Assignment 1: Search

Part 2:

In [8]:
```
from search_1_2 import *
from itertools import combinations, permutations
import numpy as np
import time

fname = 'src/tinySearch.txt'
# sname_dfs = fname[:-4] + '_sol_dfs' + fname[-4:]
sname_bfs_1 = fname[:4] + 'p1_2_sol/' + fname[4:-4] + '_sol_bfs_1' + fnam
sname_bfs_2 = fname[:4] + 'p1_2_sol/' + fname[4:-4] + '_sol_bfs_2' + fnam
sname_bfs_3 = fname[:4] + 'p1_2_sol/' + fname[4:-4] + '_sol_bfs_3' + fnam
sname_gbfs = fname[:4] + 'p1_2_sol/' + fname[4:-4] + '_sol_gbfs' + fname[
sname_astar_1 = fname[:4] + 'p1_2_sol/' + fname[4:-4] + '_sol_astar_1' +
sname_astar_2 = fname[:4] + 'p1_2_sol/' + fname[4:-4] + '_sol_astar_2' +
```

Breadth First Search (1)

In [9]:
```
maze_map, start, goal_dict, index_dict = parse_file(fname)

total_path = ''
total_expanded = 0


# print(goal_dict)
# print(index_dict)
timer_s = time.clock()

curr_node = start
while index_dict:
    dist = []
    for goal in index_dict:
        manhattan = get_manhattan(curr_node, goal)
        dist.append((manhattan, goal))
    if len(dist) == 1:
        print('DONE')
        break
    dist.sort()
#     print(dist)
```

```
    if len(dist) > 2:
        real_dist = []
        for i in range(1, len(dist) - 1):
            goal = dist[i][1]
            path, expanded = find_path_bfs(maze_map, start, goal)
            total_expanded += expanded
            bfs = len(path)
            real_dist.append((bfs, goal))
            if bfs > dist[i + 1][0]:
                break

        real_dist.sort()
#         print(real_dist)
        next_node = real_dist[0][1]
    else:
        next_node = dist[1][1]

    path, expanded = find_path_bfs(maze_map, curr_node, next_node)
    total_path += path

    print(curr_node, '--->>', next_node)
    curr_index = index_dict[curr_node]
    del index_dict[curr_node]
    del goal_dict[curr_index]
    curr_node = next_node

timer_t = time.clock()
print('bfs takes', timer_t - timer_s,
      'seconds, costs', len(total_path),
      'steps and expands', total_expanded, 'cells.')

maze_sol_bfs = draw_path(maze_map, total_path, start)
write_sol_to_file(sname_bfs_1, maze_sol_bfs)
```

```
(4, 4) --->> (5, 4)
(5, 4) --->> (4, 2)
(4, 2) --->> (5, 1)
(5, 1) --->> (7, 1)
(7, 1) --->> (2, 3)
(2, 3) --->> (1, 1)
(1, 1) --->> (3, 6)
(3, 6) --->> (3, 8)
(3, 8) --->> (1, 8)
(1, 8) --->> (5, 7)
(5, 7) --->> (6, 8)
(6, 8) --->> (7, 6)
DONE
bfs takes 0.00586099999999945 seconds, costs 45 steps and expands 527
```

cells.

Breadth First Search (2)

In [10]:
```
maze_map, start, goal_dict, index_dict = parse_file(fname)

total_path = ''
total_expanded = 0


# print(goal_dict)
# print(index_dict)
timer_s = time.clock()

length = len(goal_dict.keys())

t = Tree(length)
dist_dict = {}
for goal_i, goal_j in combinations(index_dict.keys(), r = 2):
    path, expanded = find_path_bfs(maze_map, goal_i, goal_j)
    total_expanded += expanded
    dist = len(path)
    dist_dict[(goal_i, goal_j)] = dist
    dist_dict[(goal_j, goal_i)] = dist
    i = index_dict[goal_i]
    j = index_dict[goal_j]
    t.addEdge(i, j, dist)
    # print(i, j)
mst = t.KruskalMST()
# print(mst)

curr_index = index_dict[start]
next_index = curr_index

while goal_dict.keys():

    length = len(goal_dict.keys())
    if length == 1:
        print('DONE')
        break

    edges = []
    for edge in mst:
        # print(curr_index)
        # print(edge)
        if edge[0] == curr_index or edge[1] == curr_index:
            edges.append(edge)
    for edge in edges:
```

```
        mst.remove(edge)

    if edges == []:
        dist_list = []
        curr_node = goal_dict[curr_index]
        for goal in index_dict.keys():
            if (curr_node, goal) in dist_dict.keys():
                dist_list.append((dist_dict[(curr_node, goal)], goal))
            else:
                path, expanded = find_path_bfs(maze_map, curr_node, goal)
                total_expanded += expanded
                dist = len(path)
                dist_list.append((dist, goal))
        dist_list.sort()
        if len(dist_list) == 1:
            print(index_dict)
            print(dist_list)
        next_node = dist_list[1][1]
    else:
        sorted(edges, key=lambda edge: edge[2])
        if edges[0][1] == curr_index:
            next_index = edges[0][0]
        else:
            next_index = edges[0][1]
        curr_node = goal_dict[curr_index]
        next_node = goal_dict[next_index]

    print(curr_node, '--->>', next_node)
    path, expanded = find_path_astar(maze_map, curr_node, next_node)

    total_path += path

    del goal_dict[curr_index]
    del index_dict[curr_node]
    curr_node = next_node
    curr_index = index_dict[curr_node]

timer_t = time.clock()
print('bfs takes', timer_t - timer_s,
      'seconds, costs', len(total_path),
      'steps and expands', total_expanded, 'cells.')

maze_sol_bfs = draw_path(maze_map, total_path, start)
write_sol_to_file(sname_bfs_2, maze_sol_bfs)
```

```
(4, 4) --->> (5, 4)
(5, 4) --->> (4, 2)
(4, 2) --->> (5, 1)
(5, 1) --->> (7, 1)
```

```
(7, 1) --->> (7, 6)
(7, 6) --->> (5, 7)
(5, 7) --->> (6, 8)
(6, 8) --->> (3, 8)
(3, 8) --->> (1, 8)
(1, 8) --->> (3, 6)
(3, 6) --->> (2, 3)
(2, 3) --->> (1, 1)
DONE
bfs takes 0.009282000000000679 seconds, costs 36 steps and expands 165
1 cells.
```

Breadth First Search (3)

In [11]:
```
maze_map, start, goal_dict, index_dict = parse_file(fname)

total_path = ''
total_expanded = 0


# print(goal_dict)
# print(index_dict)
timer_s = time.clock()
length = len(goal_dict)
dist_dict = {}
path_dict = {}

for goal_i, goal_j in combinations(index_dict.keys(), r = 2):
    path, expanded = find_path_bfs(maze_map, goal_i, goal_j)
    total_expanded += expanded
    dist = len(path)
    dist_dict[(goal_i, goal_j)] = dist
    dist_dict[(goal_j, goal_i)] = dist
    path_dict[(goal_i, goal_j)] = path

goals = list(index_dict.keys())
goals.remove(start)
# print(goals)

min_dist = 9999
min_path = []

for goal_order in permutations(goals):
    temp_dist = 0
    temp_path = [start]
    curr_node = start
    next_node = ()
    for goal in goal_order:
```

```
        next_node = goal
        dist = dist_dict[(curr_node, next_node)]
        temp_dist += dist
        temp_path.append(next_node)
        curr_node = next_node
    if temp_dist < min_dist:
#         print(temp_dist)
        min_dist = temp_dist
        min_path = temp_path
# print(min_path)

curr_node = start
next_node = ()
for i in range(1, len(min_path)):
    next_node = min_path[i]
    print(curr_node, '--->>', next_node)
    if (curr_node, next_node) in path_dict:
        path = path_dict[(curr_node, next_node)]
    else:
        path, _ = find_path_bfs(maze_map, curr_node, next_node)
#     print(path)
    total_path += path
    curr_node = next_node

timer_t = time.clock()
# print(total_path)
print('bfs takes', timer_t - timer_s,
      'seconds, costs', len(total_path),
      'steps and expands', total_expanded, 'cells.')

maze_sol_bfs = draw_path(maze_map, total_path, start)
write_sol_to_file(sname_bfs_3, maze_sol_bfs)
```

```
(4, 4) --->> (5, 4) :
(5, 4) --->> (2, 3) :
(2, 3) --->> (1, 1) :
(1, 1) --->> (4, 2) :
(4, 2) --->> (5, 1) :
(5, 1) --->> (7, 1) :
(7, 1) --->> (7, 6) :
(7, 6) --->> (3, 6) :
(3, 6) --->> (5, 7) :
(5, 7) --->> (6, 8) :
(6, 8) --->> (3, 8) :
(3, 8) --->> (1, 8) :
bfs takes 1977.022998 seconds, costs 35 steps and expands 1651 cells.
```

Greedy Best First Search (1)

In [12]:
```
maze_map, start, goal_dict, index_dict = parse_file(fname)

total_path = ''
total_expanded = 0


# print(goal_dict)
print(index_dict)
timer_s = time.clock()

length = len(goal_dict.keys())

t = Tree(length)
dist_dict = {}
for goal_i, goal_j in combinations(index_dict.keys(), r = 2):
    path, expanded = find_path_gbfs(maze_map, goal_i, goal_j)
    total_expanded += expanded
    dist = len(path)
    dist_dict[(goal_i, goal_j)] = dist
    dist_dict[(goal_j, goal_i)] = dist
    i = index_dict[goal_i]
    j = index_dict[goal_j]
    t.addEdge(i, j, dist)
    # print(i, j)
mst = t.KruskalMST()
# print(mst)

curr_index = index_dict[start]
next_index = curr_index

while goal_dict.keys():
    length = len(goal_dict.keys())
    if length == 1:
        print('DONE')
        break

    edges = []
    for edge in mst:
        # print(curr_index)
        # print(edge)
        if edge[0] == curr_index or edge[1] == curr_index:
            edges.append(edge)
    for edge in edges:
        mst.remove(edge)

    if edges == []:
```

```
        dist_list = []
        curr_node = goal_dict[curr_index]
        for goal in index_dict.keys():
            if (curr_node, goal) in dist_dict.keys():
                dist_list.append((dist_dict[(curr_node, goal)], goal))
            else:
                path, expanded = find_path_gbfs(maze_map, curr_node, goal
                total_expanded += expanded
                dist = len(path)
                dist_list.append((dist, goal))
        dist_list.sort()
        if len(dist_list) == 1:
            print(index_dict)
            print(dist_list)
        next_node = dist_list[1][1]
    else:
        sorted(edges, key=lambda edge: edge[2])
        if edges[0][1] == curr_index:
            next_index = edges[0][0]
        else:
            next_index = edges[0][1]
        curr_node = goal_dict[curr_index]
        next_node = goal_dict[next_index]

    print(curr_node, '--->>', next_node)
    path, expanded = find_path_astar(maze_map, curr_node, next_node)

    total_path += path

    del goal_dict[curr_index]
    del index_dict[curr_node]
    curr_node = next_node
    curr_index = index_dict[curr_node]

timer_t = time.clock()
print('gbfs takes', timer_t - timer_s,
      'seconds, costs', len(total_path),
      'steps and expands', total_expanded, 'cells.')

maze_sol_gbfs = draw_path(maze_map, total_path, start)
write_sol_to_file(sname_gbfs, maze_sol_gbfs)
```

```
{(1, 1): 0, (1, 8): 1, (2, 3): 2, (3, 6): 3, (3, 8): 4, (4, 2): 5, (4,
4): 6, (5, 1): 7, (5, 4): 8, (5, 7): 9, (6, 8): 10, (7, 1): 11, (7, 6)
: 12}
(4, 4) --->> (5, 4)
(5, 4) --->> (4, 2)
(4, 2) --->> (5, 1)
(5, 1) --->> (7, 1)
```

```
(7, 1) --->> (7, 6)
(7, 6) --->> (5, 7)
(5, 7) --->> (6, 8)
(6, 8) --->> (3, 8)
(3, 8) --->> (1, 8)
(1, 8) --->> (3, 6)
(3, 6) --->> (2, 3)
(2, 3) --->> (1, 1)
DONE
gbfs takes 0.010766999999987092 seconds, costs 36 steps and expands 49
2 cells.
```

A* Search (1)

In [13]:
```
maze_map, start, goal_dict, index_dict = parse_file(fname)

total_path = ''
total_expanded = 0


# print(goal_dict)
# print(index_dict)
timer_s = time.clock()

curr_node = start
while index_dict:
    dist = []
    for goal in index_dict:
        manhattan = get_manhattan(curr_node, goal)
        dist.append((manhattan, goal))
    if len(dist) == 1:
        print('DONE')
        break
    dist.sort()
#     print(dist)

    if len(dist) > 2:
        real_dist = []
        for i in range(1, len(dist) - 1):
            goal = dist[i][1]
            path, expanded = find_path_astar(maze_map, start, goal)
            total_expanded += expanded
            bfs = len(path)
            real_dist.append((bfs, goal))
            if bfs > dist[i + 1][0]:
                break

        real_dist.sort()
```

```
#         print(real_dist)
        next_node = real_dist[0][1]
    else:
        next_node = dist[1][1]

    path, expanded = find_path_astar(maze_map, curr_node, next_node)
    total_path += path

    print(curr_node, '--->>', next_node)
    curr_index = index_dict[curr_node]
    del index_dict[curr_node]
    del goal_dict[curr_index]
    curr_node = next_node

timer_t = time.clock()
print('astar takes', timer_t - timer_s,
      'seconds, costs', len(total_path),
      'steps and expands', total_expanded, 'cells.')

maze_sol_astar = draw_path(maze_map, total_path, start)
write_sol_to_file(sname_astar_1, maze_sol_astar)
```

```
(4, 4) --->> (5, 4)
(5, 4) --->> (4, 2)
(4, 2) --->> (5, 1)
(5, 1) --->> (7, 1)
(7, 1) --->> (2, 3)
(2, 3) --->> (1, 1)
(1, 1) --->> (3, 6)
(3, 6) --->> (3, 8)
(3, 8) --->> (1, 8)
(1, 8) --->> (5, 7)
(5, 7) --->> (6, 8)
(6, 8) --->> (7, 6)
DONE
astar takes 0.008642999999892709 seconds, costs 45 steps and expands 1
27 cells.
```

A* Search (2)

In [14]:
```
maze_map, start, goal_dict, index_dict = parse_file(fname)

total_path = ''
total_expanded = 0


# print(goal_dict)
# print(index_dict)
```

```
# print(index_dict)
timer_s = time.clock()

length = len(goal_dict.keys())

t = Tree(length)
dist_dict = {}
for goal_i, goal_j in combinations(index_dict.keys(), r = 2):
    path, expanded = find_path_astar(maze_map, goal_i, goal_j)
    total_expanded += expanded
    dist = len(path)
    dist_dict[(goal_i, goal_j)] = dist
    dist_dict[(goal_j, goal_i)] = dist
    i = index_dict[goal_i]
    j = index_dict[goal_j]
    t.addEdge(i, j, dist)
    # print(i, j)
mst = t.KruskalMST()
print(mst)

curr_index = index_dict[start]
next_index = curr_index

while goal_dict.keys():
    length = len(goal_dict.keys())
    if length == 1:
        print('DONE')
        break

    edges = []
    for edge in mst:
        # print(curr_index)
        # print(edge)
        if edge[0] == curr_index or edge[1] == curr_index:
            edges.append(edge)
    for edge in edges:
        mst.remove(edge)

    if edges == []:
        dist_list = []
        curr_node = goal_dict[curr_index]
        for goal in index_dict.keys():
            if (curr_node, goal) in dist_dict.keys():
                dist_list.append((dist_dict[(curr_node, goal)], goal))
            else:
                path, expanded = find_path_astar(maze_map, curr_node, goa
                total_expanded += expanded
                dist = len(path)
                dist_list.append((dist, goal))
        dist_list.sort()
        if len(dist list) == 1:
```

```
            print(index_dict)
            print(dist_list)
        next_node = dist_list[1][1]
    else:
        sorted(edges, key=lambda edge: edge[2])
        if edges[0][1] == curr_index:
            next_index = edges[0][0]
        else:
            next_index = edges[0][1]
        curr_node = goal_dict[curr_index]
        next_node = goal_dict[next_index]

    print(curr_node, '--->>', next_node)
    path, expanded = find_path_astar(maze_map, curr_node, next_node)

    total_path += path

    del goal_dict[curr_index]
    del index_dict[curr_node]
    curr_node = next_node
    curr_index = index_dict[curr_node]

timer_t = time.clock()
print('astar takes', timer_t - timer_s,
      'seconds, costs', len(total_path),
      'steps and expands', total_expanded, 'cells.')

maze_sol_astar = draw_path(maze_map, total_path, start)
write_sol_to_file(sname_astar_2, maze_sol_astar)
```

```
[[6, 8, 1], [1, 4, 2], [5, 7, 2], [7, 11, 2], [9, 10, 2], [0, 2, 3], [
2, 6, 3], [3, 6, 3], [3, 9, 3], [4, 9, 3], [5, 8, 3], [9, 12, 3]]
(4, 4) --->> (5, 4)
(5, 4) --->> (4, 2)
(4, 2) --->> (5, 1)
(5, 1) --->> (7, 1)
(7, 1) --->> (7, 6)
(7, 6) --->> (5, 7)
(5, 7) --->> (6, 8)
(6, 8) --->> (3, 8)
(3, 8) --->> (1, 8)
(1, 8) --->> (3, 6)
(3, 6) --->> (2, 3)
(2, 3) --->> (1, 1)
DONE
astar takes 0.009970999999950436 seconds, costs 36 steps and expands 5
65 cells.
```

In [ ]:

In [ ]:

In [ ]:

In [ ]: