|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ | | | | | | | | |
|  | | |  | | |  | | |
| ФГБОУ ВО «Пермский государственный  национальный исследовательский университет» | | | | | | | | |
|  | | |  | | |  | | |
|  | | Алгоритмы и анализ сложности  *Практическая работа*  **«Поиск числа в массиве»** | | | | |  | |
|  | | |  | | |  | | |
|  | Работу выполнили  студенты группы ПМИ 3-4  Гришин Н.А.  Вотинова Е.С. | | |  | Проверил  Профессор, доктор физико-математических наук  Постаногова И. Ю.  «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2020 | | |  |
|  |  | | |  |  | | |  |
| Пермь 2020 | | | | | | | | |

Оглавление

[1. Постановка задачи 3](#_Toc55843261)

[2. Исходные данные 4](#_Toc55843262)

[3. Решение 5](#_Toc55843263)

[4. Тестирование 19](#_Toc55843264)

[5. Краткие выводы 24](#_Toc55843265)

[6. Текст программы. 25](#_Toc55843266)

1. Постановка задачи

Дано: в первой строке записано количество чисел в массиве (*n*) и искомый элемент (*x*). Во второй строке перечислено *n* чисел ­ элементы массива.

Получить: индекс первого элемента, равного *x*, либо значение «–1», если такого элемента нет.

Алгоритмы:

линейный поиск;

бинарный поиск.

1. Краткое описание фактически реализованных алгоритмов

В ходе выполнения работы нами были использованы следующие алгоритмы поиска элемента в массиве:

**Линейный поиск**

Сложность: 

Начиная с первого, все элементы массива последовательно просматриваются и сравниваются с искомым. Если на каком-то шаге текущий элемент окажется равным искомому, тогда элемент считается найденным, и в качестве результата возвращается индекс этого элемента, и сразу выходим из цикла. Иначе, возвращаем значение -1.

**Бинарный поиск**

Сложность: 

Бинарный поиск производится в упорядоченном массиве. Для начала нужно отсортировать массив (алгоритм быстрой сортировки). При бинарном поиске искомый ключ сравнивается с ключом среднего элемента в массиве. Если они равны, то поиск успешен, выводим индекс элемента. В противном случае поиск осуществляется аналогично в левой или правой частях массива. Если элемент не найден, возвращаем -1.

1. Средства разработки

Программа была реализована с помощью языка высокого уровня С++ и использовалась среда разработки Visual Studio 2019.

Характеристики процессора:

* Intel core i5
* Тактовая частота 1.19 GHz
* ОЗУ 8 Гб

1. Полученные результаты

Экспериментально определено (168.322 с в случае бинарного поиска, с учетом сортировки и 0.422 с в случае линейного поиска).

На следующих графиках представлена зависимость времени реальных вычислений и теоретическим временем. Одна простая операция = 0,00000353 мс.

Рисунок - Соотношение времени в случае бинарного поиска.

Вывод: теоретическая оценка очень близка с фактическими результатами. Среднеквадратичное отклонение 0,033334193.

1. Краткие выводы
2. Текст программы.

#include <windows.h>

#include "iostream"

#include "cmath"

#include <consoleapi2.h>

#include <iomanip>

#include <vector>

using namespace std;

const int var = 18;

const int n = 6;

double b = 2.0;

double a = 1.0;

double h = 1 / 5.;

typedef vector<double> myVector;

typedef vector<myVector> myMatrix;

double\* yArr = new double[n];

//исходная функция

double F(double x)

{

if (var == 11)

return pow(3, x) + 2 \* x - 2;

if (var == 3)

return pow(5, x) - 3;

else

return pow(3, x) - 2 \* x + 5;

}

//производная функции

double DF(double x)

{

if (var == 11)

return log(3) \* pow(3, x) + 2;

if (var == 3)

return log(5) \* pow(5, x);

else

return log(3) \* pow(3, x) - 2;

}

void get\_X\_Y\_Arr(double\* xArr, double\* xArr1, double\* yArr)

{

double a = 1.0;

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

{

xArr[i] = 1.0 + i \* h;

}

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

{

xArr1[i] = 1.0 + (i + 0.5) \* h;

}

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

{

yArr[i] = F(xArr[i]);

}

}

// построение таблицы разделенных разностей

void FindDividedDifferenceMatrix(double\*\* DivDiff, double\* xArr, double\* yArr)

{

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

{

DivDiff[i][0] = xArr[i];

DivDiff[i][1] = yArr[i];

}

for (int j = 2; j <= n; j++)

{

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

DivDiff[i][j] = (DivDiff[i + 1][j - 1] - DivDiff[i][j - 1]) / (xArr[i + j - 1] - xArr[i]);

}

}

double fi0(double tau)

{

return (1 + 2 \* tau) \* pow(1 - tau, 2);

}

double fi1(double tau)

{

return tau \* pow(1 - tau, 2);

}

//погрешность интерполяции сплайнами

double FindErrorForSpline(double M4, double M5)

{

return (M4 / 384 + M5 \* h / 240) \* pow(h, 4);

}

// Метод прогонки

myVector SweepMethod(myMatrix m, myVector r, int const n)

{

double y;

myVector a(n + 1), B(n + 1), resV(n + 1);

y = m[0][0];

a[0] = -m[0][1] / y;

B[0] = r[0] / y;

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

y = m[i][i] + m[i][i - 1] \* a[i - 1];

a[i] = -m[i][i + 1] / y;

B[i] = (r[i] - m[i][i - 1] \* B[i - 1]) / y;

}

resV[n] = (r[n] - m[n][n - 1] \* B[n - 1]) / (m[n][n] + m[n][n - 1] \* a[n - 1]);

for (int i = n - 1; i >= 0; i--)

{

resV[i] = a[i] \* resV[i + 1] + B[i];

}

return resV;

}

//Интерполяция кубическим сплайном

void SplineInterpolation(double\* xArr1, double\* xArr2)

{

cout << "\nИнтерполяция кубическим сплайном\n";

double\* m = new double[n];

double df0 = DF(1), dfn = DF(2);

m[0] = df0;

m[n - 1] = dfn;

// находим параметры m методом прогонки

double\* alpha = new double[n];

double\* beta = new double[n];

alpha[1] = 0;

beta[1] = df0;

for (int j = 1; j < n - 1; j++)

{

alpha[j + 1] = -1 / (4 + alpha[j]);

beta[j + 1] = (3 \* (F(xArr1[j + 1]) - F(xArr1[j - 1])) / h - beta[j]) / (4 + alpha[j]);

}

for (int j = n - 2; j >= 0; j--)

m[j] = alpha[j + 1] \* m[j + 1] + beta[j + 1];

/\*for (int j = 1; j < n - 1; j++)

m[j] = (F(xArr1[j + 1]) - F(xArr1[j - 1])) / (2 \* h);\*/

//наибольшее значение 4-й и 5-й производной достигается в точке х = 2

double M4, M5;

M4 = (var == 3) ? pow(log(5), 4) \* 25 : pow(log(3), 4) \* 9;

M5 = (var == 3) ? pow(log(5), 5) \* 25 : pow(log(3), 5) \* 9;

cout << "M5 = " << M5 << "\nM4 = " << M4 << endl << endl;

// оценка погрешности аппроксимации 1-й производной

cout << " x[i] | " << " df/dx(x[i]) " << " m[i] " << " delta " << " оценка " << endl;

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

{

double DF\_xi = DF(xArr1[i]);

cout << setprecision(1) << xArr1[i] << " | " << setprecision(6) << DF\_xi << " | " << m[i] << " | " <<

abs(DF\_xi - m[i]) << " | " << M5 / 60 \* pow(h, 4) << endl;

}

// оценка погрешности аппроксимации функции

cout << endl << " x | " << " f(x) " << " S31(f;x) " << " Abs(f(x)-S31(f;x)) " << " Оценка " << endl;

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

{

double tau = 0.5; // (xArr2[i] - xArr1[i]) / h;

double S = fi0(tau) \* F(xArr1[i]) + fi0(1 - tau) \* F(xArr1[i + 1]) +

h \* (fi1(tau) \* m[i] - fi1(1 - tau) \* m[i + 1]);

double F\_ = F(xArr2[i]);

cout << setprecision(1) << xArr2[i] << " | " << setprecision(6) << F\_ << " | " <<

S << " | " << abs(F\_ - S) << " | " << FindErrorForSpline(M4, M5) << endl;

}

//delete[] b1;

}

//печать Таблицы разделенных разностей

void PrintDividedDifferenceMatrix(double\*\* DivDiff)

{

cout << "Таблица разделенных разностей:\n";

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

{

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

cout << fixed << setprecision(6) << setw(12) << DivDiff[i][j];

cout << endl;

}

}

// погрешность интерполяции по формуле Ньютона

double FindErrorForNewton(double x, double\* xArr, double M6)

{

double w = 1;

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

w \*= x - xArr[i];

int fact = 1;

for (int i = 2; i < n + 1; i++)

fact \*= i;

return M6 \* abs(w) / fact;

}

//Newton

void Newton(double\* xArr, double\* xArr1, double\* yArr)

{

cout << "Интерполяционная формула Ньютона \n";

double\*\* DivDiff = new double\* [n];

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

{

DivDiff[i] = new double[n + 1];

}

get\_X\_Y\_Arr(xArr, xArr1, yArr);

FindDividedDifferenceMatrix(DivDiff, xArr, yArr);

PrintDividedDifferenceMatrix(DivDiff);

double M6 = (var == 3) ? 25 \* pow(log(5), 6) : 9 \* pow(log(3), 6);

//double M6 = pow(log(3), 6) \* 9;

cout << "\nM6 = " << M6 << endl;

cout << "\nx | f(x) | Pn(x) | Delta | Оценка |\n";

for (int k = 0; k < n - 1; k++)

{

double Pn\_x = DivDiff[0][1];

double w = 1;

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

{

w \*= xArr1[k] - xArr[i - 1];

Pn\_x += DivDiff[0][i + 1] \* w;

}

double F\_ = F(xArr1[k]);

cout << setprecision(1) << xArr1[k] << " | " << setprecision(6) << F\_ << " | " <<

Pn\_x << " | " << abs(Pn\_x - F\_) << " | " << FindErrorForNewton(xArr1[k], xArr, M6) << " | \n";

}

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

delete[] DivDiff[i];

delete[] DivDiff;

}

// печать матрицы

void PrintMatrix(double\*\* A)

{

for (int i = 0; i < 3; i++)

{

for (int j = 0; j < 3; j++)

cout << fixed << setprecision(4) << setw(10) << A[i][j];

cout << endl;

}

cout << "\n";

}

// печать вектора

void PrintVector(double\* b)

{

for (int i = 0; i < 3; i++)

cout << setprecision(8) << b[i] << endl;

cout << "\n";

}

// определитель

double Determinant(double\*\* A)

{

return A[0][0] \* A[1][1] \* A[2][2] +

A[0][1] \* A[1][2] \* A[2][0] +

A[0][2] \* A[1][0] \* A[2][1] -

A[0][2] \* A[1][1] \* A[2][0] -

A[0][0] \* A[2][1] \* A[1][2] -

A[2][2] \* A[1][0] \* A[0][1];

}

// замена столбца матрицы на вектор

void ReplaceColumn(double\*\* A, int j, double\* b)

{

for (int i = 0; i < 3; i++)

A[i][j] = b[i];

}

// копирование столбца матрицы

void CopyColumn(double\*\* A, int j, double\* b)

{

for (int i = 0; i < 3; i++)

b[i] = A[i][j];

}

// решение СЛАУ методом Крамера

void CramerMethod(double\*\* A, double\* b, double\* x)

{

double det = Determinant(A);

double\* aj = new double[3];

CopyColumn(A, 0, aj);

ReplaceColumn(A, 0, b);

double det1 = Determinant(A);

ReplaceColumn(A, 0, aj);

CopyColumn(A, 1, aj);

ReplaceColumn(A, 1, b);

double det2 = Determinant(A);

ReplaceColumn(A, 1, aj);

CopyColumn(A, 2, aj);

ReplaceColumn(A, 2, b);

double det3 = Determinant(A);

x[0] = det1 / det;

x[1] = det2 / det;

x[2] = det3 / det;

}

// функция, получаемая методом среднеквадратичного приближения

double FuncBySquare(double x, double\* c)

{

return c[0] + c[1] \* x + c[2] \* pow(x, 2);

}

// среднеквадратичное приближение (дискретный вариант)

void MeanSquareApproximationForTable(double\* xArr1, double\* xArr2)

{

cout << "Дискретный вариант\n\n";

double\*\* A = new double\* [3];

for (int i = 0; i < 3; i++)

A[i] = new double[3];

for (int i = 0; i < 3; i++)

for (int j = 0; j < 3; j++)

A[i][j] = 0;

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

{

A[0][0] ++;

A[0][1] += xArr1[i];

A[0][2] += pow(xArr1[i], 2);

A[1][2] += pow(xArr1[i], 3);

A[2][2] += pow(xArr1[i], 4);

}

A[1][0] = A[0][1];

A[1][1] = A[0][2];

A[2][0] = A[0][2];

A[2][1] = A[1][2];

cout << "Матрица:\n";

PrintMatrix(A);

double\* b = new double[3];

for (int i = 0; i < 3; i++)

b[i] = 0;

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

{

b[0] += F(xArr1[i]);

b[1] += F(xArr1[i]) \* xArr1[i];

b[2] += F(xArr1[i]) \* pow(xArr1[i], 2);

}

cout << "\nВектор правых частей:\n";

PrintVector(b);

double\* c = new double[3];

CramerMethod(A, b, c);

cout << setprecision(4) << "P2(x) = " << c[0] << " + (" << c[1] << "\*x) + (" << c[2] << "\*x^2)\n";

double scalarFF = 0; // (f,f)

double scalarFG = 0; // (f,g[i])

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

{

scalarFF += pow(F(xArr1[i]), 2);

scalarFG += pow(FuncBySquare(xArr1[i], c), 2);

}

double error = sqrt(scalarFF - scalarFG);

cout << "Оценка погрешности: " << "sqrt(" << scalarFF << " - " << scalarFG << ") = " << error << endl;

for (int i = 0; i < 3; i++)

delete[] A[i];

delete[] A;

delete[] b;

delete[] c;

}

double g(double x, int n)

{

switch (n)

{

case 0: return 1;

case 1: return x;

case 2: return x \* x;

}

}

// вычисляем интеграл по формуле трапеций

double IntegrG(double a, double b, int n)

{

double eps = 0.000001;

double Integral = eps \* (F(a) \* g(a, n) + F(b) \* g(b, n)) / 2.0;

for (int i = 1; i <= (b - a) / eps - 1; i++)

Integral = Integral + eps \* F(a + eps \* i) \* g(a + eps \* i, n);

return Integral;

}

// вычисляем интеграл по формуле трапеций

double IntegrF(double a, double b)

{

double eps = 0.000001;

double Integral = eps \* (F(a) \* F(a) + F(b) \* F(b)) / 2.0;

for (int i = 1; i <= (b - a) / eps - 1; i++)

Integral = Integral + eps \* F(a + eps \* i) \* F(a + eps \* i);

return Integral;

}

double G(myVector B, double x)

{

return B[0] + B[1] \* x + B[2] \* x \* x;

}

double IntegrGG(double a, double b, myVector B)

{

// вычисляем интеграл по формуле трапеций

double eps = 0.000001;

double Integral = eps \* (G(B, a) \* G(B, a) + G(B, b) \* G(B, b)) / 2.0;

for (int i = 1; i <= (b - a) / eps - 1; i++)

Integral = Integral + eps \* G(B, a + eps \* i) \* G(B, a + eps \* i);

return Integral;

}

// среднеквадратичное приближение (непрерывный вариант)

void MeanSquareApproximationForInterval(double\* xArr2)

{

cout << "\nНепрерывный вариант\n\n";

double\*\* A = new double\* [3];

for (int i = 0; i < 3; i++)

A[i] = new double[3];

A[0][0] = 1;

A[0][1] = 3 / 2.;

A[0][2] = 7 / 3.;

A[1][0] = 3 / 2.;

A[1][1] = 7 / 3.;

A[1][2] = 15 / 4.;

A[2][0] = 7 / 3.;

A[2][1] = 15 / 4.;

A[2][2] = 31 / 5.;

cout << "Матрица:\n";

PrintMatrix(A);

double\* r = new double[3];

r[0] = IntegrG(a, b, 0);

r[1] = IntegrG(a, b, 1);

r[2] = IntegrG(a, b, 2);

cout << "\nВектор правых частей:\n";

PrintVector(r);

double\* c = new double[3];

CramerMethod(A, r, c);

cout << setprecision(4) << "P2(x) = " << c[0] << " + (" << c[1] << "\*x) + (" << c[2] << "\*x^2)\n";

double F = 0;

F = IntegrF(a, b) \* IntegrF(a, b);

cout << "\nОценка погрешности: " << "sqrt(" << IntegrF(a, b) << " - " << IntegrGG(a, b, { c[0], c[1], c[2] }) << ") = " << sqrt(IntegrF(a, b) - IntegrGG(a, b, { c[0], c[1], c[2] })) << endl << endl;

}

void ReverseInterpolation(double\* xArr, double\* yArr)

{

// таблица разделенных разностей

double\*\* DivDiff = new double\* [n];

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

DivDiff[i] = new double[n + 1];

cout << "\nОбратная интерполяция\n\n";

FindDividedDifferenceMatrix(DivDiff, yArr, xArr);

cout << "Таблица разделенных разностей:\n";

PrintDividedDifferenceMatrix(DivDiff);

double x = DivDiff[0][1];

double w = 1;

double c = F(1.5); // значение функции в середине отрезка

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

{

w \*= c - yArr[i - 1];

x += DivDiff[0][i + 1] \* w;

}

cout << "C= " << c << endl;

cout << "\nКорень: " << x << endl;

cout << "Невязка: " << setprecision(8) << abs(F(x) - c) << endl;

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

delete[] DivDiff[i];

delete[] DivDiff;

}

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

double\* xArr = new double[n];

double\* xArr1 = new double[n];

get\_X\_Y\_Arr(xArr, xArr1, yArr);

Newton(xArr, xArr1, yArr);

SplineInterpolation(xArr, xArr1);

cout << "\n Среднеквадратичное приближение\n\n";

MeanSquareApproximationForTable(xArr, xArr1);

MeanSquareApproximationForInterval(xArr1);

ReverseInterpolation(xArr, yArr);

cout << endl;

delete[] xArr;

delete[] xArr1;

delete[] yArr;

system("pause");

}