Short Term Scholar Weekly Progress Report 4

Ufuk Bombar

August 5, 2019

This week, the goal was to finish the remaining chapters of the book "Geoprocessing with Python" and completing the implementation of router application. Last week I finished chapter 8 and I thought the entire book. However, I did not realize, the book contains 13 chapters. I did not see the remaining chapters because the soft copy of the book was not complete.

On Monday, I did not realize this blunder and continued on my router project. At that time, the routing was handled by Mapper Graph. However, the Mapper Graph was using BFS (Breath first search) algorithm to find the shortest path. In the small scale, it was a considerable approach because BFS algorithm requires O(V + E) and easier to understand. This complexity means that it will traverse through all the edges and vertices once. The complexity of DSP (Dijkstra's Shortest Path) algorithm, on the other hand, is O(E * log(V)) since it is implemented using a binary heap. But, it is harder to understand because the procedure is much more complex. The reason why I decided to use DSP instead of BFS this is because the BFS algorithm focuses on the node distances. In other words, BFS finds the shortest path regarding the minimum number of edges between the nodes. The DSP algorithm considers the weights of edges for finding the shortest path. Thiss is why I chose the DSP algorithm for my router application. Additionally, I learned about other pathfinding algorithms for different purposes. For example; A* algorithm also used commonly on video games [2]. The enemies found the players by this algorithm. I did not choose this algorithm because it is more complex than the DSP algorithm.

On Tuesday, I started the implementation for DSP algorithm. I saw that there is more than one way to do it. The DSP uses an ordered data-structure in my implementation. It is required for keeping track of the nearest neighbor nodes. I used a min-heap for that purpose. Because min-heap provides an insertion complexity O(log(n)) where n denotes the number of elements. In Python, no collection acts like min-heap. But a module exists, called "heapq". This module converts an array to a min-heap. The functions for insertion, deletion, and heapification are also provided. I used this module to implement the DSP algorithm for my graph. The code can be found on 1.

On Wednesday, I had a briefing with Mr. Ramalingam and Mr. Altınakar. We discussed the incomplete soft copy of the book, and I showed my work. After the briefing, I have given a hard copy of the book which is complete. And cogitated on different possible directions that this project can go. I realized that the one way it will benefit me is to have a central controller and mobile agents in a city. The agents can send information about the streets and the central controller can update the database and decide which agent to take which route to reach which destination. Therefore, I started a small server-side application. I knew that Python is more than capable of this. I could have used Flask or Django, but I decided to use Go instead. The reason was, Go is much more efficient language. One reason for this is because it is a statically typed language which means it is required to compile go programs to run[3]. This is an advantage over Python because the server-side application for this purpose must be fast. Therefore, the rest of my time I work on a Go-based RestAPI which only responses with an empty JSON to requests.

On Thursday, I studied the chapter 9. I learned about reading and writing the raster files. I also learned GDAL provides these utility functions directly unlike vector operations. Vectors are read or written by OGR. I also learned how a raster file is represented. GDAL reads the raster file to the memory as a NumPy array. NumPy is a tensor library that provides high-performance functions[4]. But the raster files are, most of the time, too large to store in the memory. Therefore, most of the raster files read to the memory as chunks. This chunks can be processed separately and improve the overall performance by using parallel processing. Additionally, the geological information of a raster file can also be altered. This is called "geotransformation" [1].

On Friday, I had a briefing with Mr. Altınakar to discussed the mistakes in old reports. We examined the first 3 of my reports deeply and I realized the mistakes I made. After that, I continue studying chapter 9.

```
def doDijkstra(self, cid1, cid2):
if not (self.crossExists(cid1) and self.crossExists(cid2)):
    raise Exception('Road does not exist')

if cid1 == cid2:
    return [cid1]

visited = set()
distances = collections.defaultdict(lambda: float('inf'))
path = list()

previous = dict()
heap = _SPHeap(key=lambda x: distances[x])
distances[cid1] = 0
heap.push(cid1)
connectionDict = dict()

while len(heap) != 0:
    cur = heap.pop()
    visited.add(cur)

    if cur == cid2:
        break

    for adj, rid in self.getAdjCrosses(cur, getrids=True):
        if adj in visited:
            continue

        ndis = distances[cur] + self.getWeight(cur, adj)

        if ndis < distances[adj]:
            distances[adj] = ndis
            heap.push(adj)
            previous[adj] = cur
            connectionDict[(adj, cur)] = rid

if cid2 in visited:
    p = previous.get(cid2)

    while p != cid1:
        path = [p] + path
        p = previous.get(p)

    return [cid1] + path + [cid2], connectionDict

return None
```

Figure 1: Python code for Dijkstra's Shortest Path Algorithm

References

[1] Chris Garrard. *Geoprocessing with Python*. Manning, 2016. ISBN: 9781617292149.

[2] Amit Patel. *From Amit's Thoughts on Pathfinding: Introduction to A**. URL: `http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html`. (accessed: 05.8.2019).

[3] Go Team. *Documentation*. URL: `https://golang.org/doc/`. (accessed: 05.8.2019).

[4] Stefan van der Walt et al. *The NumPy Array: A Structure for Efficient Numerical Computation*. IEEE, 2011. ISBN: 11851135.