第一次作业（搜索问题）

彭煜峰　PB22051087

2024 年 4 月 5 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）

$\mathcal{D}_1 \circ d_s(v_2) = \min_{v\in\{v_1\}}\{d_s v + w_{vv_2}\} = 10$

b. 证明（2 分）

考虑从源点 s 到 v_k 的单源最短路径，若 $w_{sv_k} \neq +\infty$ 则目标路径为 s 直接到 v_k，此时 $d_s(v_k) = w_{sv_k}$，且显然 $\mathcal{D}_k \circ d_s(v_k) = 0 + w_{sv_k} = d_s(v_k)$ 成立；若 $w_{sv_k} = +\infty$，则目标路径为 s 经过一系列顶点之后到达 v_k，设 v_k 与 n 个顶点 $u_1, \cdots, u_n$ 直接相连，即 $\forall u \in \{u_1, \cdots, u_n\}$ 有 $w_{uv_k} \neq 0$，此时 $d_s(v_k) = \min_{u\in\{u_1,\cdots,u_n\}}\{d_s(u)+w_{uv_k}\}$。又由于 $d_s(v_k) \leqslant d_s(v_{k+1})$，满足条件的 u 一定满足：$u \in \{v_m \,|\, m \leqslant k-1\}$，因此 $\min_{u\in\{u_1,\cdots,u_n\}}\{d_s(u) + w_{uv_k}\} = \min_{v\in\{v_0,\cdots,v_{k-1}\}}\{d_s(v) + w_{vv_k}\} = \mathcal{D}_k \circ d_s(v_k)$。 □

c. 证明（4 分）

基础情形： 初始时，只有源节点 v_0 的最短路径长度被确定为 0，因此 $ds(v_0) = 0$，而对于其他节点 $v \in V$，它们的 $ds(v)$ 都被初始化为正无穷大，即 $ds(v) = \infty$。这符合假设。

归纳假设： 假设在执行算法的第 k 次迭代时，对于所有节点 $v \in V$，$ds(v)$ 是从源节点 v_0 到节点 v 的最短路径长度的上界，即 $ds(v) \leq d(v)$。

归纳步骤： 在第 k 次迭代中，选择节点 v_k 作为新的确定最短路径长度的节点。对于节点 v_k，$ds(v_k) = d(v_k)$，因为 v_k 是从源节点 v_0 到 v_k 的实际最短路径的一部分。

根据提示中给出的等式：

$$D_k \circ d(v_k) = ds(v_k) = \min_{v \in V - \{v_0, \ldots, v_{k-1}\}} \{ds(v)\} \leq \min_{v \in V - \{v_0, \ldots, v_{k-1}\}} \{d(v)\}$$

我们知道，对于任意节点 v，$ds(v) \leq d(v)$。因此，上述等式成立。

归纳结论： 根据归纳假设和归纳步骤，我们可以得出结论，在执行算法的第 k 次迭代后，对于所有节点 $v \in V$，$ds(v)$ 是从源节点 v_0 到节点 v 的最短路径长度的上界。
由于 Dijkstra 算法的每次迭代都会选择一个距离源节点最近的节点进行更新，因此在算法结束时，对于所有节点 $v \in V$，$ds(v)$ 将等于从源节点 v_0 到节点 v 的实际最短路径长度，即 $ds(v) = d(v)$。因此，Dijkstra 算法找到的是确实的最短路径。 □

d. 回答问题（2 分）

$\mathcal{D}_k$ 表示 Dijkstra 算法在执行到第 k 步时所使用的操作符，其中 k 表示当前已经确定了前 k 个顶点的最短路径。$\mathcal{D}_k$ 算符用来更新顶点的最短路径信息，同时缩小了搜索空间，只考虑前 k 个顶点的路径；Dijkstra 算法通过每一步选择当前距离最短的顶点进行松弛操作，更新其他顶点的最短路径长度。$\mathcal{D}_k$ 算符在每一步中只考虑前 k 个顶点的路径信息，从而将搜索空间缩小到这些顶点的邻居，进一步提高了算法的效率。其本质思想是通过不断更新已知的最短路径信息，并在每一步中选择距离最短的顶点，逐步求解最短路径问题。

e. 证明（2 分）

要证明所给的引理：

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

可以采用反证法。假设存在一个顶点 u，使得：

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

这意味着存在另一个顶点 v，使得从源点 s 到顶点 v 再到顶点 u 的路径长度比从源点 s 直接到顶点 u 的路径长度更短。但是这与单源最短路径问题的性质相矛盾，因为单源最短路径问题要求找到从源点 s 到顶点 u 的最短路径，而根据所给的定义，从顶点 v 再到顶点 u 的路径长度不可能比从源点 s 直接到顶点 u 的路径长度更短。因此，假设不成立，即：

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

根据此不等式的反面，即可得到所给的引理。 □

2.A* 算法，判断对错并说明原因（10 分）

a 这个说法是正确的。当 $h(u)$ 恒为 0 时，选择结点的价值估计就是 $d(u)$ 即从起始节点到当前节点的距离，选择其中最小的进行访问就是 Dijkstra 算法的思想；

b 这个说法是正确的。当不同的 $h_1(u)$ 和 $h_2(u)$ 对同一个结点 u 进行估计时可能得出不同的结果，导致 $d(u) + h(u)$ 也不同，从而影响下一个结点的选取，导致最后的 $d(u)$ 不同；

c 不正确。A* 算法中的估计函数 h 是启发式的，它可以低估或高估从当前节点到目标节点的距离。因此，尽管 d 是 h 的函数，但在算法的某一时刻，存在一个节点 u，使得 $d(u)$ 不等于图中任一路径的长度是可能的。这是因为 A* 算法会根据当前已知的信息选择下一个要探索的节点，而不一定是根据已知路径的长度。

d 这个说法是不正确的。虽然 A* 算法的最坏时间复杂度是 $O(m + n \log n)$，但这并不意味着所有情况下都能达到这个复杂度。实际上，A* 算法的性能受到启发式函数的影响，如果启发式函数选择不好，可能会导致更差的性能，甚至退化到与 Dijkstra 算法相同的时间复杂度 $O(m + n \log n)$。但是在一些情况下，A* 算法能够以更低的时间复杂度运行。

e 这个说法是正确的。给定这样的启发式估计 h，A* 算法和 Dijkstra 算法是等价的。这是因为在这种情况下，A* 算法会像 Dijkstra 算法一样工作，不再依赖于启发式估计，而是仅仅根据已知路径长度选择下一个要探索的节点，从而保证了与 Dijkstra 算法相同的最优解。

3. 网格城市（8 分）

a. 回答问题（8 分）

最低成本路径是沿着 y 轴向北移动 n 步，再沿着 x 轴向东或向西移动 m 步，到达目标点 (m, n)。这个最低成本路径是唯一的，因为不存在其他路径可以更进一步减少总成本。

问题 1：查找最短路径（12 分）

a. 代码实现 ShortestPathProblem 部分（8 分）

```
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, None)
        # 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)
        List = []
        for nextlocation, distance in self.cityMap.distances[state.location].items():
            lst = (nextlocation, State(nextlocation, None), distance)
```

```
            List.append(lst)
        return List
        # END_YOUR_CODE
```

b. 路线可视化（4 分）

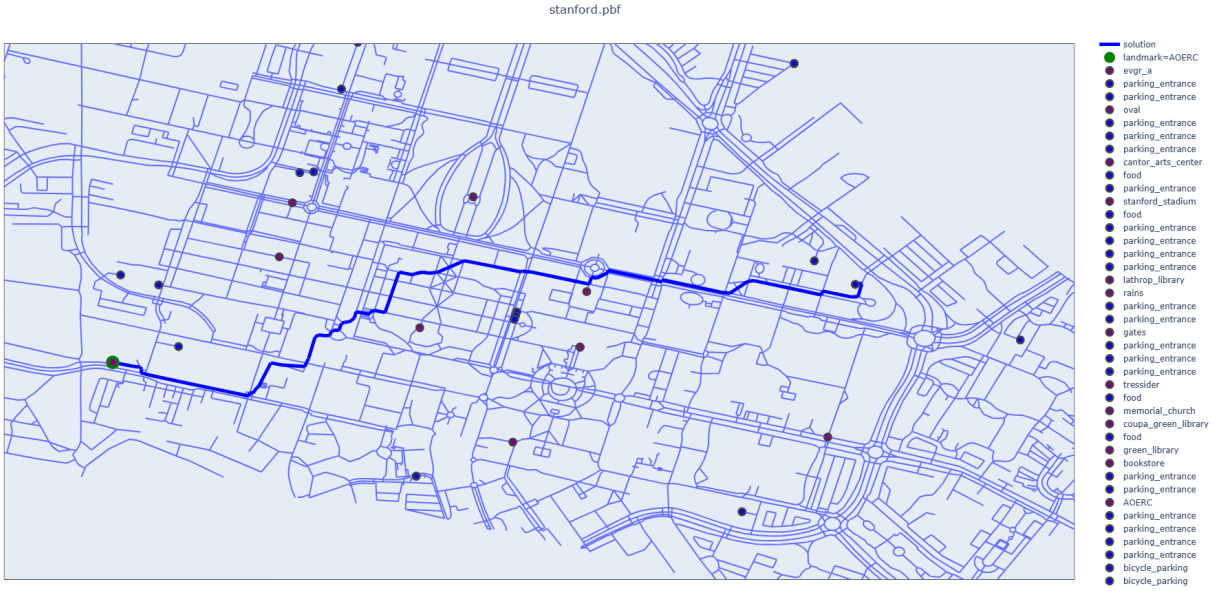

图 1: startLocation="1757902279", endTag="landmark=AOER"

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）

```
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)
        startMemory = []
        for tag in self.waypointTags:
            if tag in self.cityMap.tags[self.startLocation]: startMemory.append(1)
            else: startMemory.append(0)
        return State(self.startLocation, tuple(startMemory))
        # 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)
        if self.endTag not in self.cityMap.tags[state.location]: return False
```

```
        for value in state.memory:
            if value == 0: return False
        return True
        # 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)
        List = []
        for nextLocation, distance in self.cityMap.distances[state.location].items():
            next_memory = list(state.memory)
            for index in range(len(self.waypointTags)):
                if self.waypointTags[index] in self.cityMap.tags[nextLocation]:
                    next_memory[index] = 1
            List.append((nextLocation, State(nextLocation, tuple(next_memory)), distance
                ))
        return List
        # END_YOUR_CODE
```

b. 回答问题（4 分）

每个途径点标签有 2 种可能的位置。总共有 k 个途径点标签，所以总的路径排列数是 2^k。对于 UCS 算法来说，每个状态都代表了一个节点的状态，其中包含了该节点到达的地点以及经过的途径点。因此，最大的状态数等于 $n \times 2^k$。

c. 可视化（4 分）

图 2: startLocation="1757902279", endTag="landmark=AOER", waypointTags=["landmark=hoover_tower", "landmark=oval", "landmark=memorial_church"]

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）

```
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:
            # 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)
            succList = problem.successorsAndCosts(state)
            for element in succList:
                element = list(element)
                element[2] += heuristic.evaluate(element[1]) - heuristic.evaluate(state)
                element = tuple(element)
            return succList
            # END_YOUR_CODE

    return NewSearchProblem()
```

b. 代码实现 StraightLineHeuristic 部分（8 分）

```
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.endLocations = [location for location in self.cityMap.geoLocations.keys()
            if endTag in self.cityMap.tags[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)
        return max([computeDistance(self.cityMap.geoLocations[state.location], self.
            cityMap.geoLocations[endLocation]) for endLocation in self.endLocations])
        # END_YOUR_CODE
```

c. 代码实现 NoWaypointsHeuristic 部分（12 分）

```
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)
        class newProblem(SearchProblem):
            def startState(self) -> State:
                return State("State", None)
            def isEnd(self, state: State) -> bool:
                return False
            def successorsAndCosts(self, state: State) -> List[Tuple[str, State, float
                ]]:
                List = []
                if state.location == "State":
                    for endLocation in cityMap.geoLocations.keys():
                        if endTag in cityMap.tags[endLocation]: List.append((endLocation
                            , State(endLocation, None), 0))
                else:
                    for nextLocation, distance in cityMap.distances[state.location].
                        items(): List.append((nextLocation, State(nextLocation, None),
                        distance))
                return List
        UCS = UniformCostSearch(verbose=0)
        UCS.solve(problem=newProblem())
        self.heuristic = UCS.pastCosts
        # 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.heuristic[state.location]
        # END_YOUR_CODE
```

反馈（10 分）

- 希望课堂能结合代码实例来讲解算法；
- 希望作业代码可以有一些注释……