Rapport de projet: Phase 1

Équipe: Elahe Amiri et Louis-Philippe Proulx

Lien Github du code: **Branche Phase 1 du projet**

```
import Pkg; Pkg.add("PlutoUI")
```

```
using PlutoUI
```

```
Main.workspace3.plot_graph
```

```
begin
    include("node.jl")
    include("edge.jl")
    include("graph.jl")
    include("read_stsp.jl")
end
```

On pointe vers un des fichier sources de façon à ce que le code fonctionne sur différent poste de travail

```
"C:\\Users\\lppro\\OneDrive\\Documents\\Poly\\Cours\\MTH6412B\\code\\project\\mth6412b-starter-code\\instan
ces\\stsp\\bayg29.tsp"
```

```
begin
    filename_stsp = "bayg29.tsp"
    root = normpath(joinpath(@__FILE__,"..","..",".."))
    filepath_to_stsp = "instances\\stsp"
    filepath = joinpath(root, filepath_to_stsp)
    filepath = joinpath(filepath, filename_stsp)
end
```

Voici le type Edge que nous proposons. Il est constitué d'un couple de Noeuds et d'un poids (scalaire réel). Nous lui donnons également un nom de type "(1,2)"

```
md" Voici le type Edge que nous proposons. Il est constitué d'un couple de Noeuds et d'un poids
(scalaire réel). Nous lui donnons également un nom de type \"(1,2)\""
```

```
"""Abstract type from which other types of edges will derive."""
abstract type AbstractEdge{T} end

"""Type representing an edge as a set of nodes.
```

```
Exemple :
    node1 = Node("1", 3.14)
    node2 = Node("2", exp(1))
    E = Edge("(1,2)", edges_weight[1,2], (node1 , node2))

Be careful, all nodes must have data of the same type.
"""
mutable struct Edge{T} <: AbstractEdge{T}
  name::String
  weight::Float64
  adjacentnodes::Tuple{Node{T},Node{T}}
end


# we assume that all edges deriving from AbstractEdge
# will have `weight` and` nodes` fields.

"""Returns the weight of the edge."""
weight(edge::AbstractEdge) = edge.weight

"""Returns the adjacent nodes of the edge."""
adjacentnodes(edge::AbstractEdge) = edge.adjacentnodes


"""Display an edge"""
function show(edge::Edge)
  println("Between Node ", name(edge.adjacentnodes[1]), " and Node ", name(edge.adjacentnodes
[2]), " edge weight ", edge.weight)
end
```

Voici le type Graph que nous proposons. Nous avons ajouté un vecteur d'arrêtes et la fonction correspondante pour ajouter un arrête à la fois. La fonction "show" a été modifié afin d'afficher les arrêtes.

```
import Base.show

"""Type abstrait dont d'autres types de graphes dériveront."""
abstract type AbstractGraph{T} end

"""Type representant un graphe comme un ensemble de noeuds.

Exemple :

    node1 = Node("Joe", 3.14)
    node2 = Node("Steve", exp(1))
    node3 = Node("Jill", 4.12)
    G = Graph("Ick", [node1, node2, node3])

Attention, tous les noeuds doivent avoir des données de même type.
"""
mutable struct Graph{T} <: AbstractGraph{T}
  name::String
  nodes::Vector{Node{T}}
  edges::Vector{Edge{T}}
end

"""Adds a node to the graph."""
function add_node!(graph::Graph{T}, node::Node{T}) where T
  push!(graph.nodes, node)
  graph
end

"""Adds an edge to the graph."""
```

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


# we assume that all graphs deriving from AbstractGraph
# will have fields `name` and `nodes`.

"""Returns the name of the graph."""
name(graph::AbstractGraph) = graph.name

"""Returns the list of nodes of the graph."""
nodes(graph::AbstractGraph) = graph.nodes

"""Returns the number of nodes in the graph."""
nb_nodes(graph::AbstractGraph) = length(graph.nodes)

"""Returns the list of edges of the graph."""
edges(graph::AbstractGraph) = graph.edges

"""Returns the number of edges in the graph."""
nb_edges(graph::AbstractGraph) = length(graph.edges)

"""Display a graph"""
function show(graph::Graph)
  println("Graph ", name(graph), " has ", nb_nodes(graph), " nodes.")
  for node in nodes(graph)
    show(node)
  end
  println("Graph ", name(graph), " has ", nb_edges(graph), " edges.")
  for edge in edges(graph)
    show(edge)
  end
end
```

Exemple de création d'un graph

Lecture du fichier, la fonction a également été modifié afin de retourner un dictionnaire contenant le poids des arrêtes

```
(Dict(18 => Float64[460.0, 860.0], 2 => Float64[630.0, 1660.0], 16 => Float64[1280.0,
```

```
graph_nodes, graph_edges, edges_weight = read_stsp(filepath)
```

Création de notre graph vide. On lui donne le nom du fichier source (.tsp)

```
my_graph = Graph{Array{Float64,1}}("bayg29.tsp", Node[], Edge[])
```

```
my_graph = Graph(filename_stsp,Node{Array{Float64,1}}[],Edge{Array{Float64,1}}[])
```

On ajoute les noeuds et les arrête à notre objet "my_graph" en itérant sur les données récoltées du fichier

```
for k = 1 : length(graph_edges)
```

```
.     #T = valtype(graph_nodes)
.     #node = Node{T}(string(k),graph_nodes[k])
.     new_node1 = Node(string(k),graph_nodes[k])
.     add_node!(my_graph, new_node1)
.     for j in graph_edges[k]
.         new_node2 = Node(string(j),graph_nodes[j])
.         edge_name = "("*string(k)*","*string(j)*")"
.         new_edge = Edge(edge_name, edges_weight[k,j], (new_node1 , new_node2))
.         add_edge!(my_graph, new_edge)
.     end
. end
```

Voici le résultat. Le graph contient les noeuds et les arrêtes.

```
  Graph{Array{Float64,1}}(
  name = "bayg29.tsp"
  nodes =   Node[
              1:   Node{Array{Float64,1}}("1",  Float64[1150.0,  1760.0])
              2:   Node{Array{Float64,1}}("2",  Float64[630.0,  1660.0])
              3:   Node{Array{Float64,1}}("3",  Float64[40.0,  2090.0])
              4:   Node{Array{Float64,1}}("4",  Float64[750.0,  1100.0])
              5:   Node{Array{Float64,1}}("5",  Float64[750.0,  2030.0])
              6:   Node{Array{Float64,1}}("6",  Float64[1030.0,  2070.0])
              7:   Node{Array{Float64,1}}("7",  Float64[1650.0,  650.0])
              8:   Node{Array{Float64,1}}("8",  Float64[1490.0,  1630.0])
              9:   Node{Array{Float64,1}}("9",  Float64[790.0,  2260.0])
              10:   Node{Array{Float64,1}}("10",  Float64[710.0,  1310.0])
              11:   Node{Array{Float64,1}}("11",  Float64[840.0,  550.0])
              12:   Node{Array{Float64,1}}("12",  Float64[1170.0,  2300.0])
              13:   Node{Array{Float64,1}}("13",  Float64[970.0,  1340.0])
              14:   Node{Array{Float64,1}}("14",  Float64[510.0,  700.0])
              15:   Node{Array{Float64,1}}("15",  Float64[750.0,  900.0])
              16:   Node{Array{Float64,1}}("16",  Float64[1280.0,  1200.0])
              17:   Node{Array{Float64,1}}("17",  Float64[230.0,  590.0])
              18:   Node{Array{Float64,1}}("18",  Float64[460.0,  860.0])
              19:   Node{Array{Float64,1}}("19",  Float64[1040.0,  950.0])
              20:   Node{Array{Float64,1}}("20",  Float64[590.0,  1390.0])
              21:   Node{Array{Float64,1}}("21",  Float64[830.0,  1770.0])
              22:   Node{Array{Float64,1}}("22",  Float64[490.0,  500.0])
              23:   Node{Array{Float64,1}}("23",  Float64[1840.0,  1240.0])
              24:   Node{Array{Float64,1}}("24",  Float64[1260.0,  1500.0])
              25:   Node{Array{Float64,1}}("25",  Float64[1280.0,  790.0])
              26:   Node{Array{Float64,1}}("26",  Float64[490.0,  2130.0])
              27:   Node{Array{Float64,1}}("27",  Float64[1460.0,  1420.0])
              28:   Node{Array{Float64,1}}("28",  Float64[1260.0,  1910.0])
              29:   Node{Array{Float64,1}}("29",  Float64[360.0,  1980.0])
            ]
  edges =
    Edge[Edge{Array{Float64,1}}("(1,2)",  97.0,  (Node{Array{Float64,1}}("1",  Float64[115
)
```

```
. my_graph
```