به نام خدا

نوید نصیری ۹۸۲۹۵۴۳
تکلیف اول هوش مصنوعی
استاد : دکتر پالهنگ

الگوریتم A*

در این سوال الگوریتم A* را پیاده سازی می کنیم که در زیر به توضیح این الگوریتم می‌پردازیم.

در ورودی ایدی دو شهر را که به ترتیب حروف الفبا مرتب شده‌اند را می‌گیریم. حال از شهر ابتدا شروع میکنیم و همسایه های آن را با استفاده از اکسلی که در آن اطلاعات مربوط به همسایه ها موجود است، استخراج می‌کنیم. سپس f = g + h همه همسایه ها را به دست می‌آوریم و از بین آنها کوچکترین را انتخاب می‌کنیم.
سپس شروع میکنیم از نقطه شروع به پیداکردن بهترین همسایه و پیش رفتن. یعنی هر بار که بهترین همسایه را پیدا کردیم، current_index را برابر ایندکس آن شهر قرار میدهیم.

برنامه حل به این روش در کل، از یک کلاس ایجاد شده است که دارای توابع زیر است که کار هر کدام نیز ذکر میشود.

تابع cal_f : که برای هر نود همسایه، مقدار تابع f را محاسبه می‌کند.

```python
def cal_f(self, current_index, next_index):
    h = self.distance_matrix[next_index][self.distance_index] # calcute h parameter of next point
    g = self.g + self.distance_matrix[current_index][next_index] # calcute g parameter of next point
    f = g + h
    return f, g
```

تابع cal_next_neighbor : همسایه های نود کارند را محاسبه میکند و برمیگرداند.

```python
def cal_next_neighbors(self, current_index):
    neighbors_indexes = []
    for i in range(len(self.neighbor_matrix[current_index])):
        if self.neighbor_matrix[current_index][i] == 1 and i not in self.reached_indexes:
            neighbors_indexes.append(i)
    return neighbors_indexes
```

تابع find_best_neighbor : که از بین همسایه ها، بهترین آنها را انتخاب میکند.

```python
def find_best_neighbor(self, current_index):
    all_neighbors = self.cal_next_neighbors(current_index)
    min_f = float('inf')
    for ind in all_neighbors:
        new_f, new_g = self.cal_f(current_index, ind)
        if min_f > new_f:
            min_f = new_f
            best_ind = ind
            final_g = new_g

    self.g = final_g
    return best_ind, min_f
```

تابع solve : که دکل پاسخ دادن به برنامه در آن اجرا می‌شود.

```python
def solve(self):
    current_index = self.start_index
    self.reached_indexes.append(current_index)
    while(current_index != self.distance_index):
        best_ind, best_f = self.find_best_neighbor(current_index)
        print(best_ind, best_f, self.g)
        self.reached_indexes.append(best_ind)
        current_index = best_ind
    minimum_distance = best_f # actully in final loop, best_f is same as minimum_distance
    return minimum_distance
```

الگوریتم BFS

در الگوریتم عرض نخست به این شکل عمل می کنیم که در هر مرحله، تمام حالت های موجود در آن ارتفاع مورد برسی قرار می‌گیرد و برسی می‌شود که آیا به حالت مقصد رسیده ایم یا خیر.
در یک لیست، همواره مسیر های موجود از مبدا تا حالت های آن ارتفاع را ذخیره می‌کنیم و در نهایت مسیری که به حالت هدف رسیده است را برمیگردانیم.

بخش های موجود در کد الگوریتم به صورت زیر است:

```
def cal_next_neighbors(self, current_index):
    neighbors_indexes = []
    for i in range(len(self.neighbor_matrix[current_index])):
        if self.neighbor_matrix[current_index][i] == 1 and i not in self.reached_indexes:
            neighbors_indexes.append(i)
    return neighbors_indexes

def cal_all_distance(self, index_list):
    if len(index_list) <= 1:
        return 0
    all_distance = 0
    for i in range(len(index_list)-1):
        all_distance += self.real_distance_matrix[index_list[i]][index_list[i+1]]
    return all_distance
```

تابع اول در کلاس BFS یافتن تمام همسایه های بعدی نود فعلی می‌باشد.
تابع بعدی در زمانی استفاده می‌شود که مسیر رسیدن تا مقصد یافته شده، و ما میخواهیم جمع مسافت این مسیر را بیابیم. درواقع خروجی تابع solve زیر را به این تابع می‌دهیم.

```python
def solve(self):
    current_index = self.start_index
    self.all_routes.append([current_index])
    if current_index == self.distance_index:
        return [current_index]
    while True:
        height_nodes = deepcopy(self.all_routes)
        for route in height_nodes:
            curr = route[-1]
            self.reached_indexes.append(curr)
            frontier = self.cal_next_neighbors(curr)
            #add all new routes to all_routes
            if len(frontier) != 0:
                base_route = deepcopy(route)
                self.all_routes.remove(route)
                for n in frontier:
                    new_route = deepcopy(base_route)
                    new_route.append(n)
                    self.all_routes.append(new_route)
                    if n == self.distance_index:
                        self.goal_list = new_route
                        return new_route
```

این تابع هم در نهایت bfs را پیاده سازی میکند. در هر مرحله، یک ارتفاع پایین تر میرود و همه نود های موجود در آن ارتفاع را بررسی میکند و همسایه های آن را می‌یابد و مسیر های قبلی را تکمیل تر می‌کند. در نهایت به محض یافتن مسیر مورد نظر، لیست شهر آیدی شهر های این مسیر از مبدا تا مقصد را برمی‌گرداند.