CS1010E 16/17S1 Sum of Sub-sequences (Question)

Tags & Categories

Tags:

Categories:

Related Tutorials

Task Content

Sum of Sub-sequences

Topic Coverage

- Selection and repetition control statements
- Assignment and expressions

You may use other programming constructs, but **do not** use the math library functions.

Problem Description

Given a list of `n` non-zero integer values, a sub-sequence is defined as any non-empty sequence of at most `n` values that appears contiguously (i.e. next to each other) within the given list.

For example, the list `[ -3 4 -1 2 ]` has the following sub-sequences:

Sub-sequence	Sum of sub-sequence
`[ -3 4 -1 2 ]`	`2`
`[ -3 4 -1 ]`	`0`
`[ 4 -1 2 ]`	`5`
`[ -3 4 ]`	`1`
`[ 4 -1 ]`	`3`
`[ -1 2 ]`	`1`
`[ -3 ]`	`-3`
`[ 4 ]`	`4`
`[ -1 ]`	`-1`
`[ 2 ]`	`2`

Values within a sub-sequence can be added to give the sum as shown in the above table. Moreover, the **maximum sum of sub-sequences** is `5` (from the sub-sequence `[ 4 -1 2 ]`).

As another example, the list of integer values `[ -2 1 -3 4 -1 2 1 -5 4 ]` contains the sub-sequence `[ 4 -1 2 1 ]` with the sum `(4) + (-1) + (2) + (1) = 6` which is maximum. No other sub-sequence has a sum larger than `6`.

Task

Given a list of non-zero integer values, your task is to find the **maximum sum of sub-sequences** of the list.

This task is divided into several levels. Read through all the levels (from first to last, then from last to first) to see how the different levels are related. **You may start from any level.**

- Assume there is at least one value in the list;
- Each non-zero value in the list ranges between `-9` to `9` (excluding `0`, of course).

Within each level, sample runs are provided for you to test your program. These sample runs represent the corner test cases for small inputs. You are encouraged to expand on these and come up with your own test cases.

- **Deadline: Submit your minimal modifications to CodeCrunch by Friday, 30 September, 23:59:59.**

Level 1

Name your program `sum1.c`

Write a program that reads as input a list of non-zero integer values and outputs each value on a separate line. Input is terminated by the sentinel value `0`.

The following is a sample run of the program. User input is underlined. Ensure that the last line of output is followed by a newline character.

```
$ ./a.out
1 0
1
```

```
$ ./a.out
-2 1 -3 4 -1 2 1 -5 4 0
-2
1
-3
4
-1
2
1
-5
4
```

Click here to submit to CodeCrunch.

Check the correctness of the output by typing the following Unix command

```
./a.out < sum.in | diff - sum1.out
```

To proceed to the next level (say level 2), copy your program by typing the Unix command

```
cp sum1.c sum2.c
```

Level 2

Name your program `sum2.c`

Write a program that reads as input a list of non-zero integer values and outputs each value on a separate line. Input is terminated by the sentinel value `0`.

Alongside each i^{th} value, output the cumulative sum (i.e. the sum of all values from the first value up till and including the i^{th} one).

The following is a sample run of the program. User input is underlined. Ensure that the last line of output is followed by a newline character.

```
$ ./a.out
1 0
1,1
```

```
$ ./a.out
-1 0
-1,-1
```

```
$ ./a.out
1 2 0
1,1
2,3
```

```
$ ./a.out
-1 2 -3 0
-1,-1
2,1
-3,-2
```

```
$ ./a.out
-2 1 -3 4 -1 2 1 -5 4 0
-2,-2
1,-1
-3,-4
4,0
-1,-1
2,1
1,2
-5,-3
4,1
```

Click here to submit to CodeCrunch.

Check the correctness of the output by typing the following Unix command

```
./a.out < sum.in | diff - sum2.out
```

To proceed to the next level (say level 3), copy your program by typing the Unix command

```
cp sum2.c sum3.c
```

Level 3

Name your program `sum3.c`

Write a program that reads as input a list of non-zero integer values and outputs each value on a separate line. Input is terminated by the sentinel value `0`.

Alongside each i^{th} value, output the cumulative sum (i.e. the sum of all values from the first value until the i^{th} one). The last line of output is the maximum over all cumulative sums.

The following is a sample run of the program. User input is underlined. Ensure that the last line of output is followed by a newline character.

```
$ ./a.out
1 0
1,1
1
```

```
$ ./a.out
-1 0
-1,-1
-1
```

```
$ ./a.out
1 2 0
1,1
2,3
3
```

```
$ ./a.out
-1 2 -3 4 0
-1,-1
2,1
-3,-2
4,2
2
```

```
$ ./a.out
-2 1 -3 4 -1 2 1 -5 4 0
-2,-2
1,-1
-3,-4
4,0
-1,-1
2,1
1,2
-5,-3
4,1
2
```

Click here to submit to CodeCrunch.

Check the correctness of the output by typing the following Unix command

```
./a.out < sum.in | diff - sum3.out
```

To proceed to the next level (say level 4), copy your program by typing the Unix command

```
cp sum3.c sum4.c
```

Level 4

Name your program `sum4.c`

The intuition behind finding the **maximum sum of sub-sequences** is simple.

If the sum of the sub-sequence from [x_i ... x_{j-1}] is negative, then we will be better off checking a new sub-sequence starting with x_j as [x_j ... x_k] has a larger sum than [x_i ... x_{j-1} x_j ... x_k].

Write a program that reads as input a list of non-zero integer values and outputs each value on a separate line. Input is terminated by the sentinel value `0`.

Alongside each value, output the cumulative sum with restart (i.e. if the current cumulative sum is negative, then restart the cumulative sum for the next value that is read). The last line of output is the maximum over all cumulative sums with restart, which is also the **maximum sum of sub-sequences**.

The following is a sample run of the program. User input is underlined. Ensure that the last line of output is followed by a newline character.

```
$ ./a.out
1 0
1,1
1
```

```
$ ./a.out
-1 0
-1,-1
-1
```

```
$ ./a.out
1 1 0
1,1
1,2
2
```

```
$ ./a.out
1 -1 0
1,1
-1,0
1
```

```
$ ./a.out
-1 1 0
-1,-1
1,1
1
```

```
$ ./a.out
-1 -1 0
-1,-1
-1,-1
-1
```

```
$ ./a.out
1 1 -1 0
1,1
1,2
-1,1
2
```

```
$ ./a.out
1 -1 1 0
1,1
-1,0
1,1
1
```

```
$ ./a.out
-1 1 -1 0
-1,-1
1,1
-1,0
1
```

```
$ ./a.out
1 -1 2 0
1,1
-1,0
2,2
2
```

```
$ ./a.out
-1 2 -3 4 0
-1,-1
2,2
-3,-1
4,4
4
```

```
$ ./a.out
1 1 -2 2 0
1,1
1,2
-2,0
2,2
2
```

```
$ ./a.out
2 -1 1 0
2,2
-1,1
1,2
2
```

```
$ ./a.out
-2 1 -3 4 -1 2 1 -5 4 0
-2,-2
1,1
-3,-2
```

```
4,4
-1,3
2,5
1,6
-5,1
4,5
6
```

Click here to submit to CodeCrunch.

Check the correctness of the output by typing the following Unix command

```
./a.out < sum.in | diff - sum4.out
```

To proceed to the next level (say level 5), copy your program by typing the Unix command

```
cp sum4.c sum5.c
```

Level 5

Name your program `sum5.c`

Write a program that reads as input a list of non-zero integer values and outputs each value on a separate line. Input is terminated by the sentinel value `0`.

Alongside each value, output the cumulative sum with restart (i.e. if the current cumulative sum is negative, then restart the cumulative sum for the next value that is read).

The last line of output is the **maximum sub-sequence sum**, as well as the location within which the sub-sequence is found. For example, in the list of values `[ -3 4 -1 2 ]`, the maximum sum `5` is associated with the sub-sequence between the second and fourth values. Hence the corresponding output is `5[2,4]`.

In addition, if there are multiple sub-sequences with the same maximum sum, choose the one with the **minimum ending position**; if there are still multiple sub-sequences with the same minimum ending position, choose the **shortest sub-sequence**. As an example, the list `[ 1 -1 2 -2 2 0 ]` has a maximum sum of sub-sequences of `2` given by three sequences:

1. `[ 1 -1 2 ]` (within positions 1 to 3)
2. `[ 2 ]` (at position 3)
3. `[ 2 ]` (at position 5)

Choosing the minimum ending position would eliminate option (3). And between options (1) and (2) that ends at the same ending position, we choose the shorter one. Hence, option (2) is the final choice.

Hint:

- Take note of the starting positions of each sub-sequence;
- Among these starting positions, one of them will be the start of the sub-sequence having the maximum sum;
- Setting the end position is trivial.

The following is a sample run of the program. User input is underlined. Ensure that the last line of output is followed by a newline character.

```
$ ./a.out
1 0
1,1
1[1,1]
```

```
$ ./a.out
-1 0
-1,-1
-1[1,1]
```

```
$ ./a.out
1 1 0
1,1
1,2
2[1,2]
```

```
$ ./a.out
1 -1 0
1,1
-1,0
1[1,1]
```

```
$ ./a.out
-1 1 0
-1,-1
1,1
1[2,2]
```

```
$ ./a.out
-1 -1 0
-1,-1
-1,-1
-1[1,1]
```

```
$ ./a.out
1 1 -1 0
1,1
1,2
-1,1
2[1,2]
```

```
$ ./a.out
1 -1 1 0
1,1
-1,0
1,1
1[1,1]
```

```
$ ./a.out
-1 1 -1 0
-1,-1
1,1
-1,0
1[2,2]
```

```
$ ./a.out
1 -1 2 0
1,1
```

```
-1,0
2,2
2[3,3]
```

```
$ ./a.out
-1 2 -3 4 0
-1,-1
2,2
-3,-1
4,4
4[4,4]
```

```
$ ./a.out
1 1 -2 2 0
1,1
1,2
-2,0
2,2
2[1,2]
```

```
$ ./a.out
2 -1 1 0
2,2
-1,1
1,2
2[1,1]
```

```
$ ./a.out
1 -1 2 -2 2 0
1,1
-1,0
2,2
-2,0
2,2
2[3,3]
```

```
$ ./a.out
-2 1 -3 4 -1 2 1 -5 4 0
-2,-2
1,1
-3,-2
4,4
-1,3
2,5
1,6
-5,1
4,5
6[4,7]
```

Check the correctness of the output by typing the following Unix command

```
./a.out < sum.in | diff - sum5.out
```

Click here to submit to CodeCrunch.

Do not use the Submit button below until the deadline is over.

Submission (Course)

Select course: CS1010E (2016/2017 Sem 1) - Programming Methodology ▼

Your Files:

SUBMIT (only .java, .c, .cpp and .h extensions allowed)

To submit multiple files, click on the Browse button, then select one or more files. The selected file(s) will be added to the upload queue. You can repeat this step to add more files. Check that you have all the files needed for your submission. Then click on the Submit button to upload your submission.