offset}(j, k) = j \cdot (1 + (k \bmod 5))$.



Hashing Examples

Insert the keys $25, 4, 17, 45$ into the hash table, using the function $h(k) = k \bmod 7$ and probing to the right, $h(k) + \mathit{offset}(j, k)$:

- linear probing,
  $\mathit{offset}(j, k) = j$.
- Double Hashing,
  $\mathit{offset}(j, k) = j \cdot (1 + (k \bmod 5))$.



Hashing Examples

Insert the keys $25, 4, 17, 45$ into the hash table, using the function $h(k) = k \bmod 7$ and probing to the right, $h(k) + \mathit{offset}(j, k)$:

- linear probing,
  $\mathit{offset}(j, k) = j$.
- Double Hashing,
  $\mathit{offset}(j, k) = j \cdot (1 + (k \bmod 5))$.

0	1	2	3	4	5	6
			17	25	4	

Hashing Examples

☐ will be in follow-up next week

j: specific to each "new" k

Insert the keys $25, 4, 17, 45$ into the hash table, using the function $h(k) = k \bmod 7$ and probing to the right, $h(k) + \mathit{offset}(j, k)$:

- linear probing, $\mathit{offset}(j, k) = j$.
- Double Hashing, $\mathit{offset}(j, k) = j \cdot (1 + (k \bmod 5))$.

"second hash function"

↑ still counting collisions

			17	25	4	45
0	1	2	3	4	5	6

$t = 7$

$$H(k, j) = \left(h(k) + j \cdot h'(k)\right) \bmod t$$

← j = 0; j++ after every attmp.

Hashing Examples

Insert the keys $25, 4, 17, 45$ into the hash table, using the function $h(k) = k \bmod 7$ and probing to the right, $h(k) + \mathit{offset}(j, k)$:

- linear probing,
  $\mathit{offset}(j, k) = j$.
- Double Hashing,
  $\mathit{offset}(j, k) = j \cdot (1 + (k \bmod 5))$.

			17	25	4	45
				25		
0	1	2	3	4	5	6

Hashing Examples

Insert the keys $25, 4, 17, 45$ into the hash table, using the function $h(k) = k \bmod 7$ and probing to the right, $h(k) + \mathit{offset}(j, k)$:

- linear probing,
  $\mathit{offset}(j, k) = j$.
- Double Hashing,
  $\mathit{offset}(j, k) = j \cdot (1 + (k \bmod 5))$.

0	1	2	3	4	5	6
			17	25	4	45
		4		25		

Hashing Examples

Insert the keys $25, 4, 17, 45$ into the hash table, using the function $h(k) = k \bmod 7$ and probing to the right, $h(k) + \mathit{offset}(j, k)$:

- linear probing,
  $\mathit{offset}(j, k) = j$.
- Double Hashing,
  $\mathit{offset}(j, k) = j \cdot (1 + (k \bmod 5))$.

			17	25	4	45
		4	17	25		
0	1	2	3	4	5	6

Hashing Examples

Insert the keys $25, 4, 17, 45$ into the hash table, using the function $h(k) = k \bmod 7$ and probing to the right, $h(k) + \mathit{offset}(j, k)$:

- linear probing,
  $\mathit{offset}(j, k) = j$.
- Double Hashing,
  $\mathit{offset}(j, k) = j \cdot (1 + (k \bmod 5))$.

			17	25	4	45
		4	17	25	45	
0	1	2	3	4	5	6

←

Quiz: Hashing

A hash table of length 10 uses closed hashing with hash function $h(k) = k \bmod 10$, and linear probing (probing goes to the right). After inserting five values into an empty hash table, the table is as shown below.

0	1	2	3	4	5	6	7	8	9
		32	52	33	74	96			

Which of the following *choice(s)* give possible order(s) in which the key values could have been inserted in the table?

(A) 32, 33, 52, 96, 74

(B) 32, 52, 33, 74, 96

(C) 32, 52, 74, 96, 33

(D) 96, 32, 52, 33, 74

←

Quiz: Hashing

A hash table of length 10 uses closed hashing with hash function $h(k) = k \bmod 10$, and linear probing (probing goes to the right). After inserting five values into an empty hash table, the table is as shown below.

0	1	2	3	4	5	6	7	8	9
		32	52	33	74	96			

Which of the following *choice(s)* give possible order(s) in which the key values could have been inserted in the table?

(A) 32, 33, 52, 96, 74

(B) 32, 52, 33, 74, 96 ☺

(C) 32, 52, 74, 96, 33

(D) 96, 32, 52, 33, 74 ☺

7. Binary Tree: Simple Tasks

Hands-on example: Binary Tree

←

8. Code-Example: Hashtables, Hash-functions and Collisions

Hands-on example: importance of a well designed hashing strategy

←

9. Past Exam Questions

Past Exam 2022: Task 1d)

In der folgenden Tabelle ist ein Min-Heap in seiner üblichen Form gespeichert. Wie sieht die Tabelle aus, nachdem `ExtractMin` ausgeführt wurde?

The following table comprises a Min-Heap in its canonical form. What does the table look like after `ExtractMin` *has been executed?*

2	5	3	8	7	4	10	13	9	33	11
1	2	3	4	5	6	7	8	9	10	11

1	2	3	4	5	6	7	8	9	10

Past Exam 2022: Task 1d) – Solution

In der folgenden Tabelle ist ein Min-Heap in seiner üblichen Form gespeichert. Wie sieht die Tabelle aus, nachdem `ExtractMin` ausgeführt wurde?

The following table comprises a Min-Heap in its canonical form. What does the table look like after `ExtractMin` *has been executed?*

2	5	3	8	7	4	10	13	9	33	11
1	2	3	4	5	6	7	8	9	10	11

3	5	4	8	7	11	10	13	9	33
1	2	3	4	5	6	7	8	9	10

←

10. Tips for **code** expert

Finding a Sub-Array

- Given: two integer arrays $A = (a_0, \ldots, a_{n-1})$ and $B = (b_0, \ldots, b_{k-1})$
- Task: Find position of B in A.

←

Finding a Sub-Array

- Given: two integer arrays $A = (a_0, \ldots, a_{n-1})$ and $B = (b_0, \ldots, b_{k-1})$
- Task: Find position of B in A.
- Naive: Loop through A, check whether the following k entries match B.

Finding a Sub-Array

- Given: two integer arrays $A = (a_0, \ldots, a_{n-1})$ and $B = (b_0, \ldots, b_{k-1})$
- Task: Find position of B in A.
- Naive: Loop through A, check whether the following k entries match B.
  - $O(nk)$ comparison operations

Finding a Sub-Array

- Given: two integer arrays $A = (a_0, \ldots, a_{n-1})$ and $B = (b_0, \ldots, b_{k-1})$
- Task: Find position of B in A.
- Naive: Loop through A, check whether the following k entries match B.
  - $O(nk)$ comparison operations
- Solution using hashing: Calculate hash $h(B)$ and compare it to $h((a_i, a_{i+1}, \ldots, a_{i+k-1}))$.
- Avoid re-computing $h((a_i, a_{i+1}, \ldots, a_{i+k-1})$ for each $i \implies O(n)$ expected

Sliding Window Hash

- Possible hash function: sum of all elements:
  - Can be updated easily: subtract a_i and add a_{i+k}.
  - However: bad hash function

Sliding Window Hash

- Possible hash function: sum of all elements:
  - Can be updated easily: subtract a_i and add a_{i+k}.
  - However: bad hash function
- Better:

$$H_{c,m}((a_i, \cdots, a_{i+k-1})) = \left(\sum_{j=0}^{k-1} a_{i+j} \cdot c^{k-j-1} \right) \bmod m$$

  - Let c be a prime number: $c = 1021$
  - Let $m = 2^{15} + 3$
  - Since m is just over 2^{15}, intermediate multiplications and additions fit safely in a 32-bit integer, avoiding overflow.

Sliding Window Hash

Make sure that

- the algorithm computes c^k only once,
- all computations are modulo m for all values in order not to get an overflow (recall the rules of modular arithmetic), and
- the values are always positive (e.g., by adding multiples of m).

Computing with Modulo

$$(a + b) \bmod m = ((a \bmod m) + (b \bmod m)) \bmod m$$
$$(a - b) \bmod m = ((a \bmod m) - (b \bmod m) + m) \bmod m$$
$$(a \cdot b) \bmod m = ((a \bmod m) \cdot (b \bmod m)) \bmod m$$

Exercise: Compute

$$12746357 \bmod 11$$

Computing Modulo

Exercise: Compute

$$12746357 \bmod 11$$

Computing Modulo

Exercise: Compute

$$
\begin{aligned}
&12746357 \bmod 11 \\
&= (7 + 5 \cdot 10 + 3 \cdot 10^2 + 6 \cdot 10^3 + 4 \cdot 10^4 + 7 \cdot 10^5 + 2 \cdot 10^6 + 1 \cdot 10^7) \bmod 11
\end{aligned}
$$

Computing Modulo

Exercise: Compute

$$
\begin{aligned}
& 12746357 \bmod 11 \\
& = (7 + 5 \cdot 10 + 3 \cdot 10^2 + 6 \cdot 10^3 + 4 \cdot 10^4 + 7 \cdot 10^5 + 2 \cdot 10^6 + 1 \cdot 10^7) \bmod 11 \\
& = (7 + 50 + 3 + 60 + 4 + 70 + 2 + 10) \bmod 11
\end{aligned}
$$

For the second equality we used the fact that $10^2 \bmod 11 = 1$.

Computing Modulo

Exercise: Compute

$$\begin{aligned}
&12746357 \bmod 11\\
&= (7 + 5 \cdot 10 + 3 \cdot 10^2 + 6 \cdot 10^3 + 4 \cdot 10^4 + 7 \cdot 10^5 + 2 \cdot 10^6 + 1 \cdot 10^7) \bmod 11\\
&= (7 + 50 + 3 + 60 + 4 + 70 + 2 + 10) \bmod 11\\
&= (7 + 6 + 3 + 5 + 4 + 4 + 2 + 10) \bmod 11
\end{aligned}$$

For the second equality we used the fact that $10^2 \bmod 11 = 1$.

←

Computing Modulo

Exercise: Compute

$$\begin{aligned}
&12746357 \bmod 11 \\
&= (7 + 5 \cdot 10 + 3 \cdot 10^2 + 6 \cdot 10^3 + 4 \cdot 10^4 + 7 \cdot 10^5 + 2 \cdot 10^6 + 1 \cdot 10^7) \bmod 11 \\
&= (7 + 50 + 3 + 60 + 4 + 70 + 2 + 10) \bmod 11 \\
&= (7 + 6 + 3 + 5 + 4 + 4 + 2 + 10) \bmod 11 \\
&= 8 \bmod 11.
\end{aligned}$$

For the second equality we used the fact that $10^2 \bmod 11 = 1$.

←

11. Outro

General Questions?

See you next time!

Have a nice week!