МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №5

по дисциплине «Алгоритмы и структуры данных»

ТЕМА: КОДИРОВАНИЕ И ДЕКОДИРОВАНИЕ, БДП, ХЕШ-ТАБЛИЦЫ, СОРТИРОВКИ.

Студент гр. 9382 _____ Кодуков А.В.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2020

Цель работы:

Познакомиться с одним из часто используемых на практике алгоритмов кодирования данных.

Задание:

Индивидуальное задание 1:

Кодирование: Фано-Шеннона

Описание алгоритма:

Кодирование Фано-Шеннона:

Алгоритм использует избыточность сообщения, заключенную в неоднородном распределении частот символов. Также кодирование является префиксным, то есть никакой код не может быть префиксом другого.

Для кодирования файла необходимо два прохода. Первый необходим для подсчета частот всех символов. Затем символы сортируются по убыванию частоты. Далее необходимо построить дерево кодирования. Для этого массив символов делится пополам таким образом, чтобы обе части имели примерно одинаковую частоту. Эти части будут левым и правым поддеревом. Таки разбиение повторяется до тех пор, пока не дойдет до отдельных символов. Таким образом символ всегда является конечным листом дерева. Теперь если обозначить шаг влево по дереву как ‘0’, а вправо как ‘1’, то можно составить коды всех символов как путь до них по дереву. Так как символы хранятся исключительно в листьях, то никакой код не может быть префиксом другого.

Функции и структуры данных:

Структуры данных:

Бинарное дерево:

```
class Tree {
 public:
  // Tree node structure
  struct node {
    Elem info;      // Node data
    Tree *lt, *rt;  // Node childs

    …

  }

 private:
  node *Node = nullptr;  // Tree root

 …
```

```
}
```

Символы и частоты:

```
typedef std::map<unsigned char, long> ElemMap;
typedef std::vector<std::pair<unsigned char, long>> ElemArr;
```

Код символа:

```
struct CODE {
  bool Bits[50];
  int Len = 0;
};
```

Реализованные функции:

Построение дерева кодов:

Сигнатура: `Tree *BuildCodeTree(ElemArr CurFreq)`

Аргументы:

- CurFreq – текущий набор символов и их частот

Алгоритм:

- Массив пуст – вернуть пустой узел
- Массив содержит один символ – вернуть узел, построенный из этого элемента
- Посчитать среднюю частоту символов
- Разбить элементы массивы на две примерно равных по частоте части
- Найти левое и правое поддерево как результат работы функции для первой и второй части получившегося разбиения.
- Заполнить и вернуть узел

Построение новых кодов символов:

Сигнатура: `void BuildCodes(Tree *T)`

Аргументы:

- T –дерево кодирования

Алгоритм:

- Левое и правое поддерево пусты – алгоритм дошел до отдельного символа. Записать накопившийся код в список.
- Левое поддерево не пусто – увеличить текущую длину кода, записать в текущйи код ‘0’, запустить функцию от левого поддерева.
- Правое поддерево – аналог. с записью ‘1’.

Сжатие файла

Сигнатура: `bool Press(const char *filename)`

Аргументы:

- filename – имя файла

Алгоритм:

- Посчитать частоты символов
- Отсортировать символы по убыванию частоты
- Построить дерево кодирования
- Построить коды
- Вернуть файл в начало
- Записать метку
- Записать частоты
- Кодировать данные файлы, накапливая биты в специальном аккумуляторе

Разжатие файла

Сигнатура: `bool Depress()`

Алгоритм:

- Проверить метку
- Считать частоты
- Построить дерево кодирования
- Побитово считывать закодированный файл и восстанавливать данные по дереву кодирования

Тестирование:

№	Входные данные	Результат (коды и декодированный файл)
1	a	Коды: a: 1 Результат декодирования: a
2	aaaaaaaaaaaaaaaaaaaaa	Коды: a: 1 Результат декодирования: aaaaaaaaaaaaaaaaaaaaa aaaaaaaaaaaaaaaaaaaaa
3	aa bbb cccc ddddd	Коды: :101 a:11 b:100 c:01 d:00 Результат декодирования: aa bbb cccc ddddd
4	Текстовый файл (orwell.txt)	Приложен к тестовым файлам

Вывод:

В результате выполнения работы был изучен и реализован алгоритм кодирования Фано-Шеннона.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

```
#include <fstream>
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>

void PrintArray(int *A, int size, int left, int right, int lvl) {
  for (int i = 0; i < lvl; i++) std::cout << "  ";
  for (int k = 0; k < size; k++)
    std::cout << (k == left ? "|" : "") << A[k] << (k == right ? "| " : " ");
  if (right == size) std::cout << "|";
  std::cout << "\n";
}

void QuickSort(int *A, int size, int lvl) {
  long i = 0, j = size - 1;  // set pointers
  int temp, p;

  p = A[size / 2];  // pivot element

  for (int i = 0; i < lvl; i++) std::cout << "  ";
  std::cout << "Pivot: " << p << "\n";

  bool done = false;
  while (!done) {
    // move left pointer
    while (i <= j && A[i] < p) {
      PrintArray(A, size, i, j, lvl);
      i++;
    }
    // move right pointer
    while (i <= j && A[j] > p) {
      PrintArray(A, size, i, j, lvl);
      j--;
    }

    for (int k = 0; k < lvl; k++) std::cout << "  ";
    for (int k = 0; k < size; k++)
      std::cout << (k == i ? "|" : "") << A[k] << (k == j ? "| " : " ");

    if (A[i] != A[j] && i != j) {
      if (j < i) {
        std::cout << "\n";
        done = true;
      }
      // swap elements
      else {
        std::cout << "swap(" << A[i] << ", " << A[j] << ")\n";
        temp = A[i];
        A[i] = A[j];
        A[j] = temp;
      }
    } else {
      j--, i++;
      done = true;
      std::cout << "\n";
    }
  }
```

```
  // sort left subarray
  if (j > 0) QuickSort(A, j + 1, lvl + 1);
  // sort right subarray
  if (size > i) QuickSort(A + i, size - i, lvl + 1);
}

std::vector<int> Read(std::istream &s) {
  std::vector<int> v;
  std::string str;
  std::getline(s, str);
  int n = 0;
  bool passed = false;
  bool neg = false;
  for (auto &ch : str) {
    if (ch == '-') {
      if (!passed) {
        std::cout << "Wrong input";
        return std::vector<int>({});
      }
      neg = true;
    }
    else if (ch == ' ') {
      if (neg) n *= -1;
      v.push_back(n);
      passed = true;
      neg = false;
      n = 0;
    } else if (ch <= '9' && ch >= '0'){
      n = n * 10 + ch - '0';
      passed = false;
    }
    else {
      std::cout << "Wrong input";
      return std::vector<int>({});
    }
  }
  if (!passed) {
    if (neg) n *= -1;
    v.push_back(n);
  }
  return v;
}

int main() {
  std::vector<int> v;
  bool m;
  while (1) {
    char mode;

    system("cls");
    std::cout << "Choose mode:\n1 - console input\n2 - file input\n";
    mode = getchar();
    if (mode == '1') {
      m = false;
      break;
    } else if (mode == '2') {
      m = true;
      break;
    } else {
      std::cout << "Wrong option";
      getchar();
    }
  }
```

```
  std::cin.clear();
  std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
  system("cls");
  if (m) {
    char s[200];
    std::ifstream f("Tests/input.txt");
    f.getline(s, 200);
    if (f.is_open()) {
      std::cout << "File input: " << s;
      f.seekg(0, std::ios_base::beg);
      v = Read(f);
      f.close();
    }
  } else {
    v = Read(std::cin);
  }
  std::cout << "\n";
  if (!v.empty()) {
    std::vector<int> v1 = v, v2 = v;
    QuickSort(v1.data(), v1.size(), 0);
    std::cout << "\nSorted:  ";
    PrintArray(v1.data(), v1.size(), -1, -1, 0);
    std::sort(v2.begin(), v2.end());
    std::cout << "Control: ";
    PrintArray(v2.data(), v2.size(), -1, -1, 0);
    if (v1 == v2)
      std::cout << "Sorting is correct";
    else
      std::cout << "Wrong sorting";
  }
  getchar();
  return 30;
}
```