GRAPHS

Alunos:
Guilherme Willian Saraiva da Hora - 475980
Samuel Evangelista de Aquino Júnior - 485374

Sumário

1 Questão 1 **2**

2 Questão 2 **2**

3 Bibliografia **3**

1 Questão 1

O Problema da k-coloração de vértices de um grafo, é um problema bastante conhecido na área de estudo sobre Grafos. Na questão, foi proposto determinar se existe ou não uma coloração de vértices válida de um grafo com duas cores. Isso nos diz que se um grafo possuir uma coloração de vértices válida, dado dois vertices adjacente eles possuem cores diferentes.

Para desenvolver um algoritmo que nos permita avaliar se existe ou não essa tal coloração de vértices válida, foi utilizada a BFS(Busca em Largura). A BFS foi utilizada sobre um grafo implementado com Lista de Adjacência. A BFS nos permite realizar uma busca sistemática sobre todos os vértices do grafo.

O algoritmo consiste basicamente em pegar um vértice que está em uma estrutura auxiliar(fila) e percorrer a sua lista de adjacência. No momento que está sendo percorrida a lista de adjacência do vértice, é preciso verificar a cor do elemento, onde o elemento é um vértice pertencente a lista de adjacência do vértice retirado da fila. Existem 3 valores de cores possíveis: RED, BLACK e GRAY(Se não foi visitado ainda). Se o elemento tiver cor igual a GRAY, significa que ele não foi visitado ainda e nós podemos colorir ele com a cor contrária a cor do vértice. Se o elemento já possuir uma cor oposta a cor do vértice, nada deve ser feito pois ele já foi visitado. Caso o elemento tenha a cor igual a do vértice, significa que existe dois vértices adjancentes com cores iguais, e isso nos informa que o grafo não possui um coloração de vértices válida. Se ao terminar essa busca por todo o grafo e nenhum vértice cair nesse último caso comentando, significa que existe uma coloração de vértices válida. Essa determinada busca também realiza a coloração de grafos que não são conexos. Se após percorrer toda uma componente do grafo ainda existir outras componentes que não foram coloridas, o algoritmo implementado estende as suas buscas a essas componentes. Somente as cores RED e BLACK são usadas para colorir os vértices do grafo, a cor GRAY é utilizada apenas para saber quais vértices já foram visitados. No algoritmo, as cores relacionadas a cada vértice são armazenadas em uma estrutura auxiliar do tipo *vector*, onde cada posiçao está relacionada a um vértice de. Todo esse processo é realizado pelo método, disponibilizado na implementação, $boolBFS(Grafog)$.

Após a realização do método anterior sobre o grafo, existem dois retornos possíveis: *true* ou $false$. É retornado $false$ se não existir uma coloração válida e é impresso no arquivo de saída o valor "NAO". Se for retornado *true*, será impresso o valor "SIM"e também cada vértice acompanhado de sua respectiva cor.

A complexidade do algoritmo em questão está intimamente ligada a complexidade do algoritmo da Busca em Largura. No pior caso, será percorrido todos os vértices e suas respectivas listas de adjacência, isso nos garante a complexidade de tempo $O(|V| + |A|)$. A complexidade de memória é na ordem de $O(n^2)$.

2 Questão 2

Acreditamos que o problema relacionado a questão II é o de $Maximum\ Independent\ Vertex\ Set$ onde temos que encontrar um conjunto de vertices indepedente onde tal conjunto é o maximo possível. Logo para resolvermos esse problema teremos que ter algumas flags de status para salvar alguns dados do filhos, onde será interessante saber as flags se o resultado é livre ou nao, e se há mais de uma possibilidade ou não.

Logo para calcularmos o conjunto máximo iremos fazer com base no caso a caso de cada nó, onde cada nó pode ser pegue ou não, e essas informações são passados para os nós subsequente pela recursão da

DFS(Busca em profundidade). Para calcularmos se há mais de uma possibilidade ou não, partimos do caso em que cada nó pode ter duas possibilidade, ser escolhido ou não, então se algum nó mais abaixo tem mais de uma possibilidade de ser escolhidos, ja temos que é verdade que os nós acima também tem mais de uma possibilidade.

Para passarmos as flags de status para cima da recursão precisamos criar uma struct ja que são 4(quatro) flags de retorno, assim verificando todas as possibilidades. Com um algoritmo guloso onde verificamos um nível sim e outro não, e fazendo isso para o principal e o nível subsequente não tem como obter a solução otima do problema, por isso temos que adicionar as flags e salvar os status como em uma programção dinâmica onde verificamos todos casos.

Para tratarmos a string no grafo, fizemos uma mapeamento com a estrutura map do C++, onde cada nome tem um valor(ID) relacionado e o grafo é criado com o valor relacionado de cada chefe ou funcionário. Após criado o grafo passamos para a DFS ele e vemos qual o valor maximo e se tem mais de uma possiblidade ou não, os resultados ficam salvos na struct de retorno.

Acreditamos que a complexidade em relação ao tempo é $O(n)$ onde n é o numero de vértices, e a complexidade em relação ao armazenamento $O(n * m)$ para cada nó n na lista de adjacências podemos ter m vizinhos a ele.

3 Bibliografia

Graph Coloring — Set 2 (Greedy Algorithm), GeeksforGeeks, 2018. Disponível em: <https://www.geeksforgeeks
coloring-set-2-greedy-algorithm/>.

Graph Coloring — Set 1 (Introduction and Applications), GeeksforGeeks, 2018. Disponível em: <https://www.geeksforgeeks.org/graph-coloring-applications/>.

Busca em largura, IME USP, 2019. Disponível em: https://www.ime.usp.br/ pf/algoritmos$_p ara_g rafos/aulas/bf$

Maximal Independent Set in an Undirected Graph. Disponível em: <https://www.geeksforgeeks.org/maximal-independent-set-in-an-undirected-graph/>.

Maximum Independent Set in Bipartite Graphs. Disponível em: <https://ali-ibrahim137.github.io/competitive/independent-set-in-bipartite-graphs.html>.