

**Corso di Algoritmi e Strutture Dati**

**Progetto per la sessione estiva 2020-2021**

Studente:

Filippo Giacchetti - Matricola: 282558

*Indice:*

**1. Specifica del problema 3**

**2. Analisi del problema 4**

**3. progettazione dell’algoritmo 5**

**3.1 Scelte Progettuali 5**

**3.2 Passi dell’algoritmo 5**

**4. Implementazione del codice 7**

**3.3 Scelte Implementative 7**

**5. Testing del programma 9**

**6. Valutazione complessità del programma 15**

1. *Specifica del Problema:*

Si supponga di dover progettare un programma per un sistema informativo che gestisce i dati relativi ad un’anagrafe di autoveicoli. Il sistema permette di mantenere, per ogni veicolo, questi dati:

• veicolo: codice univoco alfanumerico rappresentato da una stringa concatenazione di quattro lettere e due numeri;

• proprietario: codice rappresentato da una stringa concatenazione di 3 lettere e tre numeri;

• modello del veicolo: stringa di massimo 20 caratteri;

• anno di immatricolazione: un numero intero.

Le informazioni sono memorizzate su un file in formato testo, secondo il seguente formato (si assumano campi separati da tabulazione o spazio):

Ad esempio:

Veicolo Proprietario Modello Anno

ZZYQ48 FCA001 Fiat 850 1973

CCGH07 DEF130 Fiat 500 1971

AAAB12 ABC001 Ford T 1908

... ... ... ...

Si scriva un programma ANSI C che esegue le seguenti elaborazioni:

1. Acquisisce il file e memorizza le relative informazioni in una struttura dati di tipo dinamico.

2. Ricerca e restituisce i dati relativi ad un dato veicolo. Ad esempio: se l’utente chiede i dati relativi

al veicolo CCGH07, il programma deve restituire le informazioni contenute nella riga

corrispondente:

Veicolo Proprietario Modello Anno

CCGH07 DEF130 Fiat 500 1971

3. Inserimento di dati relativi ad un nuovo veicolo, specificati dall’utente.

4. Rimozione di dati relativi ad un veicolo, specificati dall’utente.

5. Calcolo del valore più grande e di quello più piccolo (secondo l’ordine lessicografico) del codice

alfanumerico rappresentante il veicolo (AAAB12 e ZZYQ48, rispettivamente, nell’esempio

riportato).

Per quanto riguarda l’analisi teorica si deve fornire la complessità:

• dell’algoritmo di ricerca

• dell’algoritmo di inserimento

• dell’algoritmo di rimozione

• dell’algoritmo di calcolo del valore più grande

• dell’algoritmo di calcolo del valore più piccolo

Oltre all’analisi teorica della complessità si deve effettuare uno studio sperimentale della stessa per le operazioni sopra specificate (ricerca, inserimento, rimozione, calcolo del massimo, calcolo del minimo). Come suggerimento si può operare generando un numero N di dati-veicolo casuali (dove N rappresenta il numero di veicoli). L’analisi sperimentale deve quindi valutare la complessità al variare di N.

1. *Analisi del problema*

Il programma prende in input un file contenente quattro informazioni per riga:

1. il codice veicolo;
2. il codice proprietario;
3. identificativo del veicolo;
4. anno di immatricolazione.

Il programma analizza il file e salva i dati in memoria. Una volta che i dati sono caricati in memoria l’utente può decidere se:

* inserire un nuovo elemento;
* ricercare un elemento in memoria;
* rimuovere un elemento esistente;
* calcolare il valore più grande o quello più piccolo (secondo l’ordine lessicografico) del codice alfanumerico rappresentante il veicolo.

Il programma stampa a video l’elenco dei prodotti presi dal file a cui sono state applicate le modifiche fatte dall’utente. Le modifiche che l’utente esegue sui dati vengono anche riportate su file, in modo da tenerlo sempre aggiornato.

1. *Progettazione dell’algoritmo*
   1. *Scelte Progettuali:*

Per gestire le informazioni con cui deve lavorare il programma è stato deciso di utilizzare un array dinamico dove ogni elemento dell’array mi conterrà un dato (o oggetto) del tipo della strutturata implementata. Il dato strutturato è formato da:

* 3 stringhe: una per il codice veicolo, una per il codice proprietario e una per il nome del veicolo;
* 1 intero: per l’anno di immatricolazione.

La dimensione delle 3 stringhe è inizializzata staticamente seguendo le istruzioni presenti all’interno della specifica, quindi (contando un carattere in più per il carattere di terminazione della stringa) abbiamo:

* 7 celle per la stringa riguardante il codice veicolo;
* 7 celle per la stringa riguardante il codice proprietario;
* 21 celle per la stringa riguardante il nome del veicolo.

Per mantenere, durante l’esecuzione del programma nel momento della stampa a video delle informazioni riguardanti i veicoli e nel momento in cui vado ad aggiornare il file di input, l’indentazione dei dati uguale a quella della specifica, ovvero:

Veicolo Proprietario Modello Anno

ZZYQ48 FCA001 Fiat 850 1973

CCGH07 DEF130 Fiat 500 1971

AAAB12 ABC001 Ford T 1908

è stato deciso di riempire gli elementi vuoti della stringa rappresentante il nome del veicolo con dei caratteri spaziatura.

Durante l’esecuzione dell’algoritmo di inserimento, precisamente dopo l’acquisizione dell’ultimo valore richiesto (ovvero il nome del veicolo) sarà richiesto dal programma di premere invio una volta in più per poter andare avanti. Questa scelta è stata dettata dalla necessità di svuotare il buffer dopo quest’ultima acquisizione, in quanto se si digita un nome per il veicolo che ha più di 20 caratteri il buffer rimaneva sporco e nella successiva acquisizione (scelta voce menù) il programma andava avanti da solo.

*3.2 Passi dell’algoritmo:*

I principali passi dell’algoritmo sono:

* Acquisizione del nome del file che contiene i dati di input su cui deve lavorare il progetto. L’acquisizione è strettamente validata in modo che se il nome del file non esiste o è stato scritto male venga richiesto all’utente di riscriverlo in maniera corretta per proseguire con l’esecuzione.
* Stampa a video delle informazioni che sono state acquisite da file con la stessa indentazione che hanno su file.
* Stampa a video dell’interfaccia utente in modo che possa decidere di svolgere qualunque delle 5 operazioni richieste dalla specifica:

1. per inserire un nuovo veicolo in memoria;
2. per ricercare uno specifico veicolo in memoria tramite codice veicolo;
3. eliminare uno specifico veicolo in memoria tramite codice veicolo;
4. calcolare il valore più piccolo (secondo l’ordine lessicografico) del codice alfanumerico rappresentate il veicolo;
5. calcolare il valore più grande (secondo l’ordine lessicografico) del codice alfanumerico rappresentate il veicolo.
6. per terminare l’esecuzione del progetto e uscire;

* Dopo le operazioni di inserimento, rimozione e ricerca viene eseguita la stampa a video dell’elenco dei veicoli aggiornato, in modo da far vedere all’utente che l’operazione sia andata a buon fine.
* Dopo le operazioni di inserimento e rimozione le informazioni vengono scritte sul file che è stato acquisito all’inizio del progetto sovrascrivendolo ogni volta, in modo che rimanga sempre aggiornato.
* Dopo alcune operazioni di acquisizione in input da tastiera di dati necessari al programma, viene liberato il buffer per fare in modo di non avere problemi con acquisizioni successive.

1. *Implementazione del codice*

*4.1 Principali scelte implementative:*

Il programma inizia chiedendo il nome del file di input all’utente e successivamente (dopo aver fatto un controllo sull’esistenza del file) il programma lo scorre linea per linea (saltando la prima riga) e prende nota del numero di righe totali. Questo per fare in modo di riuscire a inizializzare dinamicamente la memoria e salvare all’interno degli elementi dell’array i valori. Successivamente viene stampato a schermo l’elenco di tutte le informazioni raccolte (con la stessa indentazione con cui sono memorizzate sul file, ovvero come viene enunciato dalla specifica) tramite la funzione “Stampa\_array”, la quale semplicemente tramite un algoritmo di visita per array attraversa tutti i suoi

elementi esattamente 1 volta e ne stampa il contenuto.

Come indicato nel capitolo precedente “Passi dell’algoritmo” è stata implementata un’interfaccia utente in modo da poter svolgere separatamente e come vuole l’utente qualunque delle 5 operazioni richieste dalla specifica.

**L’inserimento** di un nuovo elemento avviene tramite una serie di acquisizioni dall’utente riguardo le diverse informazioni necessarie per creare il nuovo veicolo (le 4 informazioni di ogni elemento).

Prima viene richiesto all’utente il numero di nuovi veicoli che vuole inserire in modo da poter

ri-allocare dinamicamente la memoria e poi ad ogni iterazione di un ciclo for viene eseguita una funzione che richiede all’utente le informazioni. Ovviamente prima di salvare in memoria le informazioni viene controllato che soddisfino i requisiti necessari e se non li soddisfano viene chiesto all’utente di ri-inserire il valore. Viene utilizzato un algoritmo di visita in modo da attraversa tutti gli elementi dell’array (a partire dalla prima cella vuota e quindi saltando i veicoli acquisiti da file) esattamente 1 volta e inserire in ogni elemento il nuovo dato di tipo “veicolo”.

L’inserimento dei dati avviene in tempo costante, mentre la visita completa dell’array avviene in tempo lineare, quindi:

**T(n) = O(k + n) = O(n)**

**La ricerca** avviene attraverso un algoritmo di ricerca lineare perché l’array non è ordinato e quindi gli elementi vengono attraversati uno dopo l’altro nell’ordine in cui sono memorizzati finché il valore cercato (che viene richiesto in input all’utente e viene validato) non viene trovato o finché non si arriva alla fine dell’array. In caso di ricerca andata a buon fine tramite la funzione “Stampa\_array” viene comunicato all’utente il veicolo con tutte le sue informazioni, altrimenti viene comunicato che il veicolo non è stato trovato.

Nel caso pessimo (valore non presente nell’array) la complessità è:

**T(n) = O(n)**

mentre nel caso ottimo (valore presente nel primo elemento dell’array) la complessità è:

**T(n) = O(1)**

**La rimozione** avviene sempre attraverso un algoritmo di ricerca lineare, in modo da andare a trovare la posizione nell’array dell’elemento che l’utente vuole eliminare. A questo punto vado tramite un algoritmo di visita a scorrere gli elementi dell’array a partite dall’elemento che voglio eliminare e ad ogni iterazione vado a scambiare l’elemento da rimuovere con il suo successivo fino a portarlo alla fine dell’array. Alla fine della funzione verrà eseguita una realloc, che essendo la combinazione di malloc e free, mi permette di andare a eliminare l’elemento contenuto nella cella dell’array perché vado a ri-dimensionarlo dinamicamente per contenere un veicolo in meno. Quindi:

* la ricerca lineare ha: T1(n) = O(n) o T1(n) = O(1);
* la visita dell’array ha: T2(n) = O(n);
* la rimozione dell’elemento ha: T3(n) = O(k);

la complessità diventa:

**T(n) = O(n)**

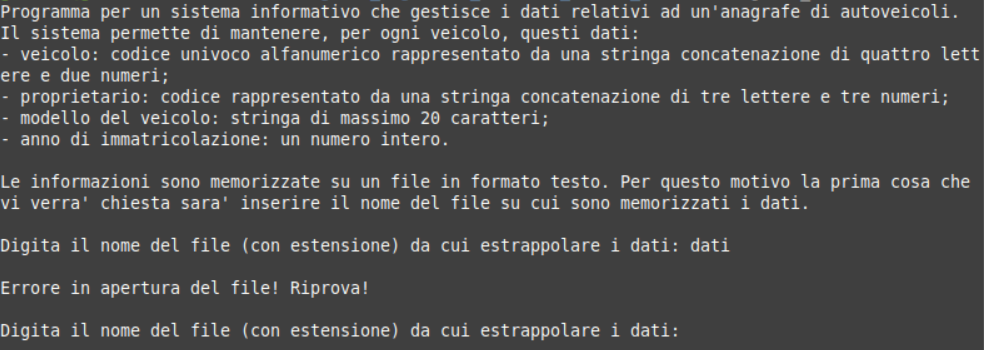
**Calcolo del valore più piccolo (secondo l’ordine lessicografico) del codice alfanumerico rappresentante il veicolo** avviene attraverso un algoritmo di visita. Alla prima iterazione prendo il primo elemento dell’array e lo confronto (secondo l’ordine lessicografico) con i successivi fino a quando non trovo un elemento più piccolo di lui oppure non arrivo alla fine dell’array. Se il confronto va a buon fine allora abbiamo il nuovo elemento più piccolo e vengono ripetute le stesse operazioni con gli elementi rimanenti fino a quando non trova un elemento a sua volta più piccolo di lui oppure non arrivo alla fine dell’array (e così via). Essendo stringhe per il confronto viene utilizzata la funzione standard “strcmp” messa a disposizione dalla libreria standard <string.h>. La complessità è:

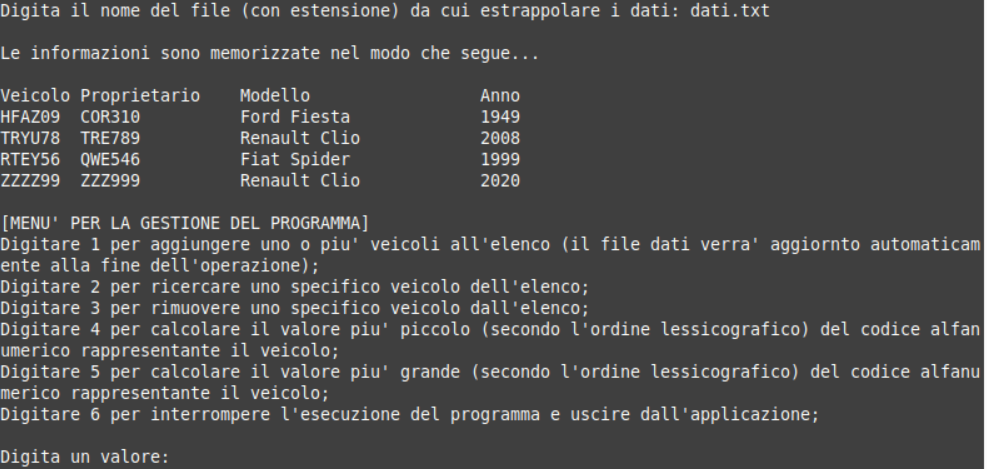
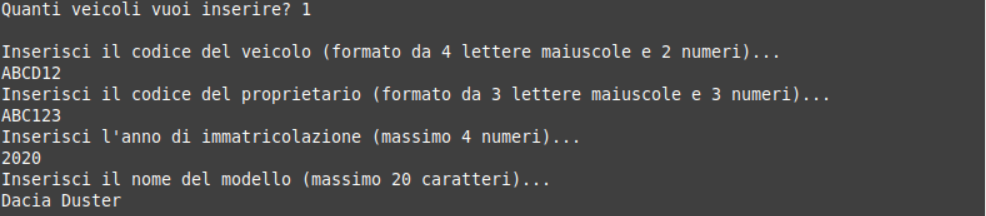
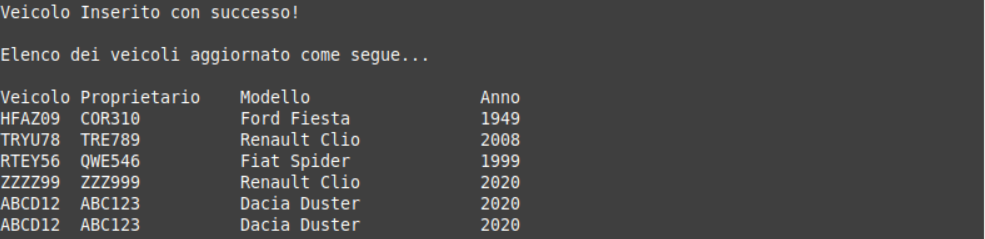
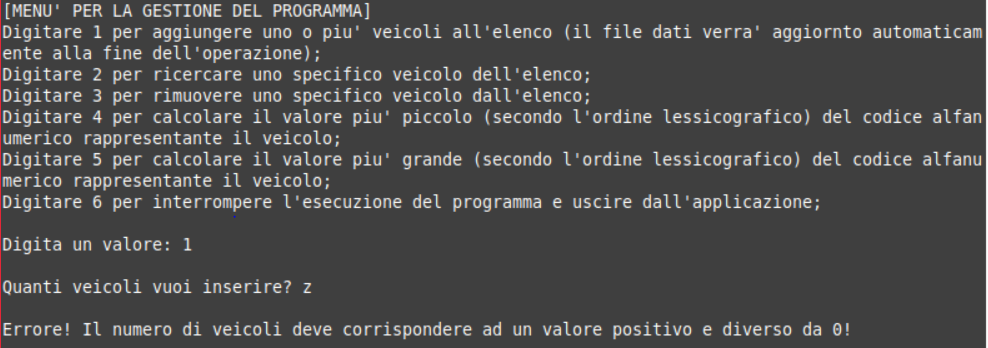
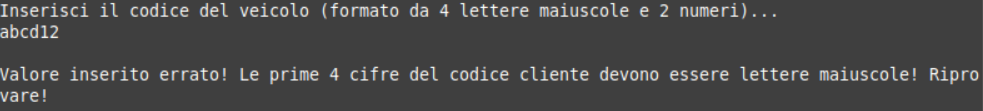
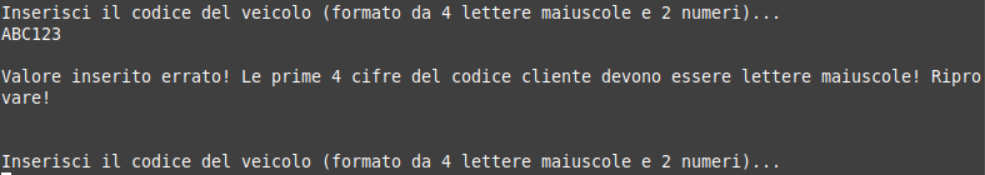
**T(n) = O(n)**

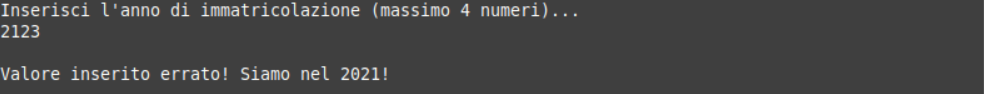
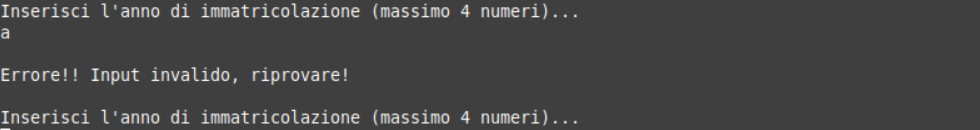
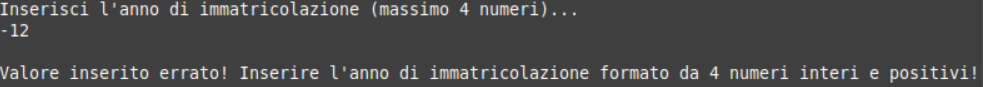
**Calcolo del valore più grande (secondo l’ordine lessicografico) del codice alfanumerico rappresentante il veicolo** avviene attraverso un algoritmo di visita. Alla prima iterazione prendo il primo elemento dell’array e lo confronto (secondo l’ordine lessicografico) con i successivi fino a quando non trovo un elemento più grande di lui oppure non arrivo alla fine dell’array. Se il confronto va a buon fine allora abbiamo il nuovo elemento più grande e vengono ripetute le stesse operazioni con gli elementi rimanenti fino a quando non trovo un elemento a sua volta più grande di lui oppure non arrivo alla fine dell’array (e così via). Essendo stringhe per il confronto viene utilizzata la funzione standard “strcmp” messa a disposizione dalla libreria standard <string.h>. La complessità è:

**T(n) = O(n)**

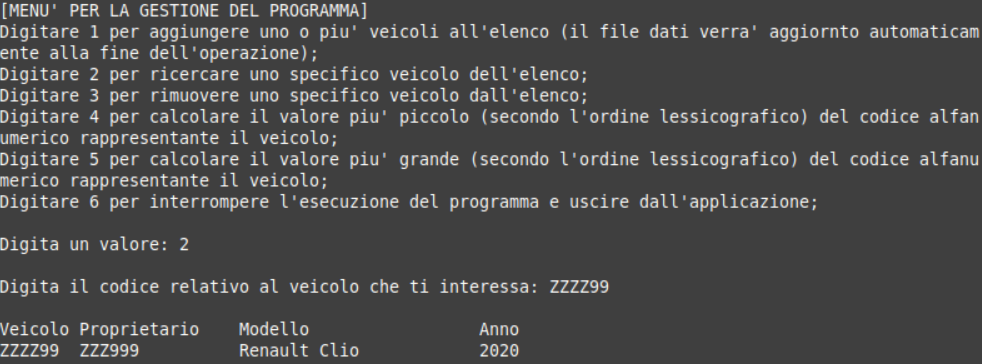
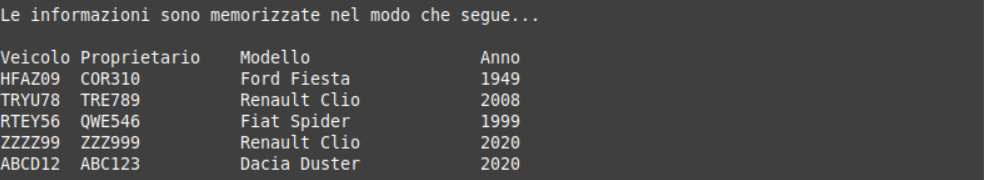
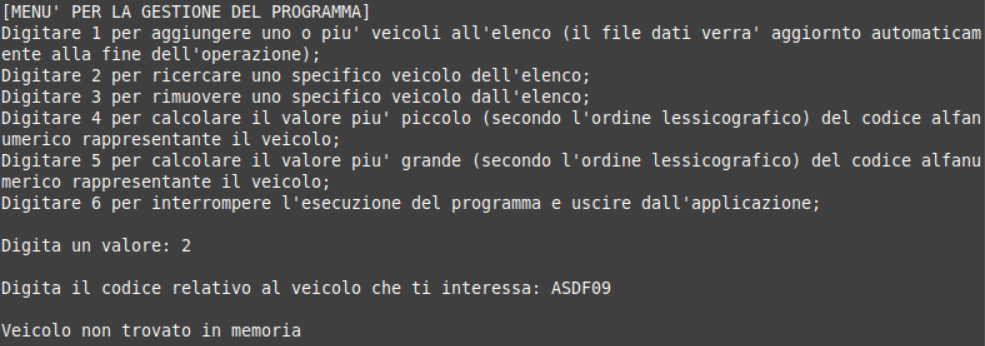
1. *Testing del programma*
2. Gestione input invalido acquisizione nome file di input per il programma;



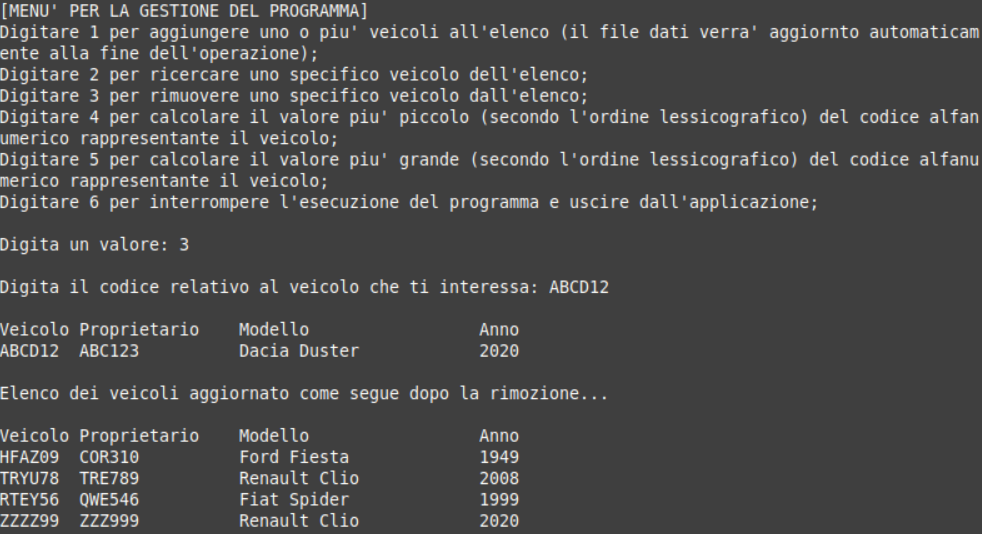
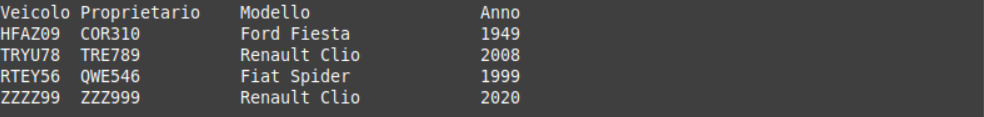
1. Presentazione programma dopo acquisizione file di input;
2. Presentazione gestione inserimento nuovo veicolo da parte del programma;
3. Gestione input invalido acquisizione nuovo veicolo;

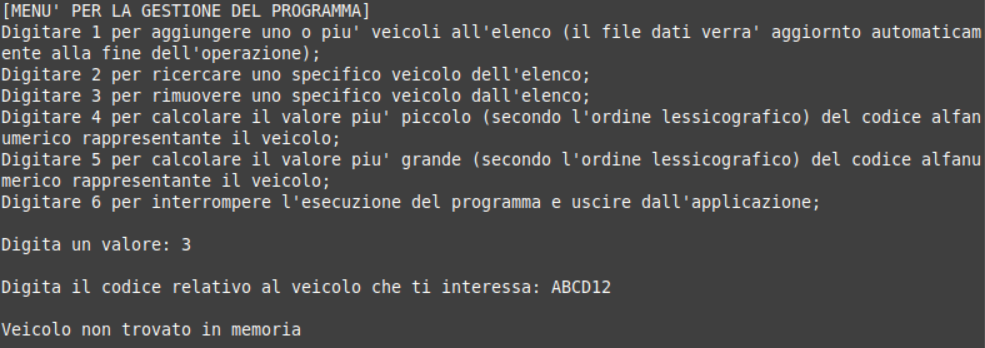


Il codice proprietario che viene acquisito dopo il codice veicolo ha la stessa gestione degli errori di quest’ultimo (ovvero quelli della pagina precedente).

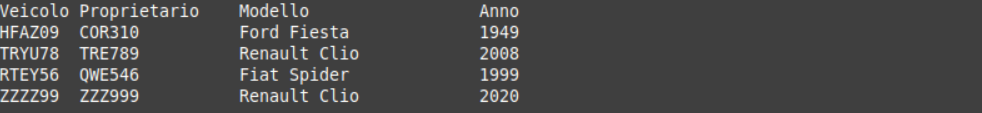
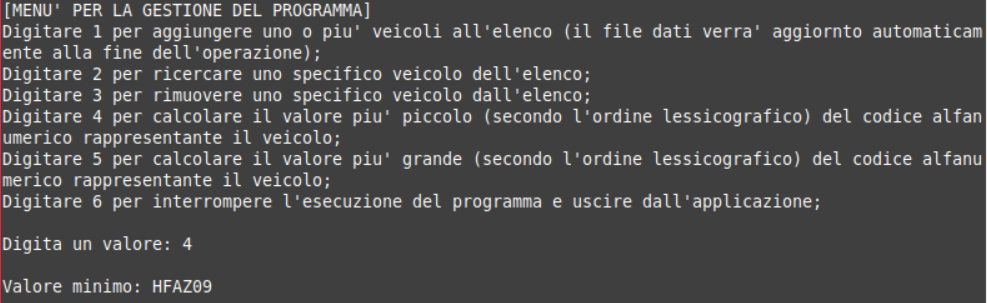
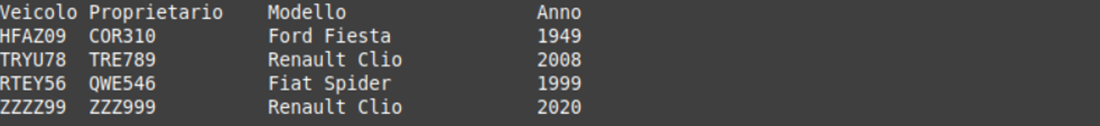
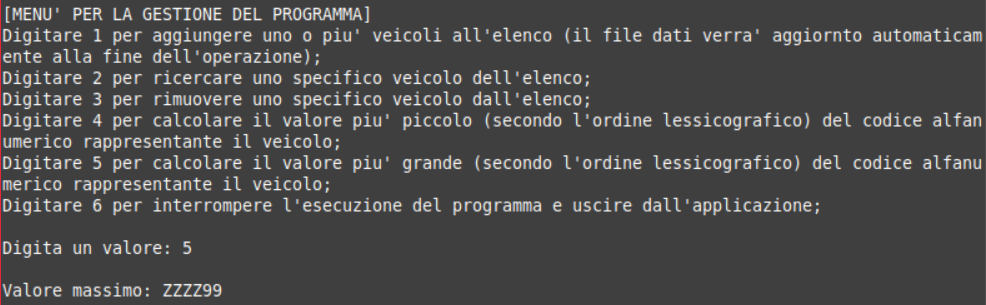
1. Presentazione gestione ricerca veicolo andata a buon fine da parte del programma;
2. Presentazione gestione ricerca veicolo non trovato in memoria;

Anche l’acquisizione del valore da ricercare è strettamente validata e i controlli che vengono eseguiti sono uguali a quelli fatti durante l’esecuzione dell’algoritmo di inserimento. Quindi nel caso in cui il codice non sia corretto viene stampato a schermo un messaggio di errore e si invita l’utente ad inserire un valore congruo alle aspettative.

1. Presentazione gestione eliminazione veicolo dalla memoria;
2. Presentazione gestione eliminazione veicolo non trovato in memoria;

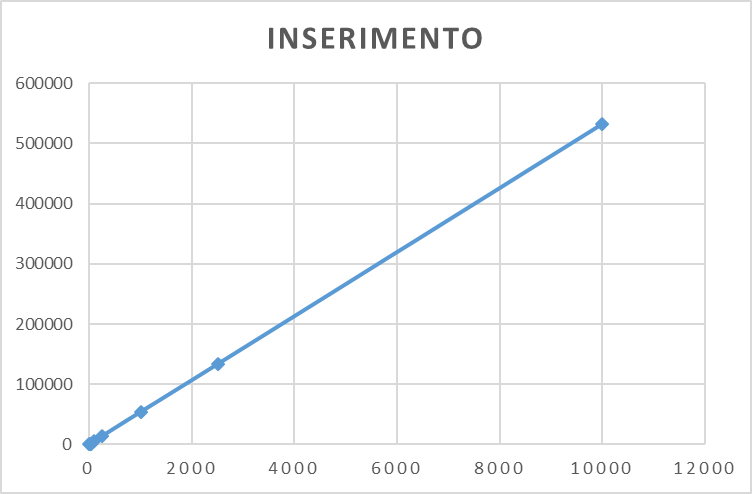


Anche l’acquisizione del valore da rimuovere dalla memoria è strettamente validato e i controlli che vengono eseguiti sono uguali a quelli fatti durante l’esecuzione dell’algoritmo di inserimento. Quindi nel caso in cui il codice non sia corretto viene stampato a schermo un messaggio di errore e si invita l’utente ad inserire un valore congruo alle aspettative.

1. Presentazione gestione calcolo del valore più piccolo (secondo l’ordine lessicografico) del codice alfanumerico rappresentante il veicolo;
2. Presentazione gestione calcolo del valore più grance (secondo l’ordine lessicografico) del codice alfanumerico rappresentante il veicolo;
3. *Valutazione complessità del programma*

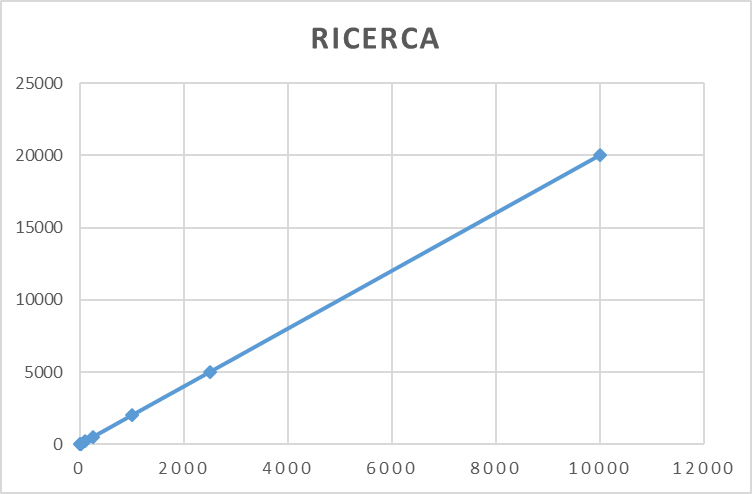
Inserimento di un elemento (progetto sviluppato tramite array)





**Lineare O(n)**

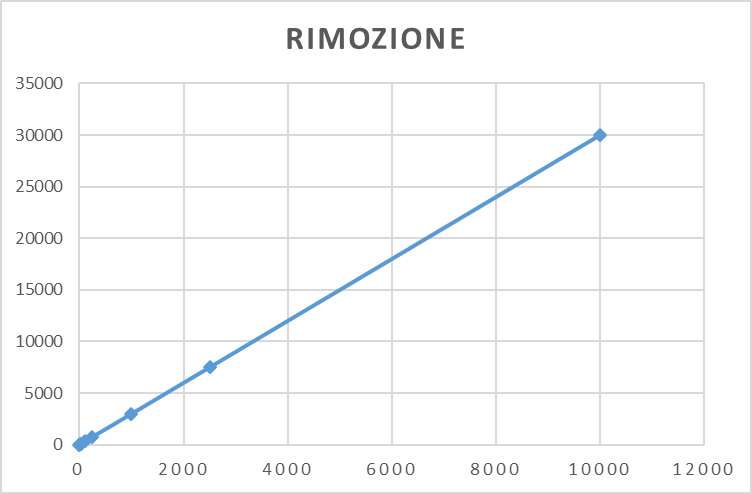
Ricerca di un elemento (progetto sviluppato tramite array)



**Lineare O(n)**

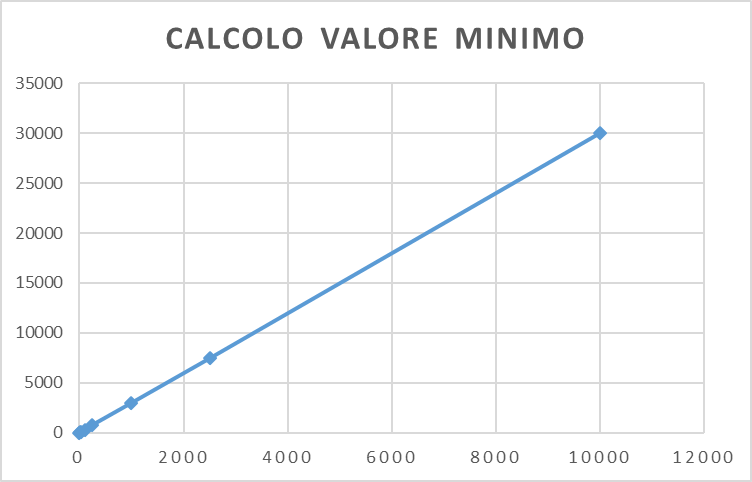
(la ricerca è stata testata scegliendo sempre l’elemento centrale nell’array)

Rimozione di un elemento (progetto sviluppato tramite array)

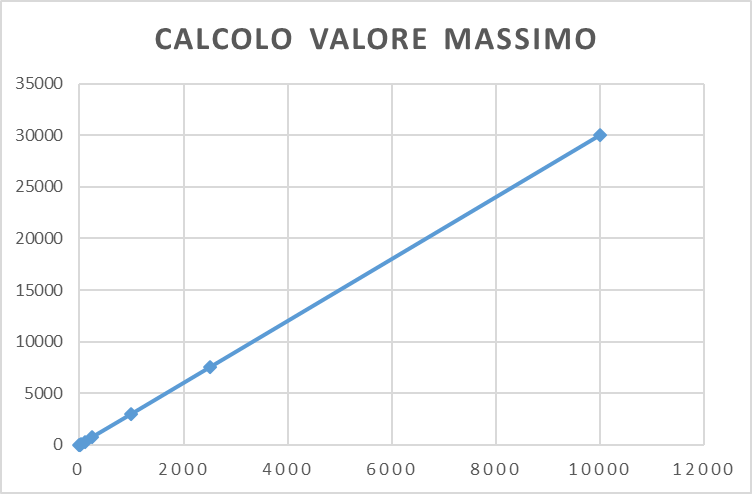


**Lineare O(n)**

(la rimozione è stata testata scegliendo sempre l’elemento centrale nell’array)

Calcolo del valore più piccolo (secondo l’ordine lessicografico) del codice alfanumerico rappresentante il veicolo (progetto sviluppato tramite array)

**Lineare O(n)**

Calcolo del valore più grande (secondo l’ordine lessicografico) del codice alfanumerico rappresentante il veicolo (progetto sviluppato tramite array)



**Lineare O(n)**