UNIVERSITAT DE LLEIDA
Escola Politècnica Superior
Grau en Enginyeria Informàtica
Algorítmica i complexitat

Pràctica 1

Ian Palacín Aliana
GM3

Professorat : Jordi Planes Cid, Aitor Corchero Rodríguez
Data : Dissabte 07 d'Abril

Contents

1 Senderscribe

L'algoritme de senderscribe, una vegada agafades les dades de terminal o mitjançant un fitxer, el que fa és fer un pseudo-CRC, és a dir, fa la operació mòdul 4 al codi asci de cada caràcter de les dades. Cada mòdul el passa a binari, i de la concatenació de mòduls en binari ho converteix a hexadecimal. Una vegada obtingut l'hexadecimal, fa una suma de validació al text original més la clau hexadecimal obtinguda, que també la passa a hexadecimal. Una vegada obtingudes les dos claus, les concatena a les dades separant-ho amb espais. Això és el que envia a la sortida.

1.1 Disseny recursiu

```
''' SENDERSCRIBE RECURSIVE '''
FUNCTION raw_input():
    ''' input from file OR stdin '''
    IF len(sys.argv) = 3:
        file_in <- open(sys.argv[1], "r")
        file_in.close()
        RETURN file_in.read()
    ENDIF
    RETURN input()
ENDFUNCTION

FUNCTION encoded_output(encoded_data):
    ''' output to file OR stdout '''
    IF len(sys.argv) = 3:
        file_out <- open(sys.argv[2], "w")
        file_out.write(encoded_data)
        file_out.close()
    ELSE:
        OUTPUT encoded_data
    ENDIF
ENDFUNCTION

FUNCTION encode_pieces(raw_data, checksum, binary_code):
    ''' encode to binary the data AND does the checksum for raw_data '''
    IF raw_data is empty:
        RETURN raw_data, checksum, binary_code
    ENDIF
    character <- raw_data[0]
    checksum += ord(character)
    binary_code += str('{0:02b}'.format(ord(character)%4))
    RETURN encode_pieces(raw_data[1:], checksum, binary_code)
ENDFUNCTION
```

```
FUNCTION checksum_code(hex_code, checksum):
    ''' does the checksum for the hexadecimal encoded part '''
    IF hex_code is empty:
        RETURN checksum
    ENDIF
    checksum += ord(hex_code[0])
    RETURN checksum_code(hex_code[1:], checksum)
ENDFUNCTION

MAIN:
    RAW_DATA <- raw_input()
    RAW_DATA <- RAW_DATA.rstrip(”\n\r”)
    CHECKSUM <- 0
    BINARY_CODE <- ”1”
    [], CHECKSUM, BINARY_CODE <- encode_pieces(RAW_DATA, CHECKSUM, BINARY_CODE)
    HEX_CODE <- format(int(BINARY_CODE, 2), 'x').upper()
    CHECKSUM <- checksum_code(HEX_CODE, CHECKSUM)
    ENCODED_DATA <- RAW_DATA + ” ” + HEX_CODE + ” ” +
                    + str(format(CHECKSUM, 'x')).upper()
    encoded_output(ENCODED_DATA)
ENDMAIN
```

1.2 Disseny iteratiu

```
’’’ SENDERSCRIBE ITERATIVE ’’’
FUNCTION raw_input():
    ’’’ input from file OR stdin ’’’
    IF len(sys.argv) = 3:
        file_in <- open(sys.argv[1], ”r”)
        file_in.close()
        RETURN file_in.read()
    ENDIF
    RETURN input()
ENDFUNCTION

FUNCTION encoded_output(encoded_data):
    ’’’ output to file OR stdout ’’’
    IF len(sys.argv) = 3:
        file_out <- open(sys.argv[2], ”w”)
        file_out.write(encoded_data)
        file_out.close()
    ELSE:
        OUTPUT encoded_data
    ENDIF
ENDFUNCTION

MAIN:
    RAW_DATA <- raw_input()
    RAW_DATA <- RAW_DATA.rstrip(”\n\r”)
    CHECKSUM <- 0
    BINARY_CODE <- ”1”
    for character in RAW_DATA:
        CHECKSUM += ord(character)
        BINARY_CODE += str(’{0:02b}’.format(ord(character)%4))
    ENDFOR
    HEX_CODE <- format(int(BINARY_CODE, 2), ’x’).upper()
    for character in HEX_CODE:
        CHECKSUM += ord(character)
    ENDFOR
    ENCODED_DATA <- RAW_DATA + ” ” + HEX_CODE + ” ” +
                                   + str(format(CHECKSUM, ’x’)).upper()
    encoded_output(ENCODED_DATA)
ENDMAIN
```

1.3 Cost teòric recursiu

L'algoritme recursiu de senderscribe té dos funcions recursives i 0 bucles. Podem dir que **n** és la llargada de l'string raw_data, i que **cx** és el cost de la línia de codi x. Donades aquestes condicions tenim que:
c1*(n-1) + c2*(n-1) + c3*1 + c4*(n-1) + c5*(n-1) + c6*(n-1) + c7*(n-1)

```
1 def encode_pieces(raw_data, checksum, binary_code):
2     if not raw_data:
3         return raw_data, checksum, binary_code
4     character = raw_data[0]
5     checksum += ord(character)
6     binary_code += str('{0:02b}'.format(ord(character)%4))
7     return encode_pieces(raw_data[1:], checksum, binary_code)
```

I pel segon bucle, on **n** és la llargada de hex_code:
c1*(n-1) + c2*(n-1) + c3*1 + c4*(n-1) + c5*(n-1)

```
1 def checksum_code(hex_code, checksum):
2     if not hex_code:
3         return checksum
4     checksum += ord(hex_code[0])
5     return checksum_code(hex_code[1:], checksum)
```

Dont de que cada funció es crida des del main una única vegada, que no hi ha cap bucle, i que el cost de les crides **cx** són inferiors a O(n), tenim que el cost de la part recursiva de senderscribe pertany a **O(n)**

1.4 Cost teòric iteratiu

L'algoritme de senderscribe iteratiu consta de 2 bucles, tenim que pel primer bucle, el cost teòric seria:
c1*(n+1) + c2*n + c3*n

```
1 for character in RAW_DATA:
2         CHECKSUM += ord(character)
3         BINARY_CODE += str('{0:02b}'.format(ord(character)%4))
```

En segon lloc, pel següent bucle, seria:
c1*(n+1) + c2*n

```
1     for character in HEX_CODE:
2         CHECKSUM += ord(character)
```

Assumint que les operacions **cx** (c1,c2,...) tenen un cost inferior a O(n), el primer bucle pertany a **O(n)**, al igual que el segon, fent així que el cost de senderscribe iteratiu sigui d'**O(n)**

1.5 Cost experimental

Per raons de problemes amb la llibreria matplotlib a última hora, no he pogut col·locar els gràfics amb els temps dels costos experimentals.

2 Receiverscribe

Receiverscribe torna a convertir les dades obtingudes a les claus parlades anteriorment, i tornant a fer el mòdul pot saber si hi ha hagut un error i on.

2.1 Disseny recursiu

```
''' RECEIVERSCRIBE RECURSIVE '''
FUNCTION encoded_input():
    ''' input from file or stdin '''
    IF len(sys.argv) = 3:
        file_in <- open(sys.argv[1], ”r”)
        encoded_data <- file_in.read()
        file_in.close()
        RETURN encoded_data.rstrip(”\n\r”)
    ENDIF
    RETURN input()
ENDFUNCTION

FUNCTION decoded_output(result):
    ''' output to file or stdout '''
    IF len(sys.argv) = 3:
        file_out <- open(sys.argv[2], ’w’)
        file_out.write(result)
        file_out.close()
    ELSE:
        OUTPUT result
    ENDIF
ENDFUNCTION

FUNCTION checksum_code(hex_code, checksum):
    ''' checksum of the hex_code part '''
    IF hex_code is empty:
        RETURN checksum
    ENDIF
    checksum += ord(hex_code[0])
    RETURN checksum_code(hex_code[1:], checksum)
ENDFUNCTION

FUNCTION scan_data(raw_data, checksum_calculated, binary_code, counter, location
    ''' converts AND scans the data in order to detect an error AND its location
    IF raw_data is empty:
        RETURN checksum_calculated, location
    ENDIF
    character <- raw_data[0]
    checksum_calculated += ord(character)
    IF ord(character)%4 != int(binary_code[2*counter:2*counter+2], 2):
        checksum_calculated -= ord(character)
        location <- counter
    ENDIF
    RETURN scan_data(raw_data[1:], checksum_calculated, binary_code,
                     counter+1, location)
ENDFUNCTION

IF __name__ = "__main__":
    ENCODED_DATA <- encoded_input()
    WALL_1 <- ENCODED_DATA.rfind(' ')
    WALL_2 <- ENCODED_DATA.rfind(' ', 0, WALL_1)
    try:
        CHECKSUM_PASSED <- int(ENCODED_DATA[WALL_1+1:], 16)
        RAW_DATA <- ENCODED_DATA[0:WALL_2]
        HEX_CODE <- ENCODED_DATA[WALL_2+1:WALL_1]
        INT_CODE <- int(HEX_CODE, 16)
        BINARY_CODE <- str(bin(INT_CODE)[2:])[1:]
    except ValueError as error:
        IF len(sys.argv) = 3:
            FILE_O <- open(sys.argv[2], 'w')
            FILE_O.write("KO")
            FILE_O.close()
            sys.exit()
        ELSE:
            OUTPUT "KO"
            sys.exit()
        ENDIF
    COUNTER <- 0
    LOCATION <- -1
    CHECKSUM_CALCULATED <- 0
    CHECKSUM_CALCULATED, LOCATION <- scan_data(RAW_DATA, CHECKSUM_CALCULATED,
                                               BINARY_CODE, COUNTER, LOCATION)
    CHECKSUM_CALCULATED <- checksum_code(HEX_CODE, CHECKSUM_CALCULATED)
    IF LOCATION != -1:
        CORRECTED_CHARACTER <- chr(CHECKSUM_PASSED - CHECKSUM_CALCULATED)
        RESULT <- "KO\n" + str(LOCATION) + " " + CORRECTED_CHARACTER
    ELSEIF CHECKSUM_PASSED != CHECKSUM_CALCULATED:
        RESULT <- "KO"
    ELSE:
        RESULT <- "OK"
    ENDIF
    decoded_output(RESULT)
```

2.2 Disseny iteratiu

```
''' RECEIVERSCRIBE ITERATIVE '''
FUNCTION encoded_input():
    ''' input from file OR stdin '''
    IF len(sys.argv) = 3:
        file_in <- open(sys.argv[1], ”r”)
        encoded_data <- file_in.read()
        file_in.close()
        RETURN encoded_data.rstrip(”\n\r”)
    ENDIF
    RETURN input()
ENDFUNCTION

FUNCTION decoded_output(result):
    ''' output to file OR stdout '''
    IF len(sys.argv) = 3:
        file_out <- open(sys.argv[2], 'w')
        file_out.write(result)
        file_out.close()
    ELSE:
        OUTPUT result
    ENDIF
ENDFUNCTION

MAIN:
    ENCODED_DATA <- encoded_input()
    WALL_1 <- ENCODED_DATA.rfind(' ')
    WALL_2 <- ENCODED_DATA.rfind(' ', 0, WALL_1)
    try:
        CHECKSUM_PASSED <- int(ENCODED_DATA[WALL_1+1:], 16)
        RAW_DATA <- ENCODED_DATA[0:WALL_2]
        HEX_CODE <- ENCODED_DATA[WALL_2+1:WALL_1]
        INT_CODE <- int(HEX_CODE, 16)
        BINARY_CODE <- str(bin(INT_CODE)[2:])[1:]
    except ValueError as error:
        IF len(sys.argv) = 3:
            FILE_O <- open(sys.argv[2], 'w')
            FILE_O.write(”KO”)
            FILE_O.close()
            sys.exit()
        ELSE:
            OUTPUT ”KO”
            sys.exit()
        ENDIF
    COUNTER <- 0
    LOCATION <− −1
    CHECKSUM_CALCULATED <− 0
    for character in RAW_DATA:
        CHECKSUM_CALCULATED += ord(character)
        IF ord(character)%4 != int(BINARY_CODE[2*COUNTER:2*COUNTER+2], 2):
            CHECKSUM_CALCULATED −= ord(character)
            LOCATION <− COUNTER
        ENDIF
        COUNTER += 1
    ENDFOR
    for character in HEX_CODE:
        CHECKSUM_CALCULATED += ord(character)
    ENDFOR
    IF LOCATION != −1:
        CORRECTED_CHARACTER <− chr(CHECKSUM_PASSED − CHECKSUM_CALCULATED)
        RESULT <− ”KO\n” + str(LOCATION) + ” ” + CORRECTED_CHARACTER
    ELSEIF CHECKSUM_PASSED != CHECKSUM_CALCULATED:
        RESULT <− ”KO”
    ELSE:
        RESULT <− ”OK”
    ENDIF
    decoded_output(RESULT)
ENDMAIN
```

2.3 Cost teòric recursiu

En la part recursiva de reciverscribe tenim:
c1*(n-1) + c2*(n-1) + c3*1 + c4*(n-1) + c5*(n-1)

```
def checksum_code(hex_code, checksum):
     if not hex_code:
        return checksum
    checksum += ord(hex_code[0])
    return checksum_code(hex_code[1:], checksum)
```

I pel segon bucle:
c1*(n-1) + c2*(n-1) + c3*1 + c4*(n-1) + c5*(n-1) + c6*(n-1) + c7*1 + c8*1 + c9*(n-1)

```
def scan_data(raw_data, checksum_calculated, binary_code, counter, location):
    if not raw_data:
        return checksum_calculated, location
    character = raw_data[0]
    checksum_calculated += ord(character)
    if ord(character)%4 != int(binary_code[2*counter:2*counter+2], 2):
        checksum_calculated -= ord(character)
        location = counter
    return scan_data(raw_data[1:], checksum_calculated, binary_code,
                   counter+1, location)
```

De la mateixa manera que en la part recursiva de l'apartat anterior, com que les funcions només es criden una única vegada des del main, el cost serà, de nou, **O(n)**

2.4 Cost teòric iteratiu

En la part iterativa del receiverscribe tenim també dos bucles. Donades les condicions establertes en la part iterativa de l'apartat anterior on tenim que:
c1*(n+1) + c2*n + c3*n + c4*1 + c5*1 * c6*n

```
for character in RAW_DATA:
        CHECKSUM_CALCULATED += ord(character)
        if ord(character)%4 != int(BINARY_CODE[2*COUNTER:2*COUNTER+2], 2):
            CHECKSUM_CALCULATED -= ord(character)
            LOCATION = COUNTER
        COUNTER += 1
```

I pel segon bucle:
c1*(n+1) + c2*n

```
   for character in HEX_CODE:
       CHECKSUM_CALCULATED += ord(character)
```

De igual forma que en la part de senderscribe, assumint que el cost de **cx** és inferior a O(n), tenim que la part iterativa de receiverscribe pertan a **O(n)**