Code Jam 2013 - Round 1C

Analysis: Pogo

The small dataset

The small dataset did not require finding the optimal solution, instead accepting any solution that solved the problem within 500 moves. This allowed for a variety of approaches. One such approach is as follows: note that with two subsequent jumps in two opposite directions you can move one unit in a chosen direction. By a sequence of at most 200 such pairs (so in total at most 400 moves) you can reach any point with |X|, |Y| ≤ 100.

There were also other approaches possible, for instance a shortest-path search, if one could prove or guess that it's OK to limit the search space somehow. It turns out that one can actually just search for a path within the points with |x|, |y| ≤ 100, we will see why in the next section. Thus, Dijkstra's algorithm or just breadth-first search will provide a short enough path to the target.

The large dataset

For the large dataset, we not only need to return the best possible solution, but also deal with more distant targets, so neither of the approaches above will work. We will begin with a few easy observations:

- First, if we want to reach the target in **N** moves, we have to have 1 + 2 + ... + **N** ≥ |**X**| + |**Y**|.
- Moreover, if we want to reach the target in **N** moves, the parity of the numbers 1 + 2 + ... + **N** and |**X**| + |**Y**| has to be the same. This is because the parity of the sum of the lengths of jumps we make in the North-South direction has to match the parity of |**Y**|, and the sum of lengths of West-East jumps has to match the parity of |**X**|.

It turns out that if **N** satisfies these two conditions, it is possible to reach (X, Y) with **N** jumps.

Let's consider a point (**X**, **Y**), and any **N** satisfying the two conditions above. For the sake of brevity, assume |**X**| ≥ |**Y**|, and **X** ≥ 0 (it's easy to provide symmetric arguments for the other four cases). In this case, we will assume the last move was East. This means the first **N**-1 moves have to reach (**X**-**N**, **Y**). We will proceed recursively, so we just have to prove that (**X**-**N**, **Y**) and **N**-1 satisfy the conditions above.

For **N** = 1 or 2, it's easy to enumerate all the possible **X** and **Y**. For **N** = 1, there are four possibilities, and in each case our strategy produces the correct move. For **N** = 2, if we assume **X** is positive and greater than |**Y**|, the only possibilities are (3, 0), (2, 1), (2, -1) and (1, 0); after the move they turn to (1, 0), (0, 1), (0, -1) and (-1, 0) — all of which satify the conditions for **N** = 1.

For larger **N**, the parity condition is trivial — both considered parities stay the same if **N** was even, and both change if **N** is odd. For the inequality condition, if **N** ≤ **X**, both sides just decrease by **N**, so the only interesting case is if **X** < **N**. However, in this case, we move to (**X** - **N**, **Y**), and the sum of absolute values is **N** - **X** + |**Y**| ≤ **N** - **X** + **X** = **N** ≤ 1 + ... + **N** - 1. Thus, the conditions are satisfied after one move, and we can continue with our strategy.

This logic again translates to simple code:

```
def Solve(x, y):
  N = 0
```

```
sum = 0
while sum < abs(x) + abs(y) or (sum + x + y) % 2 == 1:
  N += 1
  sum += N
result = ""
while N > 0:
  if abs(x) > abs(y):
    if x > 0:
      result += 'E'
      x -= N
    else:
      result += 'W'
      x += N
  else:
    if y > 0:
      result += 'N'
      y -= N
    else:
      result += 'S'
      y += N
  N -= 1
return result.reversed()
```