CSCE 750
10/10/2023

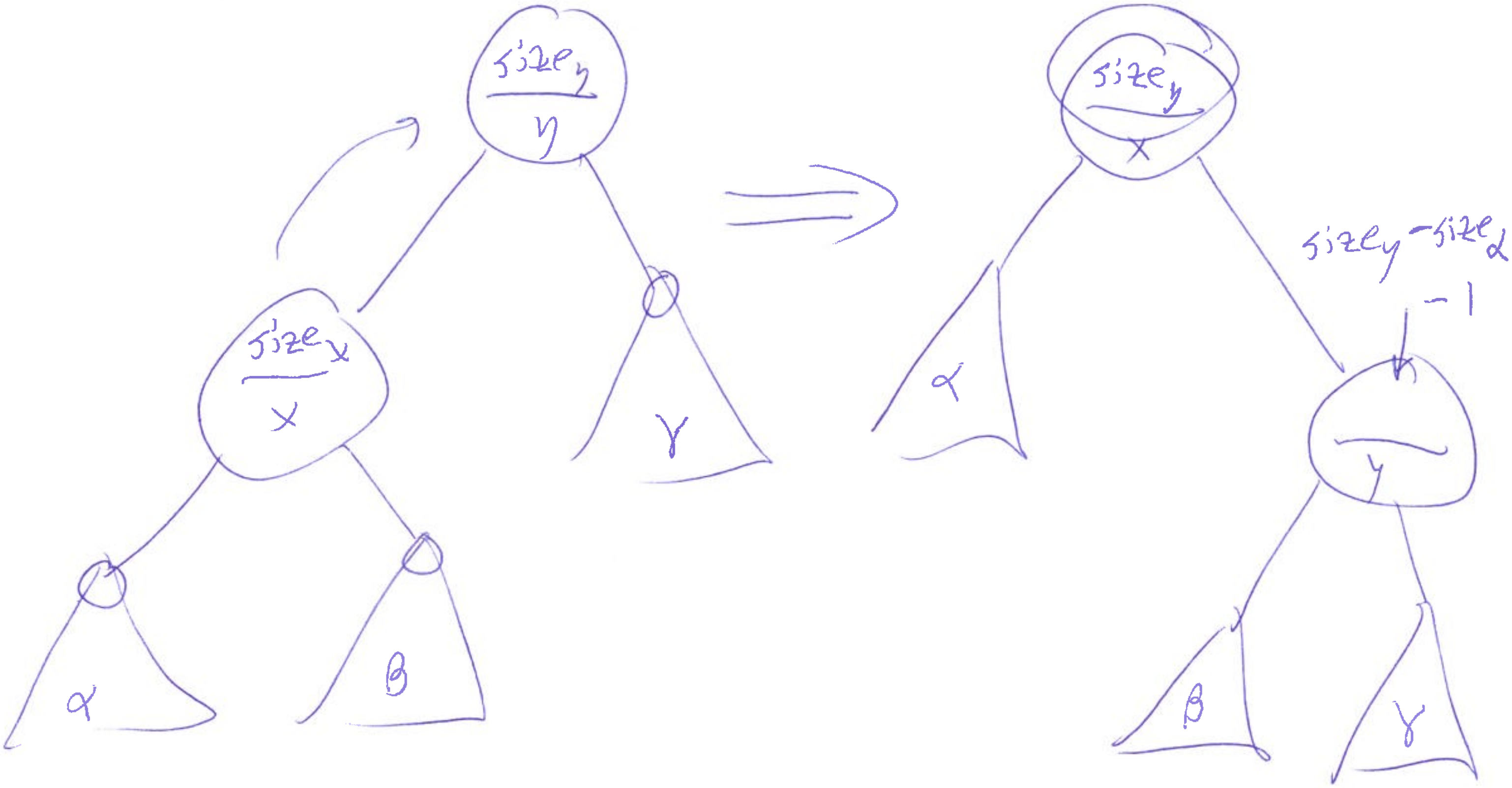
Updating an augmented BST to support dynamic order stats:

Last time: Each node contains an extra field ~~give~~ giving the size of the tree rooted at that node.

Went throug Select last time

Insertion: If successful, increment each size field ~~from~~ on path from inserted node to root.

Rotations:



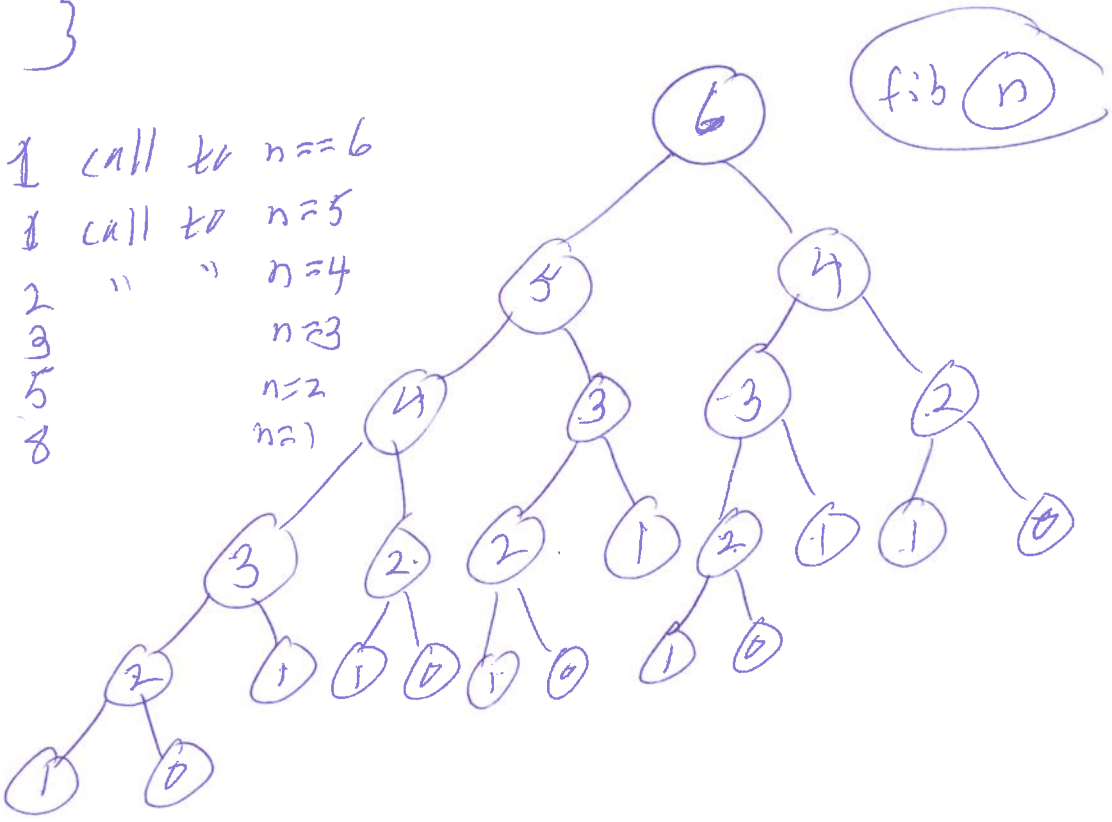
Left Rotations are similar.

Dynamic Programming

Simple example: computing the n'th fibonacci number [Assume all numeric ops take $O(1)$ time — this is not true in this case!]

$F_0 := 0$, $F_1 := 1$, $\forall n \geq 2$, $F_n := F_{n-1} + F_{n-2}$

```
int fib(int n) // n ≥ 0
{
        if (n==0) return 0;
        if (n==1) return 1;
        return(fib(n-1) + fib(n-2));
}
```



1 call to n==6
1 call to n=5
2 " " n=4
3 n=3
5 n=2
8 n=1

1 fix: memoization:

~~On first call to fib(k) do as~~

~~no~~

On calling fib(k), check if its been called before. If yes, just return the value of that call. If not, call as normal, the save the value.

```
Store (fib(0)) == 0
store (fib(1)) == 1

int memo-fib(int n)
{    int save;
     lookup fib(n)
     if there, then return the stored value;
     // n ≥ 2
     ~~return (memo~~
     save = memo-fib(n-1) + memo-fib(n-2);
     store(fib(n) == save);
     return save;
}
```

Need a dictionary structure — disadvantage

Looks a lot like the original implementation — advantage

Tabular approach: Declare an array F[0..n] ~~#~~ where F[i] = fib(i) for i=0—n.

Fill in order of increasing i:

```
F[0] = 0;
F[1] = 1;
for (i=2; i<=n; i++)
    F[i] = F[i-1] + F[i-2];
return F[n];
```

What is a candidate for dynamic programming?

1) Have a recursive solution.

for optimization problems

"Optimal substructure" — the optimal solution to a problem can be found easily from optimal solutions to subproblems.

2) Notice that the space of all recursive call parameters is small & well-organized, possibly in an array.

Knapsack problem: Input: ~~positive~~ integer $C \geq 0$ and a set $\underbrace{\{a_1, \dots, a_n\}}_{\text{global}}$ of positive integers.

Question: Is there a subset $J \subseteq \{1, \dots, n\}$ such that $C = \sum_{i \in J} a_i$?

fill(C, k) means can fill a knapsack of capacity C from $\{a_1, \dots, a_k\}$

```
boolean fill(C, k)  // C ≥ 0, k ≥ 0
{
    if (C<0) return 0;
    if (C==0) return 1;  // J = ∅
    if (k==0) return 0;
    return fill(C, k-1) || fill(C-a_k, k-1);
           T[C, k-1]     OR  T[C-a_k, k-1]
}
```

```
return fill(C, n);
```

$C, n, \{a_1, \dots, a_n\}$ — external

fill(int $\overset{\text{local}}{\text{cap}}$, int $\overset{\text{local}}{k}$) . . . .

Table T[0..C, 0..n] of boolean

```
// T[c,k] = { 1   if can fill cap c ~~with~~ from {$a_1, \ldots, a_k$}
//          { 0   otherwise
fill(C, n) {
    for (k=0; k<=n, k++)
        T[0,k] = 1;

    for (c=1; c<=C; c++)
        T[c,0] = 0;

    for (k=1; k<=n; k++)
        for (c=1; c<=C; c++) {
            T[c,k] = T[c,k-1];
            if (c-$a_k$ >= 0 && T[c-$a_k$, k-1])
                T[c,k] = 1;
        }
    }
    return T[C,n];
}
```