Compte-rendu

Charles Javerliat, Pierre Sibut-Bourde
INSA Lyon, 3-IF

13 décembre 2019

Mots-clés — Gestion de catalogue, C++, Graphes, Algorithmique, Classes héritées, Polymorphisme, Surcharge.

Résumé

Ce compte-rendu traite du TP n°2 de Programmation Orientée Objet du cours de C++. Il s'agit de réaliser une application pouvant gérer un catalogue de trajets (ajout, suppression), simples ou composés, et pouvant rechercher un itinéraire entre deux points.

Dans un premier temps, on utilisera un diagramme afin d'expliquer les classes mises en place pendant ce TP, les méthodes associées et les liens qui les régissent. Cette partie a pour but d'expliquer l'application et surtout la structure qui s'y niche.

Cette structure doit être exploitée par un utilisateur désireux de rechercher un trajet entre deux villes. Pour cela, on fonctionne en deux temps. Tout d'abord, on explicite l'algorithmique associé à une recherche simple entre deux points directement reliés. Puis l'on cherchera à implémenter un algorithme cherchant la/les combinaisons de trajets entre deux villes. Cela nous conduira à développer des outils simples de théorie des graphes, que nous expliciterons au besoin.

Table des matières

1 Classes : application catalogue de trajets **3**
1.1 Diagramme de classes . . . 3
1.2 Explication . . . 5
1.2.1 ListeChaineeTrajets . . . 5
1.2.2 MaillonListeChaineeTrajets . . . 6
1.2.3 Trajet . . . 6
1.2.4 TrajetSimple . . . 6
1.2.5 TrajetCompose . . . 7
1.2.6 Catalogue . . . 7
1.2.7 StringHelper . . . 8
1.3 État de la mémoire sur un jeu de données . . . 9

2 Algorithme de recherche de trajets **10**

3 Conclusion **13**
3.1 Complexité temporelle de l'algorithme de recherche complexe . . . 13
3.1.1 Meilleur des cas . . . 13
3.1.2 Pire des cas . . . 13
3.1.3 Cas moyen . . . 16
3.2 Normalisation . . . 17
3.3 Passage à l'échelle . . . 17
3.4 Tests unitaires . . . 17

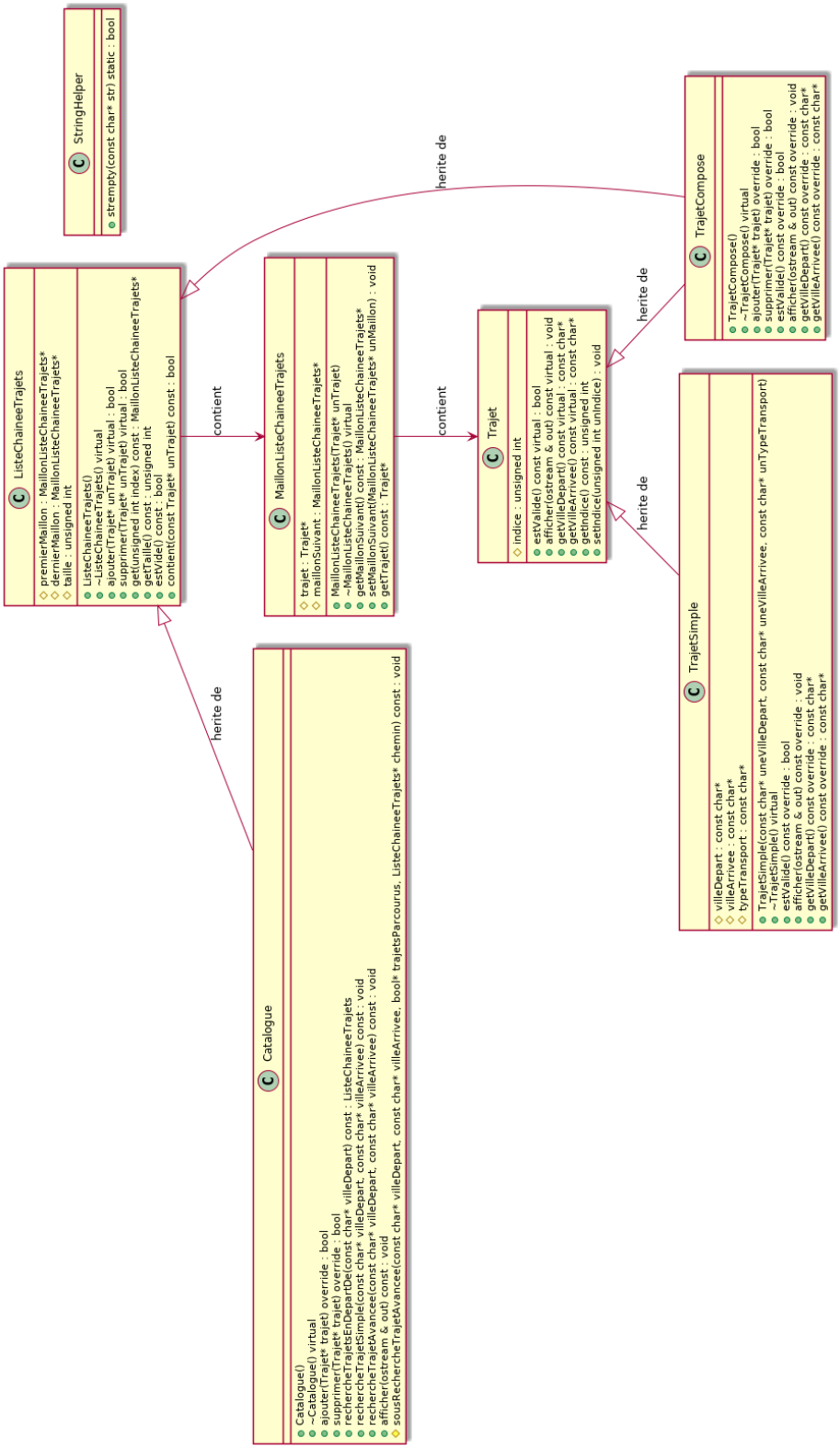
A Code source des diagrammes UML **18**
A.1 Diagramme Figure 1 . . . 18
A.2 Diagramme Figure 2 . . . 18
A.3 Diagramme Figure 3 . . . 18
A.4 Diagramme de classe . . . 20
A.5 Diagramme de la mémoire . . . 22

B Calcul empirique de $(c_n)_{n\in\mathbb{N}}$ **34**
B.1 Comptage : utilisation de nbchemins.py . . . 34
B.2 Comptage : utilisation de printchemins.py . . . 35

C Code source du programme **36**
C.1 ListeChaineeTrajets.h . . . 36
C.2 ListeChaineeTrajets.cpp . . . 39
C.3 MaillonListeChaineeTrajets.h . . . 44
C.4 MaillonListeChaineeTrajets.cpp . . . 46
C.5 Trajet.h . . . 48
C.6 Trajet.cpp . . . 50
C.7 TrajetSimple.h . . . 51
C.8 TrajetSimple.cpp . . . 53
C.9 TrajetCompose.cpp . . . 56
C.10 TrajetCompose.h . . . 60
C.11 Catalogue.h . . . 63
C.12 Catalogue.cpp . . . 66
C.13 StringHelper.h . . . 72
C.14 main.cpp . . . 73

1 Classes : application catalogue de trajets

1.1 Diagramme de classes


StringHelper
strempty(const char* str) static : bool
ListeChaineeTrajets
premierMaillon : MaillonListeChaineeTrajets*
dernierMaillon : MaillonListeChaineeTrajets*
taille : unsigned int
ListeChaineeTrajets()
~ListeChaineeTrajets() virtual
ajouter(Trajet* unTrajet) virtual : bool
supprimer(Trajet* unTrajet) virtual : bool
get(unsigned int index) const : MaillonListeChaineeTrajets*
getTaille() const : unsigned int
estVide() const : bool
contient(const Trajet* unTrajet) const : bool
contient
MaillonListeChaineeTrajets
trajet : Trajet*
maillonSuivant : MaillonListeChaineeTrajets*
MaillonListeChaineeTrajets(Trajet* unTrajet)
~MaillonListeChaineeTrajets() virtual
getMaillonSuivant() const : MaillonListeChaineeTrajets*
setMaillonSuivant(MaillonListeChaineeTrajets* unMaillon) : void
getTrajet() const : Trajet*
contient
Trajet
indice : unsigned int
estValide() const virtual : bool
afficher(ostream & out) const virtual : void
getVilleDepart() const virtual : const char*
getVilleArrivee() const virtual : const char*
getIndice() const : unsigned int
setIndice(unsigned int unIndice) : void
herite de
TrajetCompose
TrajetCompose()
~TrajetCompose() virtual
ajouter(Trajet* trajet) override : bool
supprimer(Trajet* trajet) override : bool
estValide() const override : bool
afficher(ostream & out) const override : void
getVilleDepart() const override : const char*
getVilleArrivee() const override : const char*
herite de
TrajetSimple
villeDepart : const char*
villeArrivee : const char*
typeTransport : const char*
TrajetSimple(const char* uneVilleDepart, const char* uneVilleArrivee, const char* unTypeTransport)
~TrajetSimple() virtual
estValide() const override : bool
afficher(ostream & out) const override : void
getVilleDepart() const override : const char*
getVilleArrivee() const override : const char*
herite de
Catalogue
Catalogue()
~Catalogue() virtual
ajouter(Trajet* trajet) override : bool
supprimer(Trajet* trajet) override : bool
rechercheTrajetsEnDepartDe(const char* villeDepart) const : ListeChaineeTrajets
rechercheTrajetSimple(const char* villeDepart, const char* villeArrivee) const : void
rechercheTrajetAvancee(const char* villeDepart, const char* villeArrivee) const : void
afficher(ostream & out) const : void
sousRechercheTrajetAvancee(const char* villeDepart, const char* villeArrivee, bool* trajetsParcourus, ListeChaineeTrajets* chemin) const : void
herite de

1.2 Explication

On veut créer un catalogue de trajets, c'est-à-dire une liste de trajets. Un trajet est caractérisé par un nom, et peut être ou bien *simple*, ou bien *composé*. Un trajet, s'il est simple, est caractérisé par une ville de départ, une ville d'arrivée, un moyen de transport. Un trajet, s'il est composé, est une suite ordonnée de sous-trajets reliés entre eux. On cherche à découpler au maximum les différentes classes pour faciliter la maintenance future, en rajoutant un maximum de documentation.

Pour vérifier que le contrat des différentes méthodes est bien respecté, nous avons réalisé un ensemble de tests unitaires disponibles via le binaire `catalogue-test` qui contient la version compilée des tests unitaires contenus dans le fichier `test.cpp`.

1.2.1 ListeChaineeTrajets

La classe `ListeChaineeTrajets` contient des `MaillonListeChaineeTrajets`. Elle permet de stocker de manière ordonnée des pointeurs vers des instances de `Trajet` dans ses maillons. La classe permet d'accéder à ces trajets en la parcourant de manière itérative en temps linéaire $O(n)$.

L'utilité d'avoir une classe supplémentaire plutôt que de coder directement une liste chaînée dans `Catalogue` par exemple permet la clarté du code, ainsi que la maintenance simple du fonctionnement interne de la liste. Cette classe propose plusieurs attributs et méthodes :

Attributs

- `MaillonListeChaineeTrajets* premierMaillon` Le pointeur vers le premier maillon de la liste.
- `MaillonListeChaineeTrajets* dernierMaillon` Le pointeur vers le dernier maillon de la liste.
- `unsigned int taille` La taille de la liste, correspond au nombre de maillons.

Méthodes

- `bool ajouter(Trajet* unTrajet)` Ajoute un maillon contenant le `Trajet` à la liste chaînée et retourne vrai si l'action a été réalisée avec succès.
- `bool supprimer(Trajet* unTrajet)` Supprime le maillon d'un trajet de la liste chaînée et retourne vrai si l'action a été réalisée avec succès.
- `MaillonListeChaineeTrajets* get(unsigned int index) const` Retourne le pointeur vers le $i^{\text{ème}}$ maillon de la liste, ou `nullptr` si il n'existe pas.
- `MaillonListeChaineeTrajets* getPremierMaillon() const` Retourne le pointeur vers le premier maillon.
- `MaillonListeChaineeTrajets* getDernierMaillon() const` Retourne le pointeur vers le dernier maillon.
- `unsigned int getTaille() const` Retourne la taille de la liste.
- `bool estVide() const` Retourne vrai si la liste est vide.
- `bool contient(const Trajet* unTrajet) const` Retourne vrai si le trajet est contenu dans la liste.

1.2.2 MaillonListeChaineeTrajets

La classe `MaillonListeChaineeTrajets` contient un pointeur vers un `Trajet` et le maillon suivant. Elle constitue un élément fondamental pour l'implémentation de la `ListeChaineeTrajets`. Cette classe propose plusieurs attributs et méthodes :

Attributs

— `Trajet* trajet` Le pointeur du trajet contenu dans le maillon.
— `MaillonListeChaineeTrajets* maillonSuivant` Le pointeur vers le maillon suivant.

Méthodes

— `MaillonListeChaineeTrajets* getMaillonSuivant()` Retourne le pointeur vers le maillon suivant le maillon actuel.
— `void setMaillonSuivant(MaillonListeChaineeTrajets* unMaillon)` Met à jour le pointeur vers le maillon suivant.
— `Trajet* getTrajet() const` Retourne le pointeur du trajet contenu dans le maillon.

1.2.3 Trajet

La classe `Trajet` est une classe abstraite contenant les attributs et méthodes communs à tous les types de trajets qui héritent de cette classe (`TrajetSimple` et `TrajetCompose`). L'utilisation de classe abstraite permet d'appliquer le principe de subtitution de Liskov, et d'appeler les méthodes sur un type `Trajet` sans se soucier du type réel du `Trajet` et de l'implémentation des méthodes derrière. Cela permet par exemple d'avoir une `ListeChaineeTrajets` pouvant contenir à la fois des `TrajetSimple` et des `TrajetCompose`. Cette classe propose plusieurs attributs et méthodes :

Attributs

— `unsigned int indice` L'indice du trajet dans le catalogue.

Méthodes

— `bool estValide() const` Renvoie vrai si le trajet est valide (attributs corrects, départ différent de l'arrivée, etc).
— `void afficher(ostream & out) const` Affiche le trajet sur le flux de sortie (cout, cerr, ...).
— `const char* getVilleDepart() const` Renvoie la ville de départ du trajet.
— `const char* getVilleArrivee() const` Renvoie la ville d'arrivée du trajet.
— `unsigned int getIndice() const` Renvoie l'indice du trajet dans le catalogue.
— `void setIndice(unsigned int unIndice)` Met à jour l'indice du trajet dans le catalogue.

1.2.4 TrajetSimple

La classe `TrajetSimple` est une classe héritée de `Trajet`, elle définie un trajet direct entre une ville de départ et une ville d'arrivée au moyen d'un type de transport défini. Cette classe propose plusieurs attributs et méthodes :

Attributs

— `const char* villeDepart` Nom de la ville de départ du trajet

— `const char* villeArrivee` Nom de la ville d'arrivée du trajet
— `const char* typeTransport` Nom du type de transport pour aller d'une ville à l'autre

Méthodes

— `bool estValide() const` Renvoie vrai si le trajet est valide (attributs corrects, départ différent de l'arrivée, etc).
— `void afficher(ostream & out) const` Affiche le trajet sur le flux de sortie (cout, cerr, ...).
— `const char* getVilleDepart() const` Renvoie la ville de départ du trajet.
— `const char* getVilleArrivee() const` Renvoie la ville d'arrivée du trajet.

1.2.5 TrajetCompose

La classe `TrajetCompose` est une classe héritée de `Trajet` et de `ListeChaineeTrajets`, elle définie une liste de sous-trajets. Le trajet composé s'occupe de libérer la mémoire associée à ses sous-trajets à sa destruction. Cette classe propose plusieurs méthodes :

— `bool ajouter(Trajet* unTrajet)` Ajoute un trajet au trajet composé, retourne vrai si l'action a été effectuée avec succès.
— `bool supprimer(Trajet* unTrajet)` Supprime un trajet du trajet composé, retourne vrai si l'action a été effectuée avec succès.
— `bool estValide() const` Renvoie vrai si le trajet est valide (attributs corrects, départ différent de l'arrivée, etc).
— `void afficher(ostream & out) const` Affiche le trajet sur le flux de sortie (cout, cerr, ...).
— `const char* getVilleDepart() const` Renvoie la ville de départ du trajet.
— `const char* getVilleArrivee() const` Renvoie la ville d'arrivée du trajet.

1.2.6 Catalogue

La classe `Catalogue` est une classe héritée de `ListeChaineeTrajets`, elle permet de gérer une liste de trajets, et de procéder à des recherches d'itinéraires entre de villes de manière simple (recherche directe de trajet validant la contrainte) ou avancée (recherche par composition de trajets). Le catalogue s'occupe de libérer la mémoire associée à ses trajets à sa destruction. Cette classe propose plusieurs méthodes :

— `bool ajouter(Trajet* unTrajet)` Ajoute un trajet au catalogue, retourne vrai si l'action a été effectuée avec succès.
— `bool supprimer(Trajet* unTrajet)` Supprime un trajet du catalogue, retourne vrai si l'action a été effectuée avec succès.
— `bool estValide() const` Renvoie vrai ou faux selon si le catalogue est valide.
— `ListeChaineeTrajets rechercheTrajetsEnDepartDe(const char* villeDepart) const` Renvoie la liste des trajets en départ de villeDepart.
— `bool rechercheTrajetSimple(const char* villeDepart, const char* villeArrivee) const` Affiche les trajets directs possibles pour aller de

villeDepart à villeArrivee et retourne vrai si au moins un est trouvé.
- `bool rechercheTrajetAvancee(const char* villeDepart, const char* villeArrivee) const` Affiche les trajets ou combinaisons de trajets possibles pour aller de villeDepart à villeArrivee et retourne vrai si au moins un est trouvé. Crée les variables nécessaires pour appeler la méthode récursive `sousRechercheTrajetAvancee`.
- `bool sousRechercheTrajetAvancee(const char* villeDepart, const char* villeArrivee, bool* trajetsParcourus, ListeChaineeTrajets* chemin) const` Affiche les trajets ou combinaisons de trajets possibles pour aller de villeDepart à villeArrivee et retourne vrai si au moins un est trouvé.
- `void afficher(ostream & out) const` Affiche le catalogue sur le flux de sortie.

1.2.7 StringHelper

`StringHelper` permet de rajouter une méthode utilitaire pour vérifier si une chaîne de caractère est vide ou non :
- `static bool strempty(const char* str)` Retourne vrai si la chaîne de caractères est vide.

1.3 État de la mémoire sur un jeu de données

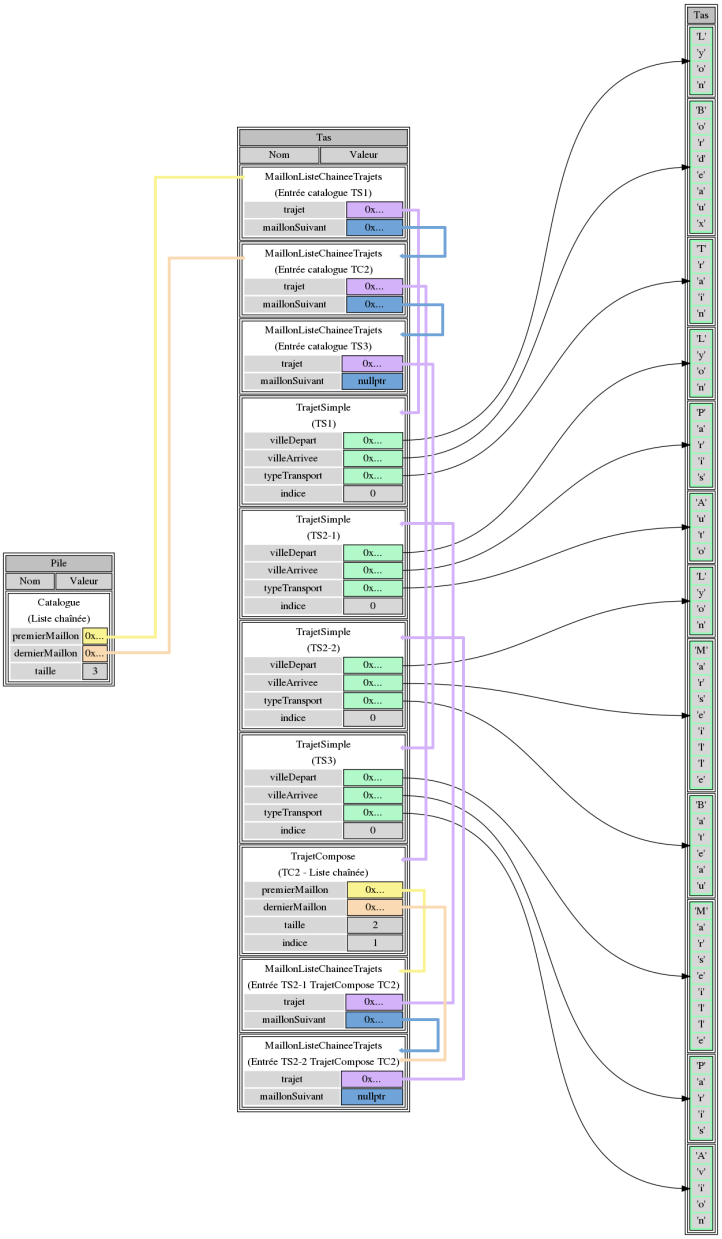
On prend l'exemple d'un catalogue constitué des trois trajets suivants :

— TS1 de Lyon à Bordeaux en Train

— TC2 (TS2-1 de Lyon à Marseille en Bateau) + (TS2-2 de Marseille à Paris en Avion)

— TS3 de Lyon à Paris en Auto

On peut ainsi représenter l'état de la mémoire lorsque tous les ajouts on fini d'être réalisés et que le système de gestion est dans un état stable.



2 Algorithme de recherche de trajets

Soit `C` un catalogue de trajet comme défini précédemment. Pour simplifier l'écriture, on utilisera une correspondance claire qu'il existe entre le catalogue et un graphe. On peut alors implémenter voir ces trajets sur un graphe $\mathcal{G}$, puis appliquer des algorithmes connus sur ce graphe (donc sur le catalogue en réalité) afin de chercher la liste des trajets possibles entre deux points donnés de ce graphe.

Définition 2.1. *Un graphe $\mathcal{G}$ est un couple (E, V) de sommets E (edges) et d'arêtes $V \subset \mathcal{P}_2(E)$ (vertices).*

Dans toute cette partie, on considérera la bijection naturelle φ entre un Catalogue et un graphe (éventuellement multi-arêtes).

Pour ce faire, on lit le catalogue et on considère que les villes forment les sommets E du graphe $\mathcal{G}$ et que les trajets sont les arêtes. Comme il est indiqué dans le sujet, il est obligatoire de suivre un trajet composé du départ à l'arrivée sans possibilité d'en sortir : ceci indique que l'on traitera un trajet composé comme une seule arête, c'est-à-dire une flèche entre la ville d'origine du premier trajet simple le composant et la ville d'arrivée du dernier trajet simple le composant. Dans l'éventualité où il existerait deux trajets entre deux villes, alors il nous faut ajouter les deux arêtes (multi-arêtes).

Exemple. *On va considérer le catalogue suivant :*

```
(Paris, Lyon, TGV)
((Bordeaux, Paris, TGV),(Paris, Genève, Avion))
((Paris, Lyon, TGV),(Lyon, Genève, TER))
(Paris, Genève, TGV)
(Lyon, Clermont-Ferrand, TER)
(Lyon, Marseille, TGV)
(Lyon, Paris, TGV)
(Genève, Paris, TGV)
```

Alors le graphe associé sera celui de FIGURE 1 *(voir page suivante).*

On va alors utiliser un algorithme pour trouver s'il existe un chemin entre deux sommets $s, s' \in E$ du graphe $\mathcal{G} = (E, V)$. Comme un trajet est caractérisé par un triplet `(villeDepart, nom, villeArrivee)`, il suffit alors d'utiliser cet algorithme.

Définition 2.2. *Soit $\mathcal{G} = (E, V)$ un graphe, et soient $s, s' \in E$ deux sommets de $\mathcal{G}$. On dit que s' est un successeur de s, et s antécédent de s' si jamais il existe $v \in V$ une arête telle que $v = (s, s')$. Dans le cas d'un graphe orienté, cette relation n'est pas symétrique, car l'arête $v^{-1} = (s', s)$ n'existe pas nécessairement. Soit $s \in E$, on note $Succ(s)$ l'ensemble des successeurs de s, $Ant(s)$ l'ensemble des antécédents de s.*

Algorithme. (Trajet direct) *Soit $\mathcal{G} = (E, V)$ un graphe, et soient $s, s' \in E$ deux sommets de $\mathcal{G}$. On part de s le point de départ voulu. On regarde les successeurs de s et si s' y apparaît, on affiche le triplet* `(s,s',nom)`.

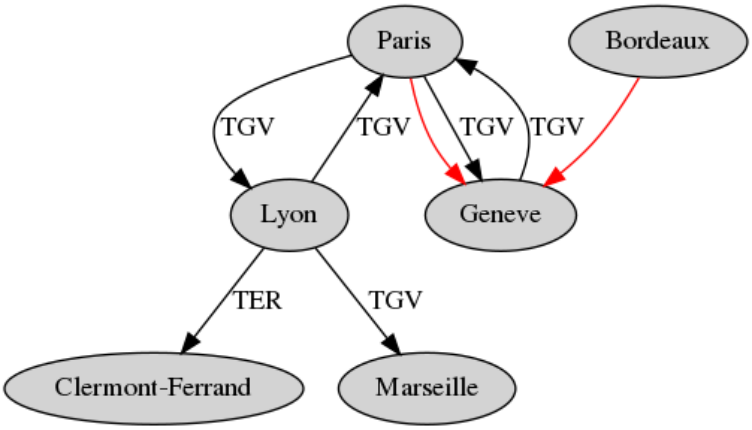


FIGURE 1 – Graphe associé au catalogue précédent. (arête noire : trajet simple, arête rouge : trajet composé)

Quoi faire si l'on veut regarder avec des escales ? On utilise le principe suivant : s'il existe un trajet entre $\widetilde{s}$ et s', alors il existe un trajet entre tous les antécédents de $\widetilde{s}$ et s'. (Preuve simple : $s \to \widetilde{s} \to s'$.)

Définition 2.3. *Soit $\mathcal{G} = (E, V)$ un graphe. Une chaîne élémentaire T de $\mathcal{G}$ est une suite $T = (s_0, ..., s_n) \in E^{n+1}$ de sommets tels que pour tous $i, j \in \{0, ..., n\}$, $s_i \neq s_j$, et s_{i+1} est successeur de s_i pour V. Le cardinal d'une telle chaîne est $n + 1$ et est noté $|C|$.*

Définition 2.4. *Soit $\mathcal{G} = (E, V)$ un graphe. On dit que $\mathcal{G}$ contient un cycle C s'il existe une chaîne élémentaire T de $\mathcal{G}$ telle que l'on ait de plus $s_0 = s_n$* [1]

Exemple. *Prenons $\mathcal{G} = (\{1, 2, 3, 4, 5\}, \{(1, 2), (2, 3), (3, 1), (3, 4), (3, 5)\})$ un graphe orienté. Une chaîne élémentaire de ce graphe est par exemple $(1, 2, 3, 4, 5)$. Un cycle est $(2, 3, 1, 2)$.*

Proposition 2.1. *Soit $\mathcal{G} = (E, V)$ un graphe admettant un cycle. Considérons T l'ensemble des trajets entre deux sommets s et s'. Alors si T est non vide et contient un trajet passant par un sommet du cycle, alors T est infini et l'ensemble $\{|t|, t \in T\}$ n'admet pas de majorant.*

Preuve : Soit t un trajet de T répondant aux hypothèses, notons $\widetilde{s}$ le sommet en question. On a $t_1 : s \to \widetilde{s}$ et $t_2 : \widetilde{s} \to s'$. Comme $\widetilde{s}$ fait partie d'un cycle, il existe $c : \widetilde{s} \to \widetilde{s}$. Mais alors il est clair que $t_2 \circ (c \circ \cdots \circ c) \circ t_1 = t_2 \circ c^n \circ t_1$ est un trajet [2] de s vers s'. De cela, $\{t_2 \circ c^n \circ t_1 | n \in \mathbb{N}\} \subset T$ et on a bien prouvé que T est infini. De plus, comme $|t_2 \circ c^n \circ t_1| = |t_1| + n|c| + |t_2| \underset{n \to +\infty}{\longrightarrow} +\infty$, cela prouve le second point.

Exemple. *Dans le cas d'un cycle (comme c'est le cas pour* `Paris -> Berlin -> Bruxelles` *dans la figure 2), et si l'on veut faire la recherche des trajets de* `Paris` *vers* `Rome`,

1. Cela dit que l'on revient au départ car le « dernier » sommet est également le premier.
2. $\circ$ est la composition usuelle et pour plus de compacité on dit que composer n fois t se note t^n.

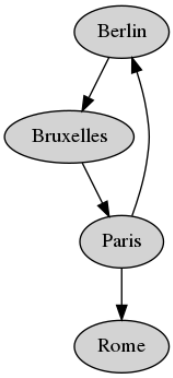


FIGURE 2 – Graphe contenant un cycle.

alors l'algorithme pourrait renvoyer les trajets suivants, en vertu de la proposition précédente :

- `Paris -> Rome`
- `Paris -> Berlin -> Bruxelles -> Rome`
- `(Paris -> Berlin -> Bruxelles -> Paris)`^{n} `-> Rome`

Dans le cadre de ce TP, on souhaiterait éviter de tels désagréments, en particulier car il n'est pas possible d'imprimer une infinité de trajets à l'écran, on va se restreindre aux trajets **ne passant pas deux fois par la même arête**. Dans l'exemple précédent, on renverrait :

- `Paris -> Rome`
- `Paris -> Berlin -> Bruxelles -> Paris -> Rome`

Tout autre parcours du cycle repasse par une arête déjà parcourue, donc le trajet associé est automatiquement exclu.

Définition 2.5. *Soit $\mathcal{G} = (E, V)$ un graphe, et soit $n \in \mathbb{N}$ son nombre de sommets. À renommage des sommets près, on peut considérer que les sommets sont étiquetés sur $E = \{1, 2, ..., n\}$. La matrice d'adjacence du graphe est la matrice $M \in \mathcal{M}_n(\mathbb{R})$ dont les coefficients sont :*

$$[M]_{i,j} = |\{v \in V ; v : i \to j\}|$$

On fait un tableau similaire pour nos algorithmes :

Définition 2.6. *Soit $\mathcal{G} = (E, V)$ un graphe, et soit $n \in \mathbb{N}$ son nombre de sommets. Le tableau des arêtes déjà vues est de type* `bool *aretesDejaVues` *de taille* `n`*. Initialement, tous les coefficients sont à* `false`*. Puis, si l'on traite dans l'algorithme l'arête (i, j), alors conformément à l'idée de la matrice d'adjacence, on fait* `aretesDejaVues[i][j] = true`*.*

Avec ces notions, on va pouvoir écrire l'algorithme de recherche voulu (on suppose comme précédemment que le graphe $\mathcal{G}$ est bien créé).

Algorithme. (Trajet composé)
Soit $\mathcal{G}(E, V)$ le graphe des trajets construit précédemment. Soient $s, s' \in E$ deux sommets de $\mathcal{G}$. On veut faire le trajet $s \rightsquigarrow s'$.
On part de s. Pour tout sommet rencontré dans l'algorithme, notons-le $\hat{s}$, et pour

toute arête non traitée (cf. Proposition 2.1 et Exemple) sortant de $\hat{s}$ on regarde si s' est ville d'arrivée, auquel cas on sait quel trajet nous permet d'arriver à s' depuis $\hat{s}$, donc depuis ses antécédents, etc. Sinon, on fait une récursion sur les voisins issus des arêtes non déjà traitées[3].

Exemple. *On fait cet exemple en se fondant sur la* FIGURE 2. *On veut faire le trajet* `Paris -> Rome`. *Les voisins de* `Paris` *sont* `Berlin` *et* `Rome`. *On affiche donc déjà un trajet. Un cas est traité et n'est plus à traiter. Pour* `Berlin`, *on indique* `true` *pour l'arête, puis l'on regarde les voisins de* `Berlin`, *le seul est* `Bruxelles` *donc on indique* `true` *pour l'arête, puis l'on regarde les voisins de* `Bruxelles`, *le seul est* `Paris` *donc on indique* `true` *pour l'arête, puis l'on regarde les voisins de* `Paris`, *le seul pour une arête non traitée est* `Rome` *donc on affiche le trajet, ceci permet de conclure.*

3 Conclusion

3.1 Complexité temporelle de l'algorithme de recherche complexe

L'algorithme de recherche complexe implémenté ici conduit à recalculer des branches dans la récursion, si bien que sa complexité n'est pas optimale. En effet :

3.1.1 Meilleur des cas

Dans le meilleur des cas, l'algorithme est en $\mathcal{O}(1)$: c'est le cas évident d'un graphe pour lequel le sommet de départ n'a aucun successeur. Dans ce cas, aucun trajet n'est affiché.

3.1.2 Pire des cas

Dans le pire des cas, il nous faut calculer effectivement la complexité de l'algorithme. Pour simplifier la chose, on ne considérera qu'un graphe simple orienté, c'est-à-dire qu'il n'existe pas de multi-arêtes entre deux sommets.

Définition 3.1. *Soit $\mathcal{G}(E,V)$ un graphe simple orienté, avec $|E| = n \in \mathbb{N}$ sommets. Il est dit complet si sa matrice d'adjacence est la matrice $M \in \mathcal{M}_n(\mathbb{R})$ définie par $M = \begin{pmatrix} 0 & 1 & \cdots & 1 \\ 1 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & 1 \\ 1 & \cdots & 1 & 0 \end{pmatrix}$. On note un tel graphe $\widetilde{K_n}$. Pour un tel graphe, il est clair que $|V| = 2\binom{n}{2} = n(n+1)$.*

Définition 3.2. *Soit $\widetilde{K}_n$ le graphe à n sommets défini précédemment. À renommage des sommets près, on peut considérer que l'étiquetage est fait sur $E = \{1, 2, ..., n\}$. Pour $i, j \in E$, on va noter $c_n(i,j)$ le nombre de chemins de*

3. C'est l'algorithme implémenté en C++, on en trouvera une version en Python| en annexe. On lira avec profit la partie qui concerne la complexité temporelle en partie 3.1.

$c : i \to j$ tels que la même arête ne soit pas parcourue deux fois, et $\widetilde{c_n}(i,j)$ le nombre de chemins $c : i \to j$ tels que le même sommet ne soit pas rencontré deux fois.

Proposition 3.1. *Soit $\mathcal{G}(E,V) \simeq \widetilde{K}_n$. Pour $i,j,i',j' \in E$, avec $i \neq j$ et $i' \neq j'$, alors on a :*

$$c_n(i,j) = c_n(i',j')$$
$$\widetilde{c_n}(i,j) = \widetilde{c_n}(i',j')$$

Preuve : On peut considérer $\sigma = (i\ i')(j\ j')$ la permutation, qui ne modifie ni le graphe ni sa matrice d'adjacence associée. De σ on définit donc φ isomorphisme de graphes et alors $c_n(i,j) = c_n(\sigma(i,j)) = c_n(i',j')$. Et de même pour $\widetilde{c_n}(i,j) = \widetilde{c_n}(i',j')$, Ce qui conclut. On va renommer cela c_n et $\widetilde{c_n}$ pour le graphe $\widetilde{K}_n$.

En implémentant la recherche complexe en ne passant pas deux fois par le même **sommet** du graphe, la complexité temporelle dans le pire des cas est de l'ordre de $\widetilde{c_n}$. Or on connaît une formule close [4] pour $\widetilde{c_n}$.

Proposition 3.2. *On a :*

$$\widetilde{c_n} = \sum_{k=0}^{n} \frac{n!}{k!}$$

Preuve : Cela revient à compter le nombre de permutations dans un ensemble à n éléments, c'est la des arrangements de k éléments. On peut retrouver la formule en disant que $\widetilde{c_n} = n\widetilde{c_{n-1}} + 1$.

Proposition 3.3. *On a :*

$$n! \sim \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

Et par suite l'algorithme naïf ne passant pas deux fois par le même sommet est dans le pire des cas : $\mathcal{O}(\frac{n^{n-\frac{1}{2}}}{e^{n-1}})$.

Preuve : Calcul avec la formule de Stirling.

Dans le pire des cas dans le cas **implémenté**, c'est-à-dire dans le cas où l'on ne passe pas deux fois par la même arête du graphe, on voit que l'algorithme doit imprimer c_n trajets entre deux sommets distincts du graphe complet. Ce qui n'est pas chose facile que de de donner une formule close pour c_n : on propose, pour se fixer les idées, de donner deux programmes en Python permettant de compter de tels chemins (c'est l'objet de `nbchemins.py`) puis, dans une amélioration, de les imprimer à l'écran (c'est ce que fait `printchemins.py`). On trouvera ces codes en annexe. Les résultats trouvés [5] sont :

4. On consultera avec profit la suite A000522 sur l'Encyclopédie des Suites d'Entiers, cf. `http://oeis.org/A000522`

5. Dans le cas $n = 6$, on sait que $c_6 > 10^{11}$, sinon largement plus grand que ce minorant. Pour cela, il faut un algorithme bien plus performant, c'est le premier travail à faire et le plus important ici.

n	c_n
1	0
2	1
3	9
4	1085
5	5092429
6	??

On remarque que le nombre de chemins a une croissance très rapide. En entrant cette suite dans l'OEIS, rien ne concorde. On a alors cherché des pistes pour donner une formule close, une formule de récurrence, sans succès à la date d'écriture. Cependant, on a exploré quelques idées :

1. Un graphe complet à n sommets est un graphe complet à $n-1$ sommets auquel on a adjoint un sommet et $2(n-1)$ arêtes (celles entre les anciens sommets et le sommet ajouté).
2. Une autre idée peut-être de construire le graphe non-orienté des arêtes puis de chercher le nombre de chemins sur un graphe non-orienté.
3. Réduire en utilisant des classes d'équivalences ($G \sim G'$ s'il existe un isomorphisme de graphes $\varphi : G \mapsto G'$ qui est clairement une relation d'équivalence [6], puis l'on quotiente par $\sim$, et compter le cardinal de telles classes.

Intuitivement, il est raisonnable de penser que la complexité pour les sommets est un petit o de la complexité concernant les arêtes, ce qui laisse à dire que l'algorithme naïf est inutilisable dans le cas de plus grands catalogues (déjà pour 6 sommets, la complexité fait défaut).

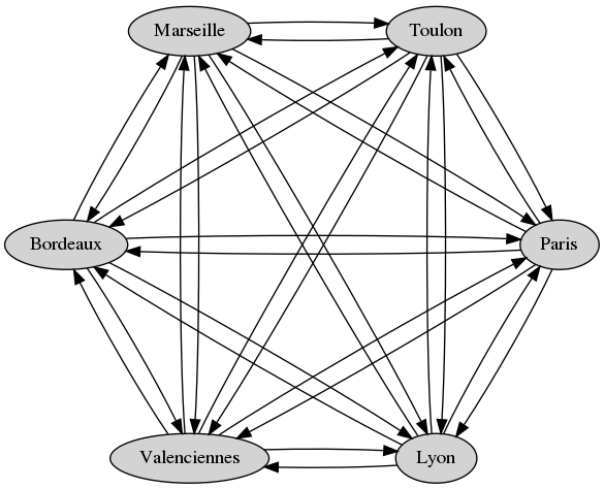


FIGURE 3 – Graphe complet à 6 sommets (le nombre de combinaisons ici est gargantuesque)

6. En effet, *id* fournit $G \sim G$, si l'on a $G \sim G'$, alors on a $\varphi : G \mapsto G'$, il suffit de considérer φ^{-1} pour avoir $G' \sim G$ et l'on peut composer deux isomorphismes, cela conclut.

3.1.3 Cas moyen

Dans le cas moyen, on veut calculer la complexité moyenne de l'algorithme considéré. On a besoin de graphes aléatoires pour définir proprement ce que l'on fait, et pour cela on se fonde sur le modèle binomial proposé par Erdős et Rényi.

Définition 3.3. *Soit $n \in \mathbb{N}\backslash\{0\}$ et $p \in]0,1[$. Une variable aléatoire X à valeur graphes (ici, non orienté) suit le modèle binomial de paramètres (n,p) si la probabilité que X soit égal à un graphe (E,V) de sommets étiquetés sur $\{1,2,...,n\}$ est :*

$$\mathbb{P}(X=(E,V)) = p^{|V|}(1-p)^{\binom{n}{2}-|V|}$$

On remarque que dans le cas $p=\frac{1}{2}$, alors cette probabilité devient

$$\mathbb{P}(X=(E,V)) = 2^{-\binom{n}{2}}$$

On va alors adapter à notre cas d'étude [7].

Définition 3.4. *Soit $n \in \mathbb{N}\backslash\{0\}$ et $p \in]0,1[$. Une variable aléatoire X à valeur graphes (ici, orienté) suit le modèle binomial de paramètres (n,p) si la probabilité que X soit égal à un graphe (E,V) de sommets étiquetés sur $\{1,2,...,n\}$ est :*

$$\mathbb{P}(X=(E,V)) = p^{|V|}(1-p)^{2\binom{n}{2}-|V|}$$

On remarque que dans le cas $p=\frac{1}{2}$, alors cette probabilité devient

$$\mathbb{P}(X=(E,V)) = 2^{-2\binom{n}{2}}$$

Cela revient à que ce que chaque arête entre deux sommets distincts existe avec probabilité p, avec mutuelle indépendance des arêtes. Mais alors, une fois fixé un tel graphe aléatoire (notons-le $\widetilde{\mathbb{G}}_{n,p}$, on peut considérer $C(i,j)$ le nombre de chemins entre i et j deux sommets du graphe (on peut éventuellement se ramener à des classes d'équivalences à isomorphisme de graphes près). Alors la complexité moyenne sera l'espérance $c_{n,p} = \mathbb{E}[C]$ où C est la variable aléatoire donnant le nombre de chemins maximal pour un graphe aléatoire du type $\widetilde{\mathbb{G}}_{n,p}$.

Pour calculer $c_{n,p}$, on peut utiliser les résultats sur le graphe binomial non orienté (nombre d'arêtes moyen, ...) et en déduire, $c_{n,p}$: c'est une possibilité pour aller plus loin, qui demande des outils mathématiques intéressants.

Conclusion de cette sous-partie : Si l'algorithme proposé pour la recherche complexe fonctionne, il apparaît désormais clair que sa complexité temporelle est très mauvaise, voire aberrante. Il faudrait, pour diminuer cela, utiliser une structure de donnée permettant de réduire les calculs. C'est une piste proposée pour l'amélioration algorithmique de l'application développée.

7. Ici comme ci-dessus, on a bien une distribution de probabilités car d'après le binôme de Newton :

$$\sum_V \mathbb{P}(X=(E,V)) = 1$$

3.2 Normalisation

Une autre question se pose assez immédiatement concernant l'étiquetage réel des villes du catalogue. Le choix de notre implémentation est d'avoir une norme minimale : si les chaînes de caractères diffèrent, alors ce n'est pas la même ville. Dans la vie réelle, cela peut poser un certain nombres de soucis car se réfèrent souvent à la même ville :

— `Lyon` et `lyon` (majuscule/minuscule)
— `Puget-sur-Argens` et `Puget sur Argens` (tirets)
— `Bellegarde-sur-Valserine` et `Bellegarde-s/-Valserine` (abréviation)
— `Florence` et `Firenze` (traduction)

Ceci peut nous conduire à l'élaboration d'une norme pour les villes, qui permet de mettre certains sommets les uns avec les autres. Si les trois premiers cas sont relativement simples à mettre en œuvre, le dernier est beaucoup plus complexe car il faut aller vers une langue cible.

3.3 Passage à l'échelle

Dans l'éventualité d'une application distribuée sur le marché, les remarques de la sous-partie 3.1 montrent le premier problème qui peut s'offrir à nous : c'est-à-dire que la complexité de l'algorithme de recherche complexe est, dans le pire des cas, bien insuffisant pour gérer un grand catalogue. Plusieurs remarques doivent être faites :

1. Tout d'abord, certaines propositions de trajet ne sont pas réalistes dans une utilisation réelle (pour un graphe, il faut cependant les considérer) : on peut alors éventuellement affecter un poids à chaque arête (par exemple le temps d'un trajet) et ensuite utiliser un algorithme pour trouver le plus court trajet entre deux points, on pensera à l'algorithme de Dijsktra (dont on sait que pour $\mathcal{G} = (E, V)$, la complexité temporelle est $\mathcal{O}(|E| + |V| \log(|V|))$.
2. L'utilisation d'une structure de donnée adaptée permet, au prix d'une insertion et d'une suppression plus coûteuse, de simplifier l'algorithme. Sauf contexte particulier, le catalogue d'une compagnie de transport ne se modifie pas de façon drastique à l'échelle d'une année, donc on pourrait gagner à la recherche.

3.4 Tests unitaires

Nous avons implémenté des tests unitaires (à la main) pour certaines fonctions utilisées dans l'application de gestion des trajets. Il serait bien mieux de générer des cas intéressants afin d'automatiser de tels tests.

A Code source des diagrammes UML

A.1 Diagramme Figure 1

```
@startuml
digraph graph1 {

  node [style=filled];
  Paris -> Lyon [label="TGV"];

  Bordeaux -> "Geneve" [color=red];
  Paris -> Geneve [color=red];

  Paris -> Geneve [label="TGV"];
  Lyon -> "Clermont-Ferrand" [label="TER"];
  Lyon -> Marseille [label="TGV"];
  Lyon -> Paris [label="TGV"];
  Geneve -> Paris [label="TGV"];

}
@enduml
```

A.2 Diagramme Figure 2

```
@startuml
digraph graph2 {

  node [style=filled];
  Berlin -> Bruxelles;

  Bruxelles -> Paris;
  Paris -> Rome;
  Paris -> Berlin;

}
@enduml
```

A.3 Diagramme Figure 3

```
@startuml
digraph graph3 {

  node [style=filled];
  layout = circo;
  Lyon -> Paris;
  Lyon -> Bordeaux;
  Lyon -> Marseille;
  Lyon -> Toulon;
  Lyon -> Valenciennes;
```

```
    Paris -> Bordeaux;
    Paris -> Marseille;
    Paris -> Toulon;
    Paris -> Valenciennes;
    Paris -> Lyon;

    Bordeaux -> Paris;
    Bordeaux -> Lyon;
    Bordeaux -> Marseille;
    Bordeaux -> Toulon;
    Bordeaux -> Valenciennes;

    Marseille -> Paris;
    Marseille -> Lyon;
    Marseille -> Bordeaux;
    Marseille -> Toulon;
    Marseille -> Valenciennes;

    Toulon -> Paris;
    Toulon -> Lyon;
    Toulon -> Bordeaux;
    Toulon -> Marseille;
    Toulon -> Valenciennes;

    Valenciennes -> Paris;
    Valenciennes -> Lyon;
    Valenciennes -> Bordeaux;
    Valenciennes -> Marseille;
    Valenciennes -> Toulon;
}
@enduml
```

A.4 Diagramme de classe

```
@startuml

class Catalogue {
  +Catalogue()
  +~Catalogue() virtual
  +ajouter(Trajet* trajet) override : bool
  +supprimer(Trajet* trajet) override : bool
  +rechercheTrajetsEnDepartDe(const char* villeDepart) const : ListeChaineeTra
  +rechercheTrajetSimple(const char* villeDepart, const char* villeArrivee) cor
  +rechercheTrajetAvancee(const char* villeDepart, const char* villeArrivee) co
  +afficher(ostream & out) const : void
  #sousRechercheTrajetAvancee(const char* villeDepart, const char* villeArrivee
}

class ListeChaineeTrajets {
  #premierMaillon : MaillonListeChaineeTrajets*
  #dernierMaillon : MaillonListeChaineeTrajets*
  #taille : unsigned int
  +ListeChaineeTrajets()
  +~ListeChaineeTrajets() virtual
  +ajouter(Trajet* unTrajet) virtual : bool
  +supprimer(Trajet* unTrajet) virtual : bool
  +get(unsigned int index) const : MaillonListeChaineeTrajets*
  +getTaille() const : unsigned int
  +estVide() const : bool
  +contient(const Trajet* unTrajet) const : bool
}

class MaillonListeChaineeTrajets {
  #trajet : Trajet*
  #maillonSuivant : MaillonListeChaineeTrajets*
  +MaillonListeChaineeTrajets(Trajet* unTrajet)
  +~MaillonListeChaineeTrajets() virtual
  +getMaillonSuivant() const : MaillonListeChaineeTrajets*
  +setMaillonSuivant(MaillonListeChaineeTrajets* unMaillon) : void
  +getTrajet() const : Trajet*
}

class StringHelper {
  +strempty(const char* str) static : bool
}

class Trajet {
  +estValide() const virtual : bool
  +afficher(ostream & out) const virtual : void
  +getVilleDepart() const virtual : const char*
  +getVilleArrivee() const virtual : const char*
  +getIndice() const : unsigned int
```

```
  +setIndice(unsigned int unIndice) : void
  #indice : unsigned int
}

class TrajetCompose  {
  +TrajetCompose()
  +~TrajetCompose() virtual
  +ajouter(Trajet* trajet) override : bool
  +supprimer(Trajet* trajet) override : bool
  +estValide() const override : bool
  +afficher(ostream & out) const override : void
  +getVilleDepart() const override : const char*
  +getVilleArrivee() const override : const char*
}

class TrajetSimple {
  #villeDepart : const char*
  #villeArrivee : const char*
  #typeTransport : const char*
  +TrajetSimple(const char* uneVilleDepart, const char* uneVilleArrivee, const
  +~TrajetSimple() virtual
  +estValide() const override : bool
  +afficher(ostream & out) const override : void
  +getVilleDepart() const override : const char*
  +getVilleArrivee() const override : const char*
}

ListeChaineeTrajets <|-- Catalogue : "herite de"

Trajet <|-- TrajetSimple : "herite de"

Trajet <|-- TrajetCompose : "herite de"

ListeChaineeTrajets <|-- TrajetCompose : "herite de"

ListeChaineeTrajets --> MaillonListeChaineeTrajets : "contient"

MaillonListeChaineeTrajets --> Trajet : "contient"

@enduml
```

A.5 Diagramme de la mémoire

```
@startuml

digraph memory {

  splines = true;
  nodesep = 2;
  ordering=out;

  subgraph cluster_0 {
  color=none;

  pile
  [
  shape = none
  label = <<table border="1" cellspacing="2">

  <tr><td colspan="2" border="1" bgcolor="gray">Pile</td></tr>
  <tr><td border="1" bgcolor="lightgray">Nom</td><td border="1" bgcolor="lightg

  'Zone memoire du catalogue'
  <tr>
  <td colspan="2">
  <table>
  <tr>
  <td colspan="2" border="0">Catalogue</td>
  </tr>
  <tr>
  <td colspan="2" border="0">(Liste chainee)</td>
  </tr>

  <tr>
  <td bgcolor="#d7d7d7" border="0">premierMaillon</td>
  <td bgcolor="#faf391" border="1" port="cpm">0x...</td>
  </tr>

  <tr>
  <td bgcolor="#d7d7d7" border="0">dernierMaillon</td>
  <td bgcolor="#f9dab2" border="1" port="cdm">0x...</td>
  </tr>

  <tr>
  <td bgcolor="#d7d7d7" border="0">taille</td>
  <td bgcolor="lightgray" border="1" port="cdm">3</td>
  </tr>

  </table>
  </td>
  </tr>
```

```
'Fin de la zone memoire du catalogue'

</table>>
]

tas
[
shape = none
label = <<table border="1" cellspacing="2">

<tr><td colspan="2" border="1" bgcolor="gray">Tas</td></tr>
<tr><td border="1" bgcolor="lightgray">Nom</td><td border="1" bgcolor="lightg

'Zone memoire maillon catalogue m1'
<tr>
<td colspan="2">
<table>
<tr>
<td port="m1" colspan="2" border="0">MaillonListeChaineeTrajets</td>
</tr>
<tr>
<td colspan="2" border="0">(Entree catalogue TS1)</td>
</tr>

<tr>
<td bgcolor="#d7d7d7" border="0">trajet</td>
<td bgcolor="#d3b2f9" border="1" port="m1t">0x...</td>
</tr>

<tr>
<td bgcolor="#d7d7d7" border="0">maillonSuivant</td>
<td bgcolor="#70a3d7" border="1" port="m1ms">0x...</td>
</tr>

</table>
</td>
</tr>
'Fin de la zone memoire maillon catalogue m1'

'Zone memoire maillon catalogue m2'
<tr>
<td colspan="2">
<table>
<tr>
<td port="m2" colspan="2" border="0">MaillonListeChaineeTrajets</td>
</tr>
<tr>
<td colspan="2" border="0">(Entree catalogue TC2)</td>
</tr>
```

```
<tr>
<td bgcolor="#d7d7d7" border="0">trajet</td>
<td bgcolor="#d3b2f9" border="1" port="m2t">0x...</td>
</tr>

<tr>
<td bgcolor="#d7d7d7" border="0">maillonSuivant</td>
<td bgcolor="#70a3d7" border="1" port="m2ms">0x...</td>
</tr>

</table>
</td>
</tr>
'Fin de la zone memoire maillon catalogue m2'

'Zone memoire maillon catalogue m3'
<tr>
<td colspan="2">
<table>
<tr>
<td port="m3" colspan="2" border="0">MaillonListeChaineeTrajets</td>
</tr>
<tr>
<td colspan="2" border="0">(Entree catalogue TS3)</td>
</tr>

<tr>
<td bgcolor="#d7d7d7" border="0">trajet</td>
<td bgcolor="#d3b2f9" border="1" port="m3t">0x...</td>
</tr>

<tr>
<td bgcolor="#d7d7d7" border="0">maillonSuivant</td>
<td bgcolor="#70a3d7" border="1" port="m3ms">nullptr</td>
</tr>

</table>
</td>
</tr>
'Fin de la zone memoire maillon catalogue m3'

'Zone memoire trajet simple TS1'
<tr>
<td colspan="2">
<table>
<tr>
<td port="ts1" colspan="2" border="0">TrajetSimple</td>
</tr>
<tr>
<td colspan="2" border="0">(TS1)</td>
```

```
</tr>

<tr>
<td bgcolor="#d7d7d7" border="0">villeDepart</td>
<td bgcolor="#b2f9ca" port="ts1villedep" border="1">0x...</td>
</tr>

<tr>
<td bgcolor="#d7d7d7" border="0">villeArrivee</td>
<td bgcolor="#b2f9ca" port="ts1villearr" border="1">0x...</td>
</tr>

<tr>
<td bgcolor="#d7d7d7" border="0">typeTransport</td>
<td bgcolor="#b2f9ca" port="ts1typetransp" border="1">0x...</td>
</tr>

<tr>
<td bgcolor="#d7d7d7" border="0">indice</td>
<td bgcolor="#d7d7d7" border="1">0</td>
</tr>

</table>
</td>
</tr>
'Fin de la zone memoire TS1'

'Zone memoire trajet simple TS2-1'
<tr>
<td colspan="2">
<table>
<tr>
<td port="ts2_1" colspan="2" border="0">TrajetSimple</td>
</tr>
<tr>
<td colspan="2" border="0">(TS2-1)</td>
</tr>

<tr>
<td bgcolor="#d7d7d7" border="0">villeDepart</td>
<td bgcolor="#b2f9ca" port="ts2_1villedep" border="1">0x...</td>
</tr>

<tr>
<td bgcolor="#d7d7d7" border="0">villeArrivee</td>
<td bgcolor="#b2f9ca" port="ts2_1villearr" border="1">0x...</td>
</tr>

<tr>
<td bgcolor="#d7d7d7" border="0">typeTransport</td>
```

```
<td bgcolor="#b2f9ca" port="ts2_1typetransp" border="1">0x...</td>
</tr>

<tr>
<td bgcolor="#d7d7d7" border="0">indice</td>
<td bgcolor="#d7d7d7" border="1">0</td>
</tr>

</table>
</td>
</tr>
'Fin de la zone memoire TS2-1'

'Zone memoire trajet simple TS2-2'
<tr>
<td colspan="2">
<table>
<tr>
<td port="ts2_2" colspan="2" border="0">TrajetSimple</td>
</tr>
<tr>
<td colspan="2" border="0">(TS2-2)</td>
</tr>

<tr>
<td bgcolor="#d7d7d7" border="0">villeDepart</td>
<td bgcolor="#b2f9ca" port="ts2_2villedep" border="1">0x...</td>
</tr>

<tr>
<td bgcolor="#d7d7d7" border="0">villeArrivee</td>
<td bgcolor="#b2f9ca" port="ts2_2villearr" border="1">0x...</td>
</tr>

<tr>
<td bgcolor="#d7d7d7" border="0">typeTransport</td>
<td bgcolor="#b2f9ca" port="ts2_2typetransp" border="1">0x...</td>
</tr>

<tr>
<td bgcolor="#d7d7d7" border="0">indice</td>
<td bgcolor="#d7d7d7" border="1">0</td>
</tr>

</table>
</td>
</tr>
'Fin de la zone memoire TS2-2'

'Zone memoire trajet simple TS3'
```

```
<tr>
<td colspan="2">
<table>
<tr>
<td port="ts3" colspan="2" border="0">TrajetSimple</td>
</tr>
<tr>
<td colspan="2" border="0">(TS3)</td>
</tr>

<tr>
<td bgcolor="#d7d7d7" border="0">villeDepart</td>
<td bgcolor="#b2f9ca" port="ts3villedep" border="1">0x...</td>
</tr>

<tr>
<td bgcolor="#d7d7d7" border="0">villeArrivee</td>
<td bgcolor="#b2f9ca" port="ts3villearr" border="1">0x...</td>
</tr>

<tr>
<td bgcolor="#d7d7d7" border="0">typeTransport</td>
<td bgcolor="#b2f9ca" port="ts3typetransp" border="1">0x...</td>
</tr>

<tr>
<td bgcolor="#d7d7d7" border="0">indice</td>
<td bgcolor="#d7d7d7" border="1">0</td>
</tr>

</table>
</td>
</tr>
'Fin de la zone memoire TS3'

'Zone memoire trajet simple TC2'
<tr>
<td colspan="2">
<table>
<tr>
<td port="tc2" colspan="2" border="0">TrajetCompose</td>
</tr>
<tr>
<td colspan="2" border="0">(TC2 - Liste chainee)</td>
</tr>

<tr>
<td bgcolor="#d7d7d7" border="0">premierMaillon</td>
<td bgcolor="#faf391" port="tc2pm" border="1">0x...</td>
</tr>
```

```
<tr>
<td bgcolor="#d7d7d7" border="0">dernierMaillon</td>
<td bgcolor="#f9dab2" port="tc2dm" border="1">0x...</td>
</tr>

<tr>
<td bgcolor="#d7d7d7" border="0">taille</td>
<td bgcolor="lightgray" border="1" port="cdm">2</td>
</tr>

<tr>
<td bgcolor="#d7d7d7" border="0">indice</td>
<td bgcolor="#d7d7d7" border="1">1</td>
</tr>

</table>
</td>
</tr>
'Fin de la zone memoire TC2'

'Zone memoire maillon trajet compose mtc1'
<tr>
<td colspan="2">
<table>
<tr>
<td port="mtc2_1" colspan="2" border="0">MaillonListeChaineeTrajets</td>
</tr>
<tr>
<td colspan="2" border="0">(Entree TS2-1 TrajetCompose TC2)</td>
</tr>

<tr>
<td bgcolor="#d7d7d7" border="0">trajet</td>
<td bgcolor="#d3b2f9" border="1" port="mtc2_1t">0x...</td>
</tr>

<tr>
<td bgcolor="#d7d7d7" border="0">maillonSuivant</td>
<td bgcolor="#70a3d7" border="1" port="mtc2_1ms">0x...</td>
</tr>

</table>
</td>
</tr>
'Fin de la zone memoire maillon catalogue mtc1'

'Zone memoire maillon trajet compose mtc2'
<tr>
<td colspan="2">
```

```
<table>
<tr>
<td port="mtc2_2" colspan="2" border="0">MaillonListeChaineeTrajets</td>
</tr>
<tr>
<td colspan="2" border="0">(Entree TS2-2 TrajetCompose TC2)</td>
</tr>

<tr>
<td bgcolor="#d7d7d7" border="0">trajet</td>
<td bgcolor="#d3b2f9" border="1" port="mtc2_2t">0x...</td>
</tr>

<tr>
<td bgcolor="#d7d7d7" border="0">maillonSuivant</td>
<td bgcolor="#70a3d7" border="1" port="mtc2_2ms">nullptr</td>
</tr>

</table>
</td>
</tr>
'Fin de la zone memoire maillon catalogue mtc2'

</table>>
]

tas2
[
shape = none
label = <<table border="1" cellspacing="2">

<tr><td colspan="2" border="1" bgcolor="gray">Tas</td></tr>

'Zone memoire chaine de caracteres 1'
<tr><td colspan="2">
<table bgcolor="#96fcb6" port="str1">
<tr><td bgcolor="#d7d7d7" border="0">'L'</td></tr>
<tr><td bgcolor="#d7d7d7" border="0">'y'</td></tr>
<tr><td bgcolor="#d7d7d7" border="0">'o'</td></tr>
<tr><td bgcolor="#d7d7d7" border="0">'n'</td></tr>
</table>
</td></tr>
'Fin de la zone memoire chaine de caracteres 1'

'Zone memoire chaine de caracteres 2'
<tr><td colspan="2">
<table bgcolor="#96fcb6" port="str2">
<tr><td bgcolor="#d7d7d7" border="0">'B'</td></tr>
<tr><td bgcolor="#d7d7d7" border="0">'o'</td></tr>
<tr><td bgcolor="#d7d7d7" border="0">'r'</td></tr>
```

```
<tr><td bgcolor="#d7d7d7" border="0">'d'</td></tr>
<tr><td bgcolor="#d7d7d7" border="0">'e'</td></tr>
<tr><td bgcolor="#d7d7d7" border="0">'a'</td></tr>
<tr><td bgcolor="#d7d7d7" border="0">'u'</td></tr>
<tr><td bgcolor="#d7d7d7" border="0">'x'</td></tr>
</table>
</td></tr>
'Fin de la zone memoire chaine de caracteres 2'

'Zone memoire chaine de caracteres 3'
<tr><td colspan="2">
<table bgcolor="#96fcb6" port="str3">
<tr><td bgcolor="#d7d7d7" border="0">'T'</td></tr>
<tr><td bgcolor="#d7d7d7" border="0">'r'</td></tr>
<tr><td bgcolor="#d7d7d7" border="0">'a'</td></tr>
<tr><td bgcolor="#d7d7d7" border="0">'i'</td></tr>
<tr><td bgcolor="#d7d7d7" border="0">'n'</td></tr>
</table>
</td></tr>
'Fin de la zone memoire chaine de caracteres 3'

'Zone memoire chaine de caracteres 4'
<tr><td colspan="2">
<table bgcolor="#96fcb6" port="str4">
<tr><td bgcolor="#d7d7d7" border="0">'L'</td></tr>
<tr><td bgcolor="#d7d7d7" border="0">'y'</td></tr>
<tr><td bgcolor="#d7d7d7" border="0">'o'</td></tr>
<tr><td bgcolor="#d7d7d7" border="0">'n'</td></tr>
</table>
</td></tr>
'Fin de la zone memoire chaine de caracteres 4'

'Zone memoire chaine de caracteres 5'
<tr><td colspan="2">
<table bgcolor="#96fcb6" port="str5">
<tr><td bgcolor="#d7d7d7" border="0">'P'</td></tr>
<tr><td bgcolor="#d7d7d7" border="0">'a'</td></tr>
<tr><td bgcolor="#d7d7d7" border="0">'r'</td></tr>
<tr><td bgcolor="#d7d7d7" border="0">'i'</td></tr>
<tr><td bgcolor="#d7d7d7" border="0">'s'</td></tr>
</table>
</td></tr>
'Fin de la zone memoire chaine de caracteres 5'

'Zone memoire chaine de caracteres 6'
<tr><td colspan="2">
<table bgcolor="#96fcb6" port="str6">
<tr><td bgcolor="#d7d7d7" border="0">'A'</td></tr>
<tr><td bgcolor="#d7d7d7" border="0">'u'</td></tr>
<tr><td bgcolor="#d7d7d7" border="0">'t'</td></tr>
```

```
<tr><td bgcolor="#d7d7d7" border="0">'o'</td></tr>
</table>
</td></tr>
'Fin de la zone memoire chaine de caracteres 6'

'Zone memoire chaine de caracteres 7'
<tr><td colspan="2">
<table bgcolor="#96fcb6" port="str7">
<tr><td bgcolor="#d7d7d7" border="0">'L'</td></tr>
<tr><td bgcolor="#d7d7d7" border="0">'y'</td></tr>
<tr><td bgcolor="#d7d7d7" border="0">'o'</td></tr>
<tr><td bgcolor="#d7d7d7" border="0">'n'</td></tr>
</table>
</td></tr>
'Fin de la zone memoire chaine de caracteres 7'

'Zone memoire chaine de caracteres 8'
<tr><td colspan="2">
<table bgcolor="#96fcb6" port="str8">
<tr><td bgcolor="#d7d7d7" border="0">'M'</td></tr>
<tr><td bgcolor="#d7d7d7" border="0">'a'</td></tr>
<tr><td bgcolor="#d7d7d7" border="0">'r'</td></tr>
<tr><td bgcolor="#d7d7d7" border="0">'s'</td></tr>
<tr><td bgcolor="#d7d7d7" border="0">'e'</td></tr>
<tr><td bgcolor="#d7d7d7" border="0">'i'</td></tr>
<tr><td bgcolor="#d7d7d7" border="0">'l'</td></tr>
<tr><td bgcolor="#d7d7d7" border="0">'l'</td></tr>
<tr><td bgcolor="#d7d7d7" border="0">'e'</td></tr>
</table>
</td></tr>
'Fin de la zone memoire chaine de caracteres 8'

'Zone memoire chaine de caracteres 9'
<tr><td colspan="2">
<table bgcolor="#96fcb6" port="str9">
<tr><td bgcolor="#d7d7d7" border="0">'B'</td></tr>
<tr><td bgcolor="#d7d7d7" border="0">'a'</td></tr>
<tr><td bgcolor="#d7d7d7" border="0">'t'</td></tr>
<tr><td bgcolor="#d7d7d7" border="0">'e'</td></tr>
<tr><td bgcolor="#d7d7d7" border="0">'a'</td></tr>
<tr><td bgcolor="#d7d7d7" border="0">'u'</td></tr>
</table>
</td></tr>
'Fin de la zone memoire chaine de caracteres 9'

'Zone memoire chaine de caracteres 10'
<tr><td colspan="2">
<table bgcolor="#96fcb6" port="str10">
<tr><td bgcolor="#d7d7d7" border="0">'M'</td></tr>
<tr><td bgcolor="#d7d7d7" border="0">'a'</td></tr>
```

```
<tr><td bgcolor="#d7d7d7" border="0">'r'</td></tr>
<tr><td bgcolor="#d7d7d7" border="0">'s'</td></tr>
<tr><td bgcolor="#d7d7d7" border="0">'e'</td></tr>
<tr><td bgcolor="#d7d7d7" border="0">'i'</td></tr>
<tr><td bgcolor="#d7d7d7" border="0">'l'</td></tr>
<tr><td bgcolor="#d7d7d7" border="0">'l'</td></tr>
<tr><td bgcolor="#d7d7d7" border="0">'e'</td></tr>
</table>
</td></tr>
'Fin de la zone memoire chaine de caracteres 10'

'Zone memoire chaine de caracteres 11'
<tr><td colspan="2">
<table bgcolor="#96fcb6" port="str11">
<tr><td bgcolor="#d7d7d7" border="0">'P'</td></tr>
<tr><td bgcolor="#d7d7d7" border="0">'a'</td></tr>
<tr><td bgcolor="#d7d7d7" border="0">'r'</td></tr>
<tr><td bgcolor="#d7d7d7" border="0">'i'</td></tr>
<tr><td bgcolor="#d7d7d7" border="0">'s'</td></tr>
</table>
</td></tr>
'Fin de la zone memoire chaine de caracteres 11'

'Zone memoire chaine de caracteres 12'
<tr><td colspan="2">
<table bgcolor="#96fcb6" port="str12">
<tr><td bgcolor="#d7d7d7" border="0">'A'</td></tr>
<tr><td bgcolor="#d7d7d7" border="0">'v'</td></tr>
<tr><td bgcolor="#d7d7d7" border="0">'i'</td></tr>
<tr><td bgcolor="#d7d7d7" border="0">'o'</td></tr>
<tr><td bgcolor="#d7d7d7" border="0">'n'</td></tr>
</table>
</td></tr>
'Fin de la zone memoire chaine de caracteres 12'

</table>>
]

{rank=same; pile; tas; tas2}


pile:cpm:e -> tas:m1:w [penwidth=2 style=invis color="#c2be6f" fillcolor="#fa
pile:cdm:e -> tas:m3:w [penwidth=2 style=invis color="#b89f80" fillcolor="#f9

tas:tc2pm:e -> tas:mtc2_1:e [penwidth=2 style=invis color="#c2be6f" fillcolor
tas:tc2dm:e -> tas:mtc2_2:e [penwidth=2 style=invis color="#b89f80" fillcolor

tas:m1ms:e -> tas:m2:e [penwidth=2 style=invis color="#5d87b3" fillcolor="#70
tas:m1ms:e -> tas:m2:e [penwidth=2 style=invis color="#5d87b3" fillcolor="#70
tas:m2ms:e -> tas:m3:e [penwidth=2 style=invis color="#5d87b3" fillcolor="#70
```

```
  tas:mtc2_1ms:e -> tas:mtc2_2:e [penwidth=2 style=invis color="#5d87b3" fillco

  tas:m1t:e -> tas:ts1:e [penwidth=2 style=invis color="#86709e" fillcolor="#d3
  tas:m2t:e -> tas:tc2:e [penwidth=2 style=invis color="#86709e" fillcolor="#d3
  tas:m3t:e -> tas:ts3:e [penwidth=2 style=invis color="#86709e" fillcolor="#d3

  tas:mtc2_1t:e -> tas:ts2_1:e [penwidth=2 style=invis color="#86709e" fillcolo
  tas:mtc2_2t:e -> tas:ts2_2:e [penwidth=2 style=invis color="#86709e" fillcolo

  tas:ts1villedep:e -> tas2:str1
  tas:ts1villearr:e -> tas2:str2
  tas:ts1typetransp:e -> tas2:str3

  tas:ts2_1villedep:e -> tas2:str4
  tas:ts2_1villearr:e -> tas2:str5
  tas:ts2_1typetransp:e -> tas2:str6

  tas:ts2_2villedep:e -> tas2:str7
  tas:ts2_2villearr:e -> tas2:str8
  tas:ts2_2typetransp:e -> tas2:str9

  tas:ts3villedep:e -> tas2:str10
  tas:ts3villearr:e -> tas2:str11
  tas:ts3typetransp:e -> tas2:str12
  }

}

@enduml
```

B Calcul empirique de $(c_n)_{n\in\mathbb{N}}$

B.1 Comptage : utilisation de nbchemins.py

```
nb=0 #variable globale

def genereMatrice(n): #On cree la matrice d'adjacence du graphe complet
    mat=[[0 for i in range(n)] for j in range(n)]
    for i in range(n):
        for j in range(n):
            if i!=j:
                mat[i][j]=1
    return mat

def recursiveRechercheComplexe(depart,arrivee,matriceAdjacence,taille):
    global nb # Utilisation de la variable globale
    for i in range(taille):
        if(matriceAdjacence[depart][i]==1): # Si jamais l'arete existe...
            matriceAdjacence[depart][i]=0
            if(i==arrivee): # si valable
                nb+=1
            recursiveRechercheComplexe(i,arrivee,matriceAdjacence,taille)
            matriceAdjacence[depart][i]=1 # reset

def nbchemins(depart, arrivee, taille):
    matriceAdjacence=genereMatrice(taille)
    recursiveRechercheComplexe(depart,arrivee,matriceAdjacence,taille)
    print(nb)
```

Résultats (avec un peu de temps d'attente pour le dernier) :

```
>>> nbchemins(0,1,2)
1
>>> nbchemins(0,1,3)
9
>>> nbchemins(0,1,4)
1085
>>> nbchemins(0,1,5)
5092429
```

Et il est impossible d'obtenir (dans l'état) une estimation de c_6.

B.2 Comptage : utilisation de printchemins.py

```
nb=0 #variable globale

def genereMatrice(n): #On cree la matrice d'adjacence du graphe complet
    mat=[[0 for i in range(n)] for j in range(n)]
    for i in range(n):
        for j in range(n):
            if i!=j:
                mat[i][j]=1
    return mat

def recursiveRechercheComplexe(depart,arrivee,matriceAdjacence,taille,l):
    global nb # Utilisation de la variable globale
    for i in range(taille):
        if(matriceAdjacence[depart][i]==1): # Si jamais l'arete existe...
            l.append(i)
            matriceAdjacence[depart][i]=0
            if(i==arrivee): # si jamais valable
                nb+=1
                print(l)
            recursiveRechercheComplexe(i,arrivee,matriceAdjacence,taille,l)
            l.pop()
            matriceAdjacence[depart][i]=1 # reprendre depuis le dernier appel

def nbchemins(depart, arrivee, taille):
    matriceAdjacence=genereMatrice(taille)
    l=[depart]
    recursiveRechercheComplexe(depart,arrivee,matriceAdjacence,taille,l)
    print(nb)
```

Résultats (seulement dans les deux premiers cas possibles) :

```
>>> printchemins(0,1,2)
[0, 1]
1

>>> printchemins(0,1,3)
[0, 1]
[0, 1, 0, 2, 1]
[0, 1, 2, 0, 2, 1]
[0, 1, 2, 1]
[0, 2, 0, 1]
[0, 2, 0, 1, 2, 1]
[0, 2, 1]
[0, 2, 1, 0, 1]
[0, 2, 1, 2, 0, 1]
9
```

C Code source du programme

C.1 ListeChaineeTrajets.h

```
/*************************************************************************
ListeChaineeTrajets  -  Liste chaînée de MaillonListeChaineeTrajets
-------------------
début                : 20/11/2019
copyright            : (C) 2019 par Charles Javerliat
e-mail               : charles.javerliat@insa-lyon.fr, pierre.sibut-bourde@insa-lyon.fr
*************************************************************************/

//---------- Interface de la classe ListeChaineeTrajets (fichier ListeChaineeTrajets.h) --
#if !defined ( LISTE_CHAINEE_TRAJETS_H )
#define LISTE_CHAINEE_TRAJETS_H

//------------------------------------------------------ Interfaces utilisées

#include "MaillonListeChaineeTrajets.h"

//------------------------------------------------------------------ Constantes

//-------------------------------------------------------------------------- Types

//-----------------------------------------------------------------------------
// Rôle de la classe ListeChaineeTrajets
//
// La classe ListeChaineeTrajets permet de gérer une liste chaînée de Trajets,
// eux mêmes encapsulés dans des maillons (Cf. MaillonListeChaineeTrajets)
//
//-----------------------------------------------------------------------------

class ListeChaineeTrajets
{
  //------------------------------------------------------------------ PUBLIC

public:
  //----------------------------------------------------- Méthodes publiques

  virtual bool ajouter(Trajet* unTrajet);
  // Mode d'emploi : Ajoute un maillon contenant le trajet à la liste chaînée et
  // retourne vrai si l'action a été réalisée avec succès.
  //
  // Contrat :
  // - N'ajoute pas le trajet si il est null
  // - N'ajoute pas le trajet si il est déjà dans la liste
  // - Met à jour la taille de la liste
  //

  virtual bool supprimer(Trajet* unTrajet);
```

```
// Mode d'emploi : Supprime le maillon d'un trajet de la liste chaînée et retourne
// vrai si l'action a été réalisée avec succès.
//
// Contrat :
// - Ne supprime pas le maillon du trajet si il est null
// - Ne supprime pas le maillon du trajet si il n'est pas dans la liste
// - Met à jour la taille de la liste
// - Met à jour les relations entre les maillons

MaillonListeChaineeTrajets* get(unsigned int index) const;
// Mode d'emploi : Retourne le pointeur vers le i_ème maillon de la liste,
// ou nullptr si il n'existe pas
//
// Contrat :
// - Renvoie nullptr si le maillon n'existe pas (index < 0 ou index >= taille)

MaillonListeChaineeTrajets* getPremierMaillon() const;
// Mode d'emploi : Retourne le pointeur vers le premier maillon

MaillonListeChaineeTrajets* getDernierMaillon() const;
// Mode d'emploi : Retourne le pointeur vers le premier maillon

unsigned int getTaille() const;
// Mode d'emploi : Retourne la taille de la liste
// Contrat:
// - La taille doit correspondre au nombre de maillons de la liste

bool estVide() const;
// Mode d'emploi : Retourne vrai si la liste est vide
//
// Contrat :
// - Doit retourner vrai si la taille est de 0 uniquement

bool contient(const Trajet* unTrajet) const;
// Mode d'emploi : Retourne vrai si le trajet est contenu dans la liste

//------------------------------------------- Constructeurs - destructeur

ListeChaineeTrajets ( );
// Mode d'emploi :
//
// Contrat :
//

virtual ~ListeChaineeTrajets ( );
// Mode d'emploi :
//
// Contrat : Supprime l'ensemble des maillons de la liste chainée en mémoire
//
```

```
  //------------------------------------------------------------------ PRIVE

protected:
  //----------------------------------------------------- Méthodes protégées

  //------------------------------------------------------ Attributs protégés

  //Le pointeur vers le premier maillon de la liste
  MaillonListeChaineeTrajets* premierMaillon;

  //Le pointeur vers le dernier maillon de la liste
  MaillonListeChaineeTrajets* dernierMaillon;

  //La taille de la liste, correspond au nombre de maillons
  unsigned int taille;

};

//-------------------------------- Autres définitions dépendantes de ListeChaineeTrajets

#endif // LISTE_CHAINEE_TRAJETS_H
```

C.2 ListeChaineeTrajets.cpp

```
/*************************************************************************
ListeChaineeTrajets  -  Liste chaînée de MaillonListeChaineeTrajets
-------------------
début                : 20/11/2019
copyright            : (C) 2019 par Charles Javerliat
e-mail               : charles.javerliat@insa-lyon.fr, pierre.sibut-bourde@insa-lyon.fr
*************************************************************************/

//---------- Réalisation de la classe ListeChaineeTrajets (fichier ListeChaineeTrajets.cpp

//------------------------------------------------------------------ INCLUDE

//-------------------------------------------------------- Include système
using namespace std;
#include <iostream>

//------------------------------------------------------ Include personnel
#include "ListeChaineeTrajets.h"

//------------------------------------------------------------------ Constantes

//------------------------------------------------------------------ PUBLIC

//---------------------------------------------------- Méthodes publiques

bool ListeChaineeTrajets::ajouter(Trajet* unTrajet)
{
  if(unTrajet == nullptr)
  {
    cerr << "Impossible d'ajouter un trajet NULL." << endl;
    return false;
  }
  else if(contient(unTrajet))
  {
    cerr << "La liste contient déjà ce trajet." << endl;
    return false;
  }
  else
  {
    MaillonListeChaineeTrajets* nouvMaillon = new MaillonListeChaineeTrajets(unTrajet);

    //Met à jour les relations entre les maillons
    if(estVide()) {
      premierMaillon = nouvMaillon;
      dernierMaillon = nouvMaillon;
    } else {
      dernierMaillon->setMaillonSuivant(nouvMaillon);
      dernierMaillon = nouvMaillon;
```

```
    }

    //Met à jour la taille de la liste
    ++taille;
    return true;
  }

} //----- Fin de ajouter

bool ListeChaineeTrajets::supprimer(Trajet* unTrajet)
{
  if(unTrajet == nullptr)
  {
    cerr << "Impossible de supprimer un trajet NULL." << endl;
    return false;
  }
  else if(!contient(unTrajet))
  {
    cerr << "La liste ne contient pas ce trajet." << endl;
    return false;
  }
  else
  {
    MaillonListeChaineeTrajets* maillonPrec = nullptr;
    MaillonListeChaineeTrajets* maillonAct = premierMaillon;

    //Cherche le trajet dans la liste
    while(maillonAct != nullptr) {

      if(maillonAct->getTrajet() == unTrajet) {

        //Met à jour la relation entre les maillons
        if(maillonAct == premierMaillon) {
          premierMaillon = maillonAct->getMaillonSuivant();
        }

        if(maillonAct == dernierMaillon) {
          dernierMaillon = maillonPrec;
        }

        if(maillonPrec != nullptr) {
          maillonPrec->setMaillonSuivant(maillonAct->getMaillonSuivant());
        }

        //On supprime et on met à jour la taille de la liste
        delete maillonAct;
        --taille;
        return true;
      } else {
        maillonPrec = maillonAct;
```

```
        maillonAct = maillonAct->getMaillonSuivant();
      }
    }

    return false;
  }
}

MaillonListeChaineeTrajets* ListeChaineeTrajets::get(unsigned int index) const
{
  if(index >= getTaille())
  {
    return nullptr;
  }
  else
  {
    unsigned int i = 0;
    MaillonListeChaineeTrajets* maillonAct = premierMaillon;

    while(maillonAct != nullptr) {

      if(i == index) {
        return maillonAct;
      }

      maillonAct = maillonAct->getMaillonSuivant();
      ++i;
    }

    return nullptr;
  }
}

MaillonListeChaineeTrajets* ListeChaineeTrajets::getPremierMaillon() const
{
  return premierMaillon;
}

MaillonListeChaineeTrajets* ListeChaineeTrajets::getDernierMaillon() const
{
  return dernierMaillon;
}

unsigned int ListeChaineeTrajets::getTaille() const
{
  return taille;
}

bool ListeChaineeTrajets::contient(const Trajet* unTrajet) const
// Algorithme : Vérifie de manière itérative qu'un des maillon contient le trajet
```

```
//
{
  MaillonListeChaineeTrajets* maillonAct = premierMaillon;

  while(maillonAct != nullptr)
  {
    if(maillonAct->getTrajet() == unTrajet)
    {
      return true;
    }
    maillonAct = maillonAct->getMaillonSuivant();
  }

  return false;

} //----- Fin de contient

bool ListeChaineeTrajets::estVide() const
{
  return taille == 0;
} //----- Fin de estVide

//------------------------------------------- Constructeurs - destructeur

ListeChaineeTrajets::ListeChaineeTrajets ( ) : premierMaillon(nullptr), dernierMaillon(nul
// Algorithme :
//
{
  #ifdef MAP
  cout << "Appel au constructeur de ListeChaineeTrajets" << endl;
  #endif

  taille = 0;
} //----- Fin de ListeChaineeTrajets


ListeChaineeTrajets::~ListeChaineeTrajets ( )
// Algorithme : Parcourt la liste chainée et supprime les maillons un à un
//
{
  #ifdef MAP
  cout << "Appel au destructeur de ListeChaineeTrajets" << endl;
  #endif

  //Supprime les maillons
  MaillonListeChaineeTrajets* maillonActuel = premierMaillon;
  MaillonListeChaineeTrajets* tmp;

  while(maillonActuel != nullptr) {
    tmp = maillonActuel;
```

```
    maillonActuel = maillonActuel->getMaillonSuivant();
    delete tmp;
  }

} //----- Fin de ~ListeChaineeTrajets


//------------------------------------------------------------------ PRIVE

//----------------------------------------------------- Méthodes protégées
```

C.3 MaillonListeChaineeTrajets.h

```
/*************************************************************************
MaillonListeChaineeTrajets  -  Maillon d'une liste chaînée contenant un trajet
-------------------
début                : 20/11/2019
copyright            : (C) 2019 par Charles Javerliat
e-mail               : charles.javerliat@insa-lyon.fr, pierre.sibut-bourde@insa-lyon.fr
*************************************************************************/

//---------- Interface de la classe MaillonListeChaineeTrajets (fichier MaillonListeChaine
#if ! defined ( MAILLON_LISTE_CHAINEE_TRAJETS_H )
#define MAILLON_LISTE_CHAINEE_TRAJETS_H

//------------------------------------------------------ Interfaces utilisées

#include "Trajet.h"

//------------------------------------------------------------------ Constantes

//------------------------------------------------------------------------ Types

//--------------------------------------------------------------------------
// Rôle de la classe MaillonListeChaineeTrajets
//
// La classe MaillonListeChaineeTrajets permet de contenir un trajet dans une
// liste chaînée (Cf ListeChaineeTrajets) et de pointer vers le maillon suivant.
//
//--------------------------------------------------------------------------

class MaillonListeChaineeTrajets
{
  //------------------------------------------------------------------ PUBLIC

public:
  //----------------------------------------------------- Méthodes publiques

  MaillonListeChaineeTrajets* getMaillonSuivant() const;
  // Mode d'emploi : Retourne le pointeur vers le maillon suivant le maillon actuel
  //
  // Contrat :
  // - Renvoie nullptr si il n'y a pas de maillon suivant
  //

  void setMaillonSuivant(MaillonListeChaineeTrajets* unMaillon);
  // Mode d'emploi : Met à jour le pointeur vers le maillon suivant
  //
  // Contrat :
  // - Met à jour le pointeur vers le maillon suivant
```

```
  Trajet* getTrajet() const;
  // Mode d'emploi : Retourne le pointeur du trajet contenu dans le maillon
  //
  // Contrat :
  // - Retourne le pointeur du trajet contenu dans le maillon

  //------------------------------------------- Constructeurs - destructeur

  MaillonListeChaineeTrajets ( Trajet* unTrajet );

  virtual ~MaillonListeChaineeTrajets ( );

  //------------------------------------------------------------------ PRIVE

protected:
  //----------------------------------------------------- Méthodes protégées

  //------------------------------------------------------ Attributs protégés

  //Le pointeur du trajet contenu dans le maillon
  Trajet* trajet;

  //Le pointeur vers le maillon suivant
  MaillonListeChaineeTrajets* maillonSuivant;

};

//-------------------------------- Autres définitions dépendantes de MaillonListeChaineeTr

#endif // MAILLON_LISTE_CHAINEE_TRAJETS_H
```

C.4 MaillonListeChaineeTrajets.cpp

```
/*************************************************************************
MaillonListeChaineeTrajets  -  Maillon d'une liste chaînée contenant un trajet
-------------------
début                : 20/11/2019
copyright            : (C) 2019 par Charles Javerliat
e-mail               : charles.javerliat@insa-lyon.fr, pierre.sibut-bourde@insa-lyon.fr
*************************************************************************/

//---------- Réalisation de la classe MaillonListeChaineeTrajets (fichier MaillonListeChai

//------------------------------------------------------------------ INCLUDE

//-------------------------------------------------------- Include système
using namespace std;
#include <iostream>

//------------------------------------------------------ Include personnel

#include "MaillonListeChaineeTrajets.h"

//------------------------------------------------------------------ Constantes

//------------------------------------------------------------------ PUBLIC

//----------------------------------------------------- Méthodes publiques

MaillonListeChaineeTrajets* MaillonListeChaineeTrajets::getMaillonSuivant() const
{
  return maillonSuivant;
}

void MaillonListeChaineeTrajets::setMaillonSuivant(MaillonListeChaineeTrajets* unMaillon)
{
  maillonSuivant = unMaillon;
}

Trajet* MaillonListeChaineeTrajets::getTrajet() const
{
  return trajet;
}

//-------------------------------------------- Constructeurs - destructeur

MaillonListeChaineeTrajets::MaillonListeChaineeTrajets ( Trajet* unTrajet ) : trajet(unTra
{
  #ifdef MAP
  cout << "Appel au constructeur de MaillonListeChaineeTrajets" << endl;
  #endif
```

```
} //----- Fin de MaillonListeChaineeTrajets

MaillonListeChaineeTrajets::~MaillonListeChaineeTrajets ( )
{
  #ifdef MAP
  cout << "Appel au destructeur de MaillonListeChaineeTrajets" << endl;
  #endif
} //----- Fin de ~MaillonListeChaineeTrajets

//------------------------------------------------------------------ PRIVE

//----------------------------------------------------- Méthodes protégées
```

C.5 Trajet.h

```
/*************************************************************************
Trajet  -  Une abstraction de trajet
-------------------
début                : 20/11/2019
copyright            : (C) 2019 par Charles Javerliat
e-mail               : charles.javerliat@insa-lyon.fr, pierre.sibut-bourde@insa-lyon.fr
*************************************************************************/

//---------- Interface de la classe Trajet (fichier Trajet.h) ----------------
#if !defined ( TRAJET_H )
#define TRAJET_H

//--------------------------------------------------- Interfaces utilisées

#include <iostream>
using namespace std;

//------------------------------------------------------------------ Constantes

//------------------------------------------------------------------------ Types

//------------------------------------------------------------------------------
// Rôle de la classe Trajet
// Classe abstraite contenant les attributs et méthodes communes de tous les types trajets
//
//------------------------------------------------------------------------------

class Trajet
{
  //------------------------------------------------------------------ PUBLIC

public:
  //----------------------------------------------------- Méthodes publiques

  virtual bool estValide() const = 0;
  // Mode d'emploi : Renvoie vrai si le trajet est valide
  //
  // Contrat : Spécifique au type de trajet (Cf. TrajetSimple et TrajetCompose)
  //

  virtual void afficher(ostream & out) const = 0;
  // Mode d'emploi : Affiche le trajet sur le flux de sortie (cout, cerr, ...)
  //
  // Contrat : Spécifique au type de trajet (Cf. TrajetSimple et TrajetCompose)
  //

  virtual const char* getVilleDepart() const = 0;
  // Mode d'emploi : Renvoie la ville de départ du trajet
```

```
  //
  // Contrat : Spécifique au type de trajet (Cf. TrajetSimple et TrajetCompose)
  //

  virtual const char* getVilleArrivee() const = 0;
  // Mode d'emploi : Renvoie la ville d'arrivée du trajet
  //
  // Contrat : Spécifique au type de trajet (Cf. TrajetSimple et TrajetCompose)
  //

  unsigned int getIndice() const;
  // Mode d'emploi : Renvoie l'indice du trajet dans le catalogue

  void setIndice(unsigned int unIndice);
  // Mode d'emploi : Met à jour l'indice du trajet dans le catalogue
  //
  // Contrat :
  // - Doit changer l'indice du trajet

  //------------------------------------------- Constructeurs - destructeur

  Trajet();

  virtual ~Trajet( );

  //------------------------------------------------------------------ PRIVE

protected:
  //----------------------------------------------------- Méthodes protégées

  //L'indice du trajet dans le catalogue
  unsigned int indice;

  //----------------------------------------------------- Attributs protégés

};

//-------------------------------- Autres définitions dépendantes de Trajet

#endif // TRAJET_H
```

C.6 Trajet.cpp

```
/*************************************************************************
Trajet  -  Une abstraction de trajet
-------------------
début                : 20/11/2019
copyright            : (C) 2019 par Charles Javerliat
e-mail               : charles.javerliat@insa-lyon.fr, pierre.sibut-bourde@insa-lyon.fr
*************************************************************************/

//---------- Réalisation de la classe Trajet (fichier Trajet.cpp) ------------

//------------------------------------------------------------------ INCLUDE

//-------------------------------------------------------- Include système

//------------------------------------------------------- Include personnel

#include "Trajet.h"

//------------------------------------------------------------- Constantes

//------------------------------------------------------------------ PUBLIC

//----------------------------------------------------- Méthodes publiques

unsigned int Trajet::getIndice() const
{
  return indice;
}

void Trajet::setIndice(unsigned int unIndice)
{
  indice = unIndice;
}

//-------------------------------------------- Constructeurs - destructeur

Trajet::Trajet() : indice(0) {}

Trajet::~Trajet( ) {}

//------------------------------------------------------------------ PRIVE

//----------------------------------------------------- Méthodes protégées
```

C.7 TrajetSimple.h

```
/*************************************************************************
TrajetSimple  -  Trajet simple allant d'une ville de départ à une ville d'arrivée
à l'aide d'un moyen de transport défini.
-------------------
début                : 20/11/2019
copyright            : (C) 2019 par Charles Javerliat
e-mail               : charles.javerliat@insa-lyon.fr, pierre.sibut-bourde@insa-lyon.fr
*************************************************************************/

//---------- Interface de la classe TrajetSimple (fichier TrajetSimple.h) ----------
#if ! defined ( TRAJET_SIMPLE_H )
#define TRAJET_SIMPLE_H

#include "Trajet.h"

//------------------------------------------------- Interfaces utilisées

//------------------------------------------------------------------ Constantes

//---------------------------------------------------------------------- Types

//-------------------------------------------------------------------------------
// Rôle de la classe TrajetSimple
//
// Classe permettant de décrire un trajet simple: sa ville de départ, d'arrivée
// et le mode de transport
//
//-------------------------------------------------------------------------------

class TrajetSimple : public Trajet
{
  //------------------------------------------------------------------ PUBLIC

public:
  //-------------------------------------------------------- Méthodes publiques

  bool estValide() const override;
  // Mode d'emploi : Renvoie vrai ou faux selon si le trajet est valide ou non
  //
  // Contrat : Renvoie faux dans les cas suivants:
  //  - Le nom de la ville de départ est NULL
  //  - Le nom de la ville de départ est vide
  //  - Le nom de la ville d'arrivée est NULL
  //  - Le nom de la ville d'arrivée est vide
  //  - Le nom de la ville de départ est égale à la ville d'arrivée (sensible à la casse)
  //  - Le nom du type de transport est NULL
  //  - Le nom du type de transport est vide
  // Affiche un message d'erreur sur la sortie standard d'erreur si une des conditions n'est
```

```
  //

  void afficher(ostream & out) const override;
  // Mode d'emploi : Affiche la définition du trajet sur le flux de sortie
  //
  // Contrat : Affiche la description du trajet sur le flux de sortie au format:
  // "Trajet simple de {villeDepart} à {villeArrivee} en {typeTransport}"
  // Sans retour à la ligne
  //

  const char* getVilleDepart() const override;

  const char* getVilleArrivee() const override;

  //------------------------------------------- Constructeurs - destructeur

  TrajetSimple ( const char* uneVilleDepart, const char* uneVilleArrivee, const char* unTy
  // Contrat : Copie en profondeur les chaines de caractères passées en paramètre

  virtual ~TrajetSimple ( );

  //------------------------------------------------------------------ PRIVE

protected:
  //----------------------------------------------------- Méthodes protégées

  //------------------------------------------------------- Attributs protégés

  /**
  * Nom de la ville de départ du trajet
  */
  const char* villeDepart;
  /**
  * Nom de la ville d'arrivée du trajet
  */
  const char* villeArrivee;
  /**
  * Nom du type de transport pour aller d'une ville à l'autre
  */
  const char* typeTransport;

};

//-------------------------------- Autres définitions dépendantes de TrajetSimple

#endif // TRAJET_SIMPLE_H
```

C.8 TrajetSimple.cpp

```
/*************************************************************************
TrajetSimple  -  Trajet simple allant d'une ville de départ à une ville d'arrivée
à l'aide d'un moyen de transport défini.
-------------------
début                : 20/11/2019
copyright            : (C) 2019 par Charles Javerliat
e-mail               : charles.javerliat@insa-lyon.fr, pierre.sibut-bourde@insa-lyon.fr
*************************************************************************/

//---------- Réalisation de la classe TrajetSimple (fichier TrajetSimple.cpp) ------------

//------------------------------------------------------------------ INCLUDE

//-------------------------------------------------------- Include système
using namespace std;
#include <iostream>
#include <cstring>

//------------------------------------------------------ Include personnel
#include "TrajetSimple.h"
#include "StringHelper.h"

//------------------------------------------------------------------ Constantes

//------------------------------------------------------------------ PUBLIC

//-------------------------------------------------------- Méthodes publiques

bool TrajetSimple::estValide() const
// Algorithme : Vérifie toutes les conditions de validité de manière séquentielle
//
{
  if(villeDepart == NULL) {
    cerr << "La ville de départ est invalide : NULL non autorisé." << endl;
    return false;
  }

  if(strempty(villeDepart)) {
    cerr << "Le nom de la ville de départ est invalide : nom vide non autorisé." << endl;
    return false;
  }

  if(villeArrivee == NULL) {
    cerr << "La ville d'arrivée est invalide : NULL non autorisé." << endl;
    return false;
  }

  if(strempty(villeArrivee)) {
```

```
    cerr << "Le nom de la ville d'arrivée est invalide : nom vide non autorisé." << endl;
    return false;
  }

  if(strcmp(villeDepart, villeArrivee) == 0) {
    cerr << "Le nom de la ville d'arrivée ne peut pas être identique au nom de la ville de
    return false;
  }

  if(typeTransport == NULL) {
    cerr << "Le type de transport est invalide : NULL non autorisé." << endl;
    return false;
  }

  if(strempty(typeTransport)) {
    cerr << "Le nom du type de transport est invalide : nom vide non autorisé." << endl;
    return false;
  }

  return true;
} //----- Fin de estValide

void TrajetSimple::afficher(ostream & out) const
{
  //Evite une erreur si un nom pointe vers nullptr
  out << "Trajet simple de ";
  out << (villeDepart == nullptr ? "NULL" : villeDepart);
  out << " à " << (villeArrivee == nullptr ? "NULL" : villeArrivee);
  out << " en " << (typeTransport == nullptr ? "NULL" : typeTransport);
}

const char* TrajetSimple::getVilleDepart() const
{
  return villeDepart;
}

const char* TrajetSimple::getVilleArrivee() const
{
  return villeArrivee;
}

//------------------------------------------- Constructeurs - destructeur

TrajetSimple::TrajetSimple ( const char* uneVilleDepart, const char* uneVilleArrivee, cons
{
  #ifdef MAP
  cout << "Appel au constructeur de TrajetSimple" << endl;
  #endif

  //Copie en profondeur les chaines de caractères passées en paramètre
```

```
  if(uneVilleDepart != nullptr) {
    char* villeDepartCopy = new char[strlen(uneVilleDepart) + 1];
    strcpy(villeDepartCopy, uneVilleDepart);
    villeDepart = villeDepartCopy;
  } else {
    villeDepart = nullptr;
  }

  if(uneVilleArrivee != nullptr) {
    char* villeArriveeCopy = new char[strlen(uneVilleArrivee) + 1];
    strcpy(villeArriveeCopy, uneVilleArrivee);
    villeArrivee = villeArriveeCopy;
  } else {
    villeArrivee = nullptr;
  }

  if(unTypeTransport != nullptr) {
    char* typeTransportCopy = new char[strlen(unTypeTransport) + 1];
    strcpy(typeTransportCopy, unTypeTransport);
    typeTransport = typeTransportCopy;
  } else {
    typeTransport = nullptr;
  }

} //----- Fin de TrajetSimple

TrajetSimple::~TrajetSimple ( )
{
  #ifdef MAP
  cout << "Appel au destructeur de TrajetSimple" << endl;
  #endif

  delete [] villeDepart;
  delete [] villeArrivee;
  delete [] typeTransport;
} //----- Fin de ~TrajetSimple

//------------------------------------------------------------------ PRIVE

//------------------------------------------------------- Méthodes protégées
```

C.9 TrajetCompose.cpp

```
/*************************************************************************
TrajetCompose  -  Trajet composé de plusieurs sous-trajets
-------------------
début                : 20/11/2019
copyright            : (C) 2019 par Charles Javerliat
e-mail               : charles.javerliat@insa-lyon.fr, pierre.sibut-bourde@insa-lyon.fr
*************************************************************************/

//---------- Réalisation de la classe TrajetCompose (fichier TrajetCompose.cpp) ----------

//------------------------------------------------------------------ INCLUDE

//-------------------------------------------------------- Include système
using namespace std;
#include <iostream>
#include <string.h>

//------------------------------------------------------ Include personnel
#include "TrajetCompose.h"

//------------------------------------------------------------------ Constantes

//------------------------------------------------------------------ PUBLIC

//---------------------------------------------------- Méthodes publiques

bool TrajetCompose::ajouter(Trajet* unTrajet)
{
  ListeChaineeTrajets::ajouter(unTrajet);

  //Si le trajet n'est pas valide en l'état, on revient en arrière
  if(!estValide())
  {
    ListeChaineeTrajets::supprimer(unTrajet);

    //Supprime le trajet du tas
    delete unTrajet;

    return false;
  }

  return true;
}

bool TrajetCompose::supprimer(Trajet* unTrajet)
{
  ListeChaineeTrajets::supprimer(unTrajet);
```

```
  //Si le trajet n'est pas valide en l'état, on revient en arrière
  if(!estValide())
  {
    ListeChaineeTrajets::ajouter(unTrajet);
    return false;
  }

  //Supprime le trajet du tas
  delete unTrajet;

  return true;
}

bool TrajetCompose::estValide() const
// Algorithme : On vérifie d'abord si la liste est vide. Sinon, on parcourt tous
// les sous-trajets pour vérifier qu'ils soient tous valides.
//
{
  if(estVide())
  {
    cerr << "Le trajet composé est vide." << endl;
    return false;
  }
  else if(strcmp(getVilleDepart(), getVilleArrivee()) == 0)
  {
    cerr << "Le trajet composé est invalide. La ville de départ ne peut pas être égale à l
    return false;
  }
  else
  {
    MaillonListeChaineeTrajets* maillonAct = premierMaillon;

    //On vérifie que chaque trajet composant le trajet composé est valide
    while(maillonAct != nullptr)
    {

      if(!maillonAct->getTrajet()->estValide())
      {
        cerr << "Le sous-trajet (";
        maillonAct->getTrajet()->afficher(cerr);
        cerr << ") est invalide." << endl;
        return false;
      }
      else if(maillonAct->getMaillonSuivant() != nullptr
              && strcmp(maillonAct->getMaillonSuivant()->getTrajet()->getVilleDepart(), ma
      {
        cerr << "Les trajets (";
        maillonAct->getTrajet()->afficher(cerr);
        cerr << ") et (";
        maillonAct->getMaillonSuivant()->getTrajet()->afficher(cerr);
```

```
        cerr << ") ne coïncident pas (t1.villeArrivee != t2.villeDepart)." << endl;
        return false;
      }

      maillonAct = maillonAct->getMaillonSuivant();
    }
  }

  return true;
} //----- Fin de estValide

void TrajetCompose::afficher(ostream & out) const
// Algorithme : On parcourt tous les sous-trajets pour les afficher un par un
// à la suite dans le flux de sortie.
//
{
  if(estVide())
  {
    out << "Le trajet composé est vide.";
  }
  else
  {
    MaillonListeChaineeTrajets* maillonAct = premierMaillon;

    out << "Trajet composé: ";

    while(maillonAct != nullptr)
    {
      out << '(';
      maillonAct->getTrajet()->afficher(out);
      out << ')';

      if(maillonAct->getMaillonSuivant() != nullptr)
      {
        out << " + ";
      }

      maillonAct = maillonAct->getMaillonSuivant();
    }
  }
}

const char* TrajetCompose::getVilleDepart() const
{
  if(estVide())
  {
    return nullptr;
  }
  else
  {
```

```
    return premierMaillon->getTrajet()->getVilleDepart();
  }
}

const char* TrajetCompose::getVilleArrivee() const
{
  if(estVide())
  {
    return nullptr;
  }
  else
  {
    return dernierMaillon->getTrajet()->getVilleArrivee();
  }
}

//-------------------------------------------- Constructeurs - destructeur

TrajetCompose::TrajetCompose ( ) : Trajet(), ListeChaineeTrajets()
{
  #ifdef MAP
  cout << "Appel au constructeur de TrajetCompose" << endl;
  #endif
} //----- Fin de TrajetCompose


TrajetCompose::~TrajetCompose ( )
{
  #ifdef MAP
  cout << "Appel au destructeur de TrajetCompose" << endl;
  #endif

  //Suppression des trajets sur le tas
  MaillonListeChaineeTrajets* maillonAct = premierMaillon;

  while(maillonAct != nullptr)
  {
    delete maillonAct->getTrajet();
    maillonAct = maillonAct->getMaillonSuivant();
  }
} //----- Fin de ~TrajetCompose


//------------------------------------------------------------------ PRIVE

//----------------------------------------------------- Méthodes protégées
```

C.10 TrajetCompose.h

```
/*************************************************************************
TrajetCompose  -  Trajet composé de plusieurs sous-trajets
-------------------
début                : 20/11/2019
copyright            : (C) 2019 par Charles Javerliat
e-mail               : charles.javerliat@insa-lyon.fr, pierre.sibut-bourde@insa-lyon.fr
*************************************************************************/

//---------- Interface de la classe TrajetCompose (fichier TrajetCompose.h) ---------------
#if !defined ( TRAJET_COMPOSE_H )
#define TRAJET_COMPOSE_H

#include "Trajet.h"
#include "TrajetSimple.h"
#include "ListeChaineeTrajets.h"

//------------------------------------------------- Interfaces utilisées

using namespace std;
#include <iostream>

//------------------------------------------------------------ Constantes

//------------------------------------------------------------------ Types

//-------------------------------------------------------------------------
// Rôle de la classe TrajetCompose
// La classe TrajetCompose permet de gérer un Trajet composé de plusieurs
// sous-trajets.
//
//-------------------------------------------------------------------------

class TrajetCompose : public Trajet, public ListeChaineeTrajets
{
  //------------------------------------------------------------------ PUBLIC

public:
  //----------------------------------------------------- Méthodes publiques

  bool ajouter(Trajet* unTrajet) override;
  // Mode d'emploi : Ajoute un trajet au trajet composé, retourne vrai si l'action a été
  // effectuée avec succès.
  //
  // Contrat :
  // - Ne rajoute pas le trajet si il rend le trajet composé invalide.
  // - Retourne vrai si le trajet a bien été rajouté.
  // - Affiche un message d'erreur sur la sortie standard d'erreur si une des conditions n'e
```

```
  bool supprimer(Trajet* unTrajet) override;
  // Mode d'emploi : Supprime un trajet du trajet composé, retourne vrai si l'action a été
  // effectuée avec succès.
  //
  // Contrat :
  // - Ne supprime pas le trajet si il rend le trajet composé invalide.
  // - Retourne vrai si le trajet a bien été supprimé.
  // - Affiche un message d'erreur sur la sortie standard d'erreur si une des conditions n'

  bool estValide() const override;
  // Mode d'emploi : Renvoie vrai ou faux selon si le trajet est valide
  //
  // Contrat : Renvoie faux dans les cas suivants:
  //  - L'un des trajets simples n'est pas valide
  //  - Pour deux trajets successifs, la ville d'arrivée du premier ne correspond
  //    pas à la ville de départ de l'autre
  //  - Le trajet composé est vide
  //  - La ville de départ est égale à la ville d'arrivée
  //  - Affiche un message d'erreur sur la sortie standard d'erreur si une des conditions n
  //

  void afficher(ostream & out) const override;
  // Mode d'emploi : Affiche la définition du trajet sur la sortie standard
  //
  // Contrat :
  // - Affiche (Sans retour à la ligne) la description du trajet sur le stdout au format:
  //    Si le trajet composé n'est pas vide:
  //      "Trajet composé: ({description trajetSimple1}) + ({description trajetSimple2}) +
  //    Sinon:
  //      "Le trajet composé est vide."
  //

  const char* getVilleDepart() const override;

  const char* getVilleArrivee() const override;

  //------------------------------------------- Constructeurs - destructeur

  TrajetCompose ( );

  virtual ~TrajetCompose ( );

  //------------------------------------------------------------------ PRIVE

protected:
  //----------------------------------------------------- Méthodes protégées

  //----------------------------------------------------- Attributs protégés

};
```

```
//-------------------------------- Autres définitions dépendantes de TrajetCompose

#endif // TRAJET_COMPOSE_H
```

C.11 Catalogue.h

```
/*************************************************************************
Catalogue  -  Classe permettant de gérer une liste de Trajets, et notamment
à procéder à des recherches de trajets entre deux villes.
-------------------
début                : 20/11/2019
copyright            : (C) 2019 par Charles Javerliat
e-mail               : charles.javerliat@insa-lyon.fr, pierre.sibut-bourde@insa-lyon.fr
*************************************************************************/

//---------- Interface de la classe Catalogue (fichier Catalogue.h) ----------------

#if !defined ( CATALOGUE_H )
#define CATALOGUE_H

//-------------------------------------------------- Interfaces utilisées

#include <iostream>
using namespace std;

#include "Trajet.h"
#include "ListeChaineeTrajets.h"

//------------------------------------------------------------------ Constantes

//------------------------------------------------------------------------ Types

//-------------------------------------------------------------------------------
// Rôle de la classe Catalogue
//
// La classe Catalogue permet de gérer une liste de Trajets, et notamment
// à procéder à des recherches simples ou avancées d'itinéraire entre deux villes.
//
//-------------------------------------------------------------------------------

class Catalogue : public ListeChaineeTrajets
{
  //------------------------------------------------------------------- PUBLIC

public:
  //-------------------------------------------------- Méthodes publiques

  bool ajouter(Trajet* unTrajet) override;
  // Mode d'emploi : Ajoute un trajet au catalogue, retourne vrai si l'action a été
  // effectuée avec succès.
  //
  // Contrat :
  // - Ne rajoute pas le trajet si il rend le catalogue invalide.
  // - Attribue un indice au trajet ajouté.
```

```
// - Retourne vrai si le trajet a bien été rajouté.
// - Affiche un message d'erreur sur la sortie standard d'erreur si une des conditions n'e

bool supprimer(Trajet* unTrajet) override;
// Mode d'emploi : Supprime un trajet du catalogue, retourne vrai si l'action a été
// effectuée avec succès.
//
// Contrat :
// - Ne supprime pas le trajet si il rend le catalogue invalide.
// - Met à jour l'indice des trajets suivant le trajet supprimé.
// - Retourne vrai si le trajet a bien été supprimé.
// - Affiche un message d'erreur sur la sortie standard d'erreur si une des conditions n'e

bool estValide() const;
// Mode d'emploi : Renvoie vrai ou faux selon si le catalogue est valide
//
// Contrat : Renvoie faux dans les cas suivants:
//  - L'un des trajets n'est pas valide
// Affiche un message d'erreur sur la sortie standard d'erreur si une des conditions n'est
//

ListeChaineeTrajets rechercheTrajetsEnDepartDe(const char* villeDepart) const;
// Mode d'emploi : Renvoie la liste des trajets en départ de villeDepart
//
// Contrat :
// - Renvoie la liste contenant les trajets en départ de villeDepart
// - Renvoie une liste vide si aucun trajet n'est trouvé.

bool rechercheTrajetSimple(const char* villeDepart, const char* villeArrivee) const;
// Mode d'emploi : Affiche les trajets directs possibles pour aller de villeDepart
// à villeArrivee et retourne vrai si au moins un est trouvé.
//
// Contrat :
// - Affiche les trajets trouvés sur la sortie standard
// - Renvoie vrai si au moins un trajet est trouvé.
//

bool rechercheTrajetAvancee(const char* villeDepart, const char* villeArrivee) const;
// Mode d'emploi : Affiche les trajets ou combinaisons de trajets possibles pour
// aller de villeDepart à villeArrivee et retourne vrai si au moins un est trouvé.
//
// Contrat :
// - Affiche les trajets trouvés sur la sortie standard
// - Renvoie vrai si au moins un trajet est trouvé.

void afficher(ostream & out) const;
// Mode d'emploi : Affiche le catalogue sur le flux de sortie
//
// Contrat : Affiche le catalogue sur le flux de sortie au format:
// * Si le catalogue n'est pas vide:
```

```
  // "Trajet i - {définition trajet_i}" n fois avec un saut de ligne entre chaque trajet
  // * Sinon:
  // "Le catalogue est vide."
  //
  // Sans retour à la ligne à la fin
  //
  //

  //------------------------------------------- Constructeurs - destructeur

  Catalogue ( );
  // Mode d'emploi :
  //
  // Contrat :
  //

  virtual ~Catalogue ( );
  // Mode d'emploi :
  //
  // Contrat :
  //

  //-------------------------------------------------------------------- PRIVE

protected:
  //----------------------------------------------------- Méthodes protégées

  bool sousRechercheTrajetAvancee(const char* villeDepart, const char* villeArrivee, bool*
  // Mode d'emploi : Affiche les trajets ou combinaisons de trajets possibles pour
  // aller de villeDepart à villeArrivee et retourne vrai si au moins un est trouvé.
  //
  // Contrat :
  // - Affiche les trajets trouvés sur la sortie standard
  // - Renvoie vrai si au moins un trajet est trouvé.
  // - trajetsParcourus[i] est vrai si le i_ème trajet a déjà été traité dans la branche d
  // - chemin contient l'ensemble des trajets empruntés dans la branche de récursion actue

  //------------------------------------------------------ Attributs protégés

};

//-------------------------------- Autres définitions dépendantes de Catalogue

#endif // CATALOGUE_H
```

C.12 Catalogue.cpp

```
/*************************************************************************
Catalogue  -  Un catalogue est un ensemble de trajets
-------------------
début                : 20/11/2019
copyright            : (C) 2019 par Charles Javerliat
e-mail               : charles.javerliat@insa-lyon.fr, pierre.sibut-bourde@insa-lyon.fr
*************************************************************************/

//---------- Réalisation de la classe Catalogue (fichier Catalogue.cpp) ------------

//------------------------------------------------------------------ INCLUDE

//-------------------------------------------------------- Include système
using namespace std;
#include <iostream>

#include <string.h>

//------------------------------------------------------ Include personnel

#include "Catalogue.h"

//------------------------------------------------------------------ Constantes

//------------------------------------------------------------------ PUBLIC

//---------------------------------------------------- Méthodes publiques

// Algorithme : Attribue un indice au trajet, puis essaie de l'ajouter,
// si une erreur survient (trajet non valide), on procède à un retour en arrière.
bool Catalogue::ajouter(Trajet* unTrajet)
{
  //Affecte un indice au trajet
  if(unTrajet != nullptr)
  {
    unTrajet->setIndice(getTaille());
  }

  ListeChaineeTrajets::ajouter(unTrajet);

  //Si le catalogue n'est pas valide en l'état, on revient en arrière
  if(!estValide())
  {
    ListeChaineeTrajets::supprimer(unTrajet);

    //Supprime le trajet du tas
    delete unTrajet;
```

```
    return false;
  }

  return true;
}

// Algorithme : Essaie de supprimer un trajet du Catalogue.
// Décrèmente l'indice de tous les trajets suivants si il est effectivement supprimé.
// Si une erreur survient (trajet non valide), on procède à un retour en arrière.
bool Catalogue::supprimer(Trajet* unTrajet)
{
  ListeChaineeTrajets::supprimer(unTrajet);

  //Si le catalogue n'est pas valide en l'état, on revient en arrière
  if(!estValide())
  {
    ListeChaineeTrajets::ajouter(unTrajet);
    return false;
  }

  //On décrémente tous les indices des trajets suivants
  MaillonListeChaineeTrajets* maillonAct = get(unTrajet->getIndice());

  while(maillonAct != nullptr)
  {
    maillonAct->getTrajet()->setIndice(maillonAct->getTrajet()->getIndice() - 1);
    maillonAct = maillonAct->getMaillonSuivant();
  }

  //Supprime le trajet du tas
  delete unTrajet;

  return true;
}

// Algorithme : Vérifie de manière itérative que tous les trajets du catalogue
// sont valides.
bool Catalogue::estValide() const
{
  MaillonListeChaineeTrajets* maillonAct = premierMaillon;

  while(maillonAct != nullptr)
  {
    if(!maillonAct->getTrajet()->estValide())
    {
      return false;
    }

    maillonAct = maillonAct->getMaillonSuivant();
  }
```

```
  return true;
}

// Algorithme : Cherche de manière itérative dans le catalogue les trajets ayant
// pour ville de départ la ville donnée en argument.
ListeChaineeTrajets Catalogue::rechercheTrajetsEnDepartDe(const char* villeDepart) const
{
  ListeChaineeTrajets liste;

  MaillonListeChaineeTrajets* maillonAct = premierMaillon;

  while(maillonAct != nullptr)
  {
    if(strcmp(maillonAct->getTrajet()->getVilleDepart(), villeDepart) == 0)
    {
      liste.ajouter(maillonAct->getTrajet());
    }

    maillonAct = maillonAct->getMaillonSuivant();
  }

  return liste;
}

// Algorithme: Cherche de manière itérative un trajet du catalogue respectant
// la contrainte de ville de départ et d'arrivée.
bool Catalogue::rechercheTrajetSimple(const char* villeDepart, const char* villeArrivee) c
{
  bool found = false;
  MaillonListeChaineeTrajets* maillonAct = premierMaillon;

  while(maillonAct != nullptr) {
    if(strcmp(maillonAct->getTrajet()->getVilleDepart(), villeDepart) == 0
    && strcmp(maillonAct->getTrajet()->getVilleArrivee(), villeArrivee) == 0)
    {
      cout << endl << " - ";
      maillonAct->getTrajet()->afficher(cout);
      found = true;
    }

    maillonAct = maillonAct->getMaillonSuivant();
  }

  return found;

} //----- Fin de rechercheTrajetSimple

// Algorithme : Cherche toutes les combinaisons de trajets respectant la contrainte
// de manière récursive dans le graphe de trajets.
```

```
bool Catalogue::rechercheTrajetAvancee(const char* villeDepart, const char* villeArrivee)
{
  //Le nombre d'arêtes du graphe
  unsigned int nbTrajets = getTaille();
  //Vrai pour une arête si elle a déjà été parcourue (initialisé à false)
  bool* trajetsParcourus = new bool[nbTrajets]();
  //Liste des trajets constituant le chemin
  ListeChaineeTrajets chemin;

  bool found = sousRechercheTrajetAvancee(villeDepart, villeArrivee, trajetsParcourus, &ch
  delete [] trajetsParcourus;

  //Vrai si un trajet a été trouvé
  return found;

} //----- Fin de rechercheTrajetAvancee

// Algorithme: Affiche de manière itérative tous les trajets dans le flux de sortie
void Catalogue::afficher(ostream & out) const
{
  if(estVide())
  {
    out << "Le catalogue est vide.";
  }
  else
  {
    int i = 1;
    MaillonListeChaineeTrajets* maillonAct = premierMaillon;

    while(maillonAct != nullptr) {

      out << "Trajet " << i << " - ";
      maillonAct->getTrajet()->afficher(out);
      out << " (indice = " << maillonAct->getTrajet()->getIndice() << ")";

      if(maillonAct->getMaillonSuivant() != nullptr)
      {
        out << endl;
      }

      maillonAct = maillonAct->getMaillonSuivant();
      ++i;
    }
  }

} //----- Fin de afficher

//------------------------------------------- Constructeurs - destructeur

Catalogue::Catalogue ( ) : ListeChaineeTrajets()
```

```
// Algorithme : Crée le tableau de trajets sur le tas
//
{
  #ifdef MAP
  cout << "Appel au constructeur de Catalogue" << endl;
  #endif
} //----- Fin de Catalogue

Catalogue::~Catalogue ( )
// Algorithme : Détruit le tableau de trajets
//
{
  #ifdef MAP
  cout << "Appel au destructeur de Catalogue" << endl;
  #endif

  //Suppression des trajets sur le tas
  MaillonListeChaineeTrajets* maillonAct = premierMaillon;

  while(maillonAct != nullptr)
  {
    delete maillonAct->getTrajet();
    maillonAct = maillonAct->getMaillonSuivant();
  }

} //----- Fin de ~Catalogue

//------------------------------------------------------------------ PRIVE

//------------------------------------------------------ Méthodes protégées

// Algorithme : Cherche toutes les combinaisons de trajets respectant la contrainte
// de manière récursive dans le graphe de trajets.
bool Catalogue::sousRechercheTrajetAvancee(const char* villeDepart, const char* villeArriv
{
  ListeChaineeTrajets trajets = rechercheTrajetsEnDepartDe(villeDepart);
  MaillonListeChaineeTrajets* maillonAct = trajets.getPremierMaillon();

  //Vrai si un trajet a été trouvé
  bool found = false;

  //Pour chaque trajet possible à partir de la ville de départ donné en argument
  //on fait un appel récursif pour afficher les trajets possibles.
  while(maillonAct != nullptr)
  {
    if(!trajetsParcourus[maillonAct->getTrajet()->getIndice()])
    {
      Trajet* trajet = maillonAct->getTrajet();

      //On ne repassera pas par cette arête dans les appels fils
```

```
      trajetsParcourus[trajet->getIndice()] = true;
      chemin->ajouter(trajet);

      //Si la ville d'arrivée du trajet correspond à la ville d'arrivée qu'on cherche
      if(strcmp(trajet->getVilleArrivee(), villeArrivee) == 0)
      {
        found = true;

        //------------------  Affichage d'un trajet solution --------------------
        MaillonListeChaineeTrajets* maillonAct = chemin->getPremierMaillon();

        cout << endl << " - ";

        while(maillonAct != nullptr)
        {
          maillonAct->getTrajet()->afficher(cout);

          if(maillonAct->getMaillonSuivant() != nullptr)
          {
            cout << " + ";
          }

          maillonAct = maillonAct->getMaillonSuivant();
        }
        //-------------------------------------------------------------------------
      }

      found |= sousRechercheTrajetAvancee(trajet->getVilleArrivee(), villeArrivee, trajetsl

      //On rend à nouveau le trajet disponible pour les autres noeuds du même niveau du gr
      chemin->supprimer(trajet);
      trajetsParcourus[trajet->getIndice()] = false;
    }

    maillonAct = maillonAct->getMaillonSuivant();
  }

  return found;
}
```

C.13 StringHelper.h

```
#if ! defined(STRING_HELPER_H)
#define STRING_HELPER_H

#include <string.h>

static bool strempty(const char* str)
// Mode d'emploi :
//    Retourne vrai si la chaîne de caractères est vide
//
// Contrat :
//    Retourne vrai si la chaîne de caractère ne contient aucun caractères, que des espace
//
// Algorithme :
//    Parcourt tous les caractères de la chaîne de caractère
//    Dès qu'un caractère différent d'une espace est rencontré, retourne faux
//    Si la chaîne de caractères ne contient aucun caractère ou que des espaces, retourne
{
  if(str == nullptr)
  {
    return true;
  }

  for(unsigned int i = 0; i < strlen(str); i++)
  {
    if(str[i] != ' ')
    {
      return false;
    }
  }
  return true;
}

#endif
```

C.14 main.cpp

```
/*************************************************************************
Main  -  Classe principale servant d'interface entre l'utilisateur et le
système de gestion du catalogue
-------------------
début                : 20/11/2019
copyright            : (C) 2019 par Charles Javerliat
e-mail               : charles.javerliat@insa-lyon.fr, pierre.sibut-bourde@insa-lyon.fr
*************************************************************************/

//---------------------------------------------------------------- INCLUDE

//------------------------------------------------------- Include système

#include <string.h>
#include <iostream>
using namespace std;

//------------------------------------------------------ Include personnel
#include "Catalogue.h"
#include "TrajetSimple.h"
#include "TrajetCompose.h"

// Contrat : Affiche le catalogue dans le terminal
static void afficherCatalogue(const Catalogue & catalogue)
{
  cout << endl << " ======  CATALOGUE ====== " << endl << endl;
  catalogue.afficher(cout);
  cout << endl << endl << " === FIN DU CATALOGUE === " << endl << endl;
}

// Contrat : Ajoute un trajet simple au Catalogue en demandant à l'utilisateur
// de renseigner les informations du trajet.
static void ajouterTrajetSimple(Catalogue & catalogue)
{
  cout << endl << " ===  AJOUT D'UN TRAJET SIMPLE === " << endl << endl;
  TrajetSimple* trajetSimple = nullptr;

  do
  {
    char villeDepart[100];
    char villeArrivee[100];
    char typeTransport[100];

    //Prompt de la ville de départ
    do {
      villeDepart[0] = '\0';
      cout << "Ville de départ: ";
      cin.clear();
```

```
      cin.getline(villeDepart, sizeof(villeDepart));
    } while(cin.fail());

    //Prompt de la ville d'arrivée
    do {
      villeArrivee[0] = '\0';
      cout << "Ville d'arrivée: ";
      cin.clear();
      cin.getline(villeArrivee, sizeof(villeArrivee));
    } while(cin.fail());

    //Prompt du type de transport
    do {
      typeTransport[0] = '\0';
      cout << "Type de transport: ";
      cin.clear();
      cin.getline(typeTransport, sizeof(typeTransport));
    } while(cin.fail());

    trajetSimple = new TrajetSimple(villeDepart, villeArrivee, typeTransport);

    //On essaie d'ajouter le trajet, si il n'est pas valide on recommence
  } while(!catalogue.ajouter(trajetSimple));

  cout << endl << " ======== FIN DE L'AJOUT ======== " << endl << endl;
}

// Contrat : Ajoute un trajet composé au Catalogue en demandant à l'utilisateur
// de renseigner les informations du trajet, aka les différents trajets simples
// qui le constitue.
static void ajouterTrajetCompose(Catalogue & catalogue)
{
  cout << endl << " ===  AJOUT D'UN TRAJET COMPOSE === " << endl;
  TrajetCompose* trajetCompose = nullptr;

  //Variable contenant la valeur de retour (o/n) à la question de si
  //on souhaite rajouter un autre trajet simple
  char ajouterTrajetSimple;

  do
  {
    delete trajetCompose;
    trajetCompose = new TrajetCompose();

    do
    {
      TrajetSimple* trajetSimple = nullptr;

      //-------------------- Ajout d'un sous-trajet simple --------------------
      do
```

```
{
  char villeDepart[100];
  char villeArrivee[100];
  char typeTransport[100];
  cout << endl << ">>> Ajout d'un trajet simple <<<" << endl;

  //Prompt de la ville de départ
  do {
    villeDepart[0] = '\0';
    cout << "Ville de départ: ";
    cin.clear();
    cin.getline(villeDepart, sizeof(villeDepart));
  } while(cin.fail());

  //Prompt de la ville d'arrivée
  do {
    villeArrivee[0] = '\0';
    cout << "Ville d'arrivée: ";
    cin.clear();
    cin.getline(villeArrivee, sizeof(villeArrivee));
  } while(cin.fail());

  //Prompt du type de transport
  do {
    typeTransport[0] = '\0';
    cout << "Type de transport: ";
    cin.clear();
    cin.getline(typeTransport, sizeof(typeTransport));
  } while(cin.fail());

  trajetSimple = new TrajetSimple(villeDepart, villeArrivee, typeTransport);

  //On essaie d'ajouter le trajet, si il n'est pas valide on recommence
} while(!trajetCompose->ajouter(trajetSimple));
//-------------------------------------------------------------------------

do
{
  cout << endl << "Ajouter un autre trajet simple (o/n) ? ";
  cin.clear();
  cin >> ajouterTrajetSimple;
  cin.ignore(10000, '\n');

  if(cin.fail() || (ajouterTrajetSimple != 'o' && ajouterTrajetSimple != 'n'))
  {
    cout << "Veuillez écrire o ou n." << endl;
  }

  //Tant que l'utilisateur ne répond pas 'o' ou 'n', on recommence
} while(cin.fail() || (ajouterTrajetSimple != 'o' && ajouterTrajetSimple != 'n'));
```

```
    } while(ajouterTrajetSimple == 'o');

    //On essaie d'ajouter le trajet, si il n'est pas valide on recommence
  } while(!catalogue.ajouter(trajetCompose));

  cout << " ======== FIN DE L'AJOUT ======== " << endl << endl;
}

// Contrat : Supprime un trajet du Catalogue en demandant à l'utilisateur
// de renseigner le numéro du trajet à supprimer.
static void supprimerTrajet(Catalogue & catalogue)
{
  cout << endl << " ===  SUPPRESSION D'UN TRAJET === " << endl;

  if(catalogue.estVide())
  {
    cout << endl << "Aucun trajet à supprimer." << endl;
  }
  else
  {
    unsigned int numeroTrajet;

    do
    {
      cout << endl << "Entrez le numéro de trajet à supprimer (entre 1 et " << catalogue.g
      cin.clear();
      cin >> numeroTrajet;

      if(cin.fail() || numeroTrajet < 1 || numeroTrajet > catalogue.getTaille())
      {
        cout << "Numéro de trajet invalide." << endl;
        cin.clear();
      }
    } while(numeroTrajet < 1 || numeroTrajet > catalogue.getTaille());

    cin.ignore(10000, '\n');

    catalogue.supprimer(catalogue.get(numeroTrajet - 1)->getTrajet());
  }
  cout << endl << " ====== FIN DE LA SUPPRESSION ====== " << endl << endl;
}

// Contrat : Liste tous les trajets allant d'une ville à l'autre sans tester
// de combinaisons possibles de trajets. Demande à l'utilisateur d'entrée la
// ville de départ et d'arrivée.
static void rechercheTrajetSimple(Catalogue & catalogue)
{
  cout << endl << " =====  RECHERCHE DE TRAJET SIMPLE ===== " << endl << endl;
  char villeDepart[100];
```

```
  char villeArrivee[100];

  //Prompt de la ville de départ
  do
  {
    villeDepart[0] = '\0';
    cout << "Ville de départ: ";
    cin.clear();
    cin.getline(villeDepart, 100);

    if(cin.fail())
    {
      cout << "Entrée invalide." << endl;
    }
  } while(cin.fail());

  //Prompt de la ville d'arrivée
  do
  {
    villeArrivee[0] = '\0';
    cout << "Ville d'arrivée: ";
    cin.clear();
    cin.getline(villeArrivee, 100);

    if(cin.fail())
    {
      cout << "Entrée invalide." << endl;
    }

  } while(cin.fail());

  bool found = catalogue.rechercheTrajetSimple(villeDepart, villeArrivee);

  if(!found)
  {
    cout << endl << "Aucun trajet trouvé entre " << villeDepart << " et " << villeArrivee;
  }

  cout << endl << endl << " ==  FIN DE RECHERCHE DE TRAJET SIMPLE == " << endl << endl;
}

// Contrat : Liste tous les trajets allant d'une ville à l'autre en testant
// les combinaisons possibles de trajets. Demande à l'utilisateur d'entrée la
// ville de départ et d'arrivée.
static void rechercheTrajetAvancee(Catalogue & catalogue)
{
  cout << endl << " =====  RECHERCHE DE TRAJET AVANCEE ===== " << endl << endl;
  char villeDepart[100];
  char villeArrivee[100];
```

```
  //Prompt de la ville de départ
  do
  {
    villeDepart[0] = '\0';
    cout << "Ville de départ: ";
    cin.clear();
    cin.getline(villeDepart, 100);

    if(cin.fail())
    {
      cout << "Entrée invalide." << endl;
    }
  } while(cin.fail());

  //Prompt de la ville d'arrivée
  do
  {
    villeArrivee[0] = '\0';
    cout << "Ville d'arrivée: ";
    cin.clear();
    cin.getline(villeArrivee, 100);

    if(cin.fail())
    {
      cout << "Entrée invalide." << endl;
    }
  } while(cin.fail());

  bool found = catalogue.rechercheTrajetAvancee(villeDepart, villeArrivee);

  if(!found)
  {
    cout << endl << "Aucun trajet trouvé entre " << villeDepart << " et " << villeArrivee;
  }

  cout << endl << endl << " ==  FIN DE RECHERCHE DE TRAJET AVANCEE == " << endl << endl;
}

int main(void)
{
  //Instance unique du Catalogue sur la pile
  Catalogue catalogue;

  unsigned short choix = 0;

  cout << " === PROGRAMME DE GESTION DE CATALOGUE === " << endl;

  do
  {
    cout << "Que désirez-vous faire ?" << endl;
```

```
    cout << "\t1 - Afficher le catalogue" << endl;
    cout << "\t2 - Ajouter un trajet simple" << endl;
    cout << "\t3 - Ajouter un trajet composé" << endl;
    cout << "\t4 - Supprimer un trajet" << endl;
    cout << "\t5 - Recherche de trajet simple" << endl;
    cout << "\t6 - Recherche de trajet avancée" << endl;
    cout << "\t7 - Quitter" << endl;

    //Prompt de l'action à effectuer sur la Catalogue
    do {
      cout << "Entrez votre choix: ";
      cin >> choix;

      if(cin.fail() || choix < 1 || choix > 7) {
        cout << "Choix invalide." << endl;
        cin.clear();
      }

      cin.ignore(10000, '\n');

    } while(choix < 1 || choix > 7);

    switch(choix) {

      case 1:
      afficherCatalogue(catalogue);
      break;
      case 2:
      ajouterTrajetSimple(catalogue);
      break;
      case 3:
      ajouterTrajetCompose(catalogue);
      break;
      case 4:
      supprimerTrajet(catalogue);
      break;
      case 5:
      rechercheTrajetSimple(catalogue);
      break;
      case 6:
      rechercheTrajetAvancee(catalogue);
      break;
      default:
      break;
    }

  } while(choix != 7);

  cout << endl << " === FERMETURE DU PROGRAMME === " << endl;
```

```
  return 0;
}
```