●●●● Not Frequent 0/4

Casework

Author: Aarav Sharma

Dividing into multiple cases and solving for each one separately.

Language: All Edit This Page

TABLE OF CONTENTS

A casework problem is one that can be broken down into different cases that can each be solved for separately.

These usually require drawing out a lot of cases and making observations about each.

We can try to spot similarities between different cases in order to solve multiple of them with the same method.

Sleepy Cow Herding
Bronze - Normal

Focus Problem – try your best to solve this problem before continuing! *View Internal Solution*

Solution - Sleepy Cow Herding

Solution

Let's solve for the minimum and maximum numbers of moves independently. We refer to the cow positions as "elements," and say that two elements are adjacent if there are no elements in between them.

Minimum

There are 3 cases for the minimum amount of moves:

1. The 3 elements are already consecutive.
2. There are two adjacent elements with exactly one empty position in between them.
3. Any other case that did not satisfy the two above.

For the first case, the answer would be 0 because the elements are already consecutive.

For the second case, the answer would be 1 because the only move required is the one that would insert the isolated element into the gap between the two other elements.

The third case would output 2 because for any other test case, the optimal solution would be to either take the minimum element to $\max - 2$ or the maximum element to $\min + 2$, and then we have reduced to the second case.

Maximum

The maximum will always be finite because every operation strictly decreases the distance between the endpoints (let's refer to this as the "span") by at least one. The best approach to maximize the amount of moves is to place each element as close to an endpoint as possible (while not remaining an endpoint). After one move, the span will equal the largest distance between two adjacent elements, and every subsequent move after that will decrease the span by one until it reaches two. Therefore, the maximum is the largest distance between two adjacent elements minus 1.

Implementation

Time Complexity: $\mathcal{O}(1)$

C++

Copy CPP

```cpp
#include <algorithm>
#include <iostream>
#include <vector>

using namespace std;

int main() {
    freopen("herding.in", "r", stdin);
    freopen("herding.out", "w", stdout);

    vector<int> a(3);
    for (int &b : a) { cin >> b; }
    sort(a.begin(), a.end());

    /*
     * The minimum number of moves can only be 0, 1, or 2.
     * 0 is if they're already consecutive,
     * 1 is if there's a difference of 2 between any 2 numbers,
     * and 2 is for all other cases.
     */
    if (a[0] == a[2] - 2) {
        cout << 0 << endl;
    } else if ((a[1] == a[2] - 2) || (a[0] == a[1] - 2)) {
        cout << 1 << endl;
    } else {
        cout << 2 << endl;
    }

    // max is equal to largest difference between end and middle, minus one.
    cout << max(a[2] - a[1], a[1] - a[0]) - 1;
}
```

Java

Copy JAVA

```java
import java.io.*;
import java.util.*;
class Main {
    public static void main(String[] args) throws IOException {
        Scanner sc = new Scanner(new File("herding.in"));
        PrintWriter pw = new PrintWriter(new File("herding.out"));
        int[] cows = new int[3];
        cows[0] = sc.nextInt();
        cows[1] = sc.nextInt();
        cows[2] = sc.nextInt();
        Arrays.sort(cows);

        /*
         * The minimum number of moves can only be 0, 1, or 2.
         * 0 is if they're already consecutive,
         * 1 is if there's a difference of 2 between any 2 numbers,
         * and 2 is for all other cases.
         */
        if (cows[2] == cows[0] + 2) {
            pw.println(0);
        } else if (cows[1] == cows[0] + 2 || cows[2] == cows[1] + 2) {
            pw.println(1);
        } else {
            pw.println(2);
        }

        // max is equal to largest difference between end and middle, minus one.
        pw.println(Math.max(cows[1] - cows[0], cows[2] - cows[1]) - 1);
        pw.close();
    }
}
```

Python

Copy PYTHON

```python
with open("herding.in", "r") as file_in:
    a, b, c = map(int, file_in.readline().split())

"""
The minimum number of moves can only be 0, 1, or 2.
0 is if they're already consecutive,
1 is if there's a difference of 2 between any 2 numbers,
and 2 is for all other cases.
"""
if c == a + 2:
    minimum = 0
elif b == a + 2 or c == b + 2:
    minimum = 1
else:
    minimum = 2

# max is equal to largest difference between end and middle, minus one.
maximum = max(b - a, c - b) - 1

print(f"{minimum}\n{maximum}", file=open("herding.out", "w"))
```

Problems

STATUS	SOURCE	PROBLEM NAME	DIFFICULTY	TAGS
	Bronze	Fence Painting	Normal	▸ Show Tags Casework
	Bronze	Hoof Paper Scissors Minus One	Normal	▸ Show Tags Casework
	Bronze	Leaders	Hard	▸ Show Tags Casework

Module Progress: Not Started