# 21. Matrizes 2

DIM0320

2015.1

# Outline


1. Multiplicação de matrizes
2. Itinerários
3. Exercícios

1 Multiplicação de matrizes

2 Itinerários

3 Exercícios

# Descrição

## Assunto

- Seja $A$ uma matriz $n \times m$ e $B$ uma matriz $m \times n$.
- Determine algoritmicamente $C = A \times B$

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \vdots & \vdots & \vdots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nm} \end{pmatrix}$$

$$\times$$

$$B = \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ b_{m1} & b_{m2} & \dots & b_{mn} \end{pmatrix}$$

$$=$$

$$C = \begin{pmatrix} c_{11} & c_{12} & \dots & c_{1n} \\ c_{21} & c_{22} & \dots & c_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ c_{n1} & c_{n2} & \dots & c_{nn} \end{pmatrix}$$

**com**

- $c_{ij} = \sum_{k=1}^{m} a_{ik} * b_{kj}$

# Cálculo para uma célula

## Inicialização

```
para i de 1 ate n faca
  para j de 1 ate m faca
    A[i, j] <- randi(10)
    B[j, i] <- randi(10)
  fimpara
fimpara
```

## Soma para uma célula

```
soma <- 0
para k de 1 ate m faca
  soma <- soma + A[i, k] * B[k, j]
fimpara
C[i, j] <- soma
```

# Algoritmo

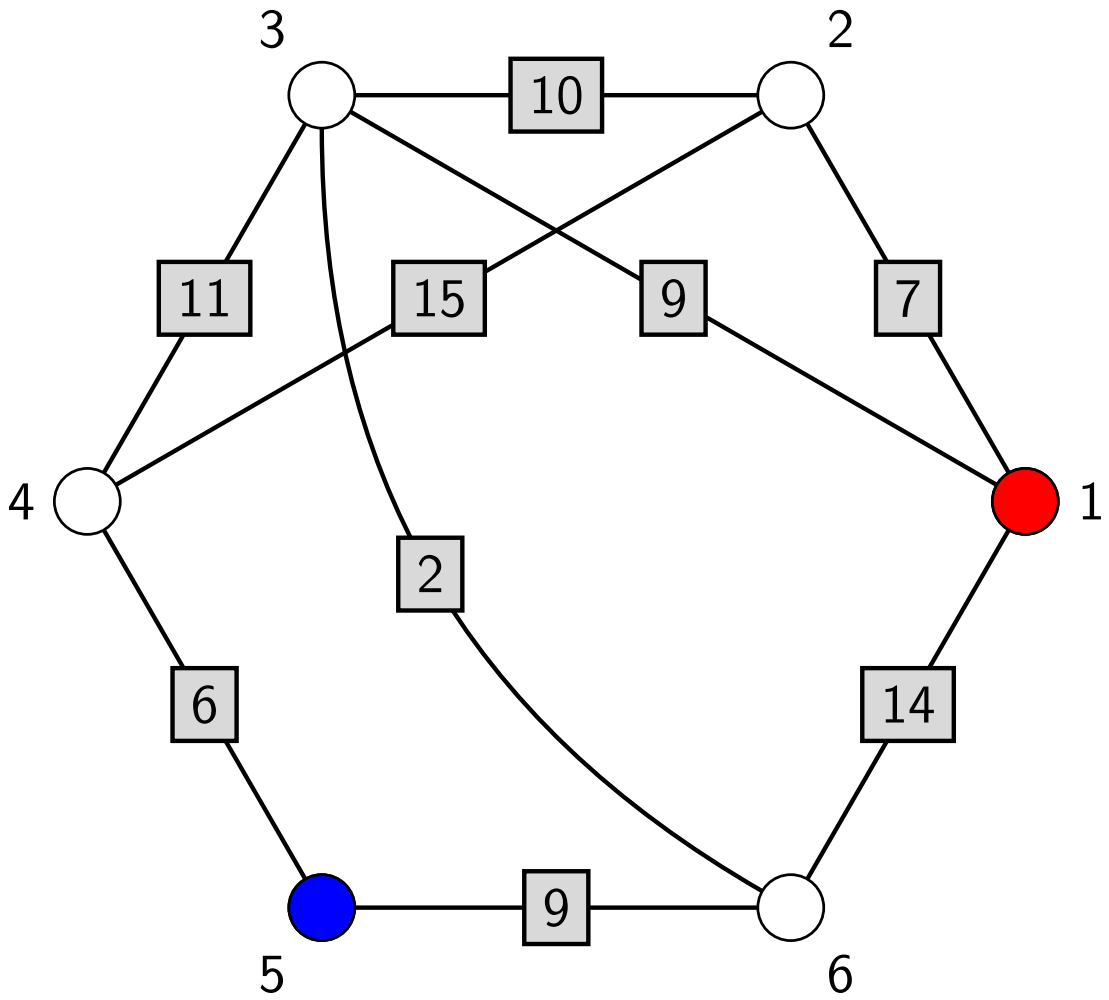
```
algoritmo "Multiplicacao de matrizes"
var n, m : inteiro
    A, B, C: vetor [1..100, 1..100] de real
    i, j, k :inteiro
    soma : real
inicio
   repita
     leia(n, m)
   ate (n > 0) e (n <= 100) e
       (m > 0) e (m <= 100)

   para i de 1 ate n faca
     para j de 1 ate m faca
       A[i, j] <- randi(10)
       B[j, i] <- randi(10)
     fimpara
   fimpara
    para i de 1 ate n faca
     para j de 1 ate n  faca
       soma <- 0
       para k de 1 ate m faca
         soma <- soma + A[i, k] * B[k, j]
       fimpara
       C[i, j] <- soma
     fimpara
   fimpara

    para i de 1 ate n faca
     para j de 1 ate n  faca
      escreva(C[i, j], " ")
     fimpara
     escreval()
    fimpara
fimalgoritmo
```

1 Multiplicação de matrizes

2 Itinerários

3 Exercícios

# Problema




1. O que é o custo duma viagem de 1 para 5, dado um itinerário ?
2. Tem um itinerário de menor custo ?

# Representação de itinerário por uma matriz

$$M = \begin{pmatrix} 0 & 7 & 9 & +\infty & +\infty & 14 \\ 7 & 0 & 10 & 15 & +\infty & +\infty \\ 9 & 10 & 0 & 11 & +\infty & 2 \\ +\infty & 15 & 11 & 0 & 6 & +\infty \\ +\infty & +\infty & +\infty & 6 & 0 & 9 \\ 14 & +\infty & 2 & +\infty & 9 & 0 \end{pmatrix}$$

## Custo de itinérario

O custo de 1 2 3 6 6 3 4 5 , com $k = 8$ cidades é

$$m_{12} + m_{23} + m_{36} + m_{66} + m_{63} + m_{34} + m_{45} = 7 + 10 + 2 + 0 + 2 + 11 + 6 = 38$$

# Cálculo du custo de itinerário

## Entrada dos dados

```
para i de 1 ate n faca
  A[i, i] <- 0
  para j de i + 1 ate n faca
    repita
      escreval("Digite a distância entre ",
               i, " e ", j)
      leia(A[i, j])
    ate (A[i, j] >= 0)
    A[j, i] <- A[i, j]
  fimpara
fimpara
```

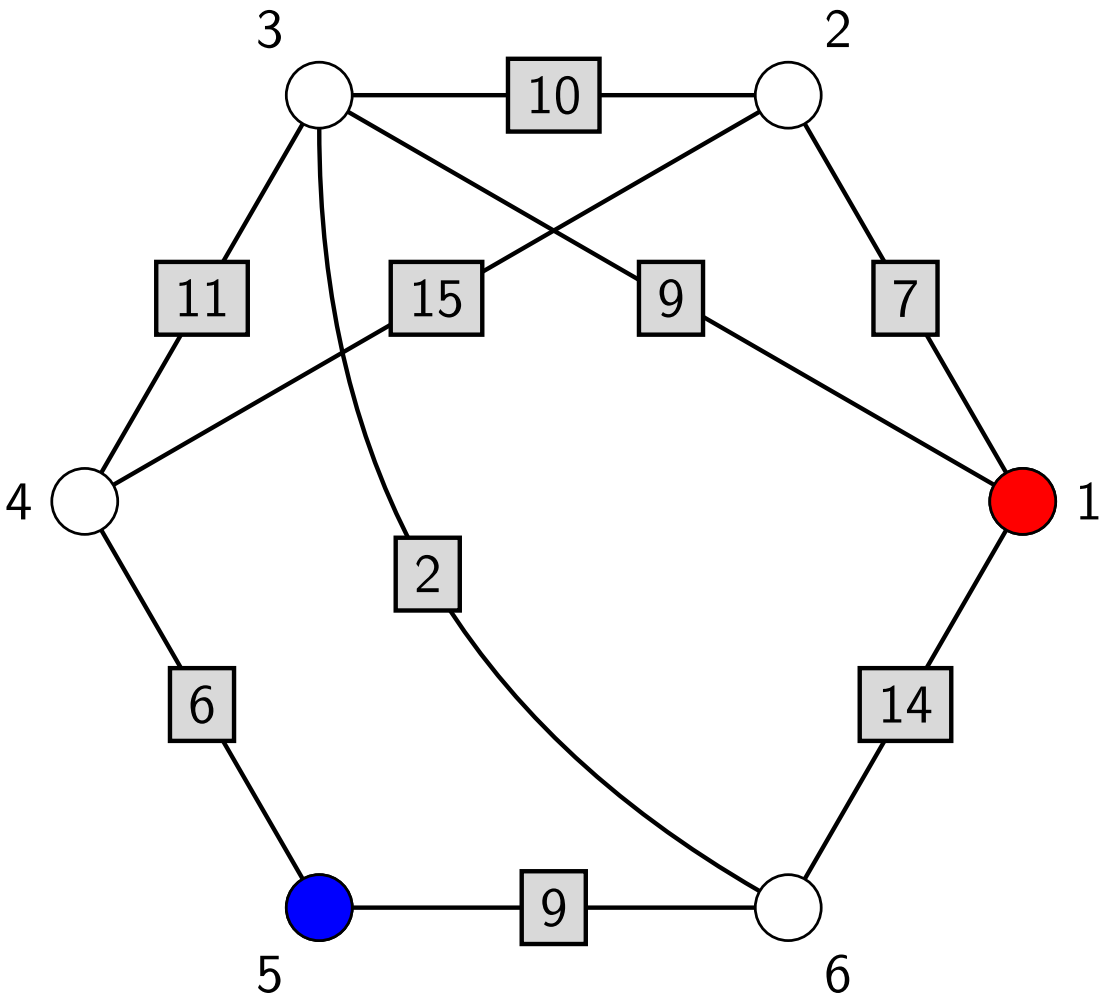
## Cálculo do custo

```
    repita
      leia(cidade1)
    ate (cidade1 > 0) e (cidade1 <= n)

    custo <- 0
    i <- 1
    enquanto i < n_paradas faca
      repita
        leia(cidade2)
      ate (cidade2 > 0) e (cidade2 <= n)
      custo <- custo + A[cidade1, cidade2]
      cidade1 <- cidade2
      i <- i + 1
    fimenquanto

   escreva("Custo total da viagem: ", custo)
fimalgoritmo
```

# Algoritmo

```
algoritmo "Custo itinerario"
var A : vetor [1..100, 1..100] de real
    n, i, j, n_paradas : inteiro
    cidade1, cidade2 : inteiro
    custo : real
inicio
    repita
      escreva("Digite um numero de cidades
               > 0 e <= 100: ")
      leia(n)
    ate (n > 0) e (n <= 100)

    para i de 1 ate n faca
      A[i, i] <- 0
      para j de i + 1 ate n faca
        repita
          escreval("Digite a distância entre "
                   i, " e ", j)
          leia(A[i, j])
        ate (A[i, j] >= 0)
        A[j, i] <- A[i, j]
      fimpara
    fimpara
    repita
      escreva("Quantos paradas tem no
               seu itinerario ? ")
      leia(n_paradas)
    ate n_paradas >= 1

    repita
      leia(cidade1)
    ate (cidade1 > 0) e (cidade1 <= n)

    custo <- 0
    i <- 1
    enquanto i < n_paradas faca
      repita
        leia(cidade2)
      ate (cidade2 > 0) e (cidade2 <= n)
      custo <- custo + A[cidade1, cidade2]
      cidade1 <- cidade2
      i <- i + 1
    fimenquanto

   escreva("Custo total da viagem: ", custo)
fimalgoritmo
```

# Algoritmo de Dijkstra (itinerário de menor custo)



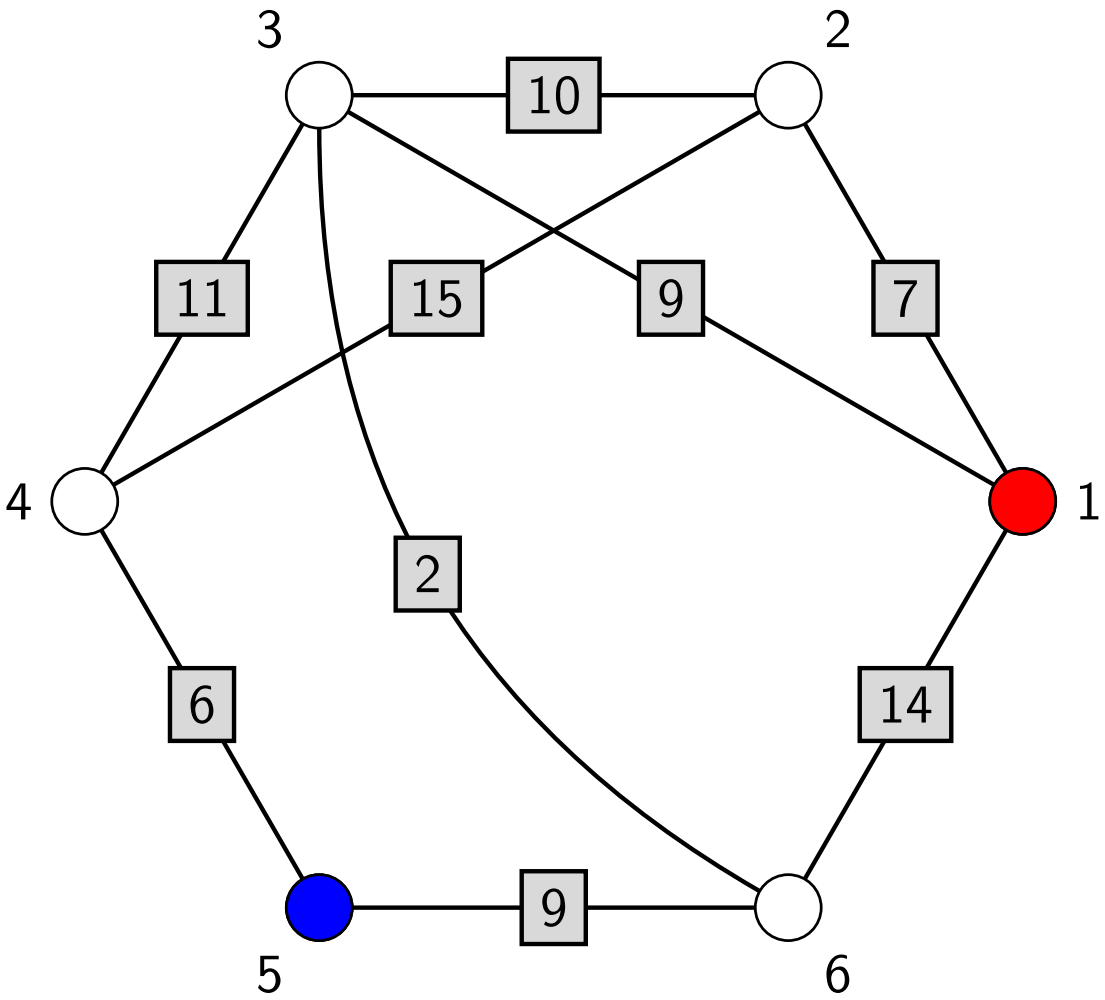

## Distâncias

| | |
|---|---|
| 1 | 0 |
| 2 | $+\infty$ |
| 3 | $+\infty$ |
| 4 | $+\infty$ |
| 5 | $+\infty$ |
| 6 | $+\infty$ |

## Visitados

$\{1\}$

# Algoritmo de Dijkstra (itinerário de menor custo)



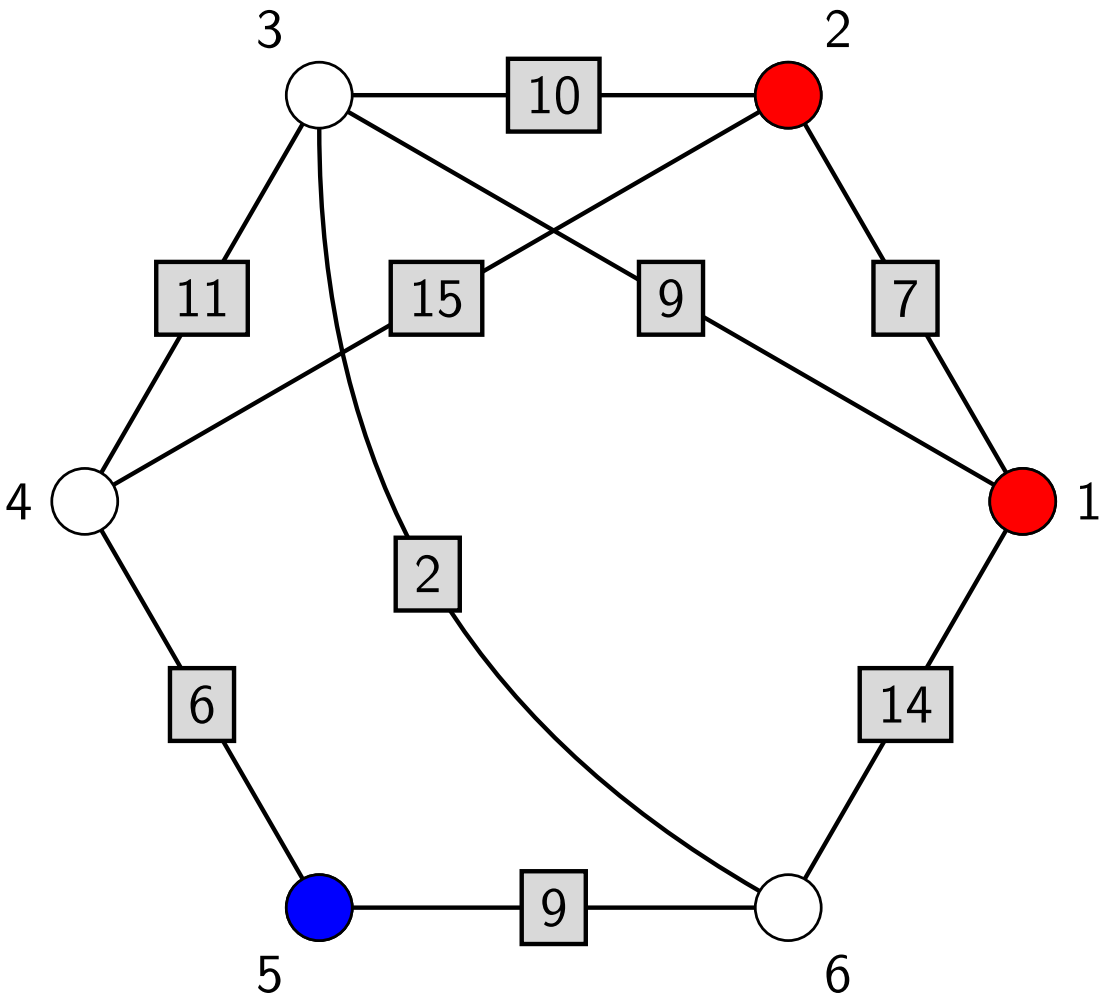

## Distâncias

| | |
|---|---|
| 1 | 0 |
| 2 | 7 |
| 3 | 9 |
| 4 | $+\infty$ |
| 5 | $+\infty$ |
| 6 | 14 |

## Visitados

$\{1\}$

# Algoritmo de Dijkstra (itinerário de menor custo)



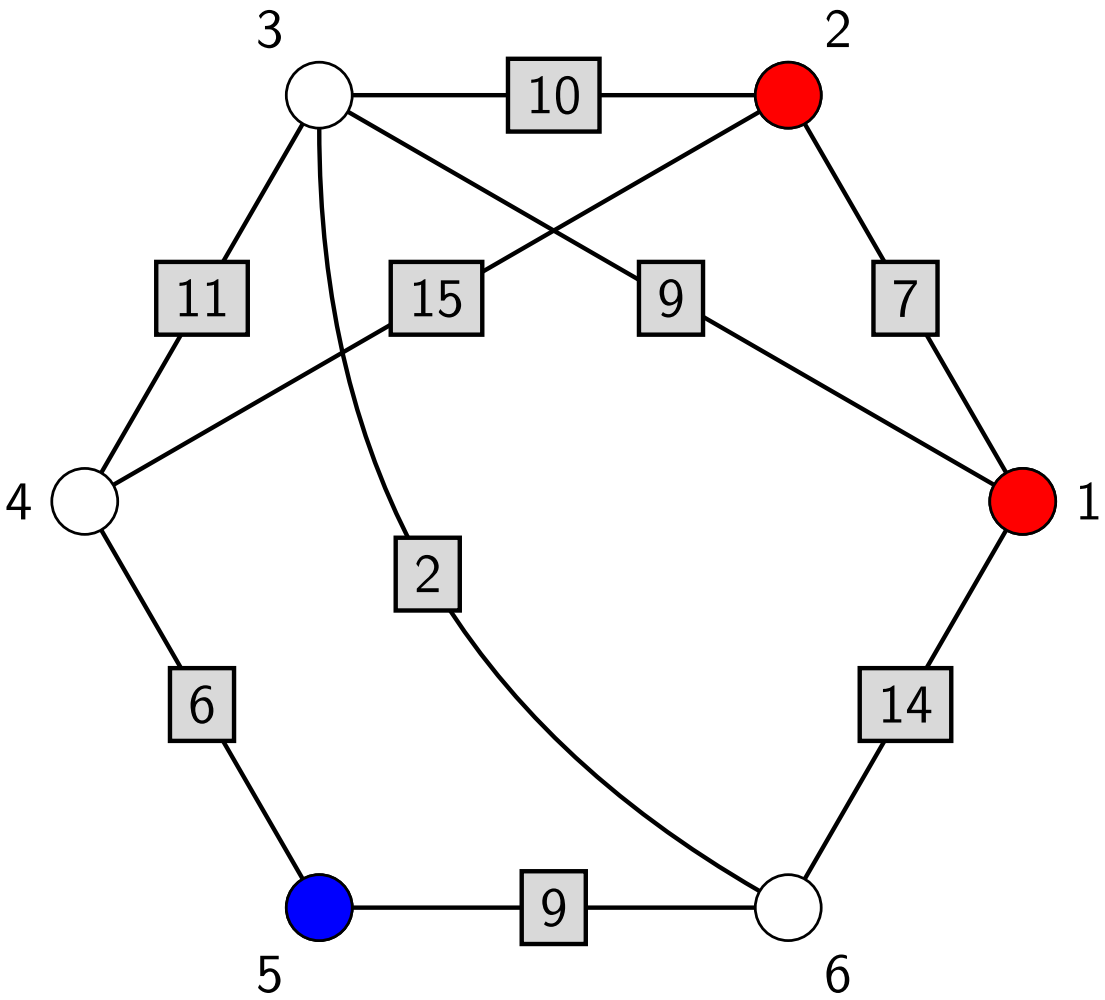

## Distâncias

| | |
|---|---|
| 1 | 0 |
| 2 | 7 |
| 3 | 9 |
| 4 | $+\infty$ |
| 5 | $+\infty$ |
| 6 | 14 |

## Visitados

$\{1, 2\}$

# Algoritmo de Dijkstra (itinerário de menor custo)



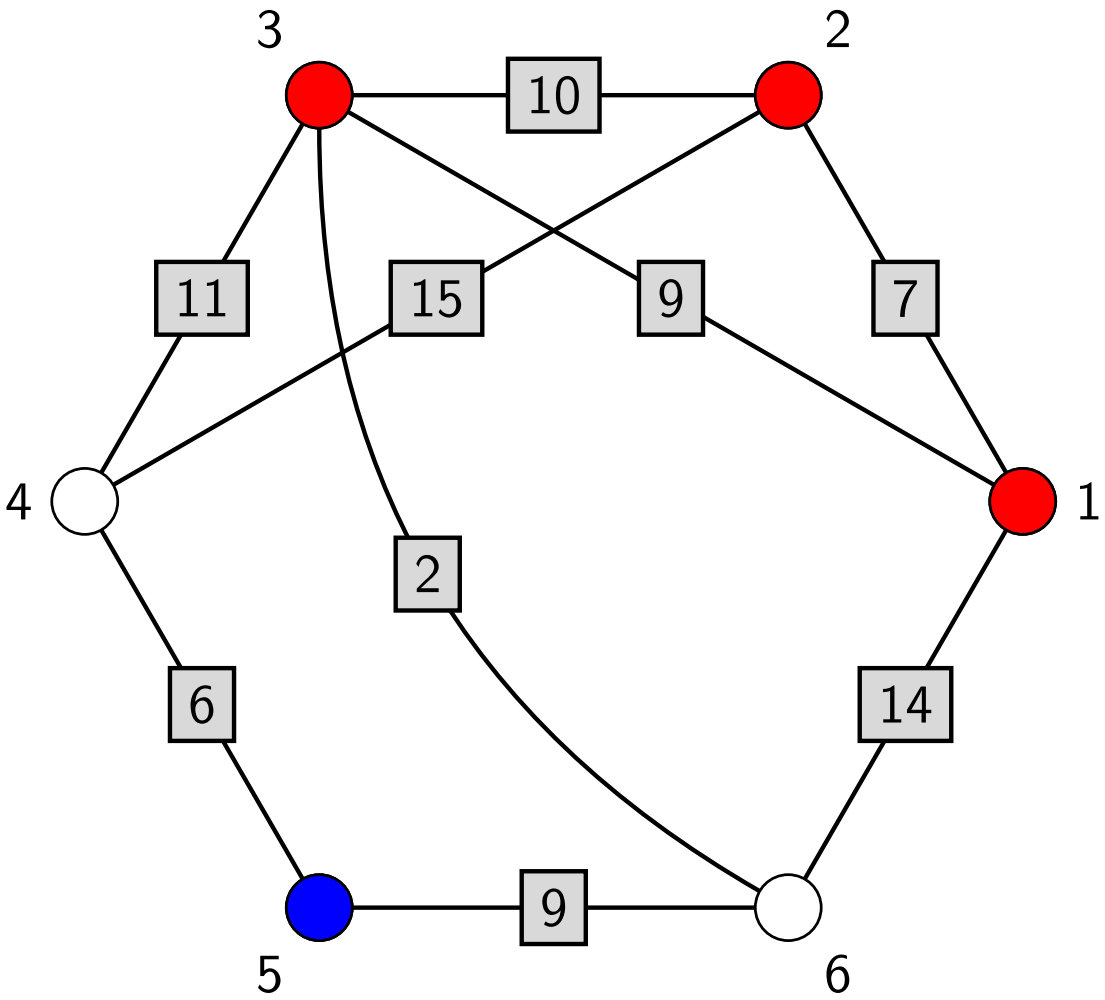

## Distâncias

| | |
|---|---|
| 1 | 0 |
| 2 | 7 |
| 3 | 9 |
| 4 | 22 |
| 5 | $+\infty$ |
| 6 | 14 |

## Visitados

$\{1, 2\}$

# Algoritmo de Dijkstra (itinerário de menor custo)



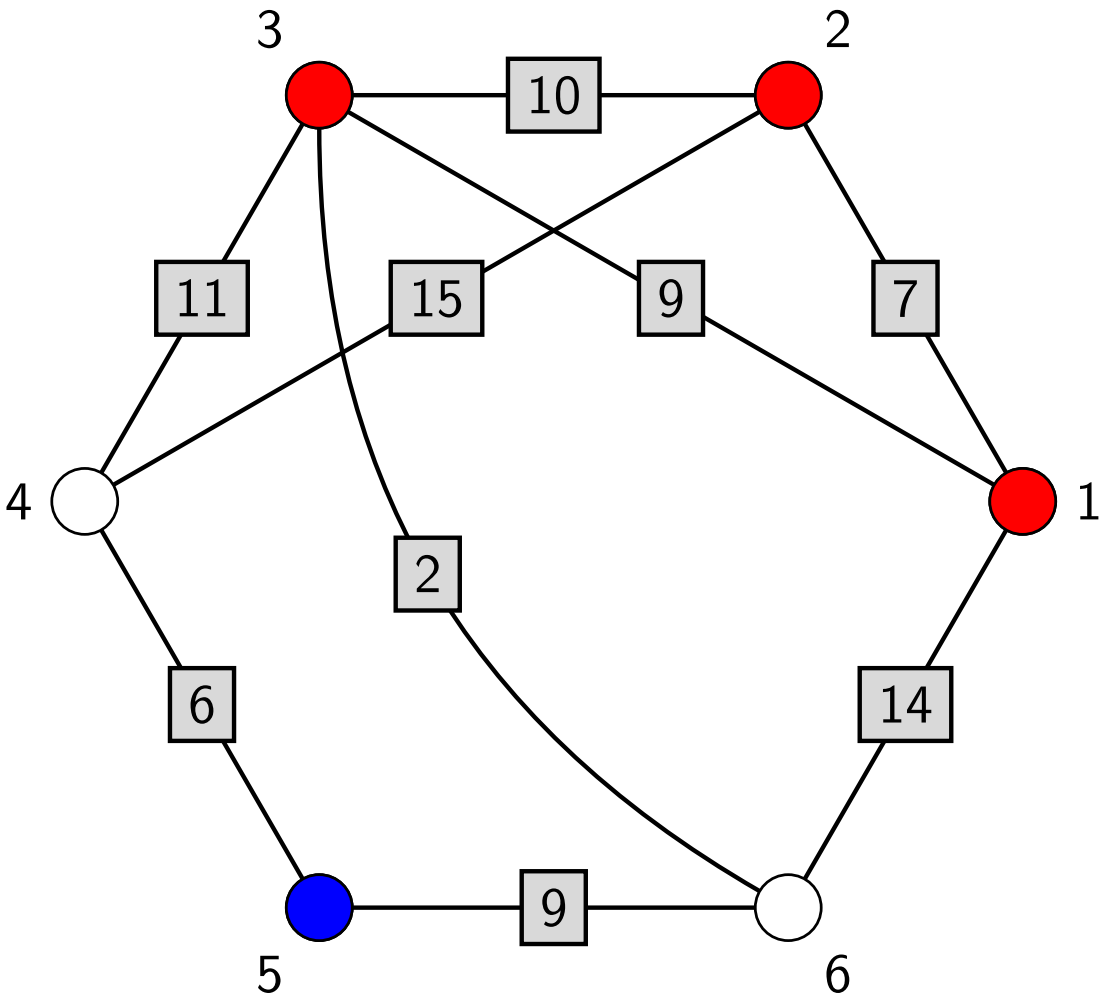

## Distâncias

| | |
|---|---|
| 1 | 0 |
| 2 | 7 |
| 3 | 9 |
| 4 | 22 |
| 5 | $+\infty$ |
| 6 | 14 |

## Visitados

$\{1, 2, 3\}$

# Algoritmo de Dijkstra (itinerário de menor custo)



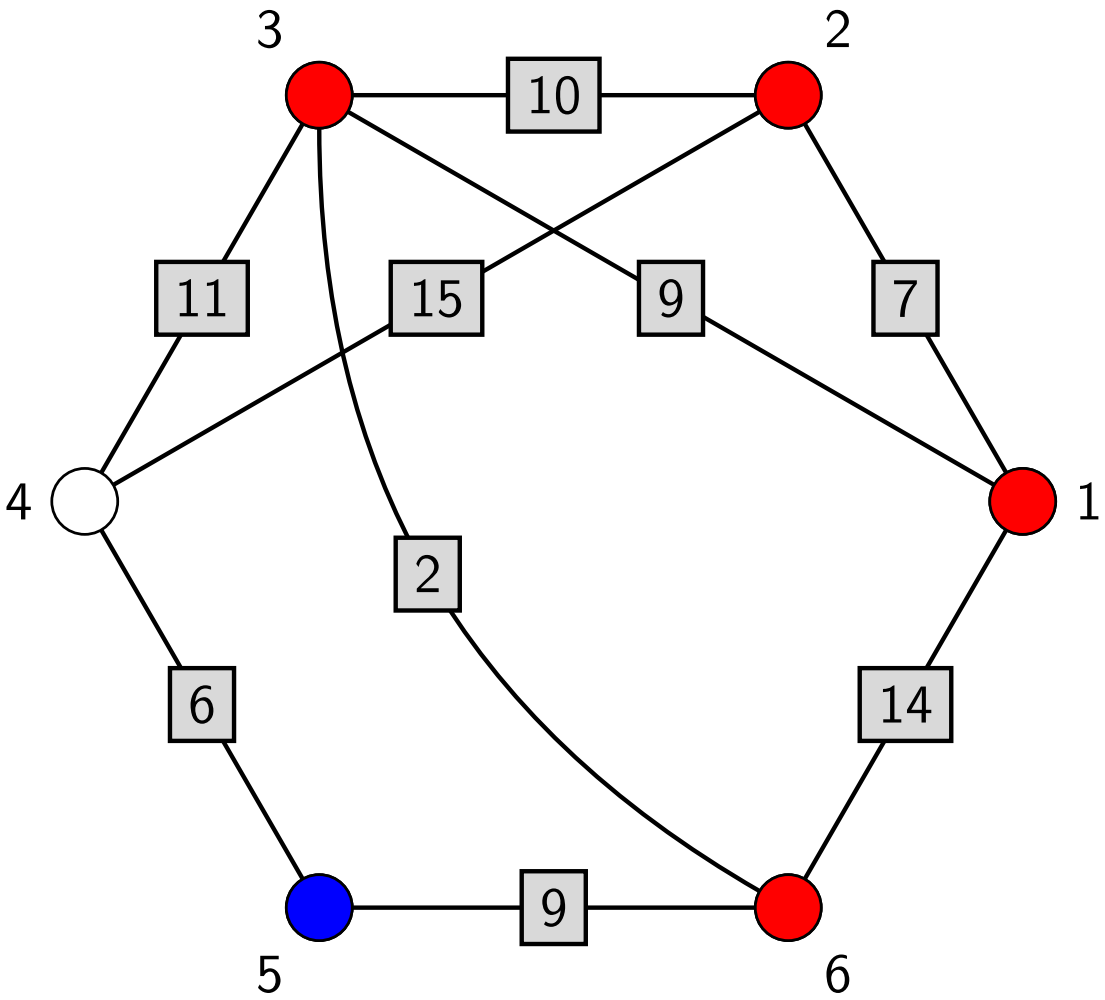

## Distâncias

| | |
|---|---|
| 1 | 0 |
| 2 | 7 |
| 3 | 9 |
| 4 | 20 |
| 5 | $+\infty$ |
| 6 | 11 |

## Visitados

$\{1, 2, 3\}$

# Algoritmo de Dijkstra (itinerário de menor custo)



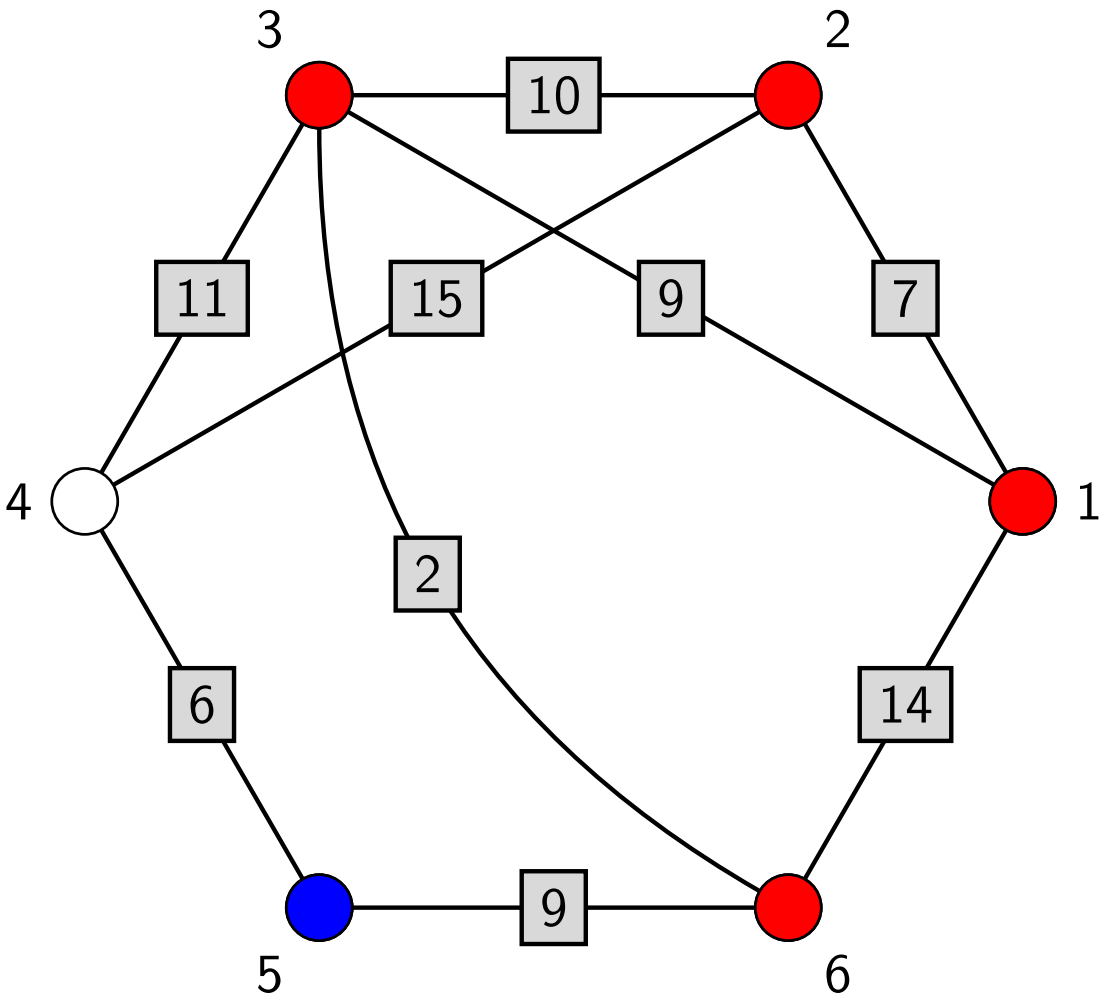

## Distâncias

| | |
|---|---|
| 1 | 0 |
| 2 | 7 |
| 3 | 9 |
| 4 | 20 |
| 5 | $+\infty$ |
| 6 | 11 |

## Visitados

$\{1, 2, 3, 6\}$

# Algoritmo de Dijkstra (itinerário de menor custo)



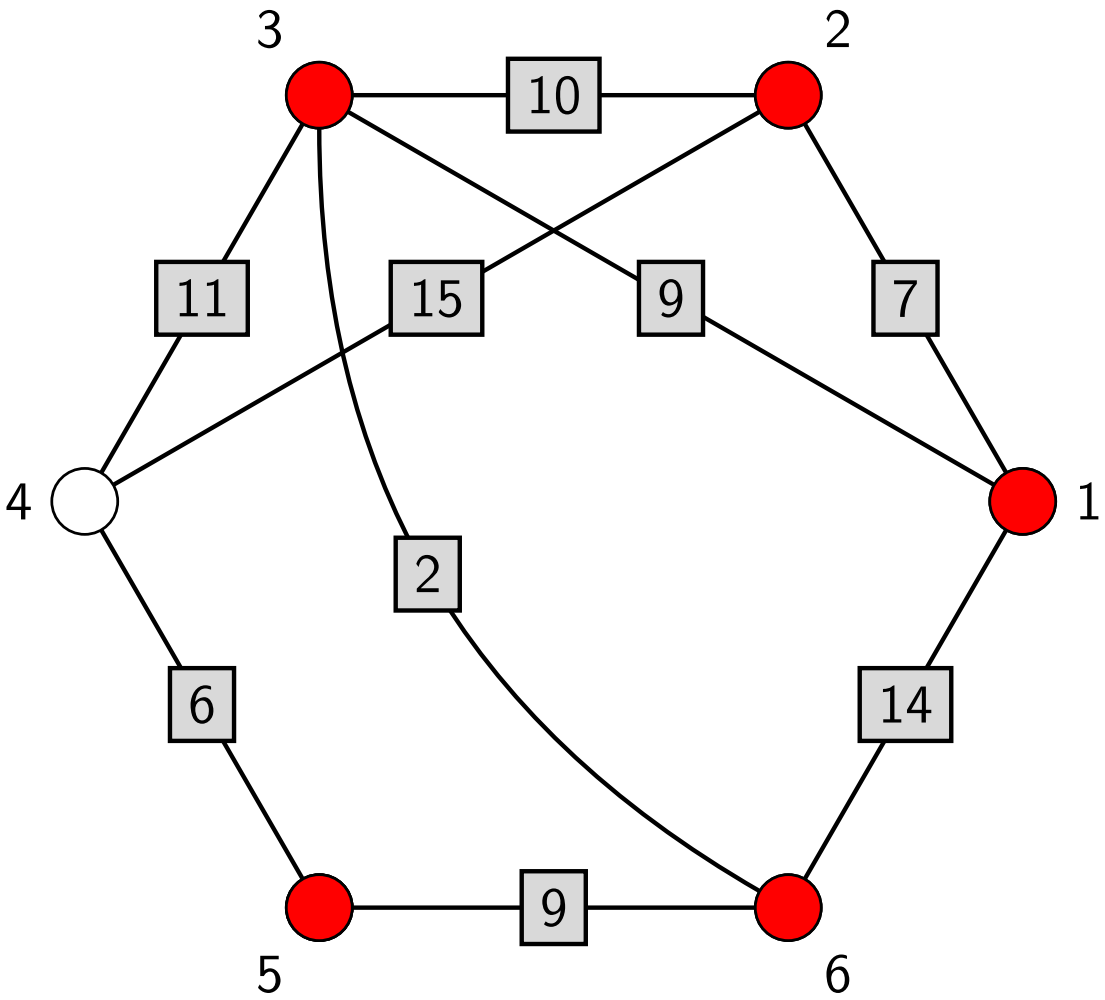

## Distâncias

| | |
|---|---|
| 1 | 0 |
| 2 | 7 |
| 3 | 9 |
| 4 | 20 |
| 5 | 20 |
| 6 | 11 |

## Visitados

$\{1, 2, 3, 6\}$

# Algoritmo de Dijkstra (itinerário de menor custo)




## Distâncias

| | |
|---|---|
| 1 | 0 |
| 2 | 7 |
| 3 | 9 |
| 4 | 20 |
| 5 | 20 |
| 6 | 11 |

## Visitados

$\{1, 2, 3, 6, 5\}$

# Extratos

## Inicialização dos dados

```
para i de 1 ate n faca
  para j de 1 ate n faca
    se i = j entao mat[i, j] <- 0
    senao
       mat[i, j] <- randi(100000)
    fimse
  fimpara
  pai[i] <- -1
  visitados[i] <- falso
  se i = partida entao
    dist[i] <- 0
  senao
    dist[i] <- -1
  fimse
fimpara

para i de 1 ate n faca
  se mat[partida, i] > 0 entao
    dist[i] <- mat[partida, i]
  fimse
fimpara

ultimo <- partida
visitados[ultimo] <- verdadeiro
```

## Cálculo incremental das distâncias

```
enquanto (ultimo <> chegada) faca
   next <- -1
   para i de 1 ate n faca
      se (nao visitados[i]) e
         ((next = -1) ou (
          (dist[i] <> -1) e
           dist[i] < dist[next])))
      entao
         next <- i
         pai[next] <- ultimo
      fimse
   fimpara
   para i de 1 ate n faca
     se (mat[next, i] > 0)
     entao
       dist_local <- mat[next, i] + dist[next]
       se (dist[i] = -1) ou
          (dist_local < dist[i]) entao
          dist[i] <- dist_local
          pai[i] <- next
       fimse
     fimse
   fimpara
   ultimo <- next
   visitados[next] <- verdadeiro
fimenquanto
```

# Algoritmo de Dijkstra

```
algoritmo "Multiplicacao de matrizes"
var n : inteiro
    mat: vetor [1..100, 1..100] de real
    dist: vetor[1..100] de real
    visitados: vetor [1..100] de logico
    pai: vetor [1..100] de inteiro
    next, i, j, dist_local : inteiro
    partida, chegada, ultimo :inteiro
inicio
   repita
     leia(n, partida, chegada)
   ate (n > 0) e (n <= 100) e
       (partida > 0) e (partida <= n) e
       (chegada > 0) e (chegada <= n)
   para i de 1 ate n faca
     para j de 1 ate n faca
       se i = j entao mat[i, j] <- 0
       senao
          mat[i, j] <- randi(100000)
       fimse
     fimpara
     pai[i] <- -1
     visitados[i] <- falso
     se i = partida entao
       dist[i] <- 0
     senao
       dist[i] <- -1
     fimse
   fimpara
```

```
ultimo <- partida
visitados[ultimo] <- verdadeiro
```

# Algoritmo de Dijkstra (fim)

```
enquanto (ultimo <> chegada) faca
   next <- -1
   para i de 1 ate n faca
      se (nao visitados[i]) e
         ((next = -1) ou (
          (dist[i] <> -1) e
           dist[i] < dist[next])))
      entao
         next <- i
         pai[next] <- ultimo
      fimse
   fimpara
   para i de 1 ate n faca
     se (mat[next, i] > 0)
     entao
       dist_local <- mat[next, i] + dist[next]
       se (dist[i] = -1) ou
          (dist_local < dist[i]) entao
          dist[i] <- dist_local
          pai[i] <- next
       fimse
     fimse
   fimpara
   ultimo <- next
   visitados[next] <- verdadeiro
fimenquanto
```

```
escreval(dist[ultimo], "? " )
repita
   escreva(ultimo, "@")
   ultimo <- pai[ultimo]
ate ultimo = -1
fimalgoritmo
```

# Perguntas ?

http://dimap.ufrn.br/~richard/dim0320

1 Multiplicação de matrizes

2 Itinerários

3 Exercícios

# Geração de matrizes

## Assunto

Escrever um algoritmo lê um inteiro $n, n > 1$ e calcula

1. A matriz identidade de dimensão $n$
2. A matriz tri-diagonal de dimensão $n$

$$\begin{pmatrix} 1 & 1 & & & (0) \\ 1 & \ddots & \ddots & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & 1 \\ (0) & & & 1 & 1 \end{pmatrix}$$

3. A matriz de dimensão $n$ formada com o modelo seguinte (em dimensão 4):

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 1 & 2 & 3 \\ 3 & 2 & 1 & 2 \\ 4 & 3 & 2 & 1 \end{pmatrix}$$

# Matriz anti-simétrica

## Assunto

Escrever um algoritmo que determina se uma matriz quadrada de dimensão $n, 0 < n \leq 100$ entrada pelo usuário é anti-simétrica.
Lembramos que uma matriz $A$ é anti-simétrica sse $A^T = -A$

# Matriz de permutação

## Assunto

Dizemos que uma matriz de inteiros, $A$, $n$ por $n$, é uma **matriz de permutação** se em cada linha e em cada coluna houver $n-1$ elementos nulos e um único elemento 1. Por exemplo,

$$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \begin{pmatrix} 2 & -1 & 0 \\ -1 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

A primeira matriz é de permutação, mas a segunda não é.

- Escreva um algoritmo para determinar se uma dada matriz quadrada de inteiros é de permutação. A entrada do algoritmo consiste de um inteiro positivo, $n$, e dos números inteiros que compõem a matriz. A saída do algoritmo é:
  - a mensagem "é de permutação" se a matriz é de permutação;
  - a mensagem "não é de permutação" caso contrário.
- Assuma que $n \leq 1000$.