Elijah Chou

Dr. Michelangelo Grigni

CS 326

3/21/2022

HW 4

1. (a): To prove that a Fibonacci Heap (FH) can have size n and height n-1 for all n, I will use induction on n when n ≥ 2.

Base case: On an empty FH, do Insert(0), Insert(1), Insert(2), then ExtractMin:

A picture containing icon

Description automatically generated

Inductive step: In a FH with a chain of n-1 nodes, need to show how to get FH of size n.   
If x is the root value of the given chain, then the following should be true:

Text, letter

Description automatically generated

(b): When examining any node x in the chain with a child y that is not a leaf, when y was made a child of x, y had at least one child. This is because y cannot gain children when it is not a root. This also implies that x had another child before y, when considering the functionality of the CONSOLIDATE function. In this case, the aforementioned other child is no longer x’s child. It is important to consider the fact that there are n-2 nodes in the chain that are like node x as explained above.

1. As mentioned in the hint, the FH should be modified to keep track of t(H). After each heap-modifying method described for the FH data structure, add the following code:

if t(H) ≥ 2 \* (D(n) + 1):

CONSOLIDATE(H)

Examining the real-time cost of the CONSOLIDATE function here shows that it should be ≤ c \* [t(H) + D(n) + 1] for some constant c. But the heap after consolidation has t(H’) ≤ D(n) + 1, so **Δ**Φ = Φ(H’) – Φ(H) ≤ D(n) + 1 – t(H) ≤ -1/2 t(H). If c’ ≥ 4c when using c’ \* Φ as the potential function, then the new amortized cost of the CONSOLIDATE function is ≤ c[t(H) + D(n) + 1] + 4c[-1/2t(H)] ≤ 0. This also means that adding the CONSOLIDATE as described earlier in this answer won’t increase the amortized cost of any of the heap-modifying function by more than O(1). However, this also means that t(H) = O(lg n) since D(n) is approximately 1.44 lg n. This also means that the worst case time for EXTRACTMIN(H) is now O(lg n). Similarly, INSERT and UNION amortized costs stay the same but have a new worst case time of O(lg n).

1. (a): Assuming that , for u ≥ 4, This simplifies to

(b): Suppose . This would make the upper bound of P(u) to become . Using this new definition, we can try to further simplify it by adding c to both sides of the equation.

For simplicity, Q(u) will be defined as P(u) + c. This simplifies the above equation as follows:

Bringing this new equation down to the base case of Q(2), we get the following:

Using for simplification, we get the following:

In the above form of the equation, everything before the “exp” term should equal to u/2, and everything inside the “exp” term should be ≤ 1 because it is less than the infinite series ½ + ¼ + 1/8 + … Considering all of this, we get that Q(u) ≤ e/2 \* u. This also means that P(u) = Q(u).

1. (c):

RS-vEB-TREE-INSERT(V,x):

if V.min == NIL:

vEB-EMPTY-TREE-INSERT(V,x)

else if x < V.min:

exchange x with V.min

if V.u > 2:

if V.cluster[high(x)] == NIL:

V.cluster[high(x)] = CREATE-NEW-RS-vEB-TREE

if vEB-TREE-MINIMUM(V.cluster[high(x)]) == NIL:

if V.summary == NIL:

V.summary = CREATE-NEW-RS-vEB-TREE

vEB-TREE-INSERT(V.summary, high(x))

vEB-EMPTY-TREE-INSERT(V.cluster[high(x)], low(x))

else:

vEB-TREE-INSERT(V.cluster[high(x)], low(x))

if x > V.max:

V.max = x

(d):

RS-vEB-TREE-SUCCESSOR(V,x)

if V.u == 2:

if x == 0 & V.max == 1:

return 1

else:

return NIL

elseif V.min != NIL & x < V.min:

return V.min

else:

if V.cluster[high(x)] == NIL:

V.cluster[high(x)] == CREATE-NEW-RS-vEB-TREE

max-low = vEB-TREE-MAXIMUM(V.cluster[high(x)])

if max-low != NIL & low(x) < max-low:

offset = vEB-TREE-SUCCESSOR(V.cluster[high(x)], low(x))

return index(high(x), offset)

else:

if V.summary == NIL:

V.summary = CREATE-NEW-RS-vEB-TREE

succ-cluster = vEB-TREE-SUCCESSOR(V.summary, high(x))

if succ-cluster == NIL:

return NIL

else:

if V.cluster[succ-cluster] == NIL:

V.cluster[succ-cluster] = CREATE-NEW-RS-vEB-TREE

offset = vEB-TREE-MINIMUM(V.cluster[succ-cluster])

return index(succ-cluster, offset)

(e): The only difference between my pseudocode for insert and successor for the RS-vEB trees and the ones for the original vEB trees is that I checked to see if the V.summary or V.cluster entry were initialized before accessing them (and initializing them if they weren’t). Since the added initializations to the original insert and successor functions of the original vEB tree are performed in constant time, the modifications don’t affect the original runtimes bound for the vEB-TREE-INSERT and vEB-TREE-SUCCESSOR functions, which were originally O(lg lg n). If it did not change, then the new RS-vEB-TREE-INSERT and RS-vEB-TREE-SUCCESSOR also have amortized expected times of O(lg lg n).