Report

Problem Statement :

There are some deer in the zoo. Each deer has two antlers. You are given int[]s antler1 andantler2. These two int[]s will contain the same number of elements. For each index i, antler1[i] and antler2[i] are the weights of the two antlers of one of the deer. You are also given an int capacity. A deer is unbalanced if the weight difference between his antlers is strictly more than capacity. You decided to perform some operations on the deer. Your goal is to make all deer balanced. In each operation, you can choose some two antlers (each on a different deer) and swap them. Return the minimal number of operations required to make all deer balanced. If this is impossible, return -1 instead.

Input Specification :

Input should be in the form of
1. no of deers
2. the first set of antlers
3. the second set of antlers
4. capacity

Output Specifications :

returns the minimum number of swappings required if its possible
returns -1 if the swappings are not possible

Design :

Assume that the antler that are to be swapped are already fixed now we have to find the minimum number swapping required to do the swapping. For that we first give a representation let each of the deer be represented as a vertex in a graph there exists an edge between vertex A and vertex B if one of the antlers of deer A and one of the antlers of deer B would be forming a pair of antlers of some deer. In this representation we can observe that every vertex in the graph will be having a degree 2 because every deer has two antlers and we can also observe that every connected component of the graph will form a loop. The minimum number of swapping required in a loop will be number of vertices's minus 1 and the total number of swappings in the graph will be total number of vertices's minus the number of connected components.

Now we can re-represent the problem as dividing the deer into maximum number of connected cycles inorder to minimize the number of swappings. Let us assume that there exists a function which when given a subset of deer returns true if they can form a cycle and false if they can't let the name of the function be iscycle[subset] we use bit manipulation to get different subsets of the main set that is we have the original set of deer the number of subsets possible are 2^(number of deer) each subset is represented from 0 to (2^(number of deer) - 1) each number representing weather an element is present in the set or not for example if the number 5 in binary (101) represents the subset that contains the elements a0 and a2 now for each of the subsets we find weather a cycle is present or not we do it by taking all the weights of the antlers and sorting them we can say that (proof in proof section) when ever we want to pair an antler with weight w we make a greedy choice of pairing it with an antler with weight just over w that is we choose w1 such that w1 > w and w1 less that all the other elements in the set that are not paired and we check if the difference between those is less than capacity if greater we can say that they cant form a pair if less we continue to pair others in the set

let max[n] represent the maximum number of non overlapping cycles for a subset n (n is a subset of total set) for all subsets of n let it be s the maximum number of non overlapping cycles is max((1+max[n-s0]) (1 + max[n-s1]) ….....) where s0 s1 s2 ….. are subsets that can form cycles the expression says that if a susbet s is selected then the maximum number of cycles it can form is 1 plus the maximum number of cycles we can form by the others we follow a bottom up approach and finally calculate max[set] and max[0] = 0 we cannot form any cycles with no elements

Proof of Correctness:

we have two proves to do one is prove the optimal substructure property of the problem for finding the maximum number of non overlapping cycles and the greedy choice in choosing the smallest in the set for determining weather a subset forms a loop or not

Optimal substructure:

let us assume that there exists a subset that do not follow the optimal substructure property but contains maximum number of cycles but by optimal substructure it is max((1+max[n-s0]) (1 + max[n-s1]) ….....) which contradicts

Greedy choice property:

In determining weather a group of antlers can form a cycle or not we make a greedy choice of choosing the smallest antler suppose if for some x1 which is smallest we can minimize the difference only by choosing the next smallest(x2) if this difference is not less than the capacity we cannot find another antler to pair up with x1 because x2 is the smallest

Sample Inputs:

1. {3, 2, 2}
   {3, 5, 5}
   0
2. {4, 2, 6, 4, 8, 5, 2, 3}
   {3, 4, 5, 2, 8, 5, 7, 6}
   1
3. {12, 34, 56, 78}
   {1234, 2345, 3456, 4567}
   100

Sample Outputs:

1. 1
2. 2
3. -1