Cash Me A Taxi

Albert Zhou

September 19, 2020

This project is meant to be a reiteration of the original Cash Me A Taxi project for SFWR ENG 2XB3 done in a similar style of SFWR ENG 2AA4 documentation. Additionally, a revamp of the implementation is made to better fit my skill set.

The original project can be found in the OldCashMeATaxi folder.

The members of the original CashMeATaxi project are Albert Zhou, Areeba Aziz, David Bednar, Dylan Smith, and Himanshu Aggarwal.

Contents

1 Intro

This project intends to provide a solution for the problem of taxi drivers in New York Citystruggling financially as modern ride-sharing apps like Uber currently dominate the market. The solution is to calculate the most optimal routes between NYC zones that earn drivers the most profit, based on 10 years' worth of taxi trips datasets provided by the City of New York. The datasets contain taxi trip data such as the distance travelled per trip, the amount of fare made, the tips earned, and the start/end location and times.

Our program first parses the intensive data that contains the taxi trip records and builds a directed graph with nodes representing a NYC taxi zone. The user then can select from various functions to run on the graph, such as search for the most optimal path between two given zones, or sort the edges (routes) by the given field (fare, tip, distance, etc). This project implements graph traversal, sorting and searching algorithms that were taught throughout the year.

Equality Interface Module

Generic Interface Module

Equality(T)

Uses

None

Syntax

Exported Types

None

Exported Constants

None

Exported Access Programs

Routine name	In	Out	Exceptions
equals	T	$\mathbb{B}$	

Considerations

When implementing in Java, Object acts as Equality.

Generic Seq Module

Generic Template Module inherits Equality(Seq(T))

Seq(T with Equality(T))

Uses

Equality

Syntax

Exported Types

Seq(T) = ?

Exported Constants

None

Exported Access Programs

Routine name	In	Out	Exceptions
new Seq(T)		Seq(T)	
new Seq(T)	Sequence of T	Seq(T)	
append	T		
rm	$\mathbb{N}$		IllegalArgumentException
member	T	$\mathbb{B}$	
size		$\mathbb{N}$	
get	$\mathbb{N}$	T	IllegalArgumentException
find	T	$\mathbb{N}$	IllegalArgumentException
count	T	$\mathbb{N}$	

Semantics

State Variables

S: seq of T

State Invariant

None

Access Routine Semantics

new Seq():

- transition: $S := \{\}$
- output: $out :=$ self
- exception: None

new Seq(s):

- transition: $S := s$
- output: $out :=$ self
- exception: None

append(e):

- transition: $S := S||[e]$
- exception: None

rm(x):

- transition: $S := [i : \mathbb{N} | 0 \leq i \leq \text{size}() \wedge i \neq x : S[i]]$
- exception: $exc := 0 \leq x < \text{size}() \Rightarrow$ IllegalArgumentException

member(x):

- output: $out := x \in S$
- exception: None

size():

- output: $out := |S|$
- exception: None

get(x):

- output: $out := S[x]$
- exception: $exc := 0 \leq x < \text{size}() \Rightarrow$ IllegalArgumentException

find(e):

- transition: $S := i : \mathbb{N} | 0 \leq i < \text{size}() \wedge e.\text{equals}(S[i]) : S[i]$
- exception: $exc := e \notin S \Rightarrow \text{IllegalArgumentException}$

count(e):

- output: $out := +(i : \mathbb{N} | 0 \leq i < \text{size}() \wedge e.\text{equals}(S[i]) : 1)$
- exception: None

equals(R):

- output: $out := R.\text{size}() = \text{size}() \wedge \forall(i : N | 0 \leq i < \text{size}() : R.\text{get}[i].\text{equals}(S[i]))$
- exception: None

2 Node Module

Template Module inherits Equality(Node)

Node

Uses

Equality

Syntax

Exported Types

Node = ?

Exported Constants

None

Exported Access Programs

Routine name	In	Out	Exceptions
new Node	$\mathbb{N}$	Node	
getID		$\mathbb{N}$	

Semantics

State Variables

id: $\mathbb{Z}$

State Invariant

None

Assumptions

- All zone IDs are valid in the dataset.

Access Routine Semantics

new Node(x):

- transition: $id := x$
- output: $out :=$ self
- exception: None

getID():

- output: $out := id$
- exception: None

equals(n):

- output: $out := id = n$.getID()
- exception: None

3 Edge Module

Template Module inherits Equality(Edge)

Edge

Uses

Equality, Node

Syntax

Exported Types

Edge = ?

Exported Constants

None

Exported Access Programs

Routine name	In	Out	Exceptions
new Edge	Node, Node, sequence of $\mathbb{N}, \mathbb{R}, \mathbb{R}, \mathbb{R}, \mathbb{R}$	Edge	IllegalArgumentExceptio
start		Node	
end		Node	
data		sequence of $\mathbb{N}, \mathbb{R}, \mathbb{R}, \mathbb{R}, \mathbb{R}$	
update	sequence of $\mathbb{N}, \mathbb{R}, \mathbb{R}, \mathbb{R}, \mathbb{R}$		IllegalArgumentExceptio
weight		$\mathbb{R}$	
comp	Edge, $\supseteq$	$\mathbb{Z}$	IllegalArgumentExceptio

Semantics

State Variables

s: Node
e: Node
$trps$: $\mathbb{N}$
fr: $\mathbb{N}$
tp: $\mathbb{N}$

dst: $\mathbb{N}$
tll: $\mathbb{N}$

State Invariant

None

Access Routine Semantics

new Edge($start, end, data$):

- transition: $s,\ e,\ trps,\ fr,\ tp,\ dst,\ fll := start, end, data[0], data[1], data[2], data[3], data[4]$
- output: $out :=$ self
- exception: $exc := |data| \neq 5 \Rightarrow$ IllegalArgumentException

start():

- output: $out := s$
- exception: None

end():

- output: $out := e$
- exception: None

data():

- output: $out := [trps,\ fr,\ tp,\ dst,\ tll]$
- exception: None

update($data$):

- transition: $trps,\ fr,\ tp,\ dst,\ fll := trps + data[0], fr + data[1], tp + data[2], dst + data[3], tll + data[4]$
- output: $out :=$ self
- exception: $exc := |data| \neq 5 \Rightarrow$ IllegalArgumentException

weight():

- output: $out := dst/(fr + tp)$
- exception: None

comp(e, i):

- output: $out :=$ data$[i] > e$.data$[i] \Rightarrow 1 |$data$[i] < e$.data$[i] \Rightarrow -1 | true \Rightarrow 0$
- exception: $exc := i < 0 \vee i > 5 \Rightarrow$ IllegalArgumentException

equals(e):

- output: $out := s$.equals$(e$.start$()) \wedge e$.equals$(e$.end$())$
- exception: None

4 Graph Module

Template Module

Graph

Uses

Seq, Node, Edge

Syntax

Exported Types

Graph = ?

Exported Constants

None

Exported Access Programs

Routine name	In	Out	Exceptions
new Graph		Graph	
nodeList		Seq(Node)	
edgeList		Seq(Seq(Edge))	
addNode	Node		
addEdge	Edge		IllegalArgumentException
adj	Node	Seq(Edge)	
find	Node	$\mathbb{N}$	
n		$\mathbb{N}$	
m		$\mathbb{N}$	

Semantics

State Variables

nodes: Seq(Node)
edges: Seq(Seq(Edge))

State Invariant

None

Access Routine Semantics

new Graph():

- transition: *n, m, nodes, edges* := $0, 0,$ new Seq(Node), new Seq(Seq(Edge))
- output: *out* := self
- exception: None

nodeList():

- output: *out* := *nodes*
- exception: None

edgeList():

- output: *out* := *edges*
- exception: None

addNode(n):

- transition: *nodes, edges* := *nodes*.member(n) = $false$ ⇒ *nodes*.append(n), *edges*.append(new Seq(Seq(Edge)))|$true$ ⇒ *nodes, edges*
- exception: None

addEdge(e):

- transition: *edges* := *edges.get*(find(e.start())).member(e) = $false$ ⇒ *edges*(find(e.start())).append(e)| $true$ ⇒ *edges*(find(e.start())).get(*edges*(find(e.start())).find(e)).update(e.data())
- exception: *exc* := *nodes*.member(e.start()) = $false \vee$ *nodes*.member(e.end()) = $false$ ⇒ IllegalArgumentException

adj(n):

- output: *out* := findNode(n) $\neq -1$ ⇒ *edges*[findNode(n)]|$true$ ⇒ new Seq(Seq(Edge))

- exception: None

n():

- output: $out := nodes$.size()
- exception: None

m():

- output: $out := +(i : \mathbb{N} | 0 \leq i < \text{n}() : edges[i].\text{size}())$
- exception: None

5 GraphSearch Module

Template Module

GraphSearch

Uses

Seq, Node, Edge, Graph

Syntax

Exported Types

GraphSearch = ?

Exported Constants

None

Exported Access Programs

Routine name	In	Out	Exceptions
new GraphSearch	Graph	GraphSearch	
find	Node	Seq(Edge)	

Semantics

State Variables

g: Graph

State Invariant

None

Access Routine Semantics

new GraphSearch(G):

- transition: $g := G$
- output: $out :=$ self

- exception: None

find(n):

- output: $out := g.\text{adj}(n)$
- exception: None

6 GraphSort Module

Template Module

GraphSort

Uses

Seq, Node, Edge, Graph

Syntax

Exported Types

GraphSort = ?

Exported Constants

None

Exported Access Programs

Routine name	In	Out	Exceptions
new GraphSort	Graph	GraphSort	
sort	$\mathbb{N}, \mathbb{Z}$	Seq(Edge)	IllegalArgumentException

Semantics

State Variables

g: Graph

State Invariant

None

Access Routine Semantics

new GraphSort(G):

- transition: $g := G$
- output: $out :=$ self

- exception: None

sort(i, j):

- output: $out := A$, such that:
  $\forall(e : \text{Edge}|e \in E : E.\text{count}(e) = A.\text{count}(e)) \wedge \forall(k : \mathbb{N}|0 \leq k < A.\text{size}() - 1 : A.\text{get}(k).\text{data}[i] * j \leq A.\text{get}(k+1).\text{data}[i])$
- exception: $exc := \neg((0 \leq i < 5) \wedge (j = 1 \vee j = -1)) \Rightarrow$ IllegalArgumentException

Local Functions

arrange: Seq(Seq(Edge)) $\rightarrow$ Seq(Edge)
sortEdges$(E) = \text{Seq}([i : \mathbb{N}|0 \leq i < E.\text{size}() : j : \mathbb{N}|0 \leq j < E[i].\text{size}() : E[i][j]])$

7 GraphPath Module

Template Module

GraphPath

Uses

Seq, Node, Edge, Graph

Syntax

Exported Types

GraphPath = ?

Exported Constants

None

Exported Access Programs

Routine name	In	Out	Exceptions
new GraphPath	Graph	GraphPath	
path	Node, Node	Seq(Edge)	IllegalArgumentException

Semantics

State Variables

g: Graph

State Invariant

None

Access Routine Semantics

new GraphPath(G):

- transition: $g := G$
- output: $out :=$ self

- exception: None

path(a, b):

- output: $out := p$, such that:
  $\forall(q : \text{Seq(Edge)} | \text{validPath}(a, b, q) : \text{cost}(p) \leq \text{cost}(q)$
- exception: $exc := g.\text{find}(a) = -1 \vee g.\text{find}(b) = -1 \Rightarrow \text{IllegalArgumentException}$

Local Functions

validPath: Node $\times$ Node $\times$ Seq(Edge) $\rightarrow \mathbb{B}$
validPath$(a, b, q) \equiv q.\text{get}(0).\text{start}() = a \wedge q.\text{get}(q.\text{size}() - 1).\text{end}() = b \wedge$
$\forall(i : \mathbb{N} | 0 \leq i < q.\text{size}() - 1 : q.\text{get}(i).\text{end}().\text{equals}(q.\text{get}(i + 1).\text{start}()))$

cost: Seq(Edge) $\rightarrow \mathbb{R}$
$\text{cost}(p) = +(i : \mathbb{N} | 0 \leq i < p.\text{size}() : p.\text{get}(i).\text{cost}())$

8 Parser Module

Module

Parser

Uses

Node, Edge, Graph

Syntax

Exported Types

Parser = ?

Exported Constants

None

Exported Access Programs

Routine name	In	Out	Exceptions
new Parser		Parser	
parse	String	Graph	IllegalArgumentException

Semantics

State Variables

None

State Invariant

The parser is used only on .csv files that have the same format as the one provided in this project.

Assumptions

Access Routine Semantics

new Parser():

- transition: None
- output: *out* := self
- exception: None

parse(f):

- output: *out* := Create a graph g such that:
  the data from the lines in f are added to g
- exception: $exc := f$ does not exist $\Rightarrow$ IllegalArgumentException

9 FileController Module

Module

FileController

Uses

Node, Edge, Graph

Syntax

Exported Types

FileController = ?

Exported Constants

None

Exported Access Programs

Routine name	In	Out	Exceptions
read	String	Graph	IllegalArgumentException
save	Graph, String		

Semantics

State Variables

None

State Invariant

None

Assumptions

The file control is used only to read files that have been created by the save function.

Access Routine Semantics

read(f):

- output: out := Create a graph g such that:
  the data from the lines in f are added to g
- exception: $exc := f$ does not exist $\Rightarrow$ IllegalArgumentException

save(g, f):

- transition: Create a file called f such that:
  the first line contains n, the number of nodes, and m the number of edges in g. The next n lines contain the id of the n nodes and the next m lines contain the starting node id, ending node id, number of trips, fare, tip, distance, and toll of the m edges.
- exception: $exc := f$ does not exist $\Rightarrow$ IllegalArgumentException

10 Cash Me A Taxi Module

Module

CashMeATaxi

Uses

Node, Edge, Graph, GraphSort, GraphSearch, Parse, FileController

Syntax

Exported Types

CashMeATaxi = ?

Exported Constants

None

Exported Access Programs

Routine name	In	Out	Exceptions
init			
read	String	Graph	
save	String		
load	String		
search	Node	Seq(Edge)	
sort	$\mathbb{N}, \mathbb{Z}$	Seq(Edge)	
path	Node, Node	Seq(Edge)	
printout	Seq(Edge), String		
main			

Semantics

State Variables

parser: Parser
control: FileController
g: Graph

State Invariant

None

Access Routine Semantics

init:

- transition: *parser, control,* $g :=$ new Parser(), new FileController(), new Graph()
- exception: None

read(f):

- transition: $g :=$ *parser*.parse(f)
- exception: none

save(f):

- output: *out* := *control*.save(f)
- exception: none

load(f):

- transition: $g :=$ *control*.load(f)
- exception: none

load(f):

- transition: $g :=$ *control*.load(f)
- exception: none

search(n):

- output: *out* := new GraphSearch(g).find(n)
- exception: none

sort(i, j):

- output: *out* := new GraphSort(g).sort(i, j)
- exception: none

path(a, b):

- output: *out* := new GraphPath(g).path(a, b)
- exception: none

printout(e, f):

- output: Create a file called f such that:
  the following data of each Edge in e is printed in order, start, end, number of trips, fare, tips, distance, toll.
- exception: none

main():

- output: Operate Cash Me A Taxi via command line:
  the operations are listed in the following order: enter data file, save graph, load graph, search, sort, and path. Perform the option specified by the user given additional input. Output is printed to a file called "CMATOut.txt".
- exception: none