Recipe for bfs_dfs:

Input: graph, which is a DiGraph object that represents the street graph

rac_class, which is a restricted access containerclass(queue or stack)

start_node, which is the starting node in graph

end_node, which is the end node in graph

Output: parent, which is a map that maps nodes in graph to their parent nodes

1. Initialize the variable rac to an empty rac_class
2. Initialize the variable dist to an empty map
3. Initialize the variable parent to an empty map
4. Iterate over each node in graph and assign the value of node to the variable node. Inside the iteration:
   i. Assign the value of infinity to the value of node in dist.
   ii. Assign the value of none to the value of node in parent
5. Assign the value of 0 to the value of start_node in the map dist
6. Push the start_node into rac
7. While length of rac doesn't equal to 0:
   i. Remove the node from rac and assign the value of the node to the variable node
   ii. Get the neighbors of the node from the graph by calling the method get_neighbors and assign the value to the variable nbrs.
   iii. Iterate each node in nbrs and assign the value to nbr. Inside the iteration:
      a) Check if the value of nbr in the map dist equals to infinity, if the output is true:
         1. the value of nbr in the map dist equals to the value of nbr in the map dist plus 1
         2. the value of nbr in the map parent equals to the value of node
         3. Push the nbr into rac
         4. Check if the value of nbr in the map dist equals to end_node, if the output is true:

            Return parent
8. Return parent

Recipe for dfs:

Input: graph, which is a DiGraph object that represents the street graph

start_node, which is the starting node in graph

end_node, which is the end node in graph

parent, which is a map that maps nodes in graph to their parent nodes

Output: parent, which is a map that maps nodes in graph to their parent nodes

Algorithm used: dfs

Base case: start_node=end_node, or counter=0, which means the entire graph has been searched

Recursive case:

start_node does not equal to end_node, or counter does not equal to 0

1. Get the keys of the map parent and assign the value to the variable key.
2. Assign the value of 0 to the variable counter
3. Iterate each node in nbrs and assign the value to element1. Inside the iteration:
   i. Determine whether parent has element1 as one of its key, if the output is False:
      a) Assign the value of 1 to the variable counter.
4. Determine whether start_node=end_node or counter equals to 0, if the output is true:
   i. Return parent

   If the output is false:
   i. Iterate each node in nbrs and assign the value to nbr. Inside the iteration:
      a) Determine whether nbr belongs to key, if the output is true:
         1. The value of nbr in the map parent equals to the value of start_node.
         2. Call the function dfs, with graph, nbr, end_node and parent as its arguments
   ii. Return parent

Recipe for astar

Input: graph, which is a DiGraph object that represents the street graph
start_node, which is the starting node in graph
end_node, which is the end node in graph
edge_distance, which is a function used to calculate the g (actual distance) between two nodes
straight_line_distance, which is a function used to calculate h (heuristic distance) between the node and the ending node

Output:
parent, which is a map that maps nodes in graph to their parent nodes

Algorithms used: edge_distance, whose parameters are node1, node2, graph, return value of g
straight_line_distance, whose parameters are node1, node2, graph, return value of h

1. Initialize openset, closedset to empty sequences. Add start_node into the openset.
2. Create a map gcost, assign the value of 0 to the key start_node. Calculate the h value by calling the straight_line_distance function with start_node, end_node, graph and then map the h value to start_node. Use the variable hcost to refer to this map. Create a map parent, assign the value of None to the key start_node.
3. While the openset is not empty:
   i. get the first element in the openset and assign its value to the variable minnode
   ii. add the value of minnode in the map gcost with the value of minnode in the map hcost, assign the value of the sum to the variable minf.
   iii. Iterate over each node in the openset and assign the value to the variable node, inside the iteration:
      a) add the value of node in the map gcost with the value of node in the map hcost, assign the value of the sum to the variable fcost.
      b) Determine whether fcost is smaller than minf, if the

ouput is true:

i. Assign the value of node to minnode. Assign the value of fcost to minf

iv. Determine if the value of minnode equals to the value of end_node. If the output is true:

a) Just stop

v. Remove minnode from the openset. Add minnode into the closedset.

vi. Get neighbors of the minnode by calling the function get_neighbors with minnode as its parameter. Iterate over each node of neighbors and assign the value to nbr. Inside the iteration:

i. Add the value of minnode in the map gcost with the value of h, calculated by calling the function edge_distance with nbr, end_node, graph as its 3 parameters. Assign the value of the sum to the variable newgcost.

ii. Determine if nbr in the openset, if the output is true:

1. Determine if newgcost is smaller than the value nbr in the map gcost. If the output is true:

a) Assign the value of newgcost to the value nbr in the map gcost. Assign the value of minnode to the value of nbr in the map parent.

iii. If the output is false, determine whether nbr is both not in openset and not in closedset, if the ouput is true:

1. Add nbr into the openset
2. Assign the value of minnode to the value of nbr in the map parent.
3. Calculate the value of h by calling the function straight_line_distance function with nbr, end_node, graph and then map the h value to the key nbr in the map hcost.
4. Assign the value of newgcost to the value of the key nbr in the map gcost.

5. Return the map parent

Discussion:

1. The same place is that they have same starting point and end point. Besides, they both give routes by exploring points. Also the route given by them both may change even you fix the starting point and end point.

The difference is that the bfs_dfs will explore fewer points than recursive dfs algorithum. In fact, recursive dfs almost explore all the points while bfs_dfs only explores partial points. However, bfs_dfs actually gives the route that is farther than the route given by recursive dfs. Also, recursive one takes more time than bfs_dfs.

Therefore recursive dfs is better because it although it needs to explore the whole map, it can find the better route (shorter).

2. It depends. When your target is deep down in the map, it is better to use A star algorithm. This is because the route given by it is more close to the google map route. In fact, the distance of the route given by A star is shortest among all the algorithms.
But if your target is around the start point, bfs and A star work both very well because the route given by them are almost same and almost matches the google map route.
The reason I think is that A star algorithm always explores the node very deeply and downwardly. Therefore, it will work well when the goal is far away.
While bfs explores the route that is with fewest steps from the start point, therefore when the target is just around the start point, it works well.

3. dfs in bfs_dfs algorithm. This is because it always gives the farthest route compared with other algorithms. Sometimes it will even not get to the target. This is because it will just keep visiting the first child of every node it sees, so if the one you're looking for isn't the first child of its parent, it will never get there.

4. Maybe google map uses a better algorithm which can find the best root quickly accurately and efficiently. Our best algorithm will be slowed down when the map becomes larger while the goolge map will not. Another reason I think is that google map may have already stored large amount of data about the route choice. Therefore it can choose the best one or do the optimization based on its big data while ours doesn’t too much existent data.