Advent of Code Day 16

Tim Linsel

Spring Term 2023

Introduction

In this assignment I will solve the advent of code Day 16. I will use depth first search but with a simplified graph.

Graph

Since there are many valves that release 0 pressure, I want to remove them from the graph. At first I thought to just remove them and have weighted edges but it is smarter to just get the shortest path between all valves that release pressure. In order to do that I implemented a version of Dijkstra's algorithm. Where I can just use a normal list as priority queue because all edges have the same weight.

Dijkstra's

In this algorithm, I have a graph, a queue and a visited list. I take the first element in the queue, put it in the visited list and add all the neighbors to the queue if they are not already in there and are not in the visited list. Then I call the function recursively until there are no more elements in the queue

```
def dijkstras(_, [], visited) do visited end
  def dijkstras(graph, [{elem, dis} | queue], visited) do
    newVisited = List.flatten(visited, [{elem, dis}])
    {:ok, map} = Map.fetch(graph, elem)
    {:ok, neighbors} = Map.fetch(map, :nbrs)
    {q, v} = List.foldl(neighbors, {queue, newVisited}, fn e, {q, v} ->
      {:ok, graph} = Map.fetch(graph, e)
      {:ok, rate} = Map.fetch(map, :rate)
      if member?(v, e) or member?(q, e)do
          {q,v}
      else
        {List.flatten(q, [{e, dis + 1}]), v}
      end
    end
    )
   dijkstras(graph, q, v)
  end
```

I now have the shortest distance from any node to every node, however, I am only interested in the nodes that release more pressure than 0. Therefore I remove the nodes with no pressure release from the neighbors list in my graph.

```
  def beautify(graph) do
    List.foldl(Map.keys(graph), Map.new(), fn elem, acc ->
      {neigh, rate} = graph[elem]
      Map.put(acc, elem, {remove0s(neigh, Map.keys(graph)), rate})
    end)
  end
```

One key value pair in my graph looks now like this:

```
"AA" => {[
     {"TR", 8},
     {"YA", 8},
     {"ES", 7},
     {"YL", 7},
     {"WZ", 6},
     {"JT", 5},
     {"AT", 5},
     {"GQ", 5},
     {"AZ", 5},
     {"RK", 4},
     {"OC", 3},
     {"EY", 3},
     {"XN", 3},
     {"VK", 3},
     {"UI", 2},
     {"AA", 0}
   ], 0}
```

search

I use depth first search as search algorithm. For all the valves that are not yet closed, I look recursively how much pressure I will release when I go to a particular valve and take the path where I release most pressure. So I look at all the possible combinations and the one that releases most pressure.

```
def search(_, _, min, press,_) when min <= 0 do press end
  def search(_, _, _, press, []) do press end
  def search(graph, valve, min, press, closed) do
    {:ok, {neighbors, _}} = Map.fetch(graph, valve)
    List.foldl(neighbors, 0, fn {neigh, dis}, acc ->
      if Map.has_key?(graph, neigh) and Enum.member?(closed, neigh) do
        removed = List.delete(closed, neigh)
        {_, rate} = graph[neigh]
        minutes = min - 1 - dis;
        pressure = search(graph, neigh, minutes, press + minutes * rate, removed )
        if pressure > acc do
          pressure
        else
          acc
        end
      else
        acc
      end
    end
      )
  end
```

Part B

For part B I have a wrapper around the depth first search algorithm where I keep track 3 clocks. One clock to keep track when I am free again, one clock to keep track when the elephant is free again and one clock that ticks down one minute at a time. When the third clock is equal to one of the other clock, that player is free to move again, then I use the almost the same algorithm as before to let the player.

```
    def player2(_graph, _valveH, _valveE, _minH, _minE, press, _closed, min) when mi
    def player2(_graph, _valveH, _valveE, _minH, _minE, press,[], _min) do press en
    def player2(graph, valveH, valveE, minH, minE, press,closed, min) do
    cond do
      minH == min -> player2_1NB(graph, valveH, valveE, minH, minE, press, closed,
      minE == min -> player2_1NB(graph, valveH, valveE, minH, minE, press, closed,
      true -> player2(graph, valveH, valveE, minH, minE, press, closed, min - 1)
    end
  end
```

If I just use this and the same algorithm from part A it takes very long to execute. Therefore I do a little heuristic. If a node is both further away and releases less pressure other nodes, I don't go down that particular path.

simplify

So I use `simplify/3` to simplify the path choices. Given all the neighbors (neighbor, distance), a list of all still closed valves and the graph, it returns a list of nodes where all the nodes that are have a greater distance also release more pressure.

```
    def simplify(list, closed, graph) do
    List.foldl(list, [], fn {elem, dis}=e1, acc ->
      if Enum.member?(closed, elem) do
        {list, add} = List.foldl(acc, {[], false}, fn {e, d} =e2, {ac, _} ->
          {_, r1} = graph[elem]
          {_, r2} = graph[e]
          cond do
            d < dis and r2 > r1 -> {[e2|ac], false}
            d > dis and r2 < r1 -> {ac, true}
            true -> {[e2|ac], true}
          end
        end)
          if add or list == [] do
            [e1 |list]
          else
            list
          end
      else
        acc
      end
    end)
  end
```

I use this function in my depth first search algorithm.

```
  def player2_1NB(graph, valveH, valveE, minH, minE, press, closed, min, person) do
    {valve1, valve2, min1, min2} = if person do
        {valveH, valveE, minH, minE}
    else
        {valveE, valveH, minE, minH}
    end
    {:ok, {neighbors, _}} = Map.fetch(graph, valve1)
    neighbors = simplify(neighbors, closed, graph)
    List.foldl(neighbors, 0, fn {neigh, dis}, acc ->
      if Map.has_key?(graph, neigh) and Enum.member?(closed, neigh) do
        removed = List.delete(closed, neigh)
        {_, rate} = graph[neigh]
        minutes = min1 - 1 - dis;
        pressure = player2(graph, neigh,valve2, minutes,min2,
        minutes * rate, removed, min)
        if pressure > acc do
          pressure
        else
          acc
        end
      else
        acc
      end
    end
      ) + press
  end
```

And voila 2 stars for the Advent of Code Day 16 with 10 sec execution time. I added all the code on GitHub: https://github.com/sonnenpelzx/AdventOfCode2022Day16