ALGORITMOS E ESTRUTURAS DE DADOS
2016/2017
EXERCÍCIOS DE TESTE/EXAME

Armanda Rodrigues

Agência Imobiliária

- Uma agência imobiliária pretende construir uma aplicação para gerir os imóveis que tem para transacionar.
- Os dados associados a cada ***imóvel*** são: um *número-chave* único que o identifica, o *tipo* (apartamento, moradia, etc.), o *número de assoalhadas*, o *ano* de construção, a *morada*, o *preço*, uma *planta*, uma *fotografia*, o *estado de conservação* (em construção, a estrear, recente, remodelado ou a necessitar de obras), o *estado de venda* (disponível, reservado ou sinalizado) e o nome do *proprietário*.
- Cada ***proprietário*** tem um *nome* (que se assume ser único, para simplificar), uma *morada*, um *número de telefone* e um *número de contribuinte*.

Operações

a) Inserir um proprietário, dando o nome, a morada, o número de telefone e o número de contribuinte.
A operação só será efetuada se não existir um proprietário com esse nome.

b) Inserir um imóvel, dando todos os seus dados.
A operação só será efetuada se não existir um imóvel com esse número-chave e se existir um proprietário com esse nome.

c) Remover um imóvel, dando o seu número-chave.
A operação só será efetuada se existir um imóvel com esse número-chave.

Operações

d) Obter todos os dados de um imóvel, dando o seu número-chave.

e) Obter a morada, o número de telefone, o número de contribuinte e os números-chave de todos os imóveis de um proprietário, dando o seu nome.
Os números-chave deverão aparecer pela ordem por que foram inseridos no sistema.

f) Listar todos os imóveis com um determinado número de assoalhadas que se encontrem dentro de um intervalo de preços.
A listagem deve estar ordenada crescentemente por preço e, em caso de igualdade no preço, por ordem crescente de número-chave.
Informação por imóvel: número-chave, tipo, morada, preço, estado de conservação e estado de venda.

Quantidades e Exercício

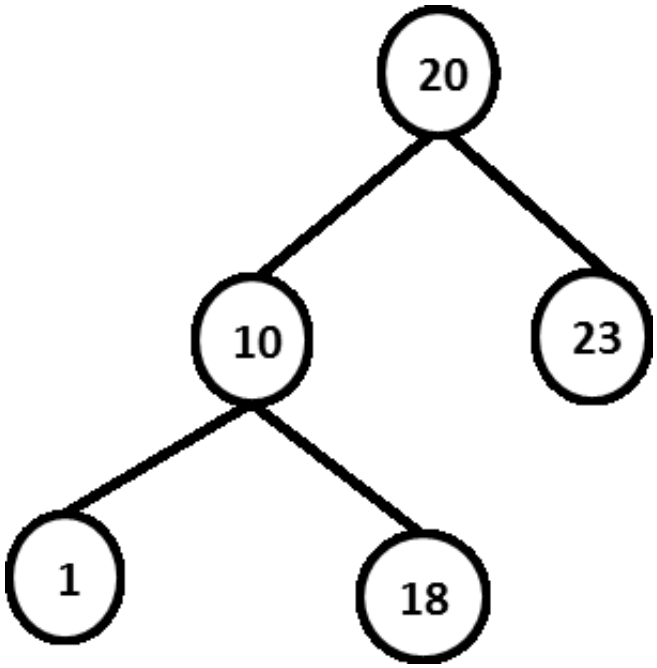
- Prevê-se que o número de imóveis e o número de proprietários não ultrapassem 100 000.
- Em cada momento, cada proprietário não deverá ter mais de dez imoveis. Assuma que qualquer imóvel tem entre uma e vinte assoalhadas.
- Explicite detalhadamente as estruturas de dados mais adequadas para implementar esta aplicação, descreva sumariamente os algoritmos para efetuar as seis operações (enumeradas de (a) a (f)) e calcule (justificando) as suas complexidades temporais, no caso esperado.

AVLs

- Considere a árvore AVL apresentada na Figura. Em cada nó está apenas representada a chave do mesmo:

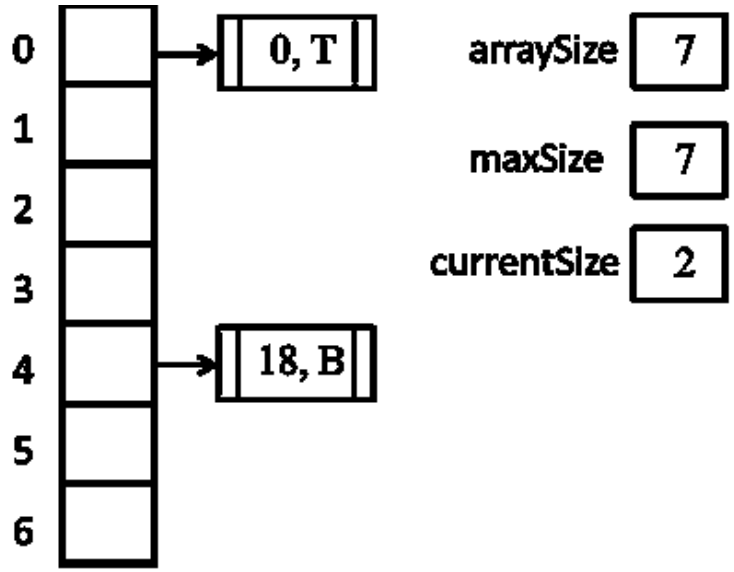
a) Desenhe a árvore AVL resultante de remover o elemento com a chave 23 da árvore apresentada;

b) Desenhe a árvore AVL resultante de inserir um elemento com a chave 21 na árvore apresentada na Figura 1. Atenção que a inserção deve ser feita na **árvore original,** representada na Figura.



TDAs

- Considere a tabela de dispersão aberta apresentada na Figura.
- Esta tabela constitui uma implementação do tipo abstrato de dados (TAD) Dicionário (Dictionary<K,V>), tal como apresentado nas aulas de AED.
- A tabela foi criada para conter no máximo 7 entradas (*maxSize*) e é constituída por um vetor de 7 posições (*arraySize*) onde, em cada posição, existe uma lista duplamente ligada, ordenada crescentemente pela chave das entradas, que guarda as colisões (entradas cujo resultado da aplicação da função de dispersão à chave da entrada é o mesmo).

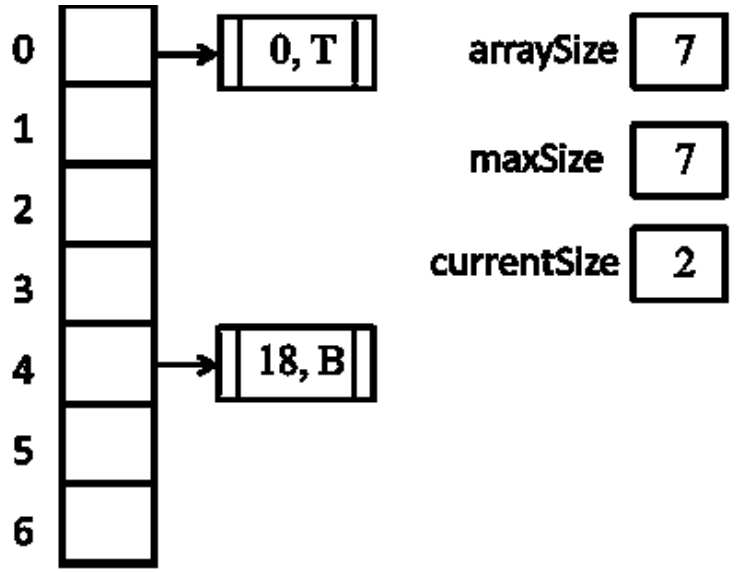


Função de Dispersão

```
// Returns the hash value of the
// specified key.
protected int hash( int key ) {
   return Math.abs( key ) %
          arraySize;
}
```

TDAs (cont.)

- Neste momento, a tabela contém duas entradas (*currentSize*).
- As entradas são do tipo Entry<Integer, Character>.
- A posição de cada entrada é encontrada pela aplicação da função de dispersão hash à chave da entrada.
  - A entrada (0,T) foi guardada na lista existente no índice 0 da tabela;
  - A entrada (18,B) foi guardada na lista pertencente ao índice 4.



Função de Dispersão

```
// Returns the hash value of the
// specified key.
protected int hash( int key ) {
   return Math.abs( key ) %
          arraySize;
}
```

TDAs (cont.)

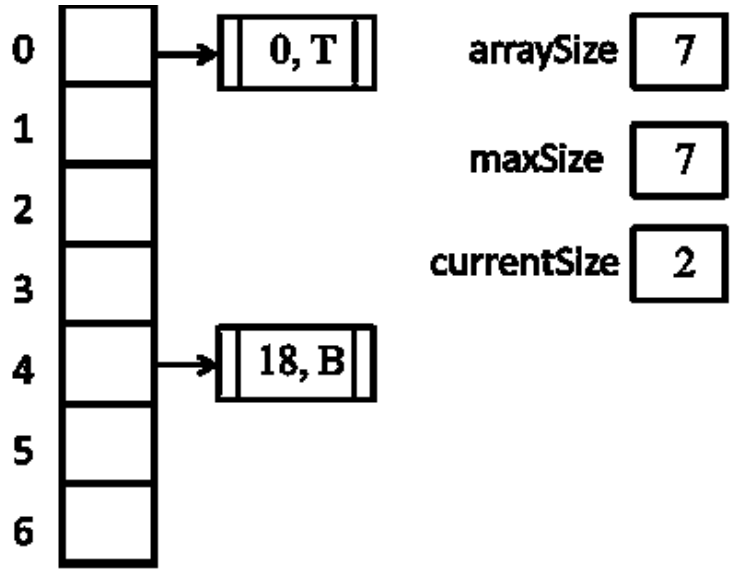
Dada a tabela de dispersão aberta apresentada na Figura, desenhe a tabela resultante de executar as inserções pedidas abaixo:

a) Inserção da entrada (9,T);

b) Inserção da entrada (18,X);

c) Inserção da entrada (11,C).

Com a tabela apresente também os valores finais das variáveis de instância arraySize, maxSize e currentSize.



StrangeIterator

- Considere pares (*e,b*), onde *e* é um elemento do tipo E e *b* é um valor booleano. O elemento *e* diz-se *verdadeiro*, se o booleano *b* for *true*; e diz-se *falso* noutro caso.
- Dada uma sequência *S*, daqueles pares, pretende-se obter uma permutação *S'* das primeiras coordenadas dos pares de *S* com as seguintes características:
  - Os elementos verdadeiros ocorrem antes dos elementos falsos;
  - A ordem dos elementos verdadeiros é a ordem original;
  - A ordem dos elementos falsos é a ordem inversa da original.
- Exemplo: Se *S* = < (1, T), (2, T), (3, F), (4, T), (5, T), (6, F), (7, F)> então *S'* = <1, 2, 4, 5, 7, 6, 3>;

Desenvolver a classe StrangeIterator

```
public interface Pair<F, S> {
   //returns the first coordinate of the pair.
   F getFirst();

   //return the second coordinate of the pair.
   S getSecond();
}

public class StrangeIterator<E> implements Iterator<E> {

   public StrangeIterator (Iterator <Pair<E, Boolean>> sequence);

   //returns true iff the iteration has more elements.
   public boolean hasNext();

   //returns the next element in the iteration.
   public E next() throws NoSuchElementException;
}
```

Calcular a complexidade no melhor e no pior caso