Project 2: Two-Sum Problem

CS 3343: Analysis of Algorithms Summer 2024

Instructor: Dr. Mohammad Imran Chowdhury

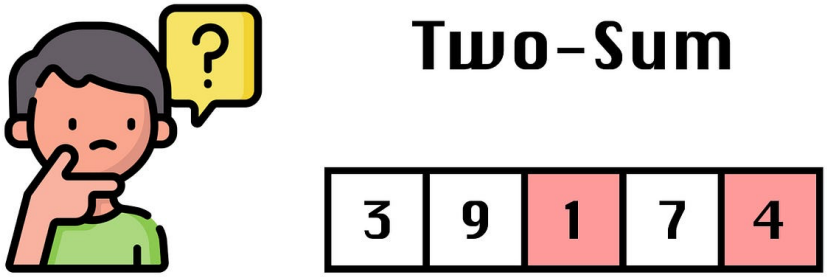
Total: 100 Points

Due: 06/25/2024 11:59 PM

In this project, I invite you to solve the ``Two-Sum Problem" using the various aspects of algorithmic thinking such as the Brute Force Approach, the Use of the HashMap Approach, and Python programming discussed in class.

Problem Description:

Consider you are given an array of integers and a target sum, return indices of two numbers in the array such that they add up to the given target. You may assume that each input would have exactly one solution. Also, you cannot use the same element twice. You are allowed to return the answer in any order.



For instance, the given array is [3,9,1,7, 4], and the given target sum = 5. If we take a look at the given array, the pair that adds to the target sum 5 is (1,4) i.e., 1+4 = 5. So, our algorithm should return (2,4) as the result because these are the indexes of elements 1 and 4 respectively in the given array. It is also stated in the problem statement that we can return the indices in any order, what does this mean? Let us understand this with the above example. The output for this example is [2,4], so when the problem statement says we can return the indices in any order what it means is that we can return either [2,4] or [4,2] as our output, both will be considered correct.

Note: If there are multiple solution pairs that add up to the target sum, we need to return the indexes of the first pair we find from the left [Brute Force Approach Only].

Brute Force Approach

A straightforward solution to this problem is to check for every possible pair present in the given array.

For a given input array **nums** we need to do the following steps:

- Run two loops and check for every combination in the given array.
- Fix the outer loop at a specific index and move the inner loop to get all the possible pairs. The outer loop runs from i=0 to i=n-2 and the inner loop runs from j=i+1 to j=n-1. **Note**: n here is the size of the array.
- In each iteration of the inner loop check if the numbers represented by the outer and inner loop indexes add up to the target sum.
- If **nums**[*outerLoopIndex*] + **nums**[*innerLoopIndex*] is equal to the **target**, return *{outerLoopIndex, innerLoopIndex}* as a result. Else continue iteration to check for the next pair.
- Repeat the above steps until you find a combination that adds up to the given target.

For example, for array [7,2,13,11] and target sum 24, we fix the outer loop at index i=0 i.e., element 7, and check it with all possible values of the inner loop from j=i+1 to j=n-1, i.e., from index 1 to 3. So, we will be checking the following pair of elements in the first iteration of the outer loop: (7,2) (7,13), and (7,11). Now we increment the outer loop index i by 1 and check it with indices 2 to 3 (i+1 to n-1) of the inner loop. We repeat this until we find the required answer.

In the worst case, this algorithm has a running time complexity of O(n^2). The worst case would occur when the required combination is the last combination to be checked by our loops.

Sample I/O:

Run 1:
Enter the input data array items : 1 2 3
Enter the target value : 4
Output: 0 2

Run 2:
Enter the input data array items : 7 2 13 11
Enter the target value : 24
Output: 2 3

Run 3:
Enter the input data array items : 2 1 7 11 15 17
Enter the target value : 18
Output: 1 5

Note: If there are multiple solution pairs that add up to the target sum, we need to return the indexes of the first pair we find from the left [Brute Force Approach Only].

HashMap Approach:

It is possible to solve this problem in linear time. The idea is to make use of a HashMap to store the indices of the elements that are already visited.

HashMap "key" is the number in the given input array (You add this to the HashMap as you visit each element). HashMap "value" is the index of the number in the array represented by the HashMap key.
For a given input array this algorithm does the following steps:

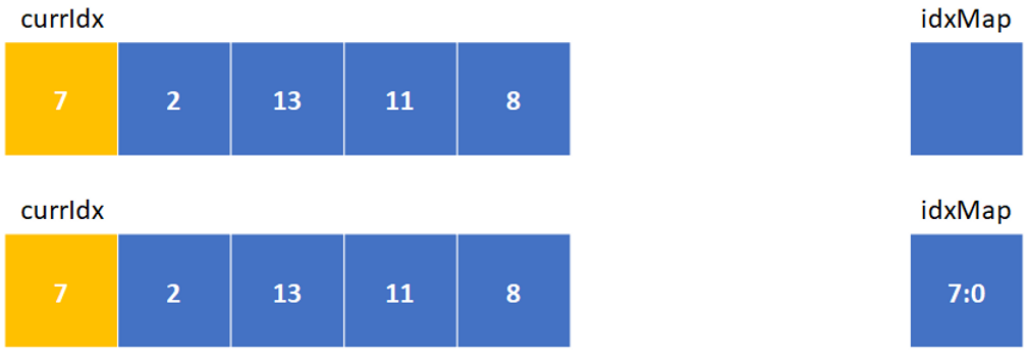
- Create a HashMap which accepts integer datatype as key and value.
- Iterate through each element in the given array starting from the first element.
- In each iteration check if the required number (*required number = target sum - current number*) is present in the hashmap.
- If present, return *{required number index, current number index}* as a result.
- Otherwise, add the current iteration number as the *key* and its index as *value* to the hashmap. Repeat this until you find the result.

Consider you are given the below input array and target = 24.

Let **currIdx** be the variable representing the current element under process and let **idxMap** be our index map. Cells marked in orange indicate the currently processed array element.

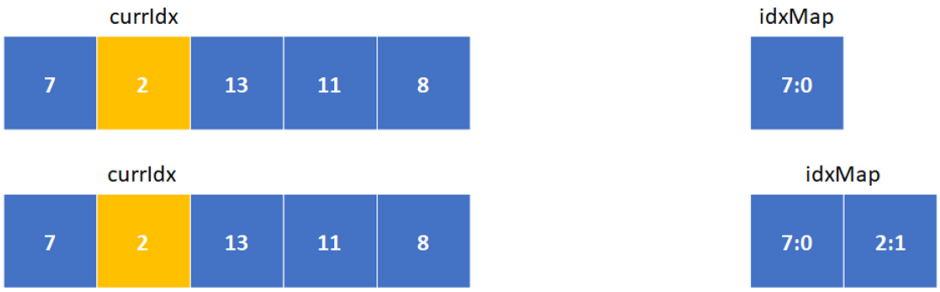
In the beginning, **currIdx** = 0, and **idxMap** is empty as shown in the first figure below. Next, we check if the *required number = target - current number* is present in the **idxMap**.

Required number = 24 - 7 = 17 is not present in our hashmap, so we add 7 as **idxMap** key and 0 as **idxMap** value (0 is the index of 7 in the input array) as shown in the figure below.

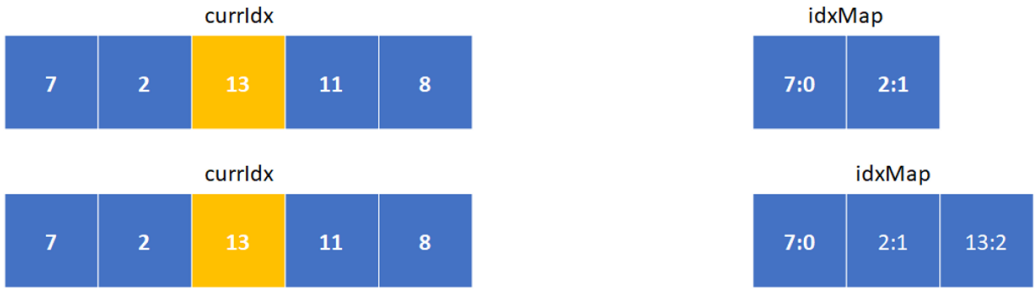


Next, we move on to the second element in the array by incrementing the current index. So **currIdx** = 1 which points to element 2 in the array. Again, we check if the required number is present in

idxMap, required number = 24 - 2 = 22 is not in our hashmap so we add to 2 to the hashmap along with its index 1.



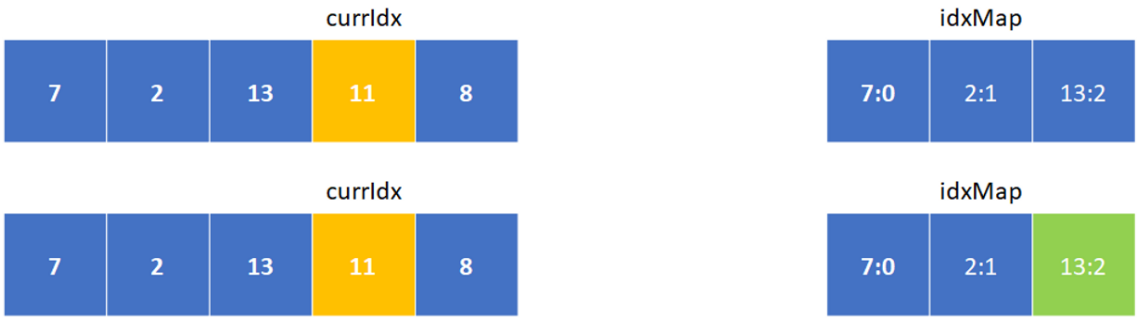
Increment current index, **currIdx** = 2, which is element 13 in the input array. Again, the required number = 24 - 13 = 11 is not in the HashMap. Add {13:2} to **idxMap**. The same is shown in the below diagram.



Our HashMap now contains 3 elements 7, 2, and 13 along with their indexes. Again, we increment **currIdx**, **currIdx** = 3 which is element 11 in the array.

Now required number = 24 - 11 = 13 is present in **idxMap** (shown by the cell highlighted in green in the figure below). That means we have found the pair which adds up to the target sum 24, i.e. (11, 13). Therefore, we return the indexes of 11 and 13 as our result. An index of 11 is nothing but **currIdx** which is 3 and an index of 13 can be found from the HashMap which is 2, therefore we return (3, 2) or (2, 3) as our result.

The worst-case running time complexity of this solution is O(n).



Sample I/O:

Run 1:

Enter the input data array items : 1 2 3
Enter the target value : 4
Adding to hash table: key:1, value: 0
Adding to hash table: key:2, value: 1
Output: 2 0

Run 2:

Enter the input data array items : 7 2 13 11 8
Enter the target value : 24
Adding to hash table: key:7, value: 0
Adding to hash table: key:2, value: 1
Adding to hash table: key:13, value: 2
Output: 3 2

Run 3:

Enter the input data array items : 2 1 7 11 15 17
Enter the target value : 18
Adding to hash table: key:2, value: 0
Adding to hash table: key:1, value: 1
Adding to hash table: key:7, value: 2
Output: 3 2

The submission grading rubric is as follows (points out of 100 total):

Project element	**Points**
Code readability such as the usage of comments in code	10
Proper coding implementation • Brute Force Approach • HashMap Approach	(25+35) = 60 25 35
Screenshots of the program output	10
Program's sample Input/Output format matching as per project writeup	20

Submission Instructions: Create a compressed file (.zip or .tar.gz files are accepted) with all your source files such as .py files. Within this compressed .zip folder, you should provide some screenshots of your running program's output as proof. Generally speaking, to complete the Two Sum Problem, you need maximum two .py files one for the Brute Force Approach and the other for the HashMap Approach. But it's better to submit everything as a compressed file. Submit the compressed file to Canvas.

Late submission policy: As described in the syllabus, any late submission will the penalized with 10% off after each 24 hours late. For example, an assignment worth 100 points turned in 2 days late will receive a 20-point penalty. Assignments turned in 5 or more days after the due date will receive a grade of 0.