Exercice Artificial Intelligence Assignment 1

1 Theory

What is the runtime complexity of

- the “insert” operation at the end of a single linked list?

  O(1); List keeps pointer to the last node (=tail). Sets next of tail node to new node. Sets the new node as tail node.

- the “insert” operation at the end of a list backed by an array?

  O(1) amortized, but O(n) worst-case since the array must be resized and copied. (see https://en.wikipedia.org/wiki/Amortized_analysis#Dynamic_Array)

- the “insert” operation of a hash table (hash set)?

  O(1) in average, but O(n) in worst-case (Handling of collisions)

- the “contains” operation of a list?

  O(n), must iterate through list
  (see http://docs.oracle.com/javase/6/docs/api/java/util/ArrayList.html: *All of the other operations run in linear time*)

- the “contains” operation of a hash table (hash set)?

  O(1) in average, but O(n) in worst-case (Handling of collisions)

Which data structure should we actually choose to implement a “closed list” as it is often called in the literature?

The closed list is used to insert visited nodes, and check if a node is already in the list (contains). Hence the `insert` and `contains` operation must be fast. The **hash table** offers a constant time complexity on both of these operations and should therefore be chosen to implement a “closed list” (as it is often called in the literature).

3 Heuristic Search

c) Which of the heuristics guarantees that Greedy Best-First Search will lead to an optimal solution?

None. GBFS doesn’t always find the optimal solution. Both heuristics generate a solution with the same amount of steps. The euclidean heuristics will find a solution which looks more like a direct line, but is more turns (zig-zag). The Manhattan heuristics generates a solutions with more “straight lines”.

Which of them guarantees obtaining an optimal solution using A* Search?

Both Heuristics are admissible (= they never overestimate the actual cost from a given node to a goal), therefore the A* search is always optimal, the (topological) differences between euclidean and Manhattan heuristics are the same as above.

d) Which of the heuristics is better?

Manhattan may find its way faster. But all together there are no big benefits or disadvantages why using one ore the other. Both Heuristics ignore the actual layout of the given level, i.e. which path you can walk on, and where is a wall. So it really depends on the level, which Heuristic offers the better solution.

4 Create a Level

a)

BFS and IDS will always deliver the same results. They work basically the same way, only that BFS uses (much) more space and IDS does part of its work redundantly.

b)

BFS (order: ^, v, <, >)

1. (1, 2) *(dequeue)*
    - ^ (1, 1) = 1 *(enqueue)*
    - v *not possible*
    - < *not possible*
    - > (2, 2) = 1
2. (1, 1)
    - ^ *not possible*
    - v *already visited*
    - < *not possible*
    - > (2, 1) = 2
3. (2, 2)
    - ^ *already visited*
    - v *not possible*
    - < *already visited*
    - > (3, 2) = 2 *solution*

IDS

- D = 0
    - (1, 2)
    - *no solution*
- D = 1
    - (1, 2)
        - ^ (1, 1)
        - > (2, 2)
    - *no solution*
- D = 2
    - (1, 2)
        - ^ (1, 1)
            - v (1, 2) *already visited*
            - > (2, 1)
        - > (2, 2)
            - ^ (2, 1) *already visited*
            - < (1, 2) *already visited*
            - > **(3, 2)**
    - *solution*

DLDFS

- (1, 2)
    - ^ (1, 1)
        - ^ *not possible*
        - v *already visited*
        - < *not possible*
        - > (2, 1)
            - ^ *not possible*
            - v (2, 2)
                - ^ *already visited*
                - *v not possible*
                - *< already visited*
                - > **(3, 2)** *solution*

UCS

1. (1, 2) = 0
    - ^ (1, 1) = 0 + 1 = 1
    - > (2, 2) = 0 + 5 = 5
2. (1, 1) = 1
    - > (2, 1) = 1 + 2 = 3
3. (2, 1) = 3
    - > (3, 1) = 3 + 1 = 4
4. (3, 1) = 4
    - v (3, 2) = 4 *solution*

GBFS (Euclidean)

1. (1, 2) = 2
   - ^ (1, 1) = 2,24…
   - > (2, 2) = 1
2. (2, 2)
   - ^ (2, 1) = 1,41…
   - < *already visited*
   - > (3, 1) = 0 *solution*

ASTAR (Euclidean)

1. (1, 2) = 0 *(cost)*
   - ^ (1, 1) = 2,24… + (0 + 1) = 3,24…
   - > (2, 2) = 1 + (0 + 5) = 6
2. (1, 1) = 1
   - v *already visited*
   - > (2, 1) = 1,41… + (1 + 2) = 4,41
3. (2, 1) = 3
   - v (2, 2) = 1 + (3 + 5) = 9
   - > (3, 1) = 1 + (3 + 1) = 5
4. (3, 1) = 4
   - v (3, 2) = 0 + (5 + 0) = 5 *solution*