Fahad sayyed - qayd qiladi DSA 6 @
fahad sayyed bilan bog'langan

Vaqt murakkabligi tahlili

Kirish

Dasturlashda muhim savol: Algoritm qanchalik samarali yoki a
kod qismi?

Samaradorlikjuda ko'p resurslarni qamrab oladi, jumladan:

- CPU (vaqt) dan foydalanish
- Xotiradan foydalanish
- Diskdan foydalanish
- Tarmoqdan foydalanish

Hammasi muhim, lekin bizni ko'proq tashvishlantiramiz**CPU vaqti**. Ehtiyot bo'ling farqlash:

1. Ishlash: Qancha vaqt/xotira/disk/va hokazo. dastur ishga tushganda ishlatiladi.

Bu mashinaga, kompilyatorga va hokazolarga, shuningdek, biz yozadigan kodga bog'liq.

2. Murakkablik: Dastur yoki algoritmning resurslarga bo'lgan talablari qanday o'lchanadi?

ya'ni kod tomonidan hal qilinayotgan muammoning o'lchami kattalashganda nima sodir bo'ladi.

Eslatma:*Murakkablik ishlashga ta'sir qiladi, lekin aksincha emas.*

Algoritm tahlili

Algoritm tahlili hisoblash murakkabligi nazariyasining muhim qismidir
hal qilish uchun algoritmning zarur resurslari uchun nazariy baho beradi
muayyan hisoblash muammosi. Algoritmlarni tahlil qilish - ni aniqlash

uni amalga oshirish uchun zarur bo'lgan vaqt va makon resurslari miqdori.

Nima uchun algoritmlarni tahlil qilish kerak?

- Algoritmni ma'lum bir narsada qo'llamasdan uning harakatini bashorat qilish kompyuter.
- An samaradorligi uchun oddiy chora-tadbirlarga ega bo'lish ancha qulayroq algoritmni amalga oshirish va har safar samaradorlikni sinab ko'rishdan ko'ra algoritm asosiy kompyuter tizimidagi ma'lum bir parametr. o'zgarishlar.
- Algoritmning aniq harakatini oldindan aytib bo'lmaydi. ham bor ko'plab ta'sir qiluvchi omillar.
- Shunday qilib, tahlil faqat taxminiy hisoblanadi; u mukammal emas.
- Eng muhimi, turli xil algoritmlarni tahlil qilish orqali biz ularni solishtirishimiz mumkin maqsadimiz uchun eng yaxshisini aniqlash.

Tahlil turlari

Berilgan algoritmni tahlil qilish uchun biz algoritm qanday kiritilishini bilishimiz kerak kamroq vaqt oladi (ya'ni algoritm yaxshi ishlaydi) va qaysi bilan kiritiladi algoritm uzoq vaqt talab etadi.

Odatda uch turdagi tahlillar amalga oshiriladi:

• **Eng yomon holatlar tahlili:**Eng yomoni, buning uchun kiritishdan iborat algoritm uning bajarilishini yakunlash uchun eng uzoq vaqtni oladi.

• **Eng yaxshi vaziyat tahlili:**Eng yaxshi holat algoritm bo'lgan kirishdan iborat uning bajarilishini yakunlash uchun eng kam vaqt talab etiladi.

• **O'rtacha holat:**O'rtacha ish vaqti o'rtacha ishlash vaqti haqida fikr beradi berilgan algoritm.

Algoritm samaradorligining ikkita asosiy murakkablik ko'rsatkichi mavjud:

- **Vaqtning murakkabligi**algoritm vaqt miqdorini tavsiflovchi funksiyadir algoritmga kirish miqdori bo'yicha oladi.
- **Kosmik murakkablik**xotira (bo'sh joy) miqdorini tavsiflovchi funksiyadir algoritm algoritmga kirish miqdori bo'yicha oladi.

Big-O belgisi

yordamida algoritmik murakkablikni ifodalashimiz mumkin**katta-O**yozuv. N o'lchamdagi muammo uchun:

- Doimiy vaqt funksiyasi/usuli*"buyurtma 1"*:**O(1)**
- Chiziqli vaqt funksiyasi/usuli*"buyurtma N"*:**O(N)**
- Kvadrat vaqt funksiyasi/usuli*"tartib N kvadrat"*:**O (N_2)**

Ta'rif:g va f natural sonlar to'plamidan o'ziga xos funktsiyalar bo'lsin. The f funksiyasi deyiladi**O(g)**(katta-oh ni o'qing), agar doimiy bo'lsa**c**va tabiiy **n_0**shu kabi**f (n)** ≤ **cg(n)**Barcha uchun**n > n_0.**

Eslatma:O(g) to'plamdir!

Belgini suiiste'mol qilish:f = O(g) f degani emas∈O(g).

Misollar:

- **$5n_2$+ 15 = O(n_2)**, beri**$5n_2$+ 15** ≤ **$6n_2$**, Barcha uchun**n > 4**.
- **5n2 + 15 = O(n_3)**, beri**$5n_2$+ 15** ≤ **n_3**, Barcha uchun**n > 6**.
- **O(1)**doimiyni bildiradi.

Katta-O belgisiga doimiylarni kiritishimiz mumkin bo'lsa-da, buni amalga oshirish uchun hech qanday sabab yo'q.

Shunday qilib, biz yozishimiz mumkin**O(5n + 4) = O(n)**.

Eslatma: The**katta-O**ifodalarda doimiy yoki past tartibli atamalar mavjud emas. Bu chunki, N yetarlicha kattalashganda, konstantalar va past tartibli atamalar muhim emas (doimiy vaqt funksiyasi/usuli chiziqli vaqt funksiyasi/usulidan tezroq bo'ladi, kvadratik vaqt funksiyasi/usulidan tezroq bo'ladi).

Vaqt murakkabliklarini nazariy jihatdan aniqlash

Umuman olganda, kod qismining ishlash vaqtini qanday aniqlash mumkin? Javob

Bu qanday turdagi bayonotlar ishlatilishiga bog'liq.

1. Bayonotlar ketma-ketligi

```
bayonot1;
bayonot2;
. . .
bayonot k;
```

Umumiy vaqt barcha bayonotlar uchun vaqtlarni qo'shish orqali topiladi:

```
totalTime = vaqt (bayonot1) + vaqt (bayonot2) +..+ vaqt (bayonotk)
```

2. `agar-boshqa` bayonotlar

```
agar (shart):
    # bayonotlar ketma-ketligi 1
boshqa:
    # bayonotlar ketma-ketligi 2
```

Bu erda ham**ketma-ketlik 1**amalga oshiradi, yoki**ketma-ketlik 2**amalga oshiradi. Shuning uchun, eng yomon vaqt ikki imkoniyatdan eng sekin hisoblanadi:

```
maksimal (vaqt (ketma-ket1), vaqt (ketma-ket2))
```

Masalan,agar 1-ket O(N) va 2-ketma O(1) bo‘lsa, if-then-else iborasi uchun eng yomon holat vaqti bo‘ladi.**O(N)**.

3. `halqalar`

```
N oralig'idagi i uchun:
    # bayonotlar ketma-ketligi
```

Bu erda tsikl bajariladi**N**marta, shuning uchun bayonotlar ketma-ketligi ham bajariladi**N** marta. Endi, barcha bayonotlar tartibda, deb faraz qiling**O(1)**, keyin jami uchun vaqt**uchun**halqa hisoblanadi**N * O(1)**, bu**O(N)**umumiy.

4. O‘rnatilgan halqalar

```
uchuniichidaN diapazoni:
    uchuniichidaM diapazoni:
        # bayonot
```

Tashqi halqa bajariladi**N**marta. Har safar tashqi halqa bajarilganda, ichki halqa bajariladi**M**marta. Natijada, ichki tsikldagi gaplar jami bajariladi**N * M**marta. Ichki pastadir ichidagi bayonotning murakkabligini hisobga olsak **O(1)**, umumiy murakkablik bo'ladi**O (N * M).**

Namuna muammosi:

'N' bo'yicha while tsiklidan keyingi vaqt murakkabligi qanday bo'ladi?

```
esaN>0:
    N = N//8
```

Biz iteratsiyalarni quyidagicha yozishimiz mumkin:

Takrorlash Raqam	**N qiymati**
1	N
2	Yo‘q//8
3	Yo'q//64
. . .	. . .
k	Yo‘q//8_k

Biz bilamizki, oxirgi, ya'nik_{th}iteratsiya, qiymati**N**aylanadi**1**, shuning uchun biz yozishimiz mumkin:

```
Yo'q//8k=1
=> N =8k
=> log(N) = log(8k) =>
k*log(8) = log(N) => k =
log(N)/log(8) => k = log8
(N)
```

Endi bu misoldagi takrorlashlar soni aniq bo'lib chiqadi

$\text{jurnal}_8(N)$. Shunday qilib, yuqoridagi while tsiklining vaqt murakkabligi bo'ladi $O(\log_8(N))$.

Sifat jihatdan shuni aytishimiz mumkinki, har bir takrorlashdan keyin berilgan sonni 8 ga bo'lamiz va raqam 0 dan katta bo'lmaguncha bo'linishda davom etamiz. Bu takrorlashlar sonini quyidagicha beradi. $O(\log_8(N))$.

Ba'zi umumiy algoritmlarning vaqt murakkabligi tahlili

Chiziqli qidiruv

Chiziqli qidiruv vaqtining murakkabligi tahlili quyida amalga oshiriladi: **Eng yaxshi holat -**Eng yaxshi holatda:

- Qidirilayotgan element birinchi holatda topiladi.
- Bunday holda, qidiruv faqat bitta taqqoslash bilan muvaffaqiyatli yakunlanadi.
- Shunday qilib, eng yaxshi holatda, chiziqli qidiruv algoritmini oladi**O(1)**operatsiyalar.

Eng yomon holat -Eng yomon holatda:

- Qidirilayotgan element oxirgi holatda bo'lishi yoki bo'lmasligi mumkin massivda umuman mavjud.
- Birinchi holda, qidiruv muvaffaqiyatli yakunlanadi**N**taqqoslashlar.
- Ikkinchi holda, qidiruv muvaffaqiyatsiz tugaydi**N**taqqoslashlar.
- Shunday qilib, eng yomon holatda, chiziqli qidiruv algoritmini oladi**O(N)**operatsiyalar.

Ikkilik qidiruv

Ikkilik qidiruv vaqtining murakkabligi tahlili quyida amalga oshiriladi -

- Har bir iteratsiyada yoki har bir rekursiv qo'ng'iroqda qidiruv yarmiga qisqaradi massiv.
- Shuning uchun**N**massivda elementlar mavjud**jurnal$_2$N**iteratsiyalar yoki rekursiv chaqiruvlar.

Shunday qilib, bizda -

- Ikkilik qidiruv algoritmining vaqt murakkabligi**O (log$_2$N)**.
- Bu yerda,**N**tartiblangan chiziqli massivdagi elementlar soni.

Ikkilik qidiruvning bu safar murakkabligi qanday bo'lishidan qat'i nazar, o'zgarishsiz qoladi elementning pozitsiyasi, hatto u massivda bo'lmasa ham.

Big-O notation amaliyotiga misollar

Misol-1uchun yuqori chegarani toping**f(n) = 3n + 8**

Yechim:$3n + 8 \leq 4n$, barcha $n \geq 8$ uchun

$\therefore 3n + 8 = O(n)$ c = 4 va n bilan$_0$= 8

Misol-2uchun yuqori chegarani toping**f(n) = n$_2$+ 1**

Yechim:$n_2 + 1 \leq 2n_2$, barcha $n \geq 1$ uchun

$\therefore n_2 + 1 = O(n_2)$ c = 2 va n bilan$_0$= 1

Misol-3uchun yuqori chegarani toping**f(n) = n$_4$+ 100n$_2$+ 50**

Yechim:$n_4 + 100n_2 + 50 \leq 2n_4$, barcha $n \geq 11$ uchun

$\therefore n_4 + 100n_2 + 50 = O(n_4)$ c = 2 va n bilan$_0$= 11