Лабораторная работа 6. Алгоритмы сжатия данных

Цель работы: изучить основные виды и алгоритмы сжатия данных и научиться решать задачи сжатия данных по методу Хаффмана и с помощью кодовых деревьев.

При выполнении лабораторной работы для каждого задания требуется написать программу на языке C++, которая получает на данные с клавиатуры или из входного файла, выполняет их обработку в соответствии с требованиями задания и выводит результат в выходной *файл*. Для обработки данных необходимо реализовать функции алгоритмов сжатия данных по методу Хаффмана и с использованием кодовых деревьев. Ограничениями на *входные данные* является максимальный размер строковых данных, допустимый *диапазон* значений используемых *числовых типов* в языке C++.

Теоретические сведения.

Ознакомьтесь с материалом лекции 5-6.

Задания к лабораторной работе.

Выполните приведенные ниже задания.

1. На основании приведенных в лекции сведений реализуйте алгоритмы сжатия по методу Хаффмана через префиксные коды и на основе кодовых деревьев.
2. *Алфавит* содержит 7 букв, которые встречаются с вероятностями 0,4; 0,2; 0,1; 0,1; 0,1; 0,05; 0,05. Осуществите кодирование по методу Хаффмана.
3. Закодируйте по алгоритму Хаффмана строку с вашим именем, отчеством, фамилией, датой и местом рождения (например, "Иванова Наталья Николаевна, 1 января 1990 года, город Тверь"). При кодировании не округляйте частоты менее, чем четыре знака после запятой – сокращение точности понижает эффективность кодирования. Подсчитайте коэффициент сжатия.
4. При кодировании по методу Фано все сообщения записываются в таблицу по степени убывания вероятности и разбиваются на две группы примерно (насколько это возможно) равной вероятности. Соответственно этой процедуре из корня кодового дерева исходят два ребра, которым в качестве весов присваиваются полученные вероятности. Двум образовавшимся вершинам приписывают кодовые символы 0 и 1. Затем каждая из групп вероятностей вновь делится на две подгруппы примерно равной вероятности. В соответствии с этим из каждой вершины 0 и 1 исходят по два ребра с весами, равными вероятностям подгрупп, а вновь образованным вершинам приписывают символы 00 и 01, 10 и 11. В результате многократного повторения процедуры разделения вероятностей и образования вершин приходим к ситуации, когда в качестве веса, приписанного ребру бинарного дерева, выступает вероятность одного из данных сообщений. В этом случае вновь образованная вершина оказывается листом дерева, т.к. процесс деления вероятностей для нее завершен. Задача кодирования считается решенной, когда на всех ветвях кодового бинарного дерева образуются листья. Закодируйте по алгоритму Фано данные текстового файла.

Указания к выполнению работы.

Каждое задание необходимо решить в соответствии с изученным алгоритмами сжатия данных по методу Хаффмана и с использованием кодовых деревьев, реализовав программный код на языке C++. Рекомендуется воспользоваться материалами лекции 6, где подробно рассматриваются описание используемых в работе алгоритмов, примеры их

реализации на языке C++ (см. Приложение). Программу для решения каждого задания необходимо разработать методом процедурной абстракции, используя функции. Этапы решения сопроводить комментариями в коде. В отчете следует отразить разработку и обоснование математической модели решения задачи и привести примеры входных и выходных файлов, полученных на этапе тестирования программ. В отчете отразить *анализ* степени сжатия данных на примере тестовых заданий.

Следует реализовать каждое задание в соответствии с приведенными этапами:

- изучить словесную постановку задачи, выделив при этом все виды данных;
- сформулировать математическую постановку задачи;
- выбрать метод решения задачи, если это необходимо;
- разработать графическую схему алгоритма;
- записать разработанный алгоритм на языке C++;
- разработать контрольный тест к программе;
- отладить программу;
- представить отчет по работе.

Требования к отчету.

Отчет по лабораторной работе должен соответствовать следующей структуре.

- Титульный лист.
- Словесная постановка задачи. В этом подразделе проводится полное описание задачи. Описывается суть задачи, анализ входящих в нее физических величин, область их допустимых значений, единицы их измерения, возможные ограничения, анализ условий при которых задача имеет решение (не имеет решения), анализ ожидаемых результатов.
- Математическая модель. В этом подразделе вводятся математические описания физических величин и математическое описание их взаимодействий. Цель подраздела – представить решаемую задачу в математической формулировке.
- Алгоритм решения задачи. В подразделе описывается разработка структуры алгоритма, обосновывается абстракция данных, задача разбивается на подзадачи. Схема алгоритма выполняется по ЕСПД (ГОСТ 19.003-80 и ГОСТ 19.002-80).
- Листинг программы. Подраздел должен содержать текст программы на языке программирования C++, реализованный в среде MS Visual Studio.
- Контрольный тест. Подраздел содержит наборы исходных данных и полученные в ходе выполнения программы результаты.
- Выводы по лабораторной работе.
- Ответы на контрольные вопросы.

Контрольные вопросы

1. При кодировании каких данных можно использовать сжатие данных с потерями? Ответ обоснуйте.
2. В чем преимущества и недостатки статических методов и словарного сжатия?
3. Каким образом кодирование по алгоритму Хаффмана через *префиксный код* гарантирует минимальную длину кода?
4. За счет чего в методе Хаффмана поддерживается однозначность соответствия кода кодируемому символу?

5. Почему алгоритм Хаффмана малоэффективен для файлов маленьких размеров?

6. Выполните кодирование по методу Хаффмана через *префиксный код* символов, которые встречаются с вероятностями 0,3; 0,2; 0,1; 0,1; 0,1; 0,05; 0,05; 0,04; 0,03; 0,03. Сравните полученный результат с данными *программной реализации*.

7. Докажите, что метод Хаффмана кодирует информацию без потерь.

Пример 1. *Программная реализация* метода Хаффмана.

```
#include "stdafx.h"
#include <iostream>
using namespace std;

void Expectancy();
long MinK();
void SumUp();
void BuildBits();
void OutputResult(char **Result);
void Clear();

const int MaxK = 1000;
long k[MaxK + 1], a[MaxK + 1], b[MaxK + 1];
char bits[MaxK + 1][40];
char sk[MaxK + 1];
bool Free[MaxK + 1];
char *res[256];
long i, j, n, m, kj, kk1, kk2;
char str[256];

int _tmain(int argc, _TCHAR* argv[]){
  char *BinaryCode;
  Clear();
  cout << "Введите строку для кодирования : ";
  cin >> str;
  Expectancy();
  SumUp();
  BuildBits();
  OutputResult(&BinaryCode);
  cout << "Закодированная строка : " << endl;
  cout << BinaryCode << endl;
  system("pause");
  return 0;
}
//описание функции обнуления данных в массивах
void Clear(){
  for (i = 0; i < MaxK + 1; i++){
    k[i] = a[i] = b[i] = 0;
    sk[i] = 0;
    Free[i] = true;
    for (j = 0; j < 40; j++)
      bits[i][j] = 0;
  }
}
/*описание функции вычисления вероятности вхождения каждого символа в
тексте*/
void Expectancy(){
  long *s = new long[256];
  for ( i = 0; i < 256; i++)
    s[i] = 0;
  for ( n = 0; n < strlen(str); n++ )
    s[str[n]]++;
  j = 0;
  for ( i = 0; i < 256; i++)
    if ( s[i] != 0 ){
      j++;
      k[j] = s[i];
      sk[j] = i;
    }
```

```
  kj = j;
}
/*описание функции нахождения минимальной частоты символа в исходном тексте*/
long MinK(){
  long min;
  i = 1;
  while ( !Free[i] && i < MaxK) i++;
  min = k[i];
  m = i;
  for ( i = m + 1; i <= kk2; i++ )
    if ( Free[i] && k[i] < min ){
      min = k[i];
      m = i;
    }
  Free[m] = false;
  return min;
}
//описание функции подсчета суммарной частоты символов
void SumUp(){
  long s1, s2, m1, m2;
  for ( i = 1; i <= kj; i++ ){
    Free[i] = true;
    a[i] = 0;
    b[i] = 0;
  }
  kk1 = kk2 = kj;
  while (kk1 > 2){
    s1 = MinK();
    m1 = m;
    s2 = MinK();
    m2 = m;
    kk2++;
    k[kk2] = s1 + s2;
    a[kk2] = m1;
    b[kk2] = m2;
    Free[kk2] = true;
    kk1--;
  }
}
//описание функции формирования префиксных кодов
void BuildBits(){
  strcpy(bits[kk2],"1");
  Free[kk2] = false;
  strcpy(bits[a[kk2]],bits[kk2]);
  strcat( bits[a[kk2]] , "0");
  strcpy(bits[b[kk2]],bits[kk2]);
  strcat( bits[b[kk2]] , "1");
  i = MinK();
  strcpy(bits[m],"0");
  Free[m] = true;
  strcpy(bits[a[m]],bits[m]);
  strcat( bits[a[m]] , "0");
  strcpy(bits[b[m]],bits[m]);
  strcat( bits[b[m]] , "1");
  for ( i = kk2 - 1; i > 0; i-- )
    if ( !Free[i] ) {
      strcpy(bits[a[i]],bits[i]);
      strcat( bits[a[i]] , "0");
      strcpy(bits[b[i]],bits[i]);
      strcat( bits[b[i]] , "1");
    }
}
//описание функции вывода данных
void OutputResult(char **Result){
```

```
  (*Result) = new char[1000];
  for (int t = 0; i < 1000 ;i++)
    (*Result)[t] = 0;
  for ( i = 1; i <= kj; i++ )
    res[sk[i]] = bits[i];
  for (i = 0; i < strlen(str); i++)
    strcat( (*Result) , res[str[i]]);
}
```

Листинг .

Пример 2. *Программная реализация* алгоритма Хаффмана с помощью кодового дерева.

```
#include "stdafx.h"
#include <iostream>
using namespace std;
struct sym {
      unsigned char ch;
      float freq;
      char code[255];
      sym *left;
      sym *right;
};
void Statistics(char *String);
sym *makeTree(sym *psym[],int k);
void makeCodes(sym *root);
void CodeHuffman(char *String,char *BinaryCode, sym *root);
void DecodeHuffman(char *BinaryCode,char *ReducedString,
                   sym *root);
int chh;//переменная для подсчета информация из строки
int k=0;
//счётчик количества различных букв, уникальных символов
int kk=0;//счетчик количества всех знаков в файле
int kolvo[256]={0};
//инициализируем массив количества уникальных символов
sym simbols[256]={0};//инициализируем массив записей
sym *psym[256];//инициализируем массив указателей на записи
float summir=0;//сумма частот встречаемости

int _tmain(int argc, _TCHAR* argv[]){
  char *String = new char[1000];
  char *BinaryCode = new char[1000];
  char *ReducedString = new char[1000];
  String[0] = BinaryCode[0] = ReducedString[0] = 0;
  cout << "Введите строку для кодирования : ";
  cin >> String;
  sym *symbols = new sym[k];
  //создание динамического массива структур simbols
  sym **psum = new sym*[k];
  //создание динамического массива указателей на simbols
  Statistics(String);
  sym *root = makeTree(psym,k);
  //вызов функции создания дерева Хаффмана
  makeCodes(root);//вызов функции получения кода
  CodeHuffman(String,BinaryCode,root);
  cout << "Закодированная строка : " << endl;
  cout << BinaryCode << endl;
  DecodeHuffman(BinaryCode,ReducedString, root);
  cout << "Раскодированная строка : " << endl;
  cout << ReducedString << endl;
  delete psum;
  delete String;
  delete BinaryCode;
```

```
  delete ReducedString;
  system("pause");
  return 0;
}
//рекурсивная функция создания дерева Хаффмана
sym *makeTree(sym *psym[],int k) {
  int i, j;
    sym *temp;
    temp = new sym;
    temp->freq = psym[k-1]->freq+psym[k-2]->freq;
    temp->code[0] = 0;
    temp->left = psym[k-1];
    temp->right = psym[k-2];
    if ( k == 2 )
        return temp;
    else {
    //внесение в нужное место массива элемента дерева Хаффмана
      for ( i = 0; i < k; i++)
        if ( temp->freq > psym[i]->freq ) {
          for( j = k - 1; j > i; j--)
            psym[j] = psym[j-1];
            psym[i] = temp;
            break;
        }
    }
  return makeTree(psym,k-1);
}
//рекурсивная функция кодирования дерева
void makeCodes(sym *root) {
  if ( root->left ) {
    strcpy(root->left->code,root->code);
    strcat(root->left->code,"0");
    makeCodes(root->left);
  }
  if ( root->right ) {
    strcpy(root->right->code,root->code);
    strcat(root->right->code,"1");
    makeCodes(root->right);
  }
}
/*функция подсчета количества каждого символа и его вероятности*/
void Statistics(char *String){
  int i, j;
  //побайтно считываем строку и составляем таблицу встречаемости
  for ( i = 0; i < strlen(String); i++){
    chh = String[i];
    for ( j = 0; j < 256; j++){
      if (chh==simbols[j].ch) {
        kolvo[j]++;
        kk++;
        break;
      }
      if (simbols[j].ch==0){
        simbols[j].ch=(unsigned char)chh;
        kolvo[j]=1;
        k++; kk++;
        break;
      }
    }
  }
  // расчет частоты встречаемости
  for ( i = 0; i < k; i++)
    simbols[i].freq = (float)kolvo[i] / kk;
  // в массив указателей заносим адреса записей
```

```
  for ( i = 0; i < k; i++)
    psym[i] = &simbols[i];
  //сортировка по убыванию
  sym tempp;
  for ( i = 1; i < k; i++)
  for ( j = 0; j < k - 1; j++)
    if ( simbols[j].freq < simbols[j+1].freq ){
      tempp = simbols[j];
      simbols[j] = simbols[j+1];
      simbols[j+1] = tempp;
    }
  for( i=0;i<k;i++)  {
    summir+=simbols[i].freq;
    printf("Ch= %d\tFreq= %f\tPPP= %c\t\n",simbols[i].ch,
            simbols[i].freq,psym[i]->ch,i);
  }
  printf("\n Slova = %d\tSummir=%f\n",kk,summir);
}
//функция кодирования строки
void CodeHuffman(char *String,char *BinaryCode, sym *root){
  for (int  i = 0; i < strlen(String); i++){
    chh = String[i];
    for (int  j = 0; j < k; j++)
      if ( chh == simbols[j].ch ){
        strcat(BinaryCode,simbols[j].code);
      }
  }
}
//функция декодирования строки
void DecodeHuffman(char *BinaryCode,char *ReducedString,
                   sym *root){
  sym *Current;// указатель в дереве
  char CurrentBit;// значение текущего бита кода
  int BitNumber;
  int CurrentSimbol;// индекс распаковываемого символа
  bool FlagOfEnd;  // флаг конца битовой последовательности
  FlagOfEnd = false;
  CurrentSimbol = 0;
  BitNumber = 0;
  Current = root;
  //пока не закончилась битовая последовательность
  while ( BitNumber != strlen(BinaryCode) ) {
    //пока не пришли в лист дерева
    while (Current->left != NULL && Current->right != NULL &&
           BitNumber != strlen(BinaryCode) ) {
      //читаем значение очередного бита
      CurrentBit = BinaryCode[BitNumber++];
      //бит - 0, то идем налево, бит - 1, то направо
      if ( CurrentBit == '0' )
        Current = Current->left;
      else
        Current = Current->right;
    }
    //пришли в лист и формируем очередной символ
    ReducedString[CurrentSimbol++] = Current->ch;
    Current = root;
  }
  ReducedString[CurrentSimbol] = 0;
}
```

Листинг .