This is my search node. I added the h g f depth and parent to my init and created the showverbose function which is just a print function for each of the nodes

```python
class SearchNode:
    def __init__(self, label, pathcost=None, g=0, h=0, depth=0, parent=None):
        self.label = label
        self.pathcost = pathcost
        self.g = g
        self.h = h
        self.f = g + h
        self.depth = depth
        self.parent = parent

    def showverbose(self):
        return f"{self.label};{self.depth};{self.g:.0f}; {self.h:.2f};{self.f:.2f}"

    def __lt__(self, other):
        return self.label < other.label

    #-------------- FROM THE LAST PART -----------
def hSLD(label, searcher):
    # Calculate straight-line distance (Euclidean) between label node and goal node
    node1 = None
    node2 = None

    # Find the two nodes in the graph
    for node in searcher.myViz.nodes:
        if node.label == label:
            node1 = node
        if node.label == searcher.goal:
            node2 = node

    # Calculate Euclidean distance
    if node1 and node2:
        dx = node1.x - node2.x
        dy = node1.y - node2.y
        return math.sqrt(dx*dx + dy*dy)

    return 0
```

My searcher class innit has added the verbose and i changed the print statement and added g h depth and path cost to any search node calls

```python
#--------- FROM PART 0 WITH UPDATED INIT ---------
class Searcher:
    def __init__(self, file_name, searchType, verbose):
        self.myViz = DRDViz()
        self.myViz.loadGraphFromFile(file_name)
        self.start = None
        self.goal = None
        self.open = []
        self.searchType = searchType
        self.verbose = verbose
        print(f"Loaded search type {self.searchType} with map in file: {file_name}")

    def setStartGoal(self, start, goal):
        self.start = start.upper()
        self.goal = goal.upper()
        self.open.append(SearchNode(self.start,pathcost=0, g=0, h=0, depth=0, parent=None))

    def successors(self, node_label):
        children = []
        for edge in self.myViz.edges:
            if edge.endlabel1 == node_label:
                children.append(SearchNode(edge.endlabel2, pathcost=int(edge.label)))
            elif edge.endlabel2 == node_label:
                children.append(SearchNode(edge.endlabel1, pathcost=int(edge.label)))
        # Sort alphabetically by label
        return sorted(children)

    def insert_front(self, node):
        # Must handle both single node and list
        if isinstance(node, list):
            for n in reversed(node):
                self.open.insert(0, n)
        else:
            self.open.insert(0, node)

    def insert_end(self, node):
        # Must handle both single node and list
        if isinstance(node, list):
            for n in node:
                self.open.append(n)
        else:
            self.open.append(node)
```

I added a function to check for duplicates

```
    def insert_ordered(self, node):
        # Must handle both single node and list
        if isinstance(node, list):
            for n in node:
                self._insert_single(n)
        else:
            self._insert_single(node)

    def _insert_single(self, node):
        # Insert in order by value
        for i, existing in enumerate(self.open):
            if node.f < existing.f:
                self.open.insert(i, node)
                return
        self.open.append(node)

    def reset(self):
        self.open = []
        if self.start:
            self.open.append(SearchNode(self.start, pathcost=0, g=0, h=0, depth=0, parent=None))

  #-------------- CREATED FUNCTION FOR DUPLICATES --------------
    def duplicate(self, label):
        return any(n.label == label for n in self.open)
```

```
#----------- NEW STUFF ----------

#this just checks what search type is being used and then calls the correct function
    def search(self):
        #if search type is depth
        if self.searchType == "DEPTH":
            #call dfs
            self.dfs()
        #if search type is breadth
        if self.searchType == "BREADTH":
            #call bfs
            self.bfs()
        #if search type is best
        if self.searchType == "BEST":
            #call best first
            self.greedy()
        #if search type is A*
        if self.searchType == "A*":
            #call A*
            self.astar()
```

```python
#----------- DFS ------------
    def dfs(self):
        #create a set to keep track of visited nodes
        self.closed = set()
        #loop through the list of nodes
        while self.open:
            #pop the first node off the list
            node = self.open.pop(0)
            #show the node that youre going to explore
            if self.verbose:
                #function print
                print(f"Exploring node: {node.label}")

            #if current node is goal node
            if node.label == self.goal:
                #create a list
                path = []
                current = node

                #while at node
                while current:
                    #add the the node to the list
                    path.append(current.label)
                    #change to the parent
                    current = current.parent
                #reverse the list which puts it in order
                path.reverse()
                #print the path
                #function print
                print(f"Success! Reached goal node {self.goal} with path: {path}")
                #return path
                return path

            #if the node is not in the closed list
            if node.label not in self.closed:
                #add to closed list
                self.closed.add(node.label)

                #create the children
                #function successors
                children = self.successors(node.label)

                #filter out visited children
                open_children = [c for c in children if c.label not in self.closed]

                #insert the new children that are not in the closed list
                if self.verbose and open_children:
                    #function print
                    print(f"Inserting new children: {[c.label for c in open_children]}")

                #create a list to hold all the children
                allowed_children = []
                #for child in children
                for child in children:
                    #if child is not in the closed list
                    if child.label not in self.closed:
                        #add the child and characteristics if not duplicate
```

```
                    #add the child and characteristics if not duplicate
                    #function duplicate
                    if not self.duplicate(child.label):
                        child.parent = node
                        child.depth = node.depth + 1
                        child.g = node.g + child.pathcost
                        #set h to zero because we dont need heuristic
                        child.h = 0
                        child.f = child.g
                        allowed_children.append(child)

            #reverse the children when coming back to the parent
            for child in reversed(allowed_children):
                self.insert_front(child)

            #print the open list
            if self.verbose and open_children:
                #function print
                print(f"Open list: {[n.showverbose() for n in self.open]}")


#----------EXACT SAME AS DFS BUT INSTEAD OF INSERT FRONT YOU INSERT END AND GET RID OF REVERSE -------
    def bfs(self):
        #create a set to keep track of visited nodes
        self.closed = set()
        #loop through list of nodes
        while self.open:
            #pop the first node off the list
            node = self.open.pop(0)

            #show the node that youre going to explore
            if self.verbose:
                #function print
                print(f"Exploring node: {node.label}")

            #if current node is goal node
            if node.label == self.goal:
                #create a list
                path = []
                current = node

                #while at node
                while current:
                    #add the the node to the list
                    path.append(current.label)
                    #change to the parent
                    current = current.parent
                #reverse the list which puts it in order
                path.reverse()
                #print the path
                    #function print
                print(f"Success! Reached goal node {self.goal} with path: {path}")
                #return the path
                return path

            #if the node is not in the closed list
```

```
            #if the node is not in the closed list
            if node.label not in self.closed:
                #add to closed list
                self.closed.add(node.label)

                #create the children
                #function successors
                children = self.successors(node.label)

                #filter out visited children
                open_children = [c for c in children if c.label not in self.closed]

                #write the new children that are not in the closed list
                if self.verbose and open_children:
                    #function print
                    print(f"Inserting new children: {[c.label for c in open_children]}")

                #create a list to hold all the children that arent duplicates
                allowed_children = []
                #for child in children
                for child in children:
                    #if child is not in the closed list
                    if child.label not in self.closed:
                        #add the child and characteristics if not duplicate
                        #function duplicate
                        if not self.duplicate(child.label):
                            child.parent = node
                            child.depth = node.depth + 1
                            child.g = node.g + child.pathcost
                            child.h = 0
                            child.f = child.g
                            allowed_children.append(child)

            #reverse the children when coming back
            for child in allowed_children:
                self.insert_end(child)

            #print the open list
            if self.verbose and open_children:
                #function print
                print(f"Open list: {[n.showverbose() for n in self.open]}")


#------------ GREEDY BEST FIRST ------------
    def greedy(self):
        #create a set to keep track of visited nodes
        self.closed = set()
        #loop through list of nodes
        while self.open:
            #pop the first node off the list
            node = self.open.pop(0)

            #show the node that youre going to explore
            if self.verbose:
                print(f"Exploring node: {node.label}")
```

```
            print(f"Exploring node: {node.label}")

        #if current node is goal node
        if node.label == self.goal:
            #create a list
            path = []
            current = node

            #while at node
            while current:
                #add the the node to the list
                path.append(current.label)
                #change to the parent
                current = current.parent
            #reverse the list which puts it in order
            path.reverse()
            #print the path
            print(f"Success! Reached goal node {self.goal} with path: {path}")
            #return path
            return path

        #if the node is not in the closed list
        if node.label not in self.closed:
            #add to closed list
            self.closed.add(node.label)

            #create the children
            children = self.successors(node.label)

            #filter out visited children
            open_children = [c for c in children if c.label not in self.closed]

            #insert the new children that are not in the closed list
            if self.verbose and open_children:
                #function print
                print(f"Inserting new children: {[c.label for c in open_children]}")

            #create a list to hold all the children
            allowed_children = []
            #for child in children
            for child in children:
                #if child is not in the closed list
                if child.label not in self.closed:
                    #add the child and characteristics if not duplicate
                    #function duplicate
                    if not self.duplicate(child.label):
                        child.parent = node
                        child.depth = node.depth + 1
                        child.g = node.g + child.pathcost
                        #set h to the heuristic
                        child.h = hSLD(child.label, self)
                        #set f to h
                        child.f = child.h
                        allowed_children.append(child)

        #order the children
```

```
            #order the children
            for child in allowed_children:
                self.insert_ordered(child)

            #print the open list
            if self.verbose and open_children:
                #function print
                print(f"Open list: {[n.showverbose() for n in self.open]}")

#------------ A* BEST FIRST ------------
    def astar(self):
        #create a set to keep track of visited nodes
        self.closed = set()
        #loop through list of nodes
        while self.open:
            #pop the first node off the list
            node = self.open.pop(0)

            #show the node that youre going to explore
            if self.verbose:
                #function print
                print(f"Exploring node: {node.label}")

            #if current node is goal node
            if node.label == self.goal:
                #create a list
                path = []
                current = node

                #while at node
                while current:
                    #add the the node to the list
                    path.append(current.label)
                    #change to the parent
                    current = current.parent
                #reverse the list which puts it in order
                path.reverse()
                #print the path
                print(f"Success! Reached goal node {self.goal} with path: {path}")
                #return
                return path

            #if the node is not in the closed list
            if node.label not in self.closed:
                #add to closed list
                self.closed.add(node.label)

                #create the children
                children = self.successors(node.label)

                #filter out visited children
                open_children = [c for c in children if c.label not in self.closed]

                #insert the new children that are not in the closed list
                if self.verbose and open_children:
                    print(f"Inserting new children: {[c.label for c in open_children]}")
```

```
            print(f"Inserting new children: {[c.label for c in open_children]}")

        #create a list to hold all the children
        allowed_children = []
        for child in children:
            #if child is not in the closed list
            if child.label not in self.closed:
                #add the child and characteristics if not duplicate
                if not self.duplicate(child.label):
                    child.parent = node
                    child.depth = node.depth + 1
                    child.g = node.g + child.pathcost
                    #set h to the heuristic
                    child.h = hSLD(child.label, self)
                    #set f to g + h
                    child.f = child.g + child.h
                    allowed_children.append(child)

    #order the children
    for child in allowed_children:
        self.insert_ordered(child)

    #print the open list
    if self.verbose and open_children:
        #function print
        print(f"Open list: {[n.showverbose() for n in self.open]}")
```

```
PS C:\Users\ewf08\OneDrive\Desktop\470 Projects\Route Finding 1> python .\06_tests.py
Loaded search type DEPTH with map in file: 10test.txt
Exploring node: H
Inserting new children: ['A', 'D', 'J']
Open list: ['A;1;72; 0.00;72.00', 'D;1;135; 0.00;135.00', 'J;1;223; 0.00;223.00']
Exploring node: A
Inserting new children: ['E', 'G']
Open list: ['E;2;680; 0.00;680.00', 'G;2;199; 0.00;199.00', 'D;1;135; 0.00;135.00', 'J;1;223; 0.00;223.00']
Exploring node: E
Inserting new children: ['C']
Open list: ['C;3;973; 0.00;973.00', 'G;2;199; 0.00;199.00', 'D;1;135; 0.00;135.00', 'J;1;223; 0.00;223.00']
Exploring node: C
Inserting new children: ['F', 'I', 'K', 'L']
Open list: ['F;4;1251; 0.00;1251.00', 'I;4;1264; 0.00;1264.00', 'K;4;1243; 0.00;1243.00', 'L;4;1107; 0.00;1107.00', 'G;2;199; 0.00;199.00', 'D;1;135; 0.00;135.00', 'J;1;223; 0.00;223.0
0']
Exploring node: F
Inserting new children: ['J', 'L']
Open list: ['I;4;1264; 0.00;1264.00', 'K;4;1243; 0.00;1243.00', 'L;4;1107; 0.00;1107.00', 'G;2;199; 0.00;199.00', 'D;1;135; 0.00;135.00', 'J;1;223; 0.00;223.00']
Exploring node: I
Inserting new children: ['B', 'D']
Open list: ['B;5;1773; 0.00;1773.00', 'K;4;1243; 0.00;1243.00', 'L;4;1107; 0.00;1107.00', 'G;2;199; 0.00;199.00', 'D;1;135; 0.00;135.00', 'J;1;223; 0.00;223.00']
Exploring node: B
Inserting new children: ['D', 'G']
Open list: ['K;4;1243; 0.00;1243.00', 'L;4;1107; 0.00;1107.00', 'G;2;199; 0.00;199.00', 'D;1;135; 0.00;135.00', 'J;1;223; 0.00;223.00']
Exploring node: K
Success! Reached goal node K with path: ['H', 'A', 'E', 'C', 'K']
Loaded search type BREADTH with map in file: 10test.txt
Exploring node: H
Inserting new children: ['A', 'D', 'J']
Open list: ['A;1;72; 0.00;72.00', 'D;1;135; 0.00;135.00', 'J;1;223; 0.00;223.00']
Exploring node: A
Inserting new children: ['E', 'G']
Open list: ['D;1;135; 0.00;135.00', 'J;1;223; 0.00;223.00', 'E;2;680; 0.00;680.00', 'G;2;199; 0.00;199.00']
Exploring node: D
Inserting new children: ['B', 'I']
Open list: ['J;1;223; 0.00;223.00', 'E;2;680; 0.00;680.00', 'G;2;199; 0.00;199.00', 'B;2;410; 0.00;410.00', 'I;2;512; 0.00;512.00']
Exploring node: J
Inserting new children: ['F', 'L']
Open list: ['E;2;680; 0.00;680.00', 'G;2;199; 0.00;199.00', 'B;2;410; 0.00;410.00', 'I;2;512; 0.00;512.00', 'F;2;350; 0.00;350.00', 'L;2;416; 0.00;416.00']
Exploring node: E
Inserting new children: ['C']
Open list: ['G;2;199; 0.00;199.00', 'B;2;410; 0.00;410.00', 'I;2;512; 0.00;512.00', 'F;2;350; 0.00;350.00', 'L;2;416; 0.00;416.00', 'C;3;973; 0.00;973.00']
Exploring node: G
Inserting new children: ['B']
Open list: ['B;2;410; 0.00;410.00', 'I;2;512; 0.00;512.00', 'F;2;350; 0.00;350.00', 'L;2;416; 0.00;416.00', 'C;3;973; 0.00;973.00']
Exploring node: B
Inserting new children: ['I']
Open list: ['I;2;512; 0.00;512.00', 'F;2;350; 0.00;350.00', 'L;2;416; 0.00;416.00', 'C;3;973; 0.00;973.00']
Exploring node: I
Inserting new children: ['C']
Open list: ['F;2;350; 0.00;350.00', 'L;2;416; 0.00;416.00', 'C;3;973; 0.00;973.00']
```

```
Exploring node: F
Inserting new children: ['C', 'L']
Open list: ['L;2;416; 0.00;416.00', 'C;3;973; 0.00;973.00']
Exploring node: L
Inserting new children: ['C']
Open list: ['C;3;973; 0.00;973.00']
Exploring node: C
Inserting new children: ['K']
Open list: ['K;4;1243; 0.00;1243.00']
Exploring node: K
Success! Reached goal node K with path: ['H', 'A', 'E', 'C', 'K']
Loaded search type BEST with map in file: 10test.txt
Exploring node: H
Inserting new children: ['A', 'D', 'J']
Open list: ['J;1;223; 504.47;504.47', 'D;1;135; 596.16;596.16', 'A;1;72; 710.03;710.03']
Exploring node: J
Inserting new children: ['F', 'L']
Open list: ['L;2;416; 349.17;349.17', 'F;2;350; 495.23;495.23', 'D;1;135; 596.16;596.16', 'A;1;72; 710.03;710.03']
Exploring node: L
Inserting new children: ['C', 'F']
Open list: ['C;3;550; 230.79;230.79', 'F;2;350; 495.23;495.23', 'D;1;135; 596.16;596.16', 'A;1;72; 710.03;710.03']
Exploring node: C
Inserting new children: ['E', 'F', 'I', 'K']
Open list: ['K;4;820; 0.00;0.00', 'E;4;843; 295.63;295.63', 'I;4;841; 383.71;383.71', 'F;2;350; 495.23;495.23', 'D;1;135; 596.16;596.16', 'A;1;72; 710.03;710.03']
Exploring node: K
Success! Reached goal node K with path: ['H', 'J', 'L', 'C', 'K']
Loaded search type A* with map in file: 10test.txt
Exploring node: H
Inserting new children: ['A', 'D', 'J']
Open list: ['J;1;223; 504.47;727.47', 'D;1;135; 596.16;731.16', 'A;1;72; 710.03;782.03']
Exploring node: J
Inserting new children: ['F', 'L']
Open list: ['D;1;135; 596.16;731.16', 'L;2;416; 349.17;765.17', 'A;1;72; 710.03;782.03', 'F;2;350; 495.23;845.23']
Exploring node: D
Inserting new children: ['B', 'I']
Open list: ['L;2;416; 349.17;765.17', 'A;1;72; 710.03;782.03', 'F;2;350; 495.23;845.23', 'I;2;512; 383.71;895.71', 'B;2;410; 838.02;1248.02']
Exploring node: L
Inserting new children: ['C', 'F']
Open list: ['C;3;550; 230.79;780.79', 'A;1;72; 710.03;782.03', 'F;2;350; 495.23;845.23', 'I;2;512; 383.71;895.71', 'B;2;410; 838.02;1248.02']
Exploring node: C
Inserting new children: ['E', 'F', 'I', 'K']
Open list: ['A;1;72; 710.03;782.03', 'K;4;820; 0.00;820.00', 'F;2;350; 495.23;845.23', 'I;2;512; 383.71;895.71', 'E;4;843; 295.63;1138.63', 'B;2;410; 838.02;1248.02']
Exploring node: A
Inserting new children: ['E', 'G']
Open list: ['K;4;820; 0.00;820.00', 'F;2;350; 495.23;845.23', 'I;2;512; 383.71;895.71', 'G;2;199; 766.99;965.99', 'E;4;843; 295.63;1138.63', 'B;2;410; 838.02;1248.02']
Exploring node: K
Success! Reached goal node K with path: ['H', 'J', 'L', 'C', 'K']
Loaded search type DEPTH with map in file: 50test.txt
Success! Reached goal node C with path: ['S', 'AB', 'G', 'N', 'AU', 'A', 'E', 'AA', 'AI', 'D', 'AV', 'F', 'AN', 'AO', 'AQ', 'AD', 'AH', 'AP', 'P', 'Y', 'C']
Loaded search type BREADTH with map in file: 50test.txt
Success! Reached goal node C with path: ['S', 'AB', 'V', 'AG', 'C']
Loaded search type BEST with map in file: 50test.txt
```

```
Loaded search type DEPTH with map in file: 50test.txt
Success! Reached goal node C with path: ['S', 'AB', 'G', 'N', 'AU', 'A', 'E', 'AA', 'AI', 'D', 'AV', 'F', 'AN', 'AO', 'AQ', 'AD', 'AH', 'AP', 'P', 'Y', 'C']
Loaded search type BREADTH with map in file: 50test.txt
Success! Reached goal node C with path: ['S', 'AB', 'V', 'AG', 'C']
Loaded search type BEST with map in file: 50test.txt
Success! Reached goal node C with path: ['S', 'AB', 'V', 'AG', 'C']
Loaded search type A* with map in file: 50test.txt
Success! Reached goal node C with path: ['S', 'AB', 'V', 'AG', 'C']
PS C:\Users\ewf08\OneDrive\Desktop\470 Projects\Route Finding 1>
```