Петров Константин Михайлович

Группа № 5130904/30003

Курсовая работа

Вариант

Коды Шеннона-Фано

1. **Общая постановка задачи**

Реализовать алгоритм кодирования и раскодирования текста по алгоритму Шеннона-Фано.

**Таблица с детальными требованиями и тест планом**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Требование | Детальные требования | Данные | Ожидаемый результат |  |
| 1. encode <txtName> <binName> - Зашифровать txtName и записать в binName | | | | |
| 1.1 txtName – текстовый файл  binName – двоичный файл | 1.1 encode <txtName> <binName> - Зашифровать данные из <txtName> и записать в <binName>.  Если <binName> не существует создать его.  Если <txtName> не существует, то вывести <INVALID READ NAME>  Если не выбрана кодировка, то вывести <NO CODES PROVIDED>  Если кодировка не подходит, то вывести <WRONG CODES> | * 1. не существует файла txtName   2. не выбрана кодировка   3. выбрана неправильная кодировка   4. Все условия соблюдены   Передан файл с  “Hello world  Hi Sun” | * 1. <INVALID READ NAME>   2. <NO CODES PROVIDED>   3. <WRONG CODES>   4. В binName записан: «0011110000000000010000101000010011010000010100010011111001010011010110001" |  |
| 1. decode <binName> <txtName> - Расшивровать binName и записать в txtName | | | | |
| 2.1 txtName – текстовый файл  binName – двоичный файл | 2.1 binName < binName > < txtName > - Расшифровать данные из < binName > и записать в < txtName >.  Если < txtName > не существует создать его.  Если < binName > не существует, то вывести <INVALID READ NAME>  Если не выбрана кодировка, то вывести <NO CODES PROVIDED>  Если кодировка не подходит, то вывести <WRONG CODES> | * 1. не существует файла txtName   2. не выбрана кодировка   3. выбрана неправильная кодировка   4. Все условия соблюдены   Передан файл с  “0011110000000000010000101000010011010000010100010011111001010011010110001” | * 1. <INVALID READ NAME>   2. <NO CODES PROVIDED>   3. <WRONG CODES>   2.4 В txtName записан: «Hello world  Hi Sun " |  |
| 1. autoCodes <txtName> - Создать коды на основе файла txtName | | | |  |
| 3.1 txtName – текстовый файл | 3.2 autoCodes <txtName> - Создать коды на основе файла txtName. Используя алгоритм Шеннона-Фаро создает таблицу кодов. Она необходима для использования остальных команд | 3.1 не существует файла txtName  3.2 Все условия соблюдены | 3.1 <INVALID READ NAME>  3.2 другие команды становятся доступны |  |

**Код программы:**

Main.cpp

#include <iostream>

#include "Shannon-Fano\_coding.hpp"

int main()

{

sfc::SFCoding a;

a.readCommands(std::cin, std::cout);

return 0;

}

Node.cpp

#include "utils.hpp"

utl::Node\* utl::getNode(Node\*& dest, size\_t& size, const std::string& src)

{

std::string alph = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789 \n";

size\_t alphLen = alph.size();

Node\* preSet = new Node[alphLen]{};

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

{

preSet[i].symbol = alph[i];

}

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

{

for (size\_t j = 0; j < alphLen; ++j)

{

if (preSet[j].symbol == src[i])

{

++preSet[j].quantity;

}

}

}

utl::quickSort(preSet, alphLen, [](const Node& n1, const Node& n2)

{

return n1.quantity <= n2.quantity;

});

size\_t i = 0;

for (;i < alphLen; ++i)

{

if (preSet[i].quantity > 0)

{

break;

}

}

Node\* set = new Node[alphLen - i];

utl::copyElements(set, &preSet[i], alphLen - i);

delete[] preSet;

size = alphLen - i;

utl::swap(dest, set);

delete[] set;

return dest;

}

Shannon-Fano\_coding.cpp

#include "Shannon-Fano\_coding.hpp"

#include <string>

#include <iostream>

#include <fstream>

#include "utils.hpp"

void sfc::SFCoding::readCommands(std::istream& in, std::ostream& out)

{

out << "> ";

std::string cmd;

while (in >> cmd)

{

try

{

if (cmd == "decode")

{

decode(in);

}

else if (cmd == "encode")

{

encode(in);

}

else if (cmd == "autoCodes")

{

autoCodes(in);

}

else

{

out << "<INVALID COMMAND>\n";

}

}

catch (const std::logic\_error& e)

{

out << e.what() << '\n';

}

out << "\n> ";

}

}

void sfc::SFCoding::decode(std::istream& in)

{

std::string bin, text;

in >> bin >> text;

if (!codes\_)

{

throw std::logic\_error("<NO CODES PROVIDED>");

}

std::ifstream fin(bin, std::ios::in | std::ios::binary);

if (!fin)

{

throw std::logic\_error("<INVALID READ NAME>");

}

std::ofstream fout(text, std::ios::out);

char ch;

std::string binary = "";

while (fin >> ch)

{

binary += ch;

ch = decode(binary);

if (ch != '\0')

{

fout << ch;

binary = "";

}

}

if (binary != "")

{

fout.close();

fin.close();

throw std::logic\_error("<WRONG CODES>");

}

fout.close();

fin.close();

}

void sfc::SFCoding::encode(std::istream& in)

{

std::string text, bin;

in >> text >> bin;

if (!codes\_)

{

throw std::logic\_error("<NO CODES PROVIDED>");

}

std::ifstream fin(text, std::ios::in);

if (!fin)

{

throw std::logic\_error("<INVALID READ NAME>");

}

std::ofstream fout(bin, std::ios::out | std::ios::binary);

char ch;

fin >> std::noskipws;

try

{

while (fin >> ch)

{

fout << encode(ch);

}

}

catch (...)

{

fout.close();

fin.close();

throw;

}

fout.close();

fin.close();

}

void sfc::SFCoding::autoCodes(std::istream& in)

{

std::string text;

in >> text;

std::ifstream fin(text, std::ios::in);

if (!fin)

{

throw std::logic\_error("<INVALID READ NAME>");

}

fin.seekg(0, std::ios::end);

size\_t size = fin.tellg();

std::string buffer(size, '\0');

fin.seekg(0);

fin.read(&buffer[0], size);

codes\_ = autoCodes(codes\_, size\_, buffer);

}

char sfc::SFCoding::decode(const std::string& src)

{

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

{

if (src == codes\_[i].code)

{

return codes\_[i].symbol;

}

}

return '\0';

}

std::string sfc::SFCoding::encode(char src)

{

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

{

if (src == codes\_[i].symbol)

{

return codes\_[i].code;

}

}

throw std::logic\_error("<WRONG CODES>");

}

utl::Node\* sfc::SFCoding::autoCodes(utl::Node\* dest, size\_t& size, const std::string& src)

{

using namespace utl;

getNode(dest, size, src);

FanosMethod(dest, size);

return dest;

}

void sfc::SFCoding::FanosMethod(utl::Node\* dest, size\_t size)

{

using namespace utl;

if (size < 2)

{

return;

}

size\_t size1 = size / 2;

Node\* arr1 = dest;

size\_t size2 = size - size1;

Node\* arr2 = &dest[size1];

for (size\_t i = 0; i < size1; ++i)

{

arr1[i].code += '1';

}

for (size\_t i = 0; i < size2; ++i)

{

arr2[i].code += '0';

}

FanosMethod(arr1, size1);

FanosMethod(arr2, size2);

}

Shannon-Fano\_coding.hpp

#ifndef SHANNON\_FANO\_CODING\_HPP

#define SHANNON\_FANO\_CODING\_HPP

#include <string>

#include <iostream>

#include "utils.hpp"

namespace sfc

{

struct SFCoding

{

void readCommands(std::istream& in, std::ostream& out);

void decode(std::istream& in);

void encode(std::istream& in);

void autoCodes(std::istream& in);

private:

utl::Node\* codes\_ = nullptr;

size\_t size\_ = 0;

char decode(const std::string& src);

std::string encode(char src);

utl::Node\* autoCodes(utl::Node\* dest, size\_t& size, const std::string& src);

void FanosMethod(utl::Node\* dest, size\_t size);

};

}

#endif

Utils.hpp

#ifndef UTILS\_HPP

#define UTILS\_HPP

#include <stdexcept>

namespace utl

{

struct Node

{

char symbol = 0;

size\_t quantity = 0;

std::string code = "";

};

Node\* getNode(Node\*& set, size\_t& size, const std::string& src);

template< class T, class Comp >

T\* quickSort(T\* arr, size\_t size, Comp comp);

template< class T, class Comp >

T\* quickSort(T\* arr, size\_t start, size\_t end, const Comp& comp);

template< class T >

void swap(T& a, T& b);

template< class T >

T\* copyElements(T\* dest, const T\* src, size\_t size);

}

template< class T, class Comp >

T\* utl::quickSort(T\* arr, size\_t size, Comp comp)

{

T\* tmpArr = nullptr;

try

{

tmpArr = new T[size]{};

copyElements(tmpArr, arr, size);

quickSort(tmpArr, 0, size - 1, comp);

return copyElements(arr, tmpArr, size);

}

catch (const std::bad\_alloc&)

{

throw std::logic\_error("<INVALID SIZE>");

}

catch (...)

{

throw;

}

}

template< class T, class Comp >

T\* utl::quickSort(T\* arr, size\_t start, size\_t end, const Comp& comp)

{

if (start < end)

{

T pivot = arr[end];

size\_t i = (start - 1);

for (size\_t j = start; j < end; j++)

{

if (comp(arr[j], pivot))

{

i++;

swap(arr[i], arr[j]);

}

}

swap(arr[i + 1], arr[end]);

size\_t pi = i + 1;

quickSort(arr, start, pi - 1, comp);

quickSort(arr, pi + 1, end, comp);

}

return arr;

}

template< class T >

void utl::swap(T& a, T& b)

{

T tmp = a;

a = b;

b = tmp;

}

template< class T >

T\* utl::copyElements(T\* dest, const T\* src, size\_t size)

{

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

{

dest[i] = src[i];

}

return dest;

}

#endif