EE205 Project 4 pdf

20180155 김준범

1. Algorithm

- Dijkstra Part.

For Dijkstra algorithm, I used the STL priority queue to get the minimum distance crossroad for every loop. I defined PQ_element which has distance argument and cross as its element. I first calculated all the adjacent crossroad for all cross road for convenience. Then, it goes into a for loop which iterates for each client.

For every client, I calculated the source and destination CID, and created the priority queue. Then, it iterates the while loop until the PQ gets empty or iterates more than n. In the while loop we get the PQ_element of the lowest distance value. For that element, we look at all adjacent crossroad.to compare the value of its original distance value and newly calculated one. If newly calculated value is smaller, we have to update the priority queue value. But, since C++ STL priority queue can't update its value I chose a way of adding a new minimum element and whenever the old element pops out, we discard that. After the while loop, I filled in the path structure and completed the implementation.

-A* Algorithm part

For A* algorithm, I also used the STL priority queue to get the minimum f value of the element. Just as dijkstra part, I started by calculating all adjacent crossroads for every crossroad. Then, it goes into the for loop which iterates for each client.

For every client, I calculated the source and destination CID, and created the priority queue and G_list. The G_list is practically the same idea of minimum distance in dijkstra. Then, we iterate the while loop until the element we got from the queue contains the same cid as the destination cid. In the while loop, we compare the G_list value and the newly calculated value. If the newly calculated g value is smaller, then we create new element and push it into the queue. After all iteration, I filed the path structure and completed the implementation.

- How know the correction.

I randomly generated the maps using the mapgen program. For those randomly generated maps, I searched for the minimum path using Dijkstra part and A* part and created the corresponding output file, a.txt and b.txt respectively. By using the linux command “cmp a.txt b.txt” I noticed that path created by two ways are always the same. Which would not be possible if one of them is wrongly implemented. Also for maps whith low number of crossroad, looked at the map

drawn by python and intuitionally looked compared the result I got from the algorithm.

** Because I couldn't update the value inside the priority queue, I chose a way of adding a new element. But This will increase the element number inside the priority queue and make the program slow. This can be solved if I created a self implemented heap. But Because I lacked time. I couldn't finish it. The Slowness would increase as the number of the crossroad increase.

2. Performance.

- Dijkstra

```
junbum@ubuntu:~/Desktop/pj4/src$ ./mapgen 10 1 > 1.txt
junbum@ubuntu:~/Desktop/pj4/src$ ./mapgen 10 1 > 2.txt
junbum@ubuntu:~/Desktop/pj4/src$ ./mapgen 10 1 > 3.txt
junbum@ubuntu:~/Desktop/pj4/src$ ./mapgen 10 1 > 4.txt
junbum@ubuntu:~/Desktop/pj4/src$ ./mapgen 10 1 > 5.txt
junbum@ubuntu:~/Desktop/pj4/src$ ./mapgen 10 1 > 6.txt
junbum@ubuntu:~/Desktop/pj4/src$ ./mapgen 10 1 > 7.txt
junbum@ubuntu:~/Desktop/pj4/src$ ./mapgen 10 1 > 8.txt
junbum@ubuntu:~/Desktop/pj4/src$ ./mapgen 10 1 > 9.txt
junbum@ubuntu:~/Desktop/pj4/src$ ./mapgen 10 1 > 10.txt
junbum@ubuntu:~/Desktop/pj4/src$ time ./pathfinder dijkstra < 1.txt > a.txt

real    0m0.009s
user    0m0.001s
sys     0m0.005s
junbum@ubuntu:~/Desktop/pj4/src$ time ./pathfinder dijkstra < 2.txt > a.txt

real    0m0.008s
user    0m0.000s
sys     0m0.006s
junbum@ubuntu:~/Desktop/pj4/src$ time ./pathfinder dijkstra < 3.txt > a.txt
```

```
junbum@ubuntu:~/Desktop/pj4/src$ ./mapgen 1000 1 > 1.txt
junbum@ubuntu:~/Desktop/pj4/src$ ./mapgen 1000 1 > 2.txt
junbum@ubuntu:~/Desktop/pj4/src$ ./mapgen 1000 1 > 3.txt
junbum@ubuntu:~/Desktop/pj4/src$ ./mapgen 1000 1 > 4.txt
junbum@ubuntu:~/Desktop/pj4/src$ ./mapgen 1000 1 > 5.txt
junbum@ubuntu:~/Desktop/pj4/src$ time ./pathfinder dijkstra < 1.txt > a.txt

real    0m0.046s
user    0m0.019s
sys     0m0.004s
junbum@ubuntu:~/Desktop/pj4/src$ time ./pathfinder dijkstra < 2.txt > a.txt

real    0m0.038s
user    0m0.016s
sys     0m0.004s
junbum@ubuntu:~/Desktop/pj4/src$ time ./pathfinder dijkstra < 3.txt > a.txt

real    0m0.035s
user    0m0.010s
sys     0m0.007s
```

The picture above is when the client number is fixed as 1. As the crossroad increases 10 to 1000, the calculation time increase about 3,4 times. We can see the logarithm increase of time due to increase in crossroad.

```
junbum@ubuntu:~/Desktop/pj4/src$ ./mapgen 1000 10 > 1.txt
junbum@ubuntu:~/Desktop/pj4/src$ ./mapgen 1000 10 > 2.txt
junbum@ubuntu:~/Desktop/pj4/src$ ./mapgen 1000 10 > 3.txt
junbum@ubuntu:~/Desktop/pj4/src$ ./mapgen 1000 10 > 4.txt
junbum@ubuntu:~/Desktop/pj4/src$ ./mapgen 1000 10 > 5.txt
junbum@ubuntu:~/Desktop/pj4/src$ time ./pathfinder dijkstra < 1.txt > a.txt

real    0m0.067s
user    0m0.038s
sys     0m0.004s
junbum@ubuntu:~/Desktop/pj4/src$ time ./pathfinder dijkstra < 2.txt > a.txt

real    0m0.069s
user    0m0.037s
sys     0m0.003s
junbum@ubuntu:~/Desktop/pj4/src$ time ./pathfinder dijkstra < 3.txt > a.txt

real    0m0.017s
user    0m0.010s
sys     0m0.003s
junbum@ubuntu:~/Desktop/pj4/src$ time ./pathfinder dijkstra < 4.txt > a.txt

real    0m0.065s
user    0m0.040s
sys     0m0.004s
junbum@ubuntu:~/Desktop/pj4/src$ time ./pathfinder dijkstra < 5.txt > a.txt

real    0m0.012s
user    0m0.011s
sys     0m0.000s
```

```
junbum@ubuntu:~/Desktop/pj4/src$ ./mapgen 1000 100 > 1.txt
junbum@ubuntu:~/Desktop/pj4/src$ ./mapgen 1000 100 > 2.txt
junbum@ubuntu:~/Desktop/pj4/src$ ./mapgen 1000 100 > 3.txt
junbum@ubuntu:~/Desktop/pj4/src$ ./mapgen 1000 100 > 4.txt
junbum@ubuntu:~/Desktop/pj4/src$ ./mapgen 1000 100 > 5.txt
junbum@ubuntu:~/Desktop/pj4/src$ time ./pathfinder dijkstra < 1.txt > a.txt

real    0m0.203s
user    0m0.160s
sys     0m0.008s
junbum@ubuntu:~/Desktop/pj4/src$ time ./pathfinder dijkstra < 2.txt > a.txt

real    0m0.131s
user    0m0.100s
sys     0m0.004s
junbum@ubuntu:~/Desktop/pj4/src$ time ./pathfinder dijkstra < 3.txt > a.txt

real    0m0.138s
user    0m0.109s
sys     0m0.000s
junbum@ubuntu:~/Desktop/pj4/src$ time ./pathfinder dijkstra < 4.txt > a.txt

real    0m0.080s
user    0m0.071s
sys     0m0.000s
junbum@ubuntu:~/Desktop/pj4/src$ time ./pathfinder dijkstra < 6.txt > a.txt

real    0m0.011s
user    0m0.002s
sys     0m0.000s
```

The two picture above is when the number of cross road is fixed. When the

number of the client increases of the scale 10, we can see the increase of the time in about 2,3 times.

- A* algorithm

```
junbum@ubuntu:~/Desktop/pj4/src$ ./mapgen 10 1 > 1.txt
junbum@ubuntu:~/Desktop/pj4/src$ ./mapgen 10 1 > 2.txt
junbum@ubuntu:~/Desktop/pj4/src$ ./mapgen 10 1 > 3.txt
junbum@ubuntu:~/Desktop/pj4/src$ ./mapgen 10 1 > 4.txt
junbum@ubuntu:~/Desktop/pj4/src$ ./mapgen 10 1 > 5.txt
junbum@ubuntu:~/Desktop/pj4/src$ time ./pathfinder a-star < 1.txt > a.txt

real    0m0.002s
user    0m0.000s
sys     0m0.001s
junbum@ubuntu:~/Desktop/pj4/src$ time ./pathfinder a-star < 2.txt > a.txt

real    0m0.003s
user    0m0.000s
sys     0m0.002s
junbum@ubuntu:~/Desktop/pj4/src$ time ./pathfinder a-star < 3.txt > a.txt

real    0m0.009s
user    0m0.003s
sys     0m0.003s
junbum@ubuntu:~/Desktop/pj4/src$ time ./pathfinder a-star < 4.txt > a.txt

real    0m0.009s
user    0m0.000s
sys     0m0.006s
```

```
junbum@ubuntu:~/Desktop/pj4/src$ ./mapgen 1000 1 > 1.txt
junbum@ubuntu:~/Desktop/pj4/src$ ./mapgen 1000 1 > 2.txt
junbum@ubuntu:~/Desktop/pj4/src$ ./mapgen 1000 1 > 3.txt
junbum@ubuntu:~/Desktop/pj4/src$ ./mapgen 1000 1 > 4.txt
junbum@ubuntu:~/Desktop/pj4/src$ ./mapgen 1000 1 > 5.txt
junbum@ubuntu:~/Desktop/pj4/src$ time ./pathfinder a-star < 1.txt > a.txt

real    0m0.033s
user    0m0.017s
sys     0m0.000s
junbum@ubuntu:~/Desktop/pj4/src$ time ./pathfinder a-star < 2.txt > a.txt

real    0m0.029s
user    0m0.010s
sys     0m0.006s
junbum@ubuntu:~/Desktop/pj4/src$ time ./pathfinder a-star < 3.txt > a.txt

real    0m0.034s
user    0m0.013s
sys     0m0.004s
junbum@ubuntu:~/Desktop/pj4/src$ time ./pathfinder a-star < 4.txt > a.txt

real    0m0.005s
user    0m0.004s
sys     0m0.000s
junbum@ubuntu:~/Desktop/pj4/src$ time ./pathfinder a-star < 5.txt > a.txt

real    0m0.009s
user    0m0.006s
sys     0m0.000s
junbum@ubuntu:~/Desktop/pj4/src$
```

The picture above is when the client number is fixed as 1. As the crossroad increases 10 to 1000, the calculation time increase about 3,4 times. We can see the logarithm increase of time due to increase in crossroad.

```
junbum@ubuntu:~/Desktop/pj4/src$ ./mapgen 1000 10 > 1.txt
junbum@ubuntu:~/Desktop/pj4/src$ ./mapgen 1000 10 > 2.txt
junbum@ubuntu:~/Desktop/pj4/src$ ./mapgen 1000 10 > 3.txt
junbum@ubuntu:~/Desktop/pj4/src$ ./mapgen 1000 10 > 4.txt
junbum@ubuntu:~/Desktop/pj4/src$ ./mapgen 1000 10 > 5.txt
junbum@ubuntu:~/Desktop/pj4/src$ time ./pathfinder a-star < 1.txt > a.txt

real    0m0.005s
user    0m0.002s
sys     0m0.002s
junbum@ubuntu:~/Desktop/pj4/src$ time ./pathfinder a-star < 2.txt > a.txt

real    0m0.034s
user    0m0.016s
sys     0m0.004s
junbum@ubuntu:~/Desktop/pj4/src$ time ./pathfinder a-star < 3.txt > a.txt

real    0m0.006s
user    0m0.003s
sys     0m0.003s
junbum@ubuntu:~/Desktop/pj4/src$ time ./pathfinder a-star < 4.txt > a.txt

real    0m0.008s
user    0m0.007s
sys     0m0.000s
junbum@ubuntu:~/Desktop/pj4/src$ time ./pathfinder a-star < 5.txt > a.txt

real    0m0.034s
user    0m0.015s
sys     0m0.004s
junbum@ubuntu:~/Desktop/pj4/src$
```

```
junbum@ubuntu:~/Desktop/pj4/src$ ./mapgen 1000 100 > 1.txt
junbum@ubuntu:~/Desktop/pj4/src$ ./mapgen 1000 100 > 2.txt
junbum@ubuntu:~/Desktop/pj4/src$ ./mapgen 1000 100 > 3.txt
junbum@ubuntu:~/Desktop/pj4/src$ ./mapgen 1000 100 > 4.txt
junbum@ubuntu:~/Desktop/pj4/src$ ./mapgen 1000 100 > 5.txt
junbum@ubuntu:~/Desktop/pj4/src$ time ./pathfinder a-star < 1.txt > a.txt

real    0m0.071s
user    0m0.036s
sys     0m0.008s
junbum@ubuntu:~/Desktop/pj4/src$ time ./pathfinder a-star < 2.txt > a.txt

real    0m0.061s
user    0m0.042s
sys     0m0.000s
junbum@ubuntu:~/Desktop/pj4/src$ time ./pathfinder a-star < 3.txt > a.txt

real    0m0.070s
user    0m0.038s
sys     0m0.006s
junbum@ubuntu:~/Desktop/pj4/src$ time ./pathfinder a-star < 4.txt > a.txt

real    0m0.070s
user    0m0.043s
sys     0m0.001s
junbum@ubuntu:~/Desktop/pj4/src$ time ./pathfinder a-star < 5.txt > a.txt

real    0m0.014s
user    0m0.013s
sys     0m0.000s
junbum@ubuntu:~/Desktop/pj4/src$
```

The two picture above is when the number of cross road is fixed. When the

number of the client increases of the scale 10, we can see the increase of the time in about 2,3 times.

3. When is the worst case for the A* algorithm?

A* algorithm is expected to be faster than dijkstra. But When we add crossroad into the closed list, there can be a case where the destination crossroad gets added at the last. In this case, since A* algorithm has additional computation (h(n)) which makes it slower than dijkstra algorithm.

4. List all collaborators who discussed with you.

I didn't collaborate with anyone and only used the book and classum.