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

Travail théorique *individuel*

Compléter ce document avec un logiciel de traitement de texte puis produire un pdf pour le téléversement

NOM en MAJUSCULES :

Prénom :

Numéro SCIPER :

Notations du pseudocode : étant donnée le lien avec le projet qui comprend l'usage de pointeurs nous autorisons d'écrire du pseudocode qui empreinte des concepts ou structures de données du C++ (attribut de structure de donnée, pointeur, vector) ou des expressions du C++ (affectation, incrémentation, opérateur d'accès à un attribut…). Veillez néanmoins à exprimer les algorithmes avec concision avec ces mots-clef. Conservez toujours la même convention pour les indices (au choix : de 1 à nb_éléments, ou de 0 à nb_éléments -1).

Exercice 1 : Vérification de conditions supplémentaires sur la ville (4 pts)

Nous supposons que le fichier décrivant la ville *contient au moins un quartier* et ne contient pas d'erreur détectable selon le rendu1 du projet. Cela ne suffit cependant pas pour lancer une simulation intéressante. En effet un fichier correspondant à la figure 1 est correct selon les règles du projet mais présente un problème pour la simulation : un quartier n'est pas connecté au reste de la ville. L'autre cas qui pose problème selon les règles du projet est illustré avec la Figure 2 : un quartier Production ne permet pas d'atteindre trois autres quartiers.

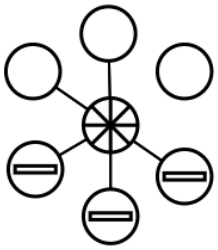

Fig 1 : un nœud n'est pas connecté au reste de la ville

1.1) A partir de ce que vous avez vu en cours, proposez une méthode pour détecter de tels cas. La question générale à répondre est : ***existe-t-il au moins un quartier qui n'est pas atteignable par un autre quartier ?***

Précision : on ne se préoccupe pas de savoir quel(s) quartier(s) pose(nt) problème, on veut seulement une réponse par **oui** ou par **non** qui permettra de commencer une simulation ou pas.

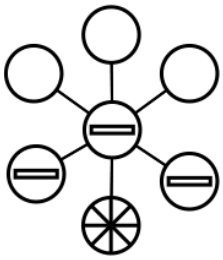

Fig 2 : un nœud Production empêche d'atteindre d'autres nœuds de la ville

Exercice 2 : Définition d'une *destination aléatoire* pour chaque pendulaire (12pts)

Nous supposons à partir de maintenant que toutes les conditions exprimées dans l'exercice 1 sont remplies. Il reste un problème concernant le choix du quartier Production de destination pour chacun des pendulaires.
Si on choisit le quartier Production le plus proche du nœud Logement du pendulaire alors le risque est grand que la capacité des nœuds Productions ne soit pas suffisante pour le nombre des pendulaires les plus proches. Par exemple pour la ville de la figure 3, tous les pendulaires devraient se rendre vers les deux quartiers Production les plus proches des quartiers de Logement et visiblement ces deux quartiers ont une capacité insuffisante.

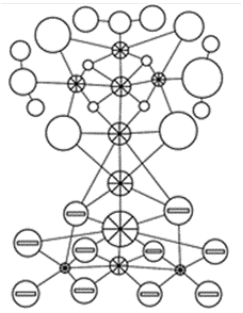

Fig 3 : exemple de ville pour laquelle le plus court chemin Logement -> Production produit un nombre trop grand de pendulaires pour les deux nœuds Production les plus proches des Logements.

C'est pourquoi nous posons que :
- chaque **pendulaire** mémorise son quartier de Logement de **départ** et son quartier Production de **destination**
- un algorithme doit choisir la destination de chaque pendulaire ***de manière aléatoire*** (précision ci-dessous) en veillant à ne pas dépasser la capacité des nœuds de production.

Rappel : un nœud peut mémoriser des données supplémentaires

Pour la mise en œuvre du hasard dans le choix de la destination, on suppose qu'on dispose d'une fonction **hasard(min, max)** qui donne à chaque appel un nombre entier dans l'intervalle [min, max] avec une probabilité uniforme. On pose que cet appel a un coût constant.

2.1) Quelle est la condition à remplir sur la capacité des nœuds de Production pour garantir d'avoir une solution ?

2.2) Quel choix faites-vous pour représenter ces deux données de départ de de destination nécessaires pour chaque pendulaire ?

2.3) Quel choix faites-vous pour représenter l'ensemble des pendulaires ?

2.4) Préciser aussi ci-dessous le nom des ensembles de nœuds dont vous avez besoin pour cet exercice et les noms des variables indiquant le nombre d'éléments de chaque ensemble. Par exemple, y a-t-il un seul ensemble qui contient tous les nœuds ? Ou alors, vos nœuds sont-ils répartis selon leur type en plusieurs ensembles ? A vous de décider. Par la suite, *on suppose que ces ensembles sont accessibles en entrée des algorithmes sans se soucier de faire des appels de fonctions* (c'est du pseudocode, pas du code).

2.5) Décrire en une à deux phrases ci-dessous comment vous envisagez de résoudre cette tâche en indiquant si vous mémorisez des données supplémentaires dans certains types de nœuds. Puis préciser cette idée en fournissant le pseudocode d'un tel algorithme en complétant selon vos besoins la page suivante (le pseudocode ne doit pas être réparti entre deux pages). Si nécessaire l'algorithme peut être décomposé en fonctions (Principes d'abstraction et de ré-utilisation).

1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	
16	
17	
18	
19	
20	
21	
22	
23	
24	
25	
26	
27	
28	
29	
30	
31	
32	
33	
34	
35	
36	
37	
38	
39	
40	
41	
42	

Exercice 3 : Détermination des plus courts chemins départ -> destination (12 pts)

A ce stade, chaque pendulaire connaît son nœud de départ et son nœud de destination.

Nous imposons que la simulation soit effectuée
selon le plus court chemin départ->destination pour chaque pendulaire.

On demande d'utiliser l'algorithme de Dijkstra présenté en cours ; cette forme générale de l'algorithme trouve le plus court chemin vers tous les noeuds du graphe à partir d'un nœud de départ donné.

3.1) Décrire en trois-quatre phrases ci-dessous comment vous envisagez de résoudre cette tâche en indiquant comment est mémorisé le chemin de chaque pendulaire.

3.2) Ecrire le pseudocode sur la page suivante (le pseudocode ne doit pas être réparti entre 2 pages). Si nécessaire l'algorithme peut être décomposé en fonctions (Principes d'abstraction et de ré-utilisation). L'algorithme de Dijkstra peut être appelé comme un simple appel de fonction en précisant ses paramètres : nœud de départ, ensemble de nœuds qui est modifié par l'appel.

3.3) Estimer l'ordre de complexité de votre algorithme.

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
```

Exercice 4 : Proposer le pseudocode de la simulation (12 pts)

A ce stade chaque pendulaire connaît sa suite des directions à suivre, c'est-à-dire la suite des nœuds consécutifs entre le nœud de départ et le nœud d'arrivée.

4.1) Nous devons maintenant préciser le choix effectué pour la représentation fine d'une **direction** selon les indications des *sections 3 à 5 du document décrivant le contexte de la simulation*. Quelle structure de donnée choisissez-vous pour représenter une **direction** ? A quel(s) endroit(s) cette information devrait-elle être mémorisée ? Rappel : on peut ajouter des données aux éléments qui représentent le graphe. Préciser quand ces structures de données doivent être initialisées (nommer les états possibles des segments).

4.2) Egalement sur la base des indications des sections 3 à 5 du document décrivant le contexte de la simulation, quelles autres informations faut-il mémoriser pour chaque pendulaire pour pouvoir mettre à jour sa position au cours de la simulation.

4.3) Ecrire le pseudocode de la boucle de mise à jour asynchrone de la simulation sur la page suivante (le pseudocode ne doit pas être réparti entre 2 pages). Si nécessaire l'algorithme peut être décomposé en fonctions (Principes d'abstraction et de ré-utilisation).

4.4) Estimer l'ordre de complexité de UN seul passage dans la boucle de la simulation (mise à jour de tous les pendulaires pour une seule seconde) en fonction de la ou des variables que vous jugerez pertinentes.

4.5) Décrire en quelques phrases le complément qu'il faudrait ajouter à cet algorithme pour *détecter* qu'un embouteillage (*traffic jam*) ne permet pas à tous les pendulaires d'atteindre leur destination, même en effectuant cette simulation sur un nombre illimité de secondes.

1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	
16	
17	
18	
19	
20	
21	
22	
23	
24	
25	
26	
27	
28	
29	
30	
31	
32	
33	
34	
35	
36	
37	
38	
39	
40	
41	
42	