UNIVERSIDADE CATÓLICA DE PELOTAS

ENGENHARIA DE COMPUTAÇÃO

INTELIGÊNCIA ARTIFICIAL

ÍCARO GONÇALVES SIQUEIRA

COMPARAÇÃO ENTRE BUSCA EM PROFUNDIDADE E BUSCA GULOSA

05/05/2020

I. INTRODUÇÃO

Este trabalho é referente à nota do primeiro bimestre da disciplina de Inteligência Artificial, ministrada pela professora Alexandra Zimpeck, e determina que sejam implementados dois algoritmos de busca na linguagem de preferência.

A escolha dos algoritmos deve ser dividida em um algoritmo de busca sem informação e um de busca com informação e em cada um deles devem ser executadas 3 árvores de busca com diferentes quantidades de nós, para uma melhor comparação das implementações.

A comparação deve ser medida em tempo de execução e analisada com a criação de gráficos, para melhor clareza no processo de execução de cada algoritmo.

II. METODOLOGIA

O algoritmo de busca sem informação escolhido foi o busca em profundidade, esta escolha foi feita devido ao bom desempenho do algoritmo, em comparação a busca em largura, por exemplo, e também pela sua simplicidade de implementação.

Na busca com informação, o algoritmo escolhido foi o busca gulosa e esta escolha foi feita devido ao fato de ser um algoritmo com baixo desempenho em comparação aos outros algoritmos com informação, então se torna uma comparação interessante para ser medida.

A busca em profundidade consiste basicamente em testar sempre os nós mais distantes, onde não é possível seguir. Já a busca gulosa segue sempre pelos nós vizinhos que possuem a menor distância entre eles e o nó objetivo.

A linguagem de programação escolhida para o desenvolvimento dos algoritmos foi o python 3, devido principalmente à afinidade com a linguagem. Pela facilidade de executar o código para teste em um simples editor de texto, o sublime text foi usado para o desenvolvimento dos códigos.

As árvores de busca implementadas foram árvores binárias, de 6, 14 e 30 nodos e preenchidas com números em ordem crescente. Nas árvores de busca da implementação com informação foram implementadas também as distâncias de cada nó em relação com o nó de destino.

Os pontos de origem e destino escolhidos foram 0 e 11, respectivamente, nas árvores de tamanhos 14 e 30 e, nas árvores de tamanho 6, os pontos foram 0 e 3. Os pontos iguais com tamanhos diferentes vão servir para avaliar a diferença de desempenho entre árvores de tamanhos diferentes, já o ponto diferente vai servir para analisar se é possível um melhor desempenho com busca sem informação.

A. Código de Busca em Profundidade

```
#
#                                       0
#
#                   1                                       2
#
#          3                    4                   5                   6
#
#    7          8          9         10         11         12         13         14
#
# 15   16   17   18   19   20   21   22   23   24   25   26   27   28   29   30

import time

origem = '0'
destino = '12'
#destino = '3'

dist = 0
testado = set()

def buscaEmProfundidade(testado, graph, node, search):
    global dist
    if node not in testado:
        print (node)
        testado.add(node)
        dist+=1

        if node == search:
            print('Encontrado!')
            fim = time.time()
            print('Distancia: %d' %dist)
            print("Tempo: %fs" % (fim - inicio))
        for vizinho in graph[node]:
            buscaEmProfundidade(testado, graph, vizinho, search)
```

O código inicia com uma representação da maior árvore a ser implementada, importando a biblioteca time e criando variáveis iniciais, como os nodos de origem e destino, o somador da distância percorrida e a lista de nodos visitados.

A função da busca em profundidade é iniciada com a lista de nodos visitados, o grafo a ser percorrido, o nodo inicial/atual e o nodo a ser buscado. Após iniciar a variável global do somador de distância, será testado se o nodo atual já foi visitado anteriormente, se não foi o código continua.

O próximo teste a ser feito já é se o nodo atual é o nodo buscado, se for igual o código irá imprimir o tempo consumido e o caminho percorrido, e se não tiver chegado no nodo buscado um loop é iniciado com chamadas recursivas, chamando sempre o próximo nodo da árvore.

```
34  #grafos em python dictionary
35  arvore1 =  {
36      '0' : ['1','2'],
37      '1' : ['3', '4'],
38      '2' : ['5', '6'],
39      '3' : ['7', '8'],
40      '4' : ['9', '10'],
41      '5' : ['11', '12'],
42      '6' : ['13', '14'],
43      '7' : ['15', '16'],
44      '8' : ['17', '18'],
45      '9' : ['19', '20'],
46      '10' : ['21', '22'],
47      '11' : ['23', '24'],
48      '12' : ['25', '26'],
49      '13' : ['27', '28'],
50      '14' : ['29', '30'],
51      '15' : [],
52      '16' : [],
53      '17' : [],
54      '18' : [],
55      '19' : [],
56      '20' : [],
57      '21' : [],
58      '22' : [],
59      '23' : [],
60      '24' : [],
61      '25' : [],
62      '26' : [],
63      '27' : [],
64      '28' : [],
65      '29' : [],
66      '30' : []
67  }
68
```

Aqui é feita a criação da árvore binária de 30 nodos, preenchida com strings de números e usando a estrutura de dicionário do python.

```
arvore2 =  {
    '0' : ['1','2'],
    '1' : ['3', '4'],
    '2' : ['5', '6'],
    '3' : ['7', '8'],
    '4' : ['9', '10'],
    '5' : ['11', '12'],
    '6' : ['13', '14'],
    '7' : [],
    '8' : [],
    '9' : [],
    '10' : [],
    '11' : [],
    '12' : [],
    '13' : [],
    '14' : []
}

arvore3 =  {
    '0' : ['1','2'],
    '1' : ['3', '4'],
    '2' : ['5', '6'],
    '3' : [],
    '4' : [],
    '5' : [],
    '6' : []
}

inicio = time.time()

buscaEmProfundidade(testado, arvore1, origem, destino)
```

As outras árvores, implementadas na mesma estrutura, com 14 e 6 nodos. O início do medidor de tempo é feito aqui, logo antes da chamada da função de busca, onde são enviados os parâmetros já citados.

B. Código de Busca Gulosa

```
12  import time
13
14  origem = 0
15  destino = 12
16  #destino = 3
17
18  #grafos em python dictionary
19  arvore1 =  {
20      0 : [(1,4),(2,3)],
21      1 : [(3,5),(4,5)],
22      2 : [(5,1),(6,2)],
23      3 : [(7,6), (8,6)],
24      4 : [(9,6), (10,6)],
25      5 : [(11,2), (12,0)],
26      6 : [(13,4), (14,4)],
27      7 : [(15,7), (16,7)],
28      8 : [(17,7), (18,7)],
29      9 : [(19,7), (20,7)],
30      10 : [(21,7), (22,7)],
31      11 : [(23,3), (24,3)],
32      12 : [(25,1), (26,1)],
33      13 : [(27,5), (28,5)],
34      14 : [(29,5), (30,5)],
35      15 : [],
36      16 : [],
37      17 : [],
38      18 : [],
39      19 : [],
40      20 : [],
41      21 : [],
42      22 : [],
43      23 : [],
44      24 : [],
45      25 : [],
46      26 : [],
47      27 : [],
48      28 : [],
49      29 : [],
50      30 : []
51  }
```

O código é iniciado com variáveis de inteiros que representam os nodos de origem e destino e, em seguida, é feita a criação da árvore binária de 30 nodos, preenchida com duplas de números inteiros que representam o número armazenado no nodo e a distância até o nodo de destino, respectivamente.

```
52
53  arvore2 =  {
54      0 : [(1,4),(2,3)],
55      1 : [(3,5),(4,5)],
56      2 : [(5,1),(6,2)],
57      3 : [(7,6), (8,6)],
58      4 : [(9,6), (10,6)],
59      5 : [(11,0), (12,2)],
60      6 : [(13,4), (14,4)],
61      7 : [],
62      8 : [],
63      9 : [],
64      10 : [],
65      11 : [],
66      12 : [],
67      13 : [],
68      14 : []
69  }
70
71  arvore3 =  {
72      0 : [(1,1),(2,3)],
73      1 : [(3,0),(4,2)],
74      2 : [(5,4),(6,4)],
75      3 : [],
76      4 : [],
77      5 : [],
78      6 : []
79  }
80
```

As outras árvores, implementadas na mesma estrutura de dicionário do python, com 14 e 6 nodos.

```
81  def buscaGulosa(graph, node, search):
82      nodoAtual = node
83      caminho = []
84      caminho.append(nodoAtual)
85      distancia = 0
86      while len(set([nodoVizinho for (nodoVizinho, distance) in graph.get(nodoAtual, [])]).
            difference(set(caminho))) > 0:
87          vizinhoProximo = None
88          menorDistancia = None
89          for vizinho, vizinhoDistancia in graph[nodoAtual]:
90              if vizinho != nodoAtual and vizinho not in caminho:
91                  if menorDistancia is not None:
92                      if menorDistancia > vizinhoDistancia:
93                          menorDistancia = vizinhoDistancia
94                          vizinhoProximo = vizinho
95                  else:
96                      menorDistancia = vizinhoDistancia
97                      vizinhoProximo = vizinho
98          vizinhoMaisProximo = (vizinhoProximo, menorDistancia)
99          nodoAtual = vizinhoMaisProximo[0]
100         caminho.append(nodoAtual)
101         distancia += vizinhoMaisProximo[1]
102         if nodoAtual == search:
103             fim = time.time()
104             print("caminho: %s" %caminho)
105             print("distancia: %d" %distancia)
106             print("Tempo: %fs" %(fim - inicio))
107             return 0
108
109 inicio = time.time()
110
111 buscaGulosa(arvore1, origem, destino)
```

O código inicia com uma representação da maior árvore a ser implementada, importando a biblioteca time e criando variáveis iniciais, como os nodos de origem e destino, o somador da distância percorrida e a lista de nodos visitados.

A função da busca gulosa tem como parâmetro de entrada o grafo a ser percorrido, o nodo inicial e o nodo a ser buscado. É iniciado o vetor de armazenamento do caminho percorrido e o somador de distância percorrida e então o loop é iniciado.

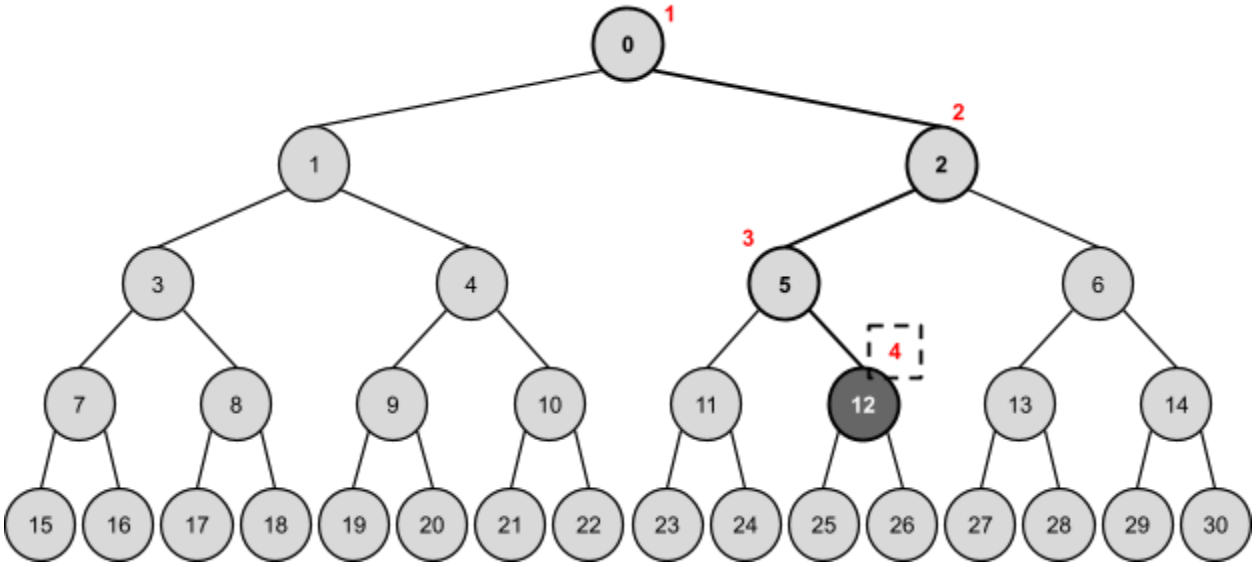
No loop são feitos testes de validação de caminho e busca pelo caminho mais curto dos próximos nodos até o nodo objetivo. São atualizadas as variáveis e o vetor de caminhos e somador de distância, até que no fim é feito um teste se o nodo atual é o nodo buscado onde, se for positivo, será imprimido o caminho percorrido, a distância total e o tempo de busca, para então o loop será encerrado.

Por fim é iniciado o medidor de tempo, logo antes da chamada da função de busca, onde são enviados os parâmetros já citados.

III. RESULTADOS

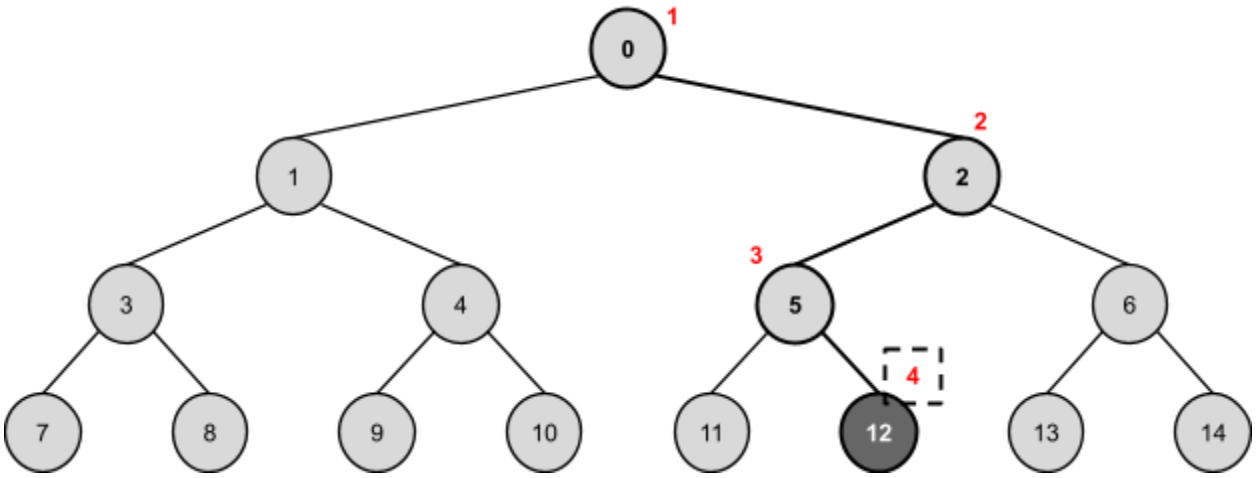
A. Gráficos de Árvore da Busca Gulosa

a. Árvore de 30 Nodos



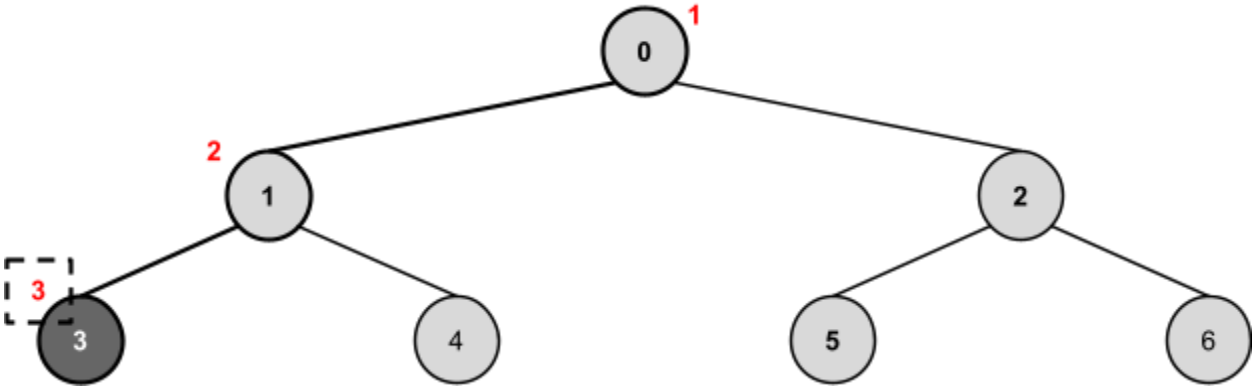
Na busca gulosa, buscando o nodo de valor 12 na árvore de 30 nodos, o caminho usado para chegar até o valor foi 0,2,5,12, usando uma distância de 4 nodos.

b. Árvore de 14 Nodos



Na busca gulosa, buscando o nodo de valor 12 na árvore de 14 nodos, o caminho usado para chegar até o valor foi 0,2,5,12, usando uma distância de 4 nodos.

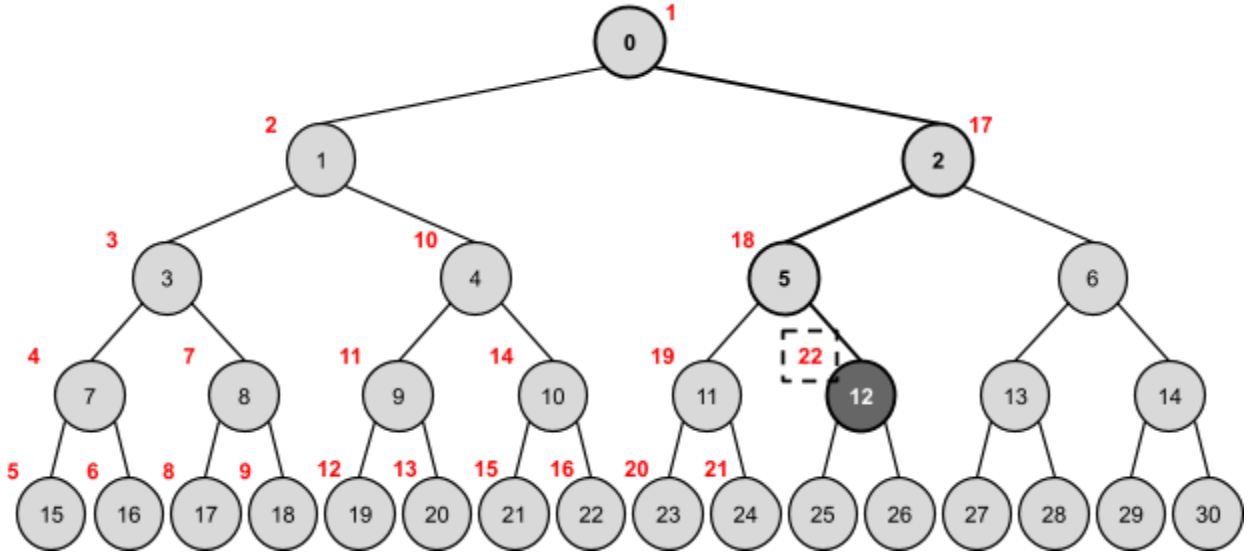
c. Árvore de 6 Nodos



Na busca gulosa, buscando o nodo de valor 3 na árvore de 6 nodos, o caminho usado para chegar até o valor foi 0,1,3, usando uma distância de 3 nodos.

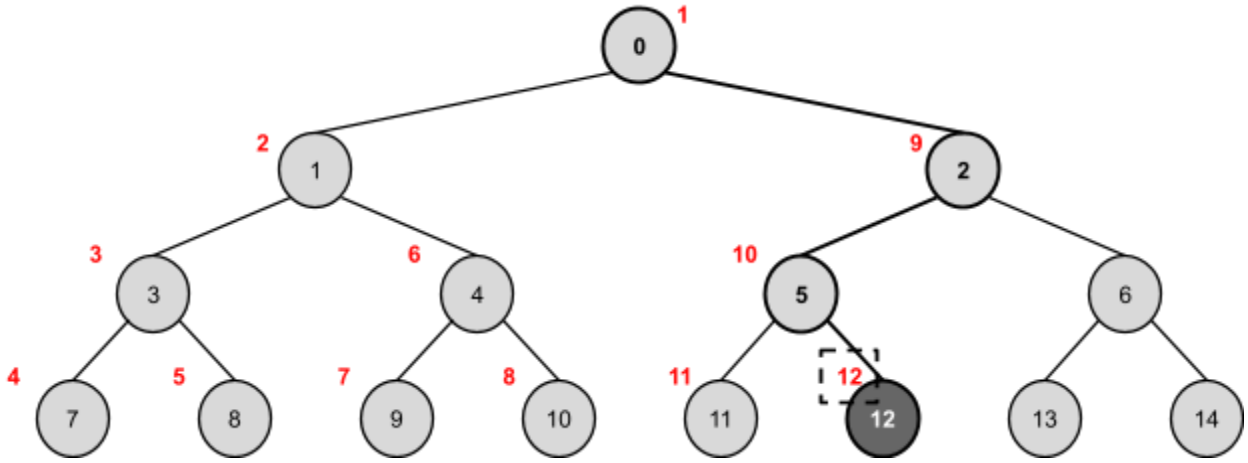
B. Gráficos de Árvore da Busca em Profundidade

a. Árvore de 30 Nodos



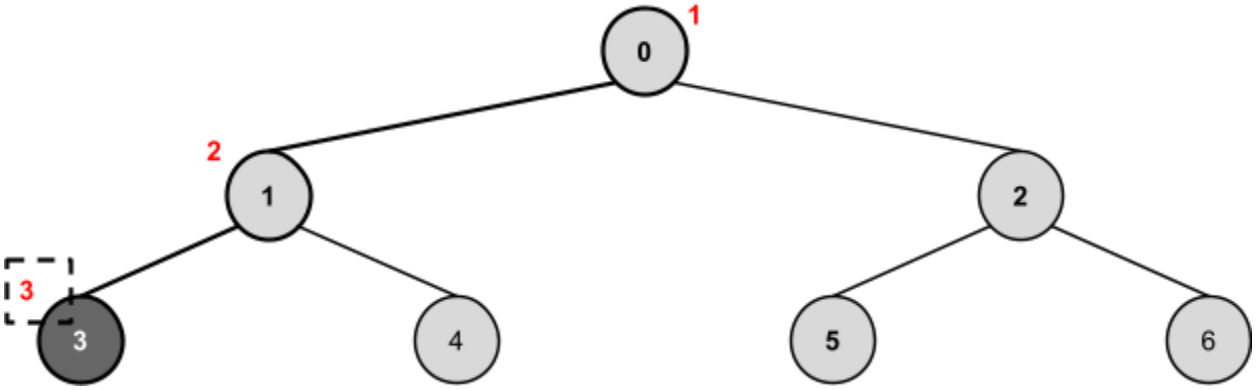
Na busca em profundidade, buscando o nodo de valor 12 na árvore de 30 nodos, o caminho usado para chegar até o valor foi 0,1,3,7,15,16,8,17,18,4,9,19,20,10,21,22,2,5,11,23,24, usando uma distância de 22 nodos.

b. Árvore de 14 Nodos



Na busca em profundidade, buscando o nodo de valor 12 na árvore de 30 nodos, o caminho usado para chegar até o valor foi 0,1,3,7,8,4,9,10,2,5,11,12, usando uma distância de 12 nodos.

c. Árvore de 6 Nodos



Na busca em profundidade, buscando o nodo de valor 3 na árvore de 6 nodos, o caminho usado para chegar até o valor foi 0,1,3, usando uma distância de 3 nodos.

C. Resultados de Tempo

No algoritmo de busca em profundidade nota-se claramente a diferença de tempo com diferentes tamanhos de árvore, tendo 25µs com 6 nodos chegando a 49µs com os 30 nodos.

Busca em Profundidade			
Número de Nodos	6	14	30
1	0.000015	0.000028	0.000046
2	0.000025	0.000046	0.000047
3	0.000034	0.000030	0.000055
Média(s)	**0.000025**	**0.000035**	**0.000049**

Os tempos do algoritmo de busca gulosa não mostram uma grande diferença de tempo com a mudança nos tamanhos de árvore, variando de apenas 6µs para até 8µs.

Busca Gulosa			
Número de Nodos	6	14	30
1	0.000006	0.000007	0.000008
2	0.000006	0.000007	0.000007
3	0.000006	0.000007	0.000008
Média(s)	**0.000006**	**0.000007**	**0.000008**

Esses resultados mostram que, independente do número de nós e até mesmo da posição do nó objetivo, a busca com informação tem um melhor desempenho comparada à busca sem informação.

IV. CONCLUSÃO

Com base nos resultados obtidos, o algoritmo de busca com informação se mostrou mais eficiente em todos os testes, então acaba sendo uma melhor opção em qualquer aplicação. Mas nota-se um maior equilíbrio em árvores maiores, pois com o aumento de nós a busca gulosa teve um aumento quase insignificante no tempo, enquanto a busca em profundidade teve um aumento significativo. Isso mostra que quanto maior a árvore de busca, mais recomendável é o uso de uma busca com informação.