Valentin Rivière **Gabriel Ménévis**

Cahier des charges

Projet C : Chemin le plus court

Table des matières

Exigences générales

L’objectif de ce programme est de calculer puis d’afficher le chemin le plus court entre deux points choisis par l’utilisateur, à partir de données réelles sur le périmètre du 13e arrondissement parisien.

Les points choisis correspondent à des intersections de rues, et l’extrémité de certaines rues et impasses. Ils sont reliés entre eux par des segments, dont on doit connaître la longueur et associés chacun à une voie.

Calcul de l’itinéraire

La fonctionnalité fondamentale du programme est la possibilité de calculer un itinéraire entre deux points. Cette fonctionnalité implique un certain nombre de sous-fonctionnalités et de règles à mettre en place concernant la circulation routière, détaillées ci-après.

Choix des points de départ et d’arrivée

Avant de lancer un calcul d’itinéraire, l’utilisateur doit pouvoir choisir un point de départ et un point d’arrivée. Il faut donc afficher les intersections de rues une première fois, lui demander de saisir un numéro correspondant à une intersection, et réitérer l’opération pour le choix du point d’arrivée.

Le calcul d’itinéraire se fait sur deux points distincts : l’utilisateur ne doit pas pouvoir choisir le même point d’arrivée que le point de départ préalablement sélectionné.

Moyen de transport et règles de circulation

L’utilisateur doit pouvoir choisir son moyen de transport (à pied ou en voiture). Le programme doit tenir compte de ce choix pour lui proposer un itinéraire qu’il puisse réellement parcourir avec le moyen de transport spécifié.

Pour cela, le programme doit prendre en compte les règles de circulation sur chaque segment de rue reliant deux croisements. Différents cas de figure sont à prévoir :

- piétons comme voitures peuvent circuler librement dans les deux sens
- la voie est à sens unique entre deux croisements : les voitures peuvent circuler dans un sens mais pas dans l’autre
- la voie est uniquement piétonne dans les deux sens

- la voie est réservée aux voitures dans les deux sens (boulevard périphérique dans le 13e arrondissement)

À notre connaissance, il n'y a pas de voie qui serait empruntable par les piétons dans un sens mais pas dans l'autre. Ce cas de figure n'est donc pas prévu par le programme.

Par ailleurs, il faut noter que le sens de circulation d'une voie peut varier. Ainsi, le sens de circulation ne doit pas être associé aux voies elles-mêmes, mais aux segments reliant deux croisements : les arcs. Un arc se voit obligatoirement affecter une et une seule règle de circulation. Si l'on observe qu'au sein d'un même arc la règle de circulation change, il conviendra de créer un nouveau noeud marquant l'endroit où la règle change, et diviser l'arc en deux nouveaux arcs.

Différents types de voies et croisements

Au-delà des règles de circulation, le programme doit prendre en compte qu'il existe différents types de voies et d'intersections. D'une part, plus de deux voies peuvent se croiser au même endroit (c'est par exemple le cas des places). D'autre part, les impasses, les villas et certains passages sont des voies qu'il faut pouvoir parcourir, mais dont une extrémité au moins ne débouche sur aucune autre voie. Enfin, deux voies peuvent se croiser plus d'une fois : c'est le cas de la rue Sœur Catherine Marie dans le 13e arrondissement, qui croise deux fois la rue de la Glacière, mais également de la rue de Sainte Hélène qui croise deux fois la rue de la Poterne des Peupliers.

Pour ces raisons, les points dans l'espace dont on peut partir, où on peut se rendre et par lesquels on peut passer, ne sauraient être définis simplement comme les croisements entre deux voies. On parlera donc plutôt de nœuds, qui sont situés sur une voie au moins mais sans maximum de voies associées. Ainsi, le fond de la Villa Auguste Blanqui est considéré comme un nœud, bien qu'il ne s'agisse pas d'un croisement. La place d'Italie est également considérée comme un nœud unique, où se croisent neuf voies différentes (boulevard de l'Hôpital, avenue des Gobelins, avenue de la Sœur Rosalie, boulevard Auguste Blanqui, rue Bobillot, avenue d'Italie, avenue de Choisy, boulevard Vincent Auriol et rue Godefroy).

Croisements spécifiques à un moyen de transport

Il peut arriver que deux voies soient empruntables par tous types de véhicules, mais qu'à leur intersection seuls les piétons puissent passer de l'une à l'autre.

Par exemple, le boulevard de Port-Royal passe par-dessus la rue Broca puis la rue Pascal : les piétons peuvent descendre l'escalier pour passer du premier à la deuxième, mais les voitures n'ont d'autre choix que de continuer sur le boulevard de Port-Royal. Le programme doit donc prendre en compte l'existence de ce croisement, mais ne pas permettre aux voitures de bifurquer.

Un deuxième cas de figure se présente : lorsqu'une voie est à double-sens pour les voitures, mais qu'il faut se situer d'un côté donné de la voie pour tourner dans une autre (par exemple lorsqu'un

terre-plein central sépare les deux sens de circulation). Ainsi, une voiture engagée sur le boulevard Masséna vers l'est peut tourner à droite dans la rue Michel Bréal, mais cette manœuvre est impossible si le véhicule va d'est en ouest : il doit alors poursuivre jusqu'à la porte d'Ivry et faire demi-tour sur le boulevard pour rejoindre la rue Bréal. La structure des données doit permettre de prendre en compte ce cas – sans oublier que, pour les piétons, il est possible de rejoindre la rue Bréal depuis le boulevard Masséna quel que soit le sens de circulation.

Itinéraire impossible entre deux croisements

Il peut arriver que deux points sélectionnés par l'utilisateur ne puissent être reliés l'un à l'autre. Cela se produit classiquement lorsqu'on cherche à atteindre en voiture le fond d'une impasse réservée aux piétons (par exemple, si l'on veut aller de Auriol – Jeanne d'Arc à Villa Blanqui). Cela ne devrait pas se produire lorsqu'on cherche à calculer un itinéraire à pied, sauf en cas de défaut dans les données chargées.

Dans tous les cas, si aucun itinéraire n'existe pour le moyen de transport choisi, le programme doit l'indiquer à l'utilisateur par le message d'erreur suivant :

« Il n'y a aucun itinéraire possible entre [nœud de départ] et [nœud d'arrivée] pour le moyen de transport choisi. »

Affichage de l'itinéraire calculé

Une fois l'itinéraire calculé, le programme doit le restituer à l'utilisateur d'une manière intuitive et au plus proche de la manière dont il peut se représenter mentalement un itinéraire : prendre la rue *X* de tel point à tel point, puis la rue *Y* de tel point à tel point, ainsi de suite jusqu'à ce que la destination soit atteinte.

L'affichage doit également faire apparaître la distance à parcourir sur chaque voie, ainsi que la distance totale parcourue entre les points de départ et d'arrivée.

Consultation des données

Pour pallier l'absence d'interface graphique et l'impossibilité de visualiser la disposition des voies et des intersections, l'utilisateur doit pouvoir consulter les données liées à une voie ou à un nœud.

Le programme doit donc proposer à l'utilisateur une interface de consultation des données chargées où il peut accéder à la liste des voies, aux nœuds liés à une voie, à la liste des nœuds et aux nœuds adjacents à un nœud donné.

Périmètre du projet

Pour récapituler, le programme que nous proposons prend en charge les fonctionnalités suivantes :

- affichage dynamique des données chargées
- sélection d'un point de départ et d'un point d'arrivée, à partir de la liste totale des voies ou d'une recherche de voies par mot-clé, dont on choisit ensuite parmi tous les points qui s'y situent
- prise en compte du mode de transport (piéton ou véhicule motorisé)
- calcul du chemin le plus court entre les deux points
- affichage simplifié (rue par rue) ou détaillé (segment de rue par segment de rue) du chemin calculé
- possibilité de chercher plusieurs itinéraires à la suite dans une même exécution du programme

Le périmètre géographique des données fournies couvre la majeure partie du 13e arrondissement parisien[1]. Toutefois, le programme est conçu pour fonctionner avec n'importe quel jeu de données, du moment que celui-ci respecte le format attendu, spécifié ci-dessous dans la section Structure des fichiers de données.

Toutes les données ont été collectées manuellement en utilisant Google Maps, qui offre une fonctionnalité de mesure de distance entre deux points.

Les fonctionnalités suivantes ne font pas partie du périmètre de notre programme :

- ajout, modification, suppression de données *via* l'interface utilisateur
- calcul d'itinéraires alternatifs entre deux mêmes points
- orientation de l'utilisateur lors des changements de rue (ex. : « Tournez à gauche rue *X* »)
- choix d'un point de départ ou d'arrivée qui ne soit pas une intersection ou l'extrémité d'une voie (par exemple un numéro de rue)
- prise en compte de la vitesse de circulation (limites de vitesse, encombrement du trafic)
- intégration des voies privées, galeries commerciales, etc. dans les données fournies

1 Confrontés à la masse importante de données à saisir et à mesurer, nous avons préféré restreindre légèrement le périmètre géographique fixé initialement et privilégier la qualité des livrables, compte tenu du fait que les données fournies permettent déjà bien de rendre compte de tous les cas de figure du programme.
Ainsi, sont exclues deux parties du 13e arrondissement : la partie nord-est délimitée à l'ouest par l'avenue des Gobelins et au sud par le boulevard Vincent Auriol ; une petite partie sud-est, délimité à l'ouest par la rue du Chevaleret et au nord par le boulevard Vincent Auriol.

Architecture du programme

Nous présentons ci-dessous l'architecture fonctionnelle prévue pour notre programme. Elle repose sur deux fonctionnalités principales : la consultation des données et le calcul de l'itinéraire. Chacune de ces fonctionnalités en implique d'autres, selon une arborescence détaillée ci-dessous. Une troisième fonctionnalité est offerte, permettant à l'utilisateur de charger ses propres fichiers de données.

I. ACCUEIL ET MENU DE CHARGEMENT

A. Affichage de l'encadré de titre et bienvenue

B. Menu de chargement

1. Affichage des données chargées

2. Option d'ajout de nouveaux jeux de données

a. Si NON, MENU PRINCIPAL

b. Si OUI, saisir un fichier de données pour les voies, nœuds, arcs

- Si les fichiers existent, affichage de chargement réussi et MENU PRINCIPAL

- Si l'un des fichiers n'existe pas, message d'erreur et fin du programme

II. MENU PRINCIPAL

A. Consulter les données chargées

1. Liste des voies : l'ensemble des voies chargées ainsi que leur identifiant s'affichent.

a. Consulter les informations d'une voie en saisissant son identifiant

- Affichage des nœuds liés à cette voie

- Consulter les informations d'un des nœuds : saisie du nœud souhaité

- Affichage des informations concernant les arcs liées à ce nœud

2. Liste des nœuds

a. Consulter les informations d'un nœud en saisissant son identifiant

- Affichage des informations concernant les arcs liées à ce nœud

3. Retour au menu principal

B. Calculer un itinéraire

1. Sélectionner une voie de départ dans la liste des voies

- Affichage de l’ensemble des voies ainsi que de leur identifiants

- Saisie de la voie de départ souhaitée et message d’erreur si saisie incorrecte

- Affichage de l’ensemble des nœuds liés à cette voix

- Saisie du nœud souhaité et message d’erreur si saisie incorrecte

- **Sélection de la voie d’arrivée, soit en sélectionnant la voie dans la liste des voies (1) soit par mot-clé (2)

- Si le nœud d’arrivée saisi est égal au nœud de départ saisi : message d’erreur tant que le nœud saisi est égal au nœud de départ

- Saisie du mode de circulation (voiture ou piéton) et message d’erreur si saisie incorrecte

- Saisie du mode d’affichage (voie par voie ou arc par arc) et message d’erreur si saisie incorrecte

- Affichage de l’itinéraire et distance parcourue

- Retour automatique au menu principal

2. Chercher une voie par mot-clé

- Saisie d’un mot-clé permettant de trouver la voie désirée. Si la voie n’est pas trouvée, message d’erreur et proposition de nouvelle recherche ou de revenir au menu précédent.

- Affichage des voies trouvées à partir du mot-clé saisi et du nombre total de voies trouvées

- Proposition de lancer une nouvelle recherche, de sélectionner l’une de ces voies ou de revenir en arrière. Si le choix est de sélectionner l’une des voies :

- Option de revenir en arrière

- Saisie de l’identifiant de la voie souhaitée et message d’erreur tant que la saisie est incorrecte

- Affichage de l’ensemble des nœuds liés à cette voix

- Saisie du nœud souhaité et message d’erreur si saisie incorrecte

- **Sélection de la voie d’arrivée, soit en sélectionnant la voie dans la liste des voies (1) soit par mot-clé (2). Même architecture à partir d’ici que dans le point II.B.1

3. Retour au menu principal

C. Charger un autre jeu de données

1. Saisir un fichier de données pour les voies, nœuds, arcs

a. Si les fichiers existent, affichage de chargement réussi et MENU PRINCIPAL

b. Si l'un des fichiers n'existe pas, message d'erreur et fin du programme

Calcul de l'itinéraire le plus court

Pour calculer l'itinéraire le plus court entre deux points, nous avons eployé un algorithme existant : l'algorithme de Dijkstra, du nom de son inventeur, l'informaticien néerlandais Edsger Dijkstra qui l'a publié en 1959. Nous expliquons ci-dessous son fonctionnement, détaillons la manière dont nous l'avons implémenté dans notre programme puis donnons un exemple précis d'exécution à partir de nos données.

Fonctionnement général de l'algorithme de Dijkstra

Une représentation en graphe de l'espace

L'algorithme de Dijkstra se fonde sur la théorie des graphes, qui étudie des modèles faits de sommets et d'arêtes reliant ces sommets.

Il s'applique notamment aux réseaux de sentiers, de routes, et peut permettre le calcul automatique de l'itinéraire le plus court entre une ville et une autre, entre un point A et un point B.

Dans la mesure où l'on cherche le chemin le plus court entre deux points en distance réelle, le graphe que nous construisons doit être valué : chaque arête doit avoir un « poids » positif, c'est en quelque sorte l'étiquette qu'on lui donne, correspondant en l'occurrence à la distance entre les deux sommets qu'elle relie, exprimée en mètres.

Par ailleurs, pour tenir compte des règles de circulation, il faut que le graphe soit orienté, c'est-à-dire que les arêtes puissent être dotées d'un sens. Une arête relie le sommet 2 au sommet 1, mais il n'existe pas forcément d'arête reliant le sommet 1 au sommet 2. Dans le contexte d'un graphe orienté, les arêtes sont communément nommées arcs et les sommets nœuds. C'est donc d'arcs et de nœuds que nous parlerons désormais. C'est aussi le nom par lequel on désigne ces données dans le programme.

Le fonctionnement général de l'algorithme peut se résumer ainsi :

On cherche à trouver le plus court chemin entre deux nœuds appartenant à un même graphe, en admettant par exemple qu'ils sont séparés par plusieurs autres nœuds. Tout au long du programme, nous allons garder en mémoire le chemin le plus court depuis le premier nœud pour chacun des

autres nœuds dans un tableau. Cet ensemble temporaire que l'on garde en mémoire des chemins les plus courts constitue un sous-graphe.

On répète alors systématiquement le même processus :

- On choisit le nœud, accessible le plus proche (ou dont la distance est la plus faible depuis le nœud où l'on se trouve) comme sommet à explorer.

- A partir de ce nœud, on explore ses voisins et, le cas échéant, on met à jour les distances pour chacun : si l'on a trouvé une nouvel itinéraire plus court que celui que l'on avait en mémoire pour ce nœud.

- On itère jusqu'à ce qu'on arrive au point d'arrivée et jusqu'à ce que tous les sommets aient été explorés.

Exemple d'application

Soit un ensemble de nœuds *S = {A, B, C, D, E, F, G, H}* et un ensemble d'arcs *A = {(A,B),(A,C),(C,A),(C,B),(B,F),(F,B),(G,F),(C,E),(C,D),(D,C),(E,D),(E,F),(F,H),(E,H),(H,E),(H,D)}*.

À chacun de ces arcs correspond un poids positif, comme on peut le voir sur la Figure 1 ci-dessous.

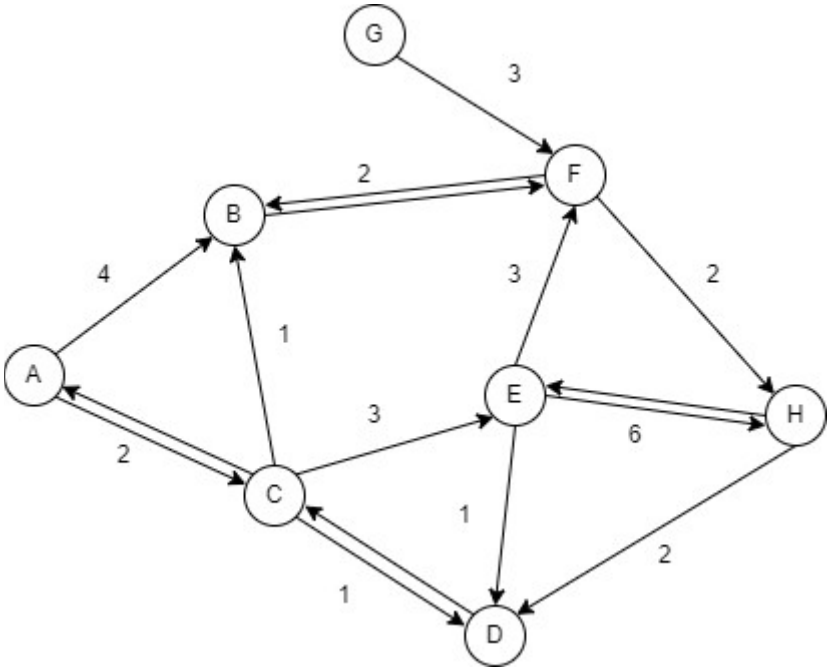

Figure 1 - Exemple de graphe orienté valué

Construire un sous-graphe pour trouver le plus court chemin

Pour trouver le plus court chemin entre deux points (par exemple, entre le point A et le point H de la Figure 1), il faut construire progressivement un sous-graphe qui intégrera petit à petit les nœuds du graphe général. À chaque nœud ajouté au sous-graphe, il faut noter la distance du chemin le plus court qui le relie au nœud de départ, ainsi que le nœud dont on provient par ce chemin.

Le sous-graphe ne comprend d'abord que le nœud de départ, ici A. Sa distance au nœud de départ est 0. Il faut explorer tous les nœuds où arrive un arc partant de A. Ici :

- on ajoute B. Sa distance au départ vaut *distance au départ de A + poids de (A,B)* soit 0 + 4 = 4
- on ajoute C. Sa distance au départ vaut *distance au départ de B + poids de (A,C)* soit 0 + 2 = 2

On vient d'ajouter les nœuds liés au sommet A : on considère donc que le nœud A est traité. On peut représenter l'état du traitement comme ci-dessous sur la Figure 2, où les nœuds noircis sont les nœuds traités ajoutés au sous-graphe, les nœuds gris sont ceux pour lesquels on connaît un chemin (pas forcément le plus court) et qu'on doit encore traités, et les nœuds blancs sont ceux qui n'ont pas encore été ajoutés.

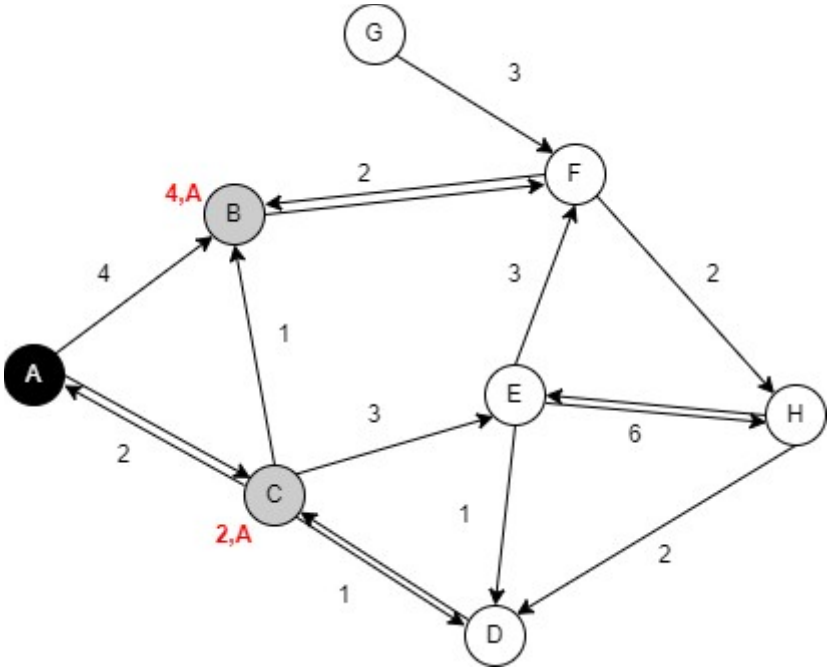

Figure 2 - Première étape du traitement

Le prochain nœud à traiter est celui pour lequel on connaît au moins un chemin, qui n'a pas été traité (apparaît grisé sur la Figure 2) et pour lequel la distance au départ est la plus faible. Il s'agit ici du nœud C. On explore alors tous les nœuds où arrive un arc partant de C. Ici :

- on ajoute D. Sa distance au départ vaut *distance au départ de C + poids de (C,D)* soit 2 + 1 = 3
- on ajoute E. Sa distance au départ vaut *distance au départ de C + poids de (C,E)* soit 2 + 3 = 5

- on connaît déjà un chemin vers B, mais il n'a pas encore été traité, c'est-à-dire qu'on n'a pas encore trouvé le plus court chemin entre A et B. On calcule donc *distance au départ de C + poids de (C,B)* qui donne 2 + 1 = 3. Or 3 < 4 : il est plus court d'aller de A à B en passant par *(A,C)* et *(C,B)* qu'en prenant *(A,B)*. On met donc à jour la distance au départ de B qui vaut maintenant 3. On note que pour B, le chemin le plus court depuis A vient de C et non de A.
- A a déjà été traité, on a déjà trouvé le plus court chemin de A à A. Il est donc inutile de calculer de nouveau sa distance au départ.

On considère que le nœud C est traité : on a trouvé son plus court chemin depuis A et on a ajouté tous ses nœuds adjacents. On le noircit donc sur la Figure 3, on grise les nœuds D et E qui ont été ajoutés et on met à jour les informations de B (plus courte distance à A et nœud de provenance pour ce chemin).

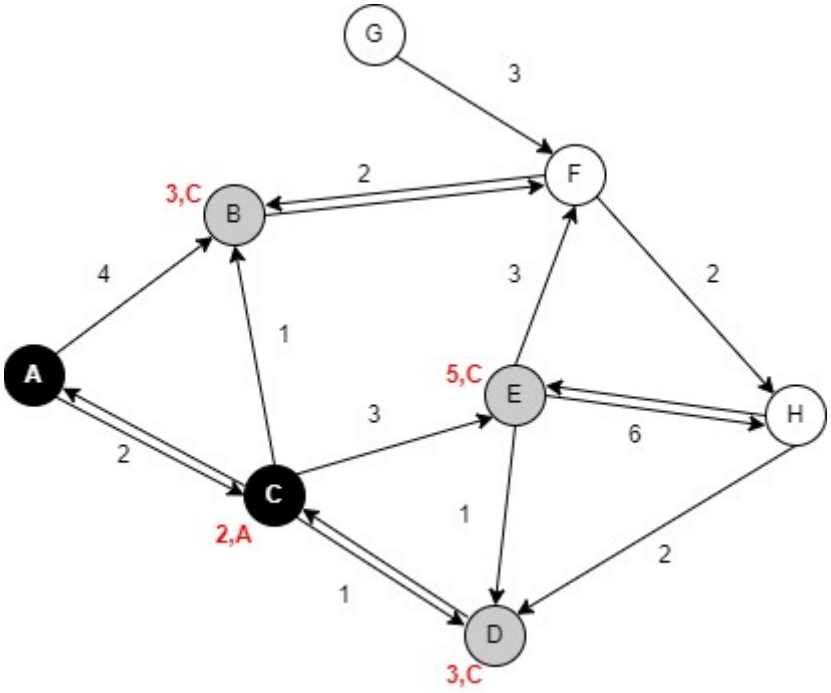

Figure 3 - Deuxième étape du traitement

De nouveau, on traite le nœud pour lequel on connaît un chemin, n'ayant pas été traité et dont la distance la plus courte à A est la plus petite. Ici, B et D ont la même distance. On choisit D :

- C a déjà été traité, il est inutile de calculer sa distance au départ venant de D.
- E et H ont tous les deux un arc menant à D, mais l'inverse est faux. Il n'y a donc aucun nouveau nœud à mettre à jour depuis D.

Le nœud D a été traité et est donc noirci sur la Figure 4 mais rien d'autre n'a changé.

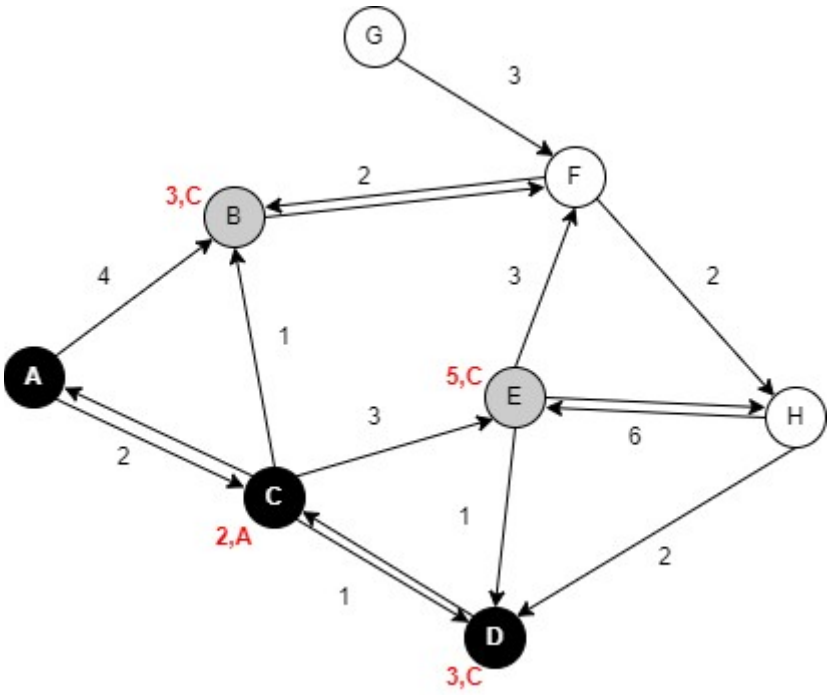

Figure 4 - Troisième étape du traitement

On traite maintenant le nœud B :

- on ajoute F. Sa distance au départ vaut *distance au départ de B + poids de (B,F)* soit 3 + 2 = 5

La Figure 5 représente l'état actuel du traitement.

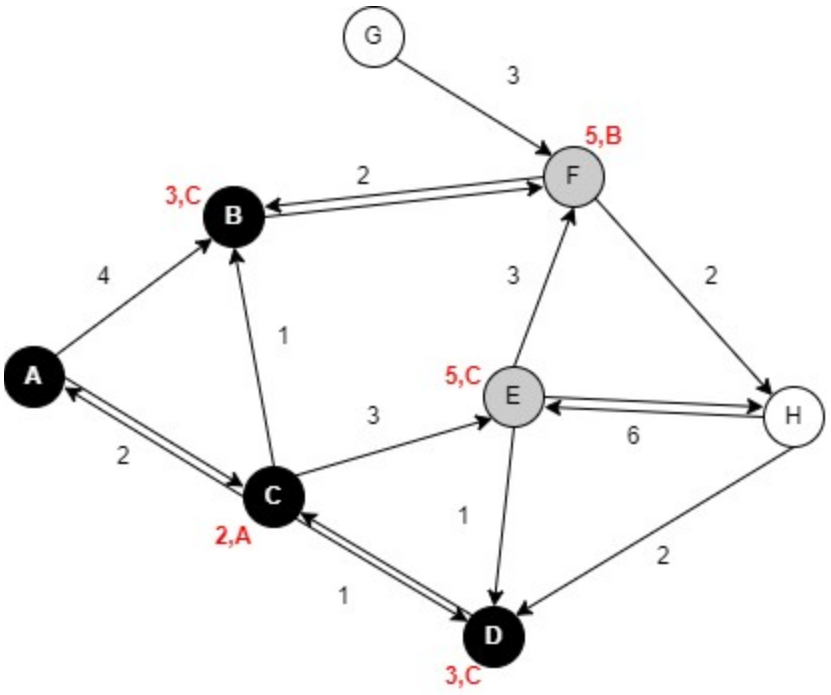

Figure 5 - Quatrième étape du traitement

Les nœuds E et F sont les deux nœuds pour lesquels on connaît un chemin mais non encore traités, c'est-à-dire que ce chemin n'est pas forcément le plus court. Leurs distances au départ respectives sont égales. On choisit de traiter ensuite le nœud E dont partent deux arcs :

- on ajoute H. Sa distance au départ vaut *distance au départ de E + poids de (E,H)* soit 5 + 6 = 11
- F n'a pas encore été traité. On calcule donc sa distance au départ venant de E : *distance au départ de E + poids de (E,F)* soit 5 + 3 = 8. Cette distance est supérieure à la distance de F au départ venant de B (5), donc on ne retient pas pour F le chemin venant de E.

Dans l'état actuel des choses, on connaît un chemin vers le nœud d'arrivée (H), comme on peut le voir sur la Figure 6. Pour autant, il ne s'agit pas encore du nœud non encore traité pour lequel la distance au départ est la plus courte : il reste possible de trouver un chemin vers H plus court passant par F. On poursuit donc le traitement.

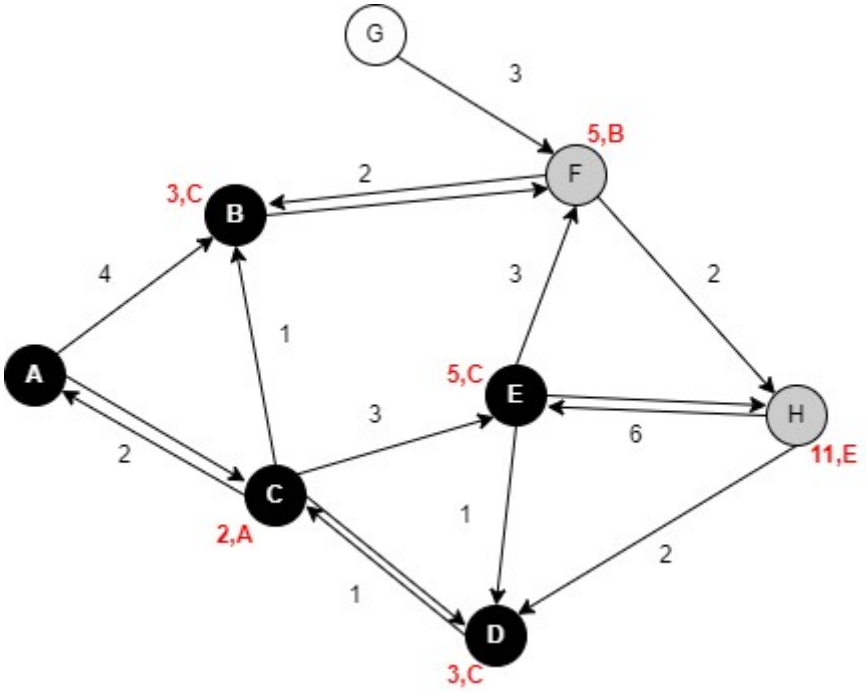

Figure 6 - Cinquième étape du traitement

On traite maintenant le nœud F, dont part un seul arc :

- on calcule la distance de H au départ passant par F : *distance au départ de F + poids de (F,H)* soit 5 + 2 = 7. Cette nouvelle distance est inférieure à la distance enregistrée précédemment (11 pour le chemin venant de E). On met donc à jour pour H la distance venant de F.

Au point représenté par la Figure 7, il ne reste qu'un seul nœud non encore traité pour lequel on connaît un chemin : c'est le nœud d'arrivée H, dont la distance au départ (A) vaut 7. Du même coup, H est le nœud pour lequel on connaît un chemin, non encore traité, dont la distance au départ est la plus petite : suivant la logique de l'algorithme, c'est le prochain nœud à traiter. C'est donc qu'on a trouvé la plus petite distance du départ A à l'arrivée H.

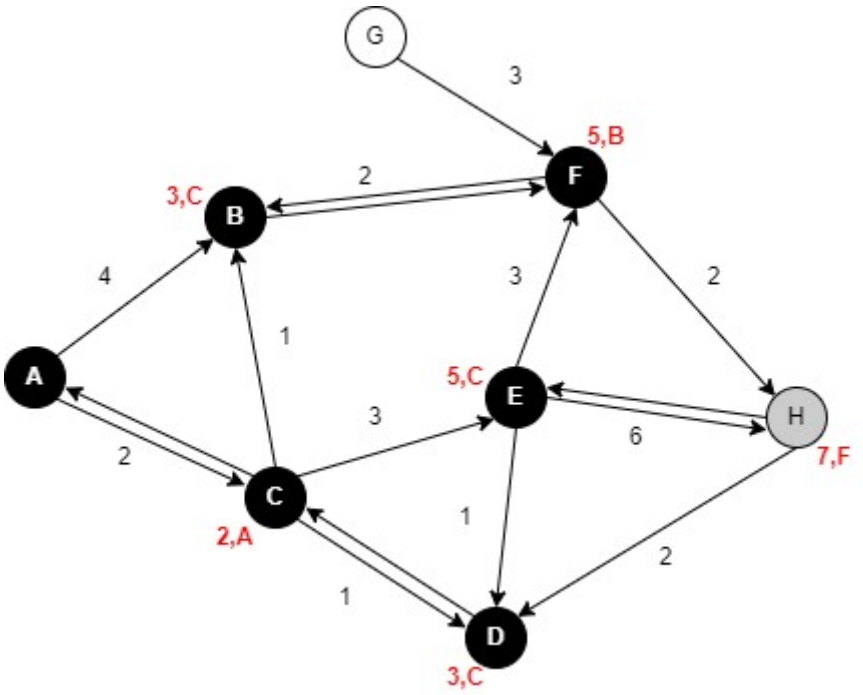

Figure 7 - Sixième étape du traitement

La Figure 8 représente l'état du graphe à l'issue du traitement. Les informations notées en rouge à côté de chaque nœud permettent de restituer le chemin correspondant à la plus courte distance de A à H, c'est-à-dire le plus court chemin de A à H.

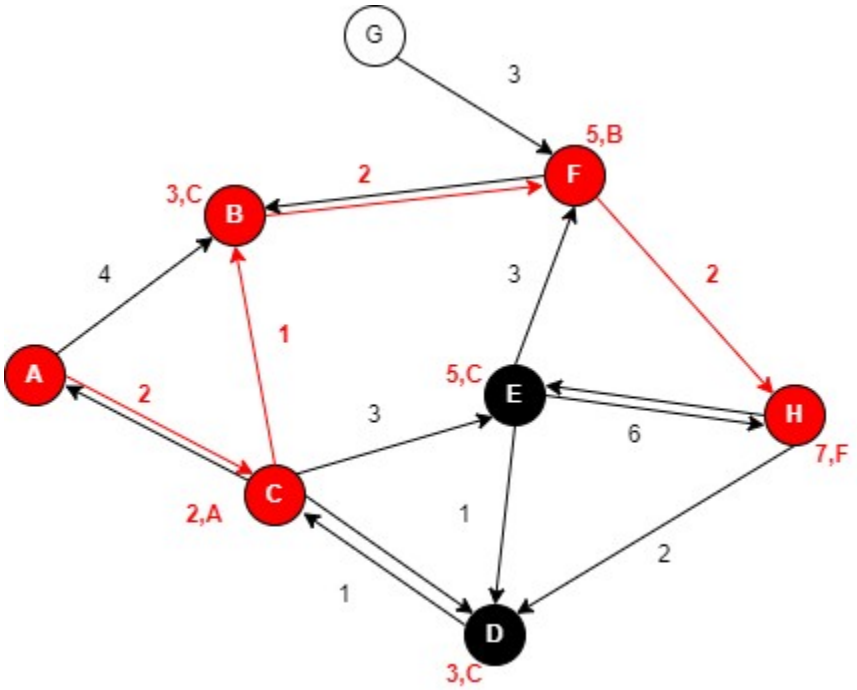

Figure 8 - À l'issue du traitement : restitution du plus court chemin

Il faut pour cela parcourir le graphe depuis le nœud d'arrivée : on vient à H depuis F ; à F depuis B ; à B depuis C ; à C depuis A. Le plus court chemin de A à H est donc A → C → B → F → H.

Implémentation de l'algorithme dans le programme

Structure des données

Pour implémenter l'algorithme de Dijkstra dans notre programme, il nous faut une structure de données qui permette de construire un graphe reliant entre eux les points géographiques identifiés sur un périmètre donné. Pour chaque lien entre deux points, il faut de surcroît connaître leur distance l'un à l'autre, les règles de circulation et la voie qui les relie.

On a donc choisi de créer trois types structurés : un type contenant les informations relatives aux nœuds (`struct noeud`), un type pour les arcs (`struct arc`), ainsi qu'un type pour les voies elles-mêmes (`struct voie`).

Ces données sont stockées dans des listes chaînées de pointeurs, ce qui permet au programme de fonctionner au mieux avec une quantité de données inconnue à l'avance. Il existe :

- une liste doublement chaînée contenant toutes les voies
- une liste doublement chaînée contenant tous les nœuds
- une liste simplement chaînée d'arcs implémentée dans chaque variable de type `struct noeud` et contenant l'ensemble des arcs partant de ce nœud

Nous détaillons ci-dessous le contenu de chaque type structuré ainsi que le mécanisme de construction du graphe au chargement des données. La construction progressive du sous-graphe lors du calcul d'itinéraire sera détaillée plus tard.

Type struct voie

Le type `struct voie` contient quatre attributs :

- `int id_voie` : identifiant numérique de la voie. Il est unique et permet de référencer la voie dans les fichiers de données, à l'image d'une clé étrangère dans une base de données relationnelles. Il sert aussi, pendant le fonctionnement du programme, au choix des voies dont on veut afficher les informations ou dont on veut partir (l'utilisateur saisit le numéro de la voie choisie parmi les voies affichées).

- `char nom_voie[MAX_NOM_VOIE]` : nom de la voie. Sa longueur maximale est définie par la constante `MAX_NOM_VOIE`.
- `struct voie *pred_liste_globale` : pointeur vers la voie précédente dans la liste chaînée de toutes les voies.
- `struct voie *succ_liste_globale` : pointeur vers la voie suivante dans la liste chaînée de toutes les voies.

L'identifiant et le nom de la voie sont lus dans les fichiers de données. La liste chaînée est construite au moment du chargement.

Type struct nœud

Le type `struct noeud` contient neuf attributs :

- `int id_noeud` : identifiant numérique du nœud. Il est unique et permet de référencer le nœud dans les fichiers de données. Il sert aussi, pendant le fonctionnement du programme, aux choix des nœuds dont on veut afficher les informations ou des nœuds de départ et d'arrivée.
- `char nom_noeud[MAX_NOM_NOEUD]` : nom du nœud. Sa longueur maximale est définie par la constante `MAX_NOM_NOEUD`.
- `struct noeud *pred_liste_globale` : pointeur vers le nœud précédent dans la liste chaînée de tous les nœuds.
- `struct noeud *succ_liste_globale` : pointeur vers le nœud suivant dans la liste chaînée de tous les nœuds.
- `struct arc *debut_liste_arc` : pointeur vers le premier élément de la liste chaînée des arcs partant de ce nœud.
- `struct noeud *pred_liste_dijkstra` : pointeur vers le nœud précédent dans la liste chaînée des nœuds à traiter pendant le calcul de l'itinéraire.
- `struct noeud *succ_liste_dijkstra` : pointeur vers le nœud suivant dans la liste chaînée des nœuds à traiter pendant le calcul de l'itinéraire.
- `struct arc *p_provenance` : pointeur vers l'arc dont vient le chemin le plus court depuis le nœud de départ à un moment donné pendant le calcul de l'itinéraire.
- `int distance_au_depart` : distance la plus courte depuis le nœud de départ à un moment donné pendant le calcul de l'itinéraire.

Les cinq premiers attributs reçoivent une valeur au moment du chargement des données et gardent la même valeur jusqu'à l'arrêt du programme ou jusqu'au chargement d'une nouvelle source de données. La liste chaînée globale des nœuds est construite au moment du chargement des nœuds. La liste chaînée des arcs partant d'un nœud est construite lors du chargement des arcs par ajout successif de chaque nouvel arc chargé partant de ce nœud.

Les quatre attributs suivants sont initialisés à `NULL` pour les trois pointeurs ou 0 pour la `distance_au_depart`. Ces attributs sont utilisés au moment du calcul de l'itinéraire, dont le fonctionnement sera détaillé plus tard.

Type struct arc

Le type `struct arc` contient six attributs :

- `int longueur` : longueur en mètres de l'arc
- `int vehicule` : information relative aux véhicules pouvant circuler sur l'arc (0 si le véhicule est indifférent, 1 si la circulation est réservée aux piétons, 2 si la circulation est réservée aux voitures)
- `struct noeud *p_noeud_depart` : pointeur vers le nœud dont part l'arc
- `struct noeud *p_noeud_arrivee` : pointeur vers le nœud où arrive l'arc
- `struct voie *p_voie_arc` : pointeur vers la voie où se situe l'arc
- `struct arc *p_successeur` : pointeur vers l'arc suivant dans la liste chaînée des arcs partant de son nœud de départ

Ces six attributs sont initialisés au moment du chargement des arcs, qui a lieu après le chargement des voies et celui des nœuds. Il restent inchangés jusqu'à l'arrêt du programme.

Schéma récapitulatif de la structure des données

La Figure 9 ci-dessous présente schématiquement la structure des données une fois le chargement des trois fichiers effectué. Il ne s'agit que d'un échantillon de données, destiné à montrer les liens des variables de différents types entre elles. On se concentre sur le croisement de l'avenue d'Italie avec la rue Caillaux, et sur ses liens avec le croisement de l'avenue d'Italie et de la rue de Tagore et le croisement de l'avenue d'Italie avec la rue de la Vistule.

On présente en jaune quatre variables de type `struct noeud`. Elles sont chaînées entre elles grâce aux attributs `pred_liste_globale` et `succ_liste_globale`. Les attributs `id_noeud` et `nom_noeud`

ont les valeurs lues dans les fichiers. Les attributs `pred_liste_dijkstra`, `succ_liste_dijkstra`, `p_provenance` pointent vers `NULL` et `distance_au_depart` vaut 0. L'attribut `debut_liste_arcs` pointe vers le premier arc de la liste d'arcs d'un nœud.

Ici, seule la liste des arcs du nœud 3 – Italie-Caillaux est montrée, en rouge. Elle comprend deux arcs, chaînés grâce à leur attribut `p_successeur`. Les attributs `longueur` et `vehicule` ont les valeurs lues dans le fichier. Les deux arcs ont `p_noeud_depart` pointant vers le nœud 3 – Italie-Caillaux. Pour le premier, `p_noeud_arrivee` pointe vers le nœud 2 – Italie-Tagore ; pour le second, `p_noeud_arrivee` pointe vers le nœud 4 – Italie-Vistule. Les deux arcs correspondent à des portions de l'avenue d'Italie, aussi dans les deux cas `p_voie_arc` pointe vers la voie 1 – Avenue d'Italie.

Cette voie se trouve elle-même chaînée (grâce à ses attributs `pred_liste_globale` et `succ_liste_globale`) à une liste de variables de type `struct voie`, présentée en bleu.

Enfin, en vert, on distingue quatre variables globales destinées à mémoriser le début et la fin de chaque liste chaînée globale : deux pointeurs de type `struct noeud` (`premier_noeud` et `dernier_noeud`) et deux pointeurs de type `struct voie` (`premiere_voie` et `derniere_voie`).

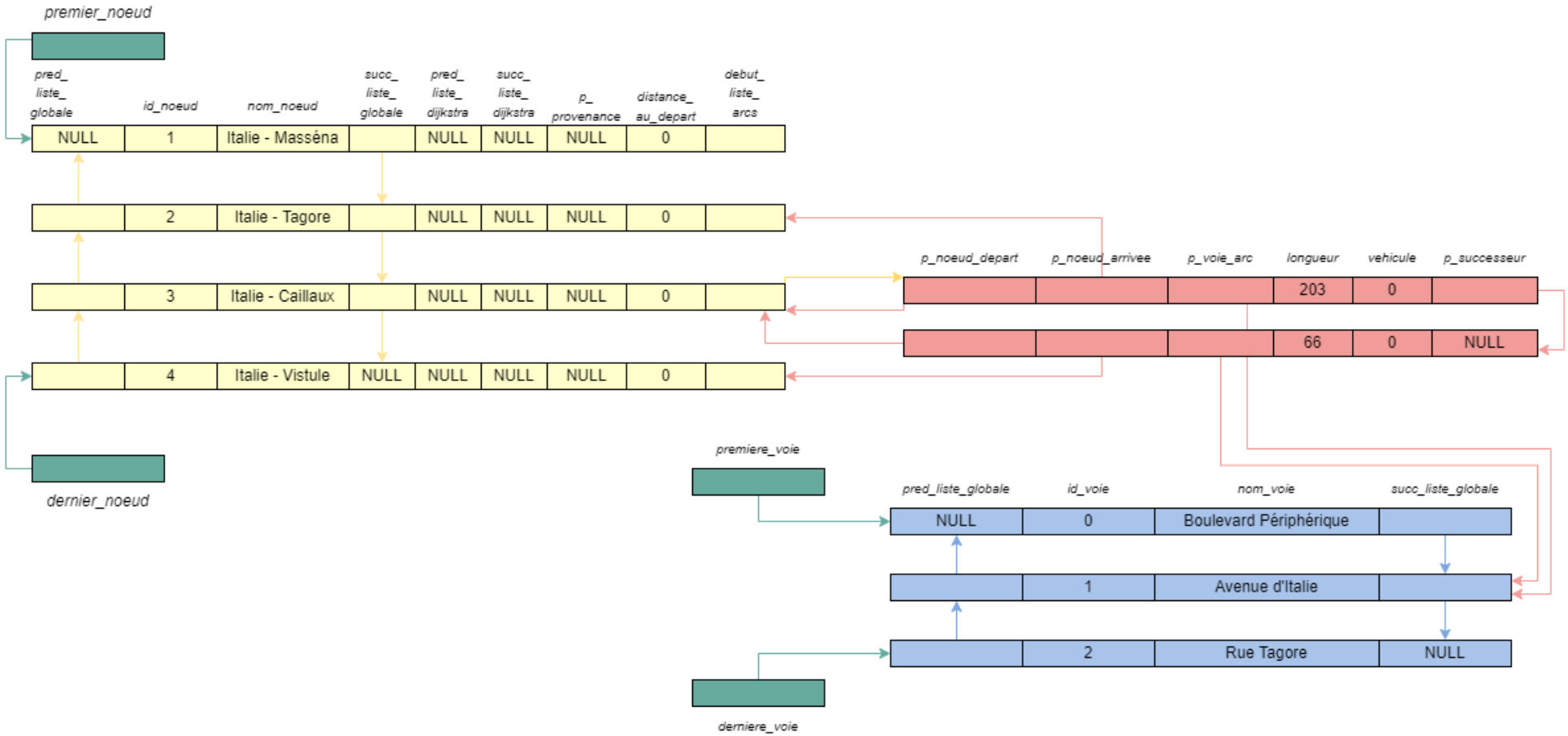

Figure 9: Structure des données au chargement

Structure des fichiers de données

Le programme est conçu pour fonctionner avec trois fichiers de données : le premier contient les informations relatives aux voies du périmètre géographique, le deuxième contient les informations sur les nœuds, le dernier contient les informations relatives aux arcs qui relient les nœuds entre eux. Il s'agit de fichiers texte (.txt).

Ces trois fichiers sont fournis avec le programme, nommés respectivement « voies_13.txt », « noeuds_13.txt », « arcs_13.txt ». Ils doivent être placés dans le même répertoire que l'exécutable du programme. L'utilisateur souhaitant utiliser d'autres données doit placer ses fichiers dans le même répertoire ou, à défaut, indiquer le chemin d'accès lors de la saisie des noms de fichiers personnalisés.

Pour que le programme fonctionne, les fichiers doivent correspondre au format attendu. Si une erreur est détectée au chargement des données, un message d'erreur s'affiche et les autres fonctionnalités de l'application ne sont pas accessibles. Le programme s'arrête et l'utilisateur doit corriger ses fichiers hors programme avant de le relancer.

Fichier des voies

Le fichier fourni avec le programme s'appelle voies_13.txt.

Une ligne du fichier correspond à une voie (type `struct voie`). Chaque ligne contient deux champs obligatoires, séparés par une virgule.

- Un identifiant numérique. Il correspond dans le programme à l'attribut `id_voie` du type `struct voie`. Les identifiants doivent être des nombres entiers consécutifs, rangés dans l'ordre croissant à partir de 0. Chaque identifiant doit être unique.
- Une chaîne de caractères correspondant au type et au nom de la voie. Elle correspond dans le programme à l'attribut `nom_voie` du type `struct voie`. Sa longueur est limitée par une constante définie dans le programme, que nous avons positionnée à 100 (ce qui correspond à 99 caractères, en enlevant le caractère fin de chaîne).

Exemple de ligne du fichier :

```
0,Boulevard Périphérique
```

Fichier des nœuds

Le fichier fourni avec le programme s'appelle noeuds_13.txt.

Une ligne du fichier correspond à un noeud (type `struct noeud`). Chaque ligne contient deux champs obligatoires, séparés par une virgule.

- Un identifiant numérique. Il correspond dans le programme à l'attribut `id_noeud` du type `struct noeud`. Les identifiants doivent être des nombres entiers consécutifs, rangés dans l'ordre croissant à partir de 0. Chaque identifiant doit être unique.
- Une chaîne de caractères correspondant au nom du noeud. Elle correspond dans le programme à l'attribut `nom_noeud` du type `struct noeud`. Sa longueur est limitée par une constante définie dans le programme, que nous avons positionnée à 100 (ce qui correspond à 99 caractères, en enlevant le caractère fin de chaîne).

Exemple de ligne du fichier :

```
48,Italie - Bourgon
```

Fichier des arcs

Le fichier fourni avec le programme s'appelle arcs_13.txt.

Une ligne du fichier correspond à un segment de voies, c'est-à-dire un arc reliant deux nœuds du graphe (type `struct noeud`). Chaque ligne contient six champs obligatoires, séparés par des virgules.

- L'identifiant numérique du sommet de départ de l'arc. Il permet de retrouver la variable de type `struct noeud` correspondante et d'affecter à l'attribut `p_noeud_depart` un pointeur sur cette variable. Ce champ doit être un nombre entier et doit correspondre à une ligne existante dans le fichier des nœuds.
- L'identifiant numérique du nœud d'arrivée de l'arc. Il permet de retrouver la variable de type `struct noeud` correspondante et d'affecter à l'attribut `p_noeud_arrivee` un pointeur sur cette variable. Ce champ doit être un nombre entier et doit correspondre à une ligne existante dans le fichier des nœuds.
- L'identifiant numérique de la voie à laquelle appartient l'arc. Il permet de retrouver la variable de type `struct voie` correspondante et d'affecter à l'attribut `p_voie_arc` un pointeur sur cette variable. Ce champ doit être un nombre entier et doit correspondre à une ligne existante dans le fichier des voies.
- La longueur en mètres de l'arc. Ce champ doit être un nombre entier strictement positif.
- Un champ indiquant si l'arc est à sens unique ou non. Ce champ peut prendre deux valeurs :
  - la valeur 1 signifie que les règles de circulation (droit des piétons ou des voitures à circuler sur l'arc) ne sont pas les mêmes dans les deux sens

    - la valeur 0 signifie au contraire que le sens dans lequel on parcourt l'arc est indifférent en ce qui concerne les règles de circulation

- Un champ déterminant les règles de circulation sur l'arc dans ce sens précis (du nœud de départ au nœud d'arrivée). Ce champ peut prendre trois valeurs :
    - la valeur 0 signifie que l'arc peut être emprunté aussi bien par les piétons que par les voitures
    - la valeur 1 signifie que l'arc ne peut être emprunté que par les piétons
    - la valeur 2 signifie que l'arc ne peut être emprunté que par les voitures

Exemple de ligne du fichier :

139,140,206,46,1,0

Cette ligne signifie qu'il y a un arc menant du sommet n°139 (ALBERT – TERRES AU CURE) au sommet n°140 (TERRES AU CURE – VILLA NIEUPORT) par la voie n°206 (RUE DES TERRES AU CURE).

Cet arc mesure 46 mètres. La sixième colonne vaut 0 donc l'arc peut être parcouru par des voitures comme par des piétons. La cinquième colonne vaut 1 donc le segment de rue est à sens unique : dans le sens inverse, seuls les piétons peuvent l'emprunter.

Duplication des arcs

Comme on construit un graphe orienté, chaque arc est à sens unique : il mène de son sommet de départ à son sommet d'arrivée. Ainsi, chaque segment de voie entre deux points doit correspondre à deux arcs dans le programme, partant du principe qu'on peut le parcourir dans les deux sens avec au moins un moyen de transport. Pour éviter les redondances dans le fichier, nous avons choisi de ne saisir qu'une ligne par segment de voie. Au chargement, le programme duplique les arcs lus dans le fichier.

Après chaque création d'arc *(A,B)*, un nouvel arc *(B,A)* est créé. Ses attributs `longueur` et `p_voie_arc` sont les mêmes que ceux de l'arc lu. Son attribut `p_sommet_depart` prend la valeur de `p_noeud_arrivee` du premier arc ; `p_noeud_arrivee` prend la valeur de `p_noeud_depart` du premier arc. *(B,A)* est inséré dans la liste des arcs du nœud B. Enfin, la valeur de l'attribut `vehicule` est déterminé selon la méthode suivante :

- si `vehicule` de *(A,B)* vaut 2, alors la circulation sur le segment est réservée aux voitures. Partant du principe que les segments réservés aux voitures le sont dans les deux sens, `vehicule` de *(B,A)* prend 2.
- sinon si `sens_unique` (cinquième champ de la ligne lue) vaut 0, alors le segment de voie n'est pas à sens unique et `vehicule` de *(B,A)* prend la même valeur que `vehicule` de *(A,B)*.

- sinon `sens_unique` vaut 1, alors `vehicule` de *(B,A)* prend la seule autre valeur possible parmi 0 et 1 que `vehicule` de *(A,B)*

La création du deuxième arc est alors achevée.

Calcul de l'itinéraire

À partir de cette structure de données, on peut mettre en place l'algorithme de Dijkstra à proprement parler pour calculer l'itinéraire le plus court entre deux nœuds.

Dans le programme, le calcul de l'itinéraire est assuré par deux sous-programmes :

- `int calcul_itineraire(struct noeud *noeud_depart, struct noeud *noeud_arrivee, int moyen_transport)`
- `void inserer_noeud(struct noeud *a_inserer, struct noeud *debut_liste)`

Fonctionnement de calcul_itineraire()

La fonction `calcul_itineraire()` consiste principalement en une boucle qui parcourt la liste des nœuds à traiter. Cette liste, distincte de la liste globale de tous les nœuds, est implémentée sous la forme d'une liste de pointeurs chaînée grâce aux attributs `*pred_liste_dijkstra` et `*succ_liste_dijkstra` du type `struct noeud`.

Elle est triée dans l'ordre croissant de la `distance_au_depart` des nœuds ajoutés. C'est la procédure `inserer_noeud()` qui, ajoutant un nœud à cette liste ou modifiant sa position lorsqu'on lui a trouvé une `distance_au_depart` plus courte, garantit que la liste reste triée.

calcul_itineraire() parcourt la liste à l'aide du curseur *noeud_actuel. Une fois qu'un nœud a été traité (on a ajouté ou mis à jour tous ses nœuds adjacents dans la liste), il ne sera plus traité car on a déjà sa `distance_au_depart` la plus courte.

1. On fait appel à `calcul_itineraire()` en lui passant en paramètre un pointeur vers le nœud de départ, un pointeur vers le nœud d'arrivée, et le moyen de transport désiré (1 pour piéton, 2 pour voiture).
2. On positionne le curseur sur *noeud_depart, le premier et seul (pour l'instant) nœud de la liste.
3. Tant qu'il reste des nœuds dans la liste ET que le nœud non traité dont la distance est la plus courte n'est pas le nœud d'arrivée, on itère la boucle

a) Positionnement sur le premier arc partant du nœud et exécution des instructions suivantes pour chaque arc de la liste du nœud seulement si les règles de circulation sur l'arc sont compatibles avec le moyen de transport choisi :

- On récupère dans *noeud_adjacent un pointeur vers le nœud d'arrivée de l'arc
- On calcule la distance au départ du nœud adjacent venant du nœud actuel
- Si le nœud adjacent n'est pas encore ajouté à la liste ou que la nouvelle distance au départ est inférieure à l'ancienne :

  i. On affecte à `distance_au_depart` de noeud_adjacent la distance nouvellement calculée

  ii. On affecte à p_provenance de noeud_adjacent un pointeur vers l'arc courant, dont vient le chemin le plus court

  iii. On appelle `inserer_noeud()` pour l'insérer dans la liste ou le déplacer

b) On passe au nœud suivant dans la liste de traitement

4. À la fin de la boucle, si le nœud d'arrivée a été atteint, on retourne 1 pour indiquer que le calcul est réussi, sinon on retourne 0.

Exemple de fonctionnement : de GANDON – TAGORE à CHOISY – LUCOT en voiture

Nous présentons maintenant un exemple concret de calcul d'itinéraire entre deux points :

- le croisement de la rue Gandon et de la rue de Tagore (GANDON – TAGORE)
- le croisement de l'avenue de Choisy avec la rue Philibert Lucot (CHOISY – LUCOT)

Cet itinéraire doit se faire en voiture, pour bien faire apparaître le traitement des arcs à sens unique.

Ces deux points se présentent comme suit sur un plan de cette zone du 13 arrondissement :

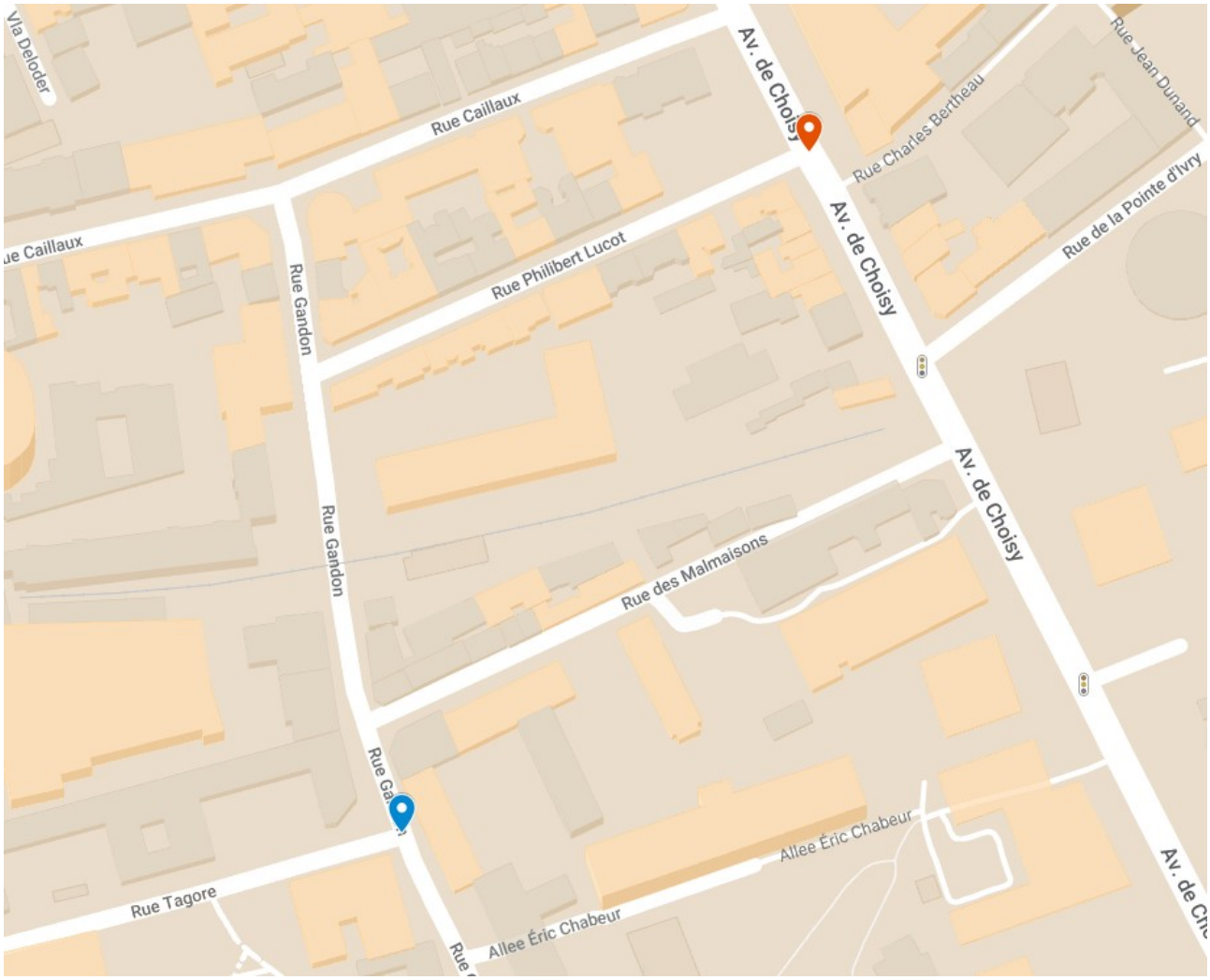

Illustration 1 - De GANDON - TAGORE à CHOISY - LUCOT

1. Depuis `menu_itineraire()`, on appelle `calcul_itineraire()` en lui passant comme paramètres un pointeur vers le nœud GANDON – TAGORE, un pointeur vers le nom CHOISY – LUCOT et la valeur 2 (constante `VOITURE`) comme troisième paramètre.

2. On positionne `noeud_actuel` sur `noeud_depart` (GANDON – TAGORE). C'est pour l'instant le seul nœud de la liste à traiter.

3. On parcourt la liste des arcs de GANDON – TAGORE (`distance_au_depart` = 0) :

   a) Le premier arc mène vers GANDON – MALMAISONS, `vehicule` vaut 0 (`INDIFFERENT`). On calcule la distance au départ du nœud adjacent venant de GANDON – TAGORE : 0 + 37 = 37. GANDON – MALMAISONS n'est pas le nœud de départ et sa `distance_au_depart` était de 0, donc c'est un nœud à ajouter dans la liste de traitement. On appelle `inserer_noeud()` pour l'insérer dans la liste et on lui ajoute un pointeur vers l'arc courant.

b) Le deuxième arc mène vers GANDON – CHABEUR, `vehicule` vaut 0 (`INDIFFERENT`). On calcule la distance au départ de ce nœud en venant de GANDON – TAGORE : 0 + 41 = 41. GANDON – CHABEUR n'est pas le nœud de départ et sa `distance_au_depart` était de 0, donc c'est un nœud à ajouter dans la liste de traitement. On appelle `inserer_noeud()` pour l'insérer dans la liste et on lui ajoute un pointeur vers l'arc courant.

c) Le troisième arc mène vers ITALIE – TAGORE mais pour cet arc `vehicule` vaut 1 (`PIETON`) donc le nœud n'est pas traité.

d) Fin des arcs.

4. Le nœud suivant dans la liste de traitement est GANDON – MALMAISONS (`distance_au_depart` = 37). On parcourt la liste de ses arcs :

   a) Le premier arc mène vers GANDON – LUCOT, `vehicule` vaut 0 (`INDIFFERENT`). On calcule la distance au départ de ce nœud en venant de GANDON – MALMAISONS : 37 + 103 = 140. GANDON – CHABEUR n'est pas le nœud de départ et sa `distance_au_depart` était de 0, donc c'est un nouveau nœud à ajouter dans la liste de traitement. On appelle `inserer_noeud()` pour l'insérer dans la liste.

   b) Le deuxième arc mène vers GANDON – TAGORE, `vehicule` vaut 0 (`INDIFFERENT`). On calcule la distance au départ de ce nœud en venant de GANDON – MALMAISONS : 37 + 37 = 74. GANDON – TAGORE est le nœud de départ donc on ne l'ajoute pas de nouveau à la liste de traitement.

   c) Le troisième arc mène vers CHOISY – MALMAISONS mais pour cet arc `vehicule` vaut 1 (`PIETON`) donc le nœud n'est pas traité.

   d) Fin des arcs.

5. Le nœud suivant dans la liste de traitement est GANDON – CHABEUR (`distance_au_depart` = 41). On parcourt la liste de ses arcs :

   a) Vers GANDON – TAGORE : `vehicule` = `INDIFFERENT`, on calcule une nouvelle distance (41 + 41 = 82). GANDON – TAGORE est le nœud de départ donc on ne l'ajoute pas de nouveau à la liste de traitement.

   b) Vers GANDON – CHAGALL : `vehicule` = `INDIFFERENT`, on calcule la distance (41 + 61 = 102), on l'insère dans la liste et on lui ajoute un pointeur vers l'arc courant car c'est un nouveau nœud.

   c) Vers CHOISY – CHABEUR : `vehicule` = `PIETON` donc on ne traite pas le nœud.

   d) Fin des arcs.

6. Nœud suivant : GANDON – CHAGALL (`distance_au_depart` = 102)

a) Vers GANDON – CHABEUR : `vehicule` = `PIETON`, on ne traite pas le nœud.

b) Vers GANDON – MASSENA : `vehicule` = `INDIFFERENT`, on calcule la distance (102 + 143 = 245), on l'insère dans la liste et on ajoute un pointeur vers l'arc courant car c'est un nouveau nœud.

c) Vers ITALIE – CHAGALL : `vehicule` = `PIETON`, on ne traite pas le nœud.

d) Fin des arcs.

7. Nœud suivant : GANDON – LUCOT (`distance_au_depart` = 140)

   a) Vers CAILLAUX – GANDON : `vehicule` = `INDIFFERENT`, on calcule la distance (140 + 57 = 197), on l'insère dans la liste et on lui ajoute un pointeur vers l'arc courant car c'est un nouveau nœud.

   b) Vers GANDON – MALMAISONS : `vehicule` = `PIETON`, on ne traite pas le nœud.

   c) Vers CHOISY – LUCOT : `vehicule` = `INDIFFERENT`, on calcule la distance (140 + 161 = 301), insertion dans la liste et ajout d'un pointeur vers l'arc car c'est un nouveau nœud.

8. À ce point, on a trouvé un itinéraire menant du nœud de départ au nœud d'arrivée (CHOISY – LUCOT) en 301m, mais on ne sait pas encore si c'est bien l'itinéraire le plus court. On poursuit donc le traitement en passant au nœud suivant dans la liste, CAILLAUX – GANDON (`distance_au_depart` = 197)

   a) Vers GANDON – LUCOT : `vehicule` = `PIETON`, on ne traite pas le nœud.

   b) Vers ITALIE – CAILLAUX : `vehicule` = `INDIFFERENT`, on calcule la distance (197 + 150 = 347), insertion dans la liste et ajout d'un pointeur vers l'arc car c'est un nouveau nœud.

   c) Vers CHOISY – CAILLAUX : `vehicule` = `PIETON`, on ne traite pas le nœud.

9. Nœud suivant : GANDON – MASSENA (`distance_au_depart` = 245)

   a) Vers MASSENA – WIDAL : `vehicule` = `PIETON`, on ne traite pas le nœud.

   b) Vers BOLLEE – ENFERT : `vehicule` = `INDIFFERENT`, on calcule la distance (245 + 149 = 394), insertion dans la liste et ajout d'un pointeur vers l'arc car c'est un nouveau nœud.

   c) Vers MASSENA – CONVENTIONNEL CHIAPPE : `vehicule` = `INDIFFERENT`, on calcule la distance (245 + 87 = 332), insertion dans la liste et ajout d'un pointeur vers l'arc car c'est un nouveau nœud.

   d) Vers GANDON – CHAGALL : `vehicule` = `PIETON`, on ne traite pas le nœud.

e) Vers CHOISY – MASSENA : `vehicule` = `INDIFFERENT`, on calcule la distance (245 + 211 = 456), insertion dans la liste et ajout d'un pointeur vers l'arc car c'est un nouveau nœud.

f) Vers ITALIE – MASSENA : `vehicule` = `INDIFFERENT`, on calcule la distance (245 + 190 = 435), insertion dans la liste et ajout d'un pointeur vers l'arc car c'est un nouveau nœud.

10. Nœud suivant : CHOISY – LUCOT (`distance_au_depart` = 301). Le premier nœud non traité de la liste est le nœud d'arrivée. Sortie de la boucle.
11. On vérifie que noeud_actuel pointe sur le même nœud que noeud_arrivee ce qui est le cas. On a donc trouvé le chemin le plus court pour une voiture entre GANDON – TAGORE et CHOISY – LUCOT. On appelle afficher_itineraire() pour l'afficher à l'utilisateur.
12. Fin du calcul de l'itinéraire.

Difficultés et arbitrages

Plusieurs points nous ont posé problème pendant la conception et la réalisation du programme. Pour chacun d'entre eux, nous expliquons sommairement la difficulté rencontrée et le choix fonctionnel ou technique qui en a résulté.

Modalités du choix des points de départ et d'arrivée

Au vu du volume important de données fournies avec le programme (près de cinq cents nœuds), il fallait réfléchir à la manière la plus ergonomique possible de sélectionner des points de départ et d'arrivée.

L'affichage de la liste globale des points à choisir aurait risqué de perdre l'utilisateur. Nous avons donc choisi de demander à l'utilisateur de choisir une voie, de lui afficher les nœuds situés sur cette voie, puis de lui demander de choisir un nœud parmi ceux affichés.

Cette solution a posé une petite difficulté technique : il n'existe pas dans notre structure des données de moyen simple de lier les voies aux nœuds ou inversement. Ce sont les arcs qui opèrent cette jonction, car ils pointent vers une voie et vers leurs nœuds de départ et d'arrivée. Pour afficher la liste des nœuds situés sur une voie, on a donc mis en place le mécanisme suivant :

- la fonction `struct noeud *choisir_noeud_voie(struct voie *p_voie)` parcourt la liste globale des nœuds ; à chaque nœud, elle appelle la fonction `int verif_noeud_sur_voie(struct noeud *p_noeud, struct voie *p_voie)`
- cette deuxième fonction parcourt la liste des arêtes du nœud et vérifie si l'un d'entre elles au moins est située sur la voie demandée ; elle renvoie 1 le cas échéant, sinon 0
- `choisir_noeud_voie()` affiche les sommets pour lesquels elle a récupéré 1, sinon elle passe au suivant sans action

Toutefois, l'affichage de la liste totale des voies présente aussi un problème : il y a également des centaines de voies. Nous avons donc choisi de présenter les voies par ordre alphabétique pour faciliter la compréhension de la liste, mais aussi d'inclure une fonctionnalité de recherche de voies par mot-clé : l'utilisateur saisit un mot-clé, le programme affiche toutes les voies qui contiennent la chaîne de caractères saisie et demande à l'utilisateur d'en choisir une avant d'afficher les sommets situés sur cette voie.

Pour que la saisie du mot-clé soit la plus intuitive possible, nous avons décidé de convertir le mot-clé en majuscule et le nom des voies cherchées également : la saisie n'est pas sensible à la casse. Nous avons également décidé de supprimer les accents dans le nom des voies, que nous souhaitions initialement conserver.

Nommage des nœuds

Dans la mesure où un nœud peut se situer à l'intersection d'un nombre important de voies, il paraissait difficile de faire apparaître le nom de chacune d'entre elles dans le nom du nœud. Aussi, en général, le nom d'une intersection est déterminé par deux des voies qui s'y croisent : celles qui nous paraissent les plus importantes.

Par exemple, le croisement de la rue Baudricourt, de l'avenue de Choisy et de la rue de la Vistule se nomme simplement « CHOISY – VISTULE ». Cependant, il est bien lié au nœud suivant sur la rue de la Vistule et apparaît bien dans la liste des points situés rue de la Vistule.

Gestion dynamique de la quantité de données

Comme nous offrons à l'utilisateur la possibilité de charger son propre jeu de données, nous avons préféré utiliser des listes chaînées de pointeurs plutôt que des tableaux pour éviter le sur- ou le sous-dimensionnement de l'espace mémoire alloué au programme.

Le choix de ne pas construire une liste globale des arcs s'explique par la nécessité de parcourir de manière simple les arcs partant de chaque nœud lors du calcul de l'itinéraire. Il nous a paru plus pertinent de construire une liste d'arcs par nœud.

Dans la mesure où l'identifiant des nœuds et des voies n'est pas lié à une case dans un tableau, on aurait pu ne pas forcer l'utilisateur à choisir des identifiants consécutifs partant de 0 pour chaque nœud et chaque voie. Le choix de contrôler tout de même cette donnée correspond à un double avantage : fonctionnel (l'affichage est plus cohérent quand les numéros des voies se suivent) et technique (il est plus facile de contrôler l'absence de doublon dans les identifiants au chargement lorsqu'on attend des identifiants consécutifs).

Chevauchements de rues et terre-pleins centraux

Comme évoqué en première partie dans la section Croisements spécifiques à un moyen de transport, certains croisements se comportent différemment selon qu'on est en voiture ou à pied, ou selon le côté de la voie sur lequel on se trouve (pour une voiture).

Reprenons l'exemple de la rue Bréal et du Boulevard Masséna, représenté sur l'Illustration 2.

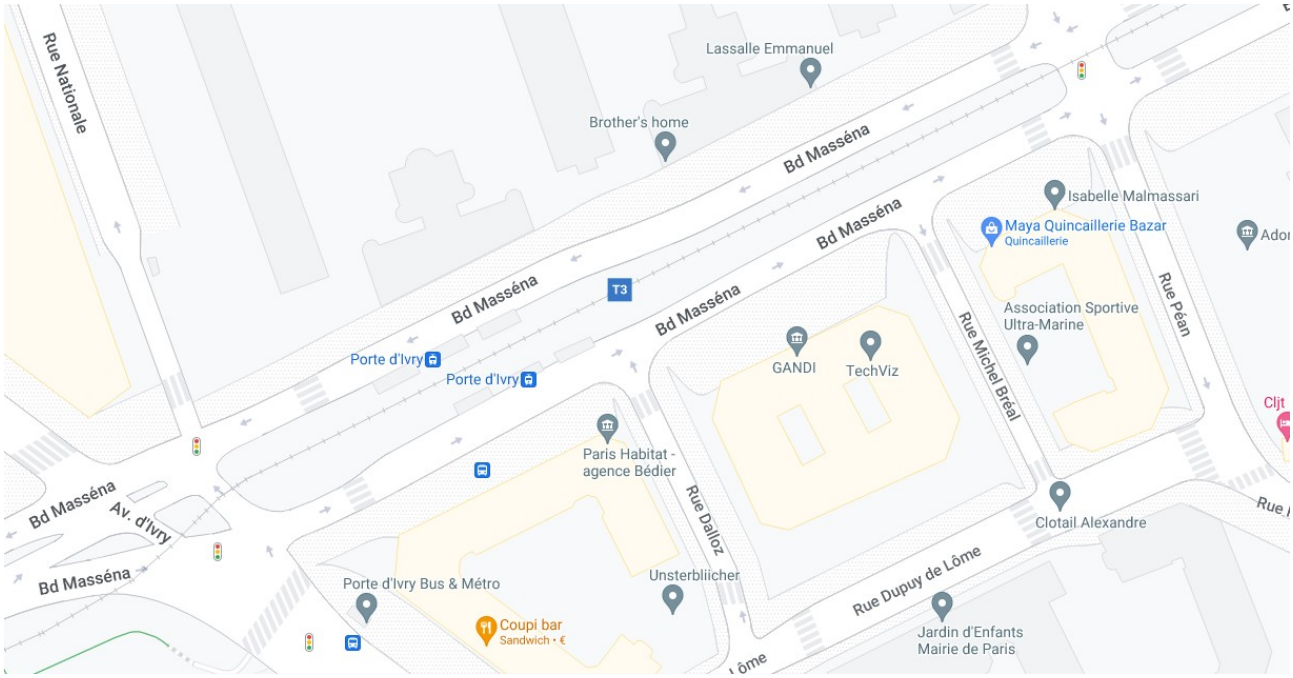

Illustration 2 - De MASSENA - RENTIERS à IVRY - MASSENA

Lorsqu'on arrive de l'ouest en voiture sur le boulevard Masséna, on peut s'engager dans la rue Bréal, mais pas lorsqu'on arrive de l'est. Nous avons donc créé un premier arc, à double sens et dont la circulation est ouverte aux piétons comme aux voitures, qui va de MASSENA – RENTIERS à IVRY – MASSENA. Dans l'autre sens, nous avons créé trois arcs à sens unique :

- de IVRY – MASSENA à MASSENA – DALLOZ

- de MASSENA – DALLOZ à MASSENA – BREAL
- de MASSENA – BREAL à MASSENA – RENTIERS

Ainsi, d'ouest en est, voitures comme piétons peuvent s'engager sur le boulevard Masséna depuis la rue Dalloz et dans la rue Bréal depuis le boulevard Masséna. Dans le sens inverse, seuls les piétons ont cette possibilité : les voitures ne peuvent aller que directement de MASSENA – RENTIERS à IVRY – MASSENA.

Affichage de l'itinéraire voie par voie

Nous avons rencontré deux difficultés pour l'affichage de l'itinéraire.

Il est facile de parcourir en sens inverse l'itinéraire calculé, puisque chaque nœud a un attribut qui pointe vers l'arc dont il provient, qui lui-même pointe vers son nœud de départ. Mais il est plus difficile de le parcourir à l'endroit : un nœud ne pointe pas vers le nœud suivant dans l'itinéraire.

Plutôt que de créer de nouveaux attributs dans le type `struct noeud` pour stocker cette information, nous avons construit `afficher_itineraire(struct arc *a)` de manière récursive. La procédure est appelée pour un arc donné du parcours (en commençant par le dernier arc, celui qui mène au nœud d'arrivée) et s'appelle elle-même autant de fois qu'il y a d'arcs dans le parcours, de manière à revenir au nœud de départ : la récursion terminale est alors atteinte. On affiche ensuite « dans l'ordre » les informations de chaque arc : point de départ, point d'arrivée, voie et distance.

La deuxième difficulté tient au regroupement des arcs appartenant à la même voie, pour éviter d'afficher autant de lignes qu'il y a de points sur une voie. Pour pallier ce problème, nous utilisons une variable statique `struct voie *v` permettant de garder en mémoire la voie de l'arc traité précédemment (donc de l'arc suivant dans l'itinéraire, comme cette opération se fait avant l'appel récursif) pour savoir si l'arc actuel est le dernier de la voie (information gardée dans une variable locale `int fin_voie`).

Dans la deuxième partie de la procédure (après l'appel récursif), qui s'exécute « dans l'ordre », on utilise d'autres variables statiques :

- `struct noeud *n_depart_voie` garde en mémoire le premier nœud d'une voie pour l'afficher
- `int longueur_voie` additionne les longueurs des arcs d'une même voie

Si l'arc traité est le dernier arc de la voie (`fin_voie` vaut 1), on affiche les informations de la voie parcourue.