人工智能基础实验一实验报告

PB19051035周佳豪

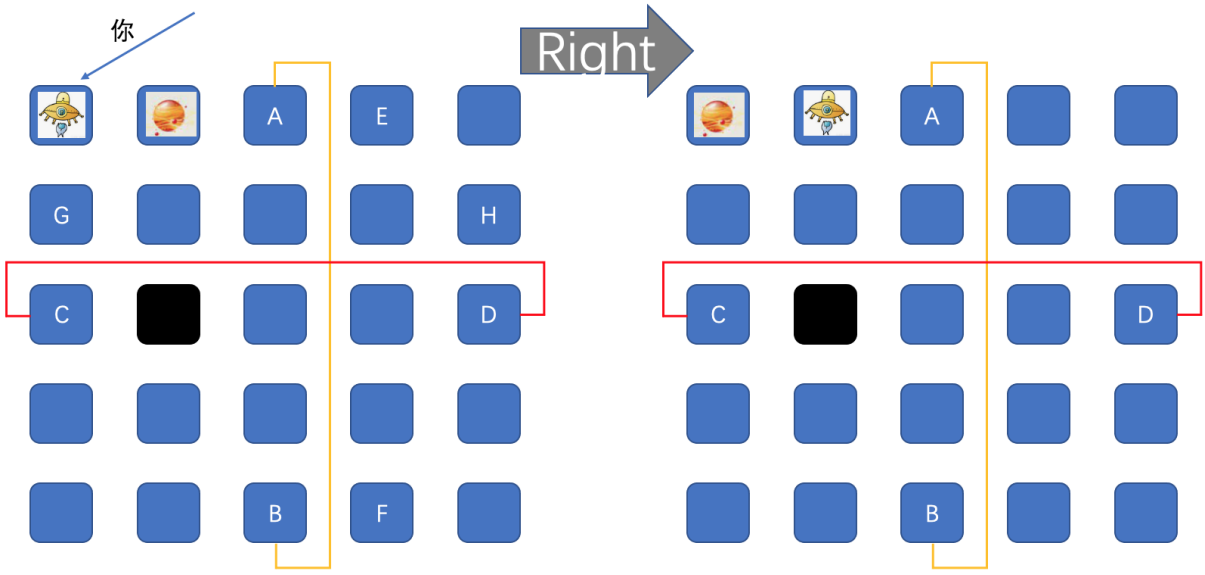
问题描述

1.1

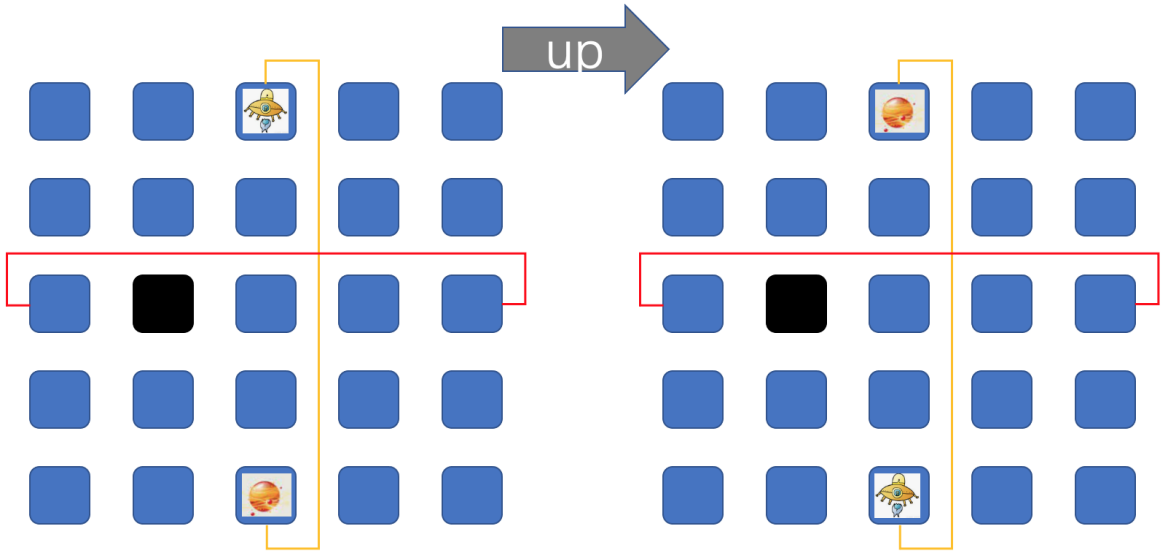
春日午后，人工智能基础课上，你犯起了春困，不小心就睡着了。。。
醒来后，你发现自己处在浩瀚的α星系，你的边上都是不同的星球，你拨弄了几下遥控手杆，发现如果你朝某个方向驾驶宇宙飞船，对应位置的星球会受到特定的电磁力和你互换位置，同时，宇宙中存在不可逾越的黑洞，以及神秘的星际通道。
这时指挥部给你下了任务，需要你驾驶宇宙飞船达到某个位置，同时使其他星球呈现某种特定排列。

输入:

我们将宇宙空间简单化为5*5的方格，其分布所示
每个星球位于其中一个方格内



其中ABCD所在区域存在星际通道（红黄线），表示你在A点往上走可以到达B点，在B点往下走可以到达A点，同理在C点往左走可以到达D点，在D点往右走可以到达C点。而其他边缘方格如GH，EF并无星际通道。黑色方格表示黑洞，为不可行走的区域。
星际通道如下图示意，在左下图的状态下，执行向上（UP）动作，会变成右下图状态：



输入分为初始星球分布和目标星球分布：

初始星球分布 对应input.txt：

```
0 2 3 4 5
1 7 8 9 10
6 12 13 14 11
16 17 18 19 -15
21 22 23 24 20
```

目标星球分布 对应target.txt：

```
1 2 3 4 5
6 7 8 9 10
11 12 13 19 14
16 17 18 24 -15
21 22 23 20 0
```

0 代表飞船，其他自然数表示不同编号的星球，黑洞是负数。所有数字互不相同，且绝对值均在[0-24]之间

输出：

输出一串动作系列（字符串形式），如 **DDLLDDR** 。
一共有4种动作，U表示向上，L表示向左，R表示向右，D表示向下。

1.2

CSP技术可以用来解决需要满足各种约束的日常生活中的调度问题。考虑下面这个场景，在一个车间中，共有 n 名工人：工人1，工人2，…， 工人n，每个工人根据工作年限都有自己的级别，比如工人1的级别是：senior，工人2的级别是：junior，现在需要 n 名工人设计一个工作表，比如将工人1和工人2安排到星期一，工人3和工人4安排到星期二，…，对于车间的工作安排，有以下5条约束要求：

1. 每个工人每周必须休息2天或2天以上
2. 每个工人每周不可以连续休息3天(不考虑跨周情况)
3. 周六周天也需要有人值班，即一周7天每天都要安排工人值班
4. 每天至少要有3个人值班
5. 每天至少要有一名级别为 senior 的工人值班

下面请大家为以下两个车间设计工作表：

1. 车间1：有7名工人
    - 工人1：senior
    - 工人2：senior
    - 工人3：junior
    - 工人4：junior
    - 工人5：junior
    - 工人6：junior
    - 工人7：junior

    由于车间规模较大，需要每天安排至少 4 个人值班，其余约束同上，其中工人1和工人4，工人2和工人3，工人3和工人6不想在同一天工作。
2. 车间2：有10名工人
    - 工人1：senior
    - 工人2：senior

- 工人3：junior
- 工人4：junior
- 工人5：junior
- 工人6：junior
- 工人7：junior
- 工人8：senior
- 工人9：junior
- 工人10：senior

由于车间规模较大，需要每天安排至少 5 个人值班，其余约束同上，其中工人1和工人5，工人2和工人6，工人8和工人10不想在同一天工作。

算法设计

1.1

以 **h1** 与 **IDAh1** 为例讲述算法思想，h2与IDAh2只需要改变启发式函数h，其余均相同。

h1

- 设计状态类 **state**,包含此时的星球分布，目标星期分布，以及如何从上一个状态转变到此时的状态(**last_move**)，指向前一个状态的指针(方便到达目标状态后递归求得路径)，**h**(启发式函数值)，**g**(已付出的代价)，**d**（h+g），**valid**（判断此状态是否合法）
- **state** 状态中包含 **state *move(char direction)** 函数，使得从该状态网某一个方向移动飞船，然后得到后继状态。一般情况下，每个状态都有4个后继状态，分别向四个方向move即可得到4个后继状态，但考虑到边界的存在，某些方向不可移动，故使用 **valid** 来判断某个后继状态是否合法，若合法则加入优先队列，否则舍弃该状态。对于隧道则只需要在边界是特殊考虑一下即可
- 关于优先队列，使用

```
priority_queue<state *, vector<state *>, cmp> open;
struct cmp
{
    bool operator()(state *a, state *b)
    {
        return a->get_h() + a->get_g() > b->get_h() + b->get_g();
    }
};
```

此时队列的头部始终是h+g最小的状态

- 关键代码

```
    open.push(start_state);
    while (!open.empty())
    {
        state *current = open.top();
        open.pop();
        vector<vector<int>> s = current->get_loaction();
        //判断是否已经到达目标
        if (current->is_over())
        {
            const state *temp = current;
            //cout << "d=" << temp->get_d() << endl;
            while (temp->get_last_state() != NULL)
            {
                close += temp->get_last_move();
                temp = temp->get_last_state();
            }
            for (int i = close.size() - 1; i >= 0; i--)
            {
                cout << close[i];
            }
            cout << endl;
            break;
        }
        //四个后继状态
        state *up_state = current->move('U');
        state *down_state = current->move('D');
        state *left_state = current->move('L');
        state *right_state = current->move('R');
        if (up_state->get_valid())
        {
            open.push(up_state);
        }
        if (down_state->get_valid())
        {
```

```
            open.push(down_state);
        }
        if (left_state->get_valid())
        {
            open.push(left_state);
        }
        if (right_state->get_valid())
        {
            open.push(right_state);
        }
    }
```

IDA_h1

- 状态类 **state** 与 **h1** 完全一样，不同的地方主要在于关键代码与优先队列
- 伪代码为

Algorithm 3 Iterative deepening A* search (IDA*)

```
1: d̂_limit ← d̂(s_0)
2: while d̂_limit < ∞ do
3:    next_d̂_limit ← ∞
4:    list ← {s_0}
5:    while list is not empty do
6:       s ← head(list)
7:       list ← rest(list)
8:       if d̂(s) > d̂_limit then
9:          next_d̂_limit ← min(next_d̂_limit, d̂(s))
10:      else
11:         if s is a goal then
12:            return s
13:         end if
14:         newstates ← apply actions to s
15:         list ← prepend(newstates, list)
16:      end if
17:   end while
18:   d̂_limit ← next_d̂_limit
19: end while
20: return fail
```

其中 d(s) 使用 状态s的g+h，优先队列使用栈

- 关键代码为：

```
state *start_state = new state(start, target, 0);
int d_limit = start_state->get_d();
bool is_end = false;
clock_t start_time, end_time;
start_time = clock();
int is_start = 0;
while (d_limit < INT_MAX)
    {
        cout << "d_limit: " << d_limit << endl;
        //每次循环都要清空栈
        while (!open.empty())
        {
            state *current = open.top();
            open.pop();
            delete current;
        }
        if (is_start == 0)
        {
            is_start = 1;
        }
        else
```

```
        {
            start_state = new state(start, target, 0);
        }
        int next_d_limit = INT_MAX;
        open.push(start_state);
        while (!open.empty())
        {
            state *current = open.top();
            open.pop();
            //更新next_d_limit
            if (current->get_d() > d_limit)
            {
                next_d_limit = min(next_d_limit, current->get_d());
            }
            else
            {
                //判断是否结束
                if (current->is_over())
                {
                    const state *temp = current;
                    while (temp->get_last_state() != NULL)
                    {
                        close += temp->get_last_move();
                        temp = temp->get_last_state();
                    }
                    for (int i = close.size() - 1; i >= 0; i--)
                    {
                        cout << close[i];
                    }
                    cout << endl;
                    end_time = clock();
                    cout << "time = " << double(end_time - start_time) / CLOCKS_PER_SEC << "s" << endl;
                    is_end = true;
                    break;
                }
                //4个后继状态
                state *up_state = current->move('U');
                state *down_state = current->move('D');
                state *left_state = current->move('L');
                state *right_state = current->move('R');
                if (up_state->get_valid())
                {
                    open.push(up_state);
                }
                if (down_state->get_valid())
                {
                    open.push(down_state);
                }
                if (left_state->get_valid())
                {
                    open.push(left_state);
                }
                if (right_state->get_valid())
                {
                    open.push(right_state);
                }
            }
        }
        d_limit = next_d_limit;
        if (is_end)
            break;
    }
```

h_2

使用状态类 **sub_state**，可以看作是 **state** 的子类，与 **state** 总体上完全一样，唯一不同的是启发式函数 h，使用考虑隧道的曼哈顿距离作为启发式函数。规定星际通道所在区域的四个地点分别为A,B,C,D。M(a,b)代表点a与点b的曼哈顿距离，s_t 表示星球 s 的目标地点，则 $h = \sum_{s \in stars} min\{M(s,s_t),\ M(s,A)+1+M(A,s_t),\ M(s,B)+1+M(B,s_t),\ M(s,C)+1+M(C,s_t), M(s,D)+1+M(D,s_t$,很明显h是一个乐观估计，故 h 是可采纳的启发式函数。

h_2的其余部分和h_1完全一样，这里不再赘述。

IDA_h2

使用状态类 **sub_state**,其余与IDA_h2完全相同，这里不再赘述

1.2

- 设计一个状态类 $state$ ，包含工人数量 `num_workers`,天数 `days`，工人某天是否工作 `vector<vector<bool>> workers`，工人在某天是否已经安排工作或休息 `vector<vector<bool>> visited`，工人的级别 `vector<*string*> level`，某两个工人不能同一天工作 `vector<vector<int>> conflicts`,一天至少有多少工人工作 `int one_day_workers`
- 实验采用的算法与 DFS 类似，维护一个栈储存状态，每个状态的后继状态均继续分配一个工人在某一天是否工作（故一个状态的后继状态有两个，分别为工作与休息），并使用前向检验优化此CSP算法,每当生成后继状态时，先判断该后继状态是否满足约束，若满足则入栈，否则舍弃
- 此问题能通过 $localsearch$算法，可根据每个状态的违反的约束数量为该状态进行打分，通过某个状态的局部变化得到邻居状态，选择分数最低(违反约束次数最少)的邻居状态，然后循环进行，直至找到不违法约束的状态。并规定最大迭代次数，若超过最大迭代次数，则重新随机一个初始状态，继续上述操作。

  伪代码为：

  ```
  int max_search_nums;//规定最大迭代次数
  while true:
      int loops=0;
      state = rand_state;//初始随机分配所有工人的工作
      while loops<max_search_nums:
          loops++;
          next_state = find_best_neighbor(state);//寻找违反约束次数最少的邻居
          if(next_state.score==0)
              return true;
          state = next_state;
  ```

- 变量集合：$\{worker[i][j]\}, i = 1, \dots, num\_workers$ ， $j = 1, \dots, days\}$
- 值域集合：每个 $worker[i][j]$ 的取值范围均为 $\{true, false\}$，取 true 代表第 i 个工人在第 j 天工作，取 false 则代表休息
- 约束集合 ：
  1. 每个工人每周必须休息2天或2天以上

     即对每个工人 i , 在days天中，$worker[i][j] = false$的次数都要大于等于2
  2. 每个工人每周不可以连续休息3天(不考虑跨周情况)

     即对每个工人 i，在days天中，都不能出现 $worker[i][j] = false, worker[i][j+1] = false, worker[i][j+2] = false$的情况，其中 $j + 2 <= days$
  3. 周六周天也需要有人值班，即一周7天每天都要安排工人值班

     对于每一天 j ， $worker[i][j] = true$的次数要大于0
  4. 每天至少要有 `one_day_workers` 个人值班

     对于每一天 j ， $worker[i][j] = true$的次数要大于等于 `one_day_workers`
  5. 每天至少要有一名级别为 senior 的工人值班

     对于每一天 j ， $worker[i][j] = true, level[i] = senior$的个数要大于等于1
  6. 工人m和n不能在同一天工作

     对于每一天j，不能出现$worker[m][j] = true, worker[n][j] = true$的情况

实验结果

1.1

h_1

样例编号	运行时间(单位:s)	移动序列	总步数
0	0.000	DDRUR	5
1	0.000	ULLUULDD	8
2	0.001	DDLUULLURR	10
3	0.002	DLDRRURRRUUURR	14
4	0.007	LUUURULLURDDRDR	15
5	0.030	LLUURRRUURDDDDLUURDD	20
6	0.024	DRDLLULULUUURDRURDRDRRR	23
7	0.003	URRRRDLLLLDRRRRDLLLLDRRRR	25
8	1.065	DRDLULLLDRUUUULDRRRRULDDDRD	27
9	20.771	RDRDLUUUURRDRDDRUUULLDRULURR	28
10	2.961	DDRRUUUULLULLUULLLLLUURRDDDDRR	30
11	217.123	DRUURDRRDRUULDLULDLDRDLDRURDRURD	32

IDA_h1

样例编号	运行时间(单位:s)	移动序列	总步数
0	0.002	DDRUR	5
1	0.002	ULLUULDD	8
2	0.001	DDLUULLURR	10
3	0.003	DLDRRURRRUUURR	14
4	0.013	LUUURULLURDDRDR	15
5	0.025	LLUURRRUURDDDDLUURDD	20
6	0.027	DRDLLULULUUURDRURDRDRRR	23
7	0.004	URRRRDLLLLDRRRRDLLLLDRRRR	25
8	1.002	DRDLULLLDRUUUULDRRRRULDDDRD	27
9	10.344	RRRRDRUUULDLDLLDRDLUUUURRURR	28
10	8.476	DDRRUUUULLULLUULLLLLUURRDDDDRR	30
11	276.608	DRUURDRRDRUULDLULDLDRDLDRURDRURD	32

h_2

样例编号	运行时间(单位:s)	移动序列	总步数
0	0.001	DDRUR	5
1	0.001	ULLUULDD	8
2	0.003	DDLUULLURR	10
3	0.005	DLDRRURRRUUURR	14
4	0.003	LUUURULLURDDRDR	15
5	0.005	LLUURRRUURDDDDLUURDD	20
6	0.02	DRDLLULULUUURDRURDRDRRR	23
7	0.004	URRRRDLLLLDRRRRDLLLLDRRRR	25
8	0.012	DRDLULLLDRUUUULDRRRRULDDDRD	27
9	1.104	RRRRDRUUULDLDLLDRDLUUUURRURR	28
10	0.005	DDRRUUUULLULLUULLLLLUURRDDDDRR	30
11	0.434	DRUURDRRDRUULDLULDLDRDLDRURDRURD	32

IDA_h2

样例编号	运行时间(单位:s)	移动序列	总步数
0	0.001	DDRUR	5
1	0.001	ULLUULDD	8
2	0.002	DDLUULLURR	10
3	0.003	DLDRRURRRUUURR	14
4	0.005	LUUURULLURDDRDR	15
5	0.004	LLUURRRUURDDDDLUURDD	20
6	0.009	DRDLLULULUUURDRURDRDRRR	23
7	0.005	URRRRDLLLLDRRRRDLLLLDRRRR	25
8	0.005	DRDLULLLDRUUUULDRRRRULDDDRD	27
9	0.541	RRRRDRUUULDLDLLDRDLUUUURRURR	28
10	0.005	DDRRUUUULLULLUULLLLLUURRDDDDRR	30
11	0.462	DRUURDRRDRUULDLULDLDRDLDRURDRURD	32

1.2

车间1

```
2 5 6 7
2 4 6 7
1 3 5 7
2 4 6 7
2 4 5 6
1 3 5 7
2 4 5 6
```

车间2

```
5 6 7 9 10
5 6 7 9 10
1 2 3 4 8
5 6 7 9 10
5 6 7 9 10
1 2 3 4 8
5 6 7 9 10
```