Rapport phase 2

Corrections et ameliorations apportees a la phase 1

Suites aux corrections recues, nous avons implemente les changements suivants au fichier ../phase 1/graph.jl:

- get_node envoi un warning et nothing si un noeud n'est pas dans le graphe
- La mise en page de la documentation a ete corrigee
- La fonction build_graph est maintenant dans le fichier graph.jl

De plus, nous avons ameliorer, dans la fonction build_graph, la recuperation du nom du graphe. Ce n'est plus le nom du fichier avec son chemin et son extension, mais simplement le nom qui est donne au graphe.

Composantes connexes

Une nouvelle structure

`AbstractConComp`

Type abstrait de composantes connexes

`Component`

Type représentant une composante connexe comme un noeud et sa racine.

La strategie est donc de représenter un arbre couvrant d'un graphe G comme un vecteur de composantes connexes: une par noeuds de G. On a donc besoin de plusieurs fonctions sur les composantes connexes, que nous détaillons dans la partie suivante.

Des fonctions pour modifier les composantes connexes et vecteurs de composantes connexes

`set_root!`

Prend en parametre une composante connexe et un noeud n. Renvoi la composante connexe en ayant changé sa racine pour le noeud n.

`add!`

Renvoi le vecteur de composantes connexes decrivant un arbre couvrant auquel on a ajouté la composante connexe new avec pour racine la composante connexe root

`add!`

Renvoi le vecteur de composantes connexes decrivant un arbre couvrant auquel on a ajouté la composante connexe new avec pour racine la composante connexe root

Renvoi le vecteur de composantes connexes auquel on a ajouté la composante connexe new

Des fonctions utilitaires

`trace_back`

Renvoi la racine de l'arbre auquel appartient new

`to_graph`

Prend en parametre un graphe g et son vecteur de composantes connexes associé.

Construit le graphe correspondant au sous graphe de g décrit par le vecteur de composantes connexes

`to_components`

Renvoi le vecteur des composantes connexes associees a un graphe. Il y en a autant que de nodes dans le graphe

`get_component`

Renvoi l'element du vecteur comp tel que s est le nom du noeud de l'element.

`get_component_index`

Renvoi l'index de l'element du vecteur comp tel que s est le nom du noeud de l'element.

`is_lonely`

Renvoi true si une composante c est sa propre racine, false sinon

Algorithme de Kruskal

`kruskal`

Prend en parametre un graphe - Construit un vecteur de composantes connexes telles que chaque element est un noeud du graphe avec elle meme pour racine - Applique l'algorithme de kruskal au graphe et en garde la progression dans le vecteur de composantes connexes - retourne un objet graphe correspondant a l'arbre couvrant minimal obtenu.

```
"""
Prend en parametre un graphe
    - Construit un vecteur de composantes connexes telles que chaque element est un noeud du graphe avec elle meme pour racine
    - Applique l'algorithme de kruskal au graphe et en garde la progression dans le vecteur de composantes connexes
    - retourne un objet graphe correspondant a l'arbre couvrant minimal obtenu.
"""
function kruskal(g::Graph{T}) where T

    #Construit un vecteur de composantes connexes telles que chaque element est un noeud du graphe avec elle meme pour racine
    comp = to_components(g)

    #Tri les aretes de g par poids croissant
    edge_sorted = sort(edges(g), by=weight)

    for e in edge_sorted
        #Recuperes la composante de chaque extremite de l'arete e
        new1 = get_component(comp, name(ends(e)[1]))
        new2 = get_component(comp, name(ends(e)[2]))

        #Si new1 et new2 ne font pas parti de la meme composante connexe
        if name_og_root(comp, new1) != name_og_root(comp, new2)

            #Si new1 est sa propre racine
            if is_lonely(new1)
                set_root!(new1, node(new2))

            #Sinon Si new2 est sa propre racine
            elseif is_lonely(new2)
                set_root!(new2, node(new1))
            end

        end
    end
    #Renvoi le graphe construit a partir du vecteur de composantes connexes
    return to_graph(comp, g)
end
```

Tests unitaires

Des tests unitaires on été implémentés, en prenant en compte un l'exemple ainsi que des cas limites.