IFT339 - Automne 2014 travail 2 (partie 1)

Utiliser les bons conteneurs, et les fichiers binaires…

Dans ce cours, nous pratiquons la bonne utilisation des conteneurs abstraits et des structures de données qui nous servent à les implanter. L'objectif ici est de faire des conteneurs commodes mais aussi efficaces que possible en temps et en espace. Trop souvent, on utilise ces conteneurs dans des "problèmes-jouets", soit des problèmes qui n'ont pas l'envergure requise pour traiter d'immenses quantités de données. Il faut être capable de traiter ces données, et très souvent les techniques apprises pour les petits problèmes ne se transfèrent pas facilement sur les gros. Par exemple, si on veut trouver le chemin le plus court entre deux points dans un graphe avec l'algorithme de Dijkstra, c'est très rapide. Mais si on doit prendre plusieurs minutes à lire tout un graphe d'un million de nœuds pour trouver un seul chemin, ce n'est pas très utile. Il n'est pas toujours nécessaire de lire tout le graphe, on peut ne lire que les informations qui sont utiles, soit celles des nœuds que nous voulons explorer.

Dans cette version, un format de données "binaires" vous est proposé. Les données sont en mode de représentation interne plutôt que sérialisées. Ainsi, si le poids d'un arc est 940.817, il n'occupera pas 8 caractères dans le fichier (les 7 de la sérialisation plus un délimiteur). Il n'occupera que le 4 octets du float qui le représente: 43346B44 en little endian ou 446B3443 en big endian. Notez que les Macs et les PC, basés sur des machines Intel, ont une architecture little endian et que les Sun ont une architecture big endian.

On peut organiser le fichier pour aller directement chercher l'information concernant le 238e nœud sans avoir à lire celle concernant les 237 qui viennent avant. Il suffit de se positionner à un endroit précis du fichier et de commencer à cet endroit la prochaine opération de lecture. Voir les notes à la page 27.

La meilleure stratégie pour le programme est de mettre toute l'information concernant un noeud dans une structure appropriée au moment où on doit lire ce noeud. Lorsqu'on a besoin d'accéder à un nœud, on peut lire cette information à la bonne place dans le fichier, et la conserver au cas où on en aurait besoin plus tard dans le programme. Dès la création du graphe, on pourrait faire un lien avec un flot, et à chaque fois que le graphe a besoin d'aller lire dans le fichier, il pourrait utiliser ce flot. Quelque chose comme ceci:

```
class graphe{
private:
   struct noeud{  //description de toutes les composantes d'un noeud
      };
   map<uint32_t,noeud> lesNoeuds;   //les noeuds deja lus
   uint32_t nbNOEUDS;               //le nombre de noeuds
   ifstream DATA;                   //le flot d'entree
   uint32_t DEBUT;                  //debut de la partie fixe
   void lire_noeud(uint32_t);
   void lire(uint16_t&);   //fonctions utilitaires de lecture binaire
   void lire(uint32_t&);   //qui dependent de l'architecture
   void lire(float&);
   graphe(const graphe&)=delete;              //copieur et
   graphe& operator=(const graphe&)=delete;   //affectateur désactivés
public:
   graphe(string);                            //constructeur
   ~graphe();                                 //destructeur
   uint32_t size()const;                      //nombre de noeuds dans le graphe
   void afficher_noeud(uint32_t);  //afficher toutes les infos sur un noeud
   ...
};
```

Dans cette représentation, on a choisi de toujours travailler avec le numéro des nœuds. On n'utilisera les noms que lorsqu'on voudra afficher l'information sur un noeud.

Votre travail...

Je vous donnerai au retour de la relâche le travail spécifique à faire. D'ici là, je vous demande d'être capable d'aller lire et afficher toutes les informations concernant un nœud. Une fois le graphe créé, on peut utiliser la fonction afficher_noeud qui affiche à l'écran toutes les informations contenues dans le fichier concernant ce nœud. Cette fonction peut à son tour appeler lire_noeud qui ira au besoin dans le fichier chercher cette information. Pour vous préparer à recevoir l'énoncé du travail pratique 2 (partie 2) commencez par faire cette première partie, dont le programme principal devrait ressembler à :

```
uint32_t numero;
graphe USA("grUSA_32.ibin");
cout<<"Le graphe contient "<<USA.size()<<noeuds<<endl;
for(;;){
   cin>>numero;
   USA.afficher_noeud(numero);
   }
```

Remise du travail

Cette première partie n'est pas à remettre, mais vous devriez viser l'avoir finie pour le retour de la relâche, pour être prêts à faire les laboratoires et travaux qui utiliseront ces fichiers.

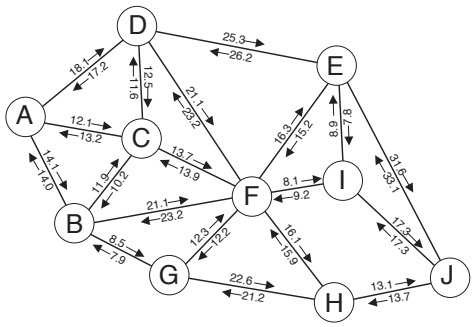
La description d'un graphe dans un fichier

Dans cette description, "numéro" est un uint32_t. On ne peut pas utiliser des types d'objets dont on ne connaît pas exactement la dimension, car c'est la base de la lecture. Ainsi, on peut être certain que l'uint32_t occupe 32 bits. Dans le fichier, chaque nœud occupe une partie fixe (de longueur égale pour tous les nœuds) et une partie variable (de longueur variable pour chaque nœud). La partie fixe comprend l'adresse du début de la partie variable du nœud concerné, la latitude et la longitude du nœud, et quatre numéros de nœuds (pour usage futur!). La partie variable comprend la description des arcs sortant du nœud concerné et le nom du nœud.

```
0000: 65 78 65 6D 70 6C 65 20 31 30 20 01 00 25 01 00  exemple 10 ..%..
0010: 00 00 00 60 C1 00 00 C0 40 04 00 00 00 05 00 00  ...`....@.......
0020: 00 00 00 00 00 00 00 00 00 43 01 00 00 00 00 20  .........C..... 
0030: C1 00 00 00 C0 00 00 00 00 00 00 00 00 00 00 00  ................
0040: 00 00 00 00 00 69 01 00 00 00 00 E0 C0 00 00 80  .....i..........
0050: 40 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  @...............
0060: 00 8F 01 00 00 00 00 E0 C0 00 00 10 41 00 00 00  ............A...
0070: 00 00 00 00 00 00 00 00 00 00 00 00 00 B5 01 00  ................
0080: 00 00 00 00 41 00 00 E0 40 00 00 00 00 00 00 00  ....A...@.......
0090: 00 03 00 00 00 00 00 00 00 DB 01 00 00 00 00 00  ................
00A0: 00 00 00 00 00 08 00 00 00 07 00 00 00 02 00 00  ................
00B0: 00 06 00 00 00 19 02 00 00 00 00 C0 C0 00 00 A0  ................
00C0: C0 00 00 00 00 00 00 00 00 01 00 00 00 00 00 00  ................
00D0: 00 37 02 00 00 00 00 E0 40 00 00 E0 C0 09 00 00  .7......@.......
00E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 55 02 00  .............U..
00F0: 00 00 00 00 41 00 00 00 40 00 00 00 00 00 00 00  ....A...@.......
0100: 00 00 00 00 00 00 00 00 00 73 02 00 00 00 00 A0  .........s......
0110: 41 00 00 C0 C0 00 00 00 00 00 00 00 00 00 00 00  A...............
0120: 00 00 00 00 00 03 00 03 00 00 00 CD CC 90 41 02  ..............A.
0130: 00 00 00 9A 99 41 41 01 00 00 00 9A 99 61 41 02  .....AA......aA.
0140: 00 41 20 04 00 05 00 00 00 CD CC A8 41 06 00 00  .A ........A...
0150: 00 00 00 08 41 02 00 00 00 66 66 3E 41 00 00 00  ....A....ff>A...
0160: 00 00 00 60 41 02 00 42 20 04 00 05 00 00 00 33  ...`A..B ......3
0170: 33 5B 41 03 00 00 00 9A 99 39 41 01 00 00 00 33  3[A......9A....3
0180: 33 23 41 00 00 00 00 33 33 53 41 02 00 43 20 04  3#A....33SA..C .
0190: 00 05 00 00 00 CD CC A8 41 04 00 00 00 66 66 CA  ........A....ff.
01A0: 41 02 00 00 00 00 00 48 41 00 00 00 00 9A 99 89  A......HA.......
01B0: 41 02 00 44 20 04 00 05 00 00 00 33 33 73 41 08  A..D ......33sA.
01C0: 00 00 00 9A 99 F9 40 03 00 00 00 9A 99 D1 41 09  ......@.......A.
01D0: 00 00 00 CD CC FC 41 02 00 45 20 07 00 04 00 00  ......A..E .....
01E0: 00 66 66 82 41 08 00 00 00 9A 99 01 41 07 00 00  .ff.A.......A...
01F0: 00 CD CC 80 41 06 00 00 00 33 33 43 41 03 00 00  ....A....33CA...
0200: 00 9A 99 B9 41 02 00 00 00 66 66 5E 41 01 00 00  ....A....ff^A...
0210: 00 9A 99 B9 41 02 00 46 20 03 00 05 00 00 00 CD  ....A..F .......
0220: CC 44 41 07 00 00 00 CD CC B4 41 01 00 00 00 CD  .DA.......A.....
0230: CC FC 40 02 00 47 20 03 00 05 00 00 00 66 66 7E  ..@..G .....ff~
0240: 41 06 00 00 00 9A 99 A9 41 09 00 00 00 9A 99 51  A.......A......Q
0250: 41 02 00 48 20 03 00 05 00 00 00 33 33 13 41 04  A..H .....33.A.
0260: 00 00 00 66 66 0E 41 09 00 00 00 66 66 8A 41 02  ...ff.A....ff.A.
0270: 00 49 20 03 00 04 00 00 00 66 66 04 42 08 00 00  .I .....ff.B...
0280: 00 66 66 8A 41 07 00 00 00 33 33 5B 41 02 00 4A  .ff.A....33[A..J
0290: 20
```

Au début du fichier, une string suivie d'un blanc donne le nom du graphe. Un uint32_t sérialisé suivi d'un blanc donne le nombre de nœuds, et un uint16_t binaire valant 1 peut servir à identifier l'architecture de la machine qui a créé ce fichier. Tout le reste du fichier est binaire. D'abord la partie fixe, puis la partie variable (une entrée dans chaque partie pour chaque nœud du graphe).

La partie fixe est de 28 octets pour chaque noeud : adresse de la partie variable (uint32_t), latitude et longitude (deux float), zones-QT (quatre numéros de nœuds, donc 16 octets, pour usage futur).



La partie variable d'un nœud débute par un uint16_t donnant le nombre d'arcs. Suivent les arcs eux-mêmes: le numéro du nœud de destination, et un float pour le poids, donc 8 octets pour chaque arc. Vous avez ici une illustration du graphe exemple.ibin et le contenu intégral de ce fichier. Les nœuds sont en ordre lexicographique de noms, à la fois dans la partie fixe comme dans la partie variable. Pour lire les informations sur le nœud 6 par exemple, il faut d'abord localiser sa partie fixe: 0x0000000d+ 6*28= 0x000000b5. On y trouve l'adresse de la partie variable de ce nœud: 0x00000219, la lattitude: 0xc0c00000, la longitude: 0xc0a00000, les quatre numéros réservés pour usage futur: 0, 0, 1 et 0. Dans la partie variable, à partir de l'adresse 0x00000291, on trouve le nombre d'arcs (3), puis les arcs (5, 0x4144cccd=12.3) (7,0x41b4cccd=22.6) (1,0x40fccccd=7.9). Ensuite le nombre de caractères plus un du nom: (0x0002 = 2), puis les caractères eux-mêmes (0x4720 = "G "). 0x4144cccd est la représentation de 12.3 dans un float.

Je déposerai les fichiers dans les deux répertoires publics et je vous en enverrai aussi une copie via le site d'envoi de gros fichiers de l'Université (le graphe d'un million de nœuds fait presque 300 megs!)

Jean Goulet