Date: 09/24/2021

Traveling Salesman Project - Phase 1

Course: MTH6412B

Under the supervision of Prof. Dominique Orban

Names:

- 1]Mozhgan Moeintaghavi
- 2]Farhad Rahbarnia

[1] Mathematics and Industrial engineering Faculty, Polytechnique Montréal, **Email**

[2] Mathematics and Industrial engineering Faculty, Polytechnique Montréal, **Email**

Github repository URL:

Code to Github: https://github.com/farhadrclass/mth6412b-starter-code

Introduction

The initial phase of the project entails creating appropriate data structures for the symmetrical traveling salesman problems. In this project, only symmetric graphs with weights in explicit format are considered.

The procedures and outputs of Phase 1 of the project are documented in this report. The Julia programming language is used to create five files of: node.jl, edge.jl, graph.jl, main.jl, and read_stsp.jl. The following sections go through all of the modifications in detail.

An overview of what has been accomplished

We developed our code in the Phase 1 branch after considering the explicit weights in the symmetric problems. We have five significant files in our program: node.jl, edge.jl, graph.jl, read stsp.jl, and main.jl.

In terms of the nodes.jl file, we did not modify it from the original code. We kept this file entirely as it was in the original code.

Also, we have modified the graphs struct to have two parameters of: nodes and edges.

The following are the main steps we have accomplished in this phase of the project:

1. Created an edge.jl file, and an Edge type was introduced to denote the edges of a graph.
2. Modified the graph.jl file, we expanded the graph type to include its edges. Considering non-oriented graphs, we built the algorithm such that makes it possible for the user to add edges to the graph.
3. We printed the edges of the graph.
4. We modified the read*edges() function in the read*stsp.jl file to read the weights of the edges, and then computed the weights.

The edge.jl file

In this code, we generated a data type that has a node for the start and a node for the end, as well as a float64 for the edge's weight. This file is not abstract since all data types (including the node type) are predefined, and the Edge type is a struct in Julia.

In fact, we explicitly included node1 ,and node2 to make it possible to read them faster in the future phases of the project, which would make it easier.

As a result, it defines the edge start and end, and then the edge's weight is determined.

Finally, we can use the show function to output the edge by importing the node.jl file.

Here is the code snippet of the file:

```
import Base.show
include("node.jl")

""" Abstract type from which other edge types will be derived.
"""
# abstract type AbstractEdge end

"""Type to representant the Edges of a graph

Exemple:

    #Old  edge1 = Edge("TO-MTL",(1,2), 100 )
    #edge1= Edge((1,2), 100 )
    edge1= Edge(node1, node2, 50.1)


"""
mutable struct Edge
  # node1::Node{K} # Start node
  node1::AbstractNode #
  # node2::Node{K} # Finish node
  node2::AbstractNode #
  weight::Float64
end



edgeStart(edge) = name(edge.node1)

edgeEnd(edge) = name(edge.node2)




"""Renvoie les données contenues dans le noeud.
EN: Returns the weight contained in the edge """
```

```
weight(edge) = edge.weight

"""Affiche un noeud.
EN: present the edge
"""
# Fix this later
function show(edge)
  println("Starting Point ",edgeStart(edge) ,", EndNode ", edgeEnd(edge), ", weight:
", weight(edge))
end
```

The read_stsp.jl file

We added weight to the one in the original code, and then added the weight to the tuple of edges. We transferred the string weight to float64 (transferring a string to a number) by reading the line by using the parse function. Here, we needed to read the weights as a value because they are explicitly defined. The file header of an STSP file also tells us how it is stored relative to the other edges (The file can be in the matrix or diagonal matrix or long sting with 0 as a marker for the starting and finishing the edges for a specific node)

Moreover, we tested this file for the gr17 instance from the stsp file, and we were unable to plot it using the plot_graph() function since it does not allow us to execute, and there is no way for fixing it. We could, however, plot for the bayg29 instance that is provided in our assignment outputs.

```
while n_data > 0
         n_on_this_line = min(n_to_read, n_data)
         for j = start : start + n_on_this_line - 1
           weight = parse(Float64, data[j+1])
           n_edges = n_edges + 1
           if edge_weight_format in ["UPPER_ROW", "LOWER_COL"]
             edge = (k+1, i+k+2, weight)
           elseif edge_weight_format in ["UPPER_DIAG_ROW", "LOWER_DIAG_COL"]
             edge = (k+1, i+k+1, weight)
           elseif edge_weight_format in ["UPPER_COL", "LOWER_ROW"]
             edge = (i+k+2, k+1, weight)
           elseif edge_weight_format in ["UPPER_DIAG_COL", "LOWER_DIAG_ROW"]
             edge = (i+1, k+1, weight)
           elseif edge_weight_format == "FULL_MATRIX"
             edge = (k+1, i+1, weight)
           else
             warn("Unknown format - function read_edges")
           end
```

The graph.jl file

We modified the graph.jl, and below is an example of how to show the graph. Here, we defined an abstract variable T (like variable K in node.jl file) as a placeholder, and then we added an edge variable. Instead of using the two addnode!() and addedge!() functions, we created the graph by constructor function. The purpose of these functions is to add nodes and edges later on in the future phases of the project.

This file creates graphs from a list of edges and nodes, and can print them using the show function. We also created a function for adding edges to the graph.

Finally, by reading the nodes and edges from the node.jl and edge.jl files, we can output the graph.

```
import Base.show

"""Type abstrait dont d'autres types de graphes dériveront.
EN: Abstract type from which other types of graphs will be derived.
"""
abstract type AbstractGraph{T} end

"""Type representant un graphe comme un ensemble de noeuds.
En: Type representing a graph as a set of nodes.

Exemple :

    node1 = Node("Joe", 3.14)
    node2 = Node("Steve", exp(1))
    node3 = Node("Jill", 4.12)

    edge1= Edge(node1, node2, 50.1)
    edge2= Edge(node2, node3, 50.1)


    G = Graph("Ick", [node1, node2, node3], [edge1, edge2])

Attention, tous les noeuds doivent avoir des données de même type.
EN: Attention, all nodes must have data of the same type.
"""
mutable struct Graph{T} <: AbstractGraph{T}
  name::String
  nodes::Vector{Node{T}}
  edges::Vector{Edge}
end

"""Ajoute un noeud au graphe.
EN: This Function adds a note to the graph
"""
function add_node!(graph::Graph{T}, node::Node{T}) where T
  push!(graph.nodes, node)
  graph
end


""" This Function adds a Edge to the graph
"""
```

```julia
function add_edge!(graph::Graph{T}, edge::Edge) where T
  push!(graph.edges, edge)
  graph
end

"""Renvoie le nom du graphe.
EN: Returns the name of the graph
"""
name(graph::AbstractGraph) = graph.name

"""Renvoie la liste des noeuds du graphe.
EN: Retuens a list of nodes of the graph
"""
nodes(graph::AbstractGraph) = graph.nodes

"""Renvoie le nombre de noeuds du graphe.
EN: Returns a size or number of nodes of a graph
"""
nb_nodes(graph::AbstractGraph) = length(graph.nodes)


"""
Return the graph edges
"""
edges(graph::AbstractGraph) = graph.edges

"""
Number of Edges
"""
nb_edges(graph::AbstractGraph) = length(graph.edges)



"""Affiche un graphe
EN: Display a graph
"""
function show(graph::Graph)
  println("Graph ", name(graph), " has ", nb_nodes(graph), " nodes.")
  println("Graph ", name(graph), " has ", nb_edges(graph), " Edges.")

  for node in nodes(graph)
    show(node)
  end

  for edge in edges(graph)
    show(edge)
  end
end
```

The main.jl file

To begin, this file imports all of the previous files that we have created.

Second, it reads a stsp file, switches the directory to the data file, and then allows us to select a graph name among several instances. The nodes and edges of the graph are then created.

In the stsp file, some of the instances have the assigned name, and location while some of them are not assigned with values (like in gr17). So, in order to prevent errors, we created a condition to check to see if the name is assigned in the stsp file. Otherwise, we assign a random name and value to it. So, when we read a TSP file, we make a Node list, and if the node has names in the TSP file, we can use them; if not, we would create one. The node list is said to be an array of Nodes. Then we created an edge List which is a type of array of Edge: edge[], and added edge's start node, end node, and the weight to it.

Then, we went through the edge list and the edges of the graph were created.

Finally, Once we have two lists, we generate the graph using data types and can display it using the graph name, edges list, and nodes list parameters (G = Graph(graphName, nodesList, edgesList). So, the final graph is shown in this step.

```
"""
This program reads a symmetric TSP instance whose weights are given in EXPLICIT form
at and builds a corresponding Graph object.
"""

# Import the other files
include("node.jl")
include("edge.jl")
include("graph.jl")
include("read_stsp.jl")


# read the graph from the file
cd("instances\\stsp\\ ")# go to the file for data
graphName = "gr17"  # this name used for name of the graph
fileName = string(graphName,".tsp")
graph_nodes, graph_edges = read_stsp(fileName)


if (length(graph_nodes) > 0) # check to see if the name is assigned in the TSP file,
if not we do something else
    nodesList = Node{typeof(graph_nodes[1][1])}[]
else
    nodesList = Node{Int64}[]
end
```

```
for k=1:length(graph_edges)
    if (length(graph_nodes) > 0) # check to see if the name is assigned in the TSP f
ile, if not we do something else
        node_buff = Node(graph_nodes[k][1], graph_nodes[k][2])
    else
        node_buff = Node(string(k), k ) #name is the same as we assign it
    end
    push!(nodesList,node_buff)
end

 # edge positions
 # go through the edge list and create the edges of the graph


# edgesList=Edge{Int64}[]
edgesList=Edge[]
# edgesList = AbstractEdge[]
 for k = 1 : length(graph_edges)
    for j = 1 : length(graph_edges[k])
        edge_buff=Edge(nodesList[k], nodesList[j], graph_edges[k][j][2])
        push!(edgesList, edge_buff)
    end
  end


# create a graph using data types
G = Graph(graphName, nodesList, edgesList)
show(G)
```

Example of an output of the system

```
Graph gr17 has 17 nodes.
Graph gr17 has 153 edges.
Node 1, data: 1
Node 2, data: 2
Node 3, data: 3
Node 4, data: 4
Node 5, data: 5
Node 6, data: 6
Node 7, data: 7
Node 8, data: 8
Node 9, data: 9
Node 10, data: 10
Node 11, data: 11
Node 12, data: 12
Node 13, data: 13
Node 14, data: 14
Node 15, data: 15
Node 16, data: 16
Node 17, data: 17
Starting Point 1, EndNode 1, weight: 0.0
Starting Point 2, EndNode 1, weight: 633.0
Starting Point 2, EndNode 2, weight: 0.0
Starting Point 3, EndNode 1, weight: 257.0
Starting Point 3, EndNode 2, weight: 390.0
Starting Point 3, EndNode 3, weight: 0.0
Starting Point 4, EndNode 1, weight: 91.0
Starting Point 4, EndNode 2, weight: 661.0
Starting Point 4, EndNode 3, weight: 228.0
Starting Point 4, EndNode 4, weight: 0.0
Starting Point 5, EndNode 1, weight: 412.0
Starting Point 5, EndNode 2, weight: 227.0
Starting Point 5, EndNode 3, weight: 169.0
Starting Point 5, EndNode 4, weight: 383.0
Starting Point 5, EndNode 5, weight: 0.0
Starting Point 6, EndNode 1, weight: 150.0
Starting Point 6, EndNode 2, weight: 488.0
Starting Point 6, EndNode 3, weight: 112.0
Starting Point 6, EndNode 4, weight: 120.0
Starting Point 6, EndNode 5, weight: 267.0
Starting Point 6, EndNode 6, weight: 0.0
Starting Point 7, EndNode 1, weight: 80.0
Starting Point 7, EndNode 2, weight: 572.0
Starting Point 7, EndNode 3, weight: 196.0
Starting Point 7, EndNode 4, weight: 77.0
Starting Point 7, EndNode 5, weight: 351.0
Starting Point 7, EndNode 6, weight: 63.0
Starting Point 7, EndNode 7, weight: 0.0
Starting Point 8, EndNode 1, weight: 134.0
Starting Point 8, EndNode 2, weight: 530.0
Starting Point 8, EndNode 3, weight: 154.0
Starting Point 8, EndNode 4, weight: 105.0
Starting Point 8, EndNode 5, weight: 309.0
Starting Point 8, EndNode 6, weight: 34.0
Starting Point 8, EndNode 7, weight: 29.0
Starting Point 8, EndNode 8, weight: 0.0
Starting Point 9, EndNode 1, weight: 259.0
Starting Point 9, EndNode 2, weight: 555.0
Starting Point 9, EndNode 3, weight: 372.0
Starting Point 9, EndNode 4, weight: 175.0
Starting Point 9, EndNode 5, weight: 338.0
Starting Point 9, EndNode 6, weight: 264.0
Starting Point 9, EndNode 7, weight: 232.0
Starting Point 9, EndNode 8, weight: 249.0
Starting Point 9, EndNode 9, weight: 0.0
Starting Point 10, EndNode 1, weight: 505.0
Starting Point 10, EndNode 2, weight: 289.0
Starting Point 10, EndNode 3, weight: 262.0
Starting Point 10, EndNode 4, weight: 476.0
Starting Point 10, EndNode 5, weight: 196.0
Starting Point 10, EndNode 6, weight: 360.0
Starting Point 10, EndNode 7, weight: 444.0
Starting Point 10, EndNode 8, weight: 402.0
Starting Point 10, EndNode 9, weight: 495.0
Starting Point 10, EndNode 10, weight: 0.0
Starting Point 11, EndNode 1, weight: 353.0
Starting Point 11, EndNode 2, weight: 282.0
Starting Point 11, EndNode 3, weight: 110.0
Starting Point 11, EndNode 4, weight: 324.0
Starting Point 11, EndNode 5, weight: 61.0
Starting Point 11, EndNode 6, weight: 208.0
Starting Point 11, EndNode 7, weight: 292.0
Starting Point 11, EndNode 8, weight: 250.0
Starting Point 11, EndNode 9, weight: 352.0
Starting Point 11, EndNode 10, weight: 154.0
Starting Point 11, EndNode 11, weight: 0.0
Starting Point 12, EndNode 1, weight: 324.0
Starting Point 12, EndNode 2, weight: 638.0
Starting Point 12, EndNode 3, weight: 437.0
Starting Point 12, EndNode 4, weight: 240.0
Starting Point 12, EndNode 5, weight: 421.0
Starting Point 12, EndNode 6, weight: 329.0
Starting Point 12, EndNode 7, weight: 297.0
Starting Point 12, EndNode 8, weight: 314.0
Starting Point 12, EndNode 9, weight: 95.0
Starting Point 12, EndNode 10, weight: 578.0
Starting Point 12, EndNode 11, weight: 435.0
Starting Point 12, EndNode 12, weight: 0.0
Starting Point 13, EndNode 1, weight: 70.0
Starting Point 13, EndNode 2, weight: 567.0
Starting Point 13, EndNode 3, weight: 191.0
Starting Point 13, EndNode 4, weight: 27.0
Starting Point 13, EndNode 5, weight: 346.0
Starting Point 13, EndNode 6, weight: 83.0
Starting Point 13, EndNode 7, weight: 47.0
Starting Point 13, EndNode 8, weight: 68.0
Starting Point 13, EndNode 9, weight: 189.0
Starting Point 13, EndNode 10, weight: 439.0
Starting Point 13, EndNode 11, weight: 287.0
Starting Point 13, EndNode 12, weight: 254.0
Starting Point 13, EndNode 13, weight: 0.0
Starting Point 14, EndNode 1, weight: 211.0
Starting Point 14, EndNode 2, weight: 466.0
Starting Point 14, EndNode 3, weight: 74.0
Starting Point 14, EndNode 4, weight: 182.0
Starting Point 14, EndNode 5, weight: 243.0
Starting Point 14, EndNode 6, weight: 105.0
Starting Point 14, EndNode 7, weight: 150.0
Starting Point 14, EndNode 8, weight: 108.0
Starting Point 14, EndNode 9, weight: 326.0
Starting Point 14, EndNode 10, weight: 336.0
Starting Point 14, EndNode 11, weight: 184.0
Starting Point 14, EndNode 12, weight: 391.0
Starting Point 14, EndNode 13, weight: 145.0
Starting Point 14, EndNode 14, weight: 0.0
Starting Point 15, EndNode 1, weight: 268.0
Starting Point 15, EndNode 2, weight: 420.0
Starting Point 15, EndNode 3, weight: 53.0
Starting Point 15, EndNode 4, weight: 239.0
Starting Point 15, EndNode 5, weight: 199.0
Starting Point 15, EndNode 6, weight: 123.0
Starting Point 15, EndNode 7, weight: 207.0
Starting Point 15, EndNode 8, weight: 165.0
Starting Point 15, EndNode 9, weight: 383.0
Starting Point 15, EndNode 10, weight: 240.0
Starting Point 15, EndNode 11, weight: 140.0
Starting Point 15, EndNode 12, weight: 448.0
Starting Point 15, EndNode 13, weight: 202.0
Starting Point 15, EndNode 14, weight: 57.0
Starting Point 15, EndNode 15, weight: 0.0
Starting Point 16, EndNode 1, weight: 246.0
Starting Point 16, EndNode 2, weight: 745.0
Starting Point 16, EndNode 3, weight: 472.0
Starting Point 16, EndNode 4, weight: 237.0
Starting Point 16, EndNode 5, weight: 528.0
Starting Point 16, EndNode 6, weight: 364.0
Starting Point 16, EndNode 7, weight: 332.0
Starting Point 16, EndNode 8, weight: 349.0
Starting Point 16, EndNode 9, weight: 202.0
Starting Point 16, EndNode 10, weight: 685.0
Starting Point 16, EndNode 11, weight: 542.0
Starting Point 16, EndNode 12, weight: 157.0
Starting Point 16, EndNode 13, weight: 289.0
Starting Point 16, EndNode 14, weight: 426.0
Starting Point 16, EndNode 15, weight: 483.0
Starting Point 16, EndNode 16, weight: 0.0
Starting Point 17, EndNode 1, weight: 121.0
Starting Point 17, EndNode 2, weight: 518.0
Starting Point 17, EndNode 3, weight: 142.0
Starting Point 17, EndNode 4, weight: 84.0
Starting Point 17, EndNode 5, weight: 297.0
Starting Point 17, EndNode 6, weight: 35.0
Starting Point 17, EndNode 7, weight: 29.0
Starting Point 17, EndNode 8, weight: 36.0
Starting Point 17, EndNode 9, weight: 236.0
Starting Point 17, EndNode 10, weight: 390.0
Starting Point 17, EndNode 11, weight: 238.0
Starting Point 17, EndNode 12, weight: 301.0
Starting Point 17, EndNode 13, weight: 55.0
Starting Point 17, EndNode 14, weight: 96.0
Starting Point 17, EndNode 15, weight: 153.0
Starting Point 17, EndNode 16, weight: 336.0
Starting Point 17, EndNode 17, weight: 0.0
```

Conclusion

In this assignment, we created a program that reads a symmetric TSP instance and generates a Graph defined by its nodes and edges based on explicit weights. Finally, we displayed the graph by printing the nodes, and edges.