Directions reduction

Once upon a time, on a way through the old wild *mountainous* west,...

... a man was given directions to go from one point to another. The directions were "NORTH", "SOUTH", "WEST", "EAST". Clearly "NORTH" and "SOUTH" are opposite, "WEST" and "EAST" too.

Going to one direction and coming back the opposite direction *right away* is a needless effort. Since this is the wild west, with dreadful weather and not much water, it's important to save yourself some energy, otherwise you might die of thirst!

How I crossed a *mountainous* desert the smart way.

The directions given to the man are, for example, the following (depending on the language):

```
["NORTH", "SOUTH", "SOUTH", "EAST", "WEST", "NORTH", "WEST"].
or
{ "NORTH", "SOUTH", "SOUTH", "EAST", "WEST", "NORTH", "WEST"
};
or
[North, South, South, East, West, North, West]
```

You can immediately see that going "NORTH" and *immediately* "SOUTH" is not reasonable, better stay to the same place! So the task is to give to the man a simplified version of the plan. A better plan in this case is simply:

```
["WEST"]
or
{ "WEST" }
or
[West]
```

Other examples:

In `["NORTH", "SOUTH", "EAST", "WEST"]`, the direction `"NORTH" + "SOUTH"` is going north and coming back *right away*.

The path becomes `["EAST", "WEST"]`, now `"EAST"` and `"WEST"` annihilate each other, therefore, the final result is `[]` (nil in Clojure).

In ["NORTH", "EAST", "WEST", "SOUTH", "WEST", "WEST"], "NORTH" and "SOUTH" are *not* directly opposite but they become directly opposite after the reduction of "EAST" and "WEST" so the whole path is reducible to ["WEST", "WEST"].

Task

Write a function `dirReduc` which will take an array of strings and returns an array of strings with the needless directions removed (W↔E or S↔N *side by side*).

- The Haskell version takes a list of directions with `data Direction = North | East | West | South`.
- The Clojure version returns nil when the path is reduced to nothing.
- The Rust version takes a slice of `enum Direction {North, East, West, South}`.

See more examples in "Sample Tests:"

Notes

- Not all paths can be made simpler. The path ["NORTH", "WEST", "SOUTH", "EAST"] is not reducible. "NORTH" and "WEST", "WEST" and "SOUTH", "SOUTH" and "EAST" are not *directly* opposite of each other and can't become such. Hence the result path is itself : ["NORTH", "WEST", "SOUTH", "EAST"].
- if you want to translate, please ask before translating.

```
def dir_reduc(arr):
    red = []
```

```
    opposite = {"NORTH": "SOUTH", "SOUTH": "NORTH", "EAST": "WES

    for direction in arr:
        if red and red[-1] == opposite[direction]:
            red.pop()  # Remove the last added direction from re
        else:
            red.append(direction)  # Add the current direction t

    return red
```

a new list is initialised for the reduction

a dictionary is created called opposite that stores key value pairs

key being the direction and value being the opposite of the direction

iterates directly through elements of the array

compares if the last added element of red is == to the **opposite direction** of the current direction

else it just adds the direction