Trial by Circuits

You sit at your desk, staring at the screen, trying to figure out how to set up the development environment for the code base. You have just been hired by *HardWired*, a medium sized circuitboard manufacturer, as a junior developer, and the documentation of their system isn't great. As you peer through the collection of readme files for some build instructions, you are interrupted by a knock on the door.

Without waiting for an answer, Roland enters your office. You know he is one of the managers with the R&D department, although you are unsure what his position is there exactly. He asks how you are settling in, but as soon as the word "alright" leaves you mouth, he starts to rave about the new *X42* ASIC miner the boys at R&D cooked up, and how they are going to break the whole crypto market wide open. Once the company gets these boards built, they should be able to do terahashes per second, and you gather this will be a very profitable thing.

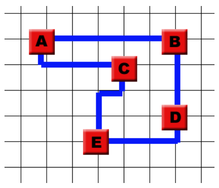
"The only problem is", Roland says "is we can't get the damn things wired up right. Now you studied AI right? So I figured you could make a *smart* program to solve it." You hold back a sigh, consider explaining the concept of *training examples* to him, but think better of it. He hands you a USB drive, you plug it in and see it contains a collection of poorly formatted Excel files. "All the information you need for the wiring should be on there" he says. "I'll need your solution by **Friday 23rd 23:59**, so the boards can get printed and shipped by Monday." And with those words, he walks back out the door.

You skim through the Excel files and gather there are multiple *gates* on a circuitboard, that need the be connected together according to a specified *netlist*. You start to doodle out the problem and try to compute the number of possible paths all the wires can take, but before you come close the the actual number, it makes your head hurt. You get the distinct impression Roland did not understand the *complexity* of the problem he put in front of you. Also, there are still some parts of the problem that are unclear to you. You decide to head down to the basement level, where the boards are actually printed.

You see Bob standing there at the CNC machine, punching in some cutting program. He greets you with a cheery "Hullo!". You met Bob at lunch the other day, after which he was eager to show off all his toys down in the basement. After a bit of chit-chat, you mention the problem Roland handed you. Bob chuckles, and says "Well, you've been given quite the trail by circuits for your first assignment." You have a bit of a back and forth with Bob about the problem, and in the end you agree on the following properties:

- Gates are located on a 2-dimensional grid on the circuit board, indicated with integer coordinates, with their origin at *0, 0*.
- Gates have 5 possible connection points for wiring, the 4 neighbouring grid points on the gate layer, and 1 connection directly above it.
- For each board-netlist combination all gates must be connected according to the netlist specification for a solution to be valid.
- Wires can only move between the integer coordinates in a straight line using the Manhattan distance, i.e. no diagonal connections.
- Wires may not cross the same points or traverse the same lines. Any conflicts in the wiring will render your solution invalid.
- Wires cannot cross gates, but may cross the connection points of gate (i.e. run directly past or over a gate), without creating an invalid solution.
- You can also add additional (empty) board layers *on top* of the gate layer, extending the problem into 3-dimensions, up to a maximum of 7 layers total, including the gate layer.
- Connections between the layers can only be made directly up or down, requiring the same 1 unit of wiring that is needed to connect between 2 neighbouring points on the 2D gird.
- The quality of your solution will solely be determined by the total length of wiring used. The less copper the signals have to travel, the faster the circuit will be.

In addition, you and Bob agree on a format in which you will deliver your solutions for the wiring problems. He grabs a bit of graph paper and makes a quick sketch:



You should mark each component of your solution as follows

- Each element on the grid should be separated by a 1 space in the x direction, and 1 newline in the y direction.
- All gates should be marked with *GA*.
- All open spaces should be marked with 2 underscores, i.e. __
- All line segments belonging to same line should be marked with the same number, with the first line starting at *01*.
- Each layer should start with *### Layer X ###*, with *X* being the layer number, starting at the bottom layer and moving to the top.

So a valid notation for the sketch would be:

```
### Layer 1 ###
__ __ __ __ __ __ __ __
__ GA 01 01 01 01 GA __
__ 02 02 02 GA __ 03 __
__ __ __ 04 04 __ 03 __
__ __ __ 04 __ __ GA __
__ __ __ GA 05 05 05 __
__ __ __ __ __ __ __ __
```

You should hand him your solutions for each board-list combination in a separate file, formatted as *boardN_listM.txt*, with *N* and *M* being the number of the circuit boards and netlists respectively.

"If you can get me your solutions in this format, I can feed them straight into old Betchy here" Bob says, as he lovingly pats what you vaguely remember to be the circuit board etching machine. You thank Bob for all this help with the problem and head back upstairs to talk with Sarah, the lead developer for your team, to discuss the solution already brewing in the back of your head.

You walk up to Sarah's office to find her staring intently at her screens. You gently knock on the open doorframe and without looking up, Sarah responds "Just give a couple of minutes." You stand there and try to get a grasp on the code she is working on, but your mind quickly drifts back to your own circuit routing problem. Just as you begin to contemplate what a strange name Edsger actually is, Sarah speaks up. "There, fixed it." she says with a smile "That was bugging me for two hours. So, what's up?"

You explain the problem you are going to be working on and she says "Ok, I guess we get to see what you are made of right away. I think it would be good idea if we did a code review afterwards, to see how you approached the problem. Why don't we pretend you are back at university" she says with a smirk "and I will give you marks out of 10:"

- 1 point for each of the 6 configurations you manage to correctly solve and get to Bob for printing.
- 2 points for the design of your code, meaning the decomposition of your code into logical functions and classes where needed, your choice of datastructures and the efficiency of the code you wrote.
- 2 points for the style of your code, meaning how well your code conforms to the PEP8 standard, and how well you document all your design choices with clear and concise comments.

"I know you're eager to get started" she says "but you only have 2 weeks, so we should also at least outline your timeframe a little. I would recommend for you to:

Spend the first week considering the representation of the grid for your problem, which could be built-in Python datastructures like *lists* and *dictionaries*, your own designed objects using *classes* or even use an external matrix library like *Numpy*. Solve the basic routing problem from gate to gate, without the wires crossing, on this grid representation using a shortest path algorithm of your choice.

For the second week you should improve on your solution using more advanced non-local search methods like simulated annealing to change the order in which you place the different connections on the grid. You could also try other strategies of your own design to solve as many of the configurations as possible. Remember to leave at least a day or two to actually let your program run and solve the boards.

In the end, you should hand in the following in a *.tar.gz* archive

- All the Python files of your program, including any tests you might have written to verify the correctness of parts of the program, and the data-files needed to run the different configurations.
- The solution files for all the configurations you managed to solve in the format Bob specified.
- A *readme.txt* in which you list any external libraries that might be required for your program to run, instructions for how to run your program for the different configurations, and a basic overview of what parts of the program are contained in which file, so the different parts of your solution are easy to find.

You walk of Sarah's office in a slight daze and head back to your own. You sit down in your chair, server lights blinking around you in red and green. You open up your favorite editor and trusty terminal, and begin to write.