

## RAPPORT : PROJET DE SYNTHESE

### PLAN

- 1) Introduction
- 2) Conditions de travail
- 3) Fonctions supplémentaires
- 4) Evaluation des structures de données
- 5) Interface Graphique
- 6) Difficultés rencontrés
- 7) Améliorations envisagées
- 8) Conclusion

#### 1) Introduction

Le projet représente une occasion d'appliquer les connaissances que nous avons acquises tout au long de l'année dans divers domaines tels que les Mathématiques Discrètes, la Programmation en C, l'Algorithmique et les structures de données, les Outils Système, la Programmation en Java et les Interfaces Graphiques. Ce projet s'effectue en binôme.

L'objectif principal de ce projet est de mettre en œuvre un algorithme qui simule l'exécution d'un ensemble de tâches sur plusieurs machines parallèles. Chaque tâche est caractérisée par deux paramètres : sa durée opératoire et sa date de libération. Nous souhaitons également offrir à l'utilisateur la possibilité de préempter une tâche, c'est-à-dire de l'interrompre momentanément pour la reprendre ultérieurement sur la même machine ou sur une autre.

Pour parvenir à nos objectifs, nous avons utilisés différentes structures de données. Nous avons utilisé des listes doublement chaînées pour représenter une instance du problème d'ordonnancement ainsi que l'ordonnancement des tâches sur chaque machine. Par ailleurs, les arbres binaires de recherche, équilibrés ou non, sont utilisés pour représenter la file d'attente des différentes tâches et l'ensemble des événements pendant l'exécution de l'algorithme.

Une part essentielle de ce projet consiste en l'évaluation des performances des structures de données et des algorithmes. Cette évaluation se déroule autour de deux axes principaux : la rapidité d'exécution et la qualité de la solution obtenue.

En ce qui concerne la rapidité d'exécution, nous avons effectué des comparaisons entre l'utilisation d'arbres équilibrés et non équilibrés dans nos algorithmes d'ordonnancement.

De plus, nous avons développé des scripts Bash dédiés à l'évaluation des performances des algorithmes et des structures de données. Ces scripts nous permettront de collecter des données afin d'analyser de manière approfondie ces performances.

Enfin, nous avons conçu une interface graphique, basée sur Java avec la plateforme Java FX pour l'interface graphique, ainsi que JNI pour l'intégration de code C dans notre application Java. Cette interface graphique permet une visualisation des performances à travers le diagramme de Gantt.

## 2) Conditions de travail

Dans le cadre de ce projet, nous avons travaillé en binôme. Nous avons utilisé Git afin de pouvoir collaborer sur le projet. Cela nous a permis d'avoir un répertoire distant nous servant de support de travail afin de partager et fusionner nos modifications de code, ainsi qu'avoir un suivi de l'ensemble du projet.

Pour faciliter la communication et le partage et pour voir en temps réel ce que l'autre personne du binôme faisait, nous avons utilisé l'extension Live Share de Visual Studio Code. Cela nous a grandement aidés à résoudre rapidement les problèmes et apporter au plus vite les modifications de code nécessaires.

Concernant les environnements de développement (IDE), nous avons utilisé Visual Studio Code pour la totalité de la partie C (partie 1 du projet). Pour la partie Java FX, nous avons utilisé à la fois Visual Studio Code et IntelliJ IDEA. Cette approche collaborative nous a permis de travailler efficacement, ainsi que partager nos connaissances et nos idées de manière rapide.

## 3) Fonctions supplémentaires

Dans notre projet, nous avons intégré une fonction supplémentaire nommée "get\_height", celle-ci a pour but d'obtenir la hauteur d'un sous-arbre raciné en se basant sur la hauteur de ses sous-arbres droit et gauche, cette fonction effectue un parcours récursif des sous-arbres. Grâce à cette fonction, nous pouvons calculer les facteurs d'équilibre (bfactor) de manière plus simple et intuitive, et ainsi faciliter les fonctions de rotation gauche et droite.

## 4) Evaluation des structures de données

Dans un premier temps, parlons de la complexité des algorithmes.

Evaluation de performances sur les arbres binaires :

Fonction	Équilibrage	Complexité
tree_find_successor	Équilibré	$\log(O)$
tree_find_successor	Non-équilibré	$\log(O)$ à $O$
insert_into_tree_node	Équilibré	$\log(O)$
insert_into_tree_node	Non-équilibré	$\log(O)$ à $O$
remove_tree_node	Équilibré	$\log(O)$
remove_tree_node	Non-équilibré	$\log(O)$ à $O$

Tous les algorithmes dans les arbres équilibrés s'exécutent en temps logarithmique car aucun parcours complet n'est nécessaire.

Toutes ces fonctions font un parcours binaire dans l'arbre qui est donc le plus souvent équivalent à  $N$  = hauteur de l'arbre, sachant que  $N = \log(M)$  où  $M$  = nombre de nœuds. Cela marche seulement quand l'arbre est équilibré et que la condition précédente reste vraie. Si l'équilibre n'est pas respecté l'arbre peut (dans le pire cas) finir en une simple liste chaînée.

## 5) Interface Graphique

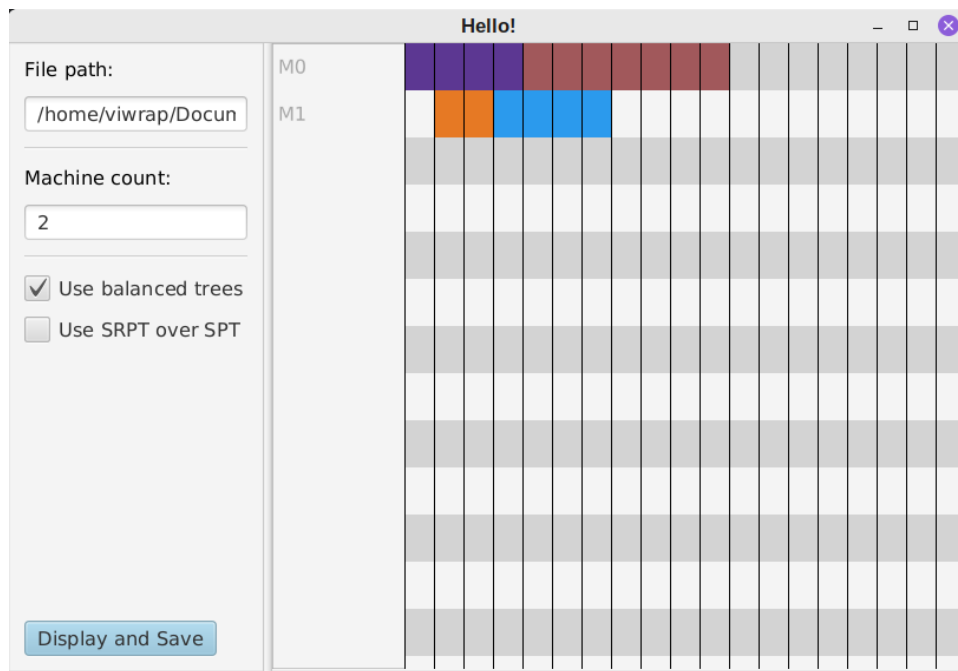


Image de l'interface graphique de l'application

Dans la partie interface de notre projet, nous avons utilisé Java FX pour implémenter une interface utilisateur afin de pouvoir lire une instance, la paramétrer selon un arbre équilibré ou non ainsi que choisir l'algorithme de tri utilisé (SPT ou SRPT), puis l'afficher à travers un diagramme de Gantt.

Nous avons créé une classe appelée "DisplayedTask" qui représente une tâche à afficher dans le diagramme et contient les informations nécessaires.

Nous avons également implémenté plusieurs fonctions nécessaires à l'affichage des tâches, toutes ces fonctions appartiennent à la classe ProjetController.

La fonction "redrawGraph" a pour objectif de préparer et afficher les tâches, "readOutputFile" lit et interprète le fichier de sortie, "intialize" gère la taille du Canvas, et submitToDiagram est appelé pour le renseignement des options choisies par l'utilisateur et la génération du fichier de sortie.

La fonction "run\_schedule", présente dans "submitToDiagram" est une fonction C intégrée via JNI, cela nous permet d'appeler directement cette fonction avec Java.

Pour finir, l'ensemble des options, le renseignement du fichier de sortie et le bouton "Display and Save" qui effectue la sauvegarde et l'affichage des tâches sont présents dans l'interface graphique.

## 6) Difficultés rencontrées

Une des plus grandes épreuves de ce projet a été de faire fonctionner l'interface Java FX. La librairie est très capricieuse sur la manière de s'exécuter. Généralement elle fonctionne car un IDE Java est déjà configuré dans le meilleur des cas pour avoir un pom.xml pour Maven, et là la librairie fonctionne immédiatement. Dans tout autre cas cependant tout effort de la faire fonctionner finit généralement en vain, surtout sans aucun gestionnaire de projet et en utilisant de simples lignes de commande Java.

Une autre difficulté rencontrée est le mauvais comportement de l'algorithme SRPT (Short Remaining Processing Time), l'utilisation de "create\_schedule" avec cet algorithme amène dans le pire des cas une erreur d'exécution, dans le meilleur des résultats faux, ce qui a été un problème dans la

comparaison des performances selon l'algorithme mais aussi et surtout lors de l'affichage des résultats dans l'interface graphique.

## 7) Amélioration envisagées

Dans un premier temps il y'a différentes améliorations possible pour l'interface graphique.

Il serait possible de rendre l'interface plus conviviale et interactif, à travers une barre d'outils, un menu déroulant ou encore un thème plus esthétique. On pourrait aussi ajouter de nouvelles fonctionnalités comme la possibilité de visualiser les résultats sous d'autres forme, de comparer plusieurs résultats ou encore d'exporter l'affichage des résultats.

Il serait aussi possible de permettre à l'utilisateur de créer, modifier et supprimer des tâches directement depuis l'interface. Cela permettrait une plus grande flexibilité et permettrait à l'utilisateur d'effectuer des changements facilement.

Dans un second temps, il y'a différentes façons d'améliorer la partie non graphique.

Il serait possible d'utiliser de nouveaux algorithmes qui pourraient améliorer les performances et fournir des solutions plus efficaces.

Une autre possibilité serait d'utiliser des structures de données différentes, là encore pour des raisons de performance et d'optimisation.

Pour finir, il peut être intéressant de rajouter des fonctionnalités de collaboration, qui permettrait à différents utilisateurs d'interagir et de partager en temps réel, cela pourrait être utile pour des projets à plusieurs ou pour l'utilisation de l'application en entreprise.

## 8) Conclusions

En conclusion, ce projet nous a permis de mettre en pratique l'ensemble des connaissances acquises tout au long de l'année de L2. Nous avons développé un algorithme d'ordonnancement de tâches sur plusieurs machines parallèles, en utilisant des structures de données telles que des listes doublement chaînées et des arbres binaires de recherche équilibrés ou non équilibrés, le tout selon un algorithme SPT (Short Processing Time) ou SRPT (Short Remaining Processing Time).

L'évaluation des performances selon les options disponibles a été l'un des points clés de notre projet, ces évaluations nous ont permis de comprendre les avantages et les limites de nos choix en termes d'algorithme et de structure de données.

Nous avons aussi développé une interface graphique, ce qui nous a permis de gérer les interactions utilisateurs et d'afficher les résultats de l'ordonnancement des tâches. Nous avons également appris à utiliser JNI afin d'appeler une fonction C directement depuis Java.

Ce projet nous a également permis de travailler en binôme sur un grand projet, à communiquer et partager nos connaissances et à faire face aux différents problèmes rencontrés. Nous avons également identifié des améliorations possibles qui pourraient rendre l'application plus performante, interactive et même la rendre plus adaptée à un travail en équipe.