



Árvore geradora mínima de Kruskal

João Victor Superbi

Survey Paper on Maze Generation Algorithms for Puzzle Solving Games

Ms. Shivani H. Shah, Ms. Jagruti M. Mohite, Mr. Anoop G. Musale, Mr. Jay L. Borade

Abstract- A maze is a sort of puzzle having many branch passages. Some are closed and some lead to the end position. Here solver's goal is to move from a starting position to an end position i.e. a valid passage between these two positions has to be revealed. Maze generation is popular in entertainment domain. Mazes are used to solve the navigational problems which indeed brought the need of automation of maze generation. This paper gives an overview of three basic two-dimensional maze generation algorithms: a) Depth-First Search (DFS), b) Kruskal's Algorithm and c) Prim's Algorithm. These algorithms describe three conceptually different approaches for generating maze.

Index Terms- Depth-First Search, Graph algorithms, Grid, Kruskal's Algorithm, Maze generation algorithms, Maze creation algorithms, Prim's Algorithm.



GERADOR DE LABIRINTOS EM PYTHON !



17:37

The Buckblog

assorted ramblings by Jamis Buck

[Home](#) | [RSS](#)  | [Archives](#) | [Basil & Fabian](#)
[Announcements](#) | [Essays and Rants](#) | [Life](#) | [Metablog](#) | [Odds & Ends](#) | [Projects](#) | [Redirect](#) |
[Reviews](#) | [Spotlight](#) | [Tips & Tricks](#) | [Under the Hood](#) | [Mazes for Programmers!](#)

Maze Generation: Kruskal's Algorithm

3 January 2011 — Using Kruskal's algorithm to generate random spanning trees—or mazes — 5-minute read

For the third article in my series on maze algorithms, I'm going to take a look at Kruskal's algorithm. (I've previously covered [recursive backtracking](#) and [Eller's algorithm](#).)



Definição de Árvore Geradora e Grafo Ponderado

Árvore Geradora

Uma árvore geradora de um grafo é uma subárvore que conecta todos os vértices do grafo sem formar ciclos. Em outras palavras, é um subconjunto das arestas do grafo que mantém todos os vértices interligados, mas com o menor número possível de arestas (ou seja, uma árvore). Para um grafo com n vértices, uma árvore geradora terá exatamente $n-1$ arestas.

Grafo Ponderado

Um grafo ponderado é um grafo no qual cada aresta tem um valor associado, chamado de peso ou custo. Esses pesos podem representar distância, custo, tempo, ou qualquer outro valor relevante, dependendo da aplicação. No caso de problemas de otimização, como o algoritmo de Kruskal, o objetivo é geralmente encontrar um subconjunto de arestas que minimize o peso total ao conectar todos os vértices.



O Algoritmo de Kruskal

O algoritmo de Kruskal é uma técnica utilizada para encontrar a Árvore Geradora Mínima (AGM) de um grafo ponderado. A AGM é uma subárvore do grafo que conecta todos os vértices com o menor peso total possível, sem formar ciclos.



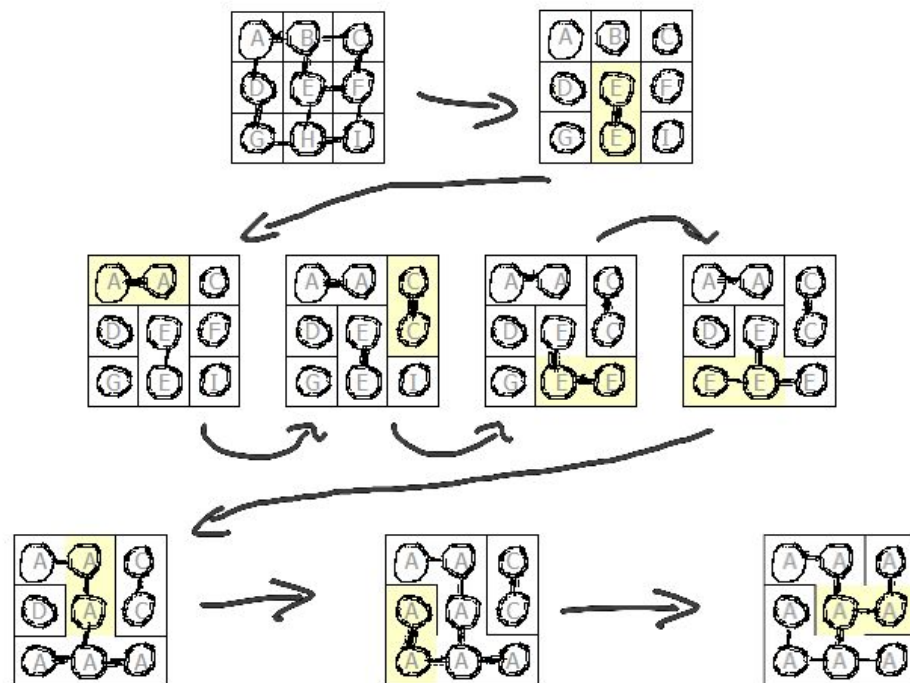
Passo a Passo do Algoritmo

Ordenação das Arestas: Primeiro, todas as arestas do grafo são ordenadas em ordem crescente de peso.

Construção da Árvore: Começando com um grafo vazio, as arestas são adicionadas uma a uma, em ordem de menor peso, mas somente se a aresta não formar um ciclo com as arestas já adicionadas.

Ciclo Evitado: Para evitar ciclos, o algoritmo utiliza uma estrutura de dados chamada Union-Find (ou Conjuntos Disjuntos), que ajuda a verificar se dois vértices já estão conectados.

Conexão Total: O algoritmo termina quando todas as arestas necessárias para conectar todos os vértices forem adicionadas, formando a árvore geradora mínima.






Possíveis Usos

Redes de computadores

Design de circuitos

Distribuição de energia



```
# Aqui cria a malha 2D do labirinto. Cada quadradinho é um vertice do grafo. E só fazem ligação na horizontal e vertical.
width, height = (WIDTH // 32, HEIGHT // 32)
grid = [[NODE() for i in range(width)] for j in range(height)]

edge_list = createEdgeList(width, height)

random.seed(42)
random.shuffle(edge_list)

clock = pygame.time.Clock()
running = True
it = 0

printGrid(grid)
```



```
# Um while que vai construindo o labirinto a cada interação.  
# uma aresta aleatória é selecionada e se os dois vértices não estão no mesmo conjunto, eles são conectados.  
# Se estão no mesmo conjunto, então não se faz nada.  
# Se em algum momento formar um ciclo, então a aresta é descartada e se pega outra.  
# O algoritmo termina quando todos os vértices estão no mesmo conjunto.  
  
# O algoritmo de kruskal é um algoritmo de construção de árvore geradora mínima.  
# Ou seja, ele cria um novo grafo chamado de árvore onde todos os vértices do grafo original são conectados  
# Não possui ciclos e a soma dos pesos das arestas é a menor possível.  
countSet = 1  
PrimeiraInteracao = True
```

#não terminou?

if flag == False:

#insere uma posição no labirinto (conecta dois quadrados do grid)

edge = edge_list.pop(0)

node_a, node_b = edge[0], edge[1]

print("aresta:", edge)

print("a:", node_a)

print("b:", node_b)

print(node_a[X] - node_b[X], node_a[Y] - node_b[Y])

print("-----")

if PrimeiraInteracao:

drawRedSquare(screen, node_a[X] * 32, node_a[Y] * 32, 32)

PrimeiraInteracao = False

xPosition = node_a[X] * 32

yPosition = node_a[Y] * 32

else:

drawBlueSquare(screen, node_a[X] * 32, node_a[Y] * 32, 32)

drawBlueSquare(screen, node_b[X] * 32, node_b[Y] * 32, 32)

if grid[node_a[Y]][node_a[X]].set == '0' and grid[node_b[Y]][node_b[X]].set == '0':

grid[node_a[Y]][node_a[X]].set = str(countSet)

grid[node_b[Y]][node_b[X]].set = str(countSet)

countSet+=1

if node_a[X] - node_b[X] == -1: #esquerda pra direita

grid[node_a[Y]][node_a[X]].right = 1

grid[node_b[Y]][node_b[X]].left = 1