Priority Queues

Luciana S. Buriol

Priority Queues

- A priority queue is designed for applications in which elements have a priority value (key) and each time we need to select an element from the set, we want to take the one with highest priority
- Data structures: organized as a balanced binary tree (simulated in a vector)
- Priority: min key, or max key
- Min-Heap property: for every node i other than the root $H[parent(i)] \leq H[i]$, that is, the value of a node is always larger then its parent.
- length(H): number of elements in the array.
- heap_size(H): number of elements in the heap stored within the array.
- Heap Operations: check/add/remove/modify values

Heapify-up Procedure

Algorithm `Heapify-up` ($\mathcal{H}$, `i`)

if $(i > 1)$ **then**
    j=parent(i)=$\lfloor i/2 \rfloor$;
    if $(H[i] < H[j])$ **then**
        swap H[i] $\leftrightarrow$ H[j];
        Heapify-up(H,j);
    end if
end if

Figura: Heapify-up procedure.

Heapify-down Procedure

Algorithm `Heapify-down` ($\mathcal{H}$, `i`)

$l \leftarrow 2i$; `//left`
$r \leftarrow 2i + 1$; `//right`
if ($l \leq heap\_size[H]$ and $H[l] < H[i]$) **then**
    $min \leftarrow l$;
else
    $min \leftarrow i$;
end if
if ($r \leq heap\_size[H]$ and $H[r] < H[min]$) **then**
    $min \leftarrow r$;
end if
if ($min \neq i$) **then**
    swap H[i] $\leftrightarrow$ H[min];
    Heapify-down(H,min);
end if

Figura: Heapify-down procedure.

Heap-Extract-Min and Heap-Insert

Algorithm `Heap-Extract-Min` ($\mathcal{H}$)

if ($heap\_size[H] < 1$) **then**

    return -1;   //heap underflow

end if

$min \leftarrow H[1]$;

$H[1] \leftarrow H[heap\_size[H]]$;

$heap\_size[H] \leftarrow heap\_size[H] - 1$;

Heapify-down(H,1);

return min;

Figura: Heap-Extract-Min

Algorithm `Heap-Insert` ($\mathcal{H}, key$)

$heap\_size[H] \leftarrow heap\_size[H] + 1$;

$i \leftarrow heap\_size[H]$;

while ($i > 1$ and $H[Parent(i)] > key$) **do**

    $H[i] \leftarrow H[Parent(i)]$;

    $i \leftarrow Parent(i)$;

end while

$H[i] \leftarrow key$;

Figura: Heap-Extract-Min

Build_Heap

Algorithm `Build_Heap` $(\mathcal{H})$

1: $heap\_size[H] \leftarrow lenght[H]$;
2: **for** $i \leftarrow lenght[H]/2$ downto 1 **do**
3: $\quad Heapify - down(H, i)$
4: **end for**

Figura: Build_Heap

Build_Heap

Algorithm `Build_Heap` $(\mathcal{H})$

1: $heap\_size[H] \leftarrow lenght[H]$;
2: **for** $i \leftarrow lenght[H]/2$ downto 1 **do**
3: $\quad Heapify - down(H, i)$
4: **end for**

Figura: Build_Heap

In an n-element heap there are at most $\lceil \frac{n}{2^{h+1}} \rceil$ nodes of height h.
$\sum_{h=0}^{\log n} \lceil \frac{n}{2^{h+1}} \rceil O(h) = O(n \sum_{h=0}^{\log n} \frac{h}{2^h}) = O(n)$