Федеральное агентство связи

Сибирский государственный университет телекоммуникаций и информатики

Кафедра прикладной математики и кибернетики

КУРСОВАЯ РАБОТА

"Структуры и алгоритмы обработки данных"

ВАРИАНТ 22

Выполнила: студентка группы ИП-413

Фараджева С. А.

Проверил: ассистент кафедры

ПМиК Климова И.В.

Новосибирск, 2015 г.

СОДЕРЖАНИЕ

1.Постановка задачи

2.Основные идеи и хаpактеpистики пpименяемых методов

2.1.Метод сортировки

2.2.Двоичный поиск

2.3.Списки и очереди

3.Описание структур данных и использованных алгоритмов

3.1.Структуры данных

3.2.Особенности реализации алгоритмов

4.Описание программы

4.1.Основные переменные и структуры

4.2.Описание подпрограмм

5.Текст пpогpаммы

6.Результаты

7.Вывод

1. Постановка задачи

Хранящуюся в файле базу данных (4000 записей) загрузить динамически в оперативную память компьютера в виде списка, вывести на экран по 20 записей (строк) на странице с возможностью отказа от показа, отсортировать базу данных с использованием очередей, затем, для проведения быстрого поиска, по отсортированному списку строится индексный массив. Предусмотреть возможность поиска по ключу в упорядоченной базе, в результате которого из записей с одинаковым ключом формируется очередь, содержимое очереди выводится на экран.

B = 4 (файл testBase4.dat)

База данных "Населенный пункт"

Стpуктуpа записи:

ФИО гражданина: текстовое поле 32 символа

фоpмат <Фамилия>\_<Имя>\_<Отчество>

Название улицы: текстовое поле 18 символов

Номер дома: целое число

Номер квартиры: целое число

Дата поселения: текстовое поле 10 символов

фоpмат дд-мм-гг

Пpимеp записи из БД:

Петpов\_Иван\_Федоpович\_\_\_\_\_\_\_\_\_\_\_

Ленина\_\_\_\_\_\_\_\_\_\_\_\_\_\_

10

67

29-02-65

Условие упорядочивания и ключи поиска (К): по дате поселения и названию улицы, К = год поселения.

Метод сортировки: метод Хоара.

2. Основные идеи и характеристики применяемых методов

2.1. Метод сортировки

*Метод Хоара*

Метод Хоара или метод быстрой сортировки заключается в следующем. Возьмём произвольный элемент массива х. Просматривая массив слева, найдём элемент ai ≥x. Просматривая массив справа, найдём aj ≤x. Поменяем местами ai и aJ . Будем продолжать процесс просмотра и обмена, до тех пор пока i не станет больше j. Тогда массив можно разбить на две части: в левой части все элементы не больше х, в правой части массива не меньше х. Затем к каждой части массива применяется тот же алгоритм.

трудоёмкость метода существенно зависит от выбора элемента х, который влияет на разделение массива. Максимальные значения М и С для метода быстрой сортировки достигаются при сортировке упорядоченных массивов (в прямом и обратном порядке). Тогда в этом случае в одной части остаётся только один элемент (минимальный или максимальный), а во второй – все остальные элементы. Выражения для М и С имеют следующий вид

M=*3(n-1),* C=*(n2+5n+4)/2*

Таким образом, в случае упорядоченных массивов трудоёмкость сортировки имеет квадратичный порядок.

Элемент *am* называется *медианой* для элементов *aL…aR,* если количество элементов меньших *am* равно количеству элементов больших *am* с точностью до одного элемента (если количество элементов нечётно).

Минимальная трудоемкость метода Хоара достигается в случае, когда на каждом шаге алгоритма в качестве ведущего элемента выбирается медиана массива. Количество сравнений в этом случае C=*(n+1)*log*(n+1)-(n+1)*. Количество пересылок зависит от положения элементов, но не может быть больше одного обмена на два сравнения. Поэтому количество пересылок – величина того же порядка, что и число сравнений. Асимптотические оценки для средних значений М и С имеют следующий вид

С=О(*n* log *n*), М=О(*n* log *n*) при *n → ∞*.

Метод Хоара неустойчив.

2.2. Двоичный поиск

Алгоритм *двоичного поиска* в упорядоченном массиве сводится к следующему. Берём средний элемент отсортированного массива и сравниваем с ключом X. Возможны три варианта:

Выбранный элемент равен X. Поиск завершён.

Выбранный элемент меньше X. Продолжаем поиск в правой половине массива.

Выбранный элемент больше X. Продолжаем поиск в левой половине массива.

Из-за необходимости найти все элементы соответствующие заданному ключу поиска в курсовой работе использовалась вторая версия двоичного поиска, которая из необходимых элементов находит самый левый, в результате чего для поиска остальных требуется просматривать лишь оставшуюся правую часть массива.

Верхняя оценка трудоёмкости алгоритма двоичного поиска такова. На каждой итерации поиска необходимо два сравнение для первой версии, одно сравнение для второй версии. Количество итераций не больше, чем C:\..\Documents and Settings\Belida\Application Data\Microsoft\Word\СибГУТИ!!!\5_САОД\SAOD-BOOK\img\sample\f_02.GIF. Таким образом, трудоёмкость двоичного поиска в обоих случаях

C:\..\Documents and Settings\Belida\Application Data\Microsoft\Word\СибГУТИ!!!\5_САОД\SAOD-BOOK\img\sample\f_03.GIF

2.3 Списки и очереди

*Очередь* — это линейный список информации, работа с которой происходит по принципу "первым пришел — первым вышел" (first-in, first-out); этот принцип (и очередь как структура данных) иногда еще называется FIFO[[1]](http://lord-n.narod.ru/download/books/walla/programming/Spr_po_C/22/2201.htm" \l "11). Это значит, что первый помещенный в очередь элемент будет получен из нее первым, второй помещенный элемент будет извлечен вторым и т.д. Это единственный способ работы с очередью; произвольный доступ к отдельным элементам не разрешается.

В программировании очереди применяются при решении многих задач. Один из наиболее популярных видов таких задач — симуляция. Очереди также применяются в планировщиках задач операционных систем и при буферизации ввода/вывода.

В отличие от очереди, *связанный список* допускает гибкие способы доступа, поскольку каждый фрагмент информации имеет ссылку на следующий элемент данных в цепочке. Кроме того, операция извлечения не приводит к удалению из списка и уничтожению элемента данных. В принципе, для этой цели необходимо ввести дополнительную специальную операцию удаления.

Связанные списки могут быть односвязными и двусвязными. В односвязном списке каждый элемент информации содержит ссылку на следующий элемент списка. Каждый элемент данных обычно представляет собой структуру, которая состоит из информационных полей и указателя связи. Существует два основных способа построения односвязного списка. Первый способ — помещать новые элементы в конец списка. Второй — вставлять элементы в определенные позиции списка, например, в порядке возрастания. От способа построения списка зависит алгоритм функции добавления элемента.

Двусвязный список содержит ссылки как на последующий, так и на предыдущий элементы списка. Наличие двух ссылок вместо одной предоставляет несколько преимуществ. Вероятно, наиболее важное из них состоит в том, что перемещение по списку возможно в обоих направлениях. Это упрощает работу со списком, в частности, вставку и удаление. Помимо этого, пользователь может просматривать список в любом направлении. Еще одно преимущество имеет значение только при некоторых сбоях. Поскольку весь список можно пройти не только по прямым, но и по обратным ссылкам, то в случае, если какая-то из ссылок станет неверной, целостность списка можно восстановить по другой ссылке.

Выбор типа применяемого списка зависит от конкретной задачи.

3.Описание структур данных и использованных алгоритмов

В ходе выполнения курсовой работы, помимо основных алгоритмов, потребовалось реализовать также несколько вспомогательных, необходимых для корректной работы программы.

1. Интерфейс программы

Для организации интерфейса используется строчка

*puts("Select void:\n1-Print\n2-Sort\n3-Search\n4-Exit")*

которая обеспечивает корректное и незатруднительное использование программы и предоставляет выбора различных вариантов обработки базы данных, в зависимости от задач пользователя.

2. Загрузка и вывод базы данных

Для построения очереди базы данных служит запись *struct srec, а* для работы с ней *struct record.* Здесь же предусмотрена проверка на наличие файла, откуда выполняется считывание. Данная процедура вызывается независимо от желания пользователя, в то время как остальные он может выбрать посредствам меню.

4. ОПИСАНИЕ ПРОГРАММЫ

4.1. Основные переменные и структуры

**struct record**

**{**

**char fio[32];**

**char street[18];**

**short int house;**

**short int flat;**

**char hd[10];**

**};**

Запись, используемая для работы с базой данных

**struct srec**

**{**

**char fio[32];**

**char street[18];**

**short int d;**

**short int kv;**

**char data[10];**

**srec \*next;**

**};**

Запись, используемая для работы с базой данных.

***\*inx[n]*** – индексный массив.

***struct record \*\*inx*** – указатель на индексный массив. ***struct record \*\*inx*** – указатель на массив записей, который следует отсортировать; ***x*** – медиана.

4.2. Описание подпрограмм

Функции и процедуры сортировки:

***int sravn*** – функция, выполняющая сравнение элементов.

***void sort*** –производит сортировку методом Хоара.

Функции и процедуры для поиска в отсортированной базе данных:

***int sravn2*** – функция производит сравнение между ключом и средним элементом.

***void sort*** – функция выполняет двоичный поиск. **struct record \*\*inx** – указатель на массив записей, в котором осуществляется поиск.

5. ТЕКСТ ПРОГРАММЫ

using namespace std;

const int n=4000;

struct srec {

char fio[32];

char street[18];

short int d;

short int kv;

char data[10];

srec \*next;

};

struct record

{

char fio[32];

char street[18];

short int d;

short int kv;

char data[10];

};

int sravn(struct record \*inx1, struct record \*inx2){

if(inx1->data[6]<inx2->data[6])return 1;

if(inx1->data[6]>inx2->data[6])return 0;

if(inx1->data[7]<inx2->data[7])return 1;

if(inx1->data[7]>inx2->data[7])return 0;

if(inx1->data[3]<inx2->data[3])return 1;

if(inx1->data[3]>inx2->data[3])return 0;

if(inx1->data[4]<inx2->data[4])return 1;

if(inx1->data[4]>inx2->data[4])return 0;

if(inx1->data[0]<inx2->data[0])return 1;

if(inx1->data[0]>inx2->data[0])return 0;

if(inx1->data[1]<inx2->data[1])return 1;

if(inx1->data[1]>inx2->data[1])return 0;

if (strcmp(inx1->street,inx2->street)<0) return 1;

else return 0;

}

int sravn2(char \*inx1, char \*inx2){

if(inx1[6]<inx2[0])return 1;

if(inx1[6]>inx2[0])return 0;

if(inx1[7]<inx2[1])return 1;

if(inx1[7]>inx2[1])return 0;

return 2;

}

void sort (struct record \*\*inx, int l, int r){

int t,i,j;

struct record \*p;

struct record \*x;

while(l<r){

x=inx[l]; i=l; j=r;

while (i<=j){

while (sravn(inx[i],x)) i++;

while (sravn(x, inx[j])) j--;

if (i<=j) {

p=inx[i];

inx[i]=inx[j];

inx[j]=p;

i++;

j--;

}

}

if (j-l<r-i){ sort(inx,i,r); r=j;}

else {sort(inx,l,j); l=i;}

}

}

void search(struct record \*\*inx, int n) {

int l=0,r=n-1,m,i;

char k[2];

srec \*p,\*pp,\*head;

puts("Search key:");

scanf("%s",k);

while(l<r){

m=(l+r)/2;

if (sravn2(inx[m]->data,k)==1) l=m+1;

else r=m;

}

if (sravn2(inx[r]->data,k)==2) printf("!");

if (sravn2(inx[r]->data,k)==2) {

p=new srec; head=p;

for (i=0; i<32; i++)head->fio[i]=inx[r]->fio[i];

for (i=0; i<18; i++) head->street[i]=inx[r]->street[i];

for (i=0; i<10; i++) head->data[i]=inx[r]->data[i];

head->kv=inx[r]->kv; head->d=inx[r]->d;

while ((++r<n)&&(sravn2(inx[r]->data,k)==2)) { pp=new srec;p->next=pp; p=pp;

for (i=0; i<32; i++)p->fio[i]=inx[r]->fio[i];

for (i=0; i<18; i++) p->street[i]=inx[r]->street[i];

for (i=0; i<10; i++) p->data[i]=inx[r]->data[i];

p->kv=inx[r]->kv; p->d=inx[r]->d;}

p->next=NULL;

for (p=head; p; p=p->next) printf("%s %s %d %3d %s\n",p->fio,p->street,p->d,p->kv,p->data);}

else printf("Not found!\n");

getch();

}

void print(struct record \*\*inx, int n){

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

printf("%4d) %s %s %d %3d %s\n",i+1,inx[i]->fio,inx[i]->street,inx[i]->d,inx[i]->kv,inx[i]->data);

if ((i+1)%20==0){puts("To exit press Esc"); if (getch()==27) break;}}

}

int main()

{ record \*inx[n];

int i,m,f=1;

srec \*p,\*head,\*pp;

FILE \*fp;

record mas[n];

fp=fopen("testBase4.dat", "rb");

i=fread((record \*)mas, sizeof(record), n, fp);

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

inx[i]=&mas[i];

while(1){

system("cls");

puts("Select void:\n1-Print\n2-Sort\n3-Search\n4-Exit");

scanf("%d",&m);

switch (m) {

case 1:system("cls"); print(inx,n);

break;

case 2:sort(inx,0,n-1); f=3;

break;

case 3:system("cls"); if (f!=3){ puts("Sorting was not performed!"); getch();}

else search(inx,n);

break;

case 4: return 4;

break;

default: puts("Error");

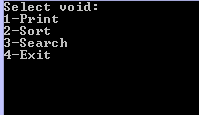
break;

}}

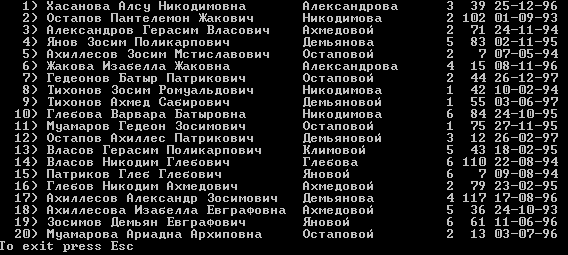
}

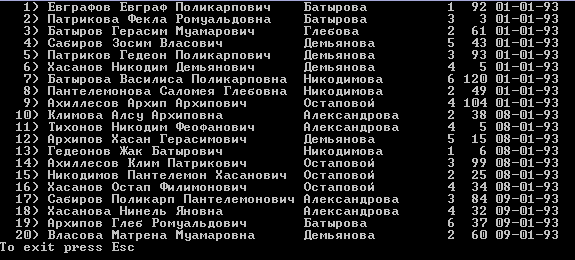
6. РЕЗУЛЬТАТЫ

1. Меню программы

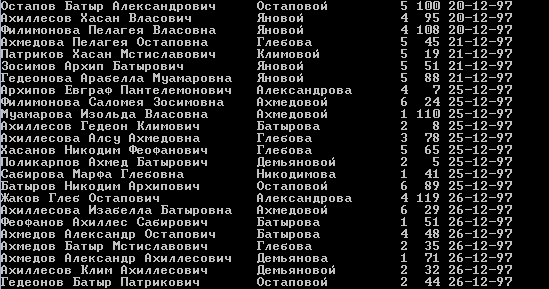


2. Неотсортированная база данных



3. Отсортированная по дате заселения и названию улицы база

4. Результат поиска по ключу



7. ВЫВОД

В проделанной курсовой работе были выполнены все поставленные задачи, реализованы такие алгоритмы, как работа со списками и очередями, сортировка и поиск. Структурированность кода, имена переменных, структур, функций и процедур сделали программу удобной в пользовании и понимании. Использованные алгоритмы дали возможность эффективно обработать программу и произвести необходимые операции с минимальным потраченным временем.