

# 人工智能导论实验报告

题 目	人 工 智 能 实 验
姓 名	郭 炼
学 号	1 1 8 0 1 0 0 2 1 7
院 系	计 算 学 部
专 业	数据科学与大数据技术
指导老师	李 钦 策
同组成员	吕岸桐、卢林烜、李衡

哈爾濱工業大學

# 目录

零 前言	3
1 实验环境说明	3
一 实验一 知识表示	4
1 问题描述	4
1.1 基本要求	4
1.2 猴子取香蕉问题	4
2 算法介绍	4
2.1 产生式表示法	4
2.2 解决方案	5
3 算法实现	5
4 讨论及结论	5
二 实验二 搜索策略	6
1 问题描述	6
2 算法介绍	6
2.1 问题一 应用深度优先算法找到一个特定的位置的豆	6
2.2 问题二 应用宽度优先算法找到一个特定的位置的豆	8
2.3 问题三 应用代价一致算法找到一个特定的位置的豆	9
2.4 问题四 应用 A* 算法找到一个特定的位置的豆	9
2.5 问题五 找到所有的角落	10
2.6 问题六 角落问题（启发式）	11
2.7 问题七 吃掉所有的豆子	12
2.8 问题八 次最优搜索	13
3 算法实现	14
4 讨论及结论	19
三 实验三 不确定性推理	20
1 问题描述	20

1.1	基本要求 . . . . .	20
1.2	输入输出描述 . . . . .	20
2	算法介绍 . . . . .	21
2.1	基本原理 . . . . .	21
2.2	代码实现 . . . . .	21
3	算法实现 . . . . .	22
4	讨论及结论 . . . . .	23

# 第 零 章 前言

## 1 实验环境说明

在本次实验中，我们小组使用的环境如下：

- 操作系统：Arch Linux x86\_64
- 内存：4GB @ 2666MHz × 2
- CPU：Intel i5-8250U (8) @ 3.400GHz
- Golang 编译器版本：go version go1.15.4 linux/amd64
- Python 解释器版本：Python 2.7.18
- C++ 编译器版本：gcc (GCC) 10.2.0

# 第一章 实验一 知识表示

## 1 问题描述

### 1.1 基本要求

实验一要求我们参照课程第二部分讲授的知识表示的方式，运用包括产生式系统、框架系统、语义网络等方法求解猴子取香蕉的问题。

### 1.2 猴子取香蕉问题

一个房间里，天花板上挂有一串香蕉，有一只猴子可在房间里任意活动（到处走动，推移箱子，攀登箱子等）。设房间里还有一只可被猴子移动的箱子，且猴子登上箱子时才能摘到香蕉，问猴子在某一状态下（设猴子位置为 A，香蕉位置在 B，箱子位置为 C），如何行动可摘取到香蕉。

## 2 算法介绍

### 2.1 产生式表示法

产生式表示法是一种知识表示方法，其基本形式为  $P \rightarrow Q$ （或 IF P THEN Q）。其中，P 是产生式的前提，也称为前件，它给出了该产生式可否使用的先决条件。Q 是一组结论或操作，也称为后件，它指出当 P 满足时，应该推出的结论或应该执行的动作。

产生式系统的基本结构包括如下三个部分：

- 综合数据库，用于存放推理过程的各种当前信息以及作为推理过程选择可用规则的依据。
- 规则库，用于存放推理所需要的所有规则，是整个产生式系统的知识集，是产生式系统能够进行推理的根本。
- 控制系统，用于控制整个产生式系统的运行，决定问题求解过程的推理线路。

## 2.2 解决方案

根据产生式表示法，我们定义如下部分

- 综合数据库: (Monkey, Banana, Box, On, Have), 其中 Monkey 表示猴子的位置, Banana 表示香蕉的位置, Box 表示箱子的位置, On 表示猴子是否在箱子上 (0 表示否, 1 表示是), Have 表示猴子有没有抓到香蕉 (同上)。

- 规则数据库包含的规则如下:

r1: IF (x, y, z, 0, 0) THEN (w, y, z, 0, 0)

r2: IF (x, y, x, 0, 0) THEN (z, y, z, 0, 0)

r3: IF (x, y, x, 0, 0) THEN (x, y, x, 1, 0)

r4: IF (x, y, x, 1, 0) THEN (x, y, x, 0, 0)

r5: IF (x, x, x, 1, 0) THEN (x, x, x, 1, 1)

根据问题, 确定终止状态是猴子摘到香蕉, 所以终止状态为 (C, C, C, 1, 1), 而初始状态为 (A, B, C, 0, 0)。控制系统需要做的就是根据初始状态和规则库中的规则, 不断地产生新的状态, 直到到达目标状态为止。

## 3 算法实现

在本实验中, 我们使用 Golang (版本 1.15.4 linux/amd64) 编写程序, 程序编译运行的结果如下:

```
./main
Monkey移动到了C点
Box移动到了B点
Monkey移动到了B点
Monkey移动到了Bon点
香蕉拿到了!
Monkey移动到了B点
```

图 3.1: 实验一运行结果

## 4 讨论及结论

该实验让我们了解了如何利用知识表示方法解决一个具体的问题, 知道了知识表示方法解决问题的具体步骤。

在综合了各个表示方法的优缺点后, 我们最终选择了产生式的表示方法。通过构建了产生式的组成 (综合数据库、规则库以及控制系统), 并定义了初始状态和目标状态, 我们最终找到了一条从初始状态到目标状态的路径进而解决了猴子取香蕉问题。

# 第二章 实验二 搜索策略

## 1 问题描述

实验二要求采用且不限于课程第四章内各种搜索算法，编写一系列吃豆人程序解决下面列出的 8 个问题，包括到达指定位置以及有效地吃豆等。具体的八个问题如下：

问题 1：应用深度优先算法找到一个特定的位置的豆

问题 2：应用宽度优先算法找到一个特定的位置的豆

问题 3：应用代价一致算法找到一个特定的位置的豆

问题 4：应用 A\* 算法找到一个特定的位置的豆

问题 5：找到所有的角落

问题 6：角落问题（启发式）

问题 7：吃掉所有的豆子

问题 8：次最优搜索

在本实验中，我们要使用深度优先算法（DFS），宽度优先算法（BFS），代价一致算法（UCS），A\* 算法等搜索算法来为吃豆人规划路径以完成特定的目标。其中前四个问题为本质是实现一个朴素的搜索算法，后四个问题则是为具体的搜索问题设计不同的搜索策略和估价函数。

## 2 算法介绍

在本章，我们将针对简单介绍实验二吃豆人中的八个问题，分析各个问题并说明解决问题的方法及算法实现。

### 2.1 问题一 应用深度优先算法找到一个特定的位置的豆

#### 2.1.1 问题描述

在本问题以及后续的三个问题中，我们需要为吃豆人规划一条在给定迷宫中从特定起点到特定终点的路径。在问题一中，我们需要使用深度优先搜索算法求

解该问题。

值得注意的是，前四个问题的问题其实是相似的，唯一的不同就是其使用的搜索算法不同。因为不同的搜索方法的不同之处仅仅在于 **open** 表的排序不同，因此我们可以使用一个通用的搜索算法解决问题一到四，通过修改数据结构来实现不同的搜索算法。

### 2.1.2 算法实现

我们首先需要实现一个通用的搜索算法，实验大纲中附录的伪代码如下：

## Graph Search Pseudo-Code

```
function GRAPH-SEARCH(problem, fringe) return a solution, or failure
  closed ← an empty set
  fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
  loop do
    if fringe is empty then return failure
    node ← REMOVE-FRONT(fringe)
    if GOAL-TEST(problem, STATE[node]) then return node
    if STATE[node] is not in closed then
      add STATE[node] to closed
      for child-node in EXPAND(STATE[node], problem) do
        fringe ← INSERT(child-node, fringe)
      end
    end
  end
```

图 2.1: 通用搜索算伪代码

在具体实现的过程中，我们使用一个三元组来代表一个 **node**，分别存储当前状态的位置，路径（即从初始状态到达本状态所要执行的操作序列）和到达当前点的代价。由于 **PriorityQueue** 的 **push** 方法同其他数据结构不同，所以对其需要特判。为了实现后面的 **A\*** 算法，我们需要传入一个 **heuristic** 参数来进行估价，其默认值为 **nullHeuristic**，该函数始终返回 0（即不进行估计）。如果不存在一条从起点到终点的路径，那么代码会返回特殊值 **None** 表示无解。具体的代码实现如下：

```
1 def generic_search(problem, open_table_type, heuristic=nullHeuristic):
2     """
3     problem: the problem to solve
4     open_table_type: the type of the open table to imply different search algorithm
5     heuristic: the heuristic function
6
7     A general search method.
8     """
9     open_table = open_table_type()
10    is_priority_queue = isinstance(open_table, util.PriorityQueue)
11
12    def push_to_open_table(state, actions, cost):
13        if is_priority_queue:
```



```

14         open_table.push((state, actions, cost),
15                           cost + heuristic(state, problem))
16     else:
17         open_table.push((state, actions, cost))
18
19     closed_table = set()
20     push_to_open_table(problem.getStartState(), [], 0)
21     while not open_table.isEmpty():
22         state, actions, cost = open_table.pop()
23         if state in closed_table: continue
24         closed_table.add(state)
25         if problem.isGoalState(state):
26             return actions
27         for successor, action, step_cost in problem.getSuccessors(state):
28             push_to_open_table(successor, actions + [action], cost + step_cost)
29     return None

```

在深度优先搜索算法中，程序会首先扩展最新产生的（即最深的）节点，当前结点无法拓展时才会回溯到祖先结点，采用的是后生成的节点先扩展的策略。所以深度优先搜索算法的 `open` 表应该使用先进后出（FILO）的栈结构来实现。基于前面已经实现的通用的搜索算法，我们只需要将 `open_table_type` 设置为栈即可。

```

1 def depthFirstSearch(problem):
2     ....
3     return generic_search(problem, util.Stack)

```

## 2.2 问题二 应用宽度优先算法找到一个特定的位置的豆

### 2.2.1 问题描述

在本问题中，我们需要使用深度优先搜索算法为吃豆人规划一条在给定迷宫中从特定起点到特定终点的路径。

### 2.2.2 算法实现

宽度优先搜索算法的基本思想是从初始结点开始扩展，当且仅当第  $n$  层的结点全部扩展完之后，才会开始扩展下一层结点，采用的时先生成的结点先扩展的策略。所以宽度优先搜索算法的 `open` 表应该使用先进先出（FIFO）的队列结构来实现。基于前面已经实现的通用的搜索算法，我们只需要将 `open_table_type` 设置为队列即可。

```

1 def breadthFirstSearch(problem):
2     ....
3     return generic_search(problem, util.Queue)

```

## 2.3 问题三 应用代价一致算法找到一个特定的位置的豆

### 2.3.1 问题描述

前面的各种搜索策略中，实际都假设状态空间中各边的代价都相同，且都为 一个单位量，从而可以用路径长度来代替路径的代价。但实际问题中，这种假设 不现实，它们的状态空间中的各个边的代价不可能完全相同。为此，我们需要在 搜索树中给每条边标上其代价。这种边上有代价的树称为代价树。

在本问题中，我们就需要使用代价一致搜索算法为吃豆人规划一条在给定路 径带权的迷宫中从特定起点到特定终点的最短路径。

### 2.3.2 算法实现

代价一致搜索算法是在宽度优先搜索上进行扩展的，它的基本原理是：代价 一致搜索总是扩展代价最小的节点。一个点的代价等于前一节点的代价加上从前 一结点到当前节点的代价。

相比于宽度优先搜索算法，代价一致搜索不是沿着等长度路径逐层进行扩展， 而是沿着等代价路径逐层进行扩展。该算法的 `open` 表需要维护一个有序集合，支 持每次取出最小值元素的操作。所以我们使用优先队列来实现这种 `open` 表。基于 前面已经实现的通用的搜索算法，我们只需要将 `open_table_type` 设置为优先队列 即可。

```
1 def uniformCostSearch(problem):  
2     ....  
3     return generic_search(problem, util.PriorityQueue)
```

## 2.4 问题四 应用 A\* 算法找到一个特定的位置的豆

### 2.4.1 问题描述

在本问题中，我们就需要使用 A\* 算法为吃豆人规划一条在给定路径带权的迷 宫中从特定起点到特定终点的最短路径。

### 2.4.2 算法实现

A\* 算法是一种启发式搜索的方法，该算法综合了宽度优先算法和迪杰斯特拉 算法算法的优点：在进行启发式搜索提高算法效率的同时，可以保证找到一条最 优路径（基于评估函数）。

在此算法中，如果以  $g(n)$  表示从起点到任意顶点  $n$  的实际距离， $h(n)$  表示顶 点  $n$  到目标顶点的估算距离（根据所采用的评估函数的不同而变化），那么 A\* 算 法的估算函数为  $f(n) = g(n) + h(n)$ 。

在 A\* 算法中，我们需要优先扩展估算函数最小的节点，所以选择优先队列作为 open 表的数据结构。

```
1 def aStarSearch(problem, heuristic=nullHeuristic):
2     ....
3     return generic_search(problem, util.PriorityQueue)
```

## 2.5 问题五 找到所有的角落

### 2.5.1 问题描述

在本问题中，我们需要实现 `CornersProblem` 类，用来表示“让吃豆人将以最短的路径找到迷宫的四个角落”这一问题。我们需要为这个类设计出对应的状态并写出对应的目标状态判定和后即状态获取。

### 2.5.2 算法实现

对于本问题，显然我们不能直接使用位置来表示当前状态。我们需要使用额外的变量来存储四个角落的访问状态。为此，我将每个状态设计为一个包含位置和角落的访问状态的二元组。四个角落的访问状态使用一个整数的四个比特位表示，使用 1 表示对应位置访问过，0 表示对应位置没有访问过。该整数的初始值为 0，而目标状态就是所有该整数为  $2^4 - 1 = 15$  的状态。

```
1 def isGoalState(self, state):
2     ....
3     return state[1] == self.ALL_GOT
```

此外，`getSuccessors` 方法也是本部分的一个核心。由于在状态表示中增加了对角落状态的表示，所以在生成后继状态时也要考虑这部分是否会变化。考虑生成的后继节点，若后继节点是未到达过的角落位置，则在该后继的表示访问状态的整数中，要将对应的比特位置为 1，表示已经到达过该角落。

```
1 def getSuccessors(self, state):
2     ....
3     x, y = state[0]
4     visited = state[1]
5     successors = []
6     for action in [
7         Directions.NORTH, Directions.SOUTH, Directions.EAST,
8         Directions.WEST
9     ]:
10        dx, dy = Actions.directionToVector(action)
11        nextx, nexty = int(x + dx), int(y + dy)
12        hitsWall = self.walls[nextx][nexty]
13        if hitsWall: continue
14        successor_position = (nextx, nexty)
15        successor_visited = visited
16        for i, position in enumerate(self.corners):
```

```

17         if successor_position == position:
18             successor_visited |= 1 << i
19             successor = (successor_position, successor_visited)
20             successors.append((successor, action, 1))
21
22     self._expanded += 1 # DO NOT CHANGE
23     return successors

```

## 2.6 问题六 角落问题（启发式）

### 2.6.1 问题描述

本问题要求我们构建合适的启发函数，完善 `cornersHeuristic` 函数，实现使用启发式函数来解决问题五中的角落问题。

大纲要求使用 A\* 启发式搜索算法来解决角落问题，并且设计出设计一个非平凡且一致的启发式函数 `cornersHeuristic`，以通过扩展尽量少的节点访问所有四个角落。其中非平凡且一致算启发式函数就是要求我们对于所有的非目标状态  $n$ ，设  $h(n)$  为  $n$  到达目标状态的真实代价， $h^*(n)$  为启发式函数估计的代价，那么  $0 < h^*(n) \leq h(n)$ ，且  $h^*(goal) = 0$ 。这样可以保证我们可以得到正确的最短路径。

### 2.6.2 算法实现

我设计的启发式函数的启发策略为依次遍历四个角落中尚未访问的所有角落的排列，按照排列的顺序访问四个结点，用宽度优先搜索算法（即假设每条边的代价为 1）来估计两个结点的距离。

显然，两个结点间的距离是小于等于它们在迷宫中的真实距离的，故上述启发式函数

$$h^*(n) = \min_{\text{所有排列}} \sum \text{BFS 距离} \leq \min_{\text{所有排列}} \sum \text{真实距离} = h(n)$$

而对于非目标结点， $h^*(n) > 0$  是显然的，所以该启发式函数是非平凡且一致的。为了提高程序的效率，我们使用一个字典来存储已计算的距离。对应的代码实现如下，：

```

1 def cornersHeuristic(state, problem):
2     ....
3     position, visited = state
4     left_corners = []
5     for i, corner in enumerate(corners):
6         if visited >> i & 1: continue
7         left_corners.append(corner)
8
9     dist = problem.distance
10
11     def get_dist(x, y):

```

```

12     if (x, y) in dist: return dist[(x, y)]
13     if (y, x) in dist: return dist[(y, x)]
14     d = mazeDistance(x, y, problem.startingGameState)
15     dist[(x, y)], dist[(y, x)] = d, d
16     return d
17
18     if not left_corners: return 0
19
20     import itertools
21     min_distance = 10**9
22     for order in itertools.permutations(left_corners):
23         cost, pre = 0, position
24         for corner in order:
25             cost += get_dist(pre, corner)
26             pre = corner
27         min_distance = min(min_distance, cost)
28     return min_distance

```

## 2.7 问题七 吃掉所有的豆子

### 2.7.1 问题描述

在本问题中，多个食物不仅仅局限在角落出现，它们可能出现在任何地方。本问题需要一个十分良好的非平凡且一致的启发式函数，用尽可能少的步数，同时尽量扩展少的节点以找到一条可以获得所有的食物路径。

### 2.7.2 算法实现

一个始终走向距离其最近食物的吃豆人行走的路径不一定是最短路径。因而在问题七中对走遍所有食物的总路径寻找下界是一件比较困难的事。因而，我们考虑在所有剩余食物中选择具有代表性的食物，使用该食物位置与吃豆人当前位置之间的距离来代表吃豆人到达目标所需要的代价下界。

我设计的启发性函数的启发策略为当剩余的食物结点较少时，我们使用类似问题六的策略，枚举排列即可；当剩余的食物结点较多时，我们先访问最近的结点，之后距离最近结点最远的结点。因为我们一定是先吃完当前点附近的食物，再去吃最远处的食物。因为我们只考虑了两个结点间的距离，远小于实际食物点间的距离，所以我们的启发式函数  $h^*(n) \leq h(n)$ ，是非平凡且一致的。

为了提高程序的效率，类似问题六，我们同样使用一个字典来存储已计算的距离。对应的代码实现如下：

```

1 def foodHeuristic(state, problem):
2     ....
3     position, foodGrid = state
4     foods = foodGrid.asList()
5     dist = problem.heuristicInfo
6

```

```

7     if not foods: return 0
8
9     def get_distance(x, y):
10         if (x, y) in dist: return dist[(x, y)]
11         if (y, x) in dist: return dist[(y, x)]
12         d = mazeDistance(x, y, problem.startingGameState)
13         dist[(x, y)], dist[(y, x)] = d, d
14         return d
15
16     if len(foods) == 1: return get_distance(foods[0], position)
17
18     if len(foods) < 9:
19         vector = tuple(foods) + (position, )
20         if vector in dist: return dist[vector]
21         import random, itertools
22         random.shuffle(foods)
23         min_cost = 10**9
24         for order in itertools.permutations(foods):
25             pre, cost = position, 0
26             for food in order:
27                 cost += get_distance(pre, food)
28                 if cost >= min_cost:
29                     break
30             pre = food
31             if min_cost > cost:
32                 min_cost = cost
33         dist[vector] = min_cost
34         return min_cost
35
36     d, closest = min([(get_distance(food, position), food) for food in foods])
37     return d + max([get_distance(food, closest) for food in foods])

```

## 2.8 问题八 次最优搜索

### 2.8.1 问题描述

本问题要求实现一个优先吃最近的食物路径规划算法，需要我们实现 AnyFoodSearchProblem 类中的 isGoalState 方法以及 ClosestDotSearchAgent 类中的 findPathToClosestDot 方法。

### 2.8.2 算法实现

通过阅读 ClosestDotSearchAgent 类下代码，我们发现该问题类的运行逻辑是不断的使用 findPathToClosestDot 寻找一条距离当前结点最近的食物路径并将该路径加入到答案路径之中，直到所有的食物均被吃完。因而在 AnyFoodSearchProblem 类中，我们定义将能够吃掉一个食物为对应的目标状态，以使 findPathToClosestDot 方法在会吃到距离最近的食物。

isGoalState 的代码如下：

```
1 def isGoalState(self, state):
2     x, y = state
3     return self.food[x][y]
```

而在 `findPathToClosestDot` 方法中，我们要给出吃豆人到最近的食物 actions 列表，因此使用宽度优先搜索算法进行搜索出一条到达最近食物的最短的路径。

`findPathToClosestDot` 方法的代码如下：

```
1 def findPathToClosestDot(self, gameState):
2     ....
3     startPosition = gameState.getPacmanPosition()
4     food = gameState.getFood()
5     walls = gameState.getWalls()
6     problem = AnyFoodSearchProblem(gameState)
7
8     min_distance, ans = 10**9, None
9     for pos in food.asList():
10         if util.manhattanDistance(startPosition, pos) >= min_distance:
11             continue
12         prob = PositionSearchProblem(gameState,
13                                     start=startPosition,
14                                     goal=pos,
15                                     warn=False,
16                                     visualize=False)
17         path = search.bfs(prob)
18         distance = len(path)
19         if distance < min_distance:
20             ans = path
21             min_distance = distance
22     return ans
```

### 3 算法实现

本节给出运行 `autograder.py` 后对于每个问题的评分结果。



```
Question q1
=====

*** PASS: test_cases/q1/graph_backtrack.test
***   solution:      ['1:A->C', '0:C->G']
***   expanded_states: ['A', 'D', 'C']
*** PASS: test_cases/q1/graph_bfs_vs_dfs.test
***   solution:      ['2:A->D', '0:D->G']
***   expanded_states: ['A', 'D']
*** PASS: test_cases/q1/graph_infinite.test
***   solution:      ['0:A->B', '1:B->C', '1:C->G']
***   expanded_states: ['A', 'B', 'C']
*** PASS: test_cases/q1/graph_manypaths.test
***   solution:      ['2:A->B2', '0:B2->C', '0:C->D', '2:D->E2', '0:E2->F', '0:F->G']
***   expanded_states: ['A', 'B2', 'C', 'D', 'E2', 'F']
*** PASS: test_cases/q1/pacman_1.test
***   pacman layout:   mediumMaze
***   solution length: 130
***   nodes expanded:   146

### Question q1: 3/3 ###
```

图 3.1: 问题一运行结果

```
Question q2
=====

*** PASS: test_cases/q2/graph_backtrack.test
***   solution:      ['1:A->C', '0:C->G']
***   expanded_states: ['A', 'B', 'C', 'D']
*** PASS: test_cases/q2/graph_bfs_vs_dfs.test
***   solution:      ['1:A->G']
***   expanded_states: ['A', 'B']
*** PASS: test_cases/q2/graph_infinite.test
***   solution:      ['0:A->B', '1:B->C', '1:C->G']
***   expanded_states: ['A', 'B', 'C']
*** PASS: test_cases/q2/graph_manypaths.test
***   solution:      ['1:A->C', '0:C->D', '1:D->F', '0:F->G']
***   expanded_states: ['A', 'B1', 'C', 'B2', 'D', 'E1', 'F', 'E2']
*** PASS: test_cases/q2/pacman_1.test
***   pacman layout:   mediumMaze
***   solution length: 68
***   nodes expanded:   269

### Question q2: 3/3 ###
```

图 3.2: 问题二运行结果



```
Question q3
=====

*** PASS: test_cases/q3/graph_backtrack.test
***   solution:          ['1:A->C', '0:C->G']
***   expanded_states:   ['A', 'B', 'C', 'D']
*** PASS: test_cases/q3/graph_bfs_vs_dfs.test
***   solution:          ['1:A->G']
***   expanded_states:   ['A', 'B']
*** PASS: test_cases/q3/graph_infinite.test
***   solution:          ['0:A->B', '1:B->C', '1:C->G']
***   expanded_states:   ['A', 'B', 'C']
*** PASS: test_cases/q3/graph_manypaths.test
***   solution:          ['1:A->C', '0:C->D', '1:D->F', '0:F->G']
***   expanded_states:   ['A', 'B1', 'C', 'B2', 'D', 'E1', 'F', 'E2']
*** PASS: test_cases/q3/ucs_0_graph.test
***   solution:          ['Right', 'Down', 'Down']
***   expanded_states:   ['A', 'B', 'D', 'C', 'G']
*** PASS: test_cases/q3/ucs_1_problemC.test
***   pacman layout:     mediumMaze
***   solution length:   68
***   nodes expanded:    269
*** PASS: test_cases/q3/ucs_2_problemE.test
***   pacman layout:     mediumMaze
***   solution length:   74
***   nodes expanded:    260
*** PASS: test_cases/q3/ucs_3_problemW.test
***   pacman layout:     mediumMaze
***   solution length:   152
***   nodes expanded:    173
*** PASS: test_cases/q3/ucs_4_testSearch.test
***   pacman layout:     testSearch
***   solution length:   7
***   nodes expanded:    14
*** PASS: test_cases/q3/ucs_5_goalAtDequeue.test
***   solution:          ['1:A->B', '0:B->C', '0:C->G']
***   expanded_states:   ['A', 'B', 'C']

### Question q3: 3/3 ###
```

图 3.3: 问题三运行结果

```

Question q4
=====

*** PASS: test_cases/q4/astar_0.test
***   solution:          ['Right', 'Down', 'Down']
***   expanded_states:   ['A', 'B', 'D', 'C', 'G']
*** PASS: test_cases/q4/astar_1_graph_heuristic.test
***   solution:          ['0', '0', '2']
***   expanded_states:   ['S', 'A', 'D', 'C']
*** PASS: test_cases/q4/astar_2_manhattan.test
***   pacman layout:     mediumMaze
***   solution length:   68
***   nodes expanded:    221
*** PASS: test_cases/q4/astar_3_goalAtDequeue.test
***   solution:          ['1:A->B', '0:B->C', '0:C->G']
***   expanded_states:   ['A', 'B', 'C']
*** PASS: test_cases/q4/graph_backtrack.test
***   solution:          ['1:A->C', '0:C->G']
***   expanded_states:   ['A', 'B', 'C', 'D']
*** PASS: test_cases/q4/graph_manypaths.test
***   solution:          ['1:A->C', '0:C->D', '1:D->F', '0:F->G']
***   expanded_states:   ['A', 'B1', 'C', 'B2', 'D', 'E1', 'F', 'E2']

### Question q4: 3/3 ###

```

图 3.4: 问题四运行结果

```

Question q5
=====

*** PASS: test_cases/q5/corner_tiny_corner.test
***   pacman layout:     tinyCorner
***   solution length:    28

### Question q5: 3/3 ###

```

图 3.5: 问题五运行结果

```

Question q6
=====

*** PASS: heuristic value less than true cost at start state
*** PASS: heuristic value less than true cost at start state
*** PASS: heuristic value less than true cost at start state
path: ['North', 'East', 'East', 'East', 'East', 'North', 'North', 'West', 'West', 'West', 'West', 'North', 'North', 'N
orth', 'North', 'North', 'North', 'North', 'North', 'West', 'West', 'West', 'West', 'South', 'South', 'East', 'East',
'East', 'East', 'South', 'South', 'South', 'South', 'South', 'South', 'South', 'West', 'West', 'South', 'South', 'South', 'West
', 'West', 'North', 'East', 'East', 'North', 'North', 'East', 'East', 'East', 'East', 'East', 'East', 'East', 'East',
'South', 'South', 'East', 'East', 'East', 'East', 'East', 'North', 'North', 'East', 'East', 'North', 'North', 'East',
'East', 'North', 'North', 'East', 'East', 'East', 'East', 'South', 'South', 'South', 'South', 'East', 'East', 'North',
'North', 'East', 'East', 'South', 'South', 'South', 'South', 'North', 'North', 'North', 'North', 'North', 'N
orth', 'North', 'West', 'West', 'North', 'North', 'East', 'East', 'North', 'North']
path length: 106
*** PASS: Heuristic resulted in expansion of 189 nodes

### Question q6: 3/3 ###

```

图 3.6: 问题六运行结果

```

Question q7
=====
*** PASS: test_cases/q7/food_heuristic_1.test
*** PASS: test_cases/q7/food_heuristic_10.test
*** PASS: test_cases/q7/food_heuristic_11.test
*** PASS: test_cases/q7/food_heuristic_12.test
*** PASS: test_cases/q7/food_heuristic_13.test
*** PASS: test_cases/q7/food_heuristic_14.test
*** PASS: test_cases/q7/food_heuristic_15.test
*** PASS: test_cases/q7/food_heuristic_16.test
*** PASS: test_cases/q7/food_heuristic_17.test
*** PASS: test_cases/q7/food_heuristic_2.test
*** PASS: test_cases/q7/food_heuristic_3.test
*** PASS: test_cases/q7/food_heuristic_4.test
*** PASS: test_cases/q7/food_heuristic_5.test
*** PASS: test_cases/q7/food_heuristic_6.test
*** PASS: test_cases/q7/food_heuristic_7.test
*** PASS: test_cases/q7/food_heuristic_8.test
*** PASS: test_cases/q7/food_heuristic_9.test
*** PASS: test_cases/q7/food_heuristic_grade_tricky.test
*** expanded nodes: 1642
*** thresholds: [15000, 12000, 9000, 7000]

### Question q7: 5/4 ###

```

图 3.7: 问题七运行结果

```

Question q8
=====
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
*** PASS: test_cases/q8/closest_dot_1.test
*** pacman layout: Test 1
*** solution length: 1
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
*** PASS: test_cases/q8/closest_dot_10.test
*** pacman layout: Test 10
*** solution length: 1
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
*** PASS: test_cases/q8/closest_dot_11.test
*** pacman layout: Test 11
*** solution length: 2
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
*** PASS: test_cases/q8/closest_dot_12.test
*** pacman layout: Test 12
*** solution length: 3
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
*** PASS: test_cases/q8/closest_dot_13.test
*** pacman layout: Test 13
*** solution length: 1
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
*** PASS: test_cases/q8/closest_dot_2.test
*** pacman layout: Test 2
*** solution length: 1
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
*** PASS: test_cases/q8/closest_dot_3.test
*** pacman layout: Test 3
*** solution length: 1
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
*** PASS: test_cases/q8/closest_dot_4.test
*** pacman layout: Test 4
*** solution length: 3
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
*** PASS: test_cases/q8/closest_dot_5.test
*** pacman layout: Test 5
*** solution length: 1
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
*** PASS: test_cases/q8/closest_dot_6.test
*** pacman layout: Test 6
*** solution length: 2
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
*** PASS: test_cases/q8/closest_dot_7.test
*** pacman layout: Test 7
*** solution length: 1
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
*** PASS: test_cases/q8/closest_dot_8.test
*** pacman layout: Test 8
*** solution length: 1
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
*** PASS: test_cases/q8/closest_dot_9.test
*** pacman layout: Test 9
*** solution length: 1

### Question q8: 3/3 ###

```

图 3.8: 问题八运行结果

## 4 讨论及结论

在本次实验中我们实现了深度优先、宽度优先、代价一致、A\* 这四种搜索算法，同时也学习了启发式函数的选取要求和方法，通过优化启发式函数来提高搜索的效率。

本次实验可以说是一次很有意思的实验了。本次前四个搜索算法，由于仅仅是 `open` 表使用的数据结构不用，其他大部分都是一样的，因此将四个算法写在了一起，组合成了一个函数，实现了抽象和复用。后面四个问题是在前四个搜索算法已经写好的条件下计算的，根据不同的要求，填写代码不同的部分并选择合适的搜索算法，不断优先自己的启发式函数，最后实现一个满足条件的高效算法。经过本次实验，我对搜索算法有了进一步的理解，掌握了一些优化启发式函数的方法。

# 第三章 实验三 不确定性推理

## 1 问题描述

### 1.1 基本要求

本实验要求我们参照课程第五部分讲授的贝叶斯网络完成。实验会给定事件和事件之间的关系，并且给出每个事件的 CPT 图，要求程序根据贝叶斯公式根据上述条件求出目标概率，编写程序实现基于贝叶斯网络的推理。在这里用到的贝叶斯算法是建立在有向无环图和 CPT 表的技术上实现的。

### 1.2 输入输出描述

首先我们需要读入形如

```
1 N
2
3 rv0 rv1 ... rvN-1
4 0 0 1 ... 0
5 1 0 0 ... 1
6 ...
7 0 1 1 ... 0
8
9 mat0
10
11 mat1
12
13 ...
14
15 matN-1
```

在这里， $N$  是贝叶斯网络中随机事件的数目， $rv$  是随机事件的名字（字符串形式表示）， $mat$  是一个二维数组，分别表示从他的父亲到其本身的可能性概率。第一个元素表示发生的概率，第二个元素表示不发生的概率，显然两个元素相加为 1。

在上述中  $mat$  即为 CPT 表（Conditional Probability Table），其被设计为如下格式：对于每个节点，如果他有  $N$  个父节点，则其 CPT 表中有  $2^N$  列，我们记为标号

$0 \dots 2^N - 1$ 。其行序号的定义方法如下：利用二进制分别表示对应的父亲为是否发生，1 为发生，0 位不发生，将得到的二进制数转化为十进制代表其对应的行号。

其次，编写程序对应的查询格式为： $P(\text{rvQ}|\text{rvE1} = \text{val}, \text{rvE2} = \text{val}, \dots)$ ，rvQ 表示查询的条件名字，即在  $\text{rvE1} = \text{val}, \text{rvE2} = \text{val}, \dots$  发生的条件下，rvQ 发生的概率；RvEx 表示条件的名称，而后面的 val 为 true 或 false，分别表示发生和不发生。最后，输出格式为两个数据分别表示  $P(\text{QueryVar} = \text{true}|\dots)$  和  $P(\text{QueryVar} = \text{false}|\dots)$ 。

## 2 算法介绍

### 2.1 基本原理

我们程序基本基于如下公式：

$$Z = \sum_q P(Q, e_1, \dots, e_k) \quad (2.1)$$

$$P(Q|e_1, \dots, e_k) = \frac{1}{Z} P(Q, e_1, \dots, e_k) \quad (2.2)$$

我们先求解出所有的  $2^N$  个事件的概率，得出联合概率表。然后读入需求解的概率，利用穷举法，计算相应的联合概率，最后根据 CPT 图消去无关变量，归一化得到待求概率的解。

### 2.2 代码实现

我们在读入贝叶斯网络后，先根据贝叶斯网络和 CPT 图计算出每个事件的全概率，将  $N$  个随机变量共  $2^N$  个事件编码为  $N$  位二进制数，最高位为 1 代表第一个随机变量取值 (1 为真，0 为假)，次高位代表第二个随机变量的取值，以此类推。对于每个事件，遍历每个随机变量，提取出该随机变量的父节点的取值，查询 CPT 进行累乘，得到该事件的联合概率。代码实现如下：

```

1 void calculateTotalProbability() {
2     int bound = 1 << n;
3     total_probability.resize(bound, 1);
4     for (int s = 0; s < bound; ++s) {
5         double &poss = total_probability[s];
6         for (int i = 0; i < n; ++i) {
7             int index = 0;
8             static auto get_bit = [&](int x) { return s >> (n - 1 - x) & 1; };
9
10            for (int x : cpts[i].parents) index = index << 1 | get_bit(x);
11            poss *= cpts[i].probabilities[index][get_bit(i)];
12        }
13    }
14 }

```

然后根据输入的待求解的概率，提取出欲求解的随机变量与已知的随机变量取值，将无关的变量消去，最后归一化即得到最终解。

```

1  std::vector<double> calculateProbability(const std::vector<int> &vars) const {
2      // var : 0: 假, 1: 真, 2: 求解, 3: 待消去
3      assert(static_cast<int>(vars.size()) == n);
4
5      int to_solve = 0, to_eliminate = 0;
6      for (int x : vars) {
7          to_solve += x == 2;
8          to_eliminate += x == 3;
9      }
10     assert(to_solve == 1);
11
12     int solve_case = 1 << to_solve;
13     int eliminate_case = 1 << to_eliminate;
14
15     std::vector<double> probability(solve_case, 0);
16     for (int solve = 0; solve < solve_case; ++solve)
17         for (int eliminate = 0; eliminate < eliminate_case; ++eliminate) {
18             auto values = vars;
19             int c_solve = to_solve, c_eliminate = to_eliminate, index = 0;
20             for (int &x : values) {
21                 if (x == 2) x = solve >> (--c_solve) & 1;
22                 if (x == 3) x = eliminate >> (--c_eliminate) & 1;
23                 index = index << 1 | x;
24             }
25             probability[solve] += total_probability[index];
26         }
27
28     double sum = 0;
29     for (auto &x : probability) sum += x;
30     for (auto &x : probability) x /= sum;
31     return probability;
32 }

```

### 3 算法实现

我们使用如下代码测试我们实现的 bayesian\_network 类

```

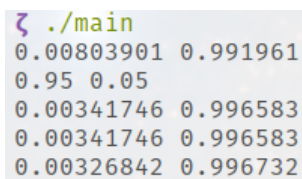
1  #include <cassert>
2  #include <fstream>
3  #include <iostream>
4  #include <regex>
5
6  #include "bayesian_network.h"
7
8  int main() {
9      std::ios::sync_with_stdio(false);
10
11     std::ifstream fin("burglarnetwork.txt");
12     assert(fin.is_open());
13     BayesianNetwork net(fin);

```



```
14  fin.close();
15
16  fin.open("burglarqueries.txt");
17  assert(fin.is_open());
18
19  std::regex split("[P()|,\\s]+");
20  std::regex assign("=");
21
22  std::vector<int> vars(net.getCPTSize());
23  for (std::string line; std::getline(fin, line);) {
24      if (line.empty()) continue;
25      std::vector<std::string> tokens(
26          std::sregex_token_iterator(line.begin(), line.end(), split, -1),
27          std::sregex_token_iterator());
28      fill(vars.begin(), vars.end(), 3);
29      for (auto &var : tokens) {
30          if (var.empty()) continue;
31          std::vector<std::string> tokens_(
32              std::sregex_token_iterator(var.begin(), var.end(), assign, -1),
33              std::sregex_token_iterator());
34          if (tokens_.size() == 1) {
35              vars[net.getVariableId(var)] = 2;
36          } else {
37              vars[net.getVariableId(tokens_[0])] = tokens_[1] == "true";
38          }
39      }
40
41      auto result = net.calculateProbability(vars);
42      std::cout << result[1] << " " << result[0] << std::endl;
43  }
44
45  return 0;
46 }
```

代码的运行结果如下：



```
ζ ./main
0.00803901 0.991961
0.95 0.05
0.00341746 0.996583
0.00341746 0.996583
0.00326842 0.996732
```

图 3.1: 实验三运行结果

## 4 讨论及结论

本次实验我们实现了给定贝叶斯网络和 CPT 计算条件概率。该实验首先利用输入的文件，构建了一个贝叶斯网络。然后读取问题文件中的概率问题，利用 CPT 表进行计算。算法利用二进制编码降低了算法复杂性，但同样也提高了代码的编写难度。通过此次实验我们更加深入地理解了贝叶斯网络。