HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD
www.heig-vd.ch

Prof. Alberto Dassati,
Assist Bruno Da Rocha
Carvalho

Laboratoire de High Performance Coding

semestre printemps 2023

High Performance Coding (HPC)

Temps à disposition: 4 périodes (deux séances de laboratoire)

Récupération du laboratoire dans l'archive sur Cyberlearn

1 Objectifs de ce laboratoire

Vous écrirez en langage C tout au long de ce cours. Ce laboratoire a pour but de vous remettre dans le bain, et surtout de vous créer une base line pour les futurs laboratoires.

2 Cahier des charges

Il vous est demandé dans ce laboratoire d'implémenter deux versions de l'algorithme A*

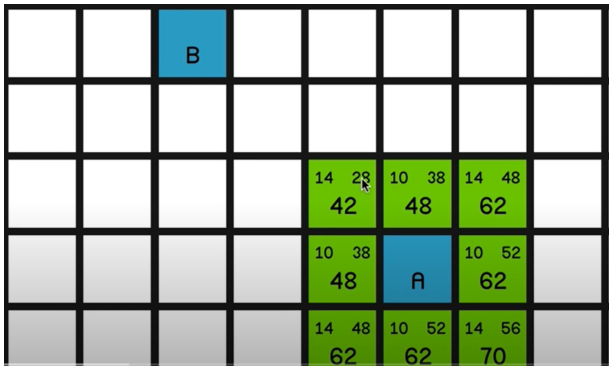
A* est un algorithme permettant de déterminer le chemin le plus court entre un point de départ et un point d'arrivée. Pour cela nous avons donc besoin d'une carte dans laquelle nous allons devoir nous déplacer afin d'y trouver le chemin par lequel passer.

A* se base sur un système de "nodes", chaque case de la map correspond donc à un node. Ils contiennent toutes les informations nécessaires à l'algorithme, comme par exemple la possibilité de se déplacer dessus ou non, sa position dans la carte, etc.

Le principe est de déterminer les coûts d'un déplacement d'une case à une autre, chaque nodes comporte trois informations de coûts :

- **gCost** : Représente la distance entre la position de départ et la position du node courant.
- **hCost** : Représente la distance entre la position d'arrivée B et la position du node courant.
- **fCost** : Est la somme du hCost et du gCost

Sur la figure ci-dessous, on trouve une représentation de ces coûts. Chaque node/case est donc définie par les trois valeurs décrites précédemment. La valeur en haut à gauche est le *gCost*, celle à droite le *hCost* et celle du milieu le *fCost*. Le node A est le point de départ et B la destination.



Dans l'exemple ci-dessus, la case qui sera ajoutée comme étant le chemin le plus court sera celle avec un fCost de 42.

Le coût d'un déplacement horizontale ou vertical vaut 10 et une diagonale 14. Ces nombres sont définis ainsi comme les carrés sont de 1cmx1cm la longueur/hauteur vaut 1 et la diagonale (par pythagore) vaut sqrt(2) et donc 1.41. Pour travailler avec des valeurs entières et simplifier les calculs ont arrondi et fait x10 sur ces valeurs.

Si vous voulez approfondir ou mieux comprendre l'algorithme je vous invite à regarder la série de vidéos par Sébastien Lague. Ces vidéos expliquent en détails les concepts et détails de l'algorithme. Également comme ressources la page Wikipedia où vous pourrez retrouver le pseudo-code de l'algorithme.

3 Travail à faire

Vous trouverez dans l'archive du laboratoire deux dossiers, src et include. Les fichiers qui vous intéressent sont **a_star.c et input.txt**:

- **a_star.c** Vous y trouverez deux fonctions à compléter. Une description de celles-ci se trouve dans include/a_star.h
- **input.txt** Ce fichier vous permet de spécifier la grille sur laquelle vous voulez travailler. Cette grille n'est composée que de 1 et de 0 où 1 définit un obstacle et 0 une case sans obstacle. Dans ce même fichier, vous pouvez ajouter un A pour le point de départ et un B pour le point d'arrivée. Le fichier input qui vous est fourni est déjà complété avec un exemple que vous pouvez modifier.

Le but et de nous fournir deux versions d'A* une utilisant un `int**` comme grille et l'autre recevant un pointeur sur la tête d'une liste chaînée de nœuds Grid_Component. Cette structure est définie dans **include/utils.h**, elle contient pour chaque nœud ses voisins, de gauche, de droite, du dessus et du dessous. Si le nœud est dans un bord de la grille le voisin du bord sera NULL.

Les deux fonctions à remplir sont `compute_path_tab` et `compute_path_struct` du fichier a_star.h. Vous êtes libre de créer des structures ou des fonctions supplémentaires pour vous simplifier la tâche. Ces fonctions prennent en paramètre les positions de départ et d'arrivée ainsi que la grille. Vous pouvez rajouter du code dans main si vous en avais besoin également.

Enfin, pour valider le fonctionnement de votre algorithme, il vous est demandé d'afficher le chemin dans la grille dans le terminal.

La fonction main a déjà été préparée pour vous. Elle attend en paramètre plusieurs informations :

1. "tab" ou "struct" qui permet de définir si on veut lancer a* avec la liste chaînée ou le simple tableau
2. le nombre de colonnes de votre fichier input.txt
3. le nombre de lignes de votre fichier input.txt

Donc par exemple `./a_star tab 20 11`. Un Makefile vous est mis à disposition également, donc pour compiler le projet rendez vous dans le dossier src et exécuter la commande make.

4 Mesure du temps d'exécution

Lorsque l'on cherche à connaître la performance d'un programme, le premier critère d'évaluation à observer est son temps d'exécution. Un outil que vous avez à connaître pour évaluer ce critère est /usr/bin/time. Nous vous invitons à lire sa documentation et à l'exécuter sur votre programme

pour le prendre en main. Une fois que vous savez mesurer le temps d'exécution, nous allons nous intéresser à l'évolution des temps d'exécution des deux implémentations en fonction de la taille des données fournies. Choisissez un ensemble de tailles que vous jugez pertinent, exécutez les deux implémentations, récupérez les temps d'exécution puis générez un graphe comparant les deux algorithmes. Analysez vos résultats et concluez.