**ASSIGNMENT 1**

**ROHAN BARICK**

M. Tech (IT) 1st Year

2024ITM006

**1.Program for file creation**

#include <iostream>

#include <fstream>

#include <vector>

int main() {

    std::string fileName;

    int numEntries;

    // Get the file name and number of entries from the user

    std::cout << "Enter the name of the file: ";

    std::cin >> fileName;

    std::cout << "Enter the number of entries to generate: ";

    std::cin >> numEntries;

    // Open the file

    std::ofstream outputFile(fileName);

    // Check if the file opened successfully

    if (!outputFile.is\_open()) {

        std::cerr << "Error: Could not open output file!" << std::endl;

        return 1;

    }

    outputFile << "index,value\n"; // Header line

    // Write the data to the file

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

        int value = 100 + i; // Value generation logic, can be adjusted as required

        outputFile << i << "," << value << "\n";

    }

    outputFile.close(); // Close the file

    // Output the details to the user

    std::cout << "Data generation complete." << std::endl;

    std::cout << "File: " << fileName << std::endl;

    std::cout << "Number of entries: " << numEntries << std::endl;

    return 0;

}

***Output***

**Outcome1.**

Enter the name of the file: my\_data1.csv

Enter the number of entries to generate: 10000

Data generation complete.

File: my\_data1.csv

Number of entries: 10000

**Outcome 2.**

Enter the name of the file: my\_data2.csv

Enter the number of entries to generate: 100000

Data generation complete.

File: my\_data2.csv

Number of entries: 100000

**Outcome 3.**

Enter the name of the file: my\_data3.csv

Enter the number of entries to generate: 1000000

Data generation complete.

File: my\_data3.csv

Number of entries: 1000000

**Outcome 4.**

Enter the name of the file: my\_data4.csv

Enter the number of entries to generate: 10000000

Data generation complete.

File: my\_data4.csv

Number of entries: 10000000

**Outcome 5.**

Enter the name of the file: my\_data5.csv

Enter the number of entries to generate: 100000000

Data generation complete.

File: my\_data5.csv

Number of entries: 100000000

**2. Binary Search Program**

#include <iostream>

#include <fstream>

#include <vector>

#include <sstream>  // For stringstream to handle CSV parsing

#include <chrono>

using namespace std;

using namespace std::chrono;

// Struct to store both index and data together

struct IndexedData {

    int index;

    int value;

};

// Function for binary search on the value column

int binarySearch(const vector<IndexedData>& arr, int left, int right, int x) {

    while (left <= right) {

        int mid = left + (right - left) / 2;

        // Check if x is present at mid

        if (arr[mid].value == x)

            return mid;

        // If x greater, ignore left half

        if (arr[mid].value < x)

            left = mid + 1;

        // If x is smaller, ignore right half

        else

            right = mid - 1;

    }

    // Element is not present in array

    return -1;

}

int main() {

    string fileName;  // The file generated from the previous program

    cout << "Enter the name of the file: ";

    cin >> fileName;

    ifstream inputFile(fileName);

    // Check if the file is opened successfully

    if (!inputFile) {

        cerr << "Error opening file!" << endl;

        return 1;

    }

    vector<IndexedData> arr;

    string line;

    // Skip the header line

    getline(inputFile, line);

    // Read the CSV file line by line

    while (getline(inputFile, line)) {

        stringstream ss(line);

        string indexStr, valueStr;

        // Split by comma

        if (getline(ss, indexStr, ',') && getline(ss, valueStr)) {

            int idx = stoi(indexStr);

            int value = stoi(valueStr);

            arr.push\_back({idx, value});

        }

    }

    inputFile.close();

    // Check if data was successfully read

    int numEntries = arr.size();

    if (numEntries == 0) {

        cerr << "The file is empty or contains no valid data!" << endl;

        return 1;

    }

    int x, numTests;

    // Input the element to search for

    cout << "Enter the element (value) to search: ";

    cin >> x;

    // Input the number of tests

    numTests = 100000;

    long long totalTime = 0;  // To store the total time taken for all tests

    int result = -1;  // To store the result of the binary search

    // Get start time

    auto start = high\_resolution\_clock::now();

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

        // Perform binary search

        result = binarySearch(arr, 0, arr.size() - 1, x);

    }

    // Get end time

    auto stop = high\_resolution\_clock::now();

    // Calculate duration in nanoseconds

    auto duration = duration\_cast<nanoseconds>(stop - start).count();

    // Calculate average time

    long long avgTime = duration / numTests;

    // Output the result of the search

    if (result != -1) {

        cout << "Element found at index (file index): " << arr[result].index << " with value: " << arr[result].value << endl;

    } else {

        cout << "Element not found" << endl;

    }

    // Output the average time taken and number of entries

    cout << "Binary search performed on " << numEntries << " entries." << endl;

    cout << "Average time taken by binary search: " << avgTime << " nanoseconds" << endl;

    return 0;

}

***Output***

**Outcome 1.**

Enter the name of the file: my\_data1.csv

Enter the element (value) to search: -1

Element not found

Binary search performed on 10000 entries.

Average time taken by binary search: 250 nanoseconds

**Outcome 2.**

Enter the name of the file: my\_data2.csv

Enter the element (value) to search: -1

Element not found

Binary search performed on 100000 entries.

Average time taken by binary search: 250 nanoseconds

**Outcome 3.**

Enter the name of the file: my\_data3.csv

Enter the element (value) to search: -1

Element not found

Binary search performed on 1000000 entries.

Average time taken by binary search: 250 nanoseconds

**Outcome 4.**

Enter the name of the file: my\_data4.csv

Enter the element (value) to search: -1

Element not found

Binary search performed on 10000000 entries.

Average time taken by binary search: 250 nanoseconds

**Outcome 5.**

Enter the name of the file: my\_data5.csv

Enter the element (value) to search: -1

Element not found

Binary search performed on 100000000 entries.

Average time taken by binary search: 250 nanoseconds

**3. Fibonacci Search Program**

#include <iostream>

#include <fstream>

#include <vector>

#include <sstream> // For stringstream to handle CSV parsing

#include <chrono>

#include <cmath>   // For the Fibonacci search

using namespace std;

using namespace std::chrono;

// Struct to store both index and data together

struct IndexedData

{

    int index;

    int value;

};

// Function to perform Fibonacci search on the value column

int fibonacciSearch(const vector<IndexedData> &arr, int x)

{

    int n = arr.size();

    int fibMm2 = 0;             // (m-2)'th Fibonacci number

    int fibMm1 = 1;             // (m-1)'th Fibonacci number

    int fibM = fibMm1 + fibMm2; // m'th Fibonacci number

    // Find the smallest Fibonacci number greater than or equal to n

    while (fibM < n)

    {

        fibMm2 = fibMm1;

        fibMm1 = fibM;

        fibM = fibMm1 + fibMm2;

    }

    int offset = -1;

    // While there are elements to be inspected

    while (fibM > 1)

    {

        // Calculate the index to be compared

        int i = min(offset + fibMm2, n - 1);

        // If x is greater than the value at index i, cut the subarray from offset to i

        if (arr[i].value < x)

        {

            fibM = fibMm1;

            fibMm1 = fibMm2;

            fibMm2 = fibM - fibMm1;

            offset = i;

        }

        // If x is less than the value at index i, cut the subarray after i+1

        else if (arr[i].value > x)

        {

            fibM = fibMm2;

            fibMm1 = fibMm1 - fibMm2;

            fibMm2 = fibM - fibMm1;

        }

        // Element found at index i

        else

            return i;

    }

    // Compare the last element

    if (fibMm1 && offset + 1 < n && arr[offset + 1].value == x)

        return offset + 1;

    // Element not found

    return -1;

}

int main()

{

    string fileName; // The file generated from the previous program

    cout << "Enter the name of the file: ";

    cin >> fileName;

    ifstream inputFile(fileName);

    // Check if the file is opened successfully

    if (!inputFile)

    {

        cerr << "Error opening file!" << endl;

        return 1;

    }

    vector<IndexedData> arr;

    string line;

    // Skip the header line

    getline(inputFile, line);

    // Read the CSV file line by line

    while (getline(inputFile, line))

    {

        stringstream ss(line);

        string indexStr, valueStr;

        // Split by comma

        if (getline(ss, indexStr, ',') && getline(ss, valueStr))

        {

            int idx = stoi(indexStr);

            int value = stoi(valueStr);

            arr.push\_back({idx, value});

        }

    }

    inputFile.close();

    int numEntries = arr.size();

    if (numEntries == 0)

    {

        cerr << "The file is empty or contains no valid data!" << endl;

        return 1;

    }

    int x, numTests;

    // Input the element to search for

    cout << "Enter the element (value) to search: ";

    cin >> x;

    // Input the number of tests

    numTests = 100000;

    long long totalTime = 0; // To store the total time taken for all tests

    int result = -1;         // To store the result of the Fibonacci search

    // Get start time

    auto start = high\_resolution\_clock::now();

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

    {

        // Perform Fibonacci search

        result = fibonacciSearch(arr, x);

    }

    // Get end time

    auto stop = high\_resolution\_clock::now();

    // Calculate duration in nanoseconds

    auto duration = duration\_cast<nanoseconds>(stop - start).count();

    // Calculate average time

    long long avgTime = duration / numTests;

    // Output the result of the search

    if (result != -1)

    {

        cout << "Element found at index (file index): " << arr[result].index << " with value: " << arr[result].value << endl;

    }

    else

    {

        cout << "Element not found" << endl;

    }

    // Output the average time taken and number of entries

    cout << "Fibonacci search performed on " << numEntries << " entries." << endl;

    cout << "Average time taken by Fibonacci search: " << avgTime << " nanoseconds" << endl;

    return 0;

}

***Output***

**Outcome 1.**

Enter the name of the file: my\_data1.csv

Enter the element (value) to search: -1

Element not found

Fibonacci search performed on 10000 entries.

Average time taken by Fibonacci search: 292 nanoseconds

**Outcome 2.**

Enter the name of the file: my\_data2.csv

Enter the element (value) to search: -1

Element not found

Fibonacci search performed on 100000 entries.

Average time taken by Fibonacci search: 292 nanoseconds

**Outcome 3.**

Enter the name of the file: my\_data3.csv

Enter the element (value) to search: -1

Element not found

Fibonacci search performed on 1000000 entries.

Average time taken by Fibonacci search: 292 nanoseconds

**Outcome 4.**

Enter the name of the file: my\_data4.csv

Enter the element (value) to search: -1

Element not found

Fibonacci search performed on 10000000 entries.

Average time taken by Fibonacci search: 292 nanoseconds

**Outcome 5.**

Enter the name of the file: my\_data5.csv

Enter the element (value) to search: -1

Element not found

Fibonacci search performed on 100000000 entries.

Average time taken by Fibonacci search: 292 nanoseconds