自动规划实验报告

指定规划问题求解

专业：	人工智能
学号：	58119125
年级：	19 级
姓名：	蒋卓洋

签名：蒋卓洋

时间：2022/6/13

一．选题

1．易用 PDDL 实现的传统规划问题：

针对老师提供了伪代码的三个规划问题：

- 积木世界
- 航空运货
- 换备胎

我尝试用 PDDL 语言复现了其结果，以达到“熟悉 PDDL 语言和规划问题思考逻辑”的目的。在复现这三个经典规划问题的过程中，我以熟悉 PDDL 工程结构为主题，尝试使用 PDDL 编译工具、构建 PDDL 工程文件、编写 PDDL 代码，对 PDDL 整个工作流程有了一个基础的了解。

2．小车搬运问题实现方法选择：

我首先尝试了用 PDDL 语言实现小车搬运问题，但由于问题的约束条件比较复杂，且有较强的数据结构特性，因此在多次尝试用 PDDL 实现失败后，我转而使用 C++的基础编程语言，依据停车场表现出来的性质，构造基础容器，从而以数据结构的方式满足了规划限制，同时结合经典的搜索算法，实现了规划求解。

为了更好地用 C++实现小车搬运问题的规划求解，我在自定规划问题中选择了八数码问题作为前置问题参考，八数码问题的操作更加复杂，因此在解决了八数码问题的解的搜索后，我了解了如何以搜索的视角观察规划问题，并迁移该方法，最终实现了小车搬运问题的规划求解。

二．实现方法概述

1．基于 PDDL 的传统规划问题实现：

1.1. 积木世界

（1）问题（规则）描述：

①情景描述与规则归纳：

桌面上有正方形积木块，一块积木上只允许直接放置一块积木。

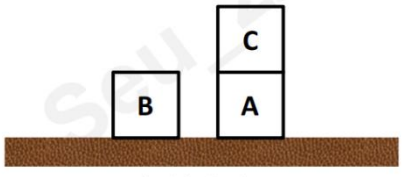
- 一个机器人手臂能将积木挪动位置——置于桌面上或其他积木上。
- 每次只能移动一块积木

②对象提取：

两种对象：木块和桌面

（2）初始状态：

①情景描述：



初始状态

②逻辑描述：

Init(On(A, Table) ∧ On(B, Table) ∧ On(C, A) ∧ Block (A) ∧ Block (B) ∧ Block (C) ∧ Clear (B) ∧ Clear (C))

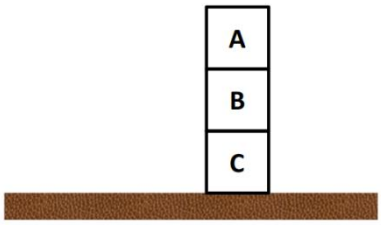
③PDDL-problem：

```
(:init (block blocka)
```

```
(block blockb)
(block blockc)
(table tablet)
(on blocka tablet)
(on blockb tablet)
(on blockc blocka)
(clear blockb)
(clear blockc))
```

（3）目标状态：

①情景描述：



目标状态

②逻辑描述：

Goal(On(A, B) ∧ On(B,C))

③PDDL-Problem：

```
(:goal (and (on blocka blockb)
(on blockb blockc)))
```

（4）动作：

①动作分解与分类：

1) 第一层分解：木块移动

2) 第二层分解：木块移动移动至另一木块上、木块移至桌面上

②动作函数构建：

A. 变量与常量：

a) 木块

b) 桌面

B. 谓词（PDDL）：

a) 【x 为空】(clear ?b)

b) 【x 是桌面】(table ?t)

c) 【x 是木块】(block ?b)

d) 【x 在 y 上】 (on ?x ?y)

e) 【x 等于 y】(isequal ?x ?y))

1) 木块移动移动至另一木块上：

a. 参数：(移动对象木块 起点位置木块 目标位置木块)

b. 前提（PDDL）：

```
:precondition (and (on ?b ?x)
                   (clear ?b)
                   (clear ?y)
                   (block ?b)
                   (block ?y)
                (not (isequal ?b ?x))
                (not (isequal ?b ?y))
                (not (isequal ?y ?x)))
```

c. 结果（PDDL）：

```
:effect (and (on ?b ?y)
             (clear ?x)
          (not (on ?b ?x))
          (not (clear ?y)))
```

2) 木块移动移动至桌面：

a. 参数：（移动对象木块，起点位置木块，桌面）

b. 前提（PDDL）：

```
:precondition (and (on ?b ?x)
                   (clear ?b)
                   (block ?b)
                   (block ?x)
                (not (isequal ?x ?b))
                     (table ?t))
```

c. 结果（PDDL）：

```
:effect (and (on ?b ?t)
             (clear ?x)
          (not (on ?b ?x)
```

1.2. 航空运货

（1） 问题（规则）描述：

①情景描述：

两架飞机在两个机场之间运送两件货物，两件货物初始分布于两个机场（起送地），要求用对应飞机将特定货物进行运输，是两件货物分别被运送至两个机场中对应的目的地。

②对象提取：

三种对象：飞机、货物、机场

（2） 初始状态：

①情景描述：

货物 C1、C2 分别在机场 SFO 和 JFK，飞机 P1、P2 分别停靠在 SFO 和 JFK。

②逻辑描述：

Init(At(C1, SFO) ∧ At(C2, JFK) ∧ At(P1, SFO) ∧ At(P2, JFK) ∧ Cargo(C1) ∧ Cargo(C2) ∧ Plane(P1) ∧ Plane(P2) ∧ Airport(JFK) ∧ Airport(SFO))

③PDDL-Problem：

```
(:init (plane plane1)
```

```
(plane plane2)
(airport SFO)
(airport JFK)
(cargo cargo1)
(cargo cargo2)
(at cargo1 SFO)
(at cargo2 JFK)
(at plane1 SFO)
(at plane2 JFK))
```

（3） 目标状态：

①情景描述：

货物 C1、C2 分别在机场 JFK 和 SFO。

②逻辑描述：

Goal(At(C1, JFK) ∧ At(C2, SFO))

③PDDL-Problem：

```
(:goal (and (at cargo1 JFK)
     (at cargo2 SFO)))
```

（4） 动作：

①动作分解与分类：

装载、卸载、飞行

②动作函数构建：

A. 变量与常量：

a) 货物
b) 飞机
c) 机场

B. 谓词（PDDL）：

a) 【x 是飞机】(plane ?p)
b) 【x 是货物】(cargo ?c)
c) 【x 是机场】(airport ?a)
d) 【飞机 x 在机场 y】 (at ?p ?a)
e) 【货物 x 在机场 y】 (at ?c ?a)
f) 【货物 x 在飞机 y 上】(in ?c ?p)

1) 装载：

a. 参数：（机场代号，在该机场的运输飞机代号，在该机场的货物代号）

b. 前提（PDDL）：

```
:precondition (and (cargo ?c)
          (plane ?p)
         (airport ?a)
          (in ?c ?p)
          (at ?p ?a))
```

c. 结果（PDDL）：

```
:effect (and (not (at ?c ?a))
             (in ?c ?p))
```

2) 卸载：

a. 参数：（机场代号，在该机场的运输飞机代号，在该机场的货物代号）

b. 前提（PDDL）：

```
:precondition (and (cargo ?c)
                   (plane ?p)
                   (airport ?a)
                   (in ?c ?p)
                   (at ?p ?a))
```

c. 结果（PDDL）：

```
:effect (and (at ?c ?a)
             (not (in ?c ?p)))
```

3) 飞行：

a. 参数：（飞机代号，起点机场代号，目标机场代号）

b. 前提（PDDL）：

```
:precondition (and (airport ?from)
                   (airport ?to)
                   (plane ?p)
                   (at ?p ?from))
```

c. 结果（PDDL）：

```
:effect (and (not (at ?p ?from))
             (at ?p ?to))
```

1.3. 换备胎

（1） 问题（规则）描述：

①情景描述：

用后备箱中的备胎替换汽车的瘪轮胎，考虑三个动作：

- 卸掉轮胎：
- 安装轮胎：
- 汽车无人值守过夜：全部轮胎会被盗走。

②对象提取：

两种对象：轮胎、轮胎去向

（2） 初始状态：

①情景描述：

轮胎在轮轴上，备胎在后备箱

②逻辑描述：

(Tire(Flat) ∧ Tire(Spare) ∧ At(Flat, Axle) ∧ At(Spare, Trunk))

③PDDL-Problem：

```
(:init (tire Flat)
       (tire Spare)
       (at Flat Axle)
       (at Spare Trunk))
```

（3） 目标状态：

①情景描述：

备胎在轮轴上

②逻辑描述：

Goal(At(Spare, Axle))

③PDDL-Problem：

```
(:goal (at Spare Axle))
```

（4） 动作：

①动作分解与分类：

两个动作：卸载轮胎与安装轮胎

一个结果性动作：汽车无人值守过夜，全部轮胎会被盗走。

②动作函数构建：

A. 变量与常量：

a) 轮轴

b) 后备箱

c) 轮胎

d) 地面

B. 谓词（PDDL）：

a) 【x 是轮胎】(tire ?t)

b) 【轮胎 x 在 y 上（中）】(at ?x ?y)

1) 卸载（拿出）轮胎：

a. 参数：（轮胎代号，卸载（拿出）位置）

b. 前提（PDDL）：

```
:precondition (at ?obj ?loc)
```

c. 结果（PDDL）：

```
:effect (and (not (at ?obj ?loc))
             (at ?obj Ground))
```

2) 安装（放入）轮胎：

a. 参数：（轮胎代号，安装（放入）位置）

b. 前提（PDDL）：

```
:precondition (and (tire ?t)
                   (at ?t Ground)
                   (not (at Flat ?Axle)))
```

c. 结果（PDDL）：

```
:effect (and (not (at ?t Ground))
             (at ?t ?Axle))
```

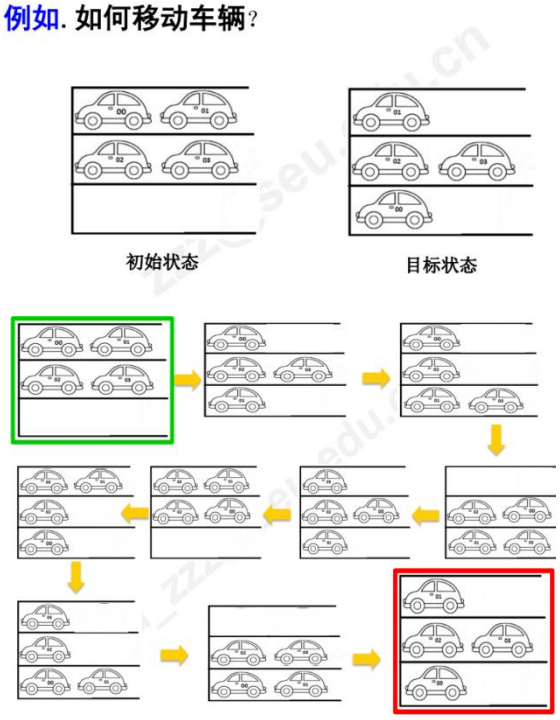
3) 汽车无人值守过夜，全部轮胎会被盗走：

a. 结果（PDDL）：

```
:effect (and (not (at Spare Ground))
             (not (at Spare Axle))
             (not (at Spare Trunk))
             (not (at Flat Ground))
             (not (at Flat Axle))
             (not (at Flat Trunk)))
```

2．基于 C++ 语言的小车搬运问题实现

（1）问题描述：



（2）问题建模与数据结构：

可以将此规划问题视作一个搜索问题，每次移动小车得到的结果是一个状态，而小车与停车场的空间关系可以形成一个状态载体，整个可能的状态集合构成了一个抽象的状态图结构，每一个状态是一个图节点，车与停车场的空间关系就是该图节点中存储的数据。

求解规划的过程就是在两个状态节点之间搜索路径，并且将路径上的操作过程展现出来。对此，我们需要做如下问题建模：

1) 定义状态的数据描述：
   a. 停车场和小车的位置关系可以用特定的抽象数据类型来描述，其中可以观察到，每个停车道“单口出入、先进后出”的规则满足栈的特性，因此三个（甚至 N 个）停车道可以各用一个栈结构表示
   b. 三个（甚至 N 个）停车道是并列同等关系的数据元素，各用一个栈结构表示的情况下，三个（甚至 N 个）停车道构成的停车场可以用一个以栈为基础元素的动态顺序表容器表示
   c. 每张车唯一需要指定的数据信息就是车辆的代号，因此可以用一个抽象数据结构表示一张车，每个抽象数据结构的结构体成员只有一个整型变量用于描述车辆代号。
2) 定义移动操作，设定操作限制：
   a. 数据结构：

      搬运小车的移动操作是在停车场抽象数据类型的对象中，在不同停车道间“单口出入、先进后出”的一移出一移入操作，这个“单口出入、先进后出”限制已经随着数据结构的选取和构建而得到满足。以目标状态为例：

      carPort: { stack0:[CAR1]，

stack1:[CAR2,CAR3]，
stack2:[CAR0] }

b. 动作定义：

在栈的基础操作中，唯一需要进行移动操作的对象就是栈顶元素，因此在移动操作实现中，无需具体指代需要移动的车辆代号，只需对栈顶元素进行操作即可，这极大地方便了操作的定义。

移动操作有三个要素：移动对象，移动起点和移动终点。因为栈只需操作栈顶元素，因此我们只需要确定移动起点栈的代号和移动终点栈的代号即可，也就是本问题移动操作只有两个要素：移动起点栈的代号和移动终点栈的代号。因此在生产移动操作的动作时，我们可以用一个二元组来表示这个移动动作：

action = （起点栈编号，终点栈编号）

c. 动作生成：

有了移动操作的二元表示，我们在生成动作的时候，需要实现在限制下找出当前状态可以进行的操作列表。包括两部分：

- 可行的起点栈编号生成：可以驶出的停车栈，即可行的起点栈，需要满足一个条件，即“栈非空”
- 可行的终点栈编号生成：可以停入的停车栈，即可行的终点栈，需要满足一个条件，即“栈不满”

在分别生成了所有可行的起点栈和终点栈之后，可以对两个要素进行组合，生成所有当前状态下可行的移动动作元组，进而生成当前状态下可行操作列表。

```
CAR_NODE updateCarNodeActionList(CAR_NODE carNode) {
    vector<int> portPost;//终点栈
    vector<int> portPre;//起点栈
    for (int x = 0; x < carNode.carPort.size(); x++) {//找出可以停入和可以驶出的停车栈
        int y = carNode.carPort[x].size();
        if (y != 2) portPost.push_back(x);//终点栈不满
        if (y != 0) portPre.push_back(x);//起点栈不空
    }
    for (int pre = 0; pre < portPre.size(); pre++) {//对可行的起点栈和终点栈进行组合，传入可行操作列表
        for (int post = 0; post < portPost.size(); post++) {
            if (portPre[pre] != portPost[post]) {
                carNode.nextActionList.push_back({ portPre[pre], portPost[post] });
            }
        }
    }
    return carNode;}
```

d. 移动过程：

之后就可以传入当前停车场状态和移动动作来阐述移动效果了。

同时，为了更好地在规划过程描述中，将操作步骤表达清晰，我们可以在基础的 action 二元组的基础上，将被移动的小车编号也放入动作信息中，可以用一个动态数组来表示这个动作。

动作生成时，这个动作只有两个要素，调用动作时也直接用数组的前两个编号调用即可。而在移动操作过程中，可以在移动同时读取被移动对象的小车编号，传入该数组的第三个位置，形成记忆，在回溯描述规划过程时，便可以从记忆中读取三元组，将移动操作的对象、起点、终点给描述清晰，即：

Action_result = （起点栈编号，终点栈编号，操作对象编号）

```
// 移动操作
vector<stack<CAR>> move(vector<stack<CAR>> state, vector<int>& action) {
    state[action[1]].push(state[action[0]].top());
    action.push_back(state[action[0]].top().carId);//将本次移动的小车编号传入action 中
    state[action[0]].pop();
    return state;}
```

3) 定义存储记忆机制：

基于广度优先，在广度优先队列中尝试搜索后，剔除每一层次不可行的操作，留下可行操作，将每一次完成操作尝试并确实改变了停车场状态的操作三元组（起点栈编号，终点栈编号，操作对象编号）给记忆下来，存入一个前置操作列表中，在完成操作搜索达到目标后，将该前置操作列表中的操作按顺序输出出来即可。

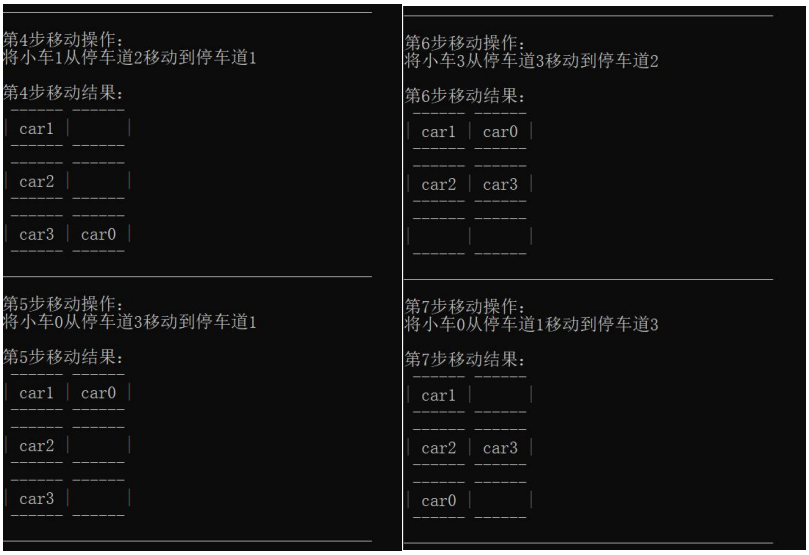
4) 选择并实现路径搜索算法：

基于一个传统的广度优先搜索并结合停车场状态节点和移动操作的定义，实现搜索与存储即可完成规划。

5) 实现规划过程可视化方法：

可视化表现分为两部分，一个是操作的文字描述，一个是操作的图形生成，都依据存储记忆机制实现

（3）求解结果：



三．迭代日志

1．三个经典规划问题迭代日志：

- 2022.4.10：伪代码分析
- 2022.4.10：伪代码翻译为 PDDL 代码
- 2022.4.10：在线编译通过得到推理结果

2．搬运小车问题迭代日志：

- 2022.3.10：数学建模
- 2022.3.10：PDDL 问题定义尝试
- 2022.3.11：PDDL 问题求解代码编写
- 2022.4.13：PDDL 问题求解代码编写失败
- 2022.6.5：八数码问题 C++代码编写
- 2022.6.5：八数码问题 C++代码迁移到搬运小车问题中
- 2022.6.5：搬运小车 C++代码编译通过

- 2022.6.5：基于广度优先搜索算法的搬运小车问题得到正确求解结果

四．总结

在分析经典规划问题的 PDDL 伪代码过程中，我初步了解了 PDDL 语言的运行逻辑，学习了如何用 PDDL 语言描述一个规划问题。

在编写 PDDL 代码的过程中，我熟悉了 PDDL 代码的运行思路，完成了由算法逻辑到 PDDL 代码实现的流程。

在求解搬运小车的规划问题时，我尝试用 PDDL 进行实现，最后虽然失败了，但也总结出 PDDL 的一些缺陷，同时，我通过学习八数码问题的求解思路，用基础编程语言，自己编写算法，完成了搬运小车问题的求解，并可视化展现了规划步骤，锻炼了自己的问题分析和求解能力。