**Постановка задачи**

Решить поставленную задачу с использованием языка программирования С++, системы контроля версий Git и средств непрерывной интеграции, предоставляемых GitLab.

**FT«1.3.2. Префиксный словарь. 2-3 дерево»**

1. Разработать и реализовать алгоритм формирования префиксного словаря: определить понятия слово и префикс; построить лексикографическое дерево из всех слов, встречающихся в тексте. Для реализации задания использовать 2-3 дерево
2. Параметром командной строки задаётся имя файла filename, который содержит внутри себя данные по некоторому количеству списков
3. Файл со списками имеет следующий вид:

<name of dict 1> <word 1 > < word 2> < word 3> < word 4> < word 5> ...

<name of dict 2> <word 1 > < word 2> < word 3> ...

...

* name of dict представляет собой имя словаря; word – слово, элемент списка.

Например input.txt:

first keyboard carrot key car knowledge mouse

second name carrot key

third

* Пустые строки игнорируются. Данные в строке разделены ровно одним пробелом

1. Реализуемая программа считывает данные словарей из файла и выполнять команды, принимаемые от пользователя со стандартного ввода.
   * Каждая строка содержит ровно одну команду. Должны поддерживаться следующие команды, все аргументы которых обязательны:

print [имя словаря]

contain [имя словаря] [слово]

words [имя словаря] [префикс]

union [имя нового словаря] [словарь1] [словарь2]

intersect [имя нового словаря] [словарь1] [словарь2]

complement [имя нового словаря] [словарь1] [словарь2]

save [имя словаря] [файл]

* + print - выводит все слова словаря в лексикографичкском порядке
  + contain - выводит True, если слово есть в словаре, false если нет
  + words - выводит все слова из словаря, начинающиеся на указанный префикс
  + union - создаёт словарь, объединяя словарь 1 и 2
  + intersect - создаёт словарь из пересечения словаря 1 и 2
  + complement - создаёт словарь вычитанием словаря2 из словаря1
  + save - запись словаря в файл
  + Если команда по каким-то причинам некорректна, то команда выводит сообщение <INVALID COMMAND>
  + Признаком конца ввода команд является EOF

**Приёмочные тесты**

|  |  |  |
| --- | --- | --- |
| **#** | **Описание** | **Результат** |
| 1 | Неверное количество аргументов командной строки | **Expected:**  Сообщение об ошибке: «Incorrect number of arguments» и ненулевой код возврата |
| 2 | В файле два словаря с одинаковым именем | **Expected:**  Сообщение об ошибке: «Dictionary with the same name already exists» и ненулевой код возврата |
| 3 | Вывод всех слов указанного словаря | **Input:**  print first  **Expected:**  first car carrot key keyboard knowledge mouse |
| **Input:**  print third  **Expected:**  <EMPTY> |
| 4 | Вывод всех слов словаря, начинающихся на указанный префикс | **Input:**  words first ca  **Expected:**  car carrot |
| **Input:**  words first as  **Expected:**  Words with prefix = as doesn't exist |
| 5 | Проверка есть ли слово в словаре | **Input:**  contain first car  **Expected:**  true |
| **Input:**  contain first card  **Expected:**  False |
| 6 | Пересечение словарей | **Input:**  intersect fourth first second  print fourth  **Expected:**  fourth carrot key |
| 7 | Объединение словарей | **Input:**  union fifth first second  print fifth  **Expected:**  fifth car carrot key keyboard knowledge mouse name |
| 8 | Вычитание словарей | **Input:**  complement sixth first second  print sixth  **Expected:**  sixth car keyboard knowledge mouse |
| 9 | Сохранение словаря в файл | **Input:**  intersect fourth first second  save fourth input.txt  **Expected in “input.txt”:**  first keyboard carrot key car knowledge mouse  second name carrot key  third  fourth carrot key |
| **Input:**  intersect fifth first second  save fifth newFile.txt  **Expected in “newFile.txt”:**  fifth carrot key |

**Исходные тексты программы**

Файлы с исходными текстами лабораторной работы располагаются в корне общего проекта (полагаем <ROOT> для папки локального репозитория)

**./<ROOT>/Nikiforova.ekaterina/FT/commandsWithDictsOfDicts.h**

#ifndef COMMANDSWITHDICTSOFDICTS\_H

#define COMMANDSWITHDICTSOFDICTS\_H

#include <set>

#include <map>

#include <string>

namespace nikiforova {

using Dict = std::set< std::string >;

using DictOfDicts = std::map< std::string, Dict >;

DictOfDicts readAllDictsFromStream(std::istream&);

Dict convertStringToDict(std::string&);

std::ostream& doPrint(std::ostream&, const std::string&, const Dict&);

std::ostream& getWordsWithPrefix(std::ostream&, const std::string&, const Dict&);

void doComplement(const std::string&, const std::string&, const std::string&, DictOfDicts&);

void doIntersect(const std::string&, const std::string&, const std::string&, DictOfDicts&);

void doUnion(const std::string&, const std::string&, const std::string&, DictOfDicts&);

std::ostream& isContain(std::ostream&, const std::string&, const Dict&);

void doSave(const std::string&, const std::string&, const Dict&);

void print(std::string&, const DictOfDicts&);

void complement(std::string&, DictOfDicts&);

void intersect(std::string&, DictOfDicts&);

void myUnion(std::string&, DictOfDicts&);

void wordsWithPrefix(std::string&, const DictOfDicts&);

void contain(std::string&, const DictOfDicts&);

void save(std::string&, const DictOfDicts&);

}

#endif

**./<ROOT>/Nikiforova.ekaterina/FT/commandsWithDictsOfDicts.cpp**

#include "commandsWithDictsOfDicts.h"

#include <string>

#include <iostream>

#include <fstream>

#include "errorMessages.h"

#include "operationsWithStrings.h"

nikiforova::DictOfDicts nikiforova::readAllDictsFromStream(std::istream& in)

{

nikiforova::DictOfDicts result;

while (!in.eof())

{

std::string str = "";

std::getline(in, str);

if (!str.empty())

{

std::string nameOfDict = nikiforova::getWord(str);

nikiforova::DictOfDicts::const\_iterator cIter = result.find(nameOfDict);

if (cIter != result.cend())

{

throw std::logic\_error("Dictionary with the same name already exists");

}

else

{

nikiforova::Dict temp = nikiforova::convertStringToDict(str);

result.insert(std::make\_pair(nameOfDict, temp));

}

}

}

return result;

}

nikiforova::Dict nikiforova::convertStringToDict(std::string& str)

{

nikiforova::Dict dict;

while (!str.empty())

{

std::string temp = nikiforova::getWord(str);

dict.insert(temp);

}

return dict;

}

std::ostream& nikiforova::doPrint(std::ostream& out, const std::string& dataset, const Dict& dict)

{

if (dict.empty())

{

nikiforova::emptyMessage(out);

out << "\n";

return out;

}

out << dataset;

for (auto&& pair: dict)

{

out << " " << pair;

}

out << "\n";

return out;

}

void nikiforova::print(std::string& str, const DictOfDicts& dicts)

{

std::string nameOfDict = nikiforova::getWord(str);

nikiforova::DictOfDicts::const\_iterator cIter = dicts.find(nameOfDict);

if (cIter == dicts.cend())

{

nikiforova::invalidCommandMessage(std::cout);

std::cout << "\n";

return;

}

nikiforova::Dict dict = cIter->second;

nikiforova::doPrint(std::cout, nameOfDict, dict);

}

void nikiforova::doComplement(const std::string& newDataset, const std::string& dataset1, const std::string& dataset2, DictOfDicts& dict)

{

nikiforova::DictOfDicts::const\_iterator cIter1 = dict.find(dataset1);

nikiforova::DictOfDicts::const\_iterator cIter2 = dict.find(dataset2);

if ((cIter1 == dict.cend()) || cIter2 == dict.cend())

{

nikiforova::invalidCommandMessage(std::cout);

std::cout << "\n";

return;

}

nikiforova::Dict result = Dict(cIter1->second);

auto listIter = cIter2->second.cbegin();

while (listIter != cIter2->second.cend())

{

auto key = \*listIter;

if (result.find(key) != result.end())

{

result.erase(key);

}

listIter++;

}

if (dict.find(newDataset) != dict.end())

{

dict.erase(newDataset);

}

dict.insert(std::make\_pair(newDataset, result));

}

void nikiforova::complement(std::string& str, DictOfDicts& dict)

{

std::string newDataset = nikiforova::getWord(str);

std::string dataset1 = nikiforova::getWord(str);

std::string dataset2 = nikiforova::getWord(str);

nikiforova::doComplement(newDataset, dataset1, dataset2, dict);

}

void nikiforova::doIntersect(const std::string& newDataset, const std::string& dataset1, const std::string& dataset2, DictOfDicts& dict)

{

nikiforova::DictOfDicts::const\_iterator cIter1 = dict.find(dataset1);

nikiforova::DictOfDicts::const\_iterator cIter2 = dict.find(dataset2);

if ((cIter1 == dict.cend()) || cIter2 == dict.cend())

{

nikiforova::invalidCommandMessage(std::cout);

std::cout << "\n";

return;

}

nikiforova::Dict result;

auto listIter1 = cIter1->second.cbegin();

while (listIter1 != cIter1->second.cend())

{

auto key1 = \*listIter1;

auto listIter2 = cIter2->second.cbegin();

while (listIter2 != cIter2->second.cend())

{

auto key2 = \*listIter2;

if ((key1 == key2) && (result.find(key1) == result.end()))

{

result.insert(\*listIter1);

}

listIter2++;

}

listIter1++;

}

if (dict.find(newDataset) != dict.end())

{

dict.erase(newDataset);

}

dict.insert(std::make\_pair(newDataset, result));

}

void nikiforova::intersect(std::string& str, DictOfDicts& dict)

{

std::string newDataset = nikiforova::getWord(str);

std::string dataset1 = nikiforova::getWord(str);

std::string dataset2 = nikiforova::getWord(str);

nikiforova::doIntersect(newDataset, dataset1, dataset2, dict);

}

void nikiforova::doUnion(const std::string& newDataset, const std::string& dataset1, const std::string& dataset2, DictOfDicts& dict)

{

nikiforova::DictOfDicts::const\_iterator cIter1 = dict.find(dataset1);

nikiforova::DictOfDicts::const\_iterator cIter2 = dict.find(dataset2);

if ((cIter1 == dict.cend()) || cIter2 == dict.cend())

{

nikiforova::invalidCommandMessage(std::cout);

std::cout << "\n";

return;

}

nikiforova::Dict result = dict.find(dataset1)->second;

auto listIter = cIter2->second.cbegin();

while (listIter != cIter2->second.cend())

{

auto key = \*listIter;

if (result.find(key) == result.end())

{

result.insert(\*listIter);

}

listIter++;

}

if (dict.find(newDataset) != dict.end())

{

dict.erase(newDataset);

}

dict.insert(std::make\_pair(newDataset, result));

}

void nikiforova::myUnion(std::string& str, DictOfDicts& dict)

{

std::string newDataset = nikiforova::getWord(str);

std::string dataset1 = nikiforova::getWord(str);

std::string dataset2 = nikiforova::getWord(str);

nikiforova::doUnion(newDataset, dataset1, dataset2, dict);

}

std::ostream& nikiforova::getWordsWithPrefix(std::ostream& out, const std::string& prefix, const Dict& dict)

{

if (dict.empty())

{

nikiforova::emptyMessage(out);

out << "\n";

return out;

}

std::set< std::string >::iterator iter = dict.begin();

std::string temp = \*iter;

while (temp < prefix)

{

iter++;

temp = \*iter;

}

if (!nikiforova::hasPrefix(temp, prefix))

{

return out << "Words with prefix = " << prefix << " doesn't exist" << "\n";

}

bool flag = 1;

while (temp >= prefix)

{

if (flag)

{

out << temp;

flag = 0;

}

else

{

out << " " << temp;

}

iter++;

if (iter == dict.end())

{

return out << "\n";

}

temp = \*iter;

if (!nikiforova::hasPrefix(temp, prefix))

{

return out << "\n";

}

}

return out;

}

void nikiforova::wordsWithPrefix(std::string& str, const DictOfDicts& dicts)

{

std::string nameOfDict = nikiforova::getWord(str);

std::string prefix = nikiforova::getWord(str);

nikiforova::DictOfDicts::const\_iterator cIter = dicts.find(nameOfDict);

if (cIter == dicts.cend())

{

nikiforova::invalidCommandMessage(std::cout);

std::cout << "\n";

return;

}

nikiforova::Dict dict = cIter->second;

nikiforova::getWordsWithPrefix(std::cout, prefix, dict);

}

std::ostream& nikiforova::isContain(std::ostream& out, const std::string& word, const Dict& dict)

{

std::set< std::string >::const\_iterator iter = dict.find(word);

return out << std::boolalpha << (iter != dict.cend()) << "\n";

}

void nikiforova::contain(std::string& str, const DictOfDicts& dicts)

{

std::string nameOfDict = nikiforova::getWord(str);

std::string word = nikiforova::getWord(str);

nikiforova::DictOfDicts::const\_iterator cIter = dicts.find(nameOfDict);

if (cIter == dicts.cend())

{

nikiforova::invalidCommandMessage(std::cout);

std::cout << "\n";

return;

}

nikiforova::Dict dict = cIter->second;

nikiforova::isContain(std::cout, word, dict);

}

void nikiforova::save(std::string& str, const DictOfDicts& dicts)

{

std::string nameOfDict = nikiforova::getWord(str);

std::string nameOfFile = nikiforova::getWord(str);

nikiforova::DictOfDicts::const\_iterator cIter = dicts.find(nameOfDict);

if (cIter == dicts.cend())

{

nikiforova::invalidCommandMessage(std::cout);

std::cout << "\n";

return;

}

nikiforova::Dict dict = cIter->second;

nikiforova::doSave(nameOfDict, nameOfFile, dict);

}

void nikiforova::doSave(const std::string& nameOfDict, const std::string& nameOfFile, const Dict& dict)

{

std::fstream fOut(nameOfFile, std::ios::app);

fOut << "\n";

nikiforova::doPrint(fOut, nameOfDict, dict);

fOut.close();

}

**./<ROOT>/Nikiforova.ekaterina/FT/operationsWithStrings.h**

#ifndef DIFFERENTUSEFULFUNCTIONS\_H

#define DIFFERENTUSEFULFUNCTIONS\_H

#include <string>

namespace nikiforova {

bool isNumber(const std::string&);

std::string getWord(std::string&);

bool hasPrefix(const std::string&, const std::string&);

}

#endif

**./<ROOT>/Nikiforova.ekaterina/FT/operationsWithStrings.cpp**

#include "operationsWithStrings.h"

bool nikiforova::isNumber(const std::string& str)

{

bool isNumber = 1;

for (size\_t i = 0; i < str.size(); i++)

{

if (!std::isdigit(str[i]) && !((str[i] == '-') && (i == 0)))

{

isNumber = 0;

}

}

return isNumber;

}

std::string nikiforova::getWord(std::string& str)

{

std::string word = "";

if (str[0] == ' ')

{

str.erase(0, 1);

}

word = str.substr(0, str.find(" "));

str = str.erase(0, str.find(" "));

return word;

}

bool nikiforova::hasPrefix(const std::string& str, const std::string& pref)

{

bool res = 1;

for (auto n = 0; n < str.length(); n++)

{

if (n >= pref.length())

{

break;

}

if(str[n] != pref[n])

{

return 0;

}

}

return 1;

}

**./<ROOT>/Nikiforova.ekaterina/FT/errorMessages.h**

#ifndef ERRORMESSAGES\_H

#define ERRORMESSAGES\_H

#include <ostream>

namespace nikiforova {

std::ostream& invalidCommandMessage(std::ostream& out);

std::ostream& emptyMessage(std::ostream& out);

}

#endif

**./<ROOT>/Nikiforova.ekaterina/FT/ errorMessages.cpp**

#include "errorMessages.h"

std::ostream& nikiforova::invalidCommandMessage(std::ostream& out)

{

return out << "<INVALID COMMAND>";

}

std::ostream& nikiforova::emptyMessage(std::ostream& out)

{

return out << "<EMPTY>";

}

**./<ROOT>/Nikiforova.ekaterina/FT/main.cpp**

﻿ #include <iostream>

#include <string>

#include <fstream>

#include <functional>

#include <set>

#include <map>

#include "errorMessages.h"

#include "commandsWithDictsOfDicts.h"

int main(int argc, char\*\* argv)

{

if (argc != 2)

{

std::cerr << "Incorrect number of arguments";

return 1;

}

std::ifstream fInput(argv[1]);

if (!fInput)

{

std::cerr << "File open error";

return 1;

}

try

{

std::map< std::string, std::set< std::string > > dict = nikiforova::readAllDictsFromStream(fInput);

using command\_t = std::function< void(std::string&, nikiforova::DictOfDicts&) >;

using constCommand\_t = std::function< void(std::string&, const nikiforova::DictOfDicts&) >;

std::map< std::string, command\_t > commands

{

{"complement", nikiforova::complement},

{"intersect", nikiforova::intersect},

{"union", nikiforova::myUnion}

};

std::map< std::string, constCommand\_t > constCommands

{

{"print", nikiforova::print},

{"words", nikiforova::wordsWithPrefix},

{"contain", nikiforova::contain},

{"save", nikiforova::save}

};

while (!std::cin.eof())

{

std::string command = "";

std::cin >> command;

if (command.empty())

{

continue;

}

auto iter = commands.find(command);

if (iter == commands.end())

{

auto cIter = constCommands.find(command);

if (cIter == constCommands.end())

{

std::string temp = "";

std::getline(std::cin, temp);

nikiforova::invalidCommandMessage(std::cout);

std::cout << "\n";

}

else

{

std::string str = "";

std::getline(std::cin, str);

cIter->second(str, dict);

}

}

else

{

std::string str = "";

std::getline(std::cin, str);

iter->second(str, dict);

}

}

}

catch (const std::exception& e)

{

std::cerr << e.what();

}

return 0;

}