Design and Analysis of Algorithms I

Data Structures

Heaps and Their Applications

Heap: Supported Operations

- A container for objects that have keys
- Employer records, network edges, events, etc.

Insert: add a new object to a heap.

Running time : O(log(n))

Equally well,
EXTRACT MAX

Extract-Min: remove an object in heap with a minimum key value. [ties broken arbitrarily]

Running time : O(log n) [n = # of objects in heap]

Also : HEAPIFY (n batched Inserts in O(n) time), DELETE(O(log(n)) time)

Application: Sorting

Canonical use of heap : fast way to do repeated minimum computations.

Example : SelectionSort ~$\theta(n)$ linear scans, $\theta(n^2)$ runtime on array of length n

Heap Sort : 1.) insert all n array elements into a heap

2.) Extract-Min to pluck out elements in sorted order

Running Time = 2n heap operations = O(nlog(n)) time.

=> optimal for a "comparison-based" sorting algorithm!

Application: Event Manager

"Priority Queue" – synonym for a heap.

Example : simulation (e.g., for a video game)

-Objects = event records [Action/update to occur at given time in the future]

- Key = time event scheduled to occur
- Extract-Min => yields the next scheduled event

Application: Median Maintenence

I give you : a sequence x1,…,xn of numbers, one-by-one.

You tell me : at each time step i, the median of {x1,….,xi}.

Constraint : use O(log(i)) time at each step i.

Solution : maintain heaps H_{Low} : supports Extract Max
H_{High} : supports Extract Min

Key Idea : maintain invariant that ~ i/2 smallest (largest) elements in H_{Low} (H_{High})

You Check : 1.) can maintain invariant with O(log(i)) work
2.) given invariant, can compute median in O(log(i)) work

Tim Roughgarden

Application: Speeding Up Dijkstra

<u>Dijkstra's Shortest-Path Algorithm</u>

-Naïve implementation => runtime = $\theta(nm)$

- with heaps => runtime = O(m log(n))

vertices

edges

loop iteratios

Work per iteration [linear scan through edges for minimum computation]