Dimanche 2 Février 2025

Rapport Final : Algorithmes pour la Résolution de Problèmes

Title (Titre): Algorithms for Problem Solving and Optimization (Algorithmes pour la résolution de problèmes et l'optimisation)

Introduction (Introduction):

Ce rapport présente l'implémentation et l'analyse de cinq algorithmes fondamentaux utilisés en informatique et en ingénierie logicielle. Ces algorithmes ont été implémentés et testés dans le cadre du cours "Search Based Software Engineering". Nous allons analyser ces algorithmes, en utilisant des exemples et une analyse de la complexité temporelle et spatiale.

- *English:* This report presents the implementation and analysis of five fundamental algorithms used in computer science and software engineering. These algorithms have been implemented and tested in the context of the "Search Based Software Engineering" course. We will analyze these algorithms, using examples, and a time and space complexity analysis.

Exercice 1 : Binary Search (Recherche Binaire)

- **Problem Representation (Représentation du Problème):**
    - Rechercher une valeur cible dans une liste triée d'entiers.
    - Représentation visuelle : liste triée, et division par 2 à chaque étape de la recherche.
- **Solution:**
    - Algorithme de recherche binaire utilisant une approche "diviser pour régner".
    - Implémentation avec une boucle while pour la recherche, des conditions if pour guider la recherche (moitié droite ou gauche).
- **Results (Résultats):**
    - Tableaux avec exemples d'entrée et sortie :
        - [2, 5, 8, 12, 16, 23, 38, 56, 72, 91], target = 23 => index = 5
        - [2, 5, 8, 12, 16, 23, 38, 56, 72, 91], target = 10 => index = -1
- **Code Snippet :** voir le fichier binary_search_interactive.py

Exercice 2 : Graph Traversal (Parcours de Graphes)

- **Problem Representation (Représentation du Problème):**
    - Exploration d'un graphe non orienté pour trouver des chemins et vérifier la connectivité.
    - Représentation visuelle : graphe avec des nœuds et des arêtes, composantes connexes.
- **Solution:**
    - Algorithme BFS : Utilise une file d'attente (queue), parcours de niveau en niveau.

    - Algorithme DFS : Utilise la récursion ou une pile, parcours en profondeur.
    - Implémentation de la fonction are_connected pour la connectivité.
- **Results (Résultats):**
    - *Exemple de graphe :*
      A -- B D -- E -- F \ / C G -- H
    - Exemples d'entrée sortie pour BFS et DFS.
    - Exemples d'entrée/sortie pour la fonction are_connected.
- **Code Snippets :** voir les fichiers graph_algorithms.py, graph_utils.py, main.py

Exercice 3 : Dynamic Programming - Knapsack Problem (Programmation Dynamique - Problème du Sac à Dos)

- **Problem Representation (Représentation du Problème):**
    - Maximiser la valeur des objets dans un sac à dos, en respectant une limite de poids (problème 0/1).
    - Représentation visuelle : un sac à dos et différents objets avec valeurs et poids.
- **Solution:**
    - Implémentation de l'algorithme avec la programmation dynamique, utilisation d'un tableau dp[i][w] pour les sous-problèmes.
    - Reconstruction du chemin pour trouver la liste des indices des objets.
- **Results (Résultats):**
    - Exemples d'entrée sortie :
        - items = [(60, 10), (100, 20), (120, 30)], max_weight = 50 => max_value = 220, indices = [2,1]
        - items = [(10, 2), (20, 3), (30, 4), (40, 5)], max_weight = 10 => max_value = 70, indices = [3, 0]
- **Code Snippet :** voir le fichier knapsack_interactive.py

Exercice 4 : Merge Intervals (Fusion d'Intervalles)

- **Problem Representation (Représentation du Problème):**
    - Fusionner les intervalles de temps qui se chevauchent.
    - Représentation visuelle : intervalle de temps avec des chevauchements.
- **Solution:**
    - Algorithme de fusion : tri par heure de début, puis itération et fusion des intervalles.
- **Results (Résultats):**
    - Exemples d'entrée/sortie :

        - intervals = [(1, 3), (2, 6), (8, 10), (15, 18)] => merged = [(1, 6), (8, 10), (15, 18)]
        - intervals = [(1, 4), (0, 2), (3, 5)] => merged = [(0, 5)]
- **Code Snippet :** voir le fichier merge_intervals_interactive.py

Exercice 5 : Maximum Subarray Sum (Somme Maximale d'un Sous-Tableau)

- **Problem Representation (Représentation du Problème):**
    - Trouver le sous-tableau contigu avec la somme maximale dans un tableau d'entiers.
    - Représentation visuelle : tableau d'entiers, sous-tableau avec somme maximale.
- **Solution:**
    - Algorithme de Kadane : utilisation de variables max_so_far, max_ending_here pour trouver la solution.
- **Results (Résultats):**
    - Exemples d'entrée/sortie :
        - arr = [-2, 1, -3, 4, -1, 2, 1, -5, 4] => max_sum = 6, indices = (3,6)
        - arr = [-1, -2, -3, -4, -5] => max_sum = -1, indices = (0, 0)
- **Code Snippet :** voir le fichier kadane_interactive.py

Performance Analysis (Analyse de la Performance)

Algorithme	Complexité Temporelle	Complexité Spatiale	Description
Binary Search (Recherche Binaire)	O(log n)	O(1)	Recherche efficace dans une liste triée.
BFS (Recherche en largeur)	O(V + E)	O(V)	Parcours de graphe, trouver le plus court chemin dans graphe non pondéré.
DFS (Recherche en profondeur)	O(V + E)	O(V)	Parcours de graphe, exploration en profondeur et détection des cycles.
Knapsack 0/1 (Sac à dos 0/1)	O(n * W)	O(n * W)	Maximisation de la valeur d'objets dans un sac à dos.
Merge Intervals (Fusion d'intervalles)	O(n log n)	O(n)	Fusion d'intervalles qui se chevauchent. Le tri est en O(n log n) et la fusion en O(n).
Kadane's algorithm (Kadane)	O(n)	O(1)	Trouver la somme maximale d'un sous-tableau contigu.

- **Explications:**
    - La **recherche binaire** est très efficace (logarithmique).
    - Les algorithmes **BFS** et **DFS** sont linéaires par rapport au nombre de nœuds et d'arêtes.
    - Le **problème du sac à dos** a une complexité qui dépend du nombre d'objets et du poids maximal.

    - La **fusion d'intervalles** a une complexité en O(n log n) car elle nécessite un tri.
    - L'algorithme de **Kadane** est linéaire, ce qui le rend très efficace.
- *English:* The table summarizes the time and space complexities of all implemented algorithms, which shows the efficiency of each algorithm.

Conclusion (Conclusion)

Ce rapport a présenté l'implémentation et l'analyse de cinq algorithmes fondamentaux.

- **Recherche Binaire :** très efficace pour les recherches dans des listes triées. Sa complexité logarithmique lui permet d'être beaucoup plus rapide que la recherche linéaire pour les grands jeux de données.
- **Algorithmes BFS et DFS :** indispensables pour l'exploration des graphes, le parcours des structures de données non linéaires, et pour les problèmes d'optimisation. Le BFS est idéal pour les chemins les plus courts et le DFS pour les composantes connexes.
- **Problème du Sac à Dos :** la programmation dynamique permet d'optimiser la solution, avec une approche systématique.
- **Fusion d'Intervalles :** l'approche de tri puis fusion permet d'obtenir les intervalles non chevauchants, et permet de résoudre le problème de planification.
- **Algorithme de Kadane :** très efficace (temps linéaire) pour trouver la somme maximale d'un sous-tableau. Son approche basée sur la mise à jour de la somme maximale est simple et très puissante.

Chaque algorithme est utile dans des contextes spécifiques. Le choix de l'algorithme approprié dépend de la nature du problème et des contraintes de performance. La compréhension des forces et des faiblesses de ces différents algorithmes, nous permet de choisir l'algorithme optimal pour un problème spécifique. La manipulation de ces algorithmes nous permettent de construire des bases solides pour aborder des problèmes complexes en informatique.

- *English:* This report presented the implementation and analysis of five fundamental algorithms. Binary search is effective for search in sorted lists. BFS and DFS are essential for graph exploration. Dynamic programming allows optimizing knapsack problem. Merge interval and Kadane's algorithm are very efficient for their respective problems. The choice of each algorithm depends on the nature of the problem and its requirements.

GitHub Repository (Répertoire GitHub) : https://github.com/leonelmaken/SBSE-Algorithms/tree/main

- Le code et ce rapport ont été sauvegardés sur un dépôt GitHub pour le partage et le contrôle de version.