CMSC 341
Lecture 19

Announcements

Expect graded exams on Wed

Truncated office hours today (until 1:30)

Proj5 up today, example solution later

Constructing a Binary Heap

A BH can be constructed in O(n) time.

Suppose an array in arbitrary order. It can be put in heap order in O(n) time.

- Create the array and store n elements in it in arbitrary order. O(n)
- Heapify the array
  - start at vertex $i = \lfloor n/2 \rfloor$
    - percolateDown(i)
  - repeat for all vertices down to i

BinaryHeap.C (cont)

```
template <class Comparable>
void BinaryHeap<Comparable>::buildHeap() {
   for(int i = currentSize/2; i >0; i--)
        percolateDown(i);
   }
```

Performance of Construction

A CBT has 2^{h-1} vertices on level h-1.

On level h-l, at most 1 swap is needed per node.

On level h-2, at most 2 swaps are needed.

…

On level 0, at most h swaps are needed.

Number of swaps = S

$= 2h*0 + 2^{h-1}*1 + 2^{h-2}*2 + \ldots + 2^{0}*h$

$= \sum_{i=0}^{h} 2^i (h-i) = h\sum_{i=0}^{h} 2^i - \sum_{i=0}^{h} i2^i$

$= h(2^{h+1}-1) - ((h-1)2^{h+1}+2)$

$= 2^{h+1}(h-(h-1))-h-2$

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

Performance of Construction (cont)

But $2^{h+1}-h-2 = O(2^h)$

But $n = 1 + 2 + 4 + \ldots + 2^h = \sum_{i=0}^{h} 2^i$

Therefore, $n = O(2^h)$

So S = O(n)

A heap of n vertices can be built in O(n) time.

Heap Sort

Given n values, can sort in O(n log n) time (in place).

- Insert values into array -- O(n)
- heapify -- O(n)
- repeatedly delete min -- O(lg n) n times

Using a min heap, this code sorts in reverse order. With a max heap, it sorts in normal order.

```
for (i = n-1; i >= 1; i--) {
  x =findMin();
  deleteMin();
  A[i+1] = x;
  }
```

Limitations

Binary heaps support `insert`, `findMin`, `deleteMin`, and `construct` efficiently.

They do not efficiently support the `meld` or `merge` operation in which 2 PQs are merged into one. If P_1 and P_2 are of size n_1 and n_2, then the merge is in $O(n_1 + n_2)$

Leftist Heap

Supports

– findMin -- O(1)
– deleteMin -- O(lg n)
– insert -- O(lg n)
– construct -- O(n)
– merge -- O(lg n)

Leftist Tree

A LT is a binary tree in which at each vertex v, the path length, $d_{r,}$ from v's right child to the nearest non-full vertex is not larger than that from the vertex's left child to the nearest non-full vertex.

An important property of leftist trees:

– At every vertex, the shortest path to a non-full vertex is along the rightmost path.
– Suppose this was not true. Then, at the same vertex the path on the left would be shorter than the path on the right.

Leftist Heap

A *leftist heap* is a leftist tree in which the values in the vertices obey heap order (the tree is partially ordered).

Since a LH is not necessarily a CBT we do not implement it in an array. An explicit tree implementation is used.

Operations

- findMin -- return root value, same as BH
- deleteMin -- done using meld operation
- insert -- done using meld operation
- construct -- done using meld operation

Meld

Algorithm:

```
Meld (H1, H2) {
   if (!root(H1) || (root_value(H1) > root_value(H2) )
         swap (H1, H2)
   if (root(H1) != NULL))
         right(H1) <-- Meld(right(H1),H2)
   if (left_length(H1) < right_length(H1)
         swap(left(H1), right(H1);
   }
```

Meld (cont)

Performance: O(lg n)

- the rightmost path of each tree has at most $\lfloor \lg(n+1) \rfloor$ vertices. So O(lg n) vertices will be involved.

Title:
leftist_meld_1.ps
Creator:
fig2dev Version 3.2 Patchlevel 0-beta2
Preview:
This EPS picture was not saved
with a preview included in it.
Comment:
This EPS picture will print to a
PostScript printer, but not to
other types of printers.

Title:
leftist_meld_2.ps
Creator:
fig2dev Version 3.2 Patchlevel 0-beta2
Preview:
This EPS picture was not saved
with a preview included in it.
Comment:
This EPS picture will print to a
PostScript printer, but not to
other types of printers.

Title:
leftist_meld_3.ps
Creator:
fig2dev Version 3.2 Patchlevel 0-beta2
Preview:
This EPS picture was not saved
with a preview included in it.
Comment:
This EPS picture will print to a
PostScript printer, but not to
other types of printers.

Leftist Heap Operations

Other operations implemented in terms of Meld

- insert (item)
  - make item into a 1-vertex LH, X
  - Meld(*this, X)
- deleteMin
  - Meld(left subtree, right subtree)
- construct from N items
  - make N LH from the N values, one element in each
  - meld each in
    - one at a time :
    - use queue and build pairwise :

LH Construct

Algorithm:

- make N heaps each with one data value
- Queue Q;
- for (I=1; I <= N; I++)

  Q.Enqueue(Hi);
- Heap H = Q.Dequeue();
- while (!Q.IsEmpty())

  Q.Enqueue(meld(H,Q.Dequeue());

  H = Q.Dequeue();