TRABALHO 1
DCC059 - TEORIA DOS GRAFOS

Revisado por Professor Gabriel Souza

Daniel Alves Thielmann - 202165020AB
Gabriel Duque Schiffner Oliveira - 201965033AB
Kauê Oliveira Paraízo Garcia - 202265517B
Michel Gomes de Andrade - 201876037
Pedro Paulo Paiva Amaral - 202235017

Codigos_do_Daniel

Funções Gerais

{

Grafo // Classe Inicial

eh_Bipartido // Verifica se é direcionado

n_conexo // Verifica quantidade de elementos conectados

get_grau // método para calcular o grau de um vértice

}

Funções Gerais

// Construtor
Grafo::Grafo(int V)
// Destrutor
Grafo::~Grafo()

eh_Bipartido:
Descrição: Determina se o grafo é bipartido utilizando um algoritmo baseado em BFS (Busca em Largura). Cada nó recebe uma cor alternada, garantindo que dois vértices adjacentes nunca tenham a mesma cor.

nConexo:
Descrição: Usa BFS para identificar todas as componentes conexas do grafo. Marca os vértices visitados e incrementa um contador sempre que uma nova busca é iniciada em um vértice não visitado.

getGrau:
Descrição: Calcula o grau de um vértice contando o número de arestas conectadas a ele. A implementação acessa a matriz de adjacência para somar as conexões.

Codigos_do_Gabriel

Funções Gerais

{

Grafo Lista

Aresta_ponderada // Verifica tem peso

Eh_completo // Verifica se todos se conectam entre si

Eh_arvore

}

Codigos_do_Kaue

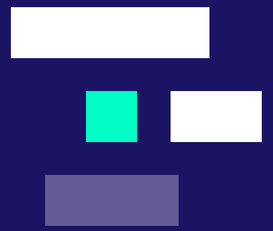
Funções Gerais

```
int get_ordem(const GrafoLista& grafo)  {
  const ListaV& vertices = grafo.getVertices();
  return vertices.tamanho();
}

int eh_direcionado(const GrafoLista& grafo) {
  return grafo.ehDirecionado();
}

int get_ordem(const GrafoLista& grafo) {
  return grafo.vertice_ponderado();
}
```

Funções Grafo Matriz



int get_ordem()

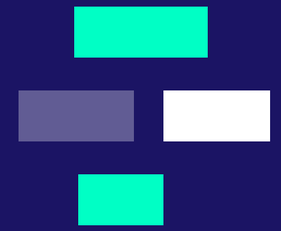
bool ehArvore()

bool temCiclo()

bool ehCompleto()

bool eh_conexo()

void dfsConexao(int vertice, bool *verticeVerificado)



Estrutura Grafo (Pai) e Matriz (Filho)

Grafo

// Construtor

Grafo(int numVertices, bool ponderadoVertices, bool ponderadosArestas, bool direcionado)

// Funções

virtual void imprime_grafo()

virtual bool ehConexo()

virtual bool ehArvore()

virtual bool ehCompleto()

Grafo Matriz : public Grafo

// Construtor

grafo_matriz(int numVertices, bool ponderadoVertices, bool ponderadosArestas, bool direcionado, int pesoVerticesices[ ])

// Funções

void imprime_grafo() override

bool ehConexo() override

bool ehArvore() override

bool ehCompleto() override

Codigos_do_Michel

Funções Gerais

```
// Construtor
Grafo::Grafo(int V)

// Destrutor
Grafo::~Grafo()

// Adiciona arestas
void Grafo::adicionarAresta(int v, int w)

// Verifica se há nó de articulação
bool Grafo::possuiArticulacao()

// Função auxiliar por DFS
bool Grafo::possuiArticulacaoUtil(int v, bool visitado[], int visit[],
int low[], int parent[ ])
```

Possui_Articulação

Grafo {

int V // Número de vértices

int ** adj // Matriz de adjacência

}

Funções Gerais

```
// Construtor
PossuiPonte::PossuiPonte()

// Adiciona arestas
void PossuiPonte::adicionarAresta(int u, int v)

// Verifica se há ponte
bool PossuiPonte::verificarPonte(int n)

// Função auxiliar por DFS
void PossuiPonte::dfs(int u, int parent, int n)
```

Possui_Ponte

```
const int MAX = 100 // Tamanho máximo do grafo

int tempo // Tempo de descoberta

bool temPonte // Indica se há ponte

int adj[MAX][MAX] // Matriz de adjacência

bool visited[MAX] // Array de nós visitados

int disc[MAX] // Array de tempo de descoberta

int low[MAX] // Array do menor tempo alcançável
```

```
// Grafo

int num_vertices // Número de vértices

Aresta ** adjacencias // Matriz de adjacências

// Construtor
Grafo::Grafo(int vertices)

// Destrutor
Grafo::~Grafo()

// Adiciona arestas
void Grafo::adicionar_aresta(int origem, int destino, int peso)

// Exibe a representação
void Grafo::exibir_grafo()

// Carregar por arquivo
Grafo carrega_grafo(const string& nome_arquivo)
```

Funções Gerais

Carrega_Arquivo

```
struct Aresta {

int origem

int destino

int peso

}
```

Codigos_do_Pedro

Funções Gerais

```
// Construtor
grafo_matriz::grafo_matriz(...)

// Destrutor
grafo_matriz::~grafo_matriz()

// Adiciona arestas
void grafo_matriz::adiciona_aresta(int a, int b, int c)

// função auxiliar para adicionar_aresta em grafo não direcionados
int grafo_matriz::tam_lista(int a)

// função imprime grafo_matriz usado em testes
void grafo_matriz::imprime_grafo()
```

```
grafo_matriz {

 int * pesoVerticesices; //

int ** grafo;
}
```

OBRIGADO!

https://github.com/Daniel-Thielmann/CPP-Teoria-Dos-Grafos.git