迷宫搜索任务实验说明文档

# 程序设计：

本次实验总共进行了5中所学的方法的代码编程实现，分别是宽度优先，深度优先，等费用算法，爬山法，A\*算法。具体可参考代码，对每种算法都有注释。

需要的模块时Gamestate，copy(python的赋值时引用传递，所以需要copy.deepcopy方法)。

Class Node—对访问位置的节点抽象数据

getDirect和StepTo合用，前置获得两个点之间的相对方向，后者执行node1到node2的actionlist。

OpenPush针对节点费用采用插入排序插入新的节点，使得费用按照从小到大排序。

Nodefee计算当前节点之前的费用g(n)，Restfee评估到达目标的最佳剩余费用h(n)。为了目标2和3，对g的计算考虑已经走过的路程长度以及获得分数，后者时费用减小，前者让费用增加。对h的计算由于目标位置未知，所以假定已经走过的步数愈多，越接近目标，同时也考虑bonus的影响，二者的增加都使得费用降低。

BFS与DFS采用策略是先query寻找目标，然后从当前位置走到目标即可，不考虑bonus等。

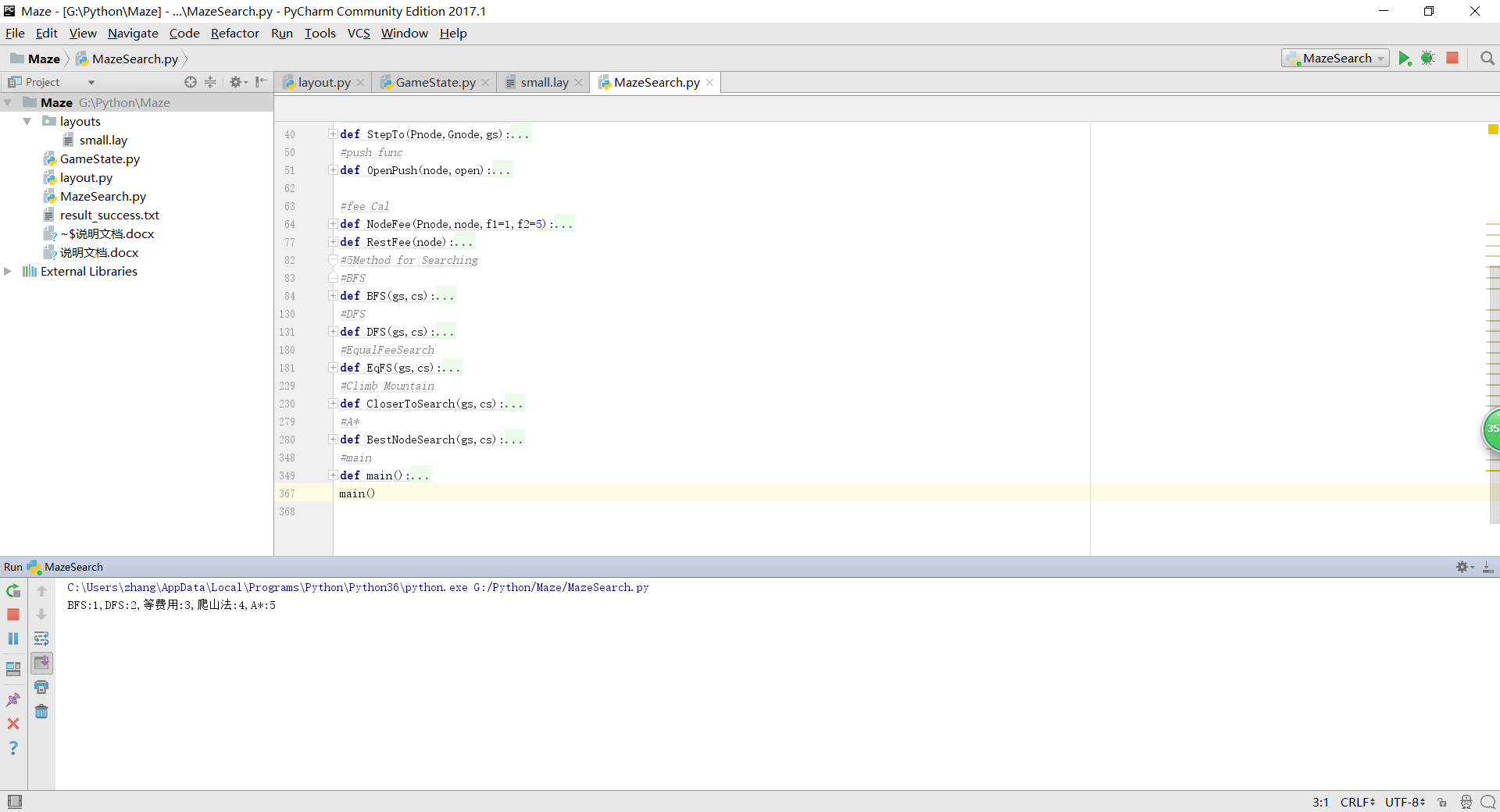
EqFS，CloserToSearch，BestNodeSearch在考虑寻找目标的同时，会优先寻找bonus之和大于0的节点并走到该节点，然后以该节点作为新的出发点继续query。

由于路程长度不是目标，所以测试中评估费用的函数主要考虑的不是路程长度而是得分与少量的query。

程序使用python3.6作为解释器，所以对助教代码里的print “”代码都改为print(“”)，所以希望测试的时候能够使用这个修改。

# 程序使用与测试：

程序在运行时，会出现如下提示：按照数字选择方法就开时search寻路，并给出路径和结果



下面是一个用例实例：

C:\Users\zhang\AppData\Local\Programs\Python\Python36\python.exe G:/Python/Maze/MazeSearch.py

BFS:1,DFS:2,等费用:3,爬山法:4,A\*:5

1

found Goal

from Pos(1,5)|Bonus(0) to Pos(9,5)|Bonus(3)

Pos(9,4)|Bonus(3) to Pos(9,5)|Bonus(3)

Pos(9,3)|Bonus(2) to Pos(9,4)|Bonus(3)

Pos(8,3)|Bonus(2) to Pos(9,3)|Bonus(2)

Pos(7,3)|Bonus(2) to Pos(8,3)|Bonus(2)

Pos(6,3)|Bonus(2) to Pos(7,3)|Bonus(2)

Pos(6,4)|Bonus(2) to Pos(6,3)|Bonus(2)

Pos(6,5)|Bonus(2) to Pos(6,4)|Bonus(2)

Pos(5,5)|Bonus(2) to Pos(6,5)|Bonus(2)

Pos(4,5)|Bonus(2) to Pos(5,5)|Bonus(2)

Pos(3,5)|Bonus(2) to Pos(4,5)|Bonus(2)

Pos(2,5)|Bonus(0) to Pos(3,5)|Bonus(2)

Pos(1,5)|Bonus(0) to Pos(2,5)|Bonus(0)

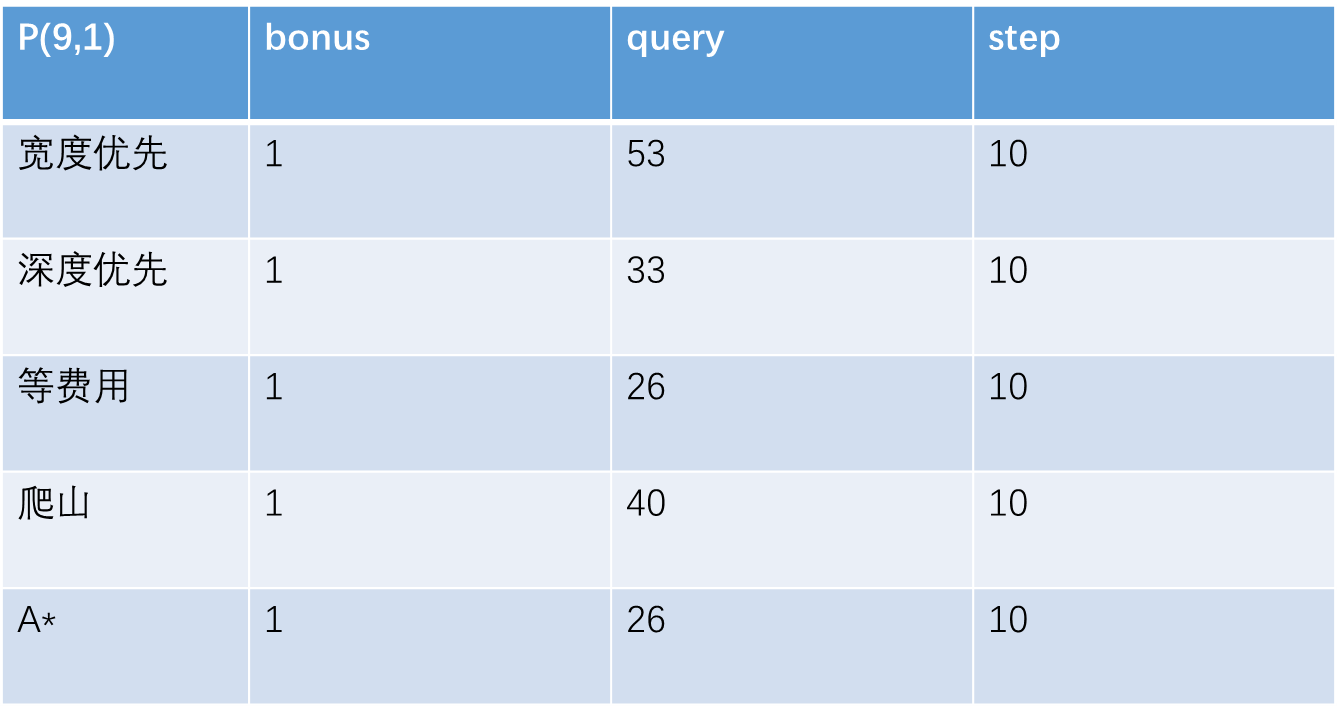
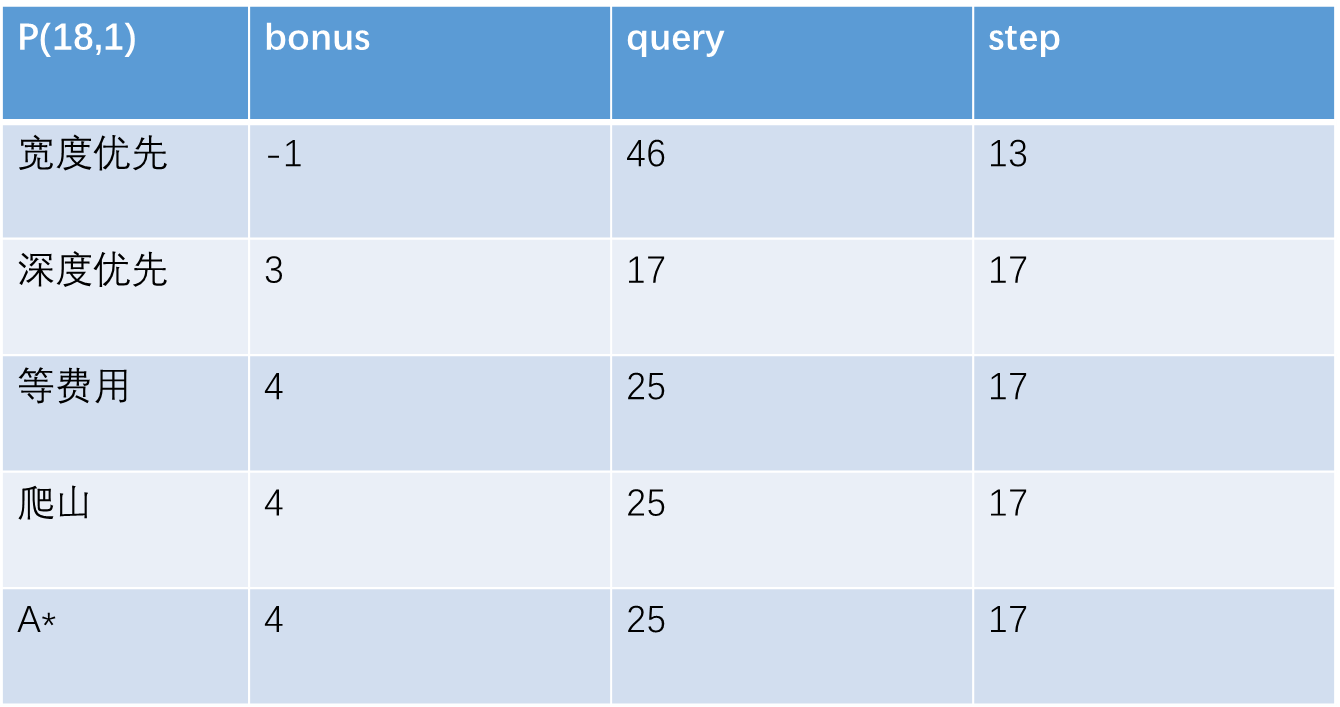
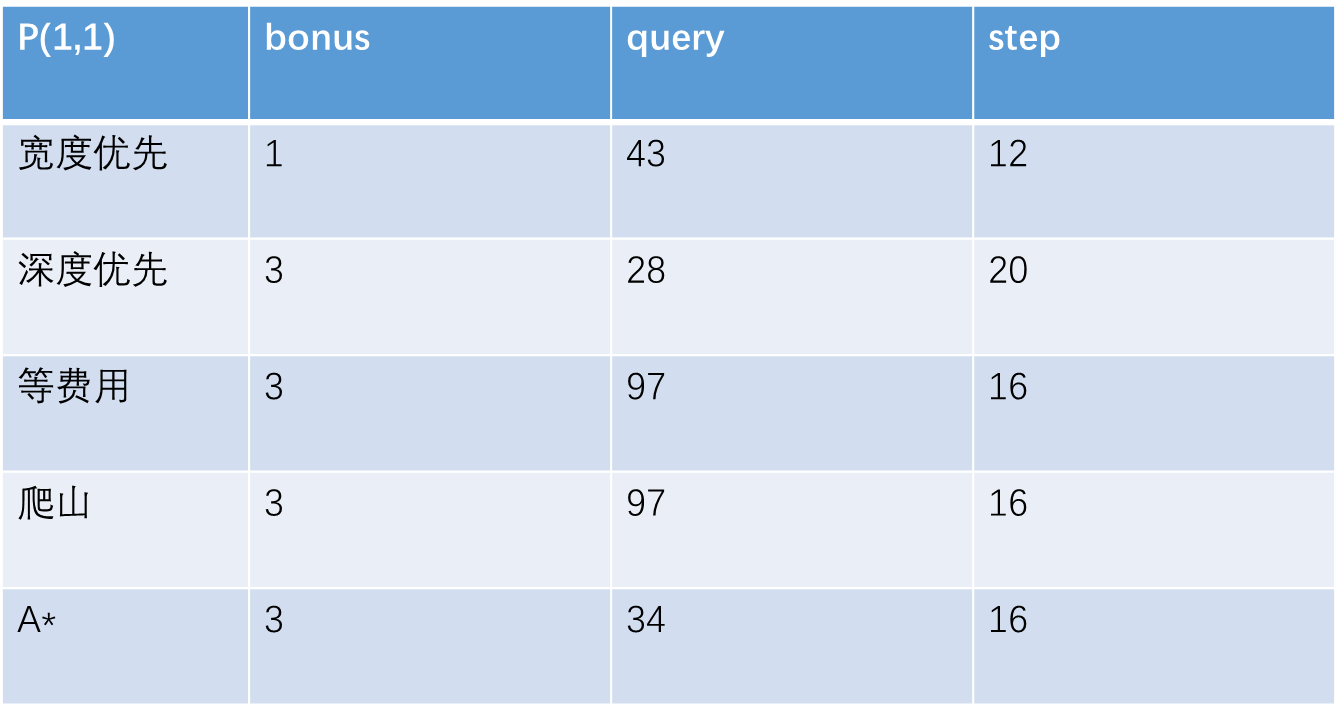
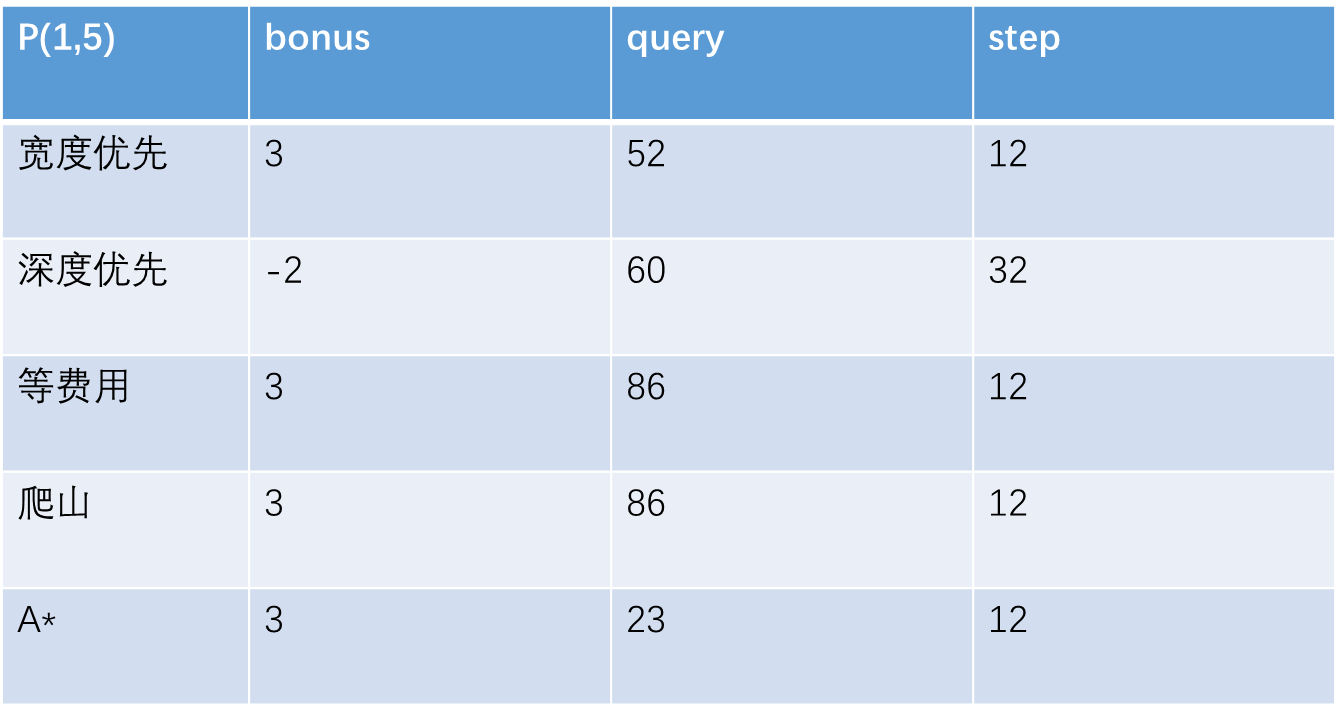
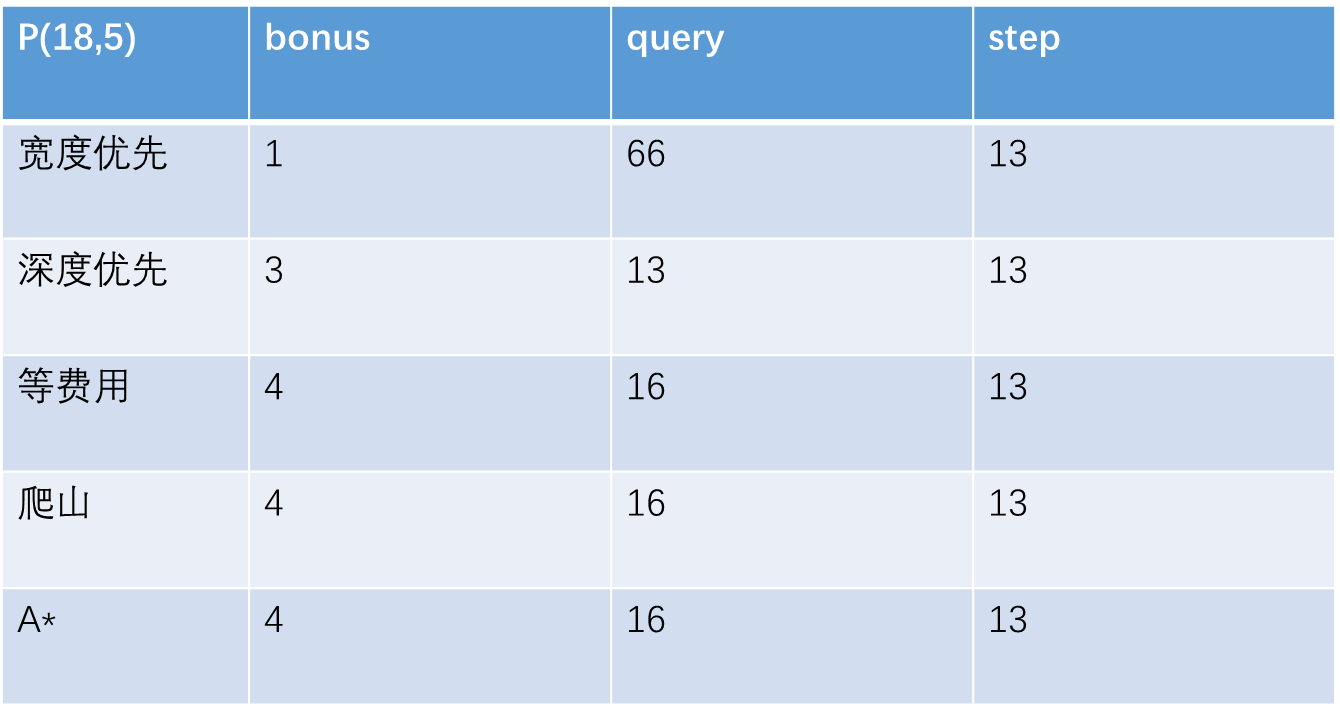
finished with

bonus: 3

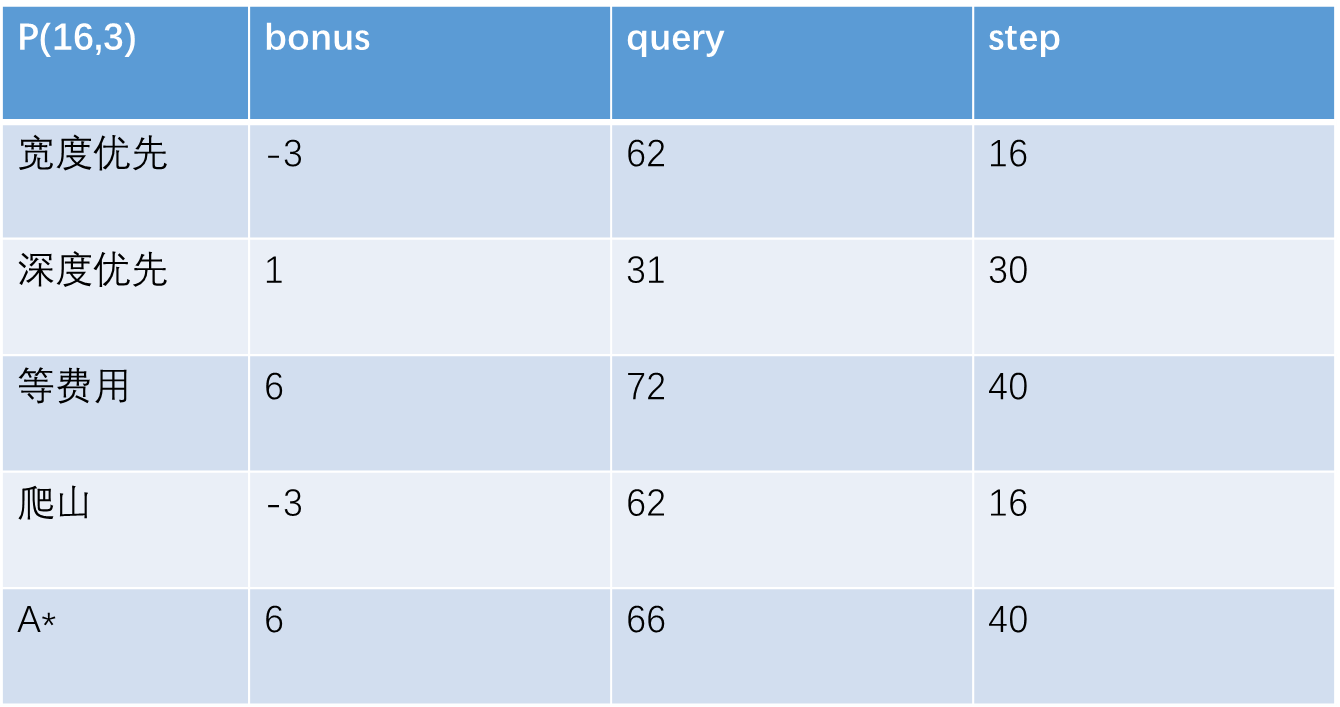
query: 52

action steps: 12

Process finished with exit code 0

选用不同起始位置的测试点测试，结果如下：目标G(9,5)

随机测试：P(16,3)🡪G(2,1)



通过对比，不难发现在套用费用f=g+h(n)的这个体系时，深度优先策略由于g最大，所以能在最少query内找到G，但是他的bonus，step等却非最优；宽度优先算法在既定框架下不考虑bonus，但能够找到最优路线，使得setp最少，同样也未考虑bonus。

在等费用算法时，为了对目标2，3找到较优解，股加入了对bonus的考虑，Nodefee不局限于找到最短路径，否则若在在这个每步费用以单位计算的情况下，肯定会使得结果于BFS相同，因为他们这样做g(n)会相同。

爬山法时，由于不清楚目标位置，所以假定走过的steps技术越多，离目标越近，然后同时考虑bonus影响，二者都与Restfee计算是负相关，使得优先访问那些距离G比较近且bonus大的节点。

A\*算法则是对等费用于爬山法在费用计算上的综合，由于g于h非线性相关，所以每步query时都得思考是否到达最优节点，通过于open及close表的比较更替实现。

总体而言A\*算法比爬山法快，但是得分与路程步数皆优于爬山法；爬山法给的启发信息不是很强，所以所以在加速时人能找到较好路径解；等费用算法的估值使得对bonus与最短路径做了平衡。

**关于一处细节的说明：**

**由于计算费用的时候可能会因为减法得到一个负数，所以为了避免回路query，我使用了对一个节点（A\*算法除外）的对已访问拓展的情况拒绝第二次访问，这一般是由于排序而不会执行的一句话，但有个别人情况，例如两个节点通向的子后代有相同的情况。**