ALGORITMI E STRUTTURE DATI

Docente: Francesco Camastra

PROGETTO: VOCABOLARIO

Capobianco Salvatore *matricola 0124/974*

Indice

Traccia

Risolvere entrambi i quesiti:

1. Costruire un vocabolario V utilizzando un **Hash Table** con il metodo della concatenazione, che abbia le seguenti funzioni:

    - Inserimento del termine
    - Cancellazione
    - Ricerca del termine

La ricerca in caso di fallimento deve restituire una lista delle parole più prossime, utilizzando un approccio basato sulla Distanza di editing. La distanza di editing (o di Levenshtein) misura il numero di operazioni (inserimento, cancellazione e correzione) che devono essere eseguite sulla tastiera per trasformare una stringa in un'altra. La distanza di Levenshtein tra due stringhe di caratteri S= (S_1 ,…,S_N) e T= (T_1 ,…, T_M) e definita nel modo seguente. Siano S_i e T_j i caratteri corrispondenti rispettivamente all' i-esimo carattere di S ed al j-esimo carattere di T allora la loro distanza di editing d[i,j] e data da:

$$d[i,j] = min \begin{cases} d[i-1,j]+1 \\ d[i,j-1]+1 \\ d[i-1,j-1]+(S_i \neq T_j) \end{cases} \quad (1)$$

$$d[0,0] = 0 \quad (2)$$

2. Costruire un vocabolario V, utilizzando un albero **RED BLACK** che abbia le stesse funzioni, dell' esercizio precedente. Nell' implementazione di entrambi i quesiti si faccia uso delle Classi Astratte.

Distanza di editing (o Levenshtein)

La Distanza di Levenshtein è la distanza tra due stringhe S1 ed S2, intesa come il numero minimo di operazioni elementari che occorrono per trasformare la stringa S1 nella stringa S2.
Queste operazioni elementari includono:

- cancellazione di un carattere
- sostituzione di un carattere con un altro
- inserimento di un carattere

L'algoritmo di Levenshtein consente di calcolare la distanza tra una stringa di lunghezza *n* e una stringa di lunghezza *m* in tempo *O(m*n)*. Si considerano due stringhe: x={$x_1,x_2,\dots,x_n$} e y={$y_1,y_2,\dots,y_m$}, costruiamo il vettore D(i,j)= distanza tra i prefissi $x_1,x_2,\dots,x_n$ e $y_1,y_2,\dots,y_m$. Il risultato cercato sarà: D(n,m) = distanza tra $x_1,x_2,\dots,x_n$ e $y_1,y_2,\dots,y_m$. Si hanno tre possibilità per calcolare D(i,j), noto D(k,l) per k < i e l < j :

- il carattere a_i va sostituito con il carattere b_j:

  D(i,j) = D(i - 1,j - 1) + t(a_i,b_j) ;

- il carattere a_i va cancellato:

  D(i,j) = D(i - 1,j) + 1 ;

- il carattere b_j va inserito:

  D(i,j) = D(i,j - 1) + 1 .

Si vuole ottenere il valore minimo. L'equazione di ricorrenza è:

$$d[i,j] = min \begin{cases} d[i-1,j]+1 \\ d[i,j-1]+1 \\ d[i-1,j-1]+(S_i \neq T_j) \end{cases} \quad (1)$$

$$d[0,0] = 0 \quad (2)$$

Considerate due stringhe *sorgente* e *destinazione,* la distanza di Levenshtein è il numero di operazioni elementari necessarie a trasformare la striga *sorgente* in quella *destinazione*.

Due stringhe uguali hanno distanza di Levenshtein pari a 0.

Esempio:

		m	a	r	i	a	n	o
	0	1	2	3	4	5	6	7
m	1	0	1	2	3	4	5	6
a	2	1	0	1	2	3	4	5
r	3	2	1	0	1	2	3	4
i	4	3	2	1	0	1	2	3
n	5	4	3	2	1	1	1	2
a	6	5	4	3	2	1	2	2
o	7	6	5	4	3	2	2	2

La matrice sopra rappresenta il calcolo che svolge l'algoritmo per arrivare al risultato finale, ovvero la distanza di editing, che è contenuta nella cella in basso a destra (D(n,m)); tutte le altre celle sono intermedie.

Le classi astratte

Una Classe Astratta è una classe, in cui sono definite una o più **funzioni virtuali pure**. Non è possibile istanziarne alcun oggetto, viene infatti, utilizzata solo come classe base. Le classi derivate, erediteranno le funzioni membro della classe astratta che, solo a questo punto, dovranno essere opportunamente definite.

Lo scopo dell'introduzione delle classi astratte è quello di fornire un'interfaccia di tipo generico, senza specificare l'implementazione di alcuni metodi, aumentando così l'ordine fra le classi e consentendo la definizione di una gerarchia di entità.

Ovviamente una classe astratta può anche possedere funzioni e dati membro completamente definiti. E' buona norma creare un puntatore ad una classe astratta.

Per l'implementazione del nostro vocabolario utilizzeremo la "ClasseAstrattaVocabolario.h", dove dichiareremo dei metodi **virtual**, ma saranno implementati solo nelle classi derivate. Saranno dunque dichiarati e posti uguali a 0. Queste funzioni servono solo da "schema di comportamento" per le classi derivate che ne definiranno a quel punto l'implementazione specifica, tali metodi sono detti **polimorfi**.

Hash Table

Un hash table è una struttura dati utilizzata per implementare insiemi dinamici, particolarmente efficace quando questi ultimi supportano solo le operazioni di dizionario, cioè **inserimento, cancellazione** e **ricerca.**

Con il **metodo hash**, un elemento con chiave k, è memorizzato nello slot h(k) della tabella. Si utilizza, cioè, una **funzione hash h( )** per calcolare lo slot della chiave; essa definisce una corrispondenza tra l'universo delle chiavi e le posizioni della tabella hash. In questo modo, si riduce l'intervallo degli indici che devono essere gestiti (rispetto alla tecnica ad indirizzamento diretto), e di conseguenza, lo spazio di memoria richiesto.

La prima caratteristica di una buona funzione hash è il soddisfacimento del **criterio di uniformità semplice**, ovvero ogni chiave viene associata ad uno degli m slot in maniera **equiprobabile**. Inoltre, una buona funzione hash, dovrebbe utilizzare tutte le cifre della chiave nella produzione del suo valore hash.

Nel vocabolario da implementare utilizzeremo **hashing per divisione**, dove la funzione hash è del tipo:

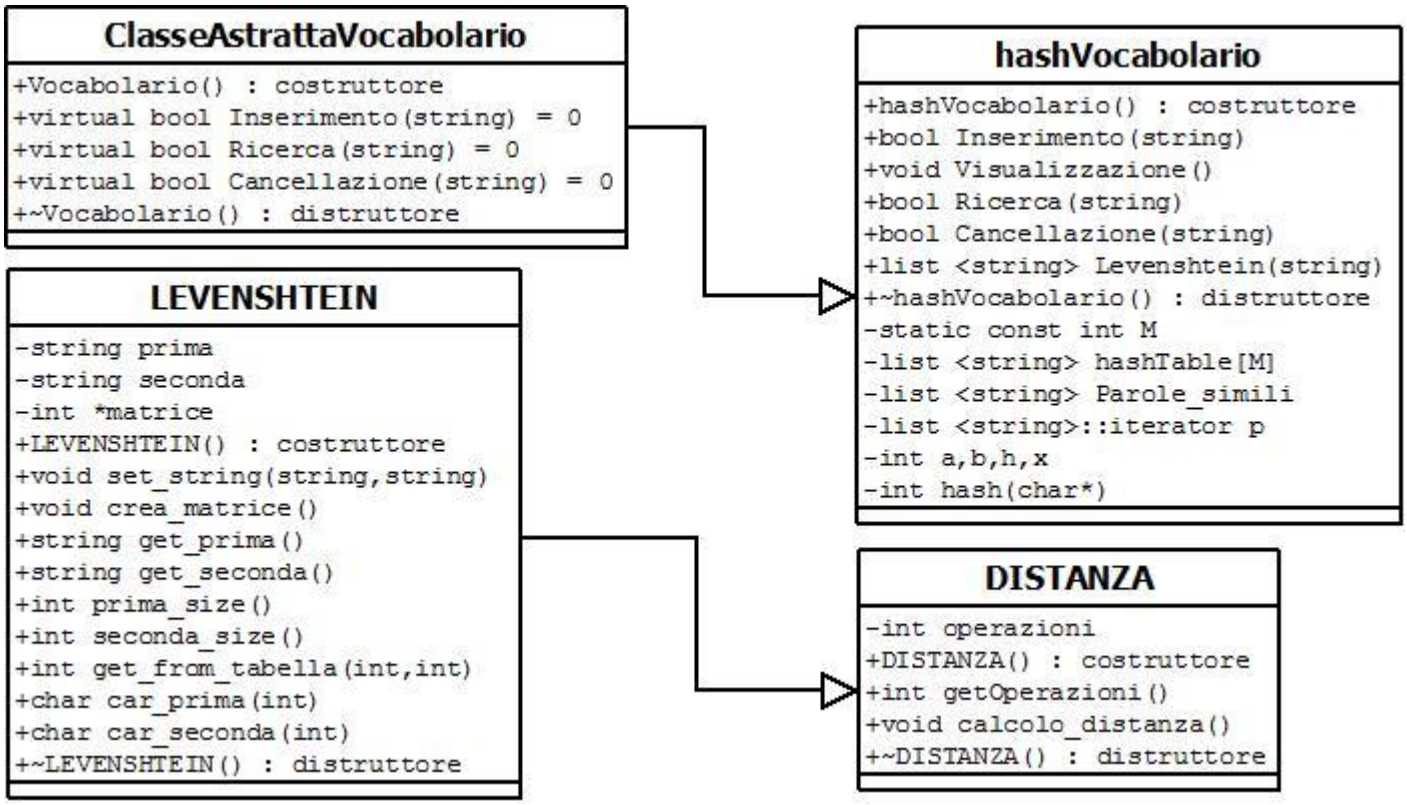
$h(k) = k \bmod m$

cioè, il valore hash prodotto è il resto della divisione della chiave k per il numero di slot m. Risulta essere un metodo veloce, ma occorre fare attenzione ai valori di m e in particolare evitarne alcuni, infatti, buoni valori di m sono i numeri primi non troppo vicini a potenze del 2, altrimenti l'operazione di modulo andrebbe a considerare solo i bit meno significativi della chiave.

Tuttavia, due chiavi distinte possono produrre il medesimo valore hash, e quindi, essere associate allo stesso slot. Questo fenomeno è detto **collisione.** Poiché l'universo delle chiavi è più grande dell'insieme delle posizioni, è impossibile evitare completamente le collisioni, anche generando una funzione hash casuale. Due sono i metodi per la risoluzione delle collisioni**: indirizzamento aperto** e **concatenamento**. Quest'ultimo sarà il metodo che utilizzeremo per la risoluzione delle collisioni nel vocabolario hash che andremo ad implementare; esso consiste nel porre tutti gli elementi collidenti, cioè quelli associati allo stesso slot, in una lista concatenata. Il j-simo slot della tabella può contenere, un puntatore alla testa della j-sima lista che contiene tutti gli elementi associati a j, oppure un puntatore nullo se non vi sono presenti elementi.

Diagramma delle classi

Le classi usate:



Implementazione:

- ClasseAstrattaVocabolario.h

```
#include <iostream>

#include <string>

#include <list>


using namespace std;


class Vocabolario

{

        public:


        Vocabolario (){};
```

```
        virtual bool Inserimento(string) = 0;
        virtual bool Ricerca(string) = 0;
        virtual bool Cancellazione(string) = 0;

        ~Vocabolario (){};
};
```

- Levenshstein.h

```
#include <iostream>
#include <string>
#include <vector>
#include <cstdio>
#include <algorithm>

using namespace std;

class LEVENSHTEIN
{
   private:
      string prima;
      string seconda;
      int *matrice;
   public:
      LEVENSHTEIN();

      void set_string(string s, string t);
      void crea_matrice();
      string get_prima();
      string get_seconda();
      int prima_size();
      int seconda_size();
      int get_from_tabella(int ,int);
      char car_prima(int);
```

```
        char car_seconda(int);


        ~LEVENSHTEIN();
};
// Costruttore.
LEVENSHTEIN::LEVENSHTEIN()
{
   prima=seconda="";
   matrice=NULL;
}

// Restituisce la prima stringa.
string LEVENSHTEIN::get_prima()
{
   return prima;
}

// Restituisce la seconda stringa.
string LEVENSHTEIN::get_seconda()
{
   return seconda;
}

// Restituisce la lunghezza della prima stringa.
int LEVENSHTEIN::prima_size()
{
   return prima.size();
}



// Restituisce la lunghezza della seconda stringa.
int LEVENSHTEIN::seconda_size()
```

```
{
   return seconda.size();
}

// Restituisce un elemento della tabella. L'accesso è per riga.
int LEVENSHTEIN::get_from_tabella(int i,int j)
{
   return *(matrice+i* (seconda.size()+1) +j);
}

// Restituisce un singolo carattere della prima stringa.
char LEVENSHTEIN::car_prima(int i)
{
   return prima[i];
}


// Restituisce un singolo carattere della seconda stringa.
char LEVENSHTEIN::car_seconda(int j)
{
   return seconda[j];
}

// Per inizializzare le stringhe.
void LEVENSHTEIN::set_string(string s, string t)
{
   prima=s; seconda=t;
   cout<<endl<<"Trasformazione di '"<<prima<<"' in '"<<seconda<<"':"<<endl;
}


// Allocazione dinamica della matrice.
```

```
void LEVENSHTEIN :: crea_matrice()
{
   int n = prima.size()+1;
   int m = seconda.size()+1;
   matrice = new int[(n)*(m)];
   if(matrice == NULL)
   {   cout << "Impossibile allocare memoria"<<endl;  exit(0); }

   for(int i=0;i<n;i++)
      *(matrice+i*m+0)=i;

   for(int j=0;j<m;j++)
      *(matrice+0*n+j)=j;

   for(int i=1;i<n;i++)
   {
      for(int j=1;j<m;j++)
      {
         int diff=0;
         if(prima[i-1] != seconda[j-1])
            diff=1;

         //cerca valore minimo
         vector<int> temp;
         temp.push_back(*(matrice+(i-1)*m+j)+1);          //cancellazione
         temp.push_back(*(matrice+i*m+(j-1))+1);          //inserimento
         temp.push_back(*(matrice+(i-1)*m+(j-1))+diff);    //sostituzione
         *(matrice+i*m+j)= *min_element(temp.begin(),temp.end());
         temp.clear();
      }
   }
}
```

```
// Distruttore.
LEVENSHTEIN::~LEVENSHTEIN()
{
   prima.clear();
   seconda.clear();
   delete []matrice;
}

class DISTANZA : public LEVENSHTEIN
{
   private:
      int operazioni;

   public:
      DISTANZA();

      int getOperazioni();
      void calcolo_distanza();

      ~DISTANZA() {};
};
// Costruttore di default.
DISTANZA::DISTANZA()
{
   operazioni=0;
}

int DISTANZA::getOperazioni(){
return operazioni;
}
```

```
// Partendo dalla soluzione finale (elemento in ultima riga, ultima colonna), ripercorre a
// ritroso, le celle con valore minimo, e in base al percorso intrapreso, risale all' operazione effettuata.
void DISTANZA :: calcolo_distanza()
{
   int sp=prima_size(), ss=seconda_size();
   int passo=1;
   if((operazioni=get_from_tabella(sp,ss))==0)
      cout<<"Non ci sono operazioni da effettuare!";
   else{
   cout<<"Operazioni effettuate: "<<operazioni;
   while(sp >= 0 && ss >= 0)   //ricostruzione del percorso a ritroso
   {
         //Caso 1: bordo superiore
      if(sp==0)
      {
         if( get_from_tabella(sp,ss) > get_from_tabella(sp,ss-1)) //inserimento
         {
            cout<<endl<<passo++<<") Ho inserito '"<<car_seconda(--ss)<<"'";
         }
         else
            ss--;
      }

      //Caso 2: bordo sinistro
      if(ss==0)
      {
         if(get_from_tabella(sp,ss) > get_from_tabella(sp-1,ss)) //cancellazione
         {
            cout<<endl<<passo++<<") Ho cancellato '"<<car_prima(--sp)<<"'";
         }
         else
            sp--;
```

```
    }



    //Caso 3: Resto della matrice
    if(sp!=0 && ss!=0 )
    {
        if(get_from_tabella(sp,ss)>(get_from_tabella(sp-1,ss-1)) && (get_from_tabella(sp-1,ss-
        1)<=get_from_tabella(sp,ss-1)) && (get_from_tabella(sp-1,ss-1)<=get_from_tabella(sp-1,ss)))
        //sostituzione
      {
        cout<<endl<<passo++<<") Ho sostituito '"<<car_prima(--sp)<<"' con '"<<car_seconda(--ss)<<"'";
      }
      else if(get_from_tabella(sp,ss)>get_from_tabella(sp,ss-1)) //inserimento
        {
          cout<<endl<<passo++<<") Ho inserito '"<<car_seconda(--ss)<<"'";
        }
        else if ( get_from_tabella(sp,ss) >get_from_tabella(sp-1,ss)) //cancellazione
          {
            cout<<endl<<passo++<<") Ho cancellato '"<<car_prima(--sp)<<"'";
          }
          else
          {
            sp--; ss--;
          }
        }
      }

  }
  cout<<endl;
}
```

- hashVocabolario.h

```
#include <iostream>
#include <string>
#include <list>
#include "levenshtein.h"

using namespace std;

class hashVocabolario: public Vocabolario
{
        public:

        hashVocabolario (){};

        bool Inserimento (string);
        void Visualizzazione();
        bool Ricerca (string);
        bool Cancellazione (string);

        list <string> Levenshtein(string);

        ~hashVocabolario (){};

        private:
           static const int M = 1000;

        list <string> hashTable[M];
        list <string> Parole_simili;
        list <string>::iterator p;

        //hash concatenato
        int a,b,h;
```

```
        int hash(char*);
        int x;




};

int hashVocabolario::hash(char *v)
{
        a = 31415; b = 27183;

        for(h=0; *v !='\0'; v++, a= a*b % (M-1))
                h = (a*h + *v) %M;

        return (h<0) ? (h+M) : h;
}


bool hashVocabolario::Inserimento(string parola)
{
        x = this->hash((char *)parola.c_str());
        this->hashTable[x].push_back(parola);
        return true;
}

void hashVocabolario::Visualizzazione()
{
int i=0;
   for(int j=0; j<M;j++){

      if(!hashTable[j].empty()){
         for(p= hashTable[j].begin(); p!=hashTable[j].end();p++){
```

```
            cout<<"Elemento\t"<<j<<"::\t"<<i<<"\t->\t"<<*p<<endl;
            i++;}
      }
   }
i=0;
}

bool hashVocabolario::Ricerca(string parola)
{
        x = this->hash((char *)parola.c_str());
        p = this->hashTable[x].begin();
        for (p = this->hashTable[x].begin(); p != this->hashTable[x].end(); p++)
        {
                if(*p == parola)
                        return true;
        }
        return false;
}

bool hashVocabolario::Cancellazione(string parola)
{
        x = this->hash((char *)parola.c_str());
        this->p = this->hashTable[x].begin();
        for (p = this->hashTable[x].begin(); p != this->hashTable[x].end(); p++)
        {
                if(*p == parola)
                {
                        p = hashTable[x].erase(p);
                        return true;
                }
        }
        return false;
```

```
}

list <string> hashVocabolario::Levenshtein(string parola)
{
        int distmin = 100;//utilizzo un valore relativamente alto

        for (int j=0; j<M; j++) // Scorre tabella hash
        {
           p= this->hashTable[j].begin();
           for (p= this->hashTable[j].begin(); p!= this->hashTable[j].end(); p++) // Scorre gli elementi con la stessa chiave
           {

              DISTANZA* stringhe = new DISTANZA;
                 string s = *p;
                 stringhe->set_string(s, parola);
                 stringhe->crea_matrice();  //alloca memoria e inizializza prima riga e colonna
       stringhe->calcolo_distanza();

       if (stringhe->getOperazioni() < distmin)
       {
          distmin = stringhe->getOperazioni();
          Parole_simili.clear();
          Parole_simili.push_back(s);
                        }
                        else if (stringhe->getOperazioni()== distmin)
                           Parole_simili.push_back(s);
           }
  }
  cout << "\nLa Distanza di Editing minore e' " << distmin << "." << endl;
        return Parole_simili;
}
```

- Main.cpp

```
#include <iostream>
#include "ClasseAstrattaVocabolario.h"
#include "hashVocabolario.h"

using namespace std;

int main ()
{
                hashVocabolario* V = new hashVocabolario;
        cout<<"     VOCABOLARIO HASH TABLE DI CAPOBIANCO SALVATORE"<<endl;
                int nav=0; // Variabile di navigazione del menù
                string parola;
do{

   cout<<"Menu: "<<endl;
   cout<<"1) Inserisci"<<endl;
   cout<<"2) Cancella"<<endl;
   cout<<"3) Ricerca"<<endl;
   cout<<"4) Visualizza"<<endl;
   cout<<"5) Exit"<<endl;
   cin>>nav;
   switch(nav){

   case 1:
          {
      cout<<"   INSERIMENTO"<<endl;
      cout <<"Digita il vocabolo da inserire: ";
                        cin >> parola;
                        if (!(V -> Ricerca (parola))) // Verifica che il vocabolo inserito non sia già presente nel
vocabolario
```

```
                        {
                           if (V -> Inserimento (parola)) // Inserisco vocabolo
                                cout <<  "INSERIMENTO DI '"<< parola<< "' AVVENUTO CON SUCCESSO" <<
endl << endl;
                                else
                              cout << "ERRORE" << endl << endl;
                        }
                        else //Ritorno nel vocabolario poichè il vocabolo già è presente.
                           cout << "ATTENZIONE: '" << parola << "' e' gia' presente nel vocabolario" << endl <<
endl;

                           V->Visualizzazione();
          break;
          }

   case 2:
          {
          cout<<"   CANCELLAZIONE"<<endl;
          system ("cls");
           V->Visualizzazione();
                        cout << "Digita il vocabolo da cancellare: ";
                        cin >> parola;
                        if (V-> Cancellazione (parola))
                           cout << "CANCELLAZIONE DI '"<< parola<<"' AVVENUTA CON SUCCESSO" << endl <<
endl;
                        else
                           cout << "ATTENZIONE: '" << parola << "' non e' presente nel vocabolario." << endl;
          break;
          }

   case 3:
          {
```

```
        cout<<"   RICERCA"<<endl;
        system ("cls");
         V->Visualizzazione();
                        cout << "Digita il vocabolo da ricercare: ";
                                cin >> parola;
                                if (V-> Ricerca(parola))
                                   cout << "RICERCA AVVENUTA CON SUCCESSO: '" << parola << "' e' presente nel vocabolario." << endl << endl;
                                else
                                {
                                        cout << "ATTENZIONE: '"<< parola <<"' non e' presente nel vocabolario." << endl;
                                        cout << "Cerco vocaboli simili con la Distanza di Editing..." << endl;
                                        list <string> Parole_simili = (V -> Levenshtein (parola));
                                        cout << "I vocaboli simili trovati sono:" << endl;
                                        list <string>::iterator p;
                        for (p = Parole_simili.begin(); p != Parole_simili.end(); p++)
                cout << *p << endl;
                        cout << endl;
                                }
        break;
        }

  case 4:
        {
        V->Visualizzazione();
        break;
        }
  case 5:
        {
        exit(1);
        break;
```

```
            }


    default:
            cout<<"Scelta non valida, riprovare..."<<endl;
        }

system("pause");
system("cls");

}while(nav!=5);

return 0;

}
```

Screen:

Menù principale del programma:

```
      VOCABOLARIO HASH TABLE DI CAPOBIANCO SALVATORE
Menu:
1) Inserisci
2) Cancella
3) Ricerca
4) Visualizza
5) Exit
```

Digitando 1 si sceglie di inserire una voce nel vocabolario:

```
      VOCABOLARIO HASH TABLE DI CAPOBIANCO SALVATORE
Menu:
1) Inserisci
2) Cancella
3) Ricerca
4) Visualizza
5) Exit
1
   INSERIMENTO
Digita il vocabolo da inserire: casa
INSERIMENTO DI 'casa' AVVENUTO CON SUCCESSO

Elemento        77::    0       ->      casa
Premere un tasto per continuare . . .
```

Un termine già inserito in precedenza non può essere reinserito:

```
Menu:
1) Inserisci
2) Cancella
3) Ricerca
4) Visualizza
5) Exit
1
   INSERIMENTO
Digita il vocabolo da inserire: casa
ATTENZIONE: 'casa' e' gia' presente nel vocabolario

Elemento        77::    0       ->      casa
Premere un tasto per continuare . . .
```

Si inseriscono altre voci nel vocabolario:

```
Menu:
1) Inserisci
2) Cancella
3) Ricerca
4) Visualizza
5) Exit
1
   INSERIMENTO
Digita il vocabolo da inserire: lampada
INSERIMENTO DI 'lampada' AVVENUTO CON SUCCESSO

Elemento        77::    0       ->      casa
Elemento        85::    1       ->      casi
Elemento        342::   2       ->      lampada
Elemento        831::   3       ->      orologio
Premere un tasto per continuare . . .
```

Per ogni termine inserito, vengono visualizzati a schermo tutte le voci presenti nel vocabolario in quel momento. Ogni elemento è individuato dal valore hash e un indice.

Ricerca con successo:

```
Elemento        77::    0       ->      casa
Elemento        85::    1       ->      casi
Elemento        342::   2       ->      lampada
Elemento        831::   3       ->      orologio
Digita il vocabolo da ricercare: casa
RICERCA AVVENUTA CON SUCCESSO: 'casa' e' presente nel vocabolario.

Premere un tasto per continuare . . .
```

Ricerca senza successo e calcolo della distanza di editing:

```
Elemento        77::    0       ->      casa
Elemento        85::    1       ->      casi
Elemento        342::   2       ->      lampada
Elemento        831::   3       ->      orologio
Digita il vocabolo da ricercare: case
ATTENZIONE: 'case' non e' presente nel vocabolario.
Cerco vocaboli simili con la Distanza di Editing...

Trasformazione di 'casa' in 'case':
Operazioni effettuate: 1
1) Ho sostituito 'a' con 'e'

Trasformazione di 'casi' in 'case':
Operazioni effettuate: 1
1) Ho sostituito 'i' con 'e'

Trasformazione di 'lampada' in 'case':
Operazioni effettuate: 6
1) Ho sostituito 'a' con 'e'
2) Ho sostituito 'd' con 's'
3) Ho cancellato 'a'
4) Ho cancellato 'p'
5) Ho cancellato 'm'
6) Ho sostituito 'l' con 'c'

Trasformazione di 'orologio' in 'case':
Operazioni effettuate: 8
1) Ho sostituito 'o' con 'e'
2) Ho sostituito 'i' con 's'
3) Ho sostituito 'g' con 'a'
4) Ho sostituito 'o' con 'c'
5) Ho cancellato 'l'
6) Ho cancellato 'o'
7) Ho cancellato 'r'
8) Ho cancellato 'o'

La Distanza di Editing minore e' 1.
I vocaboli simili trovati sono:
casa
casi

Premere un tasto per continuare . . .
```

Cancellazione con successo:

```
Elemento        77::    0       ->      casa
Elemento        85::    1       ->      casi
Elemento        342::   2       ->      lampada
Elemento        831::   3       ->      orologio
Digita il vocabolo da cancellare: casi
CANCELLAZIONE DI 'casi' AVVENUTA CON SUCCESSO

Premere un tasto per continuare . . .
```

Possiamo verificare la precedente:

- digitando nuovamente lo stesso termine (cancellazione senza successo):

```
Elemento        77::    0       ->      casa
Elemento        342::   1       ->      lampada
Elemento        831::   2       ->      orologio
Digita il vocabolo da cancellare: casi
ATTENZIONE: 'casi' non e' presente nel vocabolario.
Premere un tasto per continuare . . .
```

- visualizzando l'intero vocabolario:

```
Menu:
1) Inserisci
2) Cancella
3) Ricerca
4) Visualizza
5) Exit
4
Elemento        77::    0       ->      casa
Elemento        342::   1       ->      lampada
Elemento        831::   2       ->      orologio
Premere un tasto per continuare . . .
```

- ricercando il termine (ricerca senza successo):

```
Elemento         77::    0       ->      casa
Elemento         342::   1       ->      lampada
Elemento         831::   2       ->      orologio
Digita il vocabolo da ricercare: casi
ATTENZIONE: 'casi' non e' presente nel vocabolario.
Cerco vocaboli simili con la Distanza di Editing...

Trasformazione di 'casa' in 'casi':
Operazioni effettuate: 1
1) Ho sostituito 'a' con 'i'

Trasformazione di 'lampada' in 'casi':
Operazioni effettuate: 6
1) Ho sostituito 'a' con 'i'
2) Ho sostituito 'd' con 's'
3) Ho cancellato 'a'
4) Ho cancellato 'p'
5) Ho cancellato 'm'
6) Ho sostituito 'l' con 'c'

Trasformazione di 'orologio' in 'casi':
Operazioni effettuate: 7
1) Ho cancellato 'o'
2) Ho sostituito 'g' con 's'
3) Ho sostituito 'o' con 'a'
4) Ho sostituito 'l' con 'c'
5) Ho cancellato 'o'
6) Ho cancellato 'r'
7) Ho cancellato 'o'

La Distanza di Editing minore e' 1.
I vocaboli simili trovati sono:
casa

Premere un tasto per continuare . . .
```

ABR Red-Black

Gli alberi Red-Black sono una delle tante varianti degli Alberi Binari di Ricerca. Sono capaci di auto-bilanciarsi ogni qual volta viene eseguita un operazione di inserimento o cancellazione di un nodo, in modo da garantire buone prestazioni anche nel caso peggiore (albero fortemente sbilanciato). Un albero Red-Black, quindi, è un albero binario di ricerca in cui ogni nodo oltre ad avere i consueti attributi chiave, figlio sinistro, figlio destro e padre è caratterizzato dal **colore,** che può assumere valore "rosso" o "nero". Gli alberi Red-Black devono soddisfare le seguenti proprietà:

1. Ogni nodo può essere **rosso** o **nero.**
2. Il nodo **radice** è **nero**.
3. I nodi NIL sono **neri**.
4. I figli di un nodo **rosso** sono **neri**
5. Ogni percorso da un nodo ad una foglia contiene lo stesso numero di nodi neri.

L'**altezza nera** di un nodo x, *bh(x)* è definita come il numero di nodi neri, lungo un percorso che inizia da x (ma non lo include) e finisce in una foglia. Essa, per le proprietà viste in precedenza, risulta essere ben definita. L'altezza nera di un albero Red-Black è l'altezza nera della sua radice.

Particolare attenzione merita il **nodo NIL**: è una *sentinella* che ha lo scopo di far trattare allo stesso modo i puntatori ai nodi e i puntatori ai nodi NULL, cioè al posto di un puntatore NULL si utilizza un puntatore ad un nodo NIL. Visto che, questi ultimi sono uguali, per risparmiare risorse di memoria se ne utilizza soltanto uno, a cui si fanno puntare la radice e tutte le foglie dell'albero. Si noti che solo la radice ha un nodo NIL come padre. I campi del nodo NIL sono immateriali ma possono essere impostati diversamente con gli appositi metodi.

TH degli ALBERI RED-BLACK:

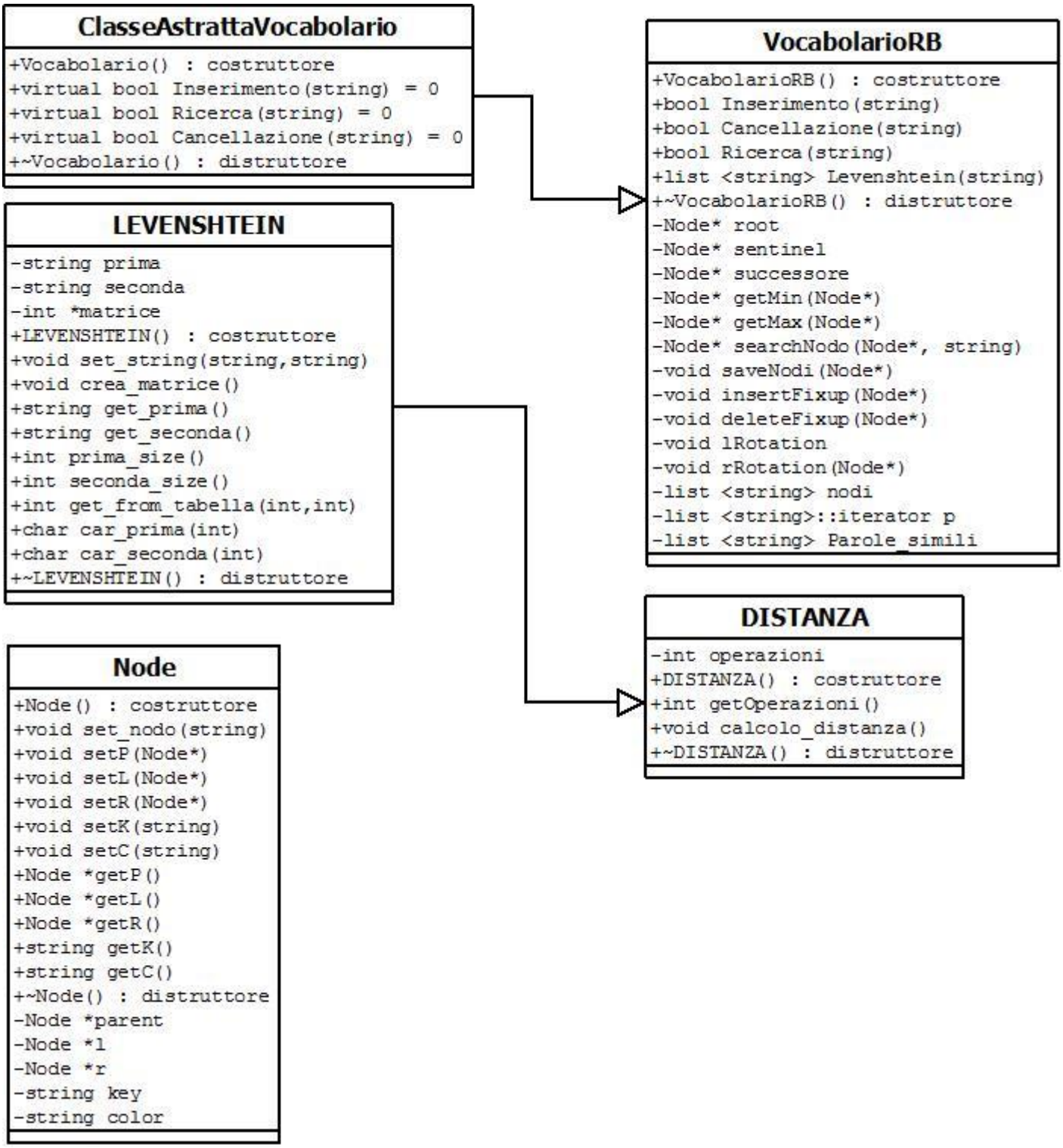
1. Ogni albero RB con radice v ha almeno *2bh(v)-1* nodi interni;
2. In un albero RB, almeno la metà dei nodi, dalla radice ad una foglia, devono essere neri;
3. In un albero RB, nessun percorso da un nodo v ad una foglia, è lungo più del doppio del percorso da v ad un'altra foglia;
4. Un albero RB con n nodi interni ha altezza massima pari a *2log(n+1)*.

L'inserimento e la cancellazione di un nodo per alberi RB si divide in due fasi: il nodo viene effettivamente inserito o cancellato, come avviene in un normale ABR; successivamente vengono ripristinate le proprietà degli alberi RB, richiamando la funzione INSERT-FIXUP o DELETE-FIXUP a seconda dei casi.

Nel dettaglio, se il nodo da inserire (che viene colorato di rosso) è la radice, l'inserimento potrebbe violare la proprietà che la radice deve essere nera, Se, invece, il padre del nodo da inserire è rosso si violerebbe la proprietà che i figli di un nodo rosso devono essere neri.

Per quanto riguarda la cancellazione, la funzione per il ripristino delle proprietà, deve essere richiamata se e solo se, il nodo da cancellare è nero, poiché non cambierebbero le altezze nere, non si creerebbero nodi rossi consecutivi e la radice resterebbe nera.

Diagramma delle classi



Implementazione:

- Node.h

```
#include <iostream>
#include <string>


using namespace std;


class Node
{
   public:


   Node ();


   void set_nodo (string value);
   void setP (Node * x);
   void setL (Node * x);
   void setR(Node * x);
   void setK(string k);
   void setC (string colore);
   Node *getP ();
   Node *getL ();
   Node *getR ();
   string getK ();
   string getC ();


   ~Node (){};
```

```
    private:

    Node *parent;
    Node *l;
    Node *r;
    string key;
    string color;
};
```

- Node.cpp

```
#include "Node.h"

Node::Node (){
    parent = NULL;
    l = NULL;
    r = NULL;
    key = "";
    color = "r";
}


void Node::set_nodo(string value)
{
    parent = NULL;
    l = NULL;
    r = NULL;
    key = value;
    color = "b";
}

void Node::setP(Node* x)
{
```

```
    parent = x;
}

void Node::setL(Node* x)
{
    l = x;
}

void Node::setR(Node* x)
{
    r = x;
}

void Node::setK(string value)
{
    key = value;
}

void Node::setC(string colore)
{
    color = colore;
}

string Node::getC()
{
    return color;
}

string Node::getK()
{
    return key;
}
```

```
Node *Node::getL()
{
   return l;
}

Node *Node::getP()
{
   return parent;
}

Node *Node::getR()
{
   return r;
}
```

- VocabolarioRB.h

```
#include "Node.cpp"
#include "Levenshtein.h"
#include "ClasseAstrattaVocabolario.h"



class VocabolarioRB : public Vocabolario
{
   public:

   VocabolarioRB();

   bool Inserimento(string);
        bool Cancellazione(string);
        bool Ricerca(string);
        list <string> Levenshtein (string);
```

```
    ~VocabolarioRB(){};

    private:

    Node* root;
    Node* sentinel;
    Node* successore (string);
            Node* getMin (Node*);
            Node* getMax (Node*);
            Node* searchNodo (Node*, string);
            void saveNodi (Node*);
    void insertFixup (Node*);
    void deleteFixup (Node*);
    void lRotation (Node*);
    void rRotation (Node*);

    list <string> nodi;
    list <string>::iterator p;
    list <string> Parole_simili;
};
```

- VocabolarioRB.cpp

```
#include "VocabolarioRB.h"

VocabolarioRB::VocabolarioRB()
{
        sentinel = new Node;
        sentinel->set_nodo("");
        root = sentinel;
};
```

```
bool VocabolarioRB::Inserimento(string parola)
{
   Node *x,*y;
          Node *z = new Node;
   z->set_nodo(parola);
          y = sentinel;
   x = root;

   while (x != sentinel)
          {
      y = x;
      if (z->getK() < x->getK())
         x = x->getL();
      else
         x = x->getR();
   }

   z->setP(y);
   if (y == sentinel)
      root = z;
   else
          {
      if (z->getK() < y->getK())
         y->setL(z);
      else
         y->setR(z);
   }
   z->setL(sentinel);
   z->setR(sentinel);
   z->setC("r");
   VocabolarioRB::insertFixup(z);
   return true;
```

```
}

void VocabolarioRB::insertFixup(Node *z)
{
   Node *y;

   while (z->getP()->getC() == "r")
   {
      if (z->getP() == z->getP()->getP()->getL())
      {
         y = z->getP()->getP()->getR();
         if (y->getC()=="r")
         {
            z->getP()->setC("b");
            y->setC("b");
            z->getP()->getP()->setC("r");
            z = z->getP()->getP();
         }
         else
         {
            if(z == z->getP()->getR())
            {
               z = z->getP();
               VocabolarioRB::lRotation(z);
            }
            z->getP()->setC("b");
            z->getP()->getP()->setC("r");
            VocabolarioRB::rRotation(z->getP()->getP());
         }
      }
      else
      {
```

```
        y = z->getP()->getP()->getP();
        if (y->getC() == "r")
        {
           z->getP()->setC("b");
           y->setC("b");
           z->getP()->getP()->setC("r");
           z = z->getP()->getP();
        }
        else
        {
           if (z == z->getP()->getL())
           {
              z = z->getP();
              VocabolarioRB::rRotation(z);
           }
           z->getP()->setC("b");
           z->getP()->getP()->setC("r");
           VocabolarioRB::lRotation(z->getP()->getP());
        }
      }
   }
   root->setC("b");
}

bool VocabolarioRB::Cancellazione(string parola)
{
          if (!Ricerca(parola))
             return false;

   Node *x,*y,*z;
          z = searchNodo(root, parola);
```

```
if (z->getL() == sentinel || z->getR() == sentinel)
   y = z;
else
   y = VocabolarioRB::successore(z->getK());

if (y->getL() != sentinel)
   x = y->getL();
else
   x = y->getR();

x->setP(y->getP());

if (y->getP() == sentinel)
   root = x;
else
        {
   if (y == y->getP()->getL())
      y->getP()->setL(x);
   else
      y->getP()->setR(x);
}

if (y != z)
   z->setK(y->getK());

if (y->getC() == "b")
   VocabolarioRB::deleteFixup(x);

if (y->getK() == z->getK())
   delete y;
else
   delete z;
```

```
    return true;
}

void VocabolarioRB::deleteFixup (Node *z)
{
    Node *y;

    while (z != root && z->getC() == "b")
    {
        if (z == z->getP()->getL())
        {
            y = z->getP()->getR();
            if (y->getC() == "r")
            {
                y->setC("b");
                y->getP()->setC("r");
                VocabolarioRB::lRotation(z->getP());
                y = z->getP()->getR();
            }
            if (y->getL()->getC() == "b" && y->getR()->getC() == "b")
            {
                y->setC("r");
                z = z->getP();
            }
            else
            {
                if (y->getC() == "b")
                {
                    y->getL()->setC("b");
                    y->setC("r");
                    VocabolarioRB::rRotation(y);
```

```
            y = z->getP()->getR();
        }
        y->setC(z->getP()->getC());
        z->getP()->setC("b");
        y->getR()->setC("b");
        VocabolarioRB::lRotation(z->getP());
        z = root;
    }
}
else
    {
    y = z->getP()->getL();
    if (y->getC() == "r")
    {
        y->setC("b");
        y->getP()->setC("r");
        VocabolarioRB::rRotation(z->getP());
        y = z->getP()->getL();
    }
    if (y->getR()->getC() == "b" && y->getL()->getC() == "b")
    {
        y->setC("r");
        z = z->getP();
    }
    else
    {
        if(y->getC() == "b")
        {
            y->getR()->setC("b");
            y->setC("r");
            VocabolarioRB::lRotation(y);
            y = z->getP()->getL();
```

```
            }
            y->setC(z->getP()->getC());
            z->getP()->setC("b");
            y->getL()->setC("b");
            VocabolarioRB::rRotation(z->getP());
            z = root;
          }
        }
    }
    z->setC("b");
}

bool VocabolarioRB::Ricerca(string parola)
{
    Node *p = root;
    int trovato = 0;
    while (p != NULL && trovato == 0)
    {
        if (p->getK() == parola)
            trovato = 1;
        if (trovato == 0)
            {
                if (p->getK() < parola)
                    p = p->getR();
                else
                    p = p->getL();
            }
    }
    if (trovato == 0)
        return false;
    else
        return true;
```

```
}

Node *VocabolarioRB::searchNodo (Node *punt, string k)
{
   while (punt != sentinel && k != punt->getK())
            {
      if (k < punt->getK())
         punt = punt->getL();
      else
         punt = punt->getR();
   }

   if (punt == sentinel)
      return NULL;
   return punt;
}



list <string> VocabolarioRB::Levenshtein(string parola)
{
          int distMin = 100;// Utilizzo un valore relativamente alto.
   saveNodi(root);
          for (p= this->nodi.begin(); p!= this->nodi.end(); p++) // Scorre gli elementi con la stessa chiave.
             {


         DISTANZA* stringhe = new DISTANZA;
                     string s = *p;
                     stringhe->set_string(s, parola);
                     stringhe->crea_matrice();  //alloca memoria e inizializza prima riga e colonna
         stringhe->calcolo_distanza();
```

```
        if (stringhe->getOperazioni() < distMin)
        {
           distMin = stringhe->getOperazioni();
           Parole_simili.clear();
           Parole_simili.push_back(s);
                        }
                        else if (stringhe->getOperazioni()== distMin)
                           Parole_simili.push_back(s);
            }
            cout << "\nDistanza di Editing minore: " << distMin << "" << endl;
            nodi.clear();
        return Parole_simili;
   }



void VocabolarioRB::saveNodi(Node *nodo)
{
        if (nodo != sentinel)
        {
                saveNodi(nodo->getL());
                nodi.push_back(nodo->getK());
                saveNodi(nodo->getR());
        }
}

Node *VocabolarioRB::getMax (Node *x)
{
   while (x->getR() != sentinel)
        {
      x = x->getR();
   }
   return x;
```

```
}

Node *VocabolarioRB::getMin (Node *x)
{
   while(x->getL() != sentinel)
          {
      x=x->getL();
   }
   return x;
}

void VocabolarioRB::lRotation (Node *z)
{
   Node *y;
   y = z->getR();
   z->setR(y->getL());

   if (y->getL() != sentinel)
      y->getL()->setP(z);

   y->setP(z->getP());

   if (y->getP() == sentinel)
      root = y;
   else
          {
      if (z == y->getP()->getL())
         z->getP()->setL(y);
      else
         z->getP()->setR(y);
   }
```

```
    y->setL(z);
    z->setP(y);
}

void VocabolarioRB::rRotation (Node *z)
{
    Node *y;
    y = z->getL();
    z->setL(y->getR());

    if (y->getR() != sentinel)
        y->getR()->setP(z);

    y->setP(z->getP());

    if (y->getP() == sentinel)
        root=y;
    else
            {
        if (z == y->getP()->getR())
            z->getP()->setR(y);
        else
            z->getP()->setL(y);
    }

    y->setR(z);
    z->setP(y);
}

Node *VocabolarioRB::successore (string k)
{
    Node *z,*y;
```

```
    z = root;
    y = sentinel;

    while (z != sentinel && z->getK() != k)
            {
       if (z->getK() < k)
          z=z->getR();
       else if (z->getK() > k)
       {
          y = z;
          z = z->getL();
       }
    }
    if (z != sentinel && z->getR() != sentinel)
       return VocabolarioRB::getMin (z->getR());
    else
       return y;
}
```

- Main.cpp

```
#include <iostream>
#include <stdlib.h>
#include "VocabolarioRB.cpp"

using namespace std;

int main ()
{
                    VocabolarioRB *V = new VocabolarioRB;
        cout<<"     VOCABOLARIO RED-BLACK DI CAPOBIANCO SALVATORE"<<endl;
                    int nav=0; // Variabile di navigazione del menù
                    string parola;
do{
```

```
   cout<<"Menu: "<<endl;
   cout<<"1) Inserisci"<<endl;
   cout<<"2) Cancella"<<endl;
   cout<<"3) Ricerca"<<endl;
   cout<<"4) Esci"<<endl;
   cin>>nav;
   switch(nav){

   case 1:
            {
               cout<<"   INSERIMENTO"<<endl;
               cout <<"Digita il vocabolo da inserire: ";
                              cin >> parola;
                              if (!(V -> Ricerca (parola))) // Verifica che il vocabolo inserito non sia già presente nel
vocabolario
                              {
                                 if (V -> Inserimento (parola)) // Inserisco vocabolo
                                       cout <<  "INSERIMENTO DI '"<< parola<< "' AVVENUTO CON SUCCESSO" <<
endl << endl;
                                       else
                                    cout << "ERRORE" << endl << endl;
                              }
                              else //Ritorno nel vocabolario poichè il vocabolo già è presente.
                                 cout << "ATTENZIONE: '" << parola << "' e' gia' presente nel vocabolario" << endl <<
endl;
            break;
            }

   case 2:
            {
            cout<<"   CANCELLAZIONE"<<endl;
            system ("cls");
```

```
                cout << "Digita il vocabolo da cancellare: ";
                cin >> parola;
                if (V-> Cancellazione (parola))
                   cout << "CANCELLAZIONE DI '"<< parola<<"' AVVENUTA CON SUCCESSO" << endl << endl;
                else
                   cout << "ATTENZIONE: '" << parola << "' non e' presente nel vocabolario." << endl;
        break;
        }

  case 3:
        {
        cout<<"   RICERCA"<<endl;
        system ("cls");
                cout << "Digita il vocabolo da ricercare: ";
                        cin >> parola;
                        if (V-> Ricerca(parola))
                           cout << "RICERCA AVVENUTA CON SUCCESSO: '" << parola << "' e' presente nel vocabolario." << endl << endl;
                        else
                        {
                                cout << "ATTENZIONE: '"<< parola <<"' non e' presente nel vocabolario." << endl;
                                cout << "Cerco vocaboli simili con la Distanza di Editing..." << endl;
                                list <string> Parole_simili = (V -> Levenshtein (parola));
                                cout << "I vocaboli simili trovati sono:" << endl;
                                list <string>::iterator p;
                for (p = Parole_simili.begin(); p != Parole_simili.end(); p++)
            cout << *p << endl;
                cout << endl;
                        }
        break;
```

```
            }

    case 4:
            {
            exit(1);
            break;
            }


    default:
            cout<<"Scelta non valida, riprovare..."<<endl;
       }

system("pause");
system("cls");

}while(nav!=4);

return 0;

}
```

Screen:

Menù principale del programma:

```
      VOCABOLARIO RED-BLACK DI CAPOBIANCO SALVATORE
Menu:
1) Inserisci
2) Cancella
3) Ricerca
4) Esci
```

Inserimento con successo:

```
Menu:
1) Inserisci
2) Cancella
3) Ricerca
4) Esci
1
   INSERIMENTO
Digita il vocabolo da inserire: mouse
INSERIMENTO DI 'mouse' AVVENUTO CON SUCCESSO

Premere un tasto per continuare . . .
```

Un termine già inserito in precedenza non può essere reinserito:

```
Menu:
1) Inserisci
2) Cancella
3) Ricerca
4) Esci
1
   INSERIMENTO
Digita il vocabolo da inserire: mouse
ATTENZIONE: 'mouse' e' gia' presente nel vocabolario

Premere un tasto per continuare . . .
```

Ricerca con successo:

```
Digita il vocabolo da ricercare: mouse
RICERCA AVVENUTA CON SUCCESSO: 'mouse' e' presente nel vocabolario.

Premere un tasto per continuare . . .
```

Ricerca senza successo e calcolo della distanza di editing:

```
Digita il vocabolo da ricercare: case
ATTENZIONE: 'case' non e' presente nel vocabolario.
Cerco vocaboli simili con la Distanza di Editing...

Trasformazione di 'monitor' in 'case':
Operazioni effettuate: 7
1) Ho sostituito 'r' con 'e'
2) Ho sostituito 'o' con 's'
3) Ho sostituito 't' con 'a'
4) Ho sostituito 'i' con 'c'
5) Ho cancellato 'n'
6) Ho cancellato 'o'
7) Ho cancellato 'm'

Trasformazione di 'mouse' in 'case':
Operazioni effettuate: 3
1) Ho sostituito 'u' con 'a'
2) Ho sostituito 'o' con 'c'
3) Ho cancellato 'm'

Trasformazione di 'stampante' in 'case':
Operazioni effettuate: 7
1) Ho sostituito 't' con 's'
2) Ho cancellato 'n'
3) Ho cancellato 'a'
4) Ho cancellato 'p'
5) Ho cancellato 'm'
6) Ho sostituito 't' con 'c'
7) Ho cancellato 's'

Trasformazione di 'tastiera' in 'case':
Operazioni effettuate: 5
1) Ho cancellato 'a'
2) Ho cancellato 'r'
3) Ho cancellato 'i'
4) Ho cancellato 't'
5) Ho sostituito 't' con 'c'

Distanza di Editing minore: 3
I vocaboli simili trovati sono:
mouse

Premere un tasto per continuare . . .
```

Cancellazione con successo:

```
Digita il vocabolo da cancellare: monitor
CANCELLAZIONE DI 'monitor' AVVENUTA CON SUCCESSO

Premere un tasto per continuare . . .
```

Cancellazione senza successo:

```
Digita il vocabolo da cancellare: monitor
ATTENZIONE: 'monitor' non e' presente nel vocabolario.
Premere un tasto per continuare . . .
```

Verifica della cancellazione tramite ricerca (senza successo) della voce appena cancellata:

```
Digita il vocabolo da ricercare: monitor
ATTENZIONE: 'monitor' non e' presente nel vocabolario.
Cerco vocaboli simili con la Distanza di Editing...

Trasformazione di 'mouse' in 'monitor':
Operazioni effettuate: 5
1) Ho sostituito 'e' con 'r'
2) Ho sostituito 's' con 'o'
3) Ho sostituito 'u' con 't'
4) Ho inserito 'i'
5) Ho inserito 'n'

Trasformazione di 'stampante' in 'monitor':
Operazioni effettuate: 8
1) Ho sostituito 'e' con 'r'
2) Ho inserito 'o'
3) Ho sostituito 'n' con 'i'
4) Ho sostituito 'a' con 'n'
5) Ho sostituito 'p' con 'o'
6) Ho cancellato 'a'
7) Ho cancellato 't'
8) Ho cancellato 's'

Trasformazione di 'tastiera' in 'monitor':
Operazioni effettuate: 7
1) Ho sostituito 'a' con 'r'
2) Ho sostituito 'r' con 'o'
3) Ho sostituito 'e' con 't'
4) Ho sostituito 't' con 'n'
5) Ho sostituito 's' con 'o'
6) Ho sostituito 'a' con 'm'
7) Ho cancellato 't'

Distanza di Editing minore: 5
I vocaboli simili trovati sono:
mouse

Premere un tasto per continuare . . .
```