1. A* algo:

h(x) was already made in the code, you can just call it.

The h(x) is calculate by the **Diagonal distance**.

```python
# estimation
def _distance(self, a, b):
    # Diagonal distance
    d = np.max([np.abs(a[0]-b[0]), np.abs(a[1]-b[1])])
    return d
```

You just need to do two things.

i. Complete this function

```python
    #####################################################
    # In a*  we need to add something in this function
    f = self.g[node]
    #####################################################
```

ii. After you find the point, you need to calculate this point's estimation.

```python
# eight direction
pts_next1 = [(p[0]+inter,p[1]), (p[0],p[1]+inter), (p[0]-inter,p[1]), (p[0],p[1]-inter)]
pts_next2 = [(p[0]+inter,p[1]+inter), (p[0]-inter,p[1]+inter), (p[0]-inter,p[1]-inter), (p[0]+inter,p[1]-inter)]
pts_next = pts_next1 + pts_next2

for pn in pts_next:
    if pn not in self.parent:
        self.queue.append(pn)
        self.parent[pn] = p
        self.g[pn] = self.g[p] + inter
        #todo
        ##############################################

        # update the estimation

        ##############################################
    elif self.g[pn]>self.g[p] + inter:
        self.parent[pn] = p
        self.g[pn] = self.g[p] + inter
```

2. RRT algo:

You just need to complete two things:

i. Check collision

```python
# todo
########################################################################################################################
# this "if-statement" is not complete, you need complete this "if-statement"
# you need to check the path is legal or illegal, you can use the function "self._check_collision"
# illegal
if new_node[1]<0 or new_node[1]>=self.map.shape[0] or new_node[0]<0 or new_node[0]>=self.map.shape[1]
########################################################################################################################
    return False, None
# legal
else:
    return new_node, self._distance(new_node, from_node)
```

ii. After you add a node, you need to maintain the tree.

In the first line you need to assign it's parent.

In the second line you need to calculate the new node's cost.

```
def planning(self, start, goal, extend_lens, img=None):
    self.ntree = {}
    self.ntree[start] = None
    self.cost = {}
    self.cost[start] = 0
    goal_node = None
    for it in range(20000):
        print("\r", it, len(self.ntree), end="")
        samp_node = self._random_node(goal, self.map.shape)
        near_node = self._nearest_node(samp_node)
        new_node, cost = self._steer(near_node, samp_node, extend_lens)
        if new_node is not False:
            # todo
            ####################################################################
            # after creat a new node in a tree, we need to maintain something
            self.ntree[""" """] =
            self.cost[""" """] =
            ####################################################################
        else:
            continue
        if self._distance(near_node, goal) < extend_lens:
            goal_node = near_node
            break
```

3. RRT* algo:

You just need to complete three things:

i. Check collision

```
# todo
########################################################################################################################
# this "if-statement" is not complete, you need complete this "if-statement"
# you need to check the path is legal or illegal, you can use the function "self._check_collision"
# illegal
if new_node[1]<0 or new_node[1]>=self.map.shape[0] or new_node[0]<0 or new_node[0]>=self.map.shape[1]
########################################################################################################################
    return False, None
```

ii. After you add a node, you need to maintain the tree.

In the first line you need to assign it's parent.

In the second line you need to calculate the new node's cost

```
        if new_node is not False:
            # todo
            ####################################################################
            # after creat a new node in a tree, we need to maintain something
            self.ntree[""" """] =
            self.cost[""" """] =
            ####################################################################
```

iii After you find a node, you need to maintain the tree.

In the first line you need to assign it's parent.

In the second line you need to update the new node's cost.

```
# Re-Parent
nlist = self._near_node(new_node, 100)
for n in nlist:
    cost = self.cost[n] + self._distance(n, new_node)
    if cost < self.cost[new_node]:
        # todo
        ###################################################################
        # update the new node's distance
        self.ntree[""" """] =
        self.cost[""" """] =
        ###################################################################

# Re-Wire
for n in nlist:
    cost = self.cost[new_node] + self._distance(n, new_node)
    if cost < self.cost[n]:
        # todo
        ###################################################################
        # update the near node's distance
        self.ntree[""" """] =
        self.cost[""" """] =
        ###################################################################
```