**Grupo: Flávia e Nathalia**

**Data de entrega: 04/09/2021**

Existem grandes semelhanças nos dois algoritmos, com a diferença de que

A \* tem um mecanismo heurístico, enquanto Dijkstra não. Com a heurística, A \*

geralmente supera Dijkstra em velocidade para encontrar um caminho, no entanto, ao contrário de Dijkstra, o caminho que A \* encontra nem sempre é o ideal.

Ou seja, o Dijkstra encontra os custos mínimos do nó inicial para todos os outros. A \* localiza o custo mínimo do nó inicial para o nó da meta.

O código para o algoritmo de Dijkstra:

// O programa é para representação da matriz de adjacência do grafo

#include <stdio.h>

#include <limits.h>

// Número de vértices no gráfico

#define V 9

// Uma função de utilidade para encontrar o vértice com valor mínimo de distância, d o conjunto de vértices ainda não incluído na árvore do caminho mais curto

int minDistance(int dist[], bool sptSet[])

{

int min = INT\_MAX, min\_index;

for (int v = 0; v < V; v++)

if (sptSet[v] == false && dist[v] <= min)

min = dist[v], min\_index = v;

return min\_index;

}

int printSolution(int dist[], int n)

{

printf("Vertex Distance from Source\n");

for (int i = 0; i < V; i++)

printf("%d \t\t %d\n", i, dist[i]);

}

void dijkstra(int graph[V][V], int src)

{

int dist[V];

bool sptSet[V];

for (int i = 0; i < V; i++)

dist[i] = INT\_MAX, sptSet[i] = false;

// A distância do vértice de origem é sempre 0

dist[src] = 0;

for (int count = 0; count < V-1; count++)

{

int u = minDistance(dist, sptSet);

// Mark the picked vertex as processed

sptSet[u] = true;

for (int v = 0; v < V; v++)

if (!sptSet[v] && graph[u][v] && dist[u] != INT\_MAX

&& dist[u]+graph[u][v] < dist[v])

dist[v] = dist[u] + graph[u][v];

}

// imprime a matriz de distância construída

printSolution(dist, V);

}

int main()

{

int graph[V][V] = {{0, 4, 0, 0, 0, 0, 0, 8, 0},

{4, 0, 8, 0, 0, 0, 0, 11, 0},

{0, 8, 0, 7, 0, 4, 0, 0, 2},

{0, 0, 7, 0, 9, 14, 0, 0, 0},

{0, 0, 0, 9, 0, 10, 0, 0, 0},

{0, 0, 4, 14, 10, 0, 2, 0, 0},

{0, 0, 0, 0, 0, 2, 0, 1, 6},

{8, 11, 0, 0, 0, 0, 1, 0, 7},

{0, 0, 2, 0, 0, 0, 6, 7, 0}

};

dijkstra(graph, 0);

return 0;

}

O código para o algoritmo A \*:

class Node:

def \_init\_(self,value,point):

self.value = value

self.point = point

self.parent = None

self.H = 0

self.G = 0

def move\_cost(self,other):

return 0 if self.value == '.' else 1

def children(point,grid):

x,y = point.point

links = [grid[d[0]][d[1]] for d in [(x-1, y),(x,y - 1),(x,y + 1),(x+1,y)]]

return [link for link in links if link.value != '%']

def manhattan(point,point2):

return abs(point.point[0] - point2.point[0]) + abs(point.point[1]-point2.point[0])

def aStar(start, goal, grid):

openset = set()

closedset = set()

current = start

openset.add(current)

while openset:

current = min(openset, key=lambda o:o.G + o.H)

if current == goal:

path = []

while current.parent:

path.append(current)

current = current.parent

path.append(current)

return path[::-1]

openset.remove(current)

closedset.add(current)

for node in children(current,grid):

if node in closedset:

continue

if node in openset:

new\_g = current.G + current.move\_cost(node)

if node.G > new\_g:

node.G = new\_g

node.parent = current

else:

node.G = current.G + current.move\_cost(node)

node.H = manhattan(node, goal)

node.parent = current

openset.add(node)

raise ValueError('No Path Found')

def next\_move(pacman,food,grid):

for x in xrange(len(grid)):

for y in xrange(len(grid[x])):

grid[x][y] = Node(grid[x][y],(x,y))

path = aStar(grid[pacman[0]][pacman[1]],grid[food[0]][food[1]],grid)

print len(path) - 1

for node in path:

x, y = node.point

print x, y

pacman\_x, pacman\_y = [ int(i) for i in raw\_input().strip().split() ]

food\_x, food\_y = [ int(i) for i in raw\_input().strip().split() ]

x,y = [ int(i) for i in raw\_input().strip().split() ]

grid = []

for i in xrange(0, x):

grid.append(list(raw\_input().strip()))

next\_move((pacman\_x, pacman\_y),(food\_x, food\_y), grid)