Steven Sawtelle Hw3 CSE294

Magic Squares

Version 1

1. Take in input 3
2. Find magic square solutions for 3
3. Print those out

Version 2

1. Main:
    a. Take in input 3
    b. Get magic constant (15) as *mag*
    c. Send 3 to process(x, mag), get back *result*
    d. Print result
2. process(x, mag):
    a. list of possible permutations as *lis*
    b. get permutation of [1,2,3,4,5,6,7,8,9]
    c. send first 3 of permutation to a0
    d. send second 3 to a1
    e. send last 3 to a2
    f. call check(a0, a1, a2, n, mag) and get back pass
    g. if pass is true add current permutation to *lis*
    h. do this for all permutations
    i. when done, return *lis*
3. check(a0, a1, a2, n, mag):
    a. start pass as true
    b. check all of the math needed to check if condition is broke
    c. if any condition breaks set pass to False
    d. return pass

Version 3

1. Main:
    a. take in input (should be 3 for this example) <- *n*
    b. determine magic constant on *n* <- *mag*
    c. process(x, mag) <- *result*
    d. print *result*
2. process(x, mag):
    a. [] <- *lis*
    b. [] <- *baseP*
    c. for x in range of 1, n^2 + 1:
        i. add x to *baseP*
    d. all permutations of *baseP* <- *perms*
    e. for *perm* in *perms*:

        i. first *n* elements of *perm* <- *a0*
        ii. second *n* elements of *perm* <- *a1*
        iii. last *n* elements of *perm* <- *a2*
        iv. *checkMag*(a0, a1, a2, n, mag) <- *pass*
        v. if *pass* is true add *perm* to *lis*
    f. return *pass*
3. check(a0, a1, a2, n, mag):
    a. True <- *pass*
    b. if sum of all elements of a0 != mag set pass to False
    c. if sum of all elements of a1 != mag set pass to False
    d. if sum of all elements of a2 != mag set pass to False
    e. if sum of all first elements != mag set pass to False
    f. if sum of all second elements != mag set pass to False
    g. if sum of all third elements != mag set pass to False
    h. if sum of first from a0, second from a1, and third from a2 != mag set pass to False
    i. return pass

Minimum Scalar Product

Version 1

1. While test cases still need to be ran do the following:
    a. Take in two vectors
    b. Determine the smallest permutation of the products of the two vectors
    c. Return that product

Version 2

1. Main:
    a. Get number of test cases
    b. Get number of contents in each vector as *size*
    c. Take in the two vectors
    d. Send these two vectors to process(x, y, size) and get back *min*
    e. Print *min*
    f. If test cases still left to perform, go to step 1b
2. Process(x, y, size):
    a. Takes in two vectors and one int
    b. Gets all permutations of vectors x and y as *Px and Py*
    c. *Min* <- maximum int
    d. For each permutation in Px - px
        i. For each permutation in Py - py
            1. *Product* is all of them multiplied by each other and added together
            2. If *product* is less than *min, min = product*
    e. Return *min*

Version 3

a. main:
    a. *n* <-get number of test cases
    b. while n>0
        i. *size* <- get number of contents in each vector
        ii. *x* <- first vector of *n* length
        iii. *y* <- second vector of *n* length
        iv. *min* <- *process(x, y, size)*
        v. print *min*
        vi. *n*-=1
b. *process*(*x, y, size*)
    a. use itertools for permutations
    b. Px <- permutations of x
    c. Py <- permutations of y
    d. *min* <- max int
    e. for *px* in *Px*
        i. *Py* <- permutations of *y*
        ii. for *py* in *Py*
            1. *product* <- px[0] * py[0] + px[1] * py[1] + px[2] * py[2]
            2. if *product* < *min*
                a. *min* <- *product*
    f. return *min*