Algorithm template

April 12, 2021

1 Code

Algorithm 1 Calcula número mínimo de intervenções a fazer para que todos os dominós caiam.

```
procedure getNumberOfInterventions($G$, $topologicalSort$)      ▷ Complexidade: O(V + E)
    let $interventions$ be a new integer
    for each vertex $u \in G.V$ do                                 ▷ Inicialização: O(V)
        $u.color$ = white
    for each vertex $u \in topologicalSort$ do                     ▷ Percorrer os vértices: O(V)
        if $u$.color == white then
            dfs-visit($G$, $u$)                                    ▷ Chamada ao DFS-VISIT: O(E)
            $interventions$++
    return $interventions$
```

Algorithm 2 Calcula tamanho da maior sequência de dominós a cair.

```
procedure getLongestSequence($G$, $topologicalSort$)             ▷ Complexidade: O(V + E)
    let $sequence$ be a new integer
    let $max$ be a new array
    for each vertex $u \in topologicalSort$.reverse do            ▷ Percorrer os vértices: O(V)
        for each vertex $v \in$ Adj[$u$] do                        ▷ Percorrer lista de adjacências: O(E)
            if $aux < max[v]$ then
                $aux = max[v]$
        $max[u] = aux + 1$
        if $sequence < max[u]$ then
            $sequence = max[u]$
    return $sequence$
```

Algorithm 3 Depth-first search.

```
procedure DFS($G$)                                          ▷ Complexidade: O(V + E)
    let topologicalOrder be a new array
    for each vertex $u \in G.V$ do                          ▷ Inicialização: O(V)
        $u.color$ = white
    for each vertex $u \in G.V$ do                          ▷ Percorrer os vértices: O(V)
        if $u$.color == white then
            DFS-VISIT($G$, $u$, topologicalSort)            ▷ Chamada ao DFS-VISIT: O(E)

procedure DFS-VISIT(G, u, topologicalSort)                  ▷ Complexidade: O(E)
    $u.color$ = gray
    let stack be a new stack
    let hasChildren be a new boolean
    stack.push(src)
    while stack not empty do
        hasChildren = false
        for each vertex $v \in$ Adj[stack.top()] do         ▷ Percorrer lista de adjacências: O(E)
            if $v$.color == white then
                hasChildren = true
                stack.push($v$)
                $v.color$ = gray
        $u.color$ = black
        if hasChildren == false then
            while stack.top().color == black do
                topologicalSort.push(stack.top())
                stack.pop()
```