CSE331 - Computer Organization

Homework 2 - Report

1901042692

Ahmet Tuğkan Ayhan

Procedures and Their Brief Explanation:

findLIS

- findLIS is the main procedure which calculates the subsequences of the given array. It also calls **isIncrementing, copyArray, printArray** procedures. (nested)
- **Arguments :** a1 (array address), a2 (array size), a3 (sequence array address)
- **Return Value :** v0 (size of the longest increasing subsequence array)

printLongest

- It just calls **printArray** but this procedure also adds “Longest Increasing Subsequence Array : ” string in front of the array before calling printArray.
- **Arguments :** a0 (lisArr address), a1 (size of the lisArr)
- **Return Value :** No return value
- lisArr : Longest Increasing Subsequence Array

copyArray

- This procedure gets two array addresses and copies first array elements to the second array. It is used in **findLIS** to copy seqArr elements to lisArr.
- **Arguments :** a0 (first array address), a1 (second array address), a2 (size of the first array)
- **Return Value :** No return value
- seqArr : Subsequence Array

isIncrementing

- If numbers inside of the given array address is in increasing order returns 1, else returns 0
- **Arguments :** a0 (array address), a1 (size of the array)
- **Return Value :** v0 (equals 1 if incrementing, else 0)

getSize

- Looks current number in the array, if it is between 0 < x < 100, then it increments current size(0 at first) and address of the array until it finds a number outside of this boundry.
- In .data part when you put **arr2** right after **arr** in memory there will be no value outside boundry between **arr - arr2** because of that size will be calculated wrong. To prevent it, I put empty values between them.
- **Arguments :** a0 (array address)
- **Return Value :** v0 (size of the array)

printArray

- It calls a basic while loop. Until t2 (which is 'i' in c code) is not equal to size of the given array, then print value at the current location and increment index
- **Arguments :** a1 (size of the array), a2 (address of the array)
- **Return Value :** No return value

Procedures' Pseudo Codes and their Time&Space Complexity

findLIS:

```
arr      : address of the array
size     : size of that array
index    : 0 (index for that array)
sNum  :  0 (number of subsequences so far)
maxS  :  2^size (max possible num of subseq)
seqArr:  address of the subSequence array
sSize  :  0 (size of the seqArr)
lSize  :  size of the longest increasing subsequence array

while(sNum < maxS)
        for(i = 0; i<size; i++)
                if(sNum AND (1 << i))
                        seqArr[sSize] = arr[i]

flag = isIncrementing(seqArr)
if(flag == 1)
        printArray(seqArr, sSize)
if(lSize < sSize)
        lSize = sSize
        copyArray(seqArr, lisArr, sSize)
sNum++
sSize = 0
```

Time Complexity : $O(2^n * n)$! Because outer while loop is iterated 2^n times and inner for loop is iterated n times. **printArray** is O(n) and isIncrementing is O(n^2) but

$O(2^n * n) + O(n^2) + O(n) = O(2^n * n)$ so they don't change complexity of the procedure.

Space Complexity : O(1) ! Because procedure doesn't allocate any extra space. They always use what is given from the start

---

printLongest:

```
arr     : address of the array
size    : size of the array
print("Longest Increasing Subsequence Array : ")
printArray(arr, size)
```

```
print("\n")
```

Time Complexity : O(n) ! Because print operation is O(1) and **printArray** procedure is O(n). If we sum, O(1) + O(n) = O(n)

Space Complexity : O(1) ! Because procedure doesn't allocate any extra space.

---

copyArray:

```
arr     : address of the array
cArr    : address of the copy array
size    : size of the base array
i       : 0 (index for loop)

while (i != size)
        cArr[i] = arr[i]
        i++
```

Time Complexity : O(n) ! Because print operation is O(1) and while loop iterates n times.

Space Complexity : O(1) ! Because procedure doesn't allocate any extra space.

---

isIncrementing:

```
arr     : address of the array
size    : size of the array
i       : 0 (index for outer loop)
j       : 0 (index for inner loop)
flag    : true

while (i != size)
        j = i + 1
        while (j != size)
                if(arr[j] < arr[])
                        flag = false
                j++
        i++

return flag
```

Time Complexity : O(n^2) ! Because outer loop iterates n times and inner loop iteraters n/2 times which is still O(n). So it becomes O(n*n) = O(n^2)

Space Complexity : O(1) ! Because procedure doesn't allocate any extra space.

---

getSize:

MAX_VAL : 100 (Max value that can be assigned to an array)
arr : address of the array
cSize : 0 (current size of the array)
cVal : 1 (current value)

```
while (cVal != 0) and (cVal < 100):
        cSize++
        cVal = arr[cSize]

return cSize
```

Time Complexity : O(n) ! Because while loop iterates n times which equals size of the array

Space Complexity : O(1) ! Because procedure doesn't allocate any extra space.

---

printArray:

arr : address of the array
size : 0 (size of the array)
cSize : 0 (current size oft the array)
cVal : 0 (current value)

```
while cSize != size:
        cVal = arr[cSize]
        print(" " + cVal)
        cSize++

print("Size : " + size + "\n")
```

Time Complexity : O(n) ! Because while loop iterates n times which equals size of the array and print and assign operation takes O(1) time

Space Complexity : O(1) ! Because procedure doesn't allocate any extra space.

Result of Test Cases and Explanation

```
main:
        ## ------------------------ 1. Array ------------------------ ##
        # 1) Find size of the array
        la      $a0,    arr             # a0 = address of the array
        jal     getSize                 # v0 = size of the array
        # 2) Find  subsequences of arr and print them
        la      $a1,    arr             # a1 = address of the array
        move    $a2,    $v0             # a2 = v0 (came from getSize)
        la      $a3,    seqArr          # a3 = address of the seqArr
        jal     findLIS                 # Finds Longest Increasing Subsequence (prints rest of them)
        # 3) Print Longest Increasing Subsequence Array
        la      $a0,    lisArr          # a0 = lisArr address
        move    $a1,    $v0             # v0 = size of the lisArr (returned from findLIS)
        jal     printLongest            # Prints lisArr and it's size to the console
```

In main, first I got size of the input array using **getSize** procedure. After that, by using v0 (which return value from getSize) I called **findLIS** procedure to find longest increasing subsequence array(lisArr) and it's size. Finally, I printed the result. I repeated this 6 times for different arr inputs. These are the arrays I tried:

```
# Input Arrays
arr:            .word   3, 10, 7, 9, 4, 11
empty:          .word   0
arr2:           .word   2, 5, 3, 95, 24, 1, 22, 10, 88
empty2:         .word   0
arr3:           .word   4, 11, 59, 12, 53, 51 , 22 , 44 , 5
empty3:         .word   0
arr4:           .word   11, 4, 9, 10, 3, 7, 51, 14, 22
empty4:         .word   0
arr5:           .word   90, 80, 70, 60, 50, 40, 30, 20, 10, 1
empty5:         .word   0
arr6:           .word   1, 2, 3, 4, 5
empty6:         .word   0
```

I explained why I put empty values between them in the code. Because I didn't implement reading and writing file part, I directly put values here. Other than that, my code prints all increasing subsequences and their sizes. Here is output for first array (3, 10, 7, 9, 4, 11):

```
, Size : 0
3 , Size : 1
10 , Size : 1
3 10 , Size : 2
7 , Size : 1
3 7 , Size : 2
9 , Size : 1
3 9 , Size : 2
7 9 , Size : 2
3 7 9 , Size : 3
4 , Size : 1
3 4 , Size : 2
11 , Size : 1
3 11 , Size : 2
10 11 , Size : 2
3 10 11 , Size : 3
7 11 , Size : 2
3 7 11 , Size : 3
9 11 , Size : 2
3 9 11 , Size : 3
7 9 11 , Size : 3
3 7 9 11 , Size : 4
4 11 , Size : 2
3 4 11 , Size : 3
Longest Increasing Subsequence Array : 3 7 9 11 , Size : 4
```