

# Tree→Heap→Priority Queue

Rivers

171860553@smail.nju.edu.cn

May 23, 2018

# Section 1

## Contents

# Contents

- Pseudocode
- C source code

# Contents

- Pseudocode
- C source code

- ① Why is array preferred?
- ② Those elementary things
- ③ How to implement
  - Tree  $\rightarrow$  Heap
    - MAX-HEAPIFY
    - BUILD-MAXHEAP-FROMTREE
  - HEAPSORT
  - Heap  $\rightarrow$  Priority Queue
    - HEAPMAX
    - EXTRACT-HEAPMAX
    - HEAP-INCREASE-KEY
    - MAXHEAP-INSERT
- ④ Easter egg
  - BUILD-COMPLETE-

## Section 2

Why is array preferred?

PARENT( $i$ )

1 **return**  $\lfloor i/2 \rfloor$

LEFT( $i$ )

1 **return**  $2i$

RIGHT( $i$ )

1 **return**  $2i + 1$

1  $A[1] = A[\text{heap-size}]$

2  $\text{heap-size} = \text{heap-size} - 1$

1  $A[i] = \text{key}$

All of them is  $\Theta(1)$

How can we  
implement these  
operations with tree  
form?

Can we maintain  
these amazing **time**  
**costs**?

## Section 3

# Those elementary things

# Structure of Tree Node

key  
parent  
left  
right

PARENT( $T$ )

1 **return**  $T.parent$

LEFT( $T$ )

1 **return**  $T.left$

RIGHT( $T$ )

1 **return**  $T.right$



PARENT( $i$ )

1 **return**  $\lfloor i/2 \rfloor$

LEFT( $i$ )

1 **return**  $2i$

RIGHT( $i$ )

1 **return**  $2i + 1$

1  $A[1] = A[\text{heap-size}]$

2  $\text{heap-size} = \text{heap-size} - 1$

1  $A[i] = \text{key}$

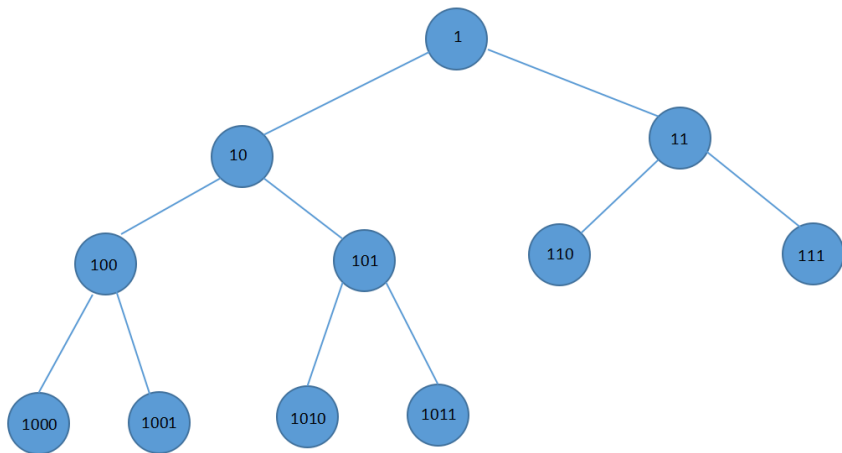


Figure: Heap Node Number and Direction of Path

$$H \begin{cases} \text{HeapTree} \\ \text{Total} \\ \text{Last} \end{cases}$$

FINDLAST(H)

```

1  H.Last = H.HeapTree
2  index = 0
3  while ( $1 \ll \textit{index} \leq \textit{H.total}$ )
4      Direction[index] =
        ( $(\textit{H.Total}) \gg \textit{index}$ ) & 1
5      index = index + 1
6  index = index - 2
7  while index  $\geq 0$ 
8      if Direction[index] == 1
9          Last = Last.right
10     else
11         Last = Last.left
12     index = index - 1

```

FIND-NODE( $H, N$ )

```

1  result = H.HeapTree
2  index = 0
3  while ( $1 \ll \textit{index} \leq N$ )
4      Direction[index] =
        ( $N \gg \textit{index}$ )&1
5      index = index + 1
6  index = index - 2
7  while index  $\geq 0$ 
8      if Direction[index] == 1
9          result = result.right
10     else
11         result = result.left
12     index = index - 1

```

One way to get **random access** to the element in a complete binary tree with time cost  $\Theta(\log N)$

## Section 4

# Tree 2 Heap

## Subsection 1

# MAXHEAPIFY

# MAXHEAPIFY

MAXHEAPIFY(*H*)

```
1  Left = H.left
2  Right = H.right
3  if Left  $\neq$  NIL and Left.key > H.key
4      largest = Left
5  else
6      largest = H
7  if Right  $\neq$  NIL and Right.key > largest.key
8      largest = Right
9  if largest  $\neq$  H
10     exchange largest.key and H.key
11     MAXHEAPIFY(largest)
```

RIGHT(*H*)=*H.right*  
LEFT(*H*)=*H.left*

## Subsection 2

# BUILD-MAXHEAP-FROMTREE



# BUILD-MAXHEAP-FROMTREE

BUILD-MAXHEAP-FROMTREE( $T$ )

```
1  if  $T == \text{NIL}$ 
2      return
3  BUILD-MAXHEAP-FROMTREE( $T.\text{left}$ )
4  BUILD-MAXHEAP-FROMTREE( $T.\text{right}$ )
5  MAXHEAPIFY( $T$ )
```

## Recurrence

$$T(n) = T(\alpha n) + T((1 - \alpha)n) + \Theta(\log n) \text{ with } 1/3 \leq \alpha \leq 2/3$$

## Solution

$$T(n) = O(n)?$$

## Proof.

- ① Claim that  $T(n) \geq dn$

$$\begin{aligned} T(n) &= T(\alpha n) + T((1 - \alpha)n) + c \log n \\ &\geq d\alpha n + d(1 - \alpha)n + c \log n \\ &= dn + c \log n \\ &\geq dn \end{aligned}$$

- ② Claim that  $T(n) \leq d_1 n - d_2 \log n$

$$\begin{aligned} T(n) &= T(\alpha n) + T((1 - \alpha)n) + c \log n \\ &\leq d_1 n - d_2 \log \alpha n - d_2 \log(1 - \alpha)n + c \log n \\ &= d_1 n - d_2 \log n \\ &\quad - (d_2 \log \alpha + d_2 \log(1 - \alpha) + d_2 \log n - c \log n) \end{aligned}$$

We only needs  $d_2$  to be big enough to overwhelm  $c \log n$

## Subsection 3

# HEAPSORT

# HEAPSORT

HEAPSORT( $H$ )

```
1  BUILD-MAXHEAP-FROMTREE( $H$ )
2  for  $i = H.total$  downto 1
3      exchange  $H.HeapTree.key$  and  $Last.key$ 
4       $A[i] = Last.key$ 
5      if  $H.Last.parent.right == NIL$ 
6           $H.Last.parent.left = NIL$ 
7      else
8           $H.Last.parent.right = NIL$ 
9       $H.total = H.total - 1$ 
10     FINDLAST( $H$ )
11     MAXHEAPIFY( $H.HeapTree$ )
```

## Section 5

# Heap 2 Priority Queue

## Subsection 1

# HEAPMAX and EXTRACT-HEAPMAX

HEAPMAX( $H$ )

**return**  $H.$ HeapTree.key

EXTRACT-HEAPMAX( $H$ )

1   **if**  $H.$ HeapTree == NIL

2         **error** “heap underflow”

3    $max = H.$ HeapTree.key

4    $H.$ HeapTree.key =  $H.$ Last.key

5   **if**  $H.$ Last.parent.right == NIL

6          $H.$ Last.parent.left = NIL

7   **else**

8          $H.$ Last.parent.right = NIL

9    $H.$ Total =  $H.$ Total - 1

10   FINDLAST( $H$ )

11   MAXHEAPIFY( $H.$ HeapTree)

## Subsection 2

# HEAP-INCREASE-KEY and MAXHEAP-INSERT



# HEAP-INCREASE-KEY

HEAP-INCREASE-KEY( $H, i, \text{key}$ )

```
1  target = FIND-NODE( $H, i$ )
2  if  $\text{key} < \text{target.key}$ 
3      error "new key is smaller than current key"
4  while  $\text{target} \neq H.\text{HeapTree}$  and
       $\text{target.parent.key} < \text{target.key}$ 
5      exchange  $\text{target.parent.key}$  and  $\text{target.key}$ 
6       $\text{target} = \text{target.parent}$ 
```

# MAXHEAP-INSERT

MAXHEAP-INSERT( $H$ ,  $key$ )

```
1   $H.Total = H.Total + 1$ 
2   $parent = \text{FIND-NODE}(H, \lfloor H.Total/2 \rfloor)$ 
3  if  $parent.left == \text{NIL}$ 
4       $parent.left = \text{CREATE}(-\infty)$ 
5       $H.Last = parent.left$ 
6  else
7       $parent.right = \text{CREATE}(-\infty)$ 
8       $H.Last = parent.right$ 
9   $\text{HEAP-INCREASE-KEY}(H, H.Last, key)$ 
```

## Section 6

# Easter Egg

## Subsection 1

BUILD-COMplete-BINARY-TREE-FROM-ARRAY

BUILD-COMplete-BINARY-  
TREE-FROM-ARRAY( $T$ ,  $A$ ,  
 $n$ )

```
1  if  $n == 0$ 
2       $T = \text{NIL}$ 
3   $T = \text{CREATE\_TNode}(A[0])$ 
4   $\text{ENQUEUE}(Q, T)$ 
5   $\text{index} = 1$ 
6  while  $\text{index} < n$ 
7       $\text{Parent} = \text{DEQUEUE}(Q)$ 
8       $\text{Left} = \text{CREATE\_TNode}(A[\text{index}])$ 
9       $\text{Parent.left} = \text{Left}$ 
10      $\text{index} = \text{index} + 1$ 
11      $\text{ENQUEUE}(Q, \text{Left})$ 
```

```
11  if  $\text{index} < n$ 
12       $\text{Right} =$ 
          $\text{CREATE\_TNode}$ 
          $(A[\text{index}])$ 
13       $\text{Parent.right} = \text{Right}$ 
14       $\text{index} = \text{index} + 1$ 
15       $\text{ENQUEUE}(Q, \text{Right})$ 
```

Loop invariant?

## Subsection 2

# BUILD-MAXHEAP-FROM-ARRAY

BUILD-MAXHEAP-FROM-ARRAY( $H, A, n$ )

- 1 BUILD-COMPLETE-BINARY-TREE-FROM-ARRAY( $H.HeapTree, A, n$ )
- 2 BUILD-MAXHEAP-FROMTREE( $H.HeapTree$ )

$$\Theta(n) + \Theta(n)$$

# A more intuitive

## BUILD-MAXHEAP-FROM-ARRAY

BUILD-MAXHEAP-FROM-ARRAY( $H, A, n$ )

1   **for**  $i = 1$  **to**  $n$

2       HEAP-INSERT( $H, A[i]$ )

Analysis of time cost

$$\sum_{i=1}^n \log i = \log n! = \Theta(n \log n)$$



