Progetto del corso di Strutture Dati a.a. 2009-10

Il presente documento contiene l'elenco delle classi che devono essere incluse nel progetto del corso di Strutture Dati.

STACK:

- La classe ArrayStack che implementa Stack mediante un array. La classe deve essere implementata in modo tale che, quando non risulta possibile effettuare ulteriori inserimenti, l'array venga sostituito con uno più grande.
- La classe NodeStack che implementa Stack mediante una lista a puntatori singoli.

QUEUE:

- La classe ArrayQueue che implementa Queue mediante un array usato in modo circolare. La classe deve essere implementata in modo tale che, quando non risulta possibile effettuare ulteriori inserimenti, l'array venga sostituito con uno più grande.

DEQUE:

- La classe NodeDeque che implementa Deque con una lista a doppi puntatori.

NODE LIST:

- La classe NodePositionList che implementa PositionList mediante una lista a doppi puntatori.

ARRAY LIST:

- La classe ArrayIndexList che implementa IndexList mediante un array. . La classe deve essere implementata in modo tale che, quando non risulta possibile effettuare ulteriori inserimenti, l'array venga sostituito con uno grande il **doppio**.

SEQUENCE

- La classe NodeSequence che implementa Sequence mediante una lista a doppi puntatori;
- La classe ArraySequence che implementa Sequence mediante un array.

ITERATOR

- La classe LinkedIterator che implementa Iterator mediante una lista a doppi puntatori;
- La classe ElementIterator che implementa Iterator per il tipo PositionList
  - L' implementazione di ElementIterator deve utilizzare un cursore.

TREE

- La classe LinkedTree che implementa Tree con una struttura a puntatori in cui ciascun nodo contiene oltre all'elemento, un riferimento al padre e un riferimento alla collezione dei figli.

  Oltre ai metodi dell'interfaccia Tree, la classe LinkedTree deve contenere i seguenti metodi di modifica:

  - Position<E> addRoot(E **elt**): se l'albero è vuoto, crea e restituisce in output un nuovo nodo contenente **elt** e fa diventare questo nuovo nodo radice dell'albero; se l'albero non è vuoto lancia l'eccezione NonEmptyTreeException;

  - Position <E> addChild(E **elt**,Position <E> **v**) : crea e restituisce in output una foglia contenente **elt** e fa diventare questo nuovo nodo figlio di **v** aggiungendolo alla fine della lista dei figli di **v**;

  - E removeExternalChild(Position<E> **v**): se il primo figlio di **v** è una foglia allora lo cancella restituendo in output il suo elemento; altrimenti lancia l'eccezione UndeletableNodeException.

BINARY TREE

- La classe EulerTour contenente il template method EulerTour;
- Una sottoclasse (a scelta dello studente) che specializza EulerTour.

- La classe LinkedBinaryTree che implementa BinaryTree con una struttura a puntatori in cui ciascun nodo contiene oltre all'elemento, un riferimento al padre, un riferimento al figlio sinistro e un riferimento al figlio destro.

  Oltre ai metodi dell'interfaccia BinaryTree, la classe LinkedBinaryTree deve contenere i seguenti metodi di modifica:

  - Position<E> addRoot(E **elt**): se l'albero è vuoto, crea e restituisce in output un nuovo nodo contenente **elt** e fa diventare questo nuovo nodo radice dell'albero; se l'albero non è vuoto lancia l'eccezione NonEmptyTreeException;

  - Position <E> insertLeft(E **elt**,Position <E> **v**) : se **v** non ha un figlio sinistro, crea e restituisce in output una foglia contenente **elt** e fa diventare questo nuovo nodo figlio sinistro di **v**; altrimenti lancia l'eccezione InvalidPositionException;

  - Position <E> insertRight(E **elt**,Position <E> **v**) : se **v** non ha un figlio sinistro, crea e restituisce in output una foglia contenente **elt** e fa diventare questo nuovo nodo figlio destro di **v**; altrimenti lancia l'eccezione InvalidPositionException;

PRIORITY QUEUE

- La classe UnsortedListPriorityQueue che implementa PriorityQueue mediante un'istanza di PositionList in cui le entrate compaiono in un ordine arbitrario.
- La classe SortedListPriorityQueue che implementa PriorityQueue mediante un'istanza di PositionList in cui le entrate sono ordinate in base ai valori delle chiavi
- La classe HeapPriorityQueue che implementa PriorityQueue mediante un heap.

ADAPTABLE PRIORITY QUEUE

- La classe HeapAdaptablePriorityQueue che implementa AdaptablePriorityQueue mediante un heap.

COMPARATOR

- La classe DefaulComparator che usa il metodo compareTo di java.lang.Comparable per effettuare i confronti
- Un comparatore per confrontare oggetti di un tipo a vostra scelta.

COMPLETE BINARY TREE

- La classe ArrayListCompleteBinaryTree che implementa CompleteBinaryTree con un'istanza di IndexList.

MAP

- La classe ListMap che implementa Map mediante un'istanza di PositionList.
- La classe HashTableMap che implementa Map mediante una tabella hash in cui i conflitti sono risolti con il metodo del linear probing. Il load factor deve essere mantenuto al di sotto di una certa soglia (scelta in modo appropriato).

DICTIONARY

- La classe LogFile che implementa Dictionary mediante un'istanza di PositionList in cui le entrate compaiono in un ordine arbitrario
- La classe ChainingHashTable che implementa Dictionary mediante una tabella hash in cui i conflitti sono risolti con il metodo del chaining
- La classe LinearProbingHashTable che implementa Dictionary mediante una tabella hash in cui i conflitti sono risolti con il metodo del linear probing. Il load factor deve essere mantenuto al di sotto di una certa soglia (scelta in modo appropriato).
- La classe BinarySearchTree che implementa Dictionary mediante un albero di ricerca binario.

SET

- La classe OrderedListSet che implementa Set mediante un'istanza di PositionList che contiene gli elementi dell'insieme ordinati secondo una certa relazione d'ordine totale. I metodi union, intersect e subtract devono far uso del template method merge.

PARTITION

- La classe ListPartition che implementa Partition mediante un'istanza di PositionList che contiene gli insiemi della partizione. Il metodo find deve avere tempo di esecuzione O(1) (almeno nel caso medio).

GRAPH

- La classe AdjacencyListGraph che implementa Graph mediante liste di adiacenza. Le interfacce Vertex ed Edge devono specializzare l'interfaccia DecorablePosition.
- Le classi DFS, BFS, Kruskal, Dijkstra, ConnectivityDFS, ComponentsDFS, FindPathDFS, CreateDFSTree e LayerBFS. Per chiarimenti su queste ultime due classi, si vedano gli esercizi sui grafi pubblicati sul sito.

NB: Le interfacce PositionList, Sequence, Tree, BinaryTree, CompleteBinaryTree devono essere tipi Iterable e fornire il metodo positions.