**Programming Project Improving Dijkstra’s algorithm**

NAME: Zheming Xu ID:301414922

**Part1(objective):**

This program will perform the Dijkstra's Algorithm (improved by binary heap), A\*-search Algorithm and, landmark algorithm each for the given nodes and the adjacent lists, and return the number of nodes visited for each algorithm after choosing the source node s and destination node t.

**Part2(implementation):**

Firstly, I need to construct graph ADT. This includes three struct type: linknode, header (for adjacent list), and Graph and three operations to initial the nodes, initial the adjacent list and complete the graph through inputting the Input Graph provided.

Secondly, I need to construct the distance function (already provided in the ppt in Python) and binary heap ADT. This ADT includes a struct type pointer PriorityQueue and six operations: Initialization, isEmpty, Precolatedown, CreateHeap, DeleteMin, DecreaseKey, which are sufficient for improving the algorithm. The last three operations have two types in the source code because the three algorithms have different keys for the heap. (Dijkstra’s key is dist[u] and A\*-search’s key and landmark’s key is dist[u]+to\_t[u]).

Thirdly, create the three algorithms. For Dijkstra’s algorithm, we firstly initial the dist[u] for all nodes and create the PriorityQueue to contain all nodes using dis[u] as key. And then while the queue is not empty, preform the DeleteMin and the DecreaseKey operation(relax) if necessary until the destination node is visited. For A\*-search algorithm, we also use the PriorityQueue but the key changes to dist[u]+to\_t[u]. In this case for to\_t[u] I choose euclidean distance from u to t for estimation. For landmark algorithm, I choose three landmarks(nodes with num = 250,500,750) for better queries, and record dist\_250[u], dist\_500[u], dist\_750[u] for all nodes through preprocessing the A\*-search for these three nodes as source nodes to all other nodes as destination nodes. And then I assign the maximum of |distz[u]-disz[t]| to to\_t[u] as a better estimation. Finally, have the same PriorityQueue operation as A\*-search algorithm.

Finally, create the main function. Use rand() to generate 20 pairs of nodes for tests and output the results we want.

**Part3:** **Determine, on an average, what are the savings if A\* and Landmark algorithms are used instead of Dijkstra’s algorithm**

Through the output in the data file, we could see for the same generated pair, the number of nodes visited of A\*-search and landmark algorithm is far less than that of Dijkstra’s algorithm, which means better efficiency. Because A\*-search and landmark use dist[u]+to\_t[u] as the key for heap operation, they will visit relatively fewer nodes and eliminate the possibility of visiting all the nodes whose dist[u] is less than dist[t], which is a big problem for Dijkstra’s algorithm. And better elimination means fewer nodes visited and better efficiency. This is the savings on an average.