Projeto da Aula de Complexidade de Algoritmos Aula 13

Aluna: Luana Roza de Oliveira

1 - Implementar o algoritmo A* para encontrar o caminho entre dois pontos em um mapa, levando em consideração o custo de locomoção nesse mapa. O mapa é uma grade definida como uma série de valores inteiros. Se o valor for positivo ou igual a zero, representa o custo (esforço) de passar pelo terreno. Se o valor for negativo (-1), significa que é impossível passar pelo terreno.

O algoritmo A* é uma ótima escolha para encontrar o caminho mais eficiente entre dois pontos em um mapa, levando em consideração o custo de locomoção. Para implementá-lo, você precisará ter uma representação do mapa como uma nota de valores inteiros.

Nessa nota, cada valor positivo ou igual a zero representa o custo ou esforço necessário para passar por determinado terreno. Valores negativos podem ser usados para representar obstáculos ou áreas intransitáveis.

O algoritmo A* utiliza uma combinação de duas heurísticas para determinar a melhor rota: o custo atual do ponto de partida até o ponto atual (custo g) e uma estimativa do custo do ponto atual até o ponto de destino (custo h). A soma desses dois custos, conhecida como custo f, é usada para determinar a ordem de exploração dos pontos.

A implementação do algoritmo A* geralmente envolve a criação de uma estrutura de dados para armazenar os pontos a serem explorados, conhecida como lista aberta, e uma estrutura para armazenar os pontos já explorados, chamada de lista fechada.

No início, você coloca o ponto de partida na lista aberta. Em cada iteração, você seleciona o ponto com o menor custo da lista aberta e move para a lista fechada. Em seguida, você verifica os pontos vizinhos desse ponto e calcula os custos g, hef para cada um deles.

Se um ponto vizinho já estiver na lista fechada ou para um obstáculo, você o ignora. Caso contrário, você o adicionará à lista aberta e atualizará seus custos ge f.

O algoritmo continua a iterar até que o ponto de destino seja promissor ou até que a lista aberta esteja vazia, o que indica que não há caminho possível para o destino.

Ao encontrar o ponto de destino, você pode rastrear o caminho percorrido, seguindo os pontos com os menores custos g. Isso lhe dará uma rota mais eficiente entre os dois pontos no mapa.

Imagem da implementação feito na linguagem python:

```python
import heapq
def heuristic(node, goal):
    """Função heurística que estima o custo do nó ao objetivo (distância Euclidiana)"""
    return ((node[0] - goal[0]) ** 2 + (node[1] - goal[1]) ** 2) ** 0.5

def astar(mapa, inicio, objetivo):
    # Verifica se o início e o objetivo estão dentro dos limites do mapa
    if not (0 <= inicio[0] < len(mapa)) or not (0 <= inicio[1] < len(mapa[0])):
        raise ValueError("O ponto de início está fora dos limites do mapa.")
    if not (0 <= objetivo[0] < len(mapa)) or not (0 <= objetivo[1] < len(mapa[0])):
        raise ValueError("O ponto de objetivo está fora dos limites do mapa.")

    # Define os movimentos possíveis (8 direções: cima, baixo, esquerda, direita e diagonais)
    movimentos = [(0, 1), (0, -1), (1, 0), (-1, 0), (1, 1), (1, -1), (-1, 1), (-1, -1)]

    # Inicializa as estruturas de dados
    fronteira = []
    heapq.heappush(fronteira, (0, inicio))  # Prioridade e posição atual
    custo_acumulado = {inicio: 0}  # Custo acumulado para chegar ao nó atual
    pais = {}  # Dicionário para rastrear o nó pai de cada nó visitado

    while fronteira:
        _, atual = heapq.heappop(fronteira)

        if atual == objetivo:
            # Constrói o caminho a partir do objetivo até o início
            caminho = [atual]
            while atual in pais:
                atual = pais[atual]
                caminho.append(atual)
            caminho.reverse()
            return caminho

        for movimento in movimentos:
            proximo = atual[0] + movimento[0], atual[1] + movimento[1]
            novo_custo = custo_acumulado[atual] + mapa[proximo[0]][proximo[1]]

            if proximo not in custo_acumulado or novo_custo < custo_acumulado[proximo]:
                custo_acumulado[proximo] = novo_custo
                prioridade = novo_custo + heuristic(proximo, objetivo)
                heapq.heappush(fronteira, (prioridade, proximo))
                pais[proximo] = atual

    raise ValueError("Não foi possível encontrar um caminho válido.")
```

```python
# Exemplo de uso
mapa = [
    [1, 1, 1, 1, 1],
    [1, 0, 0, 0, 1],
    [1, 1, -1, 1, 1],
    [1, 0, 0, 0, 1],
    [1, 1, 1, 1, 1]
]

inicio = (0, 0)
objetivo = (4, 4)

caminho = astar(mapa, inicio, objetivo)
print(caminho)
```

Link Github com o código: https://github.com/Luh022/ProjectComplex