CSE 326 Data Structures

CSE 326 : Dave Bacon

Priority Queues

Logistics

- AA 9:30 section moved to CSE 203
- Project 1 – Reverse a sound file
  - Due Wed January 10, 2007 at electronically at midnight 11:59.00000
  - Problem 5 now has two parts
  - Hard copy handed in Thursday in Section
- Homework 1
  - Due Fri January 12, 2007 at beginning of lecture 4.1
- Reading (assume you finished Chapter 1,2,3)
  - Chapter 6 : Priority Queues [next lecture]

Big-O and Friends Notation

- $2^{n/3} = O(2^n)$ means…

$\exists c, n_0$ such that $2^{n/3} \le c 2^n$ for $n \ge n_0$

- $2^{n/3} = 2^{O(n)}$ means….

$\exists c, n_0$ such that $2^{n/3} \le 2^{cn}$ $n \ge n_0$

One Final Analysis Problem

- Consider the following program segment:

```
x:= 0;
for i = 1 to N do
  for j = 1 to i do
    x := x + 1;
```

$x := x + i$

- What is the value of x at the end?

$$x = \sum_{i=1}^{N} i$$

$$\frac{N}{2}(N+1)$$

$$\left.\begin{array}{c} N+1 \\ N+1 \\ \vdots \\ + \end{array}\right\} \frac{N}{2}$$

$$\underline{1} + \underline{2} + 3 + \cdots + (N-2) + \underline{(N-1)} + \underline{N}$$

Induction

- Prove by induction $1+2+3+\ldots=\sum_{i=1}^{N} i=\frac{N(N+1)}{2}$

Base case $N=1$

LHS $\sum_{i=1}^{1} i = 1$ RHS $\left.\frac{N(N+1)}{2}\right|_{N=1} = 1$

Inductive step

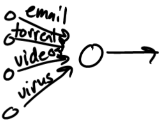
True for $N-1$

$$\sum_{i=1}^{N-1} i = \frac{(N-1)N}{2} \qquad \sum_{i=1}^{N} i = \sum_{i=1}^{N-1} i + N = \frac{(N-1)N}{2} + N$$

$$\frac{N(N+1)}{2}$$

$$\parallel$$

A Bandwidth Problem

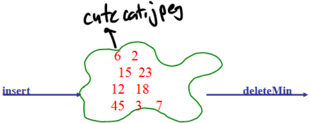


video	high
virus	low
torrent	mid
email	high

FIFO Queue?

Priority Queue ADT

- Checkout line at the supermarket ???
- Printer queues ???
- operations: insert, deleteMin



Priority Queue ADT

1. **PQueue <u>data</u>** : collection of data with priority

2. **PQueue <u>operations</u>**
   - insert
   - deleteMin

   (also: create, destroy, is_empty)

3. **PQueue <u>property</u>**: for two elements in the queue, *x* and *y*, if *x* has a **<u>lower</u>** priority value than *y*, *x* will be deleted before *y*

Applications of the Priority Q

- Select print jobs in order of decreasing length
- Forward packets on network routers in order of urgency
- Select most frequent symbols for compression
- Sort numbers, picking minimum first

- **Anything *greedy***

Implementations of Priority Queue ADT

	insert	deleteMin
Unsorted list (Array)	O(1) [O(N)]	O(N)
Unsorted list (Linked-List)	O(1)	O(N)
Sorted list (Array)	O(N)	O(1)
Sorted list (Linked-List)	O(N)	O(1)
Binary Search Tree (BST)	~O(N)	~O(N)
Binary Heap	O(logN) O(1) "on average"	O(logN)

Tree Review

root(**T**): A

leaves(**T**): DEF JKLMN I

children(B): DEF

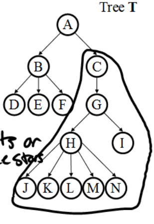
parent(H): G

siblings(E): DF

ancestors(F): AB Its parents or parents ancestors

descendents(G): HIJKLMN

subtree(C): node+descendents



More Tree Terminology

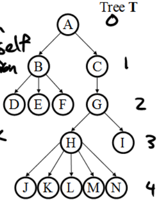
B =1

depth(**T**): # edges on path from root to self

height(G): # edges on longest from node to leaves
=2

degree(B): # children
=3

branching factor(**T**): max degree
=5



Some More Tree Terminology

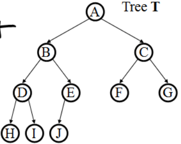
T is *binary* if …

Each node has at most 2 children

T is *n-ary* if …

" "

2 → n



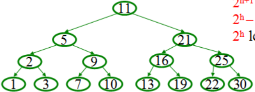
T is *complete* if …

Each row is filled left to right

How deep is a complete tree with *n* nodes?

Brief interlude: Some Definitions:

A ***Perfect*** binary tree – A binary tree with all leaf nodes at the same depth. All internal nodes have 2 children.



height h

$2^{h+1} - 1$ nodes

$2^h - 1$ non-leaves

2^h leaves

Full Binary Tree

- A binary tree in which each node has ***exactly zero or two children***.
- (also known as a proper binary tree)
- (we will use this later for Huffman trees)

Binary Heap Properties

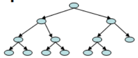
1. Structure Property
2. Ordering Property

Heap **Structure** Property

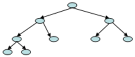
- A binary heap is a ***complete*** binary tree.

Complete binary tree – binary tree that is completely filled, with the possible exception of the bottom level, which is filled left to right.

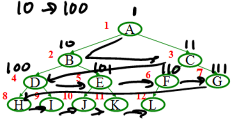
Examples:



Regular



Representing Complete Binary Trees in an Array



From node **i**:

left child: $2i$

right child: $2i+1$

parent: $\lfloor i/2 \rfloor$

implicit (array) implementation:

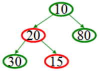
	A	B	C	D	E	F	G	H	I	J	K	L	
0	1	2	3	4	5	6	7	8	9	10	11	12	13

Why better than tree with pointers?

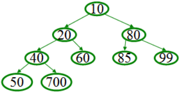
1. Less Space
2. *2, /2, +1 fast
3. Fast insert
4 Parent E2

Heap **Order** Property

Heap order property: For every non-root node X, the value in the parent of X is less than (or equal to) the value in X.

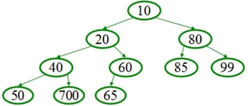


not a heap



Heap Operations

- findMin:
- insert(val): percolate up.
- deleteMin: percolate down.

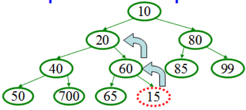


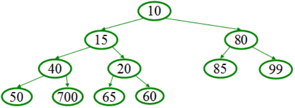
Heap – Insert(val)

Basic Idea:

1. Put val at "next" leaf position
2. Repeatedly exchange node with its parent if needed

Insert: percolate up



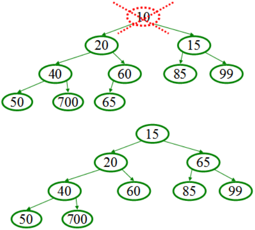


Heap – Deletemin

Basic Idea:

1. Remove root (that is always the min!)
2. Put “last” leaf node at root
3. Find smallest child of node
4. Swap node with its smallest child if needed.
5. Repeat steps 3 & 4 until no swaps needed.

DeleteMin: percolate down



Insert: 16, 32, 4, 69, 105, 43, 2

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**

Other Priority Queue Operations

- **decreaseKey**
  - given a pointer to an object in the queue, reduce its priority value

  Solution: change priority and ________________________

- **increaseKey**
  - given a pointer to an object in the queue, increase its priority value

Why do we need a *pointer*? Why not simply data value?

Solution: change priority and _________________________

Other Heap Operations

decreaseKey(objPtr, amount): raise the priority of a object, percolate up

increaseKey(objPtr, amount): lower the priority of a object, percolate down

remove(objPtr): remove a object, move to top, them delete. 1) decreaseKey(**objPtr**, ∞)

2) deleteMin()

Worst case Running time for all of these:

FindMax?

ExpandHeap – when heap fills, copy into new space.

Binary Min Heaps (summary)

- **insert**: percolate up. O(log N) time.
- **deleteMin**: percolate down. O(log N) time.

- **Next time**: Even more priority queues??