Dynamic Programming and the Periodic Table

Krish Shah

March 18, 2022

Contents

§1 Motivation and Description for the Project

I first conceived the idea for this project in the midst of chemistry class seemingly out of random - we were going our unit on the periodic table and we discussed shorthand notation. Shorthand notation is basically the crux of why the idea behind this project works. In case the reader is not familiar with shorthand notation, let me give a quick explanation of what shorthand notation (or atleast try to).

Shorthand notation is a way to write the electronic configuration of an element using the previous noble gas as a starting point and then essentially just adding in the extra electrons for the current element. I hope that's a satisfactory explanation but just in case it's not clear, here's a video [1] to clear things up a bit.

Its use in chemistry is mostly one of simplification - it helps express the electronic configuration of an element with relative ease compared to writing out the entire electronic configuration. For the purposes of this project, however, shorthand notation is basically the premise which makes dynamic programming relevant in this context. Given that the electronic configuration for an element can be determined using the previous noble gas, essentially with the electronic configuration for each element being a **state**, the noble gas configurations all serve as ways for **transitions**[1] - **states** and **transitions** are both terms associated with dynamic programming.

Using this as well as the **Aufbau principle** to dictate the order in which shells were filled, I was able to successfully determine the electronic configurations for all elements using their respective previous noble gas as part of my transitions[2].

§2 Skills Related to this Project

- classes, objects and nested class structures
- JSON parsing and manipulation
- mult-file integration
- dynamic programming

§3 Challenges Faced

Initially, when I started this project, it began as something that I wanted to implement in C++. I thought C++ would be the best language to take this through with since it naturally made sense to me choose C++ because of my previous experience with DSA in C++. However, as became apparent in later stages of the coding segment of this project, there were certain language-based disadvantages that I couldn't find ways around with in C++. I'll

[1] This does not account for all anomalies that exist - there are certainly *many* elemental exceptions.
[2] Note that this does not account for elements that are exceptions to the Aufbau principle.

give you some places I got stuck to show why I eventually decided to switch the language I coded this project in to Python.

1. JSON parsing is not natively available in C++. For that reason, I had to use an external library (JSONCpp [2]) and had to interface that with the rest of my files. This, whilst possible, took up a lot of space in the project directory as I manually included these files in my project folder[3] as I didn't really want to go through the hassle of installing a C++ package on my system that I would likely only be using for one project. This proved to be a problem when it came to storing elements as I started getting MLE errors. I tried to remedy this by eliminating many different data files that I wasn't directly using, but nonetheless, the error persisted. Eventually, this motivated me to try to switch to a language in which JSON parsing capabilities would be easier to achieve (and be less space-consuming). Among those, Python stood out to me because of its versatility and so I felt Python was a good candidate to switch over to,
2. I wanted this project to be easily accessible and executable so it could remain accessible to as many people as possible. For those reasons, as Python is more accessible than C++ (both in terms of its popularity and executing programs), I felt Python was a better choice.

I switched over to Python based on the reasons I gave above. None of the logic itself really changed (as expected) and it was just a syntactical conversion from C++ to Python.

§4 Try It Out Yourself!

After cloning the repository, you must first enter the folder containing the cloned repository. If an IDE is being used, this can be done through the IDE itself, however, if not, then the following commands can be used to run the program through the terminal:

for Linux and Mac:

```
cd Dynamic-Programming-and-the-Periodic-Table
pip3 install -r requirements.txt
python3 main.py
```

for Windows:

```
cd Dynamic-Programming-and-the-Periodic-Table
pip install -r requirements.txt
python main.py
```

The program should then execute! Cheers!

§5 Parting Shots

Thank you for taking the time to look at my project and I hope you've enjoyed reading about it just as much as I did creating it!

Thank you very much for your interest in my project and my work. You can find the latest of what I'm working on at my GitHub profile!

If you have any concerns or criticisms regarding this project or any others, please feel free to reach out to me at krishlandia@gmail.com!

§6 Attributions

§6.1 People That Made This Project Possible

- Ms. Pandey for teaching me chemistry and imparting the knowledge to me that made this project possible today
- Mr. Kim for inspiring me to take on new challenges
- the JSONCpp dev team, whose work made it possible for me to use JSON capabilities in C++

§6.2 Citations

[1] *A Level Chemistry Revision "Shorthand Electron Configuration"*. YouTube, Apr. 2020. [Online]. Available: https://www.youtube.com/watch?v=5mP0z1MAdCk.

[2] J. D. Team, *Jsoncpp*, https://github.com/open-source-parsers/jsoncpp, 2022.

[3] There were probably better ways to accomplish the same task in hindsight, but nonetheless, it would have been quite arduous either way.