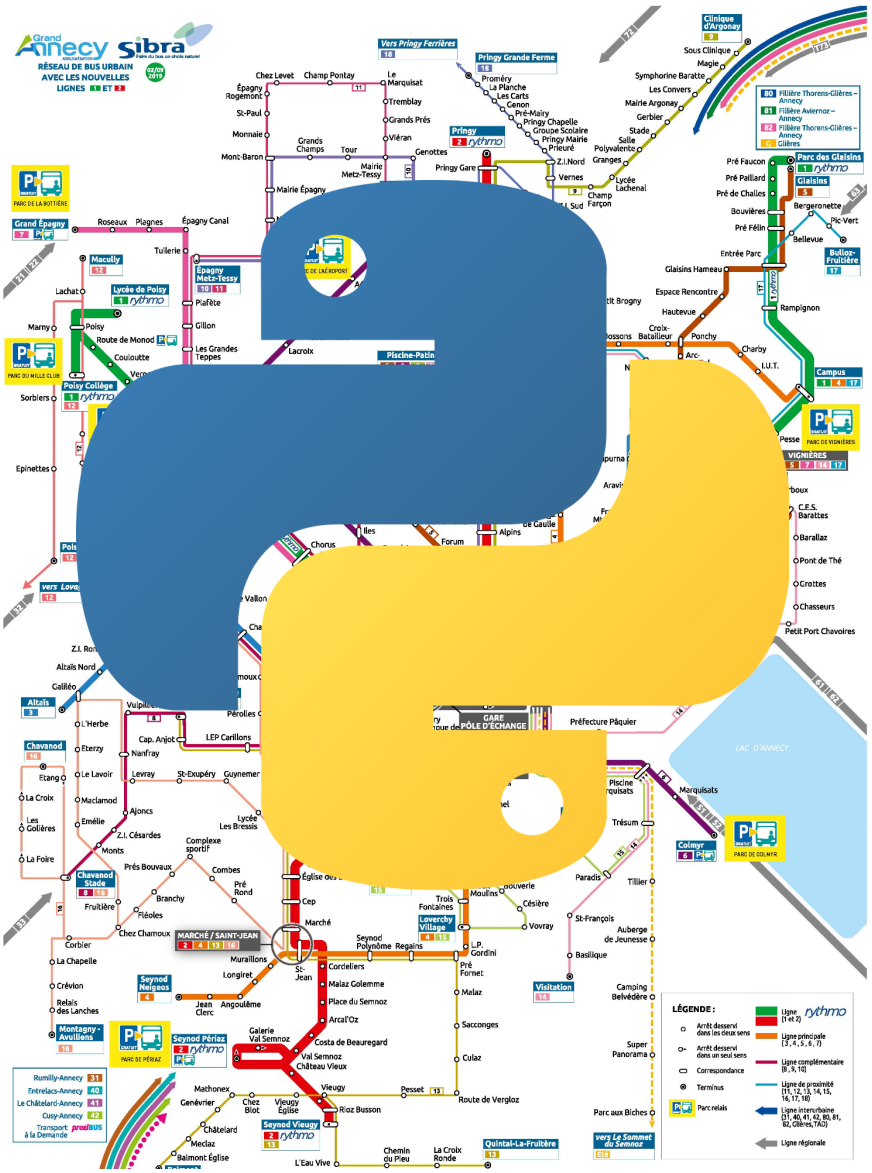
Arthur GONAY
IDU3 - PROJ631
https://github.com/Brocowlee/PROJ_ALGO

Compte rendu n°1 projet algorithmique: Gestion d'un réseau de bus



Introduction

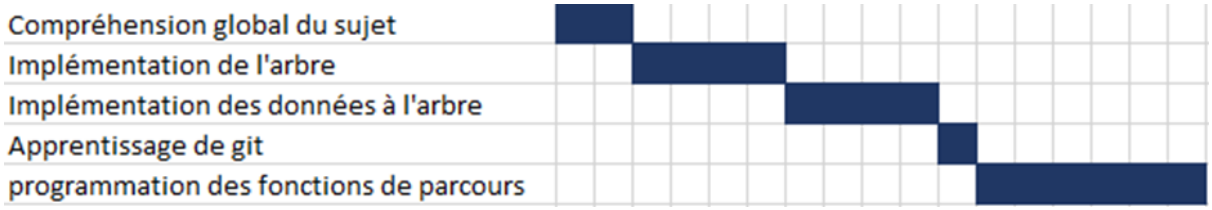
Ce projet a été réalisé dans le cadre de mon école d'ingénieur Polytech Annecy-Chambéry lors du module PROJ631.

Pour ce projet d'algorithme j'ai choisi de traiter le problème de la gestion de bus, principalement parce que l'étude des strucutres d'arbres est un sujet qui m'intéresse et qu'il était facile de comprendre l'aspect pratique de ce genre de problématique.

Le premier objectif de ce sujet est donc la création d'une structure de données afin d'y ajouter les données disponibles avec l'énoncé. Le second objectif est la création de fonctions de parcours remplissant certains critères. Dans ce rapport nous suivrons la progression chronologique du projet.

Prévision initiale des tâches:

La première étape de ce projet a été la compréhension du sujet et sa segmentation en différentes tâches distinctes. Ma planification initiale prévu était la suivante:



Je pensais qu'environ 16h de travail seraient suffisantes pour terminer ce travail.

Implémentation de la structure de donnée

La structure de données que j'ai créé est assez classique pour un arbre. Une classe Stop contient les informations liées aux arrêts. En plus du nom de l'arrêt, les informations les plus importantes telles que les arrêts suivants, les lignes qui le parcourent, les horaires normales et les horaires des vacances sont contenues dans des listes parcourue en parallèle.

Par exemple l'arrêt GARE qui est un des croisement des 2 lignes données dans l'énoncé aura dans son attribut *next_stop* les 4 arrêts qui le succède ["Mandallaz","France Barattes","Courrier","Bonlieu"].

Son attribut *ligne* contiendra les lignes que le parcours avec des indices correspondant aux indices de la liste next_stop. Par exemple pour l'arrêt gare on aura [ligne n°1 - direction:PARC_DES GLAISINS, ligne°1 - direction: LYCÉE_DE_POISY, ligne n°2 - direction: CAMPUS,ligne n°2 - direction: PISCINE-PATINOIRE].

Sur le même principe *horNormales* et *horVacances* sont des liste des horaires avec des indices correspondant aux lignes.

```
class Stop:
    def __init__(self,name):
        self.name=name
        self.next_stop=[]
        self.ligne=[]
        self.horNormales=[]
        self.horVacances=[]
```

La classe *lignes* possède l' attribut *num*, contenant le numéro de la ligne, *start*, contenant l'arret de départ de la ligne, et distination contenant le dernier arret de la ligne. Par exemple la ligne 1 allant de LYCÉE_DE_POISY à PARC_DES_GLAISINS aura *num=1, start= LYCÉE_DE_POISY, direction= PARC_DES_GLAISINS*.

```
class Ligne:
    def __init__(self,num):
        self.num=num
        self.start=None
        self.direction=None
```

La classe *ReseauBus* possède l'attribut *lignes*, contenant toutes les lignes du réseau et *allStops* contenant tous les arrêts du réseau.

```
class ReseauBus:
    def __init__(self):
        self.lignes=[]
        self.allStops=[]
```

Ce choix de structure de données n'a pas été le premier auquel j'ai pensé. En effet, j'ai d'abord pensé que créer un arrêt pour chaque ligne, avec l'arrêt qui le succède, ne posait pas problème. Cependant, la gestion des changements de ligne aurait été bien plus complexe.

Implémentation des données

Pour implémenter les données du sujet dans ma structure de donnée nous disposions du fichier *Data2py.py* qui se charge de transformer les données contenu dans les fichier texte en un dictionnaire python que nous pourrons plus facilement manipuler.

J'ai créé une fonction *imple(num,sens,reseau)* dans mon fichier main qui construit les lignes n°*num* avec les données du sujet dans un sens ou l'autre selon si la variable booléenne *sens* est à *True* ou *False*. je récupère aussi le reseau pour avoir l'informations de quels arrets ont déjà été créés

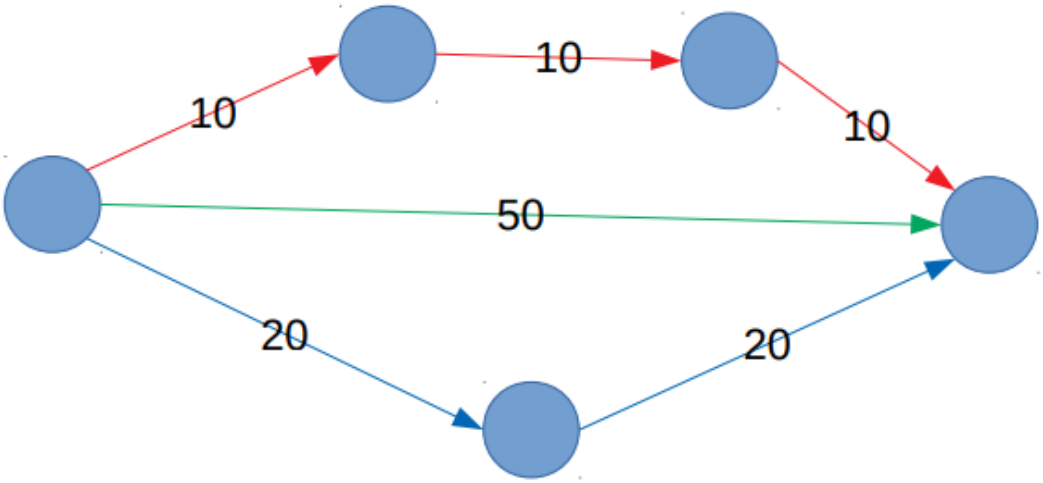
Pour cela je récupère le bon dictionnaire selon la ligne que je veux créer, parcours les clés, et si l'arrêt n'a pas encore été construit (pas présent dans la liste d'arrêt de réseau), je le crée et ajoute les différentes données. Si l'arrêt est déjà présent dans la liste de réseau, je le récupère avant, ici aussi, d'ajouter toutes les données. Lorsque le premier arrêt est parcouru, il est ajouté à la ligne en tant qu'art de départ. Même principe pour le dernier arrêt que j'ajoute à la ligne en tant que arrêt destination.

```
sybra=ReseauBus()
ligne1go=imple(1,True,sybra)
sybra.addLignes(ligne1go)
ligne1back=imple(1,False,sybra)
sybra.addLignes(ligne1back)
ligne2go=imple(2,True,sybra)
sybra.addLignes(ligne2go)
ligne2back=imple(2,False,sybra)
sybra.addLignes(ligne2back)
```

Programmation des fonctions de parcours avec dijkstra

La partie la plus complexe de ce projet était sûrement le parcours de l'arbre. Il nous a été demandé de réaliser trois fonctions de parcours de notre réseau de bus.

- "shortest"(en vert), devait définir le chemin le plus court en nombre d'arcs.
- "fastest"(en rouge, partant à 10h15), devait définir le chemin le plus rapide sans prendre en compte le nombre d'arcs
- "foremost"(en vert, partant à 10h00), devait définir le chemin arrivant le plus tôt



Chacune de ces fonctions utilise l'algorithme de Dijkstra qui permet de trouver le chemin le plus court dans un arbre. Pour cela on utilise une liste des arrêts parcourue et un dictionnaire dans lequel on écrit la distance entre l'arrêt de départ et l'arrêt actuel ainsi que l'arrêt précédent, afin de récupérer le chemin. Par le mot distance j'entends ici la distance dans l'arbre entre 2 nœuds, c'est à dire simplement 1 pour shortest dans lequel seul le nombre compte.Cependant pour "fastest" et

“foremost” il faut pondérer les arcs avec le temps de trajet pour le premier et l’heure d'arrivée pour le second.

- Shortest

Le plus simple des 3 fonctions de parcours, la plus grosse difficulté a été la compréhension et l’implémentation de l’algorithme de dijkstra. Un autre problème que j’ai rencontré était la possibilité de récupérer le chemin, car cela n’est pas présent dans l’algorithme de dijkstra de base. Pour cela j’ai ajouté l’information de l'arrêt en plus dans le dictionnaire contenant les distances.

```
def shortestDijkstra(self,start,end):
    listAllStops=[]
    for e in self.allStops:
        listAllStops.append(e)
    startingStop=self.findTheStop(start)
    endingPoint=self.findTheStop(end)
    dicoShortest=self.initDicoShortest()
    listAllStops.remove(startingStop)
    dicoShortest[startingStop][0]=0
    def shortestBis(self,current,end,listAllStops,dicoShortest):
        if current==end:
            return dicoShortest
        dicoShortest=majDicoArcs(current,dicoShortest)
        newCurrent=getNewCurrent(listAllStops,dicoShortest)
        listAllStops.remove(newCurrent)
        return shortestBis(self,newCurrent,end,listAllStops,dicoShortest)

    return shortestBis(self,startingStop,endingPoint,listAllStops,dicoShortest)
```

La fonction shortest interprète le dictionnaire retourné par la fonction *shortestDijkstra* pour retourner le nombre d’arcs ainsi que le chemin utlisé.

```
def shortest(self,start,end):
    res=""
    dico=self.shortestDijkstra(start,end)
    for cle,value in dico.items():
        if cle==self.findTheStop(end):
            current=self.findTheStop(end)
            res+="en "+ str(value[0]) + " arcs"+"\n"
            while current!= self.findTheStop(start):
                res+=str(current)+"\n"
                current=dico[current][1]
    res+=str(self.findTheStop(start))
    return res
```

- Fastest

Fastest fonctionne sur le même principe que shortest mais la gestion des temps de trajet rajoute une couche de difficulté. Pour cela il fallait créer plusieurs fonctions de gestion du temps et d’interprétation des horaires donnée dans le sujet. J’ai ainsi créé 3 fonctions:
-*nbmin(heure)* qui transforme la chaîne de caractère heure en un int correpondant au nombre de minute

-*nextBus(reseau,heure,stop,ligne)* qui indique quand le prochain bus d'une certaine ligne passe à l'arrêt stop.
-*tempsTraj(reseau,heure,stop,to)* qui calcul le temps que mettre le bus partant de *stop* pour aller à *to.*

Ces fonctions permettent de créer *fastest* qui sur le même principe repose sur la fonction *fastestDijkstra* qui renvoie un dictionnaire, lui-même interprété par la fonction *fastest*.

Je n'ai pas pu traiter la fonction foremost par manque de temps mais l'idée globale du code aurait été la même que pour les 2 autres fonctions de parcours.

Difficultés rencontrées

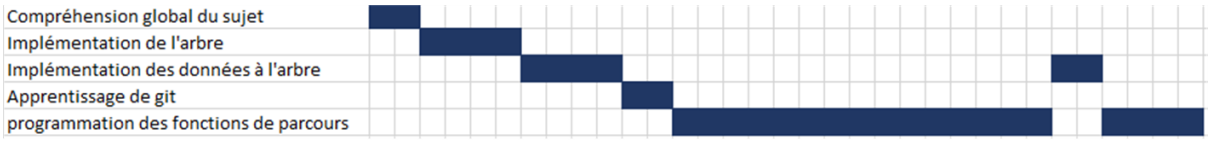
Même si mon projet s'est globalement bien passé, certaines difficultés se sont posées à moi tout le long du développement. La plus grosse a sûrement été la fonction de parcours de l'arbre que j'ai essayé moi même de coder sans utiliser l'algorithme de Dijkstra. J'ai perdu un temps considérable à essayer de créer ma propre fonction récursive mais sans succès. Ce n'est que deux jours avant la fin du projet que j'ai demandé de l'aide à M. Vernier qui m'a expliqué quel algorithme il fallait utiliser pour ce problème. J'ai donc eu très peu de temps pour écrire les 3 fonctions demandées. La prochaine fois qu'un problème de ce genre se pose à moi je saurais que regarder sur internet ce qu'il existe déjà peut économiser de longues heures de travail.

Une autre difficulté rencontrée a été le manque de méthode sur la création de structure de données. En effet je me suis attardé sur la création de fonctions de gestion du temps alors que ma classe reseauBus n'était même pas créait. Je m'étais pourtant fixé des tâches précises à suivre tout le long du projet mais je m'en suis malheureusement assez vite éloigné, me faisant perdre beaucoup de temps là encore.

Bilan

Les prévisions faites en début de projet ont au final été assez loin de la réalité, avec beaucoup de temps perdu sur certaines parties dû à des erreurs de ma part. Le temps consacré à ce travail a été d' environ 33 h soit plus du double de ce que j'avais initialement prévu, sachant qu'une des fonctions demandées est

manquante.



Ce projet m’a beaucoup apporté, autant en termes de compétences de programmation que de gestion de projet. Certaines erreurs ont été commises, surtout concernant la gestion de projet, mais je ne les reproduirai pas et m’efforcerai de mieux cadrer mes projets à l’avenir.

D’un point de vue programmation, j’ai pu apprendre à utiliser git sur un projet concret. Il ne fait aucun doute que cela me sera très utile pour mes futurs projets informatiques. De plus, j'ai pu améliorer ma compréhension des algorithmes récursif.