第一次作业（搜索问题）

崔士强　PB22151743

2024 年 4 月 7 日

本次作业需独立完成，不允许任何形式的抄袭行为，如被发现会有相应惩罚。在上方修改你的姓名学号，说明你同意本规定。

问题 0：引入（30 分）

1. 最短路径问题（12 分）

a. 回答问题（2 分）

$$\mathcal{D}_1 \circ d_s(v_2) = min_{v_1}\{d_s(v) + w_{v_1v_2}\} = 10$$

b. 证明（2 分）

在

$$d_s(v_k) = min_{i\in[n]}\{d_s(v_i) + w_{v_kv_i}\}$$

中，若 $i \geq k$，则有 $LHS > RHS$，与 $d_s(v_k) \leq d_s(v_{k+1})$ 矛盾. 则 $i < k$，$RHS = \mathcal{D}_k \circ d_s(v_k)$.

c. 证明（4 分）

定义

$$d(u) = min_{v\in S}\{d_s(v) + w_{vu}\}$$

其中 S 为已被选择的顶点集. 每轮更新即为:

$$min_{u\in V-S}\{\mathcal{D}_k \circ d(u)\} = min_{u\in V-S}\{d(u)\} = d_s(v_k)$$

也就是说，第 k 次执行算子一定能找到距源点第 k 近的点以及它到源点到最短距离，执行 n 次后整张图的搜索便可完成.

d. 回答问题（2 分）

$\mathcal{D}_k$ 表示在距源点前 k 近的点（而不是所有点）当中寻找到目标点距离最短的路径. 从而免去了对其余点的搜索，本质上是因为剩余的这些点到源点距离比目标点更长，最短路径不可能经过这些点.

e. 证明（2 分）

如果

$$d_s(u) \neq min_{v\in V}\{d_s(v) + w_{vu}\}$$

即 $\exists v_0$，$s.t.\ d_s(v_0) + w_{v_0 u} < d_s(u)$. 此时从源点到 v_0 的最短路径加上边 $v_0 u$ 为 u 的一条更短的路径，所以上面的假设不成立.

2.A* 算法，判断对错并说明原因（10 分）

a 正确. 此时选择结点的依据即为 $d(u)$，与 Dijkstra 算法等价.

b 正确. $h(u)$ 可能影响被选择的节点，从而影响对邻接节点的更新.

c 错误. 从 Algorithm 1 Line 4 可以看出，所有的 $d(u)$ 归根结底都是多次增加某条边权值的结果，这些边首尾相连的路径长度即为 $d(u)$

d 正确. 算法需要遍历所有顶点，在最坏情况下，每条边至少会被考察一次来更新顶点的 d 值。如果使用最小堆作为优先队列的数据结构，那么对于 m 条边，每次插入和删除的操作是 $O(\log n)$。

e 错误. A^* 算法与 Dijkstra 算法等价当且仅当 $h(u) = 0$，然而当 $h(u)$ 的大小排序与 $d(u)$ 相同时也满足条件.

3. 网格城市（8 分）

a. 回答问题（8 分）

先沿 y 轴走到 $(0, n)$，再沿水平方向走到 (m, n)，成本为 $n + m + \frac{(1+m)m}{2}$. 并且没有其他最短路径.

问题 1：查找最短路径（12 分）

a. 代码实现 ShortestPathProblem 部分（8 分）

```python
class ShortestPathProblem(SearchProblem):
    """The illustration and __init__ part is ommited here."""

    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(self.startLocation)
        # 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[state.location]
        # 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)
        return_list = []
        for succ_location in self.cityMap.distances[state.location]:
            cost = self.cityMap.distances[State.location][succ_location]
            return_list.append((succ_location, State(succ_location), cost))
        return return_list
        # END_YOUR_CODE
```

b. 路线可视化（4 分）

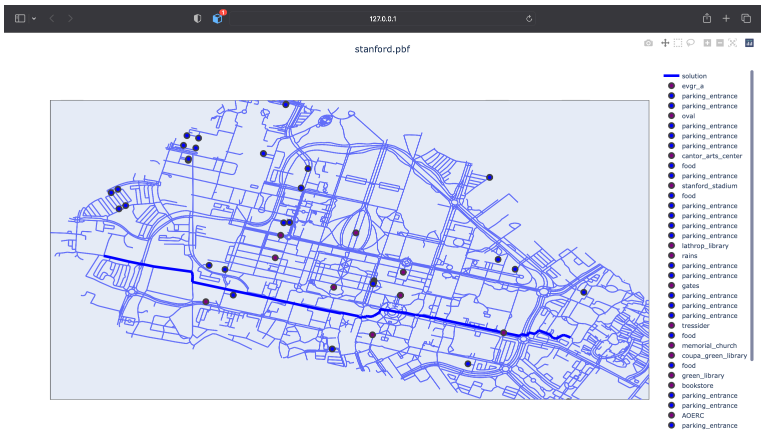

图 1: startLocation='65565059', endTag="label=9384099472"

可以看到这个路线穿过了大半个校园. 这个系统对校园旅行很有帮助，可以找到起始点到任意符合要求的点的最短路径.

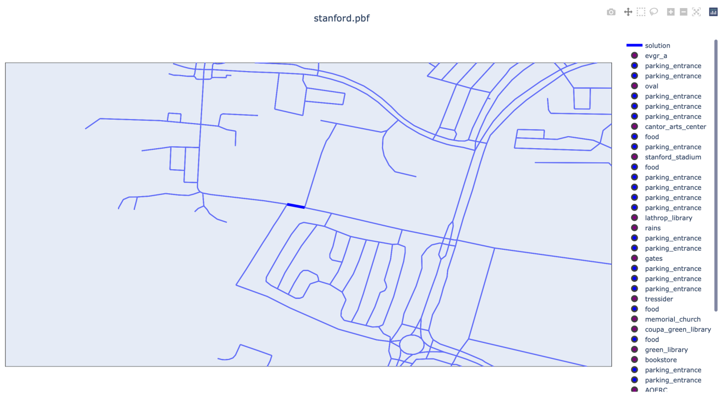

图 2: startLocation='65565059', endTag="label=65422311"

可以看到这个路径非常短，原因并不是建模不正确，而是符合条件的点距离起点很近.

问题 2：查找带无序途径点的最短路径（20 分）

a. 代码实现 WaypointsShortestPathProblem 部分（12 分）

```python
class WaypointsShortestPathProblem(SearchProblem):
    """The illustration and __init___ part is ommited here."""

    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=self.waypointTags)
        # END_YOUR_CODE

    def isEnd(self, state: State) -> bool:
        # BEGIN_YOUR_CODE (our solution is 5 lines of code, but don't worry if you deviate from this)
        return (len(state.memory) == 0) and (self.endTag in self.cityMap.tags[state.location])
        # END_YOUR_CODE

    def successorsAndCosts(self, state: State) -> List[Tuple[str, State, float]]:
        # BEGIN_YOUR_CODE (our solution is 17 lines of code, but don't worry if you deviate from
            this)
        return_list = []
        succ_memory = list(state.memory)
        for tag in self.cityMap.tags[state.location]:
            if(tag in state.memory):
                succ_memory.remove(tag)
        for succ_location in self.cityMap.distances[state.location]:
            cost = self.cityMap.distances[state.location][succ_location]
            return_list.append((succ_location, State(location=succ_location,
                memory=tuple(sorted(succ_memory))), cost))
        return return_list
        # END_YOUR_CODE
```

b. 回答问题（4 分）

$n \times 3^k$ 对于每一个确定的点，每个标签都有 3 种情况：1). 需要经过但还未经过. 2). 不需要经过 3). 需要经过且已经经过

c. 可视化（4 分）

下面的两个路径中均有 waypointTags=['amenity=food', 'label=65559196', 'label=6524008724'].

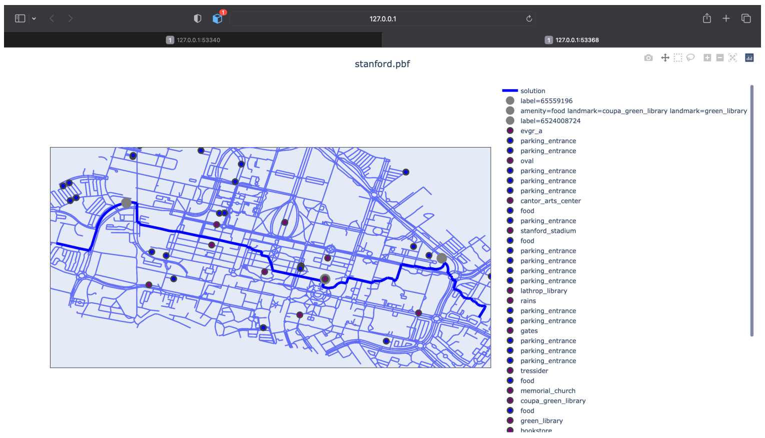

图 3: startLocation='65565059', endTag="label=9384099472"

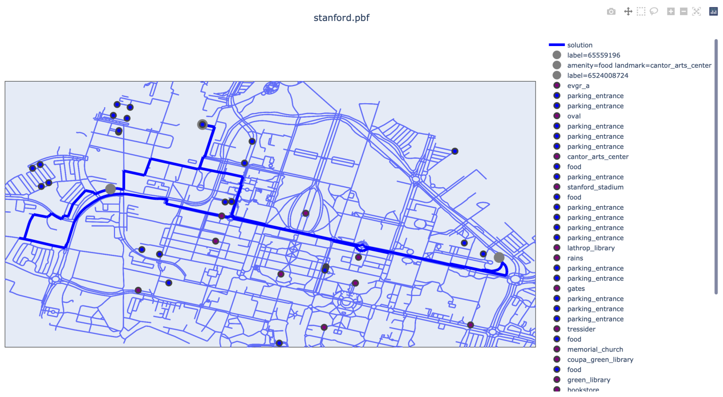

图 4: startLocation='65565059', endTag="label=65422311"

可以看到在两个路径中，设置途径点均对路径产生显著影响，使得路径可以探索更多区域.

问题 3：使用 A* 算法加快搜索速度（28 分）

a. 代码实现 aStarReduction 的 NewSearchProblem 部分（8 分）

```python
def aStarReduction(problem: SearchProblem, heuristic: Heuristic) -> SearchProblem:
    class NewSearchProblem(SearchProblem):
        def startState(self) -> State:
            # BEGIN_YOUR_CODE (our solution is 1 line of code, but don't worry if you deviate from
                this)
            return problem.startState()
            # END_YOUR_CODE

        def isEnd(self, state: State) -> bool:
```

```python
        # BEGIN_YOUR_CODE (our solution is 1 line of code, but don't worry if you deviate from
            this)
        return problem.isEnd(state)
        # END_YOUR_CODE

    def successorsAndCosts(self, state: State) -> List[Tuple[str, State, float]]:
        # BEGIN_YOUR_CODE (our solution is 8 lines of code, but don't worry if you deviate from
            this)
        org_list = problem.successorsAndCosts(state)
        return_list = []
        for tup in org_list:
            lst = list(tup)
            lst[2] += heuristic.evaluate(state)
            return_list.append(tuple(lst))
        return return_list
        # END_YOUR_CODE

  return NewSearchProblem()
```

b. 代码实现 StraightLineHeuristic 部分（8 分）

```python
class StraightLineHeuristic(Heuristic):

  def __init__(self, endTag: str, cityMap: CityMap):
      self.endTag = endTag
      self.cityMap = cityMap
      # Precompute
      # BEGIN_YOUR_CODE (our solution is 5 lines of code, but don't worry if you deviate from this)
      self.endPoints = []
      for location in cityMap.geoLocations:
          if(endTag in cityMap.tags[location]):
              self.endPoints.append(location)
      # 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)
      heuristicValue = 999999999.9
      for point in self.endPoints:
          distance = computeDistance(self.cityMap.geoLocations[state.location], point)
          if(distance < heuristicValue):
              heuristicValue = distance
      return heuristicValue
      # END_YOUR_CODE
```

c. 代码实现 NoWaypointsHeuristic 部分（12 分）

```python
class NoWaypointsHeuristic(Heuristic):

    def __init__(self, endTag: str, cityMap: CityMap):
        # Precompute
        # BEGIN_YOUR_CODE (our solution is 25 lines of code, but don't worry if you deviate from
            this)
        self.endPoints = []
        self.cityMap = cityMap
        self.endLocation = locationFromTag(endTag, cityMap)
        problem = ShortestPathProblem(self.endLocation, '1', self.cityMap)
        self.ucs = UniformCostSearch()
        self.ucs.solve(problem)
        # 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.ucs.pastCosts[state.location]
        # END_YOUR_CODE
```

反馈（10 分）

- 课堂体验还行
- 作业感觉难度有点大，花了 30h+