

Problem

You want to withdraw $\text{€}n$ ($1 \leq n \leq 10\,000$) from a cash machine. It can dispense cash in the form of 1, 2, 5, 10, 20, 50, 100, 200, 500 (in euros). Check if the machine can dispense cash adding up to $\text{€}n$ which cannot be split evenly into two piles of $\text{€}n/2$.

Problem

You want to withdraw $\text{€}n$ ($1 \leq n \leq 10\,000$) from a cash machine. It can dispense cash in the form of 1, 2, 5, 10, 20, 50, 100, 200, 500 (in euros). Check if the machine can dispense cash adding up to $\text{€}n$ which cannot be split evenly into two piles of $\text{€}n/2$.

Naive solution (Subset Sum DP + Brute force)

Given a division into coins and notes summing to $\text{€}n$, classic subset sum dynamic programming can be used to check splittability:

$DP[\text{considering notes up to position } i][\text{sum of cash so far}] = \text{can choose subset with this sum}$

Because this DP uses True / False values, it can be sped up using bitsets, to achieve a complexity of $\mathcal{O}(\# \text{ notes } \frac{n}{w})$, where w is the wordsize, typically 32 or 64.

Problem

You want to withdraw $\text{€}n$ ($1 \leq n \leq 10\,000$) from a cash machine. It can dispense cash in the form of 1, 2, 5, 10, 20, 50, 100, 200, 500 (in euros). Check if the machine can dispense cash adding up to $\text{€}n$ which cannot be split evenly into two piles of $\text{€}n/2$.

Naive solution (Subset Sum DP + Brute force)

Given a division into coins and notes summing to $\text{€}n$, classic subset sum dynamic programming can be used to check splittability:

$DP[\text{considering notes up to position } i][\text{sum of cash so far}] = \text{can choose subset with this sum}$

Because this DP uses True / False values, it can be sped up using bitsets, to achieve a complexity of $\mathcal{O}(\# \text{ notes } \frac{n}{w})$, where w is the wordsize, typically 32 or 64.

Try out all possible splittings into all the different coins and notes, and check using the knapsack DP. This is too slow.

Problem

You want to withdraw $\text{€}n$ ($1 \leq n \leq 10\,000$) from a cash machine. It can dispense cash in the form of 1, 2, 5, 10, 20, 50, 100, 200, 500 (in euros). Check if the machine can dispense cash adding up to $\text{€}n$ which cannot be split evenly into two piles of $\text{€}n/2$.

Silly solution

Optimize the brute force as much as possible, using heuristics, and precompute all the answers on your own machine.

Then submit a big file which stores all possible answers up to 10 000 in it.

Observations

- For €1, €5, €10, €50 and €100, using 2 or more of the same coin or note is unnecessary.
For example, switching $€10 \times 2 \implies €20$ is strictly better.

Observations

- For €1, €5, €10, €50 and €100, using 2 or more of the same coin or note is unnecessary.
For example, switching $€10 \times 2 \Rightarrow €20$ is strictly better.
- For €2 and €20, using 5 or more of the same coin or note is unnecessary.
Switching $€2 \times 5 \Rightarrow €10$ is also strictly better.

Observations

- For €1, €5, €10, €50 and €100, using 2 or more of the same coin or note is unnecessary.
For example, switching $€10 \times 2 \implies €20$ is strictly better.
- For €2 and €20, using 5 or more of the same coin or note is unnecessary.
Switching $€2 \times 5 \implies €10$ is also strictly better.

Easy solution

Run the brute force, but use the above observations to reduce the number of options to check. A rough upper bound for the number of options is $2^5 5^2 (10\,000/500) = 16\,000$, but in reality way less options actually sum to n . Here we brute force how many of the lower denominations we choose, also brute force the number of €500 notes.

Greedy solution

It is almost possible to solve this problem greedily, but there are tricky corner cases. This is left as an exercise to the reader.

Greedy solution

It is almost possible to solve this problem greedily, but there are tricky corner cases. This is left as an exercise to the reader.

Statistics: 322 submissions, 33 accepted, 141 unknown