04/2015

Password Finder:

UN APPROCCIO BRUTE FORCE IN MPI

Autore: **Valerio Colamatteo**

Indice:

Tale elaborato di sintesi nasce e si sviluppa come elemento di compendio informativo per il lavoro di programmazione svolto per la creazione di un *password-finder*, mosso da mire espositive **esclusivamente** finalizzate al supporto integrativo dell'applicativo pratico contestualmente realizzato.

A quanto di seguito esposto non può e non deve dunque essere attribuita capacità o pretesa alcuna di autonomia strutturale e/o indipendenza elaborativa nell'analisi delle tematiche ivi espressamente richiamate. Ciò in quanto la presa visione della presente, nonché le eventuali conseguenti analisi logico/argomentative saranno, si presume, adeguatamente coadiuvate dall'esposizione critica dell'autore nonché, si sottolinea, dal (già citato) "materiale informatico" appositamente prodotto. Quest'ultimo, in particolar modo, è da considerarsi il vero risultato dell'analisi tecnico/applicativa eseguita sui temi di riguardo.

1 Introduzione

L'applicazione realizzata prevede l'utilizzo di tecniche di ricerca esaustiva (*Brute Force*) finalizzate alla generazione di tutte le possibili combinazioni di caratteri alfanumerici sino alla ricostruzione e fedele della stringa inserita dall'utente in fase esecutiva.

Nello specifico, gli unici vincoli in tal senso, si individuano nell'impossibilità di inserimento (e conseguentemente di ricerca) di caratteri non alfanumerici e in una limitazione della lunghezza della stesa stringa a 15 caratteri[1].

Il paradigma di programmazione adottato è quello del *Massage Passing*, la libreria di riferimento adottata è stata *MPI 3.0*, Il linguaggio di programmazione il *C++*.

2 Struttura programma seriale

L'applicativo realizzato presenta una ampia modularità nella articolazione di vari sotto-programmi in esso contenuti. Si procederà quindi con l'esclusiva esposizione dei principali, relativamente, in questa fase, alla sola implementazione seriale del programma.

- ***"Initial.cpp"*** usato per creare una sorta di interfaccia iniziale utile all'utente;
- ***"Origine.cpp"*** che rappresenta il cuore del programma in quanto contiene la funzione **main()** da cui richiamare a sua volta la funzione **CrackPassword()**;
- ***"Output.cpp"***, rappresenta una sorta di interfaccia di termine programma dove vengono mostrati risultati e tempo trascorso per effettuare la ricerca, inoltre viene mostrata anche la possibilità di salvare questi risultati su file. Presenta la funzione **CrackPassword()** da cui poi viene utilizzata la funzione **passFinder()**;

[1] Tale vincolo non inficia in alcun modo l'utilità pratica dell'applicativo prodotto in quanto il limite imposto di 15 caratteri appare essere già ben oltre le potenzialità computazionali di qualsiasi singola macchina elaborativa corrente.

- ***"passFinder.cpp"***, qui è presente il cuore dell'algoritmo dove in sostanza vengono generate tutte le combinazioni. La funzione **CrackPassword()** considera 7 alfabeti diversi su cui generare le combinazioni, tali alfabeti vengono di conseguenza inviati alla funzione **passFinder()** attraverso un vettore di alfabeti, ora per poter generare tutte le possibili combinazioni si considera una password di lunghezza massima accettabile pari a 15 e si applicano in cascata 15 cicli *for()* ognuno per ogni carattere della password.
  L'elemento *digit[i]* rappresenta l'indice dell'i-esimo carattere della password che scorre tutto l'alfabeto in considerazione, per rendere il tutto più compatto si è utilizzato anche il vettore Al[] a cui viene associato per ogni elemento della password la lunghezza dell'alfabeto rappresentando di conseguenza anche l'elemento di terminazione dei cicli for. Il senso di ciò sta nel fatto che la password può naturalmente anche non essere di lunghezza 15 di conseguenza inizializzando tale vettore con ogni elemento ad 1 e modificandolo ad ogni utilizzo della funzione **passFinder()** si permette una sorta di esecuzione ad apprendimento iterativo in cui si inizia considerando un numero di cicli for pari a *passwordLength* (lunghezza attuale della password) finche al più non se ne considerano tutti e 15.
  Al centro dei 15 cicli for viene strutturato un ulteriore ciclo for che gestisce la password vera e propria e che serve per inserire all'interno della password ogni combinazione ottenuta.

3 Struttura programma parallelo

Le due implementazioni proposte, quella seriale e quella parallela, appaiono fortemente coincidenti nella logica operativa che li caratterizza. Nella realizzazione e della versione parallela, in particolare, ci si è posti la domanda su quale fosse la parte che avesse senso parallelizzare. La domanda ha avuto immediata risposta nella constatazione di fatto che la parte applicativa con nettamente maggior peso computazionale è quella attinente la generazione di tutte le possibili combinazioni.
La soluzione di fondo a questo problema è stata allora individuata nella suddivisione (antecedentemente l'inizio dell'elaborazione in questione) dello

scorrimento del vettore in un numero di sottocomponenti pari esattamente al numero di processori in dotazione all'architettura elaborativa.
Seguono alcune delle più rilevanti granularità dell'approccio di programmazione parallela *MPI* rispetto all'analoga versione seriale in precedenza introdotta:

- ***"Origine.cpp"***, qui la funzione **main()** presenta anche le definizioni per l'ambiente *MPI*, la funzione **currentDatetime()** che fornisce il tempo corrente con cui inizierà il conteggio ed infine la funzione fulcro del *main* **hackPassword()**;

- ***"Output.cpp"***, qui è presente la fuzione **Divide()** che appunto divide l'alfabeto in considerazione in base al numero di processori ed infine la funzione **crackPassword()** che richiama opportunamente la funzione **Divide()** per dividere ogni alfabeto e la funzione **passFinder()** che si occupa di trovare la password.

- ***"passFinder.cpp"***, nella funzione la stringa OriginalAlphabet rappresenta l'alfabeto originale per intero mentre la stringa alphabet rappresenta l'alfabeto diviso. Per la restante parte della funzione la generazione delle combinazioni è portata a termina con la stessa modalità dell'applicazione seriale con la differenza che l'ultima lettera della password che cicla viene divisa per il numero di processori dividendo cosi il tempo necessario a trovare la password nel complesso.

4 Risultati

Riportiamo, di seguito, alcuni dei test effettuati su ambedue gli algoritmi (seriale e parallelo). Si sottolinea che gli stessi rappresentano solo un piccolo campione arbitrariamente scelto dagli autori in rappresentazione del lavoro di *testing* effettuato e a conferma dei risultati ottenuti. Più dettagliatamente, i risultati in tabella rappresentano i valori medi conseguiti su 20 *run* indipendenti consecutivi.

Test effettuati su macchina con processore Intel *Core2Duo T9600* 2.80*Ghz*. L'architettura in questione presenta 2 *core* fisici e assenza di *SMT*.

4.1 Risultati seriale

LUNGHEZZA	PASSWORD	TEMPO MEDIO
4	casa	0.049s
5	casal	1.267s
6	casale	1.1230min
7	casa123	96.239min

4.2 Risultati parallelo

LUNGHEZZA	PASSWORD	TEMPO MEDIO
4	casa	0.0367s
5	casal	0.0675s
6	casale	33.6086s
7	casa123	47.6060min

Test effettuati su macchina con processore Intel *Core i7-2600* 3.40*GHz*.

. L'architettura in questione presenta 4 *core* fisici e 8 logici (con *SMT* attivo).

4.1 Risultati seriale (SMT non attivo)

LUNGHEZZA	PASSWORD	TEMPO MEDIO
6	casale	51.48s
7	casa123	73.387217min

4.2 Risultati parallelo (SMT non attivo)

LUNGHEZZA	PASSWORD	TEMPO MEDIO
6	casale	13.466s
7	casa123	19.018387min

4.3 Risultati parallelo (SMT attivo)

LUNGHEZZA	PASSWORD	TEMPO MEDIO
6	casale	8.6895s
7	casa123	11.673367min

5 Conclusioni

I dati forniti rilevano semplicemente una verità già nota. Ragion per cui non vanno intesi come una azione sperimentale vera e propria ma, piuttosto, null'altro che come una semplice conferma empirica circa le deduzioni logico/teoriche formulate.

Come atteso, dunque, lo *speed-up* (assoluto) segue quasi perfettamente il numero di core fisici presenti nell'architettura elaborativa rilevando, in maniera non particolarmente rilevante, solo piccole sotto-andature derivanti dall'*overhead* comunicativo gravante sulla parte parallela del programma. Al tempo stesso maggiori deviazioni temporali si sono registrate, come del resto prevedibile, per stringhe di lunghezza particolarmente limitata per l'hardware su cui è eseguito il programma.

6 Codice (parallelo)

Main.cpp

```
#include <iostream>
#include <ctime>
#include <string>
#include <conio.h>
#include<vector>
#include<fstream>
#include <windows.h>
#include <mpi.h>
#include <stdlib.h>

#include "FolderBrowserDialog.h"
#include "Initial.h"
#include "Other.h"
#include "Output.h"
#include "passFinder.h"
#include"ReverseFounder.h"
using namespace std;


extern long int attempt;
extern string test;

string pass;
const char *passChar ;

extern bool correct;
string timeCurrentStart;
char* dt;

int world_size;
int world_rank;
int root;


char msg[200];
char date[100];


int main(){


      // Initialize the MPI environment
    MPI_Init(0,0);
                                  MPI_Status status;
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

    // Get the name of the processor
    char processor_name[MPI_MAX_PROCESSOR_NAME];
    int name_len;
      MPI_Get_processor_name(processor_name, &name_len);
```

```
       if(world_rank==0){


timeCurrentStart=currentDatetime();

    starter();
       printf("\n\n\n\n\n\n\n");

       printf("**************************************************************************
                                                                          **********\n");
    cout << " Inserire Password alfanumerica (Max 15 caratteri):\n";

       printf("**************************************************************************
                                                                          **********\n");
       printf("\n\n ");


       Error_handler();



       int count=0;
       int ErrorChar=0;


        pass=msg;

        root=rootProcessorIdentifier(msg,pass);



   for (int i=1;i<world_size;i++){
   MPI_Send(&root, 1, MPI_INT, i, 0, MPI_COMM_WORLD);
  // MPI_Send(&mem,1,MPI_INT,i,0,MPI_COMM_WORLD);
   MPI_Send(timeCurrentStart.c_str(), timeCurrentStart.length(), MPI_CHAR, i, 0,
MPI_COMM_WORLD);
   MPI_Send(pass.c_str(), pass.length(), MPI_CHAR, i, 0, MPI_COMM_WORLD);}


       }

       else{

       MPI_Recv(&correct, 1, MPI_C_BOOL, 0, 0, MPI_COMM_WORLD, &status);
       if(correct==false){
              MPI_Finalize();}

       MPI_Recv(&root, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, &status);
//     MPI_Recv(&mem,1,MPI_INT,0,0,MPI_COMM_WORLD,&status); cout<<"mem
"<<*mem<<endl; //system("pause");
       MPI_Recv(&date, 100, MPI_CHAR, 0, 0, MPI_COMM_WORLD, &status);
       MPI_Recv(&msg, 20, MPI_CHAR, 0, 0, MPI_COMM_WORLD, &status);
       timeCurrentStart=date;
       pass=msg;
       //CreateArrayProc();

              }
              hackPassword();
```

```
    return 0;
}
```

Initial.cpp

```
#include <iostream>
#include<conio.h>
#include<mpi.h>
#include "Initial.h"
using namespace std;


void MyEntry(){   // customized starting point

       printf( "Premi un tasto per iniziare:\n" );
       cout<<endl;

       //char c=getchar(); //cout<<"c=  "<<c<<endl;
       system("pause>nul");
       printf("\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n");  // 2 in più rispetto al
seriale....



}

void MyQuit()     // customized ending point
{

       cout<<endl<<endl<<endl<<endl<<endl<<endl;


       printf("\n\n\n");

       printf("##########################################################################
############\n");
       printf("-------------------------  S. E. P.  --  2014 / 2015 ---------------
                                                    ----------\n\n\n");

printf("                       ESECUZIONE TERMINATA CON SUCCESSO:
                                                                    \n");

printf("             Attacco Brute Force a dizionario con paradigma MPI
                                                                  \n\n");
printf("        V. Colamatteo  matr.: 0040204    --    V. Testa  matr.: 0040445
                                                               \n\n\n");
printf("-------------------------  S. E. P.  --  2014 / 2015 ----------------------
                                                                ---\n");

printf("##########################################################################
############\n\n");
printf("\n\n\n");


       printf( "\n\nPremi un tasto per terminare definitivamente:\n" );
```

```
	exit(0);
}

void starter(){




	printf("\n\n\n");

	printf("##############################################################################
############\n");
printf("--------------------------  S. E. P.  --  2014 / 2015 ----------------------
---\n\n\n");
printf("                               Elaborato di fine corso:
\n");
printf("             Attacco Brute Force a dizionario con paradigma MPI
\n\n");
printf("                                  Valerio Colamatteo\n");
printf("                                    matr.: 0040204\n\n");
printf("                                  Valerio Testa\n");
printf("                                    matr.: 0040445\n\n\n");
printf("--------------------------  S. E. P.  --  2014 / 2015 ----------------------
---\n");

printf("##############################################################################
####\n\n");

printf("\n\n\n");


                    MyEntry();
                    //system("cls");



}


void formInseert(){

	printf("\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n"); fflush(stdout);

	printf("**************************************************************************
**********\n"); fflush(stdout);
    cout << " Reinserire Password prego...\n";  fflush(stdout);

	printf("**************************************************************************
**********\n");  fflush(stdout);
	printf("\n\n ");   fflush(stdout);

}

void myExit_Error() {


printf("\n\n\n\n\n\n\n\n\n\n");

printf("##############################################################################
####\n");
```

```
printf("-------------------------  S. E. P.  --  2014 / 2015 ----------------------
---\n\n\n");
printf("                              Elaborato di fine corso:
\n");
printf("             Attacco Brute Force a dizionario con paradigma MPI
\n\n");
printf("*****************************************************************************
****\n");
printf("                       ESECUZIONE TERMINATA CON ERRORE.
\n");

printf("*****************************************************************************
****\n\n");
printf("-------------------------  S. E. P.  --  2014 / 2015 ----------------------
---\n");
printf("###############################################################################
####\n\n");
printf("\n\n\n");

       printf("Premere un tasto per terminare definitivamente...\n");
       MPI_Finalize();

}



void FinalResult(string pass, long int attempt, double time){

       cout<<endl<<endl<<endl<<endl<<endl;
       cout<<endl<<endl<<endl<<endl<<endl;

printf("\n\n\n"); fflush(stdout);

printf("###############################################################################
####\n");fflush(stdout);

       printf("\n\n\n");fflush(stdout);
       printf("Riepilogo password identificata:       %s\n",pass);fflush(stdout);

    printf("Numero di tentativi complessivo:       %ld\n",attempt);fflush(stdout);

       if(time<60){

              printf("Time-elapsed comlessivo (secondi):
       %fl\n",time);fflush(stdout); }

else{

              printf("Time-elapsed comlessivo (minuti):
%fl\n",time/60);fflush(stdout); }


       printf("\n\n\n");fflush(stdout);
printf("###############################################################################
####\n\n");fflush(stdout);



}
```

Other.cpp

```
#include<iostream>
#include <string>
#include <windows.h>
#include <mpi.h>
#include "Initial.h"
#include "Other.h"
using namespace std;

 bool correct=false;

extern string pass;
extern char msg[20];
extern int world_size;

string AcquirePathSavingFile(){

       OPENFILENAME ofn;

    char szFileName[MAX_PATH] = "";

    ZeroMemory(&ofn, sizeof(ofn));

    ofn.lStructSize = sizeof(ofn);
    ofn.hwndOwner = NULL;
    ofn.lpstrFilter = "Text Files (*.txt)\0*.txt\0All Files (*.*)\0*.*\0";
    ofn.lpstrFile = szFileName;
    ofn.nMaxFile = MAX_PATH;
    ofn.Flags = OFN_EXPLORER | OFN_FILEMUSTEXIST | OFN_HIDEREADONLY;
    ofn.lpstrDefExt = "txt";

    if(GetSaveFileName(&ofn)){
       string path=  ofn.lpstrFile;
       return path;}

       else{MyQuit();}

}


 void Error_handler(){
    int count=0;
       int ErrorChar=0;

                    caseOne(count);

              caseTwo(count);



}


 void caseOne(int count){

              int d; int X=-1;
```

```
            correct=false;
            while(!correct){

                cin >> msg;
                    pass=msg;




            if(pass.length()>15){  //1 0 2
                        if(count>0&&count<=2)
// tasti: annulla, riprova, ignora    // icona    // tasto riprova selezionato di
default
                                        d=MessageBox(0, "ATTENZIONE!\nERRORE!  La
password deve contenere al massimo 15 caratteri alfanumerici.", "  FATAL ERROR
",MB_ABORTRETRYIGNORE|MB_SETFOREGROUND| MB_ICONWARNING|MB_DEFBUTTON2);

                                      else // 0
                                     d=MessageBox(0, "ERRORE! Lunghezza password
superiore a 15!", "    FATAL ERROR   ",MB_RETRYCANCEL|
MB_SETFOREGROUND|MB_ICONHAND|MB_DEFBUTTON1);

                    if(count>2&& d==4){
// tasti SI NO cancel
                                  d=MessageBox(0, "Password ancora errata....\n\nSI
VUOLE RIPROVARE ANCORA?", "  FATAL ERROR   ",MB_YESNOCANCEL|
MB_SETFOREGROUND|MB_ICONWARNING|MB_DEFBUTTON1);

                    if(d==6){  // SI

                          formInseert();
                          continue;
                           }

                    if(d==7)  {//No

                      d=MessageBox(0, "SI E' SICURI DI VOLERE ABBANDONARE LA
PROCEDURA DI RECUPERO PASSWORD ?\n\n", "  ULTIMATE WARNING   ",MB_YESNO|
MB_SETFOREGROUND|MB_ICONWARNING|MB_DEFBUTTON1);

          if(d==6){
                                  for (int i=1;i<world_size;i++)
                                  MPI_Send(&correct, 1, MPI_C_BOOL, i, 0,
                                                  MPI_COMM_WORLD);

                                  myExit_Error();
                          exit(-1);  }
                              if(d==7){
                                    formInseert();
                                    continue;
                              }
                          }

                                 }

                                    if(d==11){
                                          formInseert();
                                     count++;
                                                  continue;}
```

```
if(d==2){  //Annulla

     for (int i=1;i<world_size;i++){

            MPI_Send(&correct, 1, MPI_C_BOOL, i, 0, MPI_COMM_WORLD);}
           MPI_Finalize();
           myExit_Error();

            exit(-1);}

                  if(d==4)   {  // Riprova

                              count++;

                           formInseert();
                           continue;

                                       }

                     if(d==3){ // Annulla

                         for (int i=1;i<world_size;i++)

                                MPI_Send(&correct, 1, MPI_C_BOOL, i, 0,
                                                       MPI_COMM_WORLD);
                                 MPI_Finalize();
                                 myExit_Error();
                                 exit(-1);
                                }

                     }

                   if(d==5){  // Ignora

                          for (int i=1;i<world_size;i++)
                                MPI_Send(&correct, 1, MPI_C_BOOL, i, 0,
                                                       MPI_COMM_WORLD);
                                 myExit_Error();
                                  exit(-1);
                                              }


   else{

correct=true;

return ;


          }

    }


}
```

```
       boolean alphanumeric(){

                            int num=0;



size_t i = 0;  boolean findErrorChar=false;
size_t len = pass.length();

while(i < len){

    if (!isalnum(pass[i]) || pass[i] == ' '){
              num++;
         pass.erase(i,1);
         len--;
    }else
         i++;
}

if(num>0){
       findErrorChar=true;

}
return findErrorChar;

       }



 void  caseTwo(int count){

              string OrigianlPass;  int d;
              correct=false;
char NewPass[300];   boolean findErrorChar;


              while(!correct){
if(count>0)
                    cin >> msg;
                       pass=msg;

            OrigianlPass=pass;

                     findErrorChar=alphanumeric();

                     if(findErrorChar){  count++;

                            if(count>0&&count<=2){
// tasti: annulla, riprova, ignora    // icona    // tasto riprova selezionato di
default
                               d=MessageBox(0, "ATTENZIONE!\nERRORE!  La password
deve contenere esclusivamente caratteri alfanumerici.", "  FATAL ERROR
",MB_ABORTRETRYIGNORE|MB_SETFOREGROUND| MB_ICONWARNING|MB_DEFBUTTON2);

                           sprintf(NewPass," PASSWORD INSERITA:  %s\n\nPASSWORD
                            CORRETTA:  %s\n", OrigianlPass.c_str(),pass.c_str());

                                       }
```

```
                    if(count>2){
// tasti SI NO cancel
                     d=MessageBox(0, "Password ancora errata....\n\nSI VUOLE
RIPROVARE ANCORA?", "  FATAL ERROR   ",MB_YESNOCANCEL|
MB_ICONWARNING|MB_DEFBUTTON1);
                           sprintf(NewPass," PASSWORD INSERITA:  %s\n\nPASSWORD
                           CORRETTA:  %s\n", OrigianlPass.c_str(),pass.c_str());
                           //



                    if(d==6){  // SI

                           formInseert();
                           MessageBox(0, NewPass, "WARNING  ", MB_OK |
                                 MB_ICONINFORMATION | MB_SETFOREGROUND | MB_RIGHT);
                       continue;
                           }

                    if(d==7)  {//No

                           for (int i=1;i<world_size;i++)

                     MPI_Send(&correct, 1, MPI_C_BOOL, i, 0, MPI_COMM_WORLD);
                      myExit_Error();
                      exit(-1);

                        }

                    if(d==2)  //Annulla

                           for (int i=1;i<world_size;i++)
                                  MPI_Send(&correct, 1, MPI_C_BOOL, i, 0,
                                                      MPI_COMM_WORLD);
                         myExit_Error();
                           exit(-1);
                    }


                          if(d==4)   {  // Riprova

                               MessageBox(0, NewPass, "USO CORRETTO DEL PASSWORD
FINDER  ", MB_OK | MB_ICONINFORMATION | MB_SETFOREGROUND | MB_RIGHT);
                                                      formInseert();
                                 count++;
                                 continue;
                                           }

                             if(d==3){  // Annulla

                                 for (int i=1;i<world_size;i++)
                                          MPI_Send(&correct, 1, MPI_C_BOOL, i, 0,
                                                               MPI_COMM_WORLD);

                                        myExit_Error();
                                        exit(-1);}



}
```

```
                    if(d==5){  // Ignora


                      for (int i=1;i<world_size;i++)
                         MPI_Send(&correct, 1, MPI_C_BOOL, i, 0, MPI_COMM_WORLD);
                         myExit_Error();
                         exit(-1);}

                     if(d==3){  // Annulla


                      for (int i=1;i<world_size;i++)
                             MPI_Send(&correct, 1, MPI_C_BOOL, i, 0, MPI_COMM_WORLD);
                              myExit_Error();
                             exit(-1);}


                      if(findErrorChar==0){
                             correct=true;


           for (int i=1;i<world_size;i++)

                    MPI_Send(&correct, 1, MPI_C_BOOL, i, 0, MPI_COMM_WORLD);

           return;



}
              }

       }
```

Output.cpp

```
#include <iostream>
#include <ctime>
#include <string>
#include <conio.h>
#include<vector>
#include<fstream>
#include <direct.h>
#include <windows.h>
#include<mpi.h>


#include "Initial.h"
```

```
#include "Other.h"
#include "passFinder.h"
#include "Output.h"

#include "ZipFile.h"


using namespace std;

double Time;
extern string timeCurrentStart;
extern string pass;
extern long int attempt;

clock_t start_t;

 extern int world_size;

extern int world_rank;
extern int root;


char* stopwatch(clock_t stop_t,clock_t start_t){


Time=(stop_t-start_t)/double(CLOCKS_PER_SEC);

char buff[100];

if(Time>60)
       sprintf(buff,"Time-elapsed minutes:     %fl", (Time/60));
       //cout<<"Time-elapsed minutes:     "<<Time/60<<"\n";    fflush(stdout); }
else
       sprintf(buff,"Time-elapsed seconds:     %fl", Time);
 //cout<<"Time-elapsed seconds:     "<<Time<<"\n";   fflush(stdout);



return buff;
   }


string currentDatetime(){

          // current date/time based on current system
   time_t now = time(0);

   // convert now to string form
   char *dt = ctime(&now);

   string time=dt;
   return time;

}



       void OpenDialog(string Total){

       string path;
       int d=MessageBox(0, "SI DESIDERA SALVARE I RISULTATI SU FILE ?\n\n",
"Comunication  ",MB_YESNO | MB_ICONINFORMATION);
```

```
        if(d==6){

                boolean written=false;
                while(!written){
        path=AcquirePathSavingFile();
written=WriteFile(path,Total);
                }

                int d=MessageBox(0, "SI DESIDERA SALVARE I RISULTATI INSIEME ALLE
SPECIFICHE TECNICHE DI QUESTO PC ?\n\n", "Comunication  ",MB_YESNO |
MB_ICONINFORMATION);

                if(d==6){

                       // casa= FileNameWith extention   casa2=fileName
                       int c=path.find_last_of('\\');
                      string FileNameWithExt=path.substr(c+1,path.length());
                      c=FileNameWithExt.find_last_of('.');
                      string fileName=FileNameWithExt.substr(0,c);
//                    cout<<"casa2: "<<fileName<<endl;

                      int c2=path.find(FileNameWithExt);
                      string dir=path.substr(0,c2-1);
                      dir.append("\\");
                      dir.append(fileName);
                      mkdir(dir.c_str());
//                    cout<<"dir:   "<<dir<<endl;



                      dir.append("\\"); string dir2=dir;
                      dir.append(FileNameWithExt);
//                    cout<<"casa2:   "<<dir<<endl;
                      MoveFile( path.c_str(),dir.c_str());

        // add specs--

                      QuerySpecs(dir2);

                }


                MessageBox(0, " SALVATAGGIO (Dati + Specs) AVVENUTO CON SUCCESSO",
"AVVISO  ",
                 MB_OK | MB_ICONINFORMATION);


        }
        else{

                MyQuit();}

        MyQuit();


       }



       string computeDistance(string word){
     int spaceOcc;  int spaceLeft;  string t;
```

```
                        spaceOcc=88+word.length();
            spaceLeft=111-spaceOcc;

            for(int i=0;i<spaceLeft;i++)   // creo la stringa di spazi da stampare
immediatamente dopo il nominativo del file
       t=t.append(" ");
 t=t.append("*");


            return t;

      }

      void time(){

               // current date/time based on current system
   time_t now = time(0);
   char timeRunStart[100];  char timeRunEnd[100];

timeCurrentStart;
string timeCurrentEnd=currentDatetime();


   sprintf(timeRunStart, "\n                                                    Timedate
Start:     %s\n",timeCurrentStart.c_str());  //fflush(stdin);
   sprintf(timeRunEnd, "                                                Timedate
End:       %s\n",timeCurrentEnd.c_str());
   string time1=timeRunStart;  string time2=timeRunEnd;



      char buffer1[100];  char buffer2[100]; char buffer3[100];   string t;



            string c1="\n\n\n";
            string c2="
##############################################################################\n";

            string c3="                              ---------------------------
S. E. P.  --  2014 / 2015 -------------------------\n\n\n";

            string c4="
_____________________________________________________________________________\n";

            string c5="
RISULATI ESECUZIONE:                              \n";

            string c6="                                                Attacco Brute Force
a dizionario con paradigma MPI                \n";

          string c7="
_____________________________________________________________________________\n\
n";
            string c8="\n\n\n";

            string c9="
*****************************************************************************\n";

            string c10="                                  *
*\n";
```

```
            string c11="                                *
*\n";


string c11_2="                                *             Modalità di esecuzione:
MPI                    *\n";

 t=computeDistance(pass);

 sprintf(buffer1,"                                *             Riepilogo password
identificata:          %s%s\n",pass.c_str(),t.c_str());

            string c12=buffer1;



            sprintf(buffer2,"%ld",attempt);
            string temp=buffer2;  //cout<<"tentativi buffer2:  "<<temp<<endl;
system("pause");
            t=computeDistance(temp);

 sprintf(buffer3,"                                *             Numero di tentativi
complessivo:          %s%s\n",temp,t.c_str());



 string c13=buffer3;


 sprintf(buffer1,"%lf",Time);
 temp=buffer1;

 t=computeDistance(temp);
 t=t.substr(1,t.length());



 if(Time<60)
            sprintf(buffer1,"                                *             Time-
elapsed comlessivo (sec. ):         %fl%s\n",Time,t.c_str());
 else
       sprintf(buffer1,"                                *             Time-elapsed
comlessivo (min. ):         %fl%s\n",Time/60,t.c_str());

            string c15=buffer1;
            string c16="                                *
*\n";
            string c17="                                *
*\n";
            string c18="                                *
*\n";
            string c19="
*********************************************************************************";


            string c20="\n\n\n";

            string c21="
_____________________________________________________________________________\n";

            string c22="                                V. Colamatteo
matr.: 0040204    --    V. Testa  matr.: 0040445     \n";
```

```
            string c23="
_____________________________________________________________________________\n\n\
n";

            string c24="                                  -------------------------
S. E. P.  --  2014 / 2015 -------------------------\n";

            string c25="
###############################################################################\n\n
";
            string c26="\n\n\n";




String Total=
time1+time2+c1+c2+c3+c4+c5+c6+c7+c8+c9+c10+c11+c11_2+c12+c13+c15+c16+c17+c18+c19+c20
+c21+c22+c23+c24+c25+c26;
            MPI_Finalize();
            //cout<<"\n\n";

            FinalResult(pass,attempt,Time);


      OpenDialog(Total);



      }


      void result(clock_t stop_t){

       char* timeBuff=stopwatch(stop_t,start_t);
      char buff[200];

      sprintf(buff," PASSWORD SUCCESSFULLY FOUND!\n\n      Password trovata:   %s
from process: %d  %s    ", pass.c_str(),world_rank,timeBuff);

      MessageBox(0, buff, "Result    ", MB_OK | MB_SETFOREGROUND|
MB_ICONINFORMATION | MB_RIGHT);

      time();


}


      void cutAlphabet(string alphabet,string& alpha,string& alpha2){
            string tutto[64]={};
                              alpha=alphabet.substr(0,alphabet.length()/2);


      alpha2=alphabet.substr(alpha.length(),alphabet.length());

      }
```

```
	void Divide(int numProcessor, string alphabet,string* tutto){

	int last=0;
	int lungh=alphabet.length()/numProcessor;

	if(alphabet.length()-(lungh*numProcessor)>0)
		last=lungh+alphabet.length()-(lungh*numProcessor);


	for (int i=0;i<numProcessor;i++){
		if(i==numProcessor-1 && last!=0){
		   tutto[i]=alphabet.substr(i*lungh,last);
		   //cout<<"tutto["<<i<<"]= "<<tutto[i]<<endl;
		}


				if(i==numProcessor-1 && last==0){
		   tutto[i]=alphabet.substr(i*lungh,lungh);
		   //cout<<"tutto["<<i<<"]= "<<tutto[i]<<endl;
		}

		if(i==0 && i!=numProcessor-1){
			tutto[i]=alphabet.substr(0,lungh);
		//cout<<"tutto["<<i<<"]= "<<tutto[i]<<endl;
		}
		else if((i>0&&i!=numProcessor-1)){
	tutto[i]=alphabet.substr(i*lungh,lungh);
	//cout<<"tutto["<<i<<"]= "<<tutto[i]<<endl;
		}

	}
	return;
}



	void CreateArrayProc(){

		int i=0;
		int *ReducedArrayProc=new int [world_size];


		for (int x=0;x<world_size;x++)

			ReducedArrayProc[x]=x;

		while(ReducedArrayProc[i]!=world_rank)
			i++;
		//cout<<"index:  "<<i<<"  world_rank  "<<world_rank<<endl;
system("pause");
		ReducedArrayProc[i]=-1;
		//cout<<"    "<<ReducedArrayProc[i]<<"  world_rank
"<<world_rank<<endl;
		for(int y=i;y<world_size-1;y++)
			ReducedArrayProc[y]=ReducedArrayProc[y+1];
	}
```

```
void hackPassword(){

       string *alpha1=new string[world_size],*alpha2=new
string[world_size],*alpha3=new string[world_size],*alpha4=new
string[world_size],*alpha5=new string[world_size],*alpha6=new
string[world_size],*alpha7=new string[world_size];

                        string alphabet1 = "0123456789";
                                    Divide(world_size,alphabet1,alpha1);
                    string alphabet2 = "abcdefghijklmnopqrstuvwxyz";
                                    Divide(world_size,alphabet2,alpha2);
                    string alphabet3 = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
                                    Divide(world_size,alphabet3,alpha3);
                    string alphabet4 = "0123456789abcdefghijklmnopqrstuvwxyz";
                                    Divide(world_size,alphabet4,alpha4);
                    string alphabet5 = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ";
                                    Divide(world_size,alphabet5,alpha5);

                    string alphabet6 =
"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ";

                                    Divide(world_size,alphabet6,alpha6);

                    string alphabet7 =
"0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ";
                                    Divide(world_size,alphabet7,alpha7);



                    string alphabetAll[]=
{alphabet1,alphabet2,alphabet3,alphabet4,alphabet5,alphabet6,alphabet7};
                                    string
*alpha_All[]={alpha1,alpha2,alpha3,alpha4,alpha5,alpha6,alpha7};

              boolean f=1;   int num=1; int passFound=0;

              start_t=clock();



          do{

                cout<<"\n\nNUM= "<<num<<endl;  // fflush(stdout);

                for (int i=0;i<7;i++){


f=passFinder(alphabetAll[i],alpha_All[i][world_rank],num);



                    if(f){

                    clock_t stop_t=clock();

                    passFound=1;
                    MPI_Bcast(&passFound,1,MPI_INT, world_rank,MPI_COMM_WORLD);//}
                    result(stop_t);

                    MPI_Finalize();

                    delete []alpha1,alpha2,alpha3,alpha4,alpha5,alpha6,alpha7;
```

```
                    exit(0);

                    }


////////////////////////////////////////////////////////////////////////////////////////
//////////////////////
//     cout<<"root"<<root<<endl;

                    MPI_Bcast(&passFound,1,MPI_INT,root,MPI_COMM_WORLD);

                       if(passFound==1){
                                    MPI_Finalize();

                                    delete
[]alpha1,alpha2,alpha3,alpha4,alpha5,alpha6,alpha7;
                exit(0);

                            }

////////////////////////////////////////////////////////////////////////////////////////
//////////////////////



                    }

                    num++;


             }while(!passFound);



 }


bool fexists(string filename)
{
  ifstream ifile(filename);
  return ifile;
}

       boolean WriteFile(string dest,string bufferResult){

              boolean written=false;
              fstream outfile;


              string nomeFile=dest.substr(dest.find_last_of("\\")+1,dest.length());
char Path[100];


if(fexists(dest)){
       sprintf(Path,"     Il file %s è già esistente.\n     Sostituire il file?
",nomeFile.c_str());
   int d=MessageBox(0, Path, "Salva con Nome  ",
          MB_YESNO | MB_ICONWARNING );

if(d==6){
       outfile.open(dest, ios::out);
```

```
outfile<<bufferResult;
	MessageBox(0, " SALVATAGGIO DATI AVVENUTO CON SUCCESSO", "AVVISO  ",
		MB_OK | MB_ICONINFORMATION);
	outfile.close();

	written=true;
	return written;
}
else{
	written=false;
	   return written;
}
}
else
{
outfile.open(dest, ios::out);
outfile<<bufferResult;
outfile.close();
	MessageBox(0, " SALVATAGGIO AVVENUTO CON SUCCESSO", "AVVISO  ",
		MB_OK | MB_ICONINFORMATION);
	written=true;
	return written;}



	}


	string printTitle(){



string star1="\n\t\t\t
////////////////////////////////////////////////////////////////////////////////\n"
;
//string star2="\t\t\t//
//\n";
string star3="\t\t        //      ----      ----      GENERAL SPECS OF SYSTEM
----      ----      //\n";
//string star4="\t\t      //
//\n";
string star5="\t\t
////////////////////////////////////////////////////////////////////////////////\n\
n\n\n\n\n\n";



//     string
star1="\t\t\t**********************************************************************
**\n";
//string star2="\t\t\t*
*\n";
//string star3="\t\t\t*    ----        ----    GENERAL SPECS OF SYSTEM    ----
----    *\n";
//string star4="\t\t\t*
*\n";
//string
star5="\t\t\t**********************************************************************
**\n\n\n\n\n\n\n\n";
string Total=star1+star3+star5;
```

```
return Total;

}


       void QuerySpecs(string pathFolder){

              string specs="SystemInfo.txt";
              string file=pathFolder+specs;
              //string file="C:\\Users\\Valerio\\Desktop\\OneDrive\\SystemInfo.txt";
              //cout<<"file in QuerySpec:  "<<file<<endl;
              string title=printTitle();

               ofstream out;
 out.open(file,ios::out);
 out<<title;


     FILE *fp;
  char file_type[80];  string p[50];
  string query;
  string base;
  string header="**************************";

  string AllQuery[7]={"cpu get
Name,SystemName,NumberOfCores,NumberOfLogicalProcessors","computersystem get
Domain,Manufacturer,Model,Name,NumberOfLogicalProcessors,NumberOfProcessors","os get
Caption,CSDVersion,OSArchitecture,Primary,RegisteredUser",
  "diskdrive get Model,Size","memorychip get BankLabel,Capacity,Caption","path
Win32_VideoController get AdapterCompatibility,Caption,
VideoModeDescription","desktopmonitor get MonitorType,ScreenHeight,ScreenWidth"};

  string CommandExplaination[8]={" CPU System "," PC  System "," OS  Specs "," Hard
Disk "," Memory RAM "," Graph Card "," PC Monitor "};
  string spaces ="\t\t\t";
  for (int u=0;u<7;u++){
         base="wmic ";

         query=base.append(AllQuery[u]);
         // cout<<"query1  "<<query<<endl; cout<<"base  "<<base<<endl;
  string Name=spaces+header+ CommandExplaination[u]+header;

         out<<Name;
         out<<endl<<endl;

  fp = _popen(query.c_str(), "r");
  if (fp == NULL) {
      printf("Failed to run command\n" );
      exit(-1);
  }
  int i=0;

  while (fgets(file_type, sizeof(file_type)-1, fp) != NULL) {
         p[i]=file_type; //cout<<"p "<<p[i]<<endl;


         out<<'\t'<<p[i];

           i++;

  }
```

```
  out<<"        ----------------------------------------------------------------------
---------------------"<<endl;

  query.clear();
  _pclose(fp);
  }

  out.close();

  AdjustFile(pathFolder);

       }



       void find_and_replace(ifstream &in,ofstream &out,string find, string replace
) { // problem2

        string line;
    size_t len = find.length();
      while (getline(in, line))
    {
           //cout<<"U=   "<<u<<endl;
        while (true)

{
            size_t pos = line.find(find);
            if (pos != string::npos){
                        line.replace(pos, len, replace);
                //cout<<"LINE:    "<<line<<endl;
                                        }
            else
                break;


        }
             //cout<<line<<endl;
             out<<line<<'\n';

    }

        }



void AdjustFile(string folder){

       //cout<<"folder:  "<<folder<<endl;//system("pause");
       ifstream in; ofstream out;

string myfile_in= folder+"SystemInfo.txt";
string myfile_out= folder +"SystemInfo2.txt";

//string dir="C:\\Users\\Valerio\\Desktop\\OneDrive\\PROVA\\SystemInfo.txt";

string Replace[4]={"NumberOfLogica       lProcessors","Numb
       erOfProcessors","Registe   redUser","c   olori"};
string
ReplaceWith[4]={"NumberOfLogicalProcessors","NumberOfProcessors","RegisteredUser","c
olori"};

string line;
```

```
        in.open(myfile_in,ios::in);
 out.open(myfile_out,ios::out);
 //out2.open(myfile_out_temp,ios::out);
 find_and_replace(in,out,Replace[0] ,ReplaceWith[0] );
 in.close();  out.close();
 //system("pause");



 in.open(myfile_out,ios::in);
 out.open(myfile_in,ios::out);
 find_and_replace(in,out,Replace[1] ,ReplaceWith[1] );
 in.close();  out.close();


 in.open(myfile_in,ios::in);
 out.open(myfile_out,ios::out);
 find_and_replace( in,out,Replace[2] ,ReplaceWith[2] );
 in.close();  out.close();


 in.open(myfile_out,ios::in);
 out.open(myfile_in,ios::out);
 find_and_replace( in,out,Replace[3] ,ReplaceWith[3] );
 in.close();  out.close();

 remove(myfile_out.c_str());

 ZipINAction(folder);
 //cout<<"folder:  "<<folder<<endl;

}
```

PassFinder.cpp

```
#include<iostream>
#include <string>
#include <windows.h>

using namespace std;

extern string pass;
long int attempt=0;
string test;
```

```
boolean found=false;

int Al[15]={1,1,1,1,1,1,1,1,1,1,1,1,1,1,1};
int digit[15];


boolean passFinder(string &OriginalAlphabet,string &alphabet,int &passwordLength){


       //        cout<<"pid: "<<world_rank<<"\n";  fflush(stdout);


              for (int i=0;i<passwordLength;i++)
                     Al[i]=OriginalAlphabet.length();


              Al[0]=alphabet.length();



for(digit[14]=0;digit[14]<Al[14];digit[14]++)
    for(digit[13]=0;digit[13]<Al[13];digit[13]++)
       for(digit[12]=0;digit[12]<Al[12];digit[12]++)
           for(digit[11]=0;digit[11]<Al[11];digit[11]++)

                  for(digit[10]=0;digit[10]<Al[10];digit[10]++)
                 for(digit[9]=0;digit[9]<Al[9];digit[9]++)
                    for(digit[8]=0;digit[8]<Al[8];digit[8]++)

                        for(digit[7]=0;digit[7]<Al[7];digit[7]++)
                            for(digit[6]=0;digit[6]<Al[6];digit[6]++)
                                for(digit[5]=0;digit[5]<Al[5];digit[5]++)
                                   for(digit[4]=0;digit[4]<Al[4];digit[4]++)
                                       for(digit[3]=0;digit[3]<Al[3];digit[3]++)


for(digit[2]=0;digit[2]<Al[2];digit[2]++)

for(digit[1]=0;digit[1]<Al[1];digit[1]++)

for(digit[0]=0;digit[0]<Al[0];digit[0]++){
                                                        attempt++;

                                                        test=alphabet[digit[0]];
//cout<<"test1: "<<test<<endl;
                                                            for(int
i=1;i<passwordLength;i++)
                                                                {

                                        test+=OriginalAlphabet[digit[i]]; //cout<<"test2
"<<test<<"\n";




                  }

                     if(pass.compare(test)==0){
```

```
                                found=true;
return found;}

                                        }


                            return found;
        }
```

ReverseFinder.cpp

```
#include <stdlib.h>
#include <stdio.h>
#include <iostream>
#include <string>

using namespace std;


extern int world_size;


int Processor(string pass,string alphabet,int num){



        int proc;


                int val=num/world_size;
                //cout<<"valori alfabeto per ogni processo:  "<<val<<endl;
                int x=alphabet.find(pass[0])+1;
                //cout<<"posizione della cifra iniziale della pass in alfabeto:
"<<x<<endl;  // pos di pass[0] in alphabet...
                //proc=x/world_rank;
                //cout<<"x/val"<<ceil(x/val)<<endl;
                if(((float)x/val)  <=  1)
                        proc=0;
                else{
                        proc=ceil((float)x/val)-1;
                        if(proc>(world_size-1))
                                proc=world_size-1;
```

```
		}
		return proc;
}

int rootProcessorIdentifier(char* passChar,string pass) {

			string alphabet1 = "0123456789";
		string alphabet2 = "abcdefghijklmnopqrstuvwxyz";
		string alphabet3 = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
		string alphabet4 = "0123456789abcdefghijklmnopqrstuvwxyz";
		string alphabet5 = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ";
		string alphabet6 =
"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ";
		string alphabet7 =
"0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ";


	int num=0;
	int maiusc=0;
	int minusc=0;


	int alphabetNum=10; int alphabetMin=26;  int alphabetMaiusc=26;  int
alphabetNum_Minusc=36;  int alphabetNum_Maiusc=36;  int alphabetMin_Maiusc=52;  int
alphabetNum_Minusc_Maiusc=62;

	int proc;  int temp=0;

	for(int i=0;i<pass.length();i++){
	if((int)passChar[i]>=48&&(int)passChar[i]<=57)
		//cout<<"YES"<<endl;
			num++;

	if((int)passChar[i]>=65&&(int)passChar[i]<=90)
		maiusc++;
    if((int)passChar[i]>=97&&(int)passChar[i]<=122)
		minusc++;
	}

	if(num>0&&minusc==0&&maiusc==0){
		//cout<<"alfabeto soli numeri"<<endl;
		proc= Processor(pass,alphabet1,alphabetNum);
		//cout<<"Proc N:  "<<proc<<endl;
	}

	if(num>0&&minusc>0&&maiusc==0){
		//cout<<"alfabeto numeri e minuscole"<<endl;
		proc= Processor(pass,alphabet4,alphabetNum_Minusc);
		//cout<<"Proc N:  "<<proc<<endl;
	}

	if(num>0&&minusc==0&&maiusc>0){
		//cout<<"alfabeto numeri e maiscole"<<endl;
		proc= Processor(pass,alphabet5,alphabetNum_Maiusc);
		//cout<<"Proc N:  "<<proc<<endl;
	}

		if(num>0&&minusc>0&&maiusc>0){
		//cout<<"alfabeto numeri,minuscole e maiscole"<<endl;
		proc= Processor(pass,alphabet7,alphabetNum_Minusc_Maiusc);
		//cout<<"Proc N:  "<<proc<<endl;
	}
```

```
	if(num==0&&minusc>0&&maiusc==0){
		//cout<<"alfabeto sole minuscole"<<endl;
		//cout<<"Proc N:  "<<proc<<endl;
		proc= Processor(pass,alphabet2,alphabetMin);

	}

	if(num==0&&minusc>0&&maiusc>0){
		//cout<<"alfabeto minuscole e maiscole"<<endl;
		proc= Processor(pass,alphabet6,alphabetMin_Maiusc);
		//cout<<"Proc N:  "<<proc<<endl;
	}

	if(num==0&&minusc==0&&maiusc>0){
		//cout<<"alfabeto minuscole e maiscole"<<endl;
		proc= Processor(pass,alphabet3,alphabetMaiusc);
		//cout<<"Proc N:  "<<proc<<endl;
	}

	////////////////////////////////////////////////////////////////

	return proc;

	//system("pause");
}
```

ZipFile.cpp

```
#include <windows.h>
#include <tchar.h>
#include <stdio.h>
#include<vector>
#include<string>


#include "zip.h"
//#include "unzip.h"

#include<iostream>
using namespace std;


vector<string> path_vector;
vector<string> names;
```

```
vector<string> get_all_files_names_within_folder(string path)   // questa funzione
mi salva tutti nomi dei file presenti in una directory in un vector, le cartelle
invece le ignora.
{

	vector<string> get_all_files_names_within_folder(string path);
	char search_path[200];
	sprintf(search_path, "%s*.*", path.c_str());
	WIN32_FIND_DATA fd;
	HANDLE hFind = ::FindFirstFile(search_path, &fd);
	if(hFind != INVALID_HANDLE_VALUE) {
		do {
			// read all (real) files in current folder
			// , delete '!' read other 2 default folder . and ..
			if(! (fd.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY) ) {
				names.push_back(fd.cFileName);
			}
		}while(::FindNextFile(hFind, &fd));
		::FindClose(hFind);
	}
	// stampo tutti i valoridel vector
	vector <string>::iterator walk = names.begin();
while (walk != names.end())
{
//cout <<*walk << endl;
// L'iteratore va incrementato
// per accedere all'elemento successivo
walk++;
}

// qua devo ceare un altro vettore che contiene tutti i path completi:


vector<string>pathcompleti(names.size());


string p; string path_2;


for(int i=0; i<names.size();i++){
	path_2=path;
	p=path_2.append(names[i]);
	//cout<<"P= "<<p<<endl;
	pathcompleti[i]=p;
//	cout<<pathcompleti[i]<<endl<<endl;


	}


return pathcompleti;
}


// praticamente questo nn zippa la cartella direttamente ma crea un file zip in un
percorso specifico e poi ci aggiunge dei file.

void ZipINAction(string path){

//	cout<<endl<<endl<<"path:	"<<path<<endl;
	string dir=path;
	path_vector=get_all_files_names_within_folder(path);
```

```
        //system("pause");
        const int num=path_vector.size();
//      cout<<num<<endl;

                string passZip;
        //char *passZip;
        HZIP hz;
        //const TCHAR *fn="c:\\Users\\Valerio\\Desktop\\Mozilla\\Mozilla.zip";
        int f=path.find_last_of("\\");
        path=path.substr(0,f); //cout<<"NUOVO PATH: "<<path<<endl;
        path.append(".zip");
        const TCHAR *fn=path.c_str();


        hz = CreateZip(fn,0); // se metto 0 la crea senza password....La pass in ogni
caso è la stessa per ogni file

        vector<string> file_name(path_vector.size());
        vector<string> path_name;

        const char* pathNomiFile[1000];
        const char * NomiFile[1000];

        for(int i=0; i<names.size();i++){
            //t=path_vector[i];
               pathNomiFile[i]=path_vector[i].c_str();
               //cout<<t<<endl;
               NomiFile[i]=names[i].c_str();

               ZipAdd(hz,NomiFile[i], pathNomiFile[i]);

        }


        CloseZip(hz);

        // Rimuovo dir non zippata
        string deleteDir="rmdir /Q /S "; dir="\""+dir+"\"";  deleteDir.append(dir);
//cout<<deleteDir<<endl;
        system(deleteDir.c_str());

}
```