	MC	PC	CE
HT	O(1)	O(n)	O(1)
BST	O(1)	O(n)	O(log n)
AVL	O(1)	O(log n)	O(log n)
list	O(1)	O(n)	O(n)

Número: **Sala:**

Nome:

Algoritmos e Estruturas de Dados 2022/23
Segundo Teste – 16 de dezembro de 2022 (incompleto)

Atenção:

- Os Anexos ao teste poderão ser-lhe úteis.

Regras:

- O teste tem a duração de 2 horas.
- Não são permitidos portáteis nem telemóveis.
- O teste é sem consulta.
- Não há esclarecimento de dúvidas. Se suspeitar que o enunciado tem algum erro, deve avisar o docente.
- O teste pode ser resolvido a lápis.
- Na mesa pode ter consigo caneta, lápis e borracha.
- **NÃO Pode desagrafar o teste.**
- Qualquer aluno envolvido numa fraude reprova na disciplina.

Grupo I – Manipulação de Estruturas de Dados

1. Considere a implementação de tabela de dispersão aberta estudada nas aulas de AED (`SepChainHashTable<K,V>`). Esta tabela constitui uma implementação do tipo abstrato de dados (TAD) Dicionário (`Dictionary<K,V>`), tal como também apresentado nas aulas.

   Pedimos-lhe que desenhe uma tabela criada para conter no máximo 7 entradas (maxSize), sendo constituída por um vetor de 7 posições (*arraySize*). Em cada posição do vetor, 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). A função de dispersão da tabela que deverá desenhar é definida da seguinte forma:

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

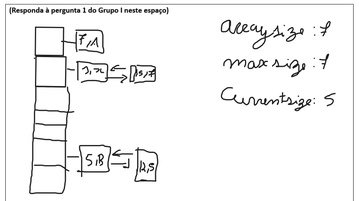
   Uma vez desenhada a tabela vazia, deverá inserir, na mesma, um conjunto de entradas, apresentadas abaixo. As entradas são do tipo `Entry<Integer, Character>`. As entradas a inserir são as seguintes:

   (1, 'K') (15, '7') (7, 'A') (15, 'K') (12, '5') (5,'8')

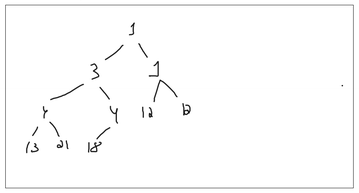
   As entradas devem ser inseridas pela ordem em que são listadas acima. Não deve, neste exercício, preocupar-se com uma possível necessidade de efetuar um processo de `rehash()`. Além do desenho da tabela de dispersão, deve ainda incluir, na sua resposta, os valores finais de *arraySize*, *maxSize* e *currentSize* (número de entradas contidas na tabela, no final do processo de inserção).

(Responda à pergunta na página seguinte)

(Responda à pergunta 1 do Grupo I neste espaço)



2. Neste exercício deve desenhar uma **Árvore Binária Completa Esquerda com prioridade (Heap), organizada por mínimos** (tal como apresentado nas aulas de AED) contendo as seguintes chaves: ~~12, 12, 1, 3~~, 4, ~~7~~, 4, 21 e 18.



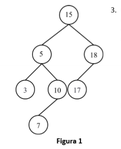
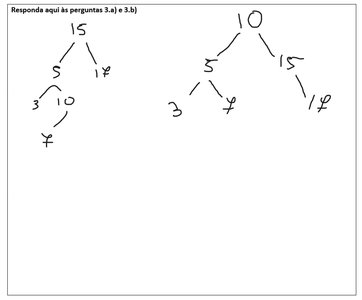


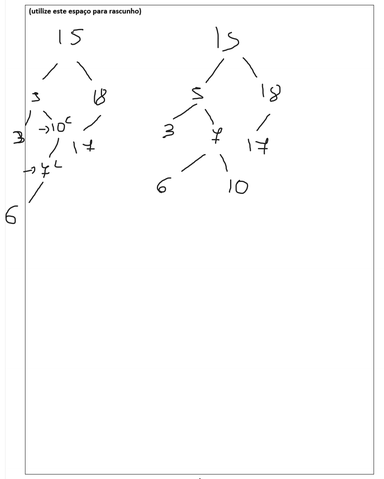
Figura 1

3. Considere a Árvore AVL apresentada na Figura 1. O valor apresentado dentro do nó de cada árvore refere a chave do mesmo nó. Tendo em conta estes dados:
   a) Desenhe a árvore binária de pesquisa resultante de remover o nó com chave 18 na árvore da Figura sem efetuar rotações. **Deve utilizar os algoritmos apresentados nas aulas teóricas para o efeito.** Considera que a árvore resultante é uma árvore AVL? Se a sua resposta for negativa, efetue a rotação necessária para transformá-la em AVL. **Deve utilizar os algoritmos apresentados nas aulas teóricas para o efeito.**
   b) Desenhe a árvore binária de pesquisa resultante de inserir um nó com chave 6 na árvore **original** da Figura, sem efetuar rotações. **Deve utilizar os algoritmos apresentados nas aulas teóricas para o efeito.** Considera que a árvore resultante é uma árvore AVL? Se a sua resposta for negativa, efetue a rotação necessária para transformá-la em AVL. **Deve utilizar os algoritmos apresentados nas aulas teóricas para o efeito.**

Responda aqui às perguntas 3.a) e 3.b)



(utilize este espaço para rascunho)



Grupo II – Tipos Abstratos de Dados e Estruturas de Dados

Neste grupo deve analisar o problema apresentado e conceber uma solução para o problema completo de acordo com os requisitos descritos. Deverá depois responder a cada uma das questões, de acordo com a solução que concebeu. LEIA TODA A PERGUNTA ATÉ AO FIM ANTES DE COMEÇAR A RESPONDER.

Pedimos-lhe que considere o desenvolvimento de um sistema de gestão de rankings de jogadores de um jogo online. O sistema contempla Jogadores (*Players*) e armazena informação sobre pontos obtidos pelos jogadores enquanto jogam. Cada sessão incrementa pontos ao jogador e os pontos totais associados a cada jogador podem ser refletidos na valorização do jogador, sendo-lhe atribuídas um número de estrelas, desta forma:

- Pontos < 100 000 – 0 estrelas;
- 100 000 <= Pontos < 200 000 – 1 estrela;
- 200 000 <= Pontos < 300 000 – 2 estrelas;
- 300 000 <= Pontos < 400 000 – 3 estrelas;
- Pontos >= 400 000 – 4 estrelas.

Com uma frequência determinada pelo site do jogo, os pontos são limpos e a valorização é reinicializada. Existem duas formas de ordenação (*ranking*) dos jogadores, por pontos e por número de estrelas.

Na sequência desta descrição, o sistema a desenvolver deve contemplar as seguintes operações:

Op1. Adicionar jogador recebendo a seguinte informação: login do jogador, nome e idade. O jogador é criado com 0 pontos e 0 estrelas. A inserção do jogador só tem sucesso se o login ainda não existir no sistema;

Op2. Adicionar pontos a jogador, recebendo o login e o número de pontos a adicionar (a somar aos pontos totais) ao perfil do jogador. Esta operação só terá sucesso se o login do jogador já existir no sistema e poderá implicar alterações nos rankings de jogadores;

Op3. Consultar dados do jogador, recebendo o login do jogador. Esta operação só terá sucesso se o login do jogador já existir no sistema e deverá devolver todos os dados do jogador, incluindo número de pontos total atual e número de estrelas;

Op4. Listar ranking de todos os jogadores totais. Esta operação deverá imprimir os dados de cada jogador, por ordem do número de pontos, descendentemente (iniciando com os jogadores com o maior número de pontos no sistema). Deve ter em conta que, num determinado momento, um número alto de jogadores (da ordem dos milhares) poderá ter os mesmos pontos. Neste caso, os jogadores com os mesmos pontos devem ser listados ordenadamente (ascendentemente) pelo seu login.

Op5. Listar todos os jogadores com uma determinada estrela, recebendo a estrela. Os jogadores a listar deverão estar ordenados ascendentemente pelo login.

O sistema deverá ser pensado tendo em conta que podem existir **milhões de jogadores** que podem adicionar pontos múltiplas vezes ao longo do dia. As listagens pedidas devem estar sempre preparadas para serem geradas, quando pedidas.

Com base nesta especificação, reflita sobre a melhor implementação para a resolução do problema e responda (**separadamente**) às questões que se seguem. Nas perguntas de escolha múltipla deve selecionar apenas uma opção como resposta. **Selecione a sua resposta com um círculo à volta da opção escolhida. Se selecionar mais de uma opção, a sua resposta não será tida em conta**.
As perguntas estão encadeadas. Assim, a resposta a algumas questões pode depender de respostas dadas anteriormente.

1. Que Tipo Abstrato de Dados (TAD) individual **de domínio** propõe para o sistema:
   a. Jogador (*Player*), contendo pelo menos login, nome e idade;
   b. Jogo (*Game*), contendo pelo menos id do Jogo e data;
   c. Equipa (*Team*), contendo id da Equipa e lista de jogadores.

Na sequência da sua escolha, liste abaixo variáveis de instância adicionais (e respetivos tipos) que o TAD escolhido necessite.

Points
Logs in
Name
nickname

2. Proponha uma **Estrutura de Dados** (ED) para apoiar a implementação de ***Op1, Op2 e Op3***. Deve escolher de entre as seguintes opções:
   a. ~~`DoublyLinkedList<Player>`~~ que implementa o interface `List`
   b. ~~`OrderedDoubleList<String, Player>`~~, que implementa o interface **`OrderedDictionary`**
   c. ~~`StackInList<Game>`~~ que implementa o interface **`Stack`**
   d. ~~`StackInArray<Team>`~~ que implementa o interface **`Stack`**
   e. **`BinarySearchTree<String, Game>`** que implementa o interface **`OrderedDictionary`**
   f. ~~`SepChainHashTable<String, Player>`~~ que implementa o interface **`Dictionary`**

Sinteticamente, justifique a sua escolha. Caso tenha escolhido qualquer tipo de dicionário, indique qual a chave (tipo **`String`**) associada ao mesmo.

Rogin

3. Proponha uma **Estrutura de Dados** (ED) para apoiar a implementação de ***Op4***. Deve escolher de entre as seguintes opções:
   a. ~~`SepChainHashTable<Long, Player>`~~ que implementa o interface **`Dictionary`**
   b. ~~`OrderedDoubleList<Long, OrderedDoubleList<String, Player>>`~~ com as duas estruturas a implementar o interface **`OrderedDictionary`**
   c. ~~`BinarySearchTree<Long, OrderedDoubleList<String, Player>>`~~ com as duas estruturas a implementar o interface **`OrderedDictionary`**
   d. **`AVLTree<Long, AVLTree<String, Player>>`** com as duas estruturas a implementar o interface **`OrderedDictionary`**
   e. ~~`SepChainHashTable<Long, OrderedDoubleList<String, Player>>`~~ com as duas estruturas a implementar o interface **`Dictionary`**

Sinteticamente justifique a sua escolha. Caso tenha escolhido qualquer tipo de dicionário (que pode ter um dicionário interno associado) indique quais as chaves associadas aos tipos escolhidos.

4. Proponha uma **Estrutura de Dados** (ED) para apoiar a implementação de ***Op5***. Deve escolher de entre as seguintes opções:
   a. **`SepChainHashTable<Integer, OrderedDoubleList<String, Player>>`** com as duas estruturas a implementar o interface **`Dictionary`**
   b. **`SepChainHashTable<Integer, Player>`** que implementa o interface **`Dictionary`**
   c. Um vetor de 5 posições (índices 0 a 4) onde, em cada posição do vetor, estará uma **`AVLTree<String, Player>`** que implementa o interface **`OrderedDictionary`**
   d. **`SepChainHashTable<Integer, AVLTree<String, Player>>`** com as duas estruturas a implementar o interface **`Dictionary`**
   e. **`OrderedDoubleList<Integer, OrderedDoubleList<String, Player>>`** com as duas estruturas a implementar o interface **`OrderedDictionary`**
   f. **`BinarySearchTree<String, OrderedDoubleList<Integer, Player>>`** com as duas estruturas a implementar o interface **`OrderedDictionary`**

Sinteticamente justifique a sua escolha. Caso tenha escolhido qualquer tipo de dicionário (que pode ter um dicionário interno associado) indique quais as chaves associadas aos tipos escolhidos.

5. Considerando as respostas anteriores, descreva brevemente (por palavras suas, ou em pseudo-código) como implementaria ***Op1***. Estude ainda a **complexidade temporal da operação**, no caso esperado, **justificando**.

6. Considerando as respostas anteriores, Estude ainda a **complexidade temporal de *Op3***, no melhor caso, no pior caso e no caso esperado, **justificando.**

7. Considerando as respostas anteriores, Estude ainda a **complexidade temporal de *Op4***, no melhor caso, no pior caso e no caso esperado, **justificando**.

Pode usar este espaço para rascunho

Número: **Sala:**

Nome:

Grupo III – Programação de Estruturas de Dados

1. Considere a implementação de árvore binária de pesquisa estudada em AED `BinarySearchTree<K,V>`. Implemente o iterador **prefixo** desta árvore. Este iterador devolve, em cada chamada do método `next()`, as entradas guardadas na árvore, pela ordem do percurso prefixo (ou *preorder*) estudado nas aulas. Deve ter em conta que o **iterador deve ser implementado apenas usando código iterativo**. Parte do código é fornecido, pelo que deve completar o iterador, onde necessário.
Exemplos:
- O percurso prefixo da árvore **da Fig. 2a)** iria visitar os nós pela seguinte ordem das chaves: 16, 7, 5, 8, 18.
- O percurso prefixo da árvore **da Fig. 2b)** iria visitar os nós pela seguinte ordem das chaves: 8, 1, 5, 6.

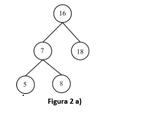


Figura 2 a)

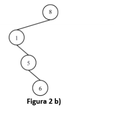


Figura 2 b)

(Responda à pergunta na página seguinte)

```
public class BSTPreorderIterator<K,V> implements Iterator<Entry<K,V>> {

    // raiz da árvore
    protected BSTNode<K,V> root;

    // Estrutura de dados de apoio à iteração
    protected Stack<BSTNode<K,V>> st;
```

```
    //Contrutor do iterador
    BSTPreorderIterator(BSTNode<K,V> root){
        this.root = root;
        st = new StackInList();
        if(root!=null){
            st.push(root.getEntry());
        }
    }
```



```
    //método que inicializa (ou re-inicializa) a iteração
    public void rewind( ){
        if(st.isEmpty())
            st.push(root.getEntry());
    }

    //devolve true se a iteração ainda não terminou
    public boolean hasNext(){
        return !st.isEmpty();
    }

    //devolve a próxima entrada na iteração e prepara a seguinte, caso exista
    public Entry<K,V> next( ) throws NoSuchElementException{
        BSTNode<K,V> no = st.pop().value();
        if(no.getRightChild()!=null){
            st.push(no.getRight());
        }
        if(no.getLeftChild()!=null){
            st.push(no.getLeft());
        }
        return no;
    }
}
```

Pode usar este espaço para rascunho

Anexo A - Recorrências

Recorrência 1

$$T(n) = \begin{cases} a & n = 0 \quad n = 1 \\ \text{ou} \\ bT(n-1) + c & n \geq 1 \quad n \geq 2 \end{cases}$$

$$T(n) = \begin{cases} O(n) & b = 1 \\ O(b^n) & b > 1 \end{cases}$$

com $a \geq 0, \quad b \geq 1, \quad c \geq 1$ **constantes**

Recorrência 2a)

$$T(n) = \begin{cases} a & n = 0 \quad n = 1 \\ \text{ou} \\ bT(\frac{n}{2}) + O(1) & n \geq 1 \quad n \geq 2 \end{cases}$$

$$T(n) = \begin{cases} O(\log n) & b = 1 \\ O(n) & b = 2 \end{cases}$$

com $a \geq 0, \quad b = 1, 2$ **constantes**

Recorrência 2b)

$$T(n) = \begin{cases} a & n = 0 \quad n = 1 \\ \text{ou} \\ bT(\frac{n}{2}) + O(n) & n \geq 1 \quad n \geq 2 \end{cases}$$

com $a \geq 0, \quad b \geq 1, \quad c > 1$ **constantes**

$$T(n) = \begin{cases} O(n) & b < c \\ O(n \log_c n) & b = c \\ O(n^{\log_c b}) & b > c \end{cases}$$

Anexo B – Interfaces e Classes de Apoio

```
public interface ComparableType{
    int compareTo( T object );
}
public interface Stack<T>{
    boolean isEmpty( );
    int size( );
    T top( ) throws EmptyStackException;
    void push( T element );
    T pop( ) throws EmptyStackException;
}
public interface Queue<T> {
    boolean isEmpty( );
    int size( );
    void enqueue( T element );
    T dequeue( ) throws EmptyQueueException;
}
public interface Iterator<E> {
    boolean hasNext( );
    E next( ) throws NoSuchElementException;
    void rewind( );
}
public interface TwoWayIterator<E>
  extends Iterator<E>{
    boolean hasPrevious( );
    E previous( ) throws NoSuchElementException;
    void fullForward( );
}
public interface List<E>
    boolean isEmpty( );
    int size( );
    Iterator<E> iterator( );
    E getFirst( ) throws EmptyListException;
    E getLast( ) throws EmptyListException;
    E get( int position ) throws
                              InvalidPositionException;
    int find( E element );
    void addFirst( E element );
    void addLast( E element );
    void add( int position, E element )
                              throws InvalidPositionException;
    E removeFirst( ) throws EmptyListException;
    E removeLast( ) throws EmptyListException;
    E remove( int position ) throws
                              InvalidPositionException;
    boolean remove( E element );
}
public interface Entry<K,V>{
    K getKey( );
    V getValue( );
}
public interface Dictionary<K,V>{
    boolean isEmpty( );
    int size( );
    Iterator<Entry<K,V>> iterator( );
    V find( K key );
    V insert( K key, V value );
    V remove( K key );
}

public interface
OrderedDictionary<K extends Comparable<K>, V>
                                  extends Dictionary<K,V>{
    Entry<K,V> minEntry( ) throws
                              EmptyDictionaryException;
    Entry<K,V> maxEntry( ) throws
                              EmptyDictionaryException;
}
```

```
class DoubleListNode<E> {
    public DoubleListNode( E theElement,
                           DoubleListNode<E> thePrevious,
                           DoubleListNode<E> theNext );
    public DoubleListNode( E theElement );
    public E getElement( );
    public DoubleListNode<E> getPrevious( );
    public DoubleListNode<E> getNext( );
    public void setElement( E newElement );
    public void setPrevious(
                    ( DoubleListNode<E> newPrevious );
    public void setNext
                    ( DoubleListNode<E> newNext );
}
public class DoubleList<E> implements List<E> {
    public boolean isEmpty( );
    public int size( );
    public Iterator<E> iterator( );
    public E getFirst( ) throws
                              EmptyListException;
    public E getLast( ) throws EmptyListException;
    protected DoubleListNode<E> getNode
                              ( int position )
    public E get( int position ) throws
                              InvalidPositionException;
    public int find( E element );
    public void addFirst( E element );
    public void addLast( E element );
    protected void addMiddle
                              ( int position, E element );
    public void add( int position, E element )
                              throws InvalidPositionException;
    protected void removeFirstNode( );
    public E removeFirst( ) throws
                              EmptyListException;
    protected void removeLastNode( );
    public E removeLast( ) throws
                              EmptyListException;
    protected void removeMiddle
                              ( DoubleListNode<E> node );
    public E remove( int position )
                              throws InvalidPositionException;
    protected DoubleListNode<E> findNode
                              ( E element );
    public boolean remove( E element );
    public void append( DoubleList<E> list )
}
class BSTNode<K,V>{
    public BSTNode( K key, V value,
                    BSTNode<K,V> left, BSTNode<K,V> right );
    public BSTNode( K key, V value );
    public Entry<K,V> getEntry( );
    public K getKey( );
    public V getValue( );
    public BSTNode<K,V> getLeft( );
    public BSTNode<K,V> getRight( );
    public void setEntry
                    ( Entry<K,V> newEntry );
    public void setKey( K newKey );
    public void setValue( V newValue );
    public void setLeft( BSTNode<K,V> newLeft );
    public void setRight( BSTNode<K,V> newRight );
    public boolean isLeaf( );
}
```