

All those colored walls,

Mazes give Pacman the blues,

So teach him to search.

所有那些彩色的墙壁，

迷宫给吃豆人带来忧郁，

所以请教他搜索。

**Introduction**介绍

In this project, your Pacman agent will find paths through his maze world, both to reach a particular location and to collect food efficiently. You will build general search algorithms and apply them to Pacman scenarios.

在这个项目中，您的吃豆子智能代理将在他的迷宫世界中找到路径，以到达特定位置并有效地收集食物。 您将构建通用搜索算法并将其应用于 Pacman 场景。

As in the Coding Diagnostic, this project includes an autograder for you to grade your answers on your machine. This can be run with the command: 与编码诊断一样，该项目包括一个自动评分器，供您在机器上对答案进行评分。 这可以使用以下命令运行：

python autograder.py

See the autograder tutorial in the Coding Diagnostic for more information about using the autograder. 有关使用自动评分器的更多信息，请参阅编码诊断中的自动评分器教程。

The code for this project consists of se veral Python files, some of which you will need to read and understand in order to complete the assignment, and some of which you can ignore. You can download all the code and supporting files as a [zip archive](https://inst.eecs.berkeley.edu/~cs188/su21/assets/files/search.zip).

该项目的代码由几个 Python 文件组成，其中一些您需要阅读和理解才能完成作业，而其中一些您可以忽略。 您可以将所有代码和支持文件下载为 zip 存档。

|  |  |
| --- | --- |
| **Files you'll edit:** **您将编辑的文件** | |
| search.py | Where all of your search algorithms will reside. |
| searchAgents.py | Where all of your search-based agents will reside. |
| **Files you might want to look at:** **您可能想查看的文件：** | |
| pacman.py | The main file that runs Pacman games. This file describes a Pacman GameState type, which you use in this project. |
| game.py | The logic behind how the Pacman world works. This file describes several supporting types like AgentState, Agent, Direction, and Grid. |
| util.py | Useful data structures for implementing search algorithms. |
| **Supporting files you can ignore:** **您可以忽略的支持文件：** | |
| graphicsDisplay.py | Graphics for Pacman |
| graphicsUtils.py | Support for Pacman graphics |
| textDisplay.py | ASCII graphics for Pacman |
| ghostAgents.py | Agents to control ghosts |
| keyboardAgents.py | Keyboard interfaces to control Pacman |
| layout.py | Code for reading layout files and storing their contents |
| autograder.py | Project autograder |
| testParser.py | Parses autograder test and solution files |
| testClasses.py | General autograding test classes |
| test\_cases/ | Directory containing the test cases for each question |
| searchTestClasses.py | Project 1 specific autograding test classes |

**Files to Edit and Submit:** **要编辑和提交的文件：**

 You will fill in portions of search.py and searchAgents.py during the assignment. Once you have completed the assignment. Please *do not* change the other files in this distribution or submit any of our original files other than these files.您将在分配期间填写 search.py 和 searchAgents.py 的部分内容。 完成作业后。 请不要更改此分发中的其他文件或提交除这些文件之外的任何我们的原始文件。

**Evaluation:** Your code will be autograded for technical correctness. Please *do not* change the names of any provided functions or classes within the code, or you will wreak havoc on the autograder. However, the correctness of your implementation – not the autograder’s judgements – will be the final judge of your score. If necessary, we will review and grade assignments individually to ensure that you receive due credit for your work. 评估：您的代码将自动评分以确保技术正确性。 请不要更改代码中提供的任何函数或类的名称，否则会对自动分级器造成严重破坏。 然而，你的实施的正确性——而不是自动评分器的判断——将是你分数的最终评判者。 如有必要，我们将单独审查和评分作业，以确保您的工作获得应有的荣誉。

**Academic Dishonesty:** We will be checking your code against other submissions in the class for logical redundancy. If you copy someone else’s code and submit it with minor changes, we will know. These cheat detectors are quite hard to fool, so please don’t try. We trust you all to submit your own work only; *please* don’t let us down. If you do, we will pursue the strongest consequences available to us.

学术不诚实：我们将根据课堂上的其他提交检查您的代码，以确保逻辑冗余。 如果你复制了别人的代码并提交了一些小的改动，我们会知道的。 这些作弊检测器很难被愚弄，所以请不要尝试。 我们相信大家只提交自己的作品； 请不要让我们失望。 如果你这样做，我们将追求我们所能得到的最强烈的后果。

**Getting Help:** You are not alone! If you find yourself stuck on something, contact the course staff for help. Office hours, section, and the discussion forum are there for your support; please use them. If you can’t make our office hours, let us know and we will schedule more. We want these projects to be rewarding and instructional, not frustrating and demoralizing. But, we don’t know when or how to help unless you ask.

获得帮助：您并不孤单！ 如果您发现自己遇到什么问题，请联系课程工作人员寻求帮助。 办公时间、版块和论坛随时为您提供支持； 请使用它们。 如果您不能安排我们的办公时间，请告诉我们，我们会安排更多。 我们希望这些项目是有益的和有指导意义的，而不是令人沮丧和士气低落。 但是，除非您提出要求，否则我们不知道何时或如何提供帮助。

**Discussion:** Please be careful not to post spoilers.

讨论：请注意不要发布剧透。

**Welcome to Pacman欢迎来到吃豆人**

After downloading the code ([search.zip](https://inst.eecs.berkeley.edu/~cs188/su20/assets/files/search.zip)), unzipping it, and changing to the directory, you should be able to play a game of Pacman by typing the following at the command line: 193 / 5000

下载代码（search.zip），解压并切换到目录后，您应该可以通过在命令行输入以下内容来玩 Pacman 游戏：

python pacman.py

Pacman lives in a shiny blue world of twisting corridors and tasty round treats. Navigating this world efficiently will be Pacman’s first step in mastering his domain. 吃豆子生活在一个闪闪发光的蓝色世界里，那里有扭曲的走廊和美味的圆形食物。 有效地导航这个世界将是 Pacman 掌握他的领域的第一步。

The simplest agent in searchAgents.py is called the GoWestAgent, which always goes West (a trivial reflex agent). This agent can occasionally win: searchAgents.py 中最简单的智能代理称为 GoWestAgent，它总是向西（一个平凡的反射智能代理）。 这个智能代理偶尔可以赢：

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

But, things get ugly for this agent when turning is required:

但是，当需要转弯时，这个智能代理的情况变得很糟糕：

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

If Pacman gets stuck, you can exit the game by typing CTRL-c into your terminal.

Soon, your agent will solve not only tinyMaze, but any maze you want.

如果 Pacman 卡住了，您可以通过在终端中输入 CTRL-c 退出游戏。

很快，您的代理不仅会解决 tinyMaze，还会解决您想要的任何迷宫。

Note that pacman.py supports a number of options that can each be expressed in a long way (e.g., --layout) or a short way (e.g., -l). You can see the list of all options and their default values via: 请注意，pacman.py 支持许多参数，每个选项都可以用长方式（例如，--layout）或短方式（例如，-l）表示。 您可以通过以下方式查看所有参数及其默认值的列表：

python pacman.py -h

Also, all of the commands that appear in this project also appear in commands.txt, for easy copying and pasting. In UNIX/Mac OS X, you can even run all these commands in order with bash commands.txt. 此外，该项目中出现的所有命令也出现在 commands.txt 中，以便于复制和粘贴。 在 UNIX/Mac OS X 中，您甚至可以使用 bash commands.txt 按顺序运行所有这些命令。

**Question 1 (3 points): Finding a Fixed Food Dot using Depth First Search问题 1（3 分）：使用深度优先搜索找到固定的食物点**

In searchAgents.py, you’ll find a fully implemented SearchAgent, which plans out a path through Pacman’s world and then executes that path step-by-step. The search algorithms for formulating a plan are not implemented – that’s your job. 在 searchAgents.py 中，您将找到一个完全实现的 SearchAgent，它规划出一条穿越吃豆人世界的路径，然后逐步执行该路径。 未实施用于制定计划的搜索算法——这是您的工作。

First, test that the SearchAgent is working correctly by running: 首先，通过运行以下命令测试 SearchAgent 是否正常工作：

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

The command above tells the SearchAgent to use tinyMazeSearch as its search algorithm, which is implemented in search.py. Pacman should navigate the maze successfully. 上面的命令告诉 SearchAgent 使用 tinyMazeSearch 作为其搜索算法，该算法在 search.py 中实现。 吃豆子应该成功地在迷宫中导航。

Now it’s time to write full-fledged generic search functions to help Pacman plan routes! Pseudocode for the search algorithms you’ll write can be found in the lecture slides. Remember that a search node must contain not only a state but also the information necessary to reconstruct the path (plan) which gets to that state. 现在是时候编写成熟的通用搜索功能来帮助 Pacman 规划路线了！ 您将编写的搜索算法的伪代码可以在讲座幻灯片中找到。 请记住，搜索节点不仅必须包含状态，还必须包含重建到达该状态的路径（计划）所需的信息。

***Important note:*** All of your search functions need to return a list of *actions* that will lead the agent from the start to the goal. These actions all have to be legal moves (valid directions, no moving through walls). 重要提示：您的所有搜索功能都需要返回一个操作列表，这些操作将引导代理从开始到目标。 这些动作都必须是合法的移动（有效的方向，不能穿墙移动）。

***Important note:*** Make sure to **use** the Stack, Queue and PriorityQueue data structures provided to you in util.py! These data structure implementations have particular properties which are required for compatibility with the autograder. 重要提示：确保使用 util.py 中提供给您的 Stack、Queue 和 PriorityQueue 数据结构！ 这些数据结构实现具有与自动评分器兼容所需的特定属性。

*Hint:* Each algorithm is very similar. Algorithms for DFS, BFS, UCS, and A\* differ only in the details of how the fringe is managed. So, concentrate on getting DFS right and the rest should be relatively straightforward. Indeed, one possible implementation requires only a single generic search method which is configured with an algorithm-specific queuing strategy. (Your implementation need *not* be of this form to receive full credit). 提示：每个算法都非常相似。 DFS、BFS、UCS 和 A\* 的算法仅在如何管理边缘的细节上有所不同。 因此，专注于获得正确的 DFS，其余的应该相对简单。 事实上，一种可能的实现只需要一个通用搜索方法，该方法配置有特定于算法的排队策略。 （您的实施无需采用这种形式即可获得全部学分）。

Implement the depth-first search (DFS) algorithm in the depthFirstSearch function in search.py. To make your algorithm *complete*, write the graph search version of DFS, which avoids expanding any already visited states.

在 search.py 中的 depthFirstSearch 函数中实现深度优先搜索 (DFS) 算法。 为了使您的算法完整，请编写 DFS 的图搜索版本，以避免扩展任何已访问的状态。

Your code should quickly find a solution for: 您的代码应该很快找到解决方案：

python pacman.py -l tinyMaze -p SearchAgent

python pacman.py -l mediumMaze -p SearchAgent

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

The Pacman board will show an overlay of the states explored, and the order in which they were explored (brighter red means earlier exploration). Is the exploration order what you would have expected? Does Pacman actually go to all the explored squares on his way to the goal? Pacman 板将显示探索状态的叠加，以及探索它们的顺序（更亮的红色表示更早的探索）。 探索顺序是否符合您的预期？ 吃豆子真的会在前往目标的路上走遍所有探索过的方格吗？

*Hint:* If you use a Stack as your data structure, the solution found by your DFS algorithm for mediumMaze should have a length of 130 (provided you push successors onto the fringe in the order provided by getSuccessors; you might get 246 if you push them in the reverse order). Is this a least cost solution? If not, think about what depth-first search is doing wrong. 提示：如果你使用 Stack 作为你的数据结构，你的 DFS 算法为 mediumMaze 找到的解决方案应该有 130 的长度（假设你按照 getSuccessors 提供的顺序将后继推到边缘；如果你推它们，你可能会得到 246 以相反的顺序）。 这是成本最低的解决方案吗？ 如果没有，请考虑深度优先搜索做错了什么。

**Question 2 (3 points): Breadth First Search问题 2（3 分）：广度优先搜索**

Implement the breadth-first search (BFS) algorithm in the breadthFirstSearch function in search.py. Again, write a graph search algorithm that avoids expanding any already visited states. Test your code the same way you did for depth-first search. 在search.py 的breadthFirstSearch 函数中实现广度优先搜索(BFS) 算法。 同样，编写一个图搜索算法，避免扩展任何已经访问过的状态。 以与深度优先搜索相同的方式测试您的代码。

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

python pacman.py -l bigMaze -p SearchAgent -a fn=bfs -z .5

Does BFS find a least cost solution? If not, check your implementation. 是否找到了成本最低的解决方案？ 如果没有，请检查您的实施。

*Hint:* If Pacman moves too slowly for you, try the option --frameTime 0. 提示：如果 Pacman 对您来说移动太慢，请尝试使用 --frameTime 0 选项。

*Note:* If you’ve written your search code generically, your code should work equally well for the eight-puzzle search problem without any changes. BFS

注意：如果您已经编写了通用的搜索代码，那么您的代码应该同样适用于八谜搜索问题，无需任何更改。

python eightpuzzle.py

**Question 3 (3 points): Varying the Cost Function问题 3（3 分）：改变成本函数**

While BFS will find a fewest-actions path to the goal, we might want to find paths that are “best” in other senses. 虽然 BFS 会找到到达目标的最少动作路径，但我们可能希望找到其他意义上“最佳”的路径。

Consider mediumDottedMaze and mediumScaryMaze. 考虑 mediumDottedMaze 和 mediumScaryMaze。

By changing the cost function, we can encourage Pacman to find different paths. For example, we can charge more for dangerous steps in ghost-ridden areas or less for steps in food-rich areas, and a rational Pacman agent should adjust its behavior in response. 通过改变成本函数，我们可以鼓励 Pacman 寻找不同的路径。 例如，我们可以对鬼怪地区的危险步骤收取更高的费用，或者对食物丰富的地区的步骤收取更少的费用，理性的吃豆代理应该相应地调整其行为。

Implement the uniform-cost graph search algorithm in the uniformCostSearch function in search.py. We encourage you to look through util.py for some data structures that may be useful in your implementation. You should now observe successful behavior in all three of the following layouts, where the agents below are all UCS agents that differ only in the cost function they use (the agents and cost functions are written for you): 在 search.py 中的 uniformCostSearch 函数中实现统一成本图搜索算法。 我们鼓励您查看 util.py 以查找可能对您的实现有用的一些数据结构。 您现在应该在以下所有三个布局中观察到成功的行为，其中下面的代理都是 UCS 代理，仅在它们使用的成本函数上有所不同（代理和成本函数是为您编写的）：

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

python pacman.py -l mediumDottedMaze -p StayEastSearchAgent

python pacman.py -l mediumScaryMaze -p StayWestSearchAgent

*Note:* You should get very low and very high path costs for the StayEastSearchAgent and StayWestSearchAgent respectively, due to their exponential cost functions (see searchAgents.py for details). 注意：由于 StayEastSearchAgent 和 StayWestSearchAgent 的指数成本函数，您应该分别获得非常低和非常高的路径成本（有关详细信息，请参阅 searchAgents.py）。

**Question 4 (3 points): A\* search**

**问题 4（3 分）：A\* 搜索**

Implement A\* graph search in the empty function aStarSearch in search.py. A\* takes a heuristic function as an argument. Heuristics take two arguments: a state in the search problem (the main argument), and the problem itself (for reference information). The nullHeuristic heuristic function in search.py is a trivial example. 在search.py中的空函数aStarSearch中实现A\*图搜索。 A\* 将启发式函数作为参数。 启发式算法采用两个参数：搜索问题中的状态（主要参数）和问题本身（用于参考信息）。 search.py 中的 nullHeuristic 启发式函数是一个简单的例子。

You can test your A\* implementation on the original problem of finding a path through a maze to a fixed position using the Manhattan distance heuristic (implemented already as manhattanHeuristic in searchAgents.py). 您可以使用曼哈顿距离启发式（已在 searchAgents.py 中作为 manhattanHeuristic 实现）在原始问题上测试您的 A\* 实现，该问题是通过迷宫找到一条到达固定位置的路径。

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

You should see that A\* finds the optimal solution slightly faster than uniform cost search (about 549 vs. 620 search nodes expanded in our implementation, but ties in priority may make your numbers differ slightly). What happens on openMaze for the various search strategies? 您应该看到 A\* 找到最佳解决方案的速度比统一成本搜索稍快（在我们的实现中扩展了大约 549 与 620 个搜索节点，但优先级的关系可能会使您的数字略有不同）。 各种搜索策略在 openMaze 上会发生什么？

**Question 5 (3 points): Finding All the Corners**

**问题 5（3 分）：找到所有的角落**

The real power of A\* will only be apparent with a more challenging search problem. Now, it’s time to formulate a new problem and design a heuristic for it.

只有在更具挑战性的搜索问题中，A\* 的真正威力才会显现出来。 现在，是时候制定一个新问题并为其设计启发式方法了。

In *corner mazes*, there are four dots, one in each corner. Our new search problem is to find the shortest path through the maze that touches all four corners (whether the maze actually has food there or not). Note that for some mazes like tinyCorners, the shortest path does not always go to the closest food first! *Hint*: the shortest path through tinyCorners takes 28 steps. 在角落迷宫中，有四个点，每个角落一个。 我们的新搜索问题是找到穿过迷宫的、触及所有四个角的最短路径（无论迷宫是否真的有食物）。 请注意，对于像 tinyCorners 这样的迷宫，最短路径并不总是先到最近的食物！ 提示：通过 tinyCorners 的最短路径需要 28 步。

*Note: Make sure to complete Question 2 before working on Question 5, because Question 5 builds upon your answer for Question 2.* 注意：请确保在处理问题 5 之前完成问题 2，因为问题 5 建立在您对问题 2 的回答之上。

Implement the CornersProblem search problem in searchAgents.py. You will need to choose a state representation that encodes all the information necessary to detect whether all four corners have been reached. Now, your search agent should solve: 在 searchAgents.py 中实现 CornersProblem 搜索问题。 您将需要选择一种状态表示，该表示对检测是否已到达所有四个角所需的所有信息进行编码。 现在，您的搜索代理应该解决：

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

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

To receive full credit, you need to define an abstract state representation that *does not* encode irrelevant information (like the position of ghosts, where extra food is, etc.). In particular, do not use a Pacman GameState as a search state. Your code will be very, very slow if you do (and also wrong). 要获得全部的分数，您需要定义一个抽象的状态表示，该表示不编码不相关的信息（例如鬼魂的位置、额外食物的位置等）。 特别是，不要使用 Pacman GameState 作为搜索状态。 如果您这样做（并且也是错误的），您的代码将非常非常缓慢。

*Hint:* The only parts of the game state you need to reference in your implementation are the starting Pacman position and the location of the four corners. 提示：您在实现中唯一需要参考的游戏状态部分是开始吃豆子的位置和四个角的位置。

Our implementation of breadthFirstSearch expands just under 2000 search nodes on mediumCorners. However, heuristics (used with A\* search) can reduce the amount of searching required. 我们的breadthFirstSearch 实现在mediumCorners 上扩展了不到2000 个搜索节点。 但是，启发式（与 A\* 搜索一起使用）可以减少所需的搜索量。

**Question 6 (3 points): Corners Problem: Heuristic问题 6（3 分）：角落问题：启发式**

*Note: Make sure to complete Question 4 before working on Question 6, because Question 6 builds upon your answer for Question 4.* *注意：确保在处理问题 6 之前完成问题 4，因为问题 6 建立在您对问题 4 的回答之上。*

Implement a non-trivial, consistent heuristic for the CornersProblem in cornersHeuristic. *为cornersHeuristic 中的CornersProblem 实现一个重要的、一致的启发式方法。*

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

*Note:* AStarCornersAgent is a shortcut for 注意：AStarCornersAgent 是一个快捷方式

-p SearchAgent -a fn=aStarSearch,prob=CornersProblem,heuristic=cornersHeuristic

***Admissibility vs. Consistency:*** Remember, heuristics are just functions that take search states and return numbers that estimate the cost to a nearest goal. More effective heuristics will return values closer to the actual goal costs. To be *admissible*, the heuristic values must be lower bounds on the actual shortest path cost to the nearest goal (and non-negative). To be *consistent*, it must additionally hold that if an action has cost *c*, then taking that action can only cause a drop in heuristic of at most *c*.

可接受性与一致性：请记住，启发式只是采用搜索状态并返回估计成本到最近目标的数字的函数。更有效的启发式方法将返回更接近实际目标成本的值。为了被接受，启发式值必须是到最近目标的实际最短路径成本的下限（并且非负）。为了保持一致，它还必须另外认为，如果一个动作的成本为 c，那么采取该动作只能导致最多 c 的启发式下降。

Remember that admissibility isn’t enough to guarantee correctness in graph search – you need the stronger condition of consistency. However, admissible heuristics are usually also consistent, especially if they are derived from problem relaxations. Therefore it is usually easiest to start out by brainstorming admissible heuristics. Once you have an admissible heuristic that works well, you can check whether it is indeed consistent, too. The only way to guarantee consistency is with a proof. However, inconsistency can often be detected by verifying that for each node you expand, its successor nodes are equal or higher in in f-value. Moreover, if UCS and A\* ever return paths of different lengths, your heuristic is inconsistent. This stuff is tricky!

请记住，可接受性不足以保证图搜索的正确性——您需要更强的一致性条件。然而，可接受的启发式通常也是一致的，特别是如果它们来自问题松弛。因此，通常最容易从头脑风暴可接受的启发式开始。一旦你有了一个运行良好的可接受的启发式方法，你也可以检查它是否确实是一致的。保证一致性的唯一方法是使用证明。但是，通过验证您扩展的每个节点，其后续节点的 f 值是否相等或更高，通常可以检测到不一致。此外，如果 UCS 和 A\* 曾经返回不同长度的路径，则您的启发式方法是不一致的。这东西很棘手！

***Non-Trivial Heuristics:*** The trivial heuristics are the ones that return zero everywhere (UCS) and the heuristic which computes the true completion cost. The former won’t save you any time, while the latter will timeout the autograder. You want a heuristic which reduces total compute time, though for this assignment the autograder will only check node counts (aside from enforcing a reasonable time limit).

非平凡启发式：平凡启发式是在任何地方都返回零 (UCS) 的启发式方法和计算真实完成成本的启发式方法。前者不会为您节省任何时间，而后者会使自动分级器超时。您需要一种减少总计算时间的启发式方法，但对于此分配，自动分级器只会检查节点数（除了强制执行合理的时间限制）。

***Grading:*** Your heuristic must be a non-trivial non-negative consistent heuristic to receive any points. 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 be graded:

评分：您的启发式必须是非平凡的非负一致启发式才能获得任何分数。确保您的启发式在每个目标状态下都返回 0，并且永远不会返回负值。根据您的启发式扩展的节点数量，您将被评分：

|  |  |
| --- | --- |
| **Number of nodes expanded**  **展开的节点数** | **Grade**  **成绩** |
| more than 2000 | 0/3 |
| at most 2000 | 1/3 |
| at most 1600 | 2/3 |
| at most 1200 | 3/3 |

*Remember:* If your heuristic is inconsistent, you will receive *no* credit, so be careful! 请记住：如果您的启发式不一致，您将得不到任何信用，所以要小心！

**Question 7 (4 points): Eating All The Dots**

**问题 7（4 分）：吃掉所有的点**

Now we’ll solve a hard search problem: eating all the Pacman food in as few steps as possible. For this, we’ll need a new search problem definition which formalizes the food-clearing problem: FoodSearchProblem in searchAgents.py (implemented for you). A solution is defined to be a path that collects all of the food in the Pacman world. For the present project, solutions do not take into account any ghosts or power pellets; solutions only depend on the placement of walls, regular food and Pacman. (Of course ghosts can ruin the execution of a solution! We’ll get to that in the next project.) If you have written your general search methods correctly, A\* with a null heuristic (equivalent to uniform-cost search) should quickly find an optimal solution to testSearch with no code change on your part (total cost of 7).

现在我们将解决一个困难的搜索问题：在尽可能少的步骤中吃掉所有的 Pacman 食物。 为此，我们需要一个新的搜索问题定义来形式化食物清理问题：searchAgents.py 中的 FoodSearchProblem（为您实现）。 一个解决方案被定义为一条收集 Pacman 世界中所有食物的路径。 对于目前的项目，解决方案没有考虑任何鬼魂或能量颗粒； 解决方案只取决于墙壁、普通食物和吃豆子的位置。 （当然，幽灵会破坏解决方案的执行！我们将在下一个项目中讨论。）如果您正确编写了通用搜索方法，带有空启发式（相当于统一成本搜索）的 A\* 应该很快 找到 testSearch 的最佳解决方案，您无需更改代码（总成本为 7）。

python pacman.py -l testSearch -p AStarFoodSearchAgent

*Note:* AStarFoodSearchAgent is a shortcut for 注意：AStarFoodSearchAgent 是一个快捷方式

-p SearchAgent -a fn=astar,prob=FoodSearchProblem,heuristic=foodHeuristic

You should find that UCS starts to slow down even for the seemingly simple tinySearch. As a reference, our implementation takes 2.5 seconds to find a path of length 27 after expanding 5057 search nodes. 您应该会发现，即使对于看似简单的 tinySearch，UCS 也开始变慢。 作为参考，我们的实现在扩展 5057 个搜索节点后需要 2.5 秒才能找到长度为 27 的路径。

*Note: Make sure to complete Question 4 before working on Question 7, because Question 7 builds upon your answer for Question 4.* 注意：确保在处理问题 7 之前完成问题 4，因为问题 7 建立在您对问题 4 的回答之上。在 searchAgents.py 中使用 FoodSearchProblem 的一致启发式填充 foodHeuristic。 在trickySearch 板上试试你的代理：

Fill in foodHeuristic in searchAgents.py with a *consistent* heuristic for the FoodSearchProblem. Try your agent on the trickySearch board: 在 searchAgents.py 中使用 FoodSearchProblem 的一致启发式填充 foodHeuristic。 在trickySearch 板上试试你的代理：

python pacman.py -l trickySearch -p AStarFoodSearchAgent

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

我们的 UCS 代理在大约 13 秒内找到了最佳解决方案，探索了 16,000 多个节点。

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: 任何非平凡非负一致启发式将获得 1 分。 确保您的启发式在每个目标状态下都返回 0，并且永远不会返回负值。 根据您的启发式扩展的节点数量，您将获得额外的积分：

*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. 请记住：如果您的启发式不一致，您将得不到任何信用，所以要小心！ 你能在短时间内解决 mediumSearch 吗？ 如果是这样，我们要么非常非常印象深刻，要么您的启发式方法不一致。

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

**问题 8（3 分）：次优搜索**

|  |  |
| --- | --- |
| **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) |

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. 有时，即使使用 A\* 和良好的启发式方法，也很难通过所有点找到最佳路径。 在这些情况下，我们仍然希望快速找到一条相当好的路径。 在本节中，您将编写一个始终贪婪地吃掉最近点的代理。 在 searchAgents.py 中为您实现了 ClosestDotSearchAgent，但它缺少一个关键函数，可以找到到最近点的路径。

Implement the function findPathToClosestDot in searchAgents.py. Our agent solves this maze (suboptimally!) in under a second with a path cost of 350: 在 searchAgents.py 中实现函数 findPathToClosestDot。 我们的代理在一秒钟内解决了这个迷宫（次优！），路径成本为 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.

提示：完成 findPathToClosestDot 的最快方法是填写 AnyFoodSearchProblem，它缺少目标测试。 然后，使用适当的搜索功能解决该问题。 解决方案应该很短！

您的 ClosestDotSearchAgent 不会总是找到穿过迷宫的最短路径。 确保您了解原因并尝试提出一个小例子，其中反复前往最近的点不会导致找到吃掉所有点的最短路径。