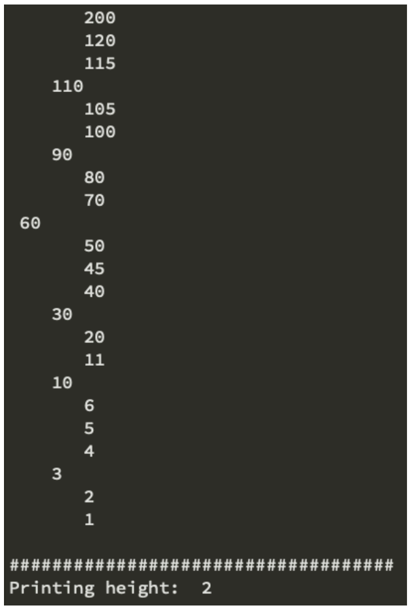
Sergio Gramer – 88521512
Lab #4: B-Trees

In lab 4 we are asked to work with B-Trees. We must complete a variety of tasks including computing the height of the tree, extract items and find nodes at certain depths. We are working with Python on Spyder and the instructor has provided us the beginning of the code which helps us begin our lab.

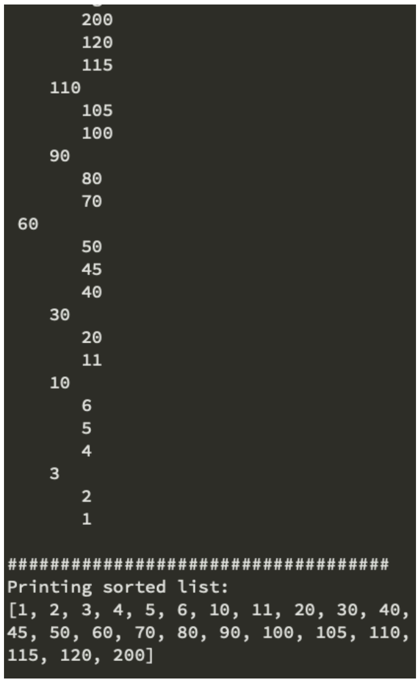
For the first part of the program we are asked to compute the height of the tree. This was done fairly easy. We just created a counter that would increase every time that we encountered a child. This was done recursively. After we found the last child it would return the counter. For part two, we were asked to extract the contents of a tree and put them into a sorted list. I was firstly attempting to do this by sending in the tree along with an empty list and navigate the tree without using recursion. I failed many times using this method because it would give me a repeated list. I would get the first half then the first half again, the second half and then the second half again. I ended up using recursion and just sending a list which would then be converted to empty inside of the method. I still ended up having problems because I wanted to get it all done inside of one for loop. I ended up splitting it and returning at the beginning of the method if it was a leaf. The third part of the lab was to return the minimum element inside of the B-Tree. I knew that the minimum element was always going to be the left most element. Although there was a twist. I had to return the minimum element at a given depth. So for the first part of it I was doing it wrong. I then realized that I had to work with a given depth. So, considering that the smallest element is still the right most element in the level that we were looking for. I would put parameters where if the depth "d" was 0 then I was the level I needed to be at. My next check was to see if I was still inside of the parameters. So if T.isLeaf was true and "d" was not zero, I could not go further down and the program would exit with error code "-1". Then finally I would recursively call the same function sending the child of the parent and subtract one from d to reach the desired level. The next step was to do the exact same thing except with the maximum element. This is pretty self explanatory except that I was looking for the right most element at a given depth. At first I was trying to do the same thing as the first one but I could navigate it once I got to the level because I was working with the left most child which has no notion of the right most child (if its in a completely separate parent). So I had to use the length of the array when recursively sending back the child. For example, on the first one I was sending T.child[0], because the left most child is always going to be zero. I tried that and then reaching the level navigating to the right most child but failed. So, I decided to send the length of the array as so: T.child[len(T.item)], so I was sending the right most child and looking through there. The fifth part of the assignment asked to return the number of nodes at a given depth. This was tricky because again I wanted to navigate it without using recursion. I found that recursively calling the function was working out easier for me. I used the fact that if the depth was zero to begin with it was going to be length one because that is the root. So I decided that would be my first parameter. Now, I would check to see if the height was less than the depth was being asked for because I kept getting an error by using the compare to T.isLeaf and equality to 0. I decided to use the method I had previously created Height(T) and see which was greater. If it was greater it would move on to calculate inside the range of the length of the child only if the depth had been reached. This was also done recursively. For part six we were tasked with printing all of the values at a given depth. I actually found this harder than I thought because I was trying to send back just the array at the given depth and then printing it outside of the method. For some reason I could not get it to work, I decided then that I would print inside of the method. I used a simple for loop when the depth was reached to print the elements inside of the array at given depth. Part six of the lab asked us to find the number of nodes that were full. This one in particular took me some time because I had not revised the code given to us by Dr. Fuentes. The initial train of that was that a node was full if the capacity had been reached. In class we had talked about a capacity of 5. I kept running trials at 5 full nodes but I would always get zero. Finally I looked at the code and saw that there was a T.max_items that I could use as my comparator. As I traversed the levels I would compare to the T.max_items, if it was equal I would add one to the counter else go onto the next one until the level before the leaves. The next step was to see if there was any full leaves. Again, I would compare to the T.max_items and add to the counter except this time I would only add if it was a leaf.

Printing the height of the given tree:



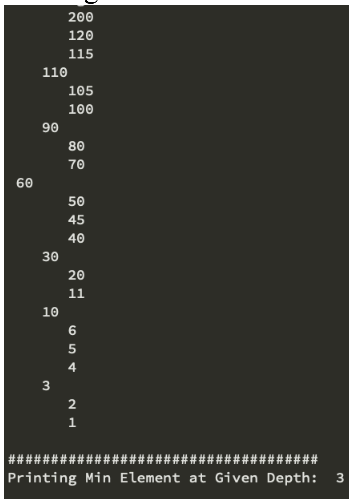
```
       200
       120
       115
    110
       105
       100
    90
       80
       70
 60
       50
       45
       40
    30
       20
       11
    10
       6
       5
       4
    3
       2
       1

####################################
Printing height:  2
```

Extracting from the B-Tree and printing in a sorted list:



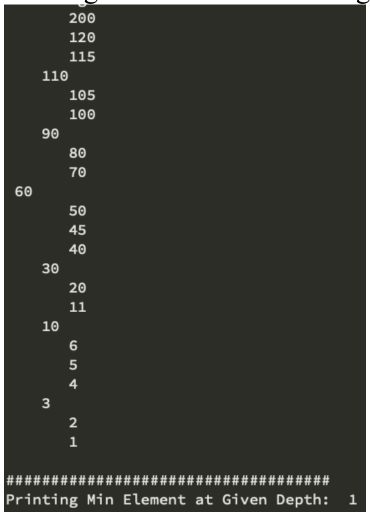
```
       200
       120
       115
    110
       105
       100
    90
       80
       70
 60
       50
       45
       40
    30
       20
       11
    10
       6
       5
       4
    3
       2
       1

####################################
Printing sorted list:
[1, 2, 3, 4, 5, 6, 10, 11, 20, 30, 40,
45, 50, 60, 70, 80, 90, 100, 105, 110,
115, 120, 200]
```

Printing minimum element at given depth of 1:



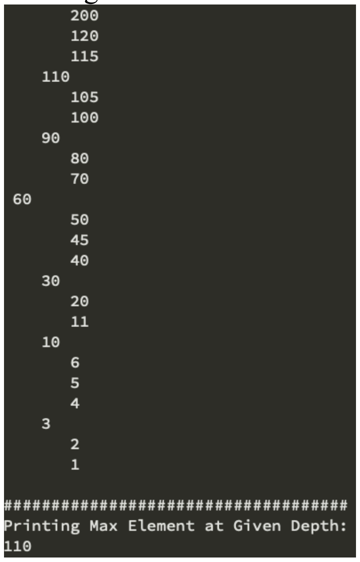
```
       200
       120
       115
    110
       105
       100
    90
       80
       70
 60
       50
       45
       40
    30
       20
       11
    10
       6
       5
       4
    3
       2
       1

###################################
Printing Min Element at Given Depth:  3
```

Printing minimum element at given depth of 2:



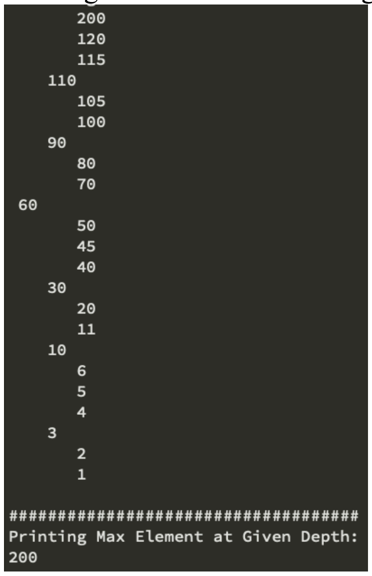
```
       200
       120
       115
    110
       105
       100
    90
       80
       70
 60
       50
       45
       40
    30
       20
       11
    10
       6
       5
       4
    3
       2
       1

###################################
Printing Min Element at Given Depth:  1
```

Printing Maximum Element at given dept of 1



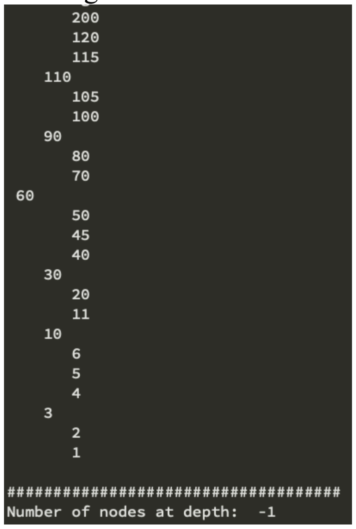
```
       200
       120
       115
    110
       105
       100
    90
       80
       70
 60
       50
       45
       40
    30
       20
       11
    10
       6
       5
       4
    3
       2
       1

####################################
Printing Max Element at Given Depth:
110
```

Printing maximum element at given depth of 2:



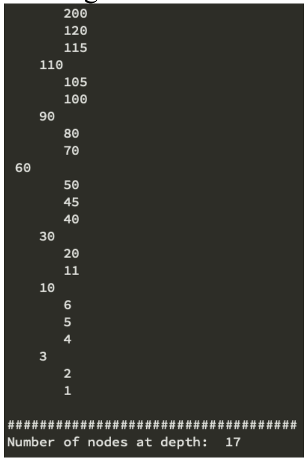
```
       200
       120
       115
    110
       105
       100
    90
       80
       70
 60
       50
       45
       40
    30
       20
       11
    10
       6
       5
       4
    3
       2
       1

####################################
Printing Max Element at Given Depth:
200
```

Printing number of nodes at depth 3: (depth 3 does not exist)



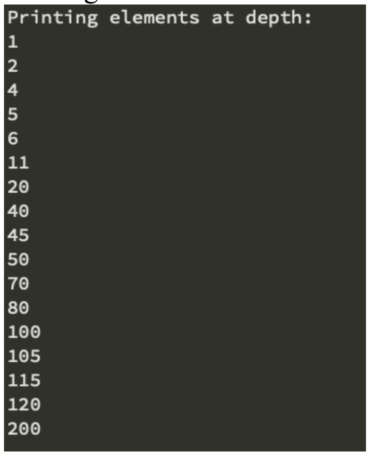
```
       200
       120
       115
    110
       105
       100
    90
       80
       70
 60
       50
       45
       40
    30
       20
       11
    10
       6
       5
       4
    3
       2
       1

####################################
Number of nodes at depth:  -1
```

Printing number of nodes at depth 2:



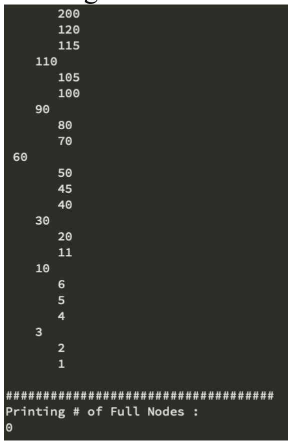
```
       200
       120
       115
    110
       105
       100
    90
       80
       70
 60
       50
       45
       40
    30
       20
       11
    10
       6
       5
       4
    3
       2
       1

####################################
Number of nodes at depth:  17
```

Printing the elements at desired depth 2:



```
Printing elements at depth:
1
2
4
5
6
11
20
40
45
50
70
80
100
105
115
120
200
```

Printing number of full nodes:



```
       200
       120
       115
    110
       105
       100
    90
       80
       70
 60
       50
       45
       40
    30
       20
       11
    10
       6
       5
       4
    3
       2
       1

####################################
Printing # of Full Nodes :
0
```

I learned from this project that B-Trees are a way of storing and accessing information is a lot easier and faster than other methods. You can easily insert and extract information using the native python lists that are embedded into the B-Trees. Gather information on the data is also easy. For example, getting the length (or size) of a particular depth is easy because you use the len() function to calculate it.

```
class BTree(object):
    # Constructor
    def __init__(self,item=[],child=[],isLeaf=True,max_items=5):
        self.item = item
        self.child = child
        self.isLeaf = isLeaf
        if max_items <3: #max_items must be odd and greater or equal to 3
            max_items = 3
        if max_items%2 == 0: #max_items must be odd and greater or equal to 3
            max_items +=1
        self.max_items = max_items

def FindChild(T,k):
    # Determines value of c, such that k must be in subtree T.child[c], if k is in the BTree
    for i in range(len(T.item)):
        if k < T.item[i]:
            return i
    return len(T.item)

def InsertInternal(T,i):
    # T cannot be Full
    if T.isLeaf:
        InsertLeaf(T,i)
    else:
        k = FindChild(T,i)
        if IsFull(T.child[k]):
            m, l, r = Split(T.child[k])
            T.item.insert(k,m)
            T.child[k] = l
            T.child.insert(k+1,r)
            k = FindChild(T,i)
        InsertInternal(T.child[k],i)

def Split(T):
    #print('Splitting')
    #PrintNode(T)
    mid = T.max_items//2
    if T.isLeaf:
        leftChild = BTree(T.item[:mid])
        rightChild = BTree(T.item[mid+1:])
    else:
        leftChild = BTree(T.item[:mid],T.child[:mid+1],T.isLeaf)
        rightChild = BTree(T.item[mid+1:],T.child[mid+1:],T.isLeaf)
    return T.item[mid], leftChild,  rightChild

def InsertLeaf(T,i):
    T.item.append(i)
    T.item.sort()

def IsFull(T):
    return len(T.item) >= T.max_items

def Insert(T,i):
    if not IsFull(T):
        InsertInternal(T,i)
    else:
        m, l, r = Split(T)
        T.item =[m]
        T.child = [l,r]
        T.isLeaf = False
        k = FindChild(T,i)
        InsertInternal(T.child[k],i)


def height(T):
    if T.isLeaf:
        return 0
    return 1 + height(T.child[0])
```

```
def Search(T,k):
    # Returns node where k is, or None if k is not in the tree
    if k in T.item:
        return T
    if T.isLeaf:
        return None
    return Search(T.child[FindChild(T,k)],k)

def Print(T):
    # Prints items in tree in ascending order
    if T.isLeaf:
        for t in T.item:
            print(t,end=' ')
    else:
        for i in range(len(T.item)):
            Print(T.child[i])
            print(T.item[i],end=' ')
        Print(T.child[len(T.item)])

def PrintD(T,space):
    # Prints items and structure of B-tree
    if T.isLeaf:
        for i in range(len(T.item)-1,-1,-1):
            A = T.item[i]
            print(space,T.item[i])
    else:
        PrintD(T.child[len(T.item)],space+'   ')
        for i in range(len(T.item)-1,-1,-1):
            print(space,T.item[i])
            PrintD(T.child[i],space+'   ')

def SearchAndPrint(T,k):
    node = Search(T,k)
    if node is None:
        print(k,'not found')

    else:
        print(k,'found',end=' ')
        print('node contents:',node.item)

def ExtractSort(T, sort) :
    sort = []
    if T.isLeaf:
         return T.item

    for i in range(len(T.child)) :
           sort += ExtractSort(T.child[i],sort)
           if i < len(T.item):
               sort.append(T.item[i])

    return sort

def MinElementAtGivenDepth(T, d) :
    if d is 0 :
        return T.item[0]

    if T.isLeaf and d != 0 :
        return -1

    return MinElementAtGivenDepth(T.child[0], d-1)

def MEAGD(T, m) : #Max Element At Given Depth
    if m is 0 :
        return T.item[(len(T.item)-1)]

    if T.isLeaf and m != 0:
        return -1

    return MEAGD(T.child[len(T.item)], m - 1)
```

```
def NodesAtDepth(T, n) :
    if n is 0 :
        return len(T.item)

    if n > height(T) :
        return -1

    c = 0
    for j in range(len(T.child)) :
        c += NodesAtDepth(T.child[j], n - 1)

    return c

def ElementsAtDepth(T, d) :
    if d is 0 :
        for i in range(len(T.item)) :
            print(T.item[i])
            i += 1

    if T.isLeaf and d != 0 :
        return print('Not found')

    for j in range(len(T.child)) :
        ElementsAtDepth(T.child[j], d-1)
        j += 1

def FullNodes(T) :
    c = 0
    if T.max_items == len(T.item) :
        c += 1
        return c

    if T.isLeaf :
        return c

    else :
        for i in range(len(T.child)) :
            c += FullNodes(T.child[i])

    return c

def FullLeaves(T) :
    if T.isLeaf :

        if len(T.item) == T.max_items :
            return 1

        else :
            return 0

    count = 0
    for i in range(len(T.child)):
        count+= FullLeaves(T.child[i])

    return count


A = []
L = [30, 50, 10, 20, 60, 70, 100, 40, 90, 80, 110, 120, 1, 11 , 3, 4, 5, 105, 115, 200, 2, 45, 6]
T = BTree()
for i in L:
    print('Inserting',i)
    Insert(T,i)
    PrintD(T,")
    #Print(T)
    print('\n###################################')

#SearchAndPrint(T,60)
#SearchAndPrint(T,200)
```

```
#SearchAndPrint(T,25)
#SearchAndPrint(T,20)
#SearchAndPrint(T,50)
#SearchAndPrint(T,10)
print('Printing # Full Leaves:')
print(FullLeaves(T))

print('Printing # of Full Nodes : ')
nodes = FullNodes(T)
print(nodes)

print('Printing elements at depth: ')
ElementsAtDepth(T, 1)

n = NodesAtDepth(T, 2)
print('Number of nodes at depth: ', n)

m = MEAGD(T, 2)
print('Printing Max Element at Given Depth: ', m)

d = MinElementAtGivenDepth(T, 2›)
print('Printing Min Element at Given Depth: ', d)

F = ExtractSort(T, A)
print('Printing sorted list: ')
print(F)

›print('Printing height: ', height(T))
```

ACADEMIC CERTIFICATION

I “Sergio Gramer” certify that this project is entirely my own work. I wrote, debugged, and tested the code being presented, performed the experiments, and wrote the report. I also certify that I did not share my code or report or provided inappropriate assistance to any student in the class.

Signed: 03/29/2019

Sergio Gramer

[illegible]

____