Une classe graphe en C++

Ce document présente le code source d'une structure de graphe codée par liste d'adjacence et disponible librement sur le net. Il s'agit d'une classe C++ `C_graphe`.

Il ne s'agit ni d'un codage optimal, ni même du plus complet, ni du plus portable,... Il s'agit simplement d'une classe simple à comprendre et à manipuler afin d'être utilisée dans un contexte d'enseignement: proposer aux étudiants un code source qui leur permette de dépasser le cadre des structures de données afin de mieux s'intéresser à l'algorithmique dans les graphes.

Cette classe `C_graphe` peut donc être modifiée, adaptée, dérivée,... de manière à correspondre au besoin de chacun. L'objectif de cette classe serait atteint si chacun de ses utilisateurs la connaissait comme s'il l'avait faite et puisse la faire évoluer selon ses besoins.

Structure de la structure

Les classe `C_graphe`, `C_sommet` et `C_arete` définissent une structure de données permettant de manipuler des graphes creux ou peu denses. Cette stucture, dite de liste d'adjacences, consiste en un vecteur de sommets et une liste d'arête. Les sommets sont donc numérotés dans l'ordre de ce vecteur. Chaque sommet contient la liste des arêtes qui lui sont incidentes.

Si cette structure est pratiquement optimale pour l'accès mémoire, elle reste à privilégier pour les graphes peu denses. Dans le cas contraire, il faut mieux utiliser une autre implémentation. Par exemple, dans le cas des graphes complets ou des graphes à formes prédéfinies, il peut être utile de dériver la classe générale en surchargeant certaines fonctions.

Prototypes

Voici les données membres et les méthodes principales de la classe `C_graphe`.

```
class C_graphe{
 public:
  int nbsom;                  // Nombre de sommets dans le graphe
  int nbarete;                // Nombre d'arêtes dans le graphe
  vector<C_sommet*> v_som;    // Vecteur de pointeur sur sommets
  list<C_arete*> l_arete;     // Liste des arêtes du graphe

  C_graphe();
  C_graphe(int nbsom);
  ~C_graphe();
  void ajoute_arete(C_arete *e);
  int renv_nbarete();
  virtual void renv_poids_vecteur(vector<double> & fctobj);  // Retourne les poids de tous les sommets
  virtual int renv_numarete(int i, int j);   // Retourne le numéro de l'arête entre i et j
};
```


La classe `C_graphe` s'appuie sur les classes `C_sommet` et `C_arete`.

```
class C_sommet{
 public:
   double poids;
   double x,y;
   list<C_arete*> l_vois;

   C_sommet();
   ~C_sommet();
   void fixe_poids(double poids);
   void fixe_coordonnees(double x, double y);
   void ajoute_voisin(C_arete* arete);
};
```

```
class C_arete{
 private:
   int num;
   double poids;
   int v1,v2;
 public:
   C_arete();
   C_arete(int v1,int v2);
   void fixe_numero(int num);
   void fixe_poids(double poids);
   int retourne_numero();
   int retourne_voisin(int v);
   int retourne_v1();   int retourne_v2();
   virtual double retourne_poids();
};
```

Un objet de class `c_graphe` peut par exemple être instancié par le constructeur `C_graphe(int nbsom)` qui définit un graphe de `nbsom` sommets sans arête. On peut alors ajouter une arête en allouant un pointeur sur `C_arete`, en l'ajoutant à la liste `l_arete` et en l'ajoutant aux listes d'adjacence des sommets avec `ajoute_voisin(C_arete* arete)`.

Entrées/Sorties

Il n'existe pas réellement de format universel de codage d'un graphe dans un fichier. Disons que tous les formats disponibles sur le web ou fournis en sortie des générateurs de graphe sont soit des matrices d'adjacence, soit des listes d'arêtes.

La classe `C_graphe` possède deux fonctions de lecture d'un graphe: l'une à partir d'une matrice triangulaire supérieure et l'autre à partir d'un format de graphe proposé par G.Rinaldi. De plus, ces deux formats peuvent être obtenus en fichier de sortie. A vous d'étendre les entrées-sorties de cette classe à vos besoins.

```
void lit_matrice_triangulaire(istream &);
void ecrit_liste_adjacence(ostream &);
void lit_rinaldi_format(istream & in);
virtual void ecrit_rinaldi_format(ostream &,bool coord);
```

Représentation graphique

Les fonctions débutant par `ecrit_xfig` permettent la création d'un fichier texte au formayt du logiciel de dessin xfig. Ce fichier se compose d'une suite de lignes correspondant chacune à une ligne ou une figure simple (rond, carré,...). Le codage xfig se trouve facilement sur le web. A vous d'étendre les capacités de dessin de cette classe en explorant les possibilités de xfig.

```
void C_graphe::ecrit_xfig_trait(ostream &fic,int couleur,int epais,int forme,
                                int x1,int y1,int x2,int y2);
void C_graphe::ecrit_xfig_numerote_point(ostream &fic,int couleur,int epais,
                                         double x, double y, int aff);
void C_graphe::ecrit_xfig_nomme_point(ostream &fic,int couleur,int epais,
                                      double x, double y, char *aff);
void C_graphe::ecrit_xfig_note_valeur(ostream &fic,int couleur,int epais,
                                      double x, double y, double valeur);
virtual void C_graphe::ecrit_xfig(ostream &fic, int epais);
void C_sommet::ecrit_xfig(ostream &fic,int couleur,int epais,int forme);
```

Si votre graphe possède des coordonnées, il peut être représenté graphiquement directement. Le cas échéant, la fonction, `void attribue_coordonnees(int taillex,int tailley)` permet d'attribuer agréablement des coordonnées à vos sommets selon un algorithme simple.

Graphes dérivés

Il est assez facile d'obtenir des classes dédiées à des graphes particuliers. Il suffit de créer de nouveaux objets par héritage des classes `C_arete`, `C_sommet` et `C_graphe`. Différents exemples ont été codés ici: une grille, un graphe complet dont les poids des arêtes sont calculés selon la distance euclidienne et un graphe orienté. A chaque fois, le prototype est obtenu par héritage et par surcharge de certaines fonctions. A vous de faire vos propres adaptations: par exemple, en ajoutant des données membres aux sommets pour les algorithmes plus complexes...

En exemple, vous pourrez trouver les classes `C_Graphe_complet` et `C_Grille` qui permettent de manipuler ces deux types de graphes bien précis. Remarquer que la fonction `attribue_coordonnees` est surchargée de manière à pourvoir dessiner très proprement ces deux types de graphes.

Vous pourrez également trouvez la classe `C_Graphe_oriente` qui se base sur la classe `C_arc` (dérivée de la classe `C_arete`. Elle permet de manipuler des graphes orientés: représentation des arcs par des flèches, lecture d'une matrice en fichier d'entrée,...

Une classe plus particulière a été dérivée de `C_graphe`. Il s'agit de la classe `C_Graphe_euclidien`. On appelle usuellement euclidien un graphe complet représentant un ensemble de points placés sur un plan. On définit ainsi une arête entre chaque couple de sommets, associée à un poids calculé comme la distance euclidienne entre ses deux extrémités. Comme ce genre de graphes représente généralement une carte ou un plan, ils se composent d'un grand nombre de sommets et conséquemment d'un nombre très important d'arêtes. Comme le graphe complet et que le poids d'une arête peut être facilement calculé en fonction de ses extrémité, la classe `C_Graphe_euclidien` possède une liste d'arête nulle. Une surcharge de certaines fonctions permet de s'adapter facilement à ce changement.