Rapport du projet C++ : Partie 1

Gabriel Dos Santos, Raphael Marouani

20 novembre 2020

Table des matières

1 Introduction

Le but principal de ce projet est de développer un système de codage et de décodage basé sur le code de Huffmann. Le codage d'Huffmann a pour but de coder un texte en binaire préfixé qui consiste à coder chaque lettre par un mot sur 0, 1 (toujours le même pour une lettre). Étant donné le pourcentage d'occurrence de chaque lettre dans le texte à coder, l'algorithme de codage des lettres est le suivant :

— Initialement, chaque lettre est un arbre binaire ramené à un sommet étiqueté par la proportion d'occurrences de cette lettre dans le texte. Tant qu'il y a plus d'un arbre, réaliser les opérations suivantes :

1. Considérer `A1` et `A2` les deux arbres dont les racines portent les plus petites étiquettes `e1` et `e2`.
2. Construire un nouvel arbre `A` dont la racine `r` a pour fils les racines de `A1` et `A2`.
3. La racine `r` est étiqueté par `e1 + e2`.

— Pour chaque noeud de l'arbre final, l'arête vers son fils gauche est étiquetée 0 et celle vers son fils droit 1.

— Le code associé à une lettre est le mot binaire composé des étiquettes sur les arêtes entre la racine de l'arbre final et la feuille étiquetée avec cette lettre.

2 Utilisation de l'application

2.1 Contenu de l'archive du projet

L'archive que vous avez reçu contient les fichiers suivants :

- Le rapport du projet, que vous êtes en train de lire.
- Le fichier `README.org`, qui résume le projet et explique rapidement comment se servir de l'application.
- Le fichier `listing.pdf`, qui est la documentation complète de la partie réalisée par Gabriel.
- Le fichier `Makefile`, qui contient les outils de compilation et d'exécution de l'application.
- Le dossier `src`, qui contient les fichiers sources du projet. Celui-ci est divisé en trois sous-dossiers :
  - `bin`, qui contient le fichier `main1.cpp` avec le programme de test.
  - `lib`, qui contient les fichiers des classes `Sommet` et `ArbreB`.
  - `test`, qui contient les fichiers de la classe de test `PartOneTests`, dont les méthodes sont utilisées dans le programme de test.

2.2 Configuration requise

Afin de pouvoir utiliser cette application, vous aurez besoin des programmes suivants :

- Le compilateur `g++` (version 7.5.0 ou supérieure). Alternativement, vous pouvez utiliser le compilateur `clang++` (testé avec la version 9.0.0 uniquement).
- Le standard C++ 17 (le minimum requis est le standard C++ 11 mais il est préférable de compiler en C++ 17).

2.3 Compilation et exécution avec l'interface en ligne de commande

Ce projet peut être compilé et exécuté dans deux modes différents : soit en mode "debug", soit en mode "release". Le mode "debug" compile le programme avec les flags `-Wall -Wextra -g` afin de permettre à l'utilisateur une meilleure compréhension des erreurs et avertissements affichés par le compilateur, ainsi que d'exécuter le programme avec des outils comme Valgrind ou gdb. Le mode "release" compile quant à lui avec le flag d'optimisation `-O2` afin de rendre le binaire moins lourd et plus rapide a l'exécution.
Les commandes qui suivent sont à taper à la racine du projet.

Afin de compiler et/ou exécuter le programme en mode "debug", vous pouvez utiliser les commandes du Makefile (fourni dans l'archive du projet) de la manière suivante :

```
# Pour compiler uniquement :
make build
# Pour compiler et executer :
make run
```

Vous pouvez également utiliser les outils de debugging gdb et Valgrind :

```
make gdb
make vg
```

Si vous souhaitez compiler en mode "release", les commandes à taper sont les suivantes :

```
# Compilation uniquement :
make build_release
# Compilation et execution :
make run_release
```

Dans l'éventualité où vous souhaiteriez compiler et exécuter le programme manuellement, voici les commandes dont vous aurez besoin :

```
mkdir -p target/release
g++ src/bin/main1.cpp src/lib/Sommet.cpp src/lib/ArbreB.cpp src/
    test/PartOneTests -std=c++17 -O2 -o target/release/main1
./target/release/main1
```

Enfin, le Makefile met à votre disposition d'autres commandes utilitaires que voici :

```
# Pour supprimer les fichiers executables :
make clean
# Pour generer une archive du projet :
make archive
```

2.4 Compilation et exécution avec l'interface graphique (Qt)

A faire par Raphael

3 Description du travail réalisé

3.1 Répartition des taches

Nous avions initialement prévu de travailler de manière équitable sur l'ensemble de la partie 1 du projet. Le but était que chacun implémente une partie des classes et des fonctionnalités demandées dans l'énoncé, afin que chaque membre du binome comprenne les choix d'implémentation et la façon dont est écrit le programme.

Devant le manque de réponses et de retours de Raphael, Gabriel a finalement réalise l'intégralité de l'implémentation des classes `Sommet`, `ArbreB`, `PartOneTests` ainsi que le programme de test `main1.cpp`. La rédaction de la documentation et des commentaires dans le code source, du README et d'une partie de ce rapport ont également été réalisées par Gabriel. Raphael s'est chargé de l'interface graphique et a rédigé les parties qui y correspondent dans ce compte-rendu.

3.2 Description des classes

3.2.1 Classe Sommet

La classe `Sommet` représente un noeud d'un arbre binaire. Par choix d'implémentation et pour faciliter les parties suivantes du projet, la classe `Sommet` contient déjà des attributs pour stocker un caractère et sa fréquence. Chaque instance de cette classe contient également deux pointeurs, qui permettent une implémentation similaire à une liste chaînée. Les attributs de la classe sont les suivants :

- `char m_Data` : le caractère affecté au Sommet.
- `double m_Freq` : la fréquence du caractère affecté au Sommet. L'énoncé demandant que la fréquence soit représentée par un pourcentage, la fréquence est représentée par un flottant à double précision.
- `Sommet* m_Left` : le fils gauche affecté au Sommet. C'est un pointeur vers un autre Sommet.
- `Sommet* m_Right` : le fils droit affecté au Sommet. C'est un également un pointeur vers un autre Sommet.

3.2.2 Classe ArbreB

La classe `ArbreB` représente un arbre binaire. Elle ne possède qu'un seul attribut, un pointeur sur la racine de cet arbre :

— `Sommet* m_Root` : la racine de l'arbre, depuis laquelle on peut accéder a tous les sommets qui le compose grâce à l'implémentation en liste chaînée de la classe `Sommet`.

3.2.3 Classe PartOneTests

La classe `PartOneTests` ne sert qu'à implémenter des tests qui valident ou non les fonctionnalités des classes `Sommet` et `ArbreB`. Les méthodes qui y sont implémentées sont utilisées dans le programme de test afin de tester de manière précise l'application. Elle possède des attributs qui ne servent qu'à suivre l'état des tests :

— `static unsigned int total_tests` : le nombre total de tests qui ont été écrit, ce qui permet de déterminer si des tests n'ont pas été exécutés (*skipped*).
— `unsigned int tests_run` : le nombre de tests qui ont été exécutés.
— `unsigned int tests_failed` : le nombre de tests qui ont échoués.

3.3 Interface en ligne de commande

Pour l'interface en ligne de commande, Gabriel a écrit un programme de test (`src/bin/main1.cpp`) qui utilise les méthodes de la classe de test `PartOneTests`. Le but était de décomposer chaque fonctionnalité des classes de façon à pouvoir les tester le plus précisément possible. Chaque test est appelé dans la fonction (`main`), puis evalue par une boucle (`if`). Le résultat du test est alors affiché dans le terminal. L'ordre dans lequel les tests sont exécutés est également logique, les tests plus avancés s'appuient sur des fonctionnalités précédemment validées. L'affichage dans le terminal est inspiré de bibliothèque de tests unitaires telle que JUnit pour le Java.

```
[root@gnix project]# make run
mkdir -p target/debug/
mkdir -p target/release/
g++ src/bin/main1.cpp src/lib/Sommet.cpp src/lib/ArbreB.cpp src/test/PartOneTests
.cpp -std=c++17 -Wall -Wextra -g -o target/debug/main1
./target/debug/main1

        Running tests for class Sommet...
Test for default constructor:           PASSED!
Test for parameterized constructor:     PASSED!
Test for copy constructor:              PASSED!
Test for setting values:                PASSED!
Test for overload of '=' operator:      PASSED!
Test that copies are not linked:        PASSED!

        Running tests for class ArbreB...
Test for default constructor:           PASSED!
Test for parameterized constructor:     PASSED!
Test for constructor from Sommet:       PASSED!
Test for copy constructor:              PASSED!
Test for overload of '=' operator:      PASSED!
Test that copies are not linked:        PASSED!
Test for inserting Sommet:              PASSED!
Test for inserting existing Sommet:     PASSED!
Test for finding a character:           PASSED!
Test for not finding a character:       PASSED!
Test for removing Sommet (leaf):        PASSED!
Test for removing Sommet (1 child):     PASSED!
Test for removing Sommet (2 children):  PASSED!
Test for fusing two ArbreBs:            PASSED!
Test for decomposing one ArbreB:        PASSED!
Test for finding a character (BFS):     PASSED!
Test for not finding a character (BFS): PASSED!

Tests run: 23,  Tests failed: 0,  Tests skipped: 0
Status: BUILD SUCCESS
[root@gnix project]# scrot -q 100 -u
```

FIGURE 1 – Voici ce que vous devriez obtenir dans le terminal en tapant la commande 'make run' à la racine du projet.

3.4 Interface graphique

A faire par Raphael