UNIVERSITATEA DE STAT DIN MOLDOVA

**Lucrare de laborator Nr.1**

**La obiectul: “Algoritmi, Structuri de date și Complexivitate”**

Realizat de: Soltan Lilian

Grupa: IA1901

Profesor: Ludmila Novac

Chișinău 2021

Sarcinile lucrării:

**„Metode de căutare în tabele”**

|  |  |
| --- | --- |
| **Termen** | **3 săptămâni** |
|  | **Să se creeze un fişier textual care conţine cel puţin 50 de înregistrări, cu cel puţin 5 câmpuri şi are cel puţin 2 tipuri de date, iar câmpul cheie trebuie să fie unic şi neordonat. Tematica structurii de date se stabileşte împreună cu profesorul la lecţiile de laborator.** |
|  | **Să fie realizat un program în C++, în care sunt implementate cel puţin două metode de căutare în tabele ordonate şi neordonate****după câmpul cheie din fişierul textual creat anterior. Analizaţi complexitatea algoritmilor. Pentru fiecare metodă de căutare de analizat lungimea medie teoretică şi lungimea practică de căutare.**  **De descris algoritmul metodelor de căutare pe paşi.** |
| 1 punct | **Metoda secvenţială de căutare** |
| 3 puncte | **Metoda de căutarea în tabele neordonate structurate arborescent** |
| 2 puncte | **Metoda binară de căutare** |
| 2 puncte | **Metoda de căutare în tabele ordonate** |
| 2 puncte | **Metoda Fibonacci de Căutare** |

***Aspecte teoretice:***

**Tipuri de Tabele:**

* Tabele neordonate

Elementele unui astfel de tabel nu sunt ordonate de valaorea cheii. Pentru a căuta un element un element cu o cheie dată, folosim algritmul de căutare secvențială.

* Tabele ordonate

Într-un astfel de tabel, toate lementele sunt odronate în ordine crescătoare a volrii cheii. Pentru a căuta un element cu o cheie dată în tabelele ordinate, se utlizează algoritmii căutării secvențiale, binare, prin interpolare și după metoda lui Fibonacci.

* Tabelele Hash

Tabelul Hash este un tabel în care poziția adresei unui element este determinată prin utilizarea unei funcții H(funcției Hash).

***Metode de căutare:***

**Căutarea secvențială sau liniară**

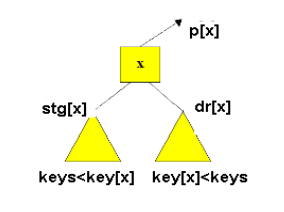
Cea mai simplă metodă de căutare a unui element dintr-un set de date neordonate, în baza valorii cheii sale, este de a analiza secvențial fiecare element al mulțimii. Căutarea continuă până când se găsește elementul dorit. Dacă întreaga mulțime este analizată, dar lementul nu este găsit, atunci cheia de căutare lipsește în mulțime.

Se compara pe rind elementele tabloului cu x pina cind, fie se gaseste egalitatea a[i]=x, fie s-a ajuns la sfirsitul tabloului.

Rezultatul teoretic așteptat: Complexitatea medie este O(n) iar în cel mai rău caz este O(n).

**Arbori binari de căutare**

Arborele este o mulţime total ordonată după valoarea cheii, având ca operaţii reprezentative: căutarea, inserarea şi ştergerea. O implementare eficientă a dicţionarelor se face cu arbori binari de căutare. Intr-un arbore binar de căutare, pentru fiecare nod, cheile din subarborele stâng sunt mai mici decât cheia din nod, care este mai mică decât cheile din subarborele drept, adică: cheie(SS(x)) < cheie(x) < cheie(SD(x)).



Proprietăţile arborilor binari de căutare:

• fiecare subarbore este un arbore binar de căutare

• cheile din subarborele stâng al unui nod sunt mai mici decât cheia din nod

• cheile din subarborele drept al unui nod sunt mai mari decât cheia din nod

• cheia minimă (/maximă) se află în cel mai din stânga (/dreapta) nod

• înălţimea arborelui binar de căutare este cuprinsă între log2n şi n-1.

Rezultatul teoretic așteptat: Complexitatea medie este O(log(n)) iar cel mai rău caz O(n).

**Căutarea Binară**

**Algoritmul de căutare binară** este un [algoritm de căutare](https://ro.wikipedia.org/w/index.php?title=Algoritm_de_c%C4%83utare&action=edit&redlink=1" \o "Algoritm de căutare — pagină inexistentă) folosit pentru a găsi un element într-o listă [ordonată](https://ro.wikipedia.org/wiki/Sortare_(informatic%C4%83)" \o "Sortare (informatică)) (tablou unidimensional/vector). Algoritmul funcționează pe baza tehnicii [divide et impera](https://ro.wikipedia.org/wiki/Divide_et_impera_(informatic%C4%83)). Valoarea căutată este comparată cu cea a elementului din mijlocul listei. Dacă e egală cu cea a acelui element, algoritmul se termină. Dacă e mai mare decât acea valoare, algoritmul se reia, de la mijlocul listei până la sfârșit, iar dacă e mai mică, algoritmul se reia pentru elementele de la începutul listei până la mijloc. Întrucât la fiecare pas cardinalul mulțimii de elemente în care se efectuează căutarea se înjumătățește, algoritmul are [complexitate](https://ro.wikipedia.org/w/index.php?title=Complexitate_(informatic%C4%83)&action=edit&redlink=1" \o "Complexitate (informatică) — pagină inexistentă) [logaritmică](https://ro.wikipedia.org/wiki/Logaritm" \o "Logaritm).

Rezultatul teoretic așteptat: Complexitatea medie este O(log(n)) iar cel mai rău caz O(log(n)).

**Metoda de căutare Fibonacci**

Se dă un tabel cu înregistrări R1, R2, ..., RN. Cheile, în baza cărora se face căutarea, sunt ordonate crescător: K1<K2< ... <KN. Căutarea se face după o careva cheie K din șirul cheilor ordonate.  
 **Algoritm:** Sirul numerelor Fibonacci se defineste recursiv astfel: F0=0, F1=1, Fk+2=Fk+1+Fk, k>=0.   
Lungimea tabelului este egala cu ultimul numar Fibonacci si este N=Fm. Daca lungimea tabelului nu este un numar Fibonacci se ia in calitate de Fm cel mai mic numar din F, imediat mai mare decat n.     
*Pas1* (Initializare):       i=Fk, p=Fk-1, q=Fk-2, unde p si q sunt doua numere Fibonacci consecutive.  
*Pas2* (Comparare):    Daca K<Ki, atunci mergi la pasul 3; daca K>Ki atunci mergi la pasul 4; si daca K=Ki algoritmul se finiseaza cu succes.   
*Pas3* (Descrestere i):  Daca q=0, atunci avem insucces. In caz contrar se fac urmatoarele atribuiri: i=i-q si p=q; q=p-q; apoi ne intoarcem la pasul 2.  
*Pas4* (Incrementare i): Daca p=1, atunci avem insucces. In caz contrar se fac urmatoarele atribuiri i=i+q p=p-q si apoi q=q-p (p se ia acel obtinut acum); apoi ne intoarcem la pasul 2.

Rezultatul teoretic așteptat: Complexitatea medie este O(log(n)) iar în cel mai rău caz O(log(n)).

**Metoda de căutare Hash(Dispersie)**

Algoritmii de căutare care folosesc hash constau din două părți distincte.

1. Prima parte este de a calcula o funcție hash care transformă cheia de căutare într-o adresă de tabel. In mod ideal, chei diferite s-ar mapa la adrese diferite, dar de multe ori două sau mai multe chei diferite pot fi distribuite la aceeași adresă de tabel.

2. A doua parte a unui algoritm de căutare care foloseşte hashing este un proces de rezolvare a coliziunilor. Una din metodele de de rezolvare a coliziunilor pe care o vom studia utilizează liste înlănţuite, și este, prin urmare, imediat utilizabilă în situații dinamice în care numărul de chei de căutare este dificil de prezis în avans.

Rezultatul teoretic așteptat: Complexitatea medie este O(1) iar în cel mai rău caz este O(n).

***Codul sursă:***

Programul de bază startsearch.cpp

#include <iostream>

#include <fstream>

#include <limits>

#include <vector>

#include <algorithm>

#include <string>

#include <math.h>

#include "search.hpp"

#include "metodaBinara.hpp"//functia de cautare binara

#include "metodaFibonacci.hpp"//functia de cautare fibonacci

#include "Arbore.hpp"//functia de cautare in arbori binari

#include "metodaHash.hpp"//functia de cautare in tabel hash

using namespace std;

int main(){

Automobil s[52];

int i,n=52,x,count;

bool found;

cout<<"Cautare ID:";

cin>>x;

for(i=0;i<n;i++)

s[i]=Automobil("Automobile-ord.txt");//atribuire din fisier

for(i=0;i<n;i++){

if(s[i].id==x){//cautare secventiala

found=true;break;}

else

found=false;

}

if(found){

cout<<"Au fost realizate "<<i <<" iteratii cu metoda de cautare Secventiala"<<endl;

cout<<"Complexitatea teoretica de cautare este: " << n / 2<<endl;

s[i].afisare();}

else

cout<<"Cautare Secventiala eronata"<<endl;

int bin=caut\_binara(x,n,s,&count);

if(bin !=-1){//cautarea binara

cout<<"Au fost realizate "<<count<<" iteratii cu metoda de cautare Binara"<<endl;

cout<<"Complexitatea teoretica de cautare este: " << (log(n)/log(2) + 2)<<endl;

s[bin].afisare();

cout<<endl;

}

else

cout<<"Cautare Binara eronota"<<endl;

count=0;

int fib=fibMonaccianSearch(x,n,s,&count);

if(fib !=-1){//cautrea fibonacci

cout<<"Au fost realizate "<<count<<" iteratii cu metoda de cautare Fiboncci"<<endl;

cout<<"Complexitatea teoretica de cautare este: " << (log(n)/log(2) + 2)<<endl;

s[fib].afisare();

}

else

cout<<"Cautare Fibonacii eronata"<<endl;

Arbore arb;//cream arborele;

for(i=0;i<n;i++)

arb.insert(s[i]);

arb.search(x);

count=0;

HashMap hash; //cream clasa HASH

for(i=0;i<n;i++)

hash.Insert(s[i].id, i);

i=hash.Search(x,&count);

if(i!=-1){

cout<<"Au fost realizate "<<cont<<" iteratii cu metoda de cautare HASH"<<endl;

cout<<"Complexitatea teoretica de cautare este: " << (n+1)/2<<endl;

s[i].afisare();}

else

cout<<"Cautare Hash eronata"<<endl;

}

**Codul pentru clasa de bază:**

#include <iostream>

#include <fstream>

using namespace std;

int poz=0;

class Automobil{

public:

int id;

string denumire;

string model;

int anproductie;

int motor;

string tipcombustibil;

Automobil(){id=-1;};

Automobil(Automobil &S){

id=S.id;

denumire=S.denumire;

anproductie=S.anproductie;

motor=S.motor;

tipcombustibil=S.tipcombustibil;

} ;

Automobil(string file){

poz++;//linia curenta

int c=0;//contor de linii citite pina la linia curenta

ifstream f;

f.open(file);

if(f.fail()){//testul de deschidere fisier

cout << "Input file opening failed.\n";

exit(1);

}

string line;

while(c<poz){//citim in gol pina ajungem la linia curenta

getline(f, line);

c++;}

f.close();

//operam cu ultima linie citita

size\_t s;//salvam pozitia caracterului in linia citita

string sir;

///id

s=line.find(" ");//cautam prima despartire

sir=line.substr(0,s);//salvam subsirul dat

id=atoi(sir.c\_str());//convertim subsirul in numar intreg

///denumire

line=line.substr(s+1);//excludem subsirul citit

s=line.find(" ");//cautam prima despartire

denumire=line.substr(0,s);//salvam subsirul dat

///model

line=line.substr(s+1);//excludem subsirul citit

s=line.find(" ");//cautam prima despartire

model=line.substr(0,s);//salvam subsirul dat

///anproductie

line=line.substr(s+1);//excludem subsirul citit

s=line.find(" ");//cautam prima despartire

sir=line.substr(0,s);//salvam subsirul dat

anproductie=atoi(sir.c\_str());//convertim subsirul in numar intreg

///motor

line=line.substr(s+1);//excludem subsirul citit

s=line.find(" ");//cautam prima despartire

sir=line.substr(0,s);//salvam subsirul dat

motor=atoi(sir.c\_str());//convertim subsirul in numar intreg

///tipcombustibil

line=line.substr(s+1);//excludem subsirul citit

s=line.find(" ");//cautam prima despartire

tipcombustibil=line.substr(0,s);//salvam subsirul dat }

void afisare(){

cout<<"ID Denumirea Model AnProductie Motor TipCombustibil"<<endl;

cout<<id<<" "<<denumire<<" "<<model<<" "<<anproductie<<" "<<motor<<" "<<tipcombustibil<<endl<<endl;}

};

***Codul pentru metoda Hash***

const int TABLE\_SIZE = 50;//numarul de locuri in tabel hash

/\*

\* HashEntry Class Declaration

\*/

class HashEntry

{

public:

int key;

int value;

HashEntry(int key, int value)

{

this->key = key;

this->value = value;

}

};

/\*

\* HashMap Class Declaration

\*/

class HashMap

{

private:

HashEntry \*\*table;

int size;

public:

HashMap()

{ size=0;

table = new HashEntry \* [TABLE\_SIZE];

for (int i = 0; i< TABLE\_SIZE; i++)

{

table[i] = NULL;

}

}

/\*

\* Hash Function

\*/

int HashFunc(int key)

{

return key % TABLE\_SIZE;

}

/\*

\* Insert Element at a key

\*/

void Insert(int key, int value)

{

int hash = HashFunc(key);

HashEntry \*temp =new HashEntry(key, value);

while (table[hash] != NULL && table[hash]->key != key && table[hash]->key != -1)

{

hash++;

hash %= TABLE\_SIZE;

}

if (table[hash] == NULL || table[hash]->key == -1)

size++;

table[hash] = temp;

}

/\*

\* Search Element at a key

\*/

int Search(int key,int \*s)

{

int hash = HashFunc(key);

int counter=0;

while (table[hash] != NULL )

{ if(counter++>TABLE\_SIZE) //to avoid infinite loop

return -1;

//if node found return its value

if(table[hash]->key == key)

return table[hash]->value;

hash++;

hash %= TABLE\_SIZE;

\*s=\*s+1;

}

return -1;

}

/\*

\* Remove Element at a key

\*/

void Remove(int key)

{

int hash = HashFunc(key);

while (table[hash] != NULL)

{

if (table[hash]->key == key)

break;

hash = HashFunc(hash + 1);

}

if (table[hash] == NULL)

{

cout<<"Nici un element gasitp entru cheia data"<<key<<endl;

return;

}

else

{

delete table[hash];

}

cout<<"Elementul a fost sters"<<endl;

}

~HashMap()

{

for (int i = 0; i < TABLE\_SIZE; i++)

{

if (table[i] != NULL)

delete table[i];

delete[] table;

}

}

};

***Codul pentru metoda de căutare Binară***

using namespace std;

int cont=0;

//definirea arborelui binar

int caut\_binara(int x,int n1,Automobil num[],int \*cont){//cautare binara

int s=0,d=n1-1,m=0;

bool flag=false;

\*cont=0;

while(s<=d){

m=(s+d)/2;

if(num[m].id>x)

d=m-1;

if(num[m].id<x)

s=m+1;

if(num[m].id==x)

{flag=true;break;}

\*cont=\*cont+1;

}

if(flag)

return m;

else

return -1;

}

**Codul pentru căutarea Fibonacci:**

int min(int x, int y) { return (x<=y)? x : y; }

/\* Returneaza indexul lui x daca se gaseste, else return -1 \*/

int fibMonaccianSearch( int x,int n,Automobil num[],int \*cont)

{

/\* Initializarea numerelor fibonacci ...,m-2,m-1,m,...\*/

int fibMMm2 = 0; // (m-2) Fibonacci No.

int fibMMm1 = 1; // (m-1) Fibonacci No.

int fibM = fibMMm2 + fibMMm1; // m Fibonacci

/\* fibM salveaza cel mai mic numar Fibonacci care este mai mare sau egal cu n \*/

while (fibM < n)

{

fibMMm2 = fibMMm1;

fibMMm1 = fibM;

fibM = fibMMm2 + fibMMm1;

}

// Marcam raza de eliminare din fata

int offset = -1;

/\* atita timp cit sunt elemente comparam vectorul cu x. Cind fibM este 1,

fibMm2 devine 0 \*/

while (fibM > 1)

{

// Verificam daca fibMm2 este o locatie valida

int i = min(offset+fibMMm2, n-1);

/\* Daca x este mai mare ca valorea din index fibMm2,

taiem vectorul pina la i \*/

if (num[i].id < x)

{

fibM = fibMMm1;

fibMMm1 = fibMMm2;

fibMMm2 = fibM - fibMMm1;

offset = i;

}

/\* Daca x este mai mare ca valoarea de pe indexul fibMm2,

taiem vectorul dupa i+1 \*/

else if (num[i].id > x)

{

fibM = fibMMm2;

fibMMm1 = fibMMm1 - fibMMm2;

fibMMm2 = fibM - fibMMm1;

}

/\* elementul sa gasit, returnam indexul \*/

else return i;

\*cont=\*cont+1;

}

/\* compram ultimul element cu x \*/

if(fibMMm1 && num[offset+1].id==x )return offset+1;

/\*element negasit. return -1 \*/

return -1;

}

**Codul pentru Arrbori binari de căutare:**

using namespace std;

class Arbore{

struct Node{

Node(Automobil x) : data(x){}

Automobil data;

struct Node\* left;

struct Node\* right;

};

Node\* root;

Node\* makeEmpty(Node\* t){

if(t == NULL)

return NULL;{

makeEmpty(t->left);

makeEmpty(t->right);

delete t;

}

return NULL;

}

Node\* insert(Automobil x, Node\*& t){

if(t == NULL){

t = new struct Node(x);

t->data = x;

t->left = t->right = NULL;

}

else if(x.id < t->data.id)

t->left = insert(x, t->left);

else if(x.id > t->data.id)

t->right = insert(x, t->right);

return t;

}

Node\* find(Node\* t, int x){

int nr=0;

while(t!=NULL){

if( t->data.id == x){

cout<<"Au fost reazliate "<<nr<<" iteratii cu metoda de cautare Arbore"<<endl;

cout<<"Complexitatea teoretica de cautare este: " << (log(nr)/log(2) + 2)<<endl;

t->data.afisare();

return t;}

else if(x < t->data.id)

t=t->left;

else if(x > t->data.id)

t=t->right;

nr++;}

cout<<"Cautare dupa Arbore eronata eronata"<<endl;

return t;

}

public:

Arbore(){

root = NULL;

}

~Arbore(){

root = makeEmpty(root);

}

void insert(Automobil x){

root = insert(x, root);

}

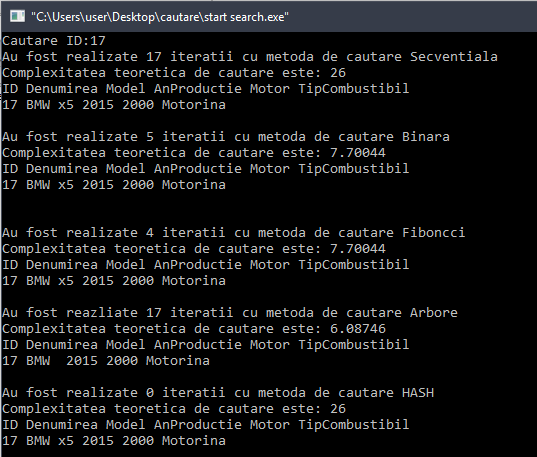
void search(int x){

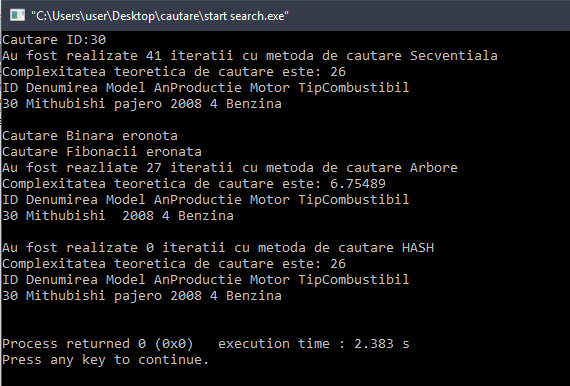
root = find(root, x);

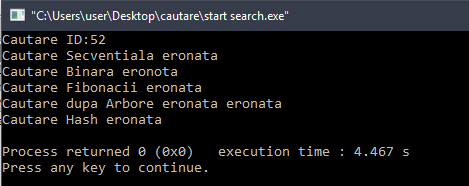
}

};

**Screenshot-uri:**







**Concluzie:**

În acest laborator, am avut posibilitatea de a studia dar și de a face căutare într-un fișier text cu ajutorul a mai multor metode. După o mica analiză a rezultatelor așteptate, și a rezultatelor obținute pot spune că metoda binară și Fibonacci pot realiza căutare doar în fișiere ordonate, iar celelalte metode indiferent de tipul fișierului pot realiza o căutare cu success.

La fel, am observat că cea mai rea metodă este liniară în cazul în care elemental căutat se află pe ultima poziție. Cea mai bună metodă de căutatare este Hash, deoarece cheia se tranformă într-o adresă din tabel.

**Fișierelul textual ordonat:**

ID Denumirea Model AnProductie Motor TipCombustibil

1 Audi a2 2002 2000 Motorina

2 Audi a4 2003 2500 Motorina

3 Audi a4 2010 4200 Benzina

4 Audi a5 2012 2000 Benzina

5 Audi a6 2005 4200 Motorina

6 Audi a8 2009 3000 Motorina

7 Audi q5 2015 2000 Motorina

8 Audi q7 2012 4500 Benzina

9 BMW 320 2002 2000 Motorina

10 BMW 325 2004 2500 Benzina

11 BMW 350 2006 5000 Benzina

12 BMW 525 2009 2500 Gaz/Benzina

13 BMW 530 2008 3000 Motorina

14 BMW 725 2010 4000 Gaz/Benzina

15 BMW 755 2005 5200 Benzina

16 BMW x5 2003 3000 Motorina

17 BMW x5 2015 2000 Motorina

18 Cadilac escalade 2005 5000 Benzina

19 Cadilac ctv 2012 4500 Motorina

20 Cadilac eldorado 1970 4000 Benzina

21 Daihatsu morena 1999 1600 Benzina

22 Daihatsu macro 2001 1600 Benzina

23 Daihatsu timpo 2009 1500 Motorina

24 Lexus ls200 2005 3200 Benzina

25 Lexus lc450 2010 4500 Motorina

26 Lexus fx350 2015 3500 Motorina

27 Lexus fx550 2018 5500 Benzina

28 MercedesBenz e-class 1998 3200 Benzina

28 MercedesBenz e-class 2018 3000 Motorina

30 MercedesBenz c-clasas 2005 2500 Motorina

31 MercedesBenz c-class 2015 2000 Benzina

32 MercedesBenz ml-class 2005 2700 Motorina

32 MercedesBenz g-class 2009 5500 Benzina

33 MercedesBenz glc-class 2014 4000 Benzina

34 MercedesBenz gl-class 2008 5000 Motorina

35 MercedesBenz s-class 2010 3200 Motorina

36 Mazda 3 2005 2000 Motorina

37 Mazda 6 2012 2000 Benzina

38 Mazda cx-5 2011 2200 Motorina

39 Mazda cx-7 2010 3000 Benzina

40 Mithubishi pajero 2008 4400 Benzina

41 Mithubishi lancer 2012 3000 Benzina

42 Toyota auris 2009 2000 Motorina

43 Toyota avensis 2006 2200 Motorina

44 Toyota landcruiser 2010 4700 Benzina

45 Volvo s60 2006 2400 Motorina

46 Volvo s80 2003 2400 Motorina

47 Volvo xc60 2012 2000 Motorina

48 Volvo xc70 2008 2500 Benzina

49 Volvo xc90 2003 2400 Motorina

50 Volvo s90 2018 3200 Motorina

**Fișierul textual neordonat:**

ID Denumirea Model AnProductie Motor TipCombustibil

50 Audi a2 2002 2.0 Motorina

45 Audi a4 2003 2.5 Motorina

4 Audi a4 2010 4.2 Benzina

3 Audi a5 2012 2.0 Benzina

9 Audi a6 2005 4.2 Motorina

6 Audi a8 2009 3.0 Motorina

47 Audi q5 2015 2.0 Motorina

12 Audi q7 2012 4.5 Benzina

5 BMW 320 2002 2.0 Motorina

10 BMW 325 2004 2.5 Benzina

49 BMW 350 2006 5.0 Benzina

8 BMW 525 2009 2.5 Gaz/Benzina

13 BMW 530 2008 3.0 Motorina

18 BMW 725 2010 4.0 Gaz/Benzina

25 BMW 755 2005 5.2 Benzina

16 BMW x5 2003 3.0 Motorina

17 BMW x5 2015 2.0 Motorina

14 Cadilac escalade 2005 5.0 Benzina

23 Cadilac ctv 2012 4.5 Motorina

36 Cadilac eldorado 1970 4.0 Benzina

21 Daihatsu morena 1999 1.6 Benzina

42 Daihatsu macro 2001 1.6 Benzina

19 Daihatsu timpo 2009 1.5 Motorina

24 Lexus ls200 2005 3.2 Benzina

15 Lexus lc450 2010 4.5 Motorina

26 Lexus fx350 2015 3.5 Motorina

29 Lexus fx550 2018 5.5 Benzina

28 MercedesBenz e-class 1998 3.2 Benzina

27 MercedesBenz e-class 2018 3.0 Motorina

40 MercedesBenz c-clasas 2005 2.5 Motorina

31 MercedesBenz c-class 2015 2.0 Benzina

5 MercedesBenz ml-class 2005 2.7 Motorina

48 MercedesBenz g-class 2009 5.5 Benzina

33 MercedesBenz glc-class 2014 4.0 Benzina

34 MercedesBenz gl-class 2008 5.0 Motorina

35 MercedesBenz s-class 2010 3.2 Motorina

20 Mazda 3 2005 2.0 Motorina

37 Mazda 6 2012 2.0 Benzina

38 Mazda cx-5 2011 2.2 Motorina

39 Mazda cx-7 2010 3.0 Benzina

30 Mithubishi pajero 2008 4.4 Benzina

41 Mithubishi lancer 2012 3.0 Benzina

22 Toyota auris 2009 2.0 Motorina

43 Toyota avensis 2006 2.2 Motorina

44 Toyota landcruiser 2010 4.7 Benzina

2 Volvo s60 2006 2.4 Motorina

46 Volvo s80 2003 2.4 Motorina

7 Volvo xc60 2012 2.0 Motorina

32 Volvo xc70 2008 2.5 Benzina

11 Volvo xc90 2003 2.4 Motorina

1 Volvo s90 2018 3.2 Motorina