Risoluzione automatica del cruciverba

Progetto di Esperienze di programmazione

Davide Coffaro Matricola 556603

2020

**DESCRIZIONE DEL PROBLEMA**  
Il problema di “Risoluzione automatica del cruciverba” è quello di generare lo schema del cruciverba a partire da un input, il quale conterrà anche la prima parola da inserire nello schema, e da qui analizzare lo schema e completarlo con la lista di parole disponibili fornite al programma.

Dato che le parole sono collegate tra loro, nel momento in cui inserisco nello schema una parola tra quelle disponibili, questo inserimento influirà sulla ricerca successiva delle parole da inserire.

Potrebbe verificarsi che durante la ricerca della parola successiva da inserire, ci siano più parole da poter inserire in quelle caselle e questa situazione potrebbe bloccare la risoluzione del cruciverba.

Inoltre la lista di parole disponibili che servirà per completare il cruciverba potrebbe contenere parole in più che non sono necessarie al suo completamento, questo potrebbe bloccare il proseguimento della risoluzione del cruciverba oppure far sì che vengano inserite parole sbagliate all’interno dello schema del cruciverba, quindi è possibile prevedere una situazione in cui bisogna ripristinare una situazione precedente.

È possibile trovare un limite inferiore alla complessità del problema di “Risoluzione automatica del cruciverba” scomponendo il problema in sotto-problemi, che saranno:

con n=numero parole da completare dello schema  
e m=lunghezza parola arrotondata alla parola più lunga per gestire il caso pessimo

* Cerca fra le parole disponibili da inserire -> L\_ricerca(n)
* Confronta le lettere delle caselle con quella della parola disponibile corrente -> L\_confronta(m)
* Inserire parole nello schema -> L\_inserimento(n-1)

Ma per ricomporre i limiti ottenuti dovremmo stabilire la dipendenza funzionale, quindi scelgo banalmente la dimensione del cruciverba come limite inferiore e ottengo quindi che il problema di completare un cruciverba con le parole disponibili ha complessità Ω(n^2).

**ALGORITMI PER RISOLVERE IL PROBLEMA**

* Algoritmo1 – parto dallo schema, cerco la parola da inserire fino a completare il cruciverba

PROCEDURA ALGORITMO1:

1. Memorizzo tutte le parole di una certa lunghezza;
2. Prendo la parola con più lettere già inserite e cerco se sono presenti parole congrue (scarto quelle con lettere errate) nella struttura dati contenente le parole da inserire
3. Se è presente solo una parola da poter inserire la inserisco aggiornando le parole ad essa collegate e la cancello dalla struttura dati contenente le parole disponibili;
4. se sono presenti altre parole della stessa lunghezza memorizzate al punto 1) torno al punto 2);
5. se il cruciverba non è ancora completo torno al punto1) cercando parole di un’altra lunghezza;
6. se il cruciverba è stato completato stampo “Cruciverba completato”

CONDIZIONI DI TERMINAZIONE:  
1) il cruciverba è stato completato;  
2) ho eseguito i passi da 1 a 5 un numero prefissato di volte -> il cruciverba non è stato completato

* Algoritmo2 – procedura simile all’algoritmo1 ma:

1. Utilizzo di strutture dati per memorizzare parole della stessa lunghezza
2. Quando inserisco la parola di una certa lunghezza nello schema, la elimino anche dalla struttura dati che la conteneva
3. Quando tutte le strutture dati contenenti parole sono vuote, termino l’algoritmo

Problemi relativi all’algoritmo2

* + Riempire la struttura dati delle parole all’inizio durante la creazione delle caselle
  + Memoria occupata dalla struttura dati delle parole
* Algoritmo4 (con AI) - utilizzo di algoritmi di intelligenza artificiale per cercare la soluzione in modo più efficiente, correggendo gli errori per cui gli altri algoritmi si bloccavano ad un certo punto non trovando altre parole da inserire nello schema del cruciverba. Con questo algoritmo, che sfrutta la strategia CSP – Constraints Satisfaction Problem, analizzo le parole che possono essere inserite nello schema e nel caso in cui arrivassi ad un punto in cui non è possibile inserire altre parole ma lo schema non è ancora completo, si possa tornare indietro e provare le parole alternative fino al completamento dello stesso.

PROCEDURA ALGORITMO4:

al momento della creazione dello schema inserisco tutte le parole in orizzontale o in verticale (non ancora completate) all’interno di una struttura dati contenente variabili; dopo analizzo i possibili valori che possono essere inseriti all’interno delle variabili inserendo anch’essi in una struttura dati collegata alle singole variabili, creando così tanti domini delle parole da inserire nello schema, uno per ogni variabile.  
  
Passi di soluzione dell’algoritmo:  
1) ricerca della variabile a cui assegnare un valore con strategia MRV (minimum remaining values) e grado maggiore a parità di MRV: MRV cioè analizzo i valori residui nel dominio di ogni variabile e prendo quelle che hanno il numero minore di valori; se sono presenti più variabili con lo stesso numero di valori nel dominio allora viene presa la variabile che vincola più variabili ad essa collegata (cioè prendo la variabile con lunghezza maggiore perché coinvolgerà un maggior numero di variabili ad essa collegata);

2) inserimento di un valore del dominio della variabile all’interno della variabile con strategia di scelta valore per la variabile attuale in ordine di come sono stati inseriti i valori nel dominio di quella variabile;

3) meccanismo di inferenza in cui vengono ridotti i domini delle variabili collegate a quella corrente e delle variabili con lo stesso numero di lettere, in caso di dominio vuoto (non sono presenti altre parole disponibili) di una variabile collegata o di una variabile con lo stesso numero di lettere e lo schema del cruciverba non è ancora completo, significa che uno degli assegnamenti di valori alle variabili precedentemente effettuato non era corretto e bisogna fare dei passi all’indietro provando un altro valore (sfrutta il backtracking cronologico ed inserisce il valore successivo del dominio della stessa variabile) tornando al passo 2 se sono presenti altri valori per la variabile corrente, altrimenti ritorno il controllo alla funzione ricorsiva chiamante. Se invece i domini di variabilli collegate o di variabili con lo stesso numero di lettere non risultano vuoti vado al punto 4;

4) se tutti i passi precedenti sono andati a buon fine proseguo la ricerca sul sottoproblema in cui adesso ho assegnato un valore alla variabile su cui stavo lavorando.

CONDIZIONI DI TERMINAZIONE:

1. quando ho effettuato un numero di assegnamenti uguale al numero di variabili dello schema;
2. quando non esistono più variabili a cui assegnare un valore.

**SCELTE IMPLEMENTATIVE**

Nell’implementazione del cruciverba ho deciso di scomporre lo schema del cruciverba nelle parole e nelle caselle, all’interno delle parole ho la stringa della parola associata che uso per cercare la parola da inserire nello schema ma scompongo ulteriormente la parola nelle singole caselle in cui ho il carattere (vocale o consonante) ad esse collegate che vengono utilizzate nei confronti per trovare la parola successiva da inserire nello schema e anche per la stampa a video della casella. Per tutto questo utilizzo una struttura dati relativa allo schema per salvare i suoi dati attuali, cioè le parole e le caselle all’interno dello schema del cruciverba, oltre alle funzioni di aggiornamento schema, ricerca di una determinata casella o di un determinato tipo di parola o di una sua caratteristica particolare.  
Lo schema del cruciverba è impostato all’interno del programma con una matrice che definisce le caselle bianche (carattere “.”) e quelle nere (carattere “\*”), la matrice è creata a partire da un file .txt (il cui nome è definito all’interno del programma) in cui la prima riga contiene il numero di righe e di colonne della matrice mentre le righe successive contengono i caratteri sopra descritti per definire le caselle. Un miglioramento dell’implementazione attuale prevede che lo schema venga individuato analizzando l’immagine dello schema stesso che verrà poi trasformata nella matrice.

La struttura dati Schema contiene strutture dati di tipo array per le parole e le caselle che hanno questa complessità temporale (n è il numero di parole da inserire nello schema):  
- inserimento, viene inserito il nuovo elemento in fondo all’array, complessità O(1);  
- ricerca o modifica, conoscendo l’indice di un oggetto all’interno dell’array l’accesso ha complessità O(1), se l’elemento deve essere ricercato per confronti abbiamo al caso pessimo una complessità di O(n);  
- non vengono fatte operazioni di cancellazione sulle parole e sulle caselle del cruciverba.

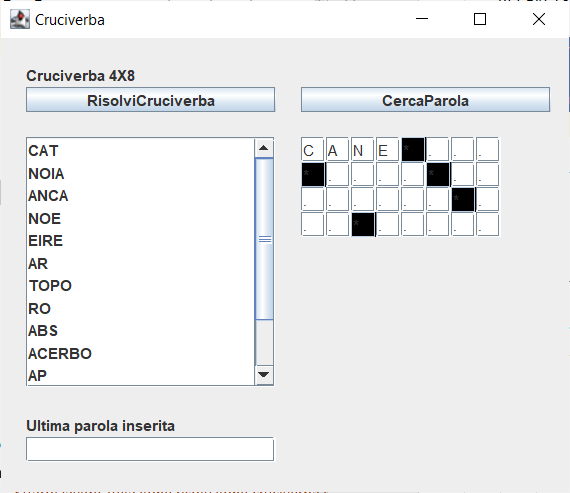
In più utilizzo una struttura dati dinamico di tipo array per memorizzare le parole che possono essere inserite all’interno dello schema; questa struttura dati ha le stesse caratteristiche di quella utilizzata per le parole e le caselle solo che per le cancellazioni si sfrutta prima l’operazione di ricerca di un elemento e poi si effettua la cancellazione, quindi al caso pessimo in cui l’elemento cercato è in fondo all’array abbiamo una complessità O(m) con m numero di parole che possono essere inserite nello schema.  
La lista di parole disponibili per l’inserimento dentro lo schema è importata all’interno del programma nella struttura dati appena descritta da un file .txt letto dal programma e il cui nome è definito nel programma (funzione migliorabile con un’implementazione per selezionare un certo file). Le liste di parole che verranno utilizzate per il completamento dello schema sono una lista creata da me contenente solo le parole che effettivamente fanno parte della soluzione del cruciverba, oppure le altre liste sono state prese da un progetto su github come indicato in bibliografia e in cui ho aggiunto le parole che fanno parte della soluzione del cruciverba.

Dato che il numero di parole dello schema n è inferiore o uguale a m numero di parole che possono essere inserite, allora dico che n=O(m) quindi pongo m come limite superiore ad n, cioè al più avrò complessità O(m) nelle operazioni di ricerca relativo alle strutture dati array delle parole e delle caselle dello schema.

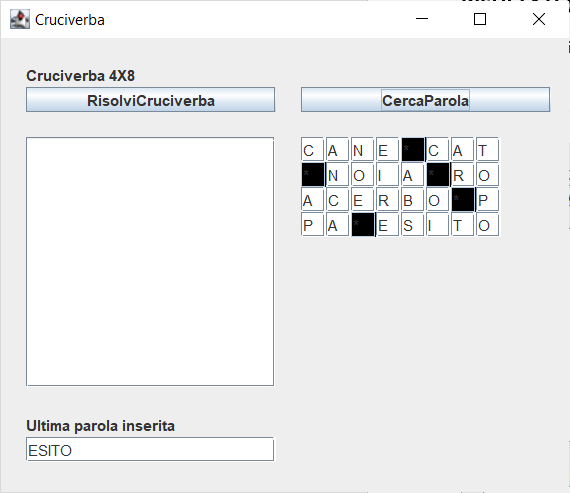
**RISULTATI OTTENUTI**

Gli esempi seguenti sono suddivisi per i 3 algoritmi implementati, utilizzando come lista di parole da utilizzare per riempire lo schema solo quelle che effettivamente fanno parte dello schema, con al più qualche parola in più (vedere esempio2)

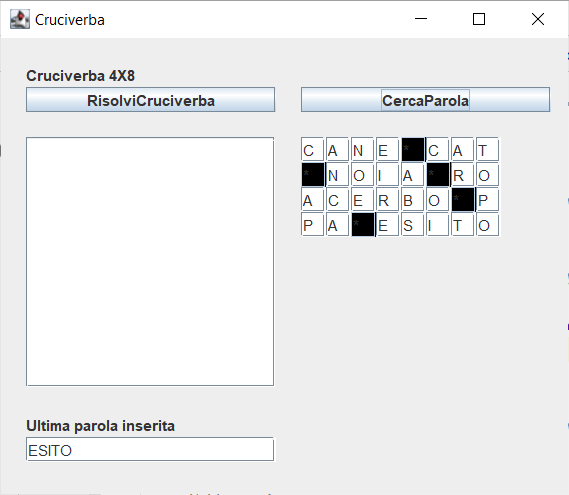
Esempio1: all’apertura del programma



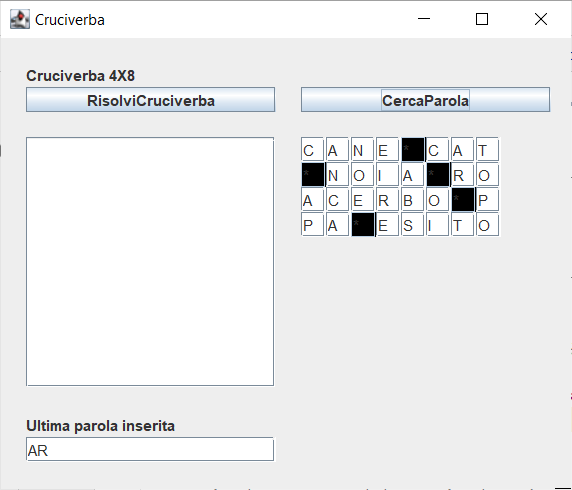
Dopo esecuzione algoritmo1



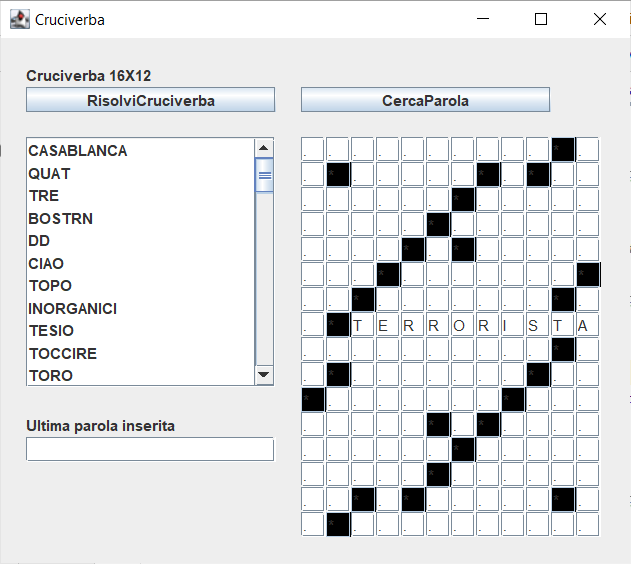
Dopo esecuzione algoritmo2



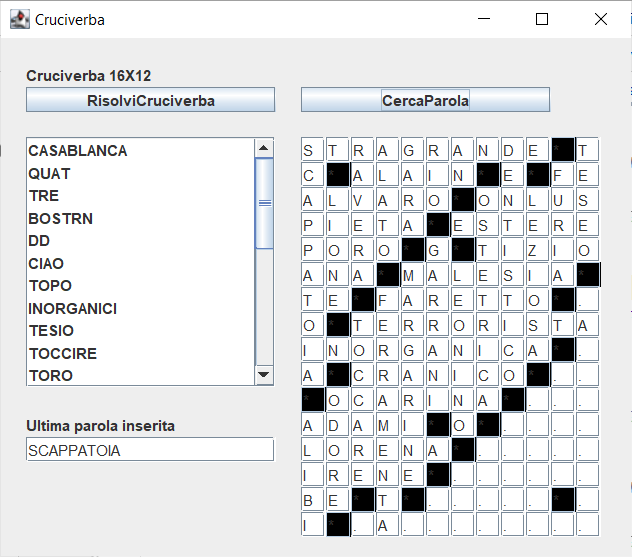
Dopo esecuzione algoritmo4\_AI



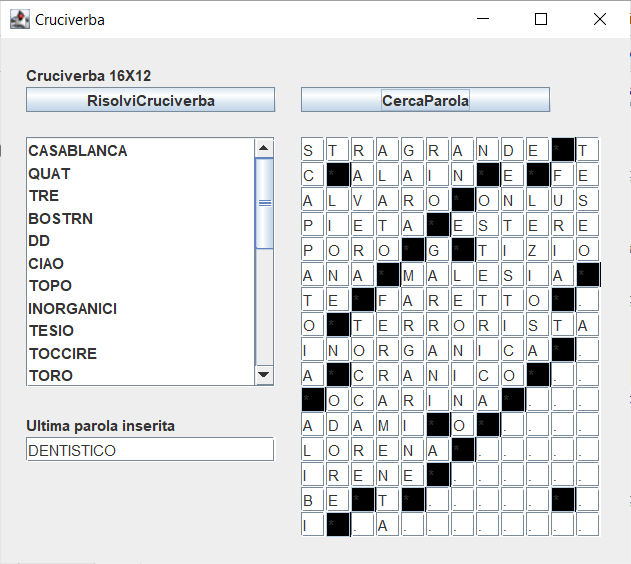
Esempio2: all’apertura del programma



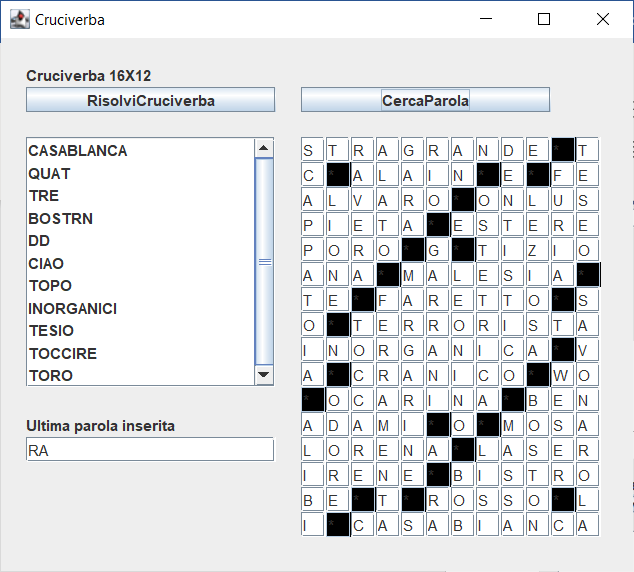
Dopo esecuzione algoritmo1



Dopo esecuzione algoritmo2



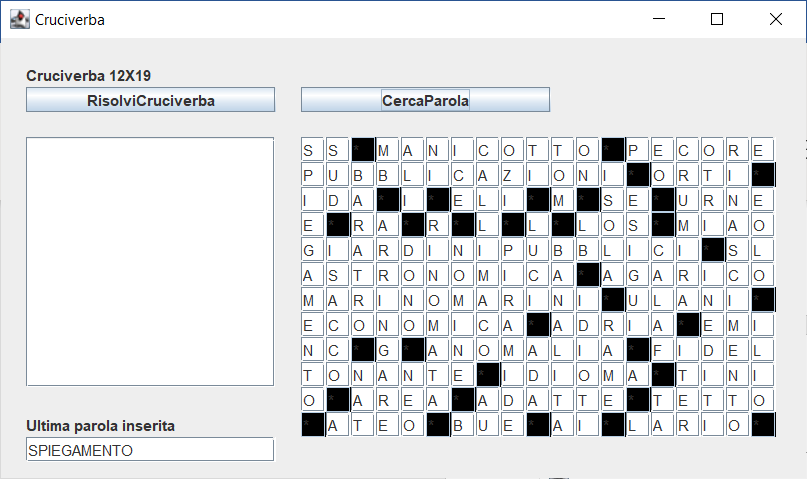
Dopo esecuzione algoritmo4\_AI



Esempio3: all’apertura del programma



Dopo esecuzione algoritmo1



Dopo esecuzione algoritmo2



Dopo esecuzione algoritmo4\_AI



Come possiamo notare dall’esempio 2 l’algoritmo1 e l’algoritmo2 non trovano sempre la soluzione, ma in alcuni casi in cui è possibile inserire più parole nelle stesse caselle non possono continuare la loro risoluzione. Questo problema non è invece presente nell’algoritmo4\_AI in cui se si verifica la precedente situazione l’algoritmo prova prima ad inserire una parola e continua la risoluzione del cruciverba, se in un certo momento non riesce ad inserire altre parole e il cruciverba non è stato ancora completato allora effettua il “backtracking” per cui elimina i valori inseriti dopo aver avuto il blocco, prova ad inserire un altro valore nella casella precedentemente riempita e effettua questa procedura fino al completamento dello schema.

Altri test effettuati utilizzando liste di parole con 1000, 21000, 60000, 280000 e 660000 parole italiane mostrano che l’algoritmo1 e l’algoritmo2 non riescono a trovare mai la soluzione per nessuno degli esempi riportati, mentre l’algoritmo4\_AI ha i seguenti risultati:  
- l’esempio1 viene risolto con tutte le liste di parole provate nei test e si può notare che con poche parole oppure molte parole nella lista di parole italiane da inserire, l’algoritmo completa il cruciverba in modo rapido, mentre per valori intermedi (ad esempio con 21000 parole) impiega più tempo.  
- l’esempio2 e l’esempio3 invece vengono risolti solo con una lista di 1000 parole italiane, con più parole l’algoritmo non riesce a completare lo schema per mancanze di risorse, infatti durante l’esecuzione viene sollevata un eccezione di OutOfMemoryError dovuto dal fatto che vengono effettuate molte chiamate ricorsive per scorrere l’albero delle soluzioni che ha una profondità più alta rispetto all’esempio1 perché sono presenti più parole da completare: si passa infatti da 14 parole dell’esempio1 alle 64 e 72 dell’esempio2 e dell’esempio3; questo fa sì che le combinazioni possibili siano molte di più anche se viene utilizzato il meccanismo di inferenza per ridurre i rami dell’albero delle soluzioni da controllare.

COMPLESSITA’ algoritmo1:  
analizzando l’algoritmo si può notare che ci sono vari cicli annidati necessari per poter completare lo schema e possono dipendere da:   
- cicliMax indicati con n, i cicli massimi superati i quali l’algoritmo termina determinando il completamento o non dello schema;   
- lunghezzaMax indicati con L, la lunghezza massima possibile per le parole dello schema, calcolata analizzando lo schema del cruciverba  
- parole schema e parole disponibili entrambi indicati con m, che indicano il numero delle parole dello schema e quelle della lista di parole da inserire.  
Per approssimare al caso peggiore imposto sia L=O(n) che m=O(n), quindi sia L che m sono dell’ordine di grandezza di n, numero molto alto rispetto agli altri input e trovo che la complessità dell’algoritmo1 è O(n^5) al caso pessimo.

COMPLESSITA’ algoritmo2:  
Anche per questo algoritmo ho diversi cicli annidati e al caso peggiore l’algoritmo ha complessità O(n^5) così come l’algoritmo1, quindi anche con l’utilizzo di liste con all’interno le parole ancora da completare invece di dover controllare una lista in cui ho le informazioni sul completamento delle parole di una certa lunghezza e poi eventualmente creare la lista delle parole di quella lunghezza e lavorarci, non migliora la complessità dell’algoritmo.

COMPLESSITA’ algoritmo4 con AI:  
considerato che viene fatta una chiamata ricorsiva per ogni variabile all’interno dello schema del cruciverba avremo una complessità esponenziale. Questo è dovuto dal fatto che la struttura del grafo dei vincoli derivante dall’implementazione dell’algoritmo CSP è complessa e potrebbe essere ridotta andando ad utilizzare delle tecniche di riduzione del grafo iniziale dei vincoli verso una struttura ad albero risolvibile con complessità minore (tree structured, cutset conditioning e tree decomposition).

Per ogni chiamata a BACKTRACK ho:

1. n\*c operazioni per selectUnassignedVariable
2. ciclo di d iterazioni = d\*c operazioni per scorrere valori presi da orderDomainValues
3. 4\*n\*c operazioni per inference
4. Chiamata ricorsiva backtrack con n-1 variabili

Complessità algoritmo4\_AI -> T(n) = n\*c + d\*c\*(4\*n\*c+T(n-1)) per n>0  
 = 0 per n=0

E dovendo fare n chiamate ricorsive risolvo l’albero di ricorsione che avrà altezza n e in cui ogni nodo avrà costo j\*c+d\*c\*j con j=numero variabili ancora da assegnare

Complessità algoritmo4\_AI = O (c+c\*sommatoria per i=1..n di ( d^i)) = O(d^n)

Quindi l’algoritmo1 e l’algoritmo2 sarebbero migliori perché hanno una complessità polinomiale al caso pessimo mentre l’algoritmo4\_AI ha complessità esponenziale sempre al caso pessimo, l’unico problema è che l’algoritmo1 e 2 non sono completi, cioè non riescono sempre a trovare una soluzione anche se questa è presente. L’algoritmo4\_AI è completo, quindi se è presente una soluzione riesce a trovarla in tempo esponenziale, ma non è ottimo perché la soluzione trovata potrebbe non essere quella meno costosa da trovare nell’albero di ricerca delle soluzioni al problema.

**APPENDICE: codice**

Classe InterfacciaCruciverba:

package com.cruciverbapackage;  
  
*//Coffaro\_Davide\_mat556603\_Progetto ESP cruciverba*import javax.swing.\*;  
import java.awt.\*;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
import java.io.File;  
import java.io.FileNotFoundException;  
import java.util.ArrayList;  
import java.util.InputMismatchException;  
import java.util.Scanner;  
  
public class InterfacciaCruciverba {  
 private JPanel panelMain;  
 private JPanel panelListaParole;  
 private JLabel labelCruciverba;  
 private JButton buttonRisolviCruciverba;  
 private JButton buttonCercaParola;  
 private JList listListaParole;  
 private JScrollPane scrollPaneListaParole;  
 private JLabel labelParolaInserita;  
 private JTextField textFieldParolaInserita;  
 private static ImplementazioneCruciverba *cruciverba1*;  
 private static int *dimensioneFinestraMinima*=500;  
 private long startTime=0;  
 private long stopTime=0;  
 private long totalTime=0;  
  
 *//inizializzazione della matrici che faranno da base dello schema del cruciverba per i 3 esempi creati  
 //proposto un miglioramento a questa implementazione all'interno della relazione (nella sezione Scelte implementative)  
 //esempio1* private static char[][] *matriceSchemaCruciverba*;  
 private ArrayList<JTextField> text;  
 private static ArrayList<String> *dizionarioInput*;  
 private static String *parolaIniziale*;  
 private static int *posizioneRigaIniziale*;  
 private static int *posizioneColonnaIniziale*;  
 private static char *orientamento*;  
  
 public InterfacciaCruciverba() {  
 createUIComponents();  
 buttonRisolviCruciverba.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 startTime=System.*currentTimeMillis*();  
  
 *//lancia procedura di risoluzione cruciverba e ritornando se è stata trovata la soluzione o meno, stampando un messaggio a video* if (*cruciverba1*.risolviCruciverba()) {  
 listListaParole.setListData(*cruciverba1*.dizionario.toArray());  
 stopTime=System.*currentTimeMillis*();  
 totalTime=stopTime-startTime;  
 JOptionPane.*showMessageDialog*(null, "Cruciverba completato in " + totalTime, "Risultato cruciverba", JOptionPane.*INFORMATION\_MESSAGE*);  
 } else {  
 listListaParole.setListData(*cruciverba1*.dizionario.toArray());  
 JOptionPane.*showMessageDialog*(null, "Cruciverba non completato", "Risultato cruciverba", JOptionPane.*ERROR\_MESSAGE*);  
 }  
  
 }  
 });  
 buttonCercaParola.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
  
 *//lancia procedura inserisci1Parola per cui esegue un passo dell'algoritmo e inserisce la parola trovata, se non trova nessuna  
 // parola controlla il risultato del cruciverba* String parolaInserita=*cruciverba1*.inserisci1Parola();  
 if (parolaInserita==null) {  
 listListaParole.setListData(*cruciverba1*.dizionario.toArray());  
 stopTime=System.*currentTimeMillis*();  
 totalTime=stopTime-startTime;  
 if (*cruciverba1*.isAlgResult()) {  
 JOptionPane.*showMessageDialog*(null, "Cruciverba completato in " + totalTime, "Risultato cruciverba", JOptionPane.*INFORMATION\_MESSAGE*);  
 }else{  
 JOptionPane.*showMessageDialog*(null, "Cruciverba non completato", "Risultato cruciverba", JOptionPane.*ERROR\_MESSAGE*);  
  
 }  
 } else {  
 textFieldParolaInserita.setText(parolaInserita);  
 listListaParole.setListData(*cruciverba1*.dizionario.toArray());  
 *//JOptionPane.showMessageDialog(null, "Cruciverba non completato", "Risultato cruciverba", JOptionPane.ERROR\_MESSAGE);* }  
  
 }  
 });  
 }  
  
 public static void main(String[] args) {  
 JFrame frame = new JFrame("Cruciverba");  
 InterfacciaCruciverba window;  
  
 *//importazione dello schema del cruciverba da un file .txt per i 3 esempi creati  
 //la prima riga del file .txt contiene le righe dello schema, la seconda riga contiene le colonne dello schema  
 //nelle successive righe si indica con . per le caselle bianche dello schema, \* per le caselle nere  
  
 //come descritto nella sezione Scelte implementative della relazione, questa implementazione sarebbe  
 //migliorabile con una procedura di analisi dell'immagine* try{  
 *//esempio1* File schema = new File("./schema1.txt");  
  
 *//esempio2  
 //File schema = new File("./schema2.txt");  
  
 //esempio3  
 //File schema = new File("./schema3.txt");* if(schema.isFile()){  
 *importaSchemaCruciverba*(schema);  
 }  
  
 }catch(NullPointerException e){  
 System.*out*.println("Percoso file errato");  
 System.*exit*(-1);  
 }  
  
 window= new InterfacciaCruciverba();  
  
 frame.setContentPane(window.panelMain);  
 frame.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
 frame.pack();  
  
 *//imposto la dimensione della finestra a seconda del numero di righe e di colonne del cruciverba, se sono troppo basse imposto una  
 //dimensione minima* int dimensioneFinestra=20\*(*matriceSchemaCruciverba*.length+ *matriceSchemaCruciverba*[0].length)+100;  
 if (dimensioneFinestra<*dimensioneFinestraMinima*){  
 frame.setSize(*dimensioneFinestraMinima*, *dimensioneFinestraMinima*);  
 }else{  
 frame.setSize(dimensioneFinestra, dimensioneFinestra);  
 }  
 frame.setVisible(true);  
  
 try {  
 *dizionarioInput* = new ArrayList<String>();  
  
 *//importo lista di parole che potranno essere inserite nello schema del cruciverba per i 3 esempi creati  
 //la prima riga del file .txt contiene la prima parola da inserire nello schema, con i riferimenti riga,  
 //colonna e orientamento.  
 //Come descritto nella sezione Scelte implementative della relazione, questa implementazione sarebbe  
 //migliorabile creando una procedura di selezione di un file .txt  
  
 //esempio1  
 //File listaParole = new File("./esempio1.txt");  
  
 //esempio2  
 //File listaParole = new File("./esempio2.txt");  
  
 //esempio3  
 //File listaParole = new File("./esempio3.txt");  
  
 //esempio4 - dizionario italiano completo* File listaParole = new File("./21000\_parole\_italiane.txt");  
  
 if (listaParole.isFile()) {  
 *importaDizionario*(listaParole);  
 }  
 }catch(NullPointerException e){  
 System.*out*.println("Percorso file errato");  
 System.*exit*(-1);  
 }  
  
 *//window.createUIComponents();* window.open();  
 frame.setContentPane(window.panelMain);  
  
 }  
  
 public void open() {  
  
 *//creazione cruciverba per l'utilizzo di funzioni dell'algoritmo1  
 //cruciverba1 = new ImplAlg1Cruciverba(panelMain, matriceSchemaCruciverba, parolaIniziale, posizioneRigaIniziale, posizioneColonnaIniziale, dizionarioInput,orientamento);  
  
 //creazione cruciverba per l'utilizzo di funzioni dell'algoritmo2  
 //cruciverba1=new ImplAlg2Cruciverba(panelMain, matriceSchemaCruciverba, parolaIniziale, posizioneRigaIniziale, posizioneColonnaIniziale, dizionarioInput, orientamento);  
  
 //creazione cruciverba per l'utilizzo di funzioni dell'algoritmo4  
 cruciverba1* = new ImplAlg4Cruciverba\_AI(panelMain,*matriceSchemaCruciverba*, *parolaIniziale*, *posizioneRigaIniziale*, *posizioneColonnaIniziale*, *dizionarioInput*,*orientamento*);  
  
 listListaParole.setListData(*dizionarioInput*.toArray());  
  
 }  
  
  
 private void createUIComponents() {  
 *// creo tutti gli elementi grafici dell'interfaccia utente per il cruciverba escluse tutte le caselle bianche e nere che creo in futuro* panelMain = new JPanel();  
 GroupLayout layoutGUI = new GroupLayout(panelMain);  
 panelMain.setLayout(layoutGUI);  
 layoutGUI.setAutoCreateGaps(true);  
 layoutGUI.setAutoCreateContainerGaps(true);  
  
 labelCruciverba = new JLabel("Cruciverba " + *matriceSchemaCruciverba*.length + "X" + *matriceSchemaCruciverba*[0].length);  
 labelCruciverba.setBounds(20, 20, 400, 20);  
  
 buttonRisolviCruciverba = new JButton("RisolviCruciverba");  
 buttonRisolviCruciverba.setBounds(20, 40, 200, 20);  
 buttonCercaParola = new JButton("CercaParola");  
 buttonCercaParola.setBounds(240, 40, 200, 20);  
  
 listListaParole=new JList();  
 listListaParole.setBounds(20, 80, 200,200);  
  
 scrollPaneListaParole = new JScrollPane(listListaParole);  
 scrollPaneListaParole.setPreferredSize(new Dimension(300,200));  
 scrollPaneListaParole.setBounds(20,80,800,800);  
  
 panelListaParole = new JPanel();  
 BorderLayout groupLayoutListaParole = new BorderLayout();  
 panelListaParole.setLayout(groupLayoutListaParole);  
 panelListaParole.add(scrollPaneListaParole);  
 panelListaParole.setBounds(20,80,200,200);  
  
 labelParolaInserita = new JLabel("Ultima parola inserita");  
 labelParolaInserita.setBounds(20, 300, 400, 20);  
  
 textFieldParolaInserita=new JTextField();  
 textFieldParolaInserita.setSize(200, 20);  
 textFieldParolaInserita.setLocation(20, 320);  
  
  
 panelMain.add(labelCruciverba);  
 panelMain.add(buttonRisolviCruciverba);  
 panelMain.add(buttonCercaParola);  
 panelMain.add(panelListaParole);  
 panelMain.add(labelParolaInserita);  
 panelMain.add(textFieldParolaInserita);  
 panelMain.revalidate();  
  
 }  
  
 *//funzione per importazione schema del cruciverba e inizializzazione variabile matriceSchemaCruciverba* private static void importaSchemaCruciverba(File f){  
 try {  
 Scanner s = new Scanner(f);  
 if (s == null) {  
 System.*out*.println("Scanner non creato");  
 } else {  
  
 if (s.hasNextInt()) {  
 int nRighe = Integer.*valueOf*(s.next());  
  
 if (s.hasNextInt()) {  
 int nColonne = Integer.*valueOf*(s.next());  
  
 *//creo matrice di caratteri che utilizzero per creare lo schema del cruciverba* char[][] matriceCruciverba = new char[nRighe][nColonne];  
 for(int i=0; i<nRighe; i++){  
 if(s.hasNextLine()){  
 s.nextLine();  
 for(int j=0; j<nColonne; j++){  
 if(s.hasNext()){  
 matriceCruciverba[i][j]=s.next().charAt(0);  
 }  
 }  
  
 }  
 }  
 *matriceSchemaCruciverba* = matriceCruciverba;  
  
 } else {  
 System.*out*.println("Numero di colonne dello schema non inserito");  
 throw new InputMismatchException();  
 }  
 } else {  
 System.*out*.println("Numero di righe dello schema non inserito");  
 throw new InputMismatchException();  
 }  
  
 s.close();  
 }  
 }catch(FileNotFoundException e){  
 System.*out*.println("File not found exception");  
 System.*exit*(-1);  
 }catch(InputMismatchException e){  
 System.*exit*(-1);  
 }  
 }  
  
 *//funzione per importazione delle parole da inserire all'interno dello schema all'interno del dizionario* private static void importaDizionario(File f){  
 try {  
 Scanner s = new Scanner(f);  
 if (s == null) {  
 System.*out*.println("Scanner non creato");  
 } else {  
 *//imposto il delimitatore per poter analizzare la prima riga contenente la parola da inserire per prima nello schema  
 // con la sua posizione e il suo orientamento  
 //s.useDelimiter(",");* if (s.hasNext()) {  
 *parolaIniziale* = s.next();  
 if (s.hasNextInt()) {  
 *posizioneRigaIniziale* = Integer.*valueOf*(s.next());  
 if (s.hasNextInt()) {  
 *posizioneColonnaIniziale* = Integer.*valueOf*(s.next());  
 if (s.hasNext()) {  
 *orientamento* = s.next().charAt(0);  
 s.nextLine();  
 *//inserimento delle parole nella lista di parole disponibili* while (s.hasNextLine()) {  
 String riga = s.nextLine();  
 *dizionarioInput*.add(riga);  
 }  
 } else {  
 System.*out*.println("Nessuna orientamento inserito");  
 throw new InputMismatchException();  
 }  
 } else {  
 System.*out*.println("Nessuna posizione colonna iniziale inserita");  
 throw new InputMismatchException();  
 }  
 } else {  
 System.*out*.println("Nessuna posizione riga iniziale inserita");  
 throw new InputMismatchException();  
 }  
 } else {  
 System.*out*.println("Nessuna parola iniziale inserita");  
 throw new InputMismatchException();  
 }  
  
 s.close();  
 }  
 }catch(FileNotFoundException e){  
 System.*out*.println("File not found exception");  
 System.*exit*(-1);  
 }catch(InputMismatchException e){  
 System.*exit*(-1);  
 }  
 }  
}

Classe Schema:

**package** com.cruciverbapackage;  
  
*//Coffaro\_Davide\_mat556603\_Progetto ESP cruciverba***import** javax.swing.\*;  
**import** java.util.ArrayList;  
  
**public class** Schema {  
 **private** ArrayList<Parola> **paroleSchema**;  
 **private** ArrayList<Casella> **caselleSchema**;  
  
  
 *//costruttore che prende la matrice in ingresso e crea lo schema aggiungendo la prima parola* **public** Schema(JPanel panel, **char** matrice[][], String parolaIniziale, Posizione posizioneParolaIniziale, **char** orientamentoInput) {  
 *//inizializzazione variabili classe* **paroleSchema** = **new** ArrayList<Parola>();  
 **caselleSchema** = **new** ArrayList<Casella>();  
 ArrayList<Casella> caselleParola = **new** ArrayList<Casella>();  
 Casella casellaAttuale;  
 StringBuilder creazioneParola = **new** StringBuilder();  
 Posizione posizioneIniziale = **new** Posizione();  
 Posizione posizioneAttuale = **new** Posizione();  
 Parola parolaCreata;  
 **int** lunghezzaParola = 0;  
 **char** carattere = **'.'**;  
 **char** casellaNera = **'\*'**;  
 **boolean** primaLettera = **true**;  
  
 *//ricerca parole orizzontali* **for** (**int** i = 0; i < matrice.**length**; i++) {  
 **for** (**int** j = 0; j < matrice[0].**length**; j++) {  
 *//inizializzazione posizione casella* posizioneAttuale.setRiga(i);  
 posizioneAttuale.setColonna(j);  
  
 *//ricerca della prima lettera della parola che verrà inserita nello schema* **if** (matrice[i][j] == carattere && primaLettera) {  
 creazioneParola.append(**' '**);  
 primaLettera = **false**;  
 posizioneIniziale.setRiga(i);  
 posizioneIniziale.setColonna(j);  
 lunghezzaParola++;  
  
 *//inizializzazione casella* casellaAttuale = **new** Casella(panel, posizioneAttuale, matrice[i][j], **false**);  
  
 **caselleSchema**.add(casellaAttuale);  
 caselleParola.add(casellaAttuale);  
 } **else if** (matrice[i][j] == carattere && (!primaLettera)) { *//ricerca lettere successive della parola che verrà inserita nello schema* creazioneParola.append(**' '**);  
 lunghezzaParola++;  
  
 *//inizializzazione casella* casellaAttuale = **new** Casella(panel, posizioneAttuale, matrice[i][j], **false**);  
  
 **caselleSchema**.add(casellaAttuale);  
 caselleParola.add(casellaAttuale);  
 } **else if** (matrice[i][j] == casellaNera) { *//ricerca delle caselle nere dello schema* **if** (lunghezzaParola >= 2) { *//lunghezza minima di una parola raggiunta, la inserisco nello schema così come la casella nera e ripristino i valori iniziali* **caselleSchema**.add(**new** Casella(panel, posizioneAttuale, matrice[i][j], **true**));  
 parolaCreata = **new** Parola(creazioneParola.toString(), posizioneIniziale, **'O'**, lunghezzaParola, caselleParola);  
 **paroleSchema**.add(parolaCreata);  
 creazioneParola = **new** StringBuilder();  
 posizioneIniziale.ripristinaPosizione();  
 lunghezzaParola = 0;  
 primaLettera = **true**;  
 caselleParola = **new** ArrayList<Casella>();  
 } **else** { *// lunghezza minima di una parola non raggiunta, inserisco solo la casella nera e ripristino i valori iniziali* **caselleSchema**.add(**new** Casella(panel, posizioneAttuale, matrice[i][j], **true**));  
 creazioneParola = **new** StringBuilder();  
 posizioneIniziale.ripristinaPosizione();  
 lunghezzaParola = 0;  
 primaLettera = **true**;  
 caselleParola = **new** ArrayList<Casella>();  
 }  
  
 }  
 }  
 **if** (lunghezzaParola >= 2) { *//raggiunta la fine della riga, inserisco la parola nello schema se ha la lunghezza minima altrimenti no* parolaCreata = **new** Parola(creazioneParola.toString(), posizioneIniziale, **'O'**, lunghezzaParola, caselleParola);  
 **paroleSchema**.add(parolaCreata);  
 creazioneParola = **new** StringBuilder();  
 posizioneIniziale.ripristinaPosizione();  
 lunghezzaParola = 0;  
 primaLettera = **true**;  
 caselleParola = **new** ArrayList<Casella>();  
 } **else** {  
 creazioneParola = **new** StringBuilder();  
 posizioneIniziale.ripristinaPosizione();  
 lunghezzaParola = 0;  
 primaLettera = **true**;  
 caselleParola = **new** ArrayList<Casella>();  
 }  
 }  
  
 *//ricerca parole verticali, stessa procedura di quelle orizzontali con l'aggiunta del controllo per le caselle già presenti* **for** (**int** j = 0; j < matrice[0].**length**; j++) {  
 **for** (**int** i = 0; i < matrice.**length**; i++) {  
 *//ricerca della prima lettera della parola che verrà inserita nello schema* **if** (matrice[i][j] == carattere && primaLettera) {  
 creazioneParola.append(**' '**);  
 primaLettera = **false**;  
 posizioneIniziale.setRiga(i);  
 posizioneIniziale.setColonna(j);  
 lunghezzaParola++;  
  
 *//cerco la casella già esistente relativa alla riga i, colonna j per poi inserirla nella parola ed averla  
 // collegata alle caselle dello schema, così come alle parole in orizzontale* cercaCasella(caselleParola, i, j);  
  
 } **else if** (matrice[i][j] == carattere && (!primaLettera)) {  
 creazioneParola.append(**' '**);  
 lunghezzaParola++;  
 cercaCasella(caselleParola, i, j);  
 } **else if** (matrice[i][j] == casellaNera) {  
 **if** (lunghezzaParola >= 2) {  
 parolaCreata = **new** Parola(creazioneParola.toString(), posizioneIniziale, **'V'**, lunghezzaParola, caselleParola);  
 **paroleSchema**.add(parolaCreata);  
 creazioneParola = **new** StringBuilder();  
 posizioneIniziale.ripristinaPosizione();  
 lunghezzaParola = 0;  
 primaLettera = **true**;  
 caselleParola = **new** ArrayList<Casella>();  
 } **else** {  
 creazioneParola = **new** StringBuilder();  
 posizioneIniziale.ripristinaPosizione();  
 lunghezzaParola = 0;  
 primaLettera = **true**;  
 caselleParola = **new** ArrayList<Casella>();  
 }  
 }  
 }  
 **if** (lunghezzaParola >= 2) {  
 parolaCreata = **new** Parola(creazioneParola.toString(), posizioneIniziale, **'V'**, lunghezzaParola, caselleParola);  
 **paroleSchema**.add(parolaCreata);  
 creazioneParola = **new** StringBuilder();  
 posizioneIniziale.ripristinaPosizione();  
 lunghezzaParola = 0;  
 primaLettera = **true**;  
 caselleParola = **new** ArrayList<Casella>();  
 } **else** {  
 creazioneParola = **new** StringBuilder();  
 posizioneIniziale.ripristinaPosizione();  
 lunghezzaParola = 0;  
 primaLettera = **true**;  
 caselleParola = **new** ArrayList<Casella>();  
 }  
 }  
 *//aggiorna schema con la prima parola iniziale data in input* aggiornaSchema(parolaIniziale, posizioneParolaIniziale, orientamentoInput);  
 }  
  
 *//metodo costruttore schema a partire da uno schema già esistente* **public** Schema(Schema s){  
 **this**.**paroleSchema**=s.getParoleSchema();  
 **this**.**caselleSchema**=s.getCaselleSchema();  
 }  
  
 **public** ArrayList<Parola> getParoleSchema() { **return new** ArrayList<Parola>(**paroleSchema**); }  
  
 **public** ArrayList<Casella> getCaselleSchema(){ **return new** ArrayList<Casella>(**caselleSchema**); }  
  
 *// inserisce la parola in input nella parola dello schema con la stessa posizione e orientamento* **public void** aggiornaSchema(String parola, Posizione posizioneParola, **char** orientamentoInput) {  
 Parola p = **new** Parola(parola, posizioneParola, orientamentoInput, parola.length());  
 **for** (Parola parolaSchema : **paroleSchema**) {  
 **if** (parolaSchema.confrontaCaselle(p)) {  
 parolaSchema.setParola(parola);  
 parolaSchema.setLunghezza(parola.length());  
 parolaSchema.aggiornaCaselleParola();  
 *//paroleSchema.set(paroleSchema.indexOf(parolaSchema), p);* **break**;  
 }  
 }  
   
 *//dopo aver aggiornato una parola dello schema devo fare in modo di aggiornare le parole dello schema che hanno lettere collegate  
 // alla parola appena aggiornata* **for**(Parola parolaSchema : **paroleSchema**){  
 parolaSchema.aggiornaParola();  
 }  
  
 }  
  
 *//cerca all'interno delle caselle dello schema e se già presente una casella con posizione riga, colonna allora la  
 // assegna alle caselleParola senza creare una nuova casella apposita* **public void** cercaCasella(ArrayList<Casella> caselleParola, **int** riga, **int** colonna) {  
 **for** (Casella c : **caselleSchema**) {  
 **if** (c.confrontaPosizione(riga, colonna)) {  
 caselleParola.add(c);  
 }  
 }  
 }  
  
 *//cerca le parole dello schema di lunghezza n non ancora completate* **public** ArrayList<Parola> ricercaLunghezzaParole(**int** n){  
 ArrayList<Parola> paroleLunghezzaN = **new** ArrayList<Parola>();  
 **for** (Parola p : **paroleSchema**){  
 **int** lunghezzaParola=p.getLunghezza();  
 **int** lettereInserite=p.getLettereInserite();  
 **if** (lunghezzaParola==n && lettereInserite<lunghezzaParola){  
 paroleLunghezzaN.add(p);  
 }  
 }  
 **return** paroleLunghezzaN;  
 }  
  
 *//cerca la lunghezza massima tra le parole dello schema* **public int** cercaLunghezzaParolaMax(){  
 **int** lunghezzaMax=0;  
 **for** (Parola p : **paroleSchema**){  
 **int** lunghezzaCorrente=p.getLunghezza();  
 **if** (lunghezzaCorrente > lunghezzaMax){  
 lunghezzaMax=lunghezzaCorrente;  
 }  
 }  
 **return** lunghezzaMax;  
 }  
  
}

Classe Parola:

package com.cruciverbapackage;  
  
*//Coffaro\_Davide\_mat556603\_Progetto ESP cruciverba*import javax.swing.\*;  
import java.util.ArrayList;  
  
public class Parola {  
 private String parola;  
 private ArrayList<Casella> caselleParola;  
 private Posizione posizioneParola;  
 private int lunghezza;  
 private int lettereInserite;  
 private char orientamento;  
  
 *//metodo costruttore crea una parola con le relative informazioni, senza caselleParola collegate* public Parola(String parola, Posizione posizioneIniziale, char orientamento, int lunghezza) {  
 try {  
 *//controllo parola* if (parola.length() > 0) {  
 this.parola = parola;  
 } else {  
 throw new Exception("Parola non corretta.");  
 }  
  
 *//controllo posizione* if (posizioneIniziale.getRiga() >= 0 && posizioneIniziale.getColonna() >= 0) {  
 posizioneParola = new Posizione(posizioneIniziale);  
 } else {  
 throw new Exception("Posizione indicata non corretta.");  
 }  
  
 *//controllo orientamento* if (orientamento == 'V' || orientamento == 'O') {  
 this.orientamento = orientamento;  
 } else {  
 throw new Exception("Orientamento (verticale o orizzontale) non corretto.");  
 }  
  
 *//controllo lunghezza* if (lunghezza > 1) {  
 this.lunghezza = lunghezza;  
 } else {  
 throw new Exception("Lunghezza non corretta. Vengono inserite le parole con almeno 2 lettere");  
 }  
  
 } catch (Exception ex) {  
 JOptionPane.*showMessageDialog*(null, ex.toString(), "Errore", JOptionPane.*ERROR\_MESSAGE*);  
 }  
  
 }  
  
 *//metodo costruttore crea una parola con le relative informazioni con le caselleParola collegate* public Parola(String parola, Posizione posizioneIniziale, char orientamento, int lunghezza, ArrayList<Casella> caselleParolaInput) {  
 try {  
  
 *//inizializzo le caselle della parola con le caselle dello schema create in precedenza* if (caselleParolaInput != null) {  
 caselleParola = caselleParolaInput;  
 }  
 *//controllo parola* if (parola.length() > 0) {  
 this.parola = parola;  
 } else {  
 throw new Exception("Parola non corretta.");  
 }  
  
 *//controllo posizione* if (posizioneIniziale.getRiga() >= 0 && posizioneIniziale.getColonna() >= 0) {  
 posizioneParola = new Posizione(posizioneIniziale);  
 } else {  
 throw new Exception("Posizione indicata non corretta.");  
 }  
  
 *//controllo orientamento* if (orientamento == 'V' || orientamento == 'O') {  
 this.orientamento = orientamento;  
 } else {  
 throw new Exception("Orientamento (verticale o orizzontale) non corretto.");  
 }  
  
 *//controllo lunghezza* if (lunghezza > 1) {  
 this.lunghezza = lunghezza;  
 } else {  
 throw new Exception("Lunghezza non corretta. Vengono inserite le parole con almeno 2 lettere");  
 }  
  
 } catch (Exception ex) {  
 JOptionPane.*showMessageDialog*(null, ex.toString(), "Errore", JOptionPane.*ERROR\_MESSAGE*);  
 }  
  
 }  
  
 *//metodo costruttore crea una parola a partire da una passata in input* public Parola(Parola p){  
 this.setParola(p.getParola());  
 this.setCaselleParola(p.getCaselleParola());  
 this.setPosizioneParola(p.getPosizioneParola());  
 this.setLunghezza(p.getLunghezza());  
 this.setLettereInserite(p.getLettereInserite());  
 this.setOrientamento(p.getOrientamento());  
 }  
  
 public ArrayList<Casella> getCaselleParola() {  
 return new ArrayList<Casella>(caselleParola);  
 }  
  
 public void setCaselleParola(ArrayList<Casella> caselleParola) {  
 this.caselleParola = caselleParola;  
 }  
  
 public String getParola() {  
 return parola;  
 }  
  
 public void setParola(String parola) {  
 this.parola = parola;  
 }  
  
 public Posizione getPosizioneParola() {  
 return posizioneParola;  
 }  
  
 public void setPosizioneParola(Posizione posizioneParola) {  
 this.posizioneParola = posizioneParola;  
 }  
  
 public int getLunghezza() {  
 return lunghezza;  
 }  
  
 public void setLunghezza(int lunghezza) {  
 this.lunghezza = lunghezza;  
 }  
  
 public char getOrientamento() {  
 return orientamento;  
 }  
  
 public void setOrientamento(char orientamento) {  
 this.orientamento = orientamento;  
 }  
  
 public int getLettereInserite() {  
 return lettereInserite;  
 }  
  
 public void setLettereInserite(int lettereInserite) {  
 this.lettereInserite = lettereInserite;  
 }  
  
 *//controlla se le due parole corrispondono alla stessa casella nello schema, sia come orientamento che come inizio e lunghezza parola* public boolean confrontaCaselle(Parola p) {  
 if (this.orientamento == p.orientamento && posizioneParola.equals(p.getPosizioneParola()) && this.lunghezza == p.lunghezza) {  
 return true;  
 } else {  
 return false;  
 }  
  
 }  
  
 *//aggiorna il testo delle caselle con la stringa dentro parola* public void aggiornaCaselleParola(){  
 lettereInserite=0;  
 try{  
 if (lunghezza==caselleParola.size()){  
 for (int i=0; i<lunghezza; i++){  
 char currentChar = parola.charAt(i);  
 if (currentChar!='.'){  
 lettereInserite++;  
 }  
 caselleParola.get(i).aggiornaCarattere(currentChar);  
 }  
 }else{  
 throw new Exception ("Lunghezza parola diversa dalla lunghezza della parola nel cruciverba");  
 }  
 }  
 catch (Exception ex){  
 JOptionPane.*showMessageDialog*(null, ex.toString(), "Errore", JOptionPane.*ERROR\_MESSAGE*);  
 }  
 }  
  
 *//aggiorna la stringa dentro parola con il testo delle caselle, contando anche le lettere inserite* public boolean aggiornaParola(){  
 StringBuilder strParolaNuova=new StringBuilder();  
 boolean result=false;  
 lettereInserite=0;  
 try{  
 if (lunghezza==caselleParola.size()){  
 for (int i=0; i<lunghezza; i++){  
 char carattereCasella=caselleParola.get(i).getCarattereCasella();  
 if (carattereCasella!='.'){  
 lettereInserite++;  
 result=true;  
 }  
 strParolaNuova.append(carattereCasella);  
 }  
 parola=strParolaNuova.toString();  
 }else{  
 throw new Exception ("Lunghezza parola diversa dalla lunghezza della parola nel cruciverba");  
 }  
 return result;  
 }  
 catch (Exception ex){  
 JOptionPane.*showMessageDialog*(null, ex.toString(), "Errore", JOptionPane.*ERROR\_MESSAGE*);  
 return false;  
 }  
 }  
  
 *//controlla se una parola è stata completata* public boolean isComplete(){  
 if (this.lunghezza==this.lettereInserite){  
 return true;  
 }else{  
 return false;  
 }  
 }  
  
 *//controllo se esistono caselle uguali tra due parole* public boolean isLinked(Parola p){  
 for (Casella casellaParolaCorrente : caselleParola){  
 for (Casella casellaParolaDaConfrontare : p.getCaselleParola()){  
 if (casellaParolaCorrente.confrontaCaselle(casellaParolaDaConfrontare)){  
 return true;  
 }  
 }  
 }  
 return false;  
 }  
  
 *//Ripristino i valori iniziali della parola, quindi tutti i caratteri a "." e le lettere inserite=0* public void resetParola(){  
 StringBuilder strParolaNuova=new StringBuilder();  
 boolean result=false;  
 lettereInserite=0;  
 try{  
 if (lunghezza==caselleParola.size()){  
 for (int i=0; i<lunghezza; i++){  
 strParolaNuova.append('.');  
 }  
 lettereInserite=0;  
 parola=strParolaNuova.toString();  
 }else{  
 throw new Exception ("Lunghezza parola diversa dalla lunghezza della parola nel cruciverba");  
 }  
 }  
 catch (Exception ex){  
 JOptionPane.*showMessageDialog*(null, ex.toString(), "Errore", JOptionPane.*ERROR\_MESSAGE*);  
 }  
 }  
  
}

Classe Casella:

**package** com.cruciverbapackage;  
  
*//Coffaro\_Davide\_mat556603\_Progetto ESP cruciverba***import** javax.swing.\*;  
**import** java.awt.\*;  
  
**public class** Casella {  
 **private** Posizione **posizioneCasella**;  
 **private** JTextField **textFieldCasella**;  
 **private char carattereCasella**;  
 **private boolean casellaNera**;  
  
 *//costruttore casella con input il Jpanel dove verrà inserita la casella* **public** Casella(JPanel panel, Posizione posizioneInput, **char** carattereInput, **boolean** casellaNeraInput) {  
 **int** xIniziale = 240;  
 **int** yIniziale = 80;  
 **int** i, j = 0;  
 **try** {  
 **if** (posizioneInput != **null**) {  
 **posizioneCasella** = **new** Posizione(posizioneInput);  
 i = **posizioneCasella**.getRiga();  
 j = **posizioneCasella**.getColonna();  
 } **else** {  
 **throw new** Exception(**"Posizione non esistente"**);  
 }  
 **carattereCasella** = carattereInput;  
 **casellaNera** = casellaNeraInput;  
 **textFieldCasella** = **new** JTextField();  
 **textFieldCasella**.setSize(20, 20);  
 *//calcolo la posizione della casella a partire dalla posizione passata in input* **textFieldCasella**.setLocation(xIniziale + j \* 20, yIniziale + i \* 20);  
 **textFieldCasella**.setText(String.*valueOf*(**carattereCasella**));  
  
  
 *//imposto lo sfondo della casella nera e la disabilito* **if** (**casellaNera**) {  
 **textFieldCasella**.setBackground(**new** Color(0, 0, 0));  
 **textFieldCasella**.setEditable(**false**);  
 **textFieldCasella**.setFocusable(**false**);  
 }  
  
 *//aggancio il textField al pannello passato in input* panel.add(**textFieldCasella**);  
 } **catch** (Exception ex) {  
 JOptionPane.*showMessageDialog*(**null**, ex.toString(), **"Errore"**, JOptionPane.***ERROR\_MESSAGE***);  
 }  
 }  
  
 **public char** getCarattereCasella() {  
 **return carattereCasella**;  
 }  
  
 **public void** setCarattereCasella(**char** carattereCasella) {  
 **this**.**carattereCasella** = carattereCasella;  
 }  
  
 *//controlla ritornando true o false se la posizione della casella dell'oggetto corrente è uguale a quella passata in input* **public boolean** confrontaPosizione(**int** riga, **int** colonna) {  
 Posizione posizioneDaConfrontare = **new** Posizione(riga, colonna);  
 **return posizioneCasella**.equals(posizioneDaConfrontare);  
 }  
  
 *//procedura di confronto caselle, ritorna true se la casella in input è la stessa della casella dell'oggetto corrente* **public boolean** confrontaCaselle(Casella c){  
 **return posizioneCasella**.equals(c.**posizioneCasella**);  
 }  
  
 *//aggiorna il carattere dell'oggetto corrente e il testo del componente visualizzato a video* **public void** aggiornaCarattere(**char** carattereInput){  
 **carattereCasella**=carattereInput;  
 **textFieldCasella**.setText(String.*valueOf*(carattereInput));  
 }  
}

Classe Posizione:

**package** com.cruciverbapackage;  
  
*//Coffaro\_Davide\_mat556603\_Progetto ESP cruciverba***public class** Posizione {  
 **int riga**;  
 **int colonna**;  
  
 *//costruttore posizione senza valori in input, inserisco valori standard* **public** Posizione() {  
 **this**.**riga** = -1;  
 **this**.**colonna** = -1;  
 }  
  
 *//costruttore posizione a partire da valori in input* **public** Posizione(**int** rigaInput, **int** colonnaInput) {  
 **this**.**riga** = rigaInput;  
 **this**.**colonna** = colonnaInput;  
 }  
  
 *//costruttore posizione da un'altra posizione già esistente* **public** Posizione(Posizione posizioneInput) {  
 **this**.**riga** = posizioneInput.getRiga();  
 **this**.**colonna** = posizioneInput.getColonna();  
 }  
  
 **public int** getRiga() {  
 **return riga**;  
 }  
  
 **public void** setRiga(**int** riga) {  
 **this**.**riga** = riga;  
 }  
  
 **public int** getColonna() {  
 **return colonna**;  
 }  
  
 **public void** setColonna(**int** colonna) {  
 **this**.**colonna** = colonna;  
 }  
  
 *//ripristina valori di default* **public void** ripristinaPosizione() {  
 **this**.**riga** = -1;  
 **this**.**colonna** = -1;  
 }  
  
 *//controlla se la posizione passata in input corrisponde alla stessa riga e colonna dell'oggetto corrente* **public boolean** equals(Posizione posizioneDaConfrontare) {  
 **if** (**this**.**riga** == posizioneDaConfrontare.getRiga() && **this**.**colonna** == posizioneDaConfrontare.getColonna()) {  
 **return true**;  
 } **else** {  
 **return false**;  
 }  
 }  
}

Classe ImplementazioneCruciverba:

**package** com.cruciverbapackage;  
*//Coffaro\_Davide\_mat556603\_Progetto ESP cruciverba***import** javax.imageio.plugins.tiff.ExifTIFFTagSet;  
**import** javax.swing.\*;  
**import** java.awt.event.ActionEvent;  
**import** java.awt.event.ActionListener;  
**import** java.lang.reflect.Array;  
**import** java.util.ArrayList;  
  
**public class** ImplementazioneCruciverba {  
 **protected** Schema **schema\_originale**;  
 **protected** ArrayList<String> **dizionario**;  
 **protected boolean algResult**;  
 **protected boolean algExecuted**;  
 *//costruttore cruciverba con una struttura passata in input* **public** ImplementazioneCruciverba(JPanel panel, **char** matrice[][], String parolaIniziale, **int** posizioneRigaIniziale, **int** posizioneColonnaIniziale, ArrayList<String> dizionarioInput, **char** orientamento) {  
 **schema\_originale** = **new** Schema(panel, matrice, parolaIniziale, **new** Posizione(posizioneRigaIniziale, posizioneColonnaIniziale), orientamento);  
 **if** (dizionarioInput != **null** && dizionarioInput.size() != 0) {  
 **dizionario** = dizionarioInput;  
 }  
  
 }  
  
 **public** ArrayList<String> getDizionario() {  
 **return dizionario**;  
 }  
  
 **public boolean** isAlgResult() {  
 **return algResult**;  
 }  
  
 **public boolean** isAlgExecuted() {  
 **return algExecuted**;  
 }  
  
 *// inserisce la parola all'interno del cruciverba nella riga e nella colonna specificata e con quell'orientamento* **public void** aggiornaParola(String parola, **int** riga, **int** colonna, **char** orientamento) {  
 **schema\_originale**.aggiornaSchema(parola,**new** Posizione(riga,colonna),orientamento);  
 }  
  
 *// ricerca la prossima parola da inserire nel cruciverba* **public** String cercaParolaDaInserire(Parola casellaDaCompletare, ArrayList<String> dizionario){  
 **return ""**;  
 }  
  
 *//corrisponde ad un ciclo di risolviCruciverba (in cui poi viene lanciata la funzione cercaParolaDaInserire)  
 //inserisce una parola nello schema del cruciverba* **public** String inserisci1Parola(){ **return null**; }  
  
 *//chiama cercaParolaDaInserire finche lo schema non è completato, cioè isComplete=true* **public boolean** risolviCruciverba(){  
 **return true**;  
 }  
  
 *//cerco la parola all'interno della lista che ha più lettere già inserite, a parità di lettere già inserite prendo la prima che ho trovato* **public** Parola cercaParolaConPiuLettere(ArrayList<Parola> listaParole){  
 **int** maxLettereInserite=-1, contatoreLettereInserite=0;  
 Parola maxParolaLettereInserite=**null**;  
 **for** (Parola p : listaParole){  
 contatoreLettereInserite=p.getLettereInserite();  
 **if** (contatoreLettereInserite>maxLettereInserite){  
 maxLettereInserite=contatoreLettereInserite;  
 maxParolaLettereInserite=p;  
 }  
 }  
 **return** maxParolaLettereInserite;  
 }  
  
 *// controllo se cruciverba è finito o no, quindi non risultano altre parole del dizionario da inserire* **public boolean** isComplete() {  
 **if** (**dizionario**.size()==0){  
 **return true**;  
 }**else**{  
 **return false**;  
 }  
 }  
   
 *//aggiornamento dizionario con le parole dello schema COMPLETATE, in questo modo quelle che parole che si sono completate automaticamente inserendo  
 //altre parole nello schema vengono eliminate dal dizionario* **public void** aggiornaDizionario(){  
 ArrayList<Parola> paroleSchema = **schema\_originale**.getParoleSchema();  
 String parolaCorrente;  
 **for** (Parola p : paroleSchema){  
 parolaCorrente=p.getParola();  
 **if** (p.getLunghezza()==p.getLettereInserite() && **dizionario**.contains(parolaCorrente)){  
 **dizionario**.remove(parolaCorrente);  
 }  
 }  
 }  
}

Classe ImplAlg1Cruciverba:

**package** com.cruciverbapackage;  
  
*//Coffaro\_Davide\_mat556603\_Progetto ESP cruciverba***import** javax.swing.\*;  
**import** java.util.ArrayList;  
  
**public class** ImplAlg1Cruciverba **extends** ImplementazioneCruciverba{  
  
 *//costruttore cruciverba con una struttura passata in input che richiama semplicemente il costruttore della classe padre* **public** ImplAlg1Cruciverba(JPanel panel, **char** matrice[][], String parolaIniziale, **int** posizioneRigaIniziale, **int** posizioneColonnaIniziale, ArrayList<String> dizionarioInput, **char** orientamento) {  
 **super**(panel,matrice,parolaIniziale,posizioneRigaIniziale,posizioneColonnaIniziale,dizionarioInput, orientamento);  
 }  
  
 *// ricerca la prossima parola da inserire nel cruciverba con algoritmo1* **public** String cercaParolaDaInserire(Parola casellaDaCompletare, ArrayList<String> dizionario){  
 ArrayList<String> paroleDizionarioTrovate=**new** ArrayList<String>();  
 **int** i=0;  
 **boolean** parolaUguale;  
 String parolaDaCompletare=casellaDaCompletare.getParola();  
 **int** lunghezzaParolaDaCompletare = casellaDaCompletare.getLunghezza();  
 */\*ciclo tutte le parole del dizionario in ricerca di una o più parole che possono entrare nelle caselle a disposizione,  
 \* a seconda dei caratteri già inseriti,  
 \* se nessuna parola viene trovata si ritorna una stringa vuota  
 \* se ci sono più parole da poter inserire si ritorna anche qui una stringa vuota  
 \* se ce n'è solo una invece ritorno la stringa da inserire all'interno di queste caselle  
 \*/* **for** (String s : dizionario){  
 parolaUguale=**true**;  
 *//se la lunghezza delle caselle da completare è diversa da quelle della parola s del dizionario salto il ciclo che confronta i caratteri  
 // delle due parole per vedere se sono compatibili* i=0;  
 **if** (s.length()==lunghezzaParolaDaCompletare){  
 *//confronto carattere per carattere quelle dell'oggetto Parola corrente con quelle del dizionario* **while** (i<lunghezzaParolaDaCompletare && parolaUguale){  
 **char** carattere = parolaDaCompletare.charAt(i);  
 *//confronto i caratteri solo se è un carattere valido (diverso da '.' impostato inizialmente)* **if** (carattere!=**'.'**){  
 **if** (carattere!=s.charAt(i)){  
 parolaUguale=**false**;  
 }  
 }  
 i++;  
  
 }  
 **if** (parolaUguale){  
 paroleDizionarioTrovate.add(s);  
  
 }  
 }  
 }  
 **if** (paroleDizionarioTrovate.size()==1){  
 **return** paroleDizionarioTrovate.get(0);  
 }**else**{  
 **return ""**;  
 }  
 }  
  
 *//corrisponde ad un ciclo di risolviCruciverba (in cui poi viene lanciata la funzione cercaParolaDaInserire)  
 //inserisce una parola nello schema del cruciverba* **public** String inserisci1Parola(){  
 **boolean** trovataParola=**false**;  
 String parolaDaInserire=**null**;  
 **int** cicliEseguiti=0,cicliMax=100,c;  
 *//analizzo le parole dello schema e prendo la lunghezza massima* **int** lunghezzaMax=**schema\_originale**.cercaLunghezzaParolaMax();  
 **int** numeroParoleLunghezzaC=0,numeroParoleLunghezzaCInserite=0;  
 ArrayList<Boolean> trovato = **new** ArrayList<Boolean>();  
  
 ArrayList<Parola> ricercaParole;  
  
 *//se già eseguito l'algoritmo ritorno null* **if** (isAlgExecuted()){  
 **return null**;  
 }  
 *//imposto tutto l'arrayList trovato a false* **for** (**int** i=0; i<=lunghezzaMax;i++){  
 trovato.add(**false**);  
 }  
  
 *//ciclo finchè tutto l'arrayList trovato è uguale a true oppure fino ad arrivare a un'iterazione>cicliMax oppure se non ho trovato  
 //nessuna parola al ciclo precedente* **while**(daTrovare(trovato) && cicliEseguiti<cicliMax && !(trovataParola)){  
 cicliEseguiti++;  
 c=0;  
 *//ciclo da 0 fino alla lunghezza massima delle parole nello schema oppure esco se non ho trovato nessuna parola al ciclo precedente* **while**(c<=lunghezzaMax && !(trovataParola)){  
 *//se ho già trovato tutte le parole con lunghezza c mi fermo e passo al numero c successivo* **if** (!(trovato.get(c))){  
 ricercaParole=**schema\_originale**.ricercaLunghezzaParole(c);  
 numeroParoleLunghezzaC=ricercaParole.size();  
 numeroParoleLunghezzaCInserite=0;  
 *//ciclo le parole di lunghezza c finchè la lista non è vuota oppure esco quando non ho trovato una parola al ciclo precedente* **while**(ricercaParole.size()>0 && !(trovataParola)){  
 Parola casellaDaCompletare=cercaParolaConPiuLettere(ricercaParole);  
 ricercaParole.remove(casellaDaCompletare);  
 *//chiamo la procedura di ricerca parola da inserire dell'algoritmo1* parolaDaInserire=cercaParolaDaInserire(casellaDaCompletare,**dizionario**);  
 **if** (!( parolaDaInserire.equals(**""**))){  
 trovataParola=**true**;  
 casellaDaCompletare.setParola(parolaDaInserire);  
 *//procedura per aggiornare lo schema con la parola trovata* aggiornaParola(casellaDaCompletare.getParola(),casellaDaCompletare.getPosizioneParola().getRiga()  
 , casellaDaCompletare.getPosizioneParola().getColonna(),casellaDaCompletare.getOrientamento());  
 numeroParoleLunghezzaCInserite++;  
  
 *//aggiorno il dizionario togliendo la parola che è stata inserita nello schema  
 // + eventuali parole che si sono autocompletate inserendo una parola nello schema* aggiornaDizionario();  
 }  
  
 }  
 **if** (numeroParoleLunghezzaC==numeroParoleLunghezzaCInserite){  
 trovato.set(c,**true**);  
 }  
 }  
  
 *//incremento il numero di caselle di cui voglio cercare le parole da inserire* c++;  
 }  
  
 }  
  
 *//ritorno la parola se è stata trovata, altrimenti ritorno null ma lanciando prima la procedura di RisolviCruciverba  
 //per vedere se è stato completato correttamente o no* **if** (trovataParola){  
 **return** parolaDaInserire;  
 }**else**{  
 risolviCruciverba();  
 **return null**;  
 }  
  
  
 }  
  
 *//risoluzione cruciverba attraverso l'utilizzo dell'algoritmo 1 e ritorno se è stato completato o no* **public boolean** risolviCruciverba(){  
  
 *//se è gia stato eseguito una volta non ripeto l'esecuzione ma ritorno il risultato ottenuto in precedenza* **if**(**algExecuted**){  
 **return algResult**;  
 }**else** {  
  
 **int** cicliEseguiti = 0, cicliMax = 100, c;  
 *//analizzo le parole dello schema e prendo la lunghezza massima* **int** lunghezzaMax = **schema\_originale**.cercaLunghezzaParolaMax();  
 **int** numeroParoleLunghezzaC = 0, numeroParoleLunghezzaCInserite = 0;  
 ArrayList<Boolean> trovato = **new** ArrayList<Boolean>();  
  
 ArrayList<Parola> ricercaParole;  
  
 *//imposto tutto l'arrayList trovato a false* **for** (**int** i = 0; i <= lunghezzaMax; i++) {  
 trovato.add(**false**);  
 }  
  
 *//ciclo finchè tutto l'arrayList trovato è uguale a true oppure fino ad arrivare a un'iterazione>cicliMax* **while** (daTrovare(trovato) && cicliEseguiti < cicliMax) {  
 cicliEseguiti++;  
 c = 0;  
 *//ciclo da 0 fino alla lunghezza massima delle parole nello schema* **while** (c <= lunghezzaMax) {  
 *//se ho già trovato tutte le parole con lunghezza c mi fermo e passo al numero c successivo* **if** (!(trovato.get(c))) {  
 ricercaParole = **schema\_originale**.ricercaLunghezzaParole(c);  
 numeroParoleLunghezzaC = ricercaParole.size();  
 numeroParoleLunghezzaCInserite = 0;  
 *//ciclo le parole di lunghezza c finchè la lista non è vuota* **while** (ricercaParole.size() > 0) {  
 Parola casellaDaCompletare = cercaParolaConPiuLettere(ricercaParole);  
 ricercaParole.remove(casellaDaCompletare);  
 *//chiamo la procedura di ricerca parola da inserire dell'algoritmo1* String parolaDaInserire = cercaParolaDaInserire(casellaDaCompletare, **dizionario**);  
 **if** (!(parolaDaInserire.equals(**""**))) {  
 casellaDaCompletare.setParola(parolaDaInserire);  
  
 *//procedura per aggiornare lo schema con la parola trovata* aggiornaParola(casellaDaCompletare.getParola(), casellaDaCompletare.getPosizioneParola().getRiga()  
 , casellaDaCompletare.getPosizioneParola().getColonna(), casellaDaCompletare.getOrientamento());  
 numeroParoleLunghezzaCInserite++;  
  
 *//aggiorno il dizionario togliendo la parola che è stata inserita nello schema  
 // + eventuali parole che si sono autocompletate inserendo una parola nello schema* aggiornaDizionario();  
 }  
  
 }  
 **if** (numeroParoleLunghezzaC == numeroParoleLunghezzaCInserite) {  
 trovato.set(c, **true**);  
 }  
 }  
  
 *//incremento il numero di caselle di cui voglio cercare le parole da inserire* c++;  
 }  
  
 }  
  
 *//aggiorno la variabile per sapere che ho eseguito una volta l'algoritmo di RisolviCruciverba* **algExecuted**=**true**;  
 *//faccio un controllo se ho completato tutte le parole dello schema, aggiorno la variabile che contiene il risultato del cruciverba  
 // e lo ritorno* **if** (!(daTrovare(trovato))){  
 **algResult**=**true**;  
 **return algResult**;  
 }**else**{  
 **algResult**=**false**;  
 **return algResult**;  
 }  
 }  
  
 }  
  
 *//controllo se sono presenti valori a false dentro l'arrayList trovato, in quel caso significa che ancora non ho trovato tutte  
 //le parole di lunghezza i* **public boolean** daTrovare(ArrayList<Boolean> trovato){  
 **int** i=0;  
 **try** {  
 **if** (trovato!=**null**){  
 **while**(i<trovato.size()){  
 **if** (trovato.get(i)){  
 i++;  
 }**else**{  
 **return true**;  
 }  
 }  
 }**else**{  
 **throw new** NullPointerException(**"Trovato è null"**);  
 }  
  
 } **catch** (NullPointerException ex) {  
 JOptionPane.*showMessageDialog*(**null**, ex.toString(), **"Errore"**, JOptionPane.***ERROR\_MESSAGE***);  
 System.*exit*(101);  
 }  
 **return false**;  
  
 }  
   
 *//cerco la parola all'interno della lista che ha più lettere già inserite, a parità di lettere già inserite prendo la prima che ho trovato* **public** Parola cercaParolaConPiuLettere(ArrayList<Parola> listaParole){  
 **int** maxLettereInserite=-1, contatoreLettereInserite=0;  
 Parola maxParolaLettereInserite=**null**;  
 **for** (Parola p : listaParole){  
 contatoreLettereInserite=p.getLettereInserite();  
 **if** (contatoreLettereInserite>maxLettereInserite){  
 maxLettereInserite=contatoreLettereInserite;  
 maxParolaLettereInserite=p;  
 }  
 }  
 **return** maxParolaLettereInserite;  
 }  
  
}

Classe ImplAlg2Cruciverba:

**package** com.cruciverbapackage;  
  
*//Coffaro\_Davide\_mat556603\_Progetto ESP cruciverba***import** javax.swing.\*;  
**import** java.util.ArrayList;  
  
**public class** ImplAlg2Cruciverba **extends** ImplementazioneCruciverba {  
  
 *//costruttore cruciverba con una struttura passata in input che richiama il costruttore del parole* **public** ImplAlg2Cruciverba(JPanel panel, **char** matrice[][], String parolaIniziale, **int** posizioneRigaIniziale, **int** posizioneColonnaIniziale, ArrayList<String> dizionarioInput, **char** orientamento) {  
 **super**(panel,matrice,parolaIniziale,posizioneRigaIniziale,posizioneColonnaIniziale,dizionarioInput, orientamento);  
 }  
  
 *//ricerca la prossima parola da inserire nel cruciverba* **public** String cercaParolaDaInserire(Parola casellaDaCompletare, ArrayList<String> dizionario){  
 ArrayList<String> paroleDizionarioTrovate=**new** ArrayList<String>();  
 **int** i=0;  
 **boolean** parolaUguale;  
 String parolaDaCompletare=casellaDaCompletare.getParola();  
 **int** lunghezzaParolaDaCompletare = casellaDaCompletare.getLunghezza();  
 */\*ciclo tutte le parole del dizionario in ricerca di una o più parole che possono entrare nelle caselle a disposizione,  
 \* a seconda dei caratteri già inseriti,  
 \* se nessuna parola viene trovata si ritorna una stringa vuota  
 \* se ci sono più parole da poter inserire si ritorna anche qui una stringa vuota  
 \* se ce n'è solo una invece ritorno la stringa da inserire all'interno di queste caselle  
 \*/* **for** (String s : dizionario){  
 parolaUguale=**true**;  
 *//se la lunghezza delle caselle da completare è diversa da quelle della parola s del dizionario salto il ciclo che confronta i caratteri  
 // delle due parole per vedere se sono compatibili* i=0;  
 **if** (s.length()==lunghezzaParolaDaCompletare){  
 *//confronto i caratteri della parola da completare con quelle delle parole nel dizionario, se non è presente  
 //nessun vincolo violato allora inserisco la parola del dizionario nella lista di parola da inserire* **while** (i<lunghezzaParolaDaCompletare && parolaUguale){  
 **char** carattere = parolaDaCompletare.charAt(i);  
 **if** (carattere!=**'.'**){  
 **if** (carattere!=s.charAt(i)){  
 parolaUguale=**false**;  
 }  
 }  
 i++;  
  
 }  
 **if** (parolaUguale){  
 paroleDizionarioTrovate.add(s);  
  
 }  
 }  
 }  
 *//se ho trovato solo una parola da inserire la ritorno alla funzione chiamante* **if** (paroleDizionarioTrovate.size()==1){  
 **return** paroleDizionarioTrovate.get(0);  
 }**else**{  
 **return ""**;  
 }  
 }  
  
 *//corrisponde ad un ciclo di risolviCruciverba (in cui poi viene lanciata la funzione cercaParolaDaInserire)  
 //inserisce una parola nello schema del cruciverba* **public** String inserisci1Parola(){  
 **boolean** trovataParola=**false**;  
 String parolaDaInserire=**null**;  
  
 **int** cicliEseguiti = 0, cicliMax = 100, c, lunghezzaMax = **schema\_originale**.cercaLunghezzaParolaMax();  
 **int** numeroParoleLunghezzaC = 0, numeroParoleLunghezzaCInserite = 0;  
  
 ArrayList<ArrayList<Parola>> listaParoleLunghezzaC = **new** ArrayList<ArrayList<Parola>>();  
  
 *//se già eseguito l'algoritmo ritorno null* **if** (isAlgExecuted()){  
 **return null**;  
 }  
 *//inserisco all'interno della lista delle parole di lunghezza i le parole di lunghezza i* **for** (**int** i = 0; i < lunghezzaMax; i++) {  
 ArrayList<Parola> paroleLunghezzaC;  
 paroleLunghezzaC = **schema\_originale**.ricercaLunghezzaParole(i + 1);  
 **if** (paroleLunghezzaC.size() > 0) {  
 listaParoleLunghezzaC.add(i, paroleLunghezzaC);  
 } **else** {  
 listaParoleLunghezzaC.add(**new** ArrayList<Parola>());  
 }  
 }  
 *//ciclo finchè il cruciverba non è completo, fino ad aver fatto n>cicliMax iterazioni e se non trovo una parola da inserire* **while** (!isComplete() && cicliEseguiti < cicliMax && !(trovataParola)) {  
 cicliEseguiti++;  
 c = 0;  
 **while** (c < lunghezzaMax && !(trovataParola)) {  
 *//prendo le parole di lunghezza=c su cui lavorerò* ArrayList<Parola> paroleLunghezzaC = listaParoleLunghezzaC.get(c);  
 **int** i=0;  
 **while** ( i < paroleLunghezzaC.size() && !(trovataParola)) {  
 Parola casellaDaCompletare = paroleLunghezzaC.get(i);  
 *//prima di cercare la parolaDaInserire faccio un controllo se la parola nella lista paroleLunghezzaC è già completa,  
 // se si la elimino dalla lista* **if** (casellaDaCompletare.getLunghezza() == casellaDaCompletare.getLettereInserite()) {  
 paroleLunghezzaC.remove(casellaDaCompletare);  
 } **else** {  
 parolaDaInserire = cercaParolaDaInserire(casellaDaCompletare, **dizionario**);  
 **if** (!(parolaDaInserire.equals(**""**))) {  
 *//è stata trovata una parola da inserire nello schema, rimuovo quindi la casellaDaCompletare dalla lista di parole di  
 // lunghezza c ancora da inserire* trovataParola=**true**;  
 paroleLunghezzaC.remove(casellaDaCompletare);  
  
 casellaDaCompletare.setParola(parolaDaInserire);  
 *//aggiorno lo schema con la nuova parola trovata* aggiornaParola(casellaDaCompletare.getParola(), casellaDaCompletare.getPosizioneParola().getRiga()  
 , casellaDaCompletare.getPosizioneParola().getColonna(), casellaDaCompletare.getOrientamento());  
  
 *//aggiorno il dizionario togliendo la parola che è stata inserita nello schema  
 // + eventuali parole che si sono autocompletate inserendo una parola nello schema* aggiornaDizionario();  
 }**else**{  
 *//incremento solo se la parola non era già completata o se non è stato inserito niente nello schema* i++;  
 }  
 }  
 }  
 c++;  
 }  
 }  
  
 *//controllo se il cruciverba è stato completato o meno* **if** (trovataParola) {  
 **return** parolaDaInserire;  
 } **else** {  
 risolviCruciverba();  
 **return null**;  
 }  
  
 }  
  
 *// risoluzione cruciverba attraverso l'utilizzo dell'algoritmo 2 ritorno true se il cruciverba è stato completato, altrimenti false* **public boolean** risolviCruciverba() {  
 **int** cicliEseguiti = 0, cicliMax = 100, c, lunghezzaMax = **schema\_originale**.cercaLunghezzaParolaMax();  
 **int** numeroParoleLunghezzaC = 0, numeroParoleLunghezzaCInserite = 0;  
 ArrayList<ArrayList<Parola>> listaParoleLunghezzaC = **new** ArrayList<ArrayList<Parola>>();  
  
 *//se ho già eseguito l'algoritmo di risoluzione ritorno il risultato salvato* **if**(**algExecuted**){  
 **return algResult**;  
 }**else** {  
 **if** (isComplete()) {  
 **algExecuted**=**true**;  
 **algResult** = **true**;  
 **return true**;  
 } **else** {  
 *//inserisco all'interno della lista delle parole di lunghezza i le parole di lunghezza i* **for** (**int** i = 0; i < lunghezzaMax; i++) {  
 ArrayList<Parola> paroleLunghezzaC;  
 paroleLunghezzaC = **schema\_originale**.ricercaLunghezzaParole(i + 1);  
 **if** (paroleLunghezzaC.size() > 0) {  
 listaParoleLunghezzaC.add(i, paroleLunghezzaC);  
 } **else** {  
 listaParoleLunghezzaC.add(**new** ArrayList<Parola>());  
 }  
 }  
 *//ciclo finchè il cruciverba non è completo, fino ad aver fatto n>cicliMax iterazioni* **while** (!isComplete() && cicliEseguiti < cicliMax) {  
 cicliEseguiti++;  
 c = 0;  
 **while** (c < lunghezzaMax) {  
 *//prendo le parole di lunghezza=c su cui lavorerò* ArrayList<Parola> paroleLunghezzaC = listaParoleLunghezzaC.get(c);  
 **int** i = 0;  
 **while** (i < paroleLunghezzaC.size()) {  
 Parola casellaDaCompletare = paroleLunghezzaC.get(i);  
 *//prima di cercare la parolaDaInserire faccio un controllo se la parola nella lista paroleLunghezzaC è già completa,  
 // se si la elimino dalla lista* **if** (casellaDaCompletare.getLunghezza() == casellaDaCompletare.getLettereInserite()) {  
 paroleLunghezzaC.remove(casellaDaCompletare);  
 } **else** {  
 String parolaDainserire = cercaParolaDaInserire(casellaDaCompletare, **dizionario**);  
 **if** (!(parolaDainserire.equals(**""**))) {  
 *//è stata trovata una parola da inserire nello schema, rimuovo quindi la casellaDaCompletare dalla lista di parole di  
 // lunghezza c ancora da inserire* paroleLunghezzaC.remove(casellaDaCompletare);  
  
 casellaDaCompletare.setParola(parolaDainserire);  
 *//aggiorno lo schema con la nuova parola trovata* aggiornaParola(casellaDaCompletare.getParola(), casellaDaCompletare.getPosizioneParola().getRiga()  
 , casellaDaCompletare.getPosizioneParola().getColonna(), casellaDaCompletare.getOrientamento());  
  
 *//aggiorno il dizionario togliendo la parola che è stata inserita nello schema  
 // + eventuali parole che si sono autocompletate inserendo una parola nello schema* aggiornaDizionario();  
 } **else** {  
 *//incremento solo se la parola non era già completata o se non è stato inserito niente nello schema* i++;  
 }  
 }  
 }  
 c++;  
 }  
 }  
  
 *//setto la variabile di esecuzione algoritmo* **algExecuted** = **true**;  
 *//controllo se il cruciverba è stato completato o meno e setto la variabile risultato* **if** (isComplete()) {  
 **algResult** = **true**;  
 **return true**;  
 } **else** {  
 **algResult** = **false**;  
 **return false**;  
 }  
 }  
 }  
 }  
  
}

Classe ImplAlg4\_AI:

package com.cruciverbapackage;  
  
*//Coffaro\_Davide\_mat556603\_Progetto ESP cruciverba*import javax.swing.\*;  
import java.lang.reflect.Array;  
import java.util.ArrayList;  
  
public class ImplAlg4Cruciverba\_AI extends ImplementazioneCruciverba{  
  
 *//creo inner class SizeException che estende Exception, perché viene utilizzata solo all'interno di questa implementazione* public class SizeException extends Exception{  
 public SizeException(){  
 super();  
 }  
 public SizeException(String s){  
 super(s);  
 }  
  
 }  
 private CSP constraintsSolver;  
 private ArrayList<Parola> listSolution;  
  
 *//costruttore cruciverba con una struttura passata in input che richiama il costruttore del padre e inizializza la variabile di tipo CPS* public ImplAlg4Cruciverba\_AI(JPanel panel, char matrice[][], String parolaIniziale, int posizioneRigaIniziale, int posizioneColonnaIniziale, ArrayList<String> dizionarioInput, char orientamento) {  
 super(panel,matrice,parolaIniziale,posizioneRigaIniziale,posizioneColonnaIniziale,dizionarioInput, orientamento);  
 constraintsSolver= new CSP(schema\_originale,dizionario);  
 }  
  
 *//corrisponde ad un ciclo di risolviCruciverba (in cui poi viene lanciata la funzione cercaParolaDaInserire)  
 //inserisce una parola nello schema del cruciverba* public String inserisci1Parola(){  
 int i=0;  
 Parola p=null;  
 *//controllo se l'algoritmo di risoluzione era già stato eseguito* if (!(constraintsSolver.isCSPExecuted())){  
  
 *//se non era stato eseguito lo eseguo e salvo il risultato nella variabile risultato di CPS, nella listSolution sarà contenuto  
 // l'ordine di inserimento delle parole nello schema* constraintsSolver.setCSPResult(backtrackSearch(constraintsSolver));  
  
 *//se risultato dell'algoritmo è true prendo un elemento di listSolution e lo inserisco nello schema del cruciverba* if (constraintsSolver.isCSPResult()){  
 if(listSolution.size()>0){  
 p=listSolution.get(i);  
 schema\_originale.aggiornaSchema(p.getParola(),p.getPosizioneParola(),p.getOrientamento());  
 listSolution.remove(i);  
 }  
 *//aggiorno le parole disponibili nel dizionario dopo l'inserimento della nuova parola nel cruciverba* aggiornaDizionario();  
 }  
 }else{  
 *//in questo ramo non rieseguo l'algoritmo e se il risultato dell'algoritmo è true prendo un elemento di listSolution e lo inserisco  
 // nello schema del cruciverba* if(constraintsSolver.isCSPResult()) {  
 if(listSolution.size()>0){  
 p=listSolution.get(i);  
 schema\_originale.aggiornaSchema(p.getParola(),p.getPosizioneParola(),p.getOrientamento());  
 listSolution.remove(i);  
 }  
 *//aggiorno la lista delle parole disponibili nel dizionario dopo l'inserimento dell'ultima parola* aggiornaDizionario();  
 }  
  
  
 }  
  
 algResult=constraintsSolver.isCSPResult();  
 try {  
 if (listSolution.size() == 0) {  
 if (p == null) {  
 return null;  
 } else {  
 return p.getParola();  
 }  
 } else {  
 return p.getParola();  
 }  
 }catch(NullPointerException e){  
 System.*out*.println("List solution è null");  
 return null;  
 }  
 }  
  
 *//implementazione algoritmo 4 con AI utilizzo CSP  
 //CSP:  
 //variabili = paroleSchema  
 //domini = x domini ognuno relativo alle parole di x lettere  
 //vincoli = relativi alle variabili, ad esempio la variabile in posizione 2^ riga - 3^ colonna in orizzontale  
 // ha la lettera 'i' nella quarta casella  
 //backtracking cronologico = se non riesco a completare lo schema del cruciverba con l'attuale assegnamento faccio passi indietro  
 // e provo ad inserire un altro valore nella variabile a cui avevo assegnato un valore errato* public boolean risolviCruciverba(){  
 int i=0;  
 if (!(constraintsSolver.isCSPExecuted())){  
  
 constraintsSolver.setCSPResult(backtrackSearch(constraintsSolver));  
  
 if (constraintsSolver.isCSPResult()){  
 *//inserisco tutte le parole nella listSolution all'interno dello schema del cruciverba, una alla volta* while(listSolution.size()>0){  
 Parola p=listSolution.get(i);  
 schema\_originale.aggiornaSchema(p.getParola(),p.getPosizioneParola(),p.getOrientamento());  
 listSolution.remove(i);  
 }  
 *//aggiorno la lista di parole disponibili del dizionario dopo l'inserimento delle parole di listSolution* aggiornaDizionario();  
 }  
 algResult=constraintsSolver.isCSPResult();  
 return algResult;  
 }else{  
 *//se avevo già eseguito l'algoritmo inserisco solo le parole rimanenti nella listSolution all'interno dello schema del cruciverba* if(constraintsSolver.isCSPResult()) {  
 while (listSolution.size() > 0) {  
 Parola p=listSolution.get(i);  
 schema\_originale.aggiornaSchema(p.getParola(),p.getPosizioneParola(),p.getOrientamento());  
 listSolution.remove(i);  
 }  
 *//aggiorno la lista di parole disponibili dopo l'inserimento delle parole nel cruciverba* aggiornaDizionario();  
 }  
 algResult=constraintsSolver.isCSPResult();  
 return algResult;  
  
 }  
  
 }  
  
  
 *//lancio procedura per soluzione cruciverba con AI* private boolean backtrackSearch( CSP csp){  
 *//setto variabile di esecuzione algoritmo a true* csp.setCSPExecuted(true);  
 *//salvo lo stato dello schema attuale per poterlo ripristinare dopo l'esecuzione dell'algoritmo  
 //che conterrà altrimenti tutti i riferimenti modificati durante l'esecuzione dell'algoritmo* Schema oldSchema=new Schema(schema\_originale);  
 backtrack(new ArrayList<Parola>(), csp);  
 if (listSolution==null){  
 System.*out*.println("Soluzione non trovata");  
 return false;  
 }else {  
 if (listSolution.size() == constraintsSolver.getNumberVariables()) {  
 *//ripristino lo schema originale e anche i valori a prima dell'esecuzione dell'algoritmo nelle caselle delle parole* schema\_originale = oldSchema;  
 for (Parola p : schema\_originale.getParoleSchema()) {  
 p.aggiornaCaselleParola();  
 }  
 return true;  
 } else {  
 *//ripristino lo schema originale e anche i valori a prima dell'esecuzione dell'algoritmo nelle caselle delle parole* schema\_originale = oldSchema;  
 for (Parola p : schema\_originale.getParoleSchema()) {  
 p.aggiornaCaselleParola();  
 }  
 return false;  
 }  
 }  
 }  
  
 *//ricerca soluzione cruciverba con AI con variable=MRV (minimum remaining values)+euristica del grado, value=prossimo valore del dominio ancora  
 // non provato,  
 // inferenza con FC (forward checking), backtracking cronologico  
 // ritorno il valore null quando sono arrivato in fondo alla procedura, altrimenti ritorno la variabile che volevo utilizzare per fare il  
 // backtracking intelligente sulle variabili collegate ad essa, adesso non è utile perché utilizzo il backtracking cronologico* private boolean backtrack(ArrayList<Parola> assignment, CSP csp){  
 int countAssignment=assignment.size();  
 boolean varResult=false;  
 if (assignment.size()==csp.getNumberVariables()){  
 listSolution=assignment;  
 return true;  
 }  
 Variable var=selectUnassignedVariable(constraintsSolver);  
 if (var==null){  
 *//termino la procedura corrente  
 //nessuna variabile a cui assegnare un valore trovata* return false;  
 }else{  
 *//mantengo una copia della vecchia variabile nel caso in cui l'assegnamento corrente non è corretto* Variable oldVar = new Variable(var);  
  
 *//elimino dal dominio della variabile i possibili valori già assegnati a variabili precedenti (visto che  
 //quando ritorno da backtrace ripristino per la variabile il dominio iniziale  
 //var.updateDomain(assignment);* ArrayList<String> varDomain=orderDomainValues(var,assignment,csp);  
  
 *//scorro i valori del dominio prendendoli uno ad uno dal dominio* for(String s : varDomain){  
 var.setNewParola(s);  
 var.aggiornaCaselleParola();  
 var.setValueAssigned(true);  
 assignment.add(var.getValue());  
 countAssignment++;  
  
 if (inference(csp,var,s,assignment)){  
 *//chiamo di nuovo backtrack per trovare il prossimo assegnamento da fare* System.*out*.print(var.getValue().getPosizioneParola().getRiga());  
 System.*out*.print(var.getValue().getOrientamento());  
 System.*out*.print(var.getValue().getPosizioneParola().getColonna());  
 System.*out*.println(var.getValue().getParola());  
 varResult = backtrack(assignment,csp);  
 if (varResult){  
 return true;  
 }else{  
 *//controllo se sono stati assegnati valori per il numero di variabili dentro lo schema del cruciverba, se si il cruciverba è stato completato  
 //altrimenti ripristino i valori precedenti* if(assignment.size()==csp.getNumberVariables()){  
 listSolution=assignment;  
 return true;  
 }else {  
 *//operazioni di ripristino nel caso in cui l'inferenza non è andata a buon fine* countAssignment--;  
 assignment.remove(countAssignment);  
 var.restore(oldVar, constraintsSolver.searchDomain(var.getValue().getLunghezza()));  
 }  
 restoreLinkedSameLengthVariables(csp, var, assignment);  
 }  
 }else{  
 *//controllo se sono stati assegnati valori per il numero di variabili dentro lo schema del cruciverba, se si il cruciverba è stato completato  
 //altrimenti ripristino i valori precedenti* if(assignment.size()==csp.getNumberVariables()){  
 listSolution=assignment;  
 return true;  
 }else {  
 *//operazioni di ripristino nel caso in cui l'inferenza non è andata a buon fine* countAssignment--;  
 assignment.remove(countAssignment);  
 var.restore(oldVar, constraintsSolver.searchDomain(var.getValue().getLunghezza()));  
 }  
 }  
  
  
 }  
 }  
 return false;  
 }  
  
  
 *//seleziono la variabile per la ricerca di un valore da inserire seguendo la strategia di CSP* private Variable selectUnassignedVariable(CSP constraintsSolver) {  
 *//imposto il valore iniziale uguale al numero di parole da inserire nel cruciverba* int minValues = dizionario.size();  
 ArrayList<Variable> listCandidateVariables = null;  
  
 *//procedura di selezione variabili per minor valore dei domini* for (Variable v : constraintsSolver.getVariables()) {  
 int variableValues = v.getValuesNumber();  
 if (!(v.isValueAssigned())) {  
 if (variableValues < minValues) {  
 *//se il numero di valori del sottodominio per questa variabile è inferiore a quella precedente creo una nuova lista  
 // (la lista precedente contenente le variabili con numero valori dominio maggiore viene scartata)  
 // in cui inserisco la variabile corrente e aggiorno il numero valore sottodominio minimo* listCandidateVariables = new ArrayList<Variable>();  
 listCandidateVariables.add(v);  
 minValues = variableValues;  
 } else if (variableValues == minValues) {  
  
 *//controllo se non era ancora stata inizializzata la lista delle variabili candidate* if (listCandidateVariables == null) {  
 listCandidateVariables = new ArrayList<Variable>();  
 }  
  
 *//inserisco la variabile corrente nella lista contenente le variabili con la stessa percentuale di completamento* listCandidateVariables.add(v);  
 }  
 }  
 }  
  
 try{  
 if(listCandidateVariables.size()==0){  
 *//se la lista contiene zero elementi sollevo un'eccezione. Inner class creata dentro questa classe perché  
 // la utilizzo solo al suo interno* throw new ImplAlg4Cruciverba\_AI.SizeException("Non è stata trovata nessuna variabile candidata per l'inserimento di un nuovo valore.");  
  
 }else if(listCandidateVariables.size()==1){  
 *//se la lista delle variabili candidate contiene un solo elemento lo passo alla return della funzione* return listCandidateVariables.get(0);  
 }else{  
 *//la lista contiene più elementi, cerco quella variabile che vincola maggiormente le altre variabili (quella con più lettere)* int maxLetters=0;  
 Variable maxLettersVariable=null;  
 for (Variable v : listCandidateVariables){  
 int currentLetters=v.getNumberLetters();  
 if(currentLetters>maxLetters){  
 maxLetters=currentLetters;  
 maxLettersVariable=v;  
 }  
  
 }  
 if(maxLettersVariable!=null){  
 return maxLettersVariable;  
 }else{  
 throw new NullPointerException("Non è stata trovata una variabile più vincolante dentro la lista delle variabili candidate");  
 }  
 }  
 }catch (SizeException e){  
 System.*out*.println(e);  
 return null;  
 }catch (NullPointerException e){  
 System.*out*.println(e);  
 return null;  
 }  
 }  
  
 *//creo la lista di assegnamenti di valori del dominio alla variabile, ordinata in ordine di inserimento nel dominio* private ArrayList<String> orderDomainValues(Variable var, ArrayList<Parola> assignment, CSP csp){  
 *//TODO implementare soluzione per poter creare una lista ordinata di valori da quello meno vincolante a quello più vincolante  
 //per adesso ritorno semplicemente la lista dei valori del dominio possibili* return var.getListValuesDomain();  
  
 }  
  
 *//procedura che mi permette di:  
 // 1) togliere dal problema csp la variabile var perché gli è stato assegnato un valore  
 // 2) cercare le variabili collegate a var di cui dovrò modificare il dominio dovuto alla nuova stringa s assegnata a var  
 // 3) fare un controllo che i domini risultanti delle variabili collegate non siano vuoti:  
 // se lo sono ritorno false altrimenti true* private boolean inference(CSP csp, Variable var, String s, ArrayList<Parola> assignment){  
  
 ArrayList<Variable> listLinkedVariables=null;  
 ArrayList<Variable> listSameLengthVariables=null;  
 int counterLinkedVariables=0;  
 int counterSameLengthVariables=0;  
 boolean result=true;  
  
 *//trovare variabili collegate a var (cioè che condividono le stesse caselle) e ridurre il loro dominio* listLinkedVariables=csp.searchLinkedVariables(var);  
 while (listLinkedVariables!=null && counterLinkedVariables<listLinkedVariables.size() && result){  
 Variable currentVar=listLinkedVariables.get(counterLinkedVariables);  
 currentVar.setOldValue(currentVar.getValue());  
 if(currentVar.aggiornaParola()){  
 currentVar.restoreDomain(constraintsSolver.searchDomain(currentVar.getValue().getLunghezza()));  
 currentVar.updateDomain(assignment);  
 }  
 *//procedura di inference sulla variabile corrente, se va a buon fine proseguo altrimenti ripristino i valori e imposto la  
 // variabile result a false* if (!(currentVar.inferenceAfterUpdateParola())){  
 currentVar.ripristinaParola();  
 result=false;  
 }  
 counterLinkedVariables++;  
 }  
  
 *//guardo adesso tutte le variabili con la stessa lunghezza della variabile corrente a cui ho assegnato il valore e rimuovo il valore  
 // assegnato dal loro dominio (se presente)* listSameLengthVariables=csp.searchSameLengthVariables(var.getNumberLetters());  
 while(listSameLengthVariables!=null && counterSameLengthVariables<listSameLengthVariables.size() && result){  
 Variable currentVar=listSameLengthVariables.get(counterSameLengthVariables);  
 *//currentVar.setOldValue(currentVar.getValue());* if (!(currentVar.inferenceAfterAssignedValue(s))){  
 result=false;  
 }  
 counterSameLengthVariables++;  
 }  
  
  
 *//se il risultato dell'inferenza è false ripristino tutti i valori dei domini che avevo modificato durante la procedura sulle variabili collegate  
 //partendo dalla penultima variabile perché l'ultima su cui si era verificato l'errore di inferenza li ha già ripristinati (torno indietro di 2)* counterLinkedVariables=counterLinkedVariables-2;  
 while(!(result) && counterLinkedVariables>=0){  
 Variable currentVar=listLinkedVariables.get(counterLinkedVariables);  
 currentVar.ripristinaParola();  
 currentVar.restoreDomain(constraintsSolver.searchDomain(currentVar.getValue().getLunghezza()));  
 currentVar.updateDomain(assignment);  
 counterLinkedVariables--;  
 }  
 *//se il risultato dell'inferenza è false ripristino tutti i valori dei domini che avevo modificato durante la procedura sulle variabili con  
 // la stessa lunghezza  
 //partendo dalla penultima variabile perché l'ultima su cui si era verificato l'errore di inferenza li ha già ripristinati (torno indietro di 2)* counterSameLengthVariables=counterSameLengthVariables-2;  
 while(!(result) && counterSameLengthVariables>=0){  
 Variable currentVar=listSameLengthVariables.get(counterSameLengthVariables);  
 *//currentVar.ripristinaParola();* currentVar.restoreDomain(constraintsSolver.searchDomain(var.getValue().getLunghezza()));  
 currentVar.updateDomain(assignment);  
 counterSameLengthVariables--;  
 }  
 return result;  
 }  
  
 *//ripristino i valori precedenti delle variabili collegate a var e delle variabili della stessa lunghezza di var* public void restoreLinkedSameLengthVariables(CSP csp, Variable var, ArrayList<Parola> assignment){  
 ArrayList<Variable> listLinkedVariables=null;  
 ArrayList<Variable> listSameLengthVariables=null;  
 int counterLinkedVariables=0;  
 int counterSameLengthVariables=0;  
  
 *//trovare variabili collegate a var (cioè che condividono le stesse caselle)* listLinkedVariables=csp.searchLinkedVariables(var);  
  
 *//guardo adesso tutte le variabili con la stessa lunghezza della variabile corrente a cui ho assegnato il valore* listSameLengthVariables=csp.searchSameLengthVariables(var.getNumberLetters());  
  
  
 *//ripristino tutti i valori dei domini che avevo modificato durante la procedura sulle variabili collegate* while(counterLinkedVariables<listLinkedVariables.size()){  
 Variable currentVar=listLinkedVariables.get(counterLinkedVariables);  
 currentVar.aggiornaParola();  
 currentVar.restoreDomain(constraintsSolver.searchDomain(currentVar.getValue().getLunghezza()));  
 currentVar.updateDomain(assignment);  
 counterLinkedVariables++;  
 }  
  
 *//ripristino tutti i valori dei domini che avevo modificato durante la procedura sulle variabili con  
 // la stessa lunghezza* while(counterSameLengthVariables<listSameLengthVariables.size()){  
 Variable currentVar=listSameLengthVariables.get(counterSameLengthVariables);  
 *//currentVar.ripristinaParola();* currentVar.restoreDomain(constraintsSolver.searchDomain(var.getValue().getLunghezza()));  
 currentVar.updateDomain(assignment);  
 counterSameLengthVariables++;  
 }  
  
 }  
  
}

Classe CSP:

**package** com.cruciverbapackage;  
  
*//Coffaro\_Davide\_mat556603\_Progetto ESP cruciverba***import** java.util.ArrayList;  
  
  
*//classe CSP = costraintsSatisfactionProblem, utilizzo di algoritmo di AI per la ricerca della soluzione del cruciverba***public class** CSP {  
 **private** ArrayList<Parola> **listSolution**;  
 **private boolean CSPExecuted**;  
 **private boolean CSPResult**;  
 **private** ArrayList<Variable> **variables**;  
 **private** ArrayList<Domain> **domains**;  
  
 **public** ArrayList<Parola> getListSolution() {  
 **return new** ArrayList<Parola>(**listSolution**);  
 }  
  
 **public** ArrayList<Variable> getVariables() {  
 **return new** ArrayList<Variable>(**variables**);  
 }  
  
 **public void** setCSPExecuted(**boolean** CSPExecuted) {  
 **this**.**CSPExecuted** = CSPExecuted;  
 }  
  
 **public boolean** isCSPExecuted() {  
 **return CSPExecuted**;  
 }  
  
 **public void** setCSPResult(**boolean** CSPResult) {  
 **this**.**CSPResult** = CSPResult;  
 }  
  
 **public boolean** isCSPResult() {  
 **return CSPResult**;  
 }  
  
 **public** CSP(Schema s, ArrayList<String> d){  
 **listSolution**= **new** ArrayList<Parola>();  
 **CSPExecuted**=**false**;  
 **CSPResult**=**false**;  
 **variables**= **new** ArrayList<Variable>();  
 **domains**=**new** ArrayList<Domain>();  
 insertValuesInDomain(d);  
 insertValuesInVariables(s.getParoleSchema());  
 }  
  
 *//inserisce tutte le stringhe all'interno del dominio con lettere corrispondenti* **private void** insertValuesInDomain(ArrayList<String> dizionario){  
 Domain foundD=**null**;  
 **for** (String s : dizionario){  
 *//cerco dominio con elementi di lunghezza s.length()* foundD=searchDomain(s.length());  
  
 **if** (foundD==**null**){  
 *//significa che non esisteva già un dominio con valori della lunghezza stringa e quindi creo un nuovo dominio,  
 // ci inserisco la stringa e lo inserisco nella lista dei domini* foundD=**new** Domain(s,s.length());  
 **domains**.add(foundD);  
 }**else**{  
 *//significa che esisteva già un dominio con valori della lunghezza stringa e quindi inserisco in coda* foundD.add(s);  
 }  
 }  
 }  
  
 **public void** insertValuesInVariables(ArrayList<Parola> list){  
 Domain foundD=**null**;  
 **for** (Parola p : list){  
 **int** i= 0;  
 **if** (!(p.isComplete())){ *// se la parola non è già completa* foundD=searchDomain(p.getLunghezza());  
  
 *//se lettereInserite!=0 devo lanciare la procedura di inferenza sui domini delle variabili che creo* **if** (foundD!=**null**){  
 Variable v = **new** Variable(p,foundD);  
 **variables**.add(v);  
 *//se lettereInserite per questa parola è diverso da zero lancio anche la procedura di inferenza per ridurre  
 // i domini di questa variabile* **if** (p.getLettereInserite()>0){  
 v.inferenceAfterUpdateParola();  
 }  
 }**else**{  
 **throw new** NullPointerException(**"Non è stato trovato un dominio per questa variabile"**);  
 }  
 }  
  
 }  
 }  
  
 *// cerco il dominio contenente elementi di lunghezza l* **public** Domain searchDomain(**int** l){  
 Domain foundD=**null**;  
 **int** i=0;  
 **while** (i<**domains**.size() && foundD==**null**){ *//esco dal while se ho scorso tutto l'array dei domini o se ho trovato un elemento* Domain checkD=**domains**.get(i);  
 **if**(checkD.getLunghezzaParole()==l){  
 foundD=checkD;  
 }  
 i++;  
 }  
 **return** foundD;  
 }  
  
 *//ritorna le variabili collegata alla variabile passata in input var* **public** ArrayList<Variable> searchLinkedVariables(Variable var){  
 ArrayList<Variable> linkedVariables=**null**;  
  
 *//scorro le variabili dello schema alla ricerca di quelle collegata a quella in input* **for**(Variable searchVar : **variables**){  
 *//faccio un controllo se è già assegnato un valore, in questo modo evito di verificare il collegamento con una variabile  
 // alla quale ho già assegnato un valore (sia variabile corrente che altre variabili all'interno dello schema)* **if** (!(searchVar.isValueAssigned())) {  
 **if** (searchVar.isLinked(var)) {  
 **if** (linkedVariables==**null**){  
 linkedVariables=**new** ArrayList<Variable>();  
 }  
 linkedVariables.add(searchVar);  
 }  
 }  
 }  
 **return** linkedVariables;  
 }  
  
 *//ritorna le variabili con la stessa lunghezza della variabile passata in input var* **public** ArrayList<Variable> searchSameLengthVariables(**int** lengthToCompare){  
 ArrayList<Variable> sameLengthVariables=**null**;  
  
 *//scorro le variabili dello schema alla ricerca di quelle con la stessa lunghezza di quella in input* **for**(Variable searchVar : **variables**){  
 *//faccio un controllo se è già assegnato un valore, in questo modo evito di verificare il collegamento con una variabile  
 // alla quale ho già assegnato un valore (sia variabile corrente che altre variabili all'interno dello schema)* **if** (!(searchVar.isValueAssigned())) {  
 **if** (searchVar.getNumberLetters()==lengthToCompare) {  
 **if** (sameLengthVariables==**null**){  
 sameLengthVariables=**new** ArrayList<Variable>();  
 }  
 sameLengthVariables.add(searchVar);  
 }  
 }  
 }  
 **return** sameLengthVariables;  
 }  
  
 *//ritorno il numero di variabili totali* **public int** getNumberVariables(){  
 **return variables**.size();  
 }  
  
}

Classe Variable:

package com.cruciverbapackage;  
  
*//Coffaro\_Davide\_mat556603\_Progetto ESP cruciverba*import java.util.ArrayList;  
  
public class Variable {  
 private Parola value;  
 private Domain variableDomain;  
 private boolean valueAssigned;  
 private ArrayList<String> oldListValues;  
 private Parola oldValue;  
  
 *//costruttore in cui inizializzo la parola e il dominio della variabile + altre variabili per lo stato dell'oggetto* public Variable(Parola var, Domain d){  
 value=new Parola(var);  
 variableDomain=new Domain(d);  
 valueAssigned=false;  
 oldListValues=null;  
 oldValue=null;  
 }  
  
 *//costruisco una variabile da una già esistente* public Variable(Variable var){  
 this.value=var.getValue();  
 this.variableDomain=var.getVariableDomain();  
 this.valueAssigned=var.isValueAssigned();  
 this.oldListValues=var.getOldListValues();  
 this.oldValue=var.getOldValue();  
 }  
  
 public Domain getVariableDomain(){ return new Domain(this.variableDomain); }  
  
 public Parola getValue(){ return new Parola(this.value); }  
  
 public boolean isValueAssigned() {  
 return valueAssigned;  
 }  
  
 public void setValueAssigned(boolean valueAssigned) {  
 this.valueAssigned = valueAssigned;  
 }  
  
 public ArrayList<String> getOldListValues() {  
 if (oldListValues==null){  
 return null;  
 }else{  
 return new ArrayList<String>(oldListValues);  
 }  
 }  
  
 public Parola getOldValue() {  
 if (oldValue==null){  
 return null;  
 }else {  
 return new Parola(oldValue);  
 }  
 }  
  
 public void setOldValue(Parola oldValue) {  
 this.oldValue = oldValue;  
 }  
  
 *//procedura di inferenza sul dominio di questa variabile per rimuovere i valori non permessi dopo assegnazione di un valore ad un'altra variabile  
 //ritorna false se il dominio risultante dall'inferenza risulta essere vuoto* public boolean inferenceAfterAssignedValue(String s){  
 ArrayList<String> listValuesDomain=variableDomain.getListValues();  
  
 *//salvo una copia della lista valore del dominio per poterla ripristinare in caso di errore di inferenza (dominio vuoto)* oldListValues=new ArrayList<String>(listValuesDomain);  
 if (listValuesDomain.contains(s)){  
 listValuesDomain.remove(s);  
 }  
  
 *//se il dominio dopo aver fatto inferenza non è vuoto lo aggiorno per la variabile corrente e ritorno true  
 //altrimenti ritorno false senza aggiornare il dominio della variabile corrente* if(listValuesDomain.size()>0){  
 variableDomain.setListValues(listValuesDomain);  
 return true;  
 }else{  
 return false;  
 }  
 }  
  
 *//procedura di inferenza sul dominio di questa variabile per rimuovere i valori non permessi dopo aggiornamento parola  
 //ritorna false se il dominio risultante dall'inferenza risulta essere vuoto* public boolean inferenceAfterUpdateParola(){  
 ArrayList<String> listValuesDomain = variableDomain.getListValues();  
  
 *//salvo una copia della lista valore del dominio per poterla ripristinare in caso di errore di inferenza (dominio vuoto)* if (listValuesDomain!=null) {  
 oldListValues = new ArrayList<String>(listValuesDomain);  
 }  
 String parolaSchema = value.getParola();  
 for (int i=0; i<parolaSchema.length(); i++){  
 int j=0;  
 char checkChar=parolaSchema.charAt(i);  
  
 *//faccio la procedura di controllo carattere solo se è diverso dal carattere '.' che è quello preimpostato  
 // nelle caselle dello schema* if(checkChar!='.'){  
 while(j<listValuesDomain.size()){  
 String s = listValuesDomain.get(j);  
  
 *//se il carattere della parola è diverso da quello del valore del dominio, tolgo quel valore dal dominio di questa variabile  
 //altrimenti incremento il valore di j e continuo sul valore successivo del dominio* if(checkChar!=s.charAt(i)){  
 listValuesDomain.remove(j);  
 }else {  
 j++;  
 }  
  
 }  
 }  
  
 }  
  
 *//se il dominio dopo aver fatto inferenza non è vuoto lo aggiorno per la variabile corrente e ritorno true  
 //altrimenti ritorno false senza aggiornare il dominio della variabile corrente* if(listValuesDomain.size()>0){  
 variableDomain.setListValues(listValuesDomain);  
 return true;  
 }else{  
 return false;  
 }  
 }  
  
 *//ritorno il numero di lettere della variabile* public int getNumberLetters(){  
 return value.getLunghezza();  
 }  
  
 *//ritorna il numero di valori dentro il dominio di questa variabile* public int getValuesNumber(){  
 return variableDomain.getValuesNumber();  
 }  
  
 public String getValueDomain(int index){  
 return variableDomain.getValueDomain(index);  
 }  
  
 public ArrayList<String> getListValuesDomain(){  
 return variableDomain.getListValues();  
 }  
  
 *//setto value con una nuova parola* public void setNewParola(String s){  
 value.setParola(s);  
 }  
  
 public void aggiornaCaselleParola(){  
 value.aggiornaCaselleParola();  
 }  
  
 public boolean aggiornaParola(){  
 return value.aggiornaParola();  
 }  
  
 *//confronto le caselle della Parola relativa a questo oggetto con quella della variabile passata in input, se una sola  
 // casella coincide allora le due variabili sono collegate e ritorno true* public boolean isLinked(Variable var){  
 return value.isLinked(var.getValue());  
 }  
  
 *//ripristino i valori precedenti del dominio* public void restoreDomain(){  
 variableDomain.setListValues(oldListValues);  
 }  
  
 *//ripristino i valori predefiniti del dominio (quelli iniziali)* public void restoreDomain(Domain initialDomain){  
 if (!(initialDomain==null)){  
 variableDomain.setListValues(initialDomain.getListValues());  
 }  
 }  
  
 *//ripristino la parola precedente della variabile (prima dell'inferenza)* public void ripristinaParola(){  
 value=oldValue;  
 }  
  
 *//ripristino i valori della variabile corrente con quelli della variabile passata in input* public void restore(Variable oldVar, Domain initialDomain){  
 if (oldVar!=null) {  
 this.value.setParola(oldVar.getValue().getParola());  
 this.value.aggiornaCaselleParola();  
 this.variableDomain.setListValues(initialDomain.getListValues());  
 this.valueAssigned = oldVar.isValueAssigned();  
 this.oldListValues = oldVar.getOldListValues();  
 Parola restoreOldValue = oldVar.getOldValue();  
 if (!(restoreOldValue == null)) {  
 this.oldValue.setParola(restoreOldValue.getParola());  
 }  
 }  
  
 }  
  
 public void updateDomain(ArrayList<Parola> assignment){  
 for(int i=0; i<assignment.size(); i++){  
 String parolaAssignment = assignment.get(i).getParola();  
 if(this.getNumberLetters()==parolaAssignment.length()) {  
 if (this.variableDomain.getListValues().contains(parolaAssignment)) {  
 ArrayList<String> newListValuesDomain=this.getListValuesDomain();  
 newListValuesDomain.remove(parolaAssignment);  
 this.variableDomain.setListValues(newListValuesDomain);  
  
 }  
 }  
 }  
 }  
  
}

Classe Domain:

**package** com.cruciverbapackage;  
  
*//Coffaro\_Davide\_mat556603\_Progetto ESP cruciverba***import** java.util.ArrayList;  
  
*//classe dei domini contenente le parole del dizionario suddivise per lunghezza parole***class** Domain{  
 **private** ArrayList<String> **listValues**;  
 **private int length**;  
  
 *//costruttore con inizializzazione delle variabili dell'oggetto Domain e con inserimento della stringa nella lista di valori del dominio* **public** Domain(String s, **int** dim){  
 **listValues**=**new** ArrayList<String>();  
 **listValues**.add(s);  
 **length**=dim;  
 }  
  
 *//costruttore a partire da un altro oggetto di tipo dominio già esistente* **public** Domain(Domain d){  
 **listValues**=d.getListValues();  
 **length**=d.getLunghezzaParole();  
 }  
  
 *//ritorna la lista di valori inseriti in questo dominio* **public** ArrayList<String> getListValues(){  
 **return new** ArrayList<String>(**listValues**);  
 }  
  
 *//modifica la lista di valori inseriti in questo dominio* **public void** setListValues(ArrayList<String> listValues) {  
 **this**.**listValues** = listValues;  
 }  
  
 *//ritorna la lunghezza delle parole inserite in questo dominio* **public int** getLunghezzaParole(){  
 **return length**;  
 }  
  
 *//aggiunge una nuova parola nella lista di valori in questo dominio* **public void** add (String s){  
 **listValues**.add(s);  
 }  
  
 *//conta il numero di valori all'interno del dominio* **public int** getValuesNumber(){  
 **return listValues**.size();  
 }  
  
 *//restituisco la stringa alla posizione index della lista di valori (listValues)* **public** String getValueDomain(**int** index){  
 **return listValues**.get(index);  
 }  
   
}

Esempio file .txt utilizzato per la creazione dello schema e per le parole da inserire nello schema del cruciverba  
  
Esempio1

4 8  
. . . .\* . . .  
\* . . . . \* . .  
. . . . . . \* .  
. . \* . . . . .

CANE 0 0 O  
CAT  
NOIA  
ANCA  
NOE  
EIRE  
AR  
TOPO  
RO  
ABS  
ACERBO  
AP  
OI  
PA  
ESITO  
  
Esempio2

16 12  
. . . . . . . . . . \* .  
. \* . . . . . \* . \* . .  
. . . . . . \* . . . . .  
. . . . . \* . . . . . .  
. . . . \* . \* . . . . .  
. . . \* . . . . . . . \*  
. . \* . . . . . . . \* .  
. \* . . . . . . . . . .  
. . . . . . . . . . \* .  
. \* . . . . . . . \* . .  
\* . . . . . . . \* . . .  
. . . . . \* . \* . . . .  
. . . . . . \* . . . . .  
. . . . . \* . . . . . .  
. . \* . \* . . . . . \* .  
. \* . . . . . . . . . .

TERRORISTA 7 2 O  
CASABLANCA  
QUAT  
TRE  
BOSTRN  
DD  
CIAO  
TOPO  
INORGANICI  
TESIO  
TOCCIRE  
TORO  
STRAGRANDE  
TESEO  
CASABIANCA  
ALAIN  
FE  
FURIA  
ROSSO  
ALVARO  
ONLUS  
LEZIOSA  
BISTRO  
PIETA  
ESTERE  
DENTISTICO  
LASER  
PORO  
TIZIO  
OSTETRICA  
MOSA  
ANA  
MALESIA  
AN  
LEONINO  
BEN  
TE  
FARETTO  
RIO  
GARRANI  
WO  
LISI  
GARA  
MARGARINE  
INORGANICA  
ALATO  
FERRAMENTA  
CRANICO  
RA  
RAVERA  
TOCCARE  
OCARINA  
BOB  
LIONE  
ODORE  
ADAMI  
SCAPPATOIA  
ALIBI  
LORENA  
MASSA  
SAVONAROLA  
IRENE  
WESER  
BOSTON  
BE  
  
Esempio3

12 19  
. . \* . . . . . . . . . \* . . . . . .  
. . . . . . . . . . . . . \* . . . . \*  
. . . \* . \* . . . \* . \* . . \* . . . .  
. \* . . \* . \* . \* . \* . . . \* . . . .  
. . . . . . . . . . . . . . . . \* . .  
. . . . . . . . . . . \* . . . . . . .  
. . . . . . . . . . . . \* . . . . . \*  
. . . . . . . . . \* . . . . . \* . . .  
. . \* . \* . . . . . . . . \* . . . . .  
. . . . . . . \* . . . . . . \* . . . .  
. \* . . . . \* . . . . . . \* . . . . .  
\* . . . . \* . . . \* . . \* . . . . . \*

DONO 4 4 V  
BANALITA  
MIAO  
TETTO  
ISOLA  
LUCI  
IDA  
TONANTE  
EO  
NOMINE  
NAT  
SPIEGAMENTO  
TA  
OTRI  
ADATTE  
EOLO  
AGARICO  
IDIOMA  
PECORE  
GIARDINIPUBBLICI  
ESIGUI  
AI  
ELI  
MANICOTTO  
ITER  
ARRINGARE  
TI  
MARINOMARINI  
FIDEL  
INEDITI  
TOM  
NI  
BARATRO  
LARIO  
ULANI  
ATEO  
ON  
CALLIMACO  
NEO  
ISACCO  
TINI  
RINASCIMENTO  
LB  
ADRIA  
SL  
NC  
PIRAMIDE  
ALI  
ADA  
ILIO  
AU  
ECONOMICA  
ICE  
SUD  
AREA  
RAME  
MB  
CALAF  
BUE  
EMI  
ASTRONOMICA  
CRUMIRA  
OZI  
RINOMATA  
ANOMALIA  
IDIOTI  
LOS  
SS  
URNE  
ORTI  
RA  
PUBBLICAZIONI  
SE

**Bibliografia:**

* Artificial Intelligence, A Moderne Approach Third Edition, Cap.6 Constraints Satisfaction Problems
* IntelliJ Idea Online Documentation about JPanel, JLabel, JButton, JList, JScrollPane, JTextField:   
  1) <https://www.jetbrains.com/help/idea/designing-gui-major-steps.html>  
  2) <https://intellij-support.jetbrains.com/hc/en-us/community/posts/360003406579-Drawing-on-a-JPanel-of-a-form>  
  3) <https://intellij-support.jetbrains.com/hc/en-us/community/posts/206233659-GUI-Designer-Question-How-to-dynamically-set-JLabel-Text>  
  4) <https://intellij-support.jetbrains.com/hc/en-us/community/posts/206327529-Default-JButton-for-a-form>  
  5) <https://intellij-support.jetbrains.com/hc/en-us/community/posts/360000366919-Add-JScrollPane-to-GUI-Form>  
  6) <https://intellij-support.jetbrains.com/hc/en-us/community/posts/206926855-JTextField-in-UI-Designer>
* StackOverFlow for JList <https://stackoverflow.com/questions/30009226/add-item-to-jlist-in-intellij-idea>
* Introduction to Algorithms 3rd edition Cormen Leiserson Rivest Stein Cap.4 Divide and Conquer
* <https://github.com/coffa94/ESP_CruciverbaIntelliJ>
* Lista di parole italiane <https://github.com/napolux/paroleitaliane/tree/master/paroleitaliane>