Esame di Programmazione II, 16 febbraio 2024

(si consegni `PrefixMap.java`)

Si crei un progetto Eclipse e il package ***`it.univr.prefix`****. Si copino al suo interno le classi del compito. Non si modifichino le dichiarazioni dei metodi e delle classi. Si possono definire altri campi, metodi o costruttori non richiesti dal compito, ma devono essere* ***`private`****. Si possono definire altre classi, che in tal caso vanno consegnate. La soluzione che verrà consegnata dovrà compilare, altrimenti non verrà corretta.*

Un *albero di prefissi* è un'implementazione di una mappa da stringhe a valori (possibilmente `null`) di tipo `E`. Il seguente `Main` crea un albero di prefissi per valori di tipo `Integer`, lo popola con dei legami chiave/valore (con il metodo `put`) che poi rilegge (con il metodo `get`):

```
public class Main {
  public static void main(String[] args) {
    PrefixMap<Integer> pm = new PrefixMap<Integer>();
    pm.put("hello", 13);         pm.put("computer", 17);     pm.put("computee", 19);
    pm.put("courage", 41);       pm.put("alliance", 17);     pm.put("help", 78);
    pm.put("computed", 91);      pm.put("courage", 42);      pm.put("", 11);
    pm.put("alliances", null);
    System.out.println("computer -> " + pm.get("computer"));
    System.out.println("computed -> " + pm.get("computed"));
    System.out.println("computee -> " + pm.get("computee"));
    System.out.println("hello -> " + pm.get("hello"));
    System.out.println("help -> " + pm.get("help"));
    System.out.println("hel -> " + pm.get("hel")); // null perche' non c'e'
    System.out.println("helpo -> " + pm.get("helpo")); // null perche' non c'e'
    System.out.println(" -> " + pm.get(""));
    System.out.println("alliance -> " + pm.get("alliance"));
    System.out.println("alliances -> " + pm.get("alliances")); // null perche' non c'e'
    System.out.println("courage -> " + pm.get("courage"));
    System.out.println("computers -> " + pm.get("computers")); // null perche' non c'e'
    System.out.println(pm); // stampa la struttura dell'albero
    pm.put(null, 113); // va in eccezione
  }
}
```

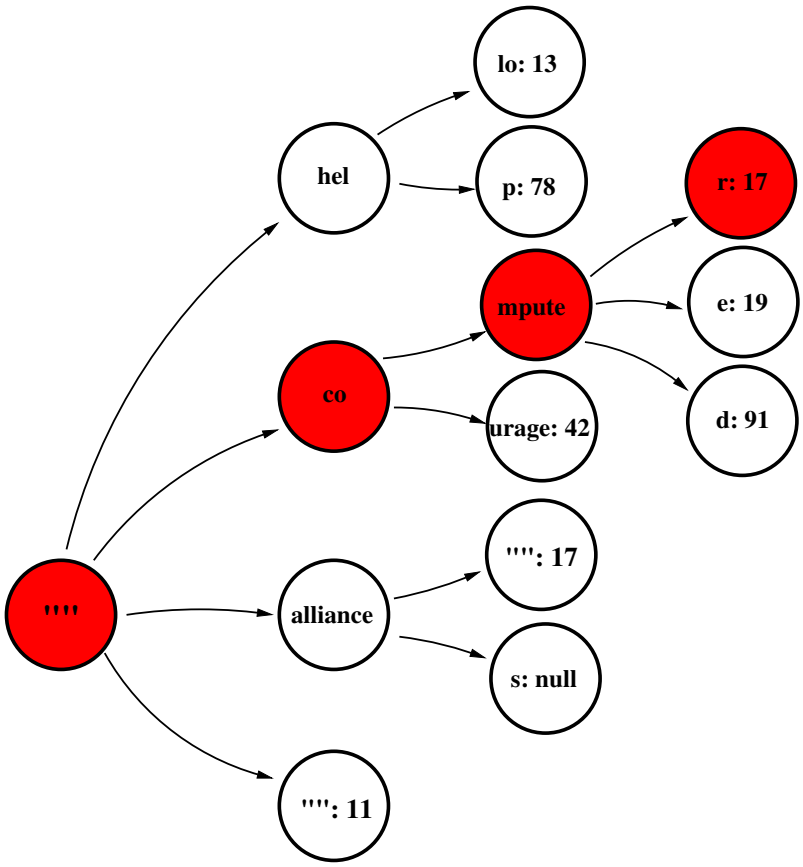
La sua esecuzione dovrà stampare:

```
computer -> 17
computed -> 91
computee -> 19
hello -> 13
help -> 78
hel -> null
helpo -> null
 -> 11
alliance -> 17
alliances -> null
courage -> 42
computers -> null
```

```
        "lo": 13
    "hel"
        "p": 78
            "r": 17
        "mpute"
            "e": 19
            "d": 91
    "co"
        "urage": 42
""
        "": 17
    "alliance"
        "s": null
    "": 11

Exception in thread "main" java.lang.NullPointerException: null keys are not allowed
```

La stampa termina con la struttura interna dell'albero di prefissi, che graficamente possiamo disegnare meglio come segue (la radice è a sinistra e le foglie sono a destra; in questo caso la radice contiene il prefisso stringa vuoto, ma non è detto che sia sempre cosí):



Per ogni cammino dalla radice a una foglia, concatenando i prefissi sul cammino, si ottiene una chiave legata al valore nella foglia. Per esempio, abbiamo evidenziato in rosso un percorso che mostra come la stringa `computer` sia associata al valore 17. Si noti che i figli di un nodo hanno sempre prefissi mutuamente esclusivi. Per esempio, i figli del nodo contenente `co` hanno prefissi `mpute` e `urage`, che non iniziano con la stessa lettera. Questo semplifica la ricerca del valore legato

a una chiave stringa, perché c'è solo un percorso in cui ci si può muovere durante la ricerca di una chiave dalla radice verso le foglie.

Esercizio 1 (2 punti). Il metodo `put` di `PrefixMap` è già scritto e funzionante: lo si completi facendogli generare una `NullPointerException`, con messaggio `null keys are not allowed`, nel caso in cui la chiave `key` fosse `null`.

Esercizio 2 (14 punti). Si completi il metodo `get` di `PrefixMap` aggiungendo un metodo pubblico ricorsivo sui nodi, ridefinito per i nodi interni e i nodi foglia, come è stato già fatto per `put`. Intuitivamente, se si cerca il valore legato a una chiave k, a partire da un nodo n, allora ci sono due casi:

1. n è una foglia: l'unica possibilità è che k sia il prefisso memorizzato dentro n, nel qual caso il valore cercato è quello scritto dentro la foglia n;
2. n è un nodo interno: se k inizia con il prefisso memorizzato dentro n, allora si elimina tale prefisso dall'inizio di k, ottenendo k'. Poi si prova a cercare k' a partire dai figli di n, ricorsivamente. Se una chiamata ricorsiva trova un valore, allora la ricerca ha avuto successo e si ritorna tale valore. Altrimenti si ritorna `null` (la chiave non è stata trovata).

Esercizio 3 (15 punti). Si completi il metodo `toString` di `PrefixMap` in modo da fargli ritornare una stringa che descrive la struttura dell'albero (si veda l'esempio di stampa alla pagina precedente). Occorrerà aggiungere un metodo pubblico ricorsivo sui nodi, ridefinito per i nodi interni e i nodi foglia, come è stato già fatto per `put`. Intuitivamente, se si vuole trasformare in stringa l'albero radicato in un nodo n, allora ci sono due casi:

1. n è una foglia con dentro un prefisso p e un valore v: si ritorna la stringa p **:** v, più un'andata a capo.
2. n è un nodo interno con f figli: si trasformano ricorsivamente in stringhe i primi $\frac{f}{2}$ figli, concatenandole; si concatena il prefisso memorizzato dentro n; si trasformano ricorsivamente in stringhe i restanti $f - \frac{f}{2}$ figli, concatenandole; si ritorna la concatenazione complessiva.

Quanto si concatenano le stampe dei figli, occorre fare un'indentazione di quattro spazi verso destra, come nell'esempio di stampa alla pagina precedente. Il suggerimento è di non preoccuparsi di tale indentazione, in una prima versione, e di aggiungerla successivamente, se il risultato sembra corretto. Si ricorda che è stato fatto in laboratorio un esempio simile di indentazione durante una discesa ricorsiva.