Exercice 25 - Graphes et STL

Un graphe dirigé peut être défini comme un couple $G = (V, E)$ où V est un ensemble de *sommets* et E un ensemble de couples $(i, j) \in V \times V$ que l'on appelle *arcs*. Un sommet $j \in V$ est dit *successeur* d'un sommet $i \in V$ si $(i, j) \in E$. Un sommet $j \in V$ est dit *prédécesseur* d'un sommet $i \in V$ si $(j, i) \in E$.

Remarque : en anglais, "sommet" se dit "vertex" (pluriel : "vertices"), et "arc" se dit "edge".

Exemple :
- $V = \{0, 1, 2, 3, 4, 5, 6\}$.
- $E = \{(2, 0), (2, 1), (2, 3), (3, 1), (1, 4), (6, 0), (6, 5), (6, 3), (0, 4), (4, 0), (4, 5), (4, 6)\}$.
- L'ensemble des successeurs du sommet 2 est $\{0, 1, 3\}$. L'ensemble des prédecesseurs de 4 est $\{0, 1\}$.

Pour manipuler un graphe dans un programme, une des structures de données les plus utilisées est la liste d'adjacence. Elle consiste en un tableau dont l'entrée i donne la liste des successeurs du sommet i.

Le but de cet exercice est de se familiariser avec la STL en implémentant une classe de graphe.

Préparation : Créer un projet vide et ajouter trois fichiers `graph.h`, `graph.cpp`, et `main.cpp`. Les situations exceptionnelles seront gérées en utilisant la classe d'exception suivante (à recopier dans le fichier `graph.h`) :

```
#if !defined(_GRAPH_H)
#define _GRAPH_H
#include<string>
#include<stdexcept>
using namespace std;
class GraphException : public exception {
 string info;
public:
 GraphException(const string& i) noexcept :info(i){}
 virtual ~GraphException() noexcept {}
 const char* what() const noexcept { return info.c_str(); }
};
#endif
```

Définir la fonction principale `main` dans le fichier `main.cpp`. S'assurer que le projet compile correctement. Implémentez une classe graphe qui utilise un objet `vector<list<unsigned int> >` pour représenter la liste d'adjacence d'un graphe dont les n sommets sont identifiés par les nombres de $\{0, 1, ..., n-1\}$. Implémentez toutes les méthodes de l'interface suivante :

```
#include<list>
#include<vector>
#include<iostream>
#include<string>
using namespace std;
class Graph {
 vector<list<unsigned int> > adj;
 string name;
public:
 Graph(const string& n, unsigned int nb);
 const string& getName() const;
 unsigned int getNbVertices() const;
 unsigned int getNbEdges() const;
 void addEdge(unsigned int i, unsigned int j);
 void removeEdge(unsigned int i, unsigned int j);
 const list<unsigned int>& getSuccessors(unsigned int i) const;
 const list<unsigned int> getPredecessors(unsigned int i) const;
};
ostream& operator<<(ostream& f, const Graph& G);
```

Le constructeur de la classe `Graph` prend en argument le nom du graphe ainsi que le nombre de sommets le constituant (qui n'évoluera pas au cours de la vie du graphe). Exploitez au mieux les algorithmes de la STL.

Exemple :

```
try{
  Graph G1("G1",5); cout<<G1;
  G1.addEdge(0,1); G1.addEdge(0,2); G1.addEdge(1,2); G1.addEdge(1,3);
  G1.addEdge(1,4); G1.addEdge(3,0);
  cout<<G1;
}catch(exception e){ std::cout<<e.what()<<"\n"; }
```

Affichage obtenu :

```
graph G1 (5 vertices and 0 edges)
0:
1:
2:
3:
4:
graph G1 (5 vertices and 6 edges)
0:1 2
1:2 3 4
2:
3:0
4:
```