Conjunto de Problemas: BFS e DFS

Prof. Rodrigo Silva

Introdução

Este conjunto de problemas divide a implementação dos algoritmos BFS (Busca em Largura) e DFS (Busca em Profundidade) em pequenas partes. O objetivo é ajudar os alunos a implementar gradualmente cada componente e, em seguida, combiná-los para formar os algoritmos completos.

Parte 1: Representando o Grafo

Problema 1.1: Representação do Grafo

Escreva uma função para representar um grafo usando um dicionário, onde cada chave é um nó e o valor é uma lista de seus vizinhos. (Veja `https://www.w3schools.com/python/python_dictionaries.asp` para aprender como usar dicionários (a estrutura de dados usada para representar o grafo) em Python).

```
Grafo = {
    'A': ['B', 'C'],
    'B': ['A', 'D'],
    'C': ['A', 'E'],
    'D': ['B'],
    'E': ['C']
}
```

A função `adicionar_aresta(nó1, nó2)` deve atualizar esta representação de dicionário.

Algorithm 1 adicionar_aresta(nó1, nó2)

```
if nó1 não está no Grafo then
  Grafo[nó1] ← lista vazia
end if
if nó2 não está no Grafo then
  Grafo[nó2] ← lista vazia
end if
adicionar nó2 à lista de Grafo[nó1]
adicionar nó1 à lista de Grafo[nó2] {como o grafo é não direcionado}
```

—

Parte 2: BFS - Busca em Largura

Problema 2.1: Inicializando a Fila

Escreva uma função que inicializa uma fila para BFS usando uma lista Python.

Algorithm 2 Inicializar_Fila(nó_inicial)

```
fila ← [nó_inicial] {Inicializa a fila com nó_inicial}
return fila
```

—

Problema 2.2: Lidando com Vizinhos

Escreva uma função para visitar um nó e enfileirar todos os seus vizinhos não visitados.

Algorithm 3 Visitar_Vizinhos(grafo, nó, visitados, fila)

```
for cada vizinho em grafo[nó] do
  if vizinho não está em visitados then
    fila.append(vizinho) {enfileira vizinho}
    visitados.add(vizinho)
  end if
end for
```

—

Problema 2.3: BFS Completo

Agora, junte tudo para completar o algoritmo BFS.

Algorithm 4 BFS(grafo, nó_inicial)

```
visitados ← set() {conjunto vazio}
fila ← Inicializar_Fila(nó_inicial)
visitados.add(nó_inicial)
while fila não está vazia do
  nó ← fila.pop(0) {desenfileira da frente}
  imprimir "Visitado", nó
  Visitar_Vizinhos(grafo, nó, visitados, fila)
end while
```

—

Parte 3: DFS - Busca em Profundidade

Problema 3.1: Inicializando a Pilha

Escreva uma função que inicializa uma pilha para DFS usando uma lista Python.

Algorithm 5 Inicializar_Pilha(nó_inicial)

```
pilha ← [nó_inicial] {Inicializa a pilha com nó_inicial}
return pilha
```

—

Problema 3.2: Lidando com Vizinhos na DFS

Escreva uma função para visitar um nó e empilhar todos os seus vizinhos não visitados.

Algorithm 6 Visitar_Vizinhos_DFS(grafo, nó, visitados, pilha)

```
for cada vizinho em grafo[nó] do
  if vizinho não está em visitados then
    pilha.append(vizinho) {empilha o vizinho}
    visitados.add(vizinho)
  end if
end for
```

—

Problema 3.3: DFS Completo

Agora, junte tudo para completar o algoritmo DFS.

Algorithm 7 DFS(grafo, nó_inicial)

```
visitados ← set() {conjunto vazio}
pilha ← Inicializar_Pilha(nó_inicial)
visitados.add(nó_inicial)
while pilha não está vazia do
  nó ← pilha.pop() {desempilha da pilha}
  imprimir "Visitado", nó
  Visitar_Vizinhos_DFS(grafo, nó, visitados, pilha)
end while
```

—

Parte 4: DFS Recursivo (Opcional)

Problema 4.1: DFS Recursivo

Escreva uma função recursiva para executar a DFS.

Algorithm 8 DFS_Recursivo(grafo, nó, visitados)

```
visitados.add(nó)
imprimir "Visitado", nó
for cada vizinho em grafo[nó] do
  if vizinho não está em visitados then
    DFS_Recursivo(grafo, vizinho, visitados)
  end if
end for
```

—

Passo Final: Juntando Tudo

No final, os alunos devem testar os algoritmos BFS e DFS em um grafo de exemplo, combinando todas as etapas. Isso ajudará a reforçar como cada parte se encaixa no algoritmo completo.