

Optimal Noise Reduction

Objective

Give practice with Recursion in C.
Give practice with Backtracking in C.
Give practice with Stacks in C.

Story

The animals are still making too much noise. Some animals antagonize nearby animals and cause them to emit screams of rage or cries of annoyance. However, some animals want to be nearby other particular animals or they emit cries of longing. You have experimented with different animals at different distances from each other. You have made some notes, and to make your life easier you have represented each animal with a unique natural number index. Now you feel that you now have enough information to move forward with your new plans.

The animals will be in a line of exhibits. Each animal will be in its own exhibit, and there are an equal number of animals as exhibits. You have found that certain pairs of animals need to be exactly some distance from each other. You want to find out if there is an arrangement of animals that satisfies all the required distances.

Problem

Given the number of animals and the required distance pairs of certain animals, write a program that prints a satisfying solution or informs the user that there is none if a solution does not exist.

Input

Input will begin with a line containing 2 integers, n and c ($1 \leq n \leq 20$; $0 \leq c \leq n$ choose 2), representing the number of animal exhibits and the number of constraints. The following c lines will each contain a constraint description in the form of 3 space separated integers, f , s , and d ($1 \leq f, s \leq n$; $f \neq s$; $0 \leq d \leq n - 2$), representing the index of the first animal, the index of the second animal, and the exact number of cages that should be in between those 2 animals.

For 7 of the 10 cases each animal will be in at most 1 constraint.

Output

Output should contain 1 line. In the event that a solution exists, you should print out n space separated integers representing the animal index given in order from the first cage to the last. If there is no solution the output should be only the phrase "No Solution." (quotes for clarity).

Sample IO on next page.

Sample Input	Sample Output
4 2 1 2 1 3 4 0	No Solution
5 4 1 2 1 1 3 1 1 5 0 2 5 2	2 4 1 5 3

Explanation

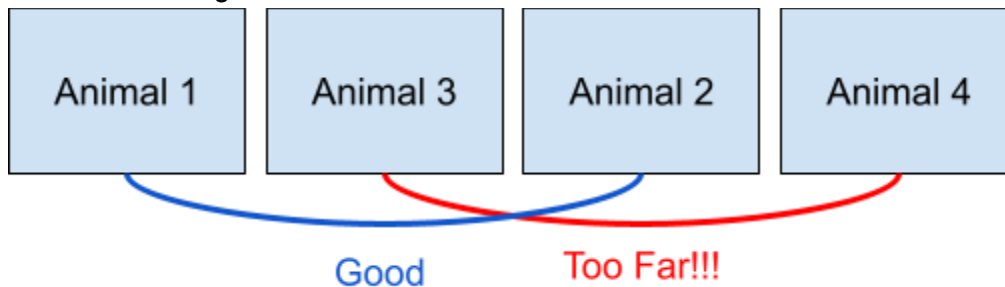
Case 1

There are 4 animals in this case. There are 2 constraints on the animals,

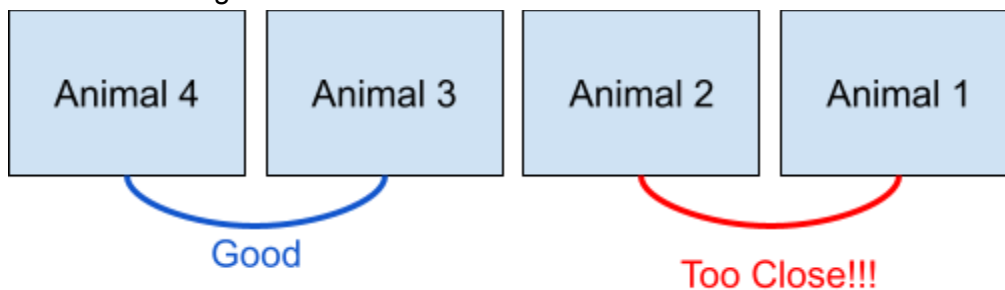
- The first constraint is that animals 1 and 2 must have exactly one exhibit in between them.
- The second constraint is that animals 3 and 4 must have no cages between them.

No matter how you arrange the animals. There is no arrangement (permutation) that will ensure that all the constraints are met.

Here is an arrangement where the first constraint is met but not the second.



Here is an arrangement where the second constraint is met but not the first.

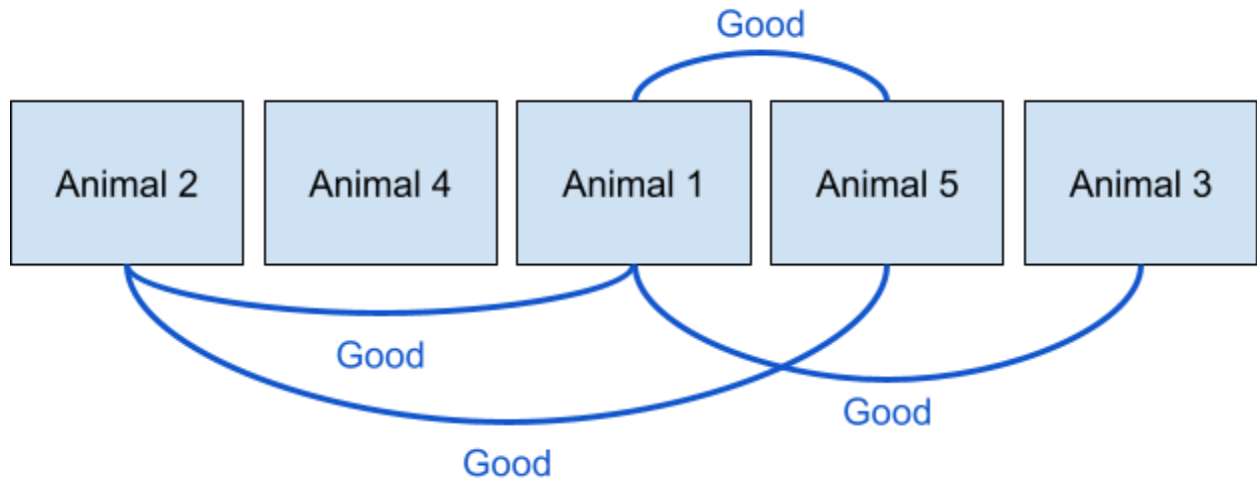


Case 2

There are 5 animals in the second case, and 4 constraints. The constraints are the following,

- Animals 1 and 2 should have a single cage between them.
- Animals 1 and 3 should have a single cage between them.
- Animals 1 and 5 should have no cages between them.
- Animals 2 and 5 should have two cages between them.

There are valid arrangements. Below is one of them



Another acceptable arrangement would be 3 5 1 4 2.

Hints

Permutation Like: You need to find a valid permutation of animals in cages.

Constraints: There are constraints on the permutations. If your partial permutation violates a constraint, then you should skip the rest of the permutation.

Unconstrained Animals: Animals that are unconstrained can cause a significant impact on performance. If you place an unconstrained animal at some location during your recursive function, and find no solution, then placing a different unconstrained animal at the same location at this point in recursion will also result in no solution. Handling this case can greatly improve performance in cases with many unconstrained animals.

Ordered placement: I recommend placing animals from one end of the permutation to the other (for reasons mentioned below).

Forced Moves: When you place an animal with constraints, any unplaced animal that is constrained with the recently placed animal should have their own placement forced. If you are placing animals from left to right, the left option will not be available for any unplaced animal (assuming this placement is not because of a different forced placement).

For example, suppose we have placed animals in spots 1, 2, and 3 and we place animal 10 at spot 4 and animal 9 (which is unplaced) needs to have 1 cage in between them and animal 10. Then we know that animal 9 **must** go in spot 6. Otherwise the constraint would be violated.

Grading Criteria

- Good comments, whitespace, and variable names
 - 15 points
- No extra input output (e.g. input prompts, "Please enter the number of friends:")
 - 10 points
- Prevent reusing an animal multiple times in your arrangement
 - 10 points
- Check for constraint violations for each placed animal.
 - 5 points
- After using an animal for some possible arrangement make sure to undo any changes to allow for reusing the animal in the future.
 - 10 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 backtracking. **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.

Challenge: If you want to give yourself a challenge, try writing a program that outputs an optimal order of animals.