Московский Авиационный Институт

(национальный исследовательский университет)

Факультет “Прикладной математики и информатики”

Кафедра 806

Курсовой проект по теме: «Сортировка и поиск»

Выполнил: Студент группы М8О-108Б-20

Борисов Я.А

Проверил: Трубченко Н.М

Москва, 2021

**ОТЧЕТ**

**Цель:** составить программу на языке Си с использованием процедур и функций для сортировки таблицы заданным методом и двоичного поиска по ключу в таблице.

**Идея:** идея заключается в том, чтобы реализовать таблицу на линейном списке с ключом типа строка-целое и сделать функции поиска по данному списку (бинарный поиск для отсортированного списка). Поиск и сортировка реализованы на итераторах. Сортировка производилась таким образом: сначала лексикографически сравнивалась string часть ключа, при совпадении этой части сравнивались целые числа ключа.

Код программы:

list.h

#ifndef KP9\_LIST\_H

#define KP9\_LIST\_H

#include "stdio.h"

#include <stdlib.h>

#include "string.h"

#include <stdbool.h>

struct str\_int{

int value;

char\* string;

};

struct Node

{

struct str\_int key;

char\* data;

struct Node\* next;

};

struct List

{

struct Node\* head;

int size;

struct Node\* end;

};

struct List createList(){

struct List list;

list.head = NULL;

list.size = 0;

return list;

}

typedef struct Node\* ListIterator;

void set\_element(struct Node\* \_node, struct str\_int key, char\* data){

if (\_node != NULL){

\_node->key = key;

\_node->data = data;

}

}

void set\_next\_element(struct Node\* \_node, struct Node\* \_next){

if (\_node != NULL){

\_node->next = \_next;

}

}

bool is\_list\_empty(struct List\* \_list){

return \_list->size == 0;

}

ListIterator end(struct List\* \_list){

return \_list->end;

}

ListIterator begin(struct List\* \_list){

if (\_list != NULL && \_list->head != NULL){

return \_list->head;

}

else

return end(\_list);

}

ListIterator next(ListIterator It){

return It->next;

}

ListIterator plus(struct List\* \_list, ListIterator It, int \_n){

if (is\_list\_empty(\_list)){

return begin(\_list);

}

ListIterator ansIterator = It;

int i = 0;

while (i < \_n && next(ansIterator) != end(\_list)){

ansIterator = next(ansIterator);

++i;

}

return ansIterator;

}

int distance(ListIterator \_begin, ListIterator \_end){

int ans = 0;

ListIterator curIt = \_begin;

while (curIt != \_end){

++ans;

curIt = next(curIt);

}

return ans;

}

void increase(struct List\* \_list, ListIterator \_lIt, int \_n){

int i = 0;

while (i < \_n && \_lIt != end(\_list)){

\_lIt = next(\_lIt);

}

}

int getListLength(struct List\* \_list){

return distance(begin(\_list), end(\_list));

}

void pushBack(struct List\* \_list, struct str\_int key, char\* data){

struct Node\* newNode = (struct Node\*) malloc(sizeof(struct Node));

set\_element(newNode, key, data);

if (is\_list\_empty(\_list)){

set\_next\_element(newNode, end(\_list));

\_list->head = newNode;

++\_list->size;

return;

}

struct Node\* lastNode = begin(\_list);

while (next(lastNode) != end(\_list)){

lastNode = next(lastNode);

}

set\_next\_element(lastNode, newNode);

set\_next\_element(newNode, end(\_list));

++\_list->size;

}

void popBack(struct List\* \_list){

ListIterator prevNode = plus(\_list, begin(\_list), \_list->size - 2);

if (getListLength(\_list) > 1){

ListIterator deletedNode = next(prevNode);

free(deletedNode);

set\_next\_element(prevNode, end(\_list));

--\_list->size;

}

else if (getListLength(\_list) == 1){

free(\_list->head);

\_list->head = end(\_list);

--\_list->size;

}

}

void printList(struct List\* \_list){

printf("list: ");

struct Node\* curNode = begin(\_list);

while (curNode != end(\_list)){

printf("%s%s**\t**%d**\t**", "key: ", curNode->key.string, curNode->key.value);

printf("%s%s**\n**", "data: ", curNode->data);

curNode = next(curNode);

}

printf("**\n**");

}

void destroyList(struct List\* \_list){

struct Node\* deletedNode = begin(\_list);

if (deletedNode == end(\_list)){

return;

}

struct Node\* nextNode = next(deletedNode);

while (nextNode != end(\_list)){

printf("%s %s**\n**", deletedNode->data, nextNode->data);

free(deletedNode->key.string);

free(deletedNode->data);

free(deletedNode);

deletedNode = nextNode;

nextNode = next(nextNode);

}

free(deletedNode->key.string);

free(deletedNode->data);

free(deletedNode);

}

bool tableSorted = **false**;

void linear\_sort(struct List\* \_list){

struct Node\* curNode = begin(\_list);

while(next(curNode) != end(\_list)){

if(strcmp(curNode->key.string, next(curNode)->key.string) > 0){

struct Node tmp = { .key = curNode->key, .data = curNode->data };

curNode->key = curNode->next->key;

curNode->data = curNode->next->data;

next(curNode)->key = tmp.key;

next(curNode)->data = tmp.data;

curNode = begin(\_list);

}

else if (strcmp(curNode->key.string, next(curNode)->key.string) == 0){

if(curNode->key.value > next(curNode)->key.value){

struct Node tmp = { .key = curNode->key, .data = curNode->data };

curNode->key = next(curNode)->key;

curNode->data = next(curNode)->data;

next(curNode)->key = tmp.key;

next(curNode)->data = tmp.data;

curNode = begin(\_list);

}

else curNode = next(curNode);

}

else curNode = next(curNode);

}

tableSorted = **true**;

}

void simplesearch(ListIterator begin, ListIterator end, struct str\_int key){

ListIterator curNode;

curNode = begin;

int i = 0;

bool element\_found = **false**;

while(curNode != end){

if ((strcmp(curNode->key.string, key.string) == 0) && (curNode->key.value == key.value)) {

printf("%s%d", "element found. it's position is ", i);

element\_found = **true**;

curNode = next(curNode);

}

else curNode = next(curNode);

i++;

}

if (element\_found == **false**) printf("%s", "element not found");

}

struct Node\* get\_Item\_by\_index(ListIterator begin, int index){

ListIterator curIterator = begin;

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

curIterator = next(curIterator);

}

return curIterator;

}

void binarysearch(ListIterator begin, ListIterator end, struct str\_int key){

bool element\_found = **false**;

int left = 0;

int right = distance(begin, end) - 1;

int mid = (distance(begin, end) + 0) / 2;

while(left <= right){

mid = (left + right) / 2;

if (strcmp(key.string, get\_Item\_by\_index(begin, mid)->key.string) == 0 && key.value == get\_Item\_by\_index(begin, mid)->key.value){

element\_found = **true**;

printf("%s%d", "element found. it's position is ", mid);

**break**;

}

if (strcmp(key.string, get\_Item\_by\_index(begin, mid)->key.string) < 0 ||

(strcmp(key.string, get\_Item\_by\_index(begin, mid)->key.string) == 0 && key.value < get\_Item\_by\_index(begin, mid)->key.value)){

right = mid - 1;

}

else left = mid + 1;

}

if (element\_found == **false**) printf("%s", "element not found");

}

#endif //KP9\_LIST\_H

main.c

#include "list.h"

int main(){

struct List list = createList();

int symb;

while ((symb = getchar()) != EOF) {

switch (symb) {

case '+': {

struct str\_int key;

char text[100];

char\* data = (char\*) malloc(sizeof(data));

char\* string = (char\*) malloc(sizeof(string));

int value;

fgets(text, 100, stdin);

sscanf(text," %s%d%s", string, &value, data);

key.string = string;

key.value = value;

pushBack(&list, key, data);

**break**;

}

case 'p': {

printList(&list);

**break**;

}

case '-': {

popBack(&list);

**break**;

}

case 's':{

linear\_sort(&list);

**break**;

}

case '?':{

struct str\_int key;

char text[100];

char\* string = (char\*) malloc(sizeof(string));

int value;

struct Node\* curNode = begin(&list);

fgets(text, 100, stdin);

sscanf(text," %s%d", string, &value);

key.string = string;

key.value = value;

struct str\_int memory;

memory.string = key.string;

memory.value = key.value;

if(tableSorted) binarysearch(begin(&list), end(&list), memory);

if(!tableSorted) simplesearch(begin(&list), end(&list), key);

free(string);

**break**;

}

}

}

destroyList(&list);

return 0;

}

**Вывод:** составил программу на языке Си с использованием процедур и функций для сортировки таблицы заданным методом и двоичного поиска по ключу в таблице.