Tema 1 – Structuri de date (seria CB)

Tabele hash

Responsabili tema:	Irina Mocanu
Data publicarii:	**30.03.2021**
Termenul de predare:	**13.04.2021 ora 23:59** Se accepta teme trimise cu penalizare 10 puncte /zi (din max 100 puncte) până la data 16.04.2021 ora 23:59.

1. Introducere

In momentul accesarii unei pagini web este lansat un DNS QUERY catre un server DNS, pentru a obtine IP-ul paginii web ce se doreste a fi accesata, urmand ca ulterior sa se initieze o conexiune care va aduce toata informatia paginii respective. Cand cererea ajunge la serverul DNS, acesta va cauta in memorie o intrare de tipul (Name, IP), unde "Name" este adresa paginii web, iar IP este IP-ul asociat.

2. Cerinta

Sunteti ingineri la Google si trebuie sa implementati structura memoriei unui server DNS. Prioritatea principala este rapiditatea accesului datelor. Problema se poate rezolva in mai multe moduri.

O varianta triviala ar fi ca toate inregistrarile (Name, IP) sa fie memorate intr-un vector.

(Name, IP)	(Name, IP)	(Name, IP)	(Name, IP)	(Name, IP)	(Name, IP)	.....	(google.com, 64.233.160.4)	(Name, IP)	(Name, IP)

In cazul unui QUERY DNS de forma (google.ro, IP=?), serverul DNS trebuie sa itereze prin vector pana gaseste inregistrarea corespunzatoare. Avand in vedere faptul ca majoritatea serverelor DNS au milioane de intrari, si sunt accesate simultan de sute de utilizatori, aceasta solutie nu numai ca este mare consumatoare de timp, cautarea executandu-se proportional cu dimensiunea vectorului, ci este si aproape imposibil de implementat din cauza slabei redundante.

Totusi, deoarece toti ne dorim ca in momentul accesarii "www.google.com", pagina web sa se incarce imediat, trebuie sa existe o alta solutie mult mai buna. Aceasta este **hashtable (tabela de dispersie)**. Un **hashtable** este o structura de date optimizata pentru functia de cautare. Acest lucru se realizeaza transformand cheia intr-un hash (folosind o functie hash). Totusi, tabelele de hash sunt foarte utile in cazul in care se stocheaza cantitati mari de date, a caror dimensiune (marime a volumului de date) poate fi anticipat.

Functia hash trebuie aleasa astfel incat sa se minimizeze numarul coliziunilor (valori diferite care produc aceleasi hash-uri). Coliziunile apar in mod inerent, deoarece lungimea hash-ului este fixa, iar obiectele de stocare pot avea lungimi și continut arbitrare. In cazul aparitiei unei coliziuni, valorile se stocheaza pe aceeasi pozitie (in acelasi bucket - lista). In acest caz, cautarea se va reduce la compararea valorilor efective in cadrul bucketului respectiv.

3. Implementare

Hashtable-ul va fi reprezentat ca un vector cu M elemente (bucket-uri) – fiecare bucket este reprezentat printr-o lista dublu inlantuita circulara generica. Elementele din liste vor fi de forma (Key, Value). Functie hash va calcula restul impartirii sumei caracterelor ce fac parte din cheile (Key) stocate in hash la numarul maxim de bucketuri ce pot fi stocate in hashtable (M).

Operatiile efectuate in hashtable sunt:

Operatia	**Descrierea operatiei**
put <Key Value>	adauga perechea (Key, Value) in hashtable (daca Key exista in hastable, aceasta nu va mai fi adăugata)
get <Key >	intoarce valoarea corespunzatoare cheii "Key"; daca “Key” nu exista, intoarce NULL.
remove <Key >	sterge perechea (Key, Value) din hashtable (in cazul in care aceasta exista)
find <Key >	caută Key in hashtable și întoarce True/False.
print	afiseaza toate valorile “**Value**” din hashtable; pentru fiecare bucket (lista) nevid; se va afisa indicele si toate elementele acestuia (**Value**). Elementele listei sunt afisate toate pe aceeasi linie, separate de un spatiu; indicii sunt considerati de la 0.
print_bucket <index_bucket>	afiseaza **valorile ("Value")** din bucketul cu indicele *index bucket*, pe o singura linie; separarea elementelor afisate se face numai prin spatii; daca lista asociata bucket-ului este vida se va afisa VIDA.

3.1. Rulare:

Programul va fi rulat:

./tema1 M hash$_i$.in hash$_i$.out

unde:

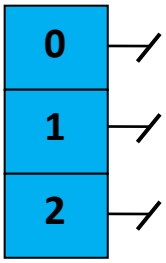
- M – reprezinta numarul de bucket-uri din tabela hash
- hash$_i$.in – fisierul de date de intrare
- hash$_i$.out – fisierul de iesire
- Fisierul de intrare va contine fiecare comanda pe o linie noua
- Rezultatele corespunzatoare fiecarei comenzi se vor afisa pe cate o linie
- In cazul aparitiei coliziunii, inserarea perechilor (Key, Value) in bucket-uri se va face ordonat dupa cheie ("Key")
- Tema se va implementa folosind liste dublu inlantuite circulare generice.

3.2. Exemple:

Observatie: in desene nu este pusa in evidenta structura de lista dublu inlantuita circulara.

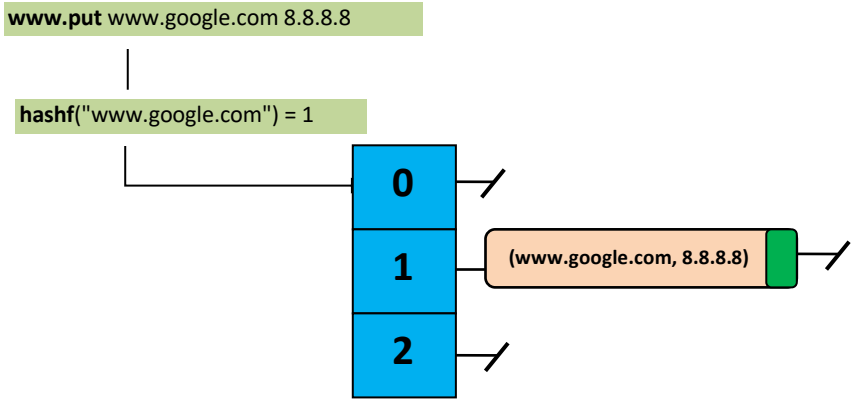
./tema1 3 hash.in hash.out

-construieste o tabela hash cu 3 intrari



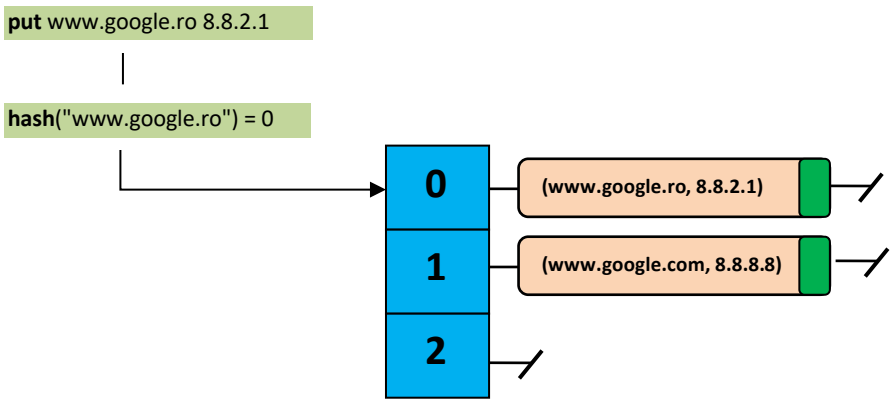
put www.google.com 8.8.8.8

-adauga perechea (www.google.com, 8.8.8.8) in tabela hash, in bucket-ul cu indice 1.



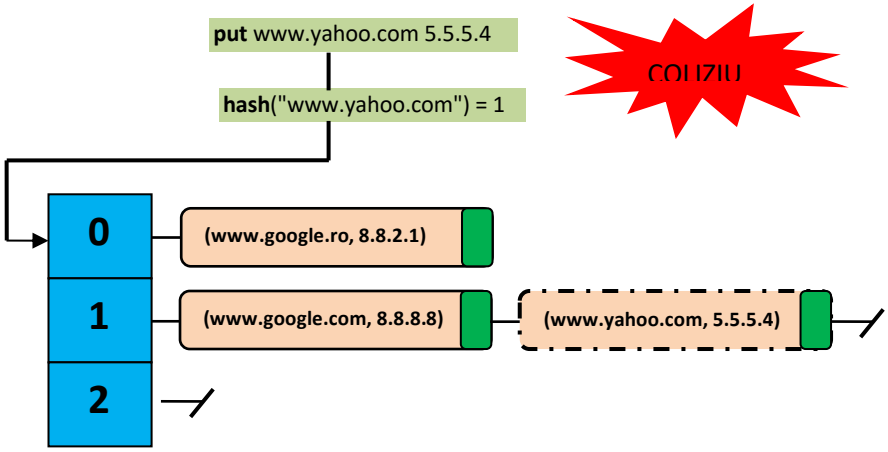
put www.google.ro 8.8.2.1

-adauga perechea (www.google.ro, 8.8.2.1) in tabela hash, in bucket-ul cu indice 0.



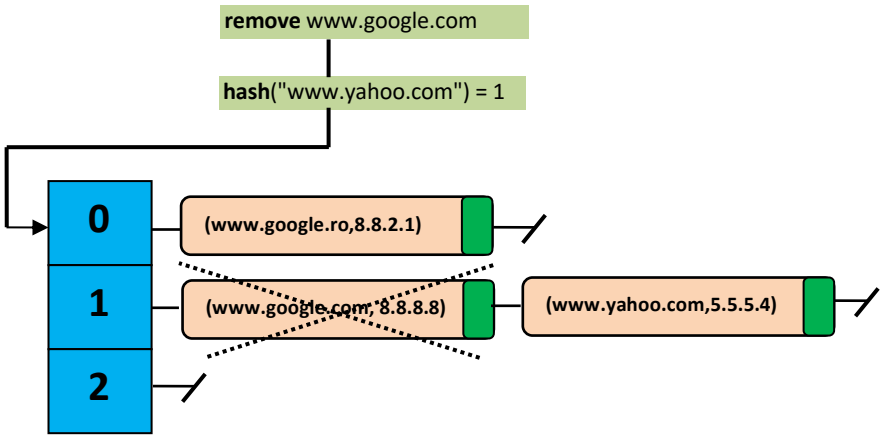
put www.yahoo.com 5.5.5.4

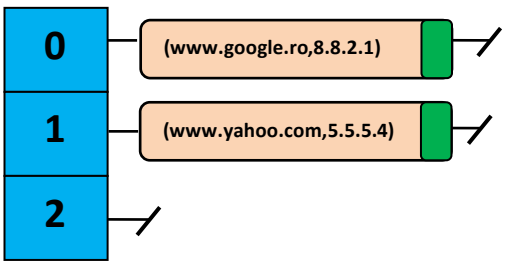
-adauga perechea (www.yahoo.com, 5.5.5.4) in tabela hash, in bucket-ul cu indice 1.



remove www.google.com

-elimina perechea www.google.com, din tabela hash, din bucket-ul cu indice 1.





3.3. Formatul datelor de intrare/iesire:

hash.in

```
put www.google.com 8.8.8.8
find www.yahoo.com
put www.yahoo.com 5.5.5.4
put www.google.ro 8.8.2.1
print
remove www.google.com
print
print_bucket 2
remove www.google.com
print_bucket 2
find www.yahoo.com
get www.yahoo.com
get www.gmail.com
```

hash.out

```
False
0: 8.8.2.1
1: 8.8.8.8 5.5.5.4
0: 8.8.2.1
1: 5.5.5.4
VIDA
VIDA
True
5.5.5.4
NULL
```

Observatie: fiecare linie scrisa in fisierul de iesire se termina cu '\n'.

4. Notare:

- 80 puncte obținute pe testele de pe vmchecker
- 10 puncte: coding style, codul trebuie sa fie comentat, consistent si usor de citit (a se vedea [1]). De exemplu, tema nu trebuie sa contina:
  - warninguri la compilare;
  - linii mai lungi de 80 de caractere
  - tab-uri amestecate cu spatii; denumire neadecvata a functiilor sau a variabilelor
- 10 puncte: README + comentarii + alte eventuale penalizari
- **Temele nu se puncteaza daca nu se foloseste structura de liste dublu inlantuite circulare generice (nu exista santinela).**
- **Bonus 10 puncte** pentru solutiile ce nu au memory leak-uri (bonusul se va considera numai in cazul in care a fost obtinut punctajul aferent testului).
- **O tema care nu va compila se va nota automat cu 0.**

5. Reguli de trimitere a temelor

A se vedea și Regulile generale de trimitere și punctare a temelor [2]

Temele vor trebui incarcate atat pe vmchecker (in secțiunea Structuri de Date (CB): **SD-CB**) cat si pe https://curs.upb.ro/, in sectiunea aferenta Temei 1.

Arhiva cu rezolvarea temei trebuie să fie .zip și să conțină:

- fisiere surse (fiecare fisier sursa creat sau modificat va trebui sa inceapa cu un comentariu de forma:

/* NUME Prenume - grupa */

- fisier README care sa contina detalii despre implementarea temei
- fișier Makefile cu doua reguli: Fișierul pentru make trebuie denumit obligatoriu Makefile și trebuie sa contina urmatoarele reguli:
    - build, care va compila sursele si va obtine executabilul, cu numele tema1.
    - clean, care va sterge executabilele generate.
- arhiva nu trebuie să contina decat fisierele sursa (nu se accepta fișiere executabile sau obiect)
- daca arhiva nu respecta specificațiile de mai sus nu va fi acceptata la upload și tema nu va fi luata in considerare

Referințe:

[1] https://developer.gnome.org/programming-guidelines/stable/c-coding-style.html.en

[2] https://curs.upb.ro/pluginfile.php/536981/mod_resource/content/2/Regulament.pdf