Documentation du projet de SAE 5.2

Victor Guilleray, Xavier Li, Yassine Elkhalki, Alexis Lemouton, Ostap Tymchyshyn

12 janvier 2025

1 Introduction

Ce document vise à expliquer les différents algorithmes de parcours ou de calculs de graphes utilisés dans le projet de SAE.

2 Algorithmes

2.1 Algorithme de parcours en largeur

L'algorithme de parcours en largeur (BFS, Breadth-First Search en anglais) permet le parcours d'un graphe ou d'un arbre. Le parcours commence à partir d'un nœud source, les voisins du nœud source sont listés, puis les voisins sont explorés à leur tour. Cet algorithme utilise le prince de file FIFO, il prend le premier élément de la file et explore tous ses voisins, ces voisins sont ajoutés à la file et anisi de suite. Les nœuds déjà visités sont marqués afin d'éviter qu'un même nœud soit exploré plusieurs fois. L'algorithme est répété jusqu'à ce qu'un chemin soit établi vers le nœud destination.
Voici les étapes de l'algorithme :

1. Nœud source ajouté dans la file,
2. Retirer le nœud de la file pour l'explorer,
3. Répertorier tous les voisins et les ajouter à la fin de la file,
4. si la file n'est pas vide, répéter l'étape 2.

Illustration :

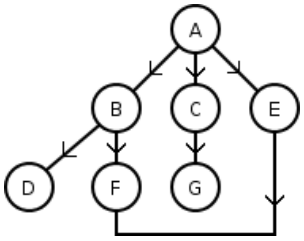

Figure 1 – Exemple pour un algorithme de parcours en largeur

Le parcours de ce graphe se fait dans ce sens : A, B, C, E, D, F, G.
Vous trouverez l'implémentation de l'algortihme ici.

2.2 Algorithme de parcours en profondeur

L'algorithme de parcours en profondeur (DFS, Depth-First Search en anglais) permet le parcours d'un graphe ou d'un arbre. On donne un nœud source à l'algorithme. Il explore un chemin dans le graphe jusqu'à un cul-de-sac (plus de voisins) ou un sommet déjà visité. À partir de ce moment, il revient au dernier sommet où on pouvait choisir un autre chemin et continue le chemin jusqu'à la fin. Le parcours se termine lorsque tous les sommets ont été visités et qu'un chemin a été établi vers le nœud destination.

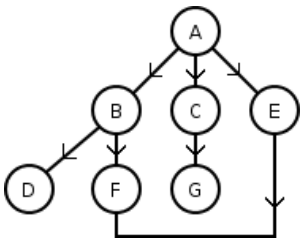

FIGURE 2 – Exemple pour un algorithme de parcours en profondeur

Illustration :
Pour ce graphe, l'algorithme de parcours en profondeur va réaliser ce chemin : A, B, D, F, C, G, E.
Vous trouverez l'implémentation de l'algorithme ici.

2.3 Problème du stable maximum

Définition d'un stable :
Considérons G, un graphe non orienté quelconque. Un ensemble stable de G est un ensemble de sommets non adjacents, qui forme donc un sous-graphe G' sans arêtes. Un ensemble stable est donc un sous-ensemble de sommets du graphe G tel que tout couple de sommets de ce sous-ensemble ne soit pas adjacents (voisins).

Le problème du stable maximum ou maximum independent set problem en anglais, est un problème d'optimisation qui consiste étant donné un graphe non orienté à trouver un stable de cardinal maximum, c'est-à-dire un sous-ensemble de sommets du graphe, le plus grand possible, tel que les éléments de ce sous-ensemble ne soient pas voisins.
Pour résoudre ce problème, L'algorithme se présente de la manière suivante :

1. Récupérer les nœuds de la grille,
2. Pour chaque nœud, déterminer s'il peut être ajouté à un ensemble indépendant (stable ou non).
3. Si le nœud peut être ajouté, il est ajouté à l'ensemble
4. Répéter jusqu'à ce que tous les nœuds aient été parcourus et déterminer quel ensemble a le plus grand cardinal.

2.4 Algorithme de Dijkstra

L'algorithme de Dijkstra est un algorithme de recherche de chemin utilisé pour trouver le chemin le plus court entre un nœud de départ et tous les autres nœuds d'un graphe pondéré. Il calcule des plus courts chemins à partir d'une source vers tous les autres sommets dans un graphe pondéré par des réels positifs. On peut aussi l'utiliser pour calculer un plus court chemin entre un sommet de départ et un sommet d'arrivée.

Il utilise une approche de type "greedy" (gloutonne) pour explorer les nœuds, en choisissant à chaque étape le nœud le plus proche du nœud de départ. Voici les étapes de son fonctionnement :

1. Chaque nœud du graphe est initialisé avec une distance infinie, sauf le nœud de départ, qui est initialisé à 0. Une file de priorité est utilisée pour gérer les nœuds à explorer, en commençant par le nœud de départ.
2. Tant que la file de priorité n'est pas vide, l'algorithme extrait le nœud avec la distance la plus courte. Pour chaque voisin de ce nœud, l'algorithme calcule la distance totale en passant par le nœud actuel. Si cette distance est inférieure à la distance précédemment enregistrée pour le voisin, la distance est mise à jour.
3. Une fois que tous les voisins d'un nœud ont été examinés, le nœud est marqué comme "visité", ce qui signifie que la distance la plus courte vers ce nœud a été trouvée.
4. L'algorithme se termine lorsque tous les nœuds ont été visités ou lorsque le nœud cible a été atteint. À ce stade, il est possible de reconstruire le chemin le plus court en suivant les nœuds précédents.

2.5 Algorithme A étoile

L'algorithme A* (A-star) est un algorithme de recherche de chemin utilisé pour trouver le chemin le plus court entre un point de départ et un point d'arrivée dans un graphe, tout en tenant compte des coûts associés aux déplacements. Voici comment l'algorithme fonctionne :
openSet, une liste de nœuds à évaluer, qui contient le nœud source,
cameFrom, un dictionnaire qui retrace le chemin,
gScore, le coût du chemin le plus court jusqu'au nœud actuel,
fScore, l'estimation du coût total jusqu'au nœud actuel.

1. Le nœud source est ajouté à openSet,
2. Chaque nœud de la grille est associé à un fScore,
3. On récupère le nœud ayant le fScore le plus petit et on l'ajoute à cameFrom,
4. A TERMINER

2.6 Algorithme Bellman-Ford

L'algorithme de Bellman-Ford est utilisé pour résoudre le problème de la recherche de chemin le plus court dans un graphe pondéré. Contrairement à d'autres algorithmes, comme Dijkstra, Bellman-Ford peut gérer des graphes avec des poids d'arêtes négatifs. L'objectif est de trouver le chemin le plus court depuis un nœud source vers tous les autres nœuds du graphe.

2.7 Algorithme de Prim

L'algorithme de Prim est une méthode gloutonne utilisée pour déterminer un arbre couvrant de poids minimal dans un graphe connexe, pondéré et non orienté. Autrement dit, cet algorithme identifie un sous-ensemble d'arêtes qui forment un arbre qui contient tous les sommets du graphe initial mais en minimisant la somme des poids de ces arêtes.

1. Choisir un sommet de départ (nœud source),
2. Sélectionner l'arête en relation de poids minimal et l'ajouter dans l'arbre à la suite du nœud source,
3. Mettre à jour l'arbre en supprimant les arêtes redondantes et refaire la sélection jusqu'à que tous les sommets soit inclus dans l'arbre