БУ ВО Ханты-Мансийского автономного округа – Югры

«Сургутский государственный университет»

Политехнический институт

Кафедра автоматики и компьютерных систем

ОТЧЁТ

по лабораторной работе №3

по дисциплине: «Алгоритмы и структуры данных»

Выполнил: студент группы №605-01,

Квапиш Н. В.

Принял: первый проректор

Даниленко И. Н.

Сургут

2022г.

**Цель работы:** изучить базовые алгоритмы поиска, исследовать их свойства, закрепить навыки структурного программирования.

**Задание:**

1. Разработать функции, реализующие три алгоритма поиска (2-ой - обязательно, еще два - по выбору).

2. Исследовать алгоритмы поиска: построить и сравнить зависимости среднего количества сравнений в случаях успешного и неуспешного поиска для заданных алгоритмов.

3. Найти зависимость оптимальной величины «прыжка» от размера последовательности.

4. Составить отчет, в котором привести графики полученных зависимостей, анализ свойств алгоритмов и выводы по работе.

**Ход работы.**

**Выбранные виды поиска:**

1. Последовательный.
2. Прыжками.
3. Бинарный.

**Результаты при удачном поиске:**

Рисунок 1 среднее количество сравнений от размера для последовательного поиска

Рисунок 2 среднее количество сравнений от размера для бинарного и одноуровневого поиска прыжками

Рисунок 3 среднее количество сравнений от размера для двухуровневого поиска прыжками

**Результаты при неудачном поиске:**

Рисунок 4 среднее количество сравнений от размера для последовательного поиска

Рисунок 5 среднее количество сравнений от размера для бинарного и поиска прыжками

**Оценка результатов:**

При поиске прыжками в худшем случае будет сделано n/k переходов, где n – размер последовательности, k – шаг. Затем, если последнее проверенное значение больше ключа, то нам предстоит сделать не более k-1 сравнения на интервале от прошлого шага до текущего. В общем получается (n/k)+(k-1) сравнение. Для нахождения оптимального шага нужно найти минимум этой функции, значит оптимальным шагом для поиска прыжками является , где n – длина последовательности.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Последовательный поиск | Поиск прыжками | Бинарный поиск |
| Сложность | Худшая: O(n)  Средняя: O(n/2) | O() | O(log n) |

Так же стоит заметить, что у двухуровневого поиска прыжками, несмотря на большее количество сравнений, время поиска в среднем меньше, чем у одноуровневого.

Рисунок 6 временная зависимость поиска прыжками от количества элементов

**Вывод:** при выполнении работы изучены принципы работы алгоритмов поиска в линейных структурах. Лучше всего себя показал бинарный поиск. Хуже всего - алгоритм последовательного поиска. У многоуровневого поиска прыжками, при увеличении “уровня” повышается количество сравнений, но при этом уменьшается среднее время поиска

**Листинг.**

#include <stdlib.h>

#include <stdio.h>

#include <math.h>

int Exchange = 0;

void Swap(int & a, int & b)

{

int temp = a;

a = b;

b = temp;

}

void upp(int \*A, int N)

{

int k = 0;

for (int i = 0; i < N-1; i++){

A[i] = k;

k += 2;

}

}

void Counter(int x)

{

Exchange += x;

}

int ConsistentSearch(int \*A, int N, int key)

{

int i;

A[N-1] = key;

for (i=0; A[i]!=key; i++)

Counter(1);

return ((i < N - 1) ? i : -1);

}

int JampSearch1(int \*A, int N, int key)

{

int step=sqrt(N), prev=0;

while(A[step]<key) {

Counter(1);

prev = step;

step += sqrt(N);

if(step>=N)

return -1;

}

while (A[prev]<key) {

Counter(1);

prev++;

if (prev == step)

return(-1);

}

return ((A[prev] == key) ? prev : -1);

}

int JampSearch2(int \*A, int N, int key)

{

int step1=sqrt(N), prev=0, step2, prev2;

while(A[step1]<key) {

Counter(1);

prev = step1;

step1 += sqrt(N);

if(step1>=N)

return -1;

}

step2=prev+sqrt(step1-prev);

while (A[step2]<key) {

Counter(1);

prev2=step2;

step2+=sqrt(step1-prev);

if (step2 >= step1-prev)

return(-1);

}

while(A[prev2]<key) {

Counter(1);

prev2++;

if (prev2 == step2)

return(-1);

}

return ((A[prev2] == key) ? prev2 : -1);

}

int BinSearch(int \*A, int N, int key)

{

int Min = 0, Max = N - 1, Pos = Max / 2;

for( ; Min <= Max; Pos = (Min + Max) / 2)

if(key < A[Pos]){

Max = Pos - 1;

Counter(1);

}

else

if (key > A[Pos]){

Min = Pos + 1;

Counter(1);

}

else

return Pos;

return -1;

}

void Out(int \*A, int N)

{

for(int i = 0; i < N-1; i++)

{

printf ("%d \n", A[i]);

}

}

main ()

{

int t2 = 0, t1 = 0, key;

int \*A = 0;

const int NMax = 50000;

A = new int [NMax];

for (int k = 0; k < 5; k++)

{

for (int N = NMax/10; N <= NMax; N+= NMax/10)

{

upp(A, N);

Exchange = 0;

//key = A[rand()%N]; // успешный поиск

key = 7; // худший случай

//ConsistentSearch(A, N, key);

JampSearch1(A, N, key);

//JampSearch2(A, N, key);

//BinSearch(A, N, key);

//Out(A, N);

printf("%d\n", Exchange);

}

printf("\n");

}

delete [] A;

system("pause");

}