Algorithms Exercise Sheet: Dijkstra's and A* Algorithms

Instructions

Answer the following questions to deepen your understanding of graph search algorithms.

Exercises

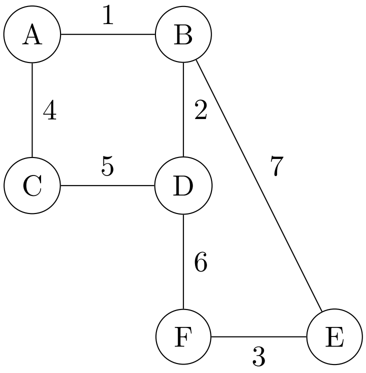
1. **Dijkstra's Algorithm:** The following pseudocode describes Dijkstra's algorithm for finding the shortest paths from a source node *s* to all other nodes in a weighted graph.

```
Algorithm Dijkstra(Graph, s):
Initialize distances d[v] = \infty for all nodes v
d[s] = 0
PriorityQueue Q = {all nodes in Graph}
while Q is not empty:
    u = node in Q with smallest d[u]
    remove u from Q
    for each neighbor v of u:
        if d[u] + weight(u, v) < d[v]:
            d[v] = d[u] + weight(u, v)
return d
```

2. **A* Algorithm:** The following pseudocode describes the A* algorithm as a generalization of Dijkstra's algorithm (with a goal node specified).

```
Algorithm A*(Graph, s, g):
Initialize distances d[v] = \infty for all nodes v
d[s] = 0
PriorityQueue Q = {all nodes in Graph}
while Q is not empty:
    u = node in Q with smallest d[u] + h(u)
    remove u from Q
    if u == g:
        return d[g]
    for each neighbor v of u:
        if d[u] + weight(u, v) < d[v]:
            d[v] = d[u] + weight(u, v)
return d
```

3. **Dijkstra's Algorithm on Example Graph:** The graph below represents a weighted graph with six nodes labeled A through F. Use Dijkstra's algorithm to compute the shortest-path tree starting from node A. Provide the shortest path and its cost to each other node.



4. **A* Algorithm on Example Graph:** Use the same graph as in question 3. Assume the heuristic $h(u)$ is defined as follows for the goal node F:

$$\begin{gathered} h(A) = 10, \quad h(B) = 8, \quad h(C) = 6, \\ h(D) = 4, \quad h(E) = 2, \quad h(F) = 0. \end{gathered}$$

Trace the execution of the A* algorithm to find the shortest path from A to F. Show the priority queue at each step.

5. **Graphs on which A* isn't very helpful:** Sometimes Dijkstra and A* search portions of the graph that are "obviously" not going to lead to the goal (although the obviousness can be hard to quantify). Write a couple graphs where A* explores many many more nodes than it actually "needs" to. Is there a way to fix the heuristic to make the algorithm faster? Can running the algorithm bidirectionally make it much faster?