Optimal Cozy Levels

Objective
Give practice with recursion in C.
Give practice with a combinations style algorithm in C.

Story
Your coworkers at the KAT Pet Shop are furiously working at catching all the elephants and placing the new baits. In lighter news the 3 families that have been fostering cats now want to permanently adopt a group of them. The only problem is that they want to adopt the same group of cats.

You are here to mediate by determining which cats will go to which family. Some cats are cozier in certain houses, and some cats will increase or decrease the coziness of other cats that wind up in the same house. Luckily, you have knowledge of exactly how much each cat affects the coziness of each other cat, and how much each family affects the coziness of each cat.

Each family will need to have at least 1 cat, and each cat will be adopted out to 1 of the 3 families. We are interested in 2 metrics,

1. The coziness sum across all cats
2. The coziness of the least cozy cat

Problem
You will be given the number of cats, the initial coziness for each family for each cat, and the amount of coziness that each cat will change for each other cat based on their presence within the same family.

Input
The input will begin with a single integer, N ($3 \leq N \leq 15$), representing the number of cats. The next N lines each contain 3 integers, the i-th line represents the coziness the i-th cat will start with at each of the corresponding families. This value could be negative.

Following this will be N lines that each contain N integers. The i-th integer on the j-th line represents the amount of coziness the j-th cat will change by (positive is an increase and negative is a decrease), if the i-th cat is adopted by the same family as the j-th cat. The diagonal will always be 0. That is the i-th cat will not affect its own coziness value.

You are guaranteed that each coziness value will be in the range of [-1000, 1000].

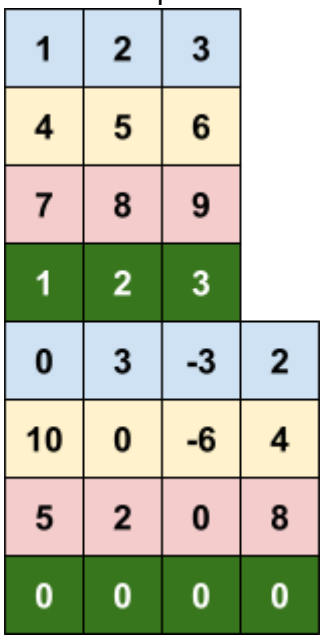
Output
Print 2 integers on a single line. The first integer should be the maximum coziness sum possible and the second integer should be the coziness of the least cozy cat when assigning the cats such that the least cozy cat is as cozy as possible.

Sample Input	Sample Output
4 1 2 3 4 5 6 7 8 9 1 2 3 0 3 -3 2 10 0 -6 4 5 2 0 8 0 0 0 0	31 3
5 5 4 1 2 4 1 1 5 3 5 2 4 2 1 1 0 6 0 3 7 -1 0 3 5 7 0 6 0 4 1 -2 6 1 0 -2 -2 6 4 -2 0	46 5

Sample Explanation

FOR CASE 1

In this case there are 4 cats. Below is a color coded picture that demonstrates which values affect different cat's coziness. Each color corresponds to a different cat's cozy level.



To maximize sum one arrangement is the following

- Cat 1 → House 3
- Cat 2 → House 3
- Cat 3 → House 1
- Cat 2 → House 2

With the above arrangement the coziness would be the following,

- Cat 1 → 6 Cozy (3 base + 3 from Cat 2)
- Cat 2 → 16 Cozy (6 base + 10 from Cat 1)
- Cat 3 → 7 Cozy (7 base)
- Cat 2 → 2 Cozy (2 base)

The sum would be 6 + 16 + 7 + 2 = 31.

To maximize the least cozy cat we could use the following arrangement,

- Cat 1 → House 2
- Cat 2 → House 2
- Cat 3 → House 1
- Cat 4 → House 3

With the above arrangement the coziness would be the following,

- Cat 1 → 5 Cozy (2 base + 3 from Cat 2)
- Cat 2 → 15 Cozy (5 base + 10 from Cat 1)
- Cat 3 → 7 Cozy (7 base)
- Cat 4 → 3 Cozy (3 base)

The least cozy cat would be Cat 4 with a value of 3.

CASE 2

An optimal arrangement to maximize the sum is,

- Cat 1 → House 1
- Cat 2 → House 2
- Cat 3 → House 2
- Cat 4 → House 3
- Cat 5 → House 2

An optimal arrangement to maximize the least cozy cat is,

- Cat 1 → House 1 (5 Cozy)
- Cat 2 → House 2 (11 Cozy)
- Cat 3 → House 3 (7 Cozy)
- Cat 4 → House 3 (5 Cozy)
- Cat 5 → House 2 (7 Cozy)

Hints

Recursion Usage: You need to for each try try all possible house assignments, but you need to allow this to happen for all previous cat assignments.

Base Case Validity: Make sure that each house has a cat assigned.

NO PROMPTS: Do NOT prompt for input. Do not print messages like, “Please enter the number of cats:” or the like.

NO LABELS: Do NOT label the output. Do not print messages like, “The cat in cozy cage 1 is…“.

Struct Recommendations: No structs were used in the instructors solution

Global Memory: I recommend using global memory to hold onto the arrays and/or the answers; it will make your function headers more manageable.

Function Recommendations: The instructor’s solution only used the main function and the recursive function.

Anwer Initialization: Make sure that your program can account for potentially negative answers.

Grading Criteria

- Read/Write from/to **standard** input/output (e.g. scanf/printf and no FILE *)
    - 5 points
- Good comments, whitespace, and variable names
    - 15 points
- No extra input output (e.g. input prompts, “Please enter the number of cats:”)
    - 5 points
- Checks the validity of the cat assignments before updating answers
    - 5 points
- Can update the answers from the base case
    - 5 points
- Tries all 3 possibilities for each cat
    - 5 points
- Computes the coziness for the cats based on the assignments
    - 5 points
- Stores all the input values in arrays of some kind
    - 5 points
- Programs will be tested on 10 cases
    - 5 points each

No points will be awarded to programs that do not compile using “gcc -std=gnu11 -lm”.

Sometimes a requested technique will be given, and solutions without the requested technique will have their maximum points total reduced. For this problem use recursion. ***Without this programs will earn at most 50 points!***

Any case that causes a program to return a non-zero return code will be treated as wrong. Additionally, any case that takes longer than the maximum allowed time (the max of {5 times my solutions time, 10 seconds}) will also be treated as wrong.

No partial credit will be awarded for an incorrect case.