CSE 5311 Notes 3: Amortized Analysis

(Last updated 5/27/13 8:52 AM)

PROBLEM: Worst case for a *single* operation is too pessimistic for analyzing a *sequence* of operations.

ELEMENTARY EXAMPLES

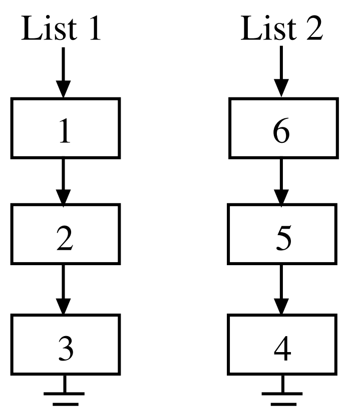
1. Stack operations with "multiple pop"

   Usual push for a single entry - $\theta(1)$

   Multi-pop for k entries - $\theta(k)$

   Sequence of n operations takes $\theta(n)$ time.

2. Queue implemented with two lists/stacks (in a functional language)



Enqueue: At head of list 2

```
Dequeue:  if list 1 is empty
              while list 2 not empty
                  Remove head of list 2
                  Insert as head of list 1
          Remove head of list 1
```

Application to maximum message length (see end of CSE 2320 Notes 10)

3. Incrementing a counter repetitively by 1 (CLRS, p. 461)

```
0 0 0 1 1 1 1
          + 1
-------------
```

ANALYSIS

1. Aggregate Method

$$\frac{\sum \text{actual cost}}{\text{\# of operations}} = \frac{\sum c_i}{n} = \hat{c}_i = \text{amortized cost}$$

2. Accounting Method - For any sequence $\sum_{i=1}^{n} \hat{c}_i \geq \sum_{i=1}^{n} c_i$

   Charge more for *early* operations in sequence to pay for *later* operations.

   Consider queue with 2 lists:

   Each item is touched 3 times

   Charge 2 for enqueue

   Charge 1 for dequeue

   Each item in list 2 has a *credit* of 1. Credit is consumed in dequeue with empty list 1.

3. Potential Method - Preferred method

   Concept:

   Generalizes accounting method.

   Tedious for initial designer, but hides details for others.

   Map entire state of data structure to a *potential*. Captures "difficulty" of future operations.

   Assuming a sequence of operations:

   c_i = actual cost of ith operation
   $\hat{c}_i$ = amortized cost of ith operation
   D_{i-1} = data structure state before ith operation
   D_i = data structure state after ith operation

   $$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$$

   Total amortized cost for a sequence is:

   $$\sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n} \left(c_i + \Phi(D_i) - \Phi(D_{i-1})\right) = \sum_{i=1}^{n} c_i + \Phi(D_n) - \Phi(D_0)$$

   Book: Multipop stack (Φ = # of items on stack)

   Binary counter (Φ = # of ones)

   Defining Φ is the hard part.

BINARY TREE TRAVERSALS - Slightly more involved than previous examples

Observation: Tree traversal on tree with n nodes requires $2n$ - 2 edges "touches"

Operations: INIT: Finds first node in traversal

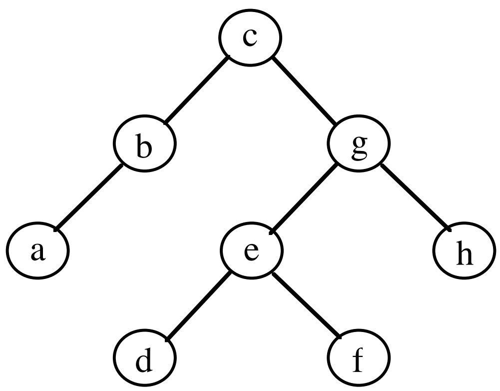
$\hat{c}_1 = 0$

SUCC(x): Finds successor of x

$\hat{c}_i = 2$ for $2 \le i \le n$ (n exits tree)

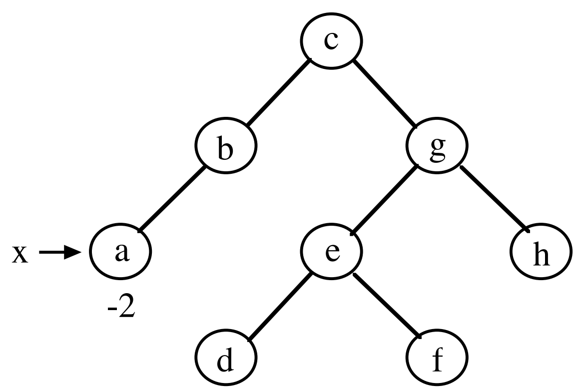
Need Φ for inorder, postorder, and preorder

Example:



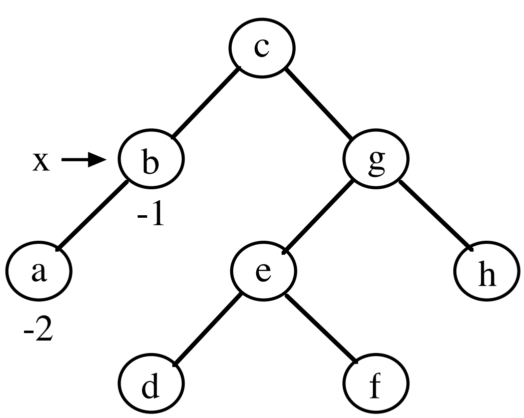
For INIT for *inorder*, must stop at a.

$$c_1 = 2, \hat{c}_1 = 0, \text{and } \hat{c}_1 = c_1 + \Phi(D_1) - \Phi(D_0)$$
$$0 = 2 + \Phi(D_1) - 0$$



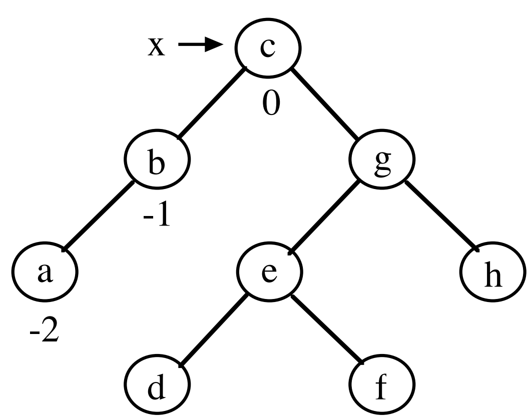
SUCC(a) = b

$$c_2 = 1, \hat{c}_2 = 2, \text{and } \hat{c}_2 = c_2 + \Phi(D_2) - \Phi(D_1)$$
$$2 = 1 + \Phi(D_2) - (-2)$$



SUCC(b) = c

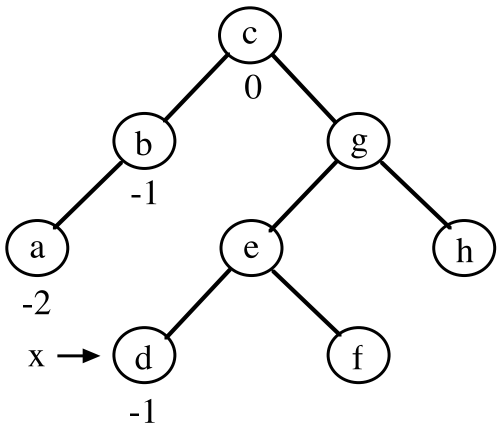
$$c_3 = 1, \hat{c}_3 = 2, \text{and } \hat{c}_3 = c_3 + \Phi(D_3) - \Phi(D_2)$$
$$2 = 1 + \Phi(D_3) - (-1)$$



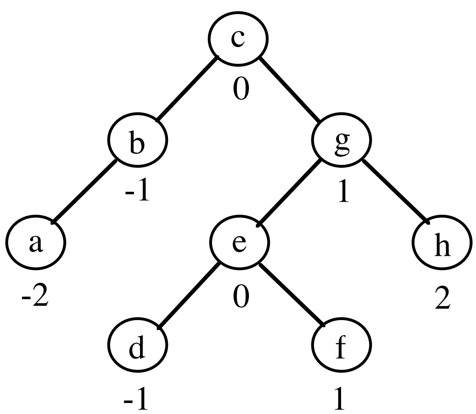
SUCC(c) = d

$c_4 = 3, \hat{c}_4 = 2$, and $\hat{c}_4 = c_4 + \Phi(D_4) - \Phi(D_3)$

$$2 = 3 + \Phi(D_4) - 0$$



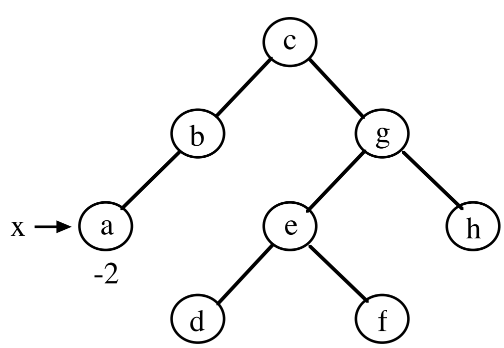
In general: rank, r(x), of node x is r(root) = 0, r(x→left) = r(x) - 1, r(x→right) = r(x) + 1



For INIT for *postorder*, must stop at a.

$c_1 = 2, \hat{c}_1 = 0$, and $\hat{c}_1 = c_1 + \Phi(D_1) - \Phi(D_0)$

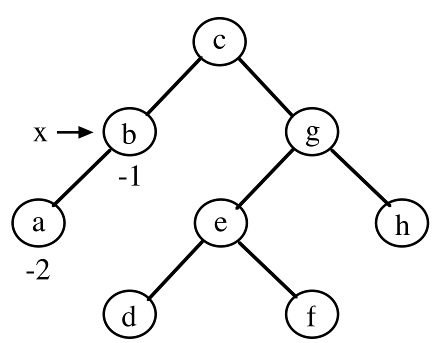
$$0 = 2 + \Phi(D_1) - 0$$



SUCC(a) = b

$c_2 = 1, \hat{c}_2 = 2$, and $\hat{c}_2 = c_2 + \Phi(D_2) - \Phi(D_1)$

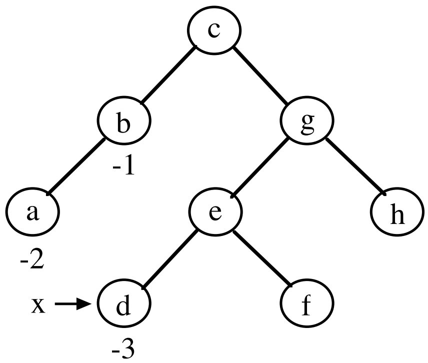
$$2 = 1 + \Phi(D_2) - (-2)$$



SUCC(b) = d

$c_3 = 4, \hat{c}_3 = 2$, and $\hat{c}_3 = c_3 + \Phi(D_3) - \Phi(D_2)$

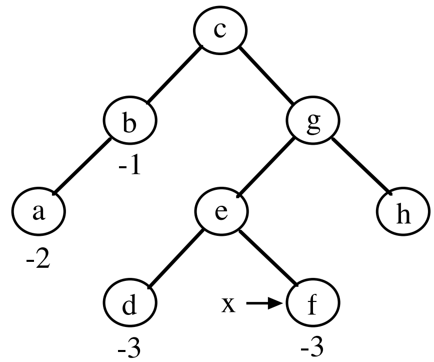
$$2 = 4 + \Phi(D_3) - (-1)$$



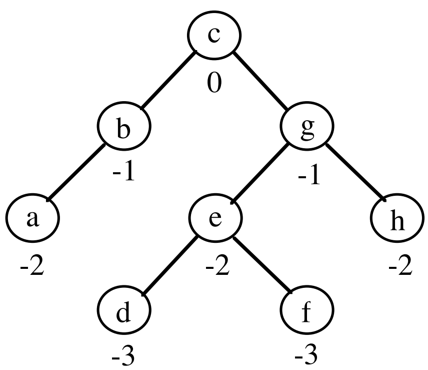
SUCC(d) = f

$c_4 = 2, \hat{c}_4 = 2$, and $\hat{c}_4 = c_4 + \Phi(D_4) - \Phi(D_3)$

$$2 = 2 + \Phi(D_4) - (-3)$$



In general: rank, r(x), of node x is r(root) = 0, r(x→left) = r(x) - 1, r(x→right) = r(x) - 1



For *preorder* (not shown): rank, r(x), of node x is r(root) = 0, r(x→left) = r(x) + 1, r(x→right) = r(x) + 1

Aside: If non-negative ranks/potential are desired (e.g. for inorder and postorder), then make r(root) = D_0 = height of tree (or number of nodes if height is unknown).

DYNAMIC TABLE GROWTH – CLRS 17.4

Applies to tables with embedded free space.

Periodic reorganization takes $\Theta(n)$ time . . .

Fixed vs. fractional growth and amortizing reorganization cost over all inserts

Deletion issues

CLRS PROBLEM 17-2 – Making binary search dynamic

Related to binomial heaps in Notes 7

Representation of dictionary with n items

Binary searches to find item

Inserting an item in $\Theta(\log n)$ amortized time using ordered merges

Deletion?

APPLICATION OF POTENTIAL FUNCTION METHOD THIS SEMESTER . . .

Comparison of online MTF lists to an (unknown) optimal strategy (Notes 4)

Splay trees (Notes 5)

Fibonacci heaps - a priority queue to improve algorithms such as Prim’s and Dijkstra’s (Notes 7)

Union-find trees (Notes 8) - not detailed

Push-relabel methods for maxflows (Notes 12) - not detailed

KMP string search (Notes 15) - easy