PRABHJOT KAUR
HOMEWORK 5
CSC 6580

Problem 1

(i) write an $O(h)$ time procedure for finding the predecessor of a key in a BST of height 'h.

Solution

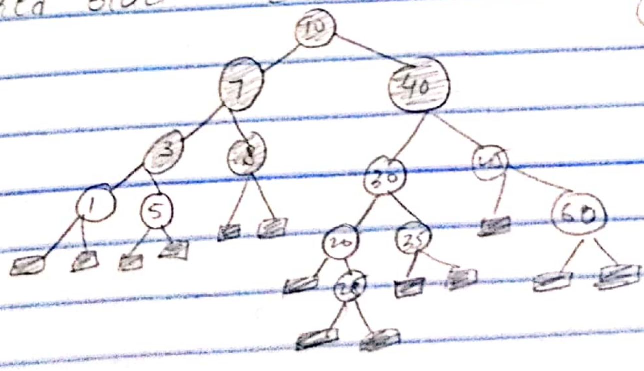
```
TREE-PREDECESSOR (x)
if x.left ≠ NIL
    return TREE-MINIMUM (x.left)
y = x.p
while y ≠ NIL and x == y.left
    x = y
    y = y.p
return y
```

This procedure runs in $O(h)$ time because we are simply following a simple path down the tree

(ii) Prove that a red-black tree with n nodes has height of $O(\log n)$

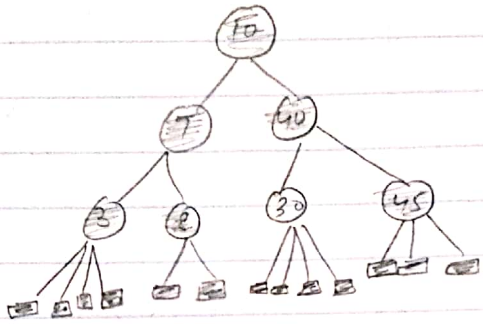
Solution

Consider red black tree whose height is h



$h = 5$

Collapse all red nodes into their black parents to get a tree whose node-degrees are b/w 2 and 4, height $\geq \frac{h}{2}$ and all external nodes are at the same level



$h' = 3$

$h' = \lceil \frac{h}{2} \rceil$

let $\frac{h}{2} \leq h'$ be the height of the collapsed tree

Then;

$$n \geq 2^{h'} - 1$$

$$n+1 \geq 2^{h'}$$

$$\log_2(n+1) \geq \log_2(2^{h'})$$

$$\log_2(n+1) \geq h' \log_2 2$$

$$\log_2(n+1) \geq h'$$

$$\log_2(n+1) \geq \frac{h}{2}$$

$$2\log_2(n+1) \geq h$$

$$O(\log n) \geq h$$

Problem 2 Prove that

(i) The fractional knapsack problem has the greedy choice property.

That is show that we can assemble a globally optimal solution by making locally optimal (greedy) choices.

Solution :-

let's define knapsack problem by $F(n, W)$. we want find This function where n = number of items and W is the total capacity.

let $F(i, j)$ where i = i^{th} item
j = capacity of knapsack

If item i is included, then,

$$F(i, j) = F(i-1, j)$$
$$= \underbrace{v_i}_{\text{value of } i^{th} \text{ item}} + F(\underbrace{i-1}_{\text{remaining items}}, \underbrace{j - w_i}_{\text{capacity decreases}})$$

w_i = weight of i^{th} item

⟹ for fractional knapsack, assume the items are arranged in increasing order of
$\frac{v_i}{w_i}$

let there be many solutions possible
$S = \{ s_1, s_2 \ldots s_n \}$

let the solution chosen by greedy algorithm be $s_n = \min(w_n, W)$ and it then goes on to solve the remaining subproblems

$$= \left(n-1, \{v_1, v_2 \ldots v_{n-1}\}, \{w_1, w_2, \ldots w_{n-1}\}, W - w_n \right)$$

Prove by contradiction this approach will gives an optimal solution.

Suppose the optimal solution to the problem is $s_1, s_2, s_3 \ldots s_n$ where $s_n < \min(w_n, W)$.
let $s_i > 0$.
By decreasing s_i to $\max(0, W - w_i)$ and increasing s_n by the same amount, we get a better solution
This is a contradiction with the assumption we made.

(ii) Prove that 0/1 knapsack does not have greedy property.

Soln. let's take an example. The optimal solution is

10	20	30	50
\$60	\$100	\$120	knapsack
$\frac{v_i}{w_i}$ = 6	5	4	
#1	#2	#3	

$\Rightarrow$

30 \$120
+
20 \$100
= \$220

Even though item #1 has the greatest value per pound, it does not yield the optimal solution.

Choose this greedy solution in the first iteration does not guarantee optimal solution at the end.

Problem 3 Give a dynamic programming solution to the 0/1 knapsack problem that runs in $O(nW)$ time where n is the # of items & W is the maximum weight of items that a thief can put in his knapsack

Solution:

We can solve this by creating a $(n+1)$ by $(W+1)$ table where n is total items that bag can take and W is the weight / capacity

At every row and column, we check if item's weight is below the bag's current capacity.

We do so by comparing the total value of a knapsack (that includes items 1 through $(i-1)$ with maximum weight j at column j and the total value of items 1 through $(i-1)$ with max weight $j - i^{th}$ weight and also item i

Recurrence:

$m[0, W] = 0$ { no item to compare }

$m[i, W] = m[i-1, W]$ if $w_i > W$
{ this item weights more than the capacity, don't include this }

$m[i, W] = \max(m[i-1, W], m[i-1, W-w_i] + v_i)$
if $w_i \leq W$

for reading the values, start from the bottom and move up.

This runs in n^2 times as we have read every row and a column.

Problem 5 Amortized Analysis

Solution $\square$ = initial size is 1

The size doubles everytime table becomes full.

1

1	2

1	2	3	4

1	2	3	4	5	6	7	8

To show that the insertion table has complexity $O(1)$

If table is not full, then cost of insertion is $O(1)$. This is trivial and no need to be proved.

However, when table is full, then we are going to create a new table double the size, copy contents from previous table, deleting contents from old table and then inserting new element in the Table just created.

$O(n)$ (i) Copy contents one by one from old to new Table

$O(n)$ (ii) Delete contents from old table.

$O(1)$ (iii) Insert new element in the new table (insert at the head will cost $O(1)$.)

Total time = $O(n) + O(n) + O(1)$

Total operations = $n + n + 1$

$$\text{Amortized Cost} = \frac{O(n) + O(n) + O(1)}{n+n+1}$$

$$= O(1)$$