Knapsack

locked

Problem | Submissions | Leaderboard | Discussions

Given an array of integers and a target sum, determine the sum nearest to but not exceeding the target that can be created. To create the sum, use any element of your array zero or more times.

For example, if $arr = [2, 3, 4]$ and your target sum is 10, you might select $[2, 2, 2, 2, 2]$, $[2, 2, 3, 3]$ or $[3, 3, 3, 1]$. In this case, you can arrive at exactly the target.

Function Description

Complete the *unboundedKnapsack* function in the editor below. It must return an integer that represents the sum nearest to without exceeding the target value.

unboundedKnapsack has the following parameter(s):

- *k*: an integer
- *arr*: an array of integers

Input Format

The first line contains an integer t, the number of test cases.

Each of the next t pairs of lines are as follows:
- The first line contains two integers n and k, the length of arr and the target sum.
- The second line contains n space separated integers $arr[i]$.

Constraints

$1 \leq t \leq 10$
$1 \leq n, k, arr[i] \leq 2000$

Output Format

Print the maximum sum for each test case which is as near as possible, but not exceeding, to the target sum on a separate line.

Sample Input

```
2
3 12
1 6 9
5 9
3 4 4 4 8
```

Sample Output

```
12
9
```

Explanation

In the first test case, one can pick {6, 6}. In the second, we can pick {3,3,3}.

Java 7

```java
import java.io.*;
import java.math.*;
import java.security.*;
import java.text.*;
import java.util.*;
import java.util.concurrent.*;
import java.util.regex.*;

class Result {

    /*
     * Complete the 'unboundedKnapsack' function below.
     *
     * The function is expected to return an INTEGER.
     * The function accepts following parameters:
     *  1. INTEGER k
     *  2. INTEGER_ARRAY arr
     */

    public static int unboundedKnapsack(int k, List<Integer> arr) {
    // Write your code here

    }

}

public class Solution {
    public static void main(String[] args) throws IOException {
        BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));
        BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter(System.getenv("OUTPUT_PATH")));

        int t = Integer.parseInt(bufferedReader.readLine().trim());

        String[] firstMultipleInput = bufferedReader.readLine().replaceAll("\\s+$", "").split(" ");

        int n = Integer.parseInt(firstMultipleInput[0]);

        int k = Integer.parseInt(firstMultipleInput[1]);

        String[] arrTemp = bufferedReader.readLine().replaceAll("\\s+$", "").split(" ");

        String[] arrTemp = bufferedReader.readLine().replaceAll("\\s+$", "").split(" ");

        List<Integer> arr = new ArrayList<>();

        for (int i = 0; i < n; i++) {
            int arrItem = Integer.parseInt(arrTemp[i]);
            arr.add(arrItem);
        }

        int result = Result.unboundedKnapsack(k, arr);

        bufferedWriter.write(String.valueOf(result));
        bufferedWriter.newLine();

        bufferedReader.close();
        bufferedWriter.close();
    }
}
```

Line: 1 Col: 1

Upload Code as File Test against custom input Run Code Submit Code

DAA-AE2(BATCH-II)-Hacker Rank Test Details ►

Challenges

Current Rank: 1 View your results

Knapsack

Success Rate: **100.00%** Max Score: **60** Difficulty: **Medium**

Try Again

Unique Divide And Conquer

Success Rate: **93.33%** Max Score: **90** Difficulty: **Advanced**

Try Again

Travelling Salesman in a Grid

Success Rate: **96.55%** Max Score: **100** Difficulty: **Expert**

Try Again

Current Leaderboard

Compare Progress

Review Submissions

Message Center