

Implémentation d'algorithmes de recherche opérationnelle. Projet phase 2

Polytechnique Montréal

MTH6412B

Xavier Lebeuf et Geoffroy Leconte

Septembre 2021

```
md"""
• # Implémentation d'algorithmes de recherche opérationnelle. Projet phase 2
•
• Polytechnique Montréal
•
• MTH6412B
•
• Xavier Lebeuf et Geoffroy Leconte
•
• Septembre 2021
• """
```

1. Importation des fichiers

```
md"""
• # 1. Importation des fichiers
• """
```

test_kruskal (generic function with 1 method)

```
• begin
•     using Plots
•     using PlutoUI
•     using Test
•
•     include("mainphase1.jl")
•     include("mainphase2.jl")
• end
```

2. Modifications à la phase 1

Nous avons d'abord ajouté à la fonction `createGraph()` l'ajout de sommets lorsque le fichier `.tsp` ne contient pas de coordonnées pour les noeuds. Ensuite, nous avons créé une fonction qui teste la création de graphe sur tous les fichiers `.tsp`.

```
• md"""
• # 2. Modifications à la phase 1
•
• Nous avons d'abord ajouté à la fonction createGraph() l'ajout de sommets lorsque le
  fichier .tsp ne contient pas de coordonnées pour les noeuds. Ensuite, nous avons
  créé une fonction qui teste la création de graphe sur tous les fichiers .tsp.
• """
```

```
bayg29.tsp ✓
bays29.tsp ✓
brazil58.tsp ✓
brg180.tsp ✓
dantzig42.tsp ✓
fri26.tsp ✓
gr120.tsp ✓
gr17.tsp ✓
gr21.tsp ✓
gr24.tsp ✓
gr48.tsp ✓
hk48.tsp ✓
pa561.tsp ✓
swiss42.tsp ✓
```

```
• with_terminal() do
•     test_creation_graphe(raw"./instances/stsp")
• end
```

3. Création d'une structure ConnexComp

Cette structure ne contient qu'un vecteur contenant les noms des sommets faisant partie de la composante connexe. Elle a été définie dans le fichier `graph.jl` à la ligne 23.

```
• md"""
• # 3. Création d'une structure ConnexComp
•
• Cette structure ne contient qu'un vecteur contenant les noms des sommets faisant
  partie de la composante connexe. Elle a été définie dans le fichier graph.jl à la
  ligne 23.
• """
```

4 et 5. Implémentation de l'algorithme de Kruskal et tests unitaires

L'algorithme a été implémenté sous forme de fonction prenant en argument un graphe et renvoyant l'arbre de moindre coût recouvrant le graphe. L'algorithme suit les étapes montrées dans les notes de cours.

Dans la cellule suivante, on performe l'algorithme sur le graphe en exemple dans le cours. Les arêtes de l'arbre trouvé sont affichées et cette solution diffère de celle des notes de cours. Toutefois, elle est également optimale. Deux tests sont ensuite effectués. Premièrement, le nombre d'arêtes dans l'arbre doit correspondre au nombre de sommets du graphe original - 1. C'est une propriété des arbres de recouvrement si le graphe est connexe. Deuxièmement, on teste l'optimalité en vérifiant que la somme des coûts est de 37, comme dans les notes de cours.

- `md"""`
- `# 4 et 5. Implémentation de l'algorithme de Kruskal et tests unitaires`
- `"""`
- L'algorithme a été implémenté sous forme de fonction prenant en argument un graphe et renvoyant l'arbre de moindre coût recouvrant le graphe. L'algorithme suit les étapes montrées dans les notes de cours.
- Dans la cellule suivante, on performe l'algorithme sur le graphe en exemple dans le cours. Les arêtes de l'arbre trouvé sont affichées et cette solution diffère de celle des notes de cours. Toutefois, elle est également optimale. Deux tests sont ensuite effectués. Premièrement, le nombre d'arêtes dans l'arbre doit correspondre au nombre de sommets du graphe original - 1. C'est une propriété des arbres de recouvrement si le graphe est connexe. Deuxièmement, on teste l'optimalité en vérifiant que la somme des coûts est de 37, comme dans les notes de cours.
- `"""`

```
Main.workspace2.Edge{String, Int64}("g↔h", ("g", "h"), 1)
Main.workspace2.Edge{String, Int64}("f↔g", ("f", "g"), 2)
Main.workspace2.Edge{String, Int64}("c↔i", ("c", "i"), 2)
Main.workspace2.Edge{String, Int64}("a↔b", ("a", "b"), 4)
Main.workspace2.Edge{String, Int64}("c↔f", ("c", "f"), 4)
Main.workspace2.Edge{String, Int64}("c↔d", ("c", "d"), 7)
Main.workspace2.Edge{String, Int64}("b↔c", ("b", "c"), 8)
Main.workspace2.Edge{String, Int64}("d↔e", ("d", "e"), 9)
G exemple du cours ✓
```

```
• begin
•     arbrecoutmin = kruskal(Gexcours)
•     with_terminal() do
•         for e in arbrecoutmin
•             println(e)
•         end
•         @test length(arbrecoutmin) == nb_nodes(Gexcours) - 1
•         @test sommeweights(arbrecoutmin) == 37
•         println("G exemple du cours ✓")
•     end
• end
```

6. Test sur toutes les instances

La fonction `test_kruskal()` performe l'algorithme sur tous les fichiers `.tsp`.

```
• md"""
• # 6. Test sur toutes les instances
•
• La fonction test_kruskal() performe l'algorithme sur tous les fichiers .tsp.
• """
```

```
bayg29.tsp ✓
bays29.tsp ✓
brazil58.tsp ✓
brg180.tsp ✓
dantzig42.tsp ✓
fri26.tsp ✓
gr120.tsp ✓
gr17.tsp ✓
gr21.tsp ✓
gr24.tsp ✓
gr48.tsp ✓
hk48.tsp ✓
pa561.tsp ✓
swiss42.tsp ✓
```

```
• with_terminal() do
•     test_kruskal(raw"./instances/stsp")
• end
```

7. Branche phase2 sur GitHub

Lien: <https://github.com/XavierLebeuf/mth6412b-starter-code/tree/phase2>

- `md"""`
- `# 7. Branche phase2 sur GitHub`
- `•`
- `Lien: https://github.com/XavierLebeuf/mth6412b-starter-code/tree/phase2`
- `"""`