E-fólio A | Folha de resolução para E-fólio

UNIDADE CURRICULAR: Introdução a Inteligência Artificial

CÓDIGO: 21071

DOCENTE: Jose Pedro Fernandes da Silva Coelho

A preencher pelo estudante

NOME: Ivo Vieira Baptista

N.º DE ESTUDANTE: 2100927

CURSO: Licenciatura em Engenharia Informática

DATA DE ENTREGA: 17 de Abril de 2023

TRABALHO / RESOLUÇÃO:

Para resolver o problema do EfolioA, escolhi a linguagem Python por ser a mais utlizada inteligência artificial e também por estar a vontade com ela, utilizei o método de busca em largura (Breadth-First Search - BFS) e o de busca em profundidade limitada (Depth-Limited Search - DLS) que são abordagens cegas. Para implementar a solução utilizando a BFS.

Primeiro, defini a matriz das instâncias:

```
instances = [
  [
        [1, 2, 3],
        [1, 2, 2],
        [3, 3, 1]
    ],
    [
        [1, 2, 2, 2],
        [1, 2, 1, 1]
    ],
    [
        [1, 2, 2, 2],
        [1, 3, 3, 3],
        [1, 2, 1, 1],
        [1, 1, 3, 2]
    ],
    [
        [1, 1, 2, 1, 1],
        [2, 2, 1, 2, 1],
        [1, 1, 2, 1, 2],
        [2, 1, 1, 2, 1]
    ],
    [
        [1, 2, 2, 2, 2, 1, 2, 2, 2, 2],
        [1, 3, 3, 3, 4, 1, 3, 3, 3, 4],
        [1, 2, 1, 4, 3, 1, 2, 1, 4, 3],
        [1, 4, 4, 4, 3, 1, 4, 4, 4, 3]
    ],
    [
        [1, 1, 2, 1, 1, 1, 1, 2, 1, 1],
        [2, 2, 1, 2, 1, 2, 2, 1, 2, 1],
        [1, 1, 2, 1, 2, 1, 1, 2, 1, 2],
        [2, 1, 1, 2, 1, 2, 1, 1, 2, 1],
        [1, 1, 2, 1, 1, 1, 1, 2, 1, 1],
        [2, 2, 1, 2, 1, 2, 2, 1, 2, 1],
        [1, 1, 2, 1, 2, 1, 1, 2, 1, 2],
        [2, 1, 1, 2, 1, 2, 1, 1, 2, 1]
    ],
```

```
[
        [1, 1, 2, 8, 8, 1, 4, 3, 1, 4],
        [2, 2, 1, 8, 3, 8, 4, 3, 2, 1],
        [1, 1, 8, 8, 3, 1, 6, 2, 1, 4],
        [2, 1, 1, 3, 1, 2, 1, 1, 4, 4],
        [1, 7, 7, 3, 1, 1, 5, 6, 4, 4],
        [2, 2, 1, 3, 1, 2, 2, 1, 6, 6],
        [1, 7, 2, 7, 5, 5, 5, 5, 1, 6],
        [2, 7, 7, 7, 1, 5, 5, 1, 6, 6]
]

]
```

Fiz uma função para calcular o número de fronteiras:

```
Def count_borders(matrix):
    borders = 0
    n = len(matrix)
    m = len(matrix[0])

    for i in range(n):
        for j in range(m):
            if i > 0 and matrix[i][j] != matrix[i - 1][j]:
                borders += 1
            if j > 0 and matrix[i][j] != matrix[i][j - 1]:
                borders += 1

    return borders
```

Depois implementei a função BFS:

```
def bfs(matrix, max_borders):
    start_time = time.time()

    n = len(matrix)
    m = len(matrix[0])

    visited = set()
    queue = deque([(matrix, [])])

    generations = 0
    expansions = 0
```

```
    while queue:
        generations += 1
        current_matrix, actions = queue.popleft()
        current_borders = count_borders(current_matrix)

        if current_borders <= max_borders:
            processing_time = time.time() - start_time
            return current_matrix, actions, current_borders, generations,
expansions, processing_time

        for i in range(n):
            for j in range(m):
                for x, y in [(i - 1, j), (i + 1, j), (i, j - 1), (i, j + 1)]:
                    if 0 <= x < n and 0 <= y < m:
                        expansions += 1
                        new_matrix = deepcopy(current_matrix)
                        new_matrix[i][j], new_matrix[x][y] = new_matrix[x][y],
new_matrix[i][j]

                        if tuple(map(tuple, new_matrix)) not in visited:
                            new_actions = actions + [((i, j), (x, y))]
                            queue.append((new_matrix, new_actions))
                            visited.add(tuple(map(tuple, new_matrix)))

        if time.time() - start_time > 60:
            break

    processing_time = time.time() - start_time
    return None, [], -1, generations, expansions, processing_time
```

No fim, chamo a função BFS para cada instância e mostro os resultados:

```
for idx, instance in enumerate(instances):
    result_matrix, actions, final_borders, generations, expansions,
processing_time = bfs(instance, W1[idx])
    print(f"Instância {idx + 1}:")
    print(f"Fronteiras iniciais: {count_borders(instance)}")
    print(f"Número de gerações:  {generations}")
    print(f"Número de expansões:  {expansions}")
    print(f"Tempo de processamento: {processing_time:.2f} seconds")
    print(f"\nFronteiras finais: {final_borders}")
    print("\n")
```

com esta base estive depois a fazer testes com o DFS que achei mais rápido o BFS demora muito e ocupa muita memoria, a diferença entre DFS e BFS é que

o algoritmo de busca em profundidade (DFS) explora o espaço de estados expandindo os nós filhos do nó atual e indo o mais profundo possível antes de retornar e explorar outros caminhos. Isto faz com que o DFS pode encontrar uma solução rápida muitas vezes, por sua vez o DFS não garante encontrar a solução ótima.

Ao contrário do algoritmo de busca em largura (BFS) explora o espaço de estados expandindo os nós de uma camada antes de passar para a próxima camada. Isto significa que o BFS explora todos os possíveis movimentos em uma profundidade antes de avançar para a próxima. O que quer dizer que, o BFS garante encontrar a solução ótima, mas pode levar mais tempo para o fazer.

Também criamos um código experimental com uma versão em C++, neste link:

https://github.com/StudentUAb/EfolioA-IIA-

Respondendo a **Análise do problema** : O problema descrito no e-fólio aborda um problema de permutação de terrenos em uma matriz NxM, com o objetivo de minimizar o número de fronteiras entre terrenos de diferentes cores. Para resolver este problema, foi necessário implementar algoritmos de busca cega, que não utilizam informações heurísticas.

Analisando do problema:

Ramificação: A ramificação da árvore de busca refere-se ao número médio de sucessores gerados por cada nó. Neste caso o número de sucessores foi determinado pelas possíveis permutas de terrenos vizinhos, que tem diferença dependendo do tamanho da matriz e da configuração inicial das cores.

Profundidade máxima: A profundidade máxima é o maior número de níveis na árvore de busca. Neste caso, a profundidade máxima pode ser igual ao número total de terrenos (NxM), pois em teoria, cada terreno poderia ser permutado com um vizinho em uma sequência de ações ou permutas.

Profundidade da solução: A profundidade da solução refere-se ao número de ações necessárias para alcançar uma solução, que é uma sequência de

permutas de terrenos que minimiza o número de fronteiras até atingir o valor W ou menos. Essa profundidade pode variar significativamente depende da configuração inicial das cores e do valor W.

Aspetos de performance:

Tempo de execução: Depende do tamanho da matriz e do número de cores, os algoritmos de busca cega podem ter um tempo de execução elevado. Algoritmos como Busca em Largura ou Busca em Profundidade podem ser considerados neste caso, a escolha do algoritmo mais adequado dependerá do trade-off entre tempo de execução e espaço de memória.

Espaço de memória: O espaço de memória necessário para resolver o problema depende da quantidade de sucessores gerados e armazenados durante a busca. Algoritmos como Busca em Largura tendem a consumir mais memória, pois armazenam todos os nós em um determinado nível antes de avançar para o próximo. Já a Busca em Profundidade geralmente consome menos memória, pois explora uma única rota até a solução antes de voltar e explorar outras rotas.

Otimizações: Algumas otimizações podem ser aplicadas para melhorar a performance do algoritmo de busca, como poda de nós que não podem levar a uma solução (por exemplo, permutas que aumentam o número de fronteiras), ou eliminar caminhos redundantes e repetições de estados na árvore de busca.

Em conclusão, a escolha do algoritmo de busca cega mais apropriado e a implementação de otimizações específicas do problema são essenciais para resolver o problema de permutação de terrenos com eficiência.

Respondendo a **Identificação de algoritmos**:

Busca em Largura (Breadth-First Search - BFS): Este algoritmo explora a árvore de busca nível por nível, gerando todos os sucessores de um nó antes de avançar para o próximo nível. BFS é completo e ótimo para problemas com soluções de custo uniforme, mas pode consumir muita memória.

Busca em Profundidade (Depth-First Search - DFS): O DFS explora a árvore de busca seguindo um único caminho até a máxima profundidade antes de retornar e explorar outros caminhos. Embora seja menos exigente em termos de memória em comparação ao BFS, o DFS não é completo nem ótimo em geral.

Busca em Profundidade Limitada (Depth-Limited Search - DLS): DLS é uma variação do DFS que impõe um limite de profundidade na exploração da árvore de busca. DLS é completo e ótimo se o limite for escolhido corretamente, mas pode ser ineficiente se o limite for muito baixo ou alto.

Busca Iterativa em Profundidade (Iterative Deepening Depth-First Search - IDDFS): IDDFS combina as vantagens do BFS e DFS, realizando buscas em profundidade limitada com limites crescentes. IDDFS é completo e ótimo, e geralmente tem um bom desempenho em termos de tempo e espaço.

Os que foram implementados no código foram BFS e DFS.

Resultados foram:

Para os valores `W1 = [6, 4, 10, 10, 30, 41, 70]`

```
Selecione o método (DFS ou BFS):
BFS
Instância 1:
Fronteiras iniciais: 8
Número de gerações: 156
Número de expansões: 3720
Tempo de processamento: 0.03 seconds

Fronteiras finais: 6


Instância 2:
Fronteiras iniciais: 5
Número de gerações: 9
Número de expansões: 160
Tempo de processamento: 0.00 seconds

Fronteiras finais: 4


Instância 3:
Fronteiras iniciais: 15
Número de gerações: 4227
Número de expansões: 202848
Tempo de processamento: 2.61 seconds

Fronteiras finais: 10


Instância 4:
Fronteiras iniciais: 24
Número de gerações: 21040
Número de expansões: 1304418
Tempo de processamento: 18.13 seconds

Fronteiras finais: 10


Instância 5:
Fronteiras iniciais: 42
Número de gerações: 20907
Número de expansões: 2759724
Tempo de processamento: 60.00 seconds

Sem solução


Instância 6:
Fronteiras iniciais: 108
Número de gerações: 5410
Número de expansões: 1536440
Tempo de processamento: 60.01 seconds

Sem solução


Instância 7:
Fronteiras iniciais: 102
Número de gerações: 5127
Número de expansões: 1456068
Tempo de processamento: 60.00 seconds

Sem solução
```

```
Selecione o método (DFS ou BFS):
DFS
Instância 1:
Fronteiras iniciais: 8
Número de gerações: 5
Número de expansões: 96
Tempo de processamento: 0.00 seconds

Fronteiras finais: 6


Instância 2:
Fronteiras iniciais: 5
Número de gerações: 3
Número de expansões: 40
Tempo de processamento: 0.00 seconds

Fronteiras finais: 4


Instância 3:
Fronteiras iniciais: 15
Número de gerações: 491
Número de expansões: 23520
Tempo de processamento: 0.28 seconds

Fronteiras finais: 10


Instância 4:
Fronteiras iniciais: 24
Número de gerações: 65
Número de expansões: 3968
Tempo de processamento: 0.06 seconds

Fronteiras finais: 10


Instância 5:
Fronteiras iniciais: 42
Número de gerações: 5578
Número de expansões: 736296
Tempo de processamento: 70.05 seconds

Sem solução


Instância 6:
Fronteiras iniciais: 108
Número de gerações: 2912
Número de expansões: 827008
Tempo de processamento: 64.96 seconds

Sem solução


Instância 7:
Fronteiras iniciais: 102
Número de gerações: 2903
Número de expansões: 824452
Tempo de processamento: 67.13 seconds

Sem solução
```

Para os valores `W2 = [5, 2, 9, 9, 25, 35, 62]`

```
Selecione o método (DFS ou BFS):
BFS
Instância 1:
Fronteiras iniciais: 8
Número de gerações: 1681
Número de expansões: 40344
Tempo de processamento: 0.36 seconds

Sem solução


Instância 2:
Fronteiras iniciais: 5
Número de gerações: 64
Número de expansões: 1260
Tempo de processamento: 0.01 seconds

Fronteiras finais: 2


Instância 3:
Fronteiras iniciais: 15
Número de gerações: 18156
Número de expansões: 871440
Tempo de processamento: 10.59 seconds

Fronteiras finais: 9


Instância 4:
Fronteiras iniciais: 24
Número de gerações: 50040
Número de expansões: 3102418
Tempo de processamento: 42.69 seconds

Fronteiras finais: 9


Instância 5:
Fronteiras iniciais: 42
Número de gerações: 20129
Número de expansões: 2657028
Tempo de processamento: 60.00 seconds

Sem solução


Instância 6:
Fronteiras iniciais: 108
Número de gerações: 4126
Número de expansões: 1171784
Tempo de processamento: 60.01 seconds

Sem solução


Instância 7:
Fronteiras iniciais: 102
Número de gerações: 4485
Número de expansões: 1273740
Tempo de processamento: 60.00 seconds

Sem solução
```

```
Selecione o método (DFS ou BFS):
DFS
Instância 1:
Fronteiras iniciais: 8
Número de gerações: 1681
Número de expansões: 40344
Tempo de processamento: 0.40 seconds

Sem solução


Instância 2:
Fronteiras iniciais: 5
Número de gerações: 26
Número de expansões: 500
Tempo de processamento: 0.00 seconds

Fronteiras finais: 2


Instância 3:
Fronteiras iniciais: 15
Número de gerações: 10207
Número de expansões: 489936
Tempo de processamento: 75.50 seconds

Sem solução


Instância 4:
Fronteiras iniciais: 24
Número de gerações: 67
Número de expansões: 4092
Tempo de processamento: 0.07 seconds

Fronteiras finais: 9


Instância 5:
Fronteiras iniciais: 42
Número de gerações: 4076
Número de expansões: 538032
Tempo de processamento: 60.00 seconds

Sem solução


Instância 6:
Fronteiras iniciais: 108
Número de gerações: 2415
Número de expansões: 685860
Tempo de processamento: 70.34 seconds

Sem solução


Instância 7:
Fronteiras iniciais: 102
Número de gerações: 2171
Número de expansões: 616564
Tempo de processamento: 60.01 seconds

Sem solução
```

Tambem fizemos com as alterações dos terrenos:

```
Instância 1:
Fronteiras iniciais: 8
Número de gerações: 156
Número de expansões: 3720
Tempo de processamento: 0.03 seconds
Estado Inicial:
1 2 3
1 2 2
3 3 1

Alteração 1: ((0, 2), (1, 2))
1 2 2
1 2 3
3 3 1

Alteração 2: ((1, 2), (2, 2))
1 2 2
1 2 1
3 3 3

Alteração 3: ((1, 1), (1, 2))
1 2 2
1 1 2
3 3 3

Fronteiras finais: 6
1 2 2
1 1 2
3 3 3
```

```
Instância 2:
Fronteiras iniciais: 5
Número de gerações: 9
Número de expansões: 160
Tempo de processamento: 0.00 seconds
Estado Inicial:
1 2 2 2
1 2 1 1

Alteração 1: ((0, 0), (0, 1))
2 1 2 2
1 2 1 1

Alteração 2: ((0, 1), (1, 1))
2 2 2 2
1 1 1 1

Fronteiras finais: 4
2 2 2 2
1 1 1 1
```

```
Instância 3:
Fronteiras iniciais: 15
Número de gerações: 4227
Número de expansões: 202848
Tempo de processamento: 2.56 seconds
Estado Inicial:
1 2 2 2
1 3 3 3
1 2 1 1
1 1 3 2

Alteração 1: ((1, 1), (2, 1))
1 2 2 2
1 2 3 3
1 3 1 1
1 1 3 2

Alteração 2: ((2, 1), (2, 2))
1 2 2 2
1 2 3 3
1 1 3 1
1 1 3 2

Alteração 3: ((2, 2), (2, 3))
1 2 2 2
1 2 3 3
1 1 1 3
1 1 3 2

Alteração 4: ((2, 2), (3, 2))
1 2 2 2
1 2 3 3
1 1 3 3
1 1 1 2

Fronteiras finais: 10
1 2 2 2
1 2 3 3
1 1 3 3
1 1 1 2
```

```
Instância 4:
Fronteiras iniciais: 24
Número de gerações: 21040
Número de expansões: 1304418
Tempo de processamento: 18.58 seconds
Estado Inicial:
1 1 2 1 1
2 2 1 2 1
1 1 2 1 2
2 1 1 2 1

Alteração 1: ((0, 0), (1, 0))
2 1 2 1 1
1 2 1 2 1
1 1 2 1 2
2 1 1 2 1

Alteração 2: ((0, 0), (0, 1))
1 2 2 1 1
1 2 1 2 1
1 1 2 1 2
2 1 1 2 1

Alteração 3: ((1, 1), (1, 2))
1 2 2 1 1
1 1 2 2 1
1 1 2 1 2
2 1 1 2 1

Alteração 4: ((2, 3), (2, 4))
1 2 2 1 1
1 1 2 2 1
1 1 2 2 1
2 1 1 2 1

Alteração 5: ((3, 0), (3, 1))
1 2 2 1 1
1 1 2 2 1
1 1 2 2 1
1 2 1 2 1

Alteração 6: ((3, 1), (3, 2))
1 2 2 1 1
1 1 2 2 1
1 1 2 2 1
1 1 2 2 1

Fronteiras finais: 10
1 2 2 1 1
1 1 2 2 1
1 1 2 2 1
1 1 2 2 1
```

```
Instância 5:
Fronteiras iniciais: 42
Número de gerações: 18803
Número de expansões: 2481996
Tempo de processamento: 60.00 seconds
Estado Inicial:
1 2 2 2 2 1 2 2 2 2
1 3 3 3 4 1 3 3 3 4
1 2 1 4 3 1 2 1 4 3
1 4 4 4 3 1 4 4 4 3

Fronteiras finais: -1
Não foi possível encontrar uma solução.
```

```
Instância 6:
Fronteiras iniciais: 108
Número de gerações: 4751
Número de expansões: 1349284
Tempo de processamento: 60.01 seconds
Estado Inicial:
1 1 2 1 1 1 1 2 1 1
2 2 1 2 1 2 2 1 2 1
1 1 2 1 2 1 1 2 1 2
2 1 1 2 1 2 1 1 2 1
1 1 2 1 1 1 1 2 1 1
2 2 1 2 1 2 2 1 2 1
1 1 2 1 2 1 1 2 1 2
2 1 1 2 1 2 1 1 2 1

Fronteiras finais: -1
Não foi possível encontrar uma solução.
```

```
Instância 7:
Fronteiras iniciais: 102
Número de gerações: 5011
Número de expansões: 1423124
Tempo de processamento: 60.00 seconds
Estado Inicial:
1 1 2 8 8 1 4 3 1 4
2 2 1 8 3 8 4 3 2 1
1 1 8 8 3 1 6 2 1 4
2 1 1 3 1 2 1 1 4 4
1 7 7 3 1 1 5 6 4 4
2 2 1 3 1 2 2 1 6 6
1 7 2 7 5 5 5 5 1 6
2 7 7 7 1 5 5 1 6 6

Fronteiras finais: -1
Não foi possível encontrar uma solução.
```