实验：搜索

引论

本实验中, Pacman 智能体将找到通过迷宫世界的路径, 既要到达一个指定的位置，也要高效地搜集食物。你需要建立通用的搜索算法。

本实验的代码在几个Python文件中，其中一些代码你需要阅读并理解，这样才能完成作业，而另外一些代码则可以忽略。

需要编辑的代码:

search.py

你的所有搜索算法都要在这个文件里实现

searchAgents.py

所有基于搜索的智能体在这里

可能需要阅读的代码:

pacman.py

运行Pacman游戏的主文件。此文件描述Pacman GameState类型,在项目中会用到.

game.py

Pacman世界背后如何工作的逻辑。此文件描述几种支持类型如 AgentState, Agent, Direction, 和 Grid.

util.py

对实现搜索算法有用的数据结构。

可以忽略的支持文件:

graphicsDisplay.py

Pacman图形

graphicsUtils.py

Pacman图形支持

textDisplay.py

Pacman的ASCII图形

ghostAgents.py

控制ghosts的智能体Agents

keyboardAgents.py

控制Pacman的键盘接口

layout.py

读取框架文件以及保存他们的代码

**提交内容**:你需要编写search.py和searchAgents.py中的部分代码。

获得帮助: 如果你发现自己陷在某问题上, 请直接联系辅导老师。建议是，如果你不知道某个变量是做什么的，或者不知道其取何值，输出它，看看它的值。

Pacman世界

进入search目录。玩Pacman游戏的方法是输入下列命令行:

python pacman.py

Pacman 居住在亮蓝色的世界里，在这个世界有弯曲的走廊和美味佳肴。高效地浏览世界将是Pacman掌握其世界的第一步。

在searchAgents.py中最简单的智能体称为GoWestAgent, 将一直向西走(极简单的反射智能体). 它将偶尔获胜：

python pacman.py --layout testMaze --pacman GoWestAgent

但, 当需要转弯时，事情变得非常糟糕：

python pacman.py --layout tinyMaze --pacman GoWestAgent

如果智能体被卡住，则按CTRL-c可退出游戏。

很快, 你的智能体不仅能求解小迷宫tinyMaze, 还能求解任何你想要的迷宫。注意pacman.py支持多种选项，这些选项既可以表达长格式(如 --layout) 也可以是短格式(如 -l). 可以通过以下方式看到所有选项的列表:

python pacman.py -h

另外，所有出现在本项目中的命令，都出现在commands.txt中，使你能方便地复制粘贴。在UNIX/Mac OS X中，还可以使用以下命令来运行所有命令：

bash commands.txt

使用搜索算法找一个固定的食物（Food Dot）

在searchAgents.py中，你会发现一个完全实现的搜索智能体, 该智能体将规划出一条通过Pacman的世界的路径，然后一步步地走过该路径。形成路径的搜索算法没有实现 -- 这是你需要实现的。在你解答后面问题时，可能需要参考代码中的对象。首先，运行下面指令来测试SearchAgent能正确的工作：

python pacman.py -l tinyMaze -p SearchAgent -a fn=tinyMazeSearch

上面命令通知SearchAgent 使用 tinyMazeSearch 作为其搜索算法, 它已经在[search.py](http://inst.eecs.berkeley.edu/~cs188/fa09/projects/search/docs/search.html)中实现。Pacman将能成功地游览迷宫。

现在，开始编写完整的通用搜索函数以帮助Pacman规划路径!记住，搜索结点不仅要包含状态，还要包含达到此状态的必要的构造路径(规划)信息。

***重要注意事项****:* 你编写的所有搜索函数都需要返回一个行为*actions*链表，它将引导智能体从始点达到终点。这些行为都是合理的(正确的方向, 不能穿墙).

*提示:* 每个算法都很类似。 UCS（uniform cost search）和 A\* 算法的区别仅仅是如何管理待处理结点（fringe，即frontier结点）。

*提示:* 强烈建议大家使用在util.py中提供的Stack, Queue 和 PriorityQueue 类型!

问题1：深度优先算法

在search.py中depthFirstSearch函数中实现深度优先算法。为了保证算法可完成, 编写深度优先算法, 避免扩展已经访问的状态。你的代码应该能很快找到下列问题的解:

python pacman.py -l tinyMaze -p SearchAgent

python pacman.py -l mediumMaze -p SearchAgent

python pacman.py -l bigMaze -z .5 -p SearchAgent

对于已经搜索过的状态Pacman棋盘上将显示一个叠加物(overlay),并显示出访问的顺序(先访问的将 以亮红色显示). 搜索的顺序是你所期待的吗? Pacman 在到达目的地的过程中,是否遍访每个正方形？

**提示*:*** 如果你使用栈Stack数据结构, 则通过DFS算法求得的mediumMaze的解长度应该为130 (假定你 将后继元素按getSuccessors得到的顺序压栈; 如果,你按相反顺序压栈,则可能是244). 这是最短的路径吗? 如果不是, 想想深度优先为什么出问题。

问题2：广度优先搜索

在search.py中breadthFirstSearch函数中,实现广度优先搜索 (BFS) 算法。同样，请实现避免扩展已经存在状态的算法。和深度优先搜索一样地测试代码。

python pacman.py -l mediumMaze -p SearchAgent -a fn=bfs

python pacman.py -l bigMaze -p SearchAgent -a fn=bfs -z

BFS求得最小费用的解了吗？如果没有，请检查你的实现。

提示*:* 如Pacman移动太慢,可以试一下选项--frameTime 0

注意*:* 如果你的搜索代码具有通用性, 则不用做任何修改,该代码将同样能对eight-puzzle搜索问题适用。

问题3：不同的费用

通过修改代价函数，我们鼓励Pacman发现不同路径。例如，有恶魔的区域，我们增加每步的代价，而在食物丰富的区域减少每步的代价，一个理性的Pacman应该相应地调整它的行为。

在search.py的uniformCostSearch函数中，实现一致代价图搜索算法。util.py中有一些数据结构，也许会对你的实现有用。现在你应该能观察到在下面三个样板中的成功行为，所使用的智能体都是UCS（uniform cost search）智能体，其唯一区别是所使用的费用函数(其智能体和费用函数已经帮你写好了):

python pacman.py -l mediumMaze -p SearchAgent -a fn=ucs

python pacman.py -l mediumDottedMaze -p StayEastSearchAgent

python pacman.py -l mediumScaryMaze -p StayWestSearchAgent

*注:* 由于其指数费用函数，在StayEastSearchAgent 和 StayWestSearchAgent中，你将分别看到很低的和很高的路径费用total  cost(详细细节可见searchAgents.py).

问题4：A\*搜索

在search.py的aStarSearch函数中实现A\*图搜索 . A\*输入参数包括一个启发式函数。启发式函数有两个输入变量：搜索问题的状态 (主参数), 和问题本身(相关参考信息). search.py中的nullHeuristic 启发函数是一个普通的实例.

可以针对求通过迷宫到达固定点的原问题来测试A\*实现，具体可使用Manhattan距离启发(已经在searchAgents.py中实现为 manhattanHeuristic).

Python2 pacman.py -l bigMaze -z .5 -p SearchAgent -a fn=astar,heuristic=manhattanHeuristic

你将看到A\*求最优解比一致费用搜索解略快。 对于openMaze问题，不同的搜索策略会产生什么样的结果?

问题5：查找所有角落

注意：确保你已经完成问题2，然后再来完成问题5，因为问题5依赖于问题2的答案。

A\*搜索的真正强大之处，在具有更大挑战性的问题上才能显现。下面，我们需要先构造一个新问题，然后为其设计一个启发式的算法。

在*角落迷宫corner mazes中*, 四个角上各有一颗豆。我们新的搜索问题是找到穿过迷宫碰到所有四个角的最短路径(不论在迷宫中是否真有食物). 注意，对于象tinyCorners这样的迷宫, 最短路径不一定总是先找最近的食物! *提示*: 通过tinyCorners的最短路径需要28步.

在searchAgents.py中实现CornersProblem搜索问题。你需要选择一种状态表示方法，该方法可以对所有必要的信息编码，以便测定所有四个角点是否达到。现在, 搜索智能体应该可解下面的问题:

python pacman.py -l tinyCorners -p SearchAgent -a fn=bfs,prob=CornersProblem

python pacman.py -l mediumCorners -p SearchAgent -a fn=bfs,prob=CornersProblem

进一步，需要定义一个抽象的状态表示，该表示不对无关信息编码(如恶魔的位置, 其他食物的位置等)。特别是不要使用Pacman的GameState作为搜索状态。如果这样，你的代码会非常、非常慢(还出错).

*提示:* 在实现中，你需要访问的唯一游戏状态是Pacman的起始位置和四个角点的位置。

问题6：角落问题：启发式

注意：确保你已经完成问题4，然后再来完成问题6，因为问题6依赖于问题4的答案。

对CornersProblem实现一个启发式搜索cornersHeuristic。请在你的实现前面加上必要的备注。

python pacman.py -l mediumCorners -p AStarCornersAgent -z 0.5

注意：AStarCornersAgent 是 -p SearchAgent -a fn=aStarSearch,prob=CornersProblem,heuristic=cornersHeuristic

的缩写。

问题7：吃掉所有的“豆”

接下来，我们求解一个困难的搜索问题: 使用尽量少的步骤吃掉所有的食物。对此次作业，我们需要定义一个新的搜索问题，在该定义中正确描述吃掉所有食物的问题: 在searchAgents.py中的FoodSearchProblem (已经实现好了). 问题的解定义为一条收集到世界中所有食物的路径。在现在的项目中，不考虑”魔鬼“或"能量药“的存在; 解仅依赖于墙和正常食物在Pacman中的位置(当然，“魔鬼”会损坏解!) 。如果你已经正确地完成了通用搜索算法, 使用null heuristic (等价于一致费用搜索UCS) 的A\* 将很快求得testSearch问题的最优解，而不用大家写任何代码(总费用7).

python pacman.py -l testSearch -p AStarFoodSearchAgent

*注:* AStarFoodSearchAgent是-p SearchAgent -a fn=astar,prob=FoodSearchProblem,heuristic=foodHeuristic的缩写.

你将看到UCS开始慢下来，即使对看起来简单的tinySearch问题。

注意：确保你已经完成问题4，然后再来完成问题7，因为问题7依赖于问题4的答案。

针对FoodSearchProblem，使用一致性启发式函数，在searchAgents.py中完成foodHeuristic。在函数开头添加必要的注释描述你的启发式函数。测试你的Agent:

python pacman.py -l trickySearch -p AStarFoodSearchAgent

问题8：次优搜索

有的时候，即使使用 A\* 加上好的启发式，求通过所有“豆”的最优路径也是困难的。此时，我们还是希望能尽快地求得一个足够好的路径。在本节中，你需要写出一个智能体，它总是吃掉最近的豆. 在searchAgents.py中已经实现了ClosestDotSearchAgent, 但缺少一个关键函数，该函数搜索到最近豆的路径。

在文件searchAgents.py中实现findPathToClosestDot函数。

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

*提示:* 完成 findPathToClosestDot 的最快方式是填满AnyFoodSearchProblem, 该问题缺少目标测试。然后，使用合适的搜索函数来求解问题。解会非常短!

你的ClosestDotSearchAgent 并不总能找到通过迷宫的可能的最短路径。事实上，如果你尝试，你可以做得更好。

对象总览

下面是基础代码中与搜索问题有关的重要对象的总览，供大家参考：

SearchProblem (search.py)

SearchProblem是一个抽象对象，该对象表示状态空间，费用，和问题的目标状态。你只能通过定义在search.py顶上的方法来与SearchProblem交互

PositionSearchProblem (searchAgents.py)

需要处理的一种特别的SearchProblem类型 --- 对应于在迷宫中搜索单个肉丸pellet.

CornersProblem (searchAgents.py)

一种需要定义的特别的SearchProblem问题 --- 目的是搜索出一条到达迷宫中所有四个角点的路径.

FoodSearchProblem (searchAgents.py)

一个特定的需要解决的搜索问题。

Search Function

搜索函数是一个函数，该函数以SearchProblem的一个实例作为输入 , 运行一些算法, 返回值一列到达目标的行为. 搜索函数的实例有depthFirstSearch 和 breadthFirstSearch, 这些都要你编写。我们提供了tinyMazeSearch函数，该函数是一个非常差的函数，只能对tinyMaze得到正确结果

SearchAgent

SearchAgent是实现智能体Agent的类(它与世界交互) 且通过搜索函数做出规划。SearchAgent首先使用所提供的搜索函数规划出到达目标状态的行为，然后一次执行一个动作。

------------------------------

**实验检查**

TA将对每个人的完成情况进行检查，并进行现场提问。

-------------------------------

提交

提交整个search目录，包含说明文档。源程序中所编写每个函数标注清楚作者姓名和学号。目录命名：学号-姓名

-----------------------

**将逐一核查雷同代码，雷同代码按0分处理**