

Homework 02: Binary Heaps

Roberto Corti

April 25, 2020

Exercise 1

Implement the array-based representation of binary heap together with the functions HEAPMIN, REMOVE MIN, HEAPIFY, BUILD HEAP, DECREASE KEY and INSERT VALUE.

The array-based representation of a binary heap is done in this folder. This version has been written in C language. First of all I defined `total_order_type` in order to make an object able to express an ordering criterion inside the heap. Then, I create a struct `binheap_type` in which it is stored:

- `void* A`: a void pointer that points array used to store heaps nodes.
- `unsigned int num_of_elem`: an integer used to store the number of nodes.
- `unsigned int max_size`: used to store the maximum number of nodes.
- `size_t key_size`: used to store the size of the data type of the node.
- `total_order_type leq`: a `total_order_type` for the ordering criterion of the heap.
- `void* max_order_value`: used to store the maximum value for a node.

The requested functions are implemented in `binheap.c` file.

Exercise 2

Implement an iterative version of HEAPIFY.

An iterative version of `HEAPIFY` has been implemented in `binheap.c`. The algorithm follows this pseudo-code:

Algorithm 1 HEAPIFY(*node*)

```
destination_node  $\leftarrow$  node
do
  node  $\leftarrow$  destination_node
  child  $\leftarrow$  RIGHT CHILD(node)
  if ADDR(node)  $\preceq$  ADDR(child) then
    destination_node  $\leftarrow$  child
  end if
  child  $\leftarrow$  LEFT CHILD(node)
  if ADDR(node)  $\preceq$  ADDR(child) then
    destination_node  $\leftarrow$  child
  end if
  if destination_node  $\neq$  node then
    SWAP(destination_node, node)
  end if
while destination_node  $\neq$  node
```

Exercise 3

Test the implementation on a set of instances of the problem and evaluate the execution time.

In order to test the implementation I create two test codes: `test_insert.c`, `heap_test.c`. The first code does a benchmark on the `insert_key` increase the size by progressively inserting nodes. The asymptotic behaviour of this function is represented by the following plot.

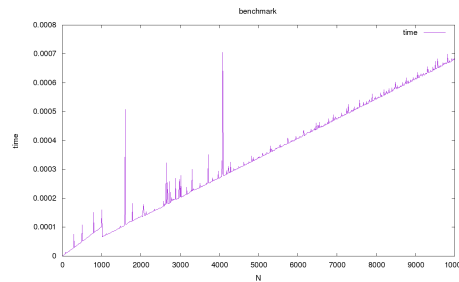


Figure 1: Benchmark of the `insert_key` function.

On the other hand, `heap_test.c` does a check of all the implemented functions just performing all the functions written down in `binheap.c`

Exercise 4

Show that, with the array representation, the leaves of a binary heap containing n nodes are indexed by $\lceil n/2 \rceil + 1, \lceil n/2 \rceil + 2, \dots, n$.

If we consider the node at the index $\lceil n/2 \rceil + 1$ and we know that its left child will be at the index

$$\text{LEFT}(\lceil n/2 \rceil + 1) = 2(\lceil n/2 \rceil + 1) > 2(n/2 - 1) = n.$$

The left child is in a position greater than `HEAP_SIZE`, so this node has no children and since a heap is a complete tree up to the leaf level, we conclude that the node $\lceil n/2 \rceil + 1$ is a leaf.

Exercise 5

Show that the worst-case running time of `MAX_HEAPIFY` on a heap of size n is $\Omega(\log_2 n)$. (Hint: For a heap with n nodes, give node values that cause `MAX_HEAPIFY` to be called recursively at every node on a simple path from the root down to a leaf).

Consider the following max-heap in which we would like to represent the worst-case scenario:

$$A[1] = 1, \text{ and } A[i] = 0 \quad \forall i \in 2, \dots, n.$$

Then, in order to conserve the heap property we have to apply `MAX_HEAPIFY` for every level of the tree for pushing the element of value 0 to the leftmost leaf. `MAX_HEAPIFY` will be then called $\log_2 n$ times, so its running time will be $\Theta(\log_2 n)$. Thus, the worst case running time of `MAX_HEAPIFY` is $\Omega(\log_2 n)$.

Exercise 6

Show that there are at most $\lceil n/2^{h+1} \rceil$ nodes of height h in any n -element binary heap.

We know that for any $n > 0$, the number of leaves of nearly complete binary tree is $\lceil n/2 \rceil$. With this result we can prove the thesis by induction.

- Consider the case $h = 0$; the thesis is satisfied since the unique level of the tree will contain $\lceil n/2 \rceil = \lceil n/2^{0+1} \rceil$.
- Consider the thesis true for $h - 1$.
- Let N_h the number of nodes at leaf level of the tree T_h that has height h . If we know consider the tree T_{h-1} which is composed by T_h up to the leaves of T_h . This tree will have $n' = n - \lceil n/2 \rceil$ nodes. Thus, if we consider

N'_{h-1} as the number of nodes at level $h - 1$, this will be equal to N_h . By induction we have

$$N'_{h-1} = N_h = \lceil n'/2^h \rceil = \lceil \lceil n/2 \rceil / 2^h \rceil \leq \lceil n/2^{h+1} \rceil.$$