Daily Coding Problem #309

Problem

This problem was asked by Walmart Labs.

There are `M` people sitting in a row of `N` seats, where `M < N`. Your task is to redistribute people such that there are no gaps between any of them, while keeping overall movement to a minimum.

For example, suppose you are faced with an input of `[0, 1, 1, 0, 1, 0, 0, 0, 1]`, where `0` represents an empty seat and `1` represents a person. In this case, one solution would be to place the person on the right in the fourth seat. We can consider the cost of a solution to be the sum of the absolute distance each person must move, so that the cost here would be five.

Given an input such as the one above, return the lowest possible cost of moving people to remove all gaps.

Solution

Intuitively, we would like to bunch people towards the center. For example, in the case where there are three people sitting spaced out evenly across the row, the optimal solution is to move the left and right person adjacent to the middle one.

```
[1, 0, 0, 0, 1, 0, 0, 0, 1] --> [0, 0, 0, 1, 1, 1, 0, 0, 0]
```

```
[1, 0, 0, 0, 1, 0, 0, 0, 1] -> [0, 0, 0, 1, 1, 1, 0, 0, 0]
```

In a more complicated situation, we must first figure out where the middle is. It turns out, though, that we can always choose the location of the median person. An informal explanation for this is as follows: first, there is no point in choosing a center where no one is currently sitting. And second, if we have to choose among the occupied positions, it makes the most sense to choose the median, since that minimizes the distance from both the left and right sides.

To implement this solution, we can start at the median and extend outward to the left, and then to the right. Each time we come across an empty seat, we swap it (if possible) with a person sitting further out. Once we have reached the end of the row, or there are no more people to swap, we know everyone has been bunched to the center.

```python
def move(seats):
    people = [i for i, x in enumerate(seats) if x == 1]
    n = len(people)
    median = people[n // 2]
    cost = 0

    # Move left seats closer to median.
    i = median - 1; j = n // 2 - 1
    while i >= 0 and j >= 0:
        if seats[i] == 0:
            cost += abs(people[j] - i)
            seats[i], seats[people[j]] = seats[people[j]], seats[i]
            j -= 1
        i -= 1

    # Move right seats closer to median.
    i = median + 1; j = n // 2 + 1
    while i < len(seats) and j < n:
        if seats[i] == 0:
            cost += abs(people[j] - i)
            seats[i], seats[people[j]] = seats[people[j]], seats[i]
            j += 1
        i += 1

    return seats, cost
```

This algorithm requires two passes over the input: one from the median to the left end, and one from the median to the right end. Since in each pass we only perform constant-time operations, the time complexity will be `O(N)`. The space required will also be `O(N)`, since we are using a helper array to store the occupied positions.