Mark Demore II, 2d Lt

CSCE686 - Dr. Lamont

Spr 2020 - Homework 2

Greedy Algorithm for the Minimum Cabs Problem

1. Introduction

The purpose of this exercise is to design and analyze a greedy algorithm approach to solving polynomial time problems. By detailing the problem and algorithm domains, along with their codependency in developing an algorithmic solution, this framework can be expanded to develop algorithms for more complex problems. For this design project, I will be applying a greedy algorithm to the minimum cabs problem [1].

1. Problem Domain Requirements [1,5]
2. Size of Solution Space: n for queue
3. Objective Function: min ∑ni=1 ci where ci = 1 if someone is waiting for a cab and cj for all j < i are in use
4. Constraint: Define Q as a queue of cab riders with start and end time of trips

min ∑ni=1 ci

End of Search, Q is empty

1. Domains: Q is queue of riders and times, C is set of cabs needed
2. The size of the solution space is n, because the range of solutions is from 0 (in the case of 0 riders) to n (all riders traveling at same time)
3. Search Algorithm Domain Requirements [2,5]
4. Name: Greedy-Search (Di)
5. Domains: Data input – Di set of candidates (riders), Data output – Do set of solutions (cab/rider pairings)
6. Operations: Assign(x) – x is in Di, pop from queue, assign to cab ci and add (x,ci) to solution set C
7. Search Algorithm Design Specification [3,5]
8. Name: Min Cab Greedy Search (Di)
9. Domains: Di – riders and travel time pairings, Do – riders and cab pairings
10. Operations: findCab(time) – search for available cab, if not available, add cab

assign(x) – x is in Di, findCab and add cab/rider pairing to C, pop(x)

pop(x) – remove rider from queue

1. Set of candidates: riders and available cabs

Next-state generator: cab/rider pairing, next waiting rider

Selection function: select first available cab or add a new cab

Feasibility function: no more than 1 rider per cab at same time

Solution function: no extra cabs

1. Expanded Algorithmic Design Specification [4]

Set C = empty

Queue Q = empty

Set c = empty

findCab(time):

for cab in c:

if cab is available at time:

return cab

c.add(cab)

return cab

pop(rider):

Q.dequeue(rider,time)

assign(rider,time):

cab = findCab(time)

C.append(rider,cab)

pop(rider,time)

MinCabGreedySearch(Di):

for pair in Di:

Q.append(rider,time)

while Q != empty:

assign(Q.top)

return C

1. Implementation and Test and Analysis

The above pseudocode was implemented on 3 data sets. The python code is copied below in Appendix A. The output from the test is below:

TEST 1:

Input Size: 5

Cabs Needed: 3

Time Taken: 0.00018289999999999973

TEST 2:

Input Size: 25

Cabs Needed: 15

Time Taken: 0.00019070000000000198

TEST 2:

Input Size: 50

Cabs Needed: 30

Time Taken: 0.00023019999999999985

Given that the problem was simple, there is little deviation in the time to compute, but it does increase with the size of input. This is to be expected with a greedy algorithm.

1. Conclusion

The problem domain/algorithm domain design process is efficient and effective as an approach to developing algorithms to solve problems. The benefits are in outlining the available input and potential solutions and creating functions to map the two together. Using this process, other algorithms could be developed for the problem, particularly non-greedy algorithms that could ensure optimality. The problem domain has a greater complexity, containing partial and suboptimal solutions, while the algorithm domains complexity is more fixed based on the selected approach. Some variations of the minimum cabs problem include other scheduling problems with limited resources and users that need deconfliction, as well as CPU scheduling, particularly on multicore systems.

1. References

[1] <https://www.hackerearth.com/practice/algorithms/greedy/basics-of-greedy-algorithms/practice-problems/algorithm/minimum-cabs-0798cfa5/>

[2] <https://en.wikipedia.org/wiki/Greedy_algorithm>

[3] <https://www.geeksforgeeks.org/greedy-algorithms/>

[4] <https://www.hackerearth.com/practice/algorithms/greedy/basics-of-greedy-algorithms/tutorial/>

[5] Lecture2\_GenSearch20.docx

Appendix A.

import time  
  
#data input  
test1 = [("Bob", 8, 9), ("Carl", 9, 10), ("Matt", 8.5, 9), ("John", 9.5, 10), ("Luke", 11, 12)]  
test2 = [("Bob", 8, 9), ("Carl", 9, 10), ("Matt", 8.5, 9), ("John", 9.5, 10), ("Luke", 11, 12), ("Bob", 8, 9), ("Carl", 9, 10), ("Matt", 8.5, 9), ("John", 9.5, 10), ("Luke", 11, 12), ("Bob", 8, 9), ("Carl", 9, 10), ("Matt", 8.5, 9), ("John", 9.5, 10), ("Luke", 11, 12), ("Bob", 8, 9), ("Carl", 9, 10), ("Matt", 8.5, 9), ("John", 9.5, 10), ("Luke", 11, 12), ("Bob", 8, 9), ("Carl", 9, 10), ("Matt", 8.5, 9), ("John", 9.5, 10), ("Luke", 11, 12)]  
test3 = [("Bob", 8, 9), ("Carl", 9, 10), ("Matt", 8.5, 9), ("John", 9.5, 10), ("Luke", 11, 12), ("Bob", 8, 9), ("Carl", 9, 10), ("Matt", 8.5, 9), ("John", 9.5, 10), ("Luke", 11, 12), ("Bob", 8, 9), ("Carl", 9, 10), ("Matt", 8.5, 9), ("John", 9.5, 10), ("Luke", 11, 12), ("Bob", 8, 9), ("Carl", 9, 10), ("Matt", 8.5, 9), ("John", 9.5, 10), ("Luke", 11, 12), ("Bob", 8, 9), ("Carl", 9, 10), ("Matt", 8.5, 9), ("John", 9.5, 10), ("Luke", 11, 12), ("Bob", 8, 9), ("Carl", 9, 10), ("Matt", 8.5, 9), ("John", 9.5, 10), ("Luke", 11, 12), ("Bob", 8, 9), ("Carl", 9, 10), ("Matt", 8.5, 9), ("John", 9.5, 10), ("Luke", 11, 12), ("Bob", 8, 9), ("Carl", 9, 10), ("Matt", 8.5, 9), ("John", 9.5, 10), ("Luke", 11, 12), ("Bob", 8, 9), ("Carl", 9, 10), ("Matt", 8.5, 9), ("John", 9.5, 10), ("Luke", 11, 12), ("Bob", 8, 9), ("Carl", 9, 10), ("Matt", 8.5, 9), ("John", 9.5, 10), ("Luke", 11, 12)]  
  
CabOut = []  
CabUse = []  
  
def findCab(riderTime):  
 index = 0 #cab number  
 flag = False #do we need to add a cab  
 for cab in CabUse:  
 if (cab < riderTime[1]): #is one of the existing cabs available at this time  
 cab = riderTime[2] #update next available time  
 flag = True  
 break  
 index += 1  
 if not flag:  
 CabUse.append(riderTime[2]) #add new cab and update next available time  
 return index  
  
def assign(riderTime):  
 cab = findCab(riderTime)  
 CabStr = "Cab # " + str(cab)  
 tup = (riderTime, CabStr)  
 CabOut.append(tup) #data output: cab + rider pairings  
  
def main():  
 # MinCabGreedySearch  
 t0 = time.clock()  
 for item in test1:  
 assign(item)  
 t1 = time.clock() - t0  
 print("TEST 1:")  
 print("Input Size: " + str(len(test1)))  
 print("Cabs Needed: " + str(len(CabUse)))  
 print("Time Taken: " + str(t1))  
  
 CabOut.clear()  
 CabUse.clear()  
 t2 = time.clock()  
 for item in test2:  
 assign(item)  
 t3 = time.clock() - t2  
 print("TEST 2:")  
 print("Input Size: " + str(len(test2)))  
 print("Cabs Needed: " + str(len(CabUse)))  
 print("Time Taken: " + str(t3))  
  
 CabOut.clear()  
 CabUse.clear()  
 t4 = time.clock()  
 for item in test3:  
 assign(item)  
 t5 = time.clock() - t4  
 print("TEST 2:")  
 print("Input Size: " + str(len(test3)))  
 print("Cabs Needed: " + str(len(CabUse)))  
 print("Time Taken: " + str(t5))  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 # execute only if run as a script  
 main()