CS1010E: Programming Methodology

Take Home Lab 4: Array & Matrix

22 Mar 2017

Preliminary: Summation [#1]

Problem Description

This is a simple exercise of finding the sum of an array.

Final Objective

Given a sequence of **`integer`** numbers, print the sum of all the continuous subset of the array that starts from index 0.

Assumptions

The following assumptions are considered to be true, they limit the inputs to the following restrictions:

- ▷ 3 ≤ `N` ≤ 1000 (*the number of* **`integer`** *in the list*)
- ▷ -1000 ≤ `n` ≤ 1000 (*the value of* **`integer`** *in the list*)

Tasks

The problem is split into 1 task(s). In the sample run, please note the following:

- ↩ is the *invisible* `[`**`newline`**`]` character.
- User input in blue and program output in purple color.
- Comments are in green color and are not part of the input and/or output.

Task 1/1

Write a program that reads a sequence of **`integer`** numbers and print the sum of all the continuous subset of the array that starts from index 0. The sequence is given as multiple lines of input with the first line is the number N denoting the number of **`integer`** in the list. The next N lines consists of a single **`integer`** of each value n in the list.

Sample Run:

Inputs:

```
5 | N = 5 -> 5 integer in the list
1 | first number
2 | second ...
3
4
5
```

Outputs:

```
1↩  | 1
3↩  | 1 + 2
6↩  | 1 + 2 + 3
10↩ | 1 + 2 + 3 + 4
15↩ | 1 + 2 + 3 + 4 + 5
```

Save your program in the file named `sum1.c`. Submit your program to CodeCrunch.

Easy: Running Median [#2]

Problem Description

“From a statistical point of view, the moving average, when used to estimate the underlying trend in a time series, is susceptible to rare events such as rapid shocks or other anomalies. A more robust estimate of the trend is the simple moving median over n time points:

$$SMM = \text{Median}(p_m, p_{m+1}, ..., p_{m-n+1}) \tag{1}$$

“where the median is found by, for example, sorting the values inside the brackets and finding the value in the middle. For larger values of n, the median can be efficiently computed by updating an indexable skiplist.

“Statistically, the moving average is optimal for recovering the underlying trend of the time series when the fluctuations about the trend are normally distributed. However, the normal distribution does not place high probability on very large deviations from the trend which explains why such deviations will have a disproportionately large effect on the trend estimate.” – Wikipedia

To put it simply, the *median* of a sequence is the middle value of the sorted sequence. It separates the higher half with the lower half of the data. When there are two numbers in the middle, we choose the larger value. In our current problem, the moving median is simply the median *up to* the current value. We will never forget older values.

Final Objective

Given a sequence of `integer` numbers, print the moving median of all the values including the intermediate subset sequence.

Example

Consider the array: `[1, 3, 6, 6, 6, 6, 7, 7, 12, 12, 17]`. When the array is being populated, the value of the median is changing. Consider the case when the array is populated from left-to-right as shown in the sequence below:

```
Array          | Median
[]             | None (ignored)
[1]            | 1
[1, 3]         | 3
[1, 3, 6]      | 3
[1, 3, 6, 6]   | 6
...            | Median does not change from here
```

Assumptions

The following assumptions are considered to be true, they limit the inputs to the following restrictions:

▷ $3 \leq$ `n` ≤ 100 (*the number of* `integer` *in the sequence*)
▷ $-2^{30} \leq$ `val` $\leq 2^{30}$ (*the value of* `integer` *in the sequence*)

Tasks

The problem is split into 3 task(s). In the sample run, please note the following:

- ↩ is the *invisible* `[newline]` character.
- User input in blue and program output in purple color.
- Comments are in green color and are not part of the input and/or output.

Task 1/3
Write a program that reads a sequence of `integer` numbers and prints the array back. The input is given as multiple lines. The first line is a value n which corresponds to the number of `integer` in the sequence. The next line consists of n `integer` numbers corresponding to the `integer` in the sequence. Note that there is no additional `[space]` at the end.
Sample Run:

Inputs:
```
5
1 6 3 6 6
```

Outputs:
```
1 6 3 6 6↩
```

Save your program in the file named `median1.c`. Submit your program to CodeCrunch. To proceed to the next task (*e.g., task 2*), copy your program using the following command:
```
cp median1.c median2.c
```

Task 2/3
Write a program that reads a sequence of `integer` numbers and prints every subarray starting from index 0 in a sorted order. The first line is a value n which corresponds to the number of `integer` in the sequence. The next line consists of n `integer` numbers corresponding to the `integer` in the sequence. Note that there is no additional `[space]` at the end.
Sample Run:

Inputs:
```
5
1 6 3 6 6
```

Outputs:
```
1↩
1 6↩
1 3 6↩
1 3 6 6↩
1 3 6 6 6↩
```

Save your program in the file named `median2.c`. Submit your program to CodeCrunch. To proceed to the next task (*e.g., task 3*), copy your program using the following command:
```
cp median2.c median3.c
```

Task 3/3
Write a program that reads a sequence of `integer` numbers and prints the moving median. The first line is a value n which corresponds to the number of `integer` in the sequence. The next line consists of n `integer` numbers corresponding to the `integer` in the sequence. Note that there is no additional `[space]` at the end.
Sample Run:

Inputs:
```
5
1 6 3 6 6
```

Outputs:
```
1↩
6↩
3↩
6↩
6↩
```

Save your program in the file named `median3.c`. Submit your program to CodeCrunch.

Easy: Spiral [#3]

Problem Description

"In mathematics, a spiral is a curve which emanates from a point, moving farther away as it revolves around the point." – Wikipedia

We will print 2D array in a spiral pattern. Consider the following matrix:

```
1 2 3
4 5 6
7 8 9
```

The spiral pattern starting from the top-left inwards is: `1 2 3 6 9 8 7 4 5`.

Final Objective

Given a sequence of **`integer`** numbers as a matrix, print the matrix in spiral pattern.

Assumptions

The following assumptions are considered to be true, they limit the inputs to the following restrictions:

▷ $2 \leq$ `size` ≤ 30 (*the size of the matrix is size* $\times$ *size*)
▷ $1 \leq$ `num` ≤ 1000 (*the number in the matrix*)

Tasks

The problem is split into 2 task(s). In the sample run, please note the following:

- ↩ is the *invisible* [**`newline`**] character.
- User input in blue and program output in purple color.
- Comments are in green color and are not part of the input and/or output.

Task 1/2

Write a program to read a matrix of **`integer`** and print the matrix in a single line. The matrix is given as multiple lines of input. The first line is an **`integer`** *size* where it denotes the size of the matrix. The next *size* lines consists of *size* number of **`integer`** which represents the element in the matrix at the given row. Note there **IS** an additional [**`space`**] at the end.

Sample Run:

Inputs:

```
3 | size is 3 x 3
1 2 3
8 9 4
7 6 5
```

Outputs:

```
1 2 3 8 9 4 7 6 5 ↩
```

Save your program in the file named `spiral1.c`. Submit your program to CodeCrunch. To proceed to the next task (*e.g., task 2*), copy your program using the following command:

```
cp spiral1.c spiral2.c
```

Task 2/2
Write a program to read a matrix of `integer` and print the matrix in a spiral pattern. The matrix is given as multiple lines of input. The first line is an `integer` *size* where it denotes the size of the matrix. The next *size* lines consists of *size* number of `integer` which represents the element in the matrix at the given row. Note there **IS** an additional `[space]` at the end. *Sorry, I can't think of an intermediate problem to this.*
Sample Run:

Inputs:

```
3 | size is 3 x 3
1 2 3
8 9 4
7 6 5
```

Outputs:

```
1 2 3 4 5 6 7 8 9 ↩
```

Save your program in the file named `spiral2.c`. Submit your program to CodeCrunch.

Medium: k-ibonacci [#4]

Problem Description

"The tribonacci numbers are like the Fibonacci numbers, but instead of starting with two predetermined terms, the sequence starts with three predetermined terms and each term afterwards is the sum of the preceding three terms. The tetranacci numbers start with four predetermined terms, each term afterwards being the sum of the preceding four terms. Pentanacci, hexanacci, and heptanacci numbers have been computed." – Wikipedia

Let's call the generalization of Fibonacci as the k-ibonacci number where k denotes the number of preceeding terms. We can then rewrite the formula as:

$$kib(n,k) = \begin{cases} 0 & \text{if } 0 \leq n < k \\ 1 & \text{if } n == k \\ \Sigma_{i=1}^{k} kib(n-i,k) & \text{otherwise} \end{cases} \tag{2}$$

And one more time, we are dealing with possibly large number. And one more time, we will be using *modular arithmetic* to limit the result to the range of **integer**. And one more time, we will add a value m as the modulo value. And one last time, the task is $kib(n,k,m) = kib(n,k) \bmod m$.

The Formula 2 can be written recursively by rewriting it to a recursive function. However, due to *efficiency* reason, you should use an array to solve the problem. On the other hand, feel free to check if your answer is correct using the following function:

```
int kibonacci(int n, int k, int m) {
  int sum = 0, i;
  if(n >= 0 && n < k) return 0;
  if(n == k)          return 1;
  for(i=1; i<=k; i++)
    sum = ((sum%m) + (kibonacci(n-i, k)%m))%m;
  return sum;
}
```

Note, *however*, to prevent you from submitting that code as the answer to CodeCrunch, we will be limiting the amount of time to 15s.

Final Objective

Given n, k, and m, compute $kib(n,k,m)$.

Assumptions

The following assumptions are considered to be true, they limit the inputs to the following restrictions:

- ▷ $0 \leq$ n $\leq 2^{15}$ (*the value of* n)
- ▷ $1 \leq$ k $\leq 2^{15}$ (*the value of* k)
- ▷ $2 \leq$ m $\leq 2^{15}$ (*the value of* m)

Tasks

The problem is split into 3 task(s). In the sample run, please note the following:

- ↩ is the *invisible* `[newline]` character.
- User input in blue and program output in purple color.
- Comments are in green color and are not part of the input and/or output.

Task 1/3

Write a program that reads n, k, and m and prints $(n \times k)$ mod m.

Sample Run:

Inputs:

```
10 3 7
```

Outputs:

```
9↩ | (10 * 3) = 30 mod 7 = 9
```

Save your program in the file named `kibonacci1.c`. Submit your program to CodeCrunch. To proceed to the next task (*e.g., task 2*), copy your program using the following command:

```
cp kibonacci1.c kibonacci2.c
```

Task 2/3

Write a program that reads n, k, and m and prints $fib(n)$ mod m.

Hint:

- Use array to compute the value of $fib(n)$ mod m

Sample Run:

Inputs:

```
10 3 7
```

Outputs:

```
6↩ | fib(10) = 55 mod 7 = 6
```

Save your program in the file named `kibonacci2.c`. Submit your program to CodeCrunch. To proceed to the next task (*e.g., task 3*), copy your program using the following command:

```
cp kibonacci2.c kibonacci3.c
```

Task 3/3

Write a program that reads n, k, and m and prints $kib(n,k)$ mod m.

Hint:

- Use array to compute the value of $kib(n,k)$ mod m

Sample Run:

Inputs:

```
10 3 7
```

Outputs:

```
2↩ | kib(10,3) = 44 mod 7 = 2
```

Save your program in the file named `kibonacci3.c`. Submit your program to CodeCrunch.

Medium: Josephus [#5]

Problem Description

"In computer science and mathematics, the Josephus problem (or Josephus permutation) is a theoretical problem related to a certain counting-out game.

"People are standing in a circle waiting to be executed. Counting begins at a specified point in the circle and proceeds around the circle in a specified direction. After a specified number of people are skipped, the next person is executed. The procedure is repeated with the remaining people, starting with the next person, going in the same direction and skipping the same number of people, until only one person remains, and is freed.

"The problem given the number of people, starting point, direction, and number to be skipped is to choose the position in the initial circle to avoid execution." – Wikipedia

In the Josephus problem, we are given a *circular* array and a number k. Then, starting from index 0, we find the number at the kth index after it and remove the number from the array. Afterwards, we continue removing the number at the kth index after the removed number and remove it as well until there are only a single number in the array. This last number is to be printed.

Note that the array in our Josephus sequence starts at index 1 instead of the usual 0. As such, there is no element at index 0.

Final Objective

Given an array and a number k, find the last number in the array based on Josephus problem.

Example

The sequence below shows the progression to find the last number in Josephus problem when $k = 2$. Note that `X` marks the removed number

```
1 2 3 4 5 6 7
1 X 3 4 5 6 7
1 X 3 X 5 6 7
1 X 3 X 5 X 7
X X 3 X 5 X 7
X X 3 X X X 7
X X X X X X 7
```

The sequence below shows the progression to find the last number in Josephus problem when $k = 3$.

```
1 2 X 4 5 6 7
1 2 X 4 5 X 7
1 X X 4 5 X 7
1 X X 4 5 X X
1 X X 4 X X X
X X X 4 X X X
```

Assumptions

The following assumptions are considered to be true, they limit the inputs to the following restrictions:

- ▷ $3 \leq$ N ≤ 1000 (*the size of the array*)
- ▷ $2 \leq$ k ≤ 21 (*the value of k*)
- ▷ $1 \leq$ num $\leq 2^{30}$ (*the element in the array*)
- ▷ k $<$ N

Tasks

The problem is split into 4 task(s). In the sample run, please note the following:

- ↩ is the *invisible* `[newline]` character.
- User input in blue and program output in purple color.
- Comments are in green color and are not part of the input and/or output.

Task 1/4

Write a program that reads a sequence of `integer` numbers and prints the array back. The input is given as multiple lines. The first line is **two (2)** `integer` value N and k which corresponds to the number of `integer` in the sequence and the Josephus skip value. The next line consists of N number of `integer` numbers corresponding to the `integer` in the sequence. Note that there is no additional `[space]` at the end.

Sample Run:

Inputs:

```
7 2 | N = 7, k = 2
1 2 3 4 5 6 7
```

Outputs:

```
1 2 3 4 5 6 7↩
```

Save your program in the file named `josephus1.c`. Submit your program to CodeCrunch. To proceed to the next task (*e.g., task 2*), copy your program using the following command:

```
cp josephus1.c josephus2.c
```

Task 2/4

Write a program that reads a sequence of `integer` numbers and prints the array starting from index $k + 1$. The input is given as multiple lines. The first line is **two (2)** `integer` value N and k which corresponds to the number of `integer` in the sequence and the Josephus skip value. The next line consists of N number of `integer` numbers corresponding to the `integer` in the sequence. Note that there is no additional `[space]` at the end.

Sample Run:

Inputs:

```
7 2 | N = 7, k = 2
1 2 3 4 5 6 7
```

Outputs:

```
2 3 4 5 6 7↩
```

Save your program in the file named `josephus2.c`. Submit your program to CodeCrunch. To proceed to the next task (*e.g., task 3*), copy your program using the following command:

```
cp josephus2.c josephus3.c
```

Task 3/4
Write a program that reads a sequence of `integer` numbers and prints the array starting from index $k+1$ and skips all the *consecutive* number equal to the value at index $k+1$. The input is given as multiple lines. The first line is **two (2)** `integer` value N and k which corresponds to the number of `integer` in the sequence and the Josephus skip value. The next line consists of N number of `integer` numbers corresponding to the `integer` in the sequence. Note that there is no additional `[space]` at the end.

Sample Run:

Inputs:
```
7 2 | N = 7, k = 2
1 2 3 4 5 6 7
```
Outputs:
```
4 5 6 7↩ | the first 2 is skipped, 3 is skipped
```

Sample Run:

Inputs:
```
7 2 | N = 7, k = 2
1 2 2 2 5 6 2
```
Outputs:
```
5 6 2↩ | consecutive 2 is skipped, but not last 2
```

Save your program in the file named `josephus3.c`. Submit your program to CodeCrunch. To proceed to the next task (*e.g., task 4*), copy your program using the following command:
```
cp josephus3.c josephus4.c
```

Task 4/4
Write a program that reads a sequence of `integer` numbers and prints the last number in the Josephus problem. The input is given as multiple lines. The first line is **two (2)** `integer` value N and k which corresponds to the number of `integer` in the sequence and the Josephus skip value. The next line consists of N number of `integer` numbers corresponding to the `integer` in the sequence. Note that there is no additional `[space]` at the end. *Hint:*

- Consider Task 2: change the value at index k to some other value, this is the visited number
- Consider Task 3: consecutive *visited number* can be skipped in this way so that it is not revisited
- Remember that in Josephus, index starts from 1 but array index starts from 0

Sample Run:

Inputs:
```
7 2 | N = 7, k = 2
1 2 3 4 5 6 7
```
Outputs:
```
7↩ | see example
```

Sample Run:

Inputs:
```
7 2 | N = 7, k = 2
1 2 2 2 5 6 2
```
Outputs:
```
2↩ | modified from example
```

Sample Run:

Inputs:
```
7 3 | N = 7, k = 3
1 2 3 4 5 6 7
```
Outputs:
```
4↩ | see example
```

Save your program in the file named `josephus4.c`. Submit your program to CodeCrunch.

Hard: Cyclic [#6]

Problem Description

"In graph theory, a cycle is a path of edges and vertices wherein a vertex is reachable from itself. There are several different types of cycles, principally a closed walk and a simple cycle; also, e.g., an element of the cycle space of the graph." – Wikipedia

Let's transform this concept into an array problem. First, we define a skip array as an `integer` array. The element of the skip array denotes the number of element to the *next element* in the array. In other words, given an array `int arr[SIZE]` and the index `idx`, the next element is located at `idx + arr[idx]`. If the value `idx` is greater than or equal to the size of the array then, then we do not consider this value at all. Similarly, if the value of `idx` is negative, we ignore this value too.

In a skip array, we can form a *chain* of elements starting from any index `idx` as:

```
+----+---------------------------------+-----------------------------------+
|Seq | index                           | element                           |
+----+---------------------------------+-----------------------------------+
|1   | idx                             | arr[idx]                          |
|2   | idx+arr[idx]                    | arr[idx+arr[idx]]                 |
|3   | idx+arr[idx]+arr[idx+arr[idx]]  | arr[idx+arr[idx]+arr[idx+arr[idx]]]|
|... | ...                             | ...                               |
+----+---------------------------------+-----------------------------------+
```

If the value `idx` points to an element outside the array, then this chain *ends*. We say that this skip array contains a *cycle* if there is an index `idx` such that the *chain* starting from `idx` contains multiple value of `idx`. In fact, since it is a *cycle*, continuing the cycle gives an infinite number of `idx`.

For instance, the array `[1, 1, -2]` starting from `idx = 0` is cyclic. We can easily trace that it is cyclic:

```
+----+---------------------------------+-----------------------------------+
|Seq | index                           | element                           |
+----+---------------------------------+-----------------------------------+
|1   | idx =0                          | arr[0] = 1                        |
|2   | 0 + 1 = 1                       | arr[1] = 1                        |
|3   | 1 + 1 = 2                       | arr[2] = -2                       |
|4   | 2 + -2 = 0                      | arr[0] = 1                        |
|... | ...                             | ...                               |
+----+---------------------------------+-----------------------------------+
```

Sometime the cycle is not so trivial as in the array `[1, 1, 1, -2]`. Starting at `idx = 0`, the cycle actually returns to `idx = 1` instead. In other cases, the cycle is *degenerate* as seen in `[0, 0, 0]` which forms **three (3)** cycles each consisting of a single element. Most of the time, however, there are no cycle in the skip array as seen in `[1, 2, 3]`. The last **two (2)** elements skip to *outside* of the array and ends the chain.

The problem we are trying to solve is to find the number of *unique* cycle in a skip array. We say that two cycles are *unique* if starting from any point in one of the cycle, we can never reach the other cycle. Thus, it is necessary for these two cycles not to share any element. We can classify the following array `[2, 1, 1, -1]` as consisting of **only one (1)** cycle.

Final Objective

Given skip array, find the number of *unique* cycle in the skip array.

Assumptions

The following assumptions are considered to be true, they limit the inputs to the following restrictions:

▷ 3 ≤ N ≤ 1000 (*the size of the array*)
▷ -1000 ≤ S ≤ 1000 (*the element of the array*)

Tasks

The problem is split into 4 task(s). In the sample run, please note the following:

- ↩ is the *invisible* `[newline]` character.
- User input in blue and program output in purple color.
- Comments are in green color and are not part of the input and/or output.

Task 1/4

Write a program that reads the skip array and prints the array back. The input is given as multiple lines. The first line is **one (1)** `integer` value N corresponding to the size of the skip array. The next line consists of N number of `integer` numbers corresponding to the `integer` in the skip array. Note that there is no additional `[space]` at the end.

Sample Run:

Inputs:

```
4
1 1 1 -2
```

Outputs:

```
1 1 1 -2↩
```

Save your program in the file named `skip1.c`. Submit your program to CodeCrunch. To proceed to the next task (*e.g., task 2*), copy your program using the following command:

```
cp skip1.c skip2.c
```

Task 2/4

Write a program that reads the skip array and check if there is a cycle starting from index 0. The input is given as multiple lines. The first line is **one (1)** `integer` value N corresponding to the size of the skip array. The next line consists of N number of `integer` numbers corresponding to the `integer` in the skip array. Note that there is no additional `[space]` at the end.

Sample Run:

Inputs:

```
4
1 1 1 -2
```

Outputs:

```
1↩ | 1 for cyclic
```

Sample Run:

Inputs:

```
3
1 2 3
```

Outputs:

```
0↩ | 0 for non-cyclic
```

Save your program in the file named `skip2.c`. Submit your program to CodeCrunch. To proceed to the next task (*e.g., task 3*), copy your program using the following command:

```
cp skip2.c skip3.c
```

Task 3/4
Write a program that reads the skip array and print the array but replace all the numbers in the chain starting from index 0 with `'X'`. The input is given as multiple lines. The first line is **one (1)** **`integer`** value N corresponding to the size of the skip array. The next line consists of N number of **`integer`** numbers corresponding to the **`integer`** in the skip array. Note that there is no additional `[space]` at the end.
Sample Run:

Inputs:
```
4
1 1 1 -2
```
Outputs:
```
X X X X↩ | all values are visited and replaced
```

Sample Run:

Inputs:
```
3
1 2 3
```
Outputs:
```
X X 3↩ | the last element is not visited
```

Save your program in the file named `skip3.c`. Submit your program to CodeCrunch. To proceed to the next task (*e.g., task 4*), copy your program using the following command:
```
cp skip3.c skip4.c
```

Task 4/4
Write a program that reads the skip array and print the number of *unique* cycle. The input is given as multiple lines. The first line is **one (1)** **`integer`** value N corresponding to the size of the skip array. The next line consists of N number of **`integer`** numbers corresponding to the **`integer`** in the skip array. Note that there is no additional `[space]` at the end.
Sample Run:

Inputs:
```
4
1 1 1 -2
```
Outputs:
```
1↩
```

Sample Run:

Inputs:
```
3
1 2 3
```
Outputs:
```
0↩
```

Save your program in the file named `skip4.c`. Submit your program to CodeCrunch.