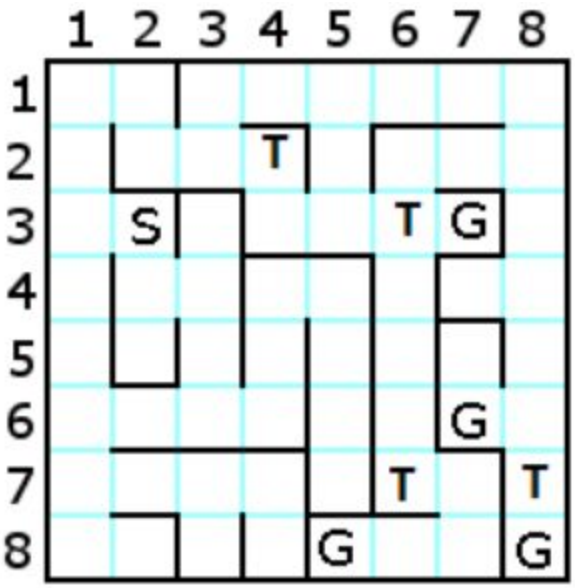
1. **Encoding the maze:**
   We started with encoding the example maze which is given in the homework document.



There are 4 types of cells in the maze, we used one lower case letter for encoding:

1. Normal cells, ‘n’
2. Start cell, ‘s’
3. Goal cells, ‘g’
4. Trap cells, ‘t’

There are 4 possible moves for each cell and we encoded these moves with True (T) and False (F), with this order: West, North, East, South. E.g Matrix[1][1] is encoded like nFFTT. So, our input file is:

```
1  nFFTT_nTFFT_nFFTT_nTFTF_nTFTT_nTFTF_nTFTF_nTFFT
2  nFTFT_nFTTF_nTTTF_tTFFT_nFTFT_nFFTT_nTFTF_nTTFT
3  nFTTT_sTFFT_nFFFT_nFTTF_nTTTF_tTTTT_gTFFF_nFTFT
4  nFTFT_nFTTT_nTTFT_nFFTT_nTFFT_nFTFT_nFFTF_nTTFT
5  nFTFT_nFTFF_nFTFT_nFTFT_nFTFT_nFTFT_nFFFT_nFTFT
6  nFTTT_nTFTF_nTTTF_nTTFF_nFTFT_nFTFT_gFTTF_nTTFT
7  nFTTT_nTFTF_nTFTT_nTFFT_nFTFF_tFTTF_nTFFT_tFTFT
8  nFTTF_nTFFF_nFTFF_nFTFF_gFFTF_nTFTF_nTTFF_gFTFF
```

2. **Creating a node class.**
   We created a class called ‘Node’ in order to hold information about a single cell.

```
class Node:
    def __init__(self, x, y, parent, cost, node_type, depth, heuristic_cost):
        self.x = x                                  # row
        self.y = y                                  # column
        self.parent = parent                        # parent of the node
        self.cost = cost                            # real cost from root to this node
        self.node_type = node_type                  # normal, trap, goal or start
        self.depth = depth
        self.heuristic_cost = heuristic_cost        # city block distance to the closest goal state
```

3. **Read the input file, locate the positions of goal states and the starting point.**
   We read the ‘input.txt’ and stored the information of each cell in a 2d array (list in python) called matrix. Then we located the start and goal states by looking the first letter of every element in matrix.
4. **Create the heuristic function.** (For A* and Greedy best first search)
   ‘heuristic func’ is a 2d array that has the same size as the matrix. We initialized the elements of the ‘heuristic func’ using city block distance to the closest goal state.

```
def create_heuristic_func(matrix, goals):
    number_of_rows, number_of_cols = len(matrix), len(matrix[0])
    heuristic_func = np.full((number_of_rows, number_of_cols), None)

    for i in range(number_of_rows):
        for j in range(number_of_cols):
            min_goal = number_of_rows + number_of_cols
            for goal in goals:
                cost = abs(goal[0]-i) + abs(goal[1]-j)
                if cost < min_goal:
                    min_goal = cost
            heuristic_func[i][j] = min_goal

    return heuristic_func
```

5. **Implementing general graph search.**
   We implemented the general graph search algorithm which is given in the textbook. The function takes following parameters as input:
   a. matrix: 2d array that represents the maze
   b. root: Starting node
   c. search_type: Defines the graph search method (BFS, DFS etc.)
   d. Iterative: max depth for the iterative deepening search

```
def graphSearch(matrix, root, search_type, iterative):
    frontier = []   # nodes to be expanded
    #state = [root.x, root.y]    # initial state

    expand_sequence = []    # list of the expanded nodes in order.
    expand_sequence.append(root)

    explored = []   # initialize the explored set to be empty
    explored.append((root.x, root.y))

    # initialize the frontier using initial state of the problem
    cell = matrix[root.x][root.y]
```

   We expanded the frontier using the matrix and starting point with the given order i in the homework document. (East, South, West, North) If the search method is DFS or IDP, we need the reversed order.

```
if search_type == 'DFS' or search_type == 'IDP':
    frontier = frontier[::-1]
```

   Then, we have an infinite loop that stops if frontier is empty or the goal is reached.

6. **Expand Frontier**
   This function expands the explored, expand sequence and the frontier.
   a. Adds the given node to the explored and the expand_sequence arrays.
   b. Determines the valid moves.
   c. Deletes the node in the frontier if and only if the node in the frontier is reachable from another node with less cost, and the search method is A* or Uniform Cost Search.
   d. If a move is valid, the function creates another node and appends to frontier.

7. **Print Solution**
   It just prints the results:
   a. Cost of the solution
   b. Path of the solution
   c. Expand sequence

Results of the search methods for the given maze:

```
============================================================
                        Search method: BFS
============================================================
The cost of the solution: 23
The solution path:  (3,2) -> (3,1) -> (2,1) -> (1,1) -> (1,2) -> (2,2) -> (2,3) -> (2,4) -> (3,4) -> (3,5) -> (3,6) -> (3,7)
The list of expanded nodes:  (3,2), (4,2), (3,1), (4,3), (5,2), (4,1), (2,1), (5,3), (3,3), (5,1), (1,1), (6,3), (6,1), (1,2),
 (6,4), (6,2), (7,1), (2,2), (5,4), (7,2), (8,1), (2,3), (4,4), (7,3), (8,2), (2,4), (1,3), (4,5), (7,4), (8,3), (3,4), (1,4),
 (5,5), (8,4), (3,5), (1,5), (6,5), (3,6), (2,5), (1,6), (7,5), (3,7)

============================================================
                        Search method: DFS
============================================================
The cost of the solution: 23
The solution path:  (3,2) -> (3,1) -> (2,1) -> (1,1) -> (1,2) -> (2,2) -> (2,3) -> (2,4) -> (3,4) -> (3,5) -> (3,6) -> (3,7)
The list of expanded nodes:  (3,2), (4,2), (4,3), (5,3), (6,3), (6,4), (5,4), (4,4), (4,5), (5,5), (6,5), (7,5), (6,2), (6,1),
 (7,1), (7,2), (7,3), (7,4), (8,4), (8,3), (8,1), (8,2), (5,1), (4,1), (3,3), (5,2), (3,1), (2,1), (1,1), (1,2), (2,2), (2,3),
 (2,4), (3,4), (3,5), (3,6), (3,7)

============================================================
                        Search method: UCS
============================================================
The cost of the solution: 18
The solution path:  (3,2) -> (3,1) -> (2,1) -> (1,1) -> (1,2) -> (2,2) -> (2,3) -> (1,3) -> (1,4) -> (1,5) -> (1,6) -> (1,7) -
> (1,8) -> (2,8) -> (3,8) -> (4,8) -> (5,8) -> (6,8) -> (6,7)
The list of expanded nodes:  (3,2), (4,2), (3,1), (4,3), (5,2), (4,1), (2,1), (5,3), (3,3), (5,1), (1,1), (6,3), (6,1), (1,2),
 (6,4), (6,2), (7,1), (2,2), (5,4), (7,2), (8,1), (2,3), (4,4), (7,3), (8,2), (1,3), (4,5), (7,4), (8,3), (1,4), (5,5), (8,4),
 (1,5), (6,5), (1,6), (2,5), (7,5), (1,7), (3,5), (1,8), (3,4), (2,4), (2,8), (3,8), (2,7), (4,8), (2,6), (5,8), (4,7), (6,8),
 (3,6), (6,7)

============================================================
                        Search method: IDP
============================================================
The cost of the solution: 23
The solution path:  (3,2) -> (3,1) -> (2,1) -> (1,1) -> (1,2) -> (2,2) -> (2,3) -> (2,4) -> (3,4) -> (3,5) -> (3,6) -> (3,7)
The list of expanded nodes:  (3,2), (4,2), (4,3), (5,3), (6,3), (6,4), (5,4), (4,4), (4,5), (5,5), (6,5), (7,5), (6,2), (6,1),
 (7,1), (7,2), (7,3), (7,4), (8,4), (8,3), (8,1), (8,2), (5,1), (4,1), (3,3), (5,2), (3,1), (2,1), (1,1), (1,2), (2,2), (2,3),
 (2,4), (3,4), (3,5), (3,6), (3,7)
Number of iterations:  11
============================================================
                        Search method: GBFS
============================================================
The cost of the solution: 23
The solution path:  (3,2) -> (3,1) -> (2,1) -> (1,1) -> (1,2) -> (2,2) -> (2,3) -> (2,4) -> (3,4) -> (3,5) -> (3,6) -> (3,7)
The list of expanded nodes:  (3,2), (4,2), (4,3), (3,3), (5,3), (6,3), (6,4), (5,4), (4,4), (4,5), (5,5), (6,5), (7,5), (6,2),
 (3,1), (5,2), (6,1), (7,1), (7,2), (7,3), (7,4), (8,4), (8,3), (8,1), (8,2), (4,1), (2,1), (5,1), (1,1), (1,2), (2,2), (2,3),
 (2,4), (3,4), (3,5), (3,6), (3,7)

============================================================
                        Search method: A*
============================================================
The cost of the solution: 18
The solution path:  (3,2) -> (3,1) -> (2,1) -> (1,1) -> (1,2) -> (2,2) -> (2,3) -> (1,3) -> (1,4) -> (1,5) -> (1,6) -> (1,7) -
> (1,8) -> (2,8) -> (3,8) -> (4,8) -> (5,8) -> (6,8) -> (6,7)
The list of expanded nodes:  (3,2), (4,2), (3,1), (4,3), (3,3), (5,2), (5,3), (6,3), (6,4), (4,1), (2,1), (6,2), (5,4), (5,1),
 (1,1), (4,4), (1,2), (4,5), (2,2), (2,3), (6,1), (5,5), (7,1), (6,5), (7,2), (8,1), (7,5), (7,3), (8,2), (7,4), (8,3), (8,4),
 (1,3), (1,4), (1,5), (1,6), (2,5), (1,7), (3,5), (1,8), (3,4), (2,8), (3,8), (2,7), (2,4), (4,8), (2,6), (4,7), (5,8), (6,8),
 (6,7)
```