uniform cost search

October 24, 2024

[1]:
```python
#uniform cost search
import heapq

def uniform_cost_search(graph, start, goal):
    priority_queue = []
    heapq.heappush(priority_queue, (0, start))
    visited = set()
    cost = {start: 0}
    parent = {start: None}

    while priority_queue:
        current_cost, current_node = heapq.heappop(priority_queue)

        if current_node in visited:
            continue

        visited.add(current_node)

        if current_node == goal:
            break

        for neighbor, weight in graph[current_node].items():
            new_cost = current_cost + weight
            if neighbor not in cost or new_cost < cost[neighbor]:
                cost[neighbor] = new_cost
                heapq.heappush(priority_queue, (new_cost, neighbor))
                parent[neighbor] = current_node

    return cost, parent

def reconstruct_path(parent, start, goal):
    path = []
    current_node = goal
    while current_node is not None:
        path.append(current_node)
        current_node = parent[current_node]
    path.reverse()
    return path

# Example usage:
graph = {
    'A': {'B': 1, 'C': 4},
    'B': {'A': 1, 'C': 2, 'D': 5},
    'C': {'A': 4, 'B': 2, 'D': 1},
    'D': {'B': 5, 'C': 1}
}

start = 'A'
goal = 'D'
cost, parent = uniform_cost_search(graph, start, goal)
path = reconstruct_path(parent, start, goal)

print("Cost from start to each node:", cost)
print("Path from start to goal:", path)
```

```
Cost from start to each node: {'A': 0, 'B': 1, 'C': 3, 'D': 4}
Path from start to goal: ['A', 'B', 'C', 'D']
```

[ ]: