Pseudocode Documentation

Group Two Project - Signature Assignment Project

Table of Contents

Sorting Algorithms

selection_sort(values, size)

Purpose: Sort an array of integers in ascending order using selection sort algorithm

Input:

- values: pointer to integer array
- size: number of elements in array

Output: Sorted array (in-place modification)

Pseudocode:

```
FUNCTION selection_sort(values, size):
    FOR i = 0 TO size - 2:
        minIndex = i

        // Find the minimum element in unsorted portion
        FOR j = i + 1 TO size - 1:
            IF values[j] < values[minIndex]:
                minIndex = j
            END IF
        END FOR

        // Swap minimum element with first unsorted element
        IF minIndex != i:
            SWAP values[i] WITH values[minIndex]
        END IF
    END FOR
END FUNCTION
```

Time Complexity: $O(n^2)$ **Space Complexity**: $O(1)$

Search Algorithms

binary_search(values, key, size)

Purpose: Helper function to initiate binary search on a sorted array

Input:

- values: pointer to sorted integer array
- key: value to search for
- size: number of elements in array

Output: true if found, false otherwise

Pseudocode:

```
FUNCTION binary_search(values, key, size):
    RETURN binary_search_recursive(values, key, 0, size - 1)
END FUNCTION
```

---

binary_search_recursive(values, key, start, end)

Purpose: Recursively search for a value in a sorted array

Input:

- values: pointer to sorted integer array
- key: value to search for
- start: starting index of search range
- end: ending index of search range

Output: true if found, false otherwise

Pseudocode:

```
FUNCTION binary_search_recursive(values, key, start, end):
    // Base case: element not found
    IF start > end:
        RETURN false
    END IF

    // Calculate middle index
    mid = start + (end - start) / 2

    // Base case: element found
    IF values[mid] == key:
        RETURN true
    END IF
```

```
    // Recursive case: search left half
    IF values[mid] > key:
        RETURN binary_search_recursive(values, key, start, mid - 1)
    END IF

    // Recursive case: search right half
    RETURN binary_search_recursive(values, key, mid + 1, end)
END FUNCTION
```

Time Complexity: O(log n) **Space Complexity**: O(log n) due to recursion

---

Binary File Operations

createBinaryFile(filename)

Purpose: Create a binary file with 1000 random integers between 0 and 999

Input: filename - name of file to create

Output: Binary file created on disk

Pseudocode:

```
FUNCTION createBinaryFile(filename):
    DATA_SIZE = 1000
    MIN_VALUE = 0
    MAX_VALUE = 999

    // Allocate memory for data
    data = ALLOCATE array of DATA_SIZE integers

    // Initialize random number generator
    seed = current_system_time
    random_generator = CREATE generator with seed
    distribution = CREATE uniform_distribution(MIN_VALUE, MAX_VALUE)

    // Generate random data
    FOR i = 0 TO DATA_SIZE - 1:
        data[i] = distribution(random_generator)
    END FOR

    // Write to file
    writeBinary(filename, data, DATA_SIZE)

    // Clean up
    DEALLOCATE data
```

```
    PRINT "Created " + filename + " with " + DATA_SIZE + " integers."
END FUNCTION
```

---

writeBinary(filename, values, length)

Purpose: Write an array of integers to a binary file

Input:

- filename: name of file to create
- values: pointer to integer array
- length: number of integers to write

Output: Binary file with raw integer data

Pseudocode:

```
FUNCTION writeBinary(filename, values, length):
    // Open file in binary write mode
    file = OPEN filename FOR BINARY WRITE

    IF file cannot be opened:
        PRINT "Error: Cannot create file " + filename
        RETURN
    END IF

    // Write raw bytes to file
    bytes_to_write = length * sizeof(int)
    WRITE values AS bytes_to_write bytes TO file

    IF write failed:
        PRINT "Error: Failed to write data to file " + filename
    END IF

    CLOSE file
END FUNCTION
```

Analyzer Classes

Base Class: Analyzer

Purpose: Abstract base class for all analyzers

Data Members:

- data: pointer to integer array (protected)

- size: size of array (protected)

Methods:

Constructor

```
FUNCTION Analyzer(data, size):
    this.size = size
    this.data = cloneValues(data, size)
END FUNCTION
```

Destructor

```
FUNCTION ~Analyzer():
    DEALLOCATE this.data
END FUNCTION
```

cloneValues

```
FUNCTION cloneValues(data, size):
    clone = ALLOCATE array of size integers
    FOR i = 0 TO size - 1:
        clone[i] = data[i]
    END FOR
    RETURN clone
END FUNCTION
```

analyze (pure virtual)

```
PURE VIRTUAL FUNCTION analyze():
    // Implemented by derived classes
END FUNCTION
```

StatisticsAnalyzer

Purpose: Calculate statistical measures (min, max, mean, median, mode)

Pseudocode:

```
FUNCTION StatisticsAnalyzer.analyze():
    IF size == 0:
        RETURN "Statistics:\n  No data to analyze."
    END IF

    // Sort the data
    selection_sort(data, size)
```

```
    // Calculate minimum and maximum
    minVal = data[0]
    maxVal = data[size - 1]

    // Calculate mean
    sum = 0
    FOR i = 0 TO size - 1:
        sum = sum + data[i]
    END FOR
    mean = sum / size

    // Calculate median
    IF size is even:
        median = (data[size/2 - 1] + data[size/2]) / 2.0
    ELSE:
        median = data[size/2]
    END IF

    // Calculate mode (most frequent value)
    counts = CREATE empty map
    FOR i = 0 TO size - 1:
        counts[data[i]] = counts[data[i]] + 1
    END FOR

    mode = data[0]
    maxCount = 0
    FOR EACH (value, count) IN counts:
        IF count > maxCount:
            maxCount = count
            mode = value
        END IF
    END FOR

    // Format and return results
    result = "Statistics:\n"
    result += "  Min: " + minVal + "\n"
    result += "  Max: " + maxVal + "\n"
    result += "  Mean: " + mean + "\n"
    result += "  Median: " + median + "\n"
    result += "  Mode: " + mode

    RETURN result
END FUNCTION
```

SearchAnalyzer

Purpose: Test binary search performance with random queries

Pseudocode:

```
FUNCTION SearchAnalyzer.Constructor(data, size):
    CALL parent constructor(data, size)
    selection_sort(this.data, this.size)
END FUNCTION

FUNCTION SearchAnalyzer.analyze():
    // Initialize random number generator
    seed = current_system_time
    generator = CREATE random_generator with seed
    distribution = CREATE uniform_distribution(0, 999)

    foundCount = 0
    numSearches = 100

    // Perform 100 random searches
    FOR i = 0 TO numSearches - 1:
        randomKey = distribution(generator)
        IF binary_search(data, randomKey, size):
            foundCount = foundCount + 1
        END IF
    END FOR

    // Format and return results
    result = "Search Results:\n"
    result += "  Found " + foundCount + " of " + numSearches + " random values."

    RETURN result
END FUNCTION
```

DuplicatesAnalyzer

Purpose: Find and report duplicate values in the dataset

Pseudocode:

```
FUNCTION DuplicatesAnalyzer.analyze():
    IF size == 0:
        RETURN "Duplicates Analysis:\n  No data to analyze."
    END IF

    // Sort the data
```

```
    selection_sort(data, size)

    // Count occurrences of each value
    counts = CREATE empty map
    FOR i = 0 TO size - 1:
        counts[data[i]] = counts[data[i]] + 1
    END FOR

    // Find duplicates (values appearing more than once)
    duplicateValues = CREATE empty list
    totalDuplicateCount = 0

    FOR EACH (value, count) IN counts:
        IF count > 1:
            ADD value TO duplicateValues
            totalDuplicateCount = totalDuplicateCount + count
        END IF
    END FOR

    // Build result string
    result = "Duplicates Analysis:\n"

    IF duplicateValues is empty:
        result += "  No duplicate values found."
    ELSE:
        result += "  Total duplicate values: " + SIZE(duplicateValues) + "\n"
        result += "  Total duplicate occurrences: " + totalDuplicateCount + "\n"
        result += "  Sample duplicate values:\n"

        // Show first 10 duplicates
        displayCount = MIN(SIZE(duplicateValues), 10)
        FOR i = 0 TO displayCount - 1:
            value = duplicateValues[i]
            result += "    " + value + " appears " + counts[value] + " times\n"
        END FOR

        IF SIZE(duplicateValues) > 10:
            remaining = SIZE(duplicateValues) - 10
            result += "    ... and " + remaining + " more"
        END IF
    END IF

    RETURN result
END FUNCTION
```

MissingAnalyzer

Purpose: Find values missing from the data range

Pseudocode:

```
FUNCTION MissingAnalyzer.analyze():
    IF size == 0:
        RETURN "Missing Values Analysis:\n  No data to analyze."
    END IF

    // Sort the data
    selection_sort(data, size)

    // Get range
    minVal = data[0]
    maxVal = data[size - 1]

    // Create set of all present values for fast lookup
    presentValues = CREATE empty set
    FOR i = 0 TO size - 1:
        ADD data[i] TO presentValues
    END FOR

    // Find missing values in range [minVal, maxVal]
    missingValues = CREATE empty list
    FOR i = minVal TO maxVal:
        IF i NOT IN presentValues:
            ADD i TO missingValues
        END IF
    END FOR

    // Build result string
    result = "Missing Values Analysis:\n"
    result += "  Range: [" + minVal + ", " + maxVal + "]\n"
    result += "  Total missing values: " + SIZE(missingValues) + "\n"

    IF missingValues is empty:
        result += "  All values in range are present."
    ELSE:
        result += "  Sample missing values: "

        // Show first 20 missing values
        displayCount = MIN(SIZE(missingValues), 20)
        FOR i = 0 TO displayCount - 1:
            result += missingValues[i]
            IF i < displayCount - 1:
```

```
            result += ", "
        END IF
    END FOR

    IF SIZE(missingValues) > 20:
        remaining = SIZE(missingValues) - 20
        result += ", ... (and " + remaining + " more)"
    END IF
END IF

RETURN result
END FUNCTION
```

Main Program Flow

Pseudocode:

```
FUNCTION main(argc, argv):
    // Determine filename
    IF argc == 1:
        filename = "data.bin"
        PRINT "No filename provided, using default: " + filename

        // Check if file exists
        IF file does not exist:
            PRINT "File not found. Creating " + filename + "..."
            createBinaryFile(filename)
        END IF
    ELSE IF argc == 2:
        filename = argv[1]
    ELSE:
        PRINT "Usage: program [filename.bin]"
        RETURN 1
    END IF

    // Open and read binary file
    file = OPEN filename FOR BINARY READ
    IF file cannot be opened:
        PRINT "Error: Cannot open file " + filename
        RETURN 1
    END IF

    fileSize = GET file size
    numIntegers = fileSize / sizeof(int)
```

```
    data = ALLOCATE array of numIntegers integers
    READ numIntegers integers FROM file INTO data
    CLOSE file

    PRINT "Data file: " + filename
    PRINT "Number of integers: " + numIntegers

    // Create copies for each analyzer (they modify data)
    dataForStats = COPY data
    dataForSearch = COPY data
    dataForDuplicates = COPY data
    dataForMissing = COPY data

    // Create analyzers
    analyzers = CREATE empty list
    ADD StatisticsAnalyzer(dataForStats, numIntegers) TO analyzers
    ADD SearchAnalyzer(dataForSearch, numIntegers) TO analyzers
    ADD DuplicatesAnalyzer(dataForDuplicates, numIntegers) TO analyzers
    ADD MissingAnalyzer(dataForMissing, numIntegers) TO analyzers

    // Run all analyzers
    FOR EACH analyzer IN analyzers:
        result = analyzer.analyze()
        PRINT result
        PRINT newline
    END FOR

    RETURN 0
END FUNCTION
```

Algorithm Summary

Algorithm	Type	Complexity	Purpose
Selection Sort	Sorting	$O(n^2)$	Sort data for analysis
Binary Search	Search	O(log n)	Find values efficiently
Statistical Calculations	Analysis	O(n)	Compute statistics
Duplicate Detection	Analysis	O(n)	Find repeated values
Gap Finding	Analysis	O(n log n)	Find missing values

Design Patterns Used

1. **Template Method Pattern**: Base Analyzer class defines common structure
2. **Strategy Pattern**: Different analysis strategies via polymorphism
3. **Factory-like Creation**: Creating multiple analyzer instances

---

Date: October 2025
Course: CSC252