Mastering Heap

March 15, 2017

Question: What are the minimum and maximum number of elements in a heap of height h?
Answer: Minimum : 2^h , Maximum: $2^{h+1} - 1$

1 Build Max-Heap

We need to know that in a heap data structure the leaves start from (len(A)/2 + 1). Because if you want to get the child of this leaf you would get 2*(len(A) / 2 + 1) = len(A) + 2, which is not possible. Even the last element of the array is the child of the array-index len(A)/2

We also need the procedure MAX-HEAPIFY(A, i)

```
In [57]: def max_heapify(A, i):
             l = 2 * i + 1
             r = 2 * i + 2
             if l >= len(A) and r >= len(A):
                 return
             if l < len(A) and A[l] >= A[i]:
                 largest = l
             else:
                 largest = i
             if r < len(A) and A[r] >= A[largest]:
                 largest = r
             if largest != i:
                 A[largest], A[i] = A[i], A[largest]
                 max_heapify(A, largest)
```

2 Non recursive implementation of max-heapify

```
In [63]: def max_heapify_non_rec(A, i):
             while i < (len(A) / 2):
                 l = 2 * i + 1
                 r = 2 * i + 2
                 if l >= len(A):
                     break
                 if l < len(A) and A[l] >= A[i]:
                     largest = l
                 else:
```

```
                    largest = i
                if r < len(A) and A[r] >= A[largest]:
                    largest = r
                if largest != i:
                    A[largest], A[i] = A[i], A[largest]
                    i = largest
```

```
In [66]: def build_max_heap(A):
             i = len(A) / 2 - 1
             while i >= 0:
                 max_heapify(A, i)
                 i = i - 1
             return A
```

```
In [67]: def build_max_heap_non_rec(A):
             i = len(A) / 2 - 1
             while i >= 0:
                 max_heapify_non_rec(A, i)
                 i = i - 1
             return A
```

```
In [68]: build_max_heap([1, 2, 3, 4, 9, 16, 7])
```

```
Out[68]: [16, 9, 7, 4, 2, 3, 1]
```

```
In [69]: build_max_heap_non_rec([1, 2, 3, 4, 9, 16, 7])
```

```
Out[69]: [16, 9, 7, 4, 2, 3, 1]
```

```
In [60]: def min_heapify(A, i):
             l = 2*i + 1
             r = 2*i + 2
             if l >= len(A) and r >= len(A):
                 return
             if l < len(A) and A[l] <= A[i]:
                 smallest = l
             else:
                 smallest = i
             if r < len(A) and A[r] <= A[smallest]:
                 smallest= r
             if smallest != i:
                 A[smallest], A[i] = A[i], A[smallest]
                 min_heapify(A, smallest)
```

```
In [61]: def build_min_heap(A):
             i = len(A) / 2
             while i >= 0:
                 min_heapify(A, i)
                 i = i - 1
             return A
```

```
In [62]: build_min_heap([12, 3, 4, 5, 2, 1])

Out[62]: [1, 2, 4, 5, 3, 12]
```

Build heap operation takes $\mathcal{O}(n)$

3 Heapsort

```
In [88]: def heapsort_descending(A):
             build_max_heap(A)
             i = len(A) - 1
             while i > 0:
                 # B is an alias for a sub array of A and B doesn't get allocated a new set of m
                 B = A[:i+1]
                 B[1], B[i] = B[i], B[1]
                 i = i - 1
                 max_heapify(B, 1)
             print A
```

```
In [85]: def heapsort(A):
             build_min_heap(A)
             i = len(A) - 1
             while i > 0:
                 B = A[:i+1]
                 B[1], B[i] = B[i], B[1]
                 i = i - 1
                 min_heapify(B, 1)
             print A
```

```
In [86]: heapsort([12, 3, 4, 5, 2, 1])

[1, 2, 4, 5, 3, 12]
```

```
In [87]: heapsort_descending([12, 3, 4, 5, 2, 1])

[12, 5, 4, 3, 2, 1]
```

4 Priority Queue

```
In [ ]:
```