CS213/293 Data Structure and Algorithms 2023

Lecture 12: Heap

Instructor: Ashutosh Gupta

IITB India

Compile date: 2023-09-14

Topic 12.1

Priority queue

Scheduling problem

On a computational server, users are submitting jobs to run on a single CPU.

- A user also declares the expected run time of the job.
- Jobs can be preempted.

Policy: shortest remaining processing time, which allows interruption of a job if a new job with smaller run time is submitted.

The policy minimizes average waiting time.

Scheduling problem operations

We need the following operations for the scheduling problem.

- Update the remaining time in every tick
- Delete a job when the remaining time is zero
- Find the next job to run
- Insert a job when arrives

Definition 12.1

In a priority queue, we dequeue the highest priority element from the enqueue elements with priorities.

Interface of priority queue

https://en.cppreference.com/w/cpp/container/priority_queue

- `priority_queue<T,Container,Compare> q` : allocates new queue q
- `q.push(e)` : adds the given element e to the queue.
- `q.pop()` : removes the highest priority element from the queue.
- `q.top()` : access the highest priority element.

- `Container` class defines the physical data structure where the queue will be stored. The default value is `Vector`.
- `Compare` class defines the method of comparing priorities of two elements.

Exercise 12.1

Give an implementation for the scheduling problem using the C++ priority queue.

Topic 12.2

Implementations of priority queue

Implementation using unsorted linked list/array

In case we use a linked list,

- We implement `q.push` by inserting the element at the front of the linked list, which is $O(1)$ operation.
- We need to scan the entire list to find the maximum for implementing `q.pop` and `q.top`

Exercise 12.2

How will we implement a priority queue over unsorted arrays?

Implementation using sorted linked list/array

In case we use a linked list,

- The maximum will be at the end of the list. We can implement `q.pop` and `q.top` in $O(1)$.
- However, `q.push(e)` needs to scan the entire list to find the right place to insert e, which is $O(n)$ operation.

Priority queue

Priority queue is one of the fundamental containers.

Many other algorithms assume access to efficient priority queues.

We will define a data structure heap that provides an efficient implementation for the priority queue.

Commentary: Heap is like the the red-black tree, which provides an efficient implementation for ordered maps.

Topic 12.3

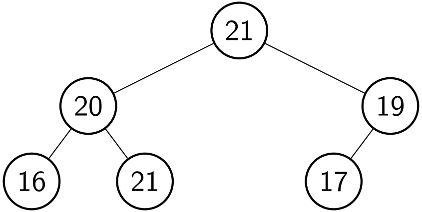
Heap - somewhat sorting!

Heap

Definition 12.2
A heap T is a binary tree such that the following holds.

- *(structural property) All levels are full except the last one and the last level is left filled.*
- *(heap property) for each non-root node n, $key(n) \leq key(parent(n))$.*

Example 12.1
An example of heap.

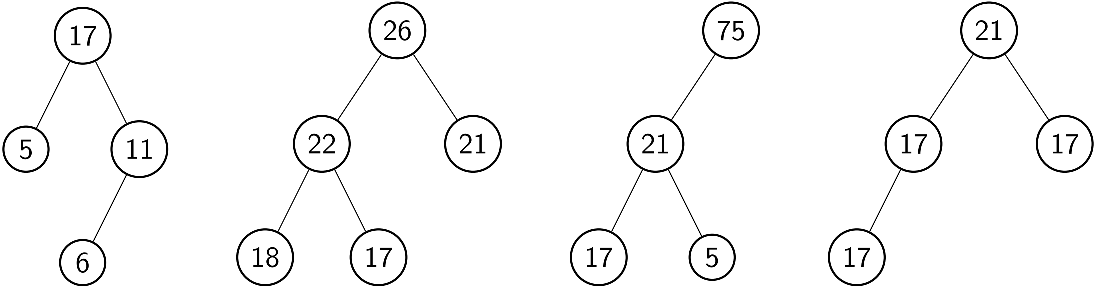


Exercise 12.3
Show that nodes on a path from root to a leaf have keys in non-increasing order.

Exercise: Identify Heap

Exercise 12.4

Which of the following are Heaps?



Algorithm: maximum

Algorithm 12.1: MAXIMUM(Heap T)

return $T[0]$

- Correctness
  - Let us suppose the maximum is not at the root.
  - There is a node n that has maximum key but $parent(n)$ has greater key, which violates heap condition.
  - Contradiction.
- Running time is $O(1)$.

Height of heap

Let us suppose a heap has n nodes and height h.

The number of nodes in a complete binary tree of height h is $2^h - 1$.

Therefore,

$$2^{h-1} - 1 < n \leq 2^h - 1.$$

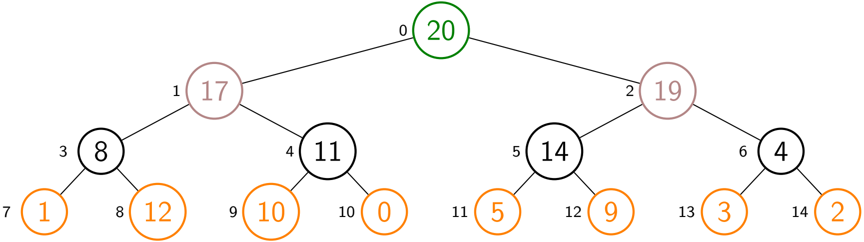
Therefore $n = \lfloor \log_2 n \rfloor$

Exercise 12.5

Give an example of a heap that touches the lower bound.

Storing heap

Let us number the nodes of a heap in the order of level.



$parent(i) = (i-1)/2$, $left(i) = 2i+1$, and $right(i) = 2i+2$.
We place the nodes on an array and traverse the heap using the above equations.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
20	17	19	8	11	14	4	1	12	10	0	5	9	3	2

Since the last level is left filled, we are guaranteed the nodes are contiguously placed.
Instead of writing $key(i)$ of node i in heap T, we will write $T[i]$ to indicate the key.

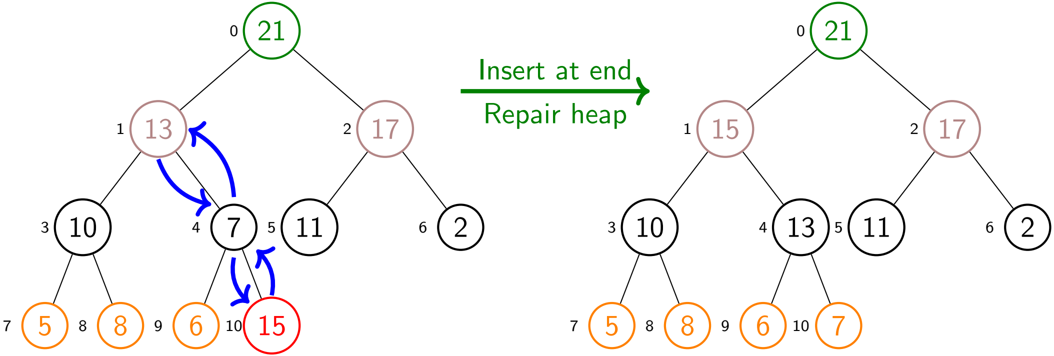
Topic 12.4

Insert in heap

Example: Insert in Heap

Example 12.2

Where do we insert 15?



- Insert at the first available place, which is easy to spot. (Why?)
- Move up the new key if the heap property is violated.

Algorithm: Insert

Algorithm 12.2: INSERT(Heap T, key k)

```
i := T.size;
T[i] := k;
while i > 0 and T[parent(i)] < T[i] do
    SWAP(T, parent(i), i);
    i := parent(i)
T.size := T.size + 1;
```

- Correctness
  - Structural property holds due to the insertion position.
  - Due to the heap property of input T, the path to i the nodes must be in non-increasing order.
  - Let i_0 be the value of i when the loop exits.
  - INSERT replaces the keys of the nodes in the path from i_0 to $T.size$ with the keys of their parents, which implies the keys do not decrease at the nodes.
  - Therefore, no introduction of a violation.
  - Therefore, we will have a heap at the end.
- Running time is $O(\log T.size)$.

Topic 12.5

Heapify

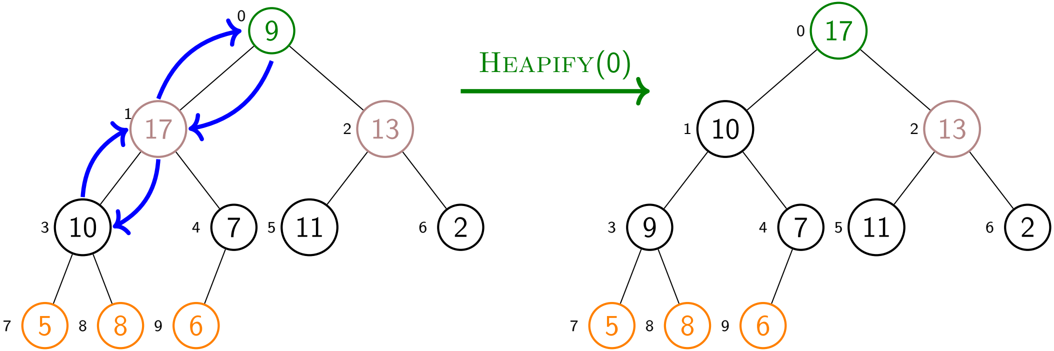
Heapify : a basic operation on a heap

- Let i be a node of heap T
- Let us suppose the binary trees rooted at $left(i)$ and $right(i)$ are valid heaps.
- $T[i]$ may be smaller than its children and violates the heap property.
- The method HEAPIFY makes the binary tree rooted at i a heap by pushing down $T[i]$ in the tree.

Example: Heapify

Example 12.3

The trees rooted at positions 1 and 2 are heaps. We have a violation at position 0*. Heapify will fix the problem by moving the key down.*



- Keep moving down to the child which has the maximum key. (Why?)

Algorithm: Heapify

Algorithm 12.3: $\textsc{Heapify}$(Heap T, i)

$c := \textsc{IndexWithLargestKey}(T, i, left(i), right(i))$ //assume $A[i] = -\infty$ if $i \geq T.size$.
if $c == i$ **then** **return**;
$\mathrm{SWAP}(T, c, i)$;
$\textsc{Heapify}(T, c)$;

- Correctness
  - Same as insert, but we are pushing down.
- Running time is $O(\log T.size)$.

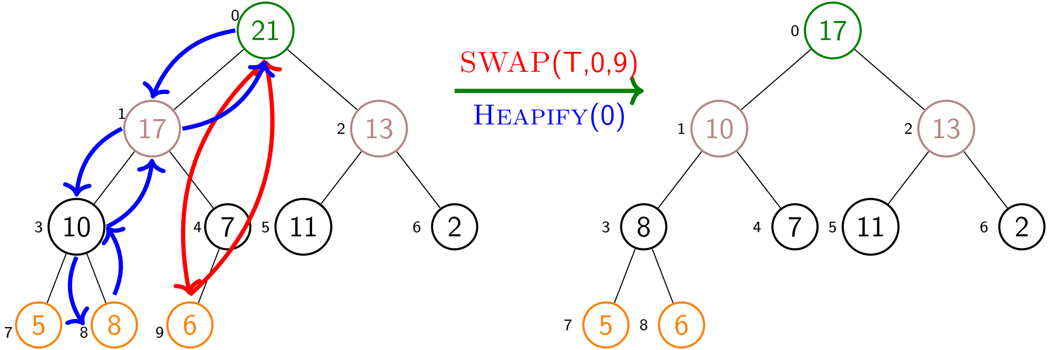
Topic 12.6

Delete maximum in heap

Example: DeleteMax

Example 12.4

Let us delete 21 at position 0.



- Swap with the last position, delete the last position, and run Heapify.

Algorithm: DeleteMax

Algorithm 12.4: $\textsc{DeleteMax}$(Heap T)

$\text{SWAP}(T, 0, T.size - 1)$;
$T.size := T.size - 1$;
$\textsc{Heapify}(T, 0)$;
return $T[T.size]$;

- Correctness
  - The maximum element is removed and heapify returns a heap.
- Running time is $O(\log T.size)$.

Topic 12.7

Build heap

Build heap https://en.cppreference.com/w/cpp/algorithm/make_heap

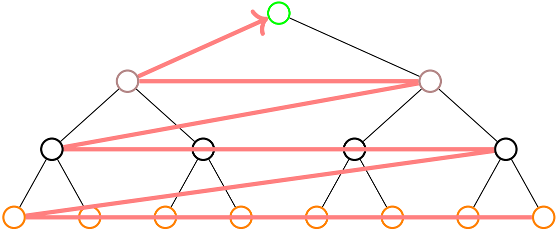
- Input: A binary tree T that has the structural property
  - If structural property holds, then the T is an array
- Output: A heap over elements of T

Algorithm: BuildHeap

Algorithm 12.5: BUILDHEAP(Heap T)

1 **for** $i := T.size - 1$ **downto** *0* **do**
2   HEAPIFY(T, i)

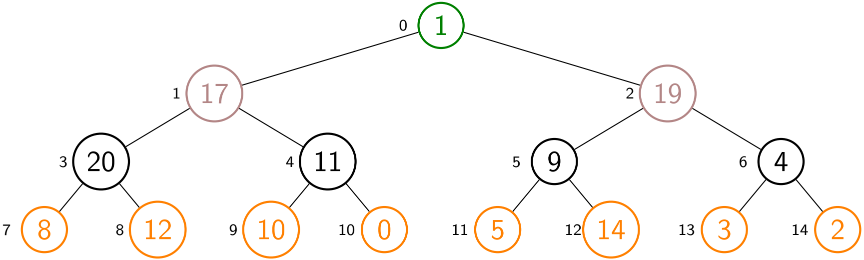
Order of processing in BUILDHEAP.



Example: BuildHeap

Example 12.5

Consider sequence 1 17 19 20 11 9 4 8 12 10 0 5 14 3 2. Let us fill them in the following tree.



BuildHeap traverses the tree bottom up. Heapify calls apply the following swap operations.

- Heapify(T,5): SWAP(T,5,12)
- Heapify(T,1): SWAP(T,1,3)
- Heapify(T,0): SWAP(T,0,1); SWAP(T,1,3); SWAP(T,3,8);

Correctness of BuildHeap

- Correctness by induction
  - **Base case:**
    If i does not have children, it is already a heap.
  - **Induction step:**
    We know $\mathit{left}(i) > i$ or $\mathit{right}(i) > i$.
    Due to the induction hypothesis, both the subtrees are heap before processing i.
    Therefore, $\textsc{Heapify}(T, i)$ will return a heap rooted at i.

Running time of BuildHeap

Heapify for i has $O(height(i))$ swaps.

Let us suppose T is a complete tree with n nodes.

At height h the number of nodes is $\lceil n/2^{h+1} \rceil$ and the height of T is $\lfloor \log n \rfloor$.

The total running time of BuildHeap is

$$\sum_{h=0}^{\lfloor \log n \rfloor} O(h) \lceil n/2^{h+1} \rceil = O(\frac{n}{2} \sum^{\lfloor \log n \rfloor} \frac{h}{2^h})$$

Since $\sum_{h=0}^{\infty} \frac{h}{2^h} = 2$, the running time is $O(n)$.

Some calculation

We know

$$\sum_{i=0}^{\infty} x^i = \frac{1}{1-x}$$

After differentiating over x,

$$\sum_{i=0}^{\infty} ix^{i-1} = \frac{1}{(1-x)^2}$$

After multiplying with x,

$$\sum_{i=0}^{\infty} ix^{i} = \frac{x}{(1-x)^2}$$

After putting $x = 1/2$,

$$\sum_{i=0}^{\infty} \frac{i}{2^i} = 2$$

Topic 12.8

Heapsort

Heapsort

Algorithm 12.6: HEAPSORT(Tree T)

```
T.size = |nodes of T|;
BUILDHEAP(T);
while T.size > 0 do
    DELETEMAX(T)
```

- Since DELETEMAX moves maximum to $T.size - 1$ position, the array is sorted in place.
- Running time:
  - BUILDHEAP is $O(n)$
  - DELETEMAX(T) is $O(\log i)$ at size i.
- Total running time: $O(n \log n)$.

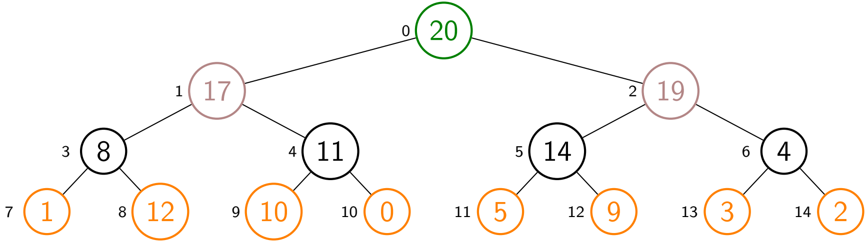
Exercise 12.6

Both BUILDHEAP *and the above loop have iterative runs of* HEAPIFY *in them. Why are their running time complexities different?*

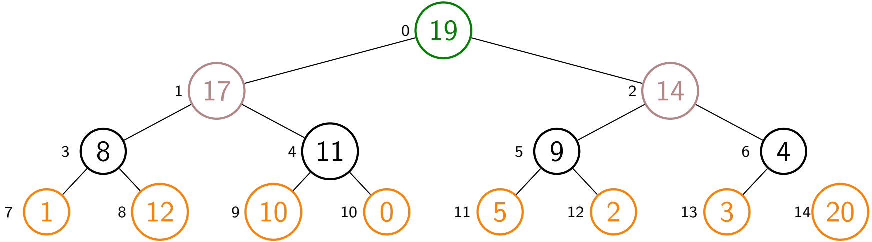
Commentary: Please solve the above exercise to clearly understand the relevant mathematics.

Example: Heapsort

Consider the following Heap obtained after running BUILDHEAP.

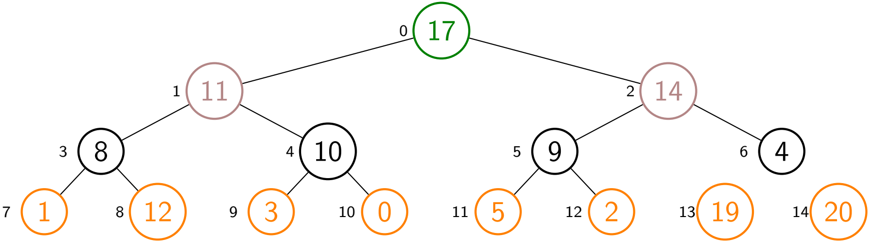


After the first DELETEMAX,



Example: Heapsort(2)

After the second DELETEMAX,



DELEATEMAX has placed 19 and 20 at their sorted position.

Topic 12.9

Tutorial problems

Exercise: Why heap?

Exercise 12.7

Can a Priority Queue be implemented as a red-black tree? What advantages does a Heap implementation has over a red-black tree implementation?

Exercise: 2D-matrix

Exercise 12.8

Suppose we have a 2D array where we maintain the following conditions: for every (i,j), we have $A(i,j) \leq A(i+1,j)$ *and* $A(i,j) \leq A(i,j+1)$. *Can this be used to implement a priority queue?*

Topic 12.10

Problems

End of Lecture 12