IHDC B232 Algorithmique 1 • Travail individuel
Année académique 2024-2025

1. Consignes générales

Dans le cadre de ce projet, vous êtes invités à choisir un algorithme parmi ceux proposés dans la section « 2. Algorithmes » de ce document. Votre tâche consistera à explorer en profondeur cet algorithme en vous concentrant sur les aspects formels de son fonctionnement et de sa correction, pour lesquels vous appliquerez les méthodes et raisonnements vus au cours.

Le délivrable prendra la forme d'un **rapport formel au format PDF,** dans lequel vous prendrez un soin particulier pour les formules mathématiques et les notations utilisées. Ce rapport peut être produit par le traitement de texte de votre choix (LaTeX, Word, …) ou même être écrit à la main. Veillez, quelle que soit votre préférence à ce niveau, à ce que vos explications et vos calculs soient précis et corrects.

Voici la structure attendue. Pour chaque étape, nous détaillons **en gras** les éléments qui doivent se retrouver dans votre rapport.

1. Effectuez des recherches pour comprendre l'algorithme choisi en profondeur. Vous devrez utiliser des ressources académiques ou en ligne pour mieux comprendre son fonctionnement et les théories qui le sous-tendent. **Listez vos sources, expliquez en quelques mots pourquoi vous avez le choix de cet algorithme en particulier, et donnez son contexte d'utilisation (dans quels cas concrets est-il typiquement utilisé ?).**

2. Spécifiez le problème : **explicitez les préconditions et postconditions formellement** (en respectant les notations mathématiques vues au cours).

3. Implémentez l'algorithme en C. Vous êtes autorisés à vous aider des ressources que vous trouvez, mais l'implémentation doit être votre propre travail : vous devez vous l'approprier et en maîtriser tous les détails. Dans le rapport, **vous donnerez le code (commenté) de votre implémentation, vous détaillerez les structures de données utilisées et expliquer pourquoi elles sont adaptées à l'algorithme choisi.**

4. **En vos propres mots, expliquez comment fonctionne l'algorithme. Décrivez de manière intuitive pourquoi l'implémentation produit un résultat correct par rapport à vos spécifications.**

5. Identifiez un invariant de boucle pertinent pour l'algorithme. **Formulez cet invariant et démontrez, de manière formelle, qu'il est vérifié à chaque itération de la boucle concernée. Expliquez en quoi il permettrait, dans une preuve de programme plus complète, de faire le pont entre pré- et post- conditions.**

6. **Analysez la complexité temporelle et spatiale « pire des cas » de l'algorithme. Justifiez votre analyse en fonction des différentes étapes de l'algorithme et des structures de données utilisées.**

7. **Proposez une version récursive de l'algorithme (ou d'une partie de celui-ci, si cela est pertinent). Formulez une hypothèse d'induction qui servira à démontrer la correction de l'algorithme sur la base des appels récursifs.** Seule l'hypothèse

d'induction, et son impact sur la correction de l'implémentation récursive, doit être formulée formellement ; les autres calculs peuvent être considérés corrects.

8. Enfin, en fonction de l'algorithme que vous aurez choisi, **ajoutez des informations qui vous semblent pertinentes ou sur des aspects caractéristiques à cet algorithme qui sont précisés dans la description de l'algorithme** (voir section 2 ci-dessous).

Remarques :

- Ce projet est individuel et l'entièreté de votre note du cours repose dessus. Vous devrez donc traiter chaque étape avec rigueur.
- N'oubliez pas de commenter votre code pour en faciliter la compréhension.
- Une attention particulière sera portée à la qualité de l'argumentation lors des preuves formelles et à la justesse des explications**.**

2. Algorithmes

Votre choix devra se faire parmi les algorithmes suivants. Nous n'en faisons qu'une très brève description. À vous de vous renseigner et de choisir celui qui vous plaira le plus. (Vous pouvez aussi proposer un algorithme qui ne figure pas dans la liste ; ce choix doit alors être validé par Gonzague avant que vous ne vous lanciez dans le travail.)

A) Algorithmes de recherche dans des graphes

Un graphe est une façon de représenter des données largement utilisée en algorithmique et en mathématique, particulièrement dans le champ de la *théorie des graphes*.

Les algorithmes de graphes sont utilisés pour résoudre des problèmes comme la recherche des plus courts chemins ou la construction d'arbres couvrants minimaux (pour ne citer que ces deux exemples). Ces algorithmes permettent de naviguer efficacement dans des graphes pondérés, où chaque arête (reliant deux nœuds) possède un *coût* (ou un *poids*) associé.

A.1) Algorithme de Dijkstra

L'algorithme de Dijkstra est utile pour trouver **le chemin le moins coûteux entre deux nœuds** d'un graphe pondéré. Il nécessite l'utilisation d'une structure de données efficace comme une file de priorité (souvent implémentée avec un tas). Il est important de prouver que les nœuds déjà visités ont leur plus court chemin correctement calculé.

A.2) Algorithme de Bellman-Ford

Bellman-Ford est une variante permettant de prendre en compte les poids négatifs associés aux arêtes. Cet algorithme inclut un test pour détecter les cycles de poids négatif.

A.3) Algorithme de Prim

L'algorithme de Prim est utilisé pour trouver un arbre couvrant minimal dans un graphe pondéré. L'algorithme s'assure que les nœuds ajoutés ne forment pas de cycles et continue jusqu'à ce que tous les nœuds soient inclus.

A.4) Algorithme de Kruskal

L'algorithme de Kruskal cherche également à trouver un arbre couvrant minimal, mais il fonctionne différemment. Il classe toutes les arêtes du graphe par ordre croissant de poids, puis les ajoute une par une à l'arbre tant qu'elles ne forment pas de cycle.

B) Algorithmes de compression de données

Les algorithmes de compression de données permettent de réduire la taille des fichiers tout en garantissant que les données d'origine peuvent être parfaitement reconstruites.

B.1) Algorithme de Huffman

Huffman utilise un arbre binaire pour attribuer des codes plus courts aux symboles fréquents. Il est fréquent d'optimiser la construction de l'arbre en utilisant une file de priorité.

B.2) Algorithme LZW

LZW se base sur la génération d'un dictionnaire au fur et à mesure que les données sont lues. Cet algorithme est intéressant car il est efficace sur les données contenant beaucoup de redondance.

C) Algorithme de la plus longue sous-séquence commune (LCS)

Le problème dit de LCS (Longest Common Subsequence) consiste à trouver la plus longue sous-séquence commune entre deux séquences données. Contrairement à la recherche de sous-chaînes contiguës, LCS trouve les éléments communs dans le même ordre, même s'ils ne sont pas adjacents dans les séquences d'origine. Par exemple, pour les séquences "ABCBDAB" et "BDCAB", la plus longue sous-séquence commune est "BDAB", car elle apparaît dans cet ordre dans les deux séquences.

Ce problème se résout souvent par programmation dynamique en construisant une matrice pour stocker les résultats intermédiaires. Cet algorithme met en lumière la méthode de programmation dynamique et nécessite une bonne gestion de la mémoire pour optimiser la complexité.

Indication (que vous n'êtes pas obligés de suivre) : construisez une matrice pour stocker les longueurs des sous-séquences communes. Chaque case de la matrice (i, j) contiendra la longueur de la plus longue sous-séquence commune entre les sous-séquences des deux chaînes jusqu'aux indices i et j.

Particularités liées au choix de cet algorithme :

- L'invariant de boucle doit montrer que chaque cellule de la matrice est calculée correctement par rapport aux cellules précédentes.
- Une optimisation de l'espace peut être proposée pour réduire l'utilisation de mémoire.

D) Algorithme de recherche de chemin optimal (A*)

L'algorithme A* est un algorithme de recherche heuristique utilisé pour trouver le chemin le plus court entre deux points dans un graphe (ou un réseau au sens large), en tenant compte à la fois du coût réel déjà parcouru et d'une estimation heuristique du coût restant à parcourir.

Cet algorithme combine des éléments de Dijkstra (coût accumulé) et une fonction heuristique (qui guide la recherche) pour trouver rapidement des solutions optimales. Le choix de la fonction heuristique est essentiel pour garantir la performance de l'algorithme.

Particularités de cet algorithme :

- Implémentez une file de priorité pour gérer les nœuds à explorer.
- Choisissez une heuristique admissible (comme la distance euclidienne ou de Manhattan) pour garantir que l'algorithme trouve le chemin optimal. En quoi est-ce une *heuristique* et non un algorithme « classique » ?
- L'invariant de boucle pourrait être que chaque nœud extrait de la file de priorité a un coût minimum estimé par rapport à tous les nœuds non explorés.
- Prouvez que votre heuristique est admissible, c'est-à-dire qu'elle ne surestime jamais le coût réel.