# Projeto e Análise de Algoritmos

## Caminhos Mínimos Utilizando Algoritmos de Dijkstra, Bellman-Ford, Floyd-Warshall, com detecção de Ciclos de Custo Negativo

Conrado C. Bicalho, Danilo S. Souza, Rodolfo L. M. Guimarães, Thiago Schons

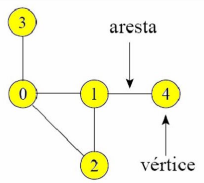
12 de maio de 2016

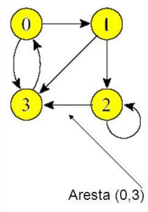
{*conradobh, danilo.gdc, rodolfolabiapari, thiagoschons2*}*@gmail.com*
Departamento de Computação – Universidade Federal de Ouro Preto
35.400-000 – Ouro Preto - MG – Brasil

# Introdução

# Teoria dos Grafos

- Estrutura $G = (V, E)$ onde [2]:
  - $V$ é um conjunto discreto e não vazio de vértices;
  - $E$ é uma família de elementos não vazios definidos em função dos elementos em $V$.
    Cada aresta tem um ou dois nós associados a ela e faz o papel de interligar suas extremidades.
- Grafo orientado $G = (V, E)$, cada aresta orientada está associada a um par ordenado de nós $(u, v)$.




- Tais arcos também podem ser ponderados sendo sua função peso $w(u, v) : E \to \mathbb{R}$ [2].

# Definição do Problema

# Definição do Problema de Caminhos Mínimos de Única Origem [1]

- Grafo $G = (V, E)$ sem laços e valorado por uma função peso $w : E \to \mathbb{R}$, sendo o peso $p = \langle v_1, v_2, \ldots, v_k \rangle$ é o somatório dos pesos de suas arestas

$$w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

- Assim, o peso do caminho mais curto de $u$ até $v$ é

$$\delta(u, v) = \begin{cases} \min\{w(p) : u \overset{p}{\to} v\} & \text{se existe um caminho de } u \text{ até } v, \\ \infty & \text{em caso contrário.} \end{cases}$$

- Propriedade de que:
  - Um caminho mais curto entre dois vértices pode conter outros caminhos mais curtos em seu interior.

## Caminhos mais Curtos de Todos para Todos

- Deseja-se encontrar o caminho mais curto entre todos os pares de vértices $u, v \in V$.
- Este problema também pode ser resolvido utilizando um algoritmo de caminho mínimo um para todos $|V|$ vezes, uma para cada vértice de origem:
  - Mas existem algoritmos específicos para a resolução deste.
- Sua estrutura de dados é lidada como uma matriz de adjacência:
  - É exibido os valores de todos para todos de acordo com sua linha e colina.

## Caminhos mais Curtos de Todos para Todos

- Supondo que existem $|V|$ vértices e enumerados de forma crescente e contínua, uma matriz de entrada seria $W$ $n \times n$ representando os peso das arestas do grafo orientado de $n$ vértices. Isto é, $W = (w_{ij})$ onde

$$w_{ij} = \begin{cases} 0 & \text{se } i = j, \\ \text{o peso da aresta orientada } (i, j) & \text{se } i \neq j \text{ e } (i,j) \in E, \\ \infty & \text{se } i \neq j \text{ e } (i,j) \notin E. \end{cases}$$

# Prova

# Subestrutura Ótima de um Caminho Mais Curto [1]

- **Lema:**
  - Dado um grafo orientado ponderado $G = (V, E)$ com função peso $w : E \rightarrow \mathbb{R}$, seja $p = \langle v_1, v_2, \ldots, v_k \rangle$ um caminho mais curto.
  - Para quaisquer $i$ e $j$ tais que $1 \leq i \leq j \leq k$, seja $p_{ij} = \langle v_i, v_{i+1}, \ldots, v_j \rangle$ o subcaminho.
  - Então, $p_{ij}$ é um caminho mais curto de $v_i$ e $v_j$.

# Subestrutura Ótima de um Caminho Mais Curto [1]

- **Lema:**
  - Dado um grafo orientado ponderado $G = (V, E)$ com função peso $w : E \to \mathbb{R}$, seja $p = \langle v_1, v_2, \ldots, v_k \rangle$ um caminho mais curto.
  - Para quaisquer $i$ e $j$ tais que $1 \leq i \leq j \leq k$, seja $p_{ij} = \langle v_i, v_{i+1}, \ldots, v_j \rangle$ o subcaminho.
  - Então, $p_{ij}$ é um caminho mais curto de $v_i$ e $v_j$.
- **Prova:**
  - Quando decompõe o caminho $p$ em $v_1 \xrightarrow{p_{1i}} v_i \xrightarrow{p_{ij}} v_j \xrightarrow{p_{jk}} v_k$, teremos $w(p) = w(p_{1i}) + w(p_{ij}) + w(p_{jk})$.
  - Supondo que existisse um caminho $p'_{ij}$ de $v_i$ até $v_j$ com peso $w(p'_{ij}) < w(p_{ij})$.
  - Então, $v_1 \xrightarrow{p_{1i}} v_i \xrightarrow{p'_{ij}} v_j \xrightarrow{p_{jk}} v_k$ é um caminho de $v_1$ até $v_k$ cujo peso $w(p) = w(p_{1i}) + w(p'_{ij}) + w(p_{jk})$ é menor que $w(p)$, o que contradiz a hipótese de que $p$ é um caminho mais curto de $v_1$ até $v_k$.

# Itens Importantes

# Arestas de Peso Negativo

- Em algumas instâncias do problema, pode haver arestas cujo pesos são negativos.

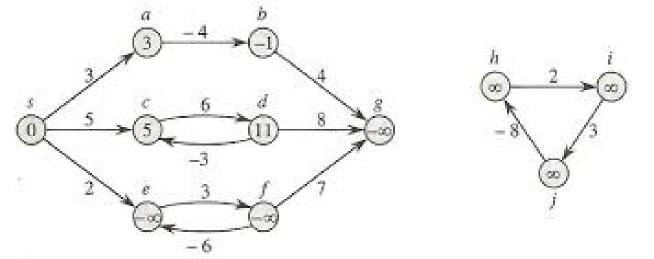
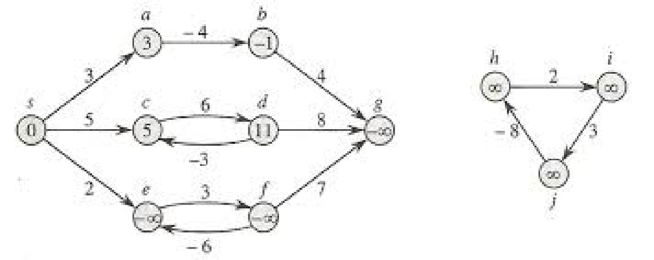


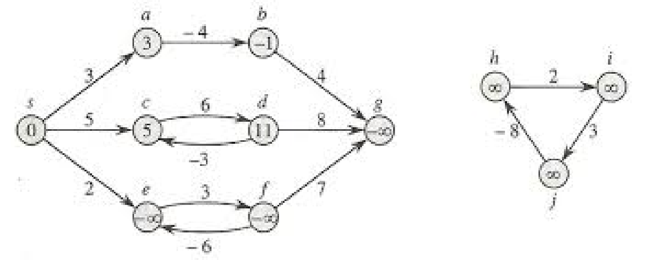

**Figura 1:** Exemplo simples de um grafo com arestas de peso negativo. Fonte: [1].

## Arestas de Peso Negativo




- Um grafo $G = (V, E)$ que não contenha nenhum ciclo de peso negativo acessível a partir da origem $s$:
  - $\forall v \in V$, o peso do caminho mais curto $\delta(s, v)$ permanece bem definido, mesmo tendo um valor negativo.

# Arestas de Peso Negativo




- Um grafo $G = (V, E)$ que não contenha nenhum ciclo de peso negativo acessível a partir da origem $s$:
  - $\forall v \in V$, o peso do caminho mais curto $\delta(s, v)$ permanece bem definido, mesmo tendo um valor negativo.
  - Caso exista um ciclo de peso negativo acessível, os pesos dos caminhos não serão bem definidos.
  - Existindo o ciclo negativo no caminho de $s$ até $v$, então $\delta(s, v) = -\infty$ [1].

# Arestas de Peso Negativo

- O algoritmo de Dijkstra possui restrições de execução.
- Já algoritmos como o Bellman-Ford e Floyd-Warshall são mais flexíveis.

# Algoritmos

# Ambiente de Hardware e Software Utilizado

**Tabela 1:** Tabela com as informações de ambiente de execução do trabalho realizado.

| Item | Descrição |
|---|---|
| Processador | 1 Processador Intel Core i7 - 2,9 GHz |
| Núcleos | 4 Núcleos |
| Cache L2 (por Núcleo) | 256 KB |
| Cache L3 | 4 MB |
| Memória RAM | 10 GB DDR3 |
| Arquitetura | Arquitetura de von Neumann |
| Sistema Operacional | OS X 10.11.4 (15E65) |
| Versão do Kernel | Darwin 15.4.0 |
| Compilador | Apple LLVM version 7.3.0 (clang-703.0.31) |

# Algoritmo Bellman-Ford

**Algorithm 1** Bellman-Ford

```
procedure BELLMAN-FORD(G, w, s)
    INICIALIZA-UNICA-FONTE(G, s);
    for i ← 1, |V[G]| − 1 do
        for cada aresta (u, v) ∈ E[G] do
            RELAXA(u, v, w);
        end for
    end for
    for cada aresta (u, v) ∈ E[G] do
        if d[v] > d[u] + w(u, v) then
            return FALSE;
        end if
    end for
    return TRUE;
end procedure
```

# Algoritmo Bellman-Ford

**Algorithm 2** Bellman-Ford

```
procedure BELLMAN-FORD(G, w, s)
    INICIALIZA-UNICA-FONTE(G, s);
    for i ← 1, |V[G]| − 1 do                    ▷ O(V)
        for cada aresta (u, v) ∈ E[G] do
            RELAXA(u, v, w);
        end for
    end for
    for cada aresta (u, v) ∈ E[G] do
        if d[v] > d[u] + w(u, v) then
            return FALSE;
        end if
    end for
    return TRUE;
end procedure
```

**Algorithm 3** Bellman-Ford

```
procedure BELLMAN-FORD(G, w, s)
    INICIALIZA-UNICA-FONTE(G, s);
    for i ← 1, |V[G]| − 1 do                          ▷ O(V)
        for cada aresta (u, v) ∈ E[G] do               ▷ O(E)
            RELAXA(u, v, w);
        end for
    end for
    for cada aresta (u, v) ∈ E[G] do
        if d[v] > d[u] + w(u, v) then
            return FALSE;
        end if
    end for
    return TRUE;
end procedure
```

# Algoritmo Bellman-Ford

**Algorithm 4** Bellman-Ford

```
procedure BELLMAN-FORD(G, w, s)
    INICIALIZA-UNICA-FONTE(G, s);
    for i ← 1, |V[G]| − 1 do                         ▷ O(V)
        for cada aresta (u, v) ∈ E[G] do             ▷ O(E)
            RELAXA(u, v, w);                         ▷ O(1)
        end for
    end for
    for cada aresta (u, v) ∈ E[G] do
        if d[v] > d[u] + w(u, v) then
            return FALSE;
        end if
    end for
    return TRUE;
end procedure
```

# Algoritmo Bellman-Ford

**Algorithm 5** Bellman-Ford

```
procedure BELLMAN-FORD(G, w, s)
    INICIALIZA-UNICA-FONTE(G, s);
    for i ← 1, |V[G]| − 1 do                         ▷ O(V)
        for cada aresta (u, v) ∈ E[G] do              ▷ O(E)
            RELAXA(u, v, w);                          ▷ O(1)
        end for
    end for
    for cada aresta (u, v) ∈ E[G] do                  ▷ O(E)
        if d[v] > d[u] + w(u, v) then
            return FALSE;
        end if
    end for
    return TRUE;
end procedure
```

# Algoritmo Bellman-Ford

**Algorithm 6** Bellman-Ford

**procedure** BELLMAN-FORD$(G, w, s)$
    INICIALIZA-UNICA-FONTE(G, s);
    **for** $i \leftarrow 1, |V[G]| - 1$ **do** $\triangleright\ \mathcal{O}(V)$
        **for** cada aresta $(u, v) \in E[G]$ **do** $\triangleright\ \mathcal{O}(E)$
            RELAXA(u, v, w); $\triangleright\ \mathcal{O}(1)$
        **end for**
    **end for**
    **for** cada aresta $(u, v) \in E[G]$ **do** $\triangleright\ \mathcal{O}(E)$
        **if** $d[v] > d[u] + w(u, v)$ **then** $\triangleright\ \mathcal{O}(1)$
            **return** FALSE;
        **end if**
    **end for**
    **return** TRUE;
**end procedure**

# Algoritmo de Dijkstra

# Algoritmo de Dijkstra

**Algorithm 7** Dijkstra

```
procedure DIJKSTRA(G, w, s)
    INICIALIZA-UNICA-FONTE(G, s);
    S ← ∅;
    Q ← V[G];
    while Q ≠ ∅ do
        u ← RETIRA-MINIMO(Q);
        S ← S ∪ {u};
        for cada vértice v ∈ Adj[u] do
            RELAXA(u, v, w);
        end for
    end while
end procedure
```

# Algoritmo de Dijkstra

**Algorithm 8** Dijkstra

**procedure** $\textsc{Dijkstra}(G, w, s)$
    INICIALIZA-UNICA-FONTE(G, s); $\triangleright\ \mathcal{O}(E)$
    $S \leftarrow \emptyset$;
    $Q \leftarrow V[G]$;
    **while** $Q \neq \emptyset$ **do**
        $u \leftarrow$ RETIRA-MINIMO(Q);
        $S \leftarrow S \cup \{u\}$;
        **for** cada vértice $v \in Adj[u]$ **do**
            RELAXA(u, v, w);
        **end for**
    **end while**
**end procedure**

# Algoritmo de Dijkstra

**Algorithm 9** Dijkstra

**procedure** $\textsc{Dijkstra}(G, w, s)$
    INICIALIZA-UNICA-FONTE(G, s); $\triangleright\ \mathcal{O}(E)$
    $S \leftarrow \emptyset$;
    $Q \leftarrow V[G]$;
    **while** $Q \neq \emptyset$ **do** $\triangleright\ \mathcal{O}(\log V)$
        $u \leftarrow$ RETIRA-MINIMO(Q);
        $S \leftarrow S \cup \{u\}$;
        **for** cada vértice $v \in Adj[u]$ **do**
            RELAXA(u, v, w);
        **end for**
    **end while**
**end procedure**

# Algoritmo de Dijkstra

**Algorithm 10** Dijkstra

**procedure** $\textsc{Dijkstra}(G, w, s)$
    INICIALIZA-UNICA-FONTE(G, s); $\triangleright\ \mathcal{O}(E)$
    $S \leftarrow \emptyset$;
    $Q \leftarrow V[G]$;
    **while** $Q \neq \emptyset$ **do** $\triangleright\ \mathcal{O}(\log V)$
        $u \leftarrow$ RETIRA-MINIMO(Q); $\triangleright\ \mathcal{O}(V)$
        $S \leftarrow S \cup \{u\}$;
        **for** cada vértice $v \in Adj[u]$ **do**
            RELAXA(u, v, w);
        **end for**
    **end while**
**end procedure**

# Algoritmo de Dijkstra

**Algorithm 11** Dijkstra

```
procedure DIJKSTRA(G, w, s)
    INICIALIZA-UNICA-FONTE(G, s);                ▷ O(E)
    S ← ∅;
    Q ← V[G];
    while Q ≠ ∅ do                               ▷ O(log V)
        u ← RETIRA-MINIMO(Q);                    ▷ O(V)
        S ← S ∪ {u};
        for cada vértice v ∈ Adj[u] do           ▷ O(V)
            RELAXA(u, v, w);
        end for
    end while
end procedure
```

# Algoritmo Floyd Warshall

# Algoritmo Floyd Warshall

**Algorithm 12** Floyd-Warshall

```
procedure FLOYD-WARSHALL(W)
    n ← linhas[W];
    D ← W;
    for k ← 1, n do
        for i ← 1, n do
            for j ← 1, n do
                d_ij^k ← min(d_ij^(k−1), d_ik^(k−1) + d_kj^(k−1));
            end for
        end for
    end for
    return D;
end procedure
```

# Algoritmo Floyd Warshall

**Algorithm 13** Floyd-Warshall

**procedure** FLOYD-WARSHALL($W$)
    $n \leftarrow linhas[W]$;
    $D \leftarrow W$;
    **for** $k \leftarrow 1, n$ **do** ▷ $\mathcal{O}(V)$
        **for** $i \leftarrow 1, n$ **do**
            **for** $j \leftarrow 1, n$ **do**
                $d_{ij}^{k} \leftarrow min(d_{ij}^{k-1}, d_{ik}^{k-1} + d_{kj}^{k-1})$;
            **end for**
        **end for**
    **end for**
    **return** $D$;
**end procedure**

# Algoritmo Floyd Warshall

**Algorithm 14** Floyd-Warshall

**procedure** FLOYD-WARSHALL($W$)
    $n \leftarrow linhas[W]$;
    $D \leftarrow W$;
    **for** $k \leftarrow 1, n$ **do** $\triangleright \mathcal{O}(V)$
        **for** $i \leftarrow 1, n$ **do** $\triangleright \mathcal{O}(V)$
            **for** $j \leftarrow 1, n$ **do**
                $d_{ij}^{k} \leftarrow min(d_{ij}^{k-1}, d_{ik}^{k-1} + d_{kj}^{k-1})$;
            **end for**
        **end for**
    **end for**
    **return** $D$;
**end procedure**

# Algoritmo Floyd Warshall

**Algorithm 15** Floyd-Warshall

**procedure** $\textsc{Floyd-Warshall}(W)$
    $n \leftarrow linhas[W]$;
    $D \leftarrow W$;
    **for** $k \leftarrow 1, n$ **do** ▷ $\mathcal{O}(V)$
        **for** $i \leftarrow 1, n$ **do** ▷ $\mathcal{O}(V)$
            **for** $j \leftarrow 1, n$ **do** ▷ $\mathcal{O}(V)$
                $d_{ij}^{k} \leftarrow min(d_{ij}^{k-1}, d_{ik}^{k-1} + d_{kj}^{k-1})$;
            **end for**
        **end for**
    **end for**
    **return** $D$;
**end procedure**

# Algoritmo Floyd Warshall

**Algorithm 16** Floyd-Warshall

**procedure** $\textsc{Floyd-Warshall}(W)$
    $n \leftarrow linhas[W]$;
    $D \leftarrow W$;
    **for** $k \leftarrow 1, n$ **do** ▷ $\mathcal{O}(V)$
        **for** $i \leftarrow 1, n$ **do** ▷ $\mathcal{O}(V)$
            **for** $j \leftarrow 1, n$ **do** ▷ $\mathcal{O}(V)$
                $d_{ij}^{k} \leftarrow min(d_{ij}^{k-1}, d_{ik}^{k-1} + d_{kj}^{k-1})$; ▷ $\mathcal{O}(1)$
            **end for**
        **end for**
    **end for**
    **return** $D$;
**end procedure**

# Experimentos

# Experimentos

- Utilizou-se de 4 instâncias:
  - *rome99.gr*, *rg300_4730.gr*, *rg300_768_floyd.gr*, *rg300_768_floyd-n.gr*.
- Para cada instância foi executado 20 vezes no mesmo ambiente de testes e colhido o tempo de execução.
- Todos os resultados estão disponíveis no relatório deste.

# Valores de Tempos de Execução

**Tabela 2:** Tabela com os valores de tempo médio de cada algoritmo nas quatro instâncias com o objetivo de obter caminhos mínimos de todos para todos.

| **Instância** | **Bellman-Ford (s)** | **Dijkstra (s)** | **Ford-Warshall (s)** |
|---|---|---|---|
| *rome99.gr* | 420.0757 | 93.59347 | 149.3292 |
| *rg300_4730.gr* | 1.788467 | 0.11464 | 0.118966 |
| *rg300_768_floyd.gr* | 0.294253 | 0.094997 | 0.116002 |
| *rg300_768_floyd-n.gr* | 0.289948 | Não se aplica. | 0.098479 |

- A Figura 2 exibe um gráfico comparando os resultados de cada algoritmo sobre a instância *rome99.gr*.

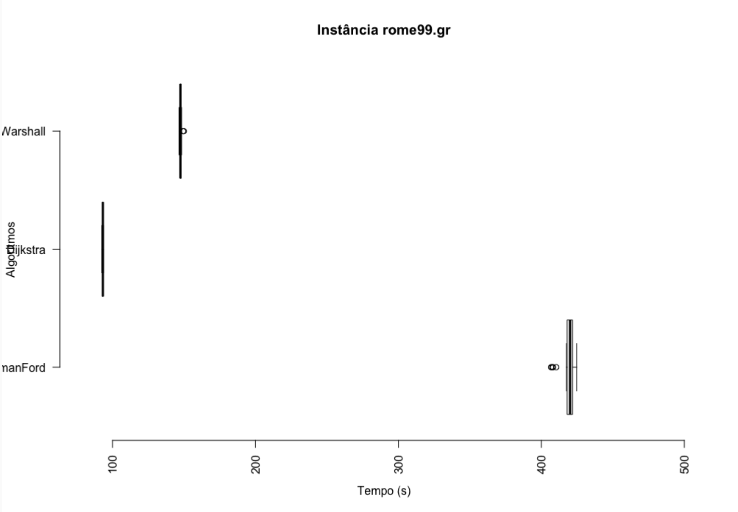



**Figura 2:** Tempo de execução de cada algoritmo sobre a instância *rome99.gr*. Fonte: Autor.

- A Figura 3 exibe um gráfico comparando os resultados de cada algoritmo sobre a instância *rg300_4730.gr*.

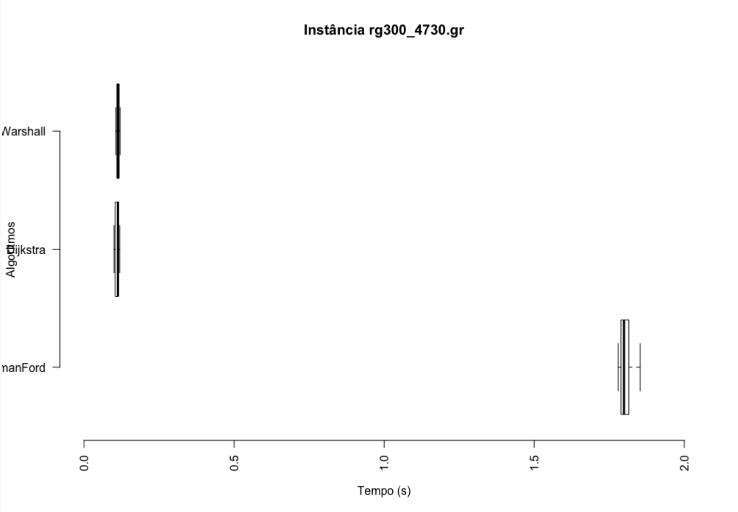



**Figura 3:** Tempo de execução de cada algoritmo sobre a instância *rg300_4730.gr*. Fonte: Autor.

- A Figura 4 exibe um gráfico comparando os resultados de cada algoritmo sobre a instância *rg300_768_floyd.gr*.

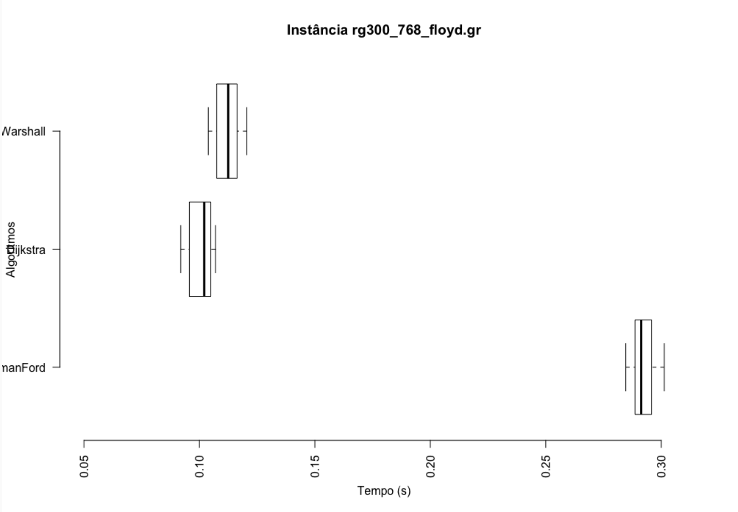



**Figura 4:** Tempo de execução de cada algoritmo sobre a instância *rg300_768_floyd.gr*. Fonte: Autor.

- A Figura 5 exibe um gráfico comparando os resultados de cada algoritmo sobre a instância *rg300_768_floyd-n.gr*.

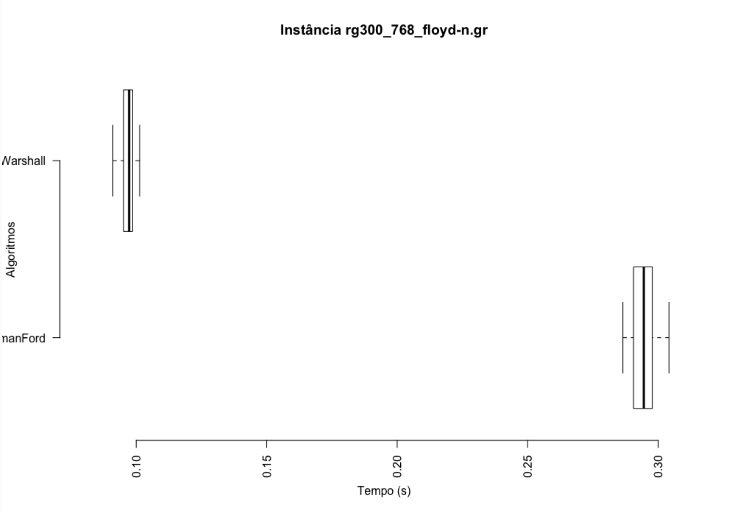



**Figura 5:** Tempo de execução de cada algoritmo sobre a instância com arestas negativas *rg300_768_floyd-n.gr*. Fonte: Autor.

# Conclusão

# Conclusão

- O problema de caminhos mínimos também está relacionado a programação linear.
  - É possível reduzir um caso especial de programação linear ao fato de encontrar caminhos mais curtos a partir de uma única origem.
  - Tal problema pode ser resolvido com algoritmo de Bellman-Ford, sendo assim resolvendo também o problema de programação linear [1].
- Realizou-se 20 iterações de teste por ser uma média razoável para análise e também devido ao tempo computacional elevado pelo tamanho das instâncias utilizadas assim como a complexidade dos algoritmos.

# Conclusão

- Em termos de implementação:
  - Todos os códigos possuem grande facilidade de implementação;
  - Principalmente o Algoritmo Floyd Warshall que se baseia numa simples estratégia mas com análise assintótica elevada.
- Sobre a análise assintótica:
  - Houve uma grande disputa entre o Bellman Ford e Dijkstra.
  - O Algoritmo de Floyd Warshall ficou fora dessa disputa por ser de complexidade de tempo $\mathcal{O}(n^3)$
  - E o Bellman tem complexidade $\mathcal{O}(n^2)$ e o Dijkstra $\mathcal{O}(E + V \log V)$ no pior caso.
  - O algoritmo Dijkstra teve sucesso em todas as execuções devida sua complexidade assintótica de tempo ser menor que todos os outros. A estrutura utilizada nele foi projetada pelos integrantes dos grupos.

# Bibliografia

T. H. Cormen.
**Algoritmos: teoria e prática.**
Elsevier, 2002.

P. O. B. Netto.
**Grafos: teoria, modelos, algoritmos.**
Edgard Blücher, 2003.

# Projeto e Análise de Algoritmos

Caminhos Mínimos Utilizando Algoritmos de Dijkstra, Bellman-Ford, Floyd-Warshall, com detecção de Ciclos de Custo Negativo

Conrado C. Bicalho, Danilo S. Souza, Rodolfo L. M. Guimarães, Thiago Schons

12 de maio de 2016

{*conradobh, danilo.gdc, rodolfolabiapari, thiagoschons2*}*@gmail.com*
Departamento de Computação – Universidade Federal de Ouro Preto
35.400-000 – Ouro Preto - MG – Brasil

# Considerações de Projeto e Análise

- Alguns algoritmos implementados tratam o ‘infinito’ como: o maior peso encontradas das arestas multiplicado por ele mesmo.
- Nenhum algoritmo faz teste de verificação de entradas inválidas.
- Foi executado em todos os algoritmos analisadores de código estáticos e dinâmicos. Executou-se primeiramente o Clang Static Analyzer e em seguida o Valgrind. Com exceção do Algoritmo de Bellman Ford, todos retornaram sucesso nas análises.

# Relaxamento [1]

- Técnica onde para cada vértice $v \in V$, mantém-se um atributo $d[v]$, que é o limite superior sobre o peso do caminho mais curto entre $s$ e $v$.
- Funciona da seguinte maneira:
  - Inicialização. Faz a estima de distância $d(v) = \infty$.
  - Relaxamento. Relaxar uma aresta $(u, v)$ consiste em testar alguma forma de melhorar o caminho mais curto para $v$ encontrado até agora por outros caminhos intermediários que utilizem $u$.


**Figura 6:** Exemplo de um relaxamento de uma aresta. Fonte: `http://wiki.icmc.usp.br/images/b/b4/7._1GrafosCaminhosLA(Graca).pdf`

# Variações deste Problema

- Variações descritas até agora:
    - Caminho mais curto de uma única origem; e

# Variações deste Problema

- Variações descritas até agora:
    - Caminho mais curto de uma única origem; e
    - De todos para todos.

# Variações deste Problema

- Variações descritas até agora:
  - Caminho mais curto de uma única origem; e
  - De todos para todos.
- Mas além destes, é possível obter outras variantes deste problema sem perder sua essência. Seriam as outras variantes:
  - Caminho mais curto de destino único; e
  - Par único.

## Aplicações

- Problemas de única origem:
  - Problemas relacionados com sequências de decisões;
  - Escolhas de itinerários ao longo de uma viagem;
  - Traçado de uma estratégia em um problema de investimentos;
  - Trata-se de decisões envolvendo alguma forma de custo a ser minimizado.

- Problema de todos para todos:
  - Elaboração de uma tabela de distância entre todos os pares de cidades de um certa região para um atlas rodoviário.