Astrazione sui dati

Tavola Rotonda v1

==> 240326_tavola_rotonda/TavolaRotonda.java <==

```
public class TavolaRotonda {

  private final int quanti;
  private final int brocca;
  private final IntSList altri;

  public TavolaRotonda(int n) {
    this.quanti = n;
    this.brocca = 1;
    this.altri = intervallo(2, n);
  }

  private TavolaRotonda(int q, int b, IntSList a) {
    this.quanti = q;
    this.brocca = b;
    this.altri = a;
  }

  public int quantiCavalieri() {
    return quanti;
  }

  public int chiHaLaBrocca() {
    return brocca;
  }

  public TavolaRotonda serve() {
    return new TavolaRotonda(quanti - 1, brocca, altri.cdr());
  }

  public TavolaRotonda passa() {
    if (altri.isNull()) return this;
    IntSList coda = IntSList.NULL_INTLIST.cons(brocca);
    IntSList nuova = altri.cdr().append(coda);
    return new TavolaRotonda(quanti, altri.car(), nuova);
  }

  private static IntSList intervallo(int inf, int sup) {
    if (inf > sup) return new IntSList();
    return intervallo(inf + 1, sup).cons(inf);
  }
}
```

==> 240326_tavola_rotonda/Test.java <==

```
public class Test {
  public static int ultimoCavaliere(int n) {
    TavolaRotonda tr = new TavolaRotonda(n);
    while (tr.quantiCavalieri() > 1) {
      tr = tr.serve().passa();
    }
    return tr.chiHaLaBrocca();
  }

  public static void main(String[] args) {
    System.out.println(ultimoCavaliere(9));
    System.out.println(ultimoCavaliere(13));
    System.out.println(ultimoCavaliere(23));
  }
}
```

Tavola Rotonda v2

==> 240408_tavola_rotonda/TavolaRotonda.java <==

```
public class TavolaRotonda {
  private final int quanti;
  private final int brocca;
  private final IntSList lista1;
  private final IntSList lista2;

  public TavolaRotonda(int n) {
    this.quanti = n;
    this.brocca = 1;
    this.lista1 = intervallo(2, n);
    this.lista2 = IntSList.NULL_INTLIST;
  }

  private TavolaRotonda(int q, int b, IntSList l1, IntSList l2) {
    this.quanti = q;
    this.brocca = b;
    this.lista1 = l1;
    this.lista2 = l2;
  }

  public int quantiCavalieri() {
    return quanti;
  }

  public int chiHaLaBrocca() {
    return brocca;
  }

  public TavolaRotonda serve() {
    if (lista1.isNull()) {
      IntSList r = lista2.reverse();
      return new TavolaRotonda(quanti - 1, brocca, r.cdr(), IntSList.NULL_INTLIST);
    } else {
      return new TavolaRotonda(quanti - 1, brocca, lista1.cdr(), lista2);
    }
  }

  public TavolaRotonda passa() {
    if (quanti == 1) return this;
    else if (lista1.isNull()) {
      IntSList r = lista2.cons(brocca).reverse();
      return new TavolaRotonda(quanti, r.car(), r.cdr(), IntSList.NULL_INTLIST);
    } else {
      return new TavolaRotonda(quanti, lista1.car(), lista1.cdr(), lista2.cons(brocca));
    }
  }

  private static IntSList intervallo(int inf, int sup) {
    if (inf > sup) return new IntSList();
    return intervallo(inf + 1, sup).cons(inf);
  }
}
```

==> 240408_tavola_rotonda/Test.java <==

```
public class Test {
  public static int ultimoCavaliere(int n) {
    TavolaRotonda tr = new TavolaRotonda(n);
    while (tr.quantiCavalieri() > 1) {
      tr = tr.serve().passa();
    }
    return tr.chiHaLaBrocca();
  }
}
```

Regine

==> 240415_scacchiera_regine/Board.java <==

```java
import java.util.function.*;

/**
 * Board b = new Board();
 *
 * <p>b.size() : int
 *
 * <p>b.queensOn() : int
 *
 * <p>b.underAttack(i, j) : boolean
 *
 * <p>b.addQueen(i, j) : Board
 *
 * <p>b.arrangement() : String
 */
public class Board {

  private static final String ROWS = " 123456789ABCDEF";
  private static final String COLS = " abcdefghijklmno";

  private final int size;
  private final int queens;
  private final BiPredicate<Integer, Integer> attack;
  private final String config;

  public Board(int n) {
    this.size = n;
    this.queens = 0;
    this.attack = (x, y) -> false; // (lambda (x y) false)
    this.config = " ";
  }

  private Board(Board b, int i, int j) {
    this.size = b.size();
    this.queens = b.queensOn() + 1;
    this.attack =
        (x, y) -> x == i || y == j || x - y == i - j || x + y == i + j || b.underAttack(x, y);
    this.config = b.arrangement() + COLS.charAt(j) + ROWS.charAt(i) + " ";
  }

  public int size() {
    return this.size;
  }

  public int queensOn() {
    return this.queens;
  }

  public boolean underAttack(int i, int j) {
    return attack.test(i, j); // (attack i j)
  }

  public Board addQueen(int i, int j) {
    return new Board(this, i, j);
  }

  public String arrangement() {
    return this.config;
  }

  public String toString() {
    return "[" + this.arrangement() + "]";
  }
}
```

==> 240415_scacchiera_regine/Test.java <==

```java
public class Test {
  public static int numSoluzioni(int n) {
    return numCompletamenti(new Board(n));
  }

  public static int numCompletamenti(Board b) {
    int n = b.size();
    int q = b.queensOn();
    if (q == n) {
      return 1;
    } else {
      int i = q + 1;
      int count = 0;
      for (int j = 1; j <= n; j++) {
        if (!b.underAttack(i, j)) {
          count += numCompletamenti(b.addQueen(i, j));
        }
      }
      return count;
    }
  }

  public static BoardSList listaSoluzioni(int n) {
    return listaCompletamenti(new Board(n));
  }

  public static BoardSList listaCompletamenti(Board b) {
    int n = b.size();
    int q = b.queensOn();
    if (q == n) {
      return BoardSList.NULL_BOARDLIST.cons(b);
    } else {
      int i = q + 1;
      BoardSList list = BoardSList.NULL_BOARDLIST;
      for (int j = 1; j <= n; j++) {
        if (!b.underAttack(i, j)) {
          list = list.append(listaCompletamenti(b.addQueen(i, j)));
        }
      }
      return list;
    }
  }

  public static void main(String[] args) {
    int n = Integer.parseInt(args[0]);
    System.out.println(numSoluzioni(n));
    System.out.println(listaSoluzioni(n));
  }
}
```

SList

```java
/**
 * Liste nello stile di Scheme
 *
 * <p>SList() // null
 *
 * <p>s.isNull() // (null? s)
 *
 * <p>s.car() // (car s)
 *
 * <p>s.cdr() // (cdr s)
 *
 * <p>s.cons(i) // (cons i s)
 */
public class SList<T> {
  private final boolean empty;
  private final T first;
  private final SList<T> rest;

  public SList() {
    empty = true;
    first = null;
    rest = null;
  }

  public SList(T e, SList<T> r) {
    empty = false;
    first = e;
    rest = r;
  }

  public boolean isNull() {
    return empty;
  }

  public T car() {
    return first;
  }

  public SList<T> cdr() {
    return rest;
  }

  public SList<T> cons(T e) {
    return new SList<T>(e, this);
  }

  public String toString() {
    if (isNull()) {
      return "()";
    } else {
      String lst = "(" + car();
      SList<T> r = cdr();
      while (!r.isNull()) {
        lst += " " + r.car();
        r = r.cdr();
      }
      return lst + ")";
    }
  }

  public int length() {
    if (isNull()) return 0;
    return 1 + cdr().length();
  }
```

```
  public T listRef(int i) {
    if (i == 0) return car();
    return cdr().listRef(i - 1);
  }

  public boolean equals(SList<T> s) {
    if (isNull()) {
      return s.isNull();
    } else if (s.isNull()) {
      return false;
    } else if (car().equals(s.car())) {
      return cdr().equals(s.cdr());
    } else {
      return false;
    }
  }

  public SList<T> append(SList<T> s) {
    if (this.isNull()) return s;
    return this.cdr().append(s).cons(car());
  }

  public SList<T> reverse() {
    return this.reverseRec(new SList<T>());
  }

  private SList<T> reverseRec(SList<T> rev) {
    if (this.isNull()) return rev;
    return this.cdr().reverseRec(rev.cons(this.car()));
  }
}
```

Programmazione Dinamica

Fibonacci Top-down (memoization)

```java
public class Fibonacci {
  public static long fibMem(int n) {
    long[] h = new long[n + 1];
    for (int i = 0; i < n + 1; i++) h[i] = UNKNOWN;
    return fibRec(n, h);
  }

  public static long fibRec(int n, long[] h) {
    if (h[n] == UNKNOWN) {
      if (n < 2) {
        h[n] = 1;
      } else {
        h[n] = fibRec(n - 2, h) + fibRec(n - 1, h);
      }
    }
    return h[n];
  }

  private static final int UNKNOWN = 0;

  public static void main(String[] args) {
    int n = Integer.parseInt(args[0]);

    long t = System.currentTimeMillis();
    long res = fibMem(n);
    System.out.println(
        "fib(" + n + ") = " + res + " in " + (System.currentTimeMillis() - t) + "ms");
  }
}
```

Manhattan Top-down

```java
public class Manhattan {
  public static long paths(int i, int j) {
    if (i == 0 || j == 0) {
      return 1;
    } else {
      return paths(i - 1, j) + paths(i, j - 1);
    }
  }

  public static long pathsMem(int i, int j) {
    long[][] h = new long[i + 1][j + 1];
    for (int x = 0; x < i + 1; x++) {
      for (int y = 0; y < j + 1; y++) {
        h[x][y] = UNKNOWN;
      }
    }
    return pathsRec(i, j, h);
  }

  private static long pathsRec(int i, int j, long[][] h) {
    if (h[i][j] == UNKNOWN) {
      if (i == 0 || j == 0) {
        h[i][j] = 1;
      } else {
        h[i][j] = pathsRec(i - 1, j, h) + pathsRec(i, j - 1, h);
      }
    }
    return h[i][j];
  }

  private static final long UNKNOWN = 0;

  public static void main(String[] args) {
    int i = Integer.parseInt(args[0]);
    int j = Integer.parseInt(args[1]);
    long nPaths = pathsMem(i, j);
    System.out.println(nPaths);
  }
}
```

Manhattan Bottom-up

```java
public class Manhattan {
  public static long pathsDp(int i, int j) {
    long[][] h = new long[i + 1][j + 1];
    for (int y = 0; y < j + 1; y++) {
      h[0][y] = 1;
    }
    for (int x = 0; x < i + 1; x++) {
      h[x][0] = 1;
    }
    for (int x = 1; x < i + 1; x++) {
      for (int y = 1; y < j + 1; y++) {
        h[x][y] = h[x - 1][y] + h[x][y - 1];
      }
    }
    return h[i][j];
  }

  public static void main(String[] args) {
    int i = Integer.parseInt(args[0]);
    int j = Integer.parseInt(args[1]);
    long nPaths = pathsDp(i, j);
    System.out.println(nPaths);
  }
}
```

Llcs

```java
/** Longest common subsequence - Dynamic Programming */
public class Llcs {
  /** Longest common subsequence (recursive) */
  // public static int llcs(String u, String v) {
  //   int m = u.length();
  //   int n = v.length();
  //
  //   if (m == 0 || n == 0) {
  //     return 0;
  //   } else if (u.charAt(0) == v.charAt(0)) {
  //     return 1 + llcs(u.substring(1), v.substring(1));
  //   } else {
  //     return Math.max(llcs(u.substring(1), v), llcs(u, v.substring(1)));
  //   }
  // }

  // Memoization (Top-down) solution
  public static int llcsMem(String u, String v) {
    int m = u.length();
    int n = v.length();
    int[][] h = new int[m + 1][n + 1];
    for (int x = 0; x < m + 1; x++) {
      for (int y = 0; y < n + 1; y++) {
        h[x][y] = UNKNOWN;
      }
    }
    return llcsRec(u, v, h);
  }

  private static int llcsRec(String u, String v, int[][] h) {
    int i = u.length();
    int j = v.length();

    if (h[i][j] == UNKNOWN) {
      if (i == 0 || j == 0) {
        h[i][j] = 0;
      } else if (u.charAt(0) == v.charAt(0)) {
        h[i][j] = 1 + llcsRec(u.substring(1), v.substring(1), h);
      } else {
        h[i][j] = Math.max(llcsRec(u.substring(1), v, h), llcsRec(u, v.substring(1), h));
      }
    }

    return h[i][j];
  }
```

```java
  // Bottom-up solution
  public static int llcsBottomUp(String u, String v) {
    int i = u.length();
    int j = v.length();

    int[][] h = new int[i + 1][j + 1];
    for (int y = 0; y < j + 1; y++) {
      h[0][y] = 0;
    }
    for (int x = 0; x < i + 1; x++) {
      h[x][0] = 0;
    }
    for (int x = 1; x < i + 1; x++) {
      for (int y = 1; y < j + 1; y++) {
        if (u.charAt(i - x) == v.charAt(j - y)) {
          h[x][y] = h[x - 1][y - 1] + 1;
        } else {
          h[x][y] = Math.max(h[x - 1][y], h[x][y - 1]);
        }
      }
    }
    return h[i][j];
  }

  private static final int UNKNOWN = -1;

  public static void main(String[] args) {
    String u = "arto";
    String v = "atrio";

    if (args.length >= 2) {
      u = args[0];
      v = args[1];
    }

    long nSolutionsTopDown = llcsMem(u, v);
    long nSolutionsBottomUp = llcsBottomUp(u, v);
    assert nSolutionsTopDown == nSolutionsBottomUp;
    System.out.println("llcsMem(" + u + ", " + v + ") = " + nSolutionsTopDown);
    System.out.println("llcsBottomUp(" + u + ", " + v + ") = " + nSolutionsBottomUp);
  }
}
```

Lcs

```java
public class Lcs {
  public static String lcs(String u, String v) {
    int m = u.length();
    int n = v.length();

    if (m == 0 || n == 0) {
      return "";
    } else if (u.charAt(0) == v.charAt(0)) {
      return u.charAt(0) + lcs(u.substring(1), v.substring(1));
    } else {
      return longer(lcs(u.substring(1), v), lcs(u, v.substring(1)));
    }
  }

  private static String longer(String u, String v) {
    int m = u.length();
    int n = v.length();

    if (m < n) {
      return v;
    } else if (m > n) {
      return u;
    } else if (Math.random() < 0.5) {
      return v;
    } else {
      return u;
    }
  }

  // Memoization (Top-doum) solution
  public static String lcsMem(String u, String v) {
    int m = u.length();
    int n = v.length();
    String[][] h = new String[m + 1][n + 1];
    for (int x = 0; x < m + 1; x++) {
      for (int y = 0; y < n + 1; y++) {
        h[x][y] = null;
      }
    }
    return lcsRec(u, v, h);
  }

  private static String lcsRec(String u, String v, String[][] h) {
    int i = u.length();
    int j = v.length();

    if (h[i][j] == null) {
      if (i == 0 || j == 0) {
        h[i][j] = "";
      } else if (u.charAt(0) == v.charAt(0)) {
        h[i][j] = u.charAt(0) + lcsRec(u.substring(1), v.substring(1), h);
      } else {
        h[i][j] = longer(lcsRec(u.substring(1), v, h), lcsRec(u, v.substring(1), h));
      }
    }

    return h[i][j];
  }
```

```java
  // Bottom-up solution
  public static String lcsBottomUp(String u, String v) {
    int i = u.length();
    int j = v.length();

    String[][] h = new String[i + 1][j + 1];
    for (int y = 0; y < j + 1; y++) {
      h[0][y] = "";
    }
    for (int x = 0; x < i + 1; x++) {
      h[x][0] = "";
    }
    for (int x = 1; x < i + 1; x++) {
      for (int y = 1; y < j + 1; y++) {
        if (u.charAt(i - x) == v.charAt(j - y)) {
          h[x][y] = u.charAt(i - x) + h[x - 1][y - 1];
        } else {
          h[x][y] = longer(h[x - 1][y], h[x][y - 1]);
        }
      }
    }
    return h[i][j];
  }

  // Smart bottom-up solution
  public static String lcsDp(String u, String v) {
    int i = u.length();
    int j = v.length();

    int[][] h = new int[i + 1][j + 1];
    for (int y = 0; y < j + 1; y++) {
      h[0][y] = 0;
    }
    for (int x = 0; x < i + 1; x++) {
      h[x][0] = 0;
    }
    for (int x = 1; x < i + 1; x++) {
      for (int y = 1; y < j + 1; y++) {
        if (u.charAt(i - x) == v.charAt(j - y)) {
          h[x][y] = h[x - 1][y - 1] + 1;
        } else {
          h[x][y] = Math.max(h[x - 1][y], h[x][y - 1]);
        }
      }
    }

    String s = "";
    int x = i, y = j;
    while (h[x][y] > 0) {
      if (u.charAt(i - x) == v.charAt(j - y)) {
        s += u.charAt(i - x);
        x--;
        y--;
      } else if (h[x - 1][y] < h[x][y - 1]) {
        y--;
      } else if (h[x - 1][y] > h[x][y - 1]) {
        x--;
      } else if (Math.random() < 0.5) {
        y--;
      } else {
        x--;
      }
    }

    return s;
  }
}
```

Astrazione sullo stato

Tavola rotonda

==> 240430_astrazione_sullo_stato_tavola_rotonda/TavolaRotonda.java <==

```java
public class TavolaRotonda {
  private int quanti;
  private int[] cavalieri;
  private int brocca;

  public TavolaRotonda(int n) {
    this.quanti = n;
    this.brocca = 0;
    this.cavalieri = new int[2 * n - 1];
    for (int i = 1; i <= n; i++) {
      cavalieri[i - 1] = i;
    }
  }

  public int quantiCavalieri() {
    return quanti;
  }

  public int chiHaLaBrocca() {
    return cavalieri[brocca];
  }

  public void serve() {
    if (quanti > 1) {
      cavalieri[brocca + 1] = cavalieri[brocca];
      brocca = brocca + 1;
      quanti--;
    }
  }

  public void passa() {
    if (quanti > 1) {
      cavalieri[brocca + quanti] = cavalieri[brocca];
      brocca = brocca + 1;
    }
  }
}
```

==> 240430_astrazione_sullo_stato_tavola_rotonda/Test.java <==

```java
public class Test {
  public static int ultimoCavaliere(int n) {
    TavolaRotonda tr = new TavolaRotonda(n);
    while (tr.quantiCavalieri() > 1) {
      tr.serve();
      tr.passa();
    }
    return tr.chiHaLaBrocca();
  }

  public static void main(String[] args) {
    int n = Integer.parseInt(args[0]);

    int u = 0;
    for (int k = 1; k <= n; k++) {
      u = ultimoCavaliere(k);
    }
    System.out.println("u: " + u);
  }
}
```

Regine

==> 240506_astrazione_sullo_stato_regine/Board.java <==

```java
public class Board {
  private static final String ROWS = " 123456789ABCDEF";
  private static final String COLS = " abcdefghijklmno";

  private final int size;
  private int queens;
  private String config;

  private int[] rowUnderAttack;
  private int[] colUnderAttack;
  private int[] dg1UnderAttack;
  private int[] dg2UnderAttack;

  public Board(int n) {
    this.size = n;
    this.queens = 0;

    this.rowUnderAttack = new int[n];
    this.colUnderAttack = new int[n];
    this.dg1UnderAttack = new int[2 * n - 1];
    this.dg2UnderAttack = new int[2 * n - 1];
    for (int i = 0; i < n; i++) {
      this.rowUnderAttack[i] = 0;
      this.colUnderAttack[i] = 0;
    }
    for (int i = 0; i < 2 * n - 1; i++) {
      this.dg1UnderAttack[i] = 0;
      this.dg2UnderAttack[i] = 0;
    }

    this.config = " ";
  }

  public int size() {
    return this.size;
  }

  public int queensOn() {
    return this.queens;
  }

  public boolean underAttack(int i, int j) {
    int n = this.size;
    return rowUnderAttack[i - 1] > 0
        || colUnderAttack[j - 1] > 0
        || dg1UnderAttack[i - j + n - 1] > 0
        || dg2UnderAttack[i + j - 2] > 0;
  }

  public void addQueen(int i, int j) {
    int n = this.size;
    rowUnderAttack[i - 1]++;
    colUnderAttack[j - 1]++;
    dg1UnderAttack[i - j + n - 1]++;
    dg2UnderAttack[i + j - 2]++;

    queens++;
    config += "" + COLS.charAt(j) + ROWS.charAt(i) + " ";
  }
```

```
  public void removeQueen(int i, int j) {
    int n = this.size;
    rowUnderAttack[i - 1]--;
    colUnderAttack[j - 1]--;
    dg1UnderAttack[i - j + n - 1]--;
    dg2UnderAttack[i + j - 2]--;

    queens--;

    String pair = "" + COLS.charAt(j) + ROWS.charAt(i);
    int k = config.indexOf(pair);
    config = config.substring(0, k) + config.substring(k + 3);
  }
  public String arrangement() {
    return this.config;
  }
  public String toString() {
    return "[" + this.arrangement() + "]";
  }
}
```

==> 240506_astrazione_sullo_stato_regine/Test.java <==

```
public class Test {
  public static int numSoluzioni(int n) {
    return numCompletamenti(new Board(n));
  }
  public static int numCompletamenti(Board b) {
    int n = b.size();
    int q = b.queensOn();
    if (q == n) {
      return 1;
    } else {
      int i = q + 1;
      int count = 0;
      for (int j = 1; j <= n; j++) {
        if (!b.underAttack(i, j)) {
          b.addQueen(i, j);
          count += numCompletamenti(b);
          b.removeQueen(i, j);
        }
      }
      return count;
    }
  }

  public static SList<String> listaSoluzioni(int n) {
    return listaCompletamenti(new Board(n));
  }
  public static SList<String> listaCompletamenti(Board b) {
    int n = b.size();
    int q = b.queensOn();
    if (q == n) {
      return NULL_STRLIST.cons("" + b);
    } else {
      int i = q + 1;
      SList<String> list = NULL_STRLIST;
      for (int j = 1; j <= n; j++) {
        if (!b.underAttack(i, j)) {
          b.addQueen(i, j);
          list = list.append(listaCompletamenti(b));
          b.removeQueen(i, j);
        }
      }
      return list;
    }
  }
  private static final SList<String> NULL_STRLIST = new SList<String>();
}
```

Huffman

==> 240514_codifica_huffman/Huffman.java <==

```
import huffman_toolkit.*;
import java.util.*;

public class Huffman {
  public static int[] chrFreq(String src) {
    InputTextFile in = new InputTextFile(src);

    int[] freq = new int[InputTextFile.CHARS];
    for (int c = 0; c < freq.length; c++) freq[c] = 0;

    while (in.bitsAvailable()) {
      char c = in.readChar();
      freq[c]++;
    }

    in.close();

    return freq;
  }

  public static Node huffmanTree(int[] freq) {
    PriorityQueue<Node> queue = new PriorityQueue<Node>();
    for (int i = 0; i < freq.length; i++) {
      if (freq[i] > 0) {
        Node n = new Node((char) i, freq[i]);
        queue.add(n);
      }
    }

    while (queue.size() > 1) {
      Node l = queue.poll();
      Node r = queue.poll();

      Node n = new Node(l, r);
      queue.add(n);
    }
    return queue.poll();
  }

  public static String[] huffmanTable(Node root) {
    String[] tab = new String[InputTextFile.CHARS];
    Huffman.fillTable(root, "", tab);
    return tab;
  }

  private static void fillTable(Node n, String hc, String[] tab) {
    if (n.isLeaf()) {
      tab[n.symbol()] = hc;
    } else {
      fillTable(n.left(), hc + "0", tab);
      fillTable(n.right(), hc + "1", tab);
    }
  }

  private static String flatTree(Node n) {
    if (n.isLeaf()) {
      char c = n.symbol();
      if (c == '@' || c == '\\') return "\\" + c;
      return "" + n.symbol();
    } else {
      return "@" + flatTree(n.left()) + flatTree(n.right());
    }
  }
```

```
  public static Node compress(String src, String dst) {
    int[] freq = chrFreq(src);
    Node tree = huffmanTree(freq);
    String[] tab = huffmanTable(tree);

    InputTextFile in = new InputTextFile(src);
    OutputTextFile out = new OutputTextFile(dst);

    out.writeTextLine("" + tree.weight());
    out.writeTextLine(flatTree(tree));

    while (in.textAvailable()) {
      char c = in.readChar();
      out.writeCode(tab[c]);
    }
    in.close();
    out.close();

    return tree;
  }

  /* Decompression */

  private static char restoreChar(InputTextFile in, Node n) {
    do {
      int bit = in.readBit();
      if (bit == 0) n = n.left();
      else n = n.right();
    } while (!n.isLeaf());
    return n.symbol();
  }

  private static Node restoreTree(InputTextFile in) {
    char c = in.readChar();
    if (c == '@') {
      return new Node(restoreTree(in), restoreTree(in));
    } else {
      if (c == '\\') c = in.readChar();
      return new Node(c, 0);
    }
  }

  public static void decompress(String src, String dst) {
    InputTextFile in = new InputTextFile(src);
    OutputTextFile out = new OutputTextFile(dst);

    int count = Integer.parseInt(in.readTextLine());

    Node root = restoreTree(in);
    in.readTextLine();

    for (int i = 0; i < count; i++) {
      char c = restoreChar(in, root);
      out.writeChar(c);
    }
    in.close();
    out.close();
  }
}
```

==> 240514_codifica_huffman/Node.java <==

```java
public class Node implements Comparable<Node> {

  private final char chr;
  private final int wgt;
  private final Node lft;
  private final Node rgt;

  public Node(char c, int w) {
    this.chr = c;
    this.wgt = w;
    this.lft = null;
    this.rgt = null;
  }

  public Node(Node l, Node r) {
    chr = (char) 0;
    this.wgt = l.weight() + r.weight();
    this.lft = l;
    this.rgt = r;
  }

  public boolean isLeaf() {
    return this.lft == null;
  }

  public char symbol() {
    return this.chr;
  }

  public int weight() {
    return this.wgt;
  }

  public Node left() {
    return this.lft;
  }

  public Node right() {
    return this.rgt;
  }

  @Override
  public int compareTo(Node n) {
    if (this.weight() < n.weight()) return -1;
    else if (this.weight() == n.weight()) return 0;
    return 1;
  }
}
```

==> 240514_codifica_huffman/Test.java <==

```java
public class Test {
  public static void main(String[] args) {
    Node root = Huffman.compress("Test.java", "C.txt");
    Huffman.decompress("C.txt", "D.txt");
  }
}
```

Stack

==> 240521_stack/Huffman.java <==

```java
import huffman_toolkit.*;
import java.util.*;

public class Huffman {
  public static int[] chrFreq(String src) {
    InputTextFile in = new InputTextFile(src);

    int[] freq = new int[InputTextFile.CHARS];
    for (int c = 0; c < freq.length; c++) freq[c] = 0;

    while (in.bitsAvailable()) {
      char c = in.readChar();
      freq[c]++;
    }

    in.close();

    return freq;
  }

  public static Node huffmanTree(int[] freq) {
    PriorityQueue<Node> queue = new PriorityQueue<Node>();
    for (int i = 0; i < freq.length; i++) {
      if (freq[i] > 0) {
        Node n = new Node((char) i, freq[i]);
        queue.add(n);
      }
    }

    while (queue.size() > 1) {
      Node l = queue.poll();
      Node r = queue.poll();

      Node n = new Node(l, r);
      queue.add(n);
    }
    return queue.poll();
  }

  public static String[] huffmanTable(Node root) {
    Stack<Coppia> stack = new Stack<Coppia>();
    stack.push(new Coppia(root, ""));

    String[] tab = new String[InputTextFile.CHARS];
    while (!stack.isEmpty()) {
      Coppia c = stack.pop();
      Node n = c.node;
      String path = c.path;

      if (n.isLeaf()) {
        tab[n.symbol()] = path;
      } else {
        stack.push(new Coppia(n.left(), path + "0"));
        stack.push(new Coppia(n.right(), path + "1"));
      }
    }
    return tab;
  }
```

```
private static String flatTree(Node root) {
  Stack<Node> stack = new Stack<Node>();
  stack.push(root);

  String solution = "";

  while (!stack.isEmpty()) {
    Node node = stack.pop();
    if (node.isLeaf()) {
      char c = node.symbol();
      if (c == '@' || c == '\\') solution += "\\" + c;
      solution += "" + node.symbol();
    } else {
      solution += "@";
      stack.push(node.right());
      stack.push(node.left());
    }
  }
  return solution;
}

public static Node compress(String src, String dst) {
  int[] freq = chrFreq(src);
  Node tree = huffmanTree(freq);
  String[] tab = huffmanTable(tree);

  InputTextFile in = new InputTextFile(src);
  OutputTextFile out = new OutputTextFile(dst);

  out.writeTextLine("" + tree.weight());
  out.writeTextLine(flatTree(tree));

  while (in.textAvailable()) {
    char c = in.readChar();
    out.writeCode(tab[c]);
  }
  in.close();
  out.close();

  return tree;
}
```

```
  /* Decompression */

  private static char restoreChar(InputTextFile in, Node n) {
    do {
      int bit = in.readBit();
      if (bit == 0) n = n.left();
      else n = n.right();
    } while (!n.isLeaf());
    return n.symbol();
  }

  private static Node restoreTree(InputTextFile in) {
    Stack<Frame> stack = new Stack<Frame>();
    stack.push(new Frame());

    Node n = null;

    while (!stack.isEmpty()) {
      Frame f = stack.peek();
      if (f.getState() == 0) {
        char c = in.readChar();
        if (c == '@') {
          f.setState(1);
          stack.push(new Frame());
        } else {
          if (c == '\\') c = in.readChar();
          n = new Node(c, 0);
          stack.pop();
        }
      } else if (f.getState() == 1) {
        f.setLeft(n);
        stack.push(new Frame());
        f.setState(2);
      } else {
        f.setRight(n);
        n = new Node(f.getLeft(), f.getRight());
        stack.pop();
      }
    }

    return n;
  }

  public static void decompress(String src, String dst) {
    InputTextFile in = new InputTextFile(src);
    OutputTextFile out = new OutputTextFile(dst);

    int count = Integer.parseInt(in.readTextLine());

    Node root = restoreTree(in);
    in.readTextLine();

    for (int i = 0; i < count; i++) {
      char c = restoreChar(in, root);
      out.writeChar(c);
    }
    in.close();
    out.close();
  }
}
```

==> 240521_stack/Coppia.java <==

```java
public class Coppia {
  public final Node node;
  public final String path;

  public Coppia(Node n, String s) {
    this.node = n;
    this.path = s;
  }
}
```

==> 240521_stack/Frame.java <==

```java
public class Frame {
  private int stato;
  private Node left;
  private Node right;

  public int getState() {
    return stato;
  }

  public void setState(int value) {
    stato = value;
  }

  public Node getLeft() {
    return left;
  }

  public void setLeft(Node value) {
    left = value;
  }

  public Node getRight() {
    return right;
  }

  public void setRight(Node value) {
    right = value;
  }

  public Frame() {
    stato = 0;
    left = null;
    right = null;
  }
}
```

==> 240521_stack/Node.java <==

```java
public class Node implements Comparable<Node> {

  private final char chr;
  private final int wgt;
  private final Node lft;
  private final Node rgt;

  public Node(char c, int w) {
    this.chr = c;
    this.wgt = w;
    this.lft = null;
    this.rgt = null;
  }

  public Node(Node l, Node r) {
    chr = (char) 0;
    this.wgt = l.weight() + r.weight();
    this.lft = l;
    this.rgt = r;
  }

  public boolean isLeaf() {
    return this.lft == null;
  }

  public char symbol() {
    return this.chr;
  }

  public int weight() {
    return this.wgt;
  }

  public Node left() {
    return this.lft;
  }

  public Node right() {
    return this.rgt;
  }

  @Override
  public int compareTo(Node n) {
    if (this.weight() < n.weight()) return -1;
    else if (this.weight() == n.weight()) return 0;
    return 1;
  }
}
```

==> 240521_stack/Test.java <==

```java
public class Test {
  public static void main(String[] args) {
    Node root = Huffman.compress("Test.java", "C.txt");
    Huffman.decompress("C.txt", "D.txt");
  }
}
```

Correttezza

```
public class Invariants {
  // Quadrato come somma di numeri dispari
  // (vedi definizione della procedura "unknown" in Scheme)
  public static int sqr( int n ) {  //  Pre:  n >= 0
    int x = 0;
    int y = 0;
    int z = 1;
    while ( x < n ) {  //  Inv:  y = x*x,  z = 2*x + 1,  x <= n
                       //  Term: n - x
      x = x + 1;
      y = y + z;
      z = z + 2;
    }
    return y;  //  Post:  y = n*n
  }

  /*
   *  a. Inv vale all'inizio
   *
   *    Pre ==> Inv(0,0,1)
   *
   *    0 = 0*0,  1 = 2*0 + 1,  0 <= n  Ok
   *
   *  b. Inv si conserva
   *
   *    x,  y,  z : valori delle variabili (stato)
   *                immediatamente prima del passo iterativo
   *                (quindi si sta assumendo x < n)
   *
   *    x', y', z': valori delle variabili (stato)
   *                all fine del passo iterativo considerato
   *
   *
   *    Inv(x,y,z) & (x < n)  ==>  Inv(x',y',z')
   *                    (d)
   *
   *    Inv(x,y,z)    :  y = x*x,  z = 2*x + 1,  x <= n
   *                       (a)       (b)            (c)
   *
   *    Inv(x',y',z') :  y' = x'*x',  z' = 2*x' + 1,  x' <= n
   *
   *                     y+z = (x+1)*(x+1),  z+2 = 2*(x+1) + 1,  x+1 <= n
   *
   *                     y+z = x*x + 2x + 1,  z+2 = 2x + 2 + 1,  x < n
   *                        (a,b)                (b)              (d)
   *
   *   c. Inv permette di dedurre il risultato
   *
   *     Inv(x,y,z) & !(x < n)  ==>  Post
   *
   *     y = n*n,  z = 2*x + 1,  x = n  ==>  y = n*n
   *
   *   t1. Term ha valori naturali
   *
   *     Inv(x,y,z)  ==>  n - x >= 0
   *
   *   t2. Term e' strettamente decrescente
   *
   *     Inv(x,y,z) & (x < n)  ==>  term(x') < term(x)
   *
   *     n - (x+1) = n - x - 1 < n - x
   */
```

```java
// Cubo attraverso somme
public static int cube( int n ) {  //  Pre:  n >= 0
  int x = 0;
  int y = 0;
  int u = 1;
  int v = 6;

  while ( x < n ) {  //  Inv:  0 <= x <= n,  y = x^3,  u = 3x^2 + 3x + 1,  v = 6x + 6
                     //  Term: n - x
    x = x + 1;
    y = y + u;
    u = u + v;
    v = v + 6;
  }
  return y;  //  Post:  y = n^3
}

// Moltiplicazione del "contadino russo"
public static int mul( int a, int b ) {  //  Pre:  a >= 0,  b >= 0
  int x = a;
  int y = b;
  int z = 0;

  while ( y > 0 ) {     //  Inv:  x*y + z = a*b,  y >= 0
                        //  Term: y
    if ( y % 2 > 0 ) {
      z = z + x;
    }
    x = 2 * x;
    y = y / 2;
  }
  return z;  //  Post:  z = a*b
}
```

```java
// Minimo comune multiplo
public static int lcm( int m, int n ) {  // Pre:  m, n > 0

  int x = m;
  int y = n;

  while ( x != y ) {  // Inv:  0 < x, y <= mcm(m,n),  x mod m = y mod n = 0
                      // Term: 2mn - x - y
    if ( x < y ) {
      x = x + m;
    } else {
      y = y + n;
    }
  }
  return x;  // Post:  x = mcm(m,n)
}

/*
 *  a. Inv vale all'inizio
 *
 *    Inv(m,n):  0 < m, n <= mcm(m,n),  m mod m = n mod n = 0  Ok
 *
 *  b. Inv si conserva
 *
 *    Inv(x,y) & (x != y)  ==>  Inv(x',y')
 *
 *    Assumo:  (a) 0 < x, y <= mcm(m,n),  (b,c) x mod m = y mod n = 0,  (d) x != y
 *
 *    Dimostro:  0 < x', y' <= mcm(m,n),  x' mod m = y' mod n = 0
 *
 *  b1. Assumo inoltre:  (e) x < y
 *
 *    Dimostro:  0 < x+m, y <= mcm(m,n),  (x+m) mod m = y mod n = 0
 *
 *  b2. Assumo inoltre:  (f) x >= y & ...  ==>  x > y
 *
 *    Dimostro:  0 < x, y+n <= mcm(m,n),  x mod m = (y+n) mod n = 0
 *
 *  c.
 *
 *    Inv(x,y) & (x = y)  ==>  Post(x,y)
 *
 *    Assumo:  (a) 0 < x, y <= mcm(m,n),  (b,c) x mod m = y mod n = 0,  (d) x = y
 *
 *    Dimostro:  x = y = mcm(m,n)
 *
 *  t1. Term ha valori interi non negativi  se vale Inv :
 *
 *        2mn - x - y >= 2 mcm(m,n) - x - y
 *                    = (mcm(m,n) - x) + (mcm(m,n) - y) >= 0
 *
 *  t2. Term decresce strettamente ad ogni iterazione
 *
 *        2mn - x' - y' <= 2mn - x - y - min(m,n) < 2mn - x - y
 *
 */
```

```java
// Fattorizzazione in fattori primi
public static int[] factorization( int n ) {  // Pre:  n >= 2
  int[] fattori = new int[ n+1 ];

  for ( int i=0; i<=n; i=i+1 ) {
    fattori[i] = 0;
  }
  int x = n;
  int p = 2;

  while ( x > 1 ) {       // Inv:  1 <= x <= n,
                          //       n = x * Prod (k: [2,n]) k^fattori[k],
                          //       x non ha fattori < p,  2 <= p <= n
                          // Term: x + n - p
    if ( x % p == 0 ) {
      fattori[p] = fattori[p] + 1;
      x = x / p;
    } else {
      p = p + 1;
  }}
  return fattori;  // Post: n = Prod (k: [2,n]) k^fattori[k]
}

// Massimo comun divisore esteso
public static int[] gcd( int a, int b ) {  //  Pre:  a > 0,  b > 0
  int x = a;
  int i = 1;
  int j = 0;

  int y = b;
  int k = 0;
  int l = 1;

  while ( x != y ) {  //  Inv:  MCD(x,y) = MCD(a,b),  x = i*a + j*b,  y = k*a + l*b
                      //  Term: x + y
    if ( x < y ) {
      y = y - x;
      k = k - i;
      l = l - j;
    } else {
      x = x - y;
      i = i - k;
      j = j - l;
    }
  }
  return new int[] { x, i, j };  //  Post:  x = MCD(a,b) = i*a + j*b
}

// Ricerca binaria in un array ordinato
public static int pos( int x, int[] v ) {  //  Pre:  v ordinato,  esiste j t.c. v[j] = x
  int l = 0;
  int r = v.length - 1;
  int m;

  while ( l < r ) {   //  Inv:  l <= r,  esiste j in [l,r] t.c. v[j] = x
                      //  Term: r - l
    m = (l + r) / 2;

    if ( v[m] < x ) {
      l = m + 1;
    } else {
      r = m;
    }
  }
  return l;  //  v[l] = x
}
```

```java
  // Valore di un numerale rappresentato in notazione binaria
  public static int val( String b ) {  //  Pre:  b stringa di 0, 1
    int v = 0;
    int i = 0,  n = b.length();

    while ( i < n ) {  //  Inv:  v = Sum( bi * 2^(i-1-j) ) per j in [0,i-1],  i <= n
                       //  Term: n - i

      v = 2 * v + ( (b.charAt(i) == '0') ? 0 : 1 );
      i = i + 1;
    }
    return v;  //  Post:  v = Sum( bi * 2^(n-1-j) ) per j in [0,n-1]
  }


  // Ripartizione delle componenti di in un array
  public static int partition( int x, int[] v ) {  //  Pre:  esiste k t.c. v[k] <= x,  esiste k t.c. v[k] > x
    int i = 0;
    int j = v.length - 1;

    while ( i <= j ) {       //  Inv:  i <= j + 1,  ogni k < i . ( v[k] <= x ),  ogni k > j . ( x < v[k] )
                             //  Term: j - i + 1
      while ( v[i] <= x ) {
        i = i + 1;
      }
      while ( x < v[j] ) {
        j = j - 1;
      }
      if ( i < j ) {
        int t = v[i];
        v[i] = v[j];
        v[j] = t;
        i = i + 1;
        j = j - 1;
    }}
    return j;  //  Post:  ogni k <= j . ( v[k] <= x ),  ogni k > j . ( x < v[k] )
  }
}  // class Invariants
```