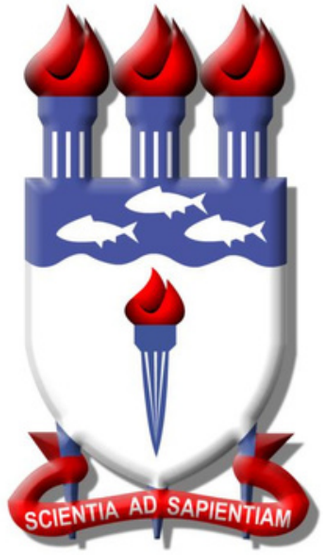


Algoritmo A*

Equipe

Bruno weverton Teixeira omena

Kauã Lessa Lima dos Santos

Efraim Leite Lopes

Pedro Resende

https://github.com/Efraim-Lopes/Estrutura-de-Dados_Huffman

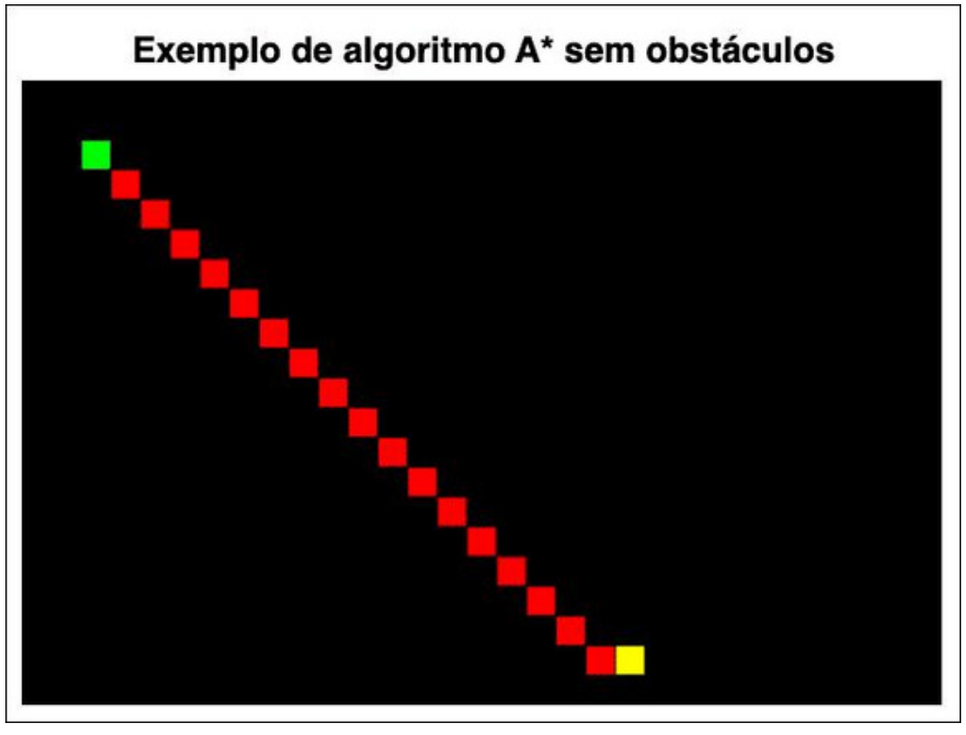
Motivação

Imagine que você está desenvolvendo um aplicativo de navegação por GPS e deseja encontrar a rota mais eficiente entre dois pontos em um mapa com estradas de diferentes comprimentos e velocidades. Isso é importante para fornecer ao usuário uma estimativa precisa do tempo de viagem. Você precisa de um algoritmo que não apenas encontre a rota mais curta, mas também leve em consideração a velocidade das estradas para calcular o tempo de viagem. Aqui, a motivação é encontrar uma solução eficiente para encontrar a rota mais rápida em um mapa.

Algoritmo A*

o que é?/para que serve?

O algoritmo A* é uma técnica de busca que combina a busca em largura (BFS) com uma heurística que estima o custo restante até o destino. Ele é usado para encontrar o caminho mais curto em um grafo ponderado. O A* é amplamente utilizado em aplicações de navegação, jogos e robótica para calcular rotas eficientes.



Definições

Nó: Cada ponto no grafo que representa uma posição possível no mapa.
Caminho: Uma sequência de nós que conectam dois pontos.
Custo: O custo associado a atravessar uma aresta entre dois nós.
Heurística: Uma função que estima o custo restante do nó atual até o destino.

Código

```
bool astar(Grid* grid, Point start, Point goal) {
    PriorityQueue openSet;
    openSet.head = NULL;

    Node* startNode = createNode(start, 0, calculateHeuristic(start, goal), NULL);
    startNode->cost = 0;

    openSet.head = startNode;

    while (openSet.head != NULL) {
        Node* current = NULL;
        if (!extractMin(&openSet, &current)) {
            break; // Fila vazia, nenhum caminho encontrado
        }

        // Remova o nó atual da lista de abertos
        if (openSet.head == current) {
            openSet.head = current->next;
        } else {
            Node* prev = openSet.head;
            while (prev->next != current) {
                prev = prev->next;
            }
            prev->next = current->next;
        }

        if (current->position.x == goal.x && current->position.y == goal.y) {
            printf("Caminho encontrado:\n");
            printPath(current, grid);
            return true;
        }

        // Gere os vizinhos do nó atual e processe-os
        Point neighbors[] = {
            {current->position.x + 1, current->position.y},
            {current->position.x - 1, current->position.y},
            {current->position.x, current->position.y + 1},
            {current->position.x, current->position.y - 1}
        };

        for (int i = 0; i < 4; i++) {
            Point neighbor = neighbors[i];
            if (isInsideGrid(neighbor, grid) && !isObstacle(neighbor, grid)) {
                int neighborCost = current->cost + 1; // Custo unitário
                int neighborHeuristic = calculateHeuristic(neighbor, goal);
                Node* neighborNode = createNode(neighbor, neighborCost, neighborHeuristic, current);
                neighborNode->next = openSet.head;
                openSet.head = neighborNode;
            }
        }
    }

    printf("Nenhum caminho encontrado.\n");
    return false;
}
```

Saída do algoritmo

```
/tmp/BE8q77qtEg.o
Caminho encontrado:
(0, 0) -> (0, 1) -> (1, 1) -> (1, 2) -> (1, 3) -> (1, 4) -> (1, 5) -> (1, 6) -> (1, 7)
    -> (1, 8) -> (1, 9) -> (2, 9) -> (3, 9) -> (4, 9) -> (5, 9) -> (6, 9) -> (7, 9) ->
    (8, 9) -> (9, 9) ->
X X # . . . . . . . .
. X X X X X X X X X
. . # . . . . . . X
. . # . . . . . . X
. . # . . . . . . X
. . # . . . . . . X
. . # . . . . . . X
. . # . . . . . # X
. . # # # # # # # X
. . . . . . . . . X
```

De volta à Motivação...

O algoritmo A* resolve o problema descrito na motivação porque leva em consideração tanto o custo acumulado até o momento quanto uma estimativa do custo restante até o destino. Isso permite que ele encontre a rota mais eficiente, considerando a velocidade das estradas ou qualquer outro fator relevante para calcular o tempo de viagem. Assim, ele fornece uma solução precisa para encontrar o caminho mais rápido em um mapa.