Algoritmos e Estruturas de Dados II

Capítulo da Aula 22: Grafos Bipartidos e Emparelhamento

Prof. Aléssio Miranda Júnior
alessio@cefetmg.br
CEFET-MG - Campus Timóteo

Fevereiro de 2026

1. Motivação: Onde encontramos esses problemas?

Muitos problemas do dia a dia envolvem separar elementos em grupos ou conectar **dois grupos distintos** de entidades.

Separação de Conflitos (Armazenamento Químico)

Imagine que você precisa armazenar um conjunto de reagentes químicos em armários. Alguns reagentes **reagem perigosamente entre si** (como Amônia e Ácido Clorídrico) e não podem ficar na mesma caixa. É possível organizar todos esses reagentes em apenas **duas caixas** seguras? Este é um clássico problema de **Verificação de Bipartição** (ou 2-Coloração).

Alocação de Registradores em Compiladores

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 (conflito). Se temos apenas K registradores, conseguimos compilar esse código? Quando $K = 2$, isso equivale a verificar se o grafo de interferência é bipartido.

Escala de Plantões (Gestão Hospitalar)

Um hospital precisa alocar **médicos** aos seus respectivos **turnos**. Cada médico só tem disponibilidade em alguns horários específicos. Como designar exatamente um turno para cada médico sem deixar o hospital vazio em nenhum momento? Este é um problema de **Emparelhamento** (Matching), onde buscamos maximizar as conexões válidas.

2. O que é um Grafo Bipartido?

Definição: Um grafo $G = (V, E)$ é **bipartido** se podemos particionar seus vértices em dois conjuntos disjuntos U e V tal que **toda aresta** conecta um vértice de U a um vértice de V. Não existem arestas dentro do mesmo conjunto.

Exemplos do Mundo Real

Grafos bipartidos modelam naturalmente relacionamentos entre **duas classes distintas** de entidades:

- **Candidatos × Vagas de Emprego:** Candidatos formam U, vagas formam V, arestas representam competência.
- **Alunos × Projetos:** Cada aluno pode ser alocado a projetos compatíveis.
- **Médicos × Turnos:** Escalonamento hospitalar.
- **Homens × Mulheres:** O clássico problema do casamento estável (Gale-Shapley).
- **Servidores × Tarefas:** Balanceamento de carga em sistemas distribuídos.

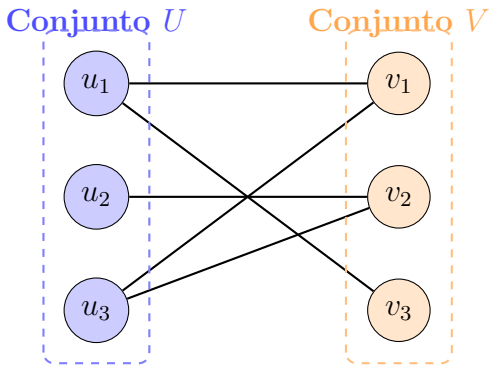

Figure 1: Grafo bipartido: vértices divididos em dois conjuntos U e V; arestas apenas entre os conjuntos.

3. Verificação de Bipartição (2-Coloração)

• Teoria

Teorema: Um grafo é bipartido se e somente se **não contém ciclos de comprimento ímpar**.

Para verificar se um grafo é bipartido, usamos uma **2-coloração** via BFS ou DFS:

1 Escolha um vértice e atribua cor 0.

2 Para cada vizinho não colorido, atribua a cor oposta.

3 Se encontrar um vizinho que já tem a **mesma cor**, o grafo **não é bipartido**.

Implementação Java (BFS)

```java
public static boolean isBipartite(int V, List<List<Integer>> adj) {
    int[] color = new int[V];
    Arrays.fill(color, -1); // -1 = não colorido

    for (int start = 0; start < V; start++) {
        if (color[start] != -1) continue; // já visitado

        Queue<Integer> queue = new LinkedList<>();
        queue.add(start);
        color[start] = 0;

        while (!queue.isEmpty()) {
            int u = queue.poll();
            for (int v : adj.get(u)) {
                if (color[v] == -1) {
                    color[v] = 1 - color[u]; // Cor oposta
                    queue.add(v);
                } else if (color[v] == color[u]) {
                    return false; // Ciclo ímpar!
                }
            }
        }
    }
    return true;
}
```

Complexidade: $O(V + E)$ — idêntica a uma BFS normal.

4. Emparelhamento Máximo (Maximum Bipartite Matching)

Um **Emparelhamento** (Matching) é um conjunto de arestas onde **nenhum vértice aparece mais de uma vez**. O **Emparelhamento Máximo** é aquele com o maior número possível de pares.

Exemplo Motivador

Temos 3 programadores e 3 vagas:

- Dev A sabe Java e C++.
- Dev B sabe Python.
- Dev C sabe Java.
- Vaga 1: Java. Vaga 2: Python. Vaga 3: C++.

Queremos alocar o **maior número de DEVs** possível. A resposta ótima é 3 (todos alocados):

- Dev A → Vaga 3 (C++), Dev B → Vaga 2 (Python), Dev C → Vaga 1 (Java).

Note que se alocássemos Dev A para Java (Vaga 1), Dev C ficaria sem vaga — matching de tamanho 2, subótimo.

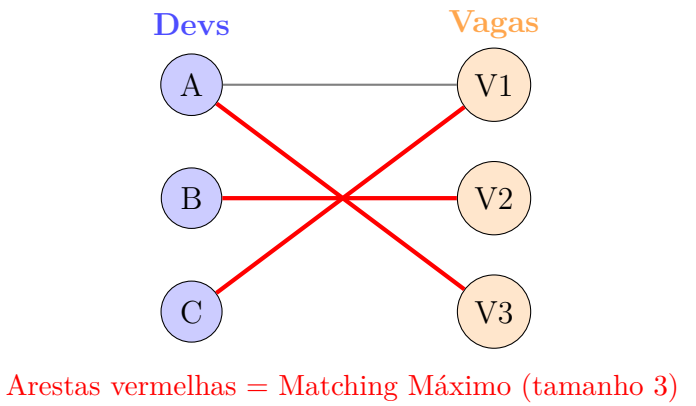

Figure 2: Emparelhamento máximo: cada dev alocado a exatamente uma vaga.

5. Redução para Fluxo Máximo

O emparelhamento máximo bipartido pode ser reduzido a um problema de **Fluxo Máximo**:

1. Crie um vértice **Super-Source** (S) e um **Super-Sink** (T).
2. Adicione arestas de S para todos os vértices de U com capacidade 1.
3. Adicione arestas de todos os vértices de V para T com capacidade 1.
4. Mantenha as arestas originais ($U \to V$) com capacidade 1.
5. O **Fluxo Máximo** de S para T é igual ao tamanho do **Emparelhamento Máximo**.

A capacidade 1 nas arestas garante que cada vértice participa de no máximo um par.

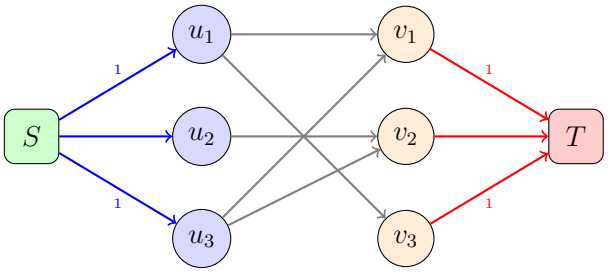

Figure 3: Redução a fluxo máximo: Fluxo Máximo = Emparelhamento Máximo.

6. Algoritmo de Kuhn (Caminhos Aumentantes)

Embora possamos usar algoritmos de fluxo (Dinic, Edmonds-Karp), o **algoritmo de Kuhn** é muito mais simples e eficiente na prática para emparelhamento bipartido.

Intuição: O Jogo das Cadeiras

Imagine um jogo onde cada pessoa (U) tenta sentar numa cadeira (V):

1. Para cada pessoa, tente encontrar uma cadeira livre compatível.
2. Se todas as cadeiras compatíveis estão ocupadas, peça ao ocupante que **tente trocar para outra cadeira** (recursivamente).
3. Se a troca deu certo em cascata, todos ficam sentados. Caso contrário, esta pessoa fica em pé.

Esse "pedir para trocar" é o conceito de **Caminho Aumentante**.

Demonstração Passo a Passo

Considere: $U = \{0, 1, 2\}$, $V = \{a, b, c\}$.
Arestas: $0 \to a$, $0 \to c$, $1 \to b$, $2 \to a$.

1. **Tentativa para $u = 0$:** Cadeira a está livre $\Rightarrow$ `match[a] = 0`. **Sucesso!**
2. **Tentativa para $u = 1$:** Cadeira b está livre $\Rightarrow$ `match[b] = 1`. **Sucesso!**
3. **Tentativa para $u = 2$:** Cadeira a está ocupada por $u = 0$. Pedimos a $u = 0$ que troque.
   - $u = 0$ tenta cadeira $c \Rightarrow$ livre! `match[c] = 0`.
   - Agora a está livre $\Rightarrow$ `match[a] = 2`. **Sucesso!**

Matching final: $\{0 \to c, 1 \to b, 2 \to a\}$. Tamanho = 3 (máximo).

Implementação Java

```java
import java.util.*;

public class BipartiteMatching {

    int n, m; // n = tamanho de U, m = tamanho de V
    List<List<Integer>> adj;
    int[] pairV; // pairV[v] = quem de U est'a casado com v (-1 = livre)
    boolean[] visited; // Para evitar ciclos na DFS

    public BipartiteMatching(int n, int m) {
        this.n = n;
        this.m = m;
```

```
        adj = new ArrayList<>(n);
        for(int i=0; i<n; i++) adj.add(new ArrayList<>());
        pairV = new int[m];
        Arrays.fill(pairV, -1);
        visited = new boolean[n];
    }

    public void addEdge(int u, int v) {
        adj.get(u).add(v); // u em [0..n-1], v em [0..m-1]
    }

    // Tenta encontrar um par (ou caminho aumentante) para u
    boolean dfs(int u) {
        visited[u] = true;
        for (int v : adj.get(u)) {
            // Se v est'a livre OU quem est'a com v pode mudar
            if (pairV[v] < 0 || (!visited[pairV[v]] && dfs(pairV[v]))) {
                pairV[v] = u; // Match!
                return true;
            }
        }
        return false;
    }

    // Retorna o tamanho do matching m'aximo
    public int maxMatching() {
        int result = 0;
        for (int u = 0; u < n; u++) {
            Arrays.fill(visited, false); // Reset a cada tentativa
            if (dfs(u)) result++;
        }
        return result;
    }
}
```

Complexidade

- **Pior caso:** $O(V \times E)$ — para cada vértice de U, a DFS percorre até E arestas.
- **Caso prático:** Geralmente muito mais rápido, comportando-se quase como $O(E)$.
- **Alternativa mais rápida:** O algoritmo de **Hopcroft-Karp** atinge $O(E\sqrt{V})$, mas é significativamente mais complexo de implementar.

7. Teorema de König e Cobertura Mínima

• Teoria

Teorema de König: Em qualquer grafo bipartido, o tamanho do **Emparelhamento Máximo** é igual ao tamanho da **Cobertura Mínima de Vértices** (Minimum Vertex Cover).

O que é Cobertura de Vértices?

Uma **cobertura de vértices** é um conjunto S de vértices tal que **toda aresta** do grafo tem pelo menos um extremo em S. A cobertura **mínima** é a menor possível.

Aplicação Prática

"Em quais cruzamentos devemos instalar câmeras para monitorar **todas as ruas**?"

- Cruzamentos = vértices, ruas = arestas.
- Manhattan é um grid, que é bipartido (linhas pares vs ímpares).
- Pelo Teorema de König: basta resolver o emparelhamento máximo para saber quantas câmeras são necessárias.

Relação com Conjunto Independente Máximo

Existe uma dualidade: **Conjunto Independente Máximo** = V - **Cobertura Mínima de Vértices**.

Um conjunto independente é um conjunto de vértices sem arestas entre eles. Em grafos bipartidos, graças ao Teorema de König:

$$|MIS| = |V| - |MVC| = |V| - |MM|$$

onde MIS = Maximum Independent Set, MVC = Minimum Vertex Cover, MM = Maximum Matching.

8. Aplicações em Engenharia de Computação

- **Alocação de Recursos:** Atribuir tarefas a trabalhadores, máquinas virtuais a servidores, canais a frequências.
- **Sistemas de Recomendação:** Emparelhar usuários com produtos, motoristas com passageiros (Uber/Lyft).
- **Compiladores:** Alocação de registradores usa coloração de grafos de interferência.
- **Redes:** Alocação de largura de banda, escalonamento de pacotes em switches.
- **Bioinformática:** Alinhamento de sequências de proteínas.

▶ Prática

Dica de Maratona: Quando o enunciado fala em "alocar", "atribuir", ou "parear" dois conjuntos distintos, pense imediatamente em **Emparelhamento Bipartido**!

9. Exercícios

9.1. Exercícios Conceituais

1 Prove que um grafo é bipartido se e somente se não contém ciclos de comprimento ímpar. *Dica: use a 2-coloração.*

2 Explique o algoritmo de Kuhn usando a analogia do "jogo das cadeiras". Por que ele encontra o emparelhamento máximo?

3 Descreva a redução de emparelhamento bipartido para fluxo máximo. Por que as capacidades devem ser 1?

4 Enuncie o Teorema de König e explique sua utilidade prática.

9.2. Exercícios Analíticos

1 **Alocação de Viaturas:** Para o grafo bipartido representando $U = \{0, 1, 2, 3\}$ (Viaturas) e $V = \{a, b, c, d\}$ (Ocorrências), onde as viaturas podem atender:

```
Arestas: (0,a), (0,b), (1,a), (1,c), (2,b), (2,d), (3,c)
```

Encontre o emparelhamento máximo usando o algoritmo de Kuhn passo a passo. Qual o tamanho e quais viaturas atendem quais ocorrências?

2 Para um grafo bipartido completo $K_{n,m}$ com $n \leq m$:

- Qual o tamanho do emparelhamento máximo?
- Qual o tamanho da cobertura mínima de vértices (König)?
- Qual o tamanho do conjunto independente máximo?

3 O grafo abaixo é bipartido? Justifique usando 2-coloração ou exibindo um ciclo ímpar.

```
0 -- 1 -- 2 -- 3 -- 0
     |          |
     4 ------- 5
```

9.3. Exercícios de Programação

1 Implemente a verificação de bipartição usando BFS com 2-coloração.

2 Implemente o algoritmo de Kuhn para emparelhamento máximo bipartido.

3 **Desafio (Tabuleiro de Xadrez):** Dado um tabuleiro de xadrez $N \times N$ com K casas bloqueadas, encontre o número máximo de **torres** que podem ser colocadas sem que duas se ataquem. *Dica: torres atacam na mesma linha ou coluna. Modelagem: linhas* $= U$*, colunas* $= V$*, casas livres = arestas.*

4 Resolva problemas de juiz online:

- Verificação de bipartição.
- Emparelhamento máximo.
- Cobertura mínima de vértices.