UNIVERSIDADE FEDERAL DE RORAIMA
CENTRO DE CIÊNCIAS E TECNOLOGIA
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

Disciplina: Inteligência Artificial
Professor: Filipe Dwan Pereira
Aluno: Kelvin Araújo Ferreira – 2019037653

O objetivo da tarefa é implementar o Algoritmo A_Estrela (Algoritmo A*). O programa deve ser capaz de solucionar o problema do Mapa da Romênia e problemas da mesma classe. A tarefa é individual e deve ser original. Um documento descrevendo a linguagem usada, a versão e o ambiente de execução, deve ser elaborado para orientar usuários a utilizar o programa.

As linguagens empregadas podem ser C, C++, Java, ou Python.

Importante: Envie o código-fonte e não o executável.

O problema do mapa da Romênia é um problema clássico de busca de caminho em um grafo. Ele envolve encontrar o caminho mais curto entre duas cidades na Romênia usando um mapa com as estradas conectando as cidades.

O problema começa com um mapa com as cidades e estradas que as conectam. Cada estrada tem um comprimento associado que representa a distância entre as cidades que ela conecta. O objetivo é encontrar o caminho mais curto entre duas cidades especificadas no mapa.

O algoritmo A* é uma técnica de busca heurística que pode ser usada para encontrar o caminho mais curto entre duas cidades. Ele usa uma heurística para estimar a distância restante do nó atual até o nó de destino, além da distância já percorrida, para decidir qual nó deve ser explorado em seguida. O algoritmo A* é uma combinação da busca de custo uniforme e da busca heurística informada.

O problema do mapa da Romênia é um exemplo comum usado para ilustrar o funcionamento do algoritmo A* e outros algoritmos de busca. Ele é frequentemente usado como um problema de benchmark para comparar a eficiência de diferentes algoritmos de busca.

Podemos resolver seguindo os seguintes passos:

1. Defina as cidades e as estradas que as conectam como um grafo. Cada cidade pode ser representada por um nó no grafo, e cada estrada pode ser representada por uma aresta que conecta dois nós.
2. Defina as distâncias entre cada cidade. Isso pode ser feito usando uma matriz de adjacência, onde o valor em cada posição i, j representa a distância entre a cidade i e a cidade j.
3. Defina uma heurística que estima a distância restante do nó atual até o nó de destino. Uma heurística comum é a distância em linha reta entre as duas cidades. Essa heurística é admissível, o que significa que ela nunca superestima a distância real entre as cidades.
4. Implemente o algoritmo A*. Comece pelo nó de origem e use a heurística para estimar a distância restante até o nó de destino. Em seguida, selecione o nó com o menor custo total (a soma

da distância já percorrida e da estimativa da distância restante) e expanda seus vizinhos. Continue selecionando e expandindo os nós com menor custo total até que o nó de destino seja alcançado.

5. Rastreie o caminho mais curto do nó de origem ao nó de destino. Isso pode ser feito armazenando o nó anterior de cada nó visitado e, em seguida, percorrendo de volta do nó de destino até o nó de origem usando essas informações.

Código-Fonte desenvolvido:

```
# Kelvin Araújo Ferreira - 2019037653

import heapq

def heuristic(node, heuristic_dict):
    return heuristic_dict.get(node, float('inf'))

def a_star(graph, start, end, heuristic_dict):
    priority_queue = [(0, start)]
    visited = set()
    g = {start: 0}
    parent = {start: None}

    while priority_queue:
        _, current_node = heapq.heappop(priority_queue)
        if current_node == end:
            path = []
            while current_node is not None:
                path.append(current_node)
                current_node = parent[current_node]
            path.reverse()
            return path, g[end]

        visited.add(current_node)

        for neighbor, cost in graph[current_node].items():
            if neighbor in visited:
                continue

            tentative_g_score = g[current_node] + cost
            if tentative_g_score < g.get(neighbor, float('inf')):
                parent[neighbor] = current_node
                g[neighbor] = tentative_g_score
                f = tentative_g_score + heuristic(neighbor, heuristic_dict)
                heapq.heappush(priority_queue, (f, neighbor))

    return None, None

def main():
    with open('Romenia.txt', 'r') as f:
```

```
        romania_map = eval(f.read())

    with open('Heuristica.txt', 'r') as f:
        heuristic_dict = eval(f.read())

    start = 'Arad'
    end = 'Bucharest'

    path, cost = a_star(romania_map, start, end, heuristic_dict)

    if path is not None:
        print(f'Path found: {path}')
        print(f'Total cost: {cost}')
    else:
        print('Path not found')


if __name__ == '__main__':
    main()
```

O código que eu criei é capaz de ler dois arquivos de texto (.txt) que contêm informações sobre o mapa da Romênia e uma heurística para o algoritmo A* encontrar o caminho mais curto entre duas cidades.

O arquivo "romenia.txt" contém um dicionário que descreve as distâncias entre cada cidade adjacente na Romênia. O arquivo "heuristica.txt" contém um dicionário que contém o valor heurístico para cada cidade, que é usado para ajudar o algoritmo a decidir qual caminho seguir.

O algoritmo A* é implementado para encontrar o caminho mais curto entre duas cidades, usando a informação contida nos dois arquivos de texto. O caminho encontrado é então impresso na tela junto com o custo total.

Segue o Link do GitHub com o código e os arquivos .txt com as informações mencionadas e necessárias para conclusão do exercício proposto: https://github.com/DilliKel/DCC607-Inteligencia-Artificial-2023.1/tree/main/Atividade%2002