Exercice 1 : Recherche et Parcours de Graphes

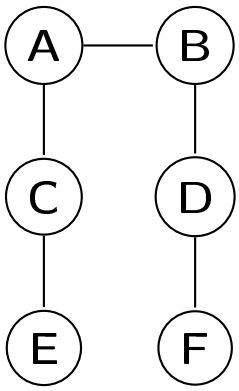

Figure 1: Exemple de graphe

1. **Implémentation de DFS et BFS**
   (a) Implémenter les algorithmes en Python.
   (b) Tester les algorithmes sur le graphe donné (voir Figure 1).
2. **Exercices sur la détection de cycles et la recherche de composantes connexes**
   (a) Utiliser DFS pour détecter des cycles dans le graphe.
   (b) Utiliser BFS pour trouver toutes les composantes connexes d'un graphe non orienté.
3. **Analyse de la complexité**
   (a) Analyser la complexité temporelle des algorithmes DFS et BFS.
   (b) Analyser la complexité spatiale des algorithmes DFS et BFS.
   (c) Discuter les cas où l'un est préférable à l'autre.

Exercice 2 : Algorithme de Dijkstra

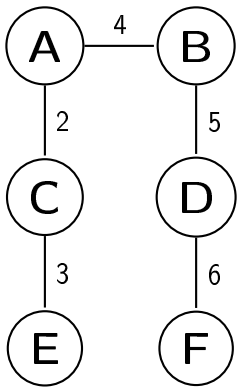

Figure 2: Exemple de graphe pondéré

1. **Implémentation de l'algorithme de Dijkstra**
   (a) Écrire le pseudo-code pour l'algorithme de Dijkstra.
   (b) Implémenter l'algorithme en Python.
   (c) Tester l'algorithme sur le graphe pondéré donné (voir Figure 2).
2. **Résolution de problèmes**
   (a) Résoudre des problèmes de plus court chemin sur le graphe pondéré donné.
   (b) Comparer les résultats avec ceux obtenus par l'algorithme de Bellman-Ford.
3. **Analyse de la complexité**
   (a) Analyser la complexité temporelle de l'algorithme.
   (b) Analyser la complexité spatiale de l'algorithme.

Exercice 3 : Algorithme de Bellman-Ford

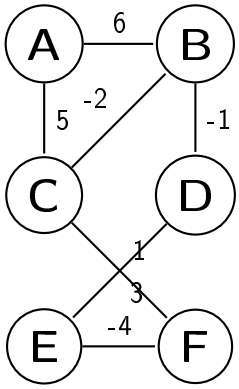

Figure 3: Exemple de graphe avec poids négatifs

1. **Implémentation de l'algorithme de Bellman-Ford**
   (a) Écrire le pseudo-code pour l'algorithme de Bellman-Ford.
   (b) Implémenter l'algorithme en Python.
   (c) Tester l'algorithme sur le graphe avec des poids négatifs donné (voir Figure 3).
2. **Détection de cycles de poids négatif**
   (a) Utiliser l'algorithme pour détecter des cycles de poids négatif dans le graphe.
   (b) Discuter des applications pratiques de cette détection.
3. **Analyse de la complexité**
   (a) Analyser la complexité temporelle de l'algorithme.
   (b) Analyser la complexité spatiale de l'algorithme.

Exercice 4 : Algorithme de Ford-Fulkerson et Edmonds-Karp

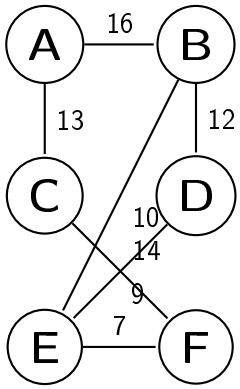

Figure 4: Exemple de réseau de capacités

1. **Implémentation de Ford-Fulkerson**
   (a) Écrire le pseudo-code pour l'algorithme de Ford-Fulkerson.
   (b) Implémenter l'algorithme en Python.
   (c) Tester l'algorithme sur le réseau de capacités donné (voir Figure 4).
2. **Implémentation d'Edmonds-Karp**
   (a) Écrire le pseudo-code pour l'algorithme d'Edmonds-Karp.
   (b) Implémenter l'algorithme en Python.
   (c) Tester l'algorithme sur le même réseau utilisé pour Ford-Fulkerson.
3. **Analyse de la complexité**
   (a) Analyser la complexité temporelle de chaque algorithme.
   (b) Analyser la complexité spatiale de chaque algorithme.
   (c) Discuter des cas où Edmonds-Karp est préférable à Ford-Fulkerson.

Exercice 5 : Tri Rapide Randomisé

10	7	8	9	1	5

Table 1: Tableau initial pour le tri rapide randomisé

1. **Implémentation du tri rapide randomisé**
   (a) Écrire le pseudo-code pour le tri rapide randomisé.
   (b) Implémenter l'algorithme en Python.
   (c) Tester l'algorithme sur différents jeux de données (par exemple : [10, 7, 8, 9, 1, 5]).
2. **Comparaison avec le tri rapide déterministe**
   (a) Comparer les performances du tri rapide randomisé avec celles du tri rapide déterministe.
   (b) Discuter des avantages et des inconvénients de chaque approche.
3. **Analyse de la complexité**
   (a) Analyser la complexité moyenne de l'algorithme.
   (b) Analyser la complexité dans le pire cas.
   (c) Discuter des situations où le tri rapide randomisé est préférable.

Exercice 6 : Arbres AVL

10	20	30	40	50	25

Table 2: Séquence d'insertion pour l'arbre AVL

1. **Implémentation des opérations de base sur les arbres AVL**
   (a) Écrire le pseudo-code pour l'insertion dans un arbre AVL.
   (b) Implémenter les fonctions d'insertion et de suppression en Python.
   (c) Tester les opérations sur différents ensembles de données (par exemple : [10, 20, 30, 40, 50, 25]).
2. **Tests de rééquilibrage**
   (a) Réaliser des tests de rééquilibrage après insertion et suppression.
   (b) Observer les rotations (simple et double) et analyser leur impact.
3. **Analyse de la complexité**
   (a) Analyser la complexité temporelle des opérations.
   (b) Analyser la complexité spatiale des opérations.
   (c) Comparer les performances des arbres AVL avec d'autres structures de données arborescentes.

Exercice 7 : Problèmes NP-complets et NP-difficiles

Clause	Variables
1	$(A \vee \neg B)$
2	$(B \vee C \vee \neg D)$
3	$(\neg A \vee D)$

Table 3: Exemple de clauses SAT

	A	B	C	D
A	0	10	15	20
B	10	0	35	25
C	15	35	0	30
D	20	25	30	0

Table 4: Distances pour le problème du voyageur de commerce

1. **Discussion théorique**
   (a) Expliquer les concepts de NP-complet et NP-difficile.
   (b) Discuter de l'importance de ces concepts en algorithmie.
2. **Implémentation de vérificateurs**
   (a) Implémenter un vérificateur pour le problème SAT.
   (b) Implémenter une heuristique pour le problème du TSP.
   (c) Tester les implémentations sur des exemples concrets (par exemple, la formule SAT et le graphe TSP donnés).
3. **Analyse et comparaison**
   (a) Analyser les performances des vérificateurs et des heuristiques.
   (b) Discuter des approches exactes et heuristiques pour résoudre les problèmes NP-complets et NP-difficiles.