University of Zurich

Department of Informatics
Database Technology Group
Prof. Dr. Anton Dignös

Informatics II
Exercise 5

Published date: March 19, 2021
Labs date: Week 5

Goal:

- Implement `heapify(A, i)` and `BuildHeap(A,n)` in C.
- Study priority queue using heap.
- Practise how partition works in quicksort.

Heap and Heapsort

Task 1. This is a follow-up question to the algorithms `Heapify(A, n, i)` and `BuildHeap(A,n)` taught in the lecture. Recall that `A` is an array of n integers. In this task, we are using max-heapify and building a max-heap.

1. Implement `Heapify(A, n, i)` and `BuildHeap(A,n)` in C.
2. In `BuildHeap(A,n)`, why is the loop that starts from $\lfloor n/2 \rfloor$ instead of n sufficient? Explain.

Algorithm: BuildHeap(A, n)

for $i = \lfloor n/2 \rfloor$ *to* 1 **do**
  Heapify(A, i, n)

Task 2. Heap can be used for designing advanced data structures. In this task, we study one of the most popular applications of a heap – *priority queue*. We use the max-heap to construct a (max) prioprity queue.

A ***priority queue*** is a data structure that maintains an array of elements, and each element has a "key" and "priority". This array of elements is a heap in terms of the "priority" values, and both values of key and priority are integers. As the struct has not been taught, we store values of key and priority of the same element in the same position of two arrays: `int priority[]` and `int key[]`, and we always keep positions for key and prioprity of an element the same. You can think of an element as a bundle of two values of the same position from `priority[]` and `key[]`. Differences in using "index" and "position" in C and pseudocode, see arrays of the cheat sheet.

A prioprity queue supports the MAXIMUM, EXTRACT-MAX, INCREASE-PRIORITY and INSERT functions. Next, we study how to implement these functions.

`priority`	15	13	9	5	12
`key`	5	2	1	4	6

Table 1: The priority queue with 5 elements. The first element has the priority = `priority[1]` = 15 and key = `key[1]` = 5. This array of elements is a prioprity queue because `priority[]` is a heap.

1. MAXIMUM(`int priority[]`, `int key[]`) returns the key of the element with the largest prioprity.

   Algorithm: MAXIMUM(`int priority[]`, `int key[]`)
   return ?

   (a) Take the priority queue in Table 1 as input, what does MAXIMUM function return?

   (b) Complete MAXIMUM function by filling in the "?".

   (c) What's the time commplexity? Explain.

2. EXTRACT-MAX(`int priority[]`, `int key[]`, `int n`) removes and returns the element with the largest priority.

   Algorithm: EXTRACT-MAX(`int priority[]`, `int key[]`, int n)
   if $n < 1$ **then**
   └ **error** "No element to be removed."
   max = ?
   priority[?] = priority[n]
   key[?] = key[n]
   $n = n - 1$
   heapify(priority, key, ?, ?) `// max-heapify`
   `/* key and priority of the same element have the same position after heapify. */`
   return max

   (a) Take the priority queue in Table 1 as input where n = 5, illustrate how values of `priority[]` and `key[]` change in the function EXTRACT-MAX.

   (b) Complete EXTRACT-MAX function by filling in the "?".

   (c) What's the time commplexity? Explain.

3. INCREASE-PRIORITY(`int priority[]`, `int key[]`, `int i`, `int p`) increases the priority of element at position i to p.

   We first update the priority of the element at position i. Increasing the priority of the element at position i might violate that its priority value is larger than its parent's prioprity, if so, we swap the element at position i and its parent. To guarantee the correctness of the heap, we might need to change the positions several elements.

Algorithm: INCREASE-PRIORITY(`int priority[]`, `int key[]`, `int i`, `int p`)

```
priority[i] = p
parent = Parent(i)
while i ≠ 0 & priority[i] > priority[parent] do
    swap(priority[i], priority[parent])
    swap(key[i], key[parent])
    i = ?
    parent = ?
```

(a) Set the priority queue in Table 1 as input; i = 5; p = 16. We increase priority of element at position i = 5 to p = 16. Illustrate how values of `priority[]` and `key[]` change in the function INCREASE-PRIORITY.

(b) Complete INCREASE-PRIORITY function by filling in the "?".

(c) What's the time commplexity? Explain.

4. INSERT(`int priority[]`, `int key[]`, `int n`, `int k`, `int p`) inserts an element with key = k and priority = p into the prioprity queue with n elements.

   We first put the inserted element in the end of the priority queue and temporially set the priority of this element to $-\infty$. Then it calls INCREASE-PRIORITY to set the priority of inserted element to p and maintain the max-heap property.

Algorithm: INSERT(`int priority[]`, `int key[]`, `int n`, `int k`, `int p`)

```
n = n + 1
priority[?] = −∞
key[?] = k
INCREASE-PRIORITY(priority, key, n, p)
```

(a) Set the priority queue in Table 1 as input; insert an element with key = 3 and priority = 14. Illustrate how values of `priority[]` and `key[]` change in the function INSERT

(b) Complete INSERT function by filling in the "?".

(c) What's the time commplexity? Explain.

Assume that we are maintaining a min priority queue using a min-heap. Consider the following functions.

a) Write pseudocode for the MINIMUM function.

b) Write pseudocode for EXTRACT-MIN function.

c) Write pseudocode for DECREASE-PRIORITY function.

d) Write pseudocode for INSERT function.

Quicksort

Task 3. The key element of the quicksort algorithm is its partitioning procedure.

1. Complete the boxes below to complete algorithm Partition(A, l, r) that partitions array $A[l..r]$.

Algorithm: Partition(A,l,r)

x = A[l];
i = l;
for $j =$ [ ] **to** [ ] **do**
  if *A[j]* [ ] x **then**
    i = [ ];
    exchange A[i] and A[j];
exchange [ ] and [ ];
return [ ];

2. Implement `Partition(A,l,r)` in C.

Task 4. Given an array `A[]` of n integers, find the k-th biggest number of `A`.
Example 1:

Input: $[3, 2, 1, 5, 6, 4]$ and $k = 2$
Output: 5

Example 2:

Input: $[3, 2, 3, 1, 2, 4, 5, 5, 6]$ and $k = 4$
Output: 4

Provide the solution using Partition function in Exercise 3 and implement the solution in C.
Hint: think of how we do partition in quick sort.