PROBLEMA DO FLUXO MÁXIMO: OTIMIZAÇÃO EM ARQUITETURAS DE *DATA CENTER*

Discente: José Bruno Barros dos Santos
Programa de Pós-Graduação em Ciência da Computação

Disciplina: Projeto e Análise de Algoritmos

São Cristóvão - SE
2025

INTRODUÇÃO

- O problema consiste em encontrar a quantidade máxima de fluxo que pode ser enviada de uma Fonte (s) para um Sumidouro (t) em uma rede de fluxo.

- A rede é um Grafo Orientado G = (V, E), onde as arestas (u, v) possuem uma capacidade c(u, v).

- Regras Fundamentais (Restrições):

1. Restrição de Capacidade: **O fluxo em cada aresta nunca pode exceder sua capacidade.**
2. Conservação do Fluxo: **Em nós intermediários (diferentes de s ou t), o fluxo que entra deve ser igual ao fluxo que sai.**

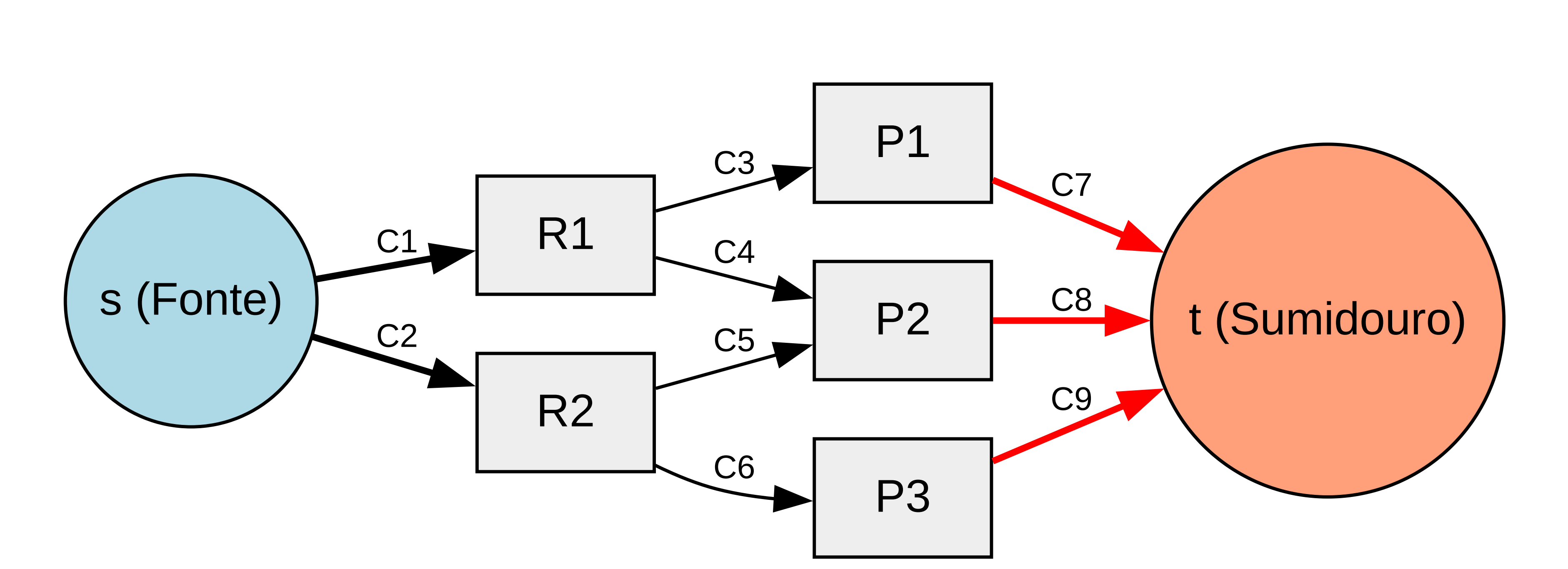
BALANCEAMENTO DE CARGA EM *DATA CENTERS*

Em sistemas de Big Data (Data Centers, Clusters), é crucial otimizar a distribuição de requisições ou dados entre múltiplos servidores (workers).

O Fluxo Máximo permite determinar o melhor caminho de distribuição para **maximizar o throughput** total de processamento.

MODELAGEM DE FLUXO EM ARQUITETURA DE *DATA CENTER*

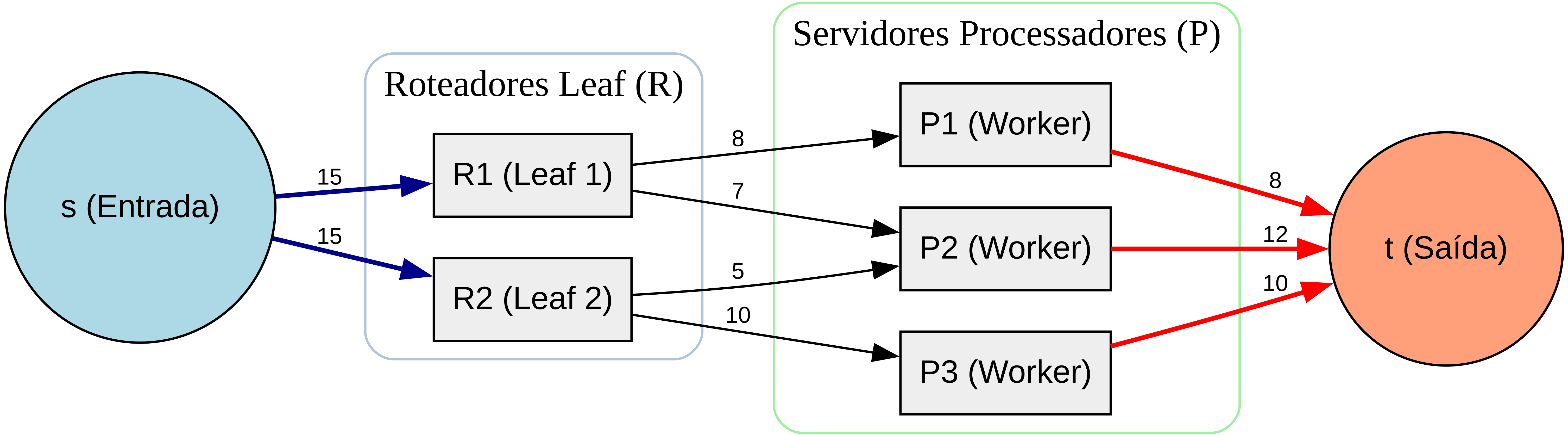
Entrada	s (Fonte)	Origem das requisições de dados.	-
Roteamento	R1, R2	Roteadores Leaf (Camada de Agregação).	Largura de banda do link de entrada.
Processamento	P1, P2, P3	Servidores/Workers (Camada de Processamento).	Largura de banda do link interno (R → P).
Saída	t (Sumidouro)	Coleta de resultados.	Capacidade de CPU/Processamento do servidor.



O ALGORITMO DE RESOLUÇÃO: MÉTODO EDMONDS-KARP

- Melhoramento Iterativo do fluxo, começando em **f=0**.
- Rede auxiliar que mostra a capacidade restante e permite o cancelamento de fluxo.
- Caminho de **s** para **t** encontrado por BFS (Busca em Largura) no grafo residual, que possui capacidade residual positiva.
- Processo: Repetir **(Encontrar Caminho → Aumentar Fluxo → Atualizar Grafo)** até que não haja mais caminhos aumentantes.

INSTÂNCIA E MODELAGEM EM PYTHON (7 NÓS)



Fluxo Máximo em Arquitetura de Data Center (Leaf-Server)

MATRIZ DE CAPACIDADES (7X7)

```
# ÍNDICES: 0=S, 1=R1, 2=R2, 3=P1, 4=P2, 5=P3, 6=T

#      0   1    2    3    4    5   6
# 0  [0, 15, 15, 0,   0,   0, 0],   <- S PARA ROTEADORES
# 1  [0, 0,   0,  8,   7,   0, 0],   <- R1 PARA PROCESSADORES
# 2  [0, 0,   0,  0,   5,  10, 0],   <- R2 PARA PROCESSADORES
# 3  [0, 0,   0,  0,   0,   0, 8],   <- P1 (GARGALO DA CPU)
# 4  [0, 0,   0,  0,   0,   0, 12],  <- P2 (GARGALO DA CPU)
# 5  [0, 0,   0,  0,   0,   0, 10],  <- P3 (GARGALO DA CPU)
# 6  [0, 0,   0,  0,   0,   0, 0]    <- T
```

IMPLEMENTAÇÃO DA SOLUÇÃO

```
#Função de Busca (bfs)

def bfs(graph, s, t, parent):
    # Encontra caminho de s para t na rede residual
    # ... código de fila e visita ...
    while queue:
        # ... lógica de exploração ...
        if v == t:
            return True # Caminho encontrado!
    return False # Sem caminho
```

```python
from collections import deque

def bfs(graph, s, t, parent):
    num_nodes = len(graph)
    visited = [False] * num_nodes
    queue = deque()

    queue.append(s)
    visited[s] = True

    while queue:
        u = queue.popleft()
        for v in range(num_nodes):
            if not visited[v] and graph[u][v] > 0:
                queue.append(v)
                visited[v] = True
                parent[v] = u

                if v == t:
                    return True # Caminho aumentante encontrado

    return False # Não há mais caminhos disponíveis
```

IMPLEMENTAÇÃO DA SOLUÇÃO

```
# Algoritmo Principal (edmonds_karp)

def edmonds_karp(graph, s, t):
    residual_graph = [row[:] for row in graph]
    max_flow = 0

    # Repete enquanto a BFS encontra um caminho aumentante
    while bfs(residual_graph, s, t, parent):

        # 1. Encontra o gargalo (path_flow)
        path_flow = min(path_flow, residual_graph[u][v])

        # 2. Aumenta o fluxo total
        max_flow += path_flow

        # 3. Atualiza o Grafo Residual (arestas de ida e volta)
        residual_graph[u][v] -= path_flow
        residual_graph[v][u] += path_flow
    return max_flow
```

```python
def edmonds_karp(graph, s, t):
    num_nodes = len(graph)
    parent = [-1] * num_nodes
    residual_graph = [row[:] for row in graph]
    max_flow = 0

    while bfs(residual_graph, s, t, parent):

        path_flow = float("Inf")
        v = t
        while v != s:
            u = parent[v]
            path_flow = min(path_flow, residual_graph[u][v])
            v = u

        max_flow += path_flow

        v = t
        while v != s:
            u = parent[v]

            residual_graph[u][v] -= path_flow
            residual_graph[v][u] += path_flow

            v = u

    return max_flow
```

SOLUÇÃO E ANÁLISE

```
Instância do Problema (7 Nós):
Nós: s=0, R1=1, R2=2, P1=3, P2=4, P3=5, t=6
--- Arestas e Capacidades ---
Aresta s -> R1 | Capacidade: 15
Aresta s -> R2 | Capacidade: 15
Aresta R1 -> P1 | Capacidade: 8
Aresta R1 -> P2 | Capacidade: 7
Aresta R2 -> P2 | Capacidade: 5
Aresta R2 -> P3 | Capacidade: 10
Aresta P1 -> t | Capacidade: 8
Aresta P2 -> t | Capacidade: 12
Aresta P3 -> t | Capacidade: 10

--- Solução do Algoritmo Edmonds-Karp ---
O Fluxo Máximo que o Data Center pode processar é: 30
```

CONCLUSÃO

- O Fluxo Máximo é uma ferramenta fundamental para otimização de gargalos em qualquer sistema de rede.
- O Edmonds-Karp resolve o problema de forma **ótima** e é robusto.
- Essencial para engenheiros de sistemas e dados que buscam a máxima performance e eficiência em infraestruturas modernas.