# DS4001-25SP-HW1：搜索

张语倢　PB24051037

2025 年 4 月 10 日

## 0　代码理解 [20%]

(a) [截图] 你的截图

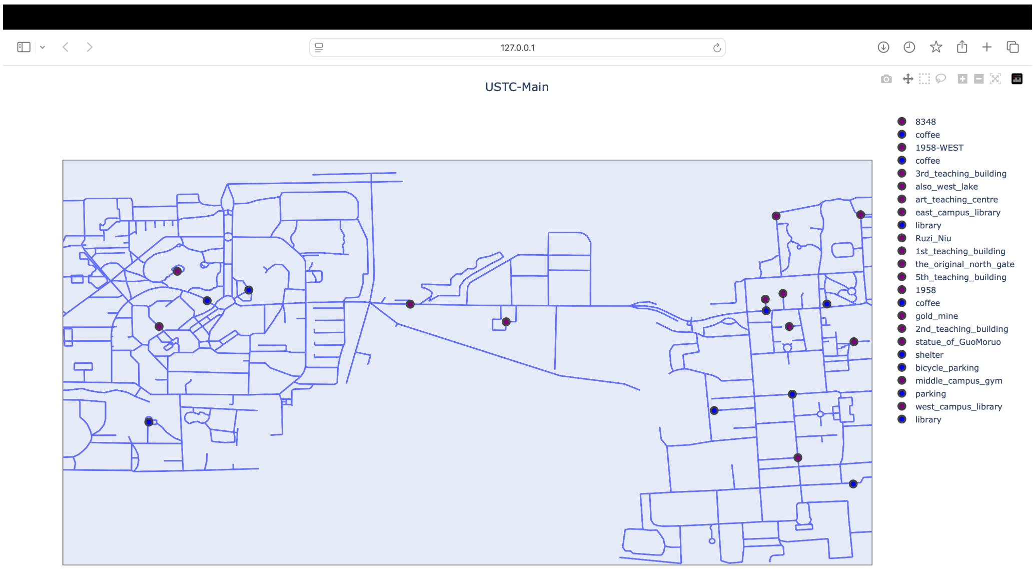


图 1: 配置环境运行结果

### 0.1　整体结构分析 [6%]

(b) [配对]

- (1):C
- (2):E
- (3):F
- (4):D
- (5):A

- (6):B
- (7):G

## 0.2 详细代码阅读 [10%=2%*5]

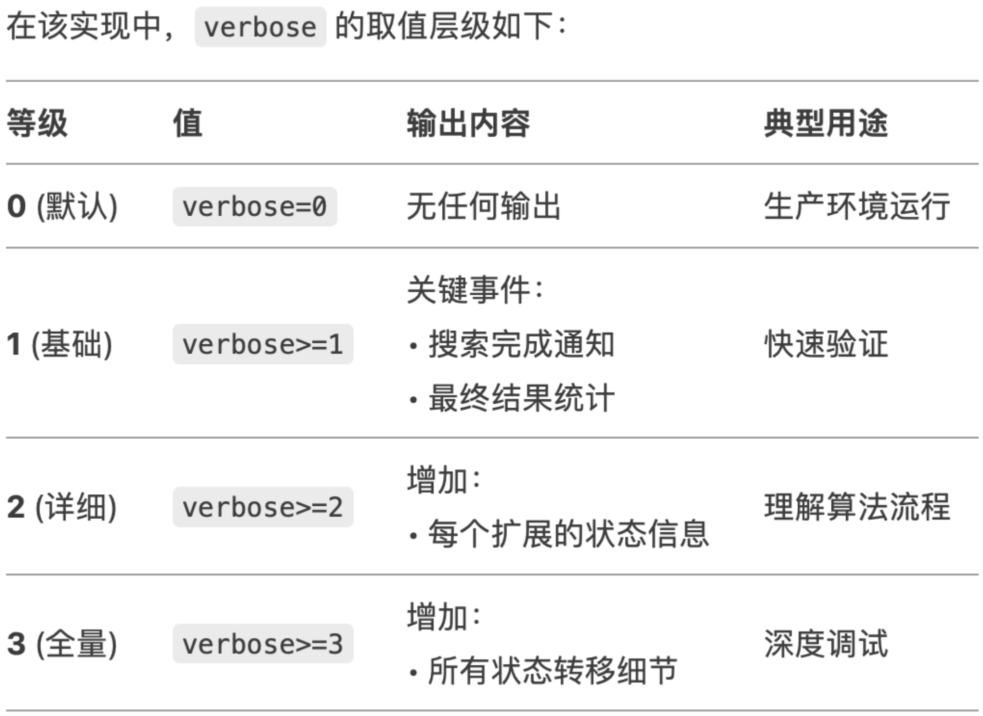
(c) [多选] ABE

(d) [单选] B

(e) [多选] BC

(f) [多选] ABCDE

(g) [简答] 在 Python 类设计中，self.verbose = verbose 这行代码的作用是为类实例添加一个控制输出详细程度的属性，verbose 是一个调试输出控制变量，用于控制算法运行时打印信息的详细程度。它的功能是通过整数值控制算法执行过程中的信息输出量。设计目的是调试时观察算法内部状态，教学时展示算法执行流程，生产环境中关闭非必要输出。



在该实现中，`verbose` 的取值层级如下：

| 等级 | 值 | 输出内容 | 典型用途 |
|---|---|---|---|
| **0** (默认) | `verbose=0` | 无任何输出 | 生产环境运行 |
| **1** (基础) | `verbose>=1` | 关键事件：<br>• 搜索完成通知<br>• 最终结果统计 | 快速验证 |
| **2** (详细) | `verbose>=2` | 增加：<br>• 每个扩展的状态信息 | 理解算法流程 |
| **3** (全量) | `verbose>=3` | 增加：<br>• 所有状态转移细节 | 深度调试 |

图 2: 截图

# 1 问题 1：查找最短路径 [29%]

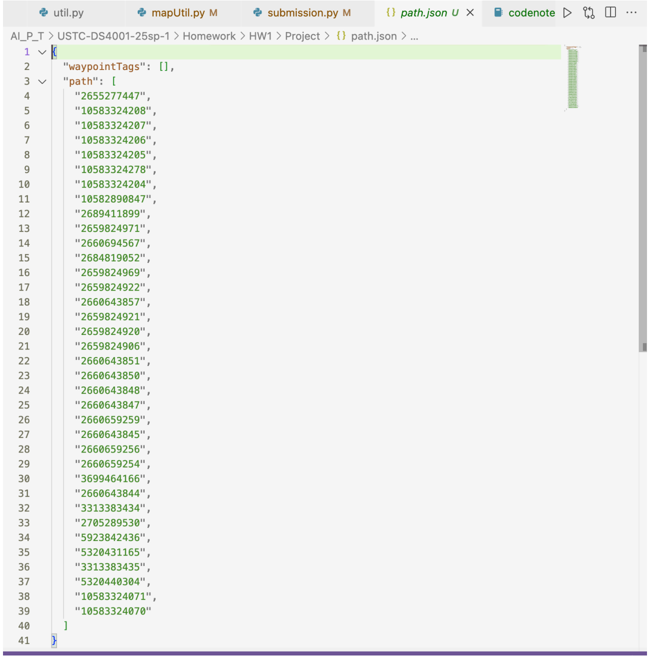
## 1.1 建模 [10%=6%+4%]

(a) [代码]

```

class ShortestPathProblem(SearchProblem):
    """
    Defines a search problem that corresponds to finding the shortest path
    from `startLocation` to any location with the specified `endTag`.
    """

    def __init__(self, startLocation: str, endTag: str, cityMap: CityMap):
        self.startLocation = startLocation
        self.endTag = endTag
        self.cityMap = cityMap



    def startState(self) -> State:
        # BEGIN_YOUR_CODE (our solution is 1 line of code, but don't worry if you deviate from this)
        return State(location=self.startLocation)
        raise NotImplementedError("Override me")
        # END_YOUR_CODE

    def isEnd(self, state: State) -> bool:
        # BEGIN_YOUR_CODE (our solution is 1 line of code, but don't worry if you deviate from this)
        return self.endTag in self.cityMap.tags.get(state.location, set())
        raise NotImplementedError("Override me")
        # END_YOUR_CODE

    def successorsAndCosts(self, state: State) -> List[Tuple[str, State, float]]:
        # BEGIN_YOUR_CODE (our solution is 4 lines of code, but don't worry if you deviate from this)
        # The successors of a state are all the locations that can be reached from
        neighbors = self.cityMap.distances.get(state.location, {})
        return [
            (neighbor, State(neighbor), distance)
            for neighbor, distance in neighbors.items()
        ]
        raise NotImplementedError("Override me")
        # END_YOUR_CODE
```

(b) [代码]

```
    def getUSTCShortestPathProblem() -> ShortestPathProblem:
        """
        Create your own search problem using the map of USTC, specifying your own
        `startLocation`/`endTag`.

        Run `python mapUtil.py > readableUSTCMap.txt` to dump a file with a list of
        locations and associated tags; you might find it useful to search for the following
        tag keys (amongst others):
            - `landmark=` - Hand-defined landmarks (from `data/USTC-landmarks.json`)
            - `amenity=`  - Various amenity types (e.g., "coffee", "food")
        """
        cityMap = createUSTCMap()

    # BEGIN_YOUR_CODE (our solution is 2 lines of code, but don't worry if you deviate from this)
    startLocation=locationFromTag(makeTag("landmark", "1958"), cityMap)
    endTag=makeTag("landmark", "1958-WEST")
    # END_YOUR_CODE
    return ShortestPathProblem(startLocation, endTag, cityMap)
```


util.py
mapUtil.py M
submission.py M
path.json U
codenote
AI_P_T > USTC-DS4001-25sp-1 > Homework > HW1 > Project > path.json > ...

```
{
  "waypointTags": [],
  "path": [
    "2655277447",
    "10583324208",
    "10583324207",
    "10583324206",
    "10583324205",
    "10583324278",
    "10583324204",
    "10582890847",
    "2689411899",
    "2659824971",
    "2660694567",
    "2684819052",
    "2659824969",
    "2659824922",
    "2660643857",
    "2659824921",
    "2659824920",
    "2659824906",
    "2660643851",
    "2660643850",
    "2660643848",
    "2660643847",
    "2660659259",
    "2660643845",
    "2660659256",
    "2660659254",
    "3699464166",
    "2660643844",
    "3313383434",
    "2705289530",
    "5923842436",
    "5320431165",
    "3313383435",
    "5320440304",
    "10583324071",
    "10583324070"
  ]
}
```

图 3: label 路线截图

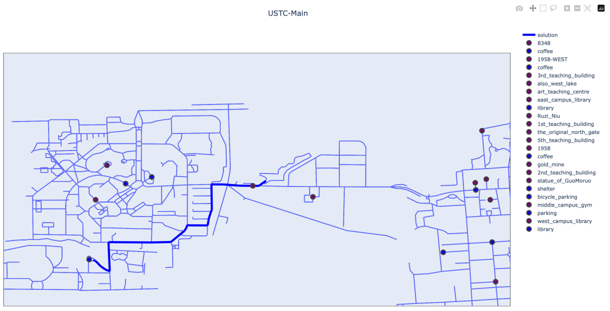


图 4: 地图路线截图

## 1.2 算法 [19%=6%+5%+2%+6%]

(c) [简答] 使用归纳法证明。即正当 k=0 时，不等式成立；假设 k=n 时满足，$\forall u, d_s(u) \leq d_s^{(n)}(u)$ 且有题目可知，$d_s^{(n+1)}(u) = \arg\min_v\{d_s^n(v) + \omega_{vu}\}\$$ $d_s(u) \leq \arg\min\{d_s(v) + \omega_{vu}\} \leq \arg\min\{d_s^{(n)}(v) + \omega_{vu}\} = d_s^{(n+1)}(u)$ 又因为每次迭代都取最小，因此显然可以知道

$$d_s^{n+1}(u) \leq d_s^n(u)$$

因此可以证明，

$$d_s(u) \leq d_s^{n+1}(u) \leq d_s^n(u)$$

**[简答]** 使用归纳法证明 $\forall k, d_s^{(k)}(v_k) = d_s(v_k)$ 正确。当 k=0 时，$d_s^(0)(v_0) = d_s^(0)(s) = d_s(v_0)$ 成立。

假设当 k=n 时，$d_s^(n)(v_n) = d_s(v_n)$ 成立，则当 k=n+1 时，s 到 $v_{n+1}$ 的最短路径，会经过某个已知的 $v_i$ 到未知的 $v_{n+1}$

$$d_s^{(n+1)}(v_{n+1}) = \arg\min_v\{d_s^{(n)}(v) + \omega_{vu}\} = \arg\min_v\{d_s(v) + \omega_{vu}\} \leq d_s(v_i) + \omega_{v_i v_{n+1}} = d_s(v_{n+1})$$

又因为 (c) 题中可知，$d_s^{(n+1)}(v) \leq d_s(v)$，因此 $d_s^{(n+1)}(v) = d_s(v)$ 得证

(e) **[判断]** 不能

(f) **[简答]** （1）Dijkstra 算法条件为非负边权。当把一个节点选入了集合时，即此时的 $v_k$ 已确定，意味着已经找到了从源点到该点的最短距离，假如存在负边权，后续则会出现经过负边权的路径找到结点，使得 $d_s(v_k)$ 更小，可能会出现得出的距离加上负边权后比已经得到的距离要短，在最终寻找路径时无法回溯。即 $\arg\min_v\{d_s(v) + \omega_{vu}\}d_s(v_i) + \omega_{v_i v_{n+1}} \leq \arg\min_v\{d_s(v) + \omega_{vu}\}$，则无法通过（d）证出 dijkstra 算法正确性

(2)

```

    set the distance to the source vetex as 0,set distance to all other vertices as infinity
    relaxation, for each edge(u,v) in the graph:
        if distance[u]+weight(u,v)<distance[v]
            update distance[v]=distance[u]+weight(u,v)
    after all edges are processed, the distance array will contain the shortest distance from the
      source vertex to all other vertices

    perform one more relaxation for each edge(u,v) in the graph:
        if distance[u]+weight(u,v)<distance[v]
            return false

    return distance
```

# 2 问题 2：查找带无序途径点的最短路径 [17%]

## 2.1 建模 [10%=6%+4%]

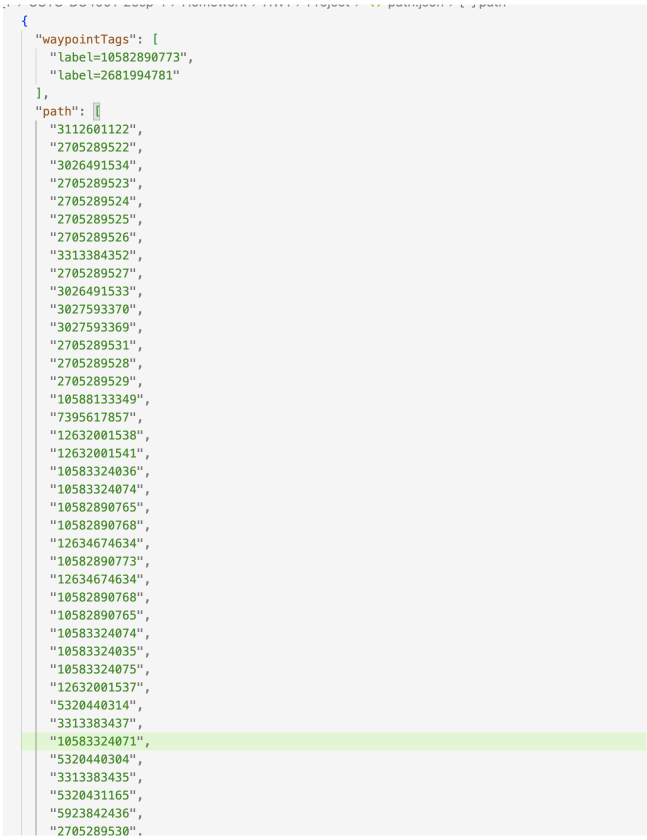
(a) **[代码]**

```

class WaypointsShortestPathProblem(SearchProblem):
    """
    Defines a search problem that corresponds to finding the shortest path from
    `startLocation` to any location with the specified `endTag` such that the path also
    traverses locations that cover the set of tags in `waypointTags`.

    Think carefully about what `memory` representation your States should have!
    """
    def __init__(
        self, startLocation: str, waypointTags: List[str], endTag: str, cityMap: CityMap
    ):
```

```python
        self.startLocation = startLocation
        self.endTag = endTag
        self.cityMap = cityMap

        # We want waypointTags to be consistent/canonical (sorted) and hashable (tuple)
        self.waypointTags = tuple(sorted(waypointTags))

    def startState(self) -> State:
        # BEGIN_YOUR_CODE (our solution is 1 line of code, but don't worry if you deviate from this)
        return State(location=self.startLocation, memory=0)
        raise NotImplementedError("Override me")
        # END_YOUR_CODE

    def isEnd(self, state: State) -> bool:
        # BEGIN_YOUR_CODE (our solution is 1 lines of code, but don't worry if you deviate from this)

        return(
            self.endTag in self.cityMap.tags.get(state.location,set()) and
            state.memory == (1<<len(self.waypointTags))-1
        )
        raise NotImplementedError("Override me")
        # END_YOUR_CODE

    def successorsAndCosts(self, state: State) -> List[Tuple[str, State, float]]:
        # BEGIN_YOUR_CODE (our solution is 13 lines of code, but don't worry if you deviate from this)
        successors = []

        for neighbor, distance in self.cityMap.distances.get(state.location,{}).items():
            newMemory = state.memory
            for i, tag in enumerate(self.waypointTags):
                if tag in self.cityMap.tags.get(neighbor, set()):
                    newMemory |= (1 << i)
            successors.append(
                (neighbor, State(neighbor,newMemory), distance)
            )
        return successors

        raise NotImplementedError("Override me")
    # END_YOUR_CODE
```

(b) [代码]

```python

def getUSTCWaypointsShortestPathProblem() -> WaypointsShortestPathProblem:
    """
    Create your own search problem using the map of USTC, specifying your own
    `startLocation`/`waypointTags`/`endTag`.

    Similar to Problem 1b, use `readableUSTCMap.txt` to identify potential
    locations and tags.
    """
    cityMap = createUSTCMap()
    # BEGIN_YOUR_CODE (our solution is 3 lines of code, but don't worry if you deviate from this)
    startLocation=locationFromTag(makeTag("landmark", "1958"), cityMap)
    endTag=makeTag("landmark", "1958-WEST")
    waypointTags=[makeTag("label",locationFromTag(makeTag("landmark","middle_campus_gym"),cityMap)), makeTag("labe

    #raise NotImplementedError("Override me")
    # END_YOUR_CODE
    return WaypointsShortestPathProblem(startLocation, waypointTags, endTag, cityMap)
```



```
{
  "waypointTags": [
    "label=10582890773",
    "label=2681994781"
  ],
  "path": [
    "3112601122",
    "2705289522",
    "3026491534",
    "2705289523",
    "2705289524",
    "2705289525",
    "2705289526",
    "3313384352",
    "2705289527",
    "3026491533",
    "3027593370",
    "3027593369",
    "2705289531",
    "2705289528",
    "2705289529",
    "10588133349",
    "7395617857",
    "12632001538",
    "12632001541",
    "10583324036",
    "10583324074",
    "10582890765",
    "10582890768",
    "12634674634",
    "10582890773",
    "12634674634",
    "10582890768",
    "10582890765",
    "10583324074",
    "10583324035",
    "10583324075",
    "12632001537",
    "5320440314",
    "3313383437",
    "10583324071",
    "5320440304",
    "3313383435",
    "5320431165",
    "5923842436",
    "2705289530",
```

图 5: label 路线截图

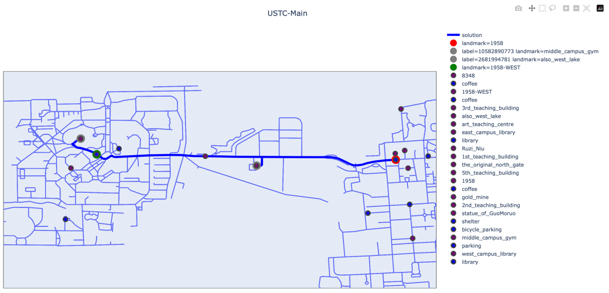


图 6: 地图路线截图

### 2.2 算法 [7%=2%+5%]

(c) **[判断]** 不会

(d) **[简答]** 小李的说法不成立，小李直接将 Dijkstra 算法应用于原始图像，即有向无环，也就是没有返回路径，在这种情况下，小李的说法正确即 $d_s(u) \leq d_s^{(k)}(u)$ 不成立。但是在有向图中，可能存在返回路径的情况，即从某个节点出发，经过一条边回到原来的节点，这种情况下，Dijkstra 算法可能会选择一条较短的路径，而不是最短路径，因此小李的说法不成立。通过 State 类中的 memory 记录期望达到途径点与已达到途径点。

小李还忽略了搜索结束判断的第二个条件，即 set(waypointTags).issubset(visitedWaypoints)，在判断 isEnd() 方法，我们设置了 waypointTags 和 visitedWaypoints 两个变量，waypointTags 表示所有途径点的标签，visitedWaypoints 表示已经访问过的途径点的集合。通过判断 visitedWaypoints 是否包含所有的 waypointTags 来判断是否经过了所有期望访问的途径点，同时判断是否达到 endTag，因此，在小李提出的情形中，程序并不会判定该路径搜索已结束，而是继续探索可能最短路径直至所有期望途径点记录进了 visitedWaypoints。

此外，从 B 出发访问 X，再出发访问 B，并不是回溯行为，而是包含在路径探索中。当第一次到达 B 时，waypointTags={'X','Y','Z'}，而 visitedWaypoints={'Y','Z'}，waypointTags 并不是 visitedWaypoints 的子集，此时路径探索尚未结束，因此并非回溯过程。

## 3 问题 3：使用 A* 加快搜索速度 [32%]

### 3.1 将 UCS 转化为 A*[4%]

(a) **[代码]**

```python
    # Your Code
def aStarReduction(problem: SearchProblem, heuristic: Heuristic) -> SearchProblem:
    class NewSearchProblem(SearchProblem):
        def __init__(self):
            # BEGIN_YOUR_CODE (our solution is 3 line of code, but don't worry if you deviate from this)
            self.problem = problem
            self.heuristic = heuristic
            #raise NotImplementedError("Override me")
            self.startLocation=self.problem.startState().location
            self.endTag=self.problem.endTag
            self.cityMap=self.problem.cityMap
            # END_YOUR_CODE

        def startState(self) -> State:
            # BEGIN_YOUR_CODE (our solution is 1 line of code, but don't worry if you deviate from this)
            return self.problem.startState()
            raise NotImplementedError("Override me")
            # END_YOUR_CODE

        def isEnd(self, state: State) -> bool:
            # BEGIN_YOUR_CODE (our solution is 1 line of code, but don't worry if you deviate from this)
            return self.problem.isEnd(state)
            raise NotImplementedError("Override me")
            # END_YOUR_CODE

        def successorsAndCosts(self, state: State) -> List[Tuple[str, State, float]]:
            # BEGIN_YOUR_CODE (our solution is 7 lines of code, but don't worry if you deviate from this)
            successors = self.problem.successorsAndCosts(state)
            newSuccessors = []
            h_current=self.heuristic.evaluate(state)
            for action, newState, cost in successors:
                #修改代价
                h_new=self.heuristic.evaluate(newState)
                newCost=cost+h_new - h_current
                newSuccessors.append((action, newState, newCost))

            return newSuccessors
            raise NotImplementedError("Override me")
            # END_YOUR_CODE

    return NewSearchProblem()
```

### 3.2 实现启发式函数 [18%=3%+6%+3%+6%]

(b) [简答]

为证明启发函数的一致性，即证明 $\forall s, a, Cost'(s,a) = Cost(s,a) + h(Succ(s,a)) - h(s) \geq 0$，且 $h(s_{end}) = 0$

对于第一个不等式，由于 $Cost(s,a), h(Sicc(s,a)), h(s) \geq 0$，由三角不等式可轻松得知，$Cost(s,a) + h(Sicc(s,a)) \geq h(s)$，第一个条件得证.

对于第二个等式，由 heauristic 的定义，“A heuristic h(s) is any extimate of FutureCost(s)”，在终点处，搜索结束，预估未来代价为 0，即 $0 \leq FutureCost(s_end) \leq d_{send}(s_{end})$

(c) [代码]

```python
    # Your Code
class StraightLineHeuristic(Heuristic):
    """
    Estimate the cost between locations as the straight-line distance.
        > Hint: you might consider using `computeDistance` defined in `mapUtil.py`
    """
    def __init__(self, endTag: str, cityMap: CityMap):
        self.endTag = endTag
        self.cityMap = cityMap

        # Precompute
        # BEGIN_YOUR_CODE (our solution is 4 lines of code, but don't worry if you deviate from this)
        '''end_label=locationFromTag(self.endTag, self.cityMap)
        self.end_location = self.cityMap.geoLocations[end_label]'''
        self.end_locations=[
            location for location,tags in self.cityMap.tags.items()
            if endTag in tags and location in self.cityMap.geoLocations
        ]
        if not self.end_locations:
            raise ValueError(f"No locations found for tag: {self.endTag}")
        self.end_geolocations = [self.cityMap.geoLocations[location] for location in self.end_locations]
        #raise NotImplementedError("Override me")
        # END_YOUR_CODE

    def evaluate(self, state: State) -> float:
        # BEGIN_YOUR_CODE (our solution is 6 lines of code, but don't worry if you deviate from this)
        current_geolocation = self.cityMap.geoLocations[state.location]
        # 判断当前位置是否在地图上
        if current_geolocation is None or self.end_geolocations is None:
            return float("inf")
        return min(
            computeDistance(current_geolocation, end_geolocation) for end_geolocation in self.end_geolocations
        )
        raise NotImplementedError("Override me")
        # END_YOUR_CODE
```

对于问题 2，我们使用不带途径点的最短路径长度作为启发式函数。

(d) **[简答]** 你的答案为证明一致性，即证明

$$\forall s, a \quad \text{Cost}(s,a) = \text{Cost}(s,a) + h(\text{Succ}(s,a)) - h(s) \geq 0$$

且 $h(\text{Succ}_{end}) = 0$。

从 NoWaypointsHeuristic 类的定义可知，

$$h(s) = \text{self.all\_distances.get}(s.\text{location}, \text{float('inf')})$$

其中 all_distances 中存储的是从每个 end_location 出发，通过 UCS 搜索得到的各 state 到 end_location 的最短路径距离。易知 $h(s_{end}) = 0$。

由于 $h(s)$ 即为 $s$ 到 end 的最短路径估计，则 $\text{Cost}(s,a) + h(\text{Succ}(s,a))$ 表示从 $s$ 经过动作 $a$ 到达 $\text{Succ}(s,a)$，再沿最短路径到 end 的总代价。根据最短路径的定义，必然有：

$$\text{Cost}(s,a) + h(\text{Succ}(s,a)) \geq h(s)$$

因此可得：

$$\text{Cost}(s,a) = \text{Cost}(s,a) + h(\text{Succ}(s,a)) - h(s) \geq 0$$

(e) **[代码]**

```python
    # Your Code
class NoWaypointsHeuristic(Heuristic):
    """
    Returns the minimum distance from `startLocation` to any location with `endTag`,
    ignoring all waypoints.
    """
    def __init__(self, endTag: str, cityMap: CityMap):
        # Precompute
        # BEGIN_YOUR_CODE (our solution is 14 lines of code, but don't worry if you deviate from this)
        self.endTag = endTag
        self.cityMap = cityMap

        self.end_locations=[
            location for location, tags in self.cityMap.tags.items()
            if self.endTag in tags and location in self.cityMap.geoLocations
        ]

        if not self.end_locations:
            raise ValueError(f"No locations found for tag: {self.endTag}")


        #initialize search algorithm and distance cache
        self.ucs=UniformCostSearch(verbose=0)
        self.distance_cache={}

        self.all_distances={}
        for end_location in self.end_locations:
            problem=ShortestPathProblem(end_location, None , self.cityMap)
            self.ucs.solve(problem)

            for loc_state, cost in self.ucs.pastCosts.items():
                if loc_state.location not in self.all_distances or cost<self.all_distances[loc_state.location]:
                    self.all_distances[loc_state.location] = cost


        #raise NotImplementedError("Override me")
        # END_YOUR_CODE

    def evaluate(self, state: State) -> float:
        # BEGIN_YOUR_CODE (our solution is 1 line of code, but don't worry if you deviate from this)

        return self.all_distances.get(state.location, float('inf'))

        raise NotImplementedError("Override me")
        # END_YOUR_CODE
```

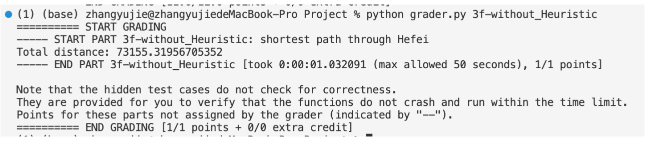
### 3.3 利用合肥市地图对比运行时间 [10%=4%+6%]

(f) [代码]

```python
    # Your Code
def getHefeiShortestPathProblem(cityMap: CityMap) -> ShortestPathProblem:
    """
    Create a search problem using the map of Hefei
    """
    startLocation=locationFromTag(makeTag("landmark", "USTC"), cityMap)
    endTag=makeTag("landmark", "Chaohu")
    # BEGIN_YOUR_CODE (our solution is 1 lines of code, but don't worry if you deviate from this)
    return ShortestPathProblem(startLocation, endTag, cityMap)
    raise NotImplementedError("Override me")
    # END_YOUR_CODE

def getHefeiShortestPathProblem_withHeuristic(cityMap: CityMap) -> ShortestPathProblem:
    """
    Create a search problem with Heuristic using the map of Hefei
    """
    startLocation=locationFromTag(makeTag("landmark", "USTC"), cityMap)
    endTag=makeTag("landmark", "Chaohu")
    # BEGIN_YOUR_CODE (our solution is 2 lines of code, but don't worry if you deviate from this)
    endTag=makeTag("label",locationFromTag(endTag, cityMap))
    return aStarReduction(
        ShortestPathProblem(startLocation, endTag, cityMap),
        NoWaypointsHeuristic(endTag, cityMap)
    )
    raise NotImplementedError("Override me")
    # END_YOUR_CODE
```

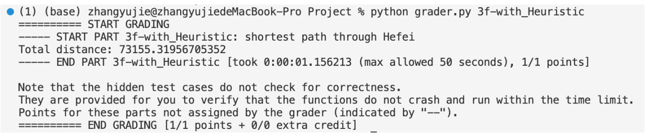
可发现，without_Heuristic 的运行时间为 0 : 00 : 01.032091，而 with_Heuristic 的运行时间为 0 : 00 : 01.156213，with_Heuristic 的运行时间更长。原因在于，with_Heuristic 中使用了启发式函数来估计从当前状态到目标状态的代价，这个估计需要额外的计算时间，因此导致了运行时间的增加。

截图：



```
● (1) (base) zhangyujie@zhangyujiedeMacBook-Pro Project % python grader.py 3f-without_Heuristic
  ========== START GRADING
  ----- START PART 3f-without_Heuristic: shortest path through Hefei
  Total distance: 73155.31956705352
  ----- END PART 3f-without_Heuristic [took 0:00:01.032091 (max allowed 50 seconds), 1/1 points]

  Note that the hidden test cases do not check for correctness.
  They are provided for you to verify that the functions do not crash and run within the time limit.
  Points for these parts not assigned by the grader (indicated by "--").
  ========== END GRADING [1/1 points + 0/0 extra credit]
```

图 7: 3f-without_Heuristic 运行结果时间为 0 : 00 : 01.032091



```
● (1) (base) zhangyujie@zhangyujiedeMacBook-Pro Project % python grader.py 3f-with_Heuristic
  ========== START GRADING
  ----- START PART 3f-with_Heuristic: shortest path through Hefei
  Total distance: 73155.31956705352
  ----- END PART 3f-with_Heuristic [took 0:00:01.156213 (max allowed 50 seconds), 1/1 points]

  Note that the hidden test cases do not check for correctness.
  They are provided for you to verify that the functions do not crash and run within the time limit.
  Points for these parts not assigned by the grader (indicated by "--").
  ========== END GRADING [1/1 points + 0/0 extra credit]  _
```

图 8: 3f-with_Heuristic 运行结果时间为 0 : 00 : 01.156213

(g) [简答] 你的答案

在合肥市地图上运行，location 数据可能过多，每次搜索需要处理的节点很多，导致 UCS 或 A* 算法运行时间较长，内存消耗高计算速度慢，在启发性函数中，使用了 UCS 算法来计算每个目标节点到所有节点的距离，这个过程需要遍历整个图，因此会消耗大量的内存和计算时间。

方案：对地图节点数据先进行预处理，将合肥市按照下辖区和县进行划分，分为瑶海区、庐阳区、蜀山区和包河区，肥东县、肥西县、长丰县和庐江县，还有一个巢湖市。将所有的区域抽象为一个集合点区域。如果是对于 UCS 算法，首先明确起点和终点，及其所在的区域，对于对应同一个 *endTag* 的多个 *end_locations*，遍历地区的所有节点，找到所有 *end_location* 所在的区域。如果在同一区域内查找，只需将 *cityMap* 定义为该区域的地图，减小搜索范围。

如果跨区域查询，即先通过宏观 ucs 搜索，找到 $startLocaiton$ 到 $end\_location$ 所在区域的边界的最短路径，在在该区域内进行局部的 ucs 搜索，找到该边界点到 $end\_location$ 的最短路径，两个最短路径的总和便是该 $startLocation$ 到该 $end\_Location$ 的最短距离。

再对所有符合 $endTag$ 的 $end\_locaiton$ 进行遍历计算最短距离后，找到这几个最短路径中的最小值，作为最终的最短路径。

## 体验反馈 [2%]

(a) [**必做**] 你的答案

我做了估计有十几个小时 [哭泣]，主要是没做过这种项目类型导致找接口比较费劲，也多次因为接口没搞对导致运行报错或运行的答案不对，最后通过 print 调试很久并且求助助教和大模型。花最长时间的可能是在 class NoWaypointsHeuristic(Heuristic) 里面的时间，改了多版，发现仍然无法输出正确答案，后面发现是因为 pastCosts 的键 State 还包含了 memory 部分，当存在 waypoints 的时候后面调用 state 中 memory 会不一致，所以在存储距离的时候应该直接用 location 存储。

(b) [**选做**] 你的答案

对于第一次做这种项目类型作业的我来说确实有点困难了...... 开始理解代码的时候确实花了不少时间。上课的时候感觉内容有点抽象，如果可以的话想问问助教学长学姐能否在实验课的时候给几个代码的例子！感恩！