Priority Queues

CS 321

Feb 03 2021

Priority Queues

- A priority queue is an ordered queue such that the highest priority entry is always at the front of the Queue.
  - Insert(S,x) – insert x into S, x is our ‘key’, ‘key’=priority.
  - Maximum(S) – return maximum key in S.
  - Extract-Max(S) – remove and return largest key.
  - Increase-Key(S,x,k) increase key x to k.
- In a full implementation x could be a structure x={key, element}.
- What might be a good data structure for this?

Max-Priority-Queue

- A Max-Heap seems an excellent choice for our Priority Queue since it keeps largest item at the root.

Heap-Maximum(A)

- Return A[1];

Heap-Extract-Max(A)

```
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]--
6. Max-Heapify(A, 1)
7. return max
```

Heap-Increase-Key(A, i, key)

```
Heap-Increase-Key(A, i, key)
1. if key < A[i]
2.    then error -- new key must be larger than current key
3. A[i] <-- key
4. while i > 1 and A[Parent(i)] < A[i]        // This procedure takes O(log n)
5.       do exchange A[i] <--> A[Parent(i)]
6.          i <-- Parent(i)
```

Max-Heap-Insert(A, key)

```
Max-Heap-Insert(A, key)
1. heap-size[A]++
2. A[heap-size[A]] <-- negative infinity
3. Heap-Increase-Key(A, heap-size[A], key)
```

Summarizing Table

Method	Worst Case	Best Case
Maximum	O(1)	O(1)
Max-Heap-Insert	O(logn) = key at leaf.	O(1) = key at root.
Heap-Increase-Key	O(logn) = key at leaf.	O(1) = key at root.
Extract-Max	O(1)	O(1)

Exercise 6.5-7

- How could we implement a first in, first out Queue with our Max-Priority-Queue?
- How could we implement a stack with our Max-Priority-Queue?
- Hint: answer is in proper selection of the keys.

Alternative Sort

- Have a list of numbers limited in range to between 0 and 2^16
- Numbers could be in any order.
- Input size could be any size (very large).
- How could we sort in Θ(n) time?