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: Декодирование: динамическое Хаффмана

Студентка гр. 9382 _____ Круглова В.Д.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург
2020

Цель работы.

Изучить алгоритм динамического декодирования Хаффмана. Реализовать алгоритм динамического декодирования с использованием алгоритма Виттера.

Задание.

Вариант 6.

Декодирование: динамическое Хаффмана

Описание основных функций и структур.

void swapNodes(Node *a, Node *b)

Назначение: меняет 2 узла/листа местами с помощью замены их собвственных указателей и указателей их родителей.

Описание аргументов: указатели на узлы a & b.

Возвращаемое значение: функция ничего не возвращает.

Node* findBlockLeader(std::vector<Node*> *vec, unsigned int weight)

Назначение: находит лидера блока (лидер блока: тот же вес, тот же тип (лист/вн.узел), максимальный номер).

Описание аргументов: указатель на вектор, хранящий узлы, искомый вес.

Возвращаемое значение: узел – лидер блока.

void splitHafTree(Node* root, std::vector<Node*> *vec, short int level)

Назначение: запишет все листы и узлы в вектор по порядку КПЛ.

Описание аргументов: указатель на вектор, хранящий узлы, счетчик уровня рекурсии.

Возвращаемое значение: функция ничего не возвращает.

void remakeNumeration(Node* p)

Назначение: расставит номера по КПЛ.

Описание аргументов: указатель на узел p.

Возвращаемое значение: функция ничего не возвращает.

char identifySymbol(std::string ascii)

Назначение: опознанет символ.

Описание аргументов: строка с кодом.

Возвращаемое значение: опознанный символ.

Node* findNextLeaf(std::string remainCode, int *count, Node* root)

Назначение: проходит по дереву, ориентируясь по значению символов строки, и возвращает узел, где он остановился.
Описание аргументов: строка с кодом, указатель на количество символов, которые нужно пройти, указатель на узел.
Возвращаемое значение: узел, в котором оказалась.

std::string vitterDecoder(std::string *code)
Назначение: декодирует код в сообщение.
Описание аргументов: указатель на строку с кодом.
Возвращаемое значение: строка с сообщением.

class Node
Назначение: содержит информацию об узле.
Node* m_parent – указатель на предка
Node* m_right – указатель на правого потомка
Node* m_left – указатель на левого потомка
unsigned int m_weight – вес узла
unsigned int m_number – номер узла
char m_symbol – символ, который хранится в листе
bool m_isLeaf – флаг листа (TRUE, если лист)
bool m_isNYT – флаг NYT (TRUE, если NYT)

Описание алгоритма.

Сначала рассматриваются первые 8 цифр кода, распознается символ, создается корень и предки, полученное значение приписывается правому предку, левый же NYT. Символ записывается в вектор.

Далее следующие цифры кода определяют действия: либо попадание в лист и тогда узел запоминается для увеличения веса внутреннего и внешнего узла и код рассматривается дальше, либо попадание в NYT, которое обеспечивает создание нового узла с описанной выше системой наследования. Символ записывается вектор.

После попадания в лист узел может не пройти проверку на соблюдение сортировки весов, тогда элементы меняются местами, после чего отдельно пересчитывается номера узлов и веса узлов, потерпевших изменения и выше до корня, при этом изменяется порядок записи символов в вектор.

Пример работы программы.

Таблица 1 – Пример работы

Входные данные	Выходные данные

0 0110000100110001010011000110110111000000 01010	Вы хотите вводить строку из терминала или из файла (1 - файл, 0 - терминал)? Для выхода из программы введите 'q' 0 Введите сообщение которое требуется декодировать 0110000100110001010011000110110111000000 01010 Был введен следующий код: 0110000100110001010011000110110111000000 01010 Декодирование началось... Анализируем первые 8 цифр: 01100001 Они соответствуют символу 'a'. Заносим этот символ в строку раскодированных символов. Строка раскодированных символов на данный момент = a Анализируем следующие 1 знач.: 0 Они ведут к узлу NYT. Значит анализируем следующие за ними 8 цифр: 01100010 Они соответствуют символу 'b'. Заносим этот символ в строку раскодированных символов. Строка раскодированных символов на данный момент = ab Анализируем следующие 2 знач.: 10 Они ведут к узлу NYT. Значит анализируем следующие за ними 8 цифр: 01100011 Они соответствуют символу 'c'. Заносим этот символ в строку раскодированных символов. Строка раскодированных символов на данный момент = abc Анализируем следующие 2 знач.: 01 Они ведут к листу, хранящему символ 'c'. Заносим этот символ в строку раскодированных символов. Строка раскодированных символов на данный момент = abcc Анализируем следующие 2 знач.: 10 Они ведут к листу, хранящему символ 'a'.

	Заносим этот символ в строку раскодированных символов. Строка раскодированных символов на данный момент = abcca Анализируем следующие 2 знач.: 11 Они ведут к листу, хранящему символ 'a'. Заносим этот символ в строку раскодированных символов. Строка раскодированных символов на данный момент = abccaa Анализируем следующие 3 знач.: 100 Они ведут к узлу NYT. Значит анализируем следующие за ними 8 цифр: 00001010 Они соответствуют символу '\n'. Заносим этот символ в строку раскодированных символов. Строка раскодированных символов на данный момент = abccaa Декодирование завершено. Итоговое декодированное сообщение: abccaa

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 — Результаты тестирования

No	Входные данные	Выходные данные	Комментарии
1	00000000	Итоговое декодированное сообщение:	Простейший случай
2	012011010	Введены некорректные данные. Остановка программы...	Некорректный ввод
3	0101	Введены некорректные данные. Остановка программы...	Некорректный ввод
4	0101010100111011010	UmU	Просто ввод
5	q	'q' was entered. Finishing program...	Выход пользователем

6	0110000100110001010011000110110111011100000001010	abccaa	Граничные данные

Выводы.

Получены знания в декодирования Хаффмана. Написана работающая программа на языке C++, способная из кода, поданного на вход переводить информацию в расшифрованную.

ИСХОДНЫЙ КОД ПРОГРАММЫ

main.cpp :

```
#include "headers.h"

unsigned int count = 187; // ( = 2*кол-во символов алфавита - 1) нужно для нумерования элементов дерева (94 печатных символа первой половины ascii)
std::vector<Node*> leafs; // здесь хранятся все адреса на узлы-листья + NYT (на нулевом месте)
std::vector<Node*> internalNodes; // здесь хранятся все адреса на внутренние узлы (корень на нулевом месте)

void swapNodes(Node *a, Node *b) // меняет 2 узла/листа местами с помощью замены их собвственных указателей и указателей их родителей
{
    if (a->m_parent == nullptr || b->m_parent == nullptr) // если одно из значений - корень
    {
        std::cout << "You can`t swap root." << std::endl;
        return;
    }

    if (a == b)
    {
        std::cout << "You can`t swap one element with itself." << std::endl;
        return;
    }

    if ((a->m_parent) == (b->m_parent)) // если у элементов один родитель
    {
        Node* parent = a->m_parent;
        if(a->m_parent->m_left == a) // если a - левый ребенок, a b - правый
        {
            a->m_parent->m_left = b;
            a->m_parent->m_right = a;
        }
        else // если же a - правый ребенок, a b - левый
        {
            a->m_parent->m_left = a;
            a->m_parent->m_right = b;
        }
        // менять указатели на родителей переданным элементам не нужно
    }
    else // если у элементов разные родители
    {
        // меняем указатели на детей в их родителях

        Node *tempVal; // временное место хранения значения

        tempVal = a->m_parent;
        if( (tempVal)->m_right == a) // если a - правый ребенок, то меняем указатель родителя на правого ребенка
```

```
            (tempVal)->m_right = b;
        else // иначе меняем указатель на левого ребенка
            (tempVal)->m_left = b;


        tempVal = b->m_parent; //
        if( (tempVal)->m_right == b) // аналогично поступаем с родителем
второго узла
            (tempVal)->m_right = a;
        else
            (tempVal)->m_left = a;


        // меняем указатели на родителей в детях

        b->m_parent = a->m_parent;
        a->m_parent = tempVal;
    };

    if(b->m_isLeaf && a->m_isLeaf) // если это 2 листа, то можно спокойно
поменять у них номера на друг друга
    {
        unsigned int k = a->m_number;
        a->m_number = b->m_number;
        b->m_number = k;
    }
};

Node* findBlockLeader(std::vector<Node*> *vec, unsigned int weight) //
находит лидера блока (лидер блока: тот же вес, тот же тип (лист/вн.узел),
максимальный номер)
{ // после вызова проверка на то, не является ли тот же элемент и лидером
(для лист-лист) и на nullptr (надо ли вообще делать swap)
    Node* p = nullptr;
    unsigned int number = 0;

    for (int i = 0; i < vec->size(); i++) // проходимся по всем элементам
    {
        if((vec->at(i))->m_weight == weight) // если вес элемента ==
нужному весу
        {
            if(p == nullptr) // первый подходящий элемент
            {
                p = vec->at(i);
                number = p->m_number;
            }
            else if( (vec->at(i))->m_number > number) // если у элемента
номер больше
            {
                p = vec->at(i);
                number = p->m_number;
            }
        }
    }
    return p;
};
```

```
void splitHafTree(Node* root, std::vector<Node*> *vec, short int
level) // запишет все листы и узлы в вектор по порядку КПЛ
{
    if (level == 1) // если это самый первый проход
    {
        vec->push_back(root); // то просто создаем первый блок
        vec->push_back(nullptr); // nullptr служат для разделения блоков
(уровней дерева)
    }
    else
    {
        short int countOfNulls = 0; // для подсчета уровней
        auto k = vec->begin();
        while (k != vec->end()) // проходимся по всему вектору
        {
            if ((*k) == nullptr) // если встретили разделитель
            {
                countOfNulls += 1;
                if (countOfNulls == level) // если количество
разделителей = уровню функции
                {
                    vec->insert(k, root); // то записываем значение в
конец нужного блока
                    break; // и останавливаем итерацию
                }
            }
            k++;
        }
        if(countOfNulls < level) // если в векторе меньше блоков, чем
уровень у функции, то добавим еще один блок
        {
            vec->push_back(root);
            vec->push_back(nullptr);
        }
    }

    if ((root->m_left) != nullptr) // если узел внутренний, то делаем
рекурсивный вызов функции для детей узла
    {
        splitHafTree(root->m_right, vec, level + 1);
        splitHafTree(root->m_left, vec, level + 1);
    }
    return;
};

void remakeNumeration(Node* p) // передаем этой функции указатель на
корень и она расставит номера по КПЛ
{
    std::vector<Node*> vec;
    splitHafTree(p, &vec, 1); // теперь в vec упорядоченно лежат все узлы
и листья дерева
    unsigned int count = 187;

    auto iter = vec.begin();
    while(iter != vec.end()) // проходимся по всем узлам и выставляем там
соответствующие номера
    {
        if (*iter != nullptr) // пропускаем все разделители
```

```
        {
            (*iter)->m_number = count;
            count -= 1;
        }
        iter++;
    }

};

void deleteAllNodes()
{
    std::vector<Node*> *allLeafs = &leafs;
    std::vector<Node*> *allIntNodes = &internalNodes;

    auto l = allLeafs->begin();
    auto iN = allIntNodes->begin();

    while(l != allLeafs->end())
    {
        delete *l;
        l++;
    }
    while(iN != allIntNodes->end())
    {
        delete *iN;
        iN++;
    }
};

char identifySymbol(std::string ascii) // принимает на вход строку с
ascii-кодом символа и возвращает опознанный символ
{
    int num = 0;
    char symbol = 0;
    for (int i = 0; i < ascii.size(); i++) // идем по всем символам
строки
    {
        if (ascii[i] == '0')
        {
            continue;
        }
        else if(ascii[i] == '1')
        {
            num = 1;
            for (int k = 7 - i; k > 0; k--)
            {
                num *= 2;
            }
            symbol += num;
        }
    }
    return symbol;
};

Node* findNextLeaf(std::string remainCode, int *count, Node* root) //
проходит по дереву, ориентируясь по значению символов строки, и
возвращает узел, где он остановился
{
```

```
    Node* currentNode = root;
    int k = 0;

    while( !(currentNode->m_isLeaf) && !(currentNode->m_isNYT))
    {
        if(remainCode[k] == '0')
        {
            currentNode = currentNode->m_left;
        }
        else if(remainCode[k] == '1')
        {
            currentNode = currentNode->m_right;
        }
        k += 1;
    }
    *count = k; // сообщает наружу, сколько символов занимает путь до
листа

    return currentNode;
};

Node* slideAndIncrement(Node* p)
{
    if(!(p->m_isLeaf)) // если это внутренний узел
    {
        Node* previousP = p->m_parent;

        // сдвигаем p в дереве выше, чем узлы-листья с весом w+1
        Node* a = findBlockLeader(&leafs, (p->m_weight) + 1);
        if(a != nullptr) // если такие листья есть
        {
            swapNodes(a, p); // т.к. здесь swap листа с внутренним узлом,
то swap не поменяет автоматически внутреннюю нумерацию
            Node* p = internalNodes.at(0); // на первом месте в списке
внутренних узлов стоит корень
            remakeNumeration(p); // пересоздаем нумерацию искуственно,
начиная с корня
        }
        p->m_weight += 1;
        return previousP;
    }
    else // если же это лист
    {
        // сдвигаем p в дереве выше, чем внутренние узлы с весом w
        Node* b = findBlockLeader(&internalNodes, p->m_weight);
        if (b != nullptr)
        {
            swapNodes(p, b);
            remakeNumeration(internalNodes.at(0));
        }
        p->m_weight += 1;
        return p->m_parent;
    }
};

int readingFile(std::string fileName, std::string *line)
{
    using namespace std;
```

```
    ifstream file(fileName, ios::in | ios::binary); // окрываем файл для
чтения
    char ch;

    if (!file)
    {
        cout << "проблема с открытием файла" << endl;
        return 1;
    }

    while(file.get(ch))
    {
        line->push_back(ch);
    }

    file.close();     // закрываем файл
    return 0;
}

std::string vitterDecoder(std::string *code) // декодирует код в
сообщение
{
    setlocale(LC_ALL, "rus");
    Node* root = new Node;          // использован конструктор для
создания первого NYT
    std::string startCode = *code;  // переданный код, который будем
раскодировывать
    std::string message = "";       // сюда будут добавляться друг за
другом раскодированные символы
    std::string temporaryStr = "";  // строка для временного хранения
значений
    int count = 0;
    char symbol;

    ::leafs.push_back(root);   // сохраняем корень в список листьев, т.к.
он пока NYT

    // Первыми всегда идут 8 цифр кода символа
    temporaryStr.assign(startCode, 0, 8);   // сохраняем 8 цифр кода во
временной переменной
    startCode.erase(0,8);                   // удаляем код символа из
общей строки кода
    std::cout << "Анализируем первые 8 цифр: " << temporaryStr <<
std::endl;
    symbol = identifySymbol(temporaryStr);  // распознаем этот символ
    message.push_back(symbol);              // записываем этот символ в
конец декодированного сообщения
    std::cout << "Они соответствуют символу \'" << symbol << "\'." <<
std::endl;
    std::cout << "Заносим этот символ в строку раскодированных символов.\
n" << std::endl;
    std::cout << "Строка раскодированных символов на данный момент = " <<
message << '\n' << std::endl;

    // начальная перестройка дерева
    root->m_left = new Node(root);          // создаем новый NYT
    root->m_right = new Node(root, symbol); // и создаем новый лист
```

```
    root->m_isNYT = false;                  // в узле, где мы
находимся(бывший NYT), указываем, что он больше не NYT

    ::leafs.at(0) = root->m_left;           // обновили NYT в списке
    ::leafs.push_back(root->m_right);       // записали новый лист в
список листьев
    ::internalNodes.push_back(root);        // записали корень в список
внутренних узлов (он занял 0 позицию и с нее не уйдет)

    root->m_weight = 1;                     // это простейший случай,
поэтому прописали веса вручную
    root->m_right->m_weight = 1;

    while(startCode != "")      // пока не проанализируем строку до конца
    {
        Node* nextLeaf = findNextLeaf(startCode, &count, root); //
находит к какому листу ведет последовательность (в count записывается
количество символов, кодирующих путь до листа)
        Node* nodeToIncrease = nullptr;                         // лист
для увеличения (нужен для отделения узлов, вес которых нужно увеличивать
в самом конце)

        std::string temp;
        temp.assign(startCode, 0 , count); // код, который ведет до листа
        std::cout << "Анализируем следующие " << count << " знач.: " <<
temp << std::endl;

        if(nextLeaf->m_isNYT) // если последовательность привела к NYT ->
встретился новый символ
        {
            startCode.erase(0, count);              // удалили путь до
NYT из общей строки кода
            temporaryStr.assign(startCode, 0, 8);   // сохранили во
временное хранилище 8 символов (код нововстреченного символа)
            startCode.erase(0,8);                   // удалили код
нововстреченного символа из общей строки кода
            symbol = identifySymbol(temporaryStr);  // идентифицируем
символ по его коду
            message.push_back(symbol);              // записываем
идентифицированный символ в строку раскодированного сообщения

            std::cout << "Они ведут к узлу NYT." << std::endl;
            std::cout << "Значит анализируем следующие за ними 8 цифр: "
<< temporaryStr << std::endl;
            std::cout << "Они соответствуют символу \'";
            if(symbol == '\n')
                std::cout << "\\n\'." << std::endl;
            else
                std::cout << symbol << "\'." << std::endl;
            std::cout << "Заносим этот символ в строку раскодированных
символов." << std::endl;
            std::cout << "Строка раскодированных символов на данный
момент = " << message << '\n' << std::endl;

            // перестройка дерева
            nextLeaf->m_left = new Node(nextLeaf);          // создаем
новый NYT
```

```
            nextLeaf->m_right = new Node(nextLeaf, symbol); // и создаем
новый лист
            nextLeaf->m_isNYT = false;                        // в узле,
где мы находимся, указываем, что он больше не NYT

            ::leafs.at(0) = nextLeaf->m_left;       // обновили NYT
            ::leafs.push_back(nextLeaf->m_right);   // указали
новосозданный лист в списке листьев
            ::internalNodes.push_back(nextLeaf);    // старый NYT
записали в список внутренних узлов

            nodeToIncrease = nextLeaf->m_right;
        }
        else // если же последовательность привела к конкретному листу ->
символ из этого листа встречается повторно
        {
            // раскодирование символа
            startCode.erase(0, count);      // удалили путь до листа из
общей строки кода
            symbol = nextLeaf->m_symbol;    // узнали, что за символ был
встречени из значения листа
            message.push_back(symbol);      // записали этот символ в
конец раскодироваемого сообщения

            std::cout << "Они ведут к листу, хранящему символ \'" <<
symbol << "\'." << std::endl;
            std::cout << "Заносим этот символ в строку раскодированных
символов.\n" << std::endl;
            std::cout << "Строка раскодированных символов на данный
момент = " << message << '\n' << std::endl;

            Node* k = findBlockLeader(&::leafs, nextLeaf-
>m_weight);    // находим лидера блока
            if (k != nextLeaf)  // если лидер блока != листу, содержащему
анализируемый символ
            {
                swapNodes(nextLeaf, k); // то меняем местами эти 2 листа
(swapNodes уже имеет внутри изменение нумерации дерева для лист swap
лист)
            }

            if ( nextLeaf->m_parent->m_left->m_isNYT ) // если p после
перемещения все равно брат NYT (это нужно во избежания пары проблем с
slideAndIncrement() )
            {
                nodeToIncrease = nextLeaf;          // лист будет
увеличен в конце отдельно
                nextLeaf = nextLeaf->m_parent;      // а цикличное
увеличение начнется с его родителя
            }
        }

        while(nextLeaf != nullptr) // цикличное увеличение веса узлов
снизу вверх
        {
            nextLeaf = slideAndIncrement(nextLeaf);
        };
```

```
        if(nodeToIncrease != nullptr) // увеличение веса узла, специально
вынесенного последним на увеличение
        {
            slideAndIncrement(nodeToIncrease);
        };
    }

    return message;
};



int main()
{
    using namespace std;
    setlocale(LC_ALL, "rus");

    string inputFile = "./text.txt";
    string line = "";
    char ch;
    bool fileFlag = false;

    { // получение от пользователя кодированной строки и сохранение ее в
line
    cout << "Вы хотите вводить строку из терминала или из файла (1 -
файл, 0 - терминал)?" << endl;
    cout << "Для выхода из программы введите 'q'" << endl;
    while(true)
    {
        ch = cin.peek();
        if(ch == 'q') // выход из программы
        {
            cin.ignore(32767,'\n');
            cout << "\'q\' was entered. Finishing program..." << endl;
            return 0;
        }
        else if(ch == '1') // ввод из файла
        {
            cin.ignore(32767,'\n');
            fileFlag = true;
            break;
        }
        else if(ch == '0') // ввод из терминала
        {
            cin.ignore(32767,'\n');
            cout << "Введите сообщение которое требуется декодировать" <<
endl;
            break;
        }
        else // введен какой-то другой символ
        {
            cout << "Ввод некорректного символа. Попробуйте снова.\n" <<
endl;
            cin.ignore(32767,'\n');
        }
    }

    if(fileFlag)    // если ввод идет через файл
```

```
    {
        readingFile(inputFile, &line); // считывает все символы в
переданную строку
        cout << "В файле был расположен следующий код:\n" << line <<
endl;
    }
    else            // если ввод идет через терминал
    {
        cin >> line;
        cout << "Был введен следующий код:\n" << line << endl;
    };
    }

    // минимальная проверка данных на корректность
    if(line.size() < 8)
    {
        cout << "\nВведены некорректные данные.\nОстановка программы..."
<< endl;
        return 0;
    }
    for(int i = 0; i < line.size(); i++)
    {
        if( (line.at(i) != '0') && (line.at(i) != '1') )
        {
            cout << "\nВведены некорректные данные.\nОстановка
программы..." << endl;
            return 0;
        }
    }

    cout << "\nДекодирование началось...\n" << endl;
    string decodedMessage = vitterDecoder(&line);
    cout << "Декодирование завершено.\n" << endl;
    cout << "Итоговое декодированное сообщение:\n" << decodedMessage;

    return 0;
}
```

headers.h :

```
#ifndef HEADERS_H
#define HEADERS_H

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

// описывает узел дерева
class Node
{
public:
    // обеспечивают построение дерева Хаффмана с другими узлами
    Node* m_parent = nullptr;
    Node* m_right = nullptr;
    Node* m_left = nullptr;

    // характеристики узла
    unsigned int m_weight = 0; // вес узла
```

```
    unsigned int m_number = 187;
    char m_symbol = '\0'; // символ который хранится в листе
    bool m_isLeaf = false; // флаг листа (TRUE, если лист)
    bool m_isNYT = true; // является ли узел NYT

    Node() = default; // базовый конструктор (исп. для корня)
    Node(Node* parent): m_parent{parent}, m_number{(parent->m_number) -
2} // конструктор для NYT с родителем
    {};
    Node(Node* parent, char symbol): m_parent{parent}, m_symbol{symbol},
m_isNYT{false}, m_isLeaf{true}, m_number{(parent->m_number) - 1} //
конструктор для листьев
    {};
};

#endif
```