# AED2 - Algoritmos e Estr. de Dados II

## Aula 22: Grafos Bipartidos e Emparelhamento

Prof. Aléssio Miranda Júnior
alessio@cefetmg.br

CEFET-MG - Campus Timoteo
Dep. Engenharia de Computação

Fevereiro de 2026

# Sumário

# Um Problema Cotidiano: Separação de Conflitos

## O Cenário

Você precisa armazenar um conjunto de reagentes químicos em armários. Alguns reagentes **reagem perigosamente entre si** e não podem ficar na mesma caixa.

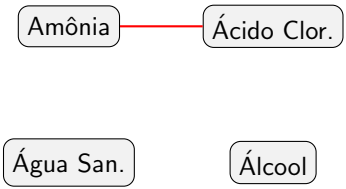
# Um Problema Cotidiano: Separação de Conflitos

## O Cenário

Você precisa armazenar um conjunto de reagentes químicos em armários. Alguns reagentes **reagem perigosamente entre si** e não podem ficar na mesma caixa.

**Restrições de Armazenamento:**

- Amônia reage com Ácido Clorídrico.



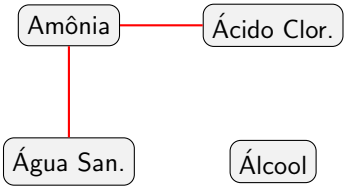
# Um Problema Cotidiano: Separação de Conflitos

## O Cenário

Você precisa armazenar um conjunto de reagentes químicos em armários. Alguns reagentes **reagem perigosamente entre si** e não podem ficar na mesma caixa.

**Restrições de Armazenamento:**

- Amônia reage com Ácido Clorídrico.
- Amônia reage com Água Sanitária (Alvejante).



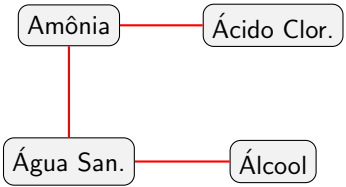
# Um Problema Cotidiano: Separação de Conflitos

## O Cenário

Você precisa armazenar um conjunto de reagentes químicos em armários. Alguns reagentes **reagem perigosamente entre si** e não podem ficar na mesma caixa.

**Restrições de Armazenamento:**

- Amônia reage com Ácido Clorídrico.
- Amônia reage com Água Sanitária (Alvejante).
- Água Sanitária reage com Álcool.



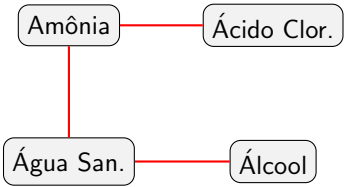
# Um Problema Cotidiano: Separação de Conflitos

## O Cenário

Você precisa armazenar um conjunto de reagentes químicos em armários. Alguns reagentes **reagem perigosamente entre si** e não podem ficar na mesma caixa.

**Restrições de Armazenamento:**

- Amônia reage com Ácido Clorídrico.
- Amônia reage com Água Sanitária (Alvejante).
- Água Sanitária reage com Álcool.




## Pergunta

É possível organizar todos esses reagentes em apenas **duas caixas** seguras?

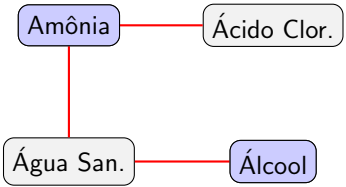
# Um Problema Cotidiano: Separação de Conflitos

## O Cenário

Você precisa armazenar um conjunto de reagentes químicos em armários. Alguns reagentes **reagem perigosamente entre si** e não podem ficar na mesma caixa.

**Restrições de Armazenamento:**

- Amônia reage com Ácido Clorídrico.
- Amônia reage com Água Sanitária (Alvejante).
- Água Sanitária reage com Álcool.




## Pergunta

É possível organizar todos esses reagentes em apenas **duas caixas** seguras?

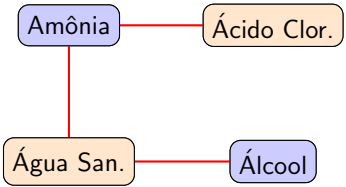
# Um Problema Cotidiano: Separação de Conflitos

## O Cenário

Você precisa armazenar um conjunto de reagentes químicos em armários. Alguns reagentes **reagem perigosamente entre si** e não podem ficar na mesma caixa.

**Restrições de Armazenamento:**

- Amônia reage com Ácido Clorídrico.
- Amônia reage com Água Sanitária (Alvejante).
- Água Sanitária reage com Álcool.




## Pergunta

É possível organizar todos esses reagentes em apenas **duas caixas** seguras?

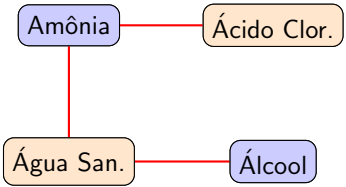
# Um Problema Cotidiano: Separação de Conflitos

## O Cenário

Você precisa armazenar um conjunto de reagentes químicos em armários. Alguns reagentes **reagem perigosamente entre si** e não podem ficar na mesma caixa.

**Restrições de Armazenamento:**

- Amônia reage com Ácido Clorídrico.
- Amônia reage com Água Sanitária (Alvejante).
- Água Sanitária reage com Álcool.




## Pergunta

É possível organizar todos esses reagentes em apenas **duas caixas** seguras?

**Sim! Isso é equivalente a pintar o grafo com 2 cores.**

# Outros Cenários do Mundo Real

## Onde mais encontramos relações em duas categorias?

Muitos problemas do dia a dia envolvem conectar **dois grupos distintos** de entidades:

# Outros Cenários do Mundo Real

## Onde mais encontramos relações em duas categorias?

Muitos problemas do dia a dia envolvem conectar **dois grupos distintos** de entidades:

- **Mobilidade Urbana (Uber):** Conjunto de *Motoristas* $\times$ Conjunto de *Passageiros*. Uma aresta existe se o motorista pode atender a corrida.

# Outros Cenários do Mundo Real

## Onde mais encontramos relações em duas categorias?

Muitos problemas do dia a dia envolvem conectar **dois grupos distintos** de entidades:

- **Mobilidade Urbana (Uber):** Conjunto de *Motoristas* $\times$ Conjunto de *Passageiros*. Uma aresta existe se o motorista pode atender a corrida.
- **Gestão Hospitalar:** Conjunto de *Médicos* $\times$ Conjunto de *Turnos*. Uma aresta existe se o médico tem disponibilidade naquele horário.

# Outros Cenários do Mundo Real

## Onde mais encontramos relações em duas categorias?

Muitos problemas do dia a dia envolvem conectar **dois grupos distintos** de entidades:

- **Mobilidade Urbana (Uber):** Conjunto de *Motoristas* $\times$ Conjunto de *Passageiros*. Uma aresta existe se o motorista pode atender a corrida.
- **Gestão Hospitalar:** Conjunto de *Médicos* $\times$ Conjunto de *Turnos*. Uma aresta existe se o médico tem disponibilidade naquele horário.
- **Sistemas de Recomendação:** Conjunto de *Usuários* $\times$ Conjunto de *Filmes* (Netflix).

# Outros Cenários do Mundo Real

## Onde mais encontramos relações em duas categorias?

Muitos problemas do dia a dia envolvem conectar **dois grupos distintos** de entidades:

- **Mobilidade Urbana (Uber):** Conjunto de *Motoristas* $\times$ Conjunto de *Passageiros*. Uma aresta existe se o motorista pode atender a corrida.
- **Gestão Hospitalar:** Conjunto de *Médicos* $\times$ Conjunto de *Turnos*. Uma aresta existe se o médico tem disponibilidade naquele horário.
- **Sistemas de Recomendação:** Conjunto de *Usuários* $\times$ Conjunto de *Filmes* (Netflix).
- **Compiladores:** Alocação de *Variáveis* em *Registradores* da CPU.

# Aprofundando: O Problema dos Compiladores

## Alocação de Registradores

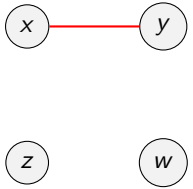
Um compilador precisa colocar variáveis da memória nos rápidos **registradores** da CPU. Duas variáveis que estão "vivas" ao mesmo tempo **não podem** ocupar o mesmo registrador.

## Alocação de Registradores

Um compilador precisa colocar variáveis da memória nos rápidos **registradores** da CPU. Duas variáveis que estão "vivas" ao mesmo tempo **não podem** ocupar o mesmo registrador.

**Conflitos de Variáveis:**

- Variável $x$ não pode ficar junto com $y$.

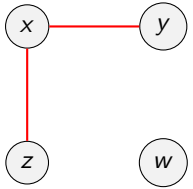


## Alocação de Registradores

Um compilador precisa colocar variáveis da memória nos rápidos **registradores** da CPU. Duas variáveis que estão "vivas" ao mesmo tempo **não podem** ocupar o mesmo registrador.

**Conflitos de Variáveis:**

- Variável $x$ não pode ficar junto com $y$.
- Variável $x$ não pode ficar junto com $z$.

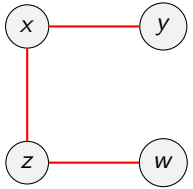


## Alocação de Registradores

Um compilador precisa colocar variáveis da memória nos rápidos **registradores** da CPU. Duas variáveis que estão "vivas" ao mesmo tempo **não podem** ocupar o mesmo registrador.

**Conflitos de Variáveis:**

- Variável $x$ não pode ficar junto com $y$.
- Variável $x$ não pode ficar junto com $z$.
- Variável $z$ não pode ficar junto com $w$.



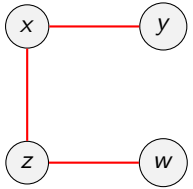
# Aprofundando: O Problema dos Compiladores

## Alocação de Registradores

Um compilador precisa colocar variáveis da memória nos rápidos **registradores** da CPU. Duas variáveis que estão "vivas" ao mesmo tempo **não podem** ocupar o mesmo registrador.

**Conflitos de Variáveis:**

- Variável $x$ não pode ficar junto com $y$.
- Variável $x$ não pode ficar junto com $z$.
- Variável $z$ não pode ficar junto com $w$.




## Pergunta

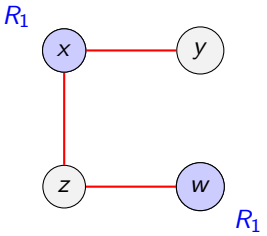
Temos apenas **2 registradores** ($R_1$ e $R_2$).
Conseguimos compilar esse código?

## Alocação de Registradores

Um compilador precisa colocar variáveis da memória nos rápidos **registradores** da CPU. Duas variáveis que estão "vivas" ao mesmo tempo **não podem** ocupar o mesmo registrador.

**Conflitos de Variáveis:**

- Variável $x$ não pode ficar junto com $y$.
- Variável $x$ não pode ficar junto com $z$.
- Variável $z$ não pode ficar junto com $w$.




## Pergunta

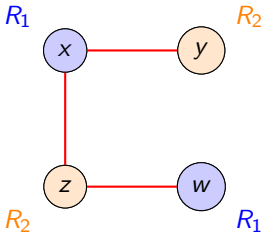
Temos apenas **2 registradores** ($R_1$ e $R_2$).
Conseguimos compilar esse código?

## Alocação de Registradores

Um compilador precisa colocar variáveis da memória nos rápidos **registradores** da CPU. Duas variáveis que estão "vivas" ao mesmo tempo **não podem** ocupar o mesmo registrador.

**Conflitos de Variáveis:**

- Variável $x$ não pode ficar junto com $y$.
- Variável $x$ não pode ficar junto com $z$.
- Variável $z$ não pode ficar junto com $w$.




## Pergunta

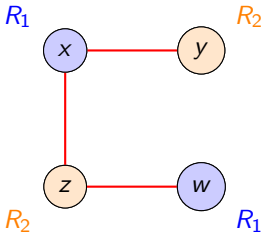
Temos apenas **2 registradores** ($R_1$ e $R_2$). Conseguimos compilar esse código?

## Alocação de Registradores

Um compilador precisa colocar variáveis da memória nos rápidos **registradores** da CPU. Duas variáveis que estão "vivas" ao mesmo tempo **não podem** ocupar o mesmo registrador.

**Conflitos de Variáveis:**

- Variável $x$ não pode ficar junto com $y$.
- Variável $x$ não pode ficar junto com $z$.
- Variável $z$ não pode ficar junto com $w$.




### Pergunta

Temos apenas **2 registradores** ($R_1$ e $R_2$). Conseguimos compilar esse código?

**Sim! Isso novamente é descobrir se o grafo é 2-colorível.**

## Escala de Plantões

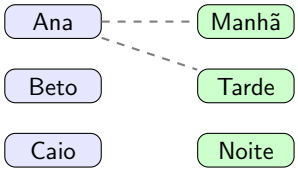
Um hospital precisa alocar **médicos** aos seus respectivos **turnos**. Cada médico só tem disponibilidade em alguns horários específicos.

## Escala de Plantões

Um hospital precisa alocar **médicos** aos seus respectivos **turnos**. Cada médico só tem disponibilidade em alguns horários específicos.


**Disponibilidades:**

- **Dra. Ana:** Manhã, Tarde.

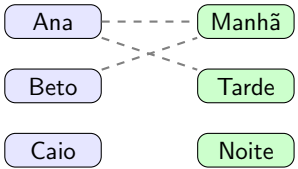


## Escala de Plantões

Um hospital precisa alocar **médicos** aos seus respectivos **turnos**. Cada médico só tem disponibilidade em alguns horários específicos.


**Disponibilidades:**

- **Dra. Ana:** Manhã, Tarde.
- **Dr. Beto:** Manhã.

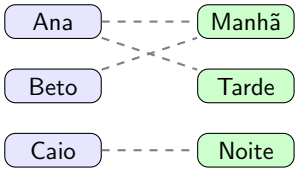


## Escala de Plantões

Um hospital precisa alocar **médicos** aos seus respectivos **turnos**. Cada médico só tem disponibilidade em alguns horários específicos.


**Disponibilidades:**

- **Dra. Ana:** Manhã, Tarde.
- **Dr. Beto:** Manhã.
- **Dr. Caio:** Noite.

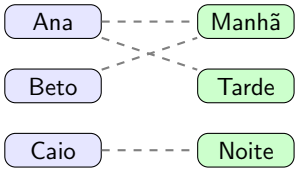


## Escala de Plantões

Um hospital precisa alocar **médicos** aos seus respectivos **turnos**. Cada médico só tem disponibilidade em alguns horários específicos.


**Disponibilidades:**

- **Dra. Ana:** Manhã, Tarde.
- **Dr. Beto:** Manhã.
- **Dr. Caio:** Noite.




### O Desafio

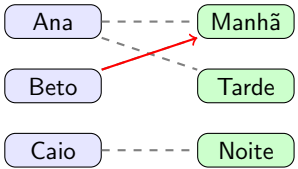
Como designar exatamente **um turno** para cada médico sem deixar o hospital vazio em nenhum momento?

## Escala de Plantões

Um hospital precisa alocar **médicos** aos seus respectivos **turnos**. Cada médico só tem disponibilidade em alguns horários específicos.


**Disponibilidades:**

- **Dra. Ana:** Manhã, Tarde.
- **Dr. Beto:** Manhã.
- **Dr. Caio:** Noite.




### O Desafio

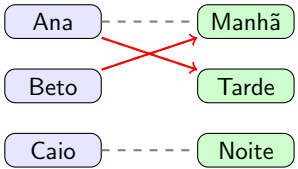
Como designar exatamente **um turno** para cada médico sem deixar o hospital vazio em nenhum momento?

## Escala de Plantões

Um hospital precisa alocar **médicos** aos seus respectivos **turnos**. Cada médico só tem disponibilidade em alguns horários específicos.


**Disponibilidades:**

- **Dra. Ana:** Manhã, Tarde.
- **Dr. Beto:** Manhã.
- **Dr. Caio:** Noite.




### O Desafio

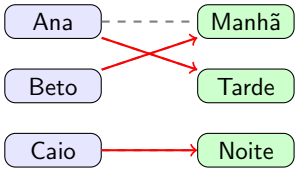
Como designar exatamente **um turno** para cada médico sem deixar o hospital vazio em nenhum momento?

## Escala de Plantões

Um hospital precisa alocar **médicos** aos seus respectivos **turnos**. Cada médico só tem disponibilidade em alguns horários específicos.


**Disponibilidades:**

- **Dra. Ana:** Manhã, Tarde.
- **Dr. Beto:** Manhã.
- **Dr. Caio:** Noite.




### O Desafio

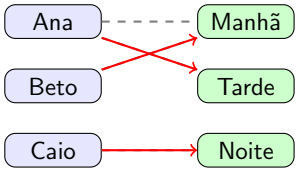
Como designar exatamente **um turno** para cada médico sem deixar o hospital vazio em nenhum momento?

## Escala de Plantões

Um hospital precisa alocar **médicos** aos seus respectivos **turnos**. Cada médico só tem disponibilidade em alguns horários específicos.


**Disponibilidades:**

- **Dra. Ana:** Manhã, Tarde.
- **Dr. Beto:** Manhã.
- **Dr. Caio:** Noite.



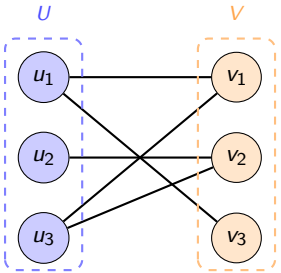

### O Desafio

Como designar exatamente **um turno** para cada médico sem deixar o hospital vazio em nenhum momento?

**Resolvido! Este é um clássico problema de Emparelhamento (Matching).**

# 1. O que é um Grafo Bipartido?

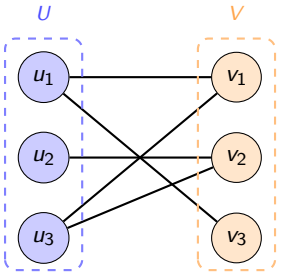
**Definição:** Grafo onde os vértices podem ser divididos em dois conjuntos $U$ e $V$ tal que **toda aresta** liga $U$ a $V$.



# 1. O que é um Grafo Bipartido?

**Definição:** Grafo onde os vértices podem ser divididos em dois conjuntos $U$ e $V$ tal que **toda aresta** liga $U$ a $V$.

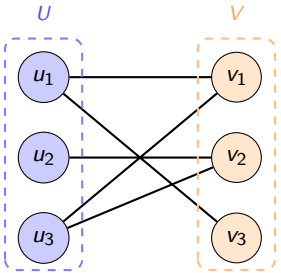
- Candidatos $\times$ Vagas de emprego.



# 1. O que é um Grafo Bipartido?

**Definição:** Grafo onde os vértices podem ser divididos em dois conjuntos $U$ e $V$ tal que **toda aresta** liga $U$ a $V$.

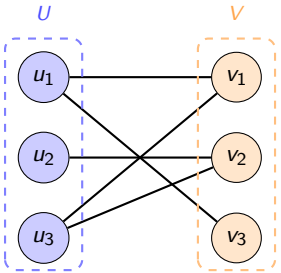
- Candidatos $\times$ Vagas de emprego.
- Alunos $\times$ Projetos.



# 1. O que é um Grafo Bipartido?

**Definição:** Grafo onde os vértices podem ser divididos em dois conjuntos $U$ e $V$ tal que **toda aresta** liga $U$ a $V$.

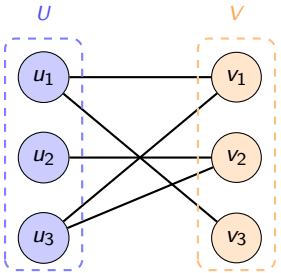
- Candidatos $\times$ Vagas de emprego.
- Alunos $\times$ Projetos.
- Médicos $\times$ Turnos hospitalares.



# 1. O que é um Grafo Bipartido?

**Definição:** Grafo onde os vértices podem ser divididos em dois conjuntos $U$ e $V$ tal que **toda aresta** liga $U$ a $V$.

- Candidatos $\times$ Vagas de emprego.
- Alunos $\times$ Projetos.
- Médicos $\times$ Turnos hospitalares.
- Motoristas $\times$ Passageiros (Uber).



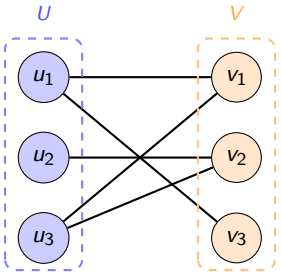
# 1. O que é um Grafo Bipartido?

**Definição:** Grafo onde os vértices podem ser divididos em dois conjuntos $U$ e $V$ tal que **toda aresta** liga $U$ a $V$.

- Candidatos $\times$ Vagas de emprego.
- Alunos $\times$ Projetos.
- Médicos $\times$ Turnos hospitalares.
- Motoristas $\times$ Passageiros (Uber).

**Critério**

Bipartido $\Leftrightarrow$ **não contém ciclo ímpar**.



# 2. Verificação: 2-Coloração via BFS

Para verificar se um grafo é bipartido, tentamos uma **2-coloração**:

## Algoritmo

# 2. Verificação: 2-Coloração via BFS

Para verificar se um grafo é bipartido, tentamos uma **2-coloração**:

## Algoritmo

1. Escolha um vértice, atribua cor 0.

# 2. Verificação: 2-Coloração via BFS

Para verificar se um grafo é bipartido, tentamos uma **2-coloração**:

## Algoritmo

1. Escolha um vértice, atribua cor 0.
2. Para cada vizinho não colorido, atribua cor oposta.

```
    int[] color = new int[V];
    Arrays.fill(color, -1);
    color[0] = 0;
    Queue<Integer> q = new LinkedList<>();
    q.add(0);
    while (!q.isEmpty()) {
        int u = q.poll();
        for (int v : adj.get(u)) {
            if (color[v] == -1) {
                color[v] = 1 - color[u];
                q.add(v);
            } else if (color[v]==color[u])
                return false; // 'impar!
        }
    }
    return true;
```

# 2. Verificação: 2-Coloração via BFS

Para verificar se um grafo é bipartido, tentamos uma **2-coloração**:

## Algoritmo

1. Escolha um vértice, atribua cor 0.
2. Para cada vizinho não colorido, atribua cor oposta.
3. Se um vizinho já tem a **mesma cor** $\Rightarrow$ **não é bipartido!**

```
1  int[] color = new int[V];
2  Arrays.fill(color, -1);
3  color[0] = 0;
4  Queue<Integer> q = new LinkedList<>();
5  q.add(0);
6  while (!q.isEmpty()) {
7      int u = q.poll();
8      for (int v : adj.get(u)) {
9          if (color[v] == -1) {
10             color[v] = 1 - color[u];
11             q.add(v);
12         } else if (color[v]==color[u])
13             return false; // 'impar!
14     }
15 }
16 return true;
```

# 2. Verificação: 2-Coloração via BFS

Para verificar se um grafo é bipartido, tentamos uma **2-coloração**:

## Algoritmo

1. Escolha um vértice, atribua cor 0.
2. Para cada vizinho não colorido, atribua cor oposta.
3. Se um vizinho já tem a **mesma cor** $\Rightarrow$ **não é bipartido!**

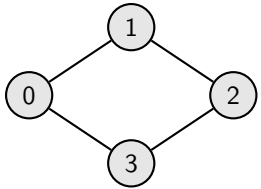
**Complexidade:** $O(V + E)$ (BFS padrão).

```
int[] color = new int[V];
Arrays.fill(color, -1);
color[0] = 0;
Queue<Integer> q = new LinkedList<>();
q.add(0);
while (!q.isEmpty()) {
    int u = q.poll();
    for (int v : adj.get(u)) {
        if (color[v] == -1) {
            color[v] = 1 - color[u];
            q.add(v);
        } else if (color[v]==color[u])
            return false; // 'impar!
    }
}
return true;
```

# 2.1 Exemplo Visual: BFS para 2-Coloração

**Passo a Passo:**

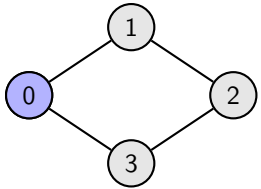
- Inicializa: Fila $=$ []



# 2.1 Exemplo Visual: BFS para 2-Coloração

**Passo a Passo:**

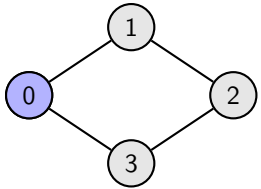
- Inicializa: Fila $=$ []
- Fonte 0 $\rightarrow$ Azul. Fila $=$ [0]



# 2.1 Exemplo Visual: BFS para 2-Coloração

**Passo a Passo:**

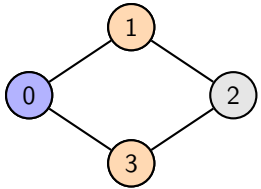
- Inicializa: Fila $=$ []
- Fonte $0 \rightarrow$ Azul. Fila $=$ [0]
- Remove 0. Vizinhos $1, 3$.



# 2.1 Exemplo Visual: BFS para 2-Coloração

**Passo a Passo:**

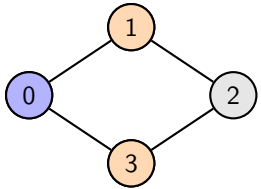
- Inicializa: Fila $=$ `[]`
- Fonte $0 \rightarrow$ Azul. Fila $=$ `[0]`
- Remove 0. Vizinhos $1, 3$.
- Pinta 1 e 3 de Laranja. Fila $=$ `[1, 3]`



# 2.1 Exemplo Visual: BFS para 2-Coloração

**Passo a Passo:**

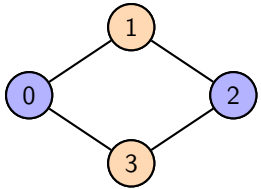
- Inicializa: Fila $=$ `[]`
- Fonte $0 \rightarrow$ Azul. Fila $=$ `[0]`
- Remove 0. Vizinhos $1, 3$.
- Pinta 1 e 3 de Laranja. Fila $=$ `[1, 3]`
- Remove 1. Vizinho 2 não visitado.



# 2.1 Exemplo Visual: BFS para 2-Coloração

**Passo a Passo:**

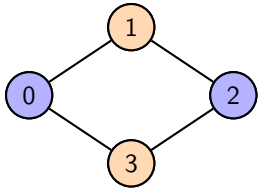
- Inicializa: Fila $=$ `[]`
- Fonte $0 \rightarrow$ Azul. Fila $=$ `[0]`
- Remove 0. Vizinhos $1, 3$.
- Pinta 1 e 3 de Laranja. Fila $=$ `[1, 3]`
- Remove 1. Vizinho 2 não visitado.
- Pinta 2 de Azul. Fila $=$ `[3, 2]`



# 2.1 Exemplo Visual: BFS para 2-Coloração

**Passo a Passo:**

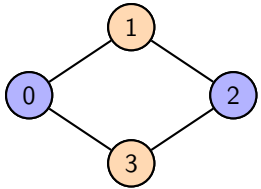
- Inicializa: Fila $=$ `[]`
- Fonte $0 \rightarrow$ Azul. Fila $=$ `[0]`
- Remove 0. Vizinhos $1, 3$.
- Pinta 1 e 3 de Laranja. Fila $=$ `[1, 3]`
- Remove 1. Vizinho 2 não visitado.
- Pinta 2 de Azul. Fila $=$ `[3, 2]`
- Remove 3 e 2. Vizinhos com cores opostas.



# 2.1 Exemplo Visual: BFS para 2-Coloração

**Passo a Passo:**

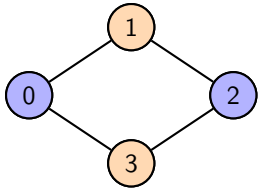
- Inicializa: Fila $=$ `[]`
- Fonte $0 \rightarrow$ Azul. Fila $=$ `[0]`
- Remove 0. Vizinhos $1, 3$.
- Pinta 1 e 3 de Laranja. Fila $=$ `[1, 3]`
- Remove 1. Vizinho 2 não visitado.
- Pinta 2 de Azul. Fila $=$ `[3, 2]`
- Remove 3 e 2. Vizinhos com cores opostas.
- Fila vazia $\Rightarrow$ **Bipartido!**



# 2.1 Exemplo Visual: BFS para 2-Coloração

**Passo a Passo:**

- Inicializa: Fila $=$ `[]`
- Fonte $0 \rightarrow$ Azul. Fila $=$ `[0]`
- Remove 0. Vizinhos $1, 3$.
- Pinta 1 e 3 de Laranja. Fila $=$ `[1, 3]`
- Remove 1. Vizinho 2 não visitado.
- Pinta 2 de Azul. Fila $=$ `[3, 2]`
- Remove 3 e 2. Vizinhos com cores opostas.
- Fila vazia $\Rightarrow$ **Bipartido!**




**E se houvesse aresta $(1, 3)$?**

Ao processar 1, veríamos vizinho 3 com a **mesma cor** (laranja). Triângulo 0-1-3 (ciclo ímpar)!

# 3. Emparelhamento Máximo (Maximum Matching)

Um **Emparelhamento** é um conjunto de arestas onde nenhum vértice aparece mais de uma vez. Queremos o **maior** possível.

**Exemplo:** 3 Devs, 3 Vagas.

- Dev A: Java, C++.
- Dev B: Python.
- Dev C: Java.

# 3. Emparelhamento Máximo (Maximum Matching)

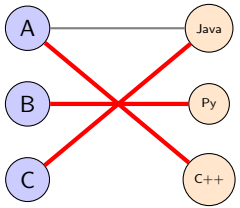
Um **Emparelhamento** é um conjunto de arestas onde nenhum vértice aparece mais de uma vez. Queremos o **maior** possível.

**Exemplo:** 3 Devs, 3 Vagas.

- Dev A: Java, C++.
- Dev B: Python.
- Dev C: Java.

**Se A → Java:** C fica sem vaga (matching = 2).

**Se A → C++:** Todos alocados (matching = 3)!




Vermelho = matching ótimo.

## Teorema de Hall

Um grafo bipartido $G = (U, V, E)$ possui um emparelhamento que cobre **todo** o conjunto $U$ se, e somente se, para todo subconjunto $S \subseteq U$, o número de vizinhos de $S$ (denotado por $N(S)$) é no mínimo o tamanho de $S$:

$$|N(S)| \geq |S| \quad \forall S \subseteq U$$

## Teorema de Hall

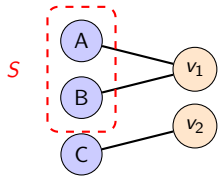
Um grafo bipartido $G = (U, V, E)$ possui um emparelhamento que cobre **todo** o conjunto $U$ se, e somente se, para todo subconjunto $S \subseteq U$, o número de vizinhos de $S$ (denotado por $N(S)$) é no mínimo o tamanho de $S$:

$$|N(S)| \geq |S| \quad \forall S \subseteq U$$

## Exemplo: Falha de Hall

$U = \{A, B, C\}$, $V = \{v_1, v_2\}$
Arestas: $A \to v_1, B \to v_1, C \to v_2$
Seja $S = \{A, B\}$. Seus vizinhos são $N(S) = \{v_1\}$.
Temos $|N(S)| = 1 < |S| = 2$.

$\Rightarrow$ Não há emparelhamento perfeito!



## 3.2 Entendendo o Teorema de Hall na Prática

### A Ideia Intuitiva (Sem Matemática!)

Imagine um grupo de amigos onde cada um vai pedir um sabor diferente de suco. O Teorema de Hall diz que **todos vão conseguir um suco** se, e somente se, **nenhum grupo de amigos for "exigente demais"**.

## 3.2 Entendendo o Teorema de Hall na Prática

### A Ideia Intuitiva (Sem Matemática!)

Imagine um grupo de amigos onde cada um vai pedir um sabor diferente de suco. O Teorema de Hall diz que **todos vão conseguir um suco** se, e somente se, **nenhum grupo de amigos for "exigente demais"**.

### O problema dos exigentes

- **Ana** e **Beto** SÓ bebem suco de Laranja.
- **Carlos** bebe Uva ou Maçã.

# 3.2 Entendendo o Teorema de Hall na Prática

## A Ideia Intuitiva (Sem Matemática!)

Imagine um grupo de amigos onde cada um vai pedir um sabor diferente de suco. O Teorema de Hall diz que **todos vão conseguir um suco** se, e somente se, **nenhum grupo de amigos for "exigente demais"**.

## O problema dos exigentes

- **Ana** e **Beto** SÓ bebem suco de Laranja.
- **Carlos** bebe Uva ou Maçã.

**O que acontece com o grupo {Ana, Beto}?**

- São **2 pessoas**.
- Juntos, eles só aceitam **1 opção** (Laranja).
- Como 2 pessoas estão brigando por 1 única opção, **alguém vai ficar sem suco!**

## A Ideia Intuitiva (Sem Matemática!)

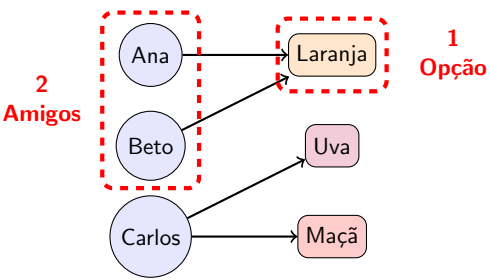
Imagine um grupo de amigos onde cada um vai pedir um sabor diferente de suco. O Teorema de Hall diz que **todos vão conseguir um suco** se, e somente se, **nenhum grupo de amigos for "exigente demais"**.

## O problema dos exigentes

- **Ana** e **Beto** SÓ bebem suco de Laranja.
- **Carlos** bebe Uva ou Maçã.

**O que acontece com o grupo {Ana, Beto}?**

- São **2 pessoas**.
- Juntos, eles só aceitam **1 opção** (Laranja).
- Como 2 pessoas estão brigando por 1 única opção, **alguém vai ficar sem suco!**



# Spoiler: O que é Fluxo Máximo?

## Uma intuição rápida (Veremos com calma no futuro!)

Embora **Fluxo Máximo** seja um tópico de próximas aulas, a sua ideia principal é simples e nos ajuda a resolver o emparelhamento.

# Spoiler: O que é Fluxo Máximo?

## Uma intuição rápida (Veremos com calma no futuro!)

Embora **Fluxo Máximo** seja um tópico de próximas aulas, a sua ideia principal é simples e nos ajuda a resolver o emparelhamento.

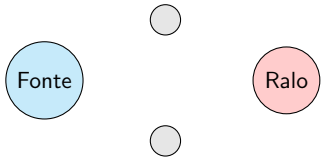
- Imagine uma rede de **canos de água**.

# Spoiler: O que é Fluxo Máximo?

## Uma intuição rápida (Veremos com calma no futuro!)

Embora **Fluxo Máximo** seja um tópico de próximas aulas, a sua ideia principal é simples e nos ajuda a resolver o emparelhamento.

- Imagine uma rede de **canos de água**.
- Existe uma **Fonte** (onde a água entra) e um **Ralo** ou Sumidouro (para onde a água vai).

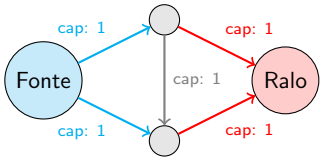


# Spoiler: O que é Fluxo Máximo?

## Uma intuição rápida (Veremos com calma no futuro!)

Embora **Fluxo Máximo** seja um tópico de próximas aulas, a sua ideia principal é simples e nos ajuda a resolver o emparelhamento.

- Imagine uma rede de **canos de água**.
- Existe uma **Fonte** (onde a água entra) e um **Ralo** ou Sumidouro (para onde a água vai).
- Cada cano possui uma **capacidade limite** (ex: 1 litro/segundo).

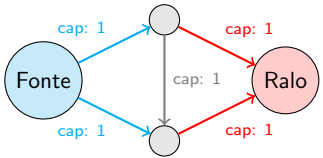


# Spoiler: O que é Fluxo Máximo?

## Uma intuição rápida (Veremos com calma no futuro!)

Embora **Fluxo Máximo** seja um tópico de próximas aulas, a sua ideia principal é simples e nos ajuda a resolver o emparelhamento.

- Imagine uma rede de **canos de água**.
- Existe uma **Fonte** (onde a água entra) e um **Ralo** ou Sumidouro (para onde a água vai).
- Cada cano possui uma **capacidade limite** (ex: 1 litro/segundo).
- **O Problema:** Qual é a quantidade *máxima* de água que conseguimos empurrar da Fonte até o Ralo sem estourar nenhum cano?



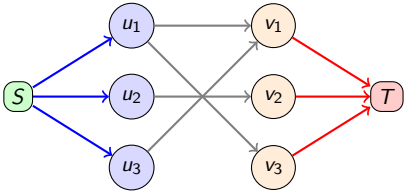
# 4. Redução para Fluxo Máximo

Podemos resolver matching via **Fluxo Máximo**:

# 4. Redução para Fluxo Máximo

Podemos resolver matching via **Fluxo Máximo**:

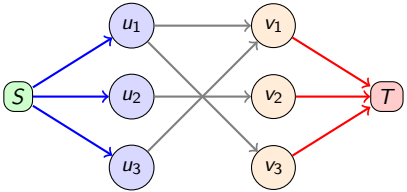
1. Super-Source $S \rightarrow$ todos de $U$ (cap. 1).



# 4. Redução para Fluxo Máximo

Podemos resolver matching via **Fluxo Máximo**:

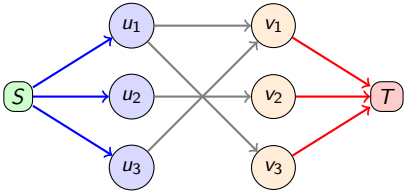
1. Super-Source $S \rightarrow$ todos de $U$ (cap. 1).
2. Todos de $V \rightarrow$ Super-Sink $T$ (cap. 1).



# 4. Redução para Fluxo Máximo

Podemos resolver matching via **Fluxo Máximo**:

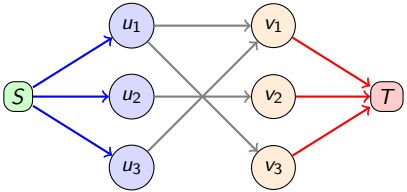
1. Super-Source $S \rightarrow$ todos de $U$ (cap. 1).
2. Todos de $V \rightarrow$ Super-Sink $T$ (cap. 1).
3. Arestas $U \rightarrow V$ originais (cap. 1).



# 4. Redução para Fluxo Máximo

Podemos resolver matching via **Fluxo Máximo**:

1. Super-Source $S \rightarrow$ todos de $U$ (cap. 1).
2. Todos de $V \rightarrow$ Super-Sink $T$ (cap. 1).
3. Arestas $U \rightarrow V$ originais (cap. 1).
4. **Fluxo Máximo = Matching Máximo!**

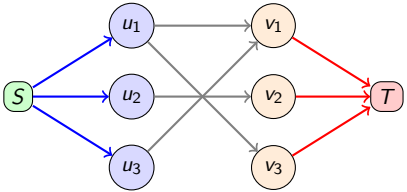


# 4. Redução para Fluxo Máximo

Podemos resolver matching via **Fluxo Máximo**:

1. Super-Source $S \to$ todos de $U$ (cap. 1).
2. Todos de $V \to$ Super-Sink $T$ (cap. 1).
3. Arestas $U \to V$ originais (cap. 1).
4. **Fluxo Máximo = Matching Máximo!**

Cap. 1 garante: cada vértice em no máximo 1 par.

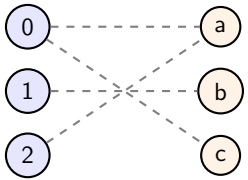


# 5. Algoritmo de Kuhn: Caminho Aumentante

## Intuição

Para cada vértice de $U$, procure um vizinho livre em $V$. Se ocupado, peça ao dono atual tentar **trocar** seu par (via DFS). A cadeia de trocas bem-sucedida é um **caminho aumentante**.

**Trace Passo a Passo:** $U = \{0, 1, 2\}$, $V = \{a, b, c\}$



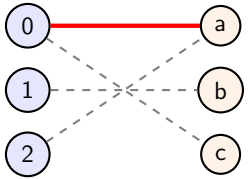
# 5. Algoritmo de Kuhn: Caminho Aumentante

## Intuição

Para cada vértice de $U$, procure um vizinho livre em $V$. Se ocupado, peça ao dono atual tentar **trocar** seu par (via DFS). A cadeia de trocas bem-sucedida é um **caminho aumentante**.

**Trace Passo a Passo:** $U = \{0, 1, 2\}$, $V = \{a, b, c\}$

- **u=0**: vizinho $a$ livre. Faz par $(0, a)$.



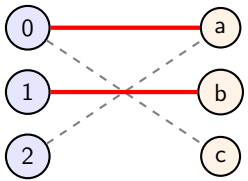
# 5. Algoritmo de Kuhn: Caminho Aumentante

## Intuição

Para cada vértice de $U$, procure um vizinho livre em $V$. Se ocupado, peça ao dono atual tentar **trocar** seu par (via DFS). A cadeia de trocas bem-sucedida é um **caminho aumentante**.

**Trace Passo a Passo:** $U = \{0, 1, 2\}$, $V = \{a, b, c\}$

- **u=0**: vizinho $a$ livre. Faz par $(0, a)$.
- **u=1**: vizinho $b$ livre. Faz par $(1, b)$.

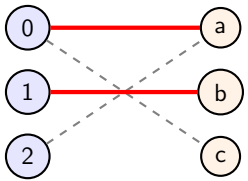


## Intuição

Para cada vértice de $U$, procure um vizinho livre em $V$. Se ocupado, peça ao dono atual tentar **trocar** seu par (via DFS). A cadeia de trocas bem-sucedida é um **caminho aumentante**.

**Trace Passo a Passo:** $U = \{0, 1, 2\}$, $V = \{a, b, c\}$

- **u=0**: vizinho $a$ livre. Faz par $(0, a)$.
- **u=1**: vizinho $b$ livre. Faz par $(1, b)$.
- **u=2**: vizinho $a$ ocupado por 0.



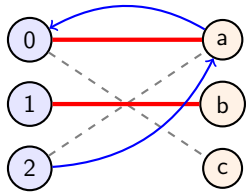
# 5. Algoritmo de Kuhn: Caminho Aumentante

## Intuição

Para cada vértice de $U$, procure um vizinho livre em $V$. Se ocupado, peça ao dono atual tentar **trocar** seu par (via DFS). A cadeia de trocas bem-sucedida é um **caminho aumentante**.

**Trace Passo a Passo:** $U = \{0, 1, 2\}$, $V = \{a, b, c\}$

- **u=0**: vizinho $a$ livre. Faz par $(0, a)$.
- **u=1**: vizinho $b$ livre. Faz par $(1, b)$.
- **u=2**: vizinho $a$ ocupado por 0.
  - $2 \to a \to 0$. Pede a 0 para mudar.



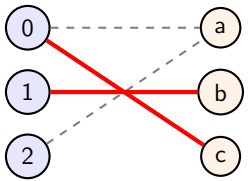
# 5. Algoritmo de Kuhn: Caminho Aumentante

## Intuição

Para cada vértice de $U$, procure um vizinho livre em $V$. Se ocupado, peça ao dono atual tentar **trocar** seu par (via DFS). A cadeia de trocas bem-sucedida é um **caminho aumentante**.

**Trace Passo a Passo:** $U = \{0, 1, 2\}$, $V = \{a, b, c\}$

- **u=0**: vizinho $a$ livre. Faz par $(0, a)$.
- **u=1**: vizinho $b$ livre. Faz par $(1, b)$.
- **u=2**: vizinho $a$ ocupado por 0.
  - $2 \to a \to 0$. Pede a 0 para mudar.
  - 0 tem vizinho $c$ livre! Faz par $(0, c)$.



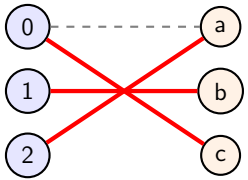
# 5. Algoritmo de Kuhn: Caminho Aumentante

## Intuição

Para cada vértice de $U$, procure um vizinho livre em $V$. Se ocupado, peça ao dono atual tentar **trocar** seu par (via DFS). A cadeia de trocas bem-sucedida é um **caminho aumentante**.

**Trace Passo a Passo:** $U = \{0, 1, 2\}$, $V = \{a, b, c\}$

- **u=0**: vizinho $a$ livre. Faz par $(0, a)$.
- **u=1**: vizinho $b$ livre. Faz par $(1, b)$.
- **u=2**: vizinho $a$ ocupado por 0.
  - $2 \to a \to 0$. Pede a 0 para mudar.
  - 0 tem vizinho $c$ livre! Faz par $(0, c)$.
  - Agora $a$ ficou livre. Faz par $(2, a)$.



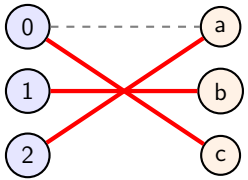
# 5. Algoritmo de Kuhn: Caminho Aumentante

## Intuição

Para cada vértice de $U$, procure um vizinho livre em $V$. Se ocupado, peça ao dono atual tentar **trocar** seu par (via DFS). A cadeia de trocas bem-sucedida é um **caminho aumentante**.

**Trace Passo a Passo:** $U = \{0, 1, 2\}$, $V = \{a, b, c\}$

- **u=0**: vizinho $a$ livre. Faz par $(0, a)$.
- **u=1**: vizinho $b$ livre. Faz par $(1, b)$.
- **u=2**: vizinho $a$ ocupado por 0.
  - $2 \to a \to 0$. Pede a 0 para mudar.
  - 0 tem vizinho $c$ livre! Faz par $(0, c)$.
  - Agora $a$ ficou livre. Faz par $(2, a)$.
- **Caminho:** $2 \to a \to 0 \to c$. **Matching = 3.**



# 6. Implementação Java – Kuhn

```java
int[] pairV;        // pairV[v] = quem de U est'a com v (-1 = livre)
boolean[] visited; // Evitar ciclos na DFS

boolean dfs(int u) {
    visited[u] = true;
    for (int v : adj.get(u)) {
        if (pairV[v] < 0 || (!visited[pairV[v]] && dfs(pairV[v]))) {
            pairV[v] = u; // Match!
            return true;
        }
    }
    return false;
}

int maxMatching() {
    int result = 0;
    for (int u = 0; u < n; u++) {
        Arrays.fill(visited, false);
        if (dfs(u)) result++;
    }
    return result;
}
```

# 6.1 Otimização: Algoritmo de Hopcroft-Karp

- Kuhn encontra **um caminho aumentante** por vez via DFS. $\mathcal{O}(V \times E)$.
- Hopcroft-Karp encontra **vários caminhos disjuntos** simultaneamente.

## Como funciona?

# 6.1 Otimização: Algoritmo de Hopcroft-Karp

- Kuhn encontra **um caminho aumentante** por vez via DFS. $\mathcal{O}(V \times E)$.
- Hopcroft-Karp encontra **vários caminhos disjuntos** simultaneamente.

## Como funciona?

1. Usa **BFS** para construir um "Grafo de Camadas" apenas com arestas não pareadas $(U \to V)$ e pareadas $(V \to U)$.

# 6.1 Otimização: Algoritmo de Hopcroft-Karp

- Kuhn encontra **um caminho aumentante** por vez via DFS. $\mathcal{O}(V \times E)$.
- Hopcroft-Karp encontra **vários caminhos disjuntos** simultaneamente.

## Como funciona?

1. Usa **BFS** para construir um "Grafo de Camadas" apenas com arestas não pareadas $(U \to V)$ e pareadas $(V \to U)$.
2. Encontra o comprimento do menor caminho aumentante.

# 6.1 Otimização: Algoritmo de Hopcroft-Karp

- Kuhn encontra **um caminho aumentante** por vez via DFS. $\mathcal{O}(V \times E)$.
- Hopcroft-Karp encontra **vários caminhos disjuntos** simultaneamente.

## Como funciona?

1. Usa **BFS** para construir um "Grafo de Camadas" apenas com arestas não pareadas $(U \to V)$ e pareadas $(V \to U)$.
2. Encontra o comprimento do menor caminho aumentante.
3. Usa **DFS** restrita a esse grafo de camadas para encontrar um **conjunto maximal de caminhos disjuntos** do mesmo comprimento.

# 6.1 Otimização: Algoritmo de Hopcroft-Karp

- Kuhn encontra **um caminho aumentante** por vez via DFS. $\mathcal{O}(V \times E)$.
- Hopcroft-Karp encontra **vários caminhos disjuntos** simultaneamente.

## Como funciona?

1. Usa **BFS** para construir um "Grafo de Camadas" apenas com arestas não pareadas $(U \to V)$ e pareadas $(V \to U)$.
2. Encontra o comprimento do menor caminho aumentante.
3. Usa **DFS** restrita a esse grafo de camadas para encontrar um **conjunto maximal de caminhos disjuntos** do mesmo comprimento.
4. Aumenta o matching usando todos esses caminhos de uma vez.

# 6.1 Otimização: Algoritmo de Hopcroft-Karp

- Kuhn encontra **um caminho aumentante** por vez via DFS. $\mathcal{O}(V \times E)$.
- Hopcroft-Karp encontra **vários caminhos disjuntos** simultaneamente.

## Como funciona?

1. Usa **BFS** para construir um "Grafo de Camadas" apenas com arestas não pareadas ($U \to V$) e pareadas ($V \to U$).
2. Encontra o comprimento do menor caminho aumentante.
3. Usa **DFS** restrita a esse grafo de camadas para encontrar um **conjunto maximal de caminhos disjuntos** do mesmo comprimento.
4. Aumenta o matching usando todos esses caminhos de uma vez.

**Complexidade:** $\mathcal{O}(E\sqrt{V})$.
Muito mais rápido para grafos grandes ou densos, e frequentemente obrigatório em problemas de Maratona mais rigorosos.

**Teorema de König**

Em grafos bipartidos: **Emparelhamento Máximo** = **Cobertura Mínima de Vértices**.

## Teorema de König

Em grafos bipartidos: **Emparelhamento Máximo = Cobertura Mínima de Vértices**.

## O que é Cobertura de Vértices?

Menor conjunto $S$ de vértices tal que **toda aresta** tem pelo menos um extremo em $S$.
Exemplo: "Onde colocar câmeras para cobrir todas as ruas?"

## Teorema de König

Em grafos bipartidos: **Emparelhamento Máximo = Cobertura Mínima de Vértices**.

## O que é Cobertura de Vértices?

Menor conjunto $S$ de vértices tal que **toda aresta** tem pelo menos um extremo em $S$.
Exemplo: "Onde colocar câmeras para cobrir todas as ruas?"

## Dualidade com Conjunto Independente

|Conjunto Independente Máximo| = $|V|$ - |Cobertura Mínima| = $|V|$ - |Matching Máximo|.
Todas essas quantidades se relacionam em bipartidos!

### Engenharia

- Escalonamento de tarefas.
- Alocação de VMs a servidores.
- Motoristas $\times$ passageiros (Uber).
- Alocação de registradores (compiladores).

### Maratona de Programação

- "Alocar" ou "parear" $\Rightarrow$ matching.
- Peões em tabuleiro sem conflito.
- Cobrir todas as arestas com mínimo de vértices.

# 8. Aplicações Práticas

## Engenharia

- Escalonamento de tarefas.
- Alocação de VMs a servidores.
- Motoristas $\times$ passageiros (Uber).
- Alocação de registradores (compiladores).

## Maratona de Programação

- "Alocar" ou "parear" $\Rightarrow$ matching.
- Peões em tabuleiro sem conflito.
- Cobrir todas as arestas com mínimo de vértices.

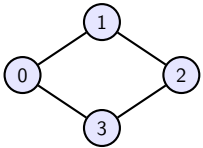
## Dica de Maratona

Se o enunciado fala em **dois conjuntos distintos** e **alocação/pareamento**, pense em Emparelhamento Bipartido!

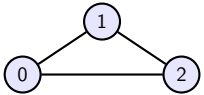
# Exercício 1: Verificação de Bipartição

Determine se os grafos abaixo são bipartidos. Justifique.

**Grafo 1:**



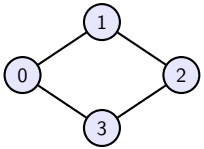

**Grafo 2:**



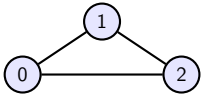
# Exercício 1: Verificação de Bipartição

Determine se os grafos abaixo são bipartidos. Justifique.

**Grafo 1:**




**Grafo 2:**




## Resposta

**Grafo 1:** **Bipartido** (ciclo de tamanho 4, par). $U = \{0, 2\}$, $V = \{1, 3\}$.
**Grafo 2:** **Não bipartido** (triângulo = ciclo ímpar de tamanho 3).

# Exercício 2: Alocação de Viaturas (Matching)

## Problema

$U = \{0, 1, 2, 3\}$ (Viaturas), $V = \{a, b, c, d\}$ (Ocorrências).
Viatura pode atender ocorrência: (0,a), (0,b), (1,a), (1,c), (2,b), (2,d), (3,c).

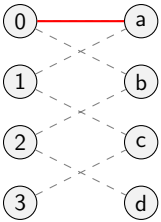
# Exercício 2: Alocação de Viaturas (Matching)

## Problema

$U = \{0, 1, 2, 3\}$ (Viaturas), $V = \{a, b, c, d\}$ (Ocorrências).
Viatura pode atender ocorrência: (0,a), (0,b), (1,a), (1,c), (2,b), (2,d), (3,c).


## Trace (Algoritmo de Kuhn)

1. $u = 0$: $a$ livre $\Rightarrow$ `match[a]=0`.



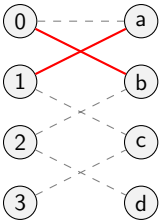
# Exercício 2: Alocação de Viaturas (Matching)

## Problema

$U = \{0, 1, 2, 3\}$ (Viaturas), $V = \{a, b, c, d\}$ (Ocorrências).
Viatura pode atender ocorrência: (0,a), (0,b), (1,a), (1,c), (2,b), (2,d), (3,c).


## Trace (Algoritmo de Kuhn)

1. $u = 0$: $a$ livre $\Rightarrow$ `match[a]=0`.
2. $u = 1$: $a$ ocupada. $1 \to a \to 0$. Pede a 0 para trocar. 0 move para $b$ livre $\Rightarrow$ `match[b]=0, match[a]=1`.



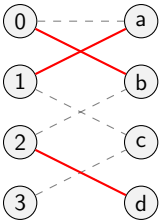
# Exercício 2: Alocação de Viaturas (Matching)

## Problema

$U = \{0, 1, 2, 3\}$ (Viaturas), $V = \{a, b, c, d\}$ (Ocorrências).
Viatura pode atender ocorrência: (0,a), (0,b), (1,a), (1,c), (2,b), (2,d), (3,c).


## Trace (Algoritmo de Kuhn)

1. $u = 0$: $a$ livre $\Rightarrow$ `match[a]=0`.
2. $u = 1$: $a$ ocupada. $1 \to a \to 0$. Pede a 0 para trocar. 0 move para $b$ livre $\Rightarrow$ `match[b]=0, match[a]=1`.
3. $u = 2$: tem arestas para $b, d$. O caminho por $b$ geraria mais trocas, mas tentando $d$ acha livre $\Rightarrow$ `match[d]=2`.



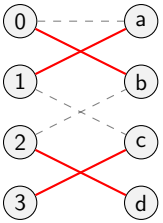
# Exercício 2: Alocação de Viaturas (Matching)

## Problema

$U = \{0, 1, 2, 3\}$ (Viaturas), $V = \{a, b, c, d\}$ (Ocorrências).
Viatura pode atender ocorrência: (0,a), (0,b), (1,a), (1,c), (2,b), (2,d), (3,c).


## Trace (Algoritmo de Kuhn)

1. $u = 0$: $a$ livre $\Rightarrow$ `match[a]=0`.
2. $u = 1$: $a$ ocupada. $1 \to a \to 0$. Pede a 0 para trocar. 0 move para $b$ livre $\Rightarrow$ `match[b]=0, match[a]=1`.
3. $u = 2$: tem arestas para $b, d$. O caminho por $b$ geraria mais trocas, mas tentando $d$ acha livre $\Rightarrow$ `match[d]=2`.
4. $u = 3$: $c$ livre $\Rightarrow$ `match[c]=3`.



# Exercício 2: Alocação de Viaturas (Matching)

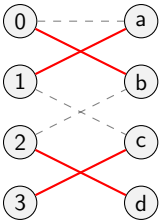
## Problema

$U = \{0, 1, 2, 3\}$ (Viaturas), $V = \{a, b, c, d\}$ (Ocorrências).
Viatura pode atender ocorrência: (0,a), (0,b), (1,a), (1,c), (2,b), (2,d), (3,c).


## Trace (Algoritmo de Kuhn)

1. $u = 0$: $a$ livre $\Rightarrow$ `match[a]=0`.
2. $u = 1$: $a$ ocupada. $1 \to a \to 0$. Pede a 0 para trocar. 0 move para $b$ livre $\Rightarrow$ `match[b]=0, match[a]=1`.
3. $u = 2$: tem arestas para $b, d$. O caminho por $b$ geraria mais trocas, mas tentando $d$ acha livre $\Rightarrow$ `match[d]=2`.
4. $u = 3$: $c$ livre $\Rightarrow$ `match[c]=3`.

**Matching Final:** $\{0 \to b, 1 \to a, 2 \to d, 3 \to c\}$. Tamanho = **4**.



# Exercício 3: Modelagem – Tabuleiro de Xadrez

## Problema

Dado um tabuleiro $N \times N$ com $K$ casas bloqueadas, qual o número máximo de torres que podem ser colocadas sem que duas se ataquem?

*Dica: Torres atacam na mesma linha ou coluna.*

# Exercício 3: Modelagem – Tabuleiro de Xadrez

## Problema

Dado um tabuleiro $N \times N$ com $K$ casas bloqueadas, qual o número máximo de torres que podem ser colocadas sem que duas se ataquem?

*Dica: Torres atacam na mesma linha ou coluna.*

## Resposta

**Modelagem:** Linhas $= U$, Colunas $= V$.
Aresta $(i, j)$ se casa $(i, j)$ **não** está bloqueada.
**Matching Máximo** = torres sem conflito.
**Algoritmo:** Kuhn com $O(N^3)$.

# Exercício 3: Modelagem – Tabuleiro de Xadrez

## Problema

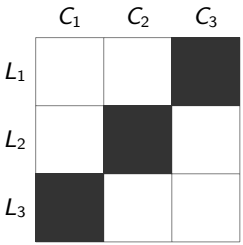
Dado um tabuleiro $N \times N$ com $K$ casas bloqueadas, qual o número máximo de torres que podem ser colocadas sem que duas se ataquem?

*Dica: Torres atacam na mesma linha ou coluna.*

## Resposta

**Modelagem:** Linhas $= U$, Colunas $= V$.
Aresta $(i, j)$ se casa $(i, j)$ **não** está bloqueada.
**Matching Máximo** = torres sem conflito.
**Algoritmo:** Kuhn com $O(N^3)$.

**Exemplo 3x3**



# Exercício 3: Modelagem – Tabuleiro de Xadrez

## Problema

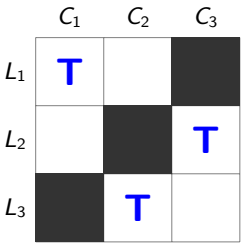
Dado um tabuleiro $N \times N$ com $K$ casas bloqueadas, qual o número máximo de torres que podem ser colocadas sem que duas se ataquem?

*Dica: Torres atacam na mesma linha ou coluna.*

## Resposta

**Modelagem:** Linhas $= U$, Colunas $= V$.
Aresta $(i, j)$ se casa $(i, j)$ **não** está bloqueada.
**Matching Máximo** = torres sem conflito.
**Algoritmo:** Kuhn com $O(N^3)$.

**Exemplo 3x3**



# Exercício 3: Modelagem – Tabuleiro de Xadrez

## Problema

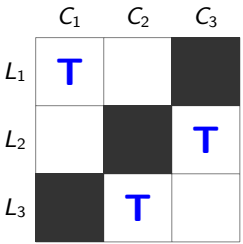
Dado um tabuleiro $N \times N$ com $K$ casas bloqueadas, qual o número máximo de torres que podem ser colocadas sem que duas se ataquem?

*Dica: Torres atacam na mesma linha ou coluna.*

## Resposta

**Modelagem:** Linhas $= U$, Colunas $= V$.
Aresta $(i, j)$ se casa $(i, j)$ **não** está bloqueada.
**Matching Máximo** = torres sem conflito.
**Algoritmo:** Kuhn com $O(N^3)$.

**Exemplo 3x3**




**Solução: 3 Torres**