-= Tema 1 SD =-
Tabelă de dispersie generică

Scop temă:

- înţelegerea funcţionării unei tabele de dispersie;
- înţelegerea genericităţii şi necesitatea lucrului cu aceasta
- înţelegerea listelor

Informatii trimitere:

- Tema se va încărca pe vmchecker
- **Soft deadline**: 5 aprilie, 2017
- **Hard deadline**: 8 aprilie, 2017
- Pentru fiecare zi de întârziere se va scădea câte 1 punct din 10

Lucreţiu a observat în cadrul laboratorului de SD că în cazul vectorilor, dacă se doreşte găsirea elementelui de pe poziţia ***K***, operaţia nu este costisitoare.

Pentru liste, accesarea unui element poate dura mai mult timp deoarece este necesară parcurgerea tuturor nodurilor până la cel de pe poziţia ***K***, însă inserarea după un element se realizează mult mai rapid.

Având aceste lucruri in minte, Lucreţiu s-a gândit dacă ar putea fi cumva o structură de date care sa îmbine cele două proprietăţi menţionate anterior.

De asemenea, el se gândeste ca dacă accesarea unui element asociat unei poziţii k a unui element din vector, ce ar fi dacă ar putea să găsească elemente asociate cu alte tipuri de date.

Introducere

O tabelă de dispersie (se mai numeşte hash table) este o structură de date folosită în cazul în care se doreşte o introducere si căutare foarte rapidă de elemente. O implementare foarte utilizată in practică este hashmap-ul care retine asocieri între un element (cheie) şi o valoare. Acest lucru este convenabil deoarece introducerea de noi asocieri sau căutarea unei valori asociate unei anumite chei este foarte rapidă.

În temă vi se cere să implementaţi un hashmap în care se stochează asocieri cheie - valoare.

Atât cheia, cât şi valoarea vor fi reprezentate generic (există un *void** care să indice spre informaţia respectivă).

Pentru a testa genericitatea structurii, va trebui să se suporte chei de tip:

- întreg (*int*);
- string (*char **);

Iar pentru genericitatea valorii:

- structura TStudent;
- structura TMaterie;

Pentru a determina bucket-ul în care trebuie introdusă o nouă pereche (cheie, valoare) se va aplica funcţia de hash asupra cheii (se va obţine un număr), după care trebuie normalizat rezultatul pentru a-l aduce în intervalul [0, număr_bucket-uri) - această operaţie se realizează cu operatorul modulo (%) - operaţia aceasta se realizează în funcţia hash din arhiva de start.

Deoarece funcţia de hash poate să dea aceeaşi valoare pentru chei diferite (va exista o coliziune între perechi) se va utiliza un mecanism de tratare al coliziunilor numit **separate chaining**, utilizând liste generice simplu înlănţuite pentru a stoca informaţia perechilor (cheie-valoare) care au căzut în acelaşi bucket.

De asemenea, cu cât se introduc mai multe perechi în hashtable există o posibilitate din ce în ce mai mare de coliziune. De aceea, se defineşte un **load factor** *k / c*, unde:

- k reprezintă numărul de perechi din hashtable;
- c reprezintă capacitatea - numărul total de bucket-uri

La atingerea/depăşirea acestui **load factor**, hashtabel-ul va trebui să îşi dubleze numărul de bucket-uri, iar perechile deja existente să fie reintroduse. Modificându-se numărul de bucket-uri, se va modifica şi poziţia unde o anumită pereche va fi introdusă în noua tabelă.

Pentru aceasta sunt definite în prealabil (se găsesc în arhiva de start):

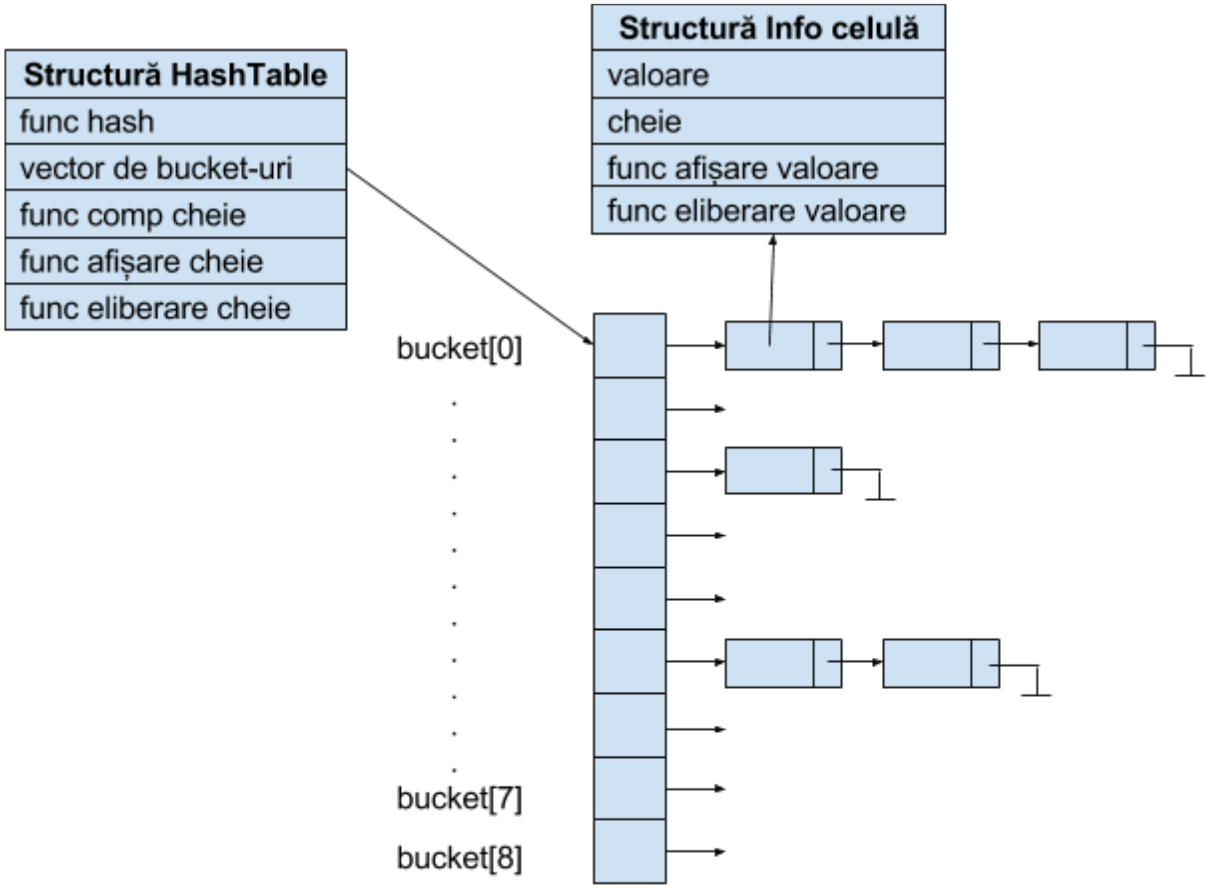
- o funcţie de hash - va fi folosită pentru determinarea bucket-ului unde se va introduce cheia cu valoarea asociată;
- structurile cu care se va lucra - pentru TStudent şi TMaterie;
- structura de listă generică (ca în laborator);

Funcţia de hash şi structurile nu trebuie modificate.

Structură internă hashmap:

- bucket-urile vor fi implementate ca **liste generice**
- camp-ul destinat informaţiei din lista generică trebuie să fie *void **, iar acesta va puncta spre o structura sau structuri de structuri unde trebuie să existe următoarele informaţii:
  - **funcţie de afişare pentru valoare;**
  - **funcţie de eliberare pentru valoare (pentru bonus);**
  - **informaţia pentru valoare** (trebuie sa fie *void **);

    - **informaţia pentru cheie** *(trebuie sa fie void *);*
- Pointerii la funcţiile specifice cheilor se vor ţine în structura hashtabel-ului:
    - **funcţie de afişare pentru cheie;**
    - **funcţie de eliberare pentru cheie (pentru bonus);**
    - **funcţie pentru compararea a doua chei;**
- Pointerii la funcţiile specifice valorile trebuie să fie iniţializaţi în momentul în care se introduce un nou element.



Load factor = k / c = 6 / 9 = 0.66

k - numărul de intrări din hashtable
c - numărul de bucket-uri

Cerinţe

Se doreşte implementarea unei tabele hash generice - va putea ţine elemente de diferite tipuri.

Datele se vor citi din fişierul **input.in**, iar afişarea se va realiza în fişierul **output.out** (ambele se află în folder-ul unde este creat executabilul).

Pe prima linie din fişier se va afla o linie de forma:
inithash [tip_cheie] [număr_iniţial_bucket-uri]

tip_cheie specifică cum trebuie interpretată *cheia*
Pentru *tip_cheie* vor exista 2 posibilităţi:

- **d** - pentru număr întreg fara semn;
- ***s*** - pentru string (**lungimea maximă va fi de 50 de caractere** - inclus şi NULL terminatorul);

O dată ce hashtabel-ul a fost iniţializat cu un tip de cheie, în acesta se vor insera doar perechi cheie-valoare, unde cheia este de acel tip.

a. Implementarea operaţiilor de inserare şi printare a hashmap-ului

ATENŢIE: Nu trebuie realizata redimensionarea hashmap-ului pentru acest task.

Inserarea unui element in hashmap se va realiza după următorul pattern:

insert cheie [tip_valoare] valoare

tip_valoare specifică cum trebuie interpretată *valoarea*
Pentru *tip_valoare* vor exista 2 valori:

- **stud** - pentru un element de tip student;
- **mat** - pentru un element de tip materie;

Datele vor fi citite în ordinea în care apar câmpurile în structură.

Dacă cheia **se găseşte deja în bucket**, intrarea veche va trebui **înlocuită** cu noua valoare.
Dacă cheia nu se găseşte în bucket, perechea se va introduce la **finalul listei.**

Exemplu de inserare (trebuie să urmăriţi câmpurile structurilor):

insert 32 stud TonyStark 10 333CQ 30
insert qwerty mat Structuri_de_Date 3 2 5 6

Prima comandă va insera în hashmap perechea 32 - (TonyStark, 10, 333CQ, 30).
A doua comandă va insera în hashmap perechea qwerty - (Structuri_de_Date, 3, 2, 5 10).

Printarea va realiza scrierea hashmap-ului în fişierul de ieşire.

Exemplu de printare:

print

Va scrie hashmap-ul în fişier sub următoarea formă:
[0] : (cheie_00 -> valoare_00) (cheie_01 -> valoare_01)…
[1] : (cheie_10 -> valoare_10) (cheie_11 -> valoare_11) …
[2] :

...
...
[k_bucket] : (cheie_k0 -> valoare_k0) (cheie_k1 -> valoare_k1)...
…
…

Afişarea pentru cheie şi valoare se va realiza apelând o funcţie de afişare specifică cheii şi valorii prin intermediul unui pointer la funcţia respectivă.

În cazul în care bucket-ul nu are intrări, se va afişa index-ul bucketului fără nici un element (vezi exemplul pentru indexul 2).

Elementele de tip float vor fi afişate cu o precizie de 2.

b. Redimensionare şi căutare dupa cheie - 30p

Se va folosi un **load factor** de 0.75 drept threshold pentru **redimensionarea hashmap-ului**.

Pentru **căutarea** după o anumită cheie se va folosi următorul pattern:
find cheie

În cazul în care valoarea nu este găsită se va afişa mesajul “Nu exista”, iar în cazul în care este gasită se va afişa valorea (se va utiliza pointerul la functia de afişare a valorii).

c. Stergere dupa cheie - 30p

Stergerea unui element in hashmap se va realiza după următorul pattern:
delete cheie

Pentru a realiza acest lucru se parcurg următorii paşi:

- se identifică bucket-ul în care se afla cheia;
- se parcurg elementele lista asociată acelui bucket;
- se verifică folosind funcţia de comparaţie dacă cheile sunt egale

Dacă **se găseşte cheia**, nodul respectiv va fi **eliminat din hashmap**.
Dacă **nu se găseşte cheia**, hashmap-ul rămâne nemodificat.

d. BONUS - 20p

Dacă nu exista erori/warning-uri/leak-uri la rularea cu toolsuite-ul valgrind.

Formatul datelor de intrare şi ieşire:

Datele de intrare se vor citi dintr-un fişier **input.in**, iar acestea for fi reprezentate în fişier în felul următor:

inithash [tip_cheie] [număr_iniţial_bucket-uri]
n // reprezintă numărul de operaţii
op_0
...
…
op_n-1

Conţinut arhivă:

- **Makefile** care să conţină regulile: build şi clean
  - *build* - crează executabilul denumit **./hash_table**
  - *clean* - şterge fişierele rezultate în urma compilării
- **Readme** în care să descrieţi modul de implementare ales
- **Fişierele** utilizate pentru compilare

Precizări

- Funcţia de afişare pentru cheie ar trebui să printeze simplu valoarea cheii;
- Funcţia de afişare pentru valoare ar trebui să respecte următorul format:
- **[Denumire câmp: valoare câmp]**;
  - Dacă existe mai multe câmpuri, ele vor fi separate prin **“,”;**
  - Exemplu1: Pentru o materie cu valorile: “SD”, 3, 2, 5, 3 se va afişa:
    - [Materie: SD, Ore_curs: 3, Ore_lab: 2, Credit: 5, Teme: 3]
  - Exemplu2: Pentru un student cu valorile: “Sherlock”, 9.99, “312CW”, 22 se va afişa:
    - [Nume: Sherlock, Grupa: 312CW, Medie: 9.99, Varsta: 22]
    - Trebuie să afişaţi prima dată grupa, iar după media.

- Apelarea funcţiilor de comparaţie, de afişare, de eliberare trebuie să se realizeze prin cadrul structurii/structurilor auxiliare definite de voi.
- Aveţi la dispoziţie in cadrul ahivei de pornire 4 fisiere: definiţiile structurilor TStudent, TMaterie, Lista si functia de hash pe care o veţi folosi.
- 10 puncte se oferă pentru coding-style si Readme corespunzător
- 100 puncte pentru teste + Readme + coding-style, 20 puncte bonus
- Temele care nu compileaza, nu ruleaza sau obtin punctaj 0 pe teste, indiferent de motive, vor primi punctaj 0
- **Nu trebuie să ţineţi un câmp special** pentru a vă da seama dacă ce trebuie să afişati/eliberaţi este un student sau o materie (pentru valoare), string sau int (pentru cheie) - acest lucru se va face la alocarea unui element prin atribuirea pointerilor corespunzători funcţiilor mai sus menţionate.
  - **ATENŢIE! Nerespectarea acestui lucru va duce la scadea punctajului pe task-ul respectiv.**