TP Graphes 6

Licence Informatique - 2nde année

Année 2022-2023

1 Compilation séparée

Dans ce premier exercice, vous allez être amenés à expérimenter la compilation séparée, à partir de la correction qui vous est fournie du TP5. Le fichier correspondant contient en effet toutes les fonctions liées à la fois aux matrices et au parcours de graphe. L'objectif sera alors de séparer les sources en trois fichiers distincts, contenant respectivement le programme principal (fichier `tp6.cpp`), les fonctions liées aux matrices (fichier `matrices.cpp`) et celles liées au parcours (fichier `parcours.cpp`). Après avoir recopié la correction fournie dans le dossier `Exercice1` sous le nom `tp6.cpp`, réalisez chacune des étapes qui suivent.

Module `matrices.cpp`

Créez le fichier `matrices.cpp`, qui contiendra toutes les fonctions liées au matrices qui se trouvent dans `tp6.cpp`. A l'issue de cette étape, la définition de ces fonctions ne devra plus être présente dans ce fichier.

Remarque : il sera nécessaire d'inclure le fichier `types.hpp`, de manière à ce que le compilateur connaisse les types à utiliser lors de la compilation de ce module.

Fichier `matrices.hpp`

En vous appuyant sur ce qui a été vu en cours, créez le fichier `matrices.hpp`, qui contiendra la définition des fonctions exportées par le module `matrices.cpp`. Mettez à jour en conséquence le fichier `tp6.cpp`.

Fichier `Makefile`

En vous appuyant sur ce qui a été vu en cours, créez le fichier `Makefile` de votre application. Utilisez ce fichier pour compiler cette application et vérifier son bon fonctionnement.

Fichiers `parcours.cpp` et `parcours.hpp`

De même que précédemment, créez les deux fichiers liés aux fonctions de parcours d'un graphe (module et fichier d'export), mettez à jour `tp6.cpp` et le fichier `Makefile` de l'application, compilez et testez celle-ci.

Modifications

Ajoutez une ligne de code au début de la fonction `main`, qui affiche à l'écran vos nom et prénom. Recompilez l'application et vérifiez que seul le fichier `tp6.cpp` est recompilé, en plus de l'étape d'édition de liens.

2 Affichage d'un graphe

Dans cet exercice, vous allez être amenés à compléter votre application pour qu'elle puisse afficher un graphe dans une fenêtre graphique. Dans un premier temps, recopiez les fichiers `.cpp`, `.hpp` et le fichier `Makefile` de l'exercice précédent dans le dossier `Exercice2` qui vous a été fourni.

Ajout des positions

Pour pouvoir afficher chaque sommet à un emplacement de la fenêtre d'affichage, le format des fichiers représentant une matice d'adjacence a été modifié de la manières suivante : en début de chaque ligne figure un coupel d'entiers qui représente les coordonnées auxquelles le sommet doit être affiché. Deux fichiers dans ce format vous sont fournis dans le dossier `Data` : `grapheC01.txt` et `grapheC02.txt` et seront utilisés dans le cadre de cet exercice.

De même, un type `coordonnees` a été défini dans le fichier `types.hpp` et un pointeur vers un tableau de coordonnées a été rajouté dans la structure de matrice d'adjacence.

```
/*
 * structure représentant une coordonnées cartesienne
 * dans une fenêtre d'affichage
 */
struct coordonnees {
  int x, y;
};

/*
 * structure représentant une matrice d'adjacence sous forme
 * de matrice creuse.
*/
struct MatriceAdjacence {
  int ordre; // nombre de sommets du graphe
  Maillon* *lignes;// tableau à allouer de taille "ordre", représentant les lignes de la matrice
  coordonnees *positions;// tableau à allouer de taille "ordre", représentant la position
                         //  du sommet dans la fenêtre d'affichage

};
```

1. Modifier la fonction `creerMatrice` de manière à ce qu'elle alloue et initialiser le tableau de coordonnées `positions` d'une matrice d'adjacence ;
2. Modifier la fonction `effacerMatrice` pour qu'elle efface ce tableau ;
3. Modifier la fonction `charger` pour qu'elle relise les coordonnées de chaque sommet et les range dans le tableau `positions` ;
4. Modifier la fonction `afficher` afin qu'elle affiche aussi la position de chaque sommet devant sa ligne.

Vous testerez chaque étape de ces modifications avec les fichiers fournis.

Ajout du module graphique

Le dossier Exercice2 contient les fichiers `graphique.cpp` et `graphique.hpp`, qui seront utilisés pour effectuer les affichages graphiques. Vous allez dans un premier temps ajouter ce module à votre application.

1. ajouter ce module à votre fichier `Makefile` de manière à ce qu'il soit considéré comme une dépendance de votre application ;
2. à la fin de la ligne d'éditions de liens, vous rajouterez `-lSDL2` qui précise au compilateur d'utiliser le code compilé de la librair `SDL` qui est utilisée ici pour gérer le graphisme ;
3. à la fin de la ligne de compilation du module `graphique.cpp`, vous ajouterez le texte `$(sdl2-config --cflags)`, qui permettra au compilateur de trouver les différents fichiers de la librairie `SDL` nécessaire à la compilation de ce module ;
4. décommentez la ligne suivante, qui figure en haut du fichier `tp6.cpp` :

   ```
   // ligne suivante à décommenter pour l'exercice 2
   //#define GRAPHIQUES
   ```

   Le fait de décommenter cette ligne permettra de compiler le code lié à la gestion de l'affichage graphique qui a été inséré dans la fonction `main`. **Ne modifiez rien dans le main ...**
5. Compilez votre application et testez là : une fenêtre graphique doit s'ouvrir, vide à ce stade. Pour quitter la fenêtre, vous pouvez soit appuyer sur une touche quelconque, soit cliquer sur la croix de fermeture de la fenêtre.

Dessin du graphe

Complétez la fonction `void drawGraph(MatriceAdjacence m)`, aux emplacements qui sont indiqués. Cette fonction a pour but de tracer le graphe, dont la matrice d'adjacence est fournie, dans la fenêtre. Vous disposez pour ce faire des deux fonctions graphiques suivantes :

- `void drawLine(coordonnees deb, coordonnees fin, SDL_Color c)` qui permet de tracer une ligne droite entre les points de coordonnées `deb` et `fin`, dans la couleur passée en troisième paramètre. Trois variables représentant les couleurs rouge, vert et bleu ont été définies dans le module, avec les noms `rouge`, `vert` et `bleu`.
- `void drawRect(coordonnees centre, int cote, SDL_Color col)` qui permet de tracer un rectangle dont le centre, le côté et la couleur
  Compilez et testez votre application.

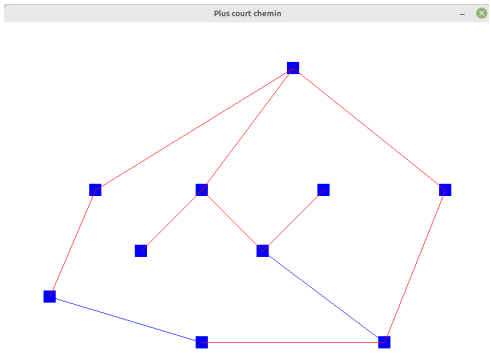
Dessin des chemins les plus courts

Après application de l'algorithme de Dijkstra, le tableau `parents` a été complété et permet de retrouver les chemins les plus courts entre chaque sommet et le sommet de départ. En vous inspirant que ce qui a été fait pour afficher les chemins trouvés dans la console (en mode texte), complétez les fonctions suivantes pour effectuer l'affichage graphique de ces chemins :

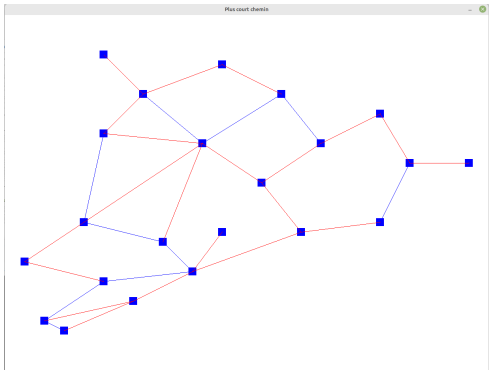
- `void drawPaths(MatriceAdjacence mat, int *parents)` : permet de lancer le tracé du chemin le plus court entre chaque sommet de graphe et le sommet de dpart ;
- `void drawPath(MatriceAdjacence mat, int sf, int *parents)` : permet de tracer **récursivement** le chemin entre le sommet d'indice `sf` et le sommet de départ.

Il est conseillé d'utiliser une autre couleur que celle utilisée pour tracer les arêtes du graphes, afin de bien les distinguer de celles constituant le chemin le plus court.

Exemples



(a) Aperçu du graphe du fichier `grapheC01.txt`



(b) Aperçu du graphe du fichier `grapheC02.txt`

FIGURE 1 – Exemples d'affichage des plus courts chemins (en rouge) pour les deux graphes fournis : depuis le sommet 4 pour le premier et depuis le dommet 0 pour le second.