UEZO
FUNDAÇÃO CENTRO UNIVERSITÁRIO
ESTADUAL DA ZONA OESTE
Rio de Janeiro / RJ / Brasil

ALGORIMO DE BUSCA BACKTRACKING

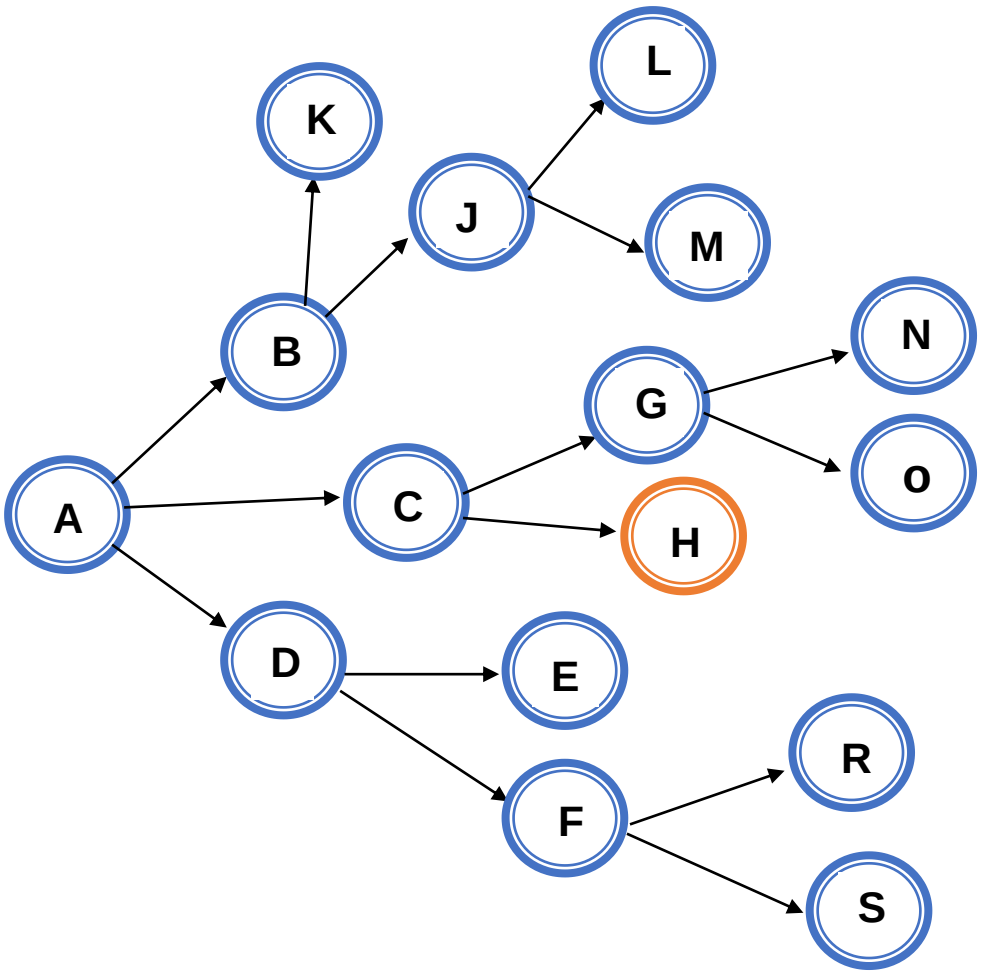
Inteligência Computacional I

Ana Cláudia Gomes Souza

Rio de Janiero, Maio 2019

Para a implementação do algoritmo de busca backtracking foi utilizada a linguagem Python. A estrutura do código é composta por um TAD de cinco arquivos. Com o auxilio de um mapa composto de letras do alfabeto será realizada a entrada de dados. O objetivo do algoritmo é percorrer os estados do mapa para encontrar a letra H.

Mapa de letras



Descrição dos arquivos do TAD

Arquivo Pilha.py

```
class Pilha:
    def __init__(self,tamanho):
        self.tamanho = tamanho
        self.letras = [None] * self.tamanho
        self.topo = -1

    def empilhar(self,letras):
        if not Pilha.pilhaCheia(self):
            self.topo += 1
            self.letras[self.topo] = letras
        else:
            print('A pilha está cheia!')

    def desempilhar(self):
        if not Pilha.pilhavazia(self):
            temp = self.letras[self.topo]
            self.topo -= 1
            return temp
        else:
            print('A pilha já está vazia!')
            return None

    def getTopo(self):
        return self.letras[self.topo]

    def pilhaVazia(self):
        return (self.topo == -1)

    def pilhaCheia(self):
        return (self.topo == self.tamanho -1)
```

Neste arquivo foi criada a classe Pilha, os objetos tamanho responsável por receber o tamanho da pilha, letras responsável por armazenar as letras que formam a pilha e topo responsável por receber o último valor inserido. Também foram criados os métodos getTopo que retorna o valor do topo da pilha, pilhaVazia responsável por verificar se a pilha está vazia, pilhaCheia responsável por verificar se a pilha está cheia, empilhar que recebe os valores de entrada e os empilha na pilha letras e o método desempilhar que retira os valores da pilha letras no sentido que o último valor inserido será o primeiro valor a ser retirado.

Arquivo Letras.py

```
class Letras:
    def __init__(self,nome):
        self.nome = nome
        self.visitado = False
        self.adjacentes = []

    def AddLetrasAdjacentes(self,letras):
        self.adjacentes.append(letras)
```

Neste arquivo foi criada a classe Letras e os objetos nome, visitado e adjacentes. Também foi implementado o método AddLetrasAdjacentes que é responsável por adicionar os valores que são adjacentes de um estado

Arquivo Adjacentes.py

```
class Adjacentes:
    def __init__(self,letras):
        self.letras = letras
```

Neste arquivo foi implementado o método que insere os valores adjacentes de um estado na pilha letras

Arquivo Mapa.py

```
from Letras import Letras
from Adjacentes import Adjacentes

class Mapa:
    A = Letras("A")
    B = Letras("B")
    C = Letras("C")
    D = Letras("D")
    E = Letras("E")
    F = Letras("F")
    G = Letras("G")
    H = Letras("H")
    J = Letras("J")
    K = Letras("H")
    L = Letras("L")
    M = Letras("M")
    N = Letras("N")
    O = Letras("O")
    R = Letras("R")
    S = Letras("S")
```

```
'''Adicionando as letras que são adjacentes a letra A'''
A.AddLetrasAdjacentes(Adjacentes(B))
A.AddLetrasAdjacentes(Adjacentes(C))
A.AddLetrasAdjacentes(Adjacentes(D))

'''Adicionando as letras que são adjacentes a letra B'''
B.AddLetrasAdjacentes(Adjacentes(K))
B.AddLetrasAdjacentes(Adjacentes(J))

'''Adicionando as letras que são adjacentes a letra J'''
J.AddLetrasAdjacentes(Adjacentes(L))
J.AddLetrasAdjacentes(Adjacentes(M))

'''Adicionando as letras que são adjacentes a letra C'''
C.AddLetrasAdjacentes(Adjacentes(G))
C.AddLetrasAdjacentes(Adjacentes(H))

'''Adicionando as letras que são adjacentes a letra G'''
G.AddLetrasAdjacentes(Adjacentes(N))
G.AddLetrasAdjacentes(Adjacentes(O))

'''Adicionando as letras que são adjacentes a letra D'''
D.AddLetrasAdjacentes(Adjacentes(E))
D.AddLetrasAdjacentes(Adjacentes(F))

'''Adicionando as letras que são adjacentes a letra F'''
F.AddLetrasAdjacentes(Adjacentes(R))
F.AddLetrasAdjacentes(Adjacentes(S))
```

Neste arquivo foi criada a classe Mapa e os objetos que repesentam a letra de cada estado, também foi implementado o método AddLetrasAdjacentes que adiciona na pilha os valores adjacentes de cada estado, para isso importamos os arquivos Letras e Adjacentes que contém as classes Letras e Adjacentes.

Arquivo Backtracking.py

```
from Pilha import Pilha

class Backtracking:
    def __init__(self,inicio,objetivo):
        self.inicio = inicio
        self.inicio.visitado = True
        self.objetivo = objetivo
        self.fronteira = Pilha(16)
        self.fronteira.empilhar(inicio)
        self.achou = False

    def buscar(self):
        topo = self.fronteira.getTopo()
        print('Topo {}'.format(topo.nome))

        if topo == self.objetivo:
            self.achou = True
            print('Objetivo alcançado!')
        else:
            for a in topo.adjacentes:
                if self.achou == False:
                    if a.letras.visitado == False:
                        a.letras.visitado = True
                        print('Estado visitado: {}'.format(a.letras.nome))
                        self.fronteira.empilhar(a.letras)
                        Backtracking.buscar(self)
            print('Desempilhou: {}'.format(self.fronteira.desempilhar().nome))

from Mapa import Mapa
mapa = Mapa()

backtracking = Backtracking(mapa.A,mapa.H)
backtracking.buscar()
```

Neste arquivo foi criada a classe Backtracking e os objetos inicio, inicio.visitado, objetivo,fronteira e achou. Também foi implementado o método buscar responsável por verificar se um estado já foi ou não visitado caso este ainda não tenha sido visitado receberá o valor False caso contrário receberá o valor True, o método buscar também verifica qual valor é o nosso objetivo e faz uma busca desse valor na pilha letras.

Para criarmos o mapa de letras importamos o arquivo Mapa.py que contém a classe Mapa, criamos uma variável mapa que é responsável por instanciar a classe Mapa. Utilizamos a variável backtracking para instanciar a classe Backtracking e informamos nosso valor inicial (A) e nosso valor objetivo (H) e por fim executamos o método buscar()

Impressão da saída do processamento dos dados da pilha letras

```
In [9]: runfile('C:/Users/aninh/.spyc
Users/aninh/.spyder-py3/AlgoritmosBus
Topo A
Estado visitado: B
Topo B
Estado visitado: H
Topo H
Desempilhou: H
Estado visitado: J
Topo J
Estado visitado: L
Topo L
Desempilhou: L
Estado visitado: M
Topo M
Desempilhou: M
Desempilhou: J
Desempilhou: B
Estado visitado: C
Topo C
Estado visitado: G
Topo G
Estado visitado: N
Topo N
Desempilhou: N
Estado visitado: O
Topo O
Desempilhou: O
Desempilhou: G
Estado visitado: H
Topo H
Objetivo alcançado!
Desempilhou: H
Desempilhou: C
```