Computer Organization CSE 331 - Homework 2

Đuro Radusinović

November 4, 2021

Abstract

This is implementation of longest subsequence dynamic programming algorithm that takes input from a file and provides a file output of the result in MIPS MARS.

1 Program's Logic

In this assignment I used the following logic.

- I use an additional array with n elements. Numbers in this array represent the length of the leftside's ascending subsequence they are part of.
In the begining since each element on its own is part of an ascending subsequence value each element of this array is 1.
Example:

Array: 4 -1 3 9 11
LengthsArray: 1 1 1 1 1

I start from the first element and go to the last. There are two loops here.
Outer loop is for the length of the subsequence we are currently calculating(from the 2nd element to the nth element)
In inner loop I start always from the leftmost element. Every time the inner loop iterates it compares the currect element of the inner loop to the element of the outer loop. If inner loop's element is smaller than the outer loop's than that means there is an ascending subset within. After that I also check for the length of the longest ascending subsequence this element is part of.
If length of the outer loop element's longest subsequence is 1 and length of the inner loop element's longest subsequence is also 1, or smaller than the length of the inner loop element's longest subsequence than I increase the length of the outer loop element's longest subsequence by adding 1 to the lenght of the inner loop element's.

For example:
arr: 4 -1 3 9 11
ls: 1 1 1 1 1

i = 0-¿ inner loop's pointer
j = 3-¿ outer loop's pointer

inner loop's element -¿ 4, its lenght -¿ 1
outer loop's element -¿ 9, its lenght -¿ 1
since 9 ¿ 4 and at the same time 1 ¿= 1
than I will increase the length by making ls[i] = ls[j] + 1 thus ls will become
ls: 1 1 1 2 1

I will do this for each element!
After that I will have some array like
ls: 1 1 2 3 4

Now after this I can see that the longest subsequence contains 4 elements
I can find this by going from right to left and follow the length of the longest subsequence 4(i:4), 3(i:3), 2(i:2), 1(i:1) and each would represent the position of our wanted array 11(i:4), 9(i:3), 3(i:2), -1(i:1) where i represents respective positions in the array.

The time complexity of this algorithm will be O(n^2) since we have 2 loops which traverse the original array. Regarding space complexity we need an additional space of array that would store the longest length of the subsequence at each position and space for the array to be returned which is at most also of n elements putting the memory at O(n) complexity.

2 Pseudocode

Logic of the program i previously explained can be represented by following C code I wrote:

```c

int* longestSubSeq( int* arr, int n, int* res_n ){
    int lis_i[n];
    for( int i = 0; i<n; ++i )
        lis_i[i] = 1;

    int max_length = 0;
    for( int i = 1; i<n; ++i ){

        int length = 1;
        for( int j = 0; j<i; ++j ){
            if( arr[i] > arr[j] && lis_i[i] <= lis_i[j] ){
                lis_i[i] = lis_i[j] + 1;
                length++;
            }
        }

        // used for the dynamic allocation later on
        if( length > max_length )
            max_length = length;

    }

    // UNTIL HERE, THIS IS LEFT TO BE WRITTEN ALSO
    int* return_arr = (int*)malloc(sizeof(max_length*sizeof(int)));

    int curr_el_index = max_length;
    for( int i = n-1; i>=0; --i){
        if( lis_i[i] == curr_el_index ){
            return_arr[curr_el_index-1] = arr[i];
            --curr_el_index;
        }
    }
    *res_n = max_length;
    return return_arr;
}
```

Figure 1: C code for longestSubSeq function.

The only difference in the MIPS MARS assembler is that longestSubSeq has an additional parameter and that the resulting sequence which is again returned alongside its length inside $v0 and $v1. When reading from I file I would read the whole file and store it in a string.
Then I would go character by character and as long as there is no space it means I would need to add that to my number

f = open(file_name)
num = 0
ch = ”
ch = read_char(f)
while(ch != 0) // until the end of string
flag_neg = 0
while(ch != ' ' and ch != 0)

if (ch == '-')

flag_neg = 1
else
num = num*10 + ch - '0'

ch = read_char(f)

if(flag_neg)
num = num * -1 (in mps, xor with -1 and than add 1 to get 2's complement or use neg pseudoinstruction)

add_to_array(num) //do this for every number in the file

After this I would execute the already mentioned C - written code for which I won't write pseudocode as I have already written clear working C code and full explaination.
After all of this is finished I need to write the found result inside a file.
To do this I have implemented an itoa function that converts the integers to string.
In order to do that I have made a space for this string of 32 spaces,meaning number can be up to 32 digits long.
Now I perform a division with 10 first, I keep the result for the next loop iteration and take the reminder and put the reminder inside the file. Of course when writing the reminder inside of the string I write it from right to left!
After this string starting position of the string is returned! For negative numebers I convert them to positive number and use a flag so that when I finish I can add the '-' sign to the string.

stringspace = 32bytes
ITOA(num)
curr_pointer = stringpasce + 30
curr_pointer[1] = ''
flag_neg = 0
if(num ¡ 0) flag_neg = 1

num = num/10
while (num != 0)
rem = take_from_register // in mips reminder of division is placed inside one of the registers
currpointer[0] = rem + '0'
–currpointer
num = num/10

if(flag_neg)
currpointer[0] = '-'

return curr_pointer
For writing to file I just make a loop and print the result of maxSubsequence using the length provided by return
arr_from_file
n
length = 0
arr = space(n)

maxSubsequent(arr_from_file, n, n, &length, arr)

fd = openFileToWrite(filename)
for i = 0 to length
string
string_n

itoa(arr_from_file[i], string, &string_n)
printToFile(fd, string, string_n)
endfor

3 Tests

1. Test 1

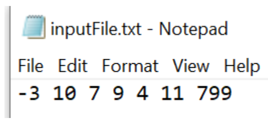

Figure 2: Test 1

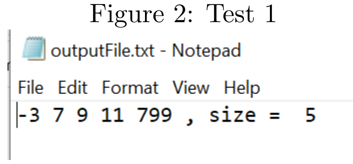

Figure 3: Test 1

2. Test 2

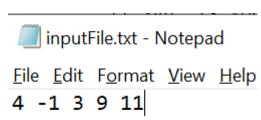

Figure 4: Test 2

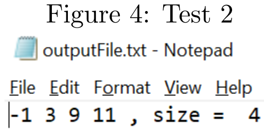

Figure 5: Test 2

3. Test 3

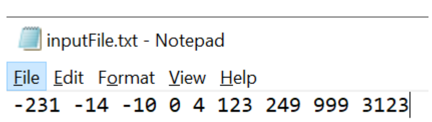

Figure 6: Test 3

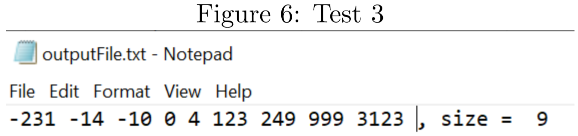

Figure 7: Test 3

4. Test 4

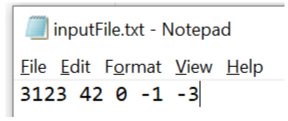

Figure 8: Test 4

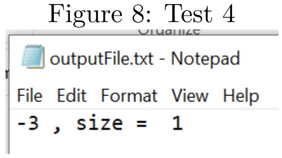

Figure 9: Test 4

5. Test 5

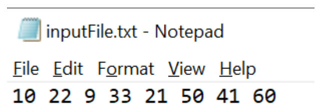

Figure 10: Test 5

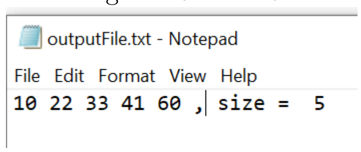

Figure 11: Test 5

6. Test 6

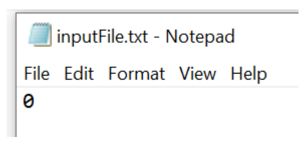

Figure 12: Test 6

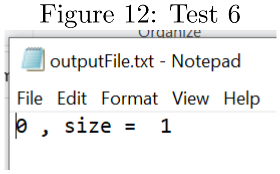

Figure 13: Test 6

References