Crowd Organizer App - Python

CS 3310 – Dr. Donna Kaminski

Index

[Log.txt 2](#_Toc402990152)

[CrowdOrganizer 3](#_Toc402990153)

[CustomerPrQ 4](#_Toc402990154)

[Node 5](#_Toc402990155)

# >>> Program starting

STORE IS OPENING

>>> Initial heap built containing 21 nodes

SERVING: Bob O'Leary Sr. (6)

SERVING: Jim O'Leary (6)

SERVING: Rosie O'Brien (19)

SERVING: Bob O'Leary (30)

SERVING: Stan Anderson (65)

SERVING: Mary Smith (66)

SERVING: Aziz Gupta (71)

SERVING: Maria Garcia (78)

SERVING: Roman Zwykowicz (82)

SERVING: Josh Miller (83)

SERVING: Rajesh Patel (94)

SERVING: John Doe (102)

SERVING: Lynn VanderCook (102)

SERVING: Ling Yin (105)

SERVING: Malia AlFaleh (105)

SERVING: Li Sung (106)

ADDING: Lottie Zipnowski (42)

ADDING: Jack Larson (113)

ADDING: Fallah Ola (74)

ADDING: Jun Sung (125)

ADDING: Mrs. VanDooran (96)

SERVING: Lottie Zipnowski (42)

SERVING: Fallah Ola (74)

SERVING: Mrs. VanDooran (96)

STORE IS CLOSING

>>> Heap currently has 7 nodes remaining

SERVING: Jamal Brown (107)

SERVING: Latisha Ford (108)

SERVING: Su Chan (109)

SERVING: Jack Larson (113)

SERVING: Mohsin Waleed (118)

SERVING: Abdalla AlSaid (119)

SERVING: Jun Sung (125)

>>> Heap is now empty

>>> Program terminating

‘’’

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Crowd Organizer App

Author: Caleb Viola

CrowdOrganizer.py

Main controller for application which manages a priority queue

for customers to be served at BetterBuy.

Reads file containing initialization data.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

‘’’

# from CustomerPrQ import \*

import os

class CrowdOrganizer:

@staticmethod

def main():

prQ = CustomerPrQ('LineAt6Am.csv')

log = open('Log.txt', 'w')

log.write('>>> Program starting\n')

with open('Events.txt', 'r') as file:

for line in file:

if not line.startswith('//', 0, 2):

raw = line.rstrip('\n')

func = {'O':(prQ.arrangeCustomerQ, log),

'C':(prQ.serveRemainingCustomers, log),

'N':(prQ.addCustomerToQ, (raw.split(',')[1:], True, log)),

'S':(prQ.serveACustomer, log)}[raw[:1]]

func[0](func[1])

log.close()

os.remove('Log.txt') if os.path.exists('Log.txt') else None

CrowdOrganizer.main()

'''

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Crowd Organizer App

Author: Caleb Viola

CustomerPrQ.py

Implementation of min-heap for managing priority queue.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

'''

import math

# 

# class CustomerPrQ():

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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

self.initialPriorityValue = 101

self.heapNodes = [] #Heap storage

self.fileName = name

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

def arrangeCustomerQ(self, log):

log.write('STORE IS OPENING\n')

with open(self.fileName, 'r') as file:

for line in file:

if not line.startswith('//', 0, 2):

self.addCustomerToQ(line.rstrip('\n').split(','), False, log)

log.write('>>> Initial heap built containing {} nodes\n\n'.format(len(self.heapNodes)))

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

def serveRemainingCustomers(self, log):

log.write('\n')

log.write('STORE IS CLOSING\n')

log.write('>>> Heap currently has {} nodes remaining\n'.format(len(self.heapNodes)))

while self.heapNodes:

self.serveACustomer(log)

log.write('>>> Heap is now empty\n')

log.write('>>> Program terminating\n')

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

def addCustomerToQ(self, raw, toPrint, log):

nextInLine = self.\_\_determinePriorityValue(raw[1:])

if toPrint:

log.write('ADDING: {} ({})\n'.format(''.join(raw[:1]), nextInLine))

self.\_\_heapInsert(''.join(raw[:1]), nextInLine)

self.initialPriorityValue += 1

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

def serveACustomer(self, log):

if self.heapNodes:

log.write('SERVING: {} ({})\n'.format(self.heapNodes[0].name,

self.heapNodes[0].priorityValue))

self.\_\_heapDelete()

else:

log.write('>>> Heap is empty\n')

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

def \_\_heapInsert(self, name, priorityValue):

x = Node(name, priorityValue)

self.heapNodes.append(x)

self.walkUp(len(self.heapNodes) - 1)

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

def \_\_heapDelete(self):

minItem = self.heapNodes[0]

self.heapNodes[0] = self.heapNodes[len(self.heapNodes) - 1]

self.heapNodes.pop()

self.walkDown(0)

return minItem

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

def \_\_determinePriorityValue(self, raw):

subtract = 0

for jumpTheQPoints in raw:

if not jumpTheQPoints == '':

subtract += {jumpTheQPoints: (15 if jumpTheQPoints.isnumeric() and

(int(jumpTheQPoints) >= 65 or int(jumpTheQPoints) >= 80) else 0),

'employee': 25,

'owner': 80,

'vip': 5,

'superVIP': 10}[jumpTheQPoints]

return self.initialPriorityValue - subtract

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

def walkUp(self, startFrom):

i = startFrom

while (i > 0) and \

(self.heapNodes[i].priorityValue < self.heapNodes[math.trunc((i - 1) / 2)].priorityValue):

self.heapNodes[i], self.heapNodes[math.trunc((i - 1) / 2)] = \

self.heapNodes[math.trunc((i - 1) / 2)], self.heapNodes[i] #Swap

i = math.trunc((i - 1) / 2)

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

def walkDown(self, startFrom):

i = startFrom

smCh = self.subOfSmCh(i)

while (2 \* i + 1) <= len(self.heapNodes) - 1 and \

(self.heapNodes[i].priorityValue > self.heapNodes[smCh].priorityValue):

self.heapNodes[i], self.heapNodes[smCh] = \

self.heapNodes[smCh], self.heapNodes[i] #Swap

i = smCh

smCh = self.subOfSmCh(i)

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

def subOfSmCh(self, i):

if ((2\*i + 2) > len(self.heapNodes) - 1 or

(self.heapNodes[2\*i + 1].priorityValue <= self.heapNodes[2\*i + 2].priorityValue)):

return 2\*i + 1

else:

return 2\*i + 2

‘’’

Min-heap node

‘’’

# class Node:

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

def \_\_init\_\_(self, name, priorityValue):

self.name = name

self.priorityValue = priorityValue