Project 1: Search in Pacman

曹旭 16307110230

Question 1 (2 points): Finding a Fixed Food Dot using Depth First Search

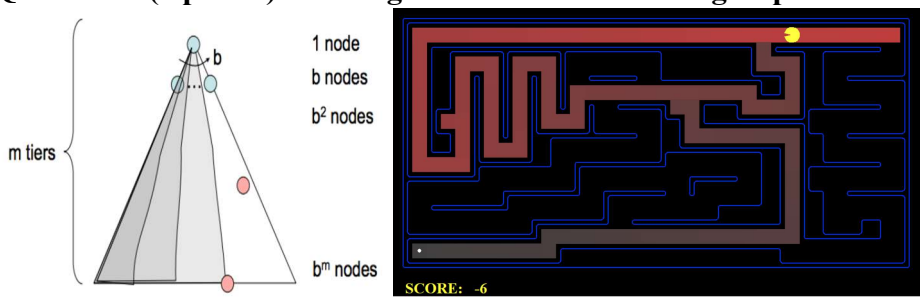


Figure.1 Depth First Search

Depth-first search (DFS) is a strategy for exploration that always selects the deepest fringe node from the start node for expansion. The core method to implement DFS is a last-in, first-out (LIFO) stack. The figure above showed us that the Pacman do not go to all the explored squares on his way to the goal. Also, the path of DFS is not the most optimized path.

Question 2 (2 points): Breadth First Search

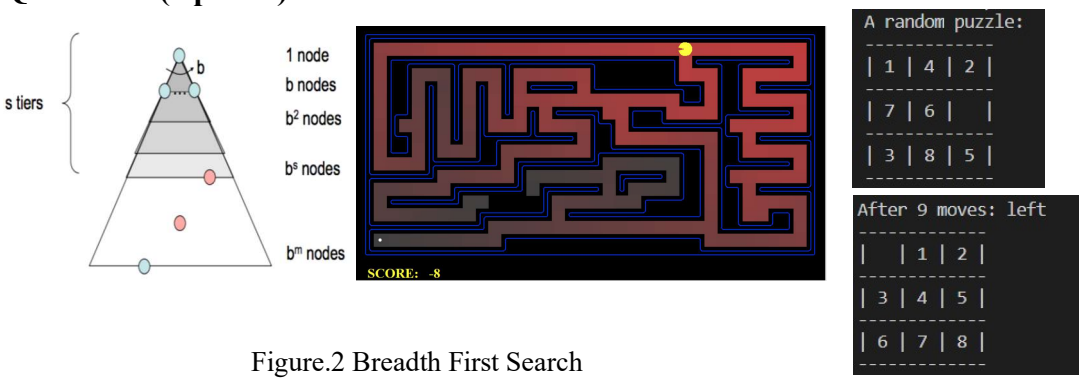


Figure.2 Breadth First Search

Breadth-first search is a strategy for exploration that always selects the shallowest fringe node from the start node for expansion. BFS's core method is a first-in, first-out (FIFO) queue. The figure above also showed us that we can use BFS to solve eight-puzzle search problem.

Question 3 (2 points): Varying the Cost Function

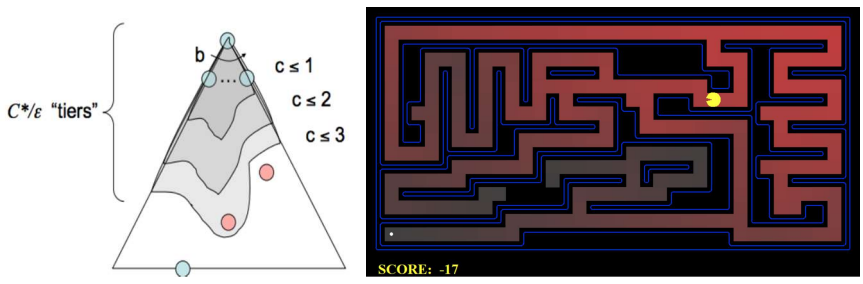


Figure.3 Uniform cost search

Uniform cost search (UCS) is a strategy for exploration that always selects the lowest cost fringe node from the start node for expansion. To represent the fringe for UCS, the choice is usually a heap-based priority queue, where the weight for a given enqueued node v is the path cost

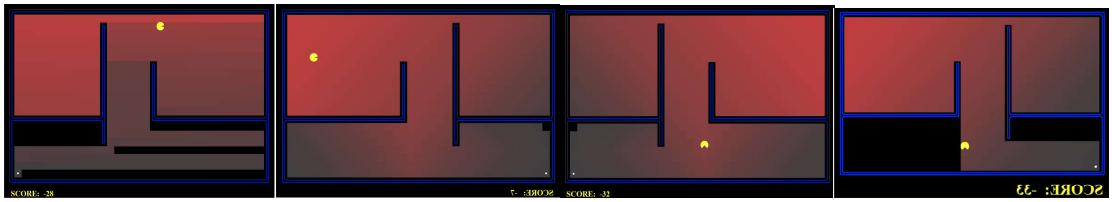
from the start node to v, or the backward cost of v. However, in this question, the result of UCS is just same to BFS, because the path cost is always 1 in this pacman problem.

Question 4 (3 points): A* search

```
Path found with total cost of 210 in 0.3 seconds
Search nodes expanded: 549
Pacman emerges victorious! Score: 300
Average Score: 300.0
Scores:        300.0
Win Rate:      1/1 (1.00)
Record:        Win
```

Figure.4 A* search

A* search is a strategy for exploration that always selects the fringe node with the lowest estimated total cost for expansion, where total cost is the entire cost from the start node to the goal node. A* finds the optimal solution slightly faster than uniform cost search (about 549 vs. 620 search nodes expanded in our implementation).



DFS BFS UCS A*

Figure.5 openMaze comparison

What happens on openMaze for the various search strategies?

As Figure.5 showed above, DFS takes an unusually tortuous route. BFS and UCS traverses almost all nodes, but the solution is the optimal solution. A* traverses less nodes than UCS and BFS, and it also find the optimal solution.

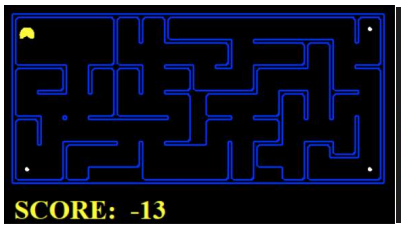
Question 5 (2 points): Finding All the Corners

This question is a sub-question of question 6, It is easy to design the function in class CornersProblem(search.SearchProblem).

```
Path found with total cost of 106 in 0.3 seconds
Search nodes expanded: 1966
Pacman emerges victorious! Score: 434
Average Score: 434.0
Scores:        434.0
Win Rate:      1/1 (1.00)
Record:        Win
```

Figure.6 Result of BFS in corner problem

Question 6 (2 points): Corners Problem: Heuristic




```
Path found with total cost of 106 in 0.2 seconds
Search nodes expanded: 741
Pacman emerges victorious! Score: 434
Average Score: 434.0
Scores:        434.0
Win Rate:      1/1 (1.00)
Record:        Win
```

Figure.7 Result of My Heuristic in corner problem

It can be proved that Manhattan distance is better than Euclidean distance, Octile and so on. So now our task is to find a heuristic function based on Manhattan distance.

At the beginning, I choose the maximam Manhattan distance of one of foods to current position of pacman, but the result can not get the highest score. Then, I search the Internet and Literature library, soon find an interesting heuristic function design.

If we know the look of labyrinth(like the walls and boundary), then we try to find Manhattan distance between two currently furthest fruits in labyrinth - let's call that x. And also find Manhattan distance from current Pacman position to the closer of previous two fruits - let's call that y. The heuristic function is x+y[1].

It can be easy to understand this thought. The heuristic function is to estimate the cost of the current distance to the end point (eat all foods), but this value is difficult to predict, however our x+y reflects the cost estimate of pacman eating the two foods furthest apart. This distance can effectively approximate the cost of the current distance to the end point.

- $\mathrm{g(n)}$ - The function representing total backwards cost computed by UCS.
- $\mathrm{h(n)}$ - The heuristic value function, or estimated forward cost, used by greedy search.
- $\mathrm{f(n)}$ - The function representing estimated total cost, used by A* search. $\mathrm{f(n)} = \mathrm{g(n)} + \mathrm{h(n)}$.
- x - Manhattan distance between two currently furthest fruits in labyrinth.
- y - Manhattan distance from current Pacman position to the closer of previous two fruits.
- P, P' - The current position and the next step position
- A, B - The current two currently furthest fruits A and B
- A', B' - The potential two future furthest fruits A and B

$$\mathrm{f(n)} = \mathrm{g(n)} + \mathrm{h(n)}$$

Defining $h^*(n)$ as the true optimal forward cost to reach a goal state from a given node n

$$\forall \mathrm{n}, \quad 0 \leq \mathrm{h(n)} \leq h^*(n)$$

$$\text{Each step: } \nabla \mathrm{h(n)} \leq 1$$

In our case, it has two situations:

1. The pacman move one step, y decrease 1, x do not change;

$$\mathrm{h(n)} = \mathrm{Manhattan(A, B)} + \min(\mathrm{Manhattan}(A, P), \mathrm{Manhattan(B, P)})$$

$$\mathrm{h(n')} = \mathrm{Manhattan(A, B)} + \min(\mathrm{Manhattan}(A, P'), \mathrm{Manhattan(B, P')})$$

$$\mathrm{h(n)} - h(n') = 1 \rightarrow \nabla \mathrm{h(n)} \leq 1$$

2. The pacman move one step and eat the closer of previous two fruits, x decrease 1 or more, y increase a value that do not bigger than the value x decrease. Both situation we have $\nabla \mathrm{h(n)} \leq 1$

$$\mathrm{h(n)} = \mathrm{Manhattan(A, B)} + \min(\mathrm{Manhattan}(A, P), \mathrm{Manhattan(B, P)})$$

$$\mathrm{h(n')} = \mathrm{Manhattan(A', B')} + \min\big(\mathrm{Manhattan}(A', P'), \mathrm{Manhattan(B', P')}\big)$$

$$= \min\Big(\mathrm{Manhattan}(A', P') + \mathrm{Manhattan(A', B')}, \mathrm{Manhattan(B', P')} + \mathrm{Manhattan(A', B')}\Big)$$

$$\text{suppose } \mathrm{P}' \ is \ A\text{:}$$

$$\mathrm{h(n)} = \ \mathrm{Manhattan(P', B)} + 1$$

$$\mathrm{h(n)} - h(n') = 1 + \ \mathrm{Manhattan(P', B)} - h(n')$$

$$= 1 - \mathrm{k}$$

$$\mathrm{k} = \mathrm{Manhattan}(A', P') + \mathrm{Manhattan(A', B')} - \ \mathrm{Manhattan(P', B)} \leq 1 \ \ or$$

$$\mathrm{k} = \mathrm{Manhattan(B', P')} + \mathrm{Manhattan(A', B')} - \ \mathrm{Manhattan(P', B)} \leq 1$$

$$\therefore \mathrm{h(n)} - h(n') \leq 1$$

Both situation we have $\nabla \mathrm{h(n)} \leq 1$. It can be proved that this heuristic is admissible and consistent.

Question 7 (3 points): Eating All The Dots

In this question, if we still choose the heuristic function of question 6, we can not achieve the goal 'at most 7000', But it close to 7000 expended node. Therefore, we will make some minor improvements to the heuristic function. Of course, these improvements must also be admissible and consistent.

This time I will replace Manhattan distances to real distance. It is not esay to do that.

First, I use UCS to calculate real distance. Then I build a dictionary to store the food position at class FoodSearchProblem. It can only expand 376 node and finish in 0.7 seconds.

```
PS D:\Desktop\人工智能\pj-1-search\search> python pacman.py -l trickySearch -p AStarFoodSearchAgent
Path found with total cost of 60 in 0.7 seconds
Search nodes expanded: 376
Pacman emerges victorious! Score: 570
Average Score: 570.0
Scores:        570.0
Win Rate:      1/1 (1.00)
Record:        Win
```

Question 8 (2 points): Suboptimal Search

Use UCS or BFS to finish it, The result:

```
Path found with cost 350.
Pacman emerges victorious! Score: 2360
Average Score: 2360.0
Scores:        2360.0
Win Rate:      1/1 (1.00)
Record:        Win
```

The final score:

```
Finished at 18:50:26

Provisional grades
==================
Question q1: 2/2
Question q2: 2/2
Question q3: 2/2
Question q4: 3/3
Question q5: 2/2
Question q6: 2/2
Question q7: 3/3
Question q8: 2/2
------------------
Total: 18/18
```

References:

[1] https://stackoverflow.com/questions/9994913/pacman-what-kinds-of-heuristics-are-mainly-used

[2] Denero J , Klein D . Teaching introductory artificial intelligence with Pac-man[J]. In Proceedings of the 1st Symposium on Educational Advances in Artificial Intelligence (EAAI, 2013.