1 Introdução

Nessa documentação é explicado o algoritmo proposto para resolução do problema no gerenciamento de voos durante a pandemia de uma companhia aérea. Devido às implicações causadas pela pandemia, diversas rotas se tornaram inviáveis ou apenas unidirecionais. Entretanto, e facultada a criacao de rotas adicionais, que tem um alto custo adicional para a empresa. Sendo assim, a companhia deseja saber qual o numero mınimo de rotas que devera adicionar a sua rede aerea para fazer com que um dado aeroporto de origem possa atingir qualquer aeroporto de destino

2 Execução do Código

2.1 Compilação e Execução

Pré-requisitos:

- Compilador moderno de C++(e.g: g++ 10.x.x)
- Programa *GNU Make* instalado (altamente recomendado)

Na pasta raiz do projeto, onde se localiza o *Makefile*, execute o comando *make* e então um arquivo chamado ***tp02*** será gerado no diretório */bin*, que também está na raiz do projeto.

```
$ make
$ cd bin
$ ./tp02 < entrada.txt
```

3 Modelagem Computacional

Para começar a tratar o problema, foi necessário analisar quais relações e informações poderíamos extrair das informações que já temos inicialmente(aeroportos e conexões existentes) e como poderíamos fazer uso disso para chegar em uma solução.

É natural imaginar inicialmente que os aeroportos são nós em um grafo direcionado G em que as arestas representam vôos existentes. Então, também, sabemos que alguns aeroportos desse grafo formam ciclos com outros aeroportos e são mutuamente alcançáveis. Esses são subgrafos fortemente conectados, ou componentes fortemente conectados do nosso grafo inicial.

Sabendo que queremos o mínimo de conexões adicionadas ao grafo inicial para que todo o grafo seja fortemente conectado, podemos pegar os componentes fortemente conectados e considerá-los como um único nó de um novo grafo H formado apenas pelos componentes fortemente conectados, onde as arestas representam conexões entre aeroportos de comnponentes diferentes. É evidente que esse novo grafo com os componentes fortemente conectados condensados é um grafo acíclico direcionado.

Sabendo que no grafo E cada nó é um componente fortemente conectado, ou seja, em que qualquer par de nós contidos ali é mutuamente alcançável, adicionar uma aresta de um nó para outro já faria com que todos os nós em G que pertencem ao nó de origem no grafo E conseguissem alcançar qualquer nó do componente que recebe a aresta. Basta resolvermos o mínimo de arestas necessárias para que esse grafo condensado E se torne fortemente conectado para que cheguemos à resposta do problema inicial.

Por fim, para resolver esse problema no grafo condensado, vamos chamar os nós que apenas recebem arestas de *sinks* e os nós que apenas tem arestas "saindo"de *sources*. Depois de realizar essa contagem, o resultado é dado por $max(qntdSources, qntdSinks)$, pois, de maneira intuitiva, caso haja mais *sources* precisamos de uma aresta entrando em cada um deles e no caso de haver mais *sinks* precisamos de uma aresta saindo de cada um deles para que possam se conectar ao componente fortemente conectado(estamos garantindo que nenhum nó é um nó sem entrada ou sem saída).

Algorithm 1

Mínimo Número de Arestas para Grafo Fortemente Conectado(G):
1- Encontre os componentes fortemente conectados CFF de G
2- Considere cada CFF um nó de um novo grafo direcionado e crie arestas para cada componente que tem arestas conectando-o à outro componente.
3- Nesse novo grafo, conte todos os nós que não são origem de nenhuma aresta(*sinks*) e os que não recebem nenhuma aresta(*sources*).
4- O resultado é o valor máximo dessas duas contagens:
return max(contagemSources, contagemSinks)

4 Análise de Complexidade Assintótica

A ênfase da análise será no algoritmo central proposto, que consiste, basicamente, da execução do algoritmo de Kosaraju para encontrar os componentes fortemente conectados, contagem do grau de entrada e de saída de cada nó do grafo condensado e, por fim, contagem do número de *sources* e *sinks* no mesmo para encontrar o resultado.

Para o primeiro passo, que é obter os componentes fortemente conectados, foi utilizado o algoritmo de Kosaraju, que tem complexidade assintótica temporal de $O(|V| + |E|)$. Uma economia de tempo é feita no momento de construção do grafo dos aeroportos pois, enquanto lemos a entrada e criamos esse grafo, o grafo com arestas invertidas, que é necessário para o algoritmo de Kosaraju, é criado simultaneamente.

O passo de contagem do grau de entrada e de saída de cada nó do novo grafo condensado consiste em iterar por cada um dos componentes fortemente conectados e verificar se algum dos seus nós tem arestas para outros nós que não pertencem ao mesmo componente. Logo, essa operação cresce proporcionalmente e é dominada pelo número de arestas presentes no grafo original, pois para cada nó adjacente, precisamos se ele está no mesmo componente. Fazendo uso de um mapa, foi possível tornar essa operação de verificação do pertencimento ou não de um nó ao componente em tempo constante, e então temos a complexidade assintótica temporal de $O(E)$ para a contagem dos graus de entrada e saída dos componentes fortemente conectados, onde E representa a cardinalidade do conjunto de arestas.

O último passo consiste em iterar pelos componentes do grafo condensado e cada um dos nós que faz parte deles para verificar se o grau de entrada e ou saída do componente é igual à zero, para ser considerado um *source* e ou *sink*. Como estamos basicamente iterando em todos os aeroportos e verificando se eles tem conexões para aeroportos fora do seu componente fortemente conectado, a complexidade temporal assintótica é $O(V)$, onde V representa a cardinalidade do conjunto de aeroportos.

Tendo isso em vista, foi possível propor um algoritmo com complexidade assintótica temporal $O(V + E)$

$$f_1 = O(V + E), f_2 = O(E), f_3 = O(V) \longrightarrow f_1 + f_2 + f_3 = O(V + E)$$