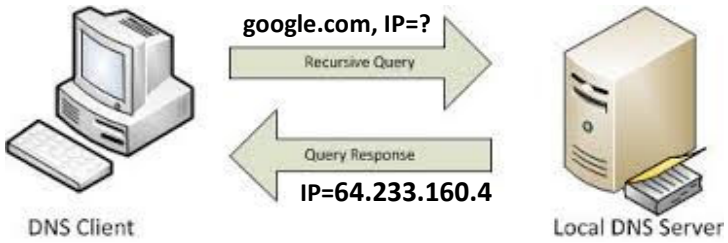
Tema 1 – Structuri de date (seria CB)

Least Recently Used Cache

Responsabili tema:	Alexandru Andrei Rotaru, Rares Mihai Taerel
Data publicarii:	**22.03.2016**
Termenul de predare:	**5.04.2016** ora **23:55** Se accepta teme trimise cu penalizare de 10 puncte / zi (din maxim 100 puncte) pana la data de 8.04.2016 ora 23:55

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.



Din cauza faptului ca aceste servere sunt publice fiind accesibile tuturor clientilor, apare o probabilitate foarte mare ca serverul sa primeasca aceleasi cereri dar de la clienti diferiti. Astfel pentru a usura operatia de cerere pentru serverele DNS, s-a convenit crearea unor servere de cache ce vor retine tupluri de forma (url, ip) pe baza frecventei cererilor deoarece daca un utilizator dintr-un anumit grup a facut o cerere DNS este foarte probabil ca un alt utilizator din acelasi grup sa faca aceeasi cerere.

2. Cerinta

Pentru implementarea backend-ului unui server DNS se poate utiliza o structura de date ce se pliaza foarte bine pe nevoile de mai sus. Aceasta structura porneste de la metoda LRU

Cache (Least Recently Used Cache) care retine in memorie doar cele mai recent folosite inregistrari. Aceasta structura utilizeaza o tabela hash.

Tabela hash este o structura de date optimizata pentru functia de cautare. Acest lucru se realizeaza transformand cheia intr-o intrare in tabela hash (folosind o functie hash). Tabelele 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 intrari in tabela hash). Coliziunile apar in mod inerent, deoarece dimensiunea (numarul de intrari) tabelei hash este fixa, iar obiectele de stocare pot avea lungimi și continut arbitrare. In cazul aparitiei unei coliziuni, valorile se stocheaza pe aceeasi pozitie (in aceeasi lista). In acest caz, cautarea se va reduce la compararea valorilor efective in cadrul listei respective.

3. Implementare

Structura de date LRUCache va fi reprezentata pe baza unei tabele hash. Aceasta este formata dintr-un vector cu M elemente – fiecare element fiind reprezentat printr-o lista dublu inlantuita circulara generica. Elementele din liste vor fi de forma (*Key, Value, Frequency*), unde *Key* si *Value* sunt siruri de caractere alocate dinamic, iar *Frequency* este o valoare intreaga ce reprezinta numarul de accesari ale cheii *Key*. Functia hash va calcula restul impartirii sumei caracterelor ce fac parte din cheia (*Key*) stocata in tabela hash la numarul maxim de liste ce pot fi stocate in tabela hash (M). Fiecare lista va fi sortata descrescator in functie de *Frequency*, in cazul elementelor cu aceeasi valoare pentru Frequency, elementele vor fi sortate alfabetic dupa *Key*.

Operatiile efectuate in tabela hash sunt:

- **set <Key, Value>**
  - daca *Key* nu exista in tabela hash, aceasta va fi adaugata in lista corespunzatoare din tabela hash avand *Frequency = 0*; in cazul in care cheia *Key* exista in tabela hash, aceasta nu va mai fi inserata.
  - in cazul in care *Key* nu exista in tabela hash:
    - daca numarul de elemente din lista corespunzatoare in care trebuie inserat elemental curent **este mai mare sau egal** decat **M**, se va sterge utlimul element din lista si se va realiza inserarea;
    - daca numarul total de elemente din tabela hash **este mai mare** decat **M * 2** se va realoca tabela hash la o noua dimensiune de **M * 2 intrari**. In cazul in care realocarea nu poate fi realizata, tabela hash va ramane nemodificata.

- **get <Key >**
  - daca in tabela hash exista elementul cu cheia *Key*, atunci intoarce valoarea (*Value*) corespunzatoare cheii *Key,* incrementeaza *Frequency* si reordoneaza in mod corespunzator lista din care face parte cheia *Key* cautata; daca cheia *Key* nu exista in tabela hash, se va intoarce NULL.

- **remove <Key>**
  - sterge elementul (*Key, Value, Frequency*) din tabela hash (in cazul in care aceasta exista).

- **print**
  - afiseaza toate valorile (*Value*) din tabela hash;
  - pentru fiecare lista nevida se va afica indicele acesteia si toate elementele acesteia, sub forma:
    i: (Valoare$_1$) (valoare$_2$) … (valoare$_n$)
  - elementele listei sunt afisate pe aceeasi linie, separate de un spatiu; indicii listelor sunt numerotati de la 0.

- **print_list <index >**
  - afiseaza valorile (*Value*) din lista cu indicele *index*, pe o singura linie; indicii listelor sunt numerotati de la 0;
  - separarea elementelor afisate se face numai prin spatii;
  - daca lista asociata intrarii *index* 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 intrari din tabela hash
- hash$_i$.in – fisierul de date de intrare
- hash$_i$.in – fisierul de date de iesire
- Fisierul de intrare va contine fiecare comanda pe o linie noua
- Tema se va implementa folosind liste dublu inlantuite circulare generice.

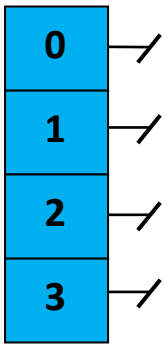
3.2 Exemple

Observatie:

- In exemplele urmatoarele este prezentata succesiunea elementelor ce fac parte din fiecare lista. Nu este pusa in evidenta structura de lista circulara dublu inlantuita generica.
- Functia hash nu este cea din enuntul temei. In cazul fiecarui exemplu este specificata valoarea acesteia pentru fiecare caz in parte

1)

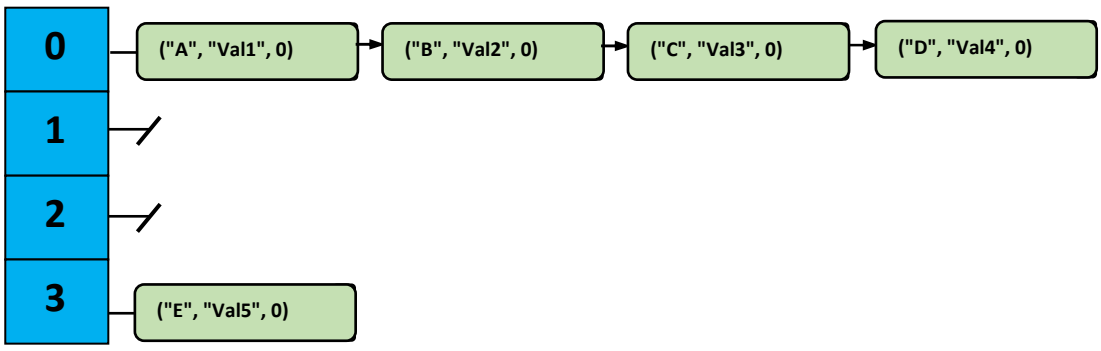
```
./tema1 4 hash.in
```



2)

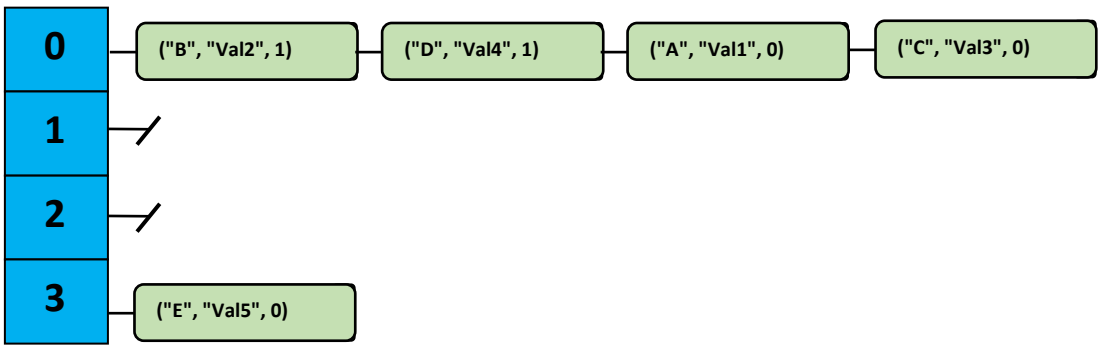
```
set A Val1
set B Val2
set C Val3
set D Val4
set E  Val5
```

```
hash_function("A") = 0
hash_function ("B") = 0
hash_function ("C") = 0
hash_function ("D") = 0
hash_function ("E") = 3
```



3)

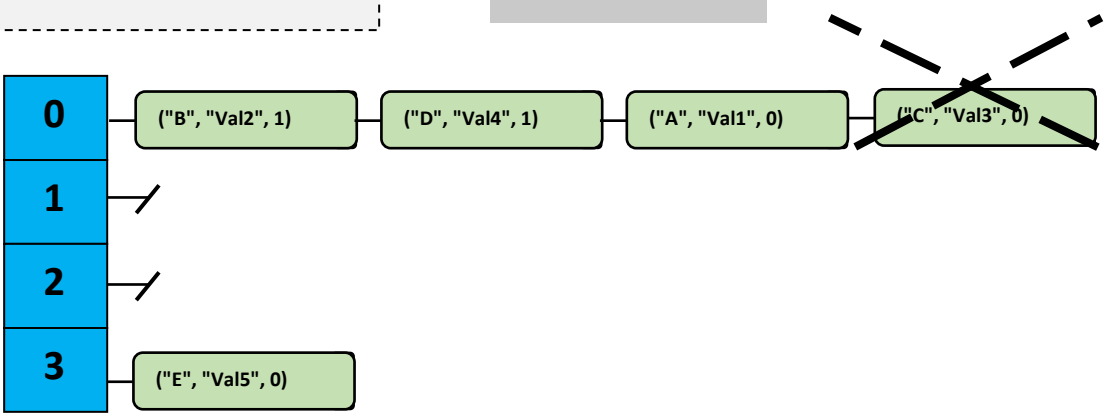
```
get B
get D
```

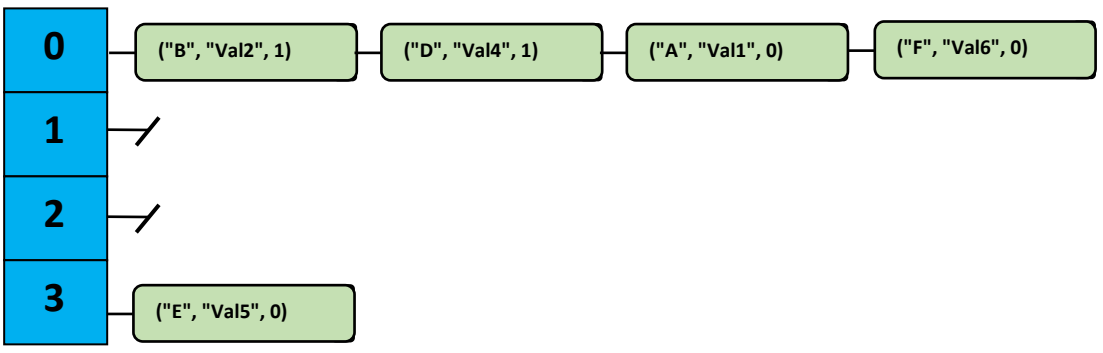


4)

```
set F  Val6
```

hash_function("F") = 0

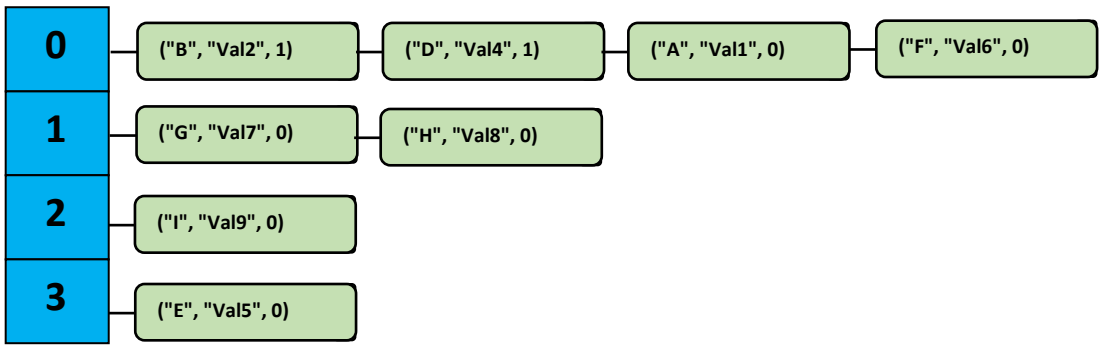




5)

set G Val7
set H Val8
set I Val9

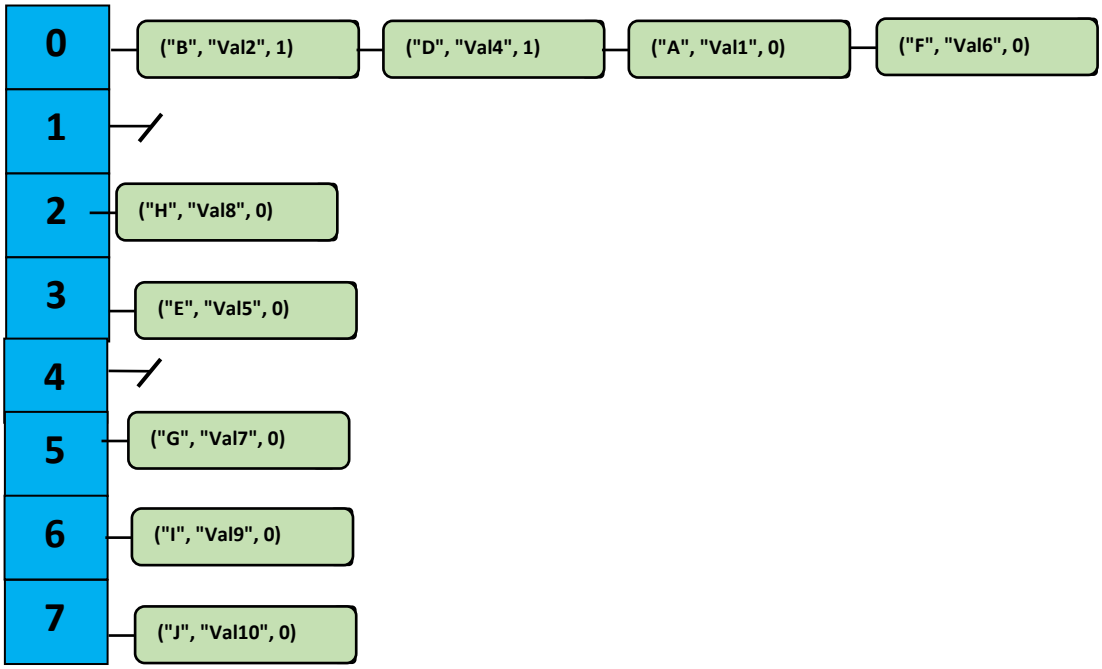
hash_function("G") = 1
hash_function("H") = 1
hash_function("I") = 2



6)

set J Val10

hash_function("J") = 7



Formatul datelor de intrare/iesire:

hash.in

```
./tema1 3 hash.in hash.out
set www.google.com 64.233.160.25
set www.yahoo.com 65.233.120.14
get www.google.com
get www.yahoo.com
print_list 1
```

hash.out

```
64.233.160.25
65.233.120.14
1: (64.233.160.25) (65.233.120.14)
```

4. Notare

- **85 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
  - folosirea incorectă de pointeri, neverificarea codurilor de eroare
  - utilizarea unor metode ce consuma resurse in mod inutil (alocare de memorie)
  - neeliberarea resurselor folosite (eliberare memoriei alocate, stergerea fisierelor temporare, inchiderea fisierelor)
  - alte situatii nespecificate aici, dar considerate inadecvate
- **5 puncte**: README – va contine detalii despre implementarea temei, precum si punctajul obtinut la teste (la rularea pe calculatorul propriu)
- **Bonus: 20 puncte** pentru solutiile ce nu au memory leak-uri (bonusul se va considera numai in cazul in care a fost obtinut punctajul aferent testului)
- **Temele care nu compileaza, nu ruleaza sau obtin punctaj 0 la teste, indiferent de motive, vor primi punctaj 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 seria CB: **SD-CB**) cat si pe cs.curs.pub.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

6. Referinte

[1] http://ocw.cs.pub.ro/courses/programare/coding-style
[2] http://cs.curs.pub.ro/ - curs SD – seria CB – sectiunea Regulament SD -> Reguli de realizare, verificare si trimitere a temelor