AI Project-1 : Search in Pacman 制作报告

基科31 蒋文韬 2013011717

2015 年 4 月 1 日

1 Tasks

1.1 Finding a Fixed Food Dot

Implement DFS algorithm in the depthFirstSearch function in search.py
Implement BFS algorithm in the breadthFirstSearch function in search.py
Implement the uniform-cost graph search algorithm in the uniformCostSearch function in search.py
Implement A* graph search in the empty function aStarSearch in search.py

1.2 Finding All the Corners

Implement the CornersProblem search problem in searchAgents.py
Implement a non-trivial, consistent heuristic for the CornersProblem in cornersHeuristic.

1.3 Eating All The Dots

Fill in foodHeuristic in searchAgents.py with a consistent heuristic for the FoodSearchProblem.

1.4 Suboptimal Search

Implement the function findPathToClosestDot in searchAgents.py

2 代码实现与运行结果分析

在开始动手编写程序之前，我从未接触过python这门语言，仅有极少量的C语言编程经验，甚至对类与对象等面向对象的编程方法几乎没有具体的了解。因此花费了数小时阅读pacman.py，game.py等文件，研究与理解其中的规律与原理，终于对程序的大致原理与待解决的问题形式有了一定程度的了解，对面向对象编程的特点与方法也有了基础的理解。

在语言方面，我也对python的风格与方法及其简明易上手的特色有了一定的感受。通过边进行编程实践边查阅相关教材的方法逐渐熟练对python语言的运用。

在有了以上准备之后，我便开始依次解决Pacman Project中的问题。

2.1 Finding a Fixed Food Dot using Depth First Search

由于对数据结构的了解不够，我最开始运用递归实现了DFS，并在测试中能够正常运行。但在BFS的实现中需要更改大部分代码，并且改写后的BFS无法通过Berkeley AI Materials网站上建议的eightpuzzle.py的测试。因此，后来在理解了堆栈等数据结构的工作原理的基础上，重写了BFS与DFS的代码。因为思路正确，BFS与DFS代码间的差距就仅是使用FIFO队列与使用堆栈的数据结构的区别而已。

但在autograder.py的测试中，q1的第四个测试没有通过，分析发现是添加已访问的节点的步骤的问题，应添加在探索过程中已展开的节点，而不是立即添加已探索到的节点。改写代码后测试通过。

在tinyMaze中，DFS展开了15个节点，花费0.0秒内找到了长度为10的路径；

在mediumMaze中，DFS展开了146个节点，花费0.0秒找到了长度为130的路径；

在bigMaze中，DFS展开了390个节点，花费0.0秒内找到了长度为210的路径。

若改变对successor的搜索次序，则：

在tinyMaze中，DFS展开了15个节点，花费0.0秒内找到了长度为8的路径；

在mediumMaze中，DFS展开了269个节点，花费0.0秒找到了长度为246的路径；

在bigMaze中，DFS展开了466个节点，花费0.0秒内找到了长度为210的路径。

2.2 Breadth First Search

在实现BFS算法时，第一次编写时我仍使用的递归，之后改写后才改为使用Queue数据结构。

在tinyMaze中，BFS展开了15个节点，花费0.0秒找到了长度为8的路径；

在mediumMaze中，BFS展开了269个节点，花费0.1秒找到了长度为68的路径；

在bigMaze中，BFS展开了620个节点，花费0.3秒找到了长度为210的路径。

与DFS的结果比较可知，DFS受搜索顺序影响，可能很快就能搜索到解，因此展开的节点数可能较少，但解的长度可能离最优较远也可能恰好就是最优解，也可能搜索完几乎所有节点后才找到解；而BFS则能保证找到最优解。

2.3 Varying the Cost Function

实现uniformCostSearch时，只需要在BFS的基础上将数据结构改为PriorityQueue，并以路径的相应权值作为优先队列的priority，加入新节点时使用getCostOfAction()函数返回相应路径的权值。

在mediumMaze中，UCS展开了269个节点，花费0.5秒找到了长度为68的路径；

在mediumDottedMaze中，UCS展开了186个节点，花费0.3秒找到了cost为1的路径；

在mediumScaryMaze中，UCS展开了108个节点，花费0.2秒找到了cost为68719479864的路径。

因为使用了不同的SearchAgent，采用了不同的指数形式的cost function，因此后两者的cost非常低或者高。可见UCS也能找到最佳路径，不过对于权值没有变化的问题等价于BFS而相对于BFS搜索耗时稍多，而UCS能够完成对路径有偏好的问题的解决。

进行autograder.py测试时，最后一组没有通过，分析后发现为会将中途搜索到的目标节点添加进已搜索节点的集合中，导致算法出错。因此在添加节点进入已搜索节点时添加了一条if语句判断是否为目标节点，随后测试通过。

2.4 A* search

实现A*算法，将UCS中计算路径的cost的语句更改为当前路径的cost加上启发式函数给出的当前节点的估值即可。

在mediumMaze中，A*展开了221个节点，花费0.1秒找到了长度为68的路径；

在bigMaze中，A*展开了549个节点，花费0.3秒找到了长度为210的路径。

对于openMaze，各算法表现如下表：

搜索算法	展开节点数	耗时(s)	路径长度
A*	535	0.3	54
UCS	682	0.4	54
BFS	682	0.4	54
DFS	808	0.1	298

除DFS外的算法均找到了最短路径，其中A*算法展开节点较少且耗时较短。DFS虽相对较快但路径很长。可见A*算法表现最优。

2.5 Finding All the Corners

开始思考这个问题的时候，我首先注意到，找到所有corner可能需要走重复的路线，并需要记录与返回已走过的路径给isGoalState判断是否已寻到所有四个corner。因此需要改变state的结构。我首先想到的是将走过的路径信息加入到state中，不过算法并未成功。分析发现为这样更改后搜索算法中对节点是否已经搜索过的判断语句将失效，程序会反复展开已经展开过的节点。于是思考后只向state中加入已经到过的corner的信息，这样判断语句的效果自动变为寻找到了新的corner后则可以走重复的节点。

在tinyCorners中，算法展开了435个节点，花费0.0秒找到了长度为28的路径，为最短路径；

在mediumCorners中，算法展开了2448个节点，花费0.2秒找到了长度为106的路径。

2.6 Corners Problem: Heuristic

我首先编写的cornersHeuristic函数将当前节点与最远corner间的manhattan距离作为返回值，运行成功并通过了autograder.py的测试，运行情况为花费0.2秒，展开了1355个节点并找到了长度为106的路径。可见启发式算法搜索的节点数大大小于BFS算法，但仍较多。

于是我又将启发函数改为返回从当前节点出发，按从近到远的顺序走遍剩余corner的总manhattan距离，程序运行成功并通过了autograder.py的测试，运行情况为花费0.1秒，展开了901个节点并找到了长度为106的路径。这次搜索的节点数更少。

2.7 Eating All The Dots

在编写foodHeuristic之前，首先测试了UCS算法在这个问题下的工作情况，在tinySearch中，UCS展开了5057个节点，花费2.8秒，找到了长度为27的路径；在trickySearch中，UCS展开了16688个节点，花费24.9秒找到了长度为60的路径。

仿照corners Problem中的做法，我将每个food均当作corner，完全类似地按从近到远的顺序走遍剩余food，并将距离改为了更加接近实际距离的mazeDistance。程序在运行时表现十分突出，在tinySearch中，仅展开了27个节点，花费0.1秒，找到了长度为27的路径；在trickySearch中，仅展

开了95个节点，花费1.3秒找到了长度为60的路径。但好景不长，这个启发函数并没有通过可采纳性的测试，并且在mediumSearch中，因为food数目十分众多，程序运行了数分钟均没有结果。

于是一番纠结后我将启发函数改为了类似corners Problem中第一次编写的启发函数，即返回当前节点与最远的food间的距离，仍然采用mazeDistance。这次，程序在tinySearch中，展开了2354个节点，花费2.7秒，找到了长度为27的路径；在trickySearch中，展开了4110个节点，花费13.8秒找到了长度为60的路径。该启发函数通过了所有测试，但仍未能在能够容忍的时间内解决mediumSearch。

考虑到每次求mazeDistance均调用了BFS，于是尝试将mazeDistance换回了manhattanDistance，程序在tinySearch中，展开了2432个节点，花费0.9秒，找到了长度为27的路径；在trickySearch中，展开了9444个节点，花费7.7秒找到了长度为60的路径。可见更改后程序运行更快，但相应地，搜索的节点数有所增加。程序也通过了测试但得分则没有那么高。但更改距离的计算方法后，程序仍未能在能够容忍的时间内解决mediumSearch，分析原因应为程序每走一步需遍历所有food的本质并未变化，而未能解决mediumSearch的关键问题便是food太多，并且距离的偏差较大也导致了搜索节点数的增加。

因此最后选择了启发函数的第二种方案。四种搜索的运行情况对比如下表：

搜索算法/启发函数返回值	展开节点数(tiny/tricky)	耗时(s) (tiny/triky)	是否可用
UCS	5057 / 16688	2.8 / 24.9	
由近至远计算总mazeDistance	27 / 95	0.1 / 1.3	不可用
距最远food的mazeDistance	2354 / 4110	2.7 / 13.8	可用
距最远food的manhattanDistance	2432 / 9444	0.9 / 7.7	可用

2.8 Suboptimal Search

完成AnyFoodSearchProblem中的isGoalState的编写后，即判断当前节点是否为目标节点，而目标节点为距当前节点最近的food。在ClosestDotSearchAgent中返回该problem下的BFS的搜索结果即可。程序运行成功，找到了吃完所有food的长度为350的路径。

3 总结与感想

这次Project前后大概花费了我5天的空闲时间，从了解python的基本语法与程序结构，到浏览pacman.py等程序中的代码，再到反复调试自己的代码与程序，排查测试中发现的错漏。

通过这次作业，我掌握并实践了基础的python编程方法，对类与对象等面向对象的编程方法也有了从初步到逐渐深入的了解，对理论课堂上学习到的若干种图搜索算法都有了更深刻的理解以及实践的经验，更加亲身体验了BFS，DFS等若干种搜索算法具体的实现方法以及各自的特点，对A*算法的思想与其优良的表现也有了更深的印象与理解。对堆栈，队列，优先队列等以前从未接触过的关于数据结构的知识也有了初步的了解与实践运用。

由于测试样例有限，我的编程经验本身也十分欠缺，程序当中一定还有一些不容易发现的错漏以及诸多有待改进的细节。总而言之，这次作业让我收获颇丰，希望下一次作业我能完成得更好更快。