1.204 Computer Algorithms in Systems Engineering

Spring 2010

Problem Set 3 Solution

Due: 12 noon, Monday March 8, 2010

1. Arlington network

Please see the Java code on the Stellar Web site.

Answers to the questions.

1. If there are two or more arcs between a pair of nodes, this algorithm traverses them correctly. It loops over all arcs out of (or into) a node, regardless of their destination (or origin).

2. If there is a loop arc, the traverse is still correct, for the same reason as above.

3. Adding an arc is O(n). One increases the dimension of the arrays, and iterates through the array to insert the arc in the correct place, updating H, to and cost after the insertion point. This only requires one pass through the data. One does not need to resort.

2. k-ary heaps

1. Array representation:
    a. If the parent node is a[i], the child nodes are from a[k*i +1] to a[k*i + k].
    b. If the child node is a[i], the parent node is a[(i-1)/k] (use integer division).

2. Using the relationship given in the lecture notes, if the height of the heap is h, then
    $h-1 < \log_k n$,
    so the height of a k-ary heap is $O(\log_k n)$.

3. You insert the element as a leaf in the heap. You would compare it with the parent element and swap places with the parent if the inserted element/leaf is greater than the root. You would keep doing until the element is in the right place so that the heap property is satisfied. In the worst case, you would need $\log_k n$ operations. On average, you would need $O(\log_k n)$ operations.

4. The insert operation is simpler in a k-ary heap (k>2) because you would need to compare only 2 elements for moving an element up the heap (towards the root). This is an $O(\log_k n)$ operation.

When you delete an element from a heap, you would move the last leaf to the root. You would then need to find the maximum of k+1 elements, which is more complex than finding the maximum of 2 elements. This is an $O(k \log_k n)$ operation.

5. Challenge question:

The insert operation is faster in a k-ary heap than in a binary heap, but the delete operation is generally slower. Building a heap (heapify) is independent of k. If the operations occur with the same frequency, what is the optimal value of k?

The analysis:

a. Inserting an element into a k-ary heap is faster as k increases.

b. Deleting the maximum element from the top of the heap takes $O(k \log_k n)$ time. What is the optimal k for deletion?

$\log_k n = (\lg n / \lg k)$

$k \log_k n = (k/ \lg k)(\lg n)$

The minimum of $(k/ \lg k)$ occurs when $k= e$, by differentiating $(k/ \lg k)$

To compare a k=2 (binary heap) with k=3 (3-ary heap), we find:

$(3/ \lg 3)/ (2/ \lg 2)= 0.95$

c. In a 3-heap, deletion is 5% faster than a binary heap. Insertion is $(\lg 2/ \lg 3)$ the time, or about 37% faster.

d. In a 4-heap, deletion takes $(4/\lg 4)(2/\lg 2)$ the time of a binary heap; this expression equals 1, so the time is the same. Insertion is $(\lg 2/\lg 4)$ or 50% faster. At larger k values, deletion is slower than a binary heap, and insertion speeds up less and less.

We can make a table of running times relative to a binary heap:

Deletion takes $(k/ \lg k)/(2/ \lg 2)$ time. Insertion takes $(\lg 2/\lg k)$ time.

k	Deletion	Insertion	Average (assuming equal frequency)
3	0.95	0.63	0.79
4	1.00	0.50	0.75
5	1.08	0.43	0.75
6	1.16	0.39	0.78

…

1.204 Computer Algorithms in Systems Engineering

Spring 2010