Fernando Marquez

CS 2302

*Lab 5 Min\_Heap*

The objective of this lab was to extract a set of numbers and sort them using the data structure min heap. Once implementation was created, 3 options were given to test the file. I first chose to implement unit test, however, unit test didn’t respond. I chose to hard coded my program next which was a success.

I first started with creating a constructor for the min heap which would build the array. After, I needed to insert items into the array in which i can append keys into the array. I then created helper methods which would help perform swap and print for the other methods. Thus going to straight into the properties of balancing in min heap, I made sure that the parent was never greater than left or right child. If the parent was greater than the two I swapped and heapified the parent. Going into my next method I could then sort the heap and extract the minimum value in the array. Which is my next and last method in the program. After I attempted to do unit testing which didn’t respond so I tested my program differently

Hard Code:

h.insert(1)

h.insert(4)

h.insert(8)

h.insert(3)

h.insert(6)

h.insert(9)

h.insert(2)

h.insert(5)

Results:

Extracted min value: 1

[2, 3, 4, 5, 6, 8, 9]

h.insert(4)

h.insert(4)

h.insert(8)

h.insert(3)

h.insert(6)

h.insert(9)

h.insert(2)

h.insert(5)

results:

Extracted min value: 2

[3, 4, 4, 5, 6, 8, 9]

I learned how to implement min heap in python and the properties of when its balancing. The program had a run time of O(log n) when it would heapfiy. O(n) when it would create and build the heap and O(n) when it ran through the program.

Appendix

import unittest

class Heap:

# constructor

def \_\_init\_\_(self):

self.heap = []

# insert items to the array

def insert(self, key):

self.heap.append(key) # append key to end of list

index = len(self.heap) - 1

while index > 0:

# parent of index

parent = (index - 1) // 2

if self.heap[index] >= self.heap[parent]:

return

else:

self.swap(parent, index)

index = parent

def is\_empty(self):

return len(self.heap) == 0

def swap(self, p, c):

self.heap[p], self.heap[c] = self.heap[c], self.heap[p]

def print(self):

print(self.heap)

# Min heap properties of balancing

def heapify(self, i):

val = len(self.heap)

p = i # abbreviated for parent

lc = 2 \* i + 1 # abbreviated for left child

rc = 2 \* i + 2 # abbreviated for right child

# left child > parent

if lc < val and self.heap[i] > self.heap[lc]:

p = lc

# right child > parent

if rc < val and self.heap[p] > self.heap[rc]:

p = rc

# Swap parent if i is greater

if p != i:

self.swap(i, p)

self.heapify(p)

# Sort in ascending order

def heap\_sort(self):

temp = [] # temp is created every time root its moved

size = len(self.heap)

# Traverse array and pop the root into the temp array

for i in range(size):

temp.append(self.heap[0])

self.swap(0, len(self.heap) - 1)

self.heap = self.heap[:len(self.heap) - 1]

self.heapify(0)

# Overwrite heap with temp

self.heap = temp

def extract\_min(self):

if self.is\_empty():

return None

ext\_min = self.heap[0]

end = len(self.heap) - 1 # heap is reduced by 1

self.swap(0, end) # swap root then heapify it

self.heap = self.heap[:end]

self.heapify(0)

return ext\_min

“I 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.”