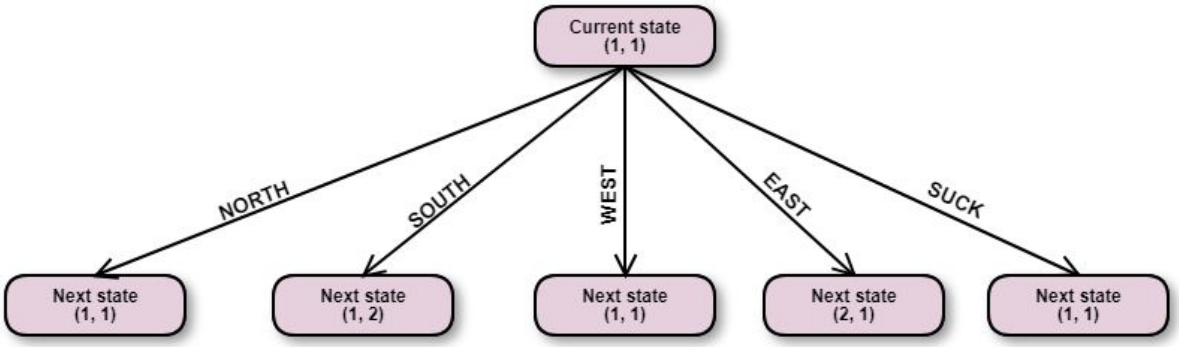
Lab 2 : Search

Part 2 : Theory

1. In the vacuum cleaner domain (Part 1), a state represents a location, by coordinates x and y. The domain has a size of 50x50, so 2500 possibles states in this domain. There are five differents actions in this domain :
   - Go North (UP)
   - Go South (DOWN)
   - Go West (LEFT)
   - Go East (RIGHT)
   - Suck dirt

   The branching factor is equals to the number of possibles actions. Example :



   The next state of the result action North and West are the same because of the walls.

2. In the case that every actions have the same cost, BFS and Uniform-Cost Search are quite similar :
   (Let *b* the branching factor and *d* the depth of the shallowest solution)
   BFS
   - BFS is complete as *b* is finite.
   - Not necessarily optimal, but here, it's optimal because all of the actions have a cost of 1.
   - Time and space complexity : $O(b^d)$

   Uniform-Cost Search
   - Complete
   - Optimal in general
   - Time and space complexity : $O(b^{d+1})$

   BFS is optimal because it always expands the shallowest unexpanded node.
   Uniform-Cost Search is similar to BFS, except that the latter stops as soon as it generates a goal, whereas Uniform-Cost Search examines all the nodes at the goal's depth to see if one has a lower cost. Thus Uniform-Cost Search does strictly more work by expanding nodes at depth *d* unnecessarily.

3. Let h_1 and h_2 admissible heuristics. So they don't overestimates the cost to reach the goal.
   a. $\frac{h_1+h_2}{2}$:
      Let C the real cost to reach the goal.
      As h_1 and h_2 admissible heuristics, then :
      $h_1 \geqslant C$
      $h_2 \geqslant C$
      $h_1 + h_2 \geqslant C + h_2 \geqslant 2C$
      $\frac{h_1+h_2}{2} \geqslant C$
      So the heuristic is **admissible**.
      Indeed, the sum of *n* admissible heuristics can possibly be inadmissible (overestimates the cost). But if the result is divided by *n* (the number of heuristics), the result becomes admissible.

   b. $2.h_1$: The result of the multiplication of an admissible heuristic by a coefficient greater than 1 could result to an **inadmissible** heuristic. Indeed, the cost could be overestimated.

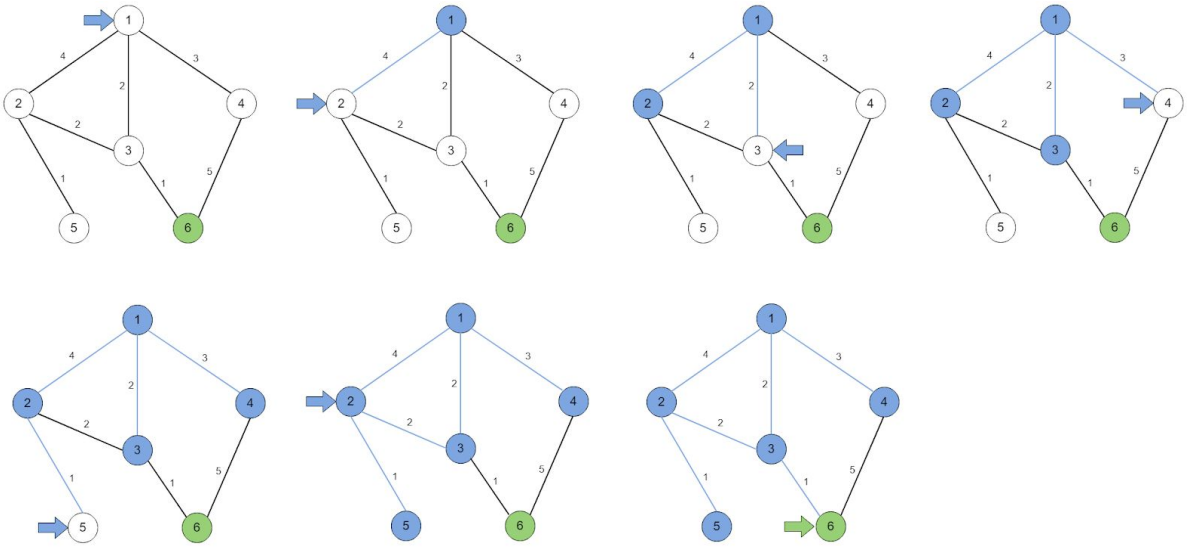
   c. $max(h_1, h_2)$: The max of two heuristics is never greater than the two of the heuristics, so the result will be an **admissible** heuristic. Moreover, this technique to determine a good heuristic from a set of heuristics.

4. $h(n)$estimates the cost to get from the node *n* to the goal. $h(n)$could be the euclidean distance between the node *n* and the goal (assuming there are no walls).

   $g(n)$estimates the cost to reach the node *n*. $g(n)$could be the euclidean distance between the start node and the node n (as the$h(n)$heuristic).

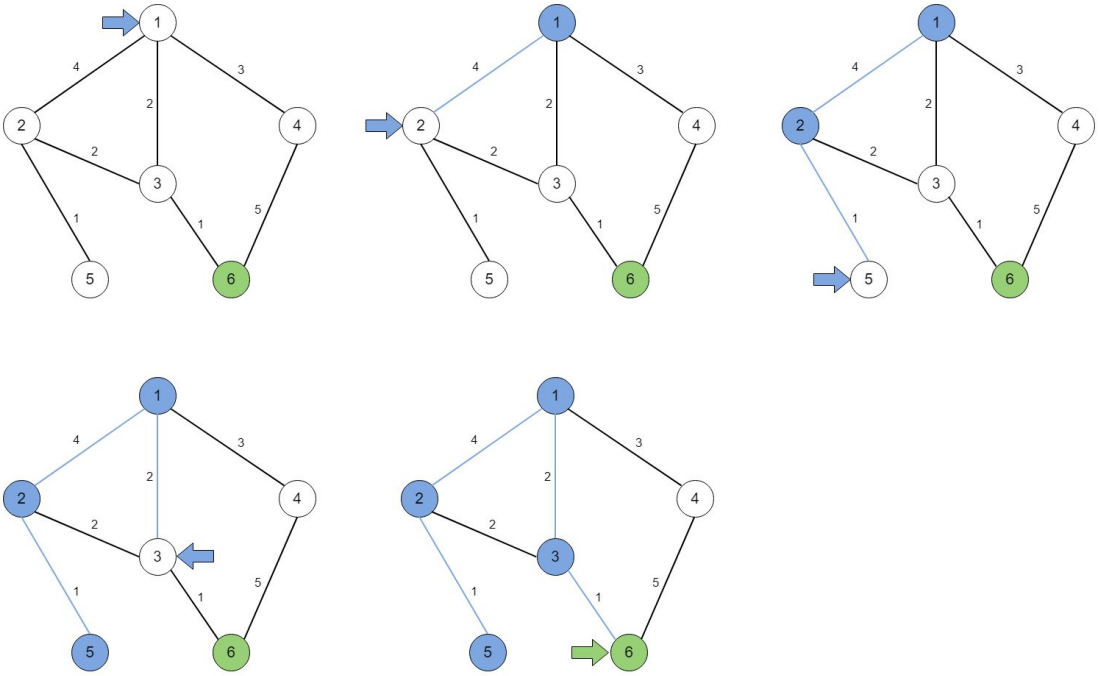
   $f(n) = g(n) + h(n)$ is clearly an **admissible** heuristic, it's an estimated cost of the cheapest solution through *n*.

5. **Breadth-first search**



BFS strategy expands current node's successors first and then continues with their successors and so on, meaning that we are looking for the shallowest node. In simple words we could say, that it searches for the goal through levels one by one. To maintain the correct sequence of the successors waiting to be visited, this strategy uses queue data structure (First In First Out). Whenever a node is expanded its children are pushed into queue. When the queue is empty, it means that the searching process is finished.

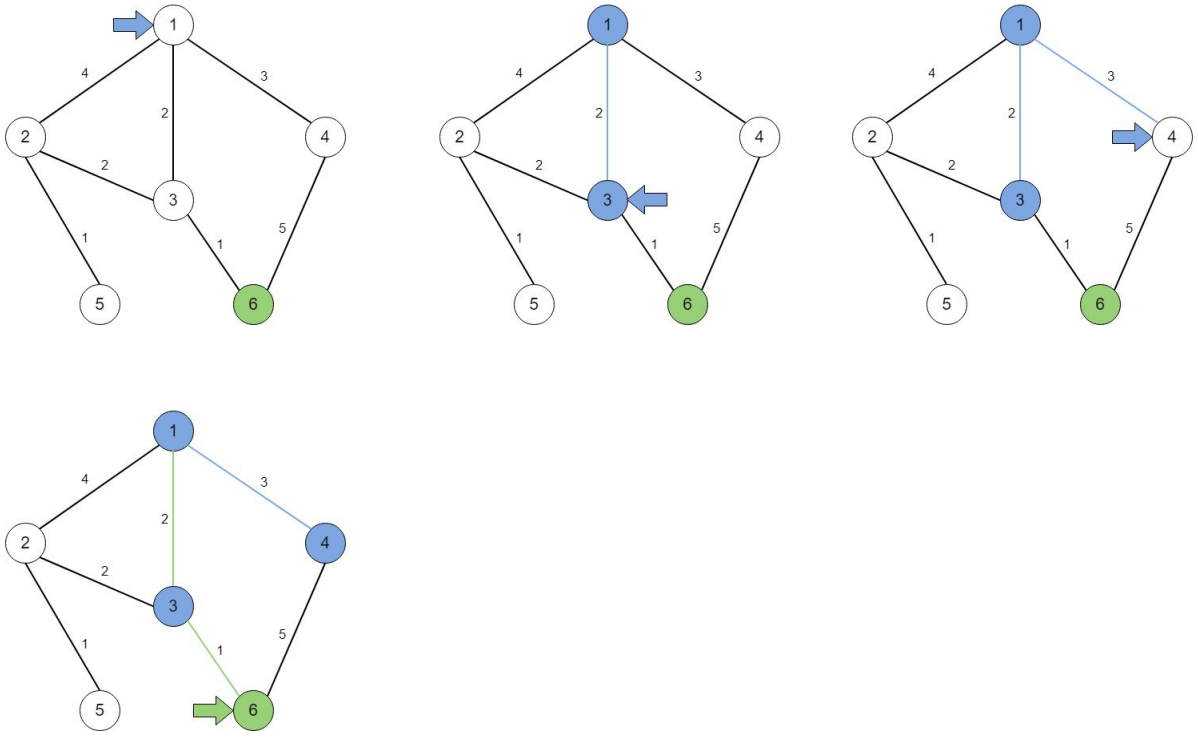
Depth-first search



In DFS the search is processed through branches one by one, meaning that we are looking for the deepest node. This approach explore next nodes of a certain branch

until all nodes, which don't have successors have been expanded. When a node is being visited it's pushed onto stack (Last In First Out) and then the search moves to its successor and so on. When we reach the node with no unvisited successors, then the we pop out the node from the stack and go back to previous one until our helper structure is empty.

Uniform-cost search
Problem: Find path to node 6



Uniform-cost search strategy is an extension of BFS, where instead of choosing the shallowest node the algorithm expands the node with the lowest path cost. So the helper structure in this case will be priority queue. We sort successors of current node by path cost before pushing them into queue. Beside that property, in every step the algorithm checks if the node that has just been added to the queue would be more beneficial as next expanded node. If that is correct, then the nodes in queue are switched in order to get the most optimal solution.

6. Uninformed search strategies:
    a. *Breadth-first search*
    Complete: if the shallowest goal node is at some finite depth *d* in the graph and assuming that branching factor *b* is also finite, then BFS will eventually find it.
    Optimal: provided the path cost is a nondecreasing function of the depth of the node, BFS is considered to be optimal search algorithm.

b. *Uniform-cost search*
   Complete: completeness is provided the cost of every step exceeds some small positive constant ε. In theoretical case where all steps are zero-cost actions the algorithm would get stuck in an infinite loop.
   Optimal: step cost is greater than or equal to zero, that's why paths never get shorter when new nodes are added. It implies that the nodes are expanded in order of their optimal path.

c. *Depth-first search*
   Complete: DFS is complete for graph-search version in finite state spaces, because it will expand every node eventually.
   Incomplete: if we are using tree-search version or infinite state spaces then the search strategy will get struck in an infinite loop (infinite path problem).
   Nonoptimal: because in this approach we are expanding whole branches and not levels, then if our goal node is placed in another branch than the one currently searched it will take time to get to it. Also if we have more than one goal node than we cannot be sure that the output is the best solution.

d. *Depth-limited search*
   Incomplete: if we choose a predetermined depth limit $l < d$, then the shallowest goal is beyond the depth limit.
   Nonoptimal: this search strategy becomes nonoptimal in case where $l > d$.

e. *Iterative deepening DFS*
   Complete & Optimal: because this strategy is a combination of BFS and DFS it's completeness and optimality is guaranteed if the same conditions are fulfilled.

f. *Bidirectional search*
   This approach depends on search strategies applied in simultaneous searches. In most cases BFS is applied for that purpose, that's why Bidirectional search is considered to be complete and optimal search strategy.

Informed (heuristic) search strategies:

A. *Greedy best-first search*
   Incomplete: in finite spaces the algorithm is complete, but only in the graph version. Otherwise, because of its mechanism, which chooses the best possible solution at the moment, it might get caught into an infinite loop.
   Nonoptimal: in every step greedy search expands node, which seems to be the most optimal at this moment - it tries to get as close to the goal as it can. However this approach does not always give the best results.

B. *A* search*

Conditions for optimality:
The A* search is optimal if the cost to get from the node to the goal *h(n)* is an admissible heuristic - it never overestimates this cost. This implies that the estimated cost of the cheapest solution through *n* node *f(n)*, where
f(n) = h(n)+g(n), shall never overestimate the true cost of a solution along the current path through *n*.

Conditions for completeness:
All steps costs exceed some finite ε and if branching factor *b* is finite.

7. In task #2 from first laboratory the most suited approach to solve the problem would be Depth-first search. Our goal was to visit every possible position in the presented world to be sure that all dirt was removed. From perspective of searching strategy we were supposed to traverse the graph and make use of already created paths (branches) in order to avoid obstacles in form of walls and already visited nodes on agent's way.