八数码实验目的

1．熟悉状态空间表示法；

2．掌握深度优先、广度优先和等代价无信息搜索算法；

3．掌握启发式函数设计，实现面向实际问题的A*搜索算法；

本次实验课主要讲第二点的**深度优先（DFS）**和**广度优先（BFS）**。

状态空间表示

搜索问题是求解如何从开始状态变成目标状态。

表示问题的状态

使用python描述八数码问题的状态，即如何描述当前八数码的9个数字，以及它包含的一些**附加信息**。

穷举法找到求解过程

首先需要穷举所有可能的状态，然后找到一条合法的路径。

> 有时存在多条合法路径，怎么找到一条最优的路径？
>
> ### 如何加速求解？
>
> #### 算法角度
>
> 如果穷举所有的状态，并尝试组合这些状态形成一条合法路径，这会比较耗时。可以有目的地穷举。(Tips: A*)
>
> #### 程序执行角度（使用C++编写的可以忽略）
>
> 因为python是解释型语言，效率一般比较慢，可以考虑在编写代码是尽可能使用python原生的数据结构，或者使用一些加速库，如numpy+numba，cuda编程等。

DFS framework

可以使用**栈**数据结构/递归实现。

具体流程

1. 创建空栈，并把初始状态压入栈中。
2. 如果栈不为空，则取出栈顶存储的状态S；如果栈为空，执行第3步。
3. 扩展S状态，并压入栈中，继续第2步。
4. 完成DFS，算法退出。

In [ ]:

```
# Stack is the data structure implemented by yourself.
# Node 表示节点状态类

class DFSSolve:
    def __init__(self, start_node, end_node, max_depth):
        # Initialize the parameters.
        self.start_node = start_node
        self.end_node = end_node
        self.max_depth = max_depth
        self.cur_depth = 0
        self.path = []

    def node_expand(self, cur_node: Node) -> Optional[List[Node]]:
        # expand the current state.
        pass

    def dfs(self) -> Optional[List[Node]]:
        """
        Return a path.
        """
        st = Stack()
        st.push(self.start_node)
        visited:dict = {} # dict
        while not st.empty():
            cur_st = st.top()
            visited[cur_st] = True
            cur_st_list = node_expand()
            st.push(cur_st_list)  # not in visited
            # save the path
        return path
```

BFS framework

可以使用**队列**数据结构实现。

具体流程

1. 创建空队列，并令初始状态入队。
2. 如果队列不为空，则取出队首存储的状态S；如果队列为空，执行第3步。
3. 扩展S状态，并令它们入队中，继续第2步。
4. 完成BFS，算法退出。

In [ ]:

```
class BFSSolve:
    def __init__(self, start_node, end_node, max_depth):
        # same as DFSSolve
        pass

    def node_expand(self, cur_node: Node) -> Optional[List[Node]]:
        # same as DFSSolve
        pass

    def bfs(self) -> Optional[List[Node]]:
        """
        Return a path.
```

```
        """
        open = Queue()
        st.put(self.start_node)
        close:dict = {} # dict
        while not st.empty():
            cur_st = st.get()
            visited[cur_st] = True
            cur_st_list = node_expand()
            # delete the state in close
            if cur_st_list is None:
                continue
            st.put(cur_st_list)  # not in visited
            # save the path
        return path
```

Tips

1. 如何确定遍历终止条件?
2. 如何扩展状态空间的节点?
3. 如何防止遍历进入死循环?
4. 如何记录遍历的路径?
5. 可以统一一下遍历框架，思考它们不一样的地方。