CS29003 Algorithms Laboratory

Assignment No: 4

Last date of submission: 05–February–2019

Dynamic-Programming Algorithms

You are given an array $A = (a_1, a_2, a_3, \ldots, a_n)$ of n positive integers. If you write the integers in the order they appear in A, and insert symbols $+$ or $-$ before each integer, you get an arithmetic expression. If the evaluation of the expression gives the value v, we say that v is realized (or realizable) by the array A. For example, consider the array $(7, 12, 1, 9, 5)$ of five positive integers. We have

$$\begin{aligned} +7-12-1+9+5 &= 8\,, \\ -7-12+1-9+5 &= -22\,, \\ -7+12-1-9+5 &= 0\,, \end{aligned}$$

that is, the integers $8, -22, 0$ are realizable by this array. You are given a target value T. Your task is to find out whether T is realizable by the given array A, and if so, how. Let $S = a_1 + a_2 + a_3 + \cdots + a_n$. Then, for T to be realizable by A, we must have $-S \leqslant T \leqslant S$. This condition is however only necessary (but not sufficient). In the five-element example above, $7 + 12 + 1 + 9 + 5 = 34$, but an odd integer like 23, despite being in the range $[-34, 34]$, cannot be realized by this array. It is easy to check that the integer 30, although even, is also unrealizable by the array. In the rest of this assignment, we will assume that $-S \leqslant T \leqslant S$, and that S and T have the same parity.

Part 1

Write a function **`realizable(A,n,T)`** to decide whether T is realizable by the array A of size n. The function should implement a dynamic-programming approach. Build a two-dimensional table $P[0 \ldots n][-S \ldots S]$ such that $P[i][j]$ would store the decision whether the value j can be realized by the prefix $(a_1, a_2, a_3, \ldots, a_i)$ of A. For $i = 0$, we consider no elements from A, so the only realizable value is 0. For $i \geqslant 1$, the value j is realizable by $(a_1, a_2, a_3, \ldots, a_i)$ if and only if either $j - a_i$ or $j + a_i$ is (or both are) realizable by $(a_1, a_2, a_3, \ldots, a_{i-1})$. Use this recursive formulation to build the table P in $\Theta(nS)$ time. The final decision is available as $P[n][T]$.

A couple of implementation issues are in order now. Each row of P should store $2S + 1$ decisions, and is indexed in the range $[-S, S]$. In C/C++, negative indexing may lead to devastating consequences, and must be avoided. Elements of a row **`P[i]`** of size $2S + 1$ in a two-dimensional C/C++ array **`P[][]`** are indexed in the range $[0, 2S]$. This means that the *logical* quantity $P[i][j]$ with $j \in [-S, S]$ is to be found in the *physical* location **`P[i][j+S]`**. The second issue pertains to the table lookup at column indices $j \pm a_i$. If any of these indices is not in the range $[-S, S]$, the corresponding lookup should not be made.

In this and the next two parts, do *not* use memoization.

Part 2

Copy the function of Part 1 to a function **`showone(A,n,T)`** that follows the same algorithm as Part 1 but additionally prints one way of realizing T (in case T is realizable). There may be multiple ways of realizing the same target value T. It suffices that this function reports any one of these realizations of T. The running time of this function should continue to remain $\Theta(nS)$.

Part 3

Write a function **`showall(A,n,T)`** to print *all* the realizations of T by A. This function may be developed from a copy of the function of Part 1 or 2. The input array A may contain duplicate elements. Rearranging the signs of the same elements is considered to lead to different realizations. For example, $-1+1+1+1-1$, $-1-1+1+1+1$, and $+1+1-1-1+1$ are considered different realizations of 1. The running time of this function should be $\Theta(n(S + r))$, where r is the number of realizations of T.

The *main*() function

- Read $n, a_1, a_2, a_3, \ldots, a_n, T$ from the user.
- Call **`realizable`** to decide whether T is realizable by A.
- Call **`showone`** to print some realization of T (provided that T is realizable).
- Call **`showall`** to print the number $r \geqslant 0$ of realizations of T, followed by these r realizations.

Sample output

```
------------------------------------------------------------------------------------------

n = 20
The input array:
81  78  38  32  78  68  30  53  54  81  76  30  97  3   70  94  64  49  66  71
T = -1059

+++ Part 1: Realizability check
   The value -1059 is not realizable

+++ Part 2: One solution
   The value -1059 is not realizable

+++ Part 3: All solutions
    Number of solutions = 0

------------------------------------------------------------------------------------------

n = 20
The input array:
26  87  97  87  53  6   29  95  45  71  28  93  52  71  75  49  82  18  26  95
T = 873

+++ Part 1: Realizability check
   The value 873 is realizable

+++ Part 2: One solution
    Solution: +26+87-97+87-53-6+29+95+45+71+28+93+52+71+75+49+82+18+26+95 =  873

+++ Part 3: All solutions
    Number of solutions = 29
    Sol   1 : +26+87-97+87-53-6+29+95+45+71+28+93+52+71+75+49+82+18+26+95 =  873
    Sol   2 : -26+87+97+87+53-6-29-95+45+71+28+93+52+71+75+49+82+18+26+95 =  873
    Sol   3 : -26+87+97+87-53-6+29+95+45-71+28+93+52+71+75+49+82+18+26+95 =  873
    Sol   4 : +26+87+97+87+53-6-29+95+45+71-28-93+52+71+75+49+82+18+26+95 =  873
    Sol   5 : +26+87+97+87-53-6+29+95-45+71+28+93-52+71+75+49+82+18+26+95 =  873
    Sol   6 : -26+87+97+87-53-6+29+95+45+71+28+93+52-71+75+49+82+18+26+95 =  873
    Sol   7 : +26+87+97+87-53+6+29+95+45+71-28+93+52+71-75+49+82+18+26+95 =  873
    Sol   8 : +26+87+97+87+53+6-29+95+45+71+28+93-52+71-75+49+82+18+26+95 =  873
    Sol   9 : -26+87+97+87-53+6+29+95+45+71-28+93+52+71+75-49+82+18+26+95 =  873
    Sol  10 : -26+87+97+87+53+6-29+95+45+71+28+93-52+71+75-49+82+18+26+95 =  873
    Sol  11 : -26+87+97+87+53-6+29+95+45+71+28+93+52+71-75-49+82+18+26+95 =  873
    Sol  12 : +26+87+97+87+53+6-29+95-45+71+28+93+52+71+75+49-82+18+26+95 =  873
    Sol  13 : +26-87+97+87+53-6+29+95-45+71+28+93+52+71+75+49+82-18+26+95 =  873
    Sol  14 : +26+87+97-87+53-6+29+95-45+71+28+93+52+71+75+49+82-18+26+95 =  873
    Sol  15 : +26+87+97+87+53+6+29+95-45+71+28-93+52+71+75+49+82-18+26+95 =  873
    Sol  16 : +26+87+97+87+53-6-29+95+45+71-28+93+52+71-75+49+82-18+26+95 =  873
    Sol  17 : -26+87+97+87+53-6-29+95+45+71-28+93+52+71+75-49+82-18+26+95 =  873
    Sol  18 : +26+87+97+87+53-6-29-95+45+71+28+93+52+71+75+49+82+18-26+95 =  873
    Sol  19 : -26+87+97+87-53-6+29+95-45+71+28+93+52+71+75+49+82+18-26+95 =  873
    Sol  20 : +26+87+97+87-53-6+29+95+45-71+28+93+52+71+75+49+82+18-26+95 =  873
    Sol  21 : +26+87+97+87-53-6+29+95+45+71+28+93+52-71+75+49+82+18-26+95 =  873
    Sol  22 : -26+87+97+87+53+6-29+95+45+71+28+93+52+71-75+49+82+18-26+95 =  873
    Sol  23 : +26+87+97+87-53+6+29+95+45+71-28+93+52+71+75-49+82+18-26+95 =  873
    Sol  24 : +26+87+97+87+53+6-29+95+45+71+28+93-52+71+75-49+82+18-26+95 =  873
    Sol  25 : +26+87+97+87+53-6+29+95+45+71+28+93+52+71-75-49+82+18-26+95 =  873
    Sol  26 : -26+87+97+87+53-6+29+95+45+71-28+93-52+71+75+49+82-18-26+95 =  873
    Sol  27 : +26+87+97+87+53-6-29+95+45+71-28+93+52+71+75-49+82-18-26+95 =  873
    Sol  28 : -26+87+97+87+53-6-29+95+45+71+28+93+52+71+75+49+82+18+26-95 =  873
    Sol  29 : +26+87+97+87+53-6-29+95+45+71+28+93+52+71+75+49+82+18-26-95 =  873

------------------------------------------------------------------------------------------
```

Submit a single C/C++ source file. Do not use global/static variables.