CPS740 - Algoritmos e Grafos - Lista 1

Thiago Guimarães Rebello Mendonca de Alcantara - DRE: 118053123

Tudo esta no repositorio https://github.com/guim4dev/CPS740

Questão 1)

- **Algoritmo de Ordenação 1:**

```
def insertion_sort(list):
    for i in range(1, len(list)):
        current_item = list[i]
        j = i - 1
        while current_item < list[j] and j >= 0:
            list[j + 1] = list[j]
            j -= 1
        list[j + 1] = current_item
        print(list)
    return list
```

Passo a passo:
[2, 7, 5, 6, 9, 0, 1, 4, 8, 5, 3]
[2, 5, 7, 6, 9, 0, 1, 4, 8, 5, 3]
[2, 5, 6, 7, 9, 0, 1, 4, 8, 5, 3]
[0, 2, 5, 6, 7, 9, 1, 4, 8, 5, 3]
[0, 1, 2, 5, 6, 7, 9, 4, 8, 5, 3]
[0, 1, 2, 4, 5, 6, 7, 9, 8, 5, 3]
[0, 1, 2, 4, 5, 6, 7, 8, 9, 5, 3]
[0, 1, 2, 4, 5, 5, 6, 7, 8, 9, 3]
[0, 1, 2, 3, 4, 5, 5, 6, 7, 8, 9]

- **Algoritmo de Ordenação 2:**

```
def quick_sort(list):
    if len(list) <= 1:
        return list

    pivot_index = len(list)//2
    pivot = list[pivot_index]
    del list[pivot_index]

    left = []
    right = []

    for item in list:
        if item <= pivot:
            left.append(item)
        else:
            right.append(item)
    print(left + [pivot] + right)
    return quick_sort(left) + [pivot] + quick_sort(right)
```

Passo a passo:
[2, 7, 5, 6, 9, 0, 1, 4, 8, 5, 3]
[0, 2, 7, 5, 6, 9, 1, 4, 8, 5, 3]
[1, 2, 7, 5, 6, 9, 4, 8, 5, 3]
[2, 7, 5, 6, 4, 8, 5, 3, 9]
[2, 3, 4, 7, 5, 6, 8, 5]
[2, 3]
[5, 5, 6, 7, 8]
[5, 5]
[7, 8]
[0, 1, 2, 3, 4, 5, 5, 6, 7, 8, 9]

Questão 2) Matrizes de adjacência possuem complexidade espacial O(n^2), enquanto Estruturas de adjacência possuem O(n + m). Portanto, a Estrutura de adjacências é preferível quando n + m < n^2 -> m < n^2 - n. Logo, Estruturas de adjacências são preferíveis em questão de armazenamento quando o número de arestas é menor que a diferença entre o quadrado do numero de vertices e o numero de vertices.

Questão 3) O Algoritmo é O(m).

```
def possui_ciclo_euleriano(arestas):
    memo = {}
    for aresta in arestas:
        for vertice in aresta:
            memo[vertice] = (memo.get(vertice) or 0) + 1
    for value in memo.values():
        if (value % 2) != 0:
            return False
    return True
```

OBS: ***Função útil que vai ser utilizada a partir de agora:***

```
def get_adjacents(vertice, edges):
    adjacents = []
    for edge in edges:
        if vertice in edges:
            edge.remove(vertice)
            adjacents += edge
    return adjacents
```

Questão 4) Utilizando o algoritmo de bi-coloração para definir se um grafo é bipartido:
Complexidade: O(m+n)

```
colors = {}
def bipartido_color(V, E):
    for vertice in V:
        colors[vertice] = -1 # sem cor

    for vertice in V:
        if colors[vertice] == -1: # tentar pintar
            if not(can_colorize_vertice(vertice, E, 0)):
                return False
    return True

# colors: 1, 0 e -1 (azul, vermelho e sem cor)
def can_colorize_vertice(vertice, edges, color):
    colors[vertice] = color
    adjacents = get_adjacents(vertice, edges) # funcao dos adjacentes
    for adjacent in adjacents:
        if colors[adjacent] == -1:
            if not(can_colorize_vertice(adjacent, edges, 1-color)):
                return False
        else:
            if colors[adjacent] == color:
                return False
    return True
```

Também fiz uma tentativa de um algoritmo de DFS por condição contrária para avaliar se eh bipartido ou nao, este encontra-se aqui: https://github.com/guim4dev/CPS740/blob/master/Lista1/questao_4.py , especificamente na linha 4.

Questão 5) a) É um algoritmo polinomial.

```
def possui_conjunto_independente_k(vertices, arestas, k):
    vertices_colors = colorized_vertices(vertices, arestas)
    independent_one_color = []
    independent_zero_color = []
    for key, value in vertices_colors:
        if value == 1:
            independent_one_color.append(key)
        elif value == 0:
            independent_zero_color.append(key)
    if (len(independent_one_color) >= k) or
(len(independent_zero_color) >= k):
        return True
    return False
```

```
colors = {}
def colorized_vertices(V, E): # funcao para colorir os vertices
    for vertice in V:
        colors[vertice] = -1 # sem cor

    for vertice in V:
        if colors[vertice] == -1: # tentar pintar
            colorize_vertice(vertice, E, 0)
    return colors

# colors: 1, 0 e -1 (azul, vermelho e sem cor)
def colorize_vertice(vertice, edges, color):
    colors[vertice] = color
    adjacents = get_adjacents(vertice, edges)
    for adjacent in adjacents:
        if colors[adjacent] == -1:
            colorize_vertice(adjacent, edges, 1-color)
    return True
```

Questão 5) b) Não é um algoritmo polinomial.

```
def possui_ciclo_hamiltoniano(V,E):
    posible_starts = get_posible_starts(E)#vertices com 2 ou mais arestas
    for start in posible_starts:
        adjacents = get_adjacents(start, E)
        if can_cycle(start, adjacents, V, E):
            return True
    return False

def can_cycle(start, start_adjacents, V, E):
    edges = edges_without_vertice(E, start)
    vertices = V - [start]
    posible_pairs = [(start_adjacents[i], start_adjacents[j]) for i
in range(len(start_adjacents)) for j in range(i+1,
len(start_adjacents))]
    for pair in posible_pairs:
        adjacent_start = pair[0]
        adjacent_end = pair[1]
        if has_hamilton_path(adjacent_start, adjacent_end, vertices,
edges):
            return True
    return False
```

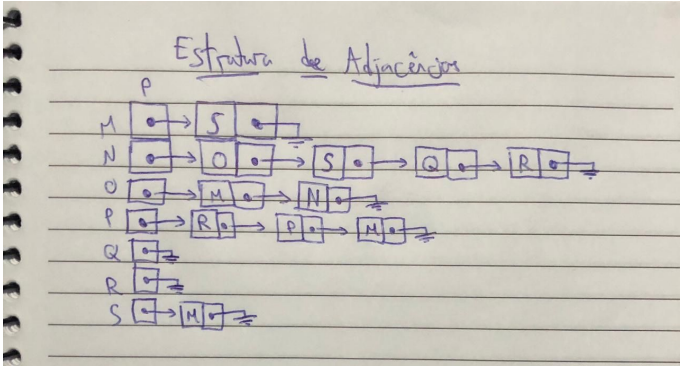
```
def has_hamilton_path(current, end, V, E, visited = []):
    adjacents = get_adjacents(current, V)
    visited.append(current)
    ways = adjacents - visited
    if ways == [end]:
        return True
    ways -= [end]
    for way in ways:
        if has_hamilton_path(way, end, V, E, visited):
            return True
    return False

def edges_without_vertice(E, vertice):
    edges = []
    for edge in E:
        if vertice in edge:
            next
        edges.append(edge)
    return edges

def get_posible_starts(edges):
    posibles = []
    memo = {}
    for edge in edges:
        for vertice in edge:
            memo[vertice] = (memo.get(vertice) or 0) + 1
    for key, value in memo:
        if value > 1:
            posibles.append(key)
    return posibles
```

Questão 6)

- ***a) (M,S)***
- ***b) Não existe***
- ***c) Fracamente Conexo***
- ***d) N: Grau de entrada - 1; Grau de saída - 2***
  R: Grau de entrada - 2 ; Grau de saída - 0

e)

	M	N	O	P	Q	R	S
M	0	0	0	0	0	0	1
N	0	0	1	0	1	1	1
O	1	1	0	0	0	0	0
P	1	0	0	1	0	1	0
Q	0	0	0	0	0	0	0
R	0	0	0	0	0	0	0
S	1	0	0	0	0	0	0



Questão 7)

a) ***O algoritmo calcula a soma de todos os quadrados de números naturais até n***
b) ***O(n)***
c) ***Sugestão O(1) abaixo.***

```
def soma_dos_quadrados(n):
    return n**3/3 + n**2/2 + n/6
```