Hersan Mathéo
Doumir Fernandes Yannis



# Compte rendu - SAE 1.02 Comparaisons d'approches algorithmiques

# Contexte :

Dans le contexte de notre SAE portant sur l'application de la méthode PERT à un projet de construction immobilière en langage C, nous avons détaillés les tâches du projet ainsi que leurs dépendances. Un planning a été élaboré afin de coordonner ces différentes tâches. Cette planification permettra de suivre de près l'évolution du projet immobilier.

# Présentation des données :

## Informations sur les Devis :

- Nom de la tâche : nomTache
- Nom de l'entreprise : entreprise
- L'adresse de l'entreprise :
    - Numéro de rue : numero
    - Nom de la rue : nomRue
    - La ville des bureaux de l'entreprise : ville
    - Le code postal de la ville : codePostal
- Le capital de l'entreprise : capital
- La durée de la tâche : duree
- Le coût de production : cout

## Informations sur les Tâches :

- Le nom de tâches : tache
- La durée de la tâche : duree
- Le nombre de tâches à faire avant de pouvoir faire la courante : nbPred
- Une liste liste des tâches à faire après la tâche courante : succ
- La date de début planifié pour cette tâche : dateDebut

# Fonctionnalitées :

1) Afficher les précédences : Affiche tous les couples de précédences.

2) Ajouter une précédence : Permet de créer un nouveau couple de tâches.

3) Afficher les devis : Affiche les devis fait pas les entreprises avec toutes les informations correspondantes.

4) Afficher l'ensemble des devis pour chaque tâche : Cette fonction affiche les devis associés à une entreprise spécifiée.

5) Afficher le devis d'une entreprise donnée pour une tâche donnée : Cette fonction parcourt un tableau d'offres et affiche les informations de chaque offre. Pour chaque offre, elle affiche le type de travaux et la liste des associés.

6) Sélectionner les meilleurs devis entreprises pour chaque tâches : Calcule quel devis est le plus intéressant pour chaque tâches.

# Jeu de donnée

Dans cette saé, nous avons deux fichiers que nous utilisons pour notre jeu de données.

## Précédences

Le premier est le fichier des précédences, qui contient les nom des tâches à faire dans l'ordre "premier -> deuxième". Ce qui veut dire que la tâche de gauche sur une ligne sera effectuée avant celle de droite. Voici un extrait du fichier :

| | |
|---|---|
| Electricité | Plomberie |
| Plomberie | Peinture |
| Plomberie | Carrelage |
| DolcéAqua | Peinture |
| Fonderie | Plomberie |
| Maçonnerie | Isolation |
| Charpente | Électricité |
| Climatisation | Serrurerie |
| Serrurerie | Revêtement |
| Menuiserie | Peinture |
| Peinture | Revêtement |
| Revêtement | Menuiserie |

## Devis

Le second fichier est le fichier des devis, il contient toutes les informations composant les différents devis. Voici un extrait de ce contient le fichier :

Plomberie // Tâche à effectuer
Pros De L'eau // Nom de l’entreprise
12 Av. la république 63000 Clermont-Ferrand // Adresse de l’entreprise
500000 // Capital de l’entreprise
35 // Durée de la tâche
25000 // Coût de réalisation
Plomberie // Autre tâche
DolcéAqua
33 zone Industriel Aubière 63170 Aubière
450000
40
27000

# Les différents algorithme utilisés :

## Les algorithme de tri :

**quickSort :**
***fichier : utils.h**

La fonction `quickSort` est une implémentation de l'algorithme de tri rapide (quicksort) qui trie un tableau d'offres en fonction du champ travaux. Elle prend en paramètre un tableau de pointeurs d'offres `tOffre` et sa taille logique `tLogique`.
L'algorithme de tri rapide divise le tableau en deux partitions, selon le pivot choisi (dans ce cas, le dernier élément du tableau). Les éléments plus petits que le pivot sont déplacés à gauche, et les éléments plus grands sont déplacés à droite. La fonction est ensuite récursivement appliquée aux deux partitions.

La complexité de l'algorithme de tri rapide est O(n log(n)), où n est la taille du tableau.

## Les algorithme de recherche :

**afficherDevisEntreprisePourType :**
***fichier : affichage.h**

Cette fonction est un algorithme de recherche qui prend en paramètre deux choses, un tableau de pointeur d'offre, `tOffre`, ainsi que sa taille logique, appelé `nbOffre`.

L'algorithme de recherche utilisé dans cette fonction est basé sur une recherche séquentielle dans le tableau d'offres et la liste de devis associée. Il parcourt chaque élément jusqu'à trouver une correspondance.

La complexité de cette fonction, dans le pire des cas,est le parcours de toute la liste jusqu'à atteindre la fin pour insérer le nouveau devis. Si la liste à `n` éléments, la complexité dans le pire des cas est O(n).
En premier lieu il recherche une correspondance avec un `for` une correspondance au niveau du nom de l'offre `tOffre->tache` pour voir si la tâche en question existe. Si elle existe, alors il regarde dans la liste des devis si l'entreprise entré par l'utilisateur existe.

**insertionCroissante :**
***fichier : utils.h**

Cette fonction est une fonction d'insertion dans une liste chaînée triée de devis. Elle prend en paramètre une liste de devis de type `ListeDevis`, ainsi que les informations

nécessaires pour créer un nouveau devis. Elle est utilisée dans la fonction `nouveauDevis`, elle permet de garder les devis trier par nom d'entreprise

L'algorithme d'insertion utilise une approche récursive. Si la liste de devis est vide (NULL), la fonction insère le nouveau devis en tête de liste en appelant la fonction insererEnTete. Sinon, elle compare le nom de l'entreprise du nouveau devis avec celui du premier devis de la liste. Si le nom de l'entreprise est inférieur, la fonction insère le nouveau devis en tête de liste. Sinon, elle appelle récursivement la fonction sur le reste de la liste.

La complexité de cet algorithme est de O(n) dans le pire des cas. Le pire des cas serait que le nom de la tâche soit à la fin dans le tableau `tOffre` car il faudrait ajouter donc l'ajouter à la fin et on aurait donc parcouru tout le tableau.

**rechercheDichotomique :**
***fichier : utils.h**

La fonction `rechercheDichotomique` réalise une recherche dichotomique dans un tableau trié d'offres pour trouver une tâche spécifique `nomTache`. Elle prend en paramètre le tableau d'offres `tOffre`, sa taille logique `tLogique`, le nom de la tâche à rechercher, et un pointeur pour indiquer si la tâche a été trouvée. La recherche dichotomique divise récursivement le tableau par deux jusqu'à trouver la tâche ou déterminer son absence.

La complexité de la recherche dichotomique est O(log(n)), où n est la taille du tableau. Cette complexité est due à la division répétée par deux du tableau jusqu'à trouver la tâche recherchée.