The goal node is denoted by `node_goal` and the source node is denoted by `node_start`.

We maintain two lists: `OPEN` and `CLOSE`:

- **OPEN** consists of nodes that have been visited but not expanded (meaning that successors have not been explored yet). This is the list of pending tasks.

- **CLOSE** consists of nodes that have been visited and expanded (successors have been explored already and included in the open list, if this was the case).

```plaintext
1. Put `node_start` in the OPEN list with \( f(node_start) = h(node_start) \) (initialization)
2. while the OPEN list is not empty {
   3. Take from the open list the node `node_current` with the lowest \( f(node_current) = g(node_current) + h(node_current) \)
   4. if \( node_current \) is `node_goal` we have found the solution; break
   5. Generate each state `node_successor` that come after `node_current`
      for each `node_successor` of `node_current`
         6. Set `successor_current_cost = g(node_current) + w(node_current, node_successor)`
         7. if `node_successor` is in the OPEN list {
             8. if \( g(node_successor) \leq \) `successor_current_cost` continue (to line 20)
         } else if `node_successor` is in the CLOSED list {
             9. if \( g(node_successor) \leq \) `successor_current_cost` continue (to line 20)
             10. Move `node_successor` from the CLOSED list to the OPEN list
         } else {
             11. Add `node_successor` to the OPEN list
             12. Set \( h(node_successor) \) to be the heuristic distance to `node_goal`
         }
         13. Set \( g(node_successor) = \) `successor_current_cost`
         14. Set the parent of `node_successor` to `node_current`
      }
   15. Add `node_current` to the CLOSED list
   16. if(`node_current` != `node_goal`) exit with error (the OPEN list is empty)
```