gianfrancojoel_condoriluna_aula11Pratico

November 22, 2024

1 Aula 11 - Exercício prático Grafos

Aluno: Gian Franco Joel Condori Luna

(1,0) Submeta no Beecrowd o problema 1621 Labyrinth:

https://www.beecrowd.com.br/judge/pt/problems/view/1621?origem=1

[ ]:
```
# DADOS DE TESTE
'''
N = 5
M = 5

matriz_1 = [
    [".", "#", ".", ".", "."],
    [".", ".", ".", "#", "#"],
    [".", "#", ".", ".", "#"],
    [".", "#", "#", ".", "."],
    ["#", "#", "#", "#", "#"],
]

matriz_2 = [
    [".", ".", ".", ".", "."],
    ["#", "#", "#", "#", "."],
    [".", ".", ".", ".", "."],
    [".", "#", "#", "#", "#"],
    [".", ".", ".", ".", "."],
]

print(matriz[0][1])
'''
```

[1]:
```
# Definição de nó de árvore
class TreeNode:
    def __init__(self, value=[], father= None, left=None, right=None,␣
 ↪center=None):
        self.value = value
        self.father = father
        self.left = left
```

```
      self.right = right
      self.center = center

    def print(self):
      print(f"{self.value}")

    def getValue(self):
       return self.value

    def getchildren(self):
      if self.left:
        print(f"{self.left.value}")
      if self.right:
        print(f"{self.right.value}")
      if self.center:
        print(f"{self.center.value}")
```

[2]:
```
# Função para calcular a profundidade da árvore
def calculate_depth(root):
    if root is None:
        return 0
    left_depth = calculate_depth(root.left)
    right_depth = calculate_depth(root.right)
    center_depth = calculate_depth(root.center)
    return max(left_depth, right_depth, center_depth) + 1
```

[3]:
```
# Função para procurar possíveis inícios nas bordas da matriz
def searchOptions(matriz, m_i, m_j, m_N, m_M):
  list_option = []
  for i in range(m_i,m_N):
    if i == m_i or i == m_N-1:
      for j in range(m_j, m_M):
        if matriz[i][j] == ".":
          list_option.append((i,j))
    else:
      if matriz[i][m_i] == ".":
        list_option.append((i,m_i))
      if matriz[i][m_M-1] == ".":
        list_option.append((i,m_M-1))
  return list_option
```

[4]:
```
# Função que me ajudará a obter caminhos possíveis do nó onde estou
def routeOption(matriz, p_tree, m_N, m_M):
  i,j = p_tree.getValue()
  routes = [(i-1,j),(i,j+1),(i+1,j),(i,j-1)]# Os 4 caminhos possíveis

  children = []
```

```
  for row, col in routes:
    if row >= 0 and col >= 0 and row<m_N and col<m_M and matriz[row][col] == ".
↪" and ( p_tree.father is None or  p_tree.father.getValue() != (row, col) ):
      children.append((row, col))

  return children
```

[5]:
```
# Função que me ajudará a inserir os filhos em cada nó
def insertChild(matriz, p_tree):
  children = routeOption(matriz, p_tree, N, M)

  # Perguntando se tem filhos
  if children:
    if len(children) == 1:
      p_tree.left = TreeNode(value=children[0], father=p_tree )
      insertChild(matriz, p_tree.left)
    elif len(children) == 2:
      p_tree.left = TreeNode(value=children[0], father=p_tree)
      p_tree.right = TreeNode(value=children[1], father=p_tree)
      insertChild(matriz, p_tree.left)
      insertChild(matriz, p_tree.right)
    else:
      p_tree.left = TreeNode(value=children[0], father=p_tree)
      p_tree.right = TreeNode(value=children[1], father=p_tree)
      p_tree.center = TreeNode(value=children[2], father=p_tree)
      insertChild(matriz, p_tree.left)
      insertChild(matriz, p_tree.right)
      insertChild(matriz, p_tree.center)
```

[8]:
```
# Função principal
#if __name__ == "__main__":
while True:
  # Leia as dimensões da matriz
  N, M = map(int, input().split())

  # Se as dimensões forem 0, interrompa o loop
  if N == 0 and M == 0:
      break

  # Valide se n e m estão dentro do intervalo permitido
  if not (5 <= N <= 500 and 5 <= M <= 500):
    print("As dimensões devem estar no intervalo 5   N, M   500. Tente novamente.
↪")
    continue  # Continue com o próximo ciclo do loop

  # Inicializar a matriz
  matriz = []
```

```
# Leia as linhas da matriz
for _ in range(N):
  fila = input().strip()  # Leia uma linha e remova espaços extras
  matriz.append(fila)     # Adicione a linha à matriz

# NOTA: Vamos trabalhar partindo do pressuposto de que sempre há um ponto "."␣
↪na borda.
#list_option = searchOptions(matriz_2,0+1+1,0+1+1,N-1-1,M-1-1)
list_option = searchOptions(matriz,0,0,N,M)
list_option

deep_max = 0
if list_option:
  for option in list_option:
    # Criamos o primeiro nó da árvore
    tree = TreeNode(option)
    insertChild(matriz, tree)
    deep = calculate_depth(tree)
    if deep > deep_max:
      deep_max = deep
print(deep_max-1)
```

```
8
16
```