Министерство цифрового развития, связи и массовых коммуникаций Российской Федерации

СибГУТИ

Кафедра ПМиК

КУРСОВОЙ ПРОЕКТ

"Структуры и алгоритмы обработки данных"

ВАРИАНТ 68

Выполнил: студент группы ИП-013

Иванов Леонид Дмитриевич

Проверил: Дьячкова Ирина Сергеевна

Новосибирск 2022

СОДЕРЖАНИЕ

1. ПОСТАНОВКА ЗАДАЧИ

2. ОСНОВНЫЕ ИДЕИ И ХАРАКТЕРИСТИКИ ПРИМЕНЯЕМЫХ МЕТОДОВ

2.1. Метод сортировки

2.2. Поиск по базе данных

2.3. Дерево и поиск по дереву

2.4. Метод кодирования

3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ АЛГОРИТМОВ

4. ОПИСАНИЕ ПРОГРАММЫ

4.1. Основные переменные и структуры

4.2. Описание подпрограмм

5. ТЕКСТ ПРОГРАММЫ

6. РЕЗУЛЬТАТЫ

7. ВЫВОДЫ

Постановка задачи

Хранящуюся в файле базу данных (№1, 4000 записей) загрузить динамически в оперативную память компьютера в виде массива, предусмотреть возможность вывода в консоль базы данных.

Упорядочить базу данных **изданию**, используя **Цифровую сортировку(digital sort)** в качестве метода сортировки.

Предусмотреть возможность поиска по ключу в упорядоченной базе **где K –три первые буквы издания**, в результате которого из записей с одинаковым ключом формируется индексный массив, который передаётся в очередь, содержимое очереди выводится на экран.

Из записей очереди построить **двоичное Дерево оптимального поиска(приближеный алгоритм) по автору**, и предусмотреть возможность поиска в дереве по запросу.

Закодировать файл базы данных статическим **кодом Гилберта-Мура**, предварительно оценив вероятности всех встречающихся в ней символов. Построенный код вывести на экран, упакованную базу данных записать в файл.

B = 1 (файл testBase1.dat)

Библиогpафическая база данных "Жизнь замечательных людей"

Стpуктуpа записи:

Автоp: текстовое поле 12 символов

фоpмат <Фамилия>\_<буква>\_<буква>

Заглавие: текстовое поле 32 символа

фоpмат <Имя>\_<Отчество>\_<Фамилия>

Издательство: текстовое поле 16 символов

Год издания: целое число

Кол-во стpаниц: целое число

Пpимеp записи из БД:

Кловский\_В\_Б

Лев\_Hиколаевич\_Толстой\_\_\_\_\_\_\_\_\_

Молодая\_гваpдия\_

1963

864

Ваpианты условий упоpядочения и ключи поиска (К):

C = 3 - по издательству и автоpу, К = тpи пеpвые буквы

издательства.

2 ОСНОВНЫЕ ИДЕИ И ХАРАКТЕРИСТИКИ ПРИМЕНЯЕМЫХ МЕТОДОВ

2.1 Метод сортировки

Другим методом сортировки последовательностей является цифровая сортировка. Пусть дана последовательность из S чисел, представленных в *m* – ичной системе счисления. Каждое число состоит из L цифр d1d2…dL, 0 ≤ di ≤ *m* – 1, i=1..L. Сначала числа из списка S распределяются по *m* очередям, причём номер очереди определяется последней цифрой каждого числа. Затем полученные очереди соединяются в список, для которого все действия повторяются, но распределение по очередям производится в соответствии со следующей цифрой.

Для цифровой сортировки М<const L(*m+n*). При фиксированных *m* и L М=O(*n*) при *n → ∞*, что значительно быстрее остальных рассмотренных методов. Однако если длина чисел L велика, то метод может проигрывать обычным методам сортировки. Кроме того, Метод применим только, если задача сортировки сводится к задаче упорядочивания чисел, что не всегда возможно.

Метод обеспечивает устойчивую сортировку. Чтобы изменить направление

сортировки на обратное, очереди нужно присоединять в обратном порядке.

2.2Поиск по базе данных

После сортировки базы данных, с помощью индексного массива на 4000 записей, создаётся цикл, который сверяет по битам 0 и 1-ый элементы ключа и года рождения, при том, изначально меняя день и год рождения, для того, чтобы сортировка произошла правильным образом. После этого, найденные совпадения записываются в индексный массив, который, в свою очередь, подаётся в очередь

2.3Дерево и поиск по дереву

До сих пор предполагалось, что частота обращения ко всем вершинам дерева поиска одинакова. Однако встречаются ситуации, когда известна информация о вероятностях обращения к отдельным ключам. Обычно для таких ситуаций характерно постоянство ключей, т.е. в дерево не включаются новые вершины и не исключаются старые и структура дерева остается неизменной. Эту ситуацию иллюстрирует сканер транслятора, который определяет, является ли каждое слово программы (идентификатор) служебным. Статистические измерения на сотнях транслируемых программ могут в этом случае дать точную информацию об относительных частотах появления в тексте отдельных ключей.

Припишем каждой вершине дерева *Vi* вес *wi*, пропорциональный частоте поиска этой вершины (например, если из каждых 100 операций поиска 15 операций приходятся на вершину *V1*, то *w1*=15). Сумма весов всех вершин дает вес дерева *W*. Каждая вершина *Vi* расположена на высоте *hi*, корень расположен на высоте 1. Высота вершины равна количеству операций сравнения, необходимых для поиска этой вершины. Определим средневзвешенную высоту дерева с *n* вершинами следующим образом: *hср=(w1h1+w2h2+…+wnhn)/W*. Дерево поиска, имеющее минимальную средневзвешенную высоту, называется *деревом оптимального поиска* (ДОП).

*Второй алгоритм* (А2) использует предварительно упорядоченный набор вершин. В качестве корня выбирается такая вершина, что разность весов левого и правого поддеревьев была минимальна. Для этого путем последовательного суммирования весов определим вершину *Vk*, для которой справедливы неравенства:

 и .

Тогда в качестве "центра тяжести" может быть выбрана вершина *Vk, Vk-1* или *Vk+1*, т. е. вершина, для которой разность весов левого и правого поддерева минимальна. Далее действия повторяются для каждого поддерева.

2.4 Методы кодирования

Рассмотрим источник с алфавитом А={*a1,a2,…,an*} и вероятностями *p1,…pn*. Пусть символы алфавита некоторым образом упорядочены, например, *a1≤a2≤…≤an*. *Алфавитным* называется код, в котором кодовые слова лексико-графически упорядочены, т.е. *φ(a1)≤φ(a2)≤…≤φ(an).*

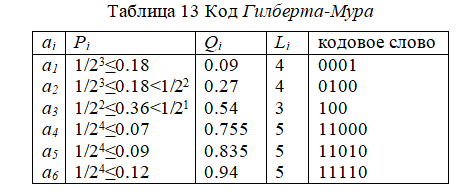
Е.Н. Гилбертом и Э.Ф. Муром предложили метод построения алфавитного кода, для которого *Lср < H+2.* Процесс построения происходит следующим образом.

1. Составим суммы *Qi, i=1,n*, вычисленные следующим образом:

*Q1=p1/2, Q2=p1+p2/2, Q3=p1+p2+p3/2,…, Qn=p1+p2+…+pn-1+pn/2*.

1. Представим суммы *Qi* в двоичном виде.
2. В качестве кодовых слов возьмем ⎡-log2*pi*⎤ +1 младших бит в двоичном представлении *Qi*.

**Пример**. Пусть дан алфавит A={*a1, a2, a3, a4, a5, a6*} с вероятностями *p1*=0.36, *p2*=0.18, *p3*=0.18, *p4*=0.12, *p5*=0.09, *p6*=0.07. Построенный код приведен в таблице.



3. ОПИСАНИЕ ПРОГРАММЫ

3.1 Основные переменные и структуры

1. Структура записей, которая инициализирует типы данных для каждой отдельно взятой записи из базы данных
2. struct record {
3. char author[12];
4. char title[32];
5. char Edition [16];
6. short int year;
7. short int page;
8. };
9. Структура списка, которая содержит данные и указатель на следующий элемент.
10. struct Node
11. {
12. record\* Record;
13. Node\* next;
14. };
15. struct Queue
16. {
17. Node\* head, \* tail;};
18. Струкутра дерево, которая содержит в себе указатели левую, правую часть и данные, а также переменную, которая отвечает за баланс дерева.
19. struct Vertex {
20. record\* data;
21. Vertex\* left;
22. Vertex\* right;

};

4.2 Описание подпрограмм

unordered\_map<char, int> get\_char\_counts\_from\_file(const string &file\_name, int n = N)

Функция выводит ошибку «не удалось открыть файл» если базу данных не удалось прочесть, в противоположном случае она вычисляет размер файла базы данных, подсчитывает символы.

int strcomp(const string& str1, const string& str2, int len = -1)

Функция для сравнивания строк.

КОД ПРОГРАММЫ:

#include <iostream>

#include <cstring>

#include <conio.h>

#include <iomanip>

#include <fstream>

#include <unordered\_map>

#include <vector>

#include <cmath>

#include <algorithm>

using namespace std;

#define Base "testBase1.dat"

const int Size = 4000;

struct Record

{

char Author[12];

char Title[32];

char Edition[16];

short int Year;

short int Page;

};

struct Node

{

Node\* Next;

Record record;

};

struct Queue

{

Node\* head, \* tail;

};

struct Vertex

{

Record\* data;

Vertex\* left;

Vertex\* right;

};

Node\* ReadFile() {

ifstream file(Base, ios::binary);

if (!file.is\_open()) {

return NULL;

}

Node\* root = NULL;

for (int i = 0; i < Size; ++i) {

Record record;

file.read((char\*)&record, sizeof(Record));

root = new Node{ root , record };

}

file.close();

return root;

}

void Print\_queue(Node\* q)

{

int i = 1, k = 0;

bool exit = false;

char input;

for (Node\* p = q; p; p = p->Next)

{

cout << i++ << "." << p->record.Author << "\t" << p->record.Title << "\t" << p->record.Edition << "\t" << p->record.Year << "\t" << p->record.Page << endl;

k++;

if (k == 20)

{

while (true)

{

input = \_getch();

if (input == 'Y' || input == 'y')

{

k = 0;

system("cls");

break;

}

else if (input == 'N' || input == 'n')

{

exit = true;

break;

}

else

{

cout << "Error!";

}

}

}

}

}

void DigitalSort(Node\*& q, bool check)

{

int l, d;

if (check == 0) { l = 12; }

else { l = 16; }

Queue Q[256];

Node\* p;

for (int j = l - 1; j >= 0; j--)

{

for (int i = 0; i < 256; i++)

{

Q[i].tail = (Node\*)&Q[i].head;

}

p = q;

while (p != NULL)

{

if (check == 0)

{

d = 255 - abs(p->record.Author[j]);

}

else

{

d = 255 - abs(p->record.Edition[j]);

}

Q[d].tail->Next = p;

Q[d].tail = p;

p = p->Next;

}

p = (Node\*)&q;

for (int i = 0; i < 256; i++)

{

if (Q[i].tail != (Node\*)&Q[i].head)

{

p->Next = Q[i].head;

p = Q[i].tail;

}

}

p->Next = NULL;

}

}

void index(Node\* head, Record\* sort\_mass[], int n = Size)

{

Node\* p = head;

for (int i = 0; i < n; i++)

{

sort\_mass[i] = &(p->record);

p = p->Next;

}

}

int strcomp(const string& str1, const string& str2, int len = -1) {

if (len == -1) {

len = (int)str1.length();

}

for (int i = 0; i < len; ++i) {

if (str1[i] == '\0' && str2[i] == '\0') {

return 0;

}

else if (str1[i] == ' ' && str2[i] != ' ') {

return -1;

}

else if (str1[i] != ' ' && str2[i] == ' ') {

return 1;

}

else if (str1[i] < str2[i]) {

return -1;

}

else if (str1[i] > str2[i]) {

return 1;

}

}

return 0;

}

int quick\_search(Record\* arr[], const string& key) {

int l = 0;

int r = Size - 1;

while (l < r) {

int m = (l + r) / 2;

if (strcomp(arr[m]->Edition, key, 3) < 0) {

l = m + 1;

}

else {

r = m;

}

}

if (strcomp(arr[r]->Edition, key, 3) == 0) {

return r;

}

return -1;

}

void search(Record\* arr[], int& ind, int& n) {

Node\* head = nullptr, \* tail = nullptr;

string key;

while (true) {

cin >> key;

break;

}

ind = quick\_search(arr, key);

if (ind == -1) {

cout << "Not found\n";

}

else {

head = new Node{ nullptr, \*arr[ind] };

tail = head;

int i;

for (i = ind + 1; i < 4000 && strcomp(arr[i]->Edition, key, 3) == 0; i++) { // ??????? Ltd

tail->Next = new Node{ nullptr, \*arr[i] };

tail = tail->Next;

}

n = i - ind;

auto find\_arr = new Record \* [n];

index(head, find\_arr, n);

Print\_queue(head);

delete[]find\_arr;

}

}

void print\_record(Record\* record) {

cout << " " << record->Author

<< " " << setw(5) << record->Title

<< " " << record->Edition

<< " " << record->Year

<< " " << record->Page << "\n";

}

void SDPREC(Record\* D, Vertex\*& p) { //добавление в рекурс

if (!p) {

p = new Vertex;

p->data = D;

p->left = nullptr;

p->right = nullptr;

}

else if (strcomp(D->Author, p->data->Author) == -1) {

SDPREC(D, p->left);

}

else if (strcomp(D->Author, p->data->Author) >= 0) {

SDPREC(D, p->right);

}

}

void A2(int L, int R, int w[], Record\* V[], Vertex\*& root) {

int wes = 0, sum = 0;

int i;

if (L <= R) {

for (i = L; i <= R; i++)

wes += w[i];

for (i = L; i <= R - 1; i++) {

if ((sum < (wes / 2)) && ((sum + w[i]) > (wes / 2))) break;

sum += w[i];

}

SDPREC(V[i - 1], root);

A2(L, i - 1, w, V, root);

A2(i + 1, R, w, V, root);

}

}

void Print\_tree(Vertex\* p, int& i)

{

if (p)

{

Print\_tree(p->left, i);

cout << i++ << ".";

print\_record(p->data);

Print\_tree(p->right, i);

}

}

void search\_in\_tree(Vertex\* root, const string& key, int& i) {

if (root) {

if (strcomp(key, root->data->Author) < 0) {

search\_in\_tree(root->left, key, i);

}

else if (strcomp(key, root->data->Author) > 0) {

search\_in\_tree(root->right, key, i);

}

else if (strcomp(key, root->data->Author) == 0) {

search\_in\_tree(root->left, key, i);

std::cout << "[" << std::setw(4) << i++ << "]";

print\_record(root->data);

search\_in\_tree(root->right, key, i);

}

}

}

void rmtree(Vertex\* root) {

if (root) {

rmtree(root->right);

rmtree(root->left);

delete root;

}

}

void tree(Record\* arr[], int n) {

Vertex\* root = nullptr;

int w[n + 1]; // ??????? ? ? ??????? Ltd

for (int i = 0; i <= n; ++i) {

w[i] = rand() % 100;

}

A2(1, n, w, arr, root);

int i = 1;

Print\_tree(root, i);

string key;

do {

getline(cin, key);

if (!key.empty() && key != "q") {

i = 1;

search\_in\_tree(root, key, i);

}

cout << "Enter the search key (lawyer's last name'), q - exit\n> ";

} while (key[0] != 'q');

search\_in\_tree(root, key, i);

rmtree(root);

}

void Mur(const int n, double p[], int Length[], char c[][20]) {

double\* q = new double[n];

Length[0] = -floor(log2(p[0])) + 1;

q[0] = p[0] / 2;

for (int i = 1; i < n; ++i) {

double tmp = 0;

for (int k = 0; k < i; k++)

tmp += p[k];

q[i] = tmp + p[i] / 2;

Length[i] = -floor(log2(p[i])) + 1;

}

for (int i = 0; i < n; ++i) {

for (int j = 0; j < Length[i]; ++j) {

q[i] \*= 2;

c[i][j] = floor(q[i]) + '0';

if (q[i] >= 1) {

q[i] -= 1;

}

}

}

}

unordered\_map<char, int> get\_char\_counts\_from\_file(const string& file\_name, int& file\_size, int n = Size) {

ifstream file(file\_name, ios::binary);

if (!file.is\_open()) {

throw runtime\_error("Нет файла");

}

char ch\_arr[sizeof(Record) \* n];

file.read((char\*)ch\_arr, sizeof(Record) \* n);

file.close();

unordered\_map<char, int> counter\_map;

file\_size = 0;

for (auto ch : ch\_arr) {

counter\_map[ch]++;

file\_size++;

}

return counter\_map;

}

vector<pair<double, char>> calc\_probabilities(const unordered\_map<char, int>& counter\_map, int count) {

vector<pair<double, char>> probabilities;

probabilities.reserve(counter\_map.size());

for (auto i : counter\_map) {

probabilities.emplace\_back(make\_pair((double)i.second / count, i.first));

}

return probabilities;

}

void coding() {

int file\_size;

unordered\_map<char, int> counter\_map;

counter\_map = get\_char\_counts\_from\_file(Base, file\_size);

auto probabilities = calc\_probabilities(counter\_map, file\_size);

counter\_map.clear();

sort(probabilities.begin(), probabilities.end(), greater<pair<char, int>>());

cout << "Вероятн. символа\n";

for (auto i : probabilities) {

cout << fixed << i.first << " | " << i.second << '\n';

}

const int n = (int)probabilities.size();

auto c = new char[n][20];

auto Length = new int[n];

auto p = new double[n];

for (int i = 0; i < n; ++i) {

p[i] = probabilities[i].first;

}

double shen = 0;

Mur(n, p, Length, c);

cout << "\nКод Гилберта-Мура:\n";

cout << "\nСимв. Вероятн. Код\n";

double avg\_len = 0;

double entropy = 0;

for (int i = 0; i < n; i++) {

avg\_len += Length[i] \* p[i];

entropy -= p[i] \* log2(p[i]);

printf("%4c | %.5lf | ", probabilities[i].second, p[i]);

for (int j = 0; j < Length[i]; ++j) {

printf("%c", c[i][j]);

}

cout << '\n';

shen += p[i];

}

cout << "Средняя длина = " << avg\_len << '\n'

<< "Энтропия = " << entropy << '\n'

<< "Средняя длина < энтропии + 2\n"

<< "N = " << n << endl;

}

void Menu(Node\* root, Record\* sort\_mass[])

{

int k;

int search\_ind, search\_n = -1;

char l ;

while (true)

{

cout << "1.DigitalSort" << endl;

cout << "2.Search" << endl;

cout << "3.Tree" << endl;

cout << "4.Coding" << endl;

cin >> k;

switch (k)

{

case 1:

{

Print\_queue(root);

}

break;

case 2:

{

search(sort\_mass, search\_ind, search\_n);

}

break;

case 3:

{

tree(sort\_mass, search\_n);

}

break;

case 4:

{

coding();

}

default:

return;

}

}

}

int main()

{

Record\* sort\_mass[Size];

Node\* root = ReadFile();

if (!root) {

cout << "File not found" << endl;

return 1;

}

DigitalSort(root, 1);

index(root, sort\_mass, Size);

Menu(root, sort\_mass);

return 0;}

