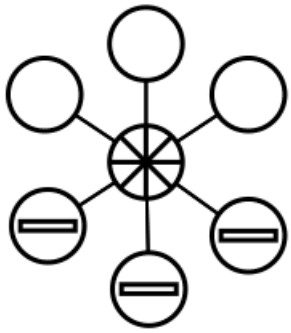
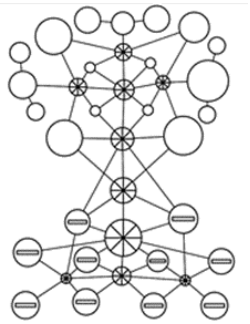
ICC MT & EL **Date limite de téléversement : dimanche 31 mai 2020 23h59**



Travail théorique *individuel*

Contexte



1. Introduction:

Nature du travail demandé: travail théorique d'analyse d'un problème avec formulation de proposition de structuration des données (du cours C++) et expression de la solution de sous-problèmes en pseudocode.

Remarque importante : ce travail se concentre sur la composante **Modèle** du problème à résoudre. Il n'a aucun élément de type « interface graphique » et n'exige aucun codage en C++.

Lien avec le projet : Le travail demandé s'appuie sur la donnée du projet afin de pouvoir tirer parti d'éléments introduits en cours (représentation de graphe, Algorithme de Dijkstra) qui ont été mis en œuvre dans vos deux premiers rendus.

Indépendance vis-à-vis du projet : il n'y a pas de dépendance vis-à-vis de votre code de ces précédents rendus. Vous pouvez faire un autre choix de représentation de graphe par exemple. De même ce travail est complètement indépendant du rendu final du projet. Le projet continue tel qu'il a été décrit (y compris le rapport final).

Travail individuel : vous pouvez vous appuyer sur les choix effectués pour le projet mais *ce travail est un travail individuel*. Les copies des membres de chaque groupe seront comparées par l'enseignant. Le détecteur de plagiat *turnitin* sera utilisé.

Sujet : le point de départ est une ville respectant les indications de la donnée du projet **Archipelago**. En résumé, on s'intéresse au problème de simuler la circulation des pendulaires, seconde par seconde, des quartiers de logement vers les quartiers de production. Certaines contraintes supplémentaires sont précisées au fil des questions.

Ce travail écrit individuel permet d'obtenir **40 points** sur un total de 100 points pour le cours complet.

2. Description de l'outil désiré :

Même si ce travail ne donne pas lieu a du codage, c'est une bonne chose d'identifier comment les différents éléments s'organisent entre eux et comment on utiliserait le programme qui en résulterait. Comme le montre la Fig1, l'analyse à effectuer porte sur des structures de données et algorithmes qui seraient dans un module appelé ici **simulation** qui utilise le **Modèle**. Un usage type du programme résultant serait le suivant :

- Appel de l'exécutable en donnant un nom de fichier de ville
- Pour chaque seconde entre 1 et un nombre max prédéfini
    - Affiche le nombre d'habitants pendulaires sur chaque direction de chaque connexion

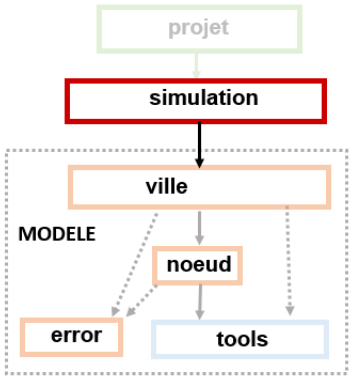


Fig1 : Architecture

Rappel : la capacité **C** d'une connexion est le minimum de la capacité des deux quartiers qu'elle relie. La section 4 précise la modélisation d'une connexion.

Etat initial : au début de la simulation, tous les habitants pendulaires sont dans leur quartier de Logement et tous les autres quartiers sont vides. De même, les connexions sont aussi vides.

Exemple :

Soit la ville illustrée par la Fig2. Par souci de clarté nous choisissons de nommer les nœuds avec une lettre. De plus, chaque lien étant bidirectionnel, une connexion est composée de deux directions et est indiquée selon la syntaxe « nœud de départ » - « nœud d'arrivée ». Ayant fourni le fichier de cette ville au programme, un tel programme produirait ce type d'affichage :

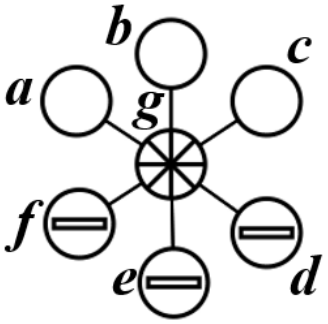


Fig2 : exemple de ville

Temps(s), a-g , g-a , b-g , g-b , c-g , g-c , d-g , g-d , e-g , g-e , f-g , g-f												
0	0	0	0	0	0	0	0	0	0	0	0	0
1	25	0	25	0	25	0	0	0	0	0	0	0
2	50	0	50	0	50	0	0	0	0	0	0	0
….												
70	0	0	0	0	0	0	0	10	0	15	0	13
….												
max	0	0	0	0	0	0	0	0	0	0	0	0

Les valeurs indiquées sont les nombres de pendulaires sur chaque direction d'une connexion à chaque seconde. La valeur *max* correspond au nombre de secondes pour l'heure prédéfinie de fin de la simulation.

Remarque sur le niveau de détail du travail demandé :

- le travail demandé est décomposé en plusieurs étapes dont trois demandent l'écriture d'un algorithme et une description de la structuration des données qui représente un pendulaire.

- Il est tout à fait autorisé et même recommandé d'ajouter des informations aux structures de données utilisées pour le graphe selon les besoins.

- On autorise l'usage des mots clef du C++ pour nommer les structures de données (ex : vector) mais on rappelle qu'il s'agit d'écrire du pseudocode et non du code.

- La dernière étape est la mise à jour des positions des pendulaires. Pour cette étape il sera suffisant de montrer comment on peut savoir le nombre de pendulaires dans chaque direction de chaque connexion à chaque seconde de la simulation ; on ne demande pas le pseudocode de l'affichage ci-dessus.

3. Hypothèses simplificatrices :

- Le nombre d'habitants qui sortent de leur quartier de Logement est de **50% de sa capacité**
    - On les appelle les **pendulaire**s et on s'occupe seulement d'eux en simulant l'évolution de leur position depuis leur nœud de départ jusqu'à leur nœud de destination

- Chaque mise à jour de la simulation correspond à **une seule** seconde de temps.
    - La mise à jour de la localisation de chaque pendulaire est **asynchrone** ; c'est-à-dire qu'on met à jour sa position seulement en examinant l'état présent, c'est-à-dire *partiellement* mis à jour, de son chemin vers sa destination. Cela simplifie énormément la mise en œuvre algorithmique car il suffit d'une boucle qui traite indépendamment chaque pendulaire, un par un : chaque pendulaire examine les informations disponibles et met à jour sa position immédiatement (détails en section 5).

4. Modélisation d'une connexion = deux directions:

Nous imposons l'approche suivante pour représenter la capacité **C** d'une connexion de manière compatible avec la vitesse V de déplacement et la distance D à parcourir :

- Tout d'abord on pose que la capacité **C** est le **nombre maximum de pendulaires** qu'elle peut contenir. On a donc dans chaque **direction** un maximum de **M=C/2** pendulaires (division entière).

- La distance **D** est la distance entre les centres des deux nœuds qu'elle relie

- La vitesse **V** peut prendre une des 2 valeurs définies dans la donnée du projet. La division entière (**D/V**) détermine le nombre de secondes pour parcourir la distance **D**.

Ce qui suit concerne chaque *direction d'une connexion*, simplement appelée une **direction**. En conséquence une **direction** est divisée en un nombre entier **S** de **segments** en prenant la partie entière de **D/V** (Fig 3). De cette manière un pendulaire se déplace de **un** segment en direction du nœud destination à chaque mise à jour (correspondant à un temps de **une** seconde). Après **S** mises à jour le pendulaire sera dans le dernier segment qui correspond à une distance (D/V)*V, c'est-à-dire environ **D**.

Cependant, la valeur de **S** n'a aucune raison d'être égale au maximum **M** d'une **direction**. C'est pourquoi on pose qu'il y a un nombre entier **P** de *passages parallèles* en prenant l'*entier supérieur ou égal* à **M/S**. Le produit S*(M/S) donne un maximum M´ légèrement supérieure à M mais cela n'est pas un problème pour le travail demandé.

Le point important à retenir est qu'à chaque mise à jour, P pendulaires peuvent entrer dans la **direction** si les premiers segments ne sont pas occupés (Fig 3).

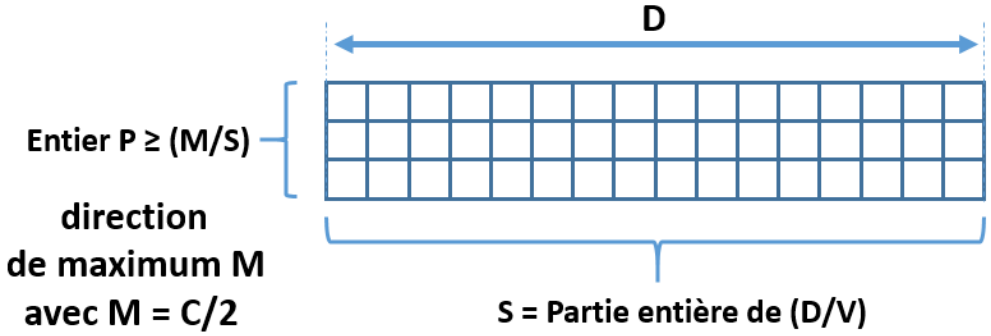


Fig 3 : Une connexion de capacité C comporte deux **directions** de sens opposés.
Une **direction** est discrétisé selon la distance D, la vitesse V et son maximum M=C/2

5. Mise à jour de la position d'un pendulaire:

Il y a trois scénarios à considérer pour un pendulaire lorsque vient son tour de mise à jour :

1. Il n'a pas encore commencé son voyage
    - il peut entrer dans un **passage** de sa première **direction** s'il y en a un dont le premier **segment** est libre (ex : segments (2,1) et (P,1) dans la Fig 4).

2. Il se trouve sur un **passage** d'une **direction** (excepté le dernier **segment** d'un **passage**)
    - *Il ne peut pas changer de passage* / il peut avancer dans le **segment** suivant s'il est libre

3. Il se trouve sur le dernier **segment** d'une **direction** (ex : (2,S) dans Fig 4)
    - Si le nœud est sa destination et que sa capacité permet d'accueillir le pendulaire, alors il est transféré dans ce nœud ; sinon il reste dans son **segment** de **direction**.

    - Si le nœud n'est pas sa destination il peut entrer dans n'importe quel **passage** de la **direction** suivante s'il y en a un dont le premier **segment** est libre ; sinon il reste dans son **segment** de **direction**.

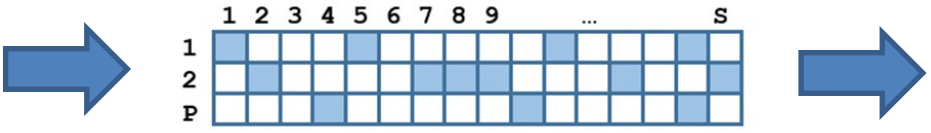


Fig 4 : exemple d'une **direction** partiellement occupée par quelques pendulaires (segments gris-bleu)
Par définition d'une **direction**, tous les pendulaires avancent dans le même sens.
Une seconde **direction** existe pour relier les deux nœuds dans le sens inverse.

6. Ordre de complexité de L'algorithme de Dijkstra:

Certaines questions demandent l'ordre de complexité d'une tâche. C'est pourquoi nous donnons l'ordre de complexité de l'algorithme de Dijkstra car il est établi à l'aide d'une analyse qui n'a pas été vue en cours. Sa complexité dépend de la manière dont la file de priorité TA est mise en oeuvre. Pour l'approche simple illustrée dans la donnée du projet on peut considérer qu'il est en $O(nbN^2)$ où ***nbN*** est le nombre total de Nœuds du graphe.

Dans la suite de ce travail, on suppose que cet algorithme est disponible dans la version vue en cours qui établit le plus court chemin depuis un nœud de départ vers tous les autres nœuds. Vous pouvez l'utiliser comme un appel de fonction en précisant le nœud de départ et l'ensemble des nœuds qui est modifié par cet appel.

Par exemple : Dijkstra (nœud de départ, ensemble des nœuds modifiés par l'appel)

On notera ***O(AlgD)*** pour indiquer le coût de **UN** appel de cet algorithme (pour UN SEUL nœud de départ). Si l'algorithme est appelé plusieurs fois il faudra exprimer le terme dominant avec la ou les variables appropriées.

7. Aperçu des tâches demandées:

Plusieurs tâches sont indépendantes mais il est bon de lire la donnée du travail à effectuer dans l'ordre des tâches car on y décrit des éléments utiles à connaitre pour la suite du travail.

- Vérification de certaines conditions supplémentaires sur la ville (ex 1)
- Définition d'une *destination aléatoire* pour chaque pendulaire (ex 2)
- Détermination des plus courts chemins départ-> destination (ex 3)
- Mise à jour de la position des pendulaires au cours du temps (ex 4)