Relatorio-3398

Guilherme Fernandes

2022-12-03

Alunos

- Guilherme Fernandes Castro de Oliveira - 3398

Instalação

```
remotes::install_github("GuilhermeFCO/grafoR")
library(grafoR)
```

TP1

Funções para criação de objetos do tipo grafo

A função `grafo(nVertices, arestas)` retorna um objeto do tipo grafo, como parâmetros ele recebe o número de vértices e um vetor com as arestas e seus respectivos pesos.

```
x <- grafo(
    6,
    c(
        "1 2 1",
        "1 3 3",
        "2 3 1",
        "2 4 3",
        "2 5 2",
        "3 4 2",
        "4 5 3",
        "4 6 2",
        "5 6 3"
    )
)
printGrafo(x, pesos = TRUE)
```

```
##
##
## [1] -> 2 (1) -> 3 (3) -> ≡
##
## [2] -> 1 (1) -> 3 (1) -> 4 (3) -> 5 (2) -> ≡
##
## [3] -> 1 (3) -> 2 (1) -> 4 (2) -> ≡
##
## [4] -> 2 (3) -> 3 (2) -> 5 (3) -> 6 (2) -> ≡
##
## [5] -> 2 (2) -> 4 (3) -> 6 (3) -> ≡
##
## [6] -> 4 (2) -> 5 (3) -> ≡
```

A função `grafoDeArquivo(path)` recebe o caminho do arquivo de entrada e retorna o grafo correspondente.

```
y <- grafoDeArquivo("./exemplo")
printGrafo(y)
```

```
##
##
## [1] -> 2 -> 5 -> ≡
##
## [2] -> 1 -> 5 -> ≡
##
## [3] -> 5 -> 4 -> ≡
##
## [4] -> 3 -> 5 -> ≡
##
## [5] -> 2 -> 3 -> 4 -> 1 -> ≡
```

Funções feitas no TP1

A função `ordem(x)` retorna a ordem do grafo `x.

```
grafoR::ordem(x)
```

```
## [1] 6
```

A função `tamanho(x)` retorna o tamanho do grafo `x`.

```
grafoR::tamanho(x)
```

```
## [1] 9
```

A função `printGrafo(x, pesos=FALSE)` printa o grafo, mostrando ou não os pesos.

```
grafoR::printGrafo(x)
```

```
##
##
## [1] -> 2 -> 3 -> ≡
##
## [2] -> 1 -> 3 -> 4 -> 5 -> ≡
##
## [3] -> 1 -> 2 -> 4 -> ≡
##
## [4] -> 2 -> 3 -> 5 -> 6 -> ≡
##
## [5] -> 2 -> 4 -> 6 -> ≡
##
## [6] -> 4 -> 5 -> ≡
```

```
grafoR::printGrafo(x, pesos = TRUE)
```

```
##
##
## [1] -> 2 (1) -> 3 (3) -> ≡
##
## [2] -> 1 (1) -> 3 (1) -> 4 (3) -> 5 (2) -> ≡
##
## [3] -> 1 (3) -> 2 (1) -> 4 (2) -> ≡
##
## [4] -> 2 (3) -> 3 (2) -> 5 (3) -> 6 (2) -> ≡
##
## [5] -> 2 (2) -> 4 (3) -> 6 (3) -> ≡
##
## [6] -> 4 (2) -> 5 (3) -> ≡
```

A função `vizinhos(x, vertice)` retorna os vizinhos do `vertice` no grafo `x`.

```
grafoR::vizinhos(x, 5)
```

```
## [1] 2 4 6
```

A função `grau(x, vertice)` retorna o grau do `vertice` no grafo `x`.

```
grafoR::grau(x, 2)
```

```
## [1] 4
```

A função `sequenciaGraus(x)` retorna a sequência de graus do grafo `x`.

```
grafoR::sequenciaGraus(x)
```

```
## [1] 4 4 3 3 2 2
```

Continuação da entrega do TP1

A função `excentricidade(x, vertice)` retorna os vértices que distancião a excentricidade e a excentricidade do `vertice` .

```
grafoR::excentricidade(x, 2)
```

```
## [[1]]
## [1] "6"
##
## [[2]]
## [1] 5
```

A função `raio(x)` retorna o raio de `x` e os vértices que formam o raio.

```
grafoR::raio(x)
```

```
## [[1]]
## [1] 3
##
## [[2]]
## [[2]][[1]]
## [[2]][[1]][[1]]
## [1] 5
##
## [[2]][[1]][[2]]
## [1] "1" "3" "4" "6"
```

A função `diametro(x)` retorna o diametro de `x` e os vértices que formam o diametro.

```
grafoR::diametro(x)
```

```
## [[1]]
## [1] 6
##
## [[2]]
## [[2]][[1]]
## [[2]][[1]][[1]]
## [1] 1
##
## [[2]][[1]][[2]]
## [1] "6"
##
##
## [[2]][[2]]
## [[2]][[2]][[1]]
## [1] 6
##
## [[2]][[2]][[2]]
## [1] "1"
```

A função `centro(x)` retorna o centro do grafo `x` .

```
grafoR::centro(x)
```

```
## [1] 5
```

A função `buscaProfundidade(x, vertice)` retorna a sequência de vértices visitados, uma matriz com 1 nas arestas que fazem parte da busca em profundidade e as arestas que não pertencem a busca em profundidade.

```
grafoR::buscaProfundidade(x, 2)
```

```
## [[1]]
## [1] 2 1 3 4 5 6
##
## [[2]]
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    0    1    1    0    0    0
## [2,]    1    0   -1   -1   -1    0
## [3,]    1   -1    0    1    0    0
## [4,]    0   -1    1    0    1   -1
## [5,]    0   -1    0    1    0    1
## [6,]    0    0    0   -1    1    0
##
## [[3]]
## [1] "2 - 3" "2 - 4" "2 - 5" "4 - 6"
```

A função `distanciaCaminhoMinimo(x, vertice, verticeDestino)` retorna a distância e o caminho do `vertice` até o `verticeDestino` . Foi utilizado o algoritmo de Floyd-Warshall.

```
grafoR::distanciaCaminhoMinimo(x, 2, 6)
```

```
## [[1]]
## [1] 5
##
## [[2]]
## [1] 2 4 6
```

A função `centralidade(x, vertice)` retorna a centralidade do `vertice` no grafo `x` .

```
grafoR::centralidade(x, 3)
```

```
## [1] 0.4166667
```

A função `grafoFromJSON(path, pathToSave)` pega o arquivo no caminho `path` que está em json e salva no padrão do tp no arquivo de caminho `pathToSave` .

```
grafoR::grafoFromJSON("./exemplo.json", "./json.txt")
```

A função `grafoToJSON(path, pathToSave)` pega o arquivo no caminho `path` que está no padrão do tp e salva em json no arquivo de caminho `pathToSave` .

```
grafoR::grafoToJSON("./exemplo", "./txt.json")
```

Entrega do TP2

A função `possuiCiclo(x)` retorna se o grafo `x` possui ciclo.

```
grafoR::possuiCiclo(x)
```

```
## [1] TRUE
```

A função `arvoreMinima(x, path=NULL)` retorna a um objeto do tipo grafo que é a árvore geradora mínima do grafo `x` e o seu peso total, caso passe um caminho de arquivo `path` ele irá salvar o grafo no padrão do TP com o peso total. Foi utilizado o algoritimo de Kruskal.

```
printGrafo(grafoR::arvoreMinima(x)[[1]])
grafoR::arvoreMinima(x)[[2]]
```

```
grafoR::arvoreMinima(x, "./arvore.txt")
```

```
## [[1]]
## $`1`
## $`1`[[1]]
## $`1`[[1]]$vertice
## [1] 2
##
## $`1`[[1]]$peso
## [1] 1
##
##
##
## $`2`
## $`2`[[1]]
## $`2`[[1]]$vertice
## [1] 1
##
## $`2`[[1]]$peso
## [1] 1
##
##
## $`2`[[2]]
## $`2`[[2]]$vertice
## [1] 3
##
## $`2`[[2]]$peso
## [1] 1
##
##
## $`2`[[3]]
## $`2`[[3]]$vertice
## [1] 5
##
## $`2`[[3]]$peso
## [1] 2
##
##
##
## $`3`
## $`3`[[1]]
## $`3`[[1]]$vertice
## [1] 2
##
## $`3`[[1]]$peso
## [1] 1
##
##
## $`3`[[2]]
## $`3`[[2]]$vertice
## [1] 4
##
## $`3`[[2]]$peso
```

```
## [1] 2
##
##
##
## $`4`
## $`4`[[1]]
## $`4`[[1]]$vertice
## [1] 3
##
## $`4`[[1]]$peso
## [1] 2
##
##
## $`4`[[2]]
## $`4`[[2]]$vertice
## [1] 6
##
## $`4`[[2]]$peso
## [1] 2
##
##
##
## $`5`
## $`5`[[1]]
## $`5`[[1]]$vertice
## [1] 2
##
## $`5`[[1]]$peso
## [1] 2
##
##
##
## $`6`
## $`6`[[1]]
## $`6`[[1]]$vertice
## [1] 4
##
## $`6`[[1]]$peso
## [1] 2
##
##
##
## attr(,"class")
## [1] "grafo"
##
## [[2]]
## [1] 8
```

A função `coberturaMinima(x)` retorna os vértices da cobertura minima e o número de vértices.

```
grafoR::coberturaMinima(x)
```

```
## [[1]]
## [1] 2 3 4 5
##
## [[2]]
## [1] 4
```

A função `emparelhamentoMaximo(x)` retorna as arestas do emparelhamento máximo do grafo `x`. Foi utilizado o algoritmo de Edmonds.

```
grafoR::emparelhamentoMaximo(x)
```

```
## [1] "1 - 3" "2 - 5" "4 - 6"
```