- La répartition des taches au sein du groupe ✅
- Le planning de réalisation ✅
- Les limitations fonctionnelles de votre application (la liste de ce qui n'est pas implémenté, et/ou de ce qui est implémenté mais que ne fonctionne pas correctement/totalement) : le temps du programme

Tout au long du projet, nous avons travaillé en groupe. Nous avons organisé notre temps et réparti les tâches en fonction des compétences de chacun pour avancer sur les différentes étapes. Travailler ensemble nous semblait important pour que tous les membres du groupe comprennent l'intégralité du programme.

La plupart de nos séances de travail se sont déroulées chez l'un des membres du groupe, ce qui nous a permis de discuter, coder et tester ensemble. Cela a facilité la communication et la résolution des problèmes.

Pour mieux organiser notre travail, chacun a travaillé sur son ordinateur personnel mais nous avons utilisé deux ordinateurs pour apporter des modifications au programme sur Github. Ainsi, pour apporter les modifications au projet, nous avons choisi d'utiliser un groupe WhatsApp, créé au préalable, pour s'envoyer notre avancement personnel et que tout le groupe puisse facilement voir ce que l'on avait fait avant de faire les modifications au programme initial. Cela nous a permis de répartir les tâches et d'avancer en parallèle tout en restant bien coordonnés grâce à des échanges réguliers.

Pour éviter les problèmes dans le dépôt Git et simplifier la gestion des versions, nous avons concentré la majorité du travail sur un seul ordinateur. Cela nous a permis de garder une version stable et d'intégrer facilement les contributions de chacun.

Pour la répartition des tâches, nous avons décidé de fonctionner de cette manière : tout d'abord nous avons commencé par faire la partie Shell en déterminant le réel objectif de la partie Shell et les différentes fonctions à intégrer au programme. Dès que cela a été fait, chaque personne s'est concentrée sur une partie :

- Partie 1 : la gestion des arguments et affichage d'aide. Cette partie a été réalisée par Océane : elle s'est concentrée sur la gestion des arguments passés au programme, y compris la validation de ces arguments et l'affichage d'aide. Elle a donc principalement contribué à faire les fonctions :
  - 'afficher_aide()' pour afficher un message d'aide expliquant les différents paramètres attendus et les combinaisons valides/invalides,
  - 'verification_demande_aide()' pour vérifier si un des arguments est '-h' et, si oui, afficher l'aide et arrêter l'exécution du programme,
  - 'verif_3arguments()' pour valider le fait que les trois premiers arguments soient corrects (fichier, type de station, type de consommateur), et afficher une erreur et appeler l'aide si l'un des arguments est invalide,
  - 'verif_tout_arguments()' pour vérifier qu'il y a entre trois et quatre arguments et afficher une erreur si le nombre d'arguments est incorrect ou si le quatrième argument est invalide, et
  - 'verif_presence_dossier()' pour vérifier que les dossiers 'tmp' et 'graphs' existent et les créer si nécessaire et vider le contenu des dossiers si des fichiers existent déjà.

Le but de cette partie est de s'assurer que le programme vérifie et valide correctement les entrées de l'utilisateur, tout en affichant des messages d'aide et d'erreur.

- Partie 2 : le traitement des fichiers en fonction des paramètres. Cette partie a été réalisée par Yaël : il s'est chargé de la gestion des fichiers en fonction des paramètres passés et a effectué le tri des données en fonction des choix de l'utilisateur. Il donc principalement contribué à faire les fonctions :

- ‘trier_fichier_3_parametre()’ pour trier les données du fichier en fonction du type de station (hvb, hva, lv) et du type de consommateur (comp, indiv, all) et générer des fichiers temporaires avec les données triées,
- ‘trier_fichier_4_parametre()’ pour trier les données du fichier en fonction d'un quatrième paramètre (identifiant de centrale) et générer des fichiers de sortie spécifiques à la centrale sélectionnée par l'utilisateur,
- ‘creation_lv_min_max()’ pour trier et traiter les données des stations de type "lv" en ajoutant une colonne avec la différence absolue entre la consommation et la capacité, puis générer un fichier final contenant les résultats triés, et
- ‘traitement_lv_all()’ pour préparer un fichier de données simplifié pour la création d'un graphique, en calculant la différence entre la capacité et la consommation et en attribuant une couleur à chaque station en fonction de la différence.

Le but de cette partie est de gérer la logique, de faire le traitement des fichiers selon les critères fournis, et de s'assurer que les fichiers de sortie sont bien créés avec les bonnes informations.

- Partie 3 : La gestion des données et validation de la sortie. Cette partie a été réalisée par Adrien : il s’est concentré sur les aspects liés à la gestion des résultats et à la validation des fichiers produits. Il a veillé à ce que les fichiers produits aient la bonne structure, et s’est assuré qu'aucune donnée erronée ne soit écrite dans les fichiers lors des fonctions ‘trier_fichier_3_parametre()’ et ‘trier_fichier_4_parametre()’. Après le traitement des fichiers, il a validé que les fichiers finaux créés contenaient bien les données attendues et dans le bon format. Aussi, il a testé que chaque paramètre de l'utilisateur était correctement interprété par les fonctions de tri et s’est assuré que les fichiers finaux étaient bien générés avec les bonnes valeurs pour chaque type de station et consommateur. Ensuite, Il a géré l'exécution générale du programme et s’est assuré qu'aucune erreur n'interrompait l'exécution avant la génération correcte des fichiers de sortie. Ce suivi comprend également la gestion des fichiers temporaires et leur nettoyage après traitement.
  De plus, il a principalement contribué à faire les fonctions :
  - ‘creation_graphique()’ pour générer un graphique PNG à partir des données préparées, en utilisant gnuplot pour afficher les différences de consommation des stations "lv" sous forme d'histogrammes colorés,
  - ‘traitement_lv_all()’ pour vérifier les conditions permettant de traiter les stations de type "lv", puis appeler les fonctions nécessaires pour créer les fichiers et graphiques correspondants,
  - ‘enregistre_resultat()’ pour vérifier si un fichier de résultats existe déjà dans le dossier de destination et le copier si nécessaire.

Le but de cette partie est de veiller à la qualité des données produites et à la validation finale du programme.

Ensuite nous avons réalisé la partie en langage C. On a commencé par déterminer le réel objectif de la partie en langage C et les différentes fonctions à intégrer au programme. Dès que cela a été fait, chaque personne s’est concentrée sur une partie :

- Partie 1 : La gestion des fichiers et de la logique principale. Cette partie a été réalisée par Océane. Elle s’est occupée de la fonction ‘main’ et des interactions entre les fichiers temporaires et finaux. Elle a géré les fonctions suivantes :
  - ‘main()’ qui permet de lire les fichiers passés en argument, gérer les erreurs, et appeler le traitement principal, et
  - ‘traitement_total()’ qui supervise le traitement global du fichier temporaire pour construire l'arbre et écrire les résultats dans le fichier final.

Le but de cette partie est d’ouvrir les fichiers fournis en arguments (fichier temporaire et fichier final), de s’assurer qu'ils soient accessibles, de traiter ces fichiers en utilisant la fonction ‘traitement_total’ qui construit un arbre AVL, d’écrire les résultats dans le fichier final et de fermer les fichiers ouverts pour éviter les fuites de ressources.

- Partie 2 : La gestion de la structure AVL et des opérations sur l'arbre. Cette partie a été réalisée par Adrien : il s'est occupé de la construction et de la manipulation des arbres AVL. Il a géré les fonctions suivantes :
  - 'creer_arbre()' qui crée un noeud AVL,
  - 'insert_AVL()' qui insère un noeud dans l'arbre AVL tout en maintenant son équilibre,
  - 'equilibrer_AVL()', 'rotation_droite()', 'rotation_gauche()', 'double_rotation_droit()', 'double_rotation_gauche()' qui effectuent les rotations pour équilibrer l'arbre, et
  - 'min()' et 'max()' qui fournissent des outils pour les calculs nécessaires à l'équilibrage.

Le but de cette partie est d'implémenter les bases d'un arbre AVL : créer des nœuds, insérer des éléments, équilibrer l'arbre après insertion (rotations simples et doubles) et de maintenir les propriétés AVL en calculant et en ajustant les facteurs d'équilibre.

- Partie 3 : la gestion des données et écriture dans les fichiers. Cette partie a été réalisée par Yaël : il s'est occupé des opérations d'écriture des données et des mises à jour des nœuds. Il a géré les fonctions suivantes :
  - 'ajout_consommation_noeud()' qui ajoute de la consommation à un nœud existant dans l'arbre,
  - 'ecrire()' qui parcourt l'arbre AVL en ordre croissant et écrit les données dans un fichier, et
  - 'Recuperer_fichier_tmp()' qui construit un arbre AVL à partir d'un fichier d'entrée.

Le but de cette partie est d'ajouter ou mettre à jour des informations dans l'arbre AVL, notamment en augmentant la consommation d'un nœud donné, de construire un arbre AVL à partir des données lues dans un fichier temporaire et d'écrire les données d'un arbre AVL dans un fichier final en parcourant l'arbre de manière ordonnée (parcours infixe).

En dernier lieu, nous avons réalisé le Makefile ainsi que le Readme. Le Makefile a été réalisé par Yaël tandis que le Readme a été réalisé par Adrien et Océane : Adrien s'est occupé de la partie en Shell et Océane s'est occupée de la partie en C.

Les commentaires du code ont été écrit pas tous les membres du groupe.

Voici maintenant notre planning de réalisation :

- Lundi 9 décembre :
  - Lecture attentive des consignes du projet
  - Clarification des objectifs de chaque partie (Shell et C).
  - Répartition des tâches entre les membres du groupe en fonction des compétences.
  - Création des outils de coordination : dépôt Git, groupe WhatsApp, planning initial.
- Mardi 10 décembre : Commencement de la partie Shell :
  - Travail sur les fonctions liées à la gestion des arguments
  - Planification de la structure générale du programme
  - Définition des bases pour la gestion des fichiers
- Mercredi 11 décembre : Avancée sur les fonctions Shell secondaires :
  - Progression sur la validation des arguments
  - Création d'un schéma logique pour les interactions entre fonctions
  - Début du développement des fonctions de tri des fichiers
- Jeudi 12 décembre :
  - Mise en place des fonctions pour la gestion des fichiers temporaires
  - Développement de tests pour la robustesse des validations d'arguments
- Vendredi 13 décembre :
  - Intégration des premières fonctions Shell dans le programme principal

    - Préparation pour tester les interactions entre les validations et la gestion des fichiers
- Samedi 14 décembre :
    - Partie Shell :
        - Finalisation des fonctions de tri des fichiers
        - Création des messages d'erreur et de l'affichage d'aide
    - Partie C :
        - Initialisation de la structure AVL
        - Développement des fonctions de base pour les rotations AVL
        - Mise en place des prototypes pour la gestion des fichiers en C
- Dimanche 15 décembre :
    - Partie Shell :
        - Intégration complète des fonctions et tests sur différents scénarios
    - Partie C :
        - Finalisation des rotations AVL et des fonctions d'équilibrage
        - Développement des fonctions pour récupérer les données d'un fichier temporaire
        - Tests préliminaires sur la fonction 'main()'
    - Début de la rédaction du Readme et du Makefile
- Lundi 16 décembre :
    - Vérification de la cohérence des fonctions entre elles
    - Ajustement des prototypes et correction des erreurs détectées
    - Début de la rédaction des commentaires du code pour la partie Shell
- Mardi 17 décembre :
    - Tests approfondis des calculs d'équilibrage AVL
    - Développement et tests des fonctions pour l'écriture dans les fichiers
    - Fin de la rédaction des commentaires du code pour la partie Shell
- Mercredi 18 décembre :
    - Corrections mineures sur les fonctions AVL et gestion des fichiers
    - Tests globaux de la partie Shell
    - Rédaction des commentaires du code pour la partie C
- Vendredi 20 décembre :
    - Préparation des fichiers pour les tests finaux
    - Vérifications finales sur le traitement des fichiers temporaires et finaux
    - Début de la rédaction du livrable PDF
- Dimanche 22 décembre :
    - Intégration complète des parties Shell et C
    - Tests finaux pour s'assurer du bon fonctionnement global et légères modifications du code
    - Finalisation du ReadMe et du Makefile
    - Fin de la rédaction du livrable au format PDF

En ce qui concerne les limites de notre projet, le code met dans le pire des cas environ 30 secondes à s'exécuter, ce qui est relativement long, mais au vu de la taille du fichier, cela nous paraît raisonnable. On a essayé au mieux de réduire la complexité temporelle mais on pourrait sûrement l'améliorer encore.