# Documentação do Trabalho Prático II da disciplina de Algoritmos I

**Cecília Kind - 2019054420**

## 1. Introdução:

Esta documentação se refere ao Trabalho Prático II da disciplina de Algoritmos I. A proposta do trabalho é desenvolver um programa para auxiliar no gerenciamento de voos de uma companhia aérea. O trabalho foi implementado em C + + seguindo os princípios da programação orientada a objetos. A solução do problema utilizou os conceitos de componentes fortemente conectados e o algoritmo de Kosaraju foi determinante na resolução. As demais seções presentes neste documento abordam a modelagem do programa (seção 2), as estruturas de dados e algoritmos (seção 3) e a complexidade assintótica dos algoritmos (seção 4).

## 2. Modelagem:

O problema foi modelado por meio de um grafo direcionado sem pesos em que cada vértice representa um aeroporto e uma aresta de um vértice P para um vértice Q indica que há um voo direto de P para Q.

O problema consiste em determinar o número mínimo de rotas que precisam ser adicionadas a uma malha aérea para permitir que a partir de um aeroporto qualquer consiga atingir todos os demais. No contexto de grafos, o problema consiste em determinar o número mínimo de arestas que precisam ser adicionadas para que o grafo seja fortemente conectado.

A solução do problema utiliza o algoritmo de Kosaraju para determinar o número de componentes fortemente conectados no grafo. A partir disso, cada componente passa a ser interpretado como um único vértice, e as arestas que conectam componentes distintos são mantidas. Finalmente, é necessário determinar quantas arestas são necessárias adicionar para tornar esse novo grafo fortemente conectado. Para isso, é considerado o número de vértices nesse novo grafo que não possuem nenhuma aresta saindo deles, e o número de vértices que não possuem nenhuma aresta dirigindo-se a eles. O resultado final é o número maior entre as duas somas, pois indica o número mínimo de arestas que devem ser acrescentadas para formar um ciclo que engloba todos os vértices do grafo, tornando-o fortemente conectado.

## 3. Estruturas de dados e algoritmos

- As principais estruturas utilizadas
  lista de adjacência - vector<int> g[n]: utilizada para armazenar o grafo original
  lista de adjacência - vector<int> g_reverso[n]: utilizada para armazenar o grafo transposto.
  Vetores auxiliares - explored, início, fim, antecessor: utilizados para auxiliar a busca em largura, registrando os nós que foram visitados, o tempo de início e término e os antecessores de cada vértice.
  Vetor - component: utilizado para armazenar em cada vértice original o índice do componente a que ele pertence.

Vetores auxiliares - count_out_edges e count_in_edges: utilizados para fazer a contagem do número de arestas entrando e saindo de cada vértice dos componentes.

- Algoritmos principais:

**DFS1 (vértice origem, grafo, vetores auxiliares)**
Seleciona um vértice V ainda não visitado
    Para cada vértice W alcançado diretamente de V ainda não visitado:
        Incrementa o tempo em uma unidade
        Marca como explorado
        Marca V como antecessor
        Determina o tempo inicial com o valor de tempo
        Chama a DFS para W
    Incrementa o tempo em uma unidade
    Determina o tempo final de V com o valor de tempo
    Marca V como explorado

**DFS2 (vértice origem, grafo, índice do componente, vetor de componentes)**
Seleciona um vértice V ainda não visitado na ordem de maior tempo de término:
    Para cada vértice W alcançado diretamente de V ainda não visitado:
        Marca no vetor de índices na posição de W o índice do componente
        Chama a DFS para W
    Incrementa o tempo em uma unidade
    Marca no vetor de índices na posição de V o índice do componente

**Kosaraju (grafo, número de vértices, estruturas auxiliares):**
Chama BFS1 até que todos os vértices sejam explorados
Inverte o grafo
Ordena os vértice na ordem de maior tempo de término da BFS
Chama BFS2 até que todos os vértices sejam explorados

**count_needed_edges( grafo, número de componentes, vetor de componentes)**
Para cada vértice V:
    Para cada vértice W alcançado diretamente a partir de V:
        Se V e W não possuem o mesmo índice no vetor de componentes:
            Incrementa em 1 a conta de arestas saindo de V
            Incrementa em 1 a conta de arestas entrando em W

Conta o número de zeros no vetor que armazena o número de arestas direcionadas a cada vértice.
Conta o número de zeros no vetor que armazena o número de arestas saindo de cada vértice.
O maior entre os dois números é o resultado.

- Corretude do algoritmo:

  Como visto em sala, o algoritmo de Kosaraju mapeia corretamente os componentes fortemente conectados em um grafo. Pela definição de um grafo fortemente conectado, é necessário que cada vértice possua pelo menos uma aresta de entrada e uma aresta de saída. Considerando que os componentes encontrados pelo algoritmo de Kosaraju são fortemente conectados, para garantir que todo o grafo seja fortemente conectado deve-se garantir que há uma aresta saindo e uma entrando em cada componente encontrado anteriormente. Assim, o algoritmo implementado conta o número de componentes que não possui arestas entrando, o número de componentes que não possui arestas saindo, e define como resultado o maior entre os dois números calculados. O maior número é utilizado pois ao se acrescentar uma aresta saindo de um componente, consequentemente aumenta o grau de entrada de outro componente. Vale lembrar que este resultado indica apenas o número necessário de arestas, mas não indica como inserir as arestas para que o resultado seja alcançado.

4. **Análise de complexidade:**

   Seja N o número de vértices e M o número de arestas:

   **Complexidade de espaço:**
   Principais estruturas:

- Lista de adjacência para representar o grafo e o grafo transposto: O(M+N)

- Vetores auxiliares: O(n) - todos os vetores auxiliares tem tamanho igual ao número de vértices.

   **Complexidade de tempo:**
   Principais funções:

- first_dfs e second_dfs: os dois algoritmos implementam a DFS, que possui complexidade O(M+N)
- Kosaraju: o algoritmo de Kosaraju aplica o DFS duas vezes O(M+N) e inverte o grafo O(M) uma vez.
- count_needed_edges: Conta as arestas externas O(M) e verifica os vértices que não possuem arestas saindo ou entrando O(N).
- main: Realiza a chamada das demais funções: O(M+N)

5. **Conclusão:**
   Este trabalho teve como base os conceitos do algoritmo de Kosaraju visto em aula. A partir dos conhecimentos de grafos direcionados e componentes fortemente conectados foi possível desenvolver uma solução simples para o problema de rotas aéreas proposto. A complexidade de tempo do programa é O(M+N) e de espaço O(M+N).