Departement of Computer Science
Markus Püschel, David Steurer

19. November 2018

Datenstrukturen & Algorithmen — Blatt P9 — HS 18

Please remember the rules of honest conduct:

- Programming exercises are to be solved alone
- Do not copy code from any source
- Do not show your code to others

Hand-in: Sunday, 2. December 2018, 23:59 clock via Online Judge (source code only).
Questions concerning the assignment will be discussed as usual in the forum.

Exercise P9.1 *Real Coins.*

Sarah decided to give away her fortune to her nephews Tim and Gordon. The fortune comes in a form of N old coins $c_1, c_2, \dots, c_N$, each of them having a value $v_1, v_2, \dots, v_N$. However, as an avid coin collector, Sarah decided that in order for her nephews to be worthy of her collection they must solve a riddle. Therefore, she asked them to come up with the number of different ways they can distribute the N coins between each other, such that each of them receives a cumulative value that is greater or equal than a given number K.

Help Tim and Gordon get the coin collection by writing an efficient program that solves the riddle.

Input The first line of the input consists of the number N and K such that $1 \leq N \leq 100$ and $1 \leq K \leq 10^5$. The next line contains N numbers: $v_1, v_2, \dots, v_N$ that indicate the values of the coins $c_1, c_2, \dots, c_N$ respectively. For each v_i, we have $1 \leq v_i \leq 10^9$ for $i \in [1 \dots N]$.

Output The output consists of a single number, that indicates the total number of different distribution of coins $c_1, \dots, c_N$ between Tim & Gordon, such that each receives cumulative value of coins greater or equal than K.

Grading You get 3 bonus points if your program works for all inputs. Your algorithm must complete the solution in $O(N \cdot K)$ time complexity, and $O(N + K)$ space complexity. Submit your `Main.java` at `https://judge.inf.ethz.ch/team/websubmit.php?cid=25012&problem=AD18H9P1`. The enrollment password is “`asymptotic`”.

Example

Input:

```
2 6
7 7
```

Output:

```
2
```

Explanation: We have two coins c_1 and c_2. To have a value greater or equal than 6 for each, there are two valid distributions, namely:

1. Tim $= \{c_1\}$ and Gordon $= \{c_2\}$
2. Tim $= \{c_2\}$ and Gordon $= \{c_1\}$

Notes The total number of distributions of coins between Tim and Gordon can grow quite fast, and exceed the precision limits imposed by the primitive types of Java such as `int` and `long`. If such case occurs, use `java.math.BigInteger`, which is an arbitrary-precision integers implementation in Java. Detailed documentation for this class is available at `https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html`.

For this exercise we provide an archive containing a program template available at `https://www.cadmo.ethz.ch/education/lectures/HS18/DA/uebungen/AD18H9P1.RealCoins.zip` The archive also contains additional test cases (which differ from the ones used for grading). Importing any additional Java class is **not allowed** (with the exception of the already imported ones `java.io.{InputStream, OutputStream}`, `java.util.Scanner` and `java.math.BigInteger` class).

Solution There are 2^N possible distributions. We need to calculate the number of invalid distributions, and subtract it from the number of all distributions. To calculate the number of invalid distributions, we need to to find all combinations of coins that sum up to v such that $v \in [0 \dots K-1]$. Then multiply that number by 2, as they are symmetric.

```java
//
// Initialize
//
for (int i = 0; i < N; i += 1) v[i] = scanner.nextInt();
d[0] = BigInteger.ONE;
for (int i = 1; i < K; i += 1) d[i] = BigInteger.ZERO;
//
// Perform dynamic programming, and get all invalid distributions
//
for (int i = 0; i < N; i += 1) {
    for (int j = K - 1; j >= v[i]; j -= 1) {
        d[j] = (d[j - v[i]].add(d[j]));
    }
}
//
// Sum-up all invalid distributions
//
BigInteger invalid = BigInteger.ZERO;
for (int i = 0; i < K;i += 1) invalid = invalid.add(d[i]);
invalid = invalid.shiftLeft(1);
//
// Substract from 2^n
//
BigInteger power2 = BigInteger.ONE.shiftLeft(N);
BigInteger result = power2.subtract(invalid);
```

Exercise P9.2 *Dyno.*

Dyno, the dinosaur of Figure 1, wants to cross a perfectly straight desert. The desert is L meters long and it is split into L segments, indexed from 0 to $L - 1$. Each segment can either be *empty* or it can contain one of the C *cacti* that inhabit the desert. When Dyno is in a generic segment i it can either walk or jump forward. Walking allows it to move from segment i to segment $i + 1$, provided that segment $i + 1$ is empty. Jumping allows it to move from segment i to segment $i + D$, provided that segment $i + D$ is empty (Dyno can jump even if there are cacti between segment $i + 1$ and segment $i + D - 1$. Dyno starts at segment 0, which is always empty, and your job is to help Dyno reach the farthest possible segment in the desert (i.e., the one with the largest possible index).

Input The input consists of a set of instances, or *test-cases*, of the previous problem. The first line of the input contains the number T of test-cases. The first line of each test case contains integers L, D and C, separated by spaces. The second line of each test case contains the locations of the C cacti as n integers separated by spaces. The segment numbers of the cacti locations appear in increasing order.

The inputs satisfy $10 \leq L \leq 1\,000\,000$, $0 \leq C \leq 1\,000\,000$, and $2 \leq D \leq 10$.

Output The output consists of T lines, each containing a single integer. The i-th line is the answer to the i-th test-case, i.e., it contains the the largest index reachable by Dyno.

Grading This exercise awards no bonus points. Your program should run in time $O(L \log(C + D))$. Submit your `Main.java` at `https://judge.inf.ethz.ch/team/websubmit.php?cid=25012&problem=AD18H9P2`. The enrollment password is "`asymptotic`".

Example

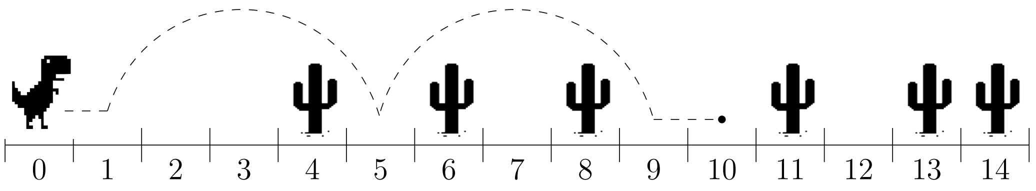

Figure 1: Example input. The dashed line represents the unique optimal solution.[1]

Input (corresponding to Figure 1):

```
1
15 4 6
4 6 8 11 13 14
```

Output:

```
10
```

Notes For this exercise we provide an archive containing a program template available at `https://www.cadmo.ethz.ch/education/lectures/HS18/DA/uebungen/AD18H9P2.Dyno.zip`. The archive also contains additional test cases (which differ from the ones used for grading). Importing any additional Java class is **not allowed** (with the exception of the already imported ones `java.io.{InputStream, OutputStream}` and `java.util.Scanner` class).

[1]Dyno is not to scale.

Solution The problem can be solved by using a dynamic programming algorithm. For $i = 0, \dots, L-1$, we let $\mathrm{OPT}[i]$ be true iff Dyno can reach segment i. Moreover, let E_i be true iff segment i is empty. By the problem definition we know that $\mathrm{OPT}[0] =$ true. For $i > 0$, $\mathrm{OPT}[i] =$ true if (i) segment i is empty and (ii) at least one of the following two conditions holds:

- Dyno can reach segment $i-1$ (as Dyno can walk from segment $i-1$ to segment i); or
- Dyno can reach segment $i-D$, if it exists (as Dyno can jump from segment $i-D$ to segment i).

Otherwise $\mathrm{OPT}[i] =$ false. In formulas: $\mathrm{OPT}[i] = E_i \wedge \big(\mathrm{OPT}[i-1] \vee \mathrm{OPT}[i-D]\big)$, where we assumed that $\mathrm{OPT}[j] =$ false for $j < 0$.

All the values $\mathrm{OPT}[i]$ can be computed in time $O(L)$ by considering the L segments in increasing order of index while keeping track of the position of the next cactus (if any). That is, if `cacti` is the array containing the positions of the C cacti (in sorted order), the algorithm maintains the following invariant: immediately before (resp. after) segment i is considered, the algorithm stores the smallest index k such that $\texttt{cacti}[k] \geq i$ (resp. $\texttt{cacti}[k] > i$), if any. Notice that updating k only requires constant time per segment and that, once k is known, it is possible to check whether $E_i =$ true in constant time.

The solution of the problem is now $\arg\max_{i=0,\dots,L-1} \mathrm{OPT}[i]$, which can be found in $O(L)$ time.