MC-202
Grafos

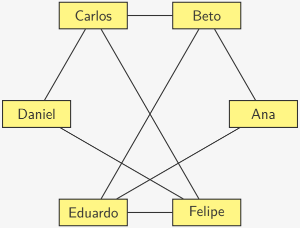
Rafael C. S. Schouery
rafael@ic.unicamp.br

Universidade Estadual de Campinas

Atualizado em: 2023-08-02 10:41

Redes Sociais

Como representar amizades em uma rede social?



Temos um conjunto de pessoas (Ana, Beto, Carlos, etc...)

- Ligamos duas pessoas se elas se conhecem

Grafos

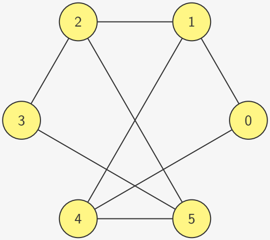
Um Grafo é um conjunto de objetos ligados entre si

- Chamamos esses objetos de vértices
  - Ex: pessoas em uma rede social
- Chamamos as conexões entre os objetos de arestas
  - Ex: relação de amizade na rede social

Representamos um grafo visualmente

- com os vértices representados por pontos e
- as arestas representadas por curvas ligando dois vértices

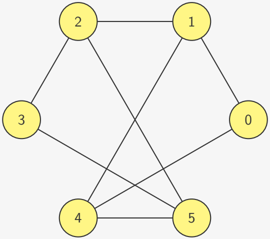
Grafos



Matematicamente, um grafo G é um par ordenado (V, E)

- V é o conjunto de vértices do grafo
  - Ex: $V = \{0, 1, 2, 3, 4, 5\}$
- E é o conjunto de arestas do grafo
  - Representamos uma aresta ligando $u, v \in V$ como $\{u, v\}$
  - Para toda aresta $\{u, v\}$ em E, temos que $u \neq v$
  - Existe no máximo uma aresta $\{u, v\}$ em E
  - Ex:
    $E = \Big\{\{0, 1\}, \{0, 4\}, \{5, 3\}, \{1, 2\}, \{2, 5\}, \{4, 5\}, \{3, 2\}, \{1, 4\}\Big\}$

Adjacência



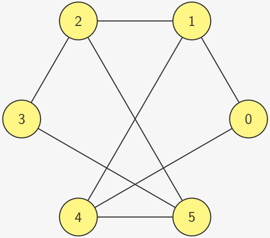
O vértice 0 é vizinho do vértice 4

- Dizemos que 0 e 4 são adjacentes
- Os vértices 0, 1 e 5 formam a vizinhança do vértice 4

Matriz de Adjacências

Vamos representar um grafo por uma matriz de adjacências

- Se o grafo tem n vértices
- Os vértices serão numerado de 0 a $n-1$
- A matriz de adjacências é $n \times n$
- `adjacencia[u][v] = 1` - u e v são vizinhos
- `adjacencia[u][v] = 0` - u e v não são vizinhos



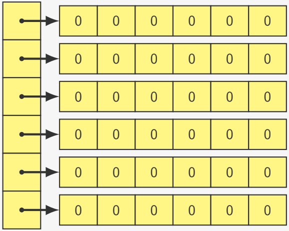
	0	1	2	3	4	5
0	0	1	0	0	1	0
1	1	0	1	0	1	0
2	0	1	0	1	0	1
3	0	0	1	0	0	1
4	1	1	0	0	0	1
5	0	0	1	1	1	0

TAD Grafo

```c
typedef grafo *p_grafo;

struct grafo {
  int **adj;
  int n;
};

p_grafo criar_grafo(int n);

void destroi_grafo(p_grafo g);

void insere_aresta(p_grafo g, int u, int v);

void remove_aresta(p_grafo g, int u, int v);

int tem_aresta(p_grafo g, int u, int v);

void imprime_arestas(p_grafo g);

...
```

Inicialização e Destruição

```c
p_grafo criar_grafo(int n) {
  int i, j;
  p_grafo g = malloc(sizeof(struct grafo));
  g->n = n;
  g->adj = malloc(n * sizeof(int *));
  for (i = 0; i < n; i++)
    g->adj[i] = malloc(n * sizeof(int));
  for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
        g->adj[i][j] = 0;
  return g;
}
```




```c
void destroi_grafo(p_grafo g) {
  int i;
  for (i = 0; i < g->n; i++)
    free(g->adj[i]);
  free(g->adj);
  free(g);
}
```

Manipulando arestas

```c
void insere_aresta(p_grafo g, int u, int v) {
  g->adj[u][v] = 1;
  g->adj[v][u] = 1;
}
```

```c
void remove_aresta(p_grafo g, int u, int v) {
  g->adj[u][v] = 0;
  g->adj[v][u] = 0;
}
```

```c
int tem_aresta(p_grafo g, int u, int v) {
  return g->adj[u][v];
}
```

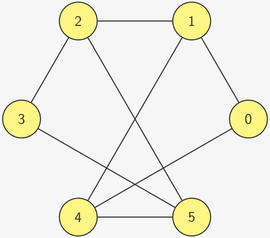
Lendo e Imprimindo um Grafo

```c
p_grafo le_grafo() {
  int n, m, i, u, v;
  p_grafo g;
  scanf("%d %d", &n, &m);
  g = criar_grafo(n);
  for (i = 0; i < m; i++) {
    scanf("%d %d", &u, &v);
    insere_aresta(g, u, v);
  }
  return g;
}
```

```c
void imprime_arestas(p_grafo g) {
  int u, v;
  for (u = 0; u < g->n; u++)
    for (v = u+1; v < g->n; v++)
      if (g->adj[u][v])
        printf("{%d,%d}\n", u, v);
}
```

Quem é o mais popular?

O grau de um vértice é o seu número de vizinhos



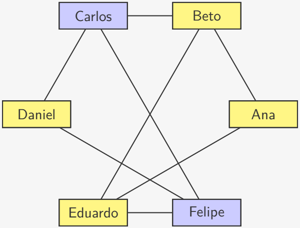

```
int grau(p_grafo g, int u) {
  int v, grau = 0;
  for (v = 0; v < g->n; v++)
    if (g->adj[u][v])
      grau++;
  return grau;
}
```

Quem é o mais popular?

```c
int mais_popular(p_grafo g) {
  int u, max, grau_max, grau_atual;
  max = 0;
  grau_max = grau(g, 0);
  for (u = 1; u < g->n; u++) {
    grau_atual = grau(g, u);
    if (grau_atual > grau_max) {
      grau_max = grau_atual;
      max = u;
    }
  }
  return max;
}
```

Indicando amigos

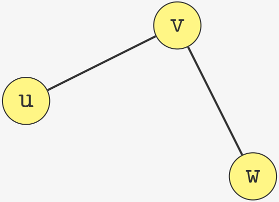
Queremos indicar novos amigos para Ana



Quem são os amigos dos amigos da Ana?

- Dentre esses quais não são ela mesma ou amigos dela?

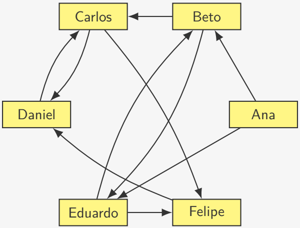
Indicando amigos




```c
void imprime_recomendacoes(p_grafo g, int u) {
  int v, w;
  for (v = 0; v < g->n; v++) {
    if (g->adj[u][v]) {
      for (w = 0; w < g->n; w++) {
        if (g->adj[v][w] && w != u && !g->adj[u][w])
          printf("%d\n", w);
      }
    }
  }
}
```

Seguindo e sendo seguido

Como representar seguidores em redes sociais?

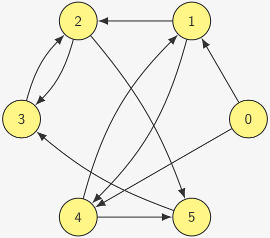


- A Ana segue o Beto e o Eduardo
- Ninguém segue a Ana
- O Daniel é seguido pelo Carlos e pelo Felipe
- O Eduardo segue o Beto que o segue de volta

Grafos dirigidos

Um Grafo dirigido (ou Digrafo)

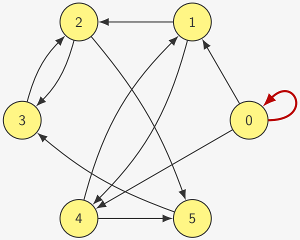
- Tem um conjunto de vértices
- Conectados através de um conjunto de arcos
  - arestas dirigidas, indicando início e fim



Representamos um digrafo visualmente

- com os vértices representados por pontos e
- os arcos representadas por curvas com uma seta na ponta ligando dois vértices

Grafos dirigidos

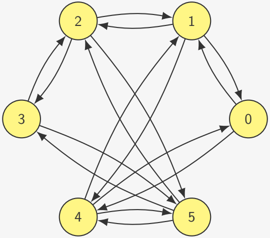


Matematicamente, um digrafo G é um par (V, A)

- V é o conjunto de vértices do grafo
- A é o conjunto de arcos do grafo
  - Representamos um arco ligando $u, v \in V$ como (u, v)
    - u é a cauda ou origem de (u, v)
    - v é a cabeça ou destino de (u, v)
  - Podemos ter laços: arcos da forma (u, u)
  - Existe no máximo um arco (u, v) em A

Grafos e digrafos

Podemos ver um grafo como um digrafo

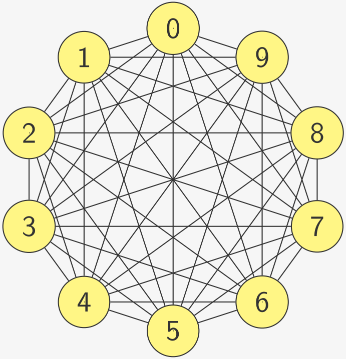


Basta considerar cada aresta como dois arcos

- É o que já estamos fazendo na matriz de adjacências
- Ou seja, podemos usar uma matriz de adjacências para representar um digrafo
  - `adjacencia[u][v] == 1`: temos um arco de `u` para `v`
  - pode ser que `adjacencia[u][v] != adjacencia[v][u]`

Número de arestas de um grafo

Quantas arestas pode ter um grafo com n vértices?



Até $\binom{n}{2} = \frac{n(n-1)}{2} = \mathrm{O}(n^2)$ arestas

Grafos esparsos

Um grafo tem no máximo $n(n-1)/2$ arestas, mas pode ter bem menos...

Facebook tem $2{,}2$ bilhões de usuários ativos/mês

- Uma matriz de adjacências teria $4{,}84 \cdot 10^{18}$ posições
  - 605 petabytes (usando um bit por posição)
- Verificar se duas pessoas são amigas leva $\mathrm{O}(1)$
  - supondo que tudo isso coubesse na memória...
- Imprimir todos os amigos de uma pessoa leva $\mathrm{O}(n)$
  - Teríamos que percorrer $2{,}2$ bilhões de posições
  - Um usuário comum tem bem menos amigos do que isso...
  - Facebook coloca um limite de 5000 amigos

Grafos esparsos

Dizemos que um grafo é esparso se ele tem "poucas" arestas

- Bem menos do que $n(n-1)/2$

Exemplos:

- Facebook:
  - Cada usuário tem no máximo 5000 amigos
  - O máximo de arestas é $5{,}5 \cdot 10^{12}$
  - Bem menos do que $2{,}4 \cdot 10^{18}$
- Grafos cujos vértices têm o mesmo grau d (constante)
  - O número de arestas é $dn/2 = \mathrm{O}(n)$
- Grafos com $\mathrm{O}(n \lg n)$ arestas

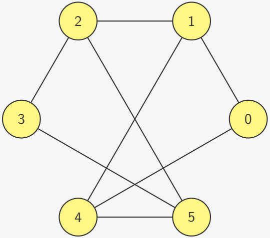
Não dizemos que um grafo com $n(n-1)/20$ arestas é esparso

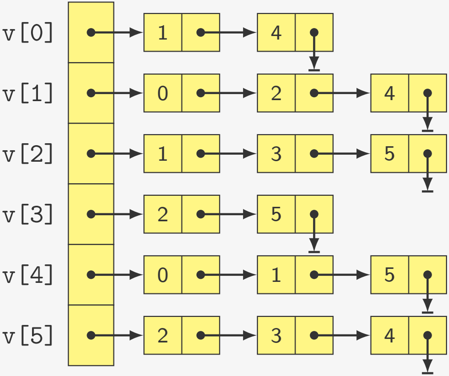
- O número de arestas não é assintoticamente menor...
- É da mesma ordem de grandeza que n^2...

Listas de Adjacência

Representando um grafo por Listas de Adjacência:

- Temos uma lista ligada para cada vértice
- A lista armazena quais são os vizinhos do vértice





TAD Grafo com Listas de Adjacência

```c
typedef struct no *p_no;

struct no {
  int v;
  p_no prox;
};

typedef struct grafo *p_grafo;

struct grafo {
  p_no *adjacencia;
  int n;
};

p_grafo criar_grafo(int n);

void destroi_grafo(p_grafo g);

void insere_aresta(p_grafo g, int u, int v);

void remove_aresta(p_grafo g, int u, int v);

int tem_aresta(p_grafo g, int u, int v);

void imprime_arestas(p_grafo g);
```

Inicialização e Destruição

```c
p_grafo criar_grafo(int n) {
  int i;
  p_grafo g = malloc(sizeof(struct grafo));
  g->n = n;
  g->adjacencia = malloc(n * sizeof(p_no));
  for (i = 0; i < n; i++)
    g->adjacencia[i] = NULL;
  return g;
}
```

```c
void libera_lista(p_no lista) {
  if (lista != NULL) {
    libera_lista(lista->prox);
    free(lista);
  }
}
```

```c
void destroi_grafo(p_grafo g) {
  int i;
  for (i = 0; i < g->n; i++)
    libera_lista(g->adjacencia[i]);
  free(g->adjacencia);
  free(g);
}
```

Inserindo uma aresta

```c
p_no insere_na_lista(p_no lista, int v) {
  p_no novo = malloc(sizeof(struct no));
  novo->v = v;
  novo->prox = lista;
  return novo;
}
```

```c
void insere_aresta(p_grafo g, int u, int v) {
  g->adjacencia[v] = insere_na_lista(g->adjacencia[v], u);
  g->adjacencia[u] = insere_na_lista(g->adjacencia[u], v);
}
```

Removendo uma aresta

```c
p_no remove_da_lista(p_no lista, int v) {
  p_no proximo;
  if (lista == NULL)
    return NULL;
  else if (lista->v == v) {
    proximo = lista->prox;
    free(lista);
    return proximo;
  } else {
    lista->prox = remove_da_lista(lista->prox, v);
    return lista;
  }
}
```

```c
void remove_aresta(p_grafo g, int u, int v) {
  g->adjacencia[u] = remove_da_lista(g->adjacencia[u], v);
  g->adjacencia[v] = remove_da_lista(g->adjacencia[v], u);
}
```

Verificando se tem uma aresta e imprimindo

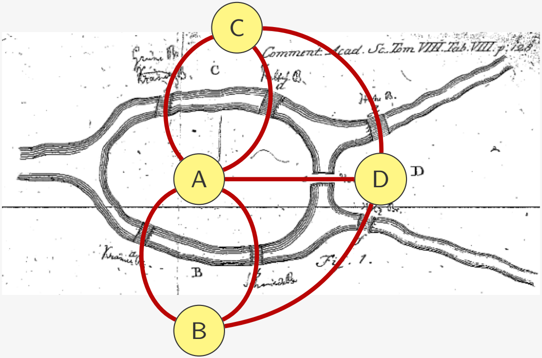
```
int tem_aresta(p_grafo g, int u, int v) {
  p_no t;
  for (t = g->adjacencia[u]; t != NULL; t = t->prox)
    if (t->v == v)
      return 1;
  return 0;
}
```

```
void imprime_arestas(p_grafo g) {
  int u;
  p_no t;
  for (u = 0; u < g->n; u++)
    for (t = g->adjacencia[u]; t != NULL; t = t->prox)
      printf("{%d,%d}\n", u, t->v);
}
```

O Problema das Pontes de Königsberg

Königsberg (hoje Kaliningrado, Rússia) tinha 7 pontes

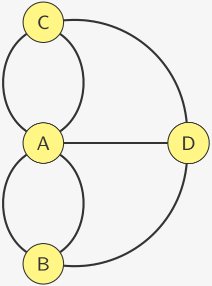
- Acreditava-se que era possível passear por toda a cidade
- Atravessando cada ponte exatamente uma vez



Leonhard Euler, em 1736, modelou o problema como um grafo

- Provou que tal passeio não é possível
- E fundou a Teoria dos Grafos...

Multigrafos



A estrutura usada por Euler é o que chamamos de Multigrafo

- Podemos ter arestas paralelas (ou múltiplas)
- Ao invés de um conjunto de arestas, temos um multiconjunto de arestas
- Pode ser representada por Listas de Adjacência
  - Por Matriz de Adjacências é mais difícil

Comparação Listas e Matrizes

Espaço para o armazenamento:

- Matriz: $\mathrm{O}(|V|^2)$
- Listas: $\mathrm{O}(|V| + |E|)$

Tempo:

Operação	Matriz	Listas
Inserir	$\mathrm{O}(1)$	$\mathrm{O}(1)$
Remover	$\mathrm{O}(1)$	$\mathrm{O}(d(v))$
Aresta existe?	$\mathrm{O}(1)$	$\mathrm{O}(d(v))$
Percorrer vizinhança	$\mathrm{O}(\lvert V\rvert)$	$\mathrm{O}(d(v))$

As duas permitem representar grafos, digrafos e multigrafos

- mas multigrafos é mais fácil com Listas de Adjacência

Qual usar?

- Depende das operações usadas e se o grafo é esparso

Importância dos Grafos

Grafos são amplamente usados na Computação e na Matemática para a modelagem de problemas:

- Redes Sociais: grafos são a forma de representar uma relação entre duas pessoas
- Mapas: podemos ver o mapa de uma cidade como um grafo e achar o menor caminho entre dois pontos
- Páginas na Internet: links são arcos de uma página para a outra - podemos querer ver qual é a página mais popular
- Redes de Computadores: a topologia de uma rede de computadores é um grafo
- Circuitos Eletrônicos: podemos criar algoritmos para ver se há curto-circuito por exemplo
- etc...