Croquefer
Gaëtan
Groupe 2B
Année 2018/2019

Compte rendu du TP n°1 de « Graphes »

I)Problématique

Lorsque l'on travaille sur des réseaux critiques, il est nécessaire de s'assurer que la configuration actuelle du réseau permet bien à toutes les parties le composant de communiquer entre elles.

Afin de prouver la fiabilité du réseau, l'approche théorique consistera à ramener le problème rencontré à un problème de connexité de graphe.

On représentera donc notre réseau par le biais d'un graphe orienté G=(S,A) où :

- chaque sommet s ∈ S représente un nœud du réseau
- chaque arrête a ∈ A représente une connexion entre deux nœuds

Dans le cas présent, l'étude de la connexité du graphe représentant le réseau sera ramené à une recherche des composantes fortement connexes de celui ci. Ainsi, si l'on arrive à décomposer notre graphe en plusieurs composantes fortement connexe, l'on aura la preuve que toutes les parties du réseaux sont bel et bien connectées.

Si c'est un travail qui est assez simple à réaliser manuellement, il faudra au technicien qui souhaite informatiser ce travail une bonne connaissance des algorithmes. En effet, l'algorithme qui sera retenu ici est l'algorithme de Kosaraju-Sharir qui se décompose en 4 étapes distinctes :

1. Effectuer un parcours en profondeur en notant les dates de fin pour chaque sommet
2. Trier les sommets par ordre décroissant de date de fin
3. Réaliser le graphe dual du graphe étudié
4. Lancer un second parcours en profondeur en notant les sommets selon les dates de fin trouvées plus tôt

Afin de résoudre le problème le plus efficacement possible, j'ai choisi d'implémenter ledit algorithme en C++ afin de pouvoir gérer la mémoire de la manière la plus optimisée possible. De surcroît, pour trier les dates de fin après la première étape de l'algorithme, j'ai opté pour un algorithme de tri à bulle, d'une complexité de n^2, et ce par facilité d'implémentation.

II)Réalisation

Avant d'écrire l'algorithme lui même, il m'a fallu choisir la manière dont j'allais représenter en mémoire un graphe orienté. J'ai opté pour une représentation par matrice, qui, bien que plus gourmande en mémoire, offre une représentation plus explicite du graphe. Cette matrice est un attribut de ma classe/mon TAD « Graphe », et est complétée par un attribut de type entier correspondant à sa taille.

```
class Graphe
{
  private:
    int _nbSommet;
    std::vector<std::vector<bool> > _matrixLiens;
```

La classe « Graphe », comporte en outre des constructeurs permettant d'initialiser la matrice, une fonction permettant d'ajouter les arcs orienté entre les sommets.

```
Graphe();
Graphe(int nbSommet);
void ajouterLien(int s1, int s2);
```

Enfin, j'ai écris deux fonctions pour l'affichage, la première réalisant un affichage textuel de la matrice :

```
friend std::ostream& operator << (std::ostream& os, const Graphe& g);
```

Et la seconde permettant, uniquement sur linux et via l'outils « Graphviz », d'afficher graphiquement le graphe. Notez que dans le « main » de démonstration fourni avec ce compte rendu, l'appel vers cette fonction est commenté, afin de prévenir d'éventuelles erreurs de compilation/execution.

```
void afficherGraphe();
```

Pour finir, le cœur de l'algorithme, décomposé en les 4 étapes précitées est réalisé par un appel successif des fonctions suivantes :

```
std::vector<std::pair<int,int> > parcoursHistorique();
void triBulle(std::vector<std::pair<int, int> >& tab);
Graphe grapheAdjacent();
std::vector<std::vector <int> > parcoursProfondeur(Graphe adj, std::vector<std::pair<int, int> > datesFin);
```

Qui elles même utilisent les utilitaires suivant, afin de modulariser le code :

```
bool contains(std::vector<int> tab, int e);
int getElementIndex(std::vector<std::pair<int, int> > tab, int e);
```

Ces utilitaires permettent respectivement de vérifier si un élément donné appartient à un vecteur précis, et de renvoyer l'index d'un élément dans un vecteur donné.

Pour plus de détail sur l'algorithme, vous trouverez en annexe le code des fonctions.

III)Bilan

Lors de ce TP sur la connexité des réseaux critiques, j'ai pu découvrir une des nombreuses applications concrète de la théorie des graphes. Si l'utilité pratique de cette théorie n'est pas si évidente à cerner de prime abord, il s'avère que dans bien des domaines, et notamment en informatique, nombre de problèmes peuvent aisément être ramené à un simple problème commun de graphe.

En outre, j'ai également appris les différentes manière de stocker des graphes programmatiquement, et, fort de cette expérience, j'ai pu apprendre à les manipuler via des algorithmes.

Pour conclure, je retiendrais l'algorithme de Kosaraju-Sharir comme l'une des implémentations les plus difficiles qui m'ait été donnée de réaliser, et j'imagine déjà les applications qui me sera possible d'en tirer.

Annexe

```
vector<pair<int, int> > Graphe::parcoursHistorique()
{
  int i=0, j=0, k=0, sommet=0;
  vector<int> datesDeb;
  vector<pair<int, int> > datesFin;
  vector<int> chemin;
  bool arret=false;

  datesDeb.reserve(this->_nbSommet);
  datesDeb.resize(this->_nbSommet);
  datesFin.reserve(this->_nbSommet);
  datesFin.resize(this->_nbSommet);

  do
  {
    if(datesDeb[sommet]==0)
    {
      i++;
      datesDeb[sommet]=i;
      chemin.push_back(sommet);
    }
    for(j=0; j<this->_nbSommet; j++)
    {
      if(this->_matrixLiens[sommet][j]==true && datesDeb[j]==0)
      {
        break;
      }
    }

    if(j!=this->_nbSommet)
    {
      sommet=j;
    }
    else
    {
      i++;
      datesFin[sommet].second=i;
      datesFin[sommet].first=sommet;
      chemin.pop_back();
```

```
      if (chemin.size()>0)
      {

        sommet=chemin[chemin.size()-1];
      }
      else
      {
        for(k=0; k<this->_nbSommet; k++)
        {
          if(datesDeb[k]==0)
          {
            break;
          }
        }
        if(k!=this->_nbSommet)
        {
          sommet=k;
        }
        else
        {
          arret=true;
        }
      }
    }

  } while(!arret);
  return datesFin;
}
```

```cpp
vector<vector <int> > Graphe::parcoursProfondeur(Graphe adj, vector<pair<int,int> > datesFin)
{
  vector<vector <int> > result;
  vector <int> sommetParcourus;
  vector <int> sommetResultats;
  bool alreadyBrowsed;
  int i=0,sommetParcourusCount=0;


  //Tant que l'on a pas parcourus tous les sommets
  while(sommetParcourusCount!=(int)datesFin.size())
  {

    //Ajout du sommet au tableau temporaire et au tableau résultat, s'il n'y est pas déjà
    //Incrémentation du compteur de sommet parcourus le cas échéant
    if(!contains(sommetResultats,datesFin[i].first))
    {
      sommetParcourusCount++;
      sommetResultats.push_back(datesFin[i].first);
      sommetParcourus.push_back(datesFin[i].first);
    }

    //Elimination des liens qui vont au sommet
    for(int j=0; j<this->_nbSommet; j++)
    {
      adj._matrixLiens[j][datesFin[i].first]=false;
    }

    //Parcours de tous les sommets du graphes
    alreadyBrowsed=true;
    for(int j=0; j<this->_nbSommet; j++)
    {
      //Si un lien part de ce sommet
      if(adj._matrixLiens[datesFin[i].first][j])
      {
        //On vérifie si la cible du lien a déjà été visitée
        alreadyBrowsed=contains(sommetResultats, j);
        if(!alreadyBrowsed)
        {
```

```
          //On assigne i à l'index de la cible dans le tableau datesFin et on quitte la boucle
          i=getElementIndex(datesFin,j);
          break;
        }
      }
    }

    //Si l'on ne trouve pas de cible valide
    if(alreadyBrowsed)
    {
      //Si c'est un graphe unaire on ajoute nos sommets au résultat, on clear nos tableaux et on recommence pour la datesFin suivante
      if(sommetParcourus.size()==1)
      {
        result.push_back(sommetResultats);
        sommetParcourus.clear();
        sommetResultats.clear();
        i++;
      }
      //Sinon on revient en arrière d'un cran pour chercher d'autres liens sur le sommet d'avant
      else
      {
        sommetParcourus.pop_back();
        i=getElementIndex(datesFin,sommetParcourus[sommetParcourus.size()-1]);
      }
    }
  }

  //On ajoute notre dernier résultat et on return
  result.push_back(sommetResultats);
  return result;
}
```