Abraham Cardenas

CMPS 102 — Winter 2019 – Homework 4

1. (10 pts)

$a.$ We can choose three different roots from 3 keys. If the middle key is chosen as the root, there woud only be 1 valid BST permutation since the middle key would be a value in between the first and third key. If the first key is chosen as the root, there would be 2 valid BST permutations since we can choose either the second key or third key as the right child of the root. Similarly, if the third key is chosen as the root, there would be 2 valid BST permutations since we can choose either the first key or second key as the left child of the root.

Adding each BST permutation, we can see that $B(3) = 1 + 2 + 2 = 5$.

$b.$ If a BST had 6 nodes, namely $\{1, 2, 3, 4, 5, 6\}$ with 3 being the root, we would have $\{1, 2\}$ in the left subtree and $\{4, 5, 6\}$ in the right subtree. This would mean that we can have 2! ways of choosing the left subtree since we can choose either 1 or 2 as a left child of the root. Similarly, we can have 3! ways of choosing the right subtree. However, when 5 is chosen as the right child of the root, 2 BST permutations will violate the BST property since 5 is the middle key of the right subtree.

Bringing these values together, we can see that there are $(2! \cdot 3!) - 2 = (2 \cdot 6) - 2 = 10$ different BSTs.

$c.$ Since we need to choose a root from n different keys, we would have $n - 1$ non-root keys. Suppose we choose a key i from the n different keys. This would imply that there are $i - 1$ keys smaller than i (left subtree) and $n - i$ keys larger than i (right subtree). Similarly, both the left and right subtrees are partitioned by having smaller keys on the left and bigger keys on the right.

Since we need to partition both the left and right subtrees, we need to sum over the product of each key in the left and right subtrees. This would give us the following recurrence:

$$\boxed{T(n) = \begin{cases} 1, & \text{if } n \leq 1 \\ \sum_{i=1}^{n} T(i-1) \cdot T(n-i), & \text{otherwise} \end{cases}}$$

$d.$

function COUNT-BSTS(n)
    $tableArr[n]$ ▷ Declare $tableArr$ of size n.
    $tableArr[0] \leftarrow 1$
    $tableArr[1] \leftarrow 1$
    for $i \leftarrow 2$ to n **do**
        $tableArr[i] \leftarrow 0$
        for $j \leftarrow 0$ to i **do**
            $tableArr[i] \leftarrow tableArr[i] + (tableArr[j] \cdot tableArr[i-j-1])$
        end for
    end for
    return $tableArr[n]$
end function

e. Both the inner and outer for loop iterate through, at most, n elements of $tableArr$. So, the algorithm would have an upper bound of $\boxed{O(n^2)}$.

2. (8 pts)

a. Since we need to choose the method with the least cost, the recurrence will be the following:

$$\boxed{OPT(n) = \begin{cases} 0, & \text{if } n = 0 \\ \sum_{i=1}^{n} c_i, & \text{if } 1 \leq n < 4 \\ \min\{\ c_n + OPT(n-1),\ \ h + OPT(n-4)\ \}, & \text{otherwise} \end{cases}}$$

The base case, when $n = 0$, implies that we don't have any tasks to do anymore so the cost would be 0. We then have the case when we have less than 4 tasks ($n < 4$), which is given in the problem that we must do each task ourselves at cost c_i. And lastly, we have the case when $n \geq 4$, where we find the least cost of performing all tasks by either doing one tasks ourselves at cost c_n or paying the handyman for doing four tasks at cost h.

b.

```
function MIN-TASK-COST(c, n, h)
    if n > 0 then
        M[n] ← [0 ... n − 1] ← −1          ▷ Declare M array of size n and init. all values to -1.
        if n < 4 then
            M[0] ← 0
            for i ← 1 to n − 1 do
                M[i] ← c[i] + M[i − 1]
            end for
        else
            Min-Cost-Rec(M, n − 1, c, h)
        end if
        return M[n − 1]
    else
        return −1
    end if
end function
function MIN-COST-REC(M, n, c, h)
    if n < 0 then
        return ∞
    else
        if M[n] = −1 then
            M[n] ← min{ c[n]+ Min-Cost-Rec(M, n − 1, c, h),
                                h+ Min-Cost-Rec(M, n − 4, c, h) }
        end if
        return M[n]
    end if
end function
```

c.

```
function MIN-TASK-COST(c, n, h)
    if n > 0 then
```

```
        M[n] ← [0 ... n − 1] ← −1        ▷ Declare M array of size n and init. all values to -1.
        if n < 4 then
            M[0] ← 0
            for i ← 1 to n − 1 do
                M[i] ← c[i] + M[i − 1]
            end for
        else
            for i ← 4 to n − 1 do
                M[i] ← min{ c[i] + M[i − 1], h + M[i − 4] }
            end for
        end if
        return M[n − 1]
    else
        return −1
    end if
end function
```

d.

```
function PRINT-WHOS-TASK(M, c, n)
    for i ← n − 1 downto 0 do
        if i < 4 then
            print Do tasks 1 to i
            break
        else if M[i] = c[i] + M[i − 1] then
            print Do task i
            i ← i − 1
        else
            print handyman does tasks i − 3 to i
            i ← i − 4
        end if
    end for
end function
```

3. (10 pts)

b. We need to find the sum of least cost of renting a canoe from post 1 to post k. So, the recurrence would be as follows:

$$\boxed{C(k) = \begin{cases} 0, & \text{if } k < 2 \\ min_{1 \leq i \leq k}\{C(i) + R_{i,k}\}, & \text{otherwise} \end{cases}}$$

c.

```
function CANOE-LEAST-COST(R, indArr n)        ▷ indArr passed by reference.
    if n < 2 then
        return 0
    end if
    costArr[n]
    costArr[0] ← 0
    for i ← 1 to n − 1 do
```

        $costArr[i] \leftarrow \infty$
        $indArr[i] \leftarrow 0$
        for $j \leftarrow 0$ to $i - 1$ **do**
            if $costArr[i] > costArr[j] + R[j][i]$ **then**
                $costArr[i] \leftarrow costArr[j] + R[j][i]$
                $indArr[i] \leftarrow j$
            end if
        end for
    end for
    return $costArr[n - 1]$
end function

d. **function** PRINT-RENT-CANOE($indArr$, n)
    for $i \leftarrow n - 1$ downto 0 **do**
        Rent from $indArr[i]$ to i
        $i \leftarrow indArr[i]$
    end for
end function

Sources

https://www.geeksforgeeks.org/binary-search-tree-data-structure/