Lab 8 - Advent of code 2022 16

Weichen Chai

Spring 2023

Introduction

This paper seeks to answer the questions presented in"Advent of code 1" for ID1019. In this assignment, we will explore how the functional language Elixir can be utilized to process quizzes and tasks given by our teachers on a website named "adventofcode day 16".

Tasks 1

In this task, we are trying to open pipes for maximum output of flow of water using data given on the website. We only have 30 minutes and we want to traverse the pipes, and each time we move to the next valve will take 1 minute and it will take another minute to open it.

Looking at the skeleton code given for this assignment, we use the parse function which returns the input statement into something we can work with in a tuple, which contains the pipe name, amount of water pressure in the valve, and which valves the pipe heads towards. We use the following parsing function to trim and remove unnecessary information that hinders our process.

```
def parse(input) do
   Enum.map(input, fn(row) ->
     [valve, rate, valves] = String.split(String.trim(row), ["=", ";"])
     [_Valve, valve | _has_flow_rate ] = String.split(valve, [" "])
     valve = String.to_atom(valve)
     {rate,_} = Integer.parse(rate)
     [_, _tunnels,_lead,_to,_valves| valves] = String.split(valves, [" "])
     valves = Enum.map(valves, fn(valve) -> String.to_atom(String.trim(valve,","))
     {valve, {rate, valves}}
   end)
 end
```

What we want to begin with is to isolate the valves out and determine which valves there are, this can be done with a simple Enum.map function, this can be used to determine the closed valves.

```
close = Enum.map(map, fn({valve, _}) -> valve end)
```

We know that the valve we begin with reading external files to the program. This can be done with the file.read function, which returns the result of things inside the file. We also start at :AA.

```
    start = :AA
    //rows = File.read!("day16.txt")
    rows = sample()
```

Furthermore, we can try to implement a traversal function, which returns the highest flow possible from the data given to us. This can be done by including the valve we are at, the time left, the opened and closed pipes which are used to know which valves can be opened or not during traversal. Other than that, an important concept is dynamic programming in this case, as brute force will not perform well and have an acceptable run time. The method we can use here is to keep the memory of the paths that we have searched through in the past. This suggests an two more inputs in the function, namely, memory and paths, apart from the already apparent inputs shown below:

```
case Dmemory[{valve, time, open}] do
  nil ->
    {maxrate, memory, path}
    = find(valve, time, close, open, rate, map, mem, path)
    // as shown above, this is the format for the find function, with 8 inputs
    //indicates that when no previous found we save a new one.
    memory = Map.put(memory, {valve, time, open}, {maxrate, path})
    {maxrate, memory, path}
...
// when we find it, return it.
```

When we encounter the same pipe, we wouldn't have to rerun therefore wasting less time when executing.

Of course, other than the case above, we also have some other cases to keep in mind of, such as the following:

```
def Dmemory(_, 0, _, _, _, _, _,path) do
  //this is when time reaches 0.
```

```
  {0, mem, path}
 end

def Dmemory(valve, time, [], open, rate, _map, mem, path) do
  //this is when all valves have been opened
    total = time *rate
.
    {total, mem, path}
 end
```

To decide what the max flow rate is, we should use a search function, which includes

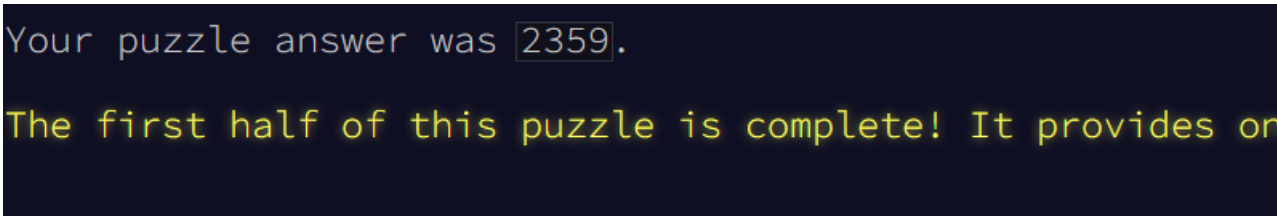
```
def search(valve, time, close, open, rate, map, mem, path) do
...
    if(Enum.member?(close, valve)) do
    // open the valve, by doing so we delete the valve from
    //closed state
    rat  = Dmemory(valve, time - 1, List.delete(close, valve), [valve | open],
...
    else
    {time* rate, mem, path}
    // cant open valve
    end
...
```

The search function should call Dmemory recursively, where it checks if the max water flow rate found by moving to next valve is higher than current flow rate, where it updates the max rate and path.

Using dynamic programming, we can compute previously impossible to compute values, when I utilized a previous version of the code which searches for the answer in a linear fashion, which was only able to run results up to time of 15, anything bigger than that would either take a lengthy amount of time, or simply crash the terminal. With this new version of the code, it is possible to go with much higher time values, such as the required 30 minutes from the task, which was impossible to run before with the function which has a horrible complexity.

In this part, after we are done implementing the program, we can run it and receive the result to put on the website, namely,

```
{2359, [:PH, :AW, :LX, :IN, :OW, :QR, :SV, :HH, :HX]}
```



The screenshot above is one that shows the answer is correct(the teacher said it was ok to have a screenshot)

Tasks 2

We were not asked to complete task 2 of the assignment with a program, but we were asked to discuss it. This task proposes that we can get an elephant to open the valves along with us, but in the 30 minutes, we have to spend 4 minutes teaching the elephant to open the valves in the right order, leaving 26 minutes to execute traversal. The question is if it will be better for two traversals to occur at the same time even if we have 4 minutes less. A method we can use to complete this task will still lie in dynamic programming, where we can have one traversing the pipes first, leaving valves open in the first run which is 24 minutes. Then the second traverses through the pipes again with the valves still open from the first traversal. This way, we can effectively open as many valves as possible.