Application Of Heap

PRIORITY QUEUES

> Applications of a heap is priority queue.
> There are two kinds of priority queue
  (1) Max-priority queues.
  (2) Min-priority queues.
> A priority queue is a data structure for maintaining a set 'S' of elements, each with an associated value called a key.
> A max-priority queue supports the following operations.
  (1) INSERT (S, x) inserts the ~~priority~~ element 'x' into the set 'S'. This operation is written as $S \leftarrow S \cup \{x\}$.
  (2) MAXIMUM(S) returns the element of S with the largest key or largest priority.
  (3) EXTRACT-MAX(S) removes and returns the element of with the largest key.
  (4) INCRESE-KEY(S, x, K) increases the value of element x's key to the new value K. i.e. Increses the priority of element x to K.

> one application of max-priority queue is to schedule jobs or programs to execute on a shared computer on the basis of priority.
> when a job is finised its operation, the next highest priority job is selected from those pending jobs using EXTRACT-MAX. This new job is added to the queue for execution using INSERT.
> when a job is executing, at the same time a higher priority job gives interrupt to the executing job, at that time the highest priority job is selected for execution using EXTRACT-MAX. This highest priority job is added to the queue for execution using INSERT.
> A min-priority queue supports different operations like INSERT, MINIMUM, EXTRACT-MIN, and DECREASE-KEY.
> A min priority queue is used in an event driven simulator.

→ To implement max priority queue, the operations are:

(1) HEAP-MAXIMUM (A)

1. return A[1]

The procedure HEAP-MAXIMUM implements the maximum operation in $\Theta(1)$ time.

(2) HEAP-EXTRACT-MAX (A)

1. If heap-size[A] < 1
2. then error "heap underflow"
3. max ← A[1]
4. A[1] ← A[heap-size[A]]
5. heap-size[A] ← heap-size[A] − 1
6. MAX-HEAPIFY (A, 1)
7. return max.

The procedure HEAP-EXTRACT-MAX implements the EXTRACT-MAX operation.

The running time of HEAP-EXTRACT-MAX is O(log n). Because MAX-HEAPIFY takes O(log n)

~~The procedure~~

HEAP-INCREASE-KEY (A, i, key)

1. If key < A[i]
2. then error "new key is smaller than current key".
3. A[i] ← key
4. while i > 1 and A[PARENT(i)] < A[i]
5. do exchange A[i] ↔ A[PARENT(i)]
6. i ← PARENT(i)

The procedure HEAP-INCREASE-KEY implements the INCREASE-KEY operation.

The running time of HEAP-INCREASE-KEY on an n-element heap is O(log n) because the path traced from node i to root has length O(log n)

MAX-HEAP-INSERT (A, key)

1. heap-size[A] ← heap-size[A] + 1
2. A[heap-size[A]] ← −∞
3. HEAP-INCREASE-KEY (A, heap-size[A], key)

The procedure MAX-HEAP-INSERT implements the INSERT operation.
The procedure takes input the key of the new element to be inserted into max-heap A.
This procedure first expands the max-heap by adding to the tree a new leaf whose key is −∞.
Then it calls HEAP-INCREASE-KEY to set the key of this new node to its correct value and maintain the max-heap property.
The running time of MAX-HEAP-INSERT on an n-element heap is O(log n).