

UFR MATHÉMATIQUES ET INFORMATIQUE



Université
de Paris

ALGORITHMIQUE AVANCÉE

IF05X040

Travaux Dirigés 6-7

Prof: Nicolas LOMÉNIE

Author: Letao WANG

November 27, 2021

Partie exercices

Exercice 2.7.

Déterminer un arbre couvrant de poids minimal pour le graphe de la figure 2.3

Réponse. On peut utiliser l'algo de Prim et Kruskal, les resultat sont meme, le poids minimal est 12.

Exercice 2.8.

On considère un graphe valué G et un ensemble U de sommets. Proposer un algorithme qui détermine un arbre couvrant de poids minimal parmi ceux tels que tous les sommets de U sont des feuilles. Démontrer sa validité et préciser sa complexité.

Réponse.

Exercice 2.9.

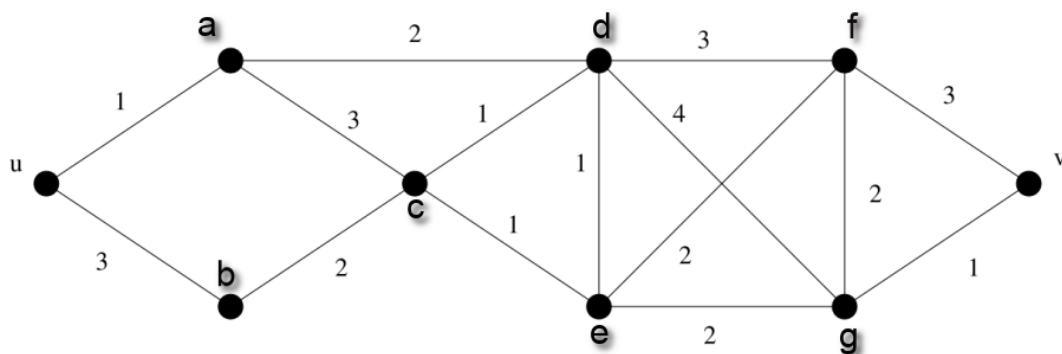


FIGURE 2.4 – .

Réponse.

l'algorithme de Dijkstra

Sommet	u	a	b	c	d	e	f	g	v
Distance	0	1	3	∞	∞	∞	∞	∞	∞
Distance	0	1	3	4	3	∞	∞	∞	∞
Distance	0	1	3	4	3	4	6	7	∞
Distance	0	1	3	4	3	4	6	6	∞
Distance	0	1	3	4	3	4	6	6	7
Distance	0	1	3	4	3	4	6	6	7

Exercice 2.10

A partir de l'algorithme de Dijkstra, construire un algorithme qui détermine le diamètre de chaque composante connexe d'un graphe. Quelle en est la complexité ?

Réponse. On fait l'algo Dijkstra pour tous les sommets dans chaque composante connexe, retourner le plus grand nombre.

Complexity : $n \cdot \text{algo Dijkstra}$ (on considère c'est $O(m + n \log n)$ à l'aide de tas de Fibonacci)

Donc c'est $O(n(m + n \log n))$

Exercice 3.1.

Réponse.

	G1	G2	G3
Chemin eulérien	true	true	false
Cycle eulérien	false	true	false
Chemin hamiltonien	true	false	true
Cycle hamiltonien	true	false	false

Exercice 3.2.

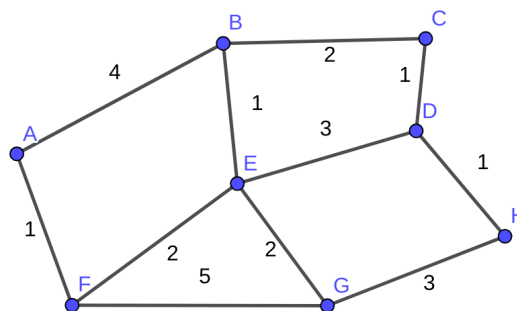


FIGURE 3.2

Réponse. Au premier, il faut trouver les sommets qui degré est impair. Donc on a sommet F, B, G, D. Ensuite on construit le Figure 3.2a

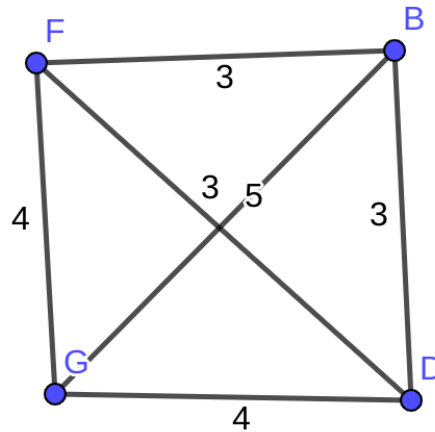


Figure 3.2a

Selon le Figure 3.2a, on veut connecter les 4 sommets avec le minimal poid, on peut choisir Edge(F, G) et Edge(B, D).

On peut ajouter un Edge(B, D) dans le Figure 3.2, donc il y a seulement 2 sommets impaire, on peut trouver un chemin eulerien.

Le poids minimal total est l'addition de tous les poids de Figure3.2 et Edge(F,G), Edge(B,D).

Tous les poids de Figure3.2 : $1 + 4 + 2 + 1 + 1 + 3 + 2 + 2 + 5 + 1 + 3 = 25$

Avec les 2 autre edges : $25 + 3 + 4 = 32$

Donc le poids minimal total est **32**.

Partie codes

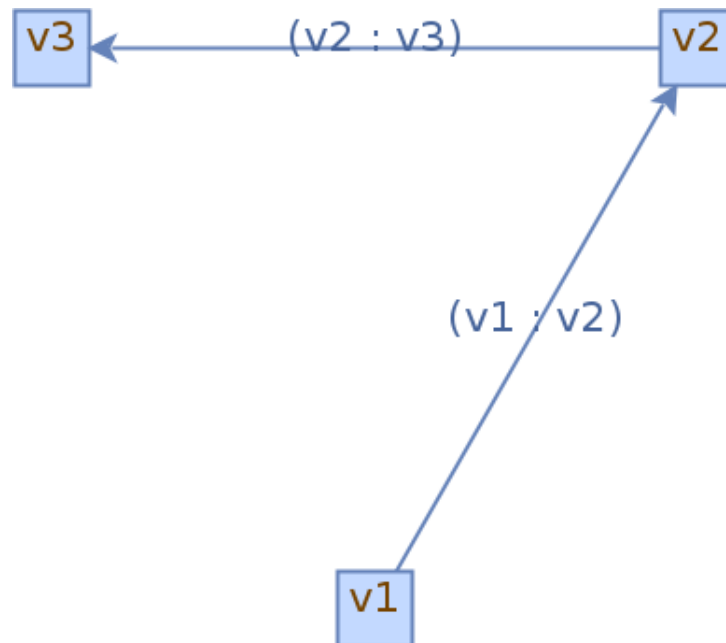
TP6.B. Visualisation de Graphes

JGraphT. Je utilise le bibliotheque JGraphT pour visualiser graphe. Une partie de mon code d'affichage.

Listing 1: GrapheGenerate

```
1 public class GrapheGenerate {
2
3     DefaultDirectedWeightedGraph<String , DefaultEdge> g;
4
5     public GrapheGenerate() {}
6
7     public void createGraph() {
8         g = new DefaultDirectedWeightedGraph<>
9             (DefaultEdge.class);
10
11         g.addVertex("v1");
12         g.addVertex("v2");
13         g.addVertex("v3");
14
15         g.addEdge("v1", "v2");
16         g.addEdge("v2", "v3");
17     }
18
19     public void visualiser() throws IOException {
20         JGraphXAdapter<String , DefaultEdge> graphAdapter =
21             new JGraphXAdapter<>(g);
22
23         mxIGraphLayout layout =
24             new mxCircleLayout(graphAdapter);
25
26         layout.execute(graphAdapter.getDefaultParent());
27         BufferedImage image =
28             mxCellRenderer.createBufferedImage
29             (graphAdapter, null, 2, Color.WHITE, true, null);
30
31         File imgFile = new File("src/main/java/graph.png");
32         ImageIO.write(image, "PNG", imgFile);
33     }
34 }
```

Voici le resultat du code:



Neo4j. Neo4j est une base de données non relationnelle, une base de données graphique native. Dans une base de données relationnelle normale, la recherche d'un élément de données nécessite souvent l'interrogation de plusieurs tables de la base de données, en particulier pour les grands projets qui nécessitent de nombreuses et longues requêtes sautantes. Les performances de ce type de base de données sont réduites dans ce scénario. La base de données de graphes introduit le concept de Edge entre les Vertex, ce qui améliore considérablement les performances dans ce scénario.

Neo4j Java

Listing 2: Neo4j avec Java exemple

```
1 ...  
2 firstNode = tx.createNode();  
3 firstNode.setProperty( "message", "Node1 " );  
4 secondNode = tx.createNode();  
5 secondNode.setProperty( "message", "Node2" );  
6  
7 relationship = firstNode.createRelationshipTo  
8 ( secondNode, RelTypes.KNOWS );  
9  
10 relationship.setProperty( "message", "relation12 " );  
11 ...
```

Voici le resultat dans le terminale:
Node1 relation12 Node2

Neo4j browser Neo4j browser est un interface graphique pour modifier les datas de database. Il utilise Cypher au lieu de SQL language.

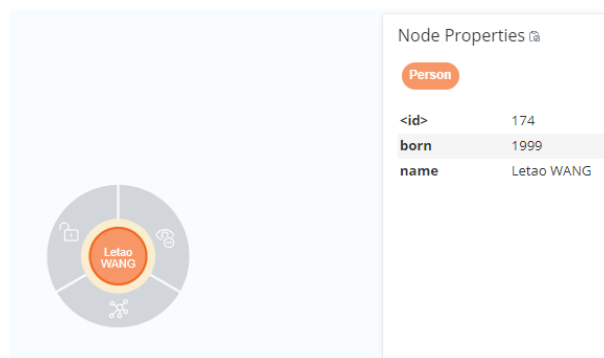
Create record:

```
CREATE (Letao:Person name:'Letao WANG', born:1999)
```

```
1 CREATE (UniversityOfParis:University
2   {title:'University of Paris', released:2019,
3     tagline:'Welcome to the University of Paris'})
4 CREATE (Taole:Person {name:'Taole WANG', born:1999})
5 CREATE (Stan:Person {name:'Stan Ma', born:1999})
6 CREATE (Paul:Person {name:'Paul Hu', born:1999})
7
8 CREATE (Taole)-[:STUDIED_IN {roles:['Student']}]>
9   (UniversityOfParis)
```

Query:

```
MATCH (Letao:Person name: "Letao WANG") RETURN Letao
```



Et on peut identifier la relation entre Person et University aussi, par exemple on cherche les personnes qui etudier a Universite de Paris

neo4j\$ `MATCH (people:Person)-[relatedTo]-(:University {title:"University of Paris"}) RETURN people.name, Type(relatedTo), relatedTo.roles`

	people.name	Type(relatedTo)	relatedTo.roles
1	"Taole WANG"	"STUDIED_IN"	["Student"]

Autre exemple de Cypher.