Report of Project 1

张奕朗 16307130242

Question 1: Finding a Fixed Food Dot using Depth First Search

In this question, DFS is implemented with some adjustment:

- `current_depth`: a marker used to record the depth, which is 0 for the root node.
- `actions`: a list used to store current actions.
- `actions` are changed dynamically according to `current_depth`: Push `current_depth+1` (the depth of successors) into `fringe` (a stack) together with successors. When it has been popped from `fringe`, compare it with `current_depth`.
    - If the `depth` popped form `fringe` is bigger than `current_path`, we can infer that we're exploring deeper nodes, so just update `current_path` with path.
    - While if `depth` is less equal than `current_path`, it indicates we're tracing back to shallower depth or still in current depth. `actions` should be adjusted to `actions[0:depth-current_depth-1]` to remove the last several actions we don't need.

  Space complexity of O(bm) has been achieved with this small trick.

Results:

	tinyMaze	mediumMaze	bigMaze
total cost	10	130	210
search nodes expanded	15	146	390

Question 2: Breadth First Search

In BFS, the details is a little different from DFS:

- `fringe`: a queue.
- `actions`: a list used to store all the actions which we have visited, since we can't decide which actions to abandon before reaching our goal. (space complexity of $O(b^m)$)
- `state2data`: a dictionary storing `state: parent_index`, where `parent_index` is the index of the action from the parent node of `state` to `state`, i.e. use `actions[state2data[state]]` to get this action. `state2data` helps us to trace the actions from our goal state to root node reversely.

Results:

	mediumMaze	bigMaze
total cost	68	210
search nodes expanded	267	617

And the algorithm also works well for eight-puzzle search problem.

Question 3: Varying the Cost Function

UCS differs from BFS in some ways:

- `fringe`: a priority queue.
- `state2data`: a dictionary storing `state: (parent_index, cost)`.
- Do goal test when visiting a node. (DFS and BFS do goal test when expanding nodes)

	mediumMaze	StayEastSearch	StayWestSearch
search nodes expanded	277	191	108

Question 4: A* search

A* resembles UCS except:

- When pushing successor into `fringe`, use `cost + heuristic` as priority.

Results: 210 cost and 551 search nodes expanded for `bigMaze`. And for `openMaze`:

	DFS	BFS	UCS	A*
total cost	183	54	54	54
search nodes expanded	576	679	1260	983

This result is not beyond my expectation, since DFS and BFS do goal-testing when expanding nodes and UCS and A* when visiting nodes. Therefore, search nodes expanded by UCS is about twice as much as BFS. DFS cannot always find the optimal solution, thus getting a total cost of 183. As for A*, it implements heuristic function to get a better performance than UCS.

Question 5: Finding All the Corners

- state representation: `(x, y, cornerVisited)`, where `x, y` is the coordinate position and `visitedCorner` is a tuple of four bool element recording whether four corners have been visited prospectively (`False` for unvisited and `True` for visited). Besides, `cornerVisited` is updated in method `getSuccessors`.

Results:

	tinyCorners	mediumCorners
total cost	28	106
search nodes expanded	260	1943

Question 6: Corners Problem: Heuristic

My heuristic function is constructed from relaxed problem: Pacman could move through the walls. Since the function is the **actual cost** of the **relaxed** problem, it satisfied **admissibility** and **consistency**, and obviously non-trivial and non-negative. It's computed according to the number of unvisited corners:

- 4 corners unvisited: go to the nearest corner first (Manhattan distance), then walk along three edges of the four whose sum is the shortest.
- 3 corners unvisited: go to the nearest of the two corner which are not adjacent (the vertex of shape 'L') first, then to the rest two along the edges.
- 2 corners unvisited: go to the nearest one first, then to another.
- 1 corners unvisited: go there.
- 1 corners unvisited: return 0.

Results (much better than Question 5):

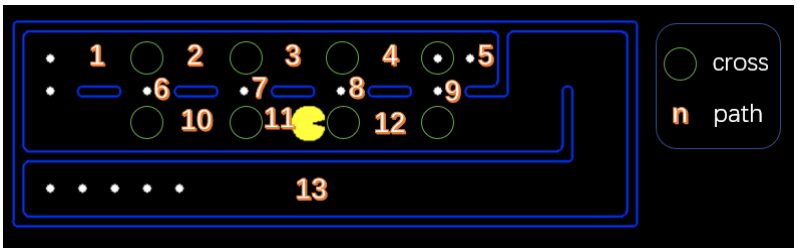
	tinyCorners	mediumCorners
total cost	28	106
search nodes expanded	210	841

Question 7: Eating All The Dots

The core idea of my algorithm is dividing the maze into several independent small regions by a global relaxed problem (this idea comes from problem dividing in CSP), then solving sub-problem in each small regions, and each region includes a `path` and its adjacent `cross`:

- `path`: a continuous range of position which is not a wall and is surrounded (WNES four directions) by more equal than two walls.
- `cross`: a position which is not a wall and is surrounded (WNES four directions) by less than two walls.

  For an example:



- Global relaxed problem: All the cross are connected, i.e. Pacman could transfer from one cross to another without any cost as if it stands on one of the cross.

Note: It seems that Pacman transferring to adjacent cross with 1 cost may be a better choice, but it would destroy the independence between each sub-problem.

- Sub-problem: Since there is only one continuous path without any bifurcation in each region, the task of eating all the foods equals to eating the farthest food in the region.
  - If Pacman is not located in current region (refer to region that we're considering), it could only eat the farthest food from the closest cross then back to the cross.

$$cost_{i \neq j} = 2 * min_{cross} max_{food}(cross_j, food_j)$$

  - If Pacman is located in current region, it could only eat the farthest food in the region then go to the closest cross.

$$cost_{i=j} = max(position_i, food_j) + min(food_{farthest}, cross_j)$$

  - When achieving our goal, we don't need to quit the region from the farthest food to the cross, so I use a marker `max_leaveRegion` to record the farthest distance to leave region, and substract it from our total cost which is simply the sum of cost in each region.
  - Call BFS algorithm when searching the distance between two positions, and it runs extreme fast, because there is only one continuous path without any bifurcation, i.e. always only one successor when expanding nodes. For an example, when searching the cost in `region13` in picture above, BFS only expands 28 nodes to get to the farthest food (just along the path), which is far more less than the nodes expanded in `foodSearchProblem` (showed below), thus can be ignored.
- Admissibility and consistency: According to my analysis above, my algorithm solved the **actual cost** of **independent** sub-problems divided from a **relaxed** problem, so it's both **admissible** and **consistent**.
- Results: Path found with total cost of 60 in 14.0 seconds. Search nodes expanded: 4232.
- Possible improvements (but I have no time to achieve):
  - Store the cost of exploring each region from its crosses, avoid repeated computation.
  - Take the location of each cross into consideration and transferring is not free for all the crosses.
  - In fact, the purpose of region dividing is for solving `mediumSearch`, but I failed, and further consideration and improvements are needed.

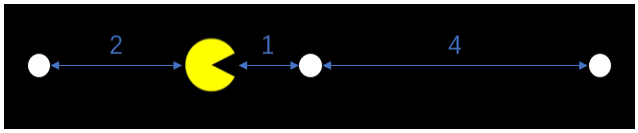
Question 8: Suboptimal Search

The goal test for `AnyFoodSearchProblem` aims to determine whether there is food on the state being tested. So it returns `True` when there is and `False` when not.

As for the method `findPathToClosestDot`, we can just implement BFS to search the nearest food based on the goal test in `AnyFoodSearchProblem`.

Result: cost of 350.

An example that greedy cannot lead to optimal:



In this example, greedy algorithm will cost: 1+3+8=11, while the optimal solution is 2+3+4=9.