Authors:

Ke Zhang, Yiyang Zhou

Run this project:

This project requires python3. Use `python main.py` to run.

In `main.py`, specify input file in `SRC_PATH` and output location in `DEST_PATH`.

Outputs:

Output1:

```
1 2 3
4 0 5
6 7 8

9 10 11
12 13 14
15 16 17

18 19 20
21 22 23
24 25 26

1 2 3
4 13 5
6 7 8

9 10 11
15 12 14
24 16 17

18 19 20
21 0 23
25 22 26

6
22
D W S D E N
6 6 6 6 6 6 6
```

Output2:

```
1 2 3
4 0 5
6 7 8

9 10 11
12 13 14
15 16 17

18 19 20
21 22 23
24 25 26

1 10 2
4 5 3
6 7 8

9 13 11
21 12 14
15 16 17

18 0 20
24 19 22
25 26 23

13
43
E N W D S W D S E E N W N
13 13 13 13 13 13 13 13 13 13 13 13 13 13
```

Output3:

```
1 2 3
4 0 5
6 7 8

9 10 11
12 13 14
15 16 17

18 19 20
21 22 23
24 25 26

0 2 3
1 7 14
6 8 5

12 9 10
4 13 11
21 16 17

18 19 20
22 25 23
15 24 26

16
58
S E N D N W W S D E S W U N U N
16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16
```

Source code:

main.py

```python
from agent import Agent
from task_environment import TaskEnvironment
SRC_PATH = "test/Input1.txt"
DEST_PATH = "outputs/Output1.txt"

def main():
    task_environment = TaskEnvironment(SRC_PATH)
    agent = Agent(task_environment)

    solutionFound = False
    while agent.frontier:
        node = agent.popFrontier()[1]
        if task_environment.isGoal(node):
            solutionFound = True
            break
        for child in agent.expand(node):
            agent.pushFrontier(child)
    if solutionFound:
        agent.output(DEST_PATH)
    else:
        print("No Solution!")

if __name__ == "__main__":
    main()
```

interface.py

```python
from typing import List, Tuple

Position = Tuple[int, int, int] # (x, y, z)
Cube = List[Position] # indices: num, values: position, length: CUBE_LENGTH
CUBE_LENGTH = 27
DIRECTION = {
  'UP': (0, 0, -1),
  'DOWN': (0, 0, 1),
  'NORTH': (0, -1, 0),
  'SOUTH': (0, 1, 0),
  'WEST': (-1, 0, 0),
  'EAST': (1, 0, 0)
}
```

agent.py

```python
import heapq
from typing import List, Optional
from interface import DIRECTION, Cube, Position


class Agent:
  def __init__(self, task_environment):
    self.task_environment = task_environment
    # keys: Cube state
    # values: [g value, h value, parent Cube, last move direction]
    self.node_history = {
      hash(tuple(task_environment.init_state)): [0,
task_environment.calculateHeuristic(task_environment.init_state), None, None]
    }
    self.node_count = 0
    self.frontier = [(0, task_environment.init_state)] # priority queue: (f=g+h, Cube)
  ## FUNCTIONS TO EXPAND NODES ##
  def act(self, cur_state: Cube, direction: str) -> Optional[Cube]:
    cur_state = cur_state[:]
    parent_hash = hash(tuple(cur_state))
    offsets = DIRECTION[direction]
    tmp_tuple = tuple((cur_state[0][i] + offsets[i] for i in range(len(offsets))))
    if any([e > 2 or e < 0 for e in tmp_tuple]):
      return None
    index = cur_state.index(tmp_tuple)
    cur_state[index], cur_state[0] = cur_state[0], cur_state[index]
    if hash(tuple(cur_state)) in self.node_history.keys():
      return None
    self.node_count += 1
    self.gValue = self.node_history[parent_hash][0] + 1
    self.node_history[hash(tuple(cur_state))] = [self.gValue,
self.task_environment.calculateHeuristic(cur_state), parent_hash, direction]

    return cur_state

  def expand(self, node):
    children = []
    for direction in DIRECTION.keys():
      child = self.act(node, direction)
      if child is not None:
        children.append(child)
    return children

  ## FRONTIER MANIPULATION
  def popFrontier(self) -> (int, Cube):
    return heapq.heappop(self.frontier)

  def pushFrontier(self, newNode: Cube):
    record = self.node_history[hash(tuple(newNode))]
    heapq.heappush(self.frontier, (record[0] + record[1], newNode))

  ## OUTPUT HANDLERS ##
  # calculate for the best solution: the number of steps, total nodes expnded, actions and f
values along the path
  def generateSolutions(self):
    solution_fValues, solution_actions = [], []
    cur_hash = hash(tuple(self.task_environment.goal_state))
    root_hash = hash(tuple(self.task_environment.init_state))
    while cur_hash != root_hash:
      gValue, hValue, cur_hash, last_move = self.node_history[cur_hash]
      solution_actions.append(last_move[0])
      solution_fValues.append(str(gValue + hValue))
    hValue = self.node_history[root_hash][1]
    solution_fValues.append(str(hValue))
    output = []
    output.append(str(len(solution_actions)))
    output.append(str(self.node_count))
    output.append(' '.join(solution_actions[::-1]))
    output.append(' '.join(solution_fValues[::-1]))
    return '\n'.join(output)

  # write the result to a file
  def output(self, path):
    f = open(path, "w")
    f.write(self.task_environment.input_file + '\n\n')
    f.write(self.generateSolutions())
    f.close()
```

task_environment.py

```
from typing import List, Tuple
from interface import CUBE_LENGTH, Cube, Position

class TaskEnvironment:
  def __init__(self, path):
    f = open(path, "r")
    self.input_file = f.read()
    f.close()

    self.init_state, self.goal_state = self.deserialize(self.input_file)

  # order = x + 3y + 9z
  def orderToPosition(self, order: int) -> Tuple[int, int, int]:
    x = order % 3
    order //= 3
    y = order % 3
    order //= 3
    z = order % 3
    return (x, y, z)

  def deserialize(self, input_file: str) -> tuple[Cube, Cube]:
    init_state = [-1] * 27
    goal_state = [-1] * 27
    nums = input_file.split()
    for i, num in enumerate(nums[:CUBE_LENGTH]):
      init_state[int(num)] = self.orderToPosition(i)
    for i, num in enumerate(nums[CUBE_LENGTH:]):
      goal_state[int(num)] = self.orderToPosition(i)
    return init_state, goal_state

  def manhattan_distance(self, pos1: Position, pos2: Position):
    distance = 0
    for i in range(len(pos1)):
      distance += abs(pos1[i] - pos2[i])
    return distance

  def calculateHeuristic(self, state: Cube) -> int:
    heuristic = 0
    for i in range(1, CUBE_LENGTH):
      heuristic += self.manhattan_distance(state[i], self.goal_state[i])
    return heuristic

  def isGoal(self, state: Cube) -> bool:
    return self.calculateHeuristic(state) == 0
```