Programmation par objets, TP 4 : Graphes

3 séances de TP: 8e, 9e et 10e

Le TP peut être préparé en binôme ou seul. Le rendu doit consister en un fichier archive `<NOM1>_<NOM2>.tar.gz` ou `<NOM1>_<NOM2>.zip` (ou "seul" à la place de `<NOM2>`) contenant :

- votre code C++, tests inclus, commenté et facilement compilable, sans erreur, par des commandes `make`.
- si la commande `make` ne suffit pas à compiler votre code, ajoutez un fichier README expliquant précisément comment compiler votre code.
- (optionnel) un compte-rendu de TP répondant aux questions qui ne demandent pas un programme ainsi que toute explication complémentaire.

Le TP doit être déposé sur moodle.
La date limite de rendu est la fin de la semaine du dernier TP.

L'objectif de ce dernier TP est d'implémenter une application de graphes génériques. On s'intéressera dans un premier temps à la programmation du noyau fonctionnel (les classes spécifiques à notre application et le graphe), puis à la généralisation de nos graphes (en utilisant la généricité par template) et enfin à la réalisation d'une interface graphique par laquelle on puisse visualiser et interagir avec nos objets.

Livrable 1. Application pour le transport aérien (barème indicatif : 7 points)

a. Partie spécifique pour le transport aérien. Implémenter et tester les classes suivantes :

- Aéroport : nom, identificateur unique (par ex. LYS pour Lyon_Saint_Exupéry), coordonnées géographiques (latitude et longitude) etc., ainsi que la liste de ses connexions aller et ses connexions retour (une seule liste ou deux listes séparées, au choix). Ici "liste" veux dire un conteneur de votre choix.
  Ajouter une méthode qui lit l'aéroport à partir d'un flux d'entrée (fichier, voir aussi fstream). On attend ce format, sur une seule ligne : identificateur, nom, latitude, longitude (et d'autres attributs éventuels).
  Ajouter une méthode qui écrit l'aéroport dans un flux de sortie. Même format que pour l'entrée.
- Connexion (directe !) : aéroport de départ, aéroport d'arrivée, durée du vol etc.
  Ajouter une méthode qui lit la connexion à partir d'un flux d'entrée. On attend ce format, sur une seule ligne : aéroport de départ, aéroport d'arrivée, durée du vol (et d'autres attributs éventuels).
  Ajouter une méthode qui écrit la connexion dans un flux de sortie. Même format que pour l'entrée.
  Ajouter aussi des méthodes qui fournissent le coût de la connexion : selon la distance, selon la durée du vol etc. Ces méthodes seront utiles pour les algorithmes de recherche selon les différents critères.

b. Partie graphe. Implémenter un graphe qui pourra facilement devenir un graphe générique (pour le livrable 2). Ou le faire directement générique, avec du template !

Dans ce livrable, pour tester notre application, les sommets du graphe représentent les aéroports et les arêtes (ou arcs) du graphe correspondent aux connexions.

c. Le graphe pourra être créé à partir d'un fichier d'entrée ayant le format suivant :

- première ligne : le nombre d'aéroports
- les lignes suivantes : les aéroports, chaque aéroport sur une nouvelle ligne : identificateur, nom, latitude, longitude (et d'autres attributs éventuels)
- la ligne suivante : le nombre de connexions
- les lignes suivantes, une ligne pour chaque connexion : aéroport de départ, aéroport d'arrivée, durée du vol (et d'autres attributs éventuels)

Indications : Représentation interne du graphe à votre choix. Pour les collections d'objets (aéroports, connexions ou autres), utiliser parmi les conteneurs de la bibliothèque Standard Template Library (STL) : vector, list, set, map, queue, stack etc. ou créer vos propres conteneurs.

d. Ajouter et supprimer une connexion.

e. Ajouter et supprimer un aéroport (et implicitement toutes ses connexions, pour la suppresion).

f. Afficher le graphe à l'écran et aussi dans un fichier de sortie (du même format que le fichier d'entrée).

g. Trouver et afficher le trajet le plus court entre deux aéroports donnés, selon différents critères :
— la durée effective de vol (somme des durées de tous les vols) ;
— la distance (pour simplicité, les distances peuvent être calculées à partir des latitudes et longitudes seules) ;
— le nombre d'escales ;
— critères multiple, par exemple : le nombre d'escales, ensuite la durée effective de vol.

h. (Facultatif) Pour aller plus loin : les connexions peuvent être des vols effectifs, ayant un numéro de vol, une compagnie aérienne, un type d'aéronave, une date et heure de départ, une date et heure d'arrivée etc. Dans ce cas on peut aussi ajouter comme critères de recherche : une durée totale du trajet (attentes entre vols incluses), une durée minimale (et/ou maximale) pour les escales ou autre.

Livrable 2. Graphe générique template (barème indicatif : 5 points)

a. Généraliser le graphe afin qu'il puisse manipuler toute sorte de sommets et arêtes : utiliser template. Et faire fonctionner notre application du livrable 1 en utilisant ce graphe générique. (si ce n'est déjà fait pour le livrable 1)

b. (Facultatif) Faire fonctionner notre graphe générique pour une autre application de graphe. Par exemple, pour un réseau informatique ou pour le transport naval ou pour le transport routier. Bien sûr, il nous faut d'autres classes à la place de Aéroport et Connexion.

c. (Facultatif) Pour aller plus loin : Créer une hierarchie de classes pour le graphes qui modélise plusieurs types de graphes : orienté, non-orienté, pondéré et non-pondéré, ainsi que des algorithmes spécifiques : les parcours en largeur, le parcours en profondeur, l'algorithme de Dijkstra (chemin le plus court deux sommets), l'algorithme de Prim (arbre couvrant minimal), l'algorithme de Floyd-Warshall (distances des plus courts chemins entre toutes les paires de sommets) etc.

Indication pour tous le points : Bien garder la modularité du code tout au long de son évolution (c.-à-d. ne pas mélanger les spécificités de notre application avec celles du graphe).

Livrable 3. Construction d'une interface graphique (barème indicatif : 8 points)

On se propose maintenant d'ajouter une interface graphique `wxWidget` à notre application pour le transport aérien.

a. Implémenter cette interface qui devra :
— dessiner les sommets et arêtes (aéroports et connexions)
— récupérer des événements de clic souris pour faire séléctionner un sommet (et éventuellement une arête)
— proposer des boutons (fonctionnels !) : "Affiche étiquettes" (c.-à-d. identificateur, durée du vol etc.), "Supprime sommet", "Chemin le plus court", "Quitter" etc.

Pour le chemin le plus court, le visualiser avec des couleurs différentes ou même une animation (en utilisant un `wxTimer`)

b. Pour une meilleure visualisation, on pourrait aussi déplacer les sommets (partie graphique). Pour simuler cela, créer une animation en utilisant un `wxTimer` pour dessiner le déplacement d'un sommet (drag and drop, glisser et déposer). Sinon, changer la position d'une arête sans animation.

c. (Facultatif) Améliorer l'interface graphique : personnaliser les sommets et/ou arêtes (différentes formes, couleurs, en ajoutant des images), ajouter des menus etc.

Indication : Bien garder la modularité du code tout au long de son évolution (c.-à-d. ne pas mélanger la partie spécifique de notre application avec celles de l'intérface graphique !).