Paris Metro

Github:

https://github.com/pjavier98/AI/tree/master/paris_metro

Archive: a_star_search.py

```
from state import *
from graph import *
from util import *
from queue import PriorityQueue

def a_star_search(graph, initial_state, goal):
  queue = PriorityQueue()
  queue.put([0, initial_state.station, initial_state])

  while(not queue.empty()):
    current_pair_state = queue.get()
    previous_cost = current_pair_state[0]
    current_state = current_pair_state[2]

    # Checking if the current state is the goal
    if current_state.final_state(goal):
      # Updating the cost in the goal
      current_state.update_total_cost(previous_cost)

      # Getting the best way from the begin to the goal
      graph.solution = [current_state]
      while current_state.dad:
        if (current_state == current_state.dad):
          break

        graph.solution.insert(0, current_state.dad)
        current_state = current_state.dad
      break

    # Mark as visited
    graph.visited_states[current_state.station] = 1

    for state in graph.adj_list[current_state.station]:
      if graph.visited_states[state.station] == 0:
        cost = previous_cost + state.real_dist + state.heurist_dist

        if not current_state.same_line(state, graph):
          cost += 2
          state.update_color(set(current_state.color) & set(state.color))
        else:
          state.update_color(current_state.color)
        state.dad = current_state
        state.update_total_cost(cost)
        queue.put([cost, state.station, state])

def main():
  print('\n#################################')
  print('# Welcome to the Metro of Paris #')
  print('#################################', end='\n\n')

  print('Select the departure station: [1-14]: ', end='')
  begin = read_stations()
  print('Select the departure station: [1-14]: ', end='')
  goal = read_stations()

  print('\nTicket: From {} to {}'.format(begin, goal), end='\n\n')

  # Read the distances
  distances_list = read_files('files/distances.txt')

  '''
    Blue: 0
    Yellow: 1
    Green: 2
    Red: 3
  '''
  # Read the colors of the station
  colors_list = read_files('files/colors.txt')

  # Creating the Graph
  graph = Graph()
  graph.adj_list = graph.generate_graph(distances_list, colors_list, goal)

  # Creating the initial state
  initial_state = State(int(begin), 0, 0, colors_list[begin - 1], begin)
  initial_state.dad = initial_state

  # Doing the A* search
  a_star_search(graph, initial_state, goal)

  # Prining the best way
  print_best_way(graph.solution)

main()
```

Archive: graph.py

```
from state import *

class Graph:

  def __init__(self):
    self.adj_list = None
    self.visited_states = [0] * 15
    self.solution = []


  def generate_graph(self, distances_list, colors_list, goal):
    graph = []
    graph.append([])
    input_adj_list = open('files/adj_list.txt', 'r')

    for i in range(14):
      adj_list = []
      for j in input_adj_list.readline().split():
        index = int(j) - 1

        real_dist = distances_list[i][index]
        # print('real distance: ' + str(real_dist))
        heurist_dist = distances_list[index][goal - 1]
        # print('heuristic distance: ' + str(heurist_dist))
        color = set(colors_list[index])

        state = State(int(j), real_dist, heurist_dist, color, i + 1)
        adj_list.append(state)

      graph.append(adj_list)

    input_adj_list.close()
    return graph

  def print_graph(self):
    for i in range(15):
      station = str(i)
      print(station + " -> ", end="")
      for j in self.adj_list[i]:
        print(str(j.station) + " ", end="")
      print()
```

Archive: state.py

```
class State:

    def __init__(self, station, heurist_dist, real_dist, color, dad):
        self.station = station
        self.heurist_dist = heurist_dist
        self.real_dist = real_dist
        self.total_cost = heurist_dist + real_dist
        self.color = color
        self.dad = None

    def __str__(self):
      # childrens_id = str_children()
      return ('station: {}\nheurist_dist: {}\nreal_dist: {}\ntotal_cost:
{}\ncolor: {}\ndad: {}\n'
            .format(self.station, self.heurist_dist, self.real_dist,
self.total_cost, self.color, self.dad.station
      ))

    def create_state():
      return State(station, heurist_dist, real_dist, color)

    def update_total_cost(self, total_cost):
      self.total_cost = total_cost

    def update_color(self, dad_color):
      self.color = dad_color

    def same_line(self, next_state, current_line):
      return set(self.color) & set(next_state.color)

    def final_state(self, goal):
      return self.station == goal
```

Archive: util.py

```
def read_stations():
  while True:
    try:
      number = int(input())
      if (number >= 1 and number <= 14):
        return number
    except:
      pass
    print("Invalid station, please choose again from [1-14]")

def read_files(path):
  fileDistances = open(path, 'r')

  inputFile = []
  for i in range(14):
      inputFile.append(list(map(int, fileDistances.readline().split())))

  fileDistances.close()
  return inputFile

def print_files(file):
  for i in file:
    print(i)

def print_best_way(solution):
  # for state in solution:
  #   print(str(state))
  previous_cost = 0
  flag = 0
  for state in solution:
    if flag == 0:
      print('Station {}'.format(state.station), end='')
      flag = 1
    else:
      print(' >> {} km >> Station {}'.format((state.total_cost - previous_cost), state.station), end='')
      previous_cost = state.total_cost
  print('\nTotal Cost: {} km'.format(previous_cost))
```