EDA - Projeto Final

Ezequiel dos Santos Melo, Gustavo Fernandes de Barros

7 de julho de 2023

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

Meus primeiros pensamentos sobre a resolução foram de certa forma intuitivos, visto que, tratando-se somente de duas cores, talvez fosse possível obter a resposta testando, ou seja, colorindo cada vértice do grafo para verificar se o mesmo pode ser colorido com duas cores ou não.

Após pesquisar sobre o problema da χ-coloração de vértices, descobri uma relação direta entre a veracidade desta propriedade para duas cores e grafos bipartidos, onde há um teorema no qual um grafo pode ser colorido com duas cores **se e somente se** ele for bipartido. Desta forma, é possível aproveitar-se deste teorema para resolver a questão.

Uma forma algorítmica de resolver essa questão é usando busca em largura para descobrir se um grafo é bipartido é de fato verificando se é possível colorir o mesmo com duas cores de forma que vértices adjacentes tenham cores distintas, percorrendo todos os vértices do grafo, colorindo-os e verificando a cor de seus vizinhos. Caso haja em algum momento vizinhos de cores semelhantes, logo conclui-se que o grafo não é bipartido, portanto não é possível que o mesmo seja colorido com duas cores.

```
bool Bicoloration(const Graph& graph) {
    int number_of_vertices = graph.get_number_of_vertices();

    vector<char> colors(number_of_vertices, ' ');
    vector<bool> visited_vertices(number_of_vertices, false);

    for(int i = 0; i < number_of_vertices; i++) {
        if(!visited_vertices[i]) {
            queue<int> queue_of_vertices;
            queue_of_vertices.push(i);
            colors[i] = 'R';

            while(!queue_of_vertices.empty()) {
                int current_vertice = queue_of_vertices.front();
                queue_of_vertices.pop();
                visited_vertices[current_vertice] = true;

                for(const auto& edge : graph.neighbors(current_vertice)) {
                    int neighbor = edge.get_destiny();

                    if(colors[neighbor] == ' ') {
                        if(colors[current_vertice] == 'R') colors[neighbor] = 'B';

                        else if(colors[current_vertice] == 'B') colors[neighbor] = 'R';

                        queue_of_vertices.push(neighbor);
                    }

                    else if(colors[neighbor] == colors[current_vertice]) return false;
```

```
                    }
                }
            }
        }

        return true;
}
```

Analisando a complexidade, temos um loop que percorre todos os vértices do grafo, sendo O(V), onde V é o número de vértices. A inicialização dos vetores auxiliares são constantes, portanto não influenciam na complexidade, enquanto o loop mais interno a função pode percorrer todas as arestas do grafo, além de realizar algumas verificações acerca dos vizinhos do vértice atual, podendo atingir uma complexidade O(A), onde A é o número de arestas. Logo, pode-se dizer que a complexidade desse algoritmo é O(V + A).

2 [Problema 2] Seis graus de Kevin Bacon

3 [Problema 3] Vias de mão dupla

Referências