实验二 实验报告

本实验源码见 code 文件夹。

智能 212 史胤隆
2006010529

实验目的

熟悉并掌握搜索算法的相关理论，包括广度优先搜索、一致代价搜索、A* 等算法；
熟悉并掌握搜索算法求解最优路径的伪代码；
熟悉并掌握利用 Python 编程实现广度优先搜索、一致代价搜索、A* 等算法。

实验内容

完成实验既定题目；具体题目见实验结果.

实验结果

罗马尼亚旅行问题

经查，实验课进行过程中发布的提示代码数据与实际题目不符，因此我们首先自行构建地图及题目给出的相关数据：

```python
# generate.py

graph = {
    'Arad': {'distance': 366, 'neighbors': {
        'Zerind': 75,
        'Sibiu': 140,
        'Timisoara': 118
    }},
    'Bucharest': {'distance': 0, 'neighbors': {
        'Fagaras': 211,
        'Pitesti': 101,
        'Giurgiu': 90,
        'Urziceni': 85
    }},
    'Craiova': {'distance': 160, 'neighbors': {
        'Drobeta': 120,
        'Rimnicu Vilcea': 146,
        'Pitesti': 138
    }},
    'Drobeta': {'distance': 242, 'neighbors': {
        'Mehadia': 75,
        'Craiova': 120
    }},
    'Eforie': {'distance': 161, 'neighbors': {
```

```
        'Hirsova': 86
    }},
    'Fagaras': {'distance': 176, 'neighbors': {
        'Sibiu': 99,
        'Bucharest': 211
    }},
    'Giurgiu': {'distance': 77, 'neighbors': {
        'Bucharest': 90
    }},
    'Hirsova': {'distance': 151, 'neighbors': {
        'Eforie': 86,
        'Urziceni': 98
    }},
    'Iasi': {'distance': 226, 'neighbors': {
        'Neamt': 87,
        'Vaslui': 92
    }},
    'Lugoj': {'distance': 244, 'neighbors': {
        'Timisoara': 111,
        'Mehadia': 70
    }},
    'Mehadia': {'distance': 241, 'neighbors': {
        'Drobeta': 75,
        'Lugoj': 70
    }},
    'Neamt': {'distance': 234, 'neighbors': {
        'Iasi': 87
    }},
    'Oradea': {'distance': 380, 'neighbors': {
        'Zerind': 71,
        'Sibiu': 151
    }},
    'Pitesti': {'distance': 100, 'neighbors': {
        'Rimnicu Vilcea': 97,
        'Craiova': 138,
        'Bucharest': 101
    }},
    'Rimnicu Vilcea': {'distance': 193, 'neighbors': {
        'Sibiu': 80,
        'Pitesti': 97,
        'Craiova': 146
    }},
    'Sibiu': {'distance': 253, 'neighbors': {
        'Oradea': 151,
        'Arad': 140,
        'Rimnicu Vilcea': 80,
        'Fagaras': 99
    }},
    'Timisoara': {'distance': 329, 'neighbors': {
        'Arad': 118,
        'Lugoj': 111}},
    'Urziceni': {'distance': 80, 'neighbors': {
        'Bucharest': 85,
        'Hirsova': 98,
```

```
        'Vaslui': 142
    }},
    'Vaslui': {'distance': 199, 'neighbors': {
        'Urziceni': 142,
        'Iasi': 92
    }},
    'Zerind': {'distance': 374, 'neighbors': {
        'Arad': 75,
        'Oradea': 71
    }}
}

neighbor_map = {i: [j for j in graph[i]['neighbors']] for i in graph}
print('neighbor_map = ', neighbor_map)

neighbormapWithweight = {i: graph[i]['neighbors'] for i in graph}
print('neighbormapWithweight = ', neighbormapWithweight)

straight_to_Bucharest = {i: graph[i]['distance'] for i in graph}
print('straight_to_Bucharest = ', straight_to_Bucharest)
```

```
neighbor_map =  {'Arad': ['Zerind', 'Sibiu', 'Timisoara'], 'Bucharest': ['Fagaras',
'Pitesti', 'Giurgiu', 'Urziceni'], 'Craiova': ['Drobeta', 'Rimnicu Vilcea', 'Pitesti'],
'Drobeta': ['Mehadia', 'Craiova'], 'Eforie': ['Hirsova'], 'Fagaras': ['Sibiu',
'Bucharest'], 'Giurgiu': ['Bucharest'], 'Hirsova': ['Eforie', 'Urziceni'], 'Iasi':
['Neamt', 'Vaslui'], 'Lugoj': ['Timisoara', 'Mehadia'], 'Mehadia': ['Drobeta', 'Lugoj'],
'Neamt': ['Iasi'], 'Oradea': ['Zerind', 'Sibiu'], 'Pitesti': ['Rimnicu Vilcea', 'Craiova',
'Bucharest'], 'Rimnicu Vilcea': ['Sibiu', 'Pitesti', 'Craiova'], 'Sibiu': ['Oradea',
'Arad', 'Rimnicu Vilcea', 'Fagaras'], 'Timisoara': ['Arad', 'Lugoj'], 'Urziceni':
['Bucharest', 'Hirsova', 'Vaslui'], 'Vaslui': ['Urziceni', 'Iasi'], 'Zerind': ['Arad',
'Oradea']}
neighbormapWithweight =  {'Arad': {'Zerind': 75, 'Sibiu': 140, 'Timisoara': 118},
'Bucharest': {'Fagaras': 211, 'Pitesti': 101, 'Giurgiu': 90, 'Urziceni': 85}, 'Craiova':
{'Drobeta': 120, 'Rimnicu Vilcea': 146, 'Pitesti': 138}, 'Drobeta': {'Mehadia': 75,
'Craiova': 120}, 'Eforie': {'Hirsova': 86}, 'Fagaras': {'Sibiu': 99, 'Bucharest': 211},
'Giurgiu': {'Bucharest': 90}, 'Hirsova': {'Eforie': 86, 'Urziceni': 98}, 'Iasi': {'Neamt':
87, 'Vaslui': 92}, 'Lugoj': {'Timisoara': 111, 'Mehadia': 70}, 'Mehadia': {'Drobeta': 75,
'Lugoj': 70}, 'Neamt': {'Iasi': 87}, 'Oradea': {'Zerind': 71, 'Sibiu': 151}, 'Pitesti':
{'Rimnicu Vilcea': 97, 'Craiova': 138, 'Bucharest': 101}, 'Rimnicu Vilcea': {'Sibiu': 80,
'Pitesti': 97, 'Craiova': 146}, 'Sibiu': {'Oradea': 151, 'Arad': 140, 'Rimnicu Vilcea':
80, 'Fagaras': 99}, 'Timisoara': {'Arad': 118, 'Lugoj': 111}, 'Urziceni': {'Bucharest':
85, 'Hirsova': 98, 'Vaslui': 142}, 'Vaslui': {'Urziceni': 142, 'Iasi': 92}, 'Zerind':
{'Arad': 75, 'Oradea': 71}}
straight_to_Bucharest =  {'Arad': 366, 'Bucharest': 0, 'Craiova': 160, 'Drobeta': 242,
'Eforie': 161, 'Fagaras': 176, 'Giurgiu': 77, 'Hirsova': 151, 'Iasi': 226, 'Lugoj': 244,
'Mehadia': 241, 'Neamt': 234, 'Oradea': 380, 'Pitesti': 100, 'Rimnicu Vilcea': 193,
'Sibiu': 253, 'Timisoara': 329, 'Urziceni': 80, 'Vaslui': 199, 'Zerind': 374}
```

实验 1. 罗马尼亚旅行问题的广度优先搜索

广度优先搜索是十分基础的搜索算法。这里我们选用生成的 `neighbor_map` 数据来进行搜索算法的设计；其中 `Node` 类的设计方便了历史路径的记录，增加了代码的可读性，为之后的题目做准备。

```python
# work1.py

from typing import *

neighbor_map = {
    'Arad': ['Zerind', 'Sibiu', 'Timisoara'],
    'Bucharest': ['Fagaras', 'Pitesti', 'Giurgiu', 'Urziceni'],
    'Craiova': ['Drobeta', 'Rimnicu Vilcea', 'Pitesti'],
    'Drobeta': ['Mehadia', 'Craiova'],
    'Eforie': ['Hirsova'],
    'Fagaras': ['Sibiu', 'Bucharest'],
    'Giurgiu': ['Bucharest'],
    'Hirsova': ['Eforie', 'Urziceni'],
    'Iasi': ['Neamt', 'Vaslui'],
    'Lugoj': ['Timisoara', 'Mehadia'],
    'Mehadia': ['Drobeta', 'Lugoj'],
    'Neamt': ['Iasi'],
    'Oradea': ['Zerind', 'Sibiu'],
    'Pitesti': ['Rimnicu Vilcea', 'Craiova', 'Bucharest'],
    'Rimnicu Vilcea': ['Sibiu', 'Pitesti', 'Craiova'],
    'Sibiu': ['Oradea', 'Arad', 'Rimnicu Vilcea', 'Fagaras'],
    'Timisoara': ['Arad', 'Lugoj'],
    'Urziceni': ['Bucharest', 'Hirsova', 'Vaslui'],
    'Vaslui': ['Urziceni', 'Iasi'],
    'Zerind': ['Arad', 'Oradea']
}


class Node:
    def __init__(self, name: str, history: List[str] = ...):
        self.name = name
        self.history = history if history is not ... else []

    def expand(self) -> List['Node']:
        return [Node(i, self.history + [self.name]) for i in neighbor_map[self.name]]


def bfs(start: 'Node', goal: 'Node'):
    queue = [start]
    while queue:
        node = queue.pop(0)
        print(*node.history, node.name, sep=' -> ')
        if node.name == goal.name:
            print('求解完成。')
            break
```

```
        queue.extend(node.expand())
    else:
        print('求解失败。')


a = Node(input('请输入起点城市名: '))   # Arad
b = Node(input('请输入目标城市名: '))   # Bucharest
bfs(a, b)
```

```
请输入起点城市名: Arad
请输入目标城市名: Bucharest
Arad
Arad -> Zerind
Arad -> Sibiu
Arad -> Timisoara
Arad -> Zerind -> Arad
Arad -> Zerind -> Oradea
Arad -> Sibiu -> Oradea
Arad -> Sibiu -> Arad
Arad -> Sibiu -> Rimnicu Vilcea
Arad -> Sibiu -> Fagaras
Arad -> Timisoara -> Arad
Arad -> Timisoara -> Lugoj
Arad -> Zerind -> Arad -> Zerind
Arad -> Zerind -> Arad -> Sibiu
Arad -> Zerind -> Arad -> Timisoara
Arad -> Zerind -> Oradea -> Zerind
Arad -> Zerind -> Oradea -> Sibiu
Arad -> Sibiu -> Oradea -> Zerind
Arad -> Sibiu -> Oradea -> Sibiu
Arad -> Sibiu -> Arad -> Zerind
Arad -> Sibiu -> Arad -> Sibiu
Arad -> Sibiu -> Arad -> Timisoara
Arad -> Sibiu -> Rimnicu Vilcea -> Sibiu
Arad -> Sibiu -> Rimnicu Vilcea -> Pitesti
Arad -> Sibiu -> Rimnicu Vilcea -> Craiova
Arad -> Sibiu -> Fagaras -> Sibiu
Arad -> Sibiu -> Fagaras -> Bucharest
求解完成。
```

实验 2 - 1. 罗马尼亚旅行问题的一致代价搜索

我们将生成的 `neighbor_map` 数据替换为带有路径距离信息的 `neighbormapWithweight` 数据来进行搜索算法的设计。相对于上一题，`Node` 类添加了 `cost` 属性以方便计算路径代价；搜索函数新增 `queue.sort()` 排序语句以实现类似优先队列的效果。

```python
# work2-1.py

from typing import *

neighbormapWithweight = {
    'Arad': {'Zerind': 75, 'Sibiu': 140, 'Timisoara': 118},
    'Bucharest': {'Fagaras': 211, 'Pitesti': 101, 'Giurgiu': 90, 'Urziceni': 85},
    'Craiova': {'Drobeta': 120, 'Rimnicu Vilcea': 146, 'Pitesti': 138},
    'Drobeta': {'Mehadia': 75, 'Craiova': 120},
    'Eforie': {'Hirsova': 86},
    'Fagaras': {'Sibiu': 99, 'Bucharest': 211},
    'Giurgiu': {'Bucharest': 90},
    'Hirsova': {'Eforie': 86, 'Urziceni': 98},
    'Iasi': {'Neamt': 87, 'Vaslui': 92},
    'Lugoj': {'Timisoara': 111, 'Mehadia': 70},
    'Mehadia': {'Drobeta': 75, 'Lugoj': 70},
    'Neamt': {'Iasi': 87},
    'Oradea': {'Zerind': 71, 'Sibiu': 151},
    'Pitesti': {'Rimnicu Vilcea': 97, 'Craiova': 138, 'Bucharest': 101},
    'Rimnicu Vilcea': {'Sibiu': 80, 'Pitesti': 97, 'Craiova': 146},
    'Sibiu': {'Oradea': 151, 'Arad': 140, 'Rimnicu Vilcea': 80, 'Fagaras': 99},
    'Timisoara': {'Arad': 118, 'Lugoj': 111},
    'Urziceni': {'Bucharest': 85, 'Hirsova': 98, 'Vaslui': 142},
    'Vaslui': {'Urziceni': 142, 'Iasi': 92},
    'Zerind': {'Arad': 75, 'Oradea': 71}
}


class Node:
    def __init__(self, name: str, history: List[str] = ..., cost: int = 0):
        self.name = name
        self.history = history if history is not ... else []
        self.cost = cost

    def expand(self) -> List['Node']:
        return [Node(
            i,
            self.history + [self.name],
            self.cost + neighbormapWithweight[self.name][i]
        ) for i in neighbormapWithweight[self.name]]


def ucs(start: 'Node', goal: 'Node'):
    queue = [start]
```

```
    while queue:
        node = queue.pop(0)
        print(*node.history, node.name, sep=' -> ')
        if node.name == goal.name:
            print('求解完成。')
            break
        queue.extend(node.expand())
        queue.sort(key=lambda x: x.cost)
    else:
        print('求解失败。')


a = Node(input('请输入起点城市名: '))   # Arad
b = Node(input('请输入目标城市名: '))   # Bucharest
ucs(a, b)
```

```
请输入起点城市名: Arad
请输入目标城市名: Bucharest
Arad
Arad -> Zerind
Arad -> Timisoara
Arad -> Sibiu
Arad -> Zerind -> Oradea
Arad -> Zerind -> Arad
Arad -> Zerind -> Oradea -> Zerind
Arad -> Sibiu -> Rimnicu Vilcea
Arad -> Zerind -> Arad -> Zerind
Arad -> Timisoara -> Lugoj
Arad -> Timisoara -> Arad
Arad -> Sibiu -> Fagaras
Arad -> Zerind -> Arad -> Timisoara
Arad -> Sibiu -> Arad
Arad -> Zerind -> Oradea -> Zerind -> Oradea
Arad -> Zerind -> Arad -> Sibiu
Arad -> Sibiu -> Oradea
Arad -> Zerind -> Oradea -> Zerind -> Arad
Arad -> Zerind -> Arad -> Zerind -> Oradea
Arad -> Zerind -> Oradea -> Sibiu
Arad -> Timisoara -> Lugoj -> Mehadia
Arad -> Sibiu -> Rimnicu Vilcea -> Sibiu
Arad -> Zerind -> Arad -> Zerind -> Arad
Arad -> Timisoara -> Arad -> Zerind
Arad -> Sibiu -> Rimnicu Vilcea -> Pitesti
Arad -> Sibiu -> Fagaras -> Sibiu
Arad -> Timisoara -> Lugoj -> Timisoara
Arad -> Timisoara -> Arad -> Timisoara
Arad -> Sibiu -> Arad -> Zerind
Arad -> Zerind -> Oradea -> Zerind -> Oradea -> Zerind
```

```
Arad -> Sibiu -> Oradea -> Zerind
Arad -> Sibiu -> Rimnicu Vilcea -> Craiova
Arad -> Zerind -> Oradea -> Zerind -> Arad -> Zerind
Arad -> Zerind -> Arad -> Zerind -> Oradea -> Zerind
Arad -> Timisoara -> Lugoj -> Mehadia -> Lugoj
Arad -> Zerind -> Arad -> Sibiu -> Rimnicu Vilcea
Arad -> Timisoara -> Lugoj -> Mehadia -> Drobeta
Arad -> Zerind -> Arad -> Zerind -> Arad -> Zerind
Arad -> Timisoara -> Arad -> Sibiu
Arad -> Zerind -> Oradea -> Sibiu -> Rimnicu Vilcea
Arad -> Zerind -> Arad -> Timisoara -> Lugoj
Arad -> Sibiu -> Rimnicu Vilcea -> Sibiu -> Rimnicu Vilcea
Arad -> Timisoara -> Arad -> Zerind -> Oradea
Arad -> Zerind -> Arad -> Timisoara -> Arad
Arad -> Timisoara -> Arad -> Zerind -> Arad
Arad -> Zerind -> Arad -> Sibiu -> Fagaras
Arad -> Zerind -> Oradea -> Sibiu -> Fagaras
Arad -> Sibiu -> Arad -> Timisoara
Arad -> Sibiu -> Rimnicu Vilcea -> Sibiu -> Fagaras
Arad -> Zerind -> Oradea -> Zerind -> Arad -> Timisoara
Arad -> Sibiu -> Rimnicu Vilcea -> Pitesti -> Rimnicu Vilcea
Arad -> Zerind -> Arad -> Zerind -> Arad -> Timisoara
Arad -> Sibiu -> Rimnicu Vilcea -> Pitesti -> Bucharest
求解完成。
```

实验 2-2. 罗马尼亚旅行问题的 A* 算法搜索

相对于上一题，新增记录剩余代价的 `straight_to_Bucharest` 数据来进行搜索算法的设计，修改 `queue.sort()` 排序规则即可。

```python
# work2-2.py

from typing import *

neighbormapWithweight = {
    'Arad': {'Zerind': 75, 'Sibiu': 140, 'Timisoara': 118},
    'Bucharest': {'Fagaras': 211, 'Pitesti': 101, 'Giurgiu': 90, 'Urziceni': 85},
    'Craiova': {'Drobeta': 120, 'Rimnicu Vilcea': 146, 'Pitesti': 138},
    'Drobeta': {'Mehadia': 75, 'Craiova': 120},
    'Eforie': {'Hirsova': 86},
    'Fagaras': {'Sibiu': 99, 'Bucharest': 211},
    'Giurgiu': {'Bucharest': 90},
    'Hirsova': {'Eforie': 86, 'Urziceni': 98},
    'Iasi': {'Neamt': 87, 'Vaslui': 92},
    'Lugoj': {'Timisoara': 111, 'Mehadia': 70},
    'Mehadia': {'Drobeta': 75, 'Lugoj': 70},
    'Neamt': {'Iasi': 87},
    'Oradea': {'Zerind': 71, 'Sibiu': 151},
    'Pitesti': {'Rimnicu Vilcea': 97, 'Craiova': 138, 'Bucharest': 101},
```

```
    'Rimnicu Vilcea': {'Sibiu': 80, 'Pitesti': 97, 'Craiova': 146},
    'Sibiu': {'Oradea': 151, 'Arad': 140, 'Rimnicu Vilcea': 80, 'Fagaras': 99},
    'Timisoara': {'Arad': 118, 'Lugoj': 111},
    'Urziceni': {'Bucharest': 85, 'Hirsova': 98, 'Vaslui': 142},
    'Vaslui': {'Urziceni': 142, 'Iasi': 92},
    'Zerind': {'Arad': 75, 'Oradea': 71}
}

straight_to_Bucharest = {
    'Arad': 366, 'Bucharest': 0, 'Craiova': 160, 'Drobeta': 242,
    'Eforie': 161, 'Fagaras': 176, 'Giurgiu': 77, 'Hirsova': 151,
    'Iasi': 226,  'Lugoj': 244, 'Mehadia': 241, 'Neamt': 234,
    'Oradea': 380, 'Pitesti': 100, 'Rimnicu Vilcea': 193, 'Sibiu': 253,
    'Timisoara': 329, 'Urziceni': 80, 'Vaslui': 199, 'Zerind': 374
}


class Node:
    def __init__(self, name: str, history: List[str] = ..., cost: int = 0):
        self.name = name
        self.history = history if history is not ... else []
        self.cost = cost

    def expand(self) -> List['Node']:
        return [Node(
            i,
            self.history + [self.name],
            self.cost + neighbormapWithweight[self.name][i]
        ) for i in neighbormapWithweight[self.name]]


def astar(start: 'Node', goal: 'Node'):
    queue = [start]
    while queue:
        node = queue.pop(0)
        print(*node.history, node.name, sep=' -> ')
        if node.name == goal.name:
            print('求解完成。')
            break
        queue.extend(node.expand())
        queue.sort(key=lambda x: x.cost + straight_to_Bucharest[x.name])
    else:
        print('求解失败。')


a = Node(input('请输入起点城市名: '))   # Arad
b = Node(input('请输入目标城市名: '))   # Bucharest
astar(a, b)
```

```
请输入起点城市名: Arad
请输入目标城市名: Bucharest
Arad
Arad -> Sibiu
Arad -> Sibiu -> Rimnicu Vilcea
Arad -> Sibiu -> Fagaras
Arad -> Sibiu -> Rimnicu Vilcea -> Pitesti
Arad -> Sibiu -> Rimnicu Vilcea -> Pitesti -> Bucharest
求解完成。
```

算法应用实验

基于相同的原因，自行构建地图，并复用之前题目的代码完成本题目。

实验 3. 基于以上三种算法的图搜索

本题目代码及输出结果如下：

```python
# work3.py

from typing import *

graph = {
    'AP': {'distance': 8, 'neighbors': {}},
    'BBY': {'distance': 8, 'neighbors': {}},
    'DT': {'distance': 2, 'neighbors': {
        'SP': 2
    }},
    'JB': {'distance': 3, 'neighbors': {
        'KB': 4
    }},
    'KB': {'distance': 3, 'neighbors': {
        'BBY': 6,
        'DT': 3
    }},
    'KD': {'distance': 6, 'neighbors': {
        'JB': 2,
        'MP': 4
    }},
    'MP': {'distance': 7, 'neighbors': {
        'BBY': 5,
        'KB': 3
    }},
    'RM': {'distance': 9, 'neighbors': {
        'SSY': 21
    }},
    'SP': {'distance': 0, 'neighbors': {}},
    'SRY': {'distance': 29, 'neighbors': {
        'BBY': 23
```

```
    }},
    'UBC': {'distance': 5, 'neighbors': {
        'JB': 3,
        'KD': 3
    }}
}

neighbor_map = {i: [j for j in graph[i]['neighbors']] for i in graph}
neighbormapWithweight = {i: graph[i]['neighbors'] for i in graph}
distance_to_SP = {i: graph[i]['distance'] for i in graph}


class Node:
    def __init__(self, name: str, history: List[str] = ..., cost: int = 0):
        self.name = name
        self.history = history if history is not ... else []
        self.cost = cost

    def expand(self) -> List['Node']:
        return [Node(
            i,
            self.history + [self.name],
            self.cost + neighbormapWithweight[self.name][i]
        ) for i in neighbormapWithweight[self.name]]


def bfs(start: 'Node', goal: 'Node'):
    queue = [start]
    while queue:
        node = queue.pop(0)
        print(*node.history, node.name, sep=' -> ')
        if node.name == goal.name:
            print('求解完成。')
            break
        queue.extend(node.expand())
    else:
        print('求解失败。')


def ucs(start: 'Node', goal: 'Node'):
    queue = [start]
    while queue:
        node = queue.pop(0)
        print(*node.history, node.name, sep=' -> ')
        if node.name == goal.name:
            print('求解完成。')
            break
        queue.extend(node.expand())
        queue.sort(key=lambda x: x.cost)
    else:
```

```
        print('求解失败。')


def astar(start: 'Node', goal: 'Node'):
    queue = [start]
    while queue:
        node = queue.pop(0)
        print(*node.history, node.name, sep=' -> ')
        if node.name == goal.name:
            print('求解完成。')
            break
        queue.extend(node.expand())
        queue.sort(key=lambda x: x.cost + distance_to_SP[x.name])
    else:
        print('求解失败。')


print('\n广度优先搜索: ')
bfs(Node('UBC'), Node('SP'))
print('\n一致代价搜索: ')
ucs(Node('UBC'), Node('SP'))
print('\nA* 搜索: ')
astar(Node('UBC'), Node('SP'))
```

```
广度优先搜索:
UBC
UBC -> JB
UBC -> KD
UBC -> JB -> KB
UBC -> KD -> JB
UBC -> KD -> MP
UBC -> JB -> KB -> BBY
UBC -> JB -> KB -> DT
UBC -> KD -> JB -> KB
UBC -> KD -> MP -> BBY
UBC -> KD -> MP -> KB
UBC -> JB -> KB -> DT -> SP
求解完成。

一致代价搜索:
UBC
UBC -> JB
UBC -> KD
UBC -> KD -> JB
UBC -> JB -> KB
UBC -> KD -> MP
UBC -> KD -> JB -> KB
UBC -> JB -> KB -> DT
```

```
UBC -> KD -> MP -> KB
UBC -> KD -> MP -> BBY
UBC -> KD -> JB -> KB -> DT
UBC -> JB -> KB -> DT -> SP
求解完成。

A* 搜索:
UBC
UBC -> JB
UBC -> KD
UBC -> KD -> JB
UBC -> JB -> KB
UBC -> KD -> JB -> KB
UBC -> JB -> KB -> DT
UBC -> JB -> KB -> DT -> SP
求解完成。
```