Hydrothermal Venture

Francesc Rocher

Hydrothermal Venture

Advent of Code 2021 / Day 5

Advent of Code is an Advent calendar of small programming puzzles for a variety of skill sets and skill levels that can be solved in any programming language you like.

Contents

List of Figures

Listings

You come across a field of hydrothermal vents on the ocean floor!

AOC 2021/05

1 Introduction

Please read Advent of Code 2021 / Day 5 carefully for all the puzzle details.

1.1 Context

The puzzle story introduces the following concepts and elements:

1. There are thermal vents in the ocean floor that produce opaque clouds.
2. Thermal vents tend to form in lines.
3. Puzzle input contains start and end points, (x_0, y_0) and (x_1, y_1), of each thermal vent.
4. Thermal vents can be drawn in a diagram where the top-left corner is the coordinate $(0, 0)$, horizontal axis is X and vertical axis is Y.

Example of the input file and the associated diagram:

```
Input                   Diagram
----------              ----------
0,9 -> 5,9              .......1..
8,0 -> 0,8              ..1....1..
9,4 -> 3,4              ..1....1..
2,2 -> 2,1              .......1..
7,0 -> 7,4              .112111211
6,4 -> 2,0              ..........
0,9 -> 2,9              ..........
3,4 -> 1,4              ..........
0,0 -> 8,8              ..........
5,5 -> 8,2              222111....
```

2 Avoid Opaque Clouds

For now, consider only those thermal vents that are straight lines, either horizontal or vertical. That is, where

$$x_0 = x_1 \quad \text{or} \quad y_0 = y_1 \tag{1}$$

Considering only horizontal and vertical lines, **at how many points at least two lines overlap?**

2.1 Geometry definitions

Drawing two-dimensional lines requires a coordinate system composed by points and segments. In this case, the definition of the `Coordinate_Range` is limited to the interval $[0, 999]$. `Point_Type` and `Segments_Type` are defined in this system, as well as the two-dimensional surface `Ocean_Floor_Type`:

```
--  __Geometry_Definitions__
subtype Coordinate_Range is Natural range 0 .. 999;

type Point_Type is record
   X : Coordinate_Range;
   Y : Coordinate_Range;
end record;

type Segment_Type is record
   Start_Point : Point_Type;
   End_Point   : Point_Type;
end record;

type Ocean_Floor_Type is
  array (Coordinate_Range, Coordinate_Range) of Natural;
```

Listing 2.1: Geometry Definitions

2.2 Counting points

To count how many points where at least two lines overlap, we need the `Ocean_Floor` to draw the lines in it and `Σ_Overlap` to count such points.

```
--  __Ocean_Floor_And_Overlap_Sum__
Ocean_Floor : Ocean_Floor_Type := (others => (others => 0));
Σ_Overlap   : Natural          := 0;
```

Listing 2.2: Ocean Floor And Overlap Sum

2.3 Straight lines

For now we have only to consider straight lines, those that satisfy equation (1) of section 1.1.

```
--  __Straight_Lines_Definition__
function Is_Straight (S : Segment_Type) return Boolean is
  ((S.Start_Point.X = S.End_Point.X) or else (S.Start_Point.Y = S.End_Point.Y));
```

Listing 2.3: Straight Lines Definition

2.4 Drawing Segments

The procedure to draw a `Vent` in the `Ocean_Floor` consist in:

- start at `Point` indicated by `Vent.Start_Point`
- *draw the point*, which mean to increment by one the `Ocean_Floor` at `Point` coordinates
- increment `Point` coordinates `X` and `Y` by Δx and Δy, respectively
- finish when `Point` reaches `Vent.End_Point`

Δ values depend on the `Vent.Start_Point` and `Vent.End_Point`. Both deltas can have values `-1`, `0` or `1`. Starting at `Vent.Start_Point` and applying their respective increment, it is guaranteed that the `Vent.End_Point` is eventually reached.

Also, once the `Ocean_Floor` is incremented by `1` at `Point` coordinates, when the resulting value is `2` the `Σ_Overlap` will be incremented. Consecutive increments at the same point must not be counted because only points where *at least two lines overlap* must be counted.

```
--  __Drawing_Segments_Definition__
procedure Draw_Segment (Vent : Segment_Type) is
   Point : Point_Type := Vent.Start_Point;

   -- compute deltas
   Δx : Integer :=
     (if Vent.Start_Point.X = Vent.End_Point.X then 0
      elsif Vent.Start_Point.X < Vent.End_Point.X then 1 else -1);

   Δy : Integer :=
     (if Vent.Start_Point.Y = Vent.End_Point.Y then 0
      elsif Vent.Start_Point.Y < Vent.End_Point.Y then 1 else -1);
begin
   loop
      Ocean_Floor (Point.X, Point.Y) := @ + 1;

      if Ocean_Floor (Point.X, Point.Y) = 2 then
```

```
            Σ_Overlap := @ + 1;
         end if;

         exit when Point = Vent.End_Point;

         Point := (X => @.X + Δx, Y => @.Y + Δy);
      end loop;
   end Draw_Segment;
```

Listing 2.4: Drawing Segments Definition

2.5 Reading and drawing hydrothermal vents

Some variables are needed to read the vent segments from the input file:

```
--  __Variables_To_Read_The_Input_File__
Input : File_Type;
Vent  : Segment_Type;
Char  : Character;
Sep   : String (1 .. 4); -- separator ' -> '
```

Listing 2.5: Variables To Read The Input File

Reading a vent segment is quite straightforward:

```
--  __Read_Vent_Segment_From_Input_File__
Get (Input, Vent.Start_Point.X); Get (Input, Char); -- skip ','
Get (Input, Vent.Start_Point.Y);

Get (Input, Sep); -- skip ' -> '

Get (Input, Vent.End_Point.X); Get (Input, Char); -- skip ','
Get (Input, Vent.End_Point.Y);
```

Listing 2.6: Read Vent Segment From Input File

This will be iterated for all the vent segments in the file.

For now, we only focus on straight vents. Because no more information is required, these vent segments can be drawn just after they have been read:

```
--  __Draw_Some_Vents__
if Is_Straight (Vent) then
   Draw_Segment (Vent);
end if;
```

Listing 2.7: Draw Some Vents

2.6 Compilation Unit

```
with Ada.Strings;         use Ada.Strings;
with Ada.Text_IO;         use Ada.Text_IO;
with Ada.Integer_Text_IO; use Ada.Integer_Text_IO;

procedure Day05_P1 is
  <<__Geometry_Definitions__>>
  <<__Ocean_Floor_And_Overlap_Sum__>>
  <<__Straight_Lines_Definition__>>
  <<__Drawing_Segments_Definition__>>
begin
  declare
    <<__Variables_To_Read_The_Input_File__>>
  begin
    Open (Input, In_File, "__Current_Path__" & "input.txt");
    loop
      <<__Read_Vent_Segment_From_Input_File__>>
      <<__Draw_Some_Vents__>>

      exit when End_Of_File (Input);
    end loop;
    Close (Input);
    <<__Export_Data_For_Plotting__>>
  end;

  Put_Line ("Answer:" & Σ_Overlap'Image);
end Day05_P1;
```

Evaluation

```
Answer: 6841
```

3 Avoid More Clouds

Unfortunately, considering only horizontal and vertical lines doesn't give you the full picture; you need to also consider **diagonal lines**.

Because of the limits of the hydrothermal vent mapping system, the lines in your list will only ever be horizontal, vertical, or a diagonal line at exactly 45 degrees. In other words:

- An entry like `1,1 -> 3,3` covers points `1,1`, `2,2`, and `3,3`.
- An entry like `9,7 -> 7,9` covers points `9,7`, `8,8`, and `7,9`.

Considering all lines from the above example would now produce the following diagram:

```
1.1....11.
.111...2..
..2.1.111.
...1.2.2..
.112313211
...1.2....
..1...1...
.1.....1..
1.......1.
222111....
```

You still need to determine **the number of points where at least two lines overlap**. In the above example, this is still anywhere in the diagram with a `2` or larger - now a total of `12` points.

Consider all of the lines. **At how many points do at least two lines overlap?**

3.1 Now drawing at 45 degrees

New considerations

We need a function to detect vent segments at 45 degrees. A vent V, with starting point (x_0, y_0) and end point (x_1, y_1), is a diagonal vent if

$$|x_0 - x_1| = |y_0 - y_1| \tag{2}$$

The function `Is_Diagonal` implements this checking:

```
--  __Diagonal_Lines_Definition__
function Is_Diagonal (Vent : Segment_Type) return Boolean is
  (abs(Integer(Vent.Start_Point.X - Vent.End_Point.X)) =
     abs(Integer(Vent.Start_Point.Y - Vent.End_Point.Y)));
```

Listing 3.1: Diagonal Lines Definition

And drawing more vents:

```
--  __Draw_More_Vents__
if Is_Straight (Vent) or else Is_Diagonal (Vent) then
   Draw_Segment (Vent);
end if;
```

Listing 3.2: Draw More Vents

Compilation unit

```
-- Advent of Code 2021 - Day 5
with Ada.Strings;          use Ada.Strings;
with Ada.Text_IO;          use Ada.Text_IO;
with Ada.Integer_Text_IO;  use Ada.Integer_Text_IO;

procedure Day05_P2 is
  <<__Geometry_Definitions__>>
  <<__Ocean_Floor_And_Overlap_Sum__>>
  <<__Straight_Lines_Definition__>>
  <<__Diagonal_Lines_Definition__>>
  <<__Drawing_Segments_Definition__>>
begin
  declare
    <<__Variables_To_Read_The_Input_File__>>
  begin
    Open (Input, In_File, "__Current_Path__" & "input.txt");
    loop
      <<__Read_Vent_Segment_From_Input_File__>>
      <<__Draw_More_Vents__>>

      exit when End_Of_File (Input);
    end loop;
    Close (Input);
  end;
  <<__Export_Data_For_Plotting__>>

  Put_Line ("Answer:" & Σ_Overlap'Image);
end Day05_P2;
```

Evaluation

```
Answer: 19258
```

4 Graphics

4.1 Generating plots

Writing drawn thermal vents

Once the thermal vents have been drawn in the ocean floor, a file can be written to lately create the graph:

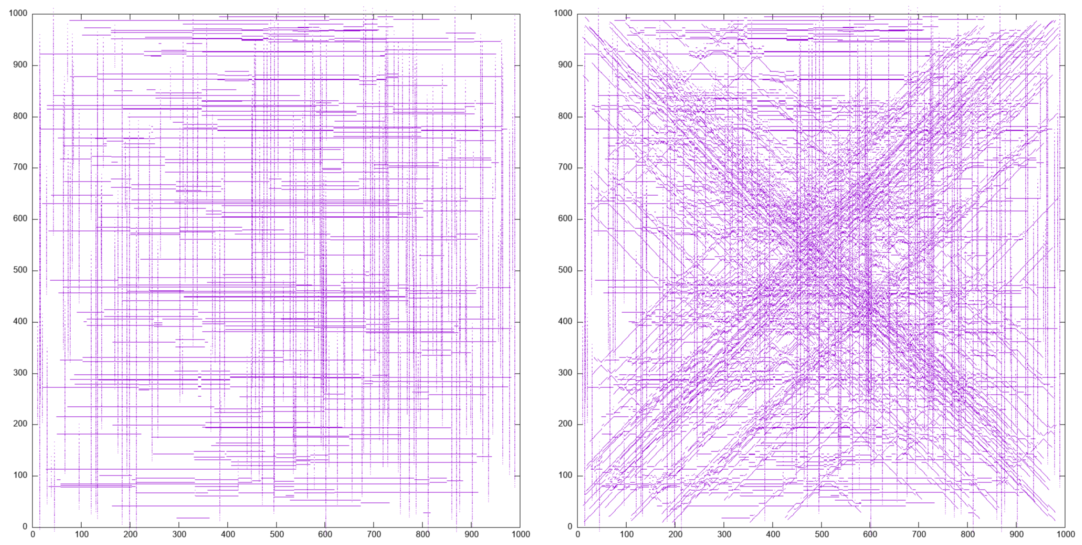
```ada
--  __Export_Data_For_Plotting__
declare
   Output : File_Type;
begin
   Create (Output, Name => "__Current_Path__" & "__Output_Name__");
   for Row in Ocean_Floor'Range (1) loop
      for Col in Ocean_Floor'Range (2) loop
         if Ocean_Floor (Row, Col) > 0 then
            Put_Line (Output, Row'Image & Col'Image &
                        Ocean_Floor (Row, Col)'Image);
         end if;
      end loop;
   end loop;
   Close (Output);
end;
```

Listing 4.1: Export Data For Plotting

The Graph:

```gnuplot
#  __GNUplot_For_Results__
set jitter vertical spread 0.25
set term png size 1024, 1024
set output "output_p1.png"
plot "output_p1.txt" with points pointtype 0 title ""
set output "output_p2.png"
plot "output_p2.txt" with points pointtype 0 title ""
```

Listing 4.2: GNUplot For Results



(a) Horizontal and vertical thermal vents **(b)** Horizontal, vertical and diagonal thermal vents

Figure 4.1: Thermal vents

5 References

- Repository rocher/adventofcode contains
  - original source file README.org to generate:
    - documentation file: README.pdf (this file)
    - tangled source code files day05_p1.adb and day05_p2.adb
  - gpr project file: day05.gpr
  - input file: input.txt
- Developed with Ada/SPARK support for Emacs org-babel
- Images generated with Stable Diffusion with entries from the text

Clone the referred repository and evaluate `README.org` file or compile source code with `gprbuild`.