Andrea Vaghi Mat. N° 790619

Lorenzo Randazzo Mat. N° 812591

Scelta elaborato*: RUZZLE CHEATER*

*SPECIFICA DEI REQUISITI:*

L’elaborato consiste nella realizzazione di un programma scritto in linguaggio C++, che ha come scopo l’individuazione di un numero di parole stabilito dall’utente che lo esegue, all’interno di una griglia quadrata di lettere, utilizzando come strategia la medesima dell’applicazione per dispositivi mobili Ruzzle™. In oltre, calcola il punteggio di ogni singola parola (assegnato in base al valore di ogni lettera che la compone, il quale verrà spiegato in seguito, e alla sua lunghezza) e le coordinate che ne identificano la posizione all’interno della griglia. Tutto ciò viene stampato su un file di testo che poi verrà visualizzato dall’utente. Il programma ordina le parole in output a partire da quella con punteggio maggiore e a seguire le altre in ordine di punteggio decrescente. Le parole con stesso punteggio sono ordinate in ordine alfabetico. Ogni singola parola viene stampata una volta sola e non sono ammesse parole di una sola lettera.

Il programma accetta in ingresso 4 parametri:

* Una sequenza di N^2 lettere, maiuscole o minuscole, che andranno a formare la griglia di gioco di dimensioni NxN, con N > 4. Es:

Input: DENISAODRITGPMTV Input: CPVHCALIAFSORTVRIGISDMTLB

|  |  |  |  |
| --- | --- | --- | --- |
| **D** | **E** | **N** | **I** |
| **S** | **A** | **O** | **D** |
| **R** | **I** | **T** | **G** |
| **P** | **M** | **T** | **V** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **C** | **P** | **V** | **H** | **C** |
| **A** | **L** | **I** | **A** | **F** |
| **S** | **O** | **R** | **T** | **V** |
| **R** | **I** | **G** | **I** | **S** |
| **D** | **M** | **T** | **L** | **B** |

Ovviamente il programma accetta in input solo sequenze la cui dimensione sotto radice quadrata dia un numero naturale, per semplicità compreso tra 4 e 9 inclusi, in caso contrario segnala un errore e obbliga l’utente a re-inserire il giusto numero di caratteri.

* Il numero di parole che l’utente desidera cercare all’interno della griglia e che verranno visualizzate tra i risultati.
* Una stringa che contiene il percorso all’interno del computer in corrispondenza del quale si trova il file di dizionario che verrà utilizzato come base per cercare le parole in esso contenute all’interno della griglia di gioco. All’interno del file ogni parola occupa una singola riga. Es: /Users/andreavaghi/Desktop/Dizionario/italiano.dic

a

abaca

abache

abachi

abacista

…

* Una stringa che contiene il percorso del file.txt (esistente o creato al momento) sul quale verranno visualizzati i risultati della ricerca.

Es: /Users/andreavaghi/Desktop/Dizionario/risultati.txt

*-INDIVIDUAZIONE PAROLE E ASSEGNAZIONE PUNTEGGIO-*

Le parole all’interno della griglia vengono identificate combinando gli spostamenti tra una cella e l’altra nelle seguenti direzioni:

* Orizzontalmente: da sinistra a destra e da destra a sinistra
* Verticalmente: dall’alto al basso e dal basso all’alto
* Diagonalmente: da nord-est a sud-ovest e viceversa
* Diagonalmente: da nord-ovest a sud-est e viceversa

Il punteggio viene attribuito sommando il valore di ogni singola lettera e moltiplicando il risultato per la lunghezza della parola.

I valori della lettere sono i seguenti:

* 1 Pto. per le lettere: A, B, C, D, E, F, G, I, J, K, L, M, N, O, P, R, S, T, V.
* 2 Pti. per le lettere: H, Q, U, Z.
* 3 Pti. per le lettere: W, X, Y.

Es. ESITANDO

|  |  |  |  |
| --- | --- | --- | --- |
| **D** | **E** | **N** | **I** |
| **S** | **A** | **O** | **D** |
| **R** | **I** | **T** | **G** |
| **P** | **M** | **T** | **V** |

**= (1+1+1+1+1+1+1+1) \* 8 = 64 pti.**

*-MEMORIZZAZIONE COORDINATE-*

Il percorso che identifica ogni parola trovata viene espresso come successione delle coordinate delle singole lettere che la compongono. Gli indici di righe e colonne vanno da 0 a N-1. Es. CALVARIO

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **C** | **P** | **V** | **H** | **C** |
| **A** | **L** | **I** | **A** | **F** |
| **S** | **O** | **R** | **T** | **V** |
| **R** | **I** | **G** | **I** | **S** |
| **D** | **M** | **T** | **L** | **B** |

**= [0,0];[1,0];[1,1];[0,2];[1,3];[2,2];[3,1];[2,1];**

*-OUTPUT DEL PROGRAMMA-*

Il programma produce come risultato in uscita un file testuale che contiene le parole individuate nella griglia, per un numero massimo di elementi decisi dall’utente, e i rispettivi punteggi/coordinate dei percorsi. Es:

Input: - DENISAODRITGPMTV

- 9

- /Users/andreavaghi/Desktop/Dizionario/italiano.dic

- /Users/andreavaghi/Desktop/Dizionario/risultati.txt

File “risultati.txt":

|  |  |  |  |
| --- | --- | --- | --- |
| **D** | **E** | **N** | **I** |
| **S** | **A** | **O** | **D** |
| **R** | **I** | **T** | **G** |
| **P** | **M** | **T** | **V** |

Risultati:

adesioni

- 64 [1][1]; [0][0]; [0][1]; [1][0]; [2][1]; [1][2]; [0][2]; [0][3];

esitando

- 64 [0][1]; [1][0]; [2][1]; [2][2]; [1][1]; [0][2]; [1][3]; [1][2];

inesatti

- 64 [0][3]; [0][2]; [0][1]; [1][0]; [1][1]; [2][2]; [3][2]; [2][1];

pittando

- 64 [3][0]; [2][1]; [3][2]; [2][2]; [1][1]; [0][2]; [1][3]; [1][2];

pritaneo

- 64 [3][0]; [2][0]; [2][1]; [2][2]; [1][1]; [0][2]; [0][1]; [1][2];

rationes

- 64 [2][0]; [1][1]; [2][2]; [2][1]; [1][2]; [0][2]; [0][1]; [1][0];

arpione

- 49 [1][1]; [2][0]; [3][0]; [2][1]; [1][2]; [0][2]; [0][1];

arpioni

- 49 [1][1]; [2][0]; [3][0]; [2][1]; [1][2]; [0][2]; [0][3];

arsione

- 49 [1][1]; [2][0]; [1][0]; [2][1]; [1][2]; [0][2]; [0][1];

*PROGETTAZIONE:*

Il programma utilizza diversi tipi di strutture dati per la realizzazione dell’algoritmo.

*-INDICAZIONI GENERALI SULL’ ALGORITMO-*

L’algoritmo è composto da diverse funzioni, ognuna atta a svolgere un compito diverso, per esempio ricerca delle parole nella griglia, calcolo dei punteggi, stampa della lista, all’ordinamento della stessa e così via.

Il metodo su cui si basa l’algoritmo per la realizzazione dell’obiettivo principale, ovvero la ricerca delle parole all’interno della griglia, è quello ricorsivo, ovvero viene invocata una funzione che svolge la ricerca (chiamandone altre a sua volta) intorno alla lettera di partenza per poi re-invocarsi (aggiornando le coordinate con gli altri indici e “spostando il cursore” sulla lettera trovata) se l’esito della ricerca è positivo. La ricerca avviene esaminando tutte le posizioni intorno all’ultima lettera trovata. Se ci si trova in vicoli cechi (la lettera da cercare non corrisponde a nessuna di quella intorno alla corrente, oppure le lettere che corrispondono si trovano in posizioni che la parola ha già occupato) si indietreggia di una posizione e si cercano strade alternative. La ricerca termina con esito negativo quando, indietreggiando fino alla prima lettera, non ci sono più strade da intraprendere. Verranno ora esaminati tutti gli aspetti e le strutture dell’algoritmo.

*-HEADERS-*

#include <iostream> :

Header per lo stream di input/output su schermo (cin, cout e funzioni correlate, es. cin.clear()).

#include <string> :

Header necessario per la manipolazione e la gestione di stringhe, per esempio

la successione di caratteri che identifica la griglia di gioco, i percorsi dei file in input e in output (dizionario e destinazione), le singole parole del dizionario, la loro acquisizione etc.

#include <fstream> :

Header per lo stream di input/output con files, permette l’accesso in lettura al file di dizionario per l’acquisizione delle singole parole e in scrittura (dopo averlo eventualmente creato) al file di destinazione per i risultati.

#include <math.h> :

Header per la realizzazione di alcuni calcoli matematici (es. radice quadrata della dimensione della sequenza che identifica la matrice di gioco, per determinare se il numero di caratteri inseriti è in linea con i requisiti del programma).

#include <vector> :

Header per la creazione e la gestione di vettori dinamici, utilizzati ad esempio per realizzare la matrice di gioco (dinamica in quanto la sua dimensione è decisa dall’utente), e i vettori per la memorizzazione delle coordinate del percorso di ogni parola all’interno della griglia.

*-VARIABILI GLOBALI-*

Le variabili dichiarate globalmente sono quelle strutture che permettono in primo luogo la realizzazione della matrice di gioco (vector<vector<char>> mat), e di un’ altra matrice che servirà per determinare in quali posizioni sarà permesso effettuare le ricerche per le lettere successive, (vector<vector<vector<int>>> mat3D). Vengono dichiarate come vettori di vettori (2 e 3 dimensioni) perché la classe <vector> permette la realizzazione di array dinamici, com’è il caso della matrice di gioco, in quanto la sua dimensione non è decisa a priori ma viene determinata dall’utente. La prima è una matrice di caratteri (la sequenza inserita da tastiera), la seconda di interi, delle stesse misure della prima, ma con una dimensione in più, formata da 8 elementi (le possibili direzioni di ricerca) che possono assumere i valori 0 e 1. Vi è corrispondenza tra gli indici delle prime due dimensioni delle matrici, ma la terza dimensione della seconda è quella che indica in quali posizioni intorno all’ultima lettera trovata può proseguire la ricerca, cioè quando l’elemento nella posizione considerata è uguale a 0. La numerazione delle posizioni comincia da ovest e procede in senso antiorario. Es:

vector<vector<char>> mat:

|  |  |  |  |
| --- | --- | --- | --- |
| **D** | **E** | **N** | **I** |
| **S** | **A** | **O** | **D** |
| **R** | **I** | **T** | **G** |
| **P** | **M** | **T** | **V** |

mat[2][1] = I

mat[2][1][0] = è possibile cercare a ovest di I?

mat[2][1][1] = è possibile cercare a sud-ovest di I?

mat[2][1][2] = è possibile cercare a sud di I?

…

se mat[2][1][0] = 0 allora è permessa la ricerca a ovest di I

se mat[2][1][0] = 1 allora non è permessa la ricerca a ovest di I,

continua la ricerca a sud-ovest, poi a sud e così via.

Sempre globalmente sono definiti i vettori che rappresentano le coordinate del percorso di ogni parola (definiti come <vector> poichè le parole hanno lunghezze differenti e inoltre c’è la necessita di cancellare o aggiungere elementi dinamicamente, man mano che le parole vengono rintracciate (vector<int> righe; vector<int> colonne). Ogni volta che una lettera viene rintracciata si aggiunge un elemento in coda a entrambi i vettori (i valori degli indici delle coordinate della lettera individuata); allo stesso modo avviene la cancellazione di una coordinata: se una lettera trovata non porta poi da nessuna parte in quanto in nessuna delle posizioni che la circondano si trova la lettera successiva, bisogna provvedere alla cancellazione della coordinata dell’ultima lettera trovata e (soprattutto) all’aggiornamento della terza dimensione della matrice 3D di interi (l’elemento corrispondente viene aggiornato a 1). Quest’ultima procedura verrà piegata più nel dettaglio in seguito.

Le altre variabili globali di rilievo sono string paroladiz, che è la stringa all’interno della quale viene copiata ogni parola prelevata dal file di dizionario, e typedef struct node, ovvero la struttura dati che contiene tutte le informazioni sugli elementi da stampare, le parole trovate, i relativi punteggi e le coordinate. Essa rappresenta ogni singolo nodo della lista dei risultati, pertanto comprende il puntatore al nodo successivo della lista struct node\* nextptr; *(continua)*

typedef struct node {

string parola;

decltype(paroladiz.size()) punti = 0;

struct node\* nextptr;

vector<int> r;

vector<int> c;

} node;

Tra le altre variabili globali compaiono gli indici i e j per l’identificazione delle coordinate nelle matrici, e l’intero D (dimensione della matrice) dichiarato float per poter permettere la segnalazione di errori nel caso la dimensione della sequenza inserita dall’utente non sia il quadrato di un numero naturale 4 < N < 9.

*-ACQUISIZIONE DATI DI INPUT E RUOLO DEL MAIN-*

L’acquisizione dei parametri di ingresso avviene all’interno della funzione main. In esso ovviamente ha luogo anche la chiamata delle altre funzioni. Una volta terminato l’inserimento (con le dovute precauzioni e segnalazioni di errori nel caso alcuni dati inseriti non siano corretti) e visualizzata la matrice di gioco, si procede alla memorizzazione della prima parola del dizionario all’interno della stringa paroladiz tramite lo stream di input da file. Da questo momento in poi la procedura è la stessa per ogni parola prelevata da esso.

Per prima cosa si inizializza la matrice delle posizioni con un ciclo for, portando tutti i valori a 0 (permesse le ricerche in ogni direzione da tutte le posizioni). Se la parola è più lunga di una lettera si procede, altrimenti viene prelevata la parola successiva dal file. Nel main avviene la chiamata alla funzione di ricerca; se l’esito di tale chiamata è positivo (in essa vengono anche memorizzate le coordinate), viene calcolato il punteggio e inserito il tutto in un nodo della lista dinamica. Successivamente si re-inizializza la matrice 3D e si svuotano i vettori delle coordinate, per ricominciare la procedura con la parola successiva del dizionario.

Una volta che la lista è stata riempita con tutte le parole trovate, quest’ultima viene ordinata in ordine decrescente in base al punteggio di ogni parola e i suoi elementi stampati su un file di testo (per un numero massimo definito dall’utente) che viene visualizzato tramite una chiamata di sistema.

*-PRELIEVO PAROLE-*

Il file di dizionario viene aperto in input con diz.open(percorso.c\_str(), ios::in), dove “percorso” è la stringa in cui è stata inserita la path del file e “ios::in” indica che il file è stato aperto in lettura. Le parole del dizionario vengono prelevate una alla volta tramite la funzione getline(diz, paroladiz), che preleva una riga di testo nel file e la copia nella stringa “paroladiz”. Per ognuna avviene la verifica di esistenza nella matrice e l’eventuale calcolo del punteggio e inserimento nella lista dinamica. La condizione di fine del prelievo da file è data da while (diz.good()), funzione che verifica se nel file c’è ancora del testo da leggere.

*-RICERCA PAROLE-*

La ricerca delle parole è il nucleo del programma. Viene fatta scorrere tutta la matrice di gioco con un ciclo *for* nidificato e per ogni coordinata vengono chiamate le funzioni di ricerca. Essa inizia con l’invocazione della funzione booleana (cercaprimo(mat, i, j), alla quale vengono passati come parametri la matrice di gioco e gli indici correnti. Si effettua un confronto tra paroladiz[0] (prima lettera della parola) e la posizione passata come parametro mat[i][j]. Se il confronto ha esito positivo (le lettere sono uguali), si aggiungono ai vettori delle coordinate i valori degli indici, si aggiorna la lettera da cercare all’interno della parola (la variabile *pos*) a 1, ad indicare che la ricerca continuerà dalla seconda lettera della parola, dal momento che la prima è già stata trovata, e si procede all’invocazione della funzione trovata(mat, mat3D, paroladiz, i, j, pos).

I parametri sono la matrice di gioco, quella per le ricerche permesse, la stringa contenente la parola che si sta cercando, gli indici correnti della matrice di gioco e la variabile pos, indice della lettera della parola che si sta cercando.

Una parola è considerata trovata quando si trova l’ultima delle sue lettere, mentre non lo è quando ci si ritrova sulla prima lettera (a causa dell’indietreggiamento) e non si hanno più strade possibili nelle quali cui cercare.

La funzione “*trovata”* a sua volta chiama altre 8 funzioni, ognuna per ogni direzione di ricerca intorno alla lettera corrente. Esse funzionano esattamente come “*cercaprimo*”, ovvero effettuano un singolo confronto tra la lettera in posizione [*pos*] (la lettera da cercare, quindi paroladiz[pos]) e l’indice corrente della matrice mat[i][j]. Es di chiamata:

cercaw(mat, paroladiz, i, j, pos);

cercasw(mat, paroladiz, i, j, pos);

cercas(mat, paroladiz, i, j, pos);

...

Prima di effettuare il confronto, le funzioni verificano che la coordinata che stanno controllando non sia già presente nei vettori del percorso, invocando la funzione presente(const vector<int>, const vector<int>, int, int), dove i due vettori rappresentano le righe e le colonne occupate e gli interi sono gli indici della casella nella quale avviene la ricerca. La verifica avviene scorrendo i vettori e confrontando gli elementi con gli indici passati come parametri.

Dopodichè, se l’esito del confronto è positivo, tali funzioni procedono inoltre alla memorizzazione delle coordinate della lettera trovata negli appositi vettori, e *trovata* si richiama con gli stessi parametri, modificando gli indici con quelli della lettera trovata e incrementando *pos* di 1.

Se invece l’esito è negativo, la funzione chiamante *trovata* ne verifica il motivo. Se cio è dato dal fatto che la funzione “*presente*” ha rilevato che la posizione era già stata occupata, *trovata* richiama se stessa con gli stessi parametri ma aggiornando a 1, l’indice della matrice 3D che verifica le posizioni in cui è possibile cercare, in corrispondenza degli indici i e j correnti e della direzione di ricerca, altrimenti procede con l’invocazione delle altre funzioni di ricerca. Es: Parola da cercare: DOMAVO

|  |  |  |  |
| --- | --- | --- | --- |
| **C** | **I** | **A** | **O** |
| **V** | **A** | **D** | **O** |
| **C** | **O** | **M** | **E** |
| **S** | **T** | **A** | **I** |

Pos = 6 (cerca la 6^ lettera), i = 1, j = 0;

invocata cersase(mat, paroladiz, i, j, pos);

ovviamente presente segnalerà true, quindi cercase restituirà

false. A questo punto allora “*trovata”* setterà mat[1][0][3] = 1,

e si richiamerà con gli stessi parametri. Questa volta salterà la

ricerca a sud-est.

La stessa procedura avviene con tutte le altre funzioni di ricerca, cercaw, cercasw etc.

Se nessuna tra queste funzioni da esito positivo, allora ci si trova di fronte a un vicolo cieco:

bisogna indietreggiare e cercare altre strade.

Se pos = 1, vuol dire che stiamo cercando la seconda lettera e quindi la strada è unica e sbagliata: la parola non esiste (almeno partendo da quella posizione).

Se invece pos > 1 si comincia la procedura di identificazione della direzione da cui si è arrivati per procedere con la regressione: si confrontano le coordinate della penultima lettera trovata recuperandole dai vettori appositi con quelle dell’ultima (gli indici correnti i e j). A questo punto si nega la ricerca nella direzione che ci ha portato al vicolo ceco aggiornando a 1 l’elemento della matrice 3D che ha indici [penultima posizione righe][penultima posizione colonne][numero che identifica la direzione], si resettano i permessi di ricerca per la posizione corrente e si eliminano dai vector *righe* e *colonne* le coordinate della lettera in posizione corrente (quella che rappresenta il vicolo cieco). Ora viene richiamata la funzione “*trovata*” con gli indici i e j corrispondenti a quelli della lettera da cui si è arrivati e decrementando pos di 1. Es:

Parola da cercare: DOMAVO

A questo punto ci troviamo in mat[3][2] = A, con pos = 5;

Nessuna delle funzioni di ricerca darà esito positivo in quanto

nessuna casella adiacente contiene la lettera V: è un vicolo

cieco. Dalla ricerca successiva capiamo di essere arrivati da nord,

il che significa che indietreggiando non dovremo più cercare a

sud.

Allora mat[2][2][2] = 1 e si richiama trovata con indici [2][2] e

pos decrementata (non stiamo più cercando V ma una A in una

posizione diversa da quella in [3][2].

La ricerca termina con esito positivo quando si trova l’ultima lettera della parola.

*-CALCOLO PUNTEGGIO-*

Il calcolo del punteggio avviene con una funzione denominata

|  |  |  |  |
| --- | --- | --- | --- |
| **C** | **I** | **A** | **O** |
| **V** | **A** | **D** | **O** |
| **C** | **O** | **M** | **E** |
| **S** | **T** | **A** | **I** |

punt(paroladiz, punteggio), dove punteggio è una variabile intera inizializzata precedentemente a 0. La funzione restituisce un int che verrà poi di nuovo assegnato alla variabile punteggio. La funzione scorre tutta la stringa passata come paramentro tramite un ciclo *for* e somma i valori di ogni singola lettera (in base alle regole descritte precedentemente). Il risultato viene poi moltiplicato per il numero di lettere che costituiscono la parola.

*-CREAZIONE E INSERIMENTO NELLA LISTA-*

Una volta che si ha a disposizione tutto quello che serve, si procede alla creazione del nodo da appendere alla lista:

* Nel campo *parola* andrà la stringa *paroladiz,* che contiene la paroladella quale si è verificata precedentemente la presenza all’interno della griglia.
* In *punti* verrà inserito il punteggio, calcolato precedentemente
* Nei vettori ci andranno le coordinate delle lettere trovate, memorizzate durante l’esecuzione delle funzioni di ricerca.

Tutti questi parametri sono passati alla funzione:

void insertnode(node\*& first, node\*& last, decltype(paroladiz.size()) punteggio, const string paroladiz, vector<int> righe, vector<int> colonne);

Essa crea un nodo allocando memoria dinamicamente con la funzione *new* e assegna gli elementi alle variabili corrispondenti. Il nodo così formato viene poi “appeso” alla lista con un inserimento in coda (mantenendo così l’ordine alfabetico), assegnando anche un puntatore di riferimento all’ultimo elemento inserito.

Questo procedimento avviene per tutte le parole trovate, fino alla fine del file del dizionario.

*-ORDINAMENTO LISTA-*

La lista ora va ordinata secondo i punteggi delle parole, in ordine decrescente. Per farlo si passa (per riferimento) la testa della lista, che è concatenata a tutti gli altri nodi. Per l’ordinamento della lista si utilizza il bubble sort, uno dei più comuni algoritmi di ordinamento. Questo opera dei confronti a 2 a 2 degli elementi della lista partendo dall’inizio fino alla fine dell’elenco e se il primo membro è minore del secondo li scambia. Questa procedura si ripete finchè la lista non risulta ordinata.  
Prestazioni del Bubble sort:

* Caso peggiore (lista ordinata al contrario) = O(N^2) (N numero di operazioni)
* Caso medio = O(N^2)
* Case migliore (se la lista è già ordinata) = O(N^2)

*-STAMPA DELLA LISTA-*

La stampa della lista è affidata alla funzione:

printlist(first, N, destinazione);

*First* la testa della lista, N il numero di parole da visualizzare definito dall’utente ad inizio programma e *destinazione* la stringa che contiene il percorso per la creazione o la sovrascrittura del file su cui visualizzare i risultati.

Quest’ultima parte avviene con la funzione ofstream ris(dest), dove ris è il nome dello stream di output assegnato al file che avrà come collocazione *dest.* Una volta creato il file in scrittura, viene posizionato un altro puntatore, *currentnode* sul primo elemento della lista e viene fatto scorrere imponendo come condizione currentnode!=NULL && counter < N (finchè *currentnode* non punta a niente o non si raggiunge il numero di parole da visualizzare) e spostandolo, terminata la stampa di ogni nodo, su quello successivo.

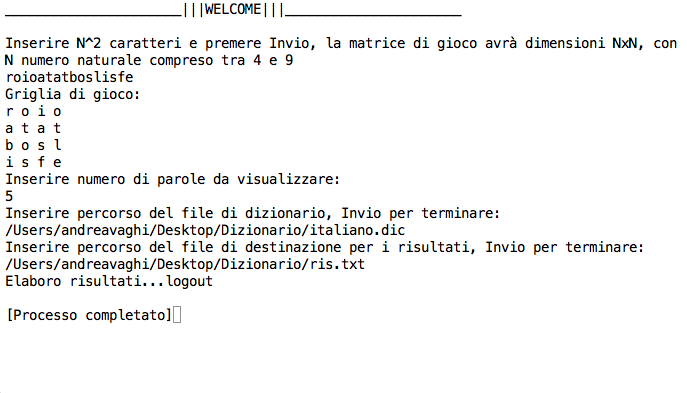
Il file testuale viene infine mostrato con una chiamata di sistema “*open”*  + la stringa di destinazione. Il programma termina a questo punto.

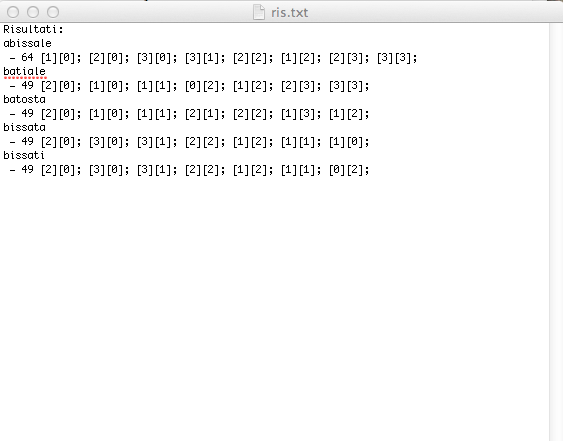
*VERIFICA:*

Prova 1

Input: roiatatboslisfe; 5; /Users/andreavaghi/Desktop/Dizionario/italiano.dic;

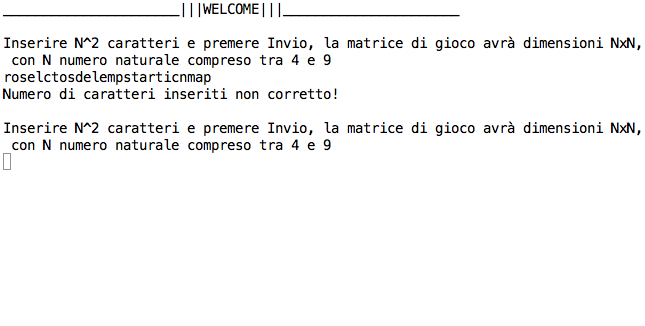
/Users/andreavaghi/Desktop/Dizionario/ris.txt





Prova 2

Input: roselctosdelempstarticnmap (dim = 26);



Prova 3

Input: roselctosdelempstarticnma (dim = 25) ; 10;

/Users/andreavaghi/Desktop/Dizionario/italiano.dic;

/Users/andreavaghi/Desktop/Dizionario/ris.txt

*(continua..)*

