Attività di didattica integrativa per Fondamenti di Informatica

Esercitazioni sulle prove di esame

AA 2021-22

Sim1 Dictionary

giulio.martini@igi.cnr.it

TEMA DI ESAME:
Rubrica telefonica

Esempio di Dizionario - La Rubrica telefonica

Si vuole gestire una rubrica telefonica realizzando un dizionario che contiene coppie di tipo “nome numero”. Il campo “nome” è una stringa che ha la funzione di chiave per il dizionario stesso (dunque la rubrica non potrà contenere omonimi), mentre il campo “numero” è un numero intero in formato long che rappresenta i recapiti telefonici associati a ciascun nome.

Materiale fornito:

- Un’interfaccia Dictionary
- Un’eccezione DictionaryItemNotFoundException
- Classe Rubrica con all’interno la classe **Pair**
- Classe RubricaTester
- File di prova

Dictionary Interface

```
interface Dictionary
{
    /*
        verifica se il dizionario contiene almeno una coppia chiave/valore
    */
    boolean isEmpty();

    /*
        svuota il dizionario
    */
    void makeEmpty();

    /*
     Inserisce un elemento nel dizionario. L'inserimento va sempre a buon fine.
     Se la chiave non esiste la coppia key/value viene aggiunta al dizionario;
     se la chiave esiste gia' il valore ad essa associato viene sovrascritto
     con il nuovo valore; se key e` null viene lanciata IllegalArgumentException
    */
    void insert(Comparable key, Object value);

    /*
     Rimuove dal dizionario l'elemento specificato dalla chiave key
     Se la chiave non esiste viene lanciata DictionaryItemNotFoundException
    */
    void remove(Comparable key);

    /*
     Cerca nel dizionario l'elemento specificato dalla chiave key
     La ricerca per chiave restituisce soltanto il valore ad essa associato
     Se la chiave non esiste viene lanciata DictionaryItemNotFoundException
    */
    Object find(Comparable key);
}
```

Implementazione Dictionary

La rubrica conterrà coppie di tipo “nome numero” appartenenti alla classe **Pair**, realizzata come classe interna alla classe Rubrica e il cui codice non va modificato.

```
//classe privata Pair: non modificare!!
private class Pair
{
    //campi di esemplare
    private String name;
    private long phone;

    // COSTRUTTORE
    public Pair(String aName, long aPhone) {
        name= aName;
        phone = aPhone;
    }

    // NAME
    public String getName()  {   return name; }

    // PHONE
    public long getPhone()   {   return phone; }

    // toString
    public String toString() {
        return name + " : " + phone;
    }

}
```

Richieste:

STEP 1

Implementare la classe Rubrica scrivendone la parte privata e realizzandone i metodi pubblici.

- Si richiede che il metodo find() abbia prestazioni O(log n).
- Si richiede inoltre di realizzare un metodo toString per la classe Rubrica, che restituisca una stringa contenente gli elementi della rubrica secondo il seguente formato:
  a. ogni coppia “nome numero” viene scritta su una riga diversa
  b. all’interno di ogni riga la coppia viene scritta seguendo il formato specificato dal metodo toString della classe Pair (si veda il corpo di tale metodo).

STEP 2

Implementare la classe Rubrica Tester che contiene il metodo main() tale che:

- Due nomi di file di testo, file1 e file2, vengono passati come argomenti sulla riga di comando.
- Vengono creati due oggetti di tipo Rubrica. Nella prima rubrica si inseriscono elementi letti dal file file1 (scritto nello stesso formato specificato sopra per il metodo toString di Rubrica).
- Viene richiesta l'immissione di un nome da input standard. Il nome immesso viene cercato nella prima rubrica e la corrispondente coppia “nome numero” viene (se trovata) spostata dalla prima alla seconda rubrica (ovvero rimossa dalla prima e inserita nella seconda rubrica).
- L’operazione descritta al punto precedente può essere ripetuta un numero non prefissato di volte. Il ciclo viene terminato tramite inserimento del carattere “Q” da input standard.
- Al termine delle ricerche effettuate dall’utente, il contenuto della seconda rubrica viene stampato sul file file2 (nello stesso formato specificato sopra per il metodo toString di Rubrica).

STEP 1: Rubrica

Soluzione: Array associativo

Supponiamo di implementare il dizionario attraverso un Array di coppie (chiave - valore).

- Il testo richiede esplicitamente che la complessità del metodo **find** sia **O(log n)**.

  IN QUESTO CASE QUALE ALGORITMO DI RICERCA UTILIZZIAMO?

  LA RICERCA BINARIA
  (BINARY SEARCH)

- Quindi bisogna saper realizzare almeno un algoritmo di **ordinamento** sull'array, ed utilizzarlo all'interno del metodo **insert** in modo da mantenere l'ordinamento ad ogni inserimento

  Dovendo sempre mantenere l'ordinamento (per poter applicare il binary search) …

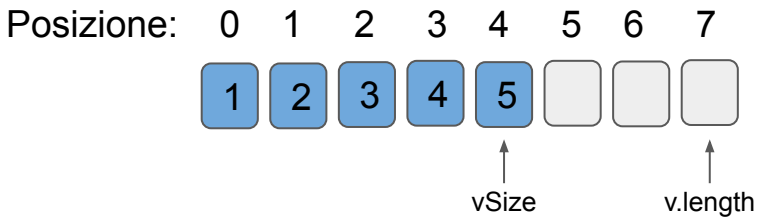
  QUALE ALGORITMO UTILIZZIAMO ?

ORDINAMENTO PER INSERZIONE
(INSERTION SORT)

InsertionSort è la scelta migliore dal punto di vista delle prestazioni, tuttavia sarebbe stato accettabile anche l'uso di selectionSort o mergeSort dal momento che il testo non fa richieste sulle prestazioni di insert

Soluzione: struttura di base della classe Rubrica

```
class Rubrica implements Dictionary
{
    //campi di esemplare e variabili statiche di Rubrica
    private Pair[] v;
    private int vSize;
    private static final int INITSIZE = 1;

    // COSTRUTTORE
    public Rubrica()
    {
        v = new Pair[INITSIZE];
        makeEmpty();
    }

    // verifica se la rubrica e' vuota
    public boolean isEmpty()
    {   return vSize == 0;  }

    // svuota la rubrica
    public void makeEmpty()
    {   vSize = 0;  }

}
```

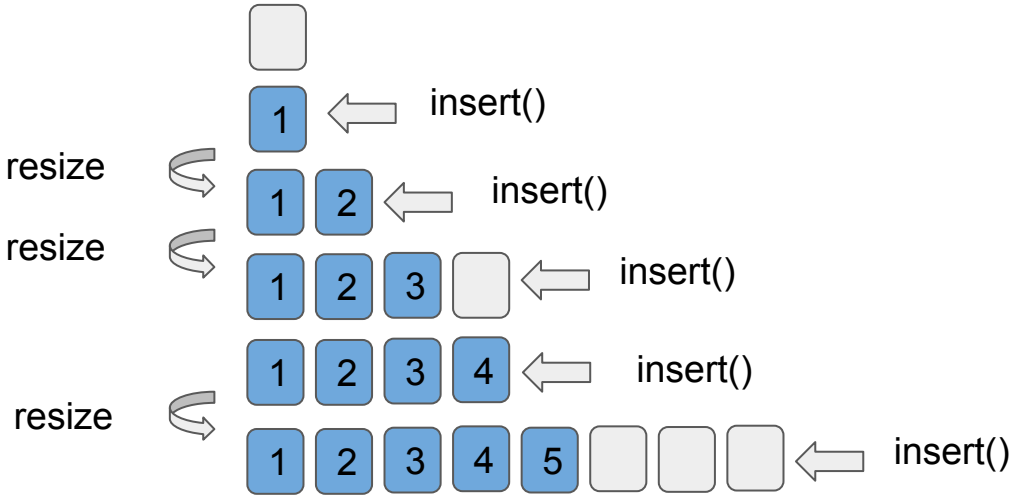


insert()

```
//Bisogna realizzare il comportamento richiesto nell'interfaccia Dictionary:
//in particolare sovrascrivere coppie gia` presenti e lanciare eccezioni
//La realizzazione qui proposta ha prestazioni O(n), perche` abbiamo usato
//un algoritmo di ordinamento per inserimento
public void insert(Comparable key, Object value)
{
    //precondizioni: controllo anche che value sia un numero long
    if (key == null || !(value instanceof Long) )
        throw new IllegalArgumentException();
    try{   remove(key); } //se la coppia c'e` gia` la rimuovo
    catch(DictionaryItemNotFoundException e){   }//altrimenti tutto ok!

    //uso array ridimensionabile!
    if(vSize == v.length) resize();

    //riordinamento per inserimento. Attenzione ai cast: v[i-1].getName()
    //e` di tipo String, e puo` essere comparato solo a String
    int i = vSize; // questo ciclo ha tempi di esecuzione O(n)
    while (i > 0 && (v[i-1].getName()).compareTo((String)key) > 0)
    {   v[i] = v[i-1];
        i--;
    }

    //creo un nuovo Pair (attenzione ai cast) e aggiungo al punto giusto
    v[i] = new Pair((String)key, (Long)value);
    vSize++; // aggiorno la dimensione
}
```

resize()

```
//metodo ausiliario: lo rendo private, non deve essere usato da altri
private void resize() //niente parametri espliciti e valori restituiti: ho
{                     // deciso che raddoppio sempre la dimensione.
    Pair[] newv = new Pair[2*v.length];
    System.arraycopy(v, 0, newv, 0, v.length);
    v = newv; //funziona: v non e` una var. locale ma un campo di esemplare
}
```



find()

```
public Object find(Comparable key)
{   //uso binSearch per cercare la chiave nell'array non ordinato
    //se la chiave non c'e` lancio ItemNotFoundException come da
    //specifiche (viene lanciata da binSearch)
    return v[binSearch(0, vSize-1, key)].getPhone();
}



//metodo ausiliario: restituisce l'indice in cui ha trovato l'elemento
private int binSearch(int from, int to, Comparable key)
{   if (from > to) throw new ItemNotFoundException();
    int mid = (from + to) / 2; // circa in mezzo
    Comparable middlekey = v[mid].getName();
    if (middlekey.compareTo(key) == 0)
    //In questo caso funzionerebbe anche if (middle.getKey().equals(key))
    //perche` le chiavi sono di tipo String, e il metodo equals e` stato
    //sovrascritto in String in modo da essere coerente con compareTo
        return mid; // elemento trovato
    else if (middlekey.compareTo(key) < 0)  //cerca a destra
        return binSearch(mid + 1, to, key);
    else // cerca a sinistra
        return binSearch(from, mid - 1, key);
}
```

remove()

```
public void remove(Comparable key)
{   //uso binSearch per cercare la chiave nell'array non ordinato
    //se la chiave non c'e` lancio ItemNotFoundException come da
    //specifiche (viene lanciata da binSearch)
    int i = binSearch(0, vSize-1, key);
    for (int j = i; j < vSize-1; j++)
        v[j] = v[j+1];
    vSize--;
}
```

```
public String toString()
{   String s = "";
    for (int i = 0; i < vSize; i++)
        s = s + v[i] + "\n"; //sfrutto il metodo toString di Pair!
    return s;
}
```

```
name : phone

// Pair toString
public String toString() {
    return name + " : " + phone;
}
```

STEP 2: RubricaTester

RubricaTester

- Due nomi di file di testo, file1 e file2, vengono passati come argomenti sulla riga di comando.

```
//controllo parametri del metodo main
if (args.length != 2 || args[0].equals(args[1])) {
  System.out.println("Non usare stesso nome file in lettura/scrittura");
  System.exit(1);
}
String filename1 = args[0];
String filename2 = args[1];
```

- Il *file1* e' usato per leggere i dati di ingresso, mentre il *file2* verrà utilizzato per la scrittura.

```
//apertura di file1 in lettura
Scanner file1 = null;
try{  file1 = new Scanner(new FileReader(filename1));  }
catch(FileNotFoundException e)
{   System.out.println("Problema in apertura File1! Termino");
    System.exit(1); }
```

Inserimento da file1

- Viene creato un primo oggetto di tipo Rubrica. Vi si inseriscono elementi letti dal file *file1* (scritto nello stesso formato specificato sopra per il metodo toString di Rubrica).

```
//Creazione e  scrittura di rubrica1
Rubrica rubrica1 = new Rubrica();
while (file1.hasNextLine())
{   String line = file1.nextLine();
    Scanner linescan = new Scanner(line);
    try{
    String name = linescan.next(); //nome : campo "key" del dizionario
    String word;
    while (!( word = linescan.next()).equals(":"))//considero anche
        name += " " + word; //nomi eventualmente composti da piu` parole
    long phone = Long.parseLong(linescan.next()); //dopo i ":" ci deve
                                    //essere un numero in formato long
    rubrica1.insert(name, phone);
    } //NoSuchElementException puo` essere lanciata da next se non vengono trovati token
    catch(NoSuchElementException e)
    {   System.out.println("Formato inserimento sbagliato");  }
     //NumberFormatException puo` essere lanciata da parseLong se la
     // stringa dopo i ":" non e` un intero
    catch(NumberFormatException e)
    {   System.out.println("Formato inserimento sbagliato");  }
}
System.out.println(rubrica1); //controllo il contenuto di rubrica1
file1.close();
```

Inserimento da file1

- Viene creato un primo oggetto di tipo Rubrica. Vi si inseriscono elementi letti dal file *file1* (scritto nello stesso formato specificato sopra per il metodo toString di Rubrica).

NOTA:

Bisogna interpretare correttamente le righe, usando ":" come carattere di separazione tra la chiave (nome) e il valore (numero telefonico) della coppia

La soluzione deve consentire di interpretare correttamente righe in cui i nomi sono composti da più parole, ad esempio: **Zio Paperone : 4683457923**

Nella fase di scansione delle righe del file devono essere gestite le eccezioni **NoSuchElementException** e **NumberFormatException**

Spostamento da Rubrica1 a Rubrica2

- Viene richiesta l'immissione di un nome da input standard. Il nome immesso viene cercato nella prima rubrica e la corrispondente coppia “nome numero” viene (se trovata) spostata dalla prima alla seconda rubrica (ovvero rimossa dalla prima e inserita nella seconda rubrica). L’operazione descritta al punto precedente può essere ripetuta un numero non prefissato di volte. Il ciclo viene terminato tramite inserimento del carattere “Q” da input standard.

```
//Creazione di rubrica2, ricerca e rimozione dati da rubrica1
//inserimento in rubrica2 di dati rimossi da rubrica1
Scanner in = new Scanner(System.in); //apertura standard input
Rubrica rubrica2 = new Rubrica();
boolean done = false;
while(!done)
{   System.out.println("Nome in rubrica1 da spostare in rubrica2?");
    System.out.println("(\"Q\" per terminare)");
    String name = in.nextLine();
    if (name.equalsIgnoreCase("Q"))
        done = true;
    else
    {   try{rubrica2.insert(name, rubrica1.find(name));//prima copio e
            rubrica1.remove(name);} //poi cancello. Non viceversa!
        catch(DictionaryItemNotFoundException e)
        { System.out.println("Nome non presente in rubrica1" ); }
        System.out.println("Rubrica1:\n" + rubrica1); //controllo i
        System.out.println("Rubrica2:\n" + rubrica2); //contenuti
    }
}
```

Salvataggio Rubrica2 su file2

- Al termine delle ricerche effettuate dall'utente, il contenuto della seconda rubrica viene stampato sul file file2 (nello stesso formato specificato sopra per il metodo toString di Rubrica).

```
//apertura di file2 in scrittura, salvataggio di rubrica2 in file2
PrintWriter file2 = null;
try{  file2 = new PrintWriter(filename2);  }
catch(FileNotFoundException e)
{   System.out.println("Problema in apertura File2! Termino");
    System.exit(1); }
file2.print(rubrica2);
file2.close();  //Se non chiudo rischio che file2 non venga scritto!!!
System.out.println("Arrivederci");
```

Caveat

- **Lettura/scrittura di file**
  - Ricordiamoci che le **eccezioni di tipo IO** sono a gestione obbligatoria

    Try/catch per catturare le eccezioni lanciabili dai costruttori di FileReader e PrintWriter

    Se non ci si sente sicuri su questo aspetto, si può evitare di gestire queste eccezioni dichiarando che il metodo main le lancia: public static void main(String[] args) throws IOException
  - Ricordarsi di "chiudere" i file, usando i metodi **close**
    Se in particolare ci dimentichiamo di chiudere File2, aperto in scrittura, il file rischia di non venire scritto!
- **Ricerca/inserimento/cancellazione**
  - Attenzione allo spostamento di una coppia dalla prima rubrica alla seconda: prima si scrive la coppia nella seconda rubrica, poi la si cancella dalla prima. Se facessimo il contrario la coppia andrebbe perduta (garbage collected) !
  - L'eventualità che un nome cercato non sia presente nella prima rubrica viene gestita catturando **ItemNotFoundException**, che può essere lanciata da find e da remove