FUNDAMENTALS OF ARTIFICIAL INTELLIGENCE - CS161

Programming Assignment 3 - Due Tuesday, January 29, 11:59 p.m.

DO NOT CONSULT ANY OUTSIDE REFERENCES OR SOURCES FOR THIS ASSIGNMENT

This assignment concerns the problem of number partitioning. Given a set of integerts, the task is to divide them into two mutually exclusive and collectively exhaustive subsets, so that the sums of the numbers in each subset are as nearly equal as possible. For example, given the set (1 4 9 16 25), an optimal partition divides the numbers into the sets (1 25) and (4 9 16). Since the sum of the numbers in the first set is 26, and the sum of the numbers in the second set is 29, the difference of the sums is 3, and this is the smallest difference that can be achieved in this case.

Your task is to write a pure LISP program to optimally partition sets of integers. Your top-level function, called PARTITION, must take a single argument which is a list of non-negative integers, and return an optimal partition of those integers. The value returned should be a three-element list, where the first element is the final partition difference, and the next two elements are the subsets that achieve that value. For example, (PARTITION '(1 4 9 16 25)) might return (3 (1 25) (4 9 16)). The order of the integers in the lists, and the order of the two lists, is irrelevant.

You will need to write a function to generate test data. The common-lisp function (random n) will return a random number in the range zero to n. Number partitioning problems generally become more difficult with increasing magnitude of the numbers, so be sure to test your program against integers at least as large as a million. Note that it is feasible to optimally solve instances with very large numbers of integers if the integers are small.

This problem is NP-Complete, meaning that you (almost certainly) have to do an exponential search to find an optimal solution. DO NOT IMPLEMENT A POLYNOMIAL-TIME ALGORITHM, BECAUSE IT WILL NOT PRODUCE CORRECT ANSWERS FOR ALL PROBLEM INSTANCES. You are encouraged to first implement and experiment with a simple search algorithm to solve the problem. In order to solve instances with more and larger integers, however, you will have to improve the efficiency of your algorithm. Hacking your code will not buy you big gains in performance, but thinking about and experimenting with your algorithm will. Remember that all your partitions must be optimal. I'm not aware of any useful heuristic evaluation functions for this problem.

If you are having trouble getting started, I suggest that you first implement a simpler version of this assignment that only returns the numerical difference of the final partition, and not the partition itself. If you call this function SIMPART, and you don't get PARTITION working correctly, then you'll receive partial credit for SIMPART.