Projeto Final:
Problemas de Grafos

Antônio Joabe Alves Morais
Iarley Natã Lopes Souza

1 [Problema 1] Colorindo os vértices do grafo com duas cores

1.1 Descrição da solução

Na solução, o primeiro vértice (v_0) no grafo é pintado de vermelho, portanto todos os vizinhos deverão ser pintados com a cor contrária: Azul.

A partir disto, forma-se um efeito cascata: Ao analizarmos um vértice v, verificamos seus vizinhos. Se um vizinho u for branco, devemos pintá-lo com a cor contrária de v, caso contrário (se u não for branco), comparamos as cores de v e u.

Se v e u possuírem cores diferentes, a execução do algoritmo irá prosseguir normalmente. Se possuírem cores iguais, o algoritmo será interrompido, assim indicando que não é possível bipartir o grafo.

1.2 Complexidade

A complexidade será dada, de acordo com a notação *Big O*, por:

$$O(V + E)$$

A justificativa se dá pelo fato do algoritmo visitar todos os vértices e arestas, já que precisamos verificar a cor de todos eles, a fim de saber se um grafo é bipartido ou não.

A notação, portanto, irá indicar que a complexidade de tempo de execução cresce de acordo com a soma desses dois termos, vértices (V) e arestas (E), de forma diretamente proporcional.

2 [Problema 2] Seis graus de Kevin Bacon

2.1 Descrição da solução

Neste problema, é necessária apenas a aplicação da *BFS* (Breadth-First Search), já que o algoritmo já conta com o armazenamento das distâncias de todos os vértices dado uma origem, neste caso, Kevin Bacon.

Portanto, dado um grafo não-direcionado, a *BFS* irá fornecer os números de Bacon que o problema demanda por meio das distâncias armazenadas na execução do algoritmo.

2.2 Complexidade

Essa solução exige um tempo de execução:

$$O(V + E \log V)$$

Analizando o algoritmo de busca de forma agregada, temos que o tempo de inicialização onde se visita os vértices se dá por $O(V)$, assim como o tempo gasto com a fila.

Ao percorrer as adjacências (arestas) - por causa da estrutura usada (`map`), que usa $O(\log V)$ para realizar operações de inserção, busca e remoção - será exigido tempo $O(E \log V)$.

Portanto, a análise de complexidade será de $O(V + E \log V)$.

3 [Problema 3] Vias de mão dupla

3.1 Descrição da solução

A solução se divide em duas etapas:

1. Identificar as arestas que são pontes: Através de uma *DFS* modificada, são encontradas as arestas que são pontes, que são depois removidas, dividindo o grafo em componentes desconectados. Essas arestas são armazenadas em um vetor de pares (u, v).

2. Direcionamento das arestas não-pontes: Em seguida, uma nova *DFS* é realizada para direcionar as arestas restantes. Para cada vértice, como o grafo ainda é não-direcionado, o algoritmo irá retirar, digamos, a aresta (v, u), deixando a aresta (u, v), onde u é o vértice analisado e v é um vizinho de u.

   Isso é feito para cada vértice, e a *DFS* é executada novamente.

   Quando um vértice não possuir mais vizinhos ainda não visitados, o tempo de descoberta do mesmo é comparado com o do seu vizinho: se for maior, a mão dupla (v, u) será removida.

Depois disso, as arestas que são pontes são adicionadas novamente ao grafo, e o mesmo é ordenado e impresso.

3.2 Complexidade

A solução tem complexidade:

$$O((V + E) + V \log V)$$

Ambas as *DFS*'s têm complexidades $O(V + E)$, já são algoritmos de busca em profundidade, com alterações que não modificam a complexidade. A ordenação tem complexidade $O(V \log V)$, que ocorre para ordenar os vizinhos na lista de adjacência.

Portanto, a complexidade total é $O((V + E) + V \log V)$.