Es wurden keine Einträge für das Inhaltsverzeichnis gefunden.

1 T9

Folgend ist die Dokumentation der Aufgabe T9 Converter angeführt, welcher Strings in T9 Codes und visa versa konvertiert.

1.1 Lösungsidee

Folgend ist die Lösungsidee für die Aufgabenstellungen des T9 Konverters angeführt.
Alle Methoden sollen in einer Klasse gekapselt werden (T9Converter). Wiederverwertbare Logik soll in eigene private Methoden ausgelagert werden. Die Klasse soll Konstruktoren zur Verfügung stellen die eine Pfadangabe für das Wörterbuch und den Beispieltext verlangen.
Im Konstruktor soll das Wörterbuch und der Beispieltext verarbeitet werden.

Da die Aufgabenstellungen mehrere Indexes verlangen, sollen diese als private Member der Klasse definiert werden.

Die Test sollen wieder mit CUTE und BOOST erfolgen.

1.1.1 word2number

Bei diesem Algorithmus soll ein String in seine T9 Code Darstellung konvertiert werden.
Dies sollte einfach sein, da der Source auch ein Mapping für die t9 Codes haben soll, wo die einzelnen Ziffern auf ihre T9 Codes abgebildet sind.
Ungültige Character sollen erkannt, eine Meldung auf die Konsole ausgegeben werden und in dem weiteren Verlauf ignoriert werden.

1.1.2 number2strings

Bei dieser Implementierung sollen alle möglichen Kombinationen eines T9 Codes ermittelt und ausgeben werden. Hierbei soll wie folgt vorgegangen werden.

1. Ermittle alle Character Sets die verwendet werden (einzelne T9 Code <-> String mappings)
2. Initialisiere ein temporäres Set mit einem Element (Lerrstring)
3. Iteriere über die ermittelten Character Sets und conatiniere alle Strings des temporären Sets mit den Characters des Sets
4. Ermittle ob sich bereits Strings mit adäquater Länge im temporären Set befinden
    a. JA: Füge sie in das Result Set hinzu
    b. NEIN: Ignoriere diese Strings
5. Fahre damit fort solange es noch Character Sets gibt.

Mit diesem Algorithmus werden alle Kombinationen erfasst auch diese, die die nicht Strings mit adäquater Länge produzieren. Dies ist allerdings erforderlich um alle benötigten Permutationen zu erfassen.
Daher ist anzumerken dass dieser Algorithmus extrem ineffizient ist da er eine Laufzeitkomplexität von ungefähr O(n^4) besitzt. Da sich hier vier verschachtelte Schleifen befinden.

1.1.3 number2words

In dieser Methode soll der gegebene T9 Code in seine Worte übersetzt werden, die sich in einem Index befinden. Daher soll ein Index in folgender Datenstruktur aufgebaut werden.

```
unordered_map<long long, unordered_set<string>>
```

Die wird die Zugriffszeiten erheblich verringern da keine Permutationen mehr berechnet werden müssen. Ein Nachteil ist aber das nur Worte unterstützt werden, die m Index vorhanden sind, daher wird man auf den Umfang des Wörterbuches begrenzt sein.

1.1.4 number2wordsByLength

In dieser Methode soll ein t9 Code seine möglichen, vom Index unterstützten Worte übersetzt werden. Hierbei soll jedoch die Länge des T9 Codes berücksichtigt werden, die ebenfalls auch die Länge des Wortes vorgibt. Daher soll eine Datenstruktur aufgebaut werden, die die Längen des T9 Code (=Wortlänge) auf die möglichen T9 Codes mit den dazugehörigen Wörtern mapped.

```
unordered_map<int, unordered_map<long long, unordered_set<string>>>
```

Bei dieser Datenstruktur wird das Laufzeitverhalten erneut verbessert weil hier bereits die Länge des T9 Codes berücksichtigt wird und daher die Zugriffe beschleunigt werden, da eine gemappter Länge nur die T9 Codes und natürlich auch die dazugehörigen Wörter beinhaltet, die auf diese Länge passen. Dadurch wird ein Baum aufgebaut dessen Knoten die Längen sind, die wiederum einen Baum darstellen.

1.1.5 number2PrefixWord

In dieser Methode soll ein T9 Code auf all seine möglichen Kombinationen die noch folgen könnten abgebildet werden. Also alle Wörter, die sich bei weiterer Eingabe von T9 Codes bilden lassen könnten.
Da bereits eine Datenstruktur verlangt wird, die die Längen auf die t9 Codes abbildet, kann diese Struktur in dieser Methode wiederverwendet werden um die möglichen Kombinationen von Wörter zu ermitteln.
Es soll wie folgt vorgegangen werden:

1. Iteriere über den Längen Index und suche alle T9 Codes, die länger sind als der gegebene

   a. NEIN: Fahre mit den Iterieren fort
   b. JA: Iteriere über diese T9 Codes und ermittle die Codes die den gegebenen Code als Prefix haben
      i. NEIN: Fahre mit den Iterieren fort
      ii. JA: Speichere diese in einer Collection

Es sind zwar mehrere Iterationen notwendig um zu den Wörtern zu kommen, jedoch muss nicht über die Wörter selbst iteriert werden. Da es sich um einen Index handelt ist auch hier eine performante Laufzeit zu erwarten.

1.1.6 numberPrefix2sortedWords

In dieser Methode sollen die Wörter für einen T9 Code ermittelt werden, die sich in Wörterbuch und einen Beispieltext befinden. Diese sollen nach der Häufigkeit im Beispieltext sortiert ausgegeben werden.
Es soll ein Index aufgebaut werden, der die Häufigkeiten im Beispieltext auf die Wörter des Wörterbuches abbildet.

```
unordered_map<string, int>>
```

Ebenso soll die vorherige Implementierung wieder verwendet werden, da das Resultat dasselbe sein muss. Als Resultat wird ein Vector geliefert, da dieser die Sortierung der eingefügten Elemente beibehält und zwar nach der Reihenfolge in der sie dem Container hinzufügt wurden.

Die Sortierung soll wie folgt erfolgen:

1. Initialisiere eine Map mit folgender Signatur
   `unordered_map<int, unordered_set<string>>`
2. Iteriere über die resultierenden Wörter und ermittle deren Häufigkeit und speichere sie in der Map
   a. Häufigkeit vorhanden: Füge das Wort dem Set hinzu
   b. Nicht vorhanden: Erstelle einen neuen Set, füge das Wort hinzu und Speichere das Set in der Map
3. Fahre fort solange noch Wörter vorhanden sind

1.2 Implementierung

Folgend ist der C Source des Programms T9Converter angeführt.

1.2.1 T9Converter.h

Folgend ist die Spezifikation des T9Converts angeführt.

```
/*
 * T9.h
 * This header file specifies the T9 converter and its needed resources
 *
 *  Created on: Mar 6, 2015
 *      Author: Thomas Herzog
 */

#ifndef T9_H_
#define T9_H_

#include <map>
#include <tr1/unordered_map>
#include <tr1/unordered_set>
#include <set>
#include <string>
#include <sstream>
#include <exception>
#include <fstream>
#include <utility>
#include <vector>

/**
 * The sorter class which gets used for sorting strings by their occurrence.
 * This sorter makes only sense when used on a std::map<int,
std::unordered_set<std::string>> where the key
 * represents the occurrences and the set the string which have the same occurrence
 */
class StringWordFrequencySorter {
    public:
        inline StringWordFrequencySorter() {
        }

        inline bool operator()(const int& left, const int& right) {
            return left > right;
        }
};

/**
 * This class handles the conversion from and to the T9 specification.
 */
class T9Converter {
    private:
        /* The references to the path of the dictonary */
        std::string& dictonaryPath;
        /* The path to the sample file which provides the occurrence count for the
dictonary entries */
        std::string& samplePath;
        /* Map which contains the character mappings to their corresponding T9 value
*/
        std::tr1::unordered_map<int, std::string> mapping;
        /* Map which contains the character mappings their special characters */
        std::tr1::unordered_map<std::string, std::string> specialCharacterMapping;
        /* The map which represents the index of the dictonary entries (T9Value ->
string) */
        std::tr1::unordered_map<long long, std::tr1::unordered_set<std::string>>
indexDictonaryT9;
        /* Map which represents the index of the dictonary entries mapped to their
length (T9Value length -> map<int, string>)*/
        std::tr1::unordered_map<int,
                std::tr1::unordered_map<long long,
                        std::tr1::unordered_set<std::string>>> indexDictonaryT9Length;
```

1.2.2 C – Source

1.2.3 Shell Script – Tests

1.3 Tests