Universidade Federal do Pampa

TIPOS ABSTRATOS DE DADOS

Aula 27: Mapas, Dicionários e Grafos

Prof. Jean Felipe Cheiran
jeancheiran@unipampa.edu.br
Novembro de 2017


Boa Vista
Roraima
Amapá
Macapá
Belém
Manaus
Amazonas
Pará
São Luís
Fortaleza
Maranhão
Teresina
Ceará
Rio Grande do Norte
Natal
Paraíba
João Pessoa
Piauí
Pernambuco
Recife
Acre
Rio Branco
Porto Velho
Rondônia
Palmas
Tocantins
Alagoas
Maceió
Aracajú
Sergipe
Bahia
Salvador
Mato Grosso
Cuiabá
Goiás
Goiânia
Brasília
Minas Gerais
Belo Horizonte
Espírito Santo
Vitória
Mato Grosso do Sul
Campo Grande
São Paulo
São Paulo
Rio de Janeiro
Rio de Janeiro
Paraná
Curitiba
Santa Catarina
Florianópolis
Rio Grande do Sul
Porto Alegre

Vetores x Mapas

- Vetores são estruturas **compostas unidimensionais homogêneas**.
- Mapas são estruturas **compostas unidimensionais (geralmente) homogêneas**.

Vetores x Mapas

- Vetores **são indexados**.
- Mapas **são indexados**.

- Vetores **possuem tamanho pré-definido**.
- Mapas são **crescem e diminuem por demanda**.

- Vetores são **indexados por números inteiros**.
- Mapas são **indexados por qualquer coisa**.

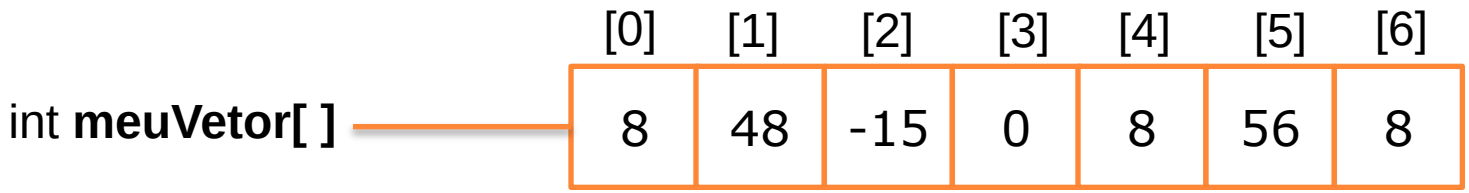
Mapas

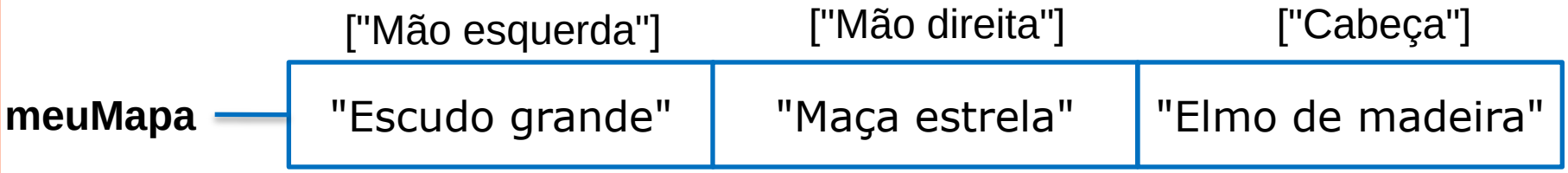
- Estruturas de dados que armazenam entradas no formato **(chave,valor)**, também conhecidos como **(key,value)** ou **(k,v)**.

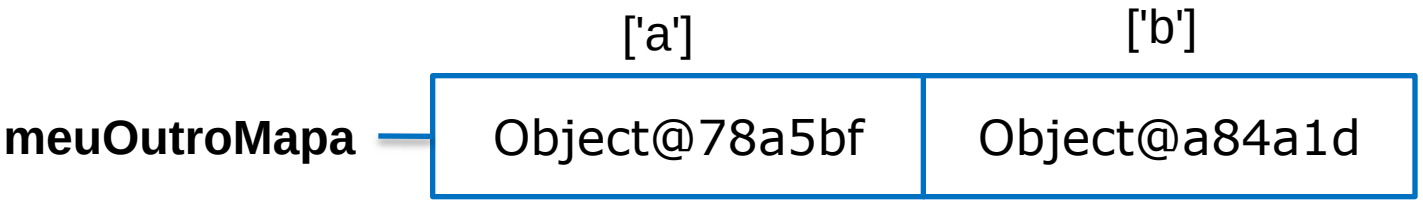
- Uma **chave** é um elemento **único** usado para endereçar a posição de um valor.

- Um **valor** são dados de qualquer natureza (que não precisam ser únicos) que é mantido pelo mapa.

Vetores x Mapas







Mapas – operações comuns

size( ) – retorna o tamanho do mapa, ou seja, a quantidade de pares (k,v) armazenados.

isEmpty( ) – retorna se o mapa está ou não vazio.

get(k) – retorna o objeto (valor) que está na posição k (chave), mas não remove ele.

put(k, v) – coloca um objeto v (valor) na posição k (chave) do mapa; se essa posição já estiver ocupada, então sobre-escreve o valor anterior.

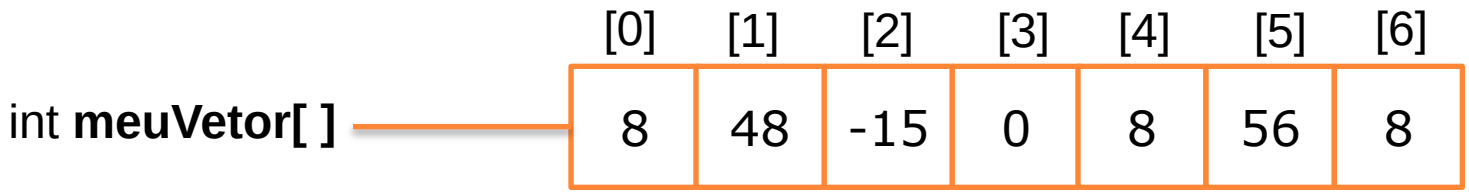
remove(k) – retorna e remove o objeto (valor) que está na posição k (chave).

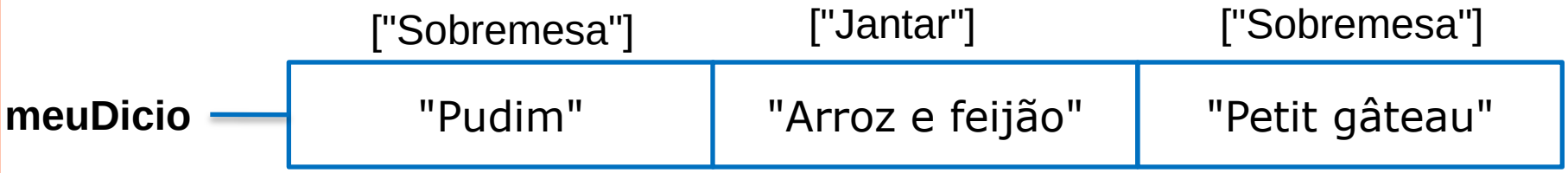

casual; acaso, eventualidade. [Cf. casual]
casualismo. [De casual + -ismo.] S. m.
atribui ao acaso a sucessão dos fenômenos
casualista. [De casualismo + -ista.] Adj.
ao, ou que é seguidor do casualismo.
do casualismo.
casualização. [De *casualizar (< casual +
S. f. Estat. V. acidentalização.
casuar. [Do malaio, pelo fr. casoar.] S. m.
ção geral para várias aves, semelhantes
que se distribuem pela Austrália
Nova Guiné e ilhas adjacentes. [Sin.,
casuariídeo. [Do tax. Casuarius
-ídeo; Casuariidae.] Zool. S. m. 1. Espécie
ariídeos. • Adj. 2. Pertencente ou relativo
casuariídeos. S. m. pl. Zool. Família de
riiformes que reúne aves cursoras, ratitas
pesadas, com penas pretas semelhantes
míferos, desprovidas de cauda e de asas
reduzidas, com três a cinco
cataplasmado.
cataplasmar.
catapléctico.
cataplético. Adj. V. catapléctico.
cataplexia

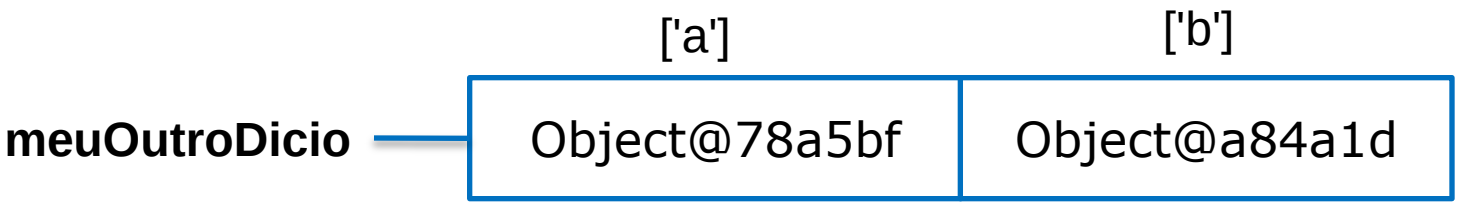
Dicionários

- Dicionários são estruturas idênticas a Mapas, mas podem manter vários valores com uma mesma chave.
- Assim, Dicionários são estruturas de dados que armazenam entradas no formato **(chave,valor)**, também conhecidos como **(key,value)** ou **(k,v)**.
- Uma **chave** é um elemento (que não precisa ser único) usado para endereçar a posição de um valor.
- Um **valor** são dados de qualquer natureza (que não precisam ser únicos) que é mantido pelo dicionário.

Vetores x Dicionários







Dicionários – operações comuns

size( ) – retorna o tamanho do dicionário, ou seja, a quantidade de pares (k,v) armazenados.

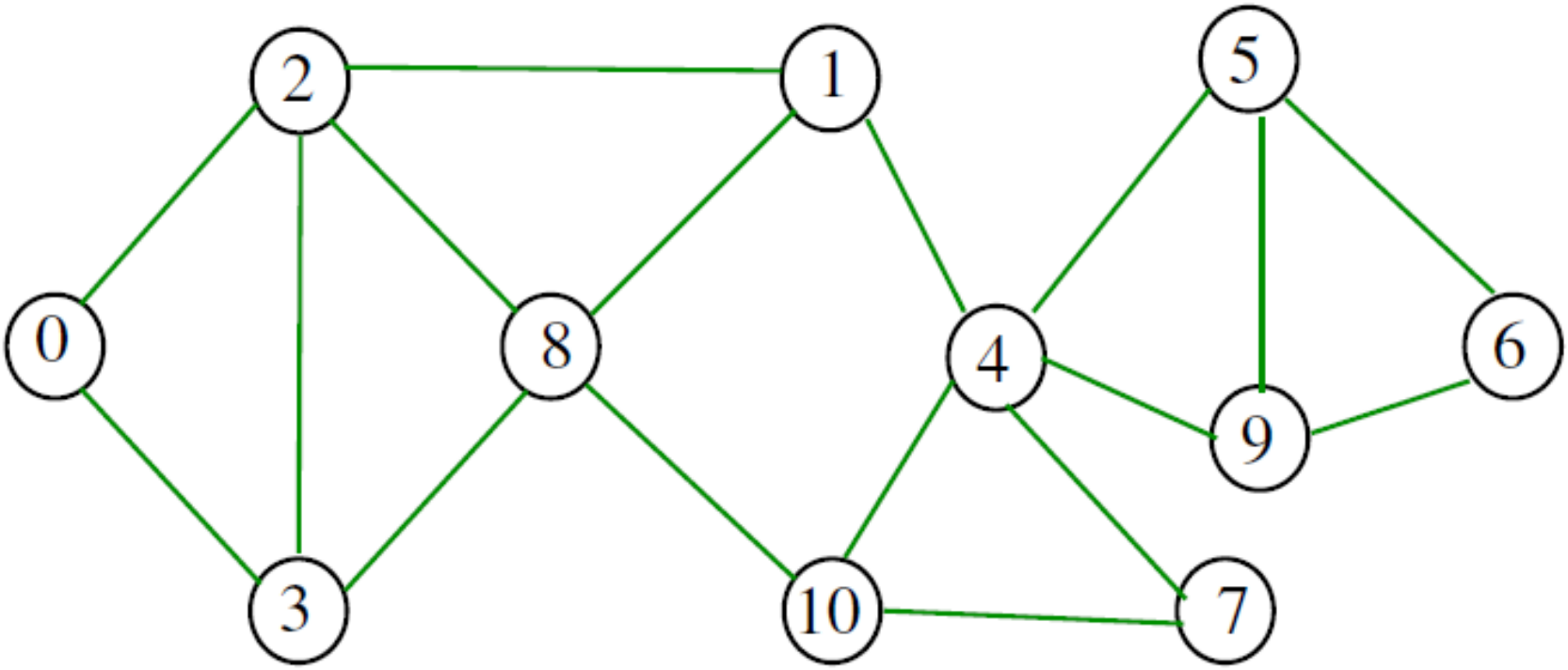
isEmpty( ) – retorna se o dicionário está ou não vazio.

find(k) – retorna o primeiro objeto (valor) achado com chave k, mas não o remove.

findAll(k) – retorna uma coleção com todos os objetos (valor) com chave k, mas não os remove.

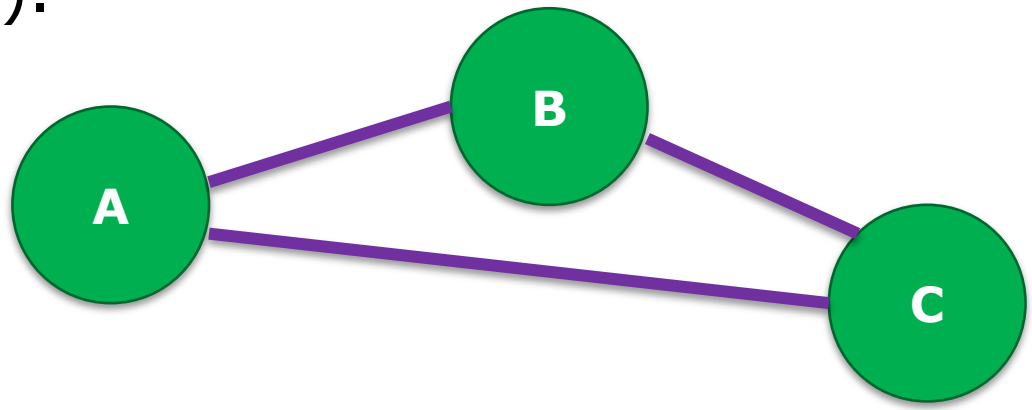
insert(k, v) – coloca um objeto v (valor) na posição k (chave) do dicionário; nunca sobre-escreve outros valores.

remove(k) – retorna e remove o primeiro objeto (valor) que está na posição k (chave).


2
1
5
0
8
4
6
9
3
10
7

Grafos

- Estruturas de dados flexíveis e amplamente usadas em computação.
- Estruturas matemáticas que modelam relacionamentos de pares de objetos.
- Compostos por **vértices** (ou nós) e **arestas** (arcos).



Grafos – operações comuns

vertices( ) – retorna uma coleção de todos os vértices do grafo.

arestas( ) – retorna uma coleção de todas as arestas do grafo.

arestasIncidentes(v) – retorna uma coleção de todas as arestas incidentes ao vértice **v**.

verticesFinais(a) – retorna uma coleção de todos os vértices finais da aresta **a**.

inserirVertice(x) – cria e retorna um novo vértice contendo o objeto **x** armazenado.

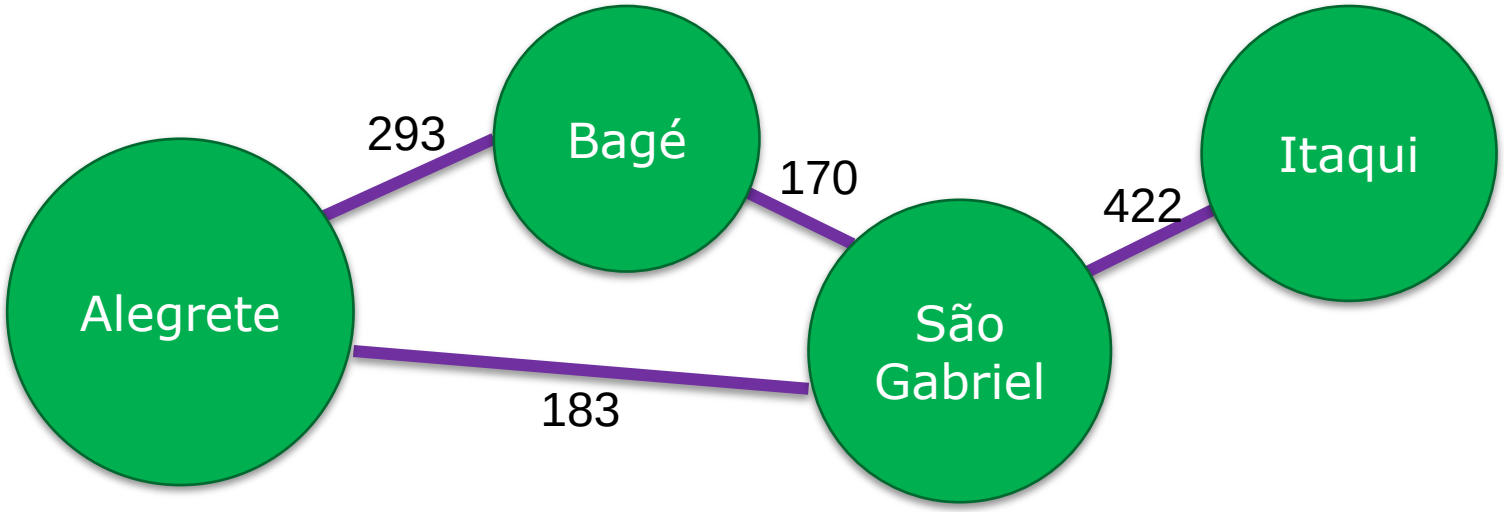
inserirAresta(v, w, x) – cria e retorna uma nova aresta não-dirigida entre os vértices **v** e **w** contendo o objeto **x** armazenado.

(ainda precisa de métodos para remover e substituir objetos)

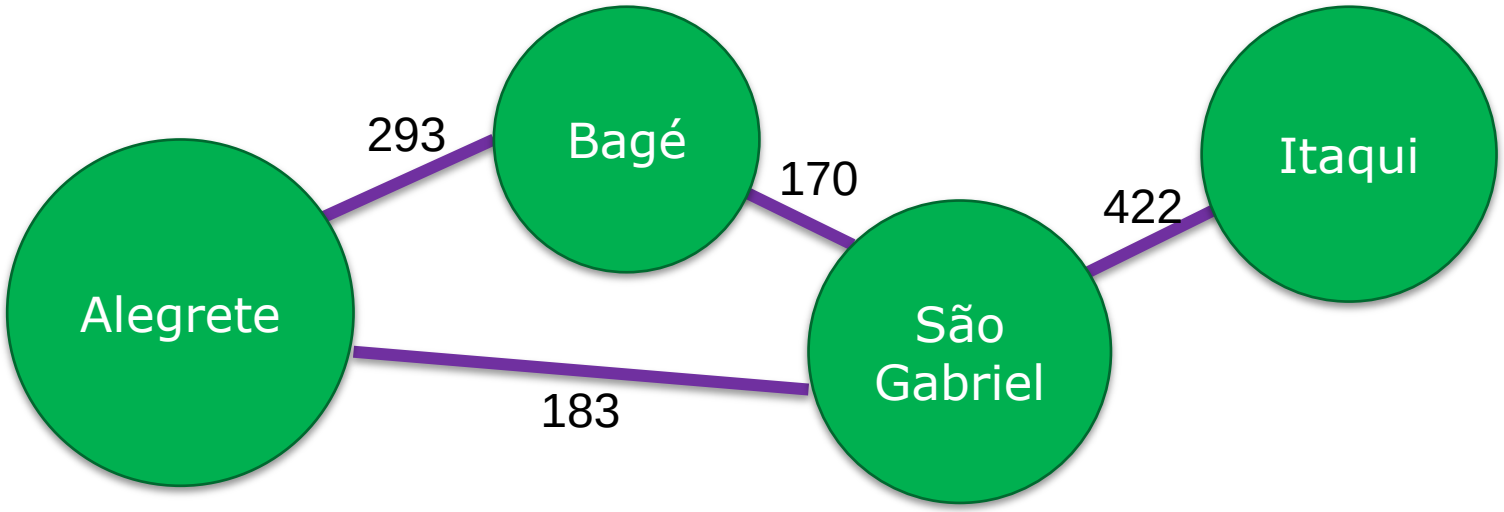
Grafos – como montar seu TAD?

- Há três técnicas bastante usadas para implementar um grafo em um TAD:
  - Lista de arestas
  - Lista de adjacência
  - Matriz de incidência
- Veremos aqui em detalhes a implementação por matriz de adjacência.

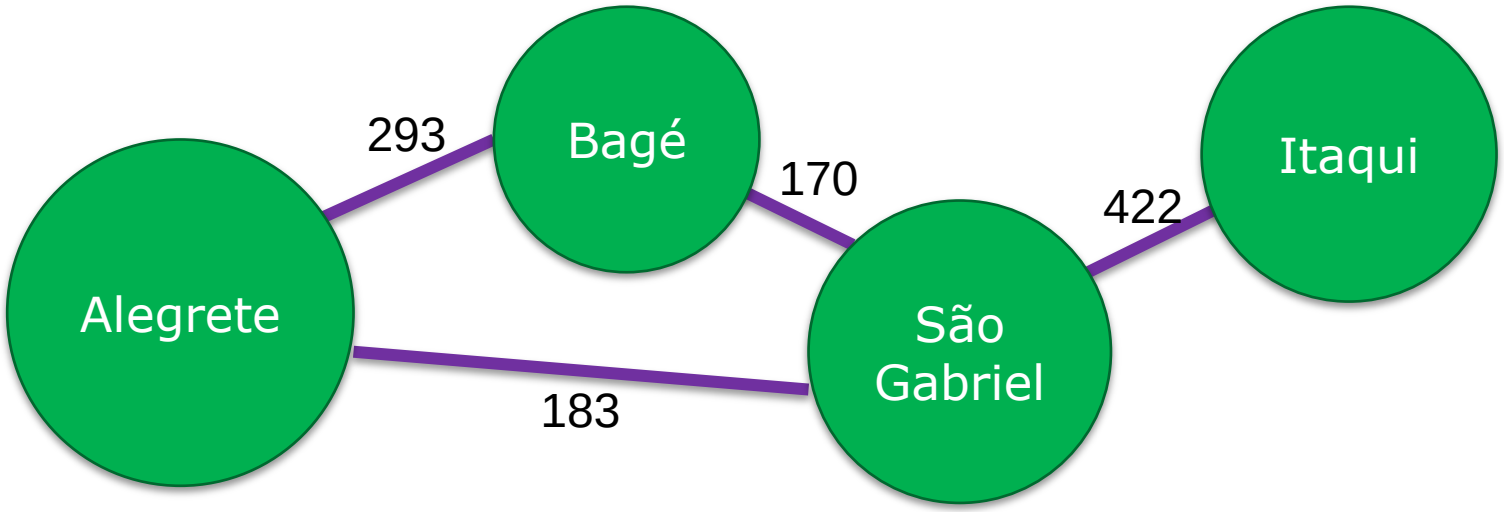
Grafos – matriz de adjacência



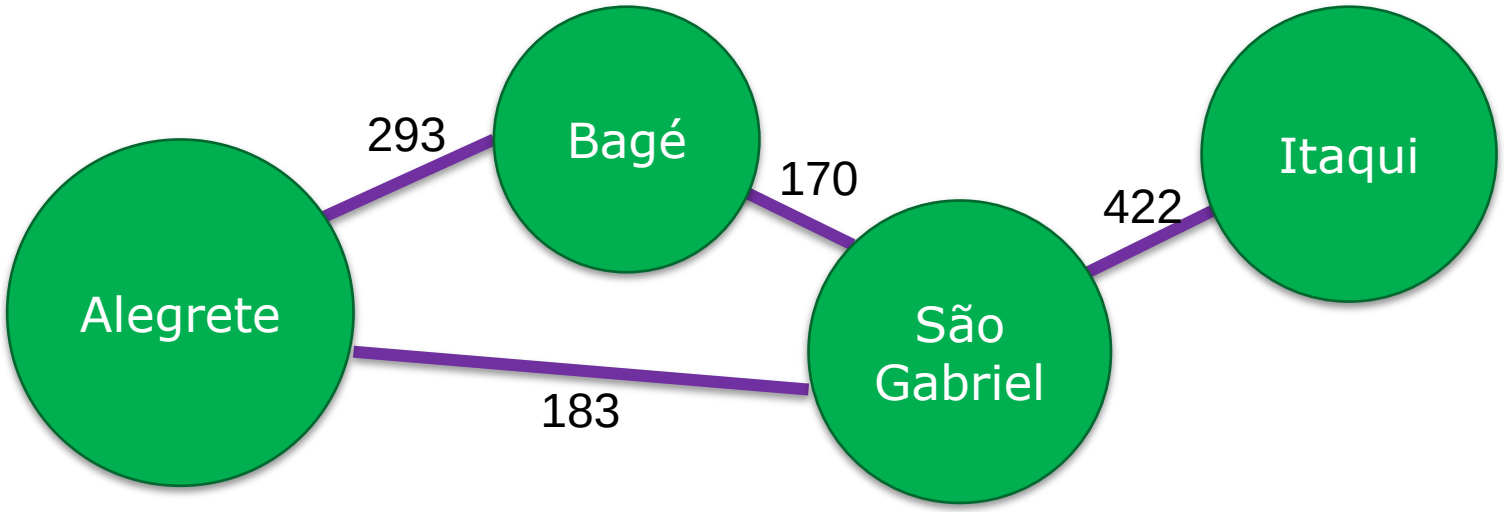
- O grafo acima é não direcionado, e armazena informação adicional nos vértices e nas arestas.
- Nesse caso, precisaremos de uma matriz indicando quais vértices estão ligados, uma lista de arestas contendo seus dados e uma lista de vértices contendo seus dados.



	Alegrete	Bagé	São G.	Itaqui
Alegrete				
Bagé				
São G.				
Itaqui				



	Alegrete	Bagé	São G.	Itaqui
Alegrete	0	1	1	0
Bagé	1	0	1	0
São G.	1	1	0	1
Itaqui	0	0	1	0

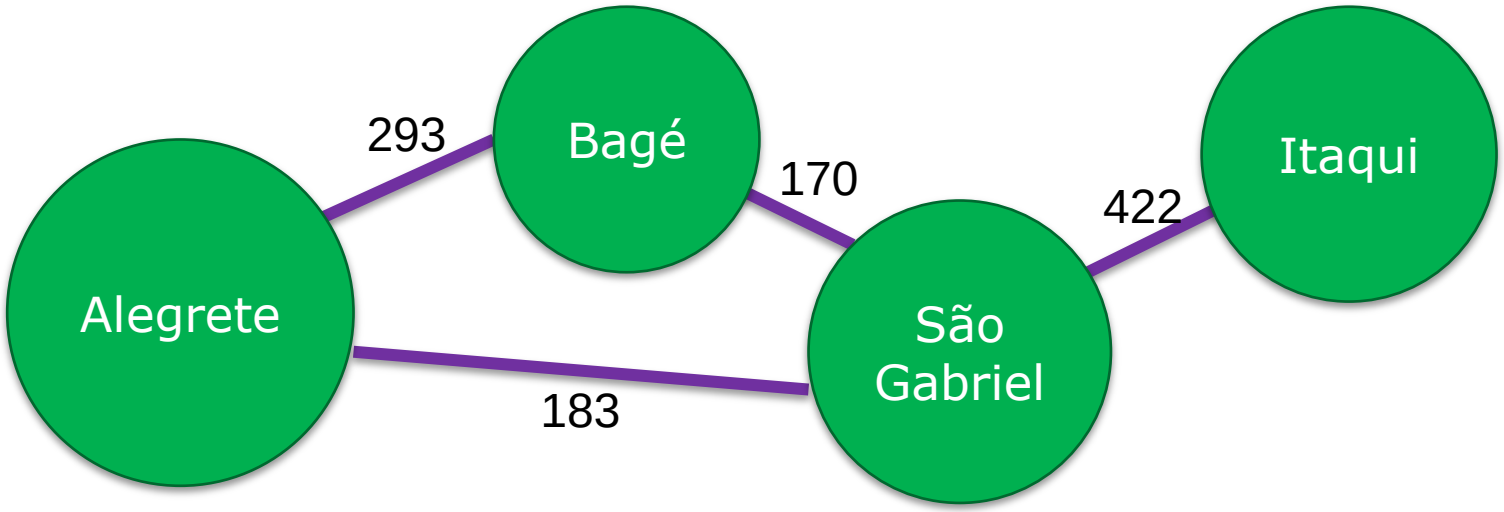


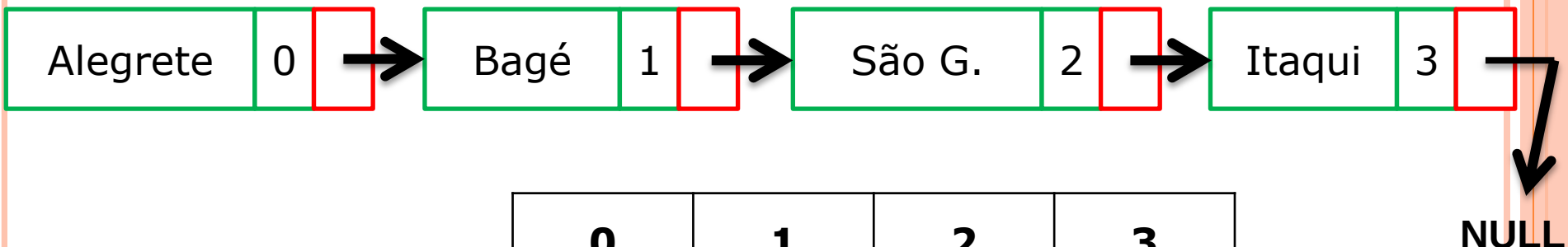
0. Alegrete
1. Bagé
2. São Gabriel
3. Itaqui

	0	**1**	**2**	**3**
0	0	1	1	0
1	1	0	1	0
2	1	1	0	1
3	0	0	1	0

- A lista dos vértices será representada por uma lista encadeada.

- Cada nodo da lista terá:
  - **Dados** (o nome da cidade, e o índice na matriz de adjacência).
  - **Referência** para o próximo nodo da lista.

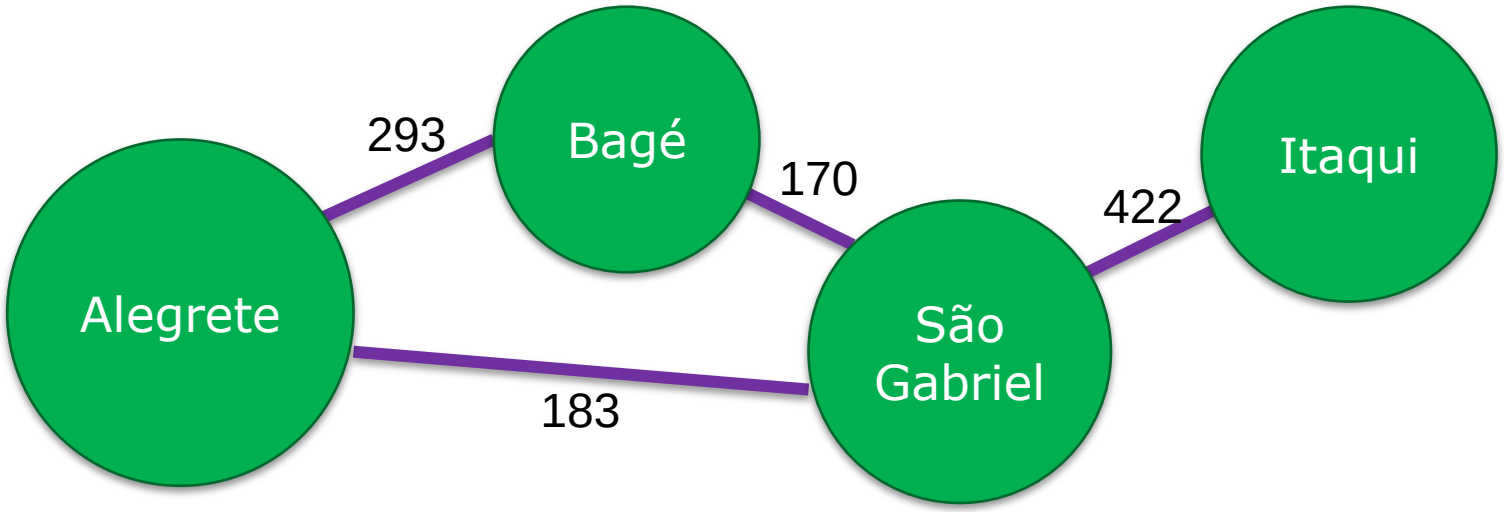


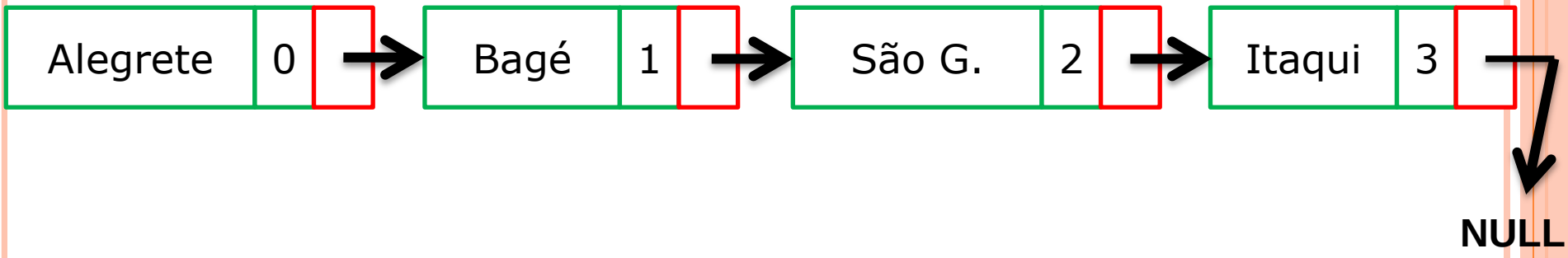


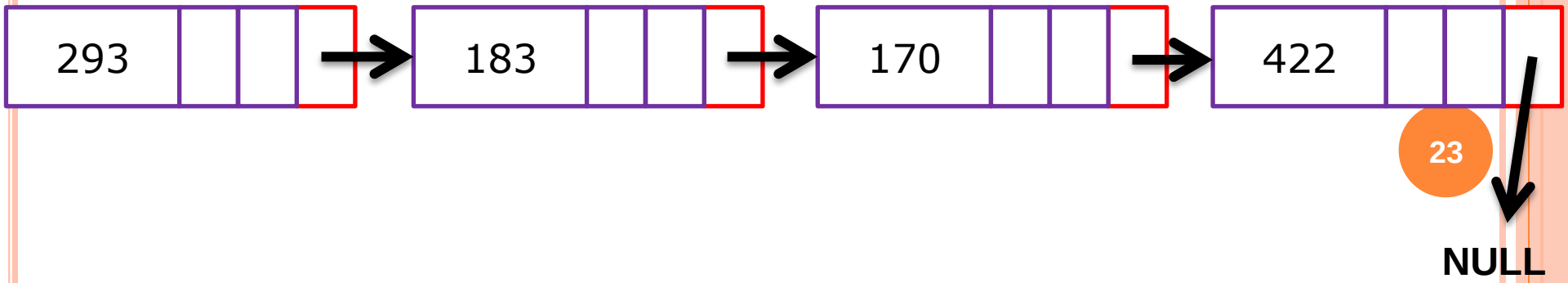
	0	**1**	**2**	**3**
0	0	1	1	0
1	1	0	1	0
2	1	1	0	1
3	0	0	1	0

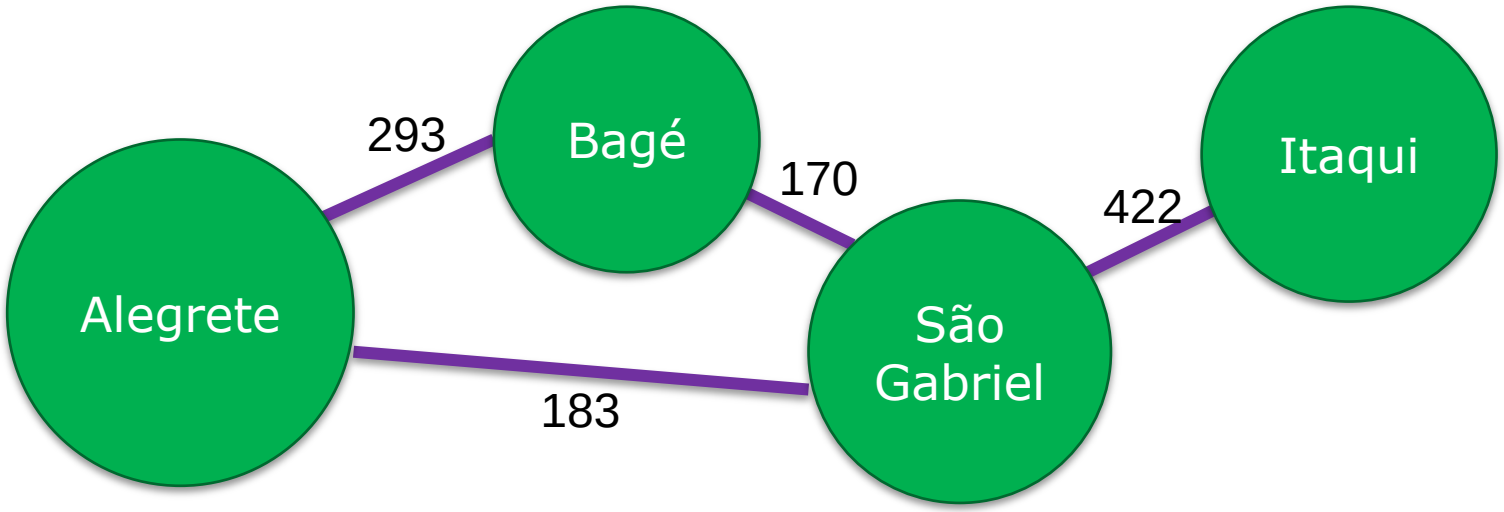
- A lista das arestas será representada por uma lista encadeada.

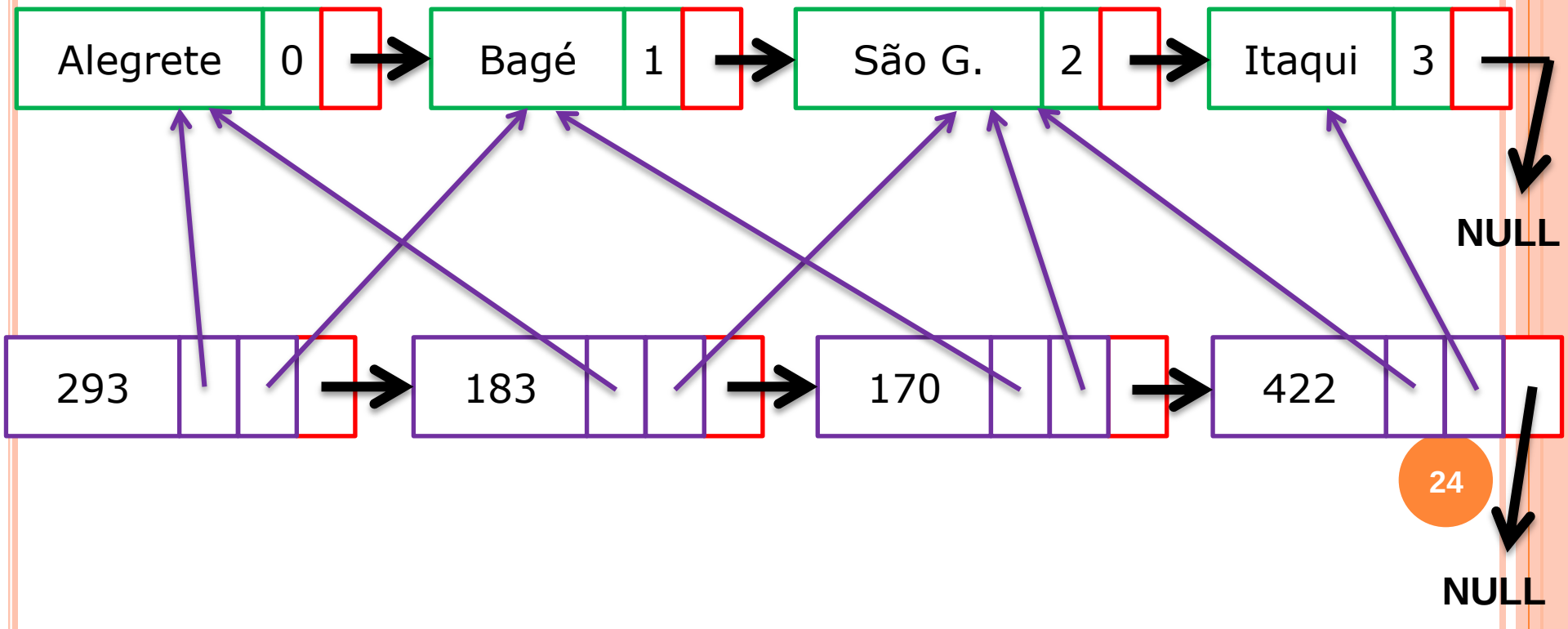
- Cada nodo da lista terá:
  - **Dados** (o número da aresta, referência para o nodo do primeiro vértice final, referência para o nodo do segundo vértice final).
  - **Referência** para o próximo nodo da lista de arestas.


Alegrete
Bagé
São Gabriel
Itaqui
293
170
422
183


Alegrete
0
Bagé
1
São G.
2
Itaqui
3
NULL

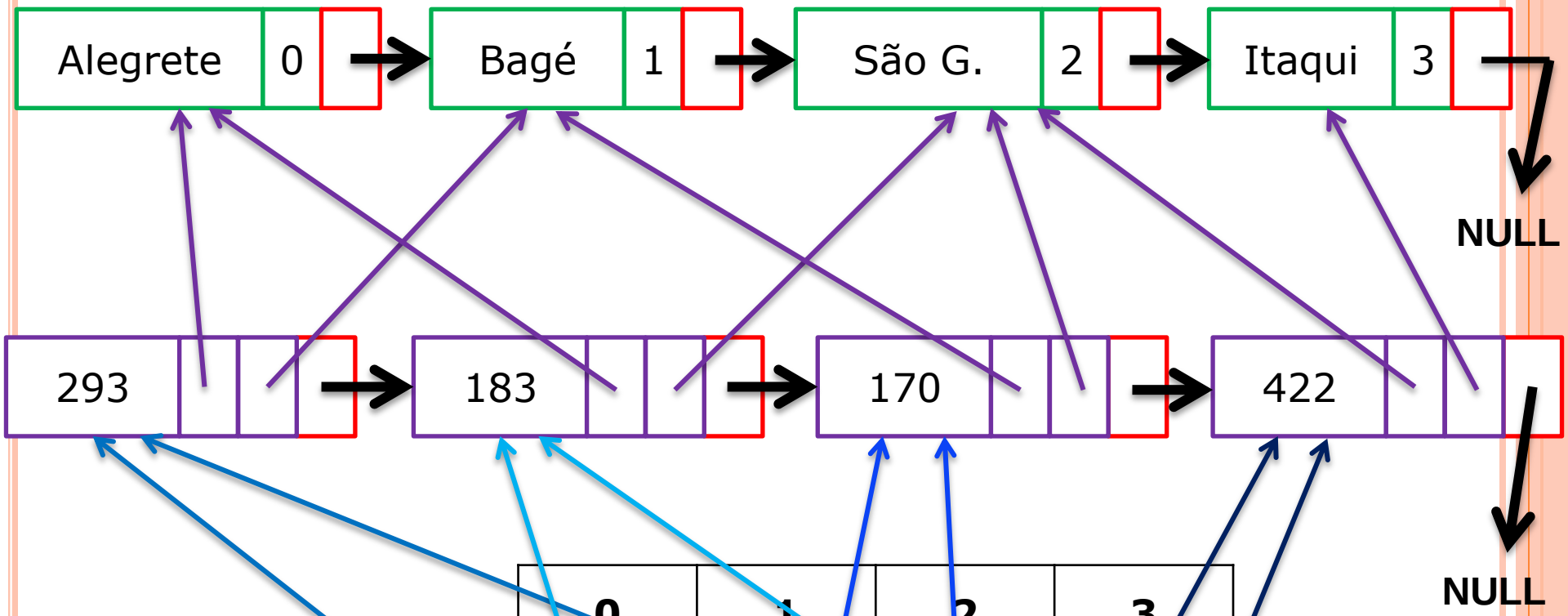

293
183
170
422
NULL


Alegrete
Bagé
São Gabriel
Itaqui
293
170
422
183


Alegrete
0
Bagé
1
São G.
2
Itaqui
3
NULL
293
183
170
422
NULL

- A matriz de adjacência deveria permitir um acesso mais rápido às arestas dos vértices envolvidos.

- Assim, em vez de armazenarmos valores inteiros ou booleanos, vamos armazenar referências para os nodos da lista de arestas.



	0	1	2	3
0	null			null
1		null		null
2			null	
3	null	null		null

Grafos – matriz de adjacência

- Os métodos que retornam todos os vértices e todas as arestas são facilmente satisfeitos percorrendo as listas.

- Achar todas as arestas de um vértice **v** exige:
  - Descobrir o índice do vértice pela lista de vértices.
  - Descobrir os nodos de arestas incidentes seguindo a linha ou a coluna na matriz de adjacência pelo índice.

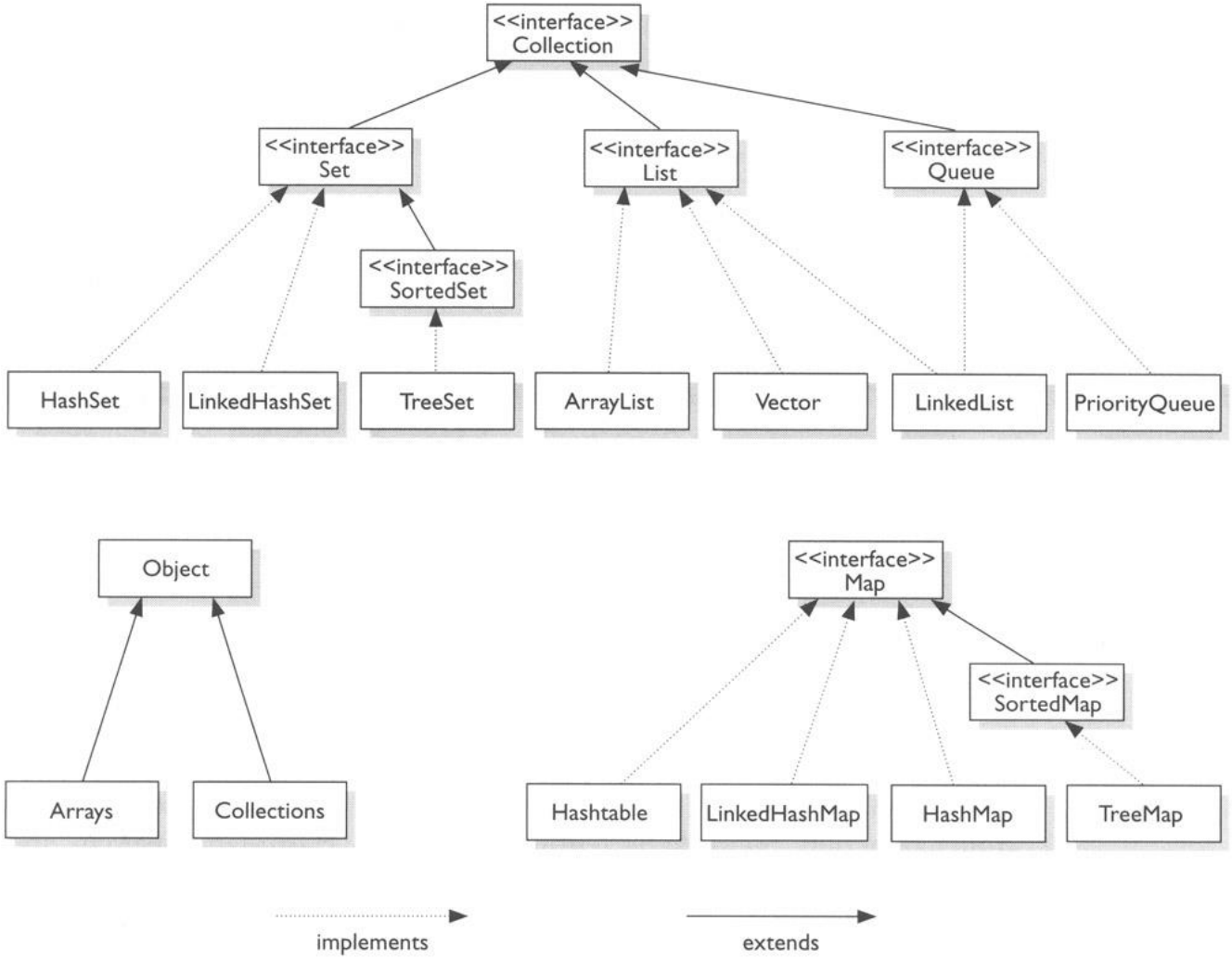
- Achar os dois vértices de uma aresta **a** exige:
  - Percorrer a lista de nodos aresta para encontrar a aresta **a**.
  - Descobrir os nodos vértice referenciados pelo nodo aresta.

Grafos – matriz de adjacência

- Adicionar um novo vértice **v** exige:
  - Criar e incluir um nodo na lista de vértices.
  - Ampliar a matriz de adjacência e preencher a nova linha e a nova coluna com **null**.

- Adicionar uma nova aresta **a** exige:
  - Descobrir as referências e os índices dos vértices finais da aresta **a**.
  - Criar e incluir um nodo na lista de arestas (incluindo referência para os vértices finais).
  - Atualizar a matriz de adjacência nos índices dos vértices finais colocando uma referência para o nodo da aresta **a**.

Java Collection / Collections



Java Collection / Collections

- **Collections** é um *framework* nativo de Java que inclui interfaces para tipos abstratos, implementações para tipos abstratos e algoritmos que atuam sobre coleções.

- **Collection** é uma interface que serve como base para inúmeros tipos abstratos de dados nativos (conjunto, lista, fila, etc.).

Coleções

- Uma coleção é um amontado de objetos organizados de alguma forma especial e com propriedades específicas.
- Uma fila é uma coleção de objetos com métodos específicos e propriedade FIFO.
- Uma lista encadeada é uma coleção de objetos ligados, ordenados e trafegáveis em uma única direção.
- Um conjunto é uma coleção de objetos sem ordem específica e que não permite duplicatas.

Coleções mais usadas

- ArrayList
  - Implementação de uma **lista indexada** (ou um vetor redimensionável).
- LinkedList
  - Implementação de uma **lista encadeada** que também oferece métodos para acesso LIFO e FIFO.
- HashMap
  - Implementação de um **mapa** que usa hashCode para endereçar objetos em sua estrutura interna.

Algoritmos mais usados

- Collections.sort
  - Aplica uma implementação de ***merge sort*** **estável** e adaptável com desempenho médio melhor que n $\log_2$n sobre uma lista.
- Collections.copy
  - Copia todos os elementos de uma lista para outra lista.
- Collections.swap
  - Troca dois elementos de uma lista de lugar.
- Collection.binarySearch
  - Aplica uma implementação de **busca binária** sobre uma lista ordenada de forma crescente.
- Collection.min e Collection.max
  - Encontra o valor mínimo ou máximo de uma coleção.