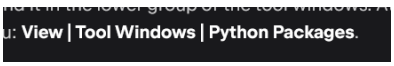
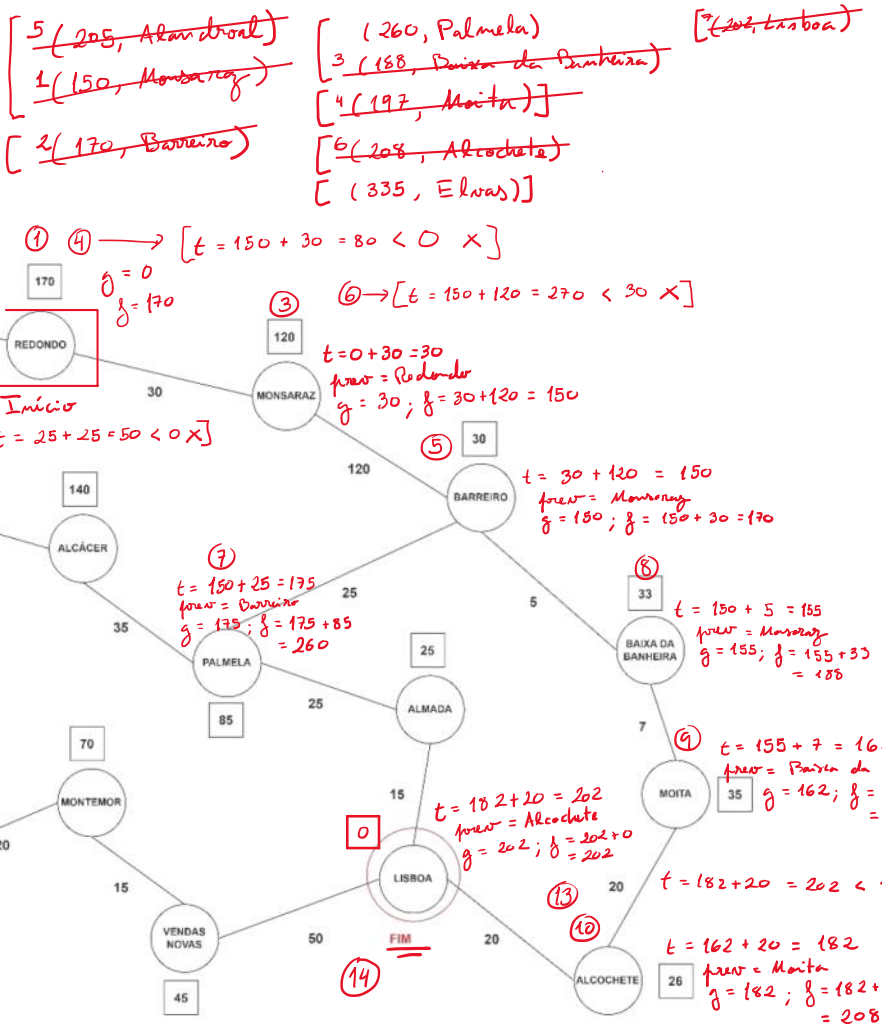
Aula PL #04

15 de outubro de 2023 14:29

Para instalar módulos externos: (pycharm)






```
# The set of discovered nodes that may need to be (re-)expanded.
# Initially, only the start node is known.
# This is usually implemented as a min-heap or priority queue rather than a hash-set.
open_set = [(0, start)]

# For node n, came_from[n] is the node immediately preceding it on the cheapest path from start to n
# currently known.
came_from = {}

# For node n, g_score[n] is the cost of the *cheapest path from start to n* currently known.
g_score = {node.get_name(): float("inf") for node in self.m_nodes}
g_score[start] = 0

# For node n, f_score[n] := g_score[n] + h(n). f_score[n] represents our current best guess as to
# how cheap a path could be from start to finish if it goes through n.
f_score = {node.get_name(): float("inf") for node in self.m_nodes}
f_score[start] = g_score[start] + self.get_h(start)  # <=> 0 + self.get_h(start)
```

Caminho de menor custo do início a outro nodo
Dicionário cuja chave é o nome, e o valor é o tal custo

```
while open_set:
    current = heapq.heappop(open_set)[1]
    if current == end:
        path = reconstruct_path(came_from, current)
        return path, g_score[current]

    for (neighbor, weight) in self.m_graph[current]:
        # d(current,neighbor) is the weight of the edge from current to neighbor
        # tentative_g_score is the distance from start to the neighbor through current

        tentative_g_score = g_score[current] + weight

        if tentative_g_score < g_score[neighbor]:
            # This path to neighbor is better than any previous one
            came_from[neighbor] = current
            g_score[neighbor] = tentative_g_score
            f_score[neighbor] = g_score[neighbor] + self.get_h(neighbor)
            if neighbor not in open_set:
                heapq.heappush(open_set, _item: (f_score[neighbor], neighbor))
```

Caminho de menor custo do início a um nodo, com a adição do valor da heurística nesse nodo, auxiliando assim uma escolha informada.
O quão curto poderá ser o caminho se se tomar uma determinada escolha; decisão informada com a heurística.

Dijkstra's Algorithm

```python
def greedy(self, start, end):

    if start not in self.m_graph.keys() or end not in self.m_graph.keys():
        return None

    open_set = [(0, start)]
    visited = set()
    came_from = {}
    score = {node.get_name(): float("inf") for node in self.m_nodes}

    while open_set:
        (current_score, current_node) = heapq.heappop(open_set)

        if current_node == end:
            path = reconstruct_path(came_from, current_node)
            return path, score[current_node]

        if current_node not in visited:
            visited.add(current_node)
            for (neighbour, weight) in self.m_graph[current_node]:
                if neighbour not in visited:
                    tentative_score = current_score + weight
                    if tentative_score < score[neighbour]:
                        score[neighbour] = tentative_score
                        came_from[neighbour] = current_node
                        heapq.heappush( *args: open_set, (score[neighbour], neighbour))

    return None
```