Simulador de Viagem entre Capitais Brasileiras

Francisco Theodoro Arantes Florencio
Universidade Federal do Rio de Janeiro
DRE: 123099825

January 24, 2025

Introdução

Imagine que você é uma pessoa planejando viajar de ônibus entre capitais brasileiras. Para otimizar sua viagem, você deseja encontrar o trajeto mais rápido entre duas cidades, considerando apenas rotas possíveis entre estados que compartilham fronteiras. Este projeto simula exatamente essa situação! A ideia surgiu da necessidade de planejar viagens mais eficientes e seguras, considerando os dados reais de distâncias rodoviárias brasileiras disponíveis no site Brazil Road Distances de Rodrigo Saldanha. Utilizando esses dados (diferenciados apenas por uma pequena curadoria manual responsável por retirar os dados que tivessem 'NA', ou seja, NULL em alguma de suas linhas para que assim pudéssemos ler como se fosse um float e não uma string), foi criado um algoritmo capaz de determinar o menor caminho entre capitais, utilizando conceitos de álgebra linear e estruturas algébricas em Julia.
O projeto nos ajuda a calcular, a partir da base de dados, a menor quantidade de capitais de forma que a sua distância fosse mínima para a qual você teria que percorrer, indo de uma capital brasileira a outra, passando apenas por outras capitais utilizando o meio rodoviário (ônibus).

1 O Contexto da Viagem

João, nosso protagonista, quer viajar de Rio Branco (AC) até Maceió (AL) de ônibus. Porém, ele percebe que nem todas as capitais estão diretamente conectadas. João precisa planejar cuidadosamente para garantir que sua rota seja viável e que ela chegue ao destino no menor tempo possível. Para isso, utilizamos um modelo matemático que leva em conta:

- As distâncias reais entre as capitais.
- As conexões entre estados vizinhos.
- A rota com a menor distância acumulada.

O algoritmo descrito neste relatório permite que João insira sua cidade de origem e destino e, em poucos segundos, obtenha a melhor rota possível.

2 Conceitos Fundamentais

2.1 Monóides e Semianéis

Um monóide é definido como uma 3-tupla $(S, \oplus, 0)$, onde:

- S é um conjunto não-vazio;
- $\oplus$ é uma operação binária chamada "soma";
- 0 é o elemento neutro em relação a $\oplus$, tal que $x \oplus 0 = 0 \oplus x = x$.

O monóide satisfaz as propriedades de totalidade, associatividade e existência de elemento neutro. Um semianel é uma generalização do monóide, definido como uma 5-tupla $(S, \oplus, \cdot, 0, 1)$, com:

- $(S, \oplus, 0)$ sendo um monóide comutativo;
- $(S, \cdot, 1)$ formando outro monóide;
- A operação $\cdot$ distribuindo sobre $\oplus$, isto é:

$$x \cdot (y \oplus z) = (x \cdot y) \oplus (x \cdot z),$$

$$(x \oplus y) \cdot z = (x \cdot z) \oplus (y \cdot z);$$

- 0 sendo elemento absorvente: $0 \cdot x = x \cdot 0 = 0$.

2.2 Fecho e Ponto Fixo

Um *semianel fechado* é uma extensão de um semianel $(S, \oplus, \cdot, 0, 1)$ com uma operação adicional de *fecho*, denotada por $*$. Formalmente, o fecho de $x \in S$, denotado por x^*, é definido como:

$$x^* = 1 \oplus x \oplus (x \cdot x) \oplus (x \cdot x \cdot x) \oplus \cdots = \bigoplus_{i=0}^{\infty} x^i,$$

onde x^i representa a aplicação iterada de $\cdot$, com:

$$x^0 = 1, \quad x^1 = x, \quad x^2 = x \cdot x, \quad \ldots$$

A operação de fecho é útil em várias aplicações, como resolver sistemas lineares em álgebra tropical e calcular caminhos mínimos em grafos.

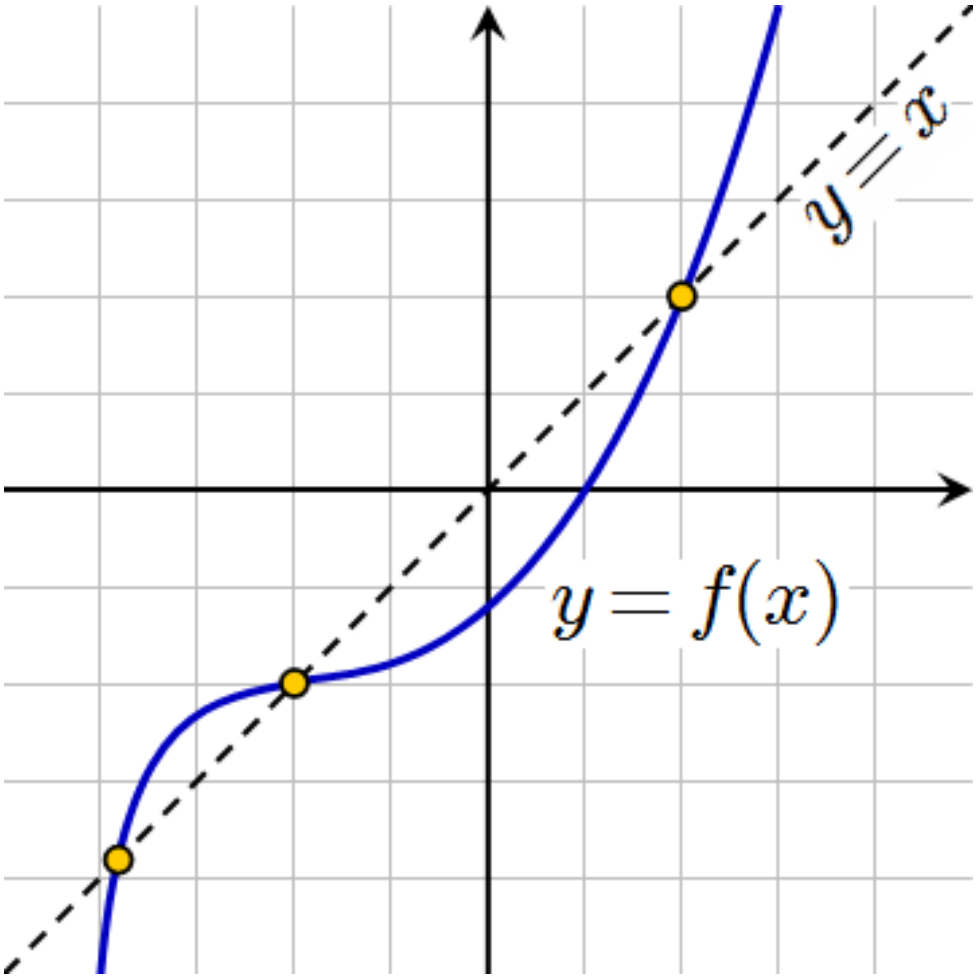

Figure 1: Ilustração do Método do Ponto Fixo

O conceito de *ponto fixo* também desempenha um papel crucial neste contexto. Dado um conjunto A e uma função $f : A \to A$, o ponto fixo de f é um elemento $a \in A$ tal que:

$$f(a) = a.$$

Em muitos casos, os pontos fixos podem ser encontrados iterativamente:

$$a_{n+1} = f(a_n), \quad n = 0, 1, 2, \ldots$$

e, sob certas condições, a sequência $\{a_n\}$ converge para o ponto fixo a:

$$\lim_{n \to \infty} a_n = a.$$

2.2.1 Aplicação em Grafos

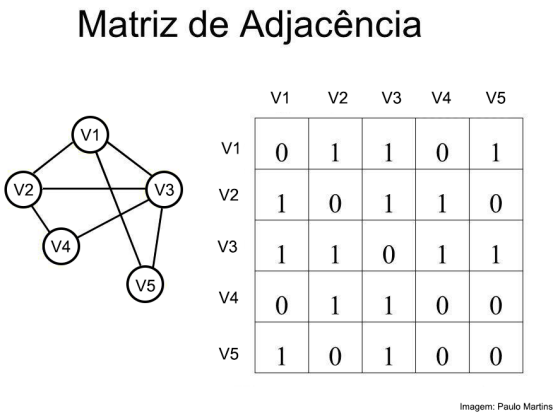

Figure 2: Matriz de adjacência genérica (Não é a que estamos utilizando no trabalho, conforme é explicado a seguir).

No caso de grafos ponderados, o fecho é utilizado para encontrar o menor caminho entre os vértices. Especificamente, no semianel *min-plus*, o fecho de uma matriz de adjacência G é definido como:

$$G^* = \bigoplus_{n=1}^{\infty} G^n,$$

onde G^n representa a matriz que contém os menores caminhos de comprimento exato n entre os vértices.

A convergência para o ponto fixo ocorre quando o valor de G^{n+1} não difere de G^n, ou seja:

$$G^n = G^{n+1}, \quad \text{para algum } n \geq 1.$$

Este método iterativo é eficiente e amplamente utilizado para resolver problemas de caminhos mínimos e de acessibilidade em grafos, sendo uma aplicação direta dos conceitos de semianéis fechados e pontos fixos.

3 Explicação Detalhada do Código

3.1 Configuração Inicial

O código inicia com as importações necessárias:

```
using DataFrames
using CSV
using LinearAlgebra
```

- `DataFrames` e `CSV`: Manipulação de dados tabulares e leitura de arquivos CSV.
- `LinearAlgebra`: Suporte a operações matriciais.

3.2 Estrutura `Path`

A estrutura `Path` é utilizada para armazenar trajetos entre cidades:

```
struct Path
    p::Union{Nothing, Vector{Int}}
    d::Float64
end
```

- `p`: Vetor com índices das cidades no caminho; `nothing` caso o caminho não exista.
- `d`: Distância total do caminho como `Float64`.

3.3 Métodos Personalizados para `Path`

O operador `+` é redefinido para comparar dois caminhos:

```
function Base.:+(a::Path, b::Path)
    isnothing(a.p) && return b
    isnothing(b.p) && return a
    return a.d < b.d ? a : b
end
```

- Retorna o caminho existente se o outro for `nothing`.
- Caso ambos existam, retorna o de menor distância.

O operador `*` é redefinido para concatenar dois caminhos:

```
function Base.:*(a::Path, b::Path)
    isnothing(a.p) && return Path(nothing, Inf)
    isnothing(b.p) && return Path(nothing, Inf)
    @assert last(a.p) == first(b.p)
    return Path([a.p; b.p[2:end]], a.d + b.d)
end
```

- Retorna um `Path` inexistente se qualquer caminho for `nothing`.
- Certifica-se de que o último ponto do primeiro caminho coincide com o início do segundo.
- Concatena os vetores e soma as distâncias.

Define o elemento neutro (*zero*):

```
Base.zero(::Type{Path}) = Path(nothing, Inf)
```

3.4 Matriz Identidade

```
function Identity(dim, tipo)
    return [Path(i == j ? [i] : nothing, i == j ? 0.0 : Inf) for i
       in 1:dim, j in 1:dim]
end
```

- Cria uma matriz identidade para `Path`.
- Na diagonal, atribui caminhos com distância zero.
- Fora da diagonal, define caminhos inexistentes com distância infinita.

3.5 Multiplicação de Matrizes

```
function Base.:*(A::Matrix{Path}, B::Matrix{Path})
    m, n = size(A)
    p = size(B, 2)
    result = Matrix{Path}(undef, m, p)

    for i in 1:m
        for j in 1:p
            paths = [A[i,k] * B[k,j] for k in 1:n]
            result[i,j] = reduce(+, filter(path -> path.p !==
               nothing, paths), init=Path(nothing, Inf))
        end
    end

    return result
end
```

- Realiza a multiplicação de duas matrizes de caminhos.
- Em cada posição, calcula todos os trajetos possíveis combinando os elementos correspondentes e escolhe o menor caminho válido.

3.6 Construção do Grafo de Capitais

Carregamento e filtragem dos dados:

```
df = CSV.read("dist_brasil_clean.csv", DataFrame)
df_fronteiras = filter(row -> (row.orig, row.dest) in fronteiras ||
    (row.dest, row.orig) in fronteiras, df)
```

- `df`: Carrega o arquivo CSV com distâncias entre cidades.
- `df_fronteiras`: Filtra apenas as fronteiras diretas entre capitais.

Construção da matriz de adjacência:

```
matriz_adj = Identity(n, Path)

for row in eachrow(df_fronteiras)
    i = capitais_dict[row.orig]
    j = capitais_dict[row.dest]
    matriz_adj[i, j] = Path([i, j], row.dist)
    matriz_adj[j, i] = Path([j, i], row.dist)
end
```

- Inicializa a matriz de adjacência com valores padrão.
- Atribui os valores de distância reais para as conexões diretas.

3.7 Algoritmo de Menor Caminho Utilizando o Método do Ponto Fixo

A implementação do algoritmo de menor caminho foi realizada utilizando o Método do Ponto Fixo, um conceito fundamental discutido na disciplina de COCADA. Este método permite encontrar o menor caminho através de uma abordagem iterativa que busca a convergência da solução.

```
function menor_caminho(matriz_adj, origem, destino)
    n = size(matriz_adj, 1)
    pm = copy(matriz_adj)
    fixed_points = 0

    while fixed_points < 2
        pm_old = copy(pm)
        pm = pm * pm
        if pm == pm_old
            fixed_points += 1
        else
            fixed_points = 0
        end
    end

    return pm[origem, destino]
end
```

A implementação do Método do Ponto Fixo neste contexto segue rigorosamente os princípios fundamentais discutidos em nossa disciplina, aplicando diretamente o teorema do ponto fixo para problemas de caminhos mínimos em grafos:

- **Iterações Sucessivas:** Utiliza multiplicações matriciais repetidas no semianel min-plus, correspondendo à função iterativa $f^n(x)$.
- **Identificação do Ponto Fixo:** Detecta a convergência através da estabilidade da matriz de caminhos, onde $f(x) = x$.

- **Critério de Parada:** Implementa o conceito de ponto fixo com um contador de iterações estáveis.

Fundamentação Teórica do Método:

O algoritmo implementa o teorema do ponto fixo de Knaster-Tarski, onde:

- f é uma função monotônica sobre um reticulado completo.
- O ponto fixo x satisfaz $f(x) = x$.
- A convergência é garantida pela propriedade de fechamento do semianel min-plus.

Processo Iterativo:

1. Parte-se da matriz de adjacência inicial A_0.
2. Aplica-se iterativamente $A_{k+1} = A_k \cdot A_k$ no semianel min-plus.
3. Compara-se A_k com A_{k-1} para verificar a convergência.
4. Interrompe-se quando $A_k = A_{k-1}$, identificando o ponto fixo.

Matematicamente, estamos computando:

$$\lim_{n\to\infty} f^n(x) = x,$$

onde $f(x)$ representa a multiplicação matricial no semianel min-plus, e x é a matriz de caminhos mínimos.

Observação Importante: A variável `fixed_points` garante que a convergência não seja um artefato de uma única iteração, exigindo estabilidade consecutiva, o que robustece a aplicação do método do ponto fixo.

3.8 Consulta Interativa

```
println("Digite o  ndice  da cidade de partida:")
origem_idx = parse(Int, readline())
println("Digite o  ndice  da cidade de destino:")
destino_idx = parse(Int, readline())

caminho = menor_caminho(matriz_adj, origem_idx, destino_idx)
println("O menor caminho entre $origem_idx e $destino_idx
   $caminho")
```

- Solicita os índices das cidades ao usuário.
- Calcula o menor caminho usando a função `menor_caminho`.
- Exibe o trajeto na tela.

4 Mapeamento de Capitais e Fronteiras

Figure 3: Estados do Brasil representados conforme o código do IBGE do município de suas respectivas capitais, conforme estamos trabalhando no código. (Adaptação de minha autoria do mapa do Brasil com os estados diferenciados pelos códigos municipais de suas capitais)

4.1 Representação Geográfica com Códigos Municipais do IBGE

4.1.1 Metodologia de Mapeamento

Cada município brasileiro possui um código numérico único, atribuído pelo IBGE, que foi utilizado como identificador principal no mapeamento.

Exemplo de Códigos Municipais das Capitais Brasileiras:

- Rio Branco: 1200401
- Maceió: 2704302
- Belo Horizonte: 3106200

4.1.2 Construção das Fronteiras no Algoritmo

O grafo foi estruturado com base nas capitais brasileiras (representadas por seus códigos municipais) e nas fronteiras que definem suas conexões diretas. As fronteiras foram inseridas manualmente no algoritmo utilizando pares de códigos municipais. Ou seja, foi atribuído a cada estado o código do IBGE referente a sua capital e os estados que faziam fronteira direta (adjacente) para com eles eram colocados num "par ordenado" conforme demonstrado a seguir.

Trecho de código ilustrativo:

```
fronteiras = [
    (1100205, 5103403),  # Porto Velho (RO) e Cuiab  (MT)
    (1200401, 1302603),  # Rio Branco (AC) e Manaus (AM)
    (3106200, 3304557),  # Belo Horizonte (MG) e Rio de Janeiro (RJ
       )
    # Outras fronteiras definidas manualmente...
]
```

Conclusão

Esse trabalho me trouxe um resultado satisfatório e interessante, algo bem legal de ver acontecendo e se observar. Agradeço, em especial, ao professor João Paixão e, o seu atual monitor de disciplina, o aluno Matheus do Ó pois sem eles a ideia inicial desse trabalho teria tomado um rumo totalmente diferente do atual e seria notoriamente não tão edificante e satisfatório. Obrigado por ter lido até aqui!

References

[1] KAMINS, B. Distances in Julia. Disponível em: `https://bkamins.github.io/julialang/2022/10/21/distances.html`. Acesso em: 3 dez. 2024.

[2] SALDANHA, R. Brazil Road Distances. Disponível em: `https://rfsaldanha.github.io/data-projects/brazil_road_distances.html`. Acesso em: 3 dez. 2024.

[3] Mth0. Algoritmos Genéricos para Multiplicação Matricial. Disponível em: `https://github.com/Mth0/Algoritmos-Genericos-Multiplicacao-Matricial`. Acesso em: 3 dez. 2024.

[4] DOLAN, S. Fun with semirings: A functional pearl on the abuse of linear algebra. In: Proceedings of the 18th ACM SIGPLAN International Conference on Functional Programming, ICFP '13, p. 101–110, New York, NY, USA, 2013. Association for Computing Machinery.

[5] EBERT, F. C. CFG Parsing and Boolean Matrix Multiplication. 2007.

[6] GRUNE, D.; JACOBS, C. J. H. Parsing Techniques - A Practical Guide. Monographs in Computer Science. Springer, 2008.

[7] KAMIŃSKI, B. What is the shortest path to AlphaTensor?

[8] LEHMANN, D. J. Algebraic Structures for Transitive Closure. Theoretical Computer Science, v. 4, n. 1, p. 59–76, 1977.

[9] Imagem 1: Ponto fixo. Disponível em: `https://pt.wikipedia.org/wiki/Ponto_fixo`. Acesso em: 3 dez. 2024.

[10] Imagem 2: Grafos — representação e implementação. Disponível em: `https://medium.com/@paulomartins_10299/grafos-representa%C3%A7%C3%A3o-e-implementa%C3%A7%C3%A3o-f260dd98823d`. Acesso em: 3 dez. 2024.