CSE 326
Lecture 16

Priority Queues and Binary Heaps

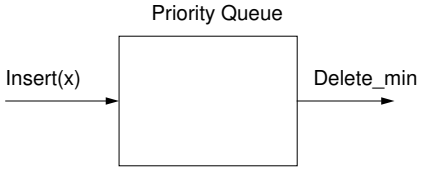
Winter Quarter 1999

Priority Queues

- Priority queues support the two operations

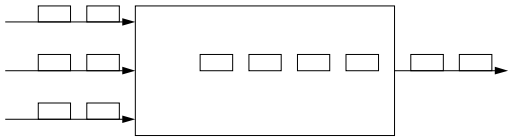
  1. Insert(x) which inserts the key (number x) into the priority queue

  2. Delete_min which removes the smallest item in the priority queue and returns it.



Applications of Priority Queues

- Queues with inherent priorities
  - Run-time queue in an operating system
  - Printer queue
  - Network multiplexer



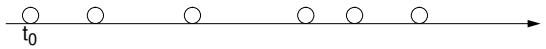
Discrete Event Simulation

- An event is a pair (x,t) where x is a description of the event and t is the time it is supposed to occur.
- A discrete event simulator (DES) maintains a set S of events which it intends to simulate in time order.
- Simulation engine of a DES:

  repeat

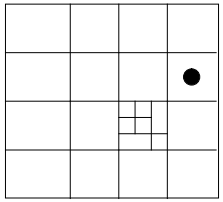
  Find and remove the event (x,t_0) from S such that t_0 is minimal;
  Do whatever x says to do, in the process new events (x_1,t_1), $(x_2,t_2),...,(x_k,t_k)$ may be generated;
  Insert the new events into S.

  until the simulation has gone long enough or S is empty



The N-Body Problem in Astrophysics

- We have millions of stars clustered in galaxies and we wish to see what happens when the galaxies collide or get close to each other.
- If there are N bodies then there are N^2 gravitational interactions. To simulate one time increment would take far too long.
- Put the stars into a oct-tree or 3-d tree. A node of the tree represents a cluster of stars. Such a cluster can be approximated by its center of mass.



Continuing N-Body Simulation

- Each node n has an error $\varepsilon(n)$ associated with it. $\varepsilon(n)$ is the error in approximating the stars in the cluster by its center of mass.
- Suppose we have some acceptable error δ for one step of the simulation. We maintain a set of nodes (clusters) C whose sum of errors is ε. Initially C contains only the root and $\varepsilon = \varepsilon(\text{root})$.
- Cluster Finding

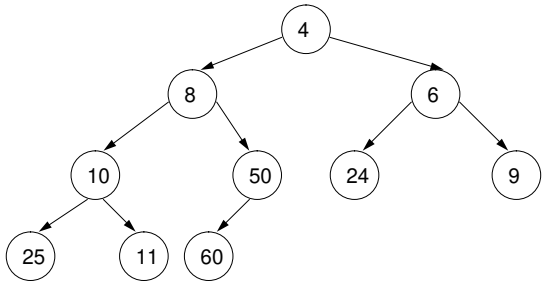
  while $\varepsilon > \delta$ do

  Find and remove the member n of C such that $\varepsilon(n)$ is maximal;
  Insert $n_1, n_2, \ldots, n_k$ the children of n which have positive mass, into C;
  $\varepsilon := \varepsilon - \varepsilon(n) + \varepsilon(n_1) + \ldots + \varepsilon(n_k)$

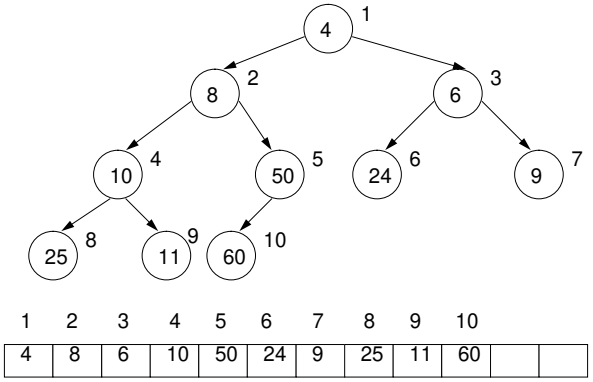
  endwhile

The Heap Data Structure

- A binary heap is a complete binary tree with the "heap property".
- Heap property: The key of the parent is smaller than the keys of its children.



Storage of a Complete Binary Tree

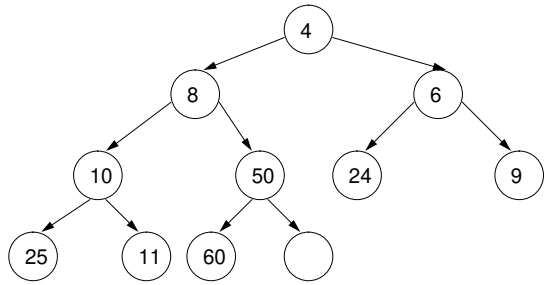


Finding Children and Parent

- We assume that there are n items in the priority queue and an array H[1..m] where m ≥ n.
- The root of the heap is in position 1.
- Children of i are in 2i and 2i+1 as long as 2i ≤ n and 2i+1 ≤ n.
- Parent of i is in i/2 as long as i > 1.

Insertion

- Suppose we want to insert the item 5. "Percolate Up"



Insertion Code

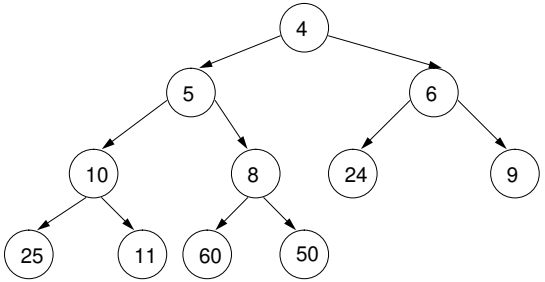
- Insert(x) inserts into the heap H.
  n is the number of keys in the heap, and m is the size of the heap array
- Insert(x: key) {

```
i: local integer;
if n = m then "heap is full"
else {
    n := n + 1;
    i := n;
    while not(i/2=0) and H[i/2] > x do
        H[i] := H[i/2];
        i := i/2
    endwhile
    H[i] := x }}
```

"Percolate Up"

Delete_Min

- "Percolate Down"



Delete_min Code

- Delete_min removes and returns the minimum in the heap and reheapifies. Again, n is the number of keys in the heap.
- Delete_min {

```
i,j : local integer; x,y: local key;
if n = 0 then "heap is empty"
else {
   i :=1;
  y := H[1]
  x := H[n];
  n := n-1;
  while 2i ≤ n do
     if 2i + 1 ≤ n and H[2i+1] < H[2i] then j := 2i+1
               else j := 2i;
     if H[j] < x then  { H[i] := H[j]; i := j }
               else break
   endwhile
   H[i] := x }
return(y)}
```

"Percolate Down"

Performance of the Binary Heap

- Insert

  - worst case O(log n)
  - average case O(1). The average insert percolates up 1.6 levels

- Delete_min

  - worst case O(log n)
  - average case also O(log n). The average delete-min percolates to the bottom of the tree.

Other Heap Operations

- Decrease(i,d) decreases the value in H[i] by d and returns the new index of the item H[i].

  Simply decrease H[i] by d and percolate up from position i.

- Increase(i,d) increases the value of H[i] by d and returns the new index of the item H[i].

  Simply increase H[i] by d and percolate down from position i.

- Delete(i) deletes the item H[i] from the heap.

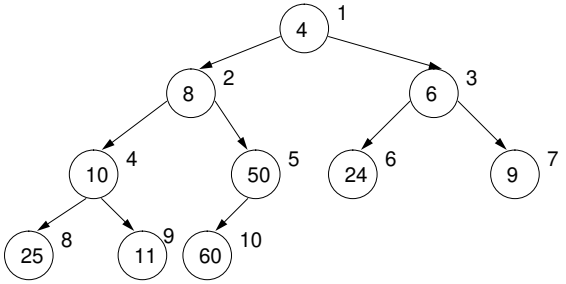
  First Decrease(i,infinity), then Delete_min.

- Build_heap builds a heap from scratch assuming items are stored in H[1..n] (not necessarily in heap order).

  Starting with i = n/2 and working down to 1 just percolate_down(i). Total time is O(n). This is called Floyd's Method.

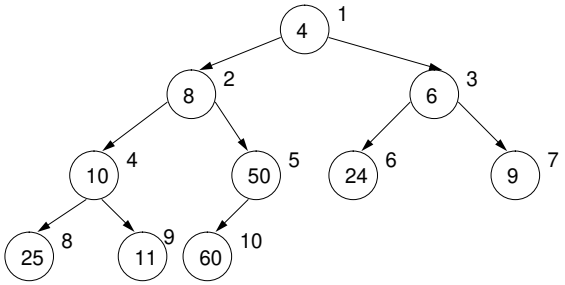
Decrease(i,d)

- Decrease(8,22)



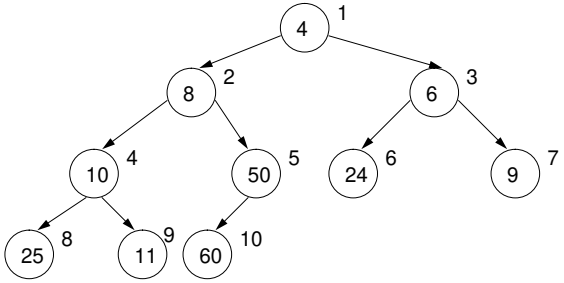
Increase(i,d)

- Increase(2,50)

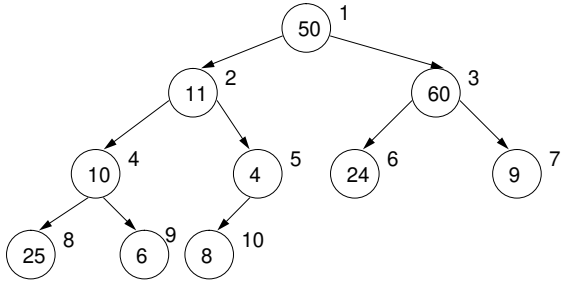


Delete(i)

- Delete(4)



Build_heap (Floyd's Method)



1	2	3	4	5	6	7	8	9	10
50	11	60	10	4	24	9	25	6	8

Complexity of Build_Heap

- At most n/4 percolate down 1 level
  at most n/8 percolate down 2 levels
  at most n/16 percolate down 3 levels

$$Total\ Time = 1 \cdot \frac{n}{4} + 2 \cdot \frac{n}{8} + 3 \cdot \frac{n}{16} + \ldots = \sum_{i=1}^{\log n} i \cdot \frac{n}{2^{i+1}} = \frac{n}{2} \cdot \sum_{i=1}^{\log n} \frac{i}{2^i} \le n$$

$$\sum_{i=0}^{\infty} x^i = \frac{1}{1-x}$$

$$\sum_{i=0}^{\infty} i \cdot x^{i-1} = \frac{d}{dx} \sum_{i=0}^{\infty} x^i = \frac{1}{(1-x)^2}$$

$$\sum_{i=1}^{\log n} \frac{i}{2^i} \le \sum_{i=1}^{\infty} \frac{i}{2^i} = \frac{1}{2} \cdot \sum_{i=1}^{\infty} \frac{i}{2^{i-1}} = \frac{1/2}{(1-1/2)^2} = 2$$

Heap Sort

- Suppose we have an unordered array A[1..N] which we wish to sort.

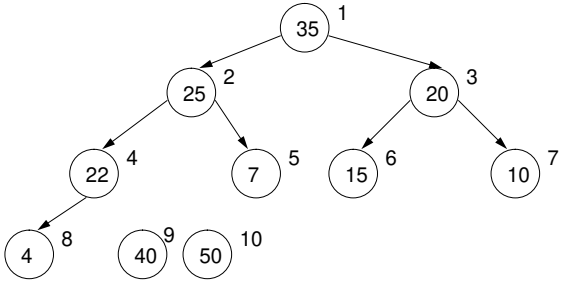
  1. Build a max heap. (The largest element is in A[1].)

  2. For i = 1 to N - 1 do
     x := delete_max;
     A[N-i+1] := x
     endfor

	1	2	3	4	5	6	7	8	9	10
After Build_heap	50	40	20	25	35	15	10	22	4	7
	40	35	20	25	7	15	10	22	4	50
	35	25	20	22	7	15	10	4	40	50

Heap Sort Step

1	2	3	4	5	6	7	8	9	10
35	25	20	22	7	15	10	4	40	50



25	22	20	4	7	15	10	35	40	50

Properties of Heap Sort

- Worst case time complexity O(n log n)

  Build_heap O(n)

  n Delete-max's for O(n log n)

- In-place sort - only constant storage beyond the array is needed.