Estruturas de Dados

Grafos (Introdução)

Aula 11

Prof. Felipe A. Louza

UFU

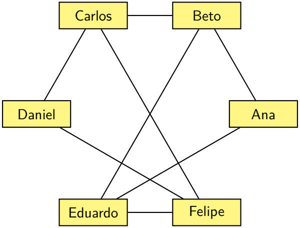
Roteiro

Roteiro

Grafos

Um Grafo é um conjunto de objetos ligados entre si

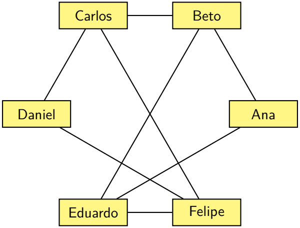
- Chamamos esses objetos de vértices
  - Ex: pessoas em uma rede social
- Chamamos as conexões entre os objetos de arestas
  - Ex: relação de amizade na rede social



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



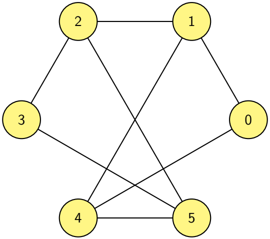
Representamos um grafo visualmente:

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

Grafos

Um Grafo é um conjunto de objetos ligados entre si

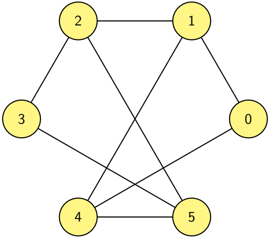
- Chamamos esses objetos de vértices
  - Ex: pessoas em uma rede social
- Chamamos as conexões entre os objetos de arestas
  - Ex: relação de amizade na rede social



Representamos um grafo visualmente:

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

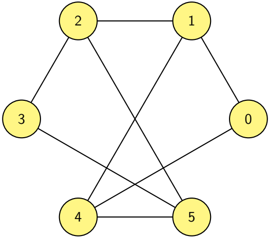
Grafos



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

- V é o conjunto de vértices do grafo
  - Ex: $V = \{0, 1, 2, 3, 4, 5\}$

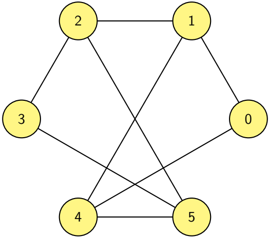
Grafos



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 $\{u, v\}$
  - 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

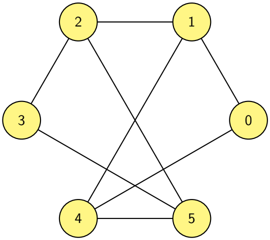
- Dizemos que 0 e 4 são adjacentes
- Os vértices 0, 1 e 5 formam a vizinhança do vértice 4

Roteiro

Matriz de Adjacências

Podemos 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 $n \times n$
- `M[u][v] = 1` $\Rightarrow$ u e v são vizinhos
- `M[u][v] = 0` $\Rightarrow$ 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 - Interface (matriz de adjacências)

grafo_matriz.h

```c
#ifndef GRAFO_MATRIZ_H
#define GRAFO_MATRIZ_H

//Dados
typedef struct {
  int **M; //Matriz de adjacências
  int n;
} Grafo;

//Funções
Grafo* criar_grafo(int n);
void destruir_grafo(Grafo *p);

void inserir_aresta(Grafo *p, int u, int v);
void remover_aresta(Grafo *p, int u, int v);

int tem_aresta(Grafo *p, int u, int v);
void imprimir_arestas(Grafo *g);

int get_vertices(Grafo *p);

#endif
```

Grafo - Criar e Destruir

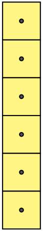
grafo_matriz.c

```c
Grafo* criar_grafo(int n) {//ex. n = 6
  int i, j;
  Grafo *p = (Grafo*) malloc(sizeof(Grafo));
  p->n = n;
  p->M = (int**) malloc(n * sizeof(int *));
  for (i = 0; i < n; i++)
    p->M[i] = (int*) malloc(n * sizeof(int));
  for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
        p->M[i][j] = 0;
  return p;
}
```

Grafo - Criar e Destruir

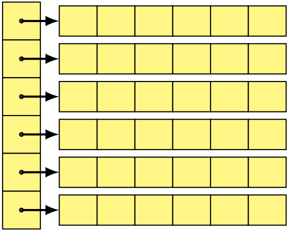
grafo_matriz.c

```c
Grafo* criar_grafo(int n) {//ex. n = 6
  int i, j;
  Grafo *p = (Grafo*) malloc(sizeof(Grafo));
  p->n = n;
  p->M = (int**) malloc(n * sizeof(int *));
  for (i = 0; i < n; i++)
    p->M[i] = (int*) malloc(n * sizeof(int));
  for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
        p->M[i][j] = 0;
  return p;
}
```



grafo_matriz.c

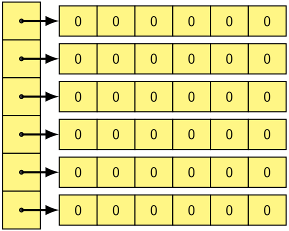
```
Grafo* criar_grafo(int n) {//ex. n = 6
  int i, j;
  Grafo *p = (Grafo*) malloc(sizeof(Grafo));
  p->n = n;
  p->M = (int**) malloc(n * sizeof(int *));
  for (i = 0; i < n; i++)
    p->M[i] = (int*) malloc(n * sizeof(int));
  for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
        p->M[i][j] = 0;
  return p;
}
```



Grafo - Criar e Destruir

grafo_matriz.c

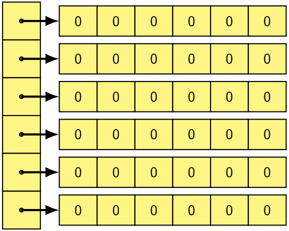
```c
Grafo* criar_grafo(int n) {//ex. n = 6
  int i, j;
  Grafo *p = (Grafo*) malloc(sizeof(Grafo));
  p->n = n;
  p->M = (int**) malloc(n * sizeof(int *));
  for (i = 0; i < n; i++)
    p->M[i] = (int*) malloc(n * sizeof(int));
  for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
        p->M[i][j] = 0;
  return p;
}
```



Grafo - Criar e Destruir

grafo_matriz.c

```c
Grafo* criar_grafo(int n) {//ex. n = 6
  int i, j;
  Grafo *p = (Grafo*) malloc(sizeof(Grafo));
  p->n = n;
  p->M = (int**) malloc(n * sizeof(int *));
  for (i = 0; i < n; i++)
    p->M[i] = (int*) malloc(n * sizeof(int));
  for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
        p->M[i][j] = 0;
  return p;
}
```



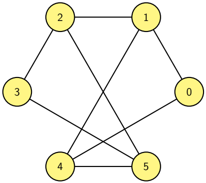

```c
void destruir_grafo(Grafo *p) {
  int i;
  for (i = 0; i < p->n; i++)
    free(p->M[i]);
  free(p->M);
  free(p);
}
```

Grafo - Manipulando arestas

grafo_matriz.c

```c
void inserir_aresta(Grafo *p, int u, int v) {
  p->M[u][v] = 1;
  p->M[v][u] = 1;
}

void remover_aresta(Grafo *p, int u, int v) {
  p->M[u][v] = 0;
  p->M[v][u] = 0;
}
```



	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

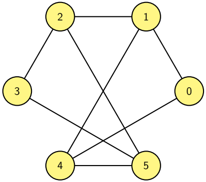
- Custo computacional: $O(1)$

Grafo - Imprimindo as arestas

grafo_matriz.c

```c
int tem_aresta(Grafo *p, int u, int v) {//custo computational: O(1)
  return p->M[u][v];
}

void imprimir_arestas(Grafo *p) {
  int u, v;
  for (u = 0; u < p->n; u++)
    for (v = u+1; v < p->n; v++)
      if (tem_aresta(p, u,v))
        printf("{%d,%d}\n", u, v);
}
```



	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

Cliente

exemplo1.c

```c
#include <stdio.h>
#include <stdlib.h>
#include "grafo_matriz.h"
```

```c
int main() {
  int n, m, i, u, v;
  scanf("%d %d", &n, &m);
  Grafo *G = criar_grafo(n);
  for (i = 0; i < m; i++) {
    scanf("%d %d", &u, &v);
    inserir_aresta(G, u, v);
  }
  imprimir_arestas(G);
  return 0;
}
```

Makefile

Vamos usar o Makefile para compilar:

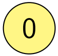
```
exemplo1: exemplo1.c grafo_matriz.o
  gcc $^ -o $@
```

Vamos executar:

```
$ ./exemplo1 < teste.in
{0,1}
{0,4}
{1,2}
{1,4}
{2,3}
{2,5}
{3,5}
{4,5}
```

Número de arestas de um grafo

Quantas arestas pode ter um grafo com n vértices?



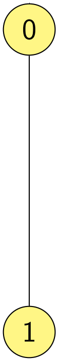
Será que a representação por matrizes é a melhor?

Número de arestas de um grafo

Quantas arestas pode ter um grafo com n vértices?

Número de arestas de um grafo

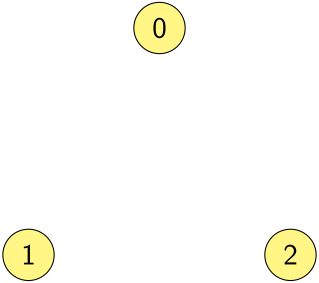
Quantas arestas pode ter um grafo com n vértices?



1

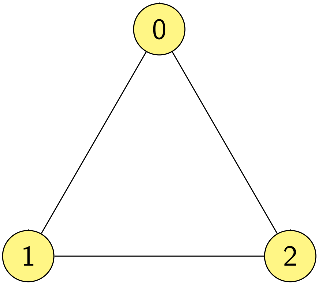
Número de arestas de um grafo

Quantas arestas pode ter um grafo com n vértices?



Número de arestas de um grafo

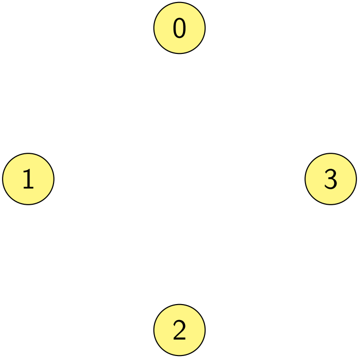
Quantas arestas pode ter um grafo com n vértices?



3

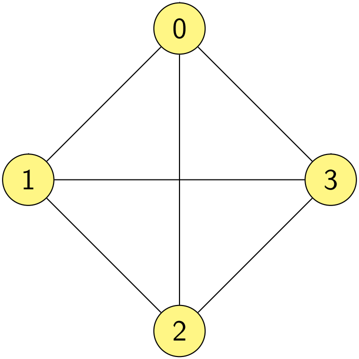
Número de arestas de um grafo

Quantas arestas pode ter um grafo com n vértices?



Número de arestas de um grafo

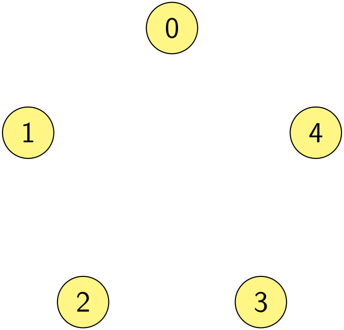
Quantas arestas pode ter um grafo com n vértices?



6

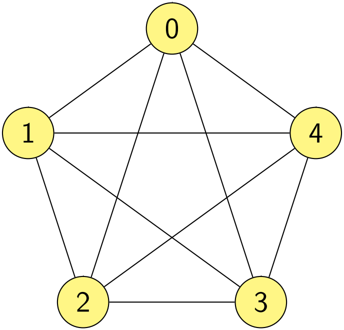
Número de arestas de um grafo

Quantas arestas pode ter um grafo com n vértices?



Número de arestas de um grafo

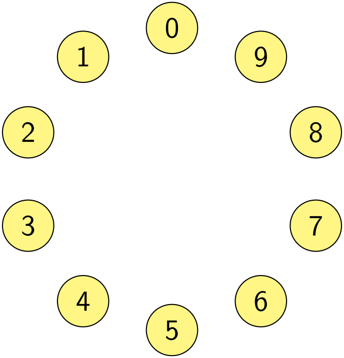
Quantas arestas pode ter um grafo com n vértices?



10

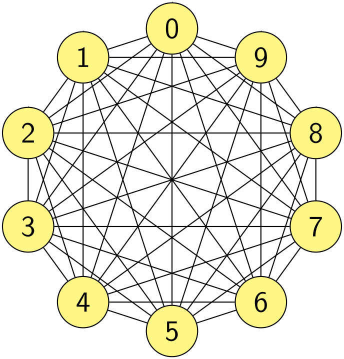
Número de arestas de um grafo

Quantas arestas pode ter um grafo com n vértices?



Número de arestas de um grafo

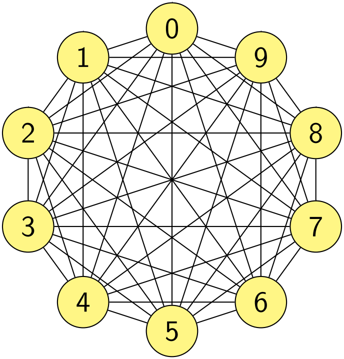
Quantas arestas pode ter um grafo com n vértices?



45

Número de arestas de um grafo

Quantas arestas pode ter um grafo com n vértices?

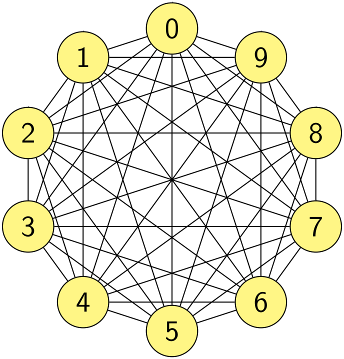


Até $\binom{n}{2}$ arestas

Número de combinações de n elementos tomados 2 a 2

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 $n(n-1)/2 = \mathrm{O}(n^2)$ arestas, mas pode ter bem menos...

Grafos esparsos

Um grafo tem no máximo $n(n-1)/2 = \mathrm{O}(n^2)$ 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)

Grafos esparsos

Um grafo tem no máximo $n(n-1)/2 = \mathrm{O}(n^2)$ 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...

Grafos esparsos

Um grafo tem no máximo $n(n-1)/2 = \mathrm{O}(n^2)$ arestas, mas pode ter bem menos...

Facebook tem 2,2 bilhões de usuários ativos/mês

- Uma matriz de adjacências teria $\underline{4{,}84 \cdot 10^{18}}$ 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...

Grafos esparsos

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

- Bem menos do que $n(n-1)/2 = \mathrm{O}(n^2)$

Não dizemos que $n(n-1)/20$ arestas é esparso (mesma ordem de grandeza)

Grafos esparsos

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

- Bem menos do que $n(n-1)/2 = \mathrm{O}(n^2)$

Exemplos reais:

- **Facebook**:
  - Cada usuário tem no máximo 5.000 amigos
  - O máximo de arestas é $5{,}5 \cdot 10^{12}$
  - Bem menos do que $2{,}4 \cdot 10^{18}$

hhttps://www.facebook.com/help/mobile-touch/187491537967631

Grafos esparsos

Dizemos que um grafo é esparso se ele tem “poucas” arestas

- Bem menos do que $n(n-1)/2 = \mathrm{O}(n^2)$

Exemplos reais:

- **Facebook**:
  - Cada usuário tem no máximo 5.000 amigos
  - O máximo de arestas é $5{,}5 \cdot 10^{12}$
  - Bem menos do que $2{,}4 \cdot 10^{18}$
- **Instagram**:
  - Cada usuário tem no máximo 7.500 conexões

https://help.instagram.com/408167069251249

Grafos esparsos

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

- Bem menos do que $n(n-1)/2 = \mathrm{O}(n^2)$

Exemplos reais:

- **Facebook**:
  - Cada usuário tem no máximo 5.000 amigos
  - O máximo de arestas é $5{,}5 \cdot 10^{12}$
  - Bem menos do que $2{,}4 \cdot 10^{18}$
- **Instagram**:
  - Cada usuário tem no máximo 7.500 conexões
- **Youtube**:
  - Cada usuário pode se inscrever em no máximo 2.000 páginas

---

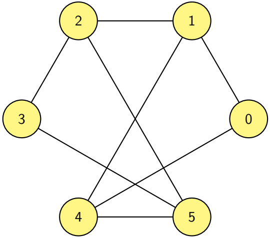
hhttps://support.google.com/youtube/answer/4489286

Roteiro

Listas de Adjacência

Podemos representar um grafo por uma Listas de Adjacências

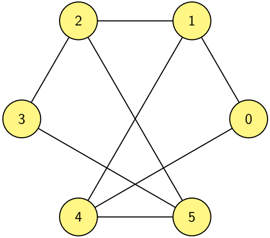
- Temos uma lista ligada para cada vértice
- A lista armazena quais são os vizinhos do vértice

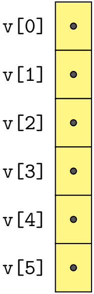


Listas de Adjacência

Podemos representar um grafo por uma Listas de Adjacências

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

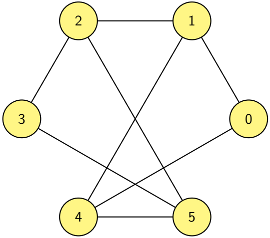


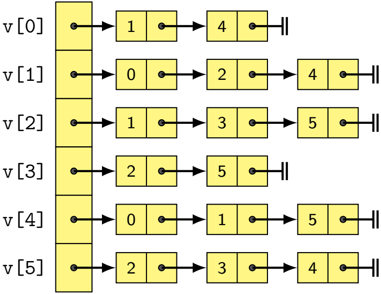


Listas de Adjacência

Podemos representar um grafo por uma Listas de Adjacências

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





TAD - Interface (lista de adjacências)

grafo_lista.h

```c
#ifndef GRAFO_LISTA_H
#define GRAFO_LISTA_H

#include "lista.h"

//Dados
typedef struct {
  No **L; //Lista de adjacências
  int n;
} Grafo;

//Funções
Grafo* criar_grafo(int n);
void destruir_grafo(Grafo *p);

void inserir_aresta(Grafo *p, int u, int v);
void remover_aresta(Grafo *p, int u, int v);

int tem_aresta(Grafo *p, int u, int v);
void imprimir_arestas(Grafo *g);

int get_vertices(Grafo *p);

#endif
```

Grafo - Criar e Destruir

grafo_matriz.c

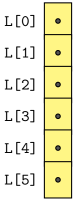
```
Grafo* criar_grafo(int n) {
  int i;
  Grafo *g = (Grafo*) malloc(sizeof(Grafo));
  g->n = n;
  g->L = (No**) malloc(n * sizeof(No*));
  for (i = 0; i < n; i++)
    g->L[i] = criar_lista();
  return g;
}
```

```
void destruir_grafo(Grafo *g) {
  int i;
  for (i = 0; i < g->n; i++)
    destruir_lista(g->L[i]);
  free(g->L);
  free(g);
}
```

Grafo - Criar e Destruir

grafo_matriz.c

```c
Grafo* criar_grafo(int n) {
  int i;
  Grafo *g = (Grafo*) malloc(sizeof(Grafo));
  g->n = n;
  g->L = (No**) malloc(n * sizeof(No*));
  for (i = 0; i < n; i++)
    g->L[i] = criar_lista();
  return g;
}
```

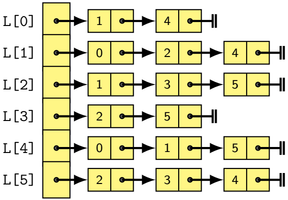



```c
void destruir_grafo(Grafo *g) {
  int i;
  for (i = 0; i < g->n; i++)
    destruir_lista(g->L[i]);
  free(g->L);
  free(g);
}
```

Grafo - Criar e Destruir

grafo_matriz.c

```c
Grafo* criar_grafo(int n) {
  int i;
  Grafo *g = (Grafo*) malloc(sizeof(Grafo));
  g->n = n;
  g->L = (No**) malloc(n * sizeof(No*));
  for (i = 0; i < n; i++)
    g->L[i] = criar_lista();
  return g;
}
```



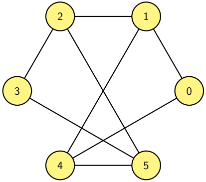

```c
void destruir_grafo(Grafo *g) {
  int i;
  for (i = 0; i < g->n; i++)
    destruir_lista(g->L[i]);
  free(g->L);
  free(g);
}
```

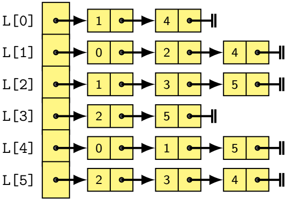
Grafo - Manipulando arestas

grafo_matriz.c

```c
void inserir_aresta(Grafo *g, int u, int v) {
  g->L[v] = inserir_na_lista(g->L[v], u);
  g->L[u] = inserir_na_lista(g->L[u], v);
}

void remover_aresta(Grafo *g, int u, int v) {
  g->L[u] = remover_da_lista(g->L[u], v);
  g->L[v] = remover_da_lista(g->L[v], u);
}
```





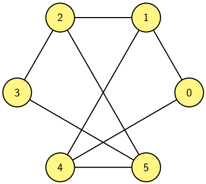
- Custo computacional: $\mathrm{O}(|V|)$

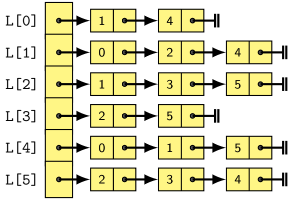
Grafo - Imprimindo as arestas

grafo_matriz.c

```c
int tem_aresta(Grafo *g, int u, int v) {//custo computational: O(|V|)
  return buscar_valor(g->L[u], v);
}

void imprimir_arestas(Grafo *g) {
  int u;
  for (u = 0; u < g->n; u++){
      printf("%d: ", u);
      imprimir_lista(g->L[u]);
  }
}
```





Cliente

exemplo2.c

```c
#include <stdio.h>
#include <stdlib.h>
#include "grafo_lista.h"
```

```c
int main() {
  int n, m, i, u, v;
  scanf("%d %d", &n, &m);
  Grafo *G = criar_grafo(n);
  for (i = 0; i < m; i++) {
    scanf("%d %d", &u, &v);
    inserir_aresta(G, u, v);
  }
  imprimir_arestas(G);
  return 0;
}
```

Makefile

Vamos usar o Makefile para compilar:

```
exemplo2: exemplo2.c grafo_matriz.o
  gcc $^ -o $@
```

Vamos executar:

```
$ ./exemplo2 < teste.in
0: 4 -> 1
1: 4 -> 2 -> 0
2: 5 -> 3 -> 1
3: 5 -> 2
4: 1 -> 0 -> 5
5: 2 -> 4 -> 3
```

Comparação Listas e Matrizes

Espaço para o armazenamento para $G = (V, E)$:

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

Comparação Listas e Matrizes

Espaço para o armazenamento para $G = (V, E)$:

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

Tempo:

Operação	Matriz	Listas
Inserir	$\mathrm{O}(1)$	$\mathrm{O}(1)$

Inserir em $\mathrm{O}(1)$: ponteiro para o final da lista.

Comparação Listas e Matrizes

Espaço para o armazenamento para $G = (V, E)$:

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

Tempo:

Operação	Matriz	Listas
Inserir	$\mathrm{O}(1)$	$\mathrm{O}(1)$
Remover	$\mathrm{O}(1)$	$\mathrm{O}(d(v))$

O grau de um vértice v, denotado por $d(v)$, é o seu número de vizinhos.

Comparação Listas e Matrizes

Espaço para o armazenamento para $G = (V, E)$:

- 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?	$\mathrm{O}(1)$	$\mathrm{O}(d(v))$

O grau de um vértice v, denotado por $d(v)$, é o seu número de vizinhos.

Comparação Listas e Matrizes

Espaço para o armazenamento para $G = (V, E)$:

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

O grau de um vértice v, denotado por $d(v)$, é o seu número de vizinhos.

Comparação Listas e Matrizes

Espaço para o armazenamento para $G = (V, E)$:

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

Qual usar?

O grau de um vértice v, denotado por $d(v)$, é o seu número de vizinhos.

Comparação Listas e Matrizes

Espaço para o armazenamento para $G = (V, E)$:

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

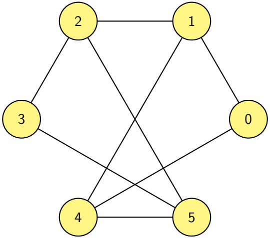
Qual usar?

- Depende das **operações usadas** e se o grafo é **esparso**

O grau de um vértice v, denotado por $d(v)$, é o seu número de vizinhos.

Roteiro

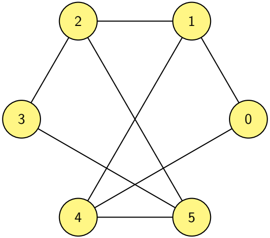
Quem é o mais popular?




```c
int grau(Grafo *p, int u) {
  int n = get_vertices(p); // n = |V|
  int v, grau = 0;
  for (v = 0; v < n; v++)
    if (tem_aresta(p, u, v)) grau++;
return grau;
}
```

Custo computacional: $\mathrm{O}(|V|)$ com matrizes de adjacências.

Quem é o mais popular?

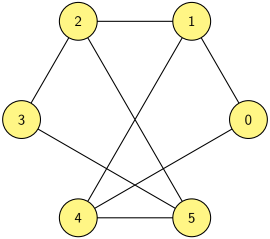



```c
int grau(Grafo *p, int u) {
  int n = get_vertices(p); // n = |V|
  int v, grau = 0;
  for (v = 0; v < n; v++)
    if (tem_aresta(p, u, v)) grau++;
return grau;
}
```

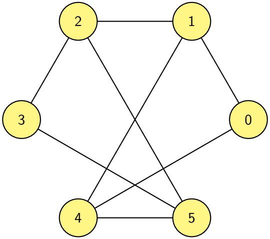
Podemos fazer uma operação mais eficiente para listas de adjacências

Quem é o mais popular?

```c
int mais_popular(Grafo *p) {
  int n = get_vertices(p); // n = |V|
  int max = 0;
  int grau_max = grau(p, 0);
  int u;
  for (u = 1; u < n; u++) {
    grau_atual = grau(p, u);//calcula 1 vez
    if (grau_atual > grau_max) {
      grau_max = grau_atual;
      max = u;
    }
  }
return max;
}
```



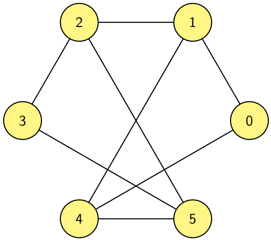
```
int mais_popular(Grafo *p) {
  int n = get_vertices(p); // n = |V|
  int max = 0;
  int grau_max = grau(p, 0);
  int u;
  for (u = 1; u < n; u++) {
    grau_atual = grau(p, u);//calcula 1 vez
    if (grau_atual > grau_max) {
      grau_max = grau_atual;
      max = u;
    }
  }
return max;
}
```



- Custo computacional:

Quem é o mais popular?

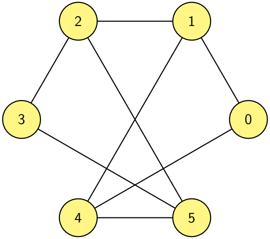
```c
int mais_popular(Grafo *p) {
  int n = get_vertices(p); // n = |V|
  int max = 0;
  int grau_max = grau(p, 0);
  int u;
  for (u = 1; u < n; u++) {
    grau_atual = grau(p, u);//calcula 1 vez
    if (grau_atual > grau_max) {
      grau_max = grau_atual;
      max = u;
    }
  }
return max;
}
```



- Custo computacional:
  - Matriz de adjacências:
  - Listas de adjacências:

Quem é o mais popular?

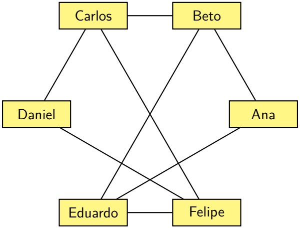
```
int mais_popular(Grafo *p) {
  int n = get_vertices(p); // n = |V|
  int max = 0;
  int grau_max = grau(p, 0);
  int u;
  for (u = 1; u < n; u++) {
    grau_atual = grau(p, u);//calcula 1 vez
    if (grau_atual > grau_max) {
      grau_max = grau_atual;
      max = u;
    }
  }
return max;
}
```



- Custo computacional:
  - Matriz de adjacências: $O(|V|^2)$
  - Listas de adjacências: $O(|E|)$

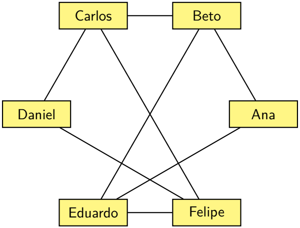
Indicando amigos

Queremos indicar novos amigos para Ana



Indicando amigos

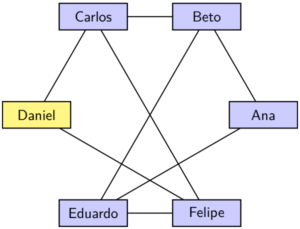
Queremos indicar novos amigos para Ana



Quem são os amigos dos amigos da Ana?

Indicando amigos

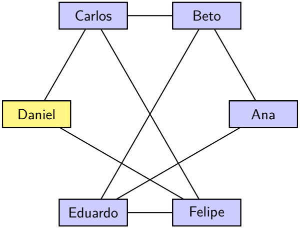
Queremos indicar novos amigos para Ana



Quem são os amigos dos amigos da Ana?

Indicando amigos

Queremos indicar novos amigos para Ana

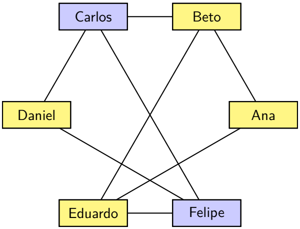


Quem são os amigos dos amigos da Ana?

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

Indicando amigos

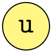
Queremos indicar novos amigos para Ana



Quem são os amigos dos amigos da Ana?

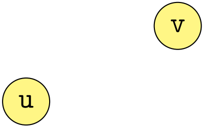
- Dentre esses quais **não são** ela mesma ou amigos dela?

Indicando amigos



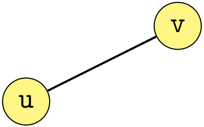

```
void imprime_recomendacoes(Grafo *p, int u) {
  int n = get_vertices(p); // n = |V|
  int v, w;
  for (v = 0; v < n; v++) {
    if (tem_aresta(p, u, v)) {
      for (w = 0; w < n; w++) {
        if (tem_aresta(p, v, w) && w != u && !tem_aresta(p, u, w))
          printf("%d\n", w);
      }
    }
  }
}
```

Indicando amigos



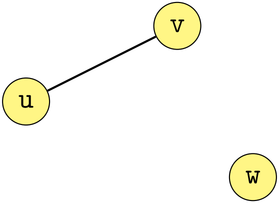

```c
void imprime_recomendacoes(Grafo *p, int u) {
  int n = get_vertices(p); // n = |V|
  int v, w;
  for (v = 0; v < n; v++) {
    if (tem_aresta(p, u, v)) {
      for (w = 0; w < n; w++) {
        if (tem_aresta(p, v, w) && w != u && !tem_aresta(p, u, w))
          printf("%d\n", w);
      }
    }
  }
}
```

Indicando amigos



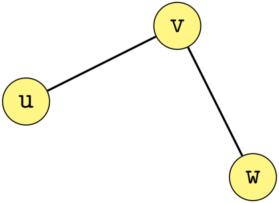

```c
void imprime_recomendacoes(Grafo *p, int u) {
  int n = get_vertices(p); // n = |V|
  int v, w;
  for (v = 0; v < n; v++) {
    if (tem_aresta(p, u, v)) {
      for (w = 0; w < n; w++) {
        if (tem_aresta(p, v, w) && w != u && !tem_aresta(p, u, w))
          printf("%d\n", w);
      }
    }
  }
}
```

Indicando amigos



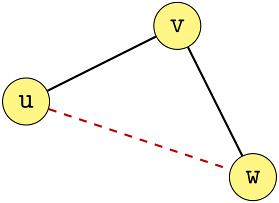

```c
void imprime_recomendacoes(Grafo *p, int u) {
  int n = get_vertices(p); // n = |V|
  int v, w;
  for (v = 0; v < n; v++) {
    if (tem_aresta(p, u, v)) {
      for (w = 0; w < n; w++) {
        if (tem_aresta(p, v, w) && w != u && !tem_aresta(p, u, w))
          printf("%d\n", w);
      }
    }
  }
}
```

Indicando amigos




```c
void imprime_recomendacoes(Grafo *p, int u) {
  int n = get_vertices(p); // n = |V|
  int v, w;
  for (v = 0; v < n; v++) {
    if (tem_aresta(p, u, v)) {
      for (w = 0; w < n; w++) {
        if (tem_aresta(p, v, w) && w != u && !tem_aresta(p, u, w))
          printf("%d\n", w);
      }
    }
  }
}
```

Indicando amigos



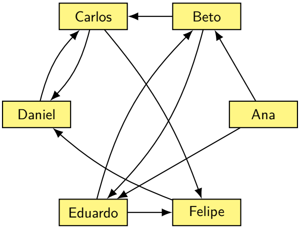

```c
void imprime_recomendacoes(Grafo *p, int u) {
  int n = get_vertices(p); // n = |V|
  int v, w;
  for (v = 0; v < n; v++) {
    if (tem_aresta(p, u, v)) {
      for (w = 0; w < n; w++) {
        if (tem_aresta(p, v, w) && w != u && !tem_aresta(p, u, w))
          printf("%d\n", w);
      }
    }
  }
}
```

Seguindo e sendo seguido

Como representar seguidores em redes sociais?

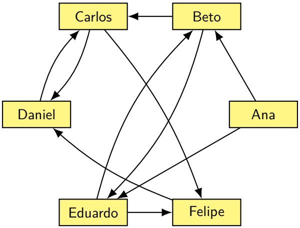
Seguindo e sendo seguido

Como representar seguidores em redes sociais?



Seguindo e sendo seguido

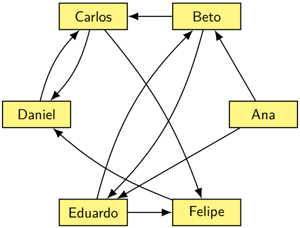
Como representar seguidores em redes sociais?



- A Ana segue o Beto e o Eduardo

Seguindo e sendo seguido

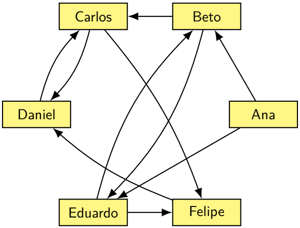
Como representar seguidores em redes sociais?



- A Ana segue o Beto e o Eduardo
- Ninguém segue a Ana

Seguindo e sendo seguido

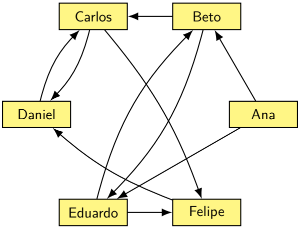
Como representar seguidores em redes sociais?



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

Seguindo e sendo seguido

Como representar seguidores em redes sociais?



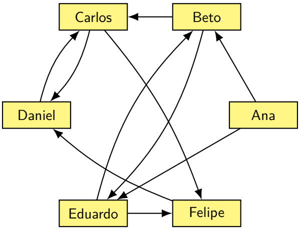
- A Ana segue o Beto e o Eduardo
- Ninguém segue a Ana
- O Daniel é seguido pelo Carlos e pelo Felipe
- O Eduardo segue o Beto que o segue de volta

Roteiro

Grafos dirigidos

Um Grafo dirigido (ou Digrafo)

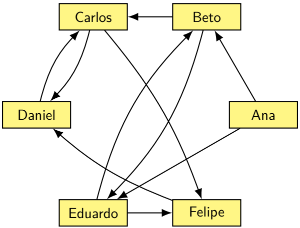
- Tem um conjunto de vértices
- Conectados através de arcos
  - arestas dirigidas, indicando início e fim



Grafos dirigidos

Um Grafo dirigido (ou Digrafo)

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

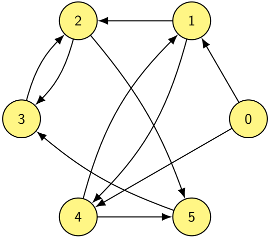


Representamos um digrafo visualmente

Grafos dirigidos

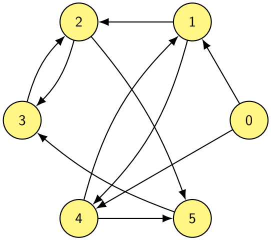
Um Grafo dirigido (ou Digrafo)

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



Representamos um digrafo visualmente

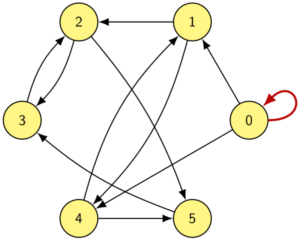
Grafos dirigidos



Matematicamente, um digrafo $G = (V, A)$

- 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)

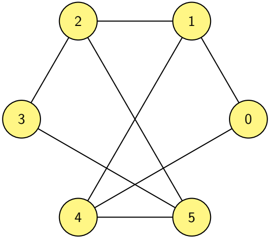
Grafos dirigidos



Matematicamente, um digrafo $G = (V, A)$

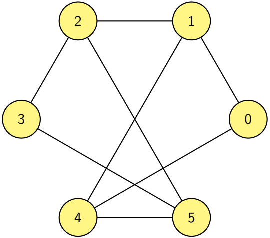
- V é o conjunto de vértices do grafo
- A é o conjunto de arcos do grafo
  - Representamos um arco ligando $u, v \in V$ como (u, v)
    - u é a cauda ou origem de (u, v)
    - v é a cabeça ou destino de (u, v)
  - Podemos ter laços: arcos da forma (u, u)

Podemos ver um grafo como um digrafo



Grafos e digrafos

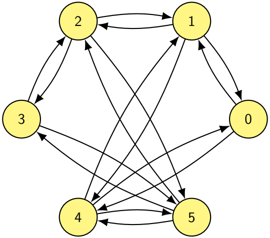
Podemos ver um grafo como um digrafo



Basta considerar cada aresta como dois arcos

Grafos e digrafos

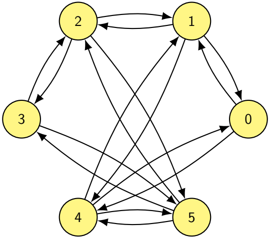
Podemos ver um grafo como um digrafo



Basta considerar cada aresta como dois arcos

Grafos e digrafos

Podemos ver um grafo como um digrafo

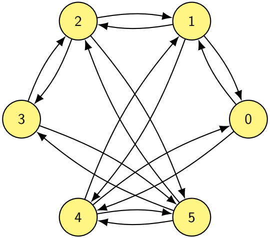


Basta considerar cada aresta como dois arcos

- É o que já estamos fazendo

Grafos e digrafos

Podemos ver um grafo como um digrafo



Basta considerar cada aresta como dois arcos

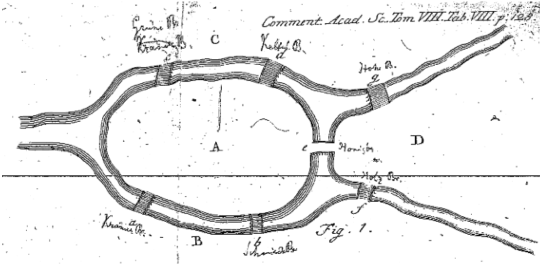
- É o que já estamos fazendo
- Ou seja, podemos usar matrizes ou listas de adjacências para representar um digrafo

Roteiro

O Problema das Pontes de Königsberg

Königsberg (hoje Kaliningrado, Rússia) tinha 7 pontes

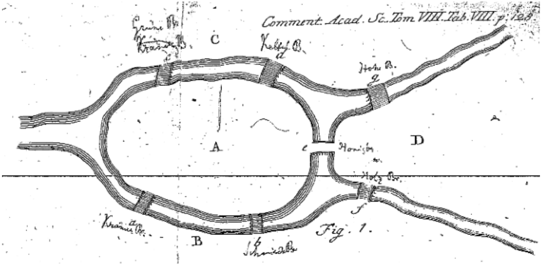
- Acreditava-se que era possível passear por toda a cidade
- Atravessando cada ponte exatamente uma vez



O Problema das Pontes de Königsberg

Königsberg (hoje Kaliningrado, Rússia) tinha 7 pontes

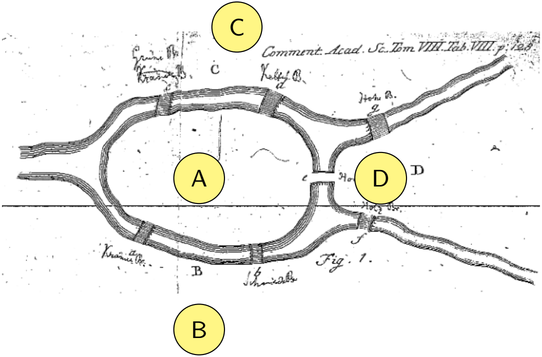
- Acreditava-se que era possível passear por toda a cidade
- Atravessando cada ponte exatamente uma vez



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

Königsberg (hoje Kaliningrado, Rússia) tinha 7 pontes

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

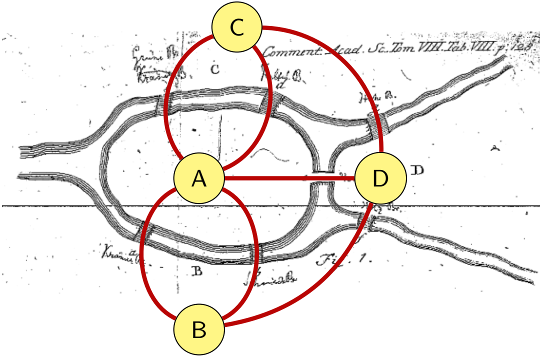


Leonhard Euler, em 1736, modelou o problema como um grafo

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

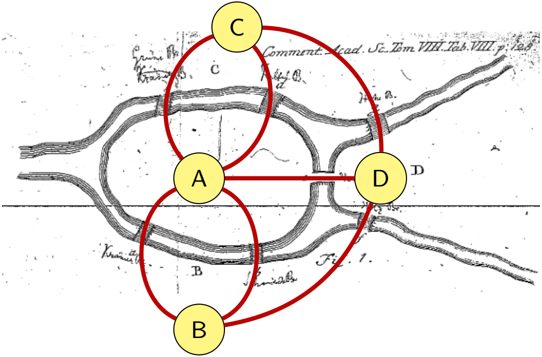


Leonhard Euler, em 1736, modelou o problema como um grafo

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



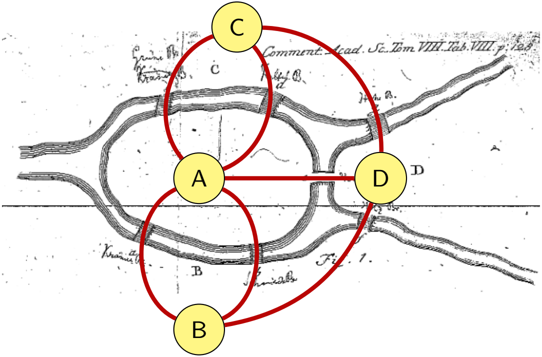
Leonhard Euler, em 1736, modelou o problema como um grafo

- Provou que tal passeio não é possível

O Problema das Pontes de Königsberg

Königsberg (hoje Kaliningrado, Rússia) tinha 7 pontes

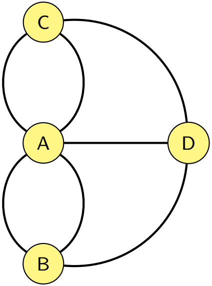
- Acreditava-se que era possível passear por toda a cidade
- Atravessando cada ponte exatamente uma vez



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

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

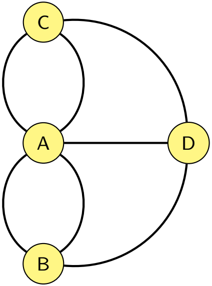
Multigrafos



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

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

Importância dos Grafos

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

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

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

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

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

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
- etc...

Fim

Dúvidas?

Roteiro

Referências

1. Materiais adaptados dos slides do Prof. Rafael C. S. Schouery, da Universidade Estadual de Campinas.
2. Thomas H. Cormen *et al.*, Algoritmos: teoria e prática, 2012.