MC-202
Grafos

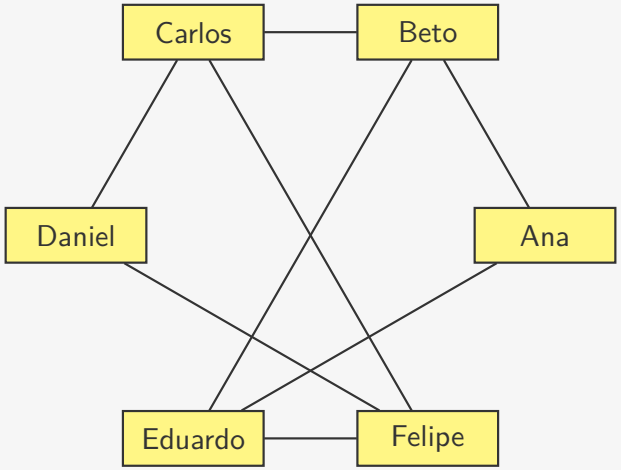
Rafael C. S. Schouery
rafael@ic.unicamp.br

Universidade Estadual de Campinas

2º semestre/2023

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

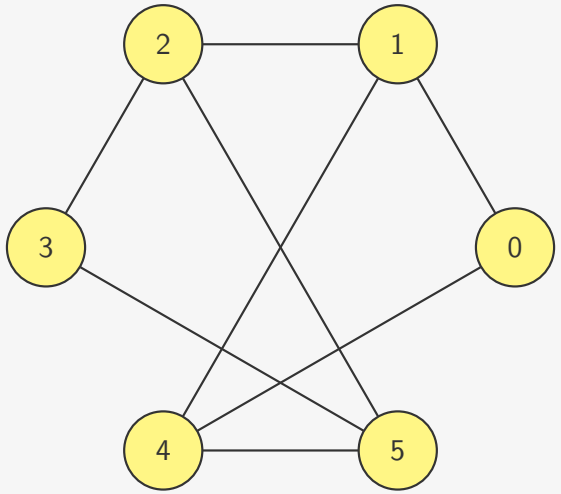
Um Grafo é um conjunto de objetos ligados entre si

- 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

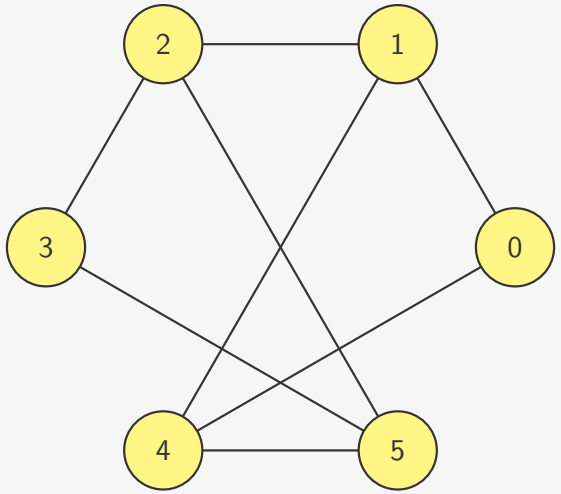
Grafos



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



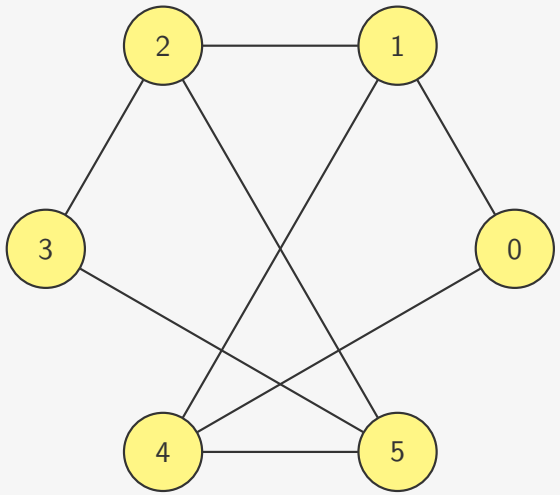
O vértice 0 é vizinho do vértice 4

- 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



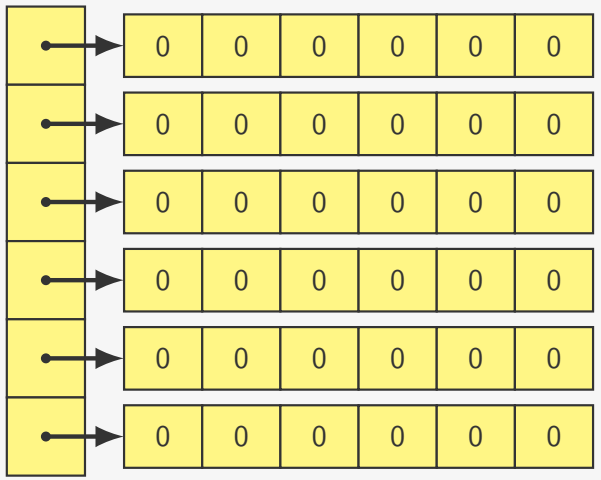
	0	1	2	3	4	5
0	0	1	0	0	1	0
1	1	0	1	0	1	0
2	0	1	0	1	0	1
3	0	0	1	0	0	1
4	1	1	0	0	0	1
5	0	0	1	1	1	0

TAD Grafo

```
1  typedef grafo *p_grafo;
2
3  struct grafo {
4    int **adj;
5    int n;
6  };
7
8  p_grafo criar_grafo(int n);
9
10 void destroi_grafo(p_grafo g);
11
12 void insere_aresta(p_grafo g, int u, int v);
13
14 void remove_aresta(p_grafo g, int u, int v);
15
16 int tem_aresta(p_grafo g, int u, int v);
17
18 void imprime_arestas(p_grafo g);
19
20 ...
```

Inicialização e Destruição

```
1  p_grafo criar_grafo(int n) {
2    int i, j;
3    p_grafo g = malloc(sizeof(struct grafo));
4    g->n = n;
5    g->adj = malloc(n * sizeof(int *));
6    for (i = 0; i < n; i++)
7      g->adj[i] = malloc(n * sizeof(int));
8    for (i = 0; i < n; i++)
9      for (j = 0; j < n; j++)
10         g->adj[i][j] = 0;
11   return g;
12 }
```




```
1  void destroi_grafo(p_grafo g) {
2    int i;
3    for (i = 0; i < g->n; i++)
4      free(g->adj[i]);
5    free(g->adj);
6    free(g);
7  }
```

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];
}
```

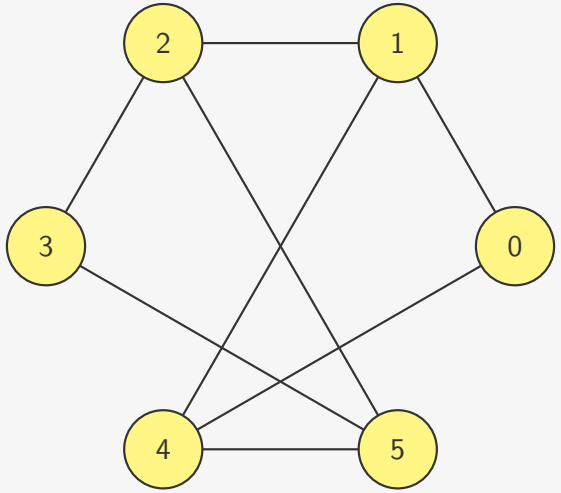
Lendo e Imprimindo um Grafo

```
1  p_grafo le_grafo() {
2    int n, m, i, u, v;
3    p_grafo g;
4    scanf("%d %d", &n, &m);
5    g = criar_grafo(n);
6    for (i = 0; i < m; i++) {
7      scanf("%d %d", &u, &v);
8      insere_aresta(g, u, v);
9    }
10   return g;
11 }
```

```
1  void imprime_arestas(p_grafo g) {
2    int u, v;
3    for (u = 0; u < g->n; u++)
4      for (v = u+1; v < g->n; v++)
5        if (g->adj[u][v])
6          printf("{%d,%d}\n", u, v);
7  }
```

Quem é o mais popular?

O grau de um vértice é o seu número de vizinhos



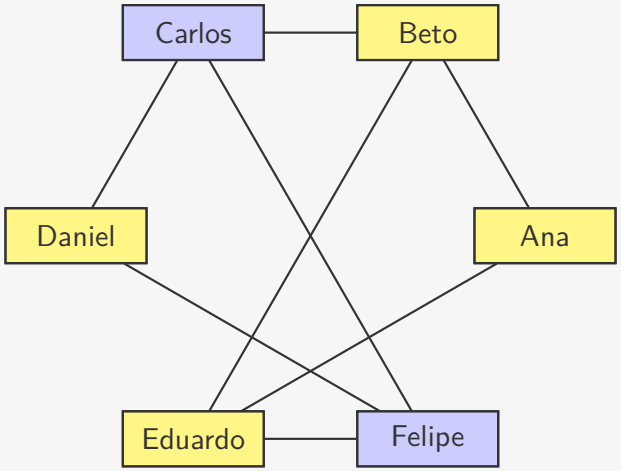

```
1 int grau(p_grafo g, int u) {
2   int v, grau = 0;
3   for (v = 0; v < g->n; v++)
4     if (g->adj[u][v])
5       grau++;
6   return grau;
7 }
```

Quem é o mais popular?

```
1  int mais_popular(p_grafo g) {
2    int u, max, grau_max, grau_atual;
3    max = 0;
4    grau_max = grau(g, 0);
5    for (u = 1; u < g->n; u++) {
6      grau_atual = grau(g, u);
7      if (grau_atual > grau_max) {
8        grau_max = grau_atual;
9        max = u;
10     }
11   }
12   return max;
13 }
```

Indicando amigos

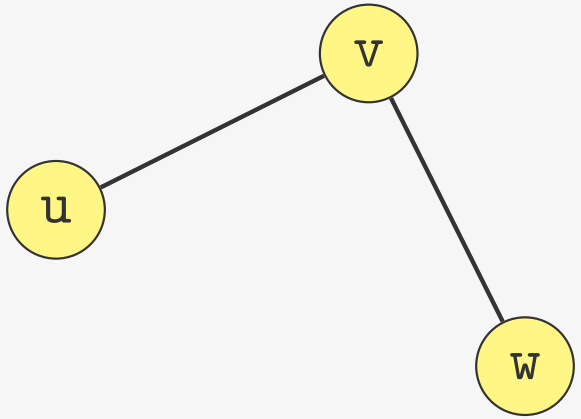
Queremos indicar novos amigos para Ana



Quem são os amigos dos amigos da Ana?

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

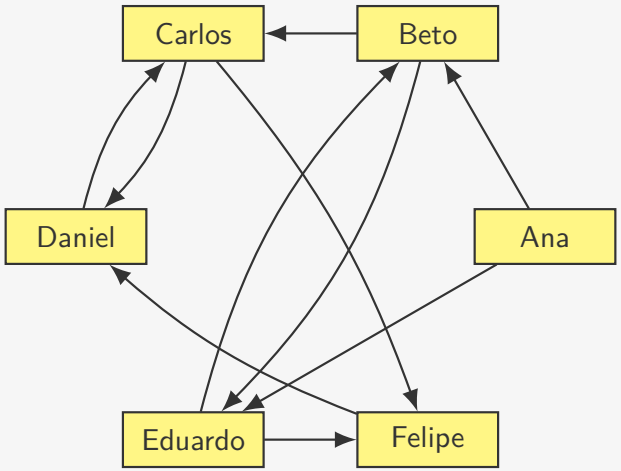
Indicando amigos




```
1  void imprime_recomendacoes(p_grafo g, int u) {
2    int v, w;
3    for (v = 0; v < g->n; v++) {
4      if (g->adj[u][v]) {
5        for (w = 0; w < g->n; w++) {
6          if (g->adj[v][w] && w != u && !g->adj[u][w])
7            printf("%d\n", w);
8        }
9      }
10   }
11 }
```

Seguindo e sendo seguido

Como representar seguidores em redes sociais?

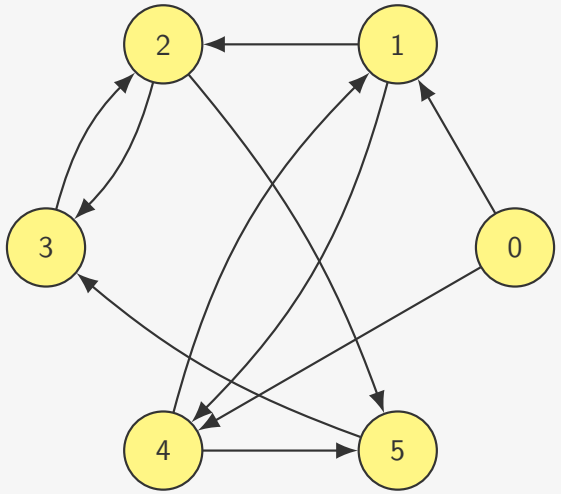


- A Ana segue o Beto e o Eduardo
- Ninguém segue a Ana
- O Daniel é seguido pelo Carlos e pelo Felipe
- O Eduardo segue o Beto que o segue de volta

Grafos dirigidos

Um Grafo dirigido (ou Digrafo)

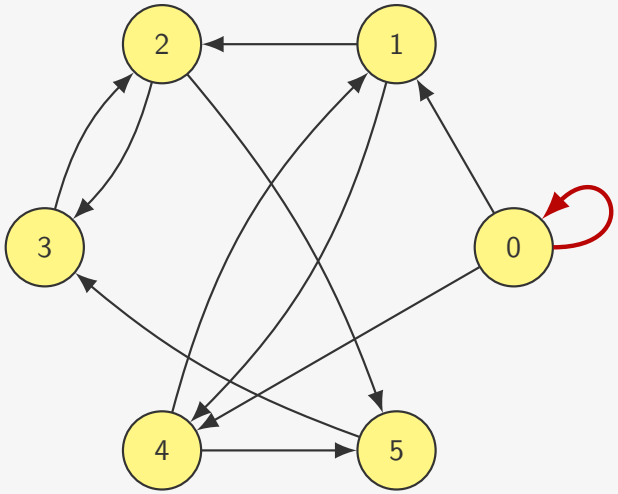
- Tem um conjunto de vértices
- Conectados através de um conjunto de arcos
  - arestas dirigidas, indicando início e fim



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

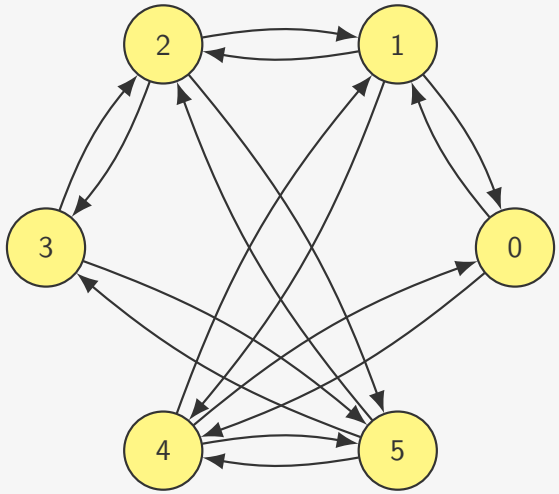


Matematicamente, um digrafo G é um par (V, A)

- 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

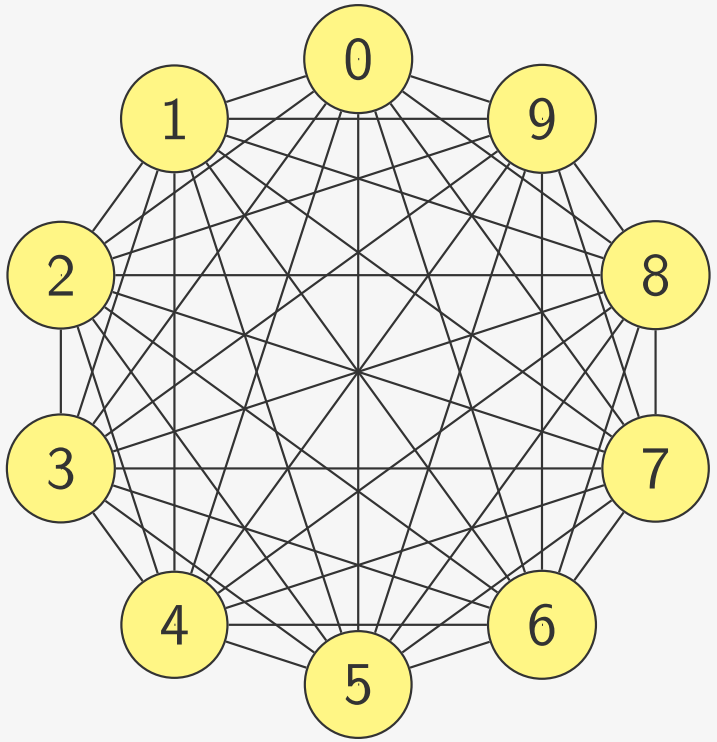


Basta considerar cada aresta como dois arcos

- 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

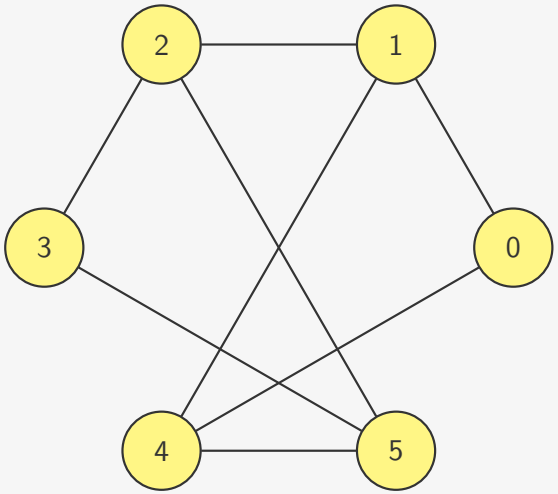
Não dizemos que um grafo com $n(n-1)/20$ arestas é esparso

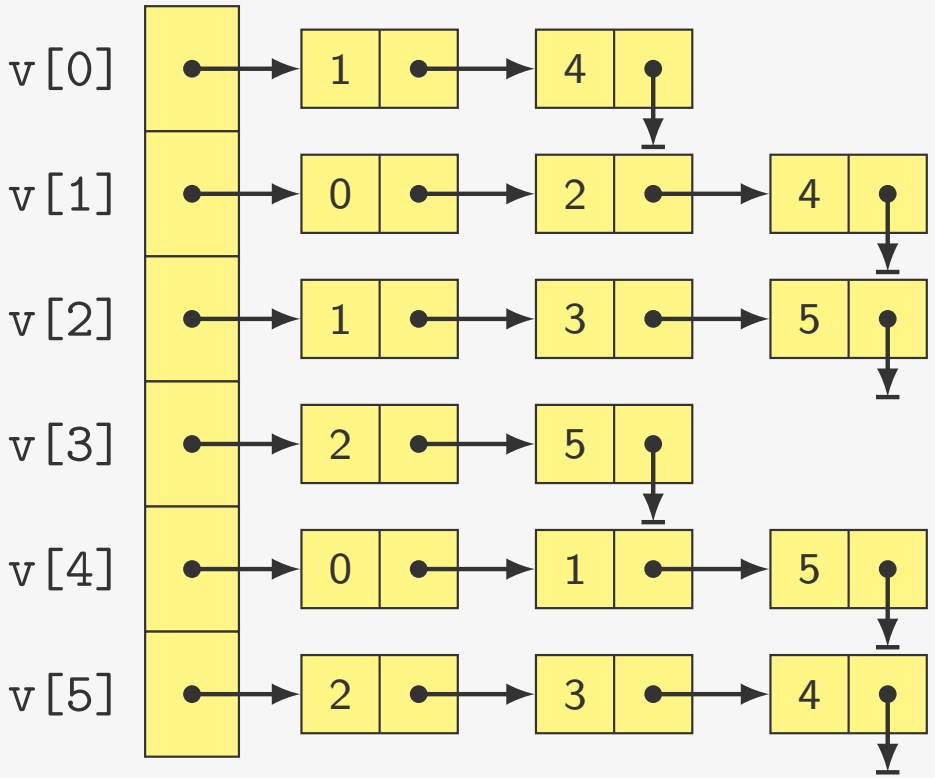
- O número de arestas não é assintoticamente menor...
- É da mesma ordem de grandeza que n^2...

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

```
1  typedef struct no *p_no;
2
3  struct no {
4    int v;
5    p_no prox;
6  };
7
8  typedef struct grafo *p_grafo;
9
10 struct grafo {
11   p_no *adjacencia;
12   int n;
13 };
14
15 p_grafo criar_grafo(int n);
16
17 void destroi_grafo(p_grafo g);
18
19 void insere_aresta(p_grafo g, int u, int v);
20
21 void remove_aresta(p_grafo g, int u, int v);
22
23 int tem_aresta(p_grafo g, int u, int v);
24
25 void imprime_arestas(p_grafo g);
```

Inicialização e Destruição

```
1 p_grafo criar_grafo(int n) {
2   int i;
3   p_grafo g = malloc(sizeof(struct grafo));
4   g->n = n;
5   g->adjacencia = malloc(n * sizeof(p_no));
6   for (i = 0; i < n; i++)
7     g->adjacencia[i] = NULL;
8   return g;
9 }
```

```
1 void libera_lista(p_no lista) {
2   if (lista != NULL) {
3     libera_lista(lista->prox);
4     free(lista);
5   }
6 }
```

```
1 void destroi_grafo(p_grafo g) {
2   int i;
3   for (i = 0; i < g->n; i++)
4     libera_lista(g->adjacencia[i]);
5   free(g->adjacencia);
6   free(g);
7 }
```

Inserindo uma aresta

```
1 p_no insere_na_lista(p_no lista, int v) {
2   p_no novo = malloc(sizeof(struct no));
3   novo->v = v;
4   novo->prox = lista;
5   return novo;
6 }
```

```
1 void insere_aresta(p_grafo g, int u, int v) {
2   g->adjacencia[v] = insere_na_lista(g->adjacencia[v], u);
3   g->adjacencia[u] = insere_na_lista(g->adjacencia[u], v);
4 }
```

Removendo uma aresta

```
1  p_no remove_da_lista(p_no lista, int v) {
2    p_no proximo;
3    if (lista == NULL)
4      return NULL;
5    else if (lista->v == v) {
6      proximo = lista->prox;
7      free(lista);
8      return proximo;
9    } else {
10     lista->prox = remove_da_lista(lista->prox, v);
11     return lista;
12   }
13 }
```

```
1  void remove_aresta(p_grafo g, int u, int v) {
2    g->adjacencia[u] = remove_da_lista(g->adjacencia[u], v);
3    g->adjacencia[v] = remove_da_lista(g->adjacencia[v], u);
4  }
```

Verificando se tem uma aresta e imprimindo

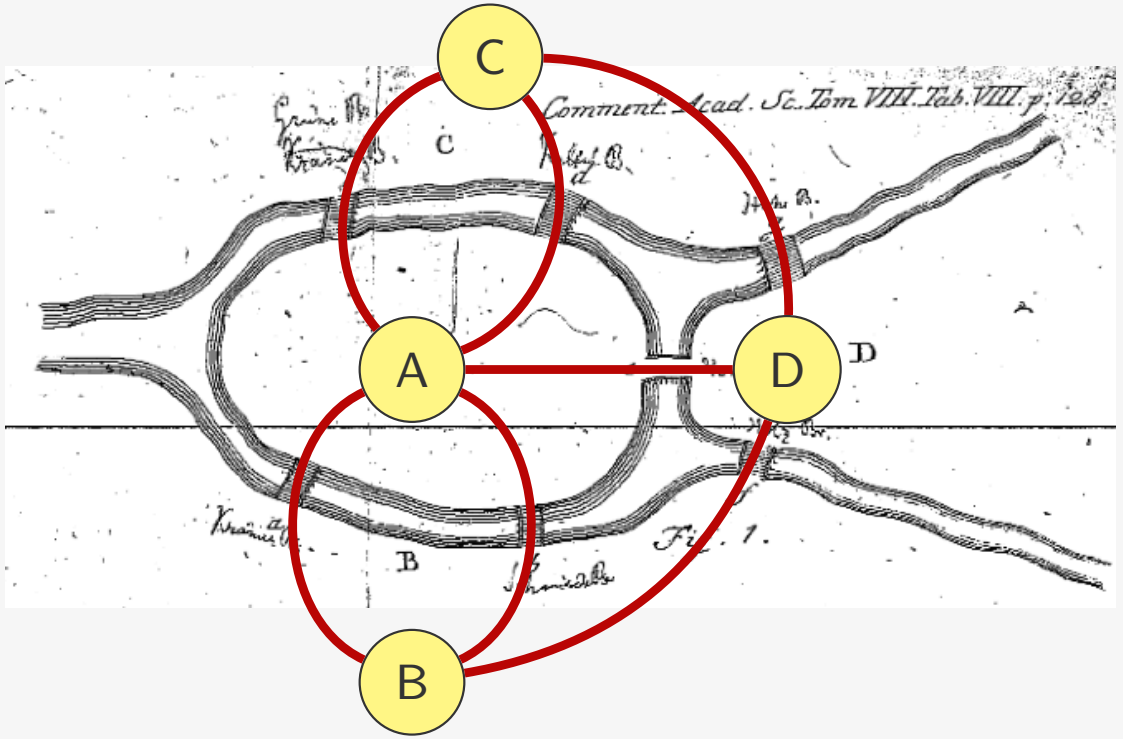
```
1 int tem_aresta(p_grafo g, int u, int v) {
2   p_no t;
3   for (t = g->adjacencia[u]; t != NULL; t = t->prox)
4     if (t->v == v)
5       return 1;
6   return 0;
7 }
```

```
1 void imprime_arestas(p_grafo g) {
2   int u;
3   p_no t;
4   for (u = 0; u < g->n; u++)
5     for (t = g->adjacencia[u]; t != NULL; t = t->prox)
6       printf("{%d,%d}\n", u, t->v);
7 }
```

O Problema das Pontes de Königsberg

Königsberg (hoje Kaliningrado, Rússia) tinha 7 pontes

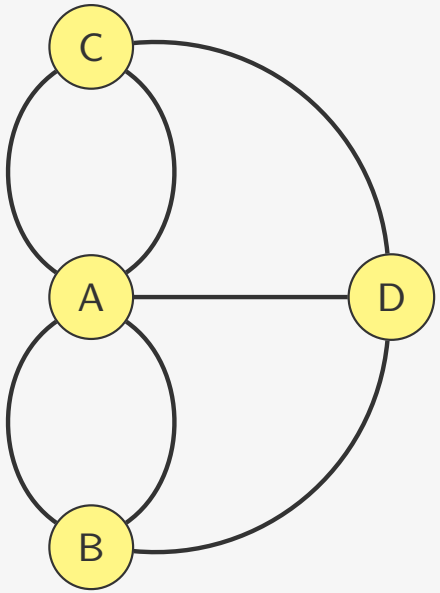
- Acreditava-se que era possível passear por toda a cidade
- Atravessando cada ponte exatamente uma vez



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	$\mathrm{O}(\|V\|)$	$\mathrm{O}(d(v))$

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...