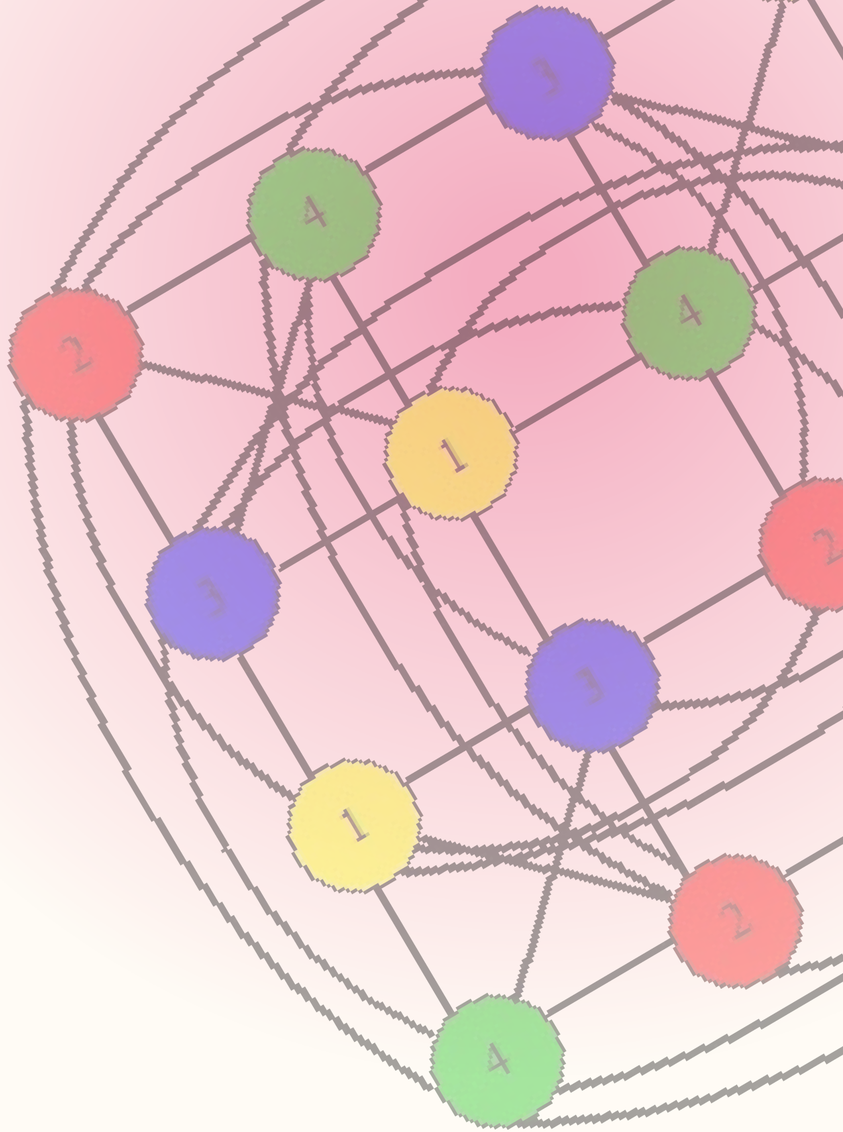
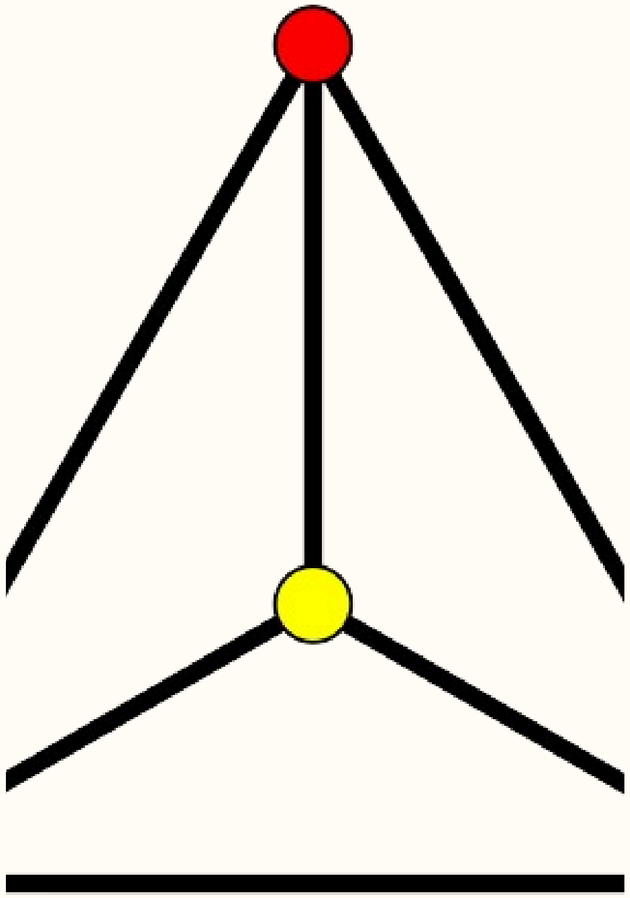
Trabalho M2
Grafos



Planaridade e Coloração

Victor Hugo
Milena Schulz
Vinícius Ritzel

01 - Conceitos



A coloração de grafos é uma atribuição de cores aos vértices de tal forma que vértices adjacentes recebem cores diferentes. O número mínimo de cores necessárias para colorir um grafo é chamado de número cromático do grafo.

Já a planaridade é a propriedade de um grafo onde suas arestas não se cruzam

Ambos conceitos possuem ligação com situações reais, como a criação de um sudoku

Como funciona o algoritmo:

A seguir, são feitas as conexões com os vértices adjacentes

Direita

```
// Cria as conexões com os vertices a direita
for (let xi = i + 1; xi < N; xi++) {
    newEdge = grafoAddConexao(i * N + j, xi * N + j, 'aresta');
    newEdge.from = newEdge.id_origem;
    newEdge.to = newEdge.id_destino;
    interface_vis_edges.add(newEdge);
}
```

Baixo

```
// Cria as conexões com os vertices a baixo
for (let xj = j + 1; xj < N; xj++) {
    newEdge = grafoAddConexao(i * N + j, i * N + xj, 'aresta');
    newEdge.from = newEdge.id_origem;
    newEdge.to = newEdge.id_destino;
    interface_vis_edges.add(newEdge);
}
```

Diagonal primária

```
// Cria as conexões na diagonal baixo-direita (primária)
ci = 1;
for (let xi = (i % sqrtN) + 1; xi < sqrtN; xi++) {
    cj = 1
    for (let xj = (j % sqrtN) + 1; xj < sqrtN; xj++) {
        newEdge = grafoAddConexao(i * N + j, (i + ci) * N + j + cj, 'aresta');
        newEdge.from = newEdge.id_origem;
        newEdge.to = newEdge.id_destino;
        interface_vis_edges.add(newEdge);
        cj++;
    }
    ci++;
}
```

Diagonal secundária

```
// Cria as conexões na diagonal baixo-esquerda (secundária)
ci = 1;
for (let xi = (i % sqrtN) + 1; xi < sqrtN; xi++) {
    cj = 1;
    for (let xj = j % sqrtN; xj > 0; xj--) {
        newEdge = grafoAddConexao(i * N + j, (i + ci) * N + j - cj, 'aresta');
        newEdge.from = newEdge.id_origem;
        newEdge.to = newEdge.id_destino;
        interface_vis_edges.add(newEdge);
        cj++;
    }
    ci++;
}
```

Colore o vértice, garantindo que não há cor repetida nos adjacentes

```
let vertice_inicial = interface_vis_network.getSelectedNodes().map(n => interface_vis_nodes.get(n))[0] ||
    interface_vis_nodes.get()[0];

interface_vis_edges.get().filter(e => e.from == vertice_inicial.id || e.to == vertice_inicial.id).forEach(e => {
    let id = e.from == vertice_inicial.id ? e.to : e.from;
    let vertice_adjacente = interface_vis_nodes.get().filter(n => n.id == id)[0];
    vertice_adjacente.saturacao++;
});

vertice_inicial.color = getColor();
vertice_inicial.colored = true;
cores.push(vertice_inicial.color);

interface_vis_nodes.update(vertice_inicial);

await new Promise(r => setTimeout(r, interface_input_speed.value));
```

Minimiza as cores necessárias

```
while (interface_vis_nodes.get().filter(node => node.colored).length < interface_vis_nodes.get().length) {

    let vertice_atual = interface_vis_nodes.get().filter(n => !n.colored).sort((a, b) => a.saturacao < b.saturacao)[0];
    let cores_disponiveis = cores.slice();

    interface_vis_edges.get().filter(e => e.from == vertice_atual.id || e.to == vertice_atual.id).forEach(e => {
        let id = e.from == vertice_atual.id ? e.to : e.from;
        let vertice_adjacente = interface_vis_nodes.get().filter(n => n.id == id)[0];
        if (vertice_adjacente.colored && cores_disponiveis.includes(vertice_adjacente.color)) {
            cores_disponiveis.splice(cores_disponiveis.indexOf(vertice_adjacente.color), 1);
        }

        vertice_adjacente.saturacao++;
    });


    if (cores_disponiveis.length >= 1) {
        vertice_atual.color = cores_disponiveis[0];
    } else {
        vertice_atual.color = getColor();
        cores.push(vertice_atual.color);
    }

    vertice_atual.colored = true;

    interface_vis_nodes.update(vertice_atual);

    await new Promise(r => setTimeout(r, interface_input_speed.value));
}
```