Bonusaufgabe + Mastersolution von letztem Jahr

Ticket Shop

You want to go to a place that is distance D away. There is an array A of n tickets that are availble, where $A[i]$ is the distance that ticket i can cover. You also have k vouchers that allow you to double the distance of **any** k of the tickets that you buy (you can also feel free to not use all of them). Your goal is to calculate the minimum amount of tickets that you need to buy to reach **EXACTLY** distance D (you can assume that this is always possible).

For example, for $D = 20$ and $A = [2, 4, 2, 3, 1, 3, 3, 4, 3]$. With $k = 0$, you can reach 20 with 6 tickets $[3, 3, 3, 3, 4, 4]$. With $k = 1$, you can reach 20 with 5 tickets $[3 \cdot 2, 3, 3, 4, 4]$. With $k = 2$, you can reach 20 with 4 tickets $[3 \cdot 2, 3 \cdot 2, 4, 4]$. With $k = 3$, you can reach 20 with 3 tickets $[3 \cdot 2, 3 \cdot 2, 4 \cdot 2]$.

You need to implement your solution as a method **minTicket(D, k, n, A)** in the file Main.java. You get one point for each passing test set. To pass both test sets, your solution is expected to run in time $O(n \cdot D \cdot k)$.

Attention: You are **NOT** allowed to use additional imports, other than the imports already included in the code template.

```
class Main {
  public static void main(String[] args) {
    // Uncomment the following two lines if you want to read from a file.
    // In.open("public/example.in");
    // Out.compareTo("public/example.out");

    int D = In.readInt();
    int k = In.readInt();
    int n = In.readInt();
    int[] A = new int[n];
    for (int i = 0; i < n; i++) {
      A[i] = In.readInt();
    }
    Out.println(minTicket(D, k, n, A));

    // Uncomment this line if you want to read from a file
    // In.close();
  }

  public static int minTicket(int D, int k, int n, int[] A) {
    // Let val(D, k, n) be the minimum ticket to reach D
    // with k vouchers and first n tickets.
    // The dynamic programming recursion is the following:
    // val(D, k, n) = min{val(D, k, n-1),
    //                    val(D-A[n-1], k, n-1)+1,
    //                    val(D-A[n-1]*2, k-1, n-1)+1}
    // The base cases are D = 0 and n = 0.
    int[][][] val = new int[D+1][k+1][n+1];

    for (int d = 0; d <= D; d++) {
      for (int i = 0; i <= k; i++) {
        for (int j = 0; j <= n; j++) {
          if(d == 0) {
            val[d][i][j] = 0;
          } else if (j == 0) {
            val[d][i][j] = n;
          } else {
            if (d < A[j-1]) {
              val[d][i][j] = val[d][i][j-1];
            } else {
              val[d][i][j] = Math.min(val[d][i][j-1], val[d-A[j-1]][i][j-1]+1);
              if (i > 0 && d >= A[j-1]*2) {
                val[d][i][j] = Math.min(val[d][i][j], val[d-A[j-1]*2][i-1][j-1]+1);
              }
            }
          }
        }
      }
    }

    return val[D][k][n];
  }
}
```