# Rapport projet de synthèse : Algorithme de Dijkstra

## Introduction

Cette année au second semestre, nous avons eu pour projet de synthèse d’implémenter l’algorithme de Dijkstra en C. Dans ce rapport nous relatons par quels moyens nous y sommes arrivés et les éventuelles difficultés rencontrées en chemin. Nous parlons aussi de la performance de notre algorithme suivant les différents types d’implémentation. Et pour finir nous conclurons sur les compétences acquises grâce à ce projet.

## Outils utilisés

Pour mener à bien ce projet, nous avons dû nous servir de différents outils.

Le premier étant git, qui nous a permis tout au long de notre projet de pouvoir se partager entre nous l’avancée du programme.

Le second outil a été VirtualBox qui nous a permis de pouvoir utiliser linux chez nous et ainsi pouvoir continuer de travailler sur notre projet.

Le troisième outil a été vim qui est l’éditeur de texte avec lequel nous avons tout codé. Une prise en main un peu compliquée, mais une fois maîtrisé vim est un très bon éditeur de texte.

Le dernier outil a été eclipse, qui nous a permis de faire l’interface graphique qui permet de visualiser le résultat de notre programme sous forme de graphique.

## Quelles étaient les difficultés ?

Tout le développement de ce projet s’est fait sans trop de difficulté à proprement parlé.

Nous avons réussi à coder toutes les fonctions demandées sans grand problème. Cependant la majeure difficulté a été dans l’élaboration de tests complets pour vérifier que nos fonctions marchaient correctement ainsi que les fonctions qui déplaçaient, inséraient ou enlevaient des éléments d’une structure de donnée, car il fallait alors replacer correctement tous les pointeurs.

En effet, à plusieurs reprises alors que nous pensions que certaines de nos fonctions marchaient correctement, nous nous sommes rendu compte lors de leur utilisation dans la fonction de Dijkstra que ce n’était pas le cas. Il s’avère que nous avions pour certaines fonctions oublié des cas particuliers ou aussi fait des erreurs qui nous causait des problèmes dans certaines conditions. Rechercher ce genre d’erreurs dans tant de fonctions n’est pas chose aisée et cela nous a fait perdre énormément de temps.

Donc pour ce qui est de la partie C, si nous devions retenir une difficulté ce serait la confection de tests complets qui nous auraient permis de détecter nos erreurs directement.

En ce qui concerne le développement de l’interface graphique, la plus grosse difficulté a été de réussir à faire le lien avec le code C et java grâce à jni. Au début nous avions essayé de le faire sous Windows mais sans grand succès. Nous avons donc essayé sous linux et avons réussi à le faire marcher sans trop de soucis.

Pour ce qui était de trouver comment représenter nos données, la question qui s’est posée est la suivante : quel type de graphe utiliser ? Nous avons opté pour un graphe circulaire, qui est assez simple à dessiner et qui reste assez lisible si le nombre de villes n’est pas excessif. Avec une difficulté modérée pour placer correctement nos éléments du graphe en fonction du nombre de ville.

## Fonctions rajoutées

Nous avons au long de ce projet rajouté plusieurs fonctions. Les plus importantes étant surtout celles nous permettant de faire une recherche dichotomique dans un arbre et une fonction qui nous permet de supprimer le last dans un cas bien particulier.

La fonction IsNotSaturated, qui nous permet de savoir si notre arbre est saturé, sinon nous donne le nombre de nœud manquant pour qu’il le soit.

La fonction countNbStage, qui nous permet de savoir le nombre d’étage de notre arbre à partir de son nombre d’élément.

La fonction removeLeftChild, qui nous permet de supprimer le last quand ce dernier se trouve être un fils gauche ne se trouvant pas aux extrémités. C’est dans cette fonction que nous faisons une recherche dichotomique nous permettant de rechercher le futur last de notre arbre et ce grâce aux deux fonctions ci-dessus.

## Quelles améliorations envisagez-vous ?

Pour la suite du projet nous pourrions envisager d’ajouter un générateur de ville permettant de pouvoir générer des fichiers textes à pouvoir utiliser avec notre programme.

## Performance

### Complexité asymptotique

Tableau :

- DTHeapInsert : log2

- DTHeapExtractMin : log2

- DTHeapIncreasePriority: log2

Arbre:

- DTHeapInsert: log2

- DTHeapExtractMin: log2

- DTHeapIncreasePriority: log2

Liste:

- DTHeapInsert: N

- DTHeapExtractMin: constant

- DTHeapIncreasePriority: N

### Temps d’exécution

#### Quelles sont les observations que nous pouvons faire grâce à ces données ?

Il est flagrant que l’implémentation avec des listes n’est pas une option viable car elle est très vite dépassée. En effet le seul moment où l’implémentation avec des listes est la plus rapide est avec 10 villes.

Il nous reste alors deux implémentations, celle avec un tableau et celle avec un arbre. On observe que jusqu’à 500 villes l’implémentation avec tableau est plus rapide, et qu’à partir de 1500 villes l’implémentation avec arbre devient la plus rapide. Il faudrait donc utiliser l’implémentation avec tableau en dessous de 1500 villes et l’implémentation avec arbre au-delà.

Cependant avec 1500 villes le temps d’exécution maximum avoisine les 0.12 seconde. Ce temps restant très minime il n’y a aucune différence à vue d’œil entre ces 3 implémentations. Mais nous pouvons supposer que plus le nombre de ville augmente plus l’implémentation avec arbre sera rapide et à ce moment-là il sera pertinent de la privilégier par rapport aux deux autres.

## Conclusion

Au bout de ces 5 mois de projet, nous avons pu acquérir certaines compétences. Une maîtrise plus poussée des pointeurs, du concept de type de donnée abstraite et de l’outil git ainsi qu’une première approche de jni. Ce projet, qui est le plus gros projet depuis le début de notre licence informatique a été très agréable à faire, l’ambiance de travail était superbe malgré les différents problèmes rencontrés et nous en sortons que plus enrichis.