哈爾濱工業大學

# 人工智能实验报告

题　　目　知识表示

专　　业　大数据

学　　号　L170300901

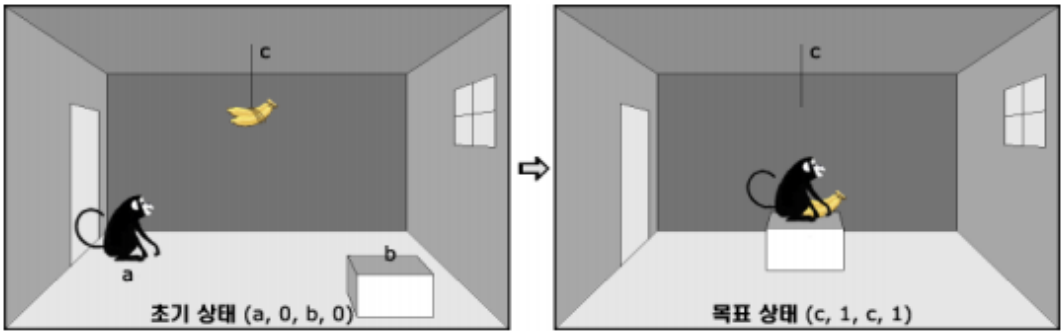
姓　　名　卢兑玑

同 组 人 员　金英雄

## 一。 简介/问题描述

1.1 相关吃豆人背景以及其跟搜索策略的关系

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。




1.2 根据搜索策略角度出发总结待完成的问题

使用知识表示单元学到的产生式表示法，从问题生成一个产生式系统。按照产生式系统的 基本结构，综合数据库可以写成(Monkey， Box，Banana，On&Get) Monkey:猴子的位置， Box:箱子的位置，Banana:香蕉的位置，On&Get=-1:猴子在地板上，On&Get=1:猴子在箱子 上并抓到了香蕉。初始状态为(A,B,C,-1)，目标状态为(C,C,C,1)。然后按照产生式系统 规则的基本形式 IF THEN ，生成规则库。

规则集的各个规则为如下：

r1：IF (x,y,z,-1) THEN (w,y,z,-1) r2: IF (x,x,y,-1) THEN (w,y,y,-1)

r3: IF (x,x,x,-1) THEN (x,x,x, 1) r4: IF (x,x,z, 1) THEN (x,x,z,-1)

## 二。 算法介绍

2.1 所用算法及的解题思路的一般介绍

首先对各个变量进行初始化，猴子，箱子和香蕉的位置，猴子在不在箱子上。然后检查初始化的数据是不是不可能的情况，比如猴子在 A 位置，箱子在 B 位置但是猴子在箱子上面。如果没问题的话，就进行产生式系统的推理方法(猴子将箱子从当前位置移动到香蕉的位置，猴子从当前位置移动到箱子的位置等)，当综合数据库里有目标状态的时候会停止，即解决该问题。

## 三。 算法实现

3.1 实验结果

实验环境：Window10，Python3.X / 问题规模：时间复杂度：O(1)

初始化状态为一个初始状态(1,0,-1,-1)相当于是(C,B,A,-1)，这时目标状态为(A,A,A, 1)。 通过产生式系统的推理过程能到达目标状态。

推理过程为如下：

第一步:猴子从 C 移动到 B (C,B,A,-1)->(B,B,A,-1)

第二步:猴子将箱子从 B 移动到 A (B,B,A,-1)->(A,A,A,-1)

第三步:猴子爬到箱子上去 (A,A,A,-1)->(A,A,A, 1)

即到达目标状态，会输出“猴子将香蕉摘下来”并停止程序。

输入: 0 1 -1 -1

输出：

步数 1:猴子从 B 移动到 C

步数 2:猴子将箱子从 C 移动到 A

步数 3:猴子爬到箱子上去

步数 4:猴子将香蕉摘下来

```
please 1 0input: monkey[-1,0,1] box[-1,0,1] banana[-1,0,1] monbox[-1,1]
0 1 -1 -1
步数1:猴子从B移动到C
步数2:猴子将箱子从C移动到A
步数3:猴子爬到箱子上去
步数4:猴子将香蕉摘下来

Process finished with exit code 0
```

## 四。 讨论及结论

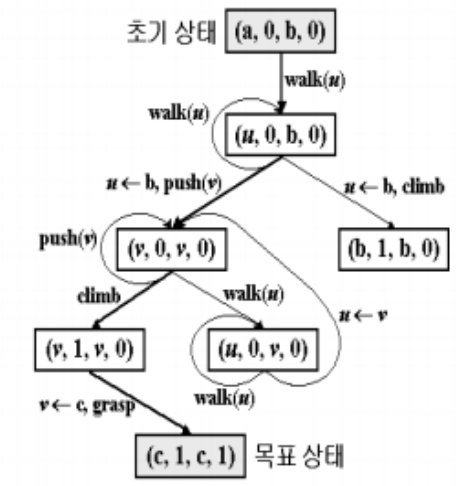
4.1 讨论（根据自己的实验过程，总结实验过程中的算法的构建过程及其优缺点，包括但不限于：不同搜索算法的对比，状态空间的抽象建模，启发函数的设计，启发函数的性质的分析，解决同一问题的不同方法及各自优缺点等等）

该实验让我们了解了利用知识表示方法解决实际问题。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。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。最终产生了一个从初始状态到目标状态的一条路径。

## 参考文献

**방법 (2)** 상태를 표현하기 위해 들 변수(schema variable)를 사용한다. 다시 말해, 위치를 나타내는 상수(constant)는 변수(variable)로 대치될 수 있고, 변수는 또한 다른 변수로 대치되거나 혹은 상수로 예화(instantiation)될 수 있다.

고전적 AI에서는 어떤 문제에 대해서도 항상 순차적으로 접근하므로, 방법 (1)의 적용은 현실적으로 힘들다. 왜냐하면 상태 공간의 크기가 커짐에 따라 목표 상태에 이르는 풀이 경로(solution path)의 길이는 급격하게 증가하게 되기 때문이다. 따라서 들 변수의 사용만 지원이 된다면, 방법 (2)를 적용시키는 편이 보다 바람직하다. 방법 (2)의 적용 하에서 상태 공간의 탐색을 통한 풀이 경로는 다음 (그림 3)에서의 굵은 화살표로 표시된다:




(그림 3) 원숭이와 바나나 문제의 그래프

2.2. Prolog를 사용한 문제 표현 및 해결

Prolog는 LISP와 더불어 고전적 AI 연구자들이 즐겨 이용하는 대표적인 언어이다. 이것은 주어진 문제에 대해 선언적(declarative) 접근, 즉 목표 지향적(goal-oriented) 프로그래밍을 가능하게 하므로, 원숭이와 바나나 문제를 해결하는 데 매우 적합한 도구이다. 나아가 Prolog의 추론 기제는 기본적으로 패턴 매칭(pattern matching)과 자동 백트래킹(backtracking)에 기반하고 있으므로, 구현된 Prolog 프로그램은 제 2.1절 방법 (2)에서의 들 변수의 사용을 만족스럽게 지원한다. Prolog 프로그램의 소스 코드는 다음과 같다([8], p.55):

```
% move( State1, Move, State2): making Move in State1 results in State2;
%     a state is represented by a term:
%     state( MonkeyHorizontal, MonkeyVertical, BoxPosition, HasBanana)

move(state( middle, onbox, middle, hasnot), % Before move
     grasp,                                 % Grasp banana
     state( middle, onbox, middle, has) ). % After move

move( state( P, onfloor, P, H),
     climb,                                 % Climb box
     state( P, onbox, P, H) ).

move( state( P1, onfloor, P1, H),
     push( P1, P2),                         % Push box from P1 to P2
     state( P2, onfloor, P2, H) ).

move( state( P1, onfloor, B, H),
     walk( P1, P2),                         % Walk from P1 to P2
     state( P2, onfloor, B, H) ).

% canget( State): monkey can get banana in State

canget( state( _ , _ , _ , has) ).          % can 1: Monkey already has it

canget( State1)  :-                         % can 2: Do some work to get it
  move( State1, Move, State2),              % Do something
  canget( State2).                          % Get it now
```

(그림 4) 원숭이와 바나나 문제 프로그램

여기서는 (그림 2)에서와 같이 목표 상태를 state(middle, onbox, middle, has)로 가정하고 있다. 이 프로그램은 "?-

注：报告中图的下方要有图题（如图 1. XXXX），表格需要用三线表，表头需列在表的上方（如表 1. XXXX）。图表居中排列。

哈爾濱工業大學

# 人工智能实验报告

题　　目　搜索策略

专　　业　大数据

学　　号　L170300901

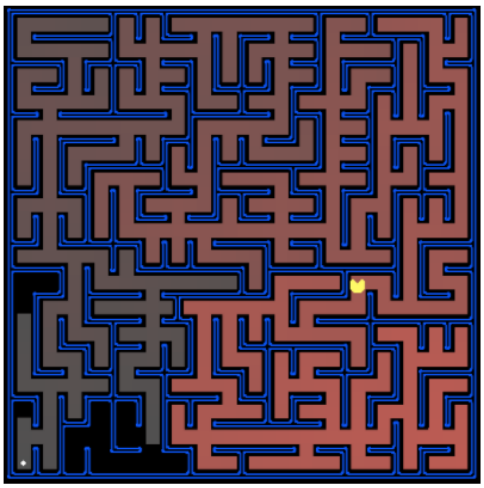
姓　　名　卢兑玑

同 组 人 员　金英雄

## 一。 简介/问题描述

1.1 相关吃豆人背景以及其跟搜索策略的关系

先参考 http://ai.berkeley.edu/search.html 上的内容编写问题 1-8 的所要求的内容，相当于是在提供的一个大的场景编写一系列 吃豆人程序解决以下列出的问题 1-8，包括到达指定位置以及有效 的吃豆等。



All those colored walls,
Mazes give Pacman the blues,
So teach him to search.

1.2 根据搜索策略角度出发总结待完成的问题

实验要求的问题 1-8 内容可以看成一个状态空间法上解决问题。因此找一个目标节点者路 径可以看成在状态空间法上找一个目标状态的路径。通过不同策略实现的搜索，路径结果 代价等的方面会不同。其中前四个问题要求编写搜索算法，有盲目式和启发式。第五个要 求找到一条访问所有四个角落的最短的路径，完成 CornersProblem 搜索问题 ，第六个要 求构建合适的启发函数，完成 cornersHeuristic 角落搜索问题。第七个要求完成 FoodSearchProblem 豆子搜索问题，第八个要求定义一个优先吃最近的豆子函数是提高搜 索 速 度 的 一 个 好 的 办 法 。 并 且 补 充 完 成 AnyFoodSearchProblem 目 标 测 试 函 数 和 ClosestDotSearchAgent 部分，在此 Agent 当中缺少一个关键的函数。

问题 1-2 是相当于在状态空间上利用盲目式搜索，其中第一个就是深度优先搜索，第二个 就是广度优先搜索，深度可以利用栈，广度可以

用队列实现。问题 3-4 是相当于在状态空 间上利用启发式搜索，第三个是代价一致算法，节点之间有代价，可以使用优先队列实现。 第四个是A*算法， 利用曼哈顿距离作为启发函数解决问题。第五个完成CornersProblem 搜索问题的时候，重新定义状态，使其能够表示角落是否被访问可以解决该问题。第六个 可以把没访问过的角落点中最近的角落点开始一步一步计算并叠加可以完成任务。第七个 也是写一个启发式函数的部分,按照距离可以选择最短的那一个。第八个就是吃掉一个食 物可以算到达目标，然后利用 A*算法和一个好的 heuristic 函数可以解决该问题。

## 二。 算法介绍

2.1 所用算法及的解题思路的一般介绍

Search.py 的问题 1-4 为了实现的方便定义一个 generalSearch，参数有 problem， opentable，是否需要代价，估计函数等，则通过这些参数来判断选择哪个搜索策略。如 果不需要代价的时候执行盲目式搜索方法，按照 opentable 可以是深度或广度优先搜索。 如果有代价的时候按照有没有估计函数可以是代价一致或 A*算法。

盲目式搜索部分是先把初始状态和有序的集合 path 放入有序的集合 table 里，然后进 入循环从 table 弹出一个状态和 path，如果这个状态节点为未访问过的节点，则扩展这个 节点并把子节点(未访问)和 path+direction(行动)放入 table 里并且把这个节点标为访问 过，直到找到目标状态节点。如果 table 利用栈，这个就是深度优先搜索，用队列就是广 度优先搜搜。

启发式搜索部分有代价函数，首先 A*算法实现的方法也是初始状态和有序的集合 path，任意值放入有序集合 table 里,然后进行循环，从 table 弹出代价+估计最小的状态和 path，如果 这个状 态 节点 为未 访问 过的节 点 ，则 扩 展 这个 节点并 把 子节 点 （ 未 访问 ）和 path+action，代价+估计值放入 table 里并且把这个节点标为访问

过，直到找到目标状态 节点，这个时候可以利用优先队列来实现算法。代价一致算法跟上述的方法不太一样，他 是每次扩展节点之后为每个子节点确定父节点，然后跟 A*算法相似，利用优先队列每次循 环选择代价最小的一个节点。别的都相同。然后找到了目标状态，则目标节点放入有序集 合 path 里，而且当前节点设为他的父节点，循环进行把当前节点放入 path，最后 path 里 面有逆序的路径。则把 path 取反一下能获得正确的路径。

问题 5 要编写四个部分，其中有初始化存储初始位置和吃的豆的数量的变量，还有三 个有功能的函数，其中 getStartState(self)获得初始状态，而初始状态由一个元组组 成。可以直接返回上边定义好的 self.startState。

```
    # in initializing the problem
    "*** YOUR CODE HERE ***"

    self.startState = (self.startingPosition, [])                # 存储位置和吃的豆的数量

def getStartState(self):
    """
    Returns the start state (in your state space, not the full Pacman state
    space)
    """
    "*** YOUR CODE HERE ***"
    return self.startState
```

isGoalState(self, state)函数就是判断这个节点是否达到目标状态，即检查这个状态的 第二个元组的大小为 4（已经吃掉四个豆为终止状态）

```
def isGoalState(self, state):
    """
    Returns whether this search state is a goal state of the problem.
    """
    "*** YOUR CODE HERE ***"
    return len(state[1]) == 4                    # 吃掉 4 个豆算终止状态
```

getSuccessors(self, state)返回后继状态列表，是一个元组列表，每个列表元组是 (successor,action, stepCost)形式，而这个 problem 里面的 successor 状态包括两个 （position,corners）position 代表位置，corners 记录了是否四个角落都访问过。首先 从 action 读取下一个节点，如果走这个方向没有碰到墙的时候，可以走这个方向，将信 息

存储起来。反复这个过程，可以获得正确的返回值。

问题 6 由于目标仅仅只是访问四个角落点，因此 Heuristic function 就是计算当前位 置到终态位置的曼哈顿距离。首先计算当前位置到最近角落的曼哈顿距离，然后计算第一 个角落到下一个最近角落的曼哈顿距离，这样直到所有角落都已经计算过或者访问过。

问题 7 的目标是吃掉所有的食物，我们需要编写的是 A 星算法的 heuristic function . 要满足可采纳性，就需要保证算法的 cost 是小于实际的 cost 的。而已经定义好的函数 mazeDistance(point1, point2, gameState)他的返回值是根据已经建立好的当前的搜索方 法计算任意两点之间的距离。然后可以使用当前位置到其他所有食物位置的最小路程作为 h 值，由于它一定会去最远的地方吃掉那个距离最远的食物，所以就用到最远的地方的距 离作为估计的总路程的最小值，这个时候估计值从小到大会吃豆子，相当于距离短的开始 吃，而这个值一定是小于等于实际代价的。

问题 8 由前边实现的 A*算法来解决搜索问题，然后判断目标的部分利用 isGoalState (self, state)函数，这个时候吃掉一个食物算是到达目标节点，则判断状态节点与食物 节点的距离，这个时候利用曼哈顿距离来算之后选择距离最短的那个值来判断是否为 0， 如果为 0 表示状态节点在食物节点上，否则不在食物节点则还没到目标节点。

## 三。 算法实现

3.1 实验结果

通过命令查看结果，结果应该为

Question q1:3/3

Question q2:3/3

Question q3:3/3

Question q4:3/3

Question q5:3/3

Question q6:3/3

Question q7:5/4

Question q8:3/3

Total: 26/25

即全功能都正确的完成，不过问题七好像有一些小问题。

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


Finished at 13:33:05

Provisional grades
==================
Question q1: 3/3
Question q2: 3/3
Question q3: 3/3
Question q4: 3/3
Question q5: 3/3
Question q6: 3/3
Question q7: 5/4
Question q8: 3/3
------------------
Total: 26/25

Your grades are NOT yet registered.  To register your grades, make sure
to follow your instructor's guidelines to receive credit on your project.
```

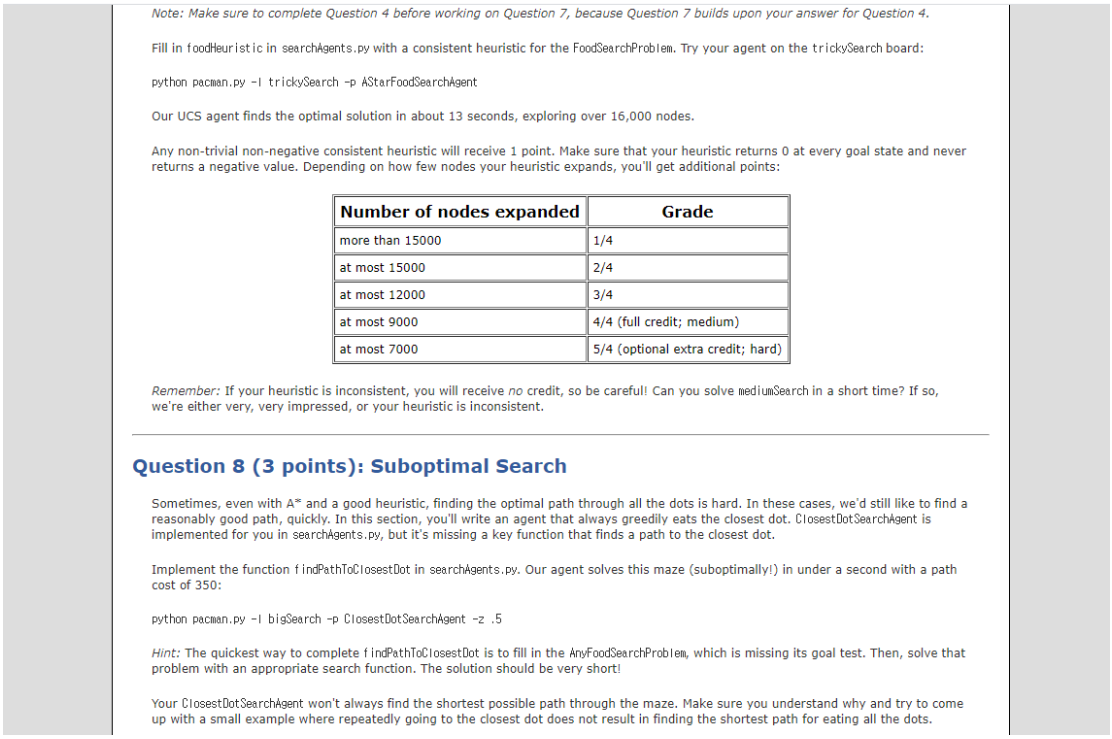
## 四。 讨论及结论

4.1 讨论（根据自己的实验过程，总结实验过程中的算法的构建过程及其优缺点，包括但不限于：不同搜索算法的对比，状态空间的抽象建模，启发函数的设计，启发函数的性质的分析，解决同一问题的不同方法及各自优缺点等等）

本次实验是利用贝叶斯网络计算各种概率的问题。该实验首先利用输入的文件，构建了一个贝叶斯网络，输出一个 CPT 表。然后读取问题文件中的概率问题，利用 CPT 表进行计算。由于计算过程中浮点数有一些误差，导致实验结果并不是一直很稳定，但还好，在可以接受的范围内。完成本次实验也是比较曲折的，经历了很多 bug ，在团队的协作下，最终还是实现出来了。

# 参考文献

http://ai.berkeley.edu/search.html



*Note: Make sure to complete Question 4 before working on Question 7, because Question 7 builds upon your answer for Question 4.*

Fill in `foodHeuristic` in `searchAgents.py` with a consistent heuristic for the `FoodSearchProblem`. Try your agent on the `trickySearch` board:

```
python pacman.py -l trickySearch -p AStarFoodSearchAgent
```

Our UCS agent finds the optimal solution in about 13 seconds, exploring over 16,000 nodes.

Any non-trivial non-negative consistent heuristic will receive 1 point. Make sure that your heuristic returns 0 at every goal state and never returns a negative value. Depending on how few nodes your heuristic expands, you'll get additional points:

| Number of nodes expanded | Grade |
|---|---|
| more than 15000 | 1/4 |
| at most 15000 | 2/4 |
| at most 12000 | 3/4 |
| at most 9000 | 4/4 (full credit; medium) |
| at most 7000 | 5/4 (optional extra credit; hard) |

*Remember:* If your heuristic is inconsistent, you will receive *no* credit, so be careful! Can you solve `mediumSearch` in a short time? If so, we're either very, very impressed, or your heuristic is inconsistent.

---

**Question 8 (3 points): Suboptimal Search**

Sometimes, even with A* and a good heuristic, finding the optimal path through all the dots is hard. In these cases, we'd still like to find a reasonably good path, quickly. In this section, you'll write an agent that always greedily eats the closest dot. `ClosestDotSearchAgent` is implemented for you in `searchAgents.py`, but it's missing a key function that finds a path to the closest dot.

Implement the function `findPathToClosestDot` in `searchAgents.py`. Our agent solves this maze (suboptimally!) in under a second with a path cost of 350:

```
python pacman.py -l bigSearch -p ClosestDotSearchAgent -z .5
```

*Hint:* The quickest way to complete `findPathToClosestDot` is to fill in the `AnyFoodSearchProblem`, which is missing its goal test. Then, solve that problem with an appropriate search function. The solution should be very short!

Your `ClosestDotSearchAgent` won't always find the shortest possible path through the maze. Make sure you understand why and try to come up with a small example where repeatedly going to the closest dot does not result in finding the shortest path for eating all the dots.

注：报告中图的下方要有图题（如图 1. XXXX），表格需要用三线表，表头需列在表的上方（如表 1. XXXX）。图表居中排列。

哈爾濱工業大學

# 人工智能实验报告

题　　目　不确定性推理

专　　业　大数据

学　　号　L170300901

姓　　名　卢兑玑

同 组 人 员　金英雄

## 一。 简介/问题描述

### 1.1 相关吃豆人背景以及其跟搜索策略的关系

参照课程第五部分讲授的贝叶斯网络完成，给定事件和事件之间的关系，并且给出每个事 件的 CPT 图，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 根据搜索策略角度出发总结待完成的问题

就是一个贝叶斯网络上分析个节点之间的独立性和概率，正确的给出 CPT 图。 首先参考老师要求里面已给的格式，给定的输入文件格式为:

```
N

rv0 rv1    ...    rvN-1
0   0      1      ...    0
1   0      0      ...    1
...
0   1      1      ...    0

mat0

mat1

...

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 - （2N1)，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

首先要理解概率论的一些基础公式和计算方法，比如条件概率，贝叶斯公式等等，通过基 本的概率公式和链式法则能够计算一些概率情况，并且还要掌握贝叶斯图的最基本的概念 和简单的计算方法（包括贝叶斯网的节点之间独立性）。 贝叶斯公式：P(A|B)=P(B|A)*P(A)/P(B)

链式法则：P(x1,x2,...xn) = x1∏i=1...nP(xi|x1,,,xi-1)

## 二。 算法介绍

2.1 所用算法及的解题思路的一般介绍

首先按照老师给定的格式读文件，并且把这个文件里的数据转化成一种可处理的形 式。使用 geteverycpt 函数从文件里读取数据，存放 Note 结构里，其中 count 为变量的 个数，t 为所有的 mat 表 p 为点的关系表，q 为各个节点的父节点个数。然后读取问题文 件，通过 getquestion 函数读取文件，利用分割函数按“P(”分，然后这些数据存放一个 数组里。这样完成文件转化为一种能处理的形式，然后接下来要处理这些数据计算全概 率，我们通过 allpro 函数计算全概率。根据贝叶斯网的知识利用 P(x1,x2,...xn) = Π i=1...nP(xi|paren ts(xi))，并且使用枚举法方法来解决这个问题。先函数里定义一些需要用 的变量，同时进行初始化，然后利用两层的循环过程处理这些数据。循环过程就是从第一 个节点开始计算，遍历能出现的所有情况下的概率，把全概率存放到要返回的数组里。当 有 N 个节点的时候需要计算 2 的 N 次方，函数初始化的时候定义的 sum 变量就是计算的次 数，然后这里还会调用 pos 函数处理一些数据。计算完全概率之后可以通过 getPT 函数求解该要求的问题，函数里调用 compute 函 数，根据上课讲的内容(下边图)能够具体 地计算解决该要求的问题的概率，每个问题的解有两个，发生事件和不发生事件概率 (T,F)计算过程当中利用二进制的数，二进制的

1 表示真(T),二进制的 0 表示假(F)，计算 之后保存为一个数组，第一个元组为真值(T)的概率，第二个元组为假值(F)的概率。

```
P(Q,e1...ek)=Σh1...hrP(Q,h1...hr,e1...ek)
Z=ΣqP(Q,e1...ek)
P(Q|e1...ek)=1/Z*P(Q,e1...ek)
```

## 三。 算法实现

### 3.1 实验结果

按照没步骤的计算和数据处理过程，首先取得全概率，然后问题解决的过程后可以能够获 得对该要求问题的正确答案，比如为如下：

P(CarWontStart | Dipstick=false, Lights=true, OilLight=true, FuelGauge=true)

= : 0.9834050864967842 : 0.01659491350321645

P(AlternatorBroken | CarWontStart=true, Dipstick=true, Lights=false, BatteryMeter=false)

= : 0.01 : 0.99

P(Alarm | Earthquake=true, Burglar=true)

= : 0.95 : 0.05

P(Burglar | John=true, Mary=false)

= : 0.001 : 0.999

P(Burglar | Alarm=true, Earthquake=true)

= : 0.001 : 0.999

P(Burglar | Alarm=true)

= : 0.001 : 0.999

第一个测试文件输出：

```
请输入文件名称

P(Burglar | Alarm=true) =
 : 0.001 : 0.999
2
P(Alarm | Earthquake=true, Burglar=true) =
 : 0.95 : 0.05
P(Burglar | John=true, Mary=false) =
 : 0.001 : 0.999
P(Burglar | John=true, Mary=false) =
 : 0.001 : 0.999
P(Burglar | Alarm=true, Earthquake=true) =
 : 0.001 : 0.999
```

第二个测试文件输出：

```
请输入文件名称

12
P(CarWontStart | Dipstick=false, Lights=true, OilLight=true, FuelGauge=true) =
 : 0.9834050864967842 : 0.01659491350321645
P(AlternatorBroken | CarWontStart=true, Dipstick=true, Lights=false, BatteryMeter=false) =
 : 0.01 : 0.99
```

## 四。 讨论及结论

4.1 讨论（根据自己的实验过程，总结实验过程中的算法的构建过程及其优缺点，包括但不限于：不同搜索算法的对比，状态空间的抽象建模，启发函数的设计，启发函数的性质的分析，解决同一问题的不同方法及各自优缺点等等）

这个实验的大部分是跟概率有关的内容，而且基于概率论和贝叶斯网络的实验要求。通过课 堂上学的概率论的基础知识和一些相关的公式或定理，能够实现了贝叶斯网上求解问题。这 次还用到了枚举法解决问题，实验中遇到了不少困难，因为计算完全概率之后存储方法，或 者后续地计算方法等等。但是通过实验熟悉了概率论和贝叶斯网络的基础知识，还提高了合 作能力及编码能力。

## 参考文献

注：报告中图的下方要有图题（如图 1. XXXX），表格需要用三线表，表头需列在表的上方（如表 1. XXXX）。图表居中排列。