Projet Pluridisciplinaire d'Informatique Intégrative - Second Semestre

Rapport du 31 Mai 2023

Kieran DALIGAUD
Arthur GÔMES
Thomas LERUEZ
Cosimo UNGARO

Table des matières

1 Première partie - Optimisation de trajet pour véhicule électrique

La première partie de ce projet consiste à créer un calculateur d'itinéraire qui prend en compte les spécificités du véhicule électrique de l'usager (notamment son autonomie) pour donner le trajet permettant de relier deux points sur une carte en indiquant tous les arrêts nécessaires pour recharger son véhicule.

1.1 État de l'art et analyse du sujet

Afin d'envisager les solutions possibles, nous avons commencé par réaliser un état de l'art. Il s'est divisé en 2 parties.

Premièrement, nous avons analysé les algorithmes disponibles. Plusieurs algorithmes de calcul d'itinéraires sont disponibles. Le plus évident est l'algorithme de Dijkstra, Il calcule le plus court chemin dans un graphe. Nous avons aussi noté plusieurs algorithme dérivés de celui-ci :

- l'algorithme A*, qui utilise une fonction auxiliaire pour réaliser des approximations progressives et être plus efficace
- l'algorithme de BellMan-Ford, qui permet d'inclure des poids négatifs aux arcs du graphe.

Une autre approche est d'utiliser un algorithme de type "colonie de fourmie", où chaque usager interagit avec l'environnement pour transmettre des informations aux autres.
Du fait de sa simplicité et de sa réponse directe au problème, nous avons choisi d'implémenter l'algorithme de Dijkstra, en le modifiant pour tenir compte des spécifités du projet. Nous gardons en tête les autres algorithmes qui pourraient servir si nous avons le temps de les implémenter pour potentiellement améliorer les performances du programme.

Deuxièmement, nous avons recherché les bases de données accessibles. Concernant les véhicules, les données extraites en CSV à partir des liens donnés ont été partagées ; concernant les bornes, le site *data.gouv.fr* nous a servi a extraire les données des bornes électriques de France. Des données publiques ont également été utiles pour avoir une interface pratique pour l'utilisateur.

1.2 Structure de notre solution

Grâce à cette analyse, nous avons établi un plan de notre solution (WBS) et commencé à répartir les taches.
Les principales étapes du projet sont les suivantes :

- Obtenir et mettre en forme les données des véhicules
- Obtenir et mettre en forme les données des bornes de recharge
- Élaborer et implémenter l'algorithme permettant de construire le trajet de l'utilisateur
- Construire une interface utilisateur et la mettre en relation avec le programme
- Tester toutes les composantes et les relier entre-elles

1.3 Scrapping des données publiques

1.3.1 Electric Vehicle Database

Nous devions ici nous attacher à extraire les données liées aux véhicules électriques. Après un début de script Python s'appuyant sur la bibliothèque Scrappy, nous avons appris qu'un fichier CSV avait, sur directive de M.Festor, été mis en commun. Nous avons donc récupéré ledit fichier et nous nous en sommes servis pour la suite du projet.

1.3.2 Bornes

Le site contenant les données relatives au bornes étant un site gouvernemental proposant directement de télécharger les données au format CSV, il n'y eut pas de problème de ce côté-là. Il est cependant à noter que le format des données ne nous permettait pas l'utilisation directe dans le code C, nous avons donc dû écrire un script Python qui extractait les données (ici, plus précisemment les coordonnées), et qui les stockaient dans un fichier séparé afin de permettre leur interprétation (ledit script python était lancé par un script .sh. Le tout est consultable dans le dossier assets de la branche Cosimo).

1.4 Structure de données mises en place

Afin d'organiser notre code, nous avons rapidement mis en place plusieurs strucures de données qui nous semblaient adaptées :

— Une structure véhicule, qui permet de recenser les caractéristiques statiques d'un modèle de véhicule particulier (nom, autonomie ...)
— Une structure borne, les noeuds du graphe dans lequel l'usager se déplace. Elle permet de stocker la position d'une borne, sa capacité et tous ses autres paramètres dont sa position
— Un type Trip fait office de structure de liste, principalement pour stocker le résultat de l'algorithme de Dijkstra
— Un type graphe : un des échec du projet. Étant la structure naturelle pour modéliser ce problème, nous avons rapidement passé du temps pour l'implémenter. Finalement, elle ne s'avère pas nécessaire.

1.5 Algorithme mis en place

Le problème a résoudre est essentiellement de trouver le plus court chemin dans un graphe. La particularité est qu'ici, le graphe est a priopri complet (il existe toujours une route reliant directement deux stations entre elles) mais les trajets subissent la contrainte de l'autonomie du véhicule qui est un paramètre dynamique. Nous avons alors élaboré une variante de l'algorithme de Dijkstra dans lequel l'autonomie du véhicule conditionne les arcs possibles d'être traversés. Le pseudo-code de l'algorithme modifié est disponible en annexe.

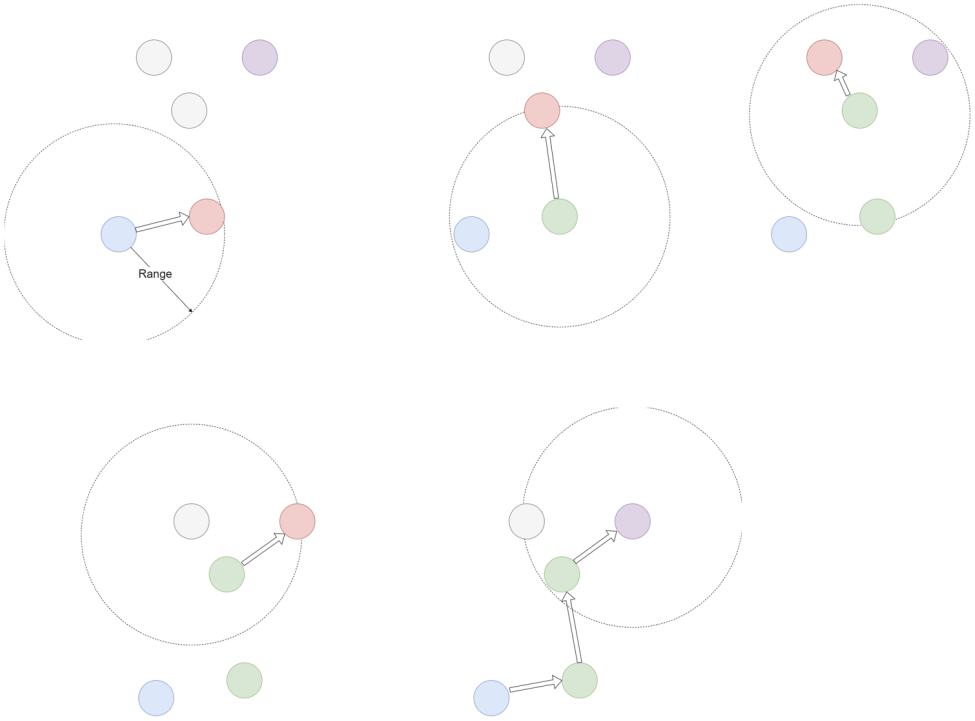


FIGURE 1 – Schéma de l'algorithme de Dijkstra modifié

1.6 Interface utilisateur

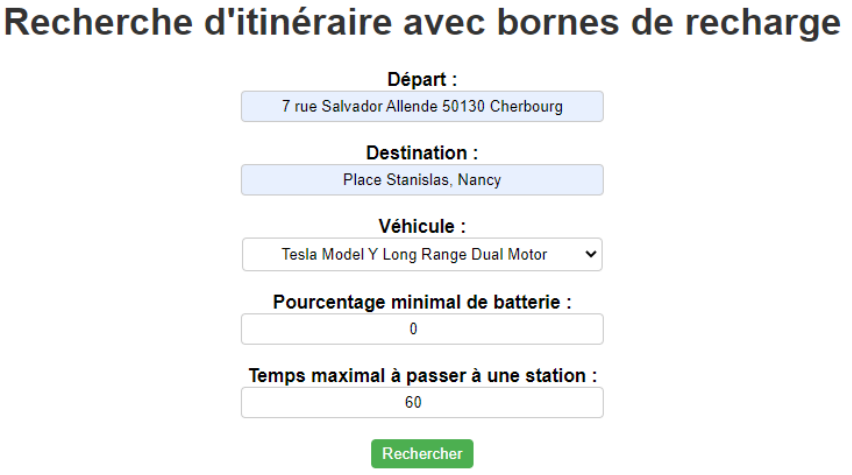
Nous avons choisi de faire une interface utilisateur sur une page Web, car il nous a semblé que ce moyen était le plus simple à prendre en main, que ce soit pour une entreprise ou un particulier, et il suffit d'avoir un serveur Web ainsi qu'une version compilée du programme en C pour que ce programme puisse être utilisé par tous les gens qui doivent y avoir accès, sans avoir à passer par la compilation et l'exécution sur chaque machine individuelle.

Dans le cadre de ce projet, il a été choisi d'utiliser un serveur Apache, dont le répertoire contient un fichier index pour la partie Web ainsi que les fichiers en C compilés. Nous avons utilisé un sous-domaine d'un nom de domaine que nous avions à disposition. Les fichiers Web contiennent du code en HTML, CSS et PHP.

1.6.1 Données d'entrée

Une fois sur la page, l'utilisateur est invité à rentrer certaines données :
- Adresse de départ
- Adresse d'arrivée
- Véhicule utilisé
- Pourcentage minimal de batterie en dessous duquel il ne désire pas descendre
- Temps maximal à passer à une station de recharge

Le code a été adapté pour que les données entrées par l'utilisateur restent en place sur la page, même après un clic sur un bouton submit.



1.6.2 Vérification données d'entrée

Des balises HTML ont été mises en place pour encadrer l'utilisateur et éviter qu'il ne mette des valeurs jugées aberrantes : un pourcentage négatif ou supérieur à 100 par exemple. Ces données sont également vérifiées par le code PHP, au cas où l'utilisateur modifie le code HTML de la page qu'il reçoit. Si un champ d'entrée comporte une erreur, celui-ci sera coloré en rouge.

1.6.3 Traitement des données d'entrée

Grâce à une API mise à disposition par le gouvernement, l'adresse envoyée par l'utilisateur est convertie en coordonnées GPS pour les utiliser comme données d'entrée du programme en C. Une fois les données d'entrée validées, PHP lance la commande d'exécution du programme en C avec les différents paramètres données par l'utilisateur avec la syntaxe adaptée. Un texte "Chargement en cours" est affiché en attendant que le programme en C retourne un résultat.

1.6.4 Sécurité

Le programme en C est lancé par PHP via une commande dans le terminal du serveur où est situé Apache, il est donc nécessaire de prendre garde à la sécurité : en effet, un utilisateur mal intentionné pourrait vouloir utiliser cela pour exécuter des commandes sur le serveur. Nous avons fait certaines vérifications pour essayer de limiter cela, comme vérifier que les caractères " ;", "&", "<" et ">" ne sont pas présents. Cependant, nous sommes conscients qu'il existe tout de même des risques. Dans notre cas, l'utilisateur qui lance le programme C est l'utilisateur d'Apache, www-data, qui n'a pas les droits de sudo ni aucun droit particulier sur le système, ses capacités d'action sont donc assez limitées, mais non nulles.

1.6.5 Traitement des données de sortie

En sortie, le programme C retourne les coordonnées des différentes bornes par lesquelles l'utilisateur va devoir passer, dans l'ordre, selon le modèle suivant : lat0,long0+...+latN,longN. Le code en PHP va séparer cette chaîne de caractères en sous-chaînes avec le séparateur "+", puis chaque sous-chaîne sera séparée de nouveau avec le séparateur ",".

Ensuite, l'utilisateur voit apparaître une carte sur la page Web, qui comporte les différentes étapes de son trajet : emplacement de départ, des différentes bornes où il doit aller, et d'arrivée. S'il le souhaite, il peut cliquer sur le bouton "Voir l'itinéraire avec Google Maps", qui lui ouvrira une page Google Maps dans un nouvel onglet, comportant l'itinéraire automatiquement rentré (via les coordonnées données dans l'URL), et envoyer cet itinéraire directement sur son téléphone portable.

1.6.6 Commentaires

L'API servant à convertir une adresse en coordonnées GPS peut aisément être changée. Dans ce projet, nous avons choisi d'utiliser l'API du gouvernement français car celle-ci ne nécessite pas de clé, et que le nombre de requêtes maximal est largement suffisant pour notre utilisation de développement. Cependant, une plus grosse structure avec des besoins plus importants pourra opter pour une API plus performante et autorisant un nombre d'appel par unité de temps plus élevé en acquérant une clé API grâce à l'un des abonnements que proposent de nombreuses entreprises spécialisées, toujours en fonction de ses besoins. Il pourrait également être possible d'implémenter un code permettant d'autoriser un nombre maximum de demande d'itinéraire, en filtrant par adresse IP par exemple. De cette manière, le nombre d'appels à l'API ne serait pas démesurément grand à cause d'un grand nombre de calcul d'itinéraire demandé par un seul utilisateur.

La carte a été plus difficile à mettre en place. En effet, Google Maps propose un service de carte à intégrer sur un site Web avec des données d'entrée, exactement ce dont nous avions besoin. Cependant, depuis quelques temps, ce service est payant, et le coût est bien trop élevé pour ce projet. Nous avons

donc choisi de nous tourner vers OpenStreetMap, un projet collaboratif de cartographie en ligne, et plus spécifiquement vers Leaflet, une bibliothèque JavaScript s'appuyant sur OpenStreetMap pour créer des cartes à intégrer sur un site Web gratuitement. Après quelques recherches sur le fonctionnement de Leaflet (et sur le JavaScript en général), nous avons réussi à afficher une carte ayant des emplacements marqués. Cependant, cette carte ne peut pas comporter d'itinéraire, mais seulement les points : Leaflet ne gère pas le calcul exact d'itinéraire. Nous avons donc choisi d'utiliser Google Maps : l'efficacité de ses itinéraires n'est plus à démontrer, et de nombreux utilisateurs se servent de Google Maps pour leurs trajets, et ils peuvent directement envoyer leur itinéraire sur leur téléphone, ce qui facilite leur trajet. Nous avons donc analysé l'URL d'un itinéraire avec étapes de Google Maps. Une fois cela compris, nous avons fait un code en PHP qui respecte la disposition des coordonnées dans l'URL de Google Maps afin de créer automatiquement un bouton redirigeant l'utilisateur vers ce dernier. Ainsi, nous optimisons le choix des bornes, et Google Maps donne le trajet exact.

2 Deuxième partie - Simulation

La deuxième partie du projet consiste à réaliser un mode simulation. Il doit permettre d'étudier l'état du réseau lorsqu'un grand nombre d'utilisateurs le sollicite en même temps. Nous avons établi une liste des fonctionnalités à implémenter :

- Discrétiser le temps par pas de 10 minutes
- Créer un tableau pour chaque borne qui recense l'état d'occupation de la borne au cours de la journée
- Générer un ensemble d'usagers avec des itinéraires différents
- Prendre en compte l'occupation de chaque borne sous forme de temps d'attente dans le calcul des bornes les plus proches dans l'algorithme de Dijkstra.

2.1 Grandes lignes de l'algorithme

Sur la base de cette analyse, nous avons écris un algorithme en pseudo-code qui permet de réaliser les fonctions attendues :

- On génère N usagers parmi un ensemble de bornes définies (les grandes villes) et leurs destinations d'arrivées (aléatoirement ou prédéfinies)
- Temps = 0
- Pour chaque borne, on initialise un tableau qui stockera le temps d'attente à chaque instant
- On calcule tous les trajets nécessaires
- tant que tout le monde n'est pas arrivé :
  - On fait avancer les voitures de l'équivalent de 10 minutes : vitesse*Temps
  - On met à jour les temps d'attente à chaque borne et on le stocke dans le tableau
  - On actualise le trajet de chaque véhicule arrêté à une borne (on n'interrompt jamais un trajet commencé)
  - Temps += 10
- Fin, on peut exploiter l'ensemble des tableaux obtenus

2.2 Structures nécessaires

Si nous n'avons pas eu le temps de terminer la partie 2, voici tout de même une liste des structures qui avaient été mises en place dans le code :

— Bien sûr, nous nous sommes resservis de toutes les structures mises en place dans la partie 1
— Un tableau d'entiers représentant les temps d'attente
— Un tableau d'entiers contenant les destinations
— Un tableau contenant les Utilisateurs
— Un tableau d'entiers contenant (nombre de bornes) entrées, chaque entrée correspondant au nombre d'utilisateurs initialisés à chaque borne
— Un tableau de "trip" contenant les trajectoires de chaque utilisateur.
— Un tableau de véhicules contenant les véhicules de tout les usagers simulés (on numérotera les usagers selon les bornes auxquels ils ont été initialisés).

3 Gestion de projet

3.1 Outils mis en places

Afin de pouvoir assurer un suivi continu et efficace de notre travail, nous avons mis en place plusieurs outils.

Premièrement, nous avons rapidement mis en place des branches différentes sur Gitlab, afin de pouvoir travailler en parallèle sur le projet. Pour cela, une de nos premières taches a été de se renseigner sur l'utilisation des branches Git. Ainsi, la branche master a permis d'ajouter les fichiers fonctionnels alors que les branches personnelles ont permis de travailler sur des fichiers qui ne fonctionnent pas encore ou qui ont besoin d'être développés.

Ensuite, un serveur Discord nous a permis de communiquer tout au long du projet, de partager des fichiers et de faire des réunions en distanciel.

Un Work Breakdown Structure nous a permis de réfléchir à l'avance sur les structures et fonctions nécessaires au développement du projet.

Des réunions hebdomadaires ont permis de contrôler l'avancement de chacun et de définir des objectifs. L'ensemble des comptes rendus est présent en annexe.

3.2 Retours d'expérience

3.2.1 Kieran DALIGAUD

Un projet comme celui-ci apprend forcément beaucoup sur les difficultés du travail en groupe sur le moyen terme. Bien qu'il ait été utile pour travailler le langage de programmation C, ce projet aurait pu aller beaucoup plus loin avec une meilleure organisation. Les tests tardifs de fonctions essentiels nous ont notamment beaucoup pénalisé.

3.2.2 Arthur GÔMES

À mon sens, le plus gros problème rencontré est celui du planning : la répartition des taches a entraîné des dépendances entre les actions des

membres sur une même semaine. Cela a entrainé beaucoup de retard. La programmation collective en C a apporté de nouveaux problèmes d'organisation comme la gestion de la compilation et des dépendances sur les différentes branches.

3.2.3 Thomas LERUEZ

Le calendrier serré sur une période d'examens a été, selon moi, le principal enjeu de ce projet. De par la construction de celui-ci, il a été très difficile de mettre des tâches en parallèle, car beaucoup d'entre elles étaient interdépendantes.

3.2.4 Cosimo UNGARO

Selon moi, le problème principal a été le planning. En plus d'avoir relativement peu de temps, nous avons dû composer avec les partiels approchant ; de plus, la dépendance des parties les unes aux autres (par exemple, la partie 2 qui s'appuie sur la partie 1) a rendu quasiment impossible le fait de prendre de l'avance.

4 Annexes

4.1 Réunion de lancement : 24/03

Ordres du jour :
— Analyse sujet
— Recherche, identifier le problème à résoudre
— Début de la gestion de projet
— Les ressources disponibles (bases de données, carte...)
— les branches Git

Il faudrait lire l'entièreté du sujet en détail, trouver des pistes de départ et l'ensemble des points exigés.
Il faudrait aussi se renseigner sur les ressources disponibles pour prendre en compte les points de bornes de recharge à utiliser dans le code.

Le site *data.gouv.fr* est évoqué, Thomas l'a déjà utilisé, ce qui pourra servir. Dans une plus longue perspective, il est évoqué le fait d'utiliser autre chose que le langage C pour présenter les résultats finaux.

TODO list :
— Les membres doivent commencer un état de l'art sur le sujet pour avoir une idée précise sur ce qui pourrait être intéressant de produire
— Les membres doivent apprendre à utiliser Git Branch pour mieux s'organiser avec les commits et la gestion de versions de chaque code

Prochaine réunion : 31/03

4.2 Réunion d'avancement : 31/03

Ordre du jour
— Retour sur le travail de chacun
— Premiers documents de la gestion de projet
— Début de l'écriture du code

Cosimo commence : la base de donnée des véhicules annoncée est payante, on va sûrement avoir besoin de faire du scrapping. Il présente aussi son analyse du sujet qu'on peut trouver sur le lien overleaf partagé.

La question de la façon dont les comportements des utilisateurs doivent être générés est levée. Une option est de faire des groupes d'utilisateurs similaires, mais il y aura alors des files d'attentes très longues et inévitables.

Pour la prise en compte des temps d'attente, deux options ressortent : donner des poids aux arcs (les distances) et des poids aux sommets (temps d'attente), ou alors donner les poids des arcs sous forme de tuples (distances / temps d'attente) et complexifier le calcul du poids d'un chemin.

Thomas poursuit, il a analysé le site *data.gouv.fr*. Il y a une grande quantité de données (points GPS) mais elles ne sont pas centralisées.

Arthur propose différents algorithmes possibles : Dijkstra est le plus classique, A* en est un variante qui peut être plus efficace. Enfin, il y a des algorithmes de colonies de fournies, ils peuvent être adapté, mais complexe. Il faudrait y réfléchir davantage dans le future.

Kieran propose l'algorithme de BellMan-Ford. Vue la diversité d'algorithmes possibles, on pourrait tous les implémenter puis choisir celui qui présente les meilleurs résultats

TODO list
— **Thomas & Cosimo** : Scrapping et mise en forme des données (*data.gouv.fr*) bornes et véhicules
— **Kieran & Arthur** : Élaborer un plan théorique de l'algorithme
— **Arthur** : Commencer le Work Breakdown Structure

prochaine réunion : 06/04 17h

4.3 Réunion d'avancement : 06/04

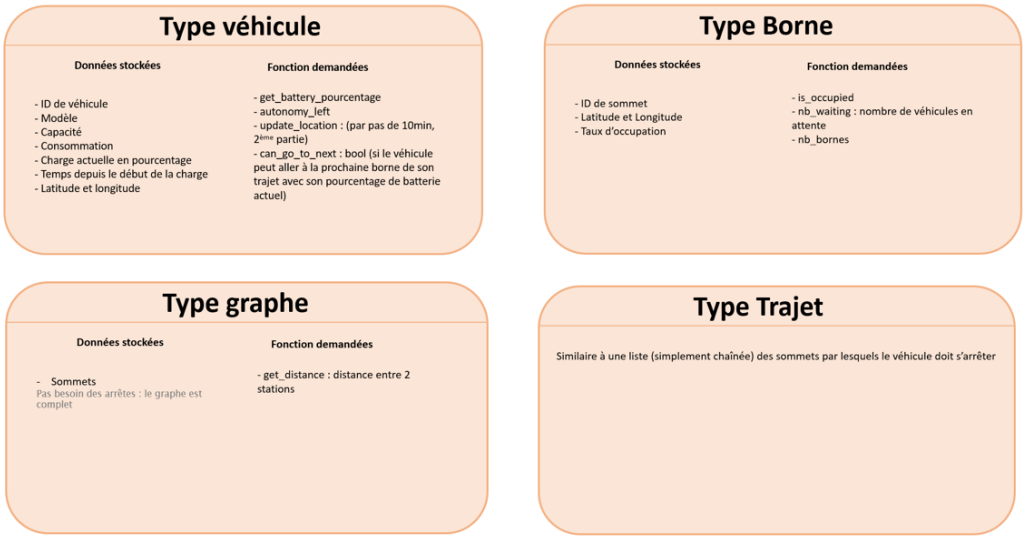
Ordre du jour

— Retour sur le travail de chacun
— Début du code et répartition du travail

Thomas a fait un script Python qui transforme un fichier CSV en un ensemble de coordonnées des stations. Cependant, le format n'est pas le même selon la source.

Cosimo indique qu'il n'a pas eu besoin d'effectuer un scrapping car un fichier CSV de l'ensemble des données à été partagé. Pour la base de donnée, qui devra ultimement être en C, Cosimo propose une table de hachage (accès en temps constant) ou une base de données non relationnelle. Arthur propose de charger entièrement le graphe depuis un fichier texte ou CSV, car le programme a besoin du graphe entier pour effectuer ses calculs.

Kieran a proposé un schéma des classes nécessaires.



Arthur présente une version modifiée de l'algorithme de Dijkstra (mettre schéma), ainsi qu'un découpage des actions à réaliser sous la forme d'un Work Breakdown Structure

TODO list

— **Thomas** : Écriture du type Borne
— **Cosimo** : Écriture du type Véhicule
— **Arthur** : Écriture de l'algorithme de Dijkstra modifié
— **Kieran** : Définition du type Graphe, fonctions de base

prochaine réunion : 13/04 17h

4.4 Réunion d'avancement : 13/04

Ordre du jour
— Retour sur le travail de chacun

Thomas a fait des types relatifs aux bornes, à mettre dans un tableau de bornes.

Kieran a fait des structures relatives aux graphes : Graph, Link, Trip

Arthur a fait un algo de Dijkstra en pseudo-code

Cosimo a fait un structure pour les véhicules Base de données : il faut la rendre utilisable en C. Problème : ce qu'on avait prévu comme structure ne correspond pas à la base de données. Pas de capacité d'un véhicule, rien pour calculer le temps de recharge. Il va tout retraduire en tableaux (à la place de liste), et voir comment exporter/importer en txt.

TODO list
— **Thomas** : Fonction de calcul de distance entre 2 coordonnées GPS et fonction déterminant la borne la plus proche avec des coordonnées GPS données.
— **Cosimo** : Fonction pour transformer un fichier txt/csv en structure C
— **Arthur** : Intégrer son algorithme aux structures de données existantes (avec Kieran), et faire que les différents éléments marchent ensemble
— **Kieran** : Intégrer son algo aux structures de données existantes (avec Arthur)

prochaine réunion : 04/05 17h

4.5 Réunion d'avancement : 04/05

Ordre du jour
— Retour sur le travail de chacun
— Attribution du travail à accomplir pour la semaine à venir
— Faire un point sur la situation (échéances, ...)
— Tableau ou liste de bornes ?
— Utilité du graphe ?

Retour travail de la semaine
— **Arthur** : il a besoin de la structure de bornes, sur laquelle Thomas est en train de travailler. Thomas précise qu'il fait un tableau d'objets de type borne pour pouvoir y accéder en temps constant (l'identifiant d'une borne sera son indice dans le tableau) Kieran explique le type graphe qu'il a implémenté, il devrait surtout servir pour la deuxième partie, autant le mettre en place le plus tôt possible. Arthur soulève que cette structure différencie les trajets qui mènent à une même borne, et ne permet donc pas de prendre en compte efficacement le temps d'attente dans le calcul
— **Thomas** : il a réalisé la fonction de calcul entre deux coordoonnées GPS ainsi que la fonction pour trouver la borne la plus proche d'une coordonnée donnée. Il a encore des difficultés pour réaliser la fonction qui transforme le fichier CSV en structure C.
— **Cosimo** : Ouvir le CSV : ok
Faut-il garder direct le CSV ? Oui
(mettre 100 en charge et demander pour L et l (depuis une addresse ? ?))

— **Kieran** : il a réalisé le type graph et commencé la fonction main. Il doit vérifier si les données CSV sont compatibles avec les fonctions qu'il a écrites.

Arthur souligne qu'il ne reste plus beaucoup de temps, surtout pour la deuxième partie du projet.

Arthur voudrait pouvoir ajouter des points un à un dans le graphe et le tableau

Thomas propose donc d'encoder un tableau "à la python"

Arthur souligne qu'il faudrait vite faire des fonctions de tests sur tout ou partie des données.

TODO list
— **Thomas** : Terminer le travail en cours
— **Cosimo** : Finaliser en rajoutant toutes les données + écrire test pour les fonctions de Arthur.
— **Arthur** : Terminer le travail en cours
— **Kieran** : Continuer à faire fonction main

prochaine Réunion : 16/05

Objectif : Terminer le travail en cours

4.6 Réunion d'avancement : 16/05

Ordre du jour
— Retour sur le travail de chacun
— Point sur les dernières taches à finir en urgence
— Format des données pour l'interface Web
— Point sur les fonctions d'initialisation

Retour travail de la semaine
— **Arthur** : il réorganise la branche master, il explique les quelques changements qu'il a faits. Il a des questions sur les fichiers d'initialisation des structures de véhicules et de bornes à partir des fichiers CSV. Cosimo doit encore les terminer. Arthur demande aussi sous quel format l'interface Web a besoin des données du trajet. Thomas répond : il faut la liste des coordonnées sous le format suivant 'lat1,long1+...+latn,longn'. Arthur présente un algorithme de simulation auquel il a pensé pour la deuxième partie.
— **Kieran** : il souligne qu'il faut prendre en compte le temps maximum d'attente à une borne et le niveau minimale en dessous duquel il ne faut pas descendre. Kieran et Arthur implémenteront ces deux fonctionnalités. Kieran a modifié des fonctions pour qu'elles soient compatibles entre elles. Kieran remarque que le tableau n'est nécessaire que pour le choix du véhicule, on peut ensuite ne garder que ses caractéristiques.
— **Thomas** : il présente son interface Web. Il demande ce qu'il doit ajouter, réponse : une liste déroulante des véhicules, temps maximum d'attente pendant un recharge et pourcentage de batterie en dessous duquel ne pas descendre. Thomas explique qu'il a besoin de pouvoir donner toutes les entrées de la fonction main en argument et récupérer le résultat sur la sortie standart. Thomas propose d'ajouter une option à la fonction main, pour pouvoir différencier le cas où l'on n'a besoin que du tableau de véhicules, où l'appel véritable à main.
— **Cosimo** : il explique qu'il a travaillé sur les fonctions d'initialisations, pour l'instant elles ne stockent pas les structures dans un tableau.

TODO :
— Arthur : ajouter les fonctionnalités à l'algorithme de Dijkstra, commencer le rapport
— Cosimo : finir les fonctions d'initialisations, réfléchir aux structures

nécessaires à l'algorithme de simulation (Deuxième partie)
— Thomas : finir l'interface et la connecter à fonctions d'initialisations des données
— Kieran : fonction pour affihcer les coordonnées d'un objet de type Trip, finir le fichier main

4.7 Pseudo-code de l'algorithme de Dijkstra modifié

- On initialise une liste d telle que $d[s] = +\inf$ pour tout sommet s
- On initialise une liste pre de prédécesseurs.
- $d[a] = 0$
- $P = \{a\}$
- tant que b n'est pas dans P
  - Choisir a n'étant pas dans P de valeur de d minimale
  - Pour chaque sommet b voisin de a et tel que distance(b, a) <range
  - Si $d[b] > d[a] + distance(a, b)$ alors
  - $d[b] = d[a] + d(a, b)$
  - $pre[b] = a$
  - finsi
- finpour
- finsi
- fintantque