H. Precise Shopping

Editorial

Let's reformulate the problem a little bit. Given an integer B and N coins numbered from 1 to N, where d_i is the value of the i^{th} coin and a_i stands for its weight, your task is to find a set of coins, let's call it S, such that for all $k \in \{1, 2, \ldots, B\}$, there is a subset of S with sum of values equal to k. If there is more than one such set, you have find the one with the smallest sum of weights.

First of all, you should take a look at constraints here. You can see that N is at most 42, which may be a hint here. On one hand, N is too big to iterate over all possible subsets of input coins, but on the other hand, it is small enough to iterate over half of them. This can lead to a very common general approach in this kind of problems. The idea is to split the input coins into two sets, computed some results using exponential algorithm for each of them separately, and finally, combine achieved results into the final solution.

Based on the above general idea, we can came up with the following approach. Let A be the set of $\lfloor \frac{N}{2} \rfloor$ coins with smallest values, and B be the set of the remaining $\lceil \frac{N}{2} \rceil$ coins with largest values (you can decide draws arbitrary here). We say that some set of coins covers an integer k if there exists its subset with a sum of values equal to k. Similarly, we say that some set covers a set of integers $\{1, \ldots, k\}$, if and only if it covers all integers in that set. Moreover, let s_X be the sum of values of coins in s. Having these definitions, for each $X \subseteq A$, we are going check if X can cover the set $\{1, \ldots, s_X\}$, and for each such X, we are going to store put it into the result for A, along with its sum of values and sum of weights. You may wonder how to find out if a subset X can cover some set of values? Well, it can be checked in linear time in terms of the size of X, using a greedy approach. The idea is to process the elements of X in ascending order, and check if the current element x has a value less or equal to the sum of already processed values plus one. If it has, then we add the value of x to the sum and process the next element. In the other case, we know that there is an integer smaller than x which cannot be covered by X. In follows that X can cover a set $\{1, \ldots, k\}$, if and only if we accumulated the sum of elements of at least k during this process.

After performing the above calculations for all subsets of A, we computed the set of subsets of A, covering the sets of integers up to their sums. Next, we are going to iterate over all sets of B, and for each such set Y, we want to compute the biggest gap of uncovered integers in a set $\{1, \ldots, s_Y\}$. The idea behind this approach is that we are going to combine these sets with the sets computed as the result for A, in order to produce a set which covers all integers less or equal to B. Notice that if G is the size of the biggest gap in a set Y, and $X \subseteq A$ which cover a set of integers from 1 to at least G, then $X \cup Y$ covers the set of integers from 1 to B, if and only if the sum of elements in $X \cup Y$ is at least B. This is true, because we can cover each integer in any gap in Y using values from A, and thus all positive integers not greater than B can be covered. Following this fact, we can define a **requirement** of a set Y as the maximum value from the size of the largest sequence of consecutive uncovered integers between 1 and s_Y by Y and the minimum sum of values required to achieve the total sum of B, so the value of $B - s_Y$.

The last remaining thing to show is how to combine two sets $X \subseteq A$ and $Y \subseteq B$, into the resulting set, and moreover, find the ones minimizing the sum of weights of their elements. The idea here is that if a set $Y \subseteq B$ has a requirement R, then it requires a matching set $X \subseteq A$, covering a set of elements from 1 to at least R, so if we sort the subsets returned as a result computed for A in increasing order of sum of their values, and subsets from B in decreasing order of their requirements. In more details, we can find the result by iterating over subsets of B and moving a pointer over the resulting subsets for A, trying to find a match, updating the minimum sum of weight in any previously considered subset of A. The crucial observation here is that any $X \subseteq A$ matched with $Y \subseteq B$ with a requirement R, can be also matched with any $Z \subseteq B$ with a requirement smaller or equal to R. Following this fact, we can compute the minimum weight matching using a linear scan over elements computed as a results for A and B.

The total time complexity of this method is $O(2^{N/2} \cdot N)$, because this is the time needed for computing results for sets A and B. Combining these result into the final one, takes $O(2^{N/2})$ time, so it does not have an affect on overall running time.