Pesquisa sobre Grafos

Alunos envolvidos:
1. **RUBENS NETO MARTINS SUAREZ;**
2. **SARAH LAVYNE MELO MIRANDA.**

O'que são grafos?

Grafos são estruturas matemáticas utilizadas para representar relações entre objetos. São formados por duas categorias: vértice e arestas.()

Vértices

- Representam os objetos do estudo.
- O conjunto de todos os vértices é finito e não vazio, denotado por V(G).
- Cada vértice é um elemento único, geralmente representado por letras como u, v, w ou índices como v1, v2.

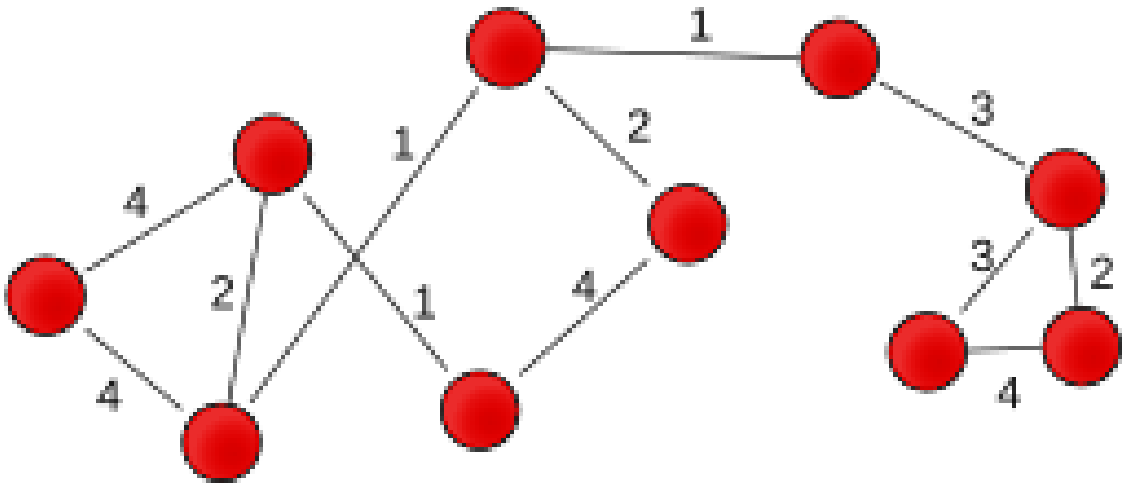
Arestas

- Conectam pares de vértices, representando a relação entre os objetos.
- O conjunto de todas as arestas é denotado por E(G).
- Cada aresta é um par não ordenado de vértices distintos {u, v}, onde u e v pertencem a V(G).
- Uma aresta {u, v} é dita incidir em u e em v, e os vértices u e v são ditos adjacentes.
- Uma aresta pode ser representada por letras como a, b, c ou índices como e1, e2, ou simplesmente pela notação uv.

Um grafo, como estrutura matemática que modela relações entre objetos, pode ser representado de diversas formas, cada uma com suas vantagens e desvantagens. A escolha da representação ideal depende do contexto e das operações que serão realizadas sobre o grafo.

Número de Arestas

Em um grafo de n vértices, a quantidade máxima de arestas é determinada por n(n-1)/2. Esse valor é obtido através da análise combinatória, que calcula o número de combinações de 2 vértices distintos que podem ser formadas a partir do conjunto de n vértices.



Tipos de Grafos

1. Há vários tipos de grafos, categorizados conforme suas propriedades:
2. Grafos não orientados: as arestas não possuem orientação, isto é, a aresta {u, v} é equivalente à aresta {v, u}.
3. Grafos orientados: as arestas possuem orientação, isto é, a aresta (u, v) difere da aresta (v, u).
4. Grafos ponderados: as linhas possuem pesos correspondentes, que simbolizam custos, distâncias ou outras medidas.
5. Grafos simples: não têm laços que ligam um vértice a si mesmo, nem arestas múltiplas que ligam os mesmos vértices.
6. Grafos completos: todas as combinações de vértices diferentes são ligadas por arestas.

Conceitos Complementares

1. Número de arestas que atingem um vértice: grau de um vértice.
2. Caminho: conjunto de vértices ligados através de arestas.
3. Ciclo: trajeto que inicia e se encerra no mesmo ponto.
4. Conectividade: característica de um grafo que sinaliza a existência de uma rota entre qualquer par de vértices.

Código em C (demonstração):

Código desenvolvido Por: Paulo Martins

```c
//Desenvolvido com o objetivo de ensino

#include <stdio.h>

#include <stdlib.h>

#define true 1

#define false 0

typedef int bool;

typedef int TIPOPESO;


typedef struct adjacencia {

    int vertice; // vertice de destino

    TIPOPESO peso; // peso associado a aresta que leva ao vertice de destino

    struct adjacencia *prox; // O próximo elemento da lista de adjacencias

}ADJACENCIA;


typedef struct vertice {

    /* Dados armazenados vão aqui */

    ADJACENCIA *cab; //possui apenas a cabeça da lista de adjacencia

}VERTICE;


typedef struct grafo { //lembrando que cada grafo possui:

    int vertices; // numero de vertice total do grafo
```

```
    int arestas; // numero de arestas totais do grafo
    VERTICE *adj; // Arranjo de vertices da estrutura
}GRAFO;


/**função para criar um GRAFO**/
GRAFO *criaGrafo (int v) {
  int i;


  GRAFO *g = (GRAFO *)malloc(sizeof(GRAFO)); //aloca espaço para estrtura grafo
  g->vertices = v; //atualizo o numero de vertice
  g->arestas = 0;  //atualizo o numero de vertice
  g->adj = (VERTICE *)malloc(v*sizeof(VERTICE)); //ler abaixo
  //Dentro da estrturua tem só o arranjo para o ponteiro de vertice, não o arranjo em si
  // então aloco o arranjo com (v) o numero de vertice desejado
  for (i=0; i<v; i++){
    g->adj[i].cab=NULL; //coloco NULL em todas arestas
  }
  return(g);
}
```

```
        /**função para adicionar arestas no GRAFO**/


        ADJACENCIA *criaAdj(int v, int peso){
          ADJACENCIA *temp = (ADJACENCIA *) malloc (sizeof(ADJACENCIA)); //aloca espaço para um nó
          temp->vertice =v; //vertice alvo da adjacencia
          temp->peso = peso; //peso da aresta
          temp->prox = NULL;
          return(temp); //retorno endereço da adjacencia
        }


        bool criaAresta(GRAFO *gr, int vi, int vf, TIPOPESO p) { //vai de vi a vf
          if(!gr) return (false);  //validações se o grafo existe
          if((vf<0)||(vf >= gr->vertices))return(false); //validações se os valores não são neg
          if((vi<0)||(vf >= gr->vertices))return(false); //ou maiores que o numero de vértice do grafo


          ADJACENCIA *novo = criaAdj(vf,p); //crio adjacencia com o vértice final e o peso
          //coloco a adjacencia na lista do vértice inicial
          novo->prox = gr->adj[vi].cab; //o campo prox da adjacencia vai receber a cabeça da lista
```

```
  gr->adj[vi].cab=novo; // e a cabeça da lista passa a ser o novo elemento
  gr->arestas++; // atualizo o numero de aresta no grafo
  return (true);
 }

 void imprime(GRAFO *gr){
  //validações se o grafo existe

  printf("Vertices: %d. Arestas: %d. \n",gr->vertices,gr->arestas); //imprime numero de vértice e arestas
  int i;

  for(i=0; i<gr->vertices; i++){
    printf("v%d: ",i); //Imprimo em qual aresta estou
    ADJACENCIA *ad = gr->adj[i].cab; //chamo a cabeça da lista de adjacencia desta aresta
     while(ad){ //enquanto as adjacencias não forem nula
       printf("v%d(%d) ",ad->vertice,ad->peso); //imprimo a adjacencia e seu peso
       ad=ad->prox; //passo para proxima adjacencia
     }
    printf("\n");
  }
```

```
}


int main()

{


  GRAFO * gr = criaGrafo(5);

  criaAresta(gr, 0, 1, 2);

  criaAresta(gr, 1, 2, 4);

  criaAresta(gr, 2, 0, 12);

  criaAresta(gr, 2, 4, 40);

  criaAresta(gr, 3, 1, 3);

  criaAresta(gr, 4, 3, 8);


    imprime(gr);

}
```

PDF, site, etc usados na pesquisa:
https://wiki.inf.ufpr.br/computacao/doku.php?id=t:teoria_dos_grafos#:~:text=Grafos%20s%C3%A3o%20uma%20%C3%A1rea%20da,v%20w%2C%20ou%20vi%2C%20vj.
https://medium.com/@paulomartins_10299/grafos-representa%C3%A7%C3%A3o-e-implementa%C3%A7%C3%A3o-f260dd98823d
https://github.com/MartinsPaulo/grafosC/blob/master/grafo-lista-adj-completo.c