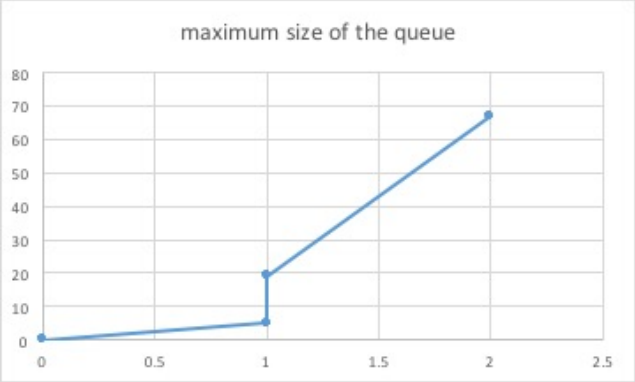
Xinyi Ma xim002@ucsd.edu
Yuanchi Ha yuha@ucsd.edu
Yijun Zhang yiz160@ucsd.edu

21.2:

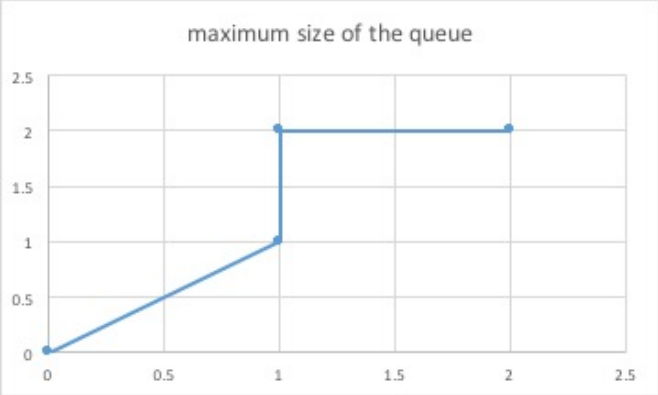
Description: there are only two rooms in a line and some of them are dirty. There is a vacuum in either of the two room and we want to use the BFS algorithm to clean all rooms.
Data structure: queue
Efficiency analysis:
input: 0,0,0 maximum size of the queue: 0
input: 0,1,1 maximum size of the queue: 5
input: 0,1,0 maximum size of the queue: 19
input: 1,1,1 maximum size of the queue: 67
In the initial state, if all rooms are clean, then it is the fastest case. If one room is dirty, and the vacuum is in that room, then the algorithm needs to expand some nodes to find the goal state. If one room is dirty and the vacuum is not in that room, the algorithm will expand more nodes. If both rooms are dirty, the algorithm will expand much more nodes.
Big-O analysis: O(b^m) where b is the maximum forward branch and m is the maximum path length m
Graph: (the x-axis is the number of dirty rooms)



21.3:

Description: there are only two rooms in a line and some of them are dirty. There is a vacuum in either of the two room and we want to use the DFS algorithm to clean all rooms.
Data structure: stack
Efficiency analysis:
input: 0,0,0 maximum size of the stack: 0
input: 0,1,1 maximum size of the stack: 2
input: 0,1,0 maximum size of the stack: 1
input: 1,1,1 maximum size of the stack: 2
The maximum size of the stack doesn't change a lot when the algorithm deal with different kinds of situation, which means that DFS is more efficient in finding the goal state
Big-O analysis: O(b^m) where b is the maximum forward branch and m is the maximum path length m

Graph: (the x-axis is the number of dirty rooms)



21.4: (use visited state of nodes)

Description: there are only two rooms in a line and some of them are dirty. There is a vacuum in either of the two room and we want to use the depth-limit search(DLS) algorithm to clean all rooms.

Data structure: stack

Efficiency analysis:

input: 0,0,0 maximum size of the stack: 0
input: 0,1,1 maximum size of the stack: 2
input: 0,1,0 maximum size of the stack: 1
input: 1,1,1 maximum size of the stack: 2

The maximum size of the stack doesn't change a lot when the algorithm deal with different kinds of situation, which means that DLS is also more efficient in finding the goal state.

We used recursion in this case

Big-O analysis: O(b^m) where b is the maximum forward branch and m is the maximum path length m

21.5: (didn't use visited state of nodes)

Description: there are only two rooms in a line and some of them are dirty. There is a vacuum in either of the two room and we want to use the iterative deepening depth-limit search(IDS) algorithm to clean all rooms.

Data structure: stack

Efficiency analysis:

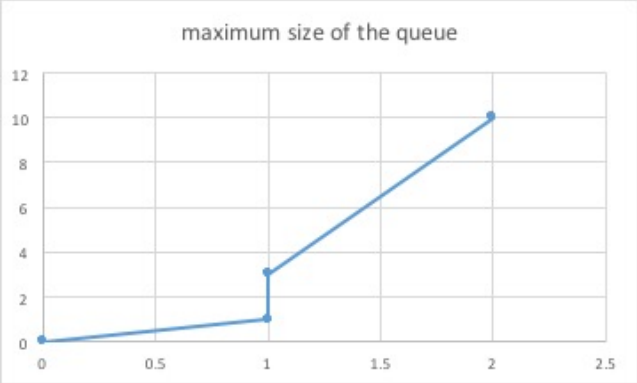
input: 0,0,0 expanded node: 0
input: 0,1,1 expanded node: 1
input: 0,1,0 expanded node: 3
input: 1,1,1 expanded node: 10

When two rooms are both dirty, the algorithm will expand more nodes, but less than the expanded nodes in BFS.

We used recursion in this case

Big-O analysis: O(b^m) where b is the maximum forward branch and m is the maximum path length m

Graph: (the x-axis is the number of dirty rooms)



21.6: (use visited state of nodes)

Description: there are only two rooms in a line and some of them are dirty. There is a vacuum in either of the two room and we want to use the A* algorithm to clean all rooms.

Data structure: priority queue

Efficiency analysis:

input: 0,0,0 expanded node: 0
input: 0,1,1 expanded node: 1
input: 0,1,0 expanded node: 1
input: 1,1,1 expanded node: 3

When two rooms are both dirty, the algorithm will expand more nodes, but less than the expanded nodes in BFS.

Big-O analysis: O(b^m) where b is the maximum forward branch and m is the maximum path length m

Graph: (the x-axis is the number of dirty rooms)

22.2:

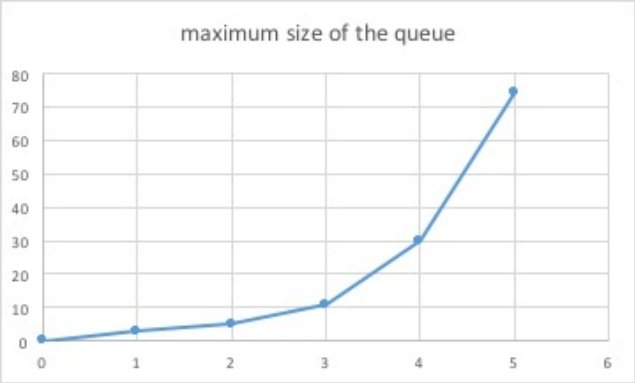
Description: there are n rooms in a line and some of them are dirty. There is a vacuum in one of the rooms and we want to use the BFS algorithm to clean all rooms.

Data structure: queue

Efficiency analysis:

input: 0,0,0,0 maximum size of the queue: 0
input: 0,1,0,1 maximum size of the queue: 3
input: 0,1,1,1 maximum size of the queue: 5
input: 1,1,1,1 maximum size of the queue: 11
input: 1,1,1,1,1 maximum size of the queue: 30
input: 1,1,1,1,1,1 maximum size of the queue: 74

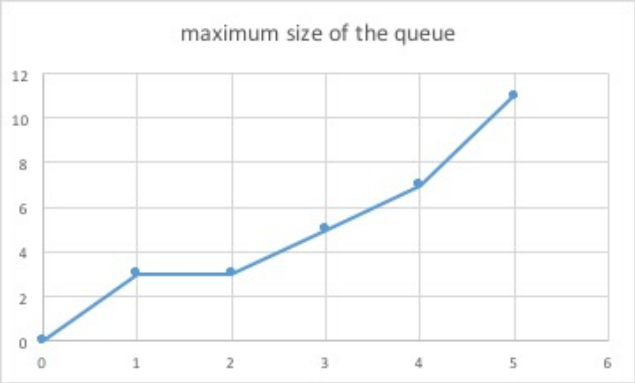
With the increase number of the dirty rooms, the number of the expanded nodes also increases. In addition, if the number of rooms increases, then the number of expanded nodes also increases.
Big-O analysis: O(b^m) where b is the maximum forward branch and m is the maximum path length m
Graph: (the x-axis is the number of dirty rooms)



22.3:

Description: there are n rooms in a line and some of them are dirty. There is a vacuum in one of the rooms and we want to use the DFS algorithm to clean all rooms.
Data structure: stack
Efficiency analysis:
input: 0,0,0,0 maximum size of the stack: 0
input: 0,1,0,1 maximum size of the stack: 3
input: 0,1,1,1 maximum size of the stack: 3
input: 1,1,1,1 maximum size of the stack: 5
input: 1,1,1,1,1 maximum size of the stack: 7
input: 1,1,1,1,1,1 maximum size of the stack: 11

With the increase number of the dirty rooms, the number of the expanded nodes also increases. In addition, if the number of rooms increases, then the number of expanded nodes also increases. However, compared to BFS, the maximum size of the stack doesn't change a lot when the algorithm deal with different kinds of situation, which means that DFS is more efficient in finding the goal state
Big-O analysis: O(b^m) where b is the maximum forward branch and m is the maximum path length m
Graph: (the x-axis is the number of dirty rooms)



22.4: (use visited state of nodes)

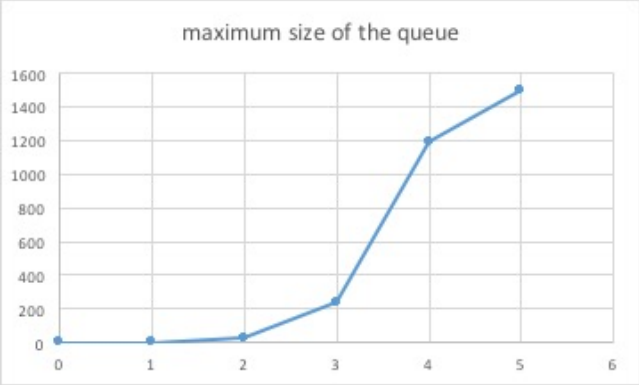
Description: there are n rooms in a line and some of them are dirty. There is a vacuum in one of the rooms and we want to use the depth-limit search(DLS) algorithm to clean all rooms.
Data structure: stack
Efficiency analysis:
input: 0,0,0,0 maximum size of the stack: 0
input: 0,1,0,1 maximum size of the stack: 3
input: 0,1,1,1 maximum size of the stack: 3
input: 1,1,1,1 maximum size of the stack: NONE
input: 1,1,1,1,1 maximum size of the stack: NONE
input: 1,1,1,1,1,1 maximum size of the stack: NONE

The maximum size of the stack doesn't change a lot when the algorithm deal with different kinds of situation, which means that DLS is also more efficient in finding the goal state. However, DLS is not guarantee to find the goal state and return a path.
We used recursion in this case
Big-O analysis: O(b^m) where b is the maximum forward branch and m is the maximum path length m

22.5: (didn't use visited state of nodes)

Description: there are n rooms in a line and some of them are dirty. There is a vacuum in one of the rooms and we want to use the iterative-deepening-depth-limit search(IDDLS) algorithm to clean all rooms.
Data structure: stack
Efficiency analysis:
input: 0,0,0,0 maximum size of the stack: 0
input: 0,1,0,1 maximum size of the stack: 3
input: 0,1,1,1 maximum size of the stack: 29
input: 1,1,1,1 maximum size of the stack: 236
input: 1,1,1,1,1 maximum size of the stack: 1191
input: 1,1,1,1,1,1 maximum size of the stack: 1498

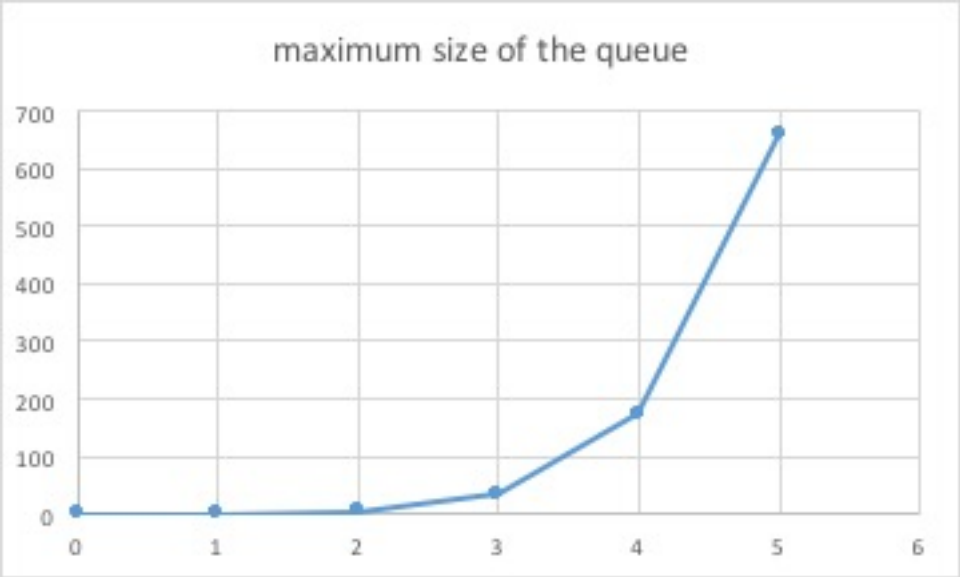
The maximum size of the stack changes a lot when the algorithm deal with different kinds of situation, which means that IDDLS is also not as more efficient in finding the goal state. However, IDDLS is guaranteed to find the optimal state and return a path.
Big-O analysis: O(b^m) where b is the maximum forward branch and m is the maximum path length m
We used recursion in this case
Graph: (the x-axis is the number of dirty rooms)



22.6: (didn't use visited state of nodes)

Description: there are n rooms in a line and some of them are dirty. There is a vacuum in one of the rooms and we want to use the A* search algorithm to clean all rooms.
Data structure: priority queue
Efficiency analysis:
input: 0,0,0,0 maximum size of the queue: 0
input: 0,1,0,1 maximum size of the queue: 1
input: 0,1,1,1 maximum size of the queue: 5
input: 1,1,1,1 maximum size of the queue: 34
input: 1,1,1,1,1 maximum size of the queue: 174
input: 1,1,1,1,1,1 maximum size of the queue: 660

With the increase number of the dirty rooms, the number of the expanded nodes also increases. In addition, if the number of rooms increases, then the number of expanded nodes also increases. However, it takes more time to find a smallest cost path.
Big-O analysis: O(b^m) where b is the maximum forward branch and m is the maximum path length m
Graph: (the x-axis is the number of dirty rooms)



23.2:

Description: there are m*n rooms in 2 dimension and some of them are dirty. There is a vacuum in one of the rooms and we want to use the BFS algorithm to clean all rooms.
Data structure: queue
Efficiency analysis:

input:
```
0,0,0
0,0,0
0,0,1
```
0,1 maximum size of the queue: 6

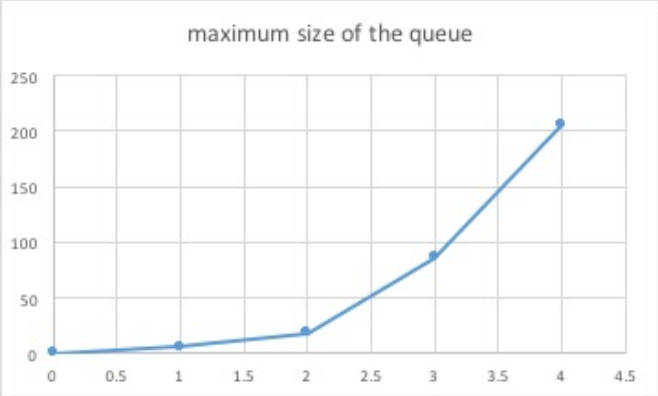
input:
```
0,0,0
1,0,0
0,1,0
```
2,1 maximum size of the queue: 10

input:
```
0,0,1
1,0,0
0,1,0
```
2,0 maximum size of the queue: 18

input:
```
0,1,1
1,0,0
0,1,0
```
2,0 maximum size of the queue: 24

input:
```
0,0,0,0
0,0,0,0
0,0,1,0
```
0,1 maximum size of the queue: 16

With the increase number of the dirty rooms, the number of the expanded nodes also increases. In addition, if the number of rooms increases, then the number of expanded nodes also increases.
Big-O analysis: O(b^m) where b is the maximum forward branch and m is the maximum path length m
Graph: (the x-axis is the number of dirty rooms)

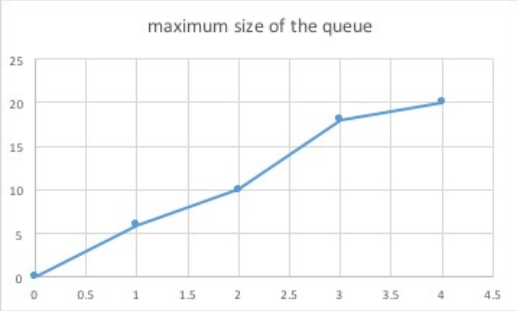


23.3:

Description: there are m*n rooms in 2 dimension and some of them are dirty. There is a vacuum in one of the rooms and we want to use the DFS algorithm to clean all rooms.
Data structure: stack
Efficiency analysis:
input:
```
0,0,0
0,0,0
0,0,1
0,1
```
maximum size of the stack: 6
input:
```
0,0,0
1,0,0
0,1,0
2,1
```
maximum size of the stack: 10
input:
```
0,0,1
1,0,0
0,1,0
2,0
```
maximum size of the stack: 18
input:
```
0,1,1
1,0,0
0,1,0
2,0
```
maximum size of the stack: 20
input:
```
0,0,0,0
```

```
0,0,0,0
0,0,1,0
0,1
```
maximum size of the stack: 11

With the increase number of the dirty rooms, the number of the expanded nodes also increases. In addition, if the number of rooms increases, then the number of expanded nodes also increases. However, compared to BFS, the maximum size of the stack doesn't change a lot when the algorithm deal with different kinds of situation, which means that DFS is more efficient in finding the goal state.
Big-O analysis: O(b^m) where b is the maximum forward branch and m is the maximum path length m
Graph: (the x-axis is the number of dirty rooms)



23.4: (use visited state of nodes)

Description: there are m*n rooms in 2 dimension and some of them are dirty. There is a vacuum in one of the rooms and we want to use the depth-limit search(DLS) algorithm to clean all rooms.
Data structure: stack
Efficiency analysis:
input:
```
0,0,0
0,0,0
0,0,1
0,1
```
maximum size of the stack: 6
input:
```
0,0,0
1,0,0
0,1,0
2,1
```
maximum size of the stack:
input:
```
0,0,1
1,0,0
0,1,0
2,0
```
maximum size of the stack: NONE
input:
```
0,1,1
1,0,0
0,1,0
```

2,0 maximum size of the stack: NONE
input:
0,0,0,0
0,0,0,0
0,0,1,0
0,1 maximum size of the stack: NONE

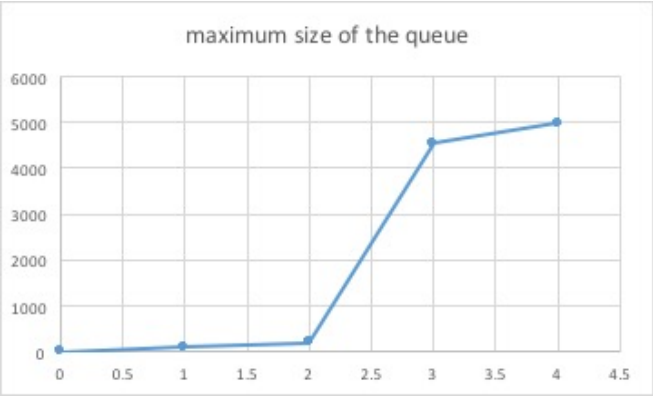
With the increase number of the dirty rooms, the number of the expanded nodes also increases. In addition, if the number of rooms increases, then the number of expanded nodes also increases. However, compared to BFS, the maximum size of the stack doesn't change a lot when the algorithm deal with different kinds of situation, which means that DFS is more efficient in finding the goal state.
We used recursion in this case.
Big-O analysis: O(b^m) where b is the maximum forward branch and m is the maximum path length m

23.5: (didn't use visited state of nodes)

Description: there are m*n rooms in 2 dimension and some of them are dirty. There is a vacuum in one of the rooms and we want to use the Iterative deepening depth-limit search(IDDLS) algorithm to clean all rooms.
Data structure: stack
Efficiency analysis:
input:
0,0,0
0,0,0
0,0,1
0,1 maximum size of the stack: 100
input:
0,0,0
1,0,0
0,1,0
2,1 maximum size of the stack: 197
input:
0,0,1
1,0,0
0,1,0
2,0 maximum size of the stack: 4540
input:
0,1,1
1,0,0
0,1,0
2,0 maximum size of the stack: 4975
input:
0,0,0,0
0,0,0,0
0,0,1,0

`0,1` maximum size of the stack: 100

The maximum size of the stack changes a lot when the algorithm deal with different kinds of situation, which means that IDDLS is also not as more efficient in finding the goal state. However, IDDLS is guaranteed to find the optimal state and return a path.
Big-O analysis: O(b^m) where b is the maximum forward branch and m is the maximum path length m
We used recursion in this case
Graph: (the x-axis is the number of dirty rooms)

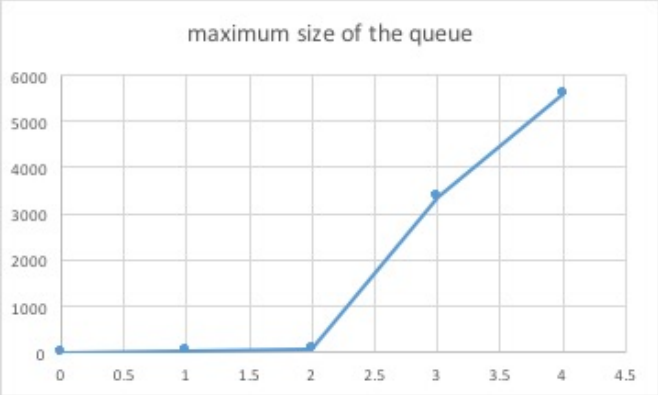


23.6: (use visited state of nodes)

Description: there are m*n rooms in 2 dimension and some of them are dirty. There is a vacuum in one of the rooms and we want to use the A* algorithm to clean all rooms.
Data structure: priority queue
Efficiency analysis:
input:
`0,0,0`
`0,0,0`
`0,0,1`
`0,1` maximum size of the queue: 43
input:
`0,0,0`
`1,0,0`
`0,1,0`
`2,0` maximum size of the queue: 84
input:
`0,0,1`
`1,0,0`
`0,1,0`
`2,0` maximum size of the queue: 3382
input:
`0,1,1`
`1,0,0`
`0,1,0`
`2,0` maximum size of the queue: 5592

input:
0,0,0,0
0,0,0,0
0,0,1,0
0,1 maximum size of the queue: 56

With the increase number of the dirty rooms, the number of the expanded nodes also increases. In addition, if the number of rooms increases, then the number of expanded nodes also increases. However, it takes more time to find a smallest cost path.
Big-O analysis: tutaO(b^m) where b is the maximum forward branch and m is the maximum path length m
We used recursion in this case
Graph: (the x-axis is the number of dirty rooms)



In conclusion:

From above analysis, we can know that BFS is complete and always return the shortest path to find a node, while DFS cannot deal with loops. However, DFS requires less memory space than BFS so it is a better choice when the goal node is known to be far from the root. DLS makes an improvement from DFS so that it does not run into infinite loops, but it is not guarantee to find a path if the goal state is too deep (actual depth > depth limit). The iterative deepening depth-first search is guaranteed to return the shortest path from root if it can find the goal state, but it may take more time to find such path. A* search can deal with graph with edge cost, and it takes the distance already traveled into account so that it usually finds a path with lower cost than greedy BFS.

Paragraph from group member Xinyi Ma:

I mainly contribute to problem21_2,3,4, problem22_1,2,3,4, problem23_1,2,3. I gain a deeper knowledge about DFS, BFS and DLS. I also learn more about python, for example, how to copy a list. If I simply use a=b to copy list b to a, then a and b still share the same id, and if I change one of them, the other one will also be changed. So I need to use a list() function to copy a list with a different id. Moreover, in order to copy an independent list of lists, I need to go through every single item in the list and copy it to a new list.

Paragraph from group member Yuanchi Ha:

In this assignment, I helped to write problem22_5, 23_1,2 and revised several other files. This assignment helps me to understand DFS, BFS, DLS, IDDLS, and heuristic search and their pros and cons better, and also reminds me again how nice python is.

Paragraph from group member Yijun Zhan:

I wrote problem 21_5, 6, 22_ 6, 23_5, 6, and debugged problem 22_4, 22_5, and 23_4. I found that using recursion for DFS is more intuitive and less error-prone than using loops. By dealing with multiple dimension arrays, I also learned about list compression and list slicing. They are both elegant language features which leads to easy manipulation of lists.