Student 1: Vo Hoai Bao
ID: ITITIU21038

Student 2: Tran Vu Khanh Hung
ID: ITCSIU21182

Net Centric Report
Lab: 01

Table of Contents

Problem 1: Hamming

Algorithm

The program would implement two main algorithms:
First, the Hamming Distance function calculates differences between DNA sequences by comparing each position and incrementing a counter whenever characters differ (O(n) time complexity.

1. Check if both DNA strings have equal length; if not, panic with error
2. Initialize distance counter to zero
3. Iterate through each position (i) of both strings simultaneously
4. Compare characters at position i in both strings
5. If characters differ, increment distance counter
6. Return final distance count after completing iteration

Student 1: Vo Hoai Bao
ID: ITITIU21038

Student 2: Tran Vu Khanh Hung
ID: ITCSIU21182

```go
func HammingDistance(dna1, dna2 string) int {
    if len(dna1) != len(dna2) {
        panic("DNA sequences must have the same lennth!")
    }

    distance := 0
    for i := 0; i < len(dna1); i++ {
        if dna1[i] != dna2[i] {
            distance++
        }
    }
    return distance
}
```

Second, the Random DNA generator creates DNA sequences by selecting random nucleotides (A, C, G, T) for each position in a sequence of specified length.

1. Define possible nucleotides as "ACGT"
2. Seed random number generator with current timestamp
3. Create an empty byte slice of requested length
4. For each position in the slice:
    - Generate random number between 0-3
    - Select corresponding nucleotide from "ACGT"
    - Assign selected nucleotide to current position
5. Convert completed byte slice to string
6. Return generated DNA sequence

```go
func RandomDNA(length int) string {
    dnaBases := "ACGT"
    rand.Seed(time.Now().UnixNano())

    dna := make([]byte, length)
    for i := 0; i < length; i++ {
        dna[i] = dnaBases[rand.Intn(len(dnaBases))]
    }
    return string(dna)
}
```

The program compare two sequences, then generating and comparing 1000 pairs of random DNA sequences.

1. Define two example DNA sequences
2. Calculate and display the Hamming distance
3. Set parameters: 1000 tests with 18-character sequences
4. For each test (0 to 999):

Student 1: Vo Hoai Bao
ID: ITITIU21038

Student 2: Tran Vu Khanh Hung
ID: ITCSIU21182

a. Generate first random DNA sequence
b. Generate second random DNA sequence
c. Calculate Hamming distance between sequences
d. Display test number and distance result

5. Display completion message

Output

```
func main() {
    dna1 := "GAGCCTACTAACGGGAT"
    dna2 := "CATCGTAATGACGGCCT"
    fmt.Println("Test with 2 common sample: ")
    fmt.Println("DNA 1:", dna1)
    fmt.Println("DNA 2:", dna2)
    fmt.Println("Hamming Distance: ", HammingDistance(dna1, dna2))

    const numTests = 1000
    const dnaLength = 18
    fmt.Println("\nRunning 1000 random DNA tests...")

    for i := 0; i < numTests; i++ {
        randDNA1 := RandomDNA(dnaLength)
        randDNA2 := RandomDNA(dnaLength)
        result := HammingDistance(randDNA1, randDNA2)
        fmt.Println("Distance ", i, ": ", result)
    }
    fmt.Println("Completed 1000 random DNA tests.")
}
```

Screenshot 1:

```
[hopeka@hopeka hamming]$ go run hamming.go
Test with 2 common sample:
DNA 1: GAGCCTACTAACGGGAT
DNA 2: CATCGTAATGACGGCCT
Hamming Distance:  7

Running 1000 random DNA tests...
Distance  0 :  14
Distance  1 :  11
Distance  2 :  12
Distance  3 :  14
Distance  4 :  18
Distance  5 :  15
Distance  6 :  15
Distance  7 :  11
Distance  8 :  14
Distance  9 :  11
Distance  10 :  13
```

Student 1: Vo Hoai Bao
ID: ITITIU21038

Student 2: Tran Vu Khanh Hung
ID: ITCSIU21182

Screenshot 2:

```
Distance  984 :  16
Distance  985 :  14
Distance  986 :  13
Distance  987 :  12
Distance  988 :  13
Distance  989 :  13
Distance  990 :  14
Distance  991 :  12
Distance  992 :  14
Distance  993 :  15
Distance  994 :  12
Distance  995 :  9
Distance  996 :  10
Distance  997 :  15
Distance  998 :  13
Distance  999 :  17
Completed 1000 random DNA tests.
```

Problem 2: Scrabble Score

Algorithm

```go
func ScrabbleScore(word string) int {
    scrabbleValues := map[rune]int{
        'A': 1, 'E': 1, 'I': 1, 'O': 1, 'U': 1, 'L': 1, 'N': 1, 'R': 1, 'S': 1, 'T': 1,
        'D': 2, 'G': 2,
        'B': 3, 'C': 3, 'M': 3, 'P': 3,
        'F': 4, 'H': 4, 'V': 4, 'W': 4, 'Y': 4,
        'K': 5,
        'J': 8, 'X': 8,
        'Q': 10, 'Z': 10,
    }
    totalScore := 0

    word = strings.ToUpper(word)

    for _, letter := range word {
        if val, exists := scrabbleValues[letter]; exists {
            totalScore += val
        }
    }
    return totalScore
}
```

The program calculates the Scrabble score of a given word.

1. Create a map that pass the values to each letter based on Scrabble rules
2. Convert the given input string to uppercase to make sure case insensitivity
3. Initialize total score, which will store the final calculated score
4. Loop through each character in the input string
    a. If the character exists in the map, retrieve its score

Student 1: Vo Hoai Bao
ID: ITITIU21038

Student 2: Tran Vu Khanh Hung
ID: ITCSIU21182

b. Add the score to total score

5. Return the final score after iterating through all character

Output

```
Codeium: Refactor | Explain | Generate GoDoc | X
31  func main() {
32      var input string
33      fmt.Println("Input your word or string for Scrabble Score:")
34      fmt.Scanln(&input)
35
36      score := ScrabbleScore(input)
37      fmt.Printf("Scrabble Score of'%s' is: %d\n", input, score)
38  }
39
```

```
[hopeka@hopeka scrabble]$ go run scrabble.go
Input your word or string for Scrabble Score:
cabbage
Scrabble Score of'cabbage' is: 14
```

Student 1: Vo Hoai Bao
ID: ITITIU21038

Student 2: Tran Vu Khanh Hung
ID: ITCSIU21182

Problem 3: Luhn

Algorithm

```go
func isValidLuhn(number string) bool {
    var digits []int

    // loc ra cac chi so tu chi so dau vao
    for _, char := range number {
        if unicode.IsDigit(char) {
            digits = append(digits, int(char-'0'))
        }
    }

    // neu so luong chu so nho hon 10, khong hop le
    if len(digits) <= 10 {
        fmt.Println("Credit number must be larger than 10")
        return false
    }

    sum := 0
    double := false

    for i := len(digits) - 1; i >= 0; i-- {
        n := digits[i]
        if double {
            n *= 2
            if n > 9 {
                n -= 9
            }
        }
        sum += n
        double = !double
    }

    return sum%10 == 0
}
```

The code implements the Luhn algorithm, used for validating credit card numbers.

1. Extract all digits from the input string, removing spaces and non-digit characters.
2. Check if the number of digits is greater than 10, if not, return false
3. Initialize sum to 0 and a boolean flag for altering digits
4. Starting from the rightmost digit:
    a. If the current position needs doubling:
        i. Double the digit
        ii. if the doubled value is greater than 0, subtract 9
    b. Add the resulting digit to the running sum

Student 1: Vo Hoai Bao
ID: ITITIU21038

Student 2: Tran Vu Khanh Hung
ID: ITCSIU21182

   c. Toggle the doubling flag for the next iteration
5. Check if the final sum is divisible by 10
6. Return true if divisible by 10, false otherwise

Output

```go
func main() {
    testNumbers := []string{
        "4539 3195 0343 6467",
        "8273 1232 7352 0569",
        "79927398713",
    }

    for i := 0; i < len(testNumbers); i++ {
        if isValidLuhn(testNumbers[i]) {
            fmt.Println(testNumbers[i], "- This number is valid!")
        } else {
            fmt.Println(testNumbers[i], "- This number is not valid!")
        }
    }
```

```
[hopeka@hopeka Luhn]$ go run luhn.go
4539 3195 0343 6467 - This number is valid!
8273 1232 7352 0569 - This number is not valid!
79927398713 - This number is valid!
```

Problem 4: Minesweeper

Algorithm

The program generates a Minesweeper game board with randomly placed mines and calculates the number of adjacent mines for each empty cell.

1. Initialize the Mineflied by creating a 2D board and fillingl all cells with dots to represent empty spaces
2. Place the random seed to make sure different placementws each run
3. Place the mines in random positions on the board as ‘*‘
   a. Checking if the cell is not already a mine, place one and increment the mine counter

Student 1: Vo Hoai Bao
ID: ITITIU21038

Student 2: Tran Vu Khanh Hung
ID: ITCSIU21182

```
Codeium: Refactor | Explain | ×
14  func generateMinefield() [][]rune {
15      board := make([][]rune, rows)
16      for i := range board {
17          board[i] = make([]rune, cols)
18          for j := range board[i] {
19              board[i][j] = '.'
20          }
21      }
22      rand.Seed(time.Now().UnixNano())
23      for mines := 0; mines < mineCount; {
24          r, c := rand.Intn(rows), rand.Intn(cols)
25          if board[r][c] != '*' {
26              board[r][c] = '*'
27              mines++
28          }
29      }
30      return board
31  }
```

4. Count adjacent mines for each cell by iterating over the 8 directions and checking if the neighboring cell contains a mine
5. Return the total count of mines around the cell

```
34  func countMines(board [][]rune, r, c int) int {
35      count := 0
36      directions := []struct{ dr, dc int }{
37          {-1, -1}, {-1, 0}, {-1, 1}, {0, -1}, {0, 1}, {1, -1}, {1, 0}, {1, 1},
38      }
39
40      for _, d := range directions {
41          nr, nc := r+d.dr, c+d.dc
42          if nr >= 0 && nr < rows && nc >= 0 && nc < cols && board[nr][nc] == '*' {
43              count++
44          }
45      }
46      return count
47  }
```

6. Update the board with mine counts by iterating through all cells in the board
    a. If a cell is empty, call the function countMines to determine the number of adjacent mines
    b. If at least one mine is found nearby, update the cell with the corresponding number from 1 to 8

Student 1: Vo Hoai Bao
ID: ITITIU21038

Student 2: Tran Vu Khanh Hung
ID: ITCSIU21182

```
func updateBoard(board [][]rune) {
    for r := 0; r < rows; r++ {
        for c := 0; c < cols; c++ {
            if board[r][c] == '.' { // Nếu là ô trống, tính số mìn xung quanh
                mineCount := countMines(board, r, c)
                if mineCount > 0 {
                    board[r][c] = rune('0' + mineCount)
                }
            }
        }
    }
}

```

7. Print the final minefield by looping through each row of the board and printing each cell in a formatted way to display the board clearly

```
// Hàm in bảng
Codeium: Refactor | Explain | ×
func printBoard(board [][]rune) {
    for _, row := range board {
        for _, cell := range row {
            fmt.Printf("%c ", cell)
        }
        fmt.Println()
    }
}
```

Output

```
func main() {
    // Tạo và cập nhật bảng mìn
    board := generateMinefield()
    updateBoard(board)

    // In bảng kết quả
    printBoard(board)
}
```

Student 1: Vo Hoai Bao
ID: ITITIU21038

Student 2: Tran Vu Khanh Hung
ID: ITCSIU21182

```
[hopeka@hopeka minesweeper]$ go run ms.go
. 2 * 2 1 2 * 2 2 * 1 . 1 * 1 1 2 * 1 1 2 * * 3 1
1 3 * 3 2 * 4 * 3 2 2 . 1 1 1 1 * 2 1 1 * 5 * 4 *
3 * 4 4 * 4 * 2 2 * 2 1 . 1 1 3 3 3 1 1 1 3 * 3 1
* * 4 * * 3 1 1 1 2 * 1 . 1 * 3 * * 1 . . 1 1 1 .
3 * 4 3 3 2 1 1 . 2 2 2 . 1 2 * 3 2 1 . . . 1 1 1
2 2 3 * 1 1 * 1 1 3 * 2 1 1 2 1 1 . 1 2 2 1 1 * 1
1 * 2 1 1 1 2 2 2 * * 3 2 * 1 . . . 1 * * 2 3 2 2
1 1 1 . . . 1 * 3 3 3 * 2 1 1 . . 1 2 3 4 * 3 * 1
. 1 1 1 1 1 2 2 * 1 1 1 2 2 2 1 . 1 * 2 3 * 3 1 1
. 1 * 1 1 * 1 1 1 1 . . 1 * * 2 1 1 2 * 2 2 3 2 1
. 1 1 1 1 1 1 . 1 1 2 1 2 3 5 * 3 1 2 1 1 1 * * 1
1 2 1 1 . . . . 1 * 4 * 2 1 * * 3 * 1 . . 2 3 3 1
* 2 * 2 1 1 1 2 2 4 * * 2 1 3 3 3 1 1 . . 1 * 2 1
2 3 3 * 2 2 * 2 * 4 * 4 2 2 3 * 2 1 1 . 1 2 3 * 2
1 * 3 4 * 3 1 3 2 4 * 2 1 * * 3 4 * 2 . 1 * 2 3 *
1 2 * 4 * 2 . 1 * 2 1 1 2 3 5 * 4 * 2 . 1 2 2 3 *
1 2 3 * 3 2 2 2 2 1 . . 1 * 3 * 3 1 1 1 1 2 * 2 1
2 * 4 2 2 * 2 * 1 1 2 2 2 1 2 1 1 . . 1 * 3 3 3 1
2 * * 1 1 1 2 1 1 2 * * 1 . . . . . . 1 1 2 * * 1
1 2 2 1 . . . . . 2 * 3 1 . . . . . . . . 1 2 2 1
```

Problem 5: Matching Brackets

Algorithm

```go
func isValidBrackets(s string) bool {
    stack := []rune{}
    bracketMap := map[rune]rune{')': '(', '}': '{', ']': '['}

    for _, char := range s {
        switch char {
        case '(', '{', '[':
            stack = append(stack, char)
        case ')', '}', ']':
            if len(stack) == 0 || stack[len(stack)-1] != bracketMap[char] {
                return false
            }
            // pop from stack
            stack = stack[:len(stack)-1]
        }
    }

    return len(stack) == 0
}
```

The program checks if a given string contains valid, balanced brackets using a stack.

1. Create an empty stack to store opening brackets
2. Define a mapping that associates closing brackets with their corresponding
3. Loop through each character in the input string
    a. If the character is an opening bracket push it onto the stack

Student 1: Vo Hoai Bao
ID: ITITIU21038

Student 2: Tran Vu Khanh Hung
ID: ITCSIU21182

   b. if the character is a closing bracket
      i. Check if the stack is empty, if so, return false
      ii. Compare the top of stack with the opening bracket using map
      iii. If they match, pop the top element from the stack, otherwise, return false because of mismatched brackets
4. After processing all characters, check if the stack is empty
   a. If empty, return true as valid
   b. If not. return false as invalid

Output

```
27  func main() {
28      // Các test case
29      tests := []string{
30          "([]{})",
31          "([)]",
32          "{[()()]}",
33          "{{[[(())]]}}",
34          "{[}",
35      }
36                              var tests []string
37      for _, test := range tests {
38          fmt.Printf("String \"%s\" -> %v\n", test, isValidBrackets(test))
39      }
40  }
```

```
[hopeka@hopeka matching_brackets]$ go run mb.go
String "([]{})" -> true
String "([)]" -> false
String "{[()()]}" -> true
String "{{[[(())]]}}" -> true
String "{[}" -> false
[hopeka@hopeka matching_brackets]$
```