**2020-2 Fall Algorithm Analysis – HW3**

2016147014 유승수

1. **Code Review & Algorithm Introduction**
2. Define Data-Structure

There are several algorithms for finding the shortest distance in a graph. Representatively, there are Floyd Warshall algorithm, Dijkstra's algorithm, and Balman's algorithm. Since the given graph is a directed and weighted graph and the longest route out of all routes needs to be found, we tried to use the Floyd Warshall algorithm in which all routes are calculated. However, the time complexity was expected to be enormous, and the memory (space complexity) to store data seemed to be insufficient. So, this task was implemented through the edge of the graph as an adjacency list and solved with the Dijkstra algorithm using priority queues.

First, the graph data structure is expressed in various ways, and it seems that there is a lot of unnecessary data to be expressed as a matrix, so in this assignment, the graph is expressed as a list of edges for each starting node. The method to read the given graph file is implemented as readGraph of Graph class. The Edge class is newly defined to manage the list of Edges starting from one node. Therefore, the graph is expressed as an array having an Edge array as an element.

1. Dijkstra Algorithm

Priority queue was used to implement Dijkstra's algorithm. When defining an edge, src, dst, and weight are specified, and the class is defined so that the comparison of the edge is made by weight size. The edge that enters the priority queue has src as a pivot node and dst as each node. In other words, this edge becomes the route from src to dst, and the weight is the sum of all weights to the destination. The reason for using the priority queue is that when selecting a node to be investigated, the point that currently has the smallest weight should be searched first. This is because if a random investigation is performed, the route to the node may not be optimal, and even if route to dst is improved later, additional computational cost is required.

First, the weight of the starting node is initialized to 0, and the route to other nodes has not been determined yet, so initialize it to a very large value. Then put them all in the priority queue and loop until all routes have been examined.

In each loop, it first finds out node that is obtained through pq.poll() by priority queue. And it updates the shortest route of all nodes that the node can go to through the node. At this time, the adjacent node whose shortest route is updated must be a node that has not been investigated, and if the weight of the route that has passed through the node being investigated is less than the current route, shortest path is updated. However, given the graph input, there are nodes that cannot be reached in the end because the graph has a direction. This is called an external graph, but Integer.MAX\_VALUE is not updated because it cannot be reached for such a node. Therefore, every time an edge is searched, such a node is not searched. A more detailed explanation is provided in the code comments.

1. Find longest shortest path

In order to, find the longest shortest distance in a given graph, all nodes have to be searched. However, if you do a complete search, it will take too long. Since the graph is a directional graph, the nodes whose shortest distance has already been searched through the previous node cannot find the longer shortest distance even if the shortest distance is calculated based on the corresponding node. The reason is similar to Dijkstra's algorithm, but since the node that has already been searched is a node that passes through in the previous step, the value found in the previous step is larger when the shortest distance is calculated based on that criterion. So this does not help in finding the longest shortest distance.

So, by defining the CheckTable class, I can reduce the operation by checking the nodes that have already been searched. In the process of checking that the corresponding node has been searched, the index is used, so the space complexity is O(n), and the temporal complexity for modification and retrieval is O(1).

This simple method was able to speed up the inspection.

1. Heuristic Algorithm to solve problem #3.

In an algorithm that finds the longest shortest path, you need to find the node at the edge to find the answer quickly. At this time, there are nodes that become the source of many edges and the nodes that become destinations. In order to quickly find the longest shortest path with Dijkstra, I thought it would be better to search for nodes that do not become destinations of many edges. In that case, I thought that the nodes of the endpoints that do not become transit nodes will be searched first and result in a quick result.

1. **Algorithm Analysis**
2. Dijkstra Algorithm

Priority queues were used to implement the Dijkstra algorithm. At this time, in the worst case, the root of the node is updated every time all edges of the graph are inspected, and insertion and deletion in the priority queue continues. In the worst case, the search node E is put in the priority queue. In this case, the time complexity of comes out. And if the graph does not allow redundant edges, it is the same as because . At this time, since , if the constant is removed, it becomes . Of course, the search time is less than , so ignore it.

1. Find longest shortest Path

CheckTable was used to implement the algorithm. At this time, it is to check whether all the CheckTables are checked, and to bring out the next node to be checked.

In order to find the longest shortest path for all nodes, the method of excluding the node already searched by Dijkstra from the node to be searched was used. This reduces the need to search all nodes, but if you think about the worst case, it is . But in practice, you can find the solution much faster. Dijkstra with time complexity is executed at each iteration, and the time complexity of the process of bringing out the next search node is as mentioned above. This is executed until all nodes are discovered, so the total time complexity is as follows.

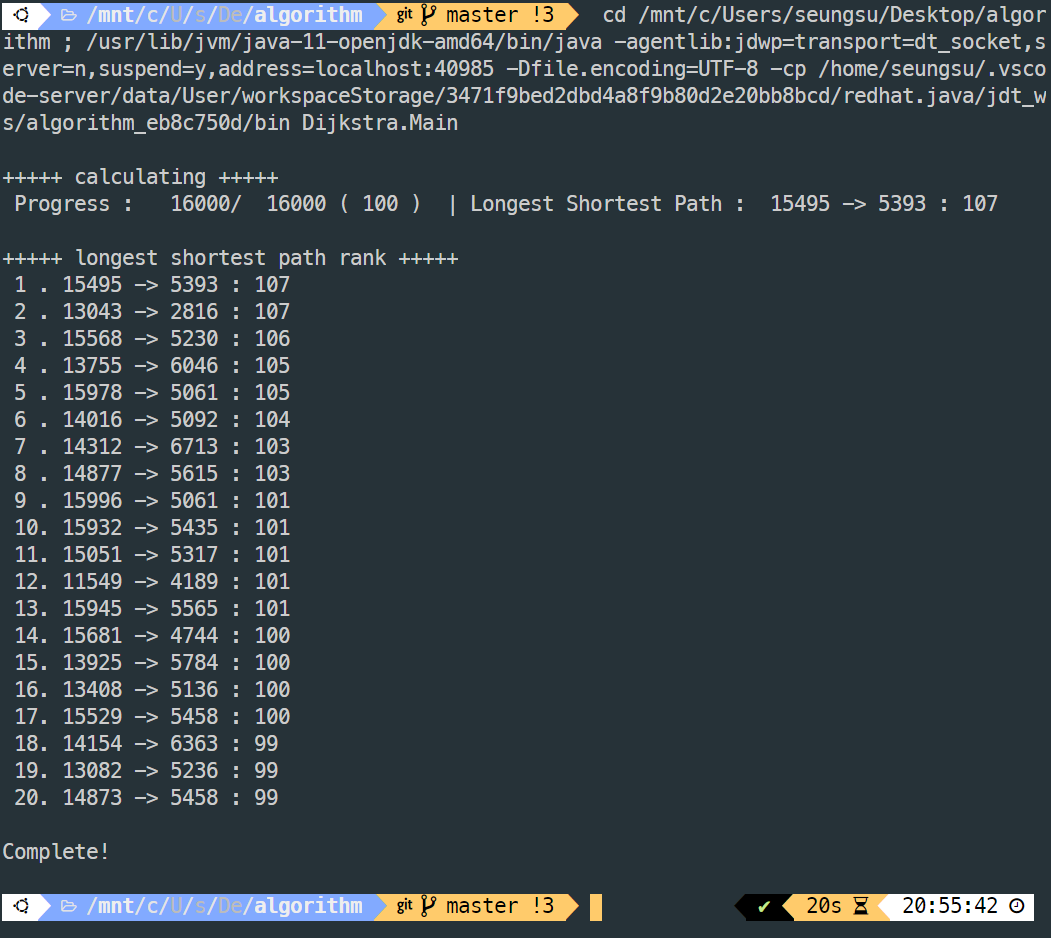
1. Find longest shortest Path by Heuristic

(Search in reverse order of the number of edges entering the node)

As above, it should be searched for all nodes, so theoretically, it should be searched times (actually shorter). In order to use the heuristic, the node to be searched was pulled out through the priority queue. It is because it is pulled from the heap. And the Dijkstra algorithm with time complexity of is executed. Therefore, this time complexity is expressed as an equation as follows.

1. **Case Study**
2. 16000.graph

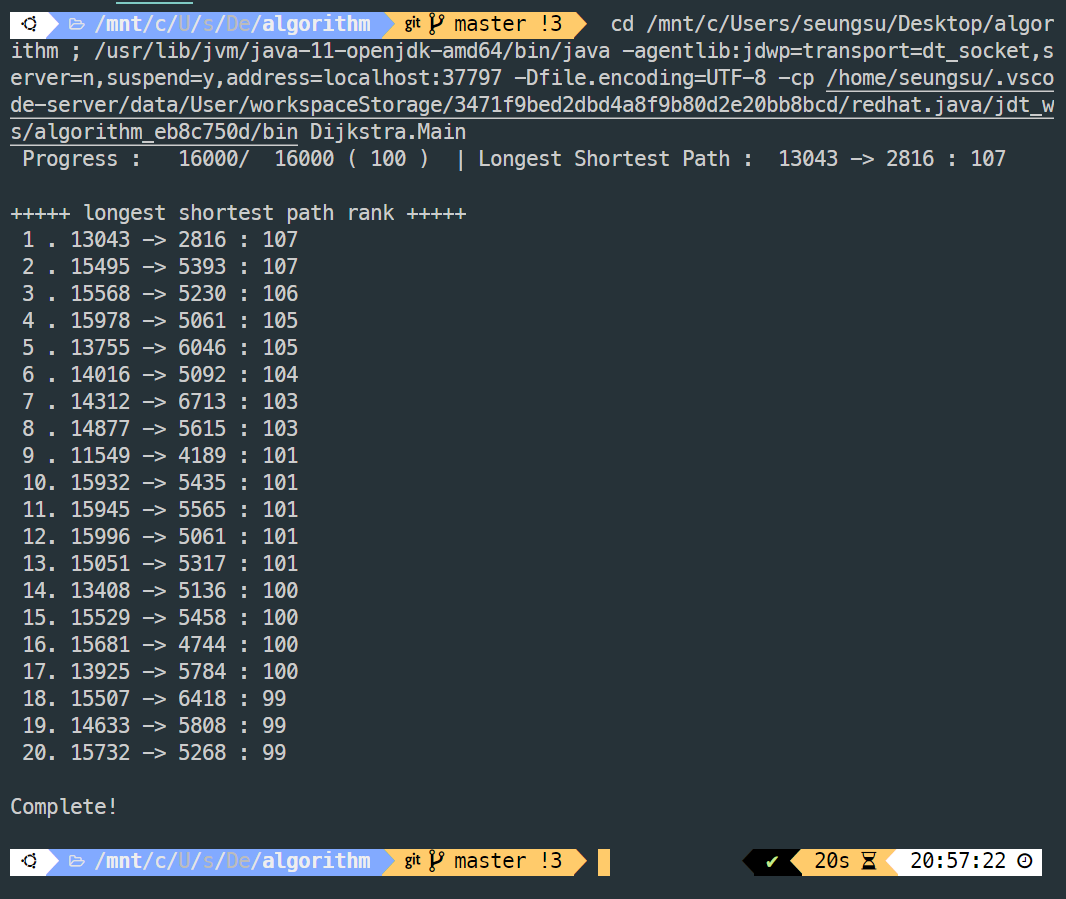
* Base Result



This is the result of finding the default shortest find for graph data with 16000 nodes. We can see that the route from node 15495 to node 5393 has a weight of 107, which is the longest and shortest distance in this graph. And the route from node 13043 to node 2816 is also the shortest distance with 107 weights.

It took a total of 20 seconds to scan all 16000 nodes. The time was reduced a lot by excluding the discovered nodes than when examining all the nodes one by one. How to exclude nodes is described above. In the basic algorithm without applying heuristics, it took 9 seconds to find the shortest path and 20 seconds to check all nodes.

* Result with heuristic



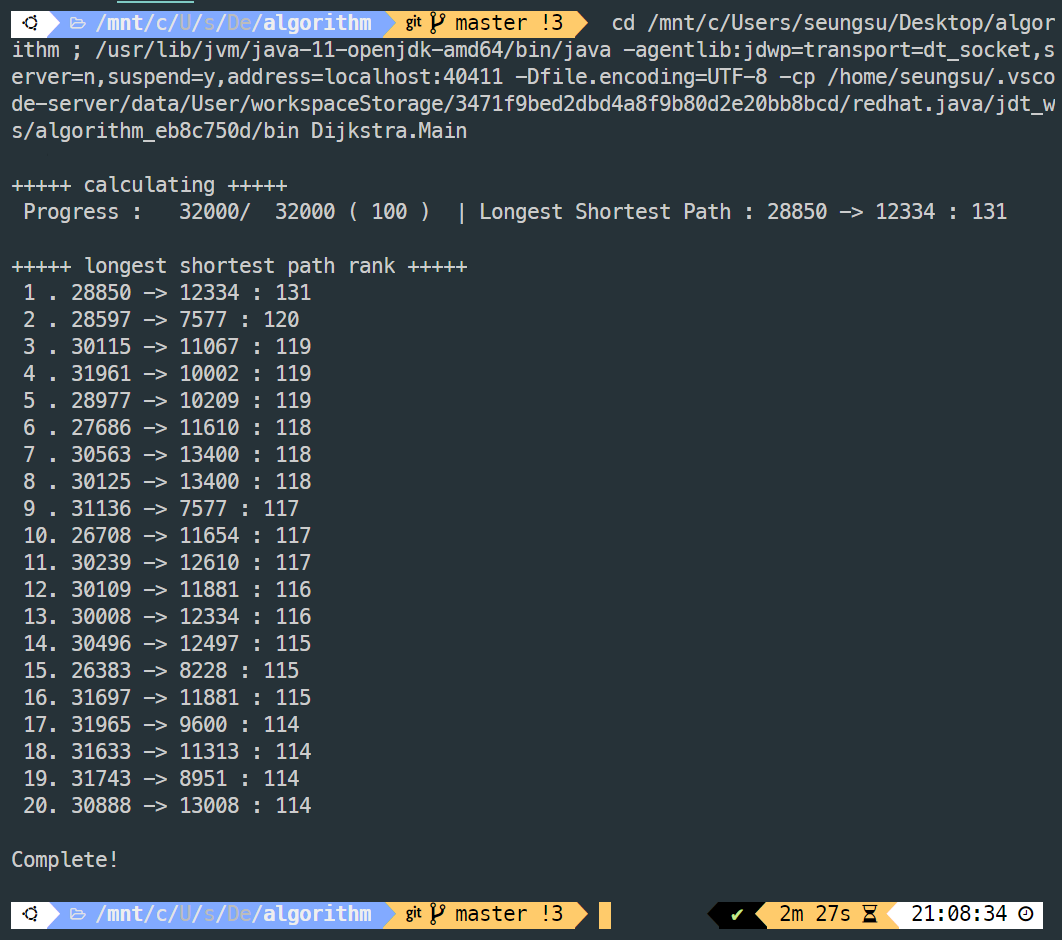
To apply the heuristic idea to a graph with 1 million nodes, I re-tested the heuristic on a graph with 16000 nodes. The result of finding the shortest path through the algorithm applying heuristics is as above and the same as the result obtained earlier. With heuristics, the total execution time was 20 seconds, but the time it took to reach the final result was reduced to 5 seconds.

* Analysis

Looking at the two results, it can be seen that the path from node 15495 to the 5393th node and the path from 13043 to 2816 are the longest shortest path with a weight of 107. And there was no difference in the execution time of the two experiments. However, the speed approaching the correct answer of 107 was shortened from 9 seconds to 5 seconds, indicating that the heuristic algorithm is meaningful.

1. 32000.graph

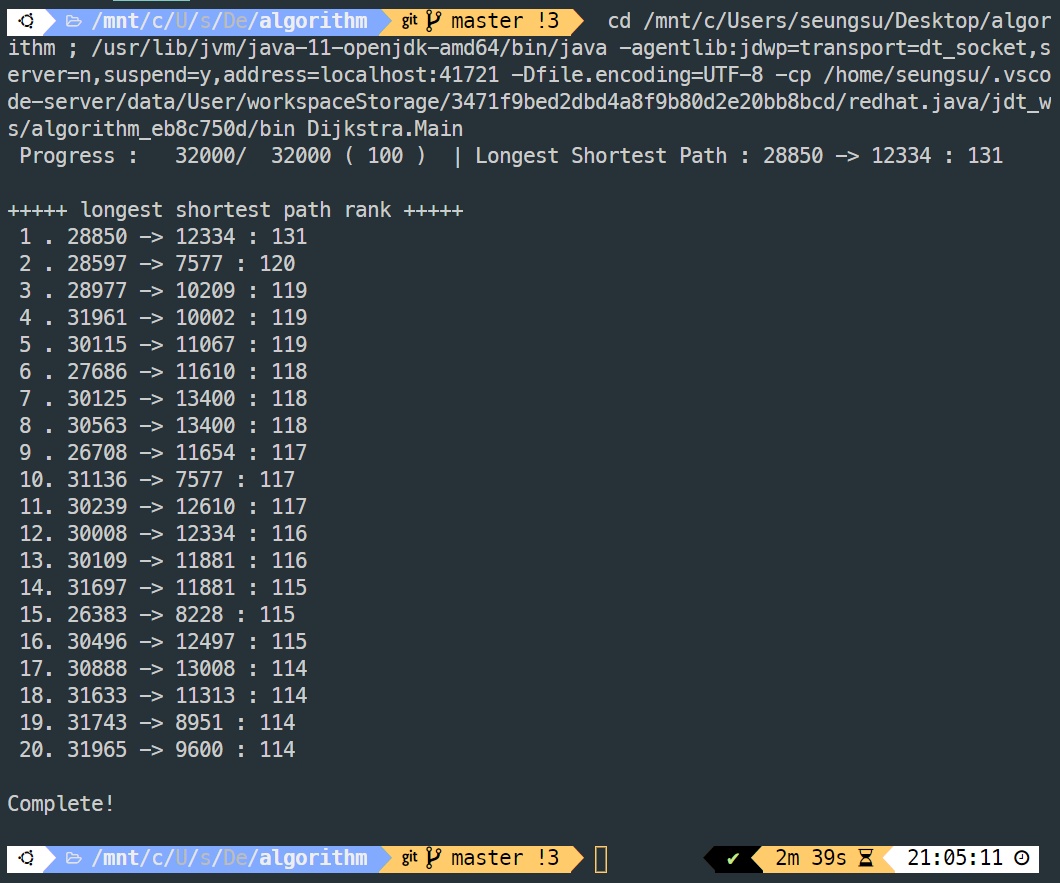
* Base Result



The result of obtaining the longest shortest path by graphing the data of 32000.graph is as in the image above. Looking at the results, it can be seen that the route from node 28850 to node 12334 is the longest shortest path with a weight of 131. After that, the weights are 120, 119, ..., so it can be seen that the gap with the longest shortest path is large.

When executing the base code, the execution time was 2 minutes and 27 seconds, and it was confirmed that it took more time than when searching the graph with 16000 nodes. And it took 1 minute 50 seconds to find the final longest shortest path.

* Result with heuristic



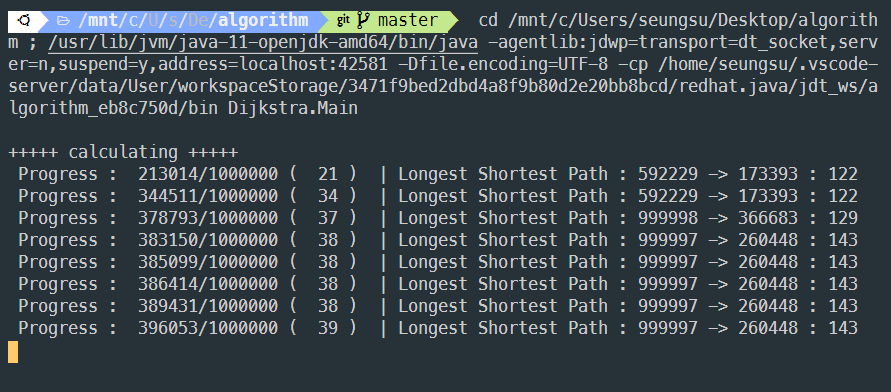
The result of finding the shortest path through an algorithm applying heuristics is as above and is the same as the result obtained earlier. The experiment conducted through the heuristic algorithm took 2 minutes and 37 seconds to run, and it took 55 seconds to find the longest shortest path.

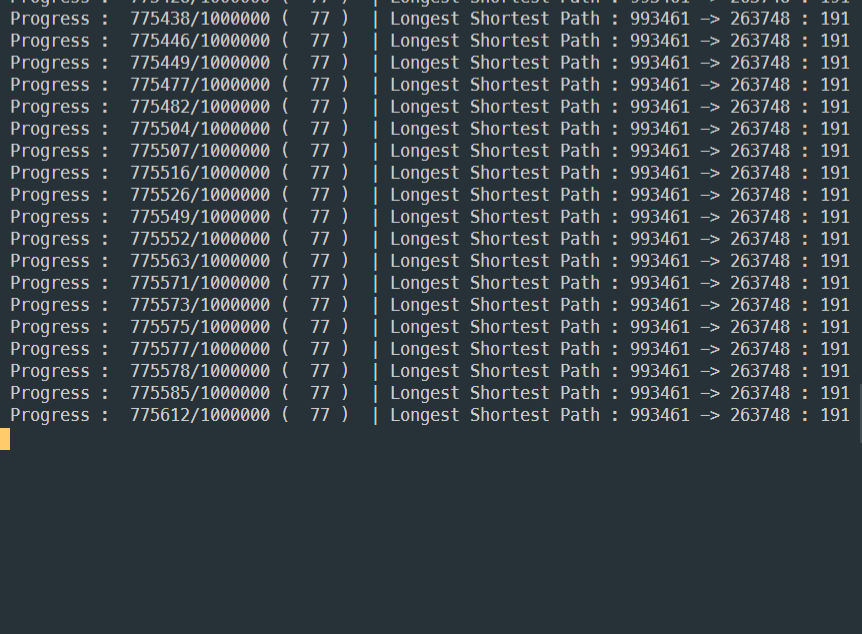
* Analysis

The performance of the heuristic algorithm could be more dramatically confirmed when exploring a graph composed of 32,000 nodes than a graph composed of 16000 nodes. When the amount of data was small, it improved about 5 seconds, but as the data increased, it took 55 seconds to derive the correct answer. The overall lead time increased, but it ended faster to get the correct answer. It is thought that the difference in the total time required was different in determining the search order (priority queue) in implementing the heuristic algorithm.

1. 1000000.graph

* Base Result

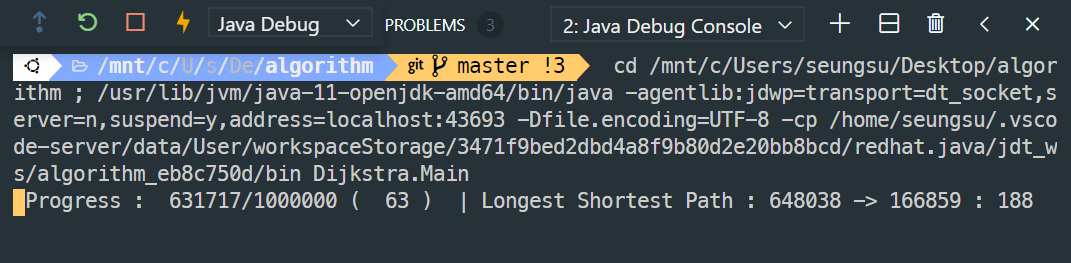


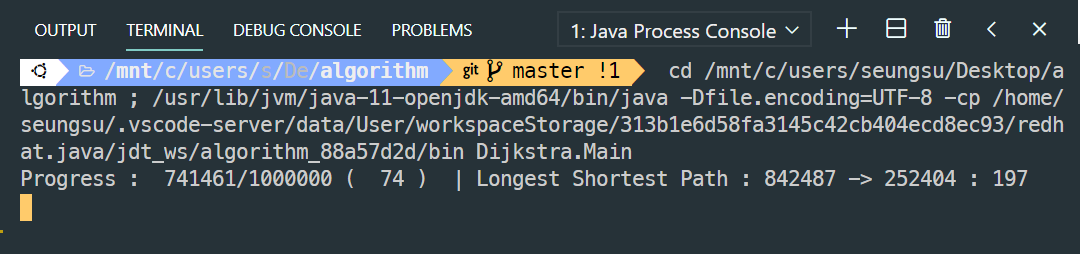


Initially, I tried to find the longest shortest path of a graph with a million nodes using the base code. The image above is the result of executing the code for about 20 hours. Even if the number of nodes to be searched was reduced, it was impossible to search all nodes because it only took 15 seconds to search for one node with Dijkstra. When it was run for 20 hours, 770,000 nodes were searched. At first, it was searched at a high speed, and as the process progressed, the number of nodes that can be excluded with one search decreases, and thus it is slow.

When 770,000 nodes were searched, the longest shortest path was the route from the 993461th node to the 263748th. The weight at this time is 191. A better solution may come out because not all nodes are explored.

* Result with heuristic





This is the result of running the heuristic algorithm. Many nodes have not been searched, but the result obtained is 188, which is quite similar to the result obtained earlier. The image above shows the situation when the algorithm has been run for about 9 hours. You can see that the answer was found quite quickly. I remember that the method without the heuristic was finding a route with a weight of about 160 after about 10 hours (I am sorry that the data in the experiment result is poor because the execution time is long). It is correct to find the correct answer quickly, but it seemed to take a long time to check all nodes. In the end, the process could not be completed, and the longest shortest path found when checking 740,000 nodes was the route from the 842487th node to the 252404th node with a weight of 197.

* Analysis

Another heuristic is used, which is an idea based on the distribution of the data, and searches for the node with the highest number first. The structure of the data was due to the idea that there is a high probability that there is a node with the longest shortest path because many front nodes are connected and the nodes with the rear number have many outgoing edges. After searching for 20 hours, a path with a weight of **197** was obtained as above. **The final result** is the shortest path with the longest path from node 842487 to node 252404.

I tried applying heuristics in various ways, but it seemed to be different depending on how quickly the answer was explored. By applying heuristics, I was able to quickly approach the correct answer, but it seems that the luck factor also worked.