MTH6412B: Projet Voyageur de Commerce

Phase 2: Arbres de recouvrement minimaux

- **Auteur**: El Hadji Abdou Aziz Ndiaye (1879468)
- **Code source**: **Repertoire Github**

Importation du code

```
run_test_kruskal (generic function with 1 method)
```

```
begin
    using PlutoUI
    using Plots
    include("node.jl")
    include("edge.jl")
    include("read_stsp.jl")
    include("graph.jl")
    include("min_span_tree.jl")
    include("tests/test_node.jl")
    include("tests/test_edge.jl")
    include("tests/test_graph.jl")
    include("tests/test_min_span_tree.jl")
```

Révision du code de la phase 1

La structure `Edge` a été modifiée. Elle contient maintenant 4 champs:

- `name` : nom de l'arête
- `start_node` : premier noeud de l'arête
- `end_node` : deuxième noeud de l'arête
- `weight` : poids de l'arête

Lors de la construction d'un graphe à travers un fichier *stsp* , les arêtes sont maintenant stockés une seule fois.

Des tests unitaires sont ajoutés pour les structures **`Node`** , **`Edge`** et **`Graph`** .

```
Test de la structure de données `Node` : -v
Test de la structure de données `Edge` : -v
Test de la structure de données `Graph` : -v
```

Structure de données des composantes connexes

Implémentation

La structure de données des composantes connexes est un graphe. Chaque noeud de ce graphe est de type **`Component`** .

Chaque noeud du graph est associé à un `Component` .

La structure `Component` comporte 2 champs:

- `node` : Le noeud auquel `Component` est associé.
- `parent` : Le parent du noeud `node` (le lien de parenté est issu de la connexité).

La structure de données **`ConnectedComponnents`** permet de faire le lien entre les différents `Component` . Cette structure est représentée par un dictionnaire qui associe chaque noeud du graphe à son composante.

Avec la structure de `ConnectedComponnents` , il est possible d'implémenter deux fonctions:

- *find_root* qui pour un noeud **n1** donné, retourne un noeud **n2** racine de la composante connexe où se trouve le noeud **n1**.
- *union_components!* permet de fusionner deux composantes connexes distinctes.

Tests

Des tests unitaires accompagne cette implémentation.

```
Test de la structure de données `Component` : -v
Test de la structure de données `ConnectedComponents` : -v
```

Algorithme de Kruskal

Implémentation

Grâce à la structure de données des composantes connexes, l'algorithme de Kruskal s'implémente de façon assez directe.

Pour ce faire, les arêtes sont d'abord triées par poids puis parcourues une à une.

Pour chaque arête, on vérifie si ses deux noeuds sont dans la même composante connexes. Pour faire cette vérification, il faut utiliser la fonction *find_root* et comparer les racines des noeuds de l'arête.

Si les deux composantes connexes sont différentes, on les fusionne grâce à la fonction *union_components!*

Test

On utilise le graphe de l'exemple du cours pour tester l'implémentation de l'algorithme de Kruskal:

```
Test de l'algorithme de Kruskal : -v
```

Programme principal

La fonction *main* permet de lire l'ensemble des fichiers contenus dans le repertoire `instances/stsp/`. Pour chaque fichier, on construit le graphe correspondant et on calcule l'arbre de recouvrement minimal.

Résultats du programme principal:

```
File: bayg29.tsp
Reading of header : -v
Reading of nodes : -v
Reading of edges : -v
Arbre de recouvrement minimal : 1319 (poids total) -v
File: bays29.tsp
Reading of header : -v
Reading of nodes : -v
Reading of edges : -v
Arbre de recouvrement minimal : 1557 (poids total) -v
File: brazil58.tsp
Reading of header : -v
Reading of nodes : -v
Reading of edges : -v
Arbre de recouvrement minimal : 17514 (poids total) -v
File: brg180.tsp
Reading of header : -v
Reading of nodes : -v
Reading of edges : -v
```