HESH JADVALLAR, ULARNING ISHLAB CHIQILISHI VA QO‘LLANISHI

Abduaziz Ziyodov
Toshkent shahridagi Inha Universiteti talabasi
abduaziz.ziyodov@mail.ru

Abstrakt: Hesh jadvallar massivlar va hesh funksiyalar orqali ishlab chiqilgan ma’lumot turi hisoblanadi. Ma’lumotlarni izlashda, olishda juda effektiv hisoblanadilar. Ushbu maqola hesh jadvallar haqida umumiy tushunchani yoritadi, qo‘shimcha sifatida ularning yaratilishi uslublari, ustunlik va kamchilik tomonlarini ham. Maqolada hesh jadvallarda ko’p marotaba takrorlanadigan muammolardan biri “collisions”(kolliziya) va samarador hesh funksiyalar ham muhokama qilinadi.

Kalit so‘zlar: Ma’lumot turlari, hesh, hesh jadvallar, hesh funksiyalar, Big-O, algoritm, xotira, qidiruv, collision, “kolliziya”

Kirish

Hesh jadvallar kalitlarni qiymatlarga bog’laydigan abstrakt ma’lumot turidir, va ular ba’zan “hash map” sifatida ham mashxur. Kompyuter ta’limotida ko’plab ma’lumot turlari mavjud va ularning har biri o’ziga yarasha ustunlik tomoniga ega. Misol uchun ketma-ket uzluksiz kelgan ma’lumotlarni saqlashda biz massivdan foydalanishimiz mumkin. Lekin undagi muammolar bu qidiruv, qo’shish va o’chirish amallari. Massivdan elementni qidirishni samaradorligini oshirish uchun biz “binary” qidiruv algoritmidan foydalanib, samaradorlikni O(N)dan O(logN)ga cha oshirishimiz mumkin. Ammo, ma’lumot qo’shish va o’chirish amallari o’rtacha holatda O(N) bo’lib qolaveradi.

Muammoni hal etish uchun “linked-list”(bog’lama massivlar) ishlab chiqilib, ushbu amallar samaradorligini O(1)gacha olib chiqdi. Qidiruv amalining vaqti esa O(N) bo’lib qoldi. Lekin massivlar va “linked-listlar”ga qaraganda murakkab mantiq asosida qurilgan ya’ngi abstrakt ma’lumot turi hesh jadvallar ushbu amallarning

barchasini O(1) vaqtda bajarib berish qobiliyatiga ega. Chuqur muhokamani boshlashdan avval Big-O yoki Katta O funksiyasi haqida qisqa ma'lumot bilan tanishsak.

Big-O

O'rta maktab algebrasidan bizga matematik funksiya atamasi tanish. Funksiya argumentlar oladi, va ma'lum bir hisob-kitoblar hamda qonuniyat asosida natija qaytaradi.

$$funksiya(argument) = argument^2$$

Sodda tilda argument kalit so'zini sonlar bilan o'zgartirsak, sonli ifodaga ega bo'lamiz.

$$funksiya(argument = 5) = 5^2 = 25$$

Oson to'grimi? Buning Big-O funksiyasiga aloqador tomoni shundaki, Big-O yoki O(x) ham funksiya, u faqat funksiyaga berilgan argumentning qiymati oshgani sari, uning bajarilish vaqti qanday o'zgarishini xarakterlaydi. Yuqorida berilgan

Argument	0	1	2	3	4	5
Natija	0	1	4	9	16	25

funksiyaga argument sifatida 0 yuborilsa natija ham 0, 1 bo'lsa 1 va hokazo.

6ta qadamda funksiya qiymati 25ga yetdi. Ammo oddiy chiziqli funksiya ya'ni f(x) = x da bu ko'rsatkich 6ni ko'rsatgan bo'lar edi ya'ni unda argument kattalashgani sa'ri kvadrat funksiyada natijaning o'sishi, oddiy chiziqli funksiyaga nisbatan juda jadal ekanligini ko'ramiz.

Big-Oda argument sifatida ham son qabul qilinadi ammo ushbu sonnig ma'nosi kiritilayotgan ma'lumotning hajmi hisoblanadi. U bayt, kilobaytlarda emas oddiy

sonlarda o'lchanadi. Funksiyaning asosiy maqsadi esa ushbu kirilayotgan qiymatning hajmi kattalashgani sari, siz ishlab chiqgan algoritm/funksiyaning bajarilish vaqti qancha kattalashishini hisoblashdir. Funksiya natijani sekundlarda qaytarmaydi, shunchaki uni xarakterlab qanchalik tez yoki sekin degan savolga javob beradi.

Jurnaldan 100ta odam ma'lumotlari ichidan Ziyodov nomli familiya katta ehtimol bilan oxiridan topiladi (saralangan bo'lsa) agar ketma-ketlikda izlansa, ya'ni 100ta element bo'lsa biz deyarli 100ta elementni ko'zdan kechirib chiqishimiz lozim. Bu esa bizga O(N) ya'ni N ta element, N ta amal tushunchasini beradi. Ammo, "binary" qidiruv algoritmi esa buni O(logN) gacha tushurib beradi ya'ni O(logN) funksiyani O(N) ga nisbatan sekin o'sgani uchun bu juda tez degani. Hesh jadvallarda ushbu ko'rsatgich O(1) ni tashkil etadi.

Hesh jadvallar

Massivdan elementni qidirish vaqti eng yaxshi holatda ya'ni "binary" qidiruv algoritmi yordamida O(logN)ni tashkil etishini bilamiz. Ammo, xotiraning ma'lum qismidan ma'lumotni yuklash vaqti O(1) ekanligini unutmaslik lozim. Ya'ni maxsus indeks bo'yicha massivdan element olinsa, uning vaqti O(1) ya'ni o'zgarmas("constant")ni tashkil etadi. Hesh jadvallar ushbu prinspdan foydalanadi.

```
>>> massiv = [1,2,3,4,5,6]
>>> massiv[0] # too fast
1
```

Sababi, siz foydalanayotgan dasturlash tili massiv elementlari xotiraga ketma-ket qo'yilishi va ularning indeks bo'yicha qayerda joylashganini aniq biladi. Bitta son

$$K - element = 1 - element\ adresi + K \times (1\ element\ hajmi)$$

("integer") N bayt egallaganda massivning to'liq hajmi (baytlarda) uzunlik*N baytga teng bo'lar edi. K-inchi indeksda turgan element xotiraning qaysi blokida turganligini("address") aniqlash uchun ushu mantiqdan foydalaniladi:

Ushbu funksiya aniq joylashuv manzilini tezda yetkazib bergani uchun millioninchi indeksdagi va birinchi indeksdagi elementni olish (“access”) o’zgarmas

$$K - element = 1 - element\ adresi + K \times (1\ element\ hajmi)$$

vaqtni tashkil etadi O(1). Demak bizda Kni argument sifatida olib, joylashuvini qaytaradigan qandaydur funksiya mavjud va shuning uchun biz O(1) ga egamiz.

Bundan kelib chiqadiki, biz ma’lum bir abstrakt ma’lumot turini shu turdagi funksiya bilan ishlab chiqa olsak, biz yana O(1) ko’rsatkichiga erisha olamiz. Ha, to’gri topdingiz - hesh jadvallar.

Hesh jadvallarda esa ushbu funksiyalar hesh funksiyalar deb ataladi. Ularning yagona farqi esa ular xotira joylashuvi/blokini natija sifatida qaytarmaydi. Boshqa ma’noda aytganda ular boshqa maqsadda ham ishlatilishi mumkin bo’lgan universal funksiya sanaladi. Ular satr (“string”) ni argument sifatida olib, har safar, istalgan vaziyatda yagona sonni natija sifatida qaytaradi. Uning ushbu xususiyatini qidirilayotgan elementning massivdagi o’rnini aniqlashda foydalanishimiz mumkin. Quyida esa “pseudo” kod parchasi:

```
>>> funksiya hesh(satr) -> son:
            ...
>>> ism = "abduaziz"
>>> son_qiymat = hesh(ism)
>>> ismlar = yangi Massiv()
>>> ismlar[son_qiymat] = ism
```

uni indeks sifatida foydalanib “gigant” massiv ichidan ham elementni O(1) vaqtda topish mumkin.

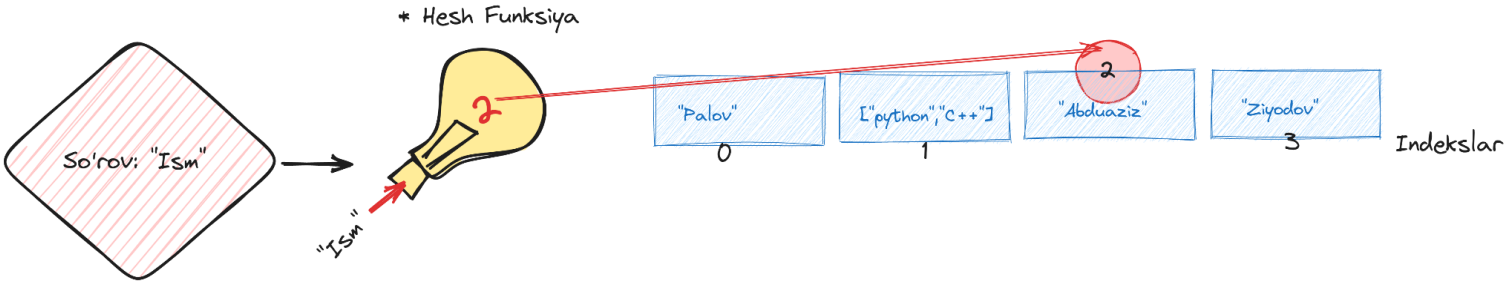
Yaratish

Tushunganingizdek, hesh jadvallar bu hesh funksiya va massivdan tashkil topgan ma’lumot turidir. Quyida esa “Jenkins One-at-a-time hash" funksiyasini misol

tariqasida ko’rishingiz mumkin(Java misolida albatta):

```
int joaat_hash(byte[] key) {
    int hash = 0;
    for (byte b : key) {
        hash += (b & 0xFF);
        hash += (hash << 10);
        hash ^= (hash >>> 6);
    }
    hash += (hash << 3);
    hash ^= (hash >>> 11);
    hash += (hash << 15);
    return hash;
}
```

Soddalashtirilgan Ishlash Prinspi



Python dasturlash tilining “hash” funksiyasidan foydalanib ham yaxshi hesh

jadval qurish mumkin:

Obyekt yaratilish vaqtida jadval hajmi belgilanadi, va shu qiymatga teng bo'lgan

```
BLANK = object()

class HashTable:
    def __init__(self, capacity):
        self.values = capacity * [BLANK]

    def __len__(self):
        return len(self.values)

    def __setitem__(self, key, value):
        index = hash(key) % len(self)
        self.values[index] = value

    def __getitem__(self, key):
        index = hash(key) % len(self)
        return self.values[index]

# real python sahifasidan olindi
```

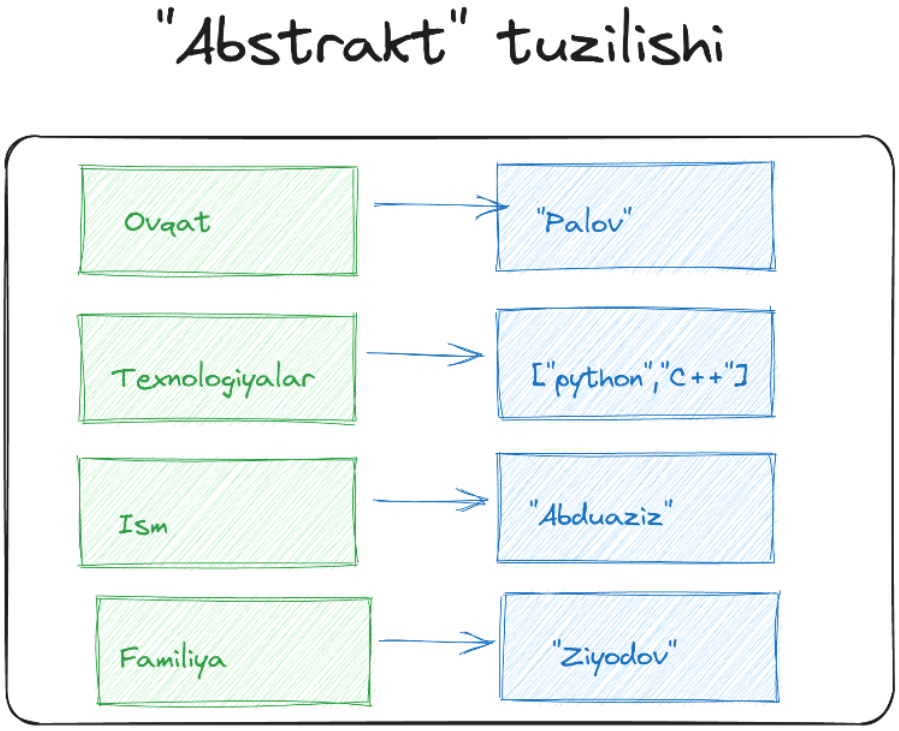
miqdorda elementlar yaratiladi. Hesh jadval uzunligi uning ichidagi massiv uzunligiga teng etib belgilanadi (__len__). Element qo'shishda __setitem__ funksiyasi ishlaydi va u kalit hamda qiymat argumentlarini qabul qiladi. So'ng, hesh funksiya orqali kalitning sonlik qiymatini oladi. Ko'p holatlarda ushbu qiymat, hesh jadval uzunligidan katta bo'ladi agarda ular tog'ridan-to'g'ri ishlatilsa IndexError xatoligi yuzaga kelishi mumkin (element indeksi, massiv hajmidan katta bo'lsaa). Oldini olish uchun mod amalidan foydalanamiz, va hajmga mos indeksga ega bo'lamiz. Elementni olishda ham

shu amal, __getitem__ ishlaydi va massiv ichidagi maxsus indeksga ega element qaytarib qo'yiladi.

Endi, maqola hesh jadvallarda uchraydigan muammolarni muhokama qiladi.

Kolliziyalar("Collisions")

Ideal sharoitda hesh jadvallarda bir kalit so'zga faqat birgina qiymat to'g'ri keladi.



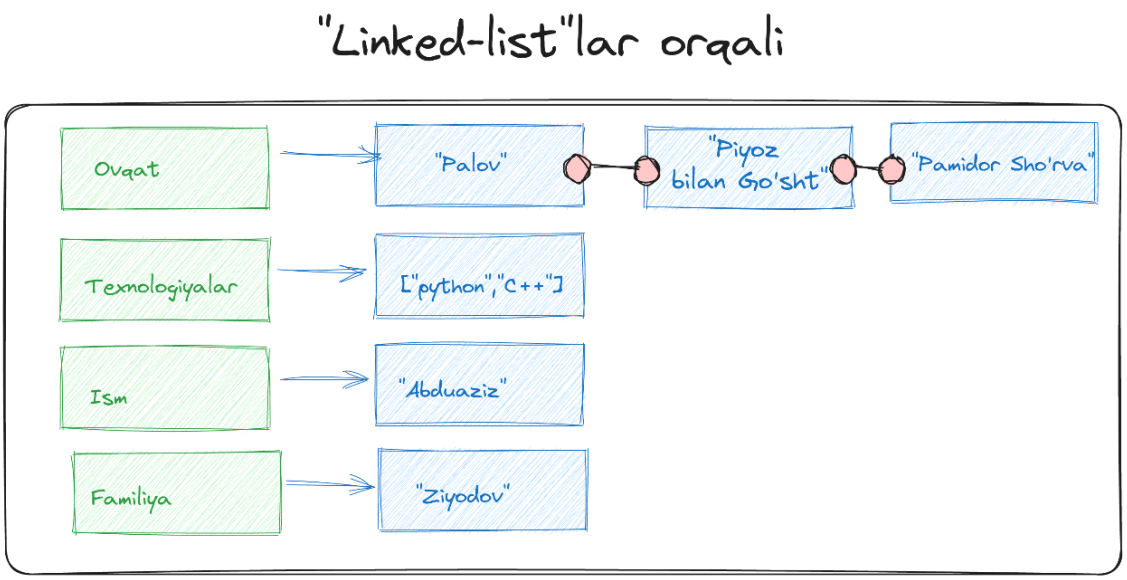
Lekin ba'zi holatlarda esa vaziyat jiddiy tus oladi. Ya'ni hesh funksiya 2 xil turdagi kalit uchun ham bir xil qiymat qaytaradi. Bu esa juda yomon, chunki buning oqibatida avvalgi qiymat ustiga yangi qiymat yozilib ketilishi va bu ma'lumotning yo'qotilishiga sabab bo'lishi mumkin. Muammoning ildizi asosan 2ta:

- Kuchsiz, sifatsiz hesh funksiya.
- Hesh jadvalning hajmi.

Hesh funksiya tasodifiy qiymat qaytarmasligi lozim, f(x) != random(). Tasodifiy qiymat qaytarilganda takrorlanishlar soni ham kolliziyaga sabab bo'lishi mumkin. Hesh funksiyalar ASC II yoki alifbo tartibida ham ishlamasligi mumkin aks holda "Abduaziz" va "Abror" kalit so'zlari sababli kolliziya yuzaga kelishi mumkin (bosh harflari, ASC II qiymatlari bir xil).

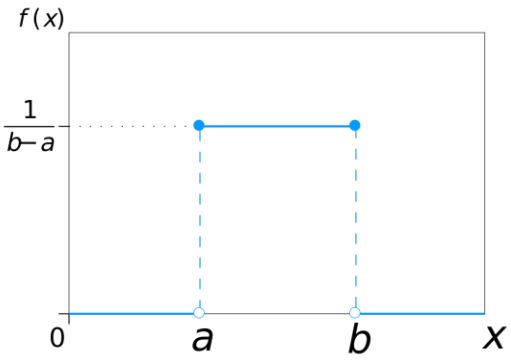
Ingliz tilida ushbu xususiyat “consistency”(“consistent”) deb ataladi(doimiylik, ketma-ketlik). Ma’nosi esa funksiya har doim bir xil argument uchun bir qiymat qaytarishi, va boshqa argumentlarda takrorlanmasligidadir. $f(5)=5^2$ funksiyada siz 5da olgan qiymatingizni boshqa argument bilan olishingizni imkoni mavjud emas.

Kolliziyalarning oldini olishning yana bir uslubi bu “linked-list”lardan foydalanishdir. Kolliziyaga uchrash ehtimoli bo’lgan elementlar ketma-ketlikda qo’yiladi ya’ni endi bizda bir kalitga bir qiymat emas qiymatlar ketma-ketligi mavjud:



Bu usulning samaradorligining pastligi shundaki, jadvalda faqat “P” bilan boshlanadigan elementlar bo’lsa (1000ta deylik), bizda bitta kaliti mavjud bo’lgan ulkan “linked-list” paydo bo’ladi. Bu orqali hesh jadval shunchaki keraksiz matoh kabi qoladi, samaradorlik “linked-list”niki bilan bir bo’lib qoladi.

Yaxshi hesh funksiyaning asosiy xususiyatlari: u hisoblashga oson va tez hamda qiymatlar “uniform” tarzda taqsimlangan:

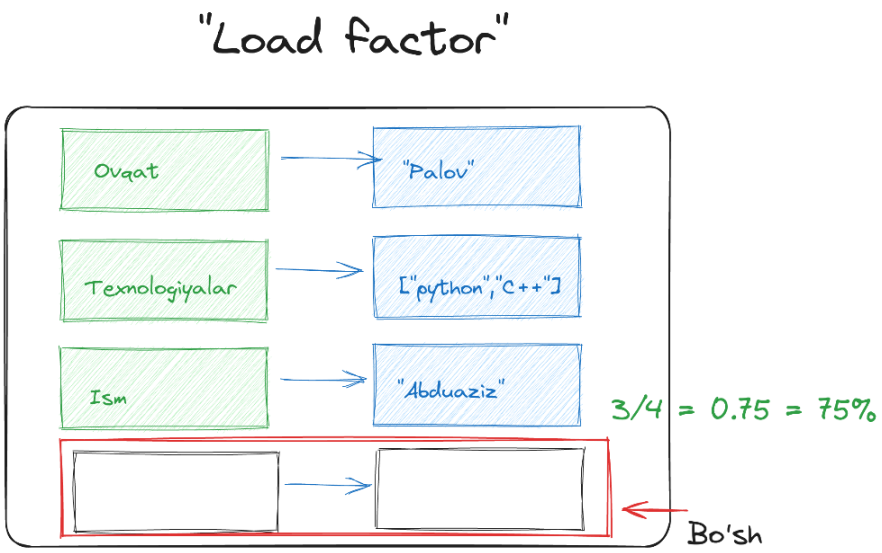


Load Faktor

Hesh jadvalning hajmining yetmasligi oqibatida yuzaga keladigan kolliziya esa hajm to'lgani sababli elementlar ustma-ust yozilib ketishidan bo'ladi. Buni kuzatib borishning usuli esa maxsus load-faktorning qiymati bilan bog'liq. Ushbu ko'rsatkich, hesh jadval uchun kritik hisoblanib quyidagicha hisoblanadi:

$$loadFactor(\alpha) = N/M$$

Bunda N mavjud, egallangan joylar soni va M esa to'liq "savat"lar soni deb olingan. Ushbu faktor 100%ga yoki 1ga teng bo'lganda jadval to'lgan hisoblanadi.



Manbalarda ushbu ko'rsatgich 0.7dan katta bo'lganida jadval hajmini kattalashtirish maqsadda muvofiqligi takidlangan. Load faktorning kichikligi, kollizya holatlarini ehtimolini ham sezilarli darajada kamaytiradi.

Agarda kolliziyalar va load-faktorlar hisobga olinmaganida, hesh jadvallar qidiruv, qo'shish va o'chirish amallarini O(N) vaqtda bajargan bo'lar edilar.

	Hesh jadval(o'rta)	Hesh jadval(eng yomon)	Massiv	Linked-list
Qidiruv	O(1)	O(N)	O(1)	O(N)
Qo'shish	O(1)	O(N)	O(N)	O(1)
O'chirish	O(1)	O(N)	O(N)	O(1)

Qo‘ llanishi

Hesh jadvallar dastur ishlab chiqish vaqtida juda ko'p marotaba foydalaniladi. Telefonda kontaktlarni izlashda ham foydalanish mumkin. Bu yerda telefon raqam bilan shaxsning ismni o'rtasida aloqa mavjud, ulardan birini kalit, ikkinchisini qiymat sifatida berilsa qidiruv jarayoni tezlashadi.

Veb ilovalar qurish vaqtida, katta hajmdagi ma'lumotlarni o'qish vaqtida keshlash uchun foydalanish mumkin. Feysbukning sayt haqidagi sahifasi deyarli o'zgarmaydi (statik), dinamik kontent mavjud emas va uni har safar HTMLga render qilish shart bo'lmagani uchun sahifa kontentini xotiraga saqlab qo'yiladi va maxsus kalit so'z bilan eslab qolingan kontentni foydalanuvchiga ko'rsatib beriladi. "Cookie"lar ham ushbu prinspda ishlaydi. Kalit va qiymat asosida ishlaydigan eng mashxur Redis NoSQL ma'lumotlar ombori ham bundan juda keng tarzda foydalanib keladi. Dastur komponentlarida takrorlanishlar("duplikatlar") oldini olishda ham foydalanishi mumkin (M: saylovda qayta ovoz berishni oldini olish uchun).

Xulosa

Ushbu abstrakt ma'lumot turi o'z o'rnida qo'llanilsa yuqori samaradorlikka erishish mumkin. Dasturchiga ko'p holatlarda hesh jadvallarni qo'lda yozishga to'g'ri kelmaydi, ammo uning sahna ortidan qanday ishlashini tushunish faqat foyda olib keladi. Ular bir element va boshqa element o'rtasidagi munosabatlarni modellashtirishda juda foydali. Dasturchi kolliziya holatlaridan qochishi, jadvalni

nazorat qilib borishi lozim, ishonchli hesh funksiyadan foydalanish maqsadga muvofiq. Hesh funksiyalar kriptografiya sohasida ham keng qo'llaniladi. Ushbu ma'lumot turining qo'llanish hajmi ancha keng, keshlashda, qidiruvda, takrorlanishlarni oldini olishda va umuman har qanday ma'lumotni eslab qolish va tezda yetkazish bilan bog'liq vaziyatlarda sizni "sharmanda" qilib qo'ymaydi.

Manbalar va adabiyotlar

- Grokking Algorithms, Aditya Y. Bhargava.
- https://en.wikipedia.org/wiki/Hash_table
- https://courses.csail.mit.edu/6.006/spring11/rec/rec07.pdf
- http://www.burtleburtle.net/bob/hash/doobs.html
- https://www.hackerearth.com/practice/data-structures/hash-tables/basics-of-hash-tables/tutorial/
- https://khalilstemmler.com/blogs/data-structures-algorithms/hash-tables