

P03 Planning and Uncertainty

学号	姓名	专业(方向)
18340052	何 泽	计算机科学与技术 (超级计算方向)
18340032	邓俊锋	计算机科学与技术 (大数据与人工智能方向)

I STRIPS planner

In this part, you will implement a simple STRIPS planner. The input of your planner is a PDDL domain file and a problem file in the STRIPS restriction, that is, preconditions of actions and the goal are conjunctions of atoms, and effects of actions are conjunctions of literals. The output of your planner is a sequence of actions to achieve the goal.

1

Describe with sentences the main ideas behind computing the heuristic for a state using reachability analysis from lecture notes.

One way to design heuristic function is to relax the original problem. The operation of STRIPS on action is to add and delete the state. So to make it easier to find the target state, we can ignore the impact of delete, and find the cost to the target. This is the reachability analysis.

2

Implement a STRIPS planner by using A* search and the heuristic function you implemented.

I use a pddl-parser on github and the link is here: [thiagopbueno/pypddl-parser: Domain and problem PDDL parser in Python3 using ply. \(github.com\)](https://github.com/thiagopbueno/pypddl-parser)

And the heuristic function is shown below:

```
1 def heuristiccc(fringe, ground_actions, visited, goal_pos,
2   goal_not):
3     while fringe:
4       state = fringe.pop(0)
5       plan = fringe.pop(0)
6       for act in ground_actions:
7         if applicable(state, act.positive_preconditions,
8           act.negative_preconditions):
8           new_state = apply(state, act.add_effects, []) # ignore the impact of delete
9           if new_state not in visited:
10             if applicable(new_state, goal_pos, goal_not):
11               full_plan = [act]
12               while plan:
13                 act, plan = plan
14                 full_plan.insert(0, act)
15               return len(full_plan)
16               visited.append(new_state)
17               fringe.append(new_state)
18               fringe.append((act, plan))
18     return -1
```

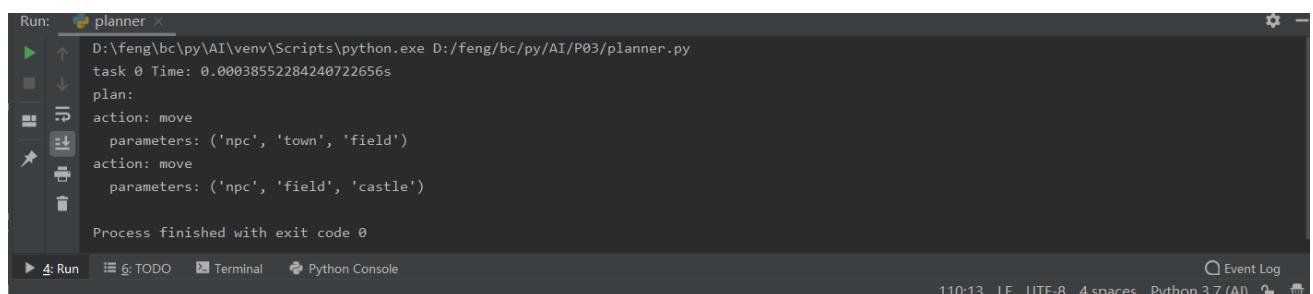
3

Explain any ideas you use to speed up the implementation.

My idea to speed up the implementation is to compute the heuristic function lazily. Because of the high cost of computing the real heuristic function, I think sometimes we only need to calculate to a certain size, such as 3. As long as it can be used for comparison.

Run your planner on the 5 test cases, and report the returned plans and the running times.
 Analyse the experimental results.

- task 0 :

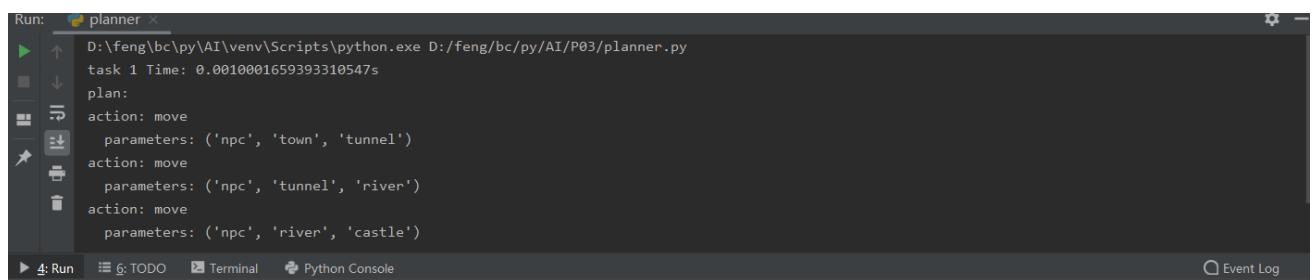


```
Run: planner
D:\feng\bc\py\AI\venv\Scripts\python.exe D:/feng/bc/py/AI/P03/planner.py
task 0 Time: 0.0003855284240722656s
plan:
action: move
  parameters: ('npc', 'town', 'field')
action: move
  parameters: ('npc', 'field', 'castle')

Process finished with exit code 0
```

Run: 4: Run TODO Terminal Python Console Event Log 110:13 LF UTF-8 4 spaces Python 3.7 (AI)

- task 1:

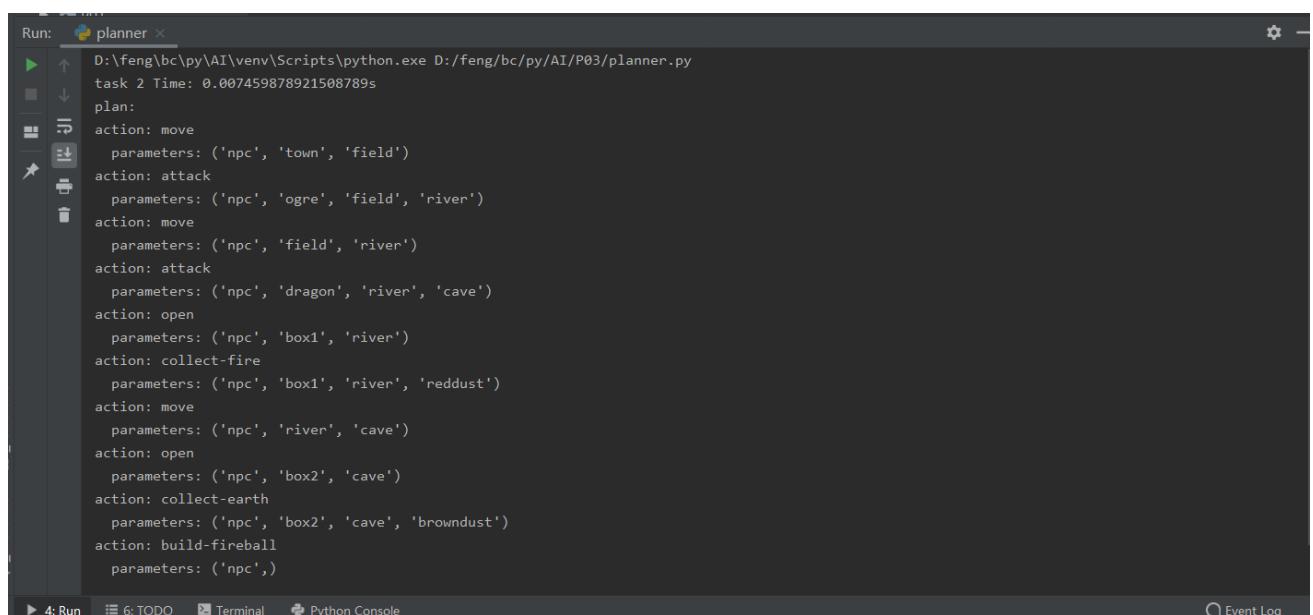


```
Run: planner
D:\feng\bc\py\AI\venv\Scripts\python.exe D:/feng/bc/py/AI/P03/planner.py
task 1 Time: 0.0010001659393310547s
plan:
action: move
  parameters: ('npc', 'town', 'tunnel')
action: move
  parameters: ('npc', 'tunnel', 'river')
action: move
  parameters: ('npc', 'river', 'castle')

Process finished with exit code 0
```

Run: 4: Run TODO Terminal Python Console Event Log

- task 2:



```
Run: planner
D:\feng\bc\py\AI\venv\Scripts\python.exe D:/feng/bc/py/AI/P03/planner.py
task 2 Time: 0.007459878921508789s
plan:
action: move
  parameters: ('npc', 'town', 'field')
action: attack
  parameters: ('npc', 'ogre', 'field', 'river')
action: move
  parameters: ('npc', 'field', 'river')
action: attack
  parameters: ('npc', 'dragon', 'river', 'cave')
action: open
  parameters: ('npc', 'box1', 'river')
action: collect-fire
  parameters: ('npc', 'box1', 'river', 'reddust')
action: move
  parameters: ('npc', 'river', 'cave')
action: open
  parameters: ('npc', 'box2', 'cave')
action: collect-earth
  parameters: ('npc', 'box2', 'cave', 'browndust')
action: build-fireball
  parameters: ('npc',)
```

Run: 4: Run TODO Terminal Python Console Event Log

- task 3:

```
Run: planner x
D:\feng\bc\py\AI\venv\Scripts\python.exe D:/feng/bc/py/AI/P03/planner.py
task 3 Time: 0.002999544143676758s
plan:
action: unstack
parameters: ('b', 'a', 'x')
action: move
parameters: ('a', 'x', 'y')
action: move
parameters: ('b', 'x', 'y')
action: stack
parameters: ('a', 'b', 'y')

▶ Run TODO Terminal Python Console Event Log
```

- task 4:

```
Run: planner x
D:\feng\bc\py\AI\venv\Scripts\python.exe D:/feng/bc/py/AI/P03/planner.py
task 4 Time: 0.002514362335205078s
plan:
action: unstack
parameters: ('b', 'a', 't1', 't3')
action: stack
parameters: ('a', 't1', 'b', 't3')

Process finished with exit code 0

▶ Run TODO Terminal Python Console Event Log
```

It can be seen that, the results are all right.

II Diagnosing by Bayesian Networks

1. Variables and their domains

- PatientAge:[‘0=30’, ‘31=65’, ‘65+’]
- CTScanResult : [‘ Ischemic Stroke ’ , ‘ Hemorrhagic Stroke ’]
- MRIScanResult: [‘Ischemic Stroke’, ‘Hemorrhagic Stroke’]
- StrokeType : [‘ Ischemic Stroke ’ , ‘ Hemorrhagic Stroke ’ , ‘ Stroke Mimic ’]
- Anticoagulants : [‘ Used ’ , ‘ Not used ’]
- Mortality :[‘True’, ‘False’]
- Disability : [‘ Negligible ’ , ‘Moderate ’ , ‘Severe ’]

2. CPTs

Note: [CTScanResult, MRIScanResult, StrokeType] means:

$P(StrokeType = \dots | CTScanResult = \dots \wedge MRIScanResult = \dots)$

- [PatientAge]

['0-30', 0.10],
['31-65', 0.30],
['65+', 0.60]

- [CTScanResult]

['Ischemic Stroke', 0.7],
['Hemorrhagic Stroke', 0.3]

- [MRIScanResult]

['Ischemic Stroke', 0.7],
['Hemorrhagic Stroke', 0.3]

- [Anticoagulants]

['Used', 0.5],
['Not used', 0.5]

- [CTScanResult, MRIScanResult, StrokeType]

['Ischemic Stroke', 'Ischemic Stroke', 'Ischemic Stroke', 0.8],
['Ischemic Stroke', 'Hemorrhagic Stroke', 'Ischemic Stroke', 0.5],
['Hemorrhagic Stroke', 'Ischemic Stroke', 'Ischemic Stroke', 0.5],
['Hemorrhagic Stroke', 'Hemorrhagic Stroke', 'Ischemic Stroke', 0],

```
[['Ischemic Stroke', 'Ischemic Stroke', 'Hemorrhagic Stroke', 0],  
 ['Ischemic Stroke', 'Hemorrhagic Stroke', 'Hemorrhagic Stroke', 0.4],  
 ['Hemorrhagic Stroke', 'Ischemic Stroke', 'Hemorrhagic Stroke', 0.4],  
 ['Hemorrhagic Stroke', 'Hemorrhagic Stroke', 'Hemorrhagic Stroke', 0.9],  
  
[['Ischemic Stroke', 'Ischemic Stroke', 'Stroke Mimic', 0.2],  
 ['Ischemic Stroke', 'Hemorrhagic Stroke', 'Stroke Mimic', 0.1],  
 ['Hemorrhagic Stroke', 'Ischemic Stroke', 'Stroke Mimic', 0.1],  
 ['Hemorrhagic Stroke', 'Hemorrhagic Stroke', 'Stroke Mimic', 0.1],
```

- [StrokeType, Anticoagulants, Mortality]

```
[['Ischemic Stroke', 'Used', 'False', 0.28],  
 ['Hemorrhagic Stroke', 'Used', 'False', 0.99],  
 ['Stroke Mimic', 'Used', 'False', 0.1],  
 ['Ischemic Stroke', 'Not used', 'False', 0.56],  
 ['Hemorrhagic Stroke', 'Not used', 'False', 0.58],  
 ['Stroke Mimic', 'Not used', 'False', 0.05],  
  
[['Ischemic Stroke', 'Used', 'True', 0.72],  
 ['Hemorrhagic Stroke', 'Used', 'True', 0.01],  
 ['Stroke Mimic', 'Used', 'True', 0.9],  
 ['Ischemic Stroke', 'Not used', 'True', 0.44],  
 ['Hemorrhagic Stroke', 'Not used', 'True', 0.42 ],  
 ['Stroke Mimic', 'Not used', 'True', 0.95]]
```

- [StrokeType, PatientAge, Disability]

```
[['Ischemic Stroke', '0-30', 'Negligible', 0.80],  
 ['Hemorrhagic Stroke', '0-30', 'Negligible', 0.70],  
 ['Stroke Mimic', '0-30', 'Negligible', 0.9],  
 ['Ischemic Stroke', '31-65', 'Negligible', 0.60],  
 ['Hemorrhagic Stroke', '31-65', 'Negligible', 0.50],  
 ['Stroke Mimic', '31-65', 'Negligible', 0.4],
```

```
[['Ischemic Stroke',    '65+', 'Negligible',0.30],  
 ['Hemorrhagic Stroke', '65+', 'Negligible',0.20],  
 ['Stroke Mimic',      '65+', 'Negligible',0.1],  
  
[['Ischemic Stroke',    '0-30', 'Moderate',0.1],  
 ['Hemorrhagic Stroke', '0-30', 'Moderate',0.2],  
 ['Stroke Mimic',      '0-30', 'Moderate',0.05],  
 [['Ischemic Stroke',    '31-65','Moderate',0.3],  
 ['Hemorrhagic Stroke', '31-65','Moderate',0.4],  
 ['Stroke Mimic',      '31-65','Moderate',0.3],  
 [['Ischemic Stroke',    '65+', 'Moderate',0.4],  
 ['Hemorrhagic Stroke', '65+', 'Moderate',0.2],  
 ['Stroke Mimic',      '65+', 'Moderate',0.1],  
  
[['Ischemic Stroke',    '0-30', 'Severe',0.1],  
 ['Hemorrhagic Stroke', '0-30', 'Severe',0.1],  
 ['Stroke Mimic',      '0-30', 'Severe',0.05],  
 [['Ischemic Stroke',    '31-65','Severe',0.1],  
 ['Hemorrhagic Stroke', '31-65','Severe',0.1],  
 ['Stroke Mimic',      '31-65','Severe',0.3],  
 [['Ischemic Stroke',    '65+', 'Severe',0.3],  
 ['Hemorrhagic Stroke', '65+', 'Severe',0.6],  
 ['Stroke Mimic',      '65+', 'Severe',0.8]]
```

3. Tasks

①

Briefly describe with sentences the main ideas of the VE algorithm.

VE即为Variable Elimination，变量消除算法，即给定贝叶斯网络，条件概率表F，询问Q，证据E，其余变量为Z，计算 $P(Q|E)$ 的算法，步骤如下：

- 对于 $f \in F$ 中每一变量，将其替换为 $f_{E=e}$

- 对于每一 $Z_j \in Z$, 按给定 Z_j 顺序, 并按照以下步骤消除:
 - f_1, f_2, \dots, f_k 为含有 Z_j 的因子
 - 计算新的因子 $g_j = \sum_{Z_j} f_1 \times f_2 \times \dots \times f_k$
 - 将 f_i 从 F 中移除, 并将新的因子 g_j 添加到 F 中
- 剩下的因子只包含询问 Q 中的变量, 则计算它们的乘积归一化得到 $P(Q | E)$

②

Implement the VE algorithm (C++ or Python) to calculate the following probability values:

- $p1 = P(\text{Mortality}=\text{'True'} \wedge \text{CTScanResult}=\text{'Ischemic Stroke'} | \text{PatientAge}=\text{'31-65'})$
- $p2 = P(\text{Disability}=\text{'Moderate'} \wedge \text{CTScanResult}=\text{'Hemorrhagic Stroke'} | \text{PatientAge}=\text{'65+'} \wedge \text{MRIScanResult}=\text{'Hemorrhagic Stroke'})$
- $p3 = P(\text{StrokeType}=\text{'Hemorrhagic Stroke'} | \text{PatientAge}=\text{'65+'} \wedge \text{CTScanResult}=\text{'Hemorrhagic Stroke'} \wedge \text{MRIScanResult}=\text{'Ischemic Stroke'})$
- $p4 = P(\text{Anticoagulants}=\text{'Used'} | \text{PatientAge}=\text{'31-65'})$
- $p5 = P(\text{Disability}=\text{'Negligible'})$

- 首先, VE算法的实现在E09中已完成, 并且变量消除的顺序是依据输入的变量列表的, 此时VE算法如下:

```

1  class VariableElimination:
2      @staticmethod
3      def inference(factorList, queryVariables,
4                    orderdListOfHiddenVariables, evidenceList):
5          for evidence in evidenceList:
6              for factor in factorList:
7                  if evidence in factor.varList:
8
9                      factorList.append(factor.restrict(evidence,
10                           evidenceList[evidence]))
11                     factorList.remove(factor)
12
13             for variable in orderdListOfHiddenVariables:
14                 eliminationList = list(filter(lambda factor:
15                     variable in factor.varList, factorList))

```

```

12             new_var = eliminationList[0]
13
14             for e in eliminationList:
15                 for i in factorList:
16                     if i.name == e.name:
17                         factorList.remove(i)
18                     if not e == eliminationList[0]:
19                         new_var = new_var.multiply(e)
20
21             new_var = new_var.sumout(variable)
22             factorList.append(new_var)
23
24             print("RESULT:")
25             res = factorList[0]
26             for factor in factorList[1:]:
27                 res = res.multiply(factor)
28             total = sum(res.cpt.values())
29             res.cpt = {k: v/total for k, v in res.cpt.items()}
30             res.printInf()
31
32
33     @staticmethod
34     def printFactors(factorList):
35         for factor in factorList:
36             factor.printInf()
37
38
39     class Util:
40         @staticmethod
41         def to_binary(num, len):
42             return format(num, '0' + str(len) + 'b')
43
44
45     class Node:
46         def __init__(self, name, var_list):
47             self.name = name
48             self.varList = var_list
49             self.cpt = {}
50
51         def setCpt(self, cpt):
52             self.cpt = cpt
53
54         def printInf(self):
55             print("Name = " + self.name)
56             print(" vars " + str(self.varList))
57             for key in self.cpt:
58                 print("   key: " + key + " val : " +
59                      str(self.cpt[key]))

```

```

50     print()
51     def multiply(self, factor):
52         newList = [var for var in self.varList]
53         new_cpt = {}
54         idx1 = []
55         idx2 = []
56         for var1 in self.varList:
57             for var2 in factor.varList:
58                 if var1 == var2:
59                     idx1.append(self.varList.index(var1))
60                     idx2.append(factor.varList.index(var2))
61                 else:
62                     newList.append(var2)
63         for k1, v1 in self.cpt.items():
64             for k2, v2 in factor.cpt.items():
65                 flag = True
66                 for i in range(len(idx1)):
67                     if k1[idx1[i]] != k2[idx2[i]]:
68                         flag = False
69                         break
70                 if flag:
71                     new_key = k1
72                     for i in range(len(k2)):
73                         if i in idx2: continue
74                         new_key += k2[i]
75                     new_cpt[new_key] = v1 * v2
76         new_node = Node("f" + str(newList), newList)
77         new_node.setCpt(new_cpt)
78         return new_node
79     def sumout(self, variable):
80         new_var_list = [var for var in self.varList]
81         new_var_list.remove(variable)
82         new_cpt = {}
83         idx = self.varList.index(variable)
84         for k, v in self.cpt.items():
85             if k[:idx] + k[idx+1:] not in new_cpt.keys():
86                 new_cpt[k[:idx] + k[idx+1:]] = v
87             else: new_cpt[k[:idx] + k[idx+1:]] += v
88         new_node = Node("f" + str(new_var_list), new_var_list)

```

```

89         new_node.setCpt(new_cpt)
90
91     def restrict(self, variable, value):
92         new_var_list = [i for i in self.varList]
93         new_var_list.remove(variable)
94         new_cpt = {}
95         idx = self.varList.index(variable)
96         for k, v in self.cpt.items():
97             if k[idx] == str(value):
98                 new_cpt[k[:idx] + k[idx+1:]] = v
99
100    new_node = Node("f" + str(new_var_list), new_var_list)
101    new_node.setCpt(new_cpt)
102
103    return new_node

```

- 由上，调用VE的方法如下：

```

1 | VariableElimination.inference(factorList,queryVariables,orderdLi
| stOfHiddenVariables,evidenceList)

```

例如，

$p1 = P(\text{Mortality} = \text{'True'} \wedge \text{CTScanResult} = \text{'IschemicStroke'} | \text{PatientAge} = 31 - 65)$

的调用代码如下：

```

1 | VariableElimination.inference(
2 |   [P, M, C, A, S, D, Mo], ['Mo', 'C'],
3 |   ['M', 'A', 'S', 'D'], {'P': 1})

```

其余的类似，调用代码便不再举例。

之后便可以运行计算，结果如下：

```

heze@HeZes-MacBook-Pro:~/大三上/人工智能实验/P03$ python VE.py
p1 = P(Mortality='True' ∧ CTScanResult='Ischemic Stroke' | PatientAge='31-65')
RESULT:
Name = f['C', 'Mo']
vars ['C', 'Mo']
key: 00 val : 0.283605
key: 01 val : 0.4163949999999999
key: 10 val : 0.1758749999999999
key: 11 val : 0.12412500000000001

```

```
key: 11 val : 0.12412300000000001

-----
p2=P(Disability='Moderate' ∧ CTScanResult='HemorrhagicStroke' | PatientAge='65+' ∧ MRIScanResult='Hemorrhagic Stroke')
RESULT:
Name = f['C', 'D']
vars ['C', 'D']
key: 00 val : 0.16800000000000004
key: 01 val : 0.20300000000000004
key: 02 val : 0.329
key: 10 val : 0.057
key: 11 val : 0.057
key: 12 val : 0.18600000000000005

-----
p3=P(StrokeType='HemorrhagicStroke' | PatientAge='65+' ∧ CTScanResult='HemorrhagicStroke' ∧ MRIScanResult='Ischemic Stroke')
RESULT:
Name = f['S']
vars ['S']
key: 0 val : 0.5000000000000001
key: 1 val : 0.4
key: 2 val : 0.1000000000000002

-----
p4 = P(Anticoagulants='Used' | PatientAge='31-65')
RESULT:
Name = f['A']
vars ['A']
key: 0 val : 0.5000000000000001
key: 1 val : 0.4999999999999994

-----
p5 = P(Disability='Negligible')
RESULT:
Name = f['D']
vars ['D']
key: 0 val : 0.38977
key: 1 val : 0.292515
key: 2 val : 0.317715
```

```
[ heze@HeZes-MacBook-Pro .../大三上/人工智能实验/P03 main ✘ 19:06:22 ]
```

③

Implement an algorithm to select a good order of variable elimination.

我的实现方法为优先消除临接边最少的变量：

```
1 def select_order(orderedListOfHiddenVariables, factorList):
2     for var in orderedListOfHiddenVariables:
3         c = 0
4         for times in List:
5             if var in times.varList:
6                 c += 1
7         if b is None or c < b:
8             a = var
9             b = c
10    return a
```

结果如下：

```
..智能实验/P03 (-zsh)
heze@HeZes-MacBook-Pro ..大三上/人工智能实验/P03 main 20:00:18
python VE_select.py
p1 = P(Mortality='True' ∧ CTScanResult='Ischemic Stroke' | PatientAge='31-65')
RESULT:
Name = f['C', 'Mo']
vars ['C', 'Mo']
key: 00 val : 0.283605
key: 01 val : 0.4163949999999999
key: 10 val : 0.1758749999999998
key: 11 val : 0.1241250000000001

-----
p2=P(Disability='Moderate' ∧ CTScanResult='HemorrhagicStroke' | PatientAge='65+' ∧ MRIScanResult='Hemorrhagic Stroke')
RESULT:
Name = f['C', 'D']
vars ['C', 'D']
key: 00 val : 0.1680000000000004
key: 01 val : 0.2030000000000004
key: 02 val : 0.329
key: 10 val : 0.057
key: 11 val : 0.057
key: 12 val : 0.1860000000000005

-----
p3=P(StrokeType='HemorrhagicStroke' | PatientAge='65+' ∧ CTScanResult='HemorrhagicStroke' ∧ MRIScanResult='Ischemic Stroke')
RESULT:
Name = f['S']
vars ['S']
key: 0 val : 0.5000000000000001
key: 1 val : 0.4
key: 2 val : 0.1000000000000002
```

```
key: 2 val : 0.10000000000000002
```

```
p4 = P(Anticoagulants='Used' | PatientAge='31-65')
RESULT:
Name = f['A']
vars ['A']
key: 0 val : 0.5000000000000001
key: 1 val : 0.4999999999999994
```

```
p5 = P(Disability='Negligible')
RESULT:
Name = f['D']
vars ['D']
key: 0 val : 0.38977
key: 1 val : 0.292515
key: 2 val : 0.317715
```

```
heze@HeZes-MacBook-Pro ~ /.../大三上/人工智能实验/P03 main ↻ 20:00:40
```

可以看出计算结果与之前是一致的。

④

Compare the running times of the VE algorithm for different orders of variable elimination, and fill out the following table:

For test cases p4 and p5, for each of the order selected by your algorithm and 5 other orders, report the elimination width, and the total running time of the VE algorithm. For each case, the first order of elimination should be the one chosen by your algorithm. Analyze the results.

由于单次的计算时间较短，总时间我统计的是计算了1000次之后的运行时间总和

Test Case	Elimination order	Elimination width	Total time
p4	Mo->D->C->MR->S	3	1.185396s
p4	MR->C->S->Mo->D	3	1.247903s
p4	C->MR->S->Mo->D	3	1.230876s
p4	MR->S->Mo->C->D	4	1.415672s
p4	MR->S->Mo->D->C	4	1.396785s
p4	MR->S->C->Mo->D	4	1.403376s
p5	Mo->P->C->MR->A->S	3	1.419637s
p5	P->MR->C->A->S->Mo	3	1.422635s
p5	A->P->C->MR->S->Mo	3	1.417334s
p5	P->MR->S->A->C->Mo	4	1.629973s
p5	S->A->P->MR->C->Mo	6	2.793629s
p5	S->A->P->C->MR->Mo	6	2.800693s

- 首先，可以从结果看出前一题的顺序选择算法是正确的，因为Elimination width都是最小的
- 另外可以看出实际的运行时间和Elimination width的相关性很高，呈正相关的趋势，但是和实际的Elimination order关系不大