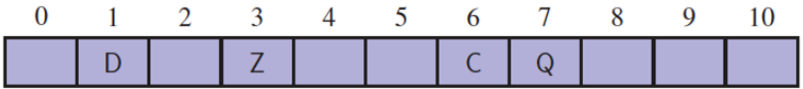
Upravljanje memorijom i B-stabla

Katedra za informatiku, Fakultet tehničkih nauka, Univerzitet u Novom Sadu

2023.

Memorija računara

- memorija je potrebna za implementaciju svake strukture podataka
- memorija je organizovana kao sekvenca reči gde se svaka reč sastoji od 4, 8 ili 16 bajtova (zavisno od računara)
- ove reči su numerisane od 0 do $N-1$, gde je N broj reči dostupnih računaru
- broj povezan sa svakom od reči zove se adresa



Kreiranje objekata

- u Python programu svi objekti se čuvaju u delu memorije koji se zove memory heap ili Python heap – ne treba mešati sa strukturom podataka koja se zove heap
- šta se dešava kada izvršimo nešto kao:

  ```
  w = Widget()
  ```

- kreira se nova instanca klase i skladišti se negde na heapu

Lista slobodnih blokova

- memorijski heap je podeljen u blokove – kontinualne „parčiće“ memorije koji mogu biti fiksne ili promenljive veličine
- mora biti moguće brzo zauzimanje memorije za nove objekte
- jedno popularno rešenje – čuvanje slobodnih „rupa“ u heapu u povezanoj listi, zvanoj lista slobodnih blokova
- odlučivanje kako dodeljivati blokove iz liste slobodnih prilikom zauzimanja (alokacije) memorije je deo upravljanja memorijom

Upravljanje memorijom

- postoji više načina za alokaciju memorije na heapu koji minimizuju fragmentaciju
  - best fit: pronađi u celoj listi onaj blok čija veličina je najbliža traženoj veličini
  - first fit: kreni od početka liste i pronađi prvi blok koji je dovoljno velik
  - next fit: traži se prvi sledeći dovoljno veliki blok počevši od prethodne pozicije; lista je cirkularna
  - worst fit: pronađi najveći slobodan blok

Sakupljanje đubreta

- garbage collection: proces otkrivanja „ustajalih“ objekata, oslobađanje memorije koju ti objekti zauzimaju, i vraćanje toga u listu slobodnih blokova
- da bi program mogao da pristupi objektu, mora imati referencu na njega (direktnu ili indirektnu)
  - takvi objekti su živi objekti
- živi objekti koji su direktno dostupni (postoji promenljiva koja sadrži referencu na njih) su korenski objekti
- indirektna referenca na živi objekat je referenca koja se nalazi u nekom drugom živom objektu

Brojanje referenci

- reference counting: svaki objekat ima uz sebe i brojač referenci na sebe; brojač se ažurira prilikom operacija dodele vrednosti
- kada brojač padne na nulu, objekat može da se ukloni jer je nedostupan
- ali šta kada dva objekta imaju reference jedan na drugog, a nisu dostupni spolja (cirkularne reference)?

Python i brojanje referenci

```
>>> import sys
>>> a = 'test'
>>> b = [a]
>>> c = {'key': a}
>>> sys.getrefcount(a)
4
>>>
```

- referenca `a`
- referenca u listi `b`
- referenca u rečniku `c`
- referenca u parametru funkcije `getrefcount` prilikom poziva :)

Cirkularne reference

- šta kada dva objekta imaju reference jedan na drugog, a nisu dostupni spolja (cirkularne reference)?
- ili objekat ima referencu na samog sebe?

```
>>> class MyClass:
...     pass
...
>>> a = MyClass()
>>> a.obj = a
>>> del a
```

Generational garbage collection

- GC prati sve objekte u memoriji
- svaki novi objekat počinje život u „prvoj generaciji“
- kada se pokrene GC proces i objekat preživi, seli se u narednu (drugu) generaciju
- svaka generacija ima limit na broj objekata koji može da primi
- Python ima 3 generacije za GC
- statistika kaže: većina objekata u programu su kratkog veka

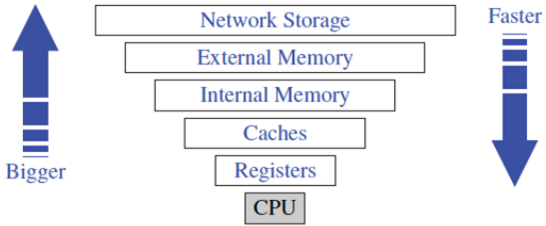
```
>>> import gc
>>> gc.get_threshold()
(700, 10, 10)
>>> gc.get_count()
(445, 3, 3)
>>> gc.collect()
115
>>> gc.get_count()
(24, 0, 0)
```

Java: mark-and-sweep algoritam

- svakom objektu dodeljena je oznaka (mark) da li je objekat živ
- kada odlučimo da je potrebno skupljati đubre, **zaustavimo sve druge aktivnosti** i
  - ukinemo mark za sve objekte na heapu
  - prođemo kroz sve module i sve korenske objekte označimo kao žive
  - odredimo da li su ostali objekti dostupni preko korenskih objekata – pretragom grafa po dubini

Hijerarhija memorije

- računari imaju hijerarhiju sa različitim vrstama memorije
- nivoi hijerarhije se razlikuju po veličini i udaljenosti od procesora
  - najbliži su interni registri procesora; pristup je vrlo brz ali ih ima vrlo malo
  - drugi nivo: cache memorija
  - treći nivo: operativna memorija (RAM)
  - četvrti nivo: spoljašnja memorija (diskovi)



Virtuelna memorija

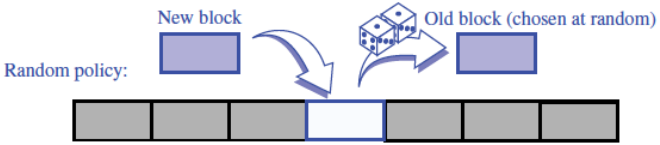
- virtuelna memorija: adresni prostor velik kao kapacitet spoljne memorije
- stranice se premeštaju iz spoljne u operativnu memoriju kada su potrebne
  - virtuelna memorija ukida ograničenje veličine operativne memorije
- koji deo čuvati u operativnoj memoriji: caching
- zahvaljujući vremenskoj lokalnosti
- učitavanjem stranice u operativnu memoriju nadamo se da će ona biti uskoro potrebna
- i da ćemo moći brzo da odgovorimo na sve zahteve za tom stranicom u bliskoj budućnosti

Strategije zamene blokova u operativnoj memoriji

- kada se traži nova stranica a operativna memorija je popunjena moramo izbaciti neku postojeću stranicu
- strategije za page replacement
  - LIFO
  - FIFO
  - random

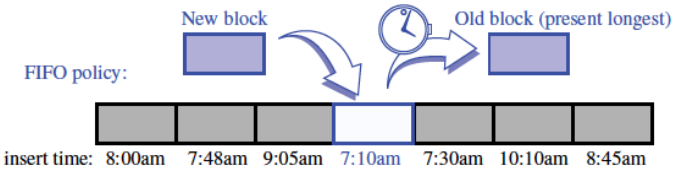
Random strategija

- izaberi stranicu koju ćeš izbaciti slučajnim putem
  - traje $O(1)$
  - ali ne pokušava da iskoristi vremensku lokalnost



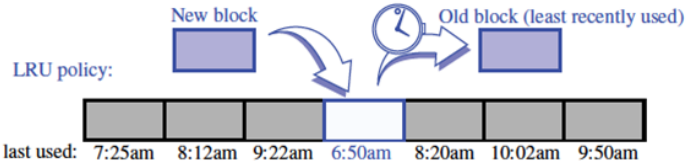
FIFO strategija

- jednostavna za implementaciju – potreban je red koji čuva reference na stranice u kešu
  - stranice se dodaju u red prilikom učitavanja
  - kada treba izbaciti stranicu, uklanja se prva stranica iz reda – $O(1)$
  - pokušava da iskoristi vremensku lokalnost



LRU strategija

- least recently used – najdavnije korišćena stranica
  - odlična politika ali implementacija može biti komplikovana
  - potreban je adaptivni red sa prioritetom
  - ako se implementira kao sortirana sekvenca pomoću povezane liste, uklanjanje je $O(1)$



Blokovi na disku

- čuvamo veliku kolekciju elemenata koja ne može stati u operativnu memoriju
- spoljnu memoriju smo podelili na disk blokove – red veličine 8KB
- prenos bloka između spoljne i operativne memorije je disk transfer ili I/O
- velika razlika između vremena pristupa operativnoj i spoljnoj memoriji
- $\Rightarrow$ želimo da minimizujemo broj disk transfera da bismo izvršili pretragu ili ažuriranje
- ovaj broj zovemo I/O kompleksnost algoritma

(a,b) stablo

- možemo predstaviti mapu za pretragu pomoću n-arnog stabla
- (a,b) stablo predstavlja uopštenje (2,4) stabla
  - (a,b) stablo je n-arno stablo u kome svaki čvor ima između a i b dece
- podešavanjem parametara a i b u odnosu na veličinu disk bloka možemo postići dobre I/O performanse

(a,b) stablo

- (a,b) stablo gde su a i b celobrojni parametri takvi da je $2 \leq a \leq (b+1)/2$ je n-arno stablo pretrage sa sledećim osobinama
  - veličina: svaki interni čvor osim korena ima najmanje a dece; koren ima najviše b dece
  - dubina: svi listovi imaju istu dubinu

Visina (a,b) stabla

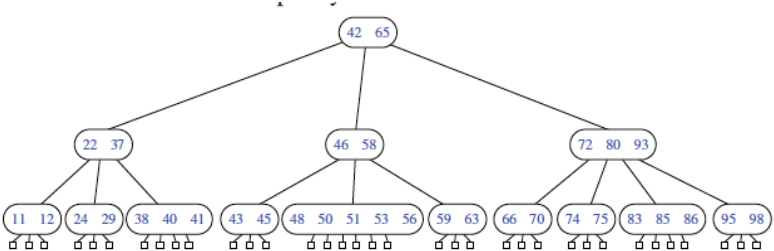
- visina (a,b) stabla sa n elemenata je
  - $\Omega(\log n / \log b)$
  - $O(\log n / \log a)$

Pretraga i ažuriranje u (a,b) stablu

- pretraga se odvija kao u n-arnom stablu
- dodavanje slično (2,4) stablu
  - overflow nastupa kada se dodaje element u b-čvor
  - čvor se deli pomeranjem median vrednosti u roditelja i zamenom čvora sa dva nova $(b+1)/2$-čvora
- uklanjanje slično (2,4) stablu
  - underflow nastupa kada se ukloni element iz a-čvora
  - ako je brat a-čvor radi se fuzija
  - ako brat nije a-čvor radi se transfer

B-stablo

- najpoznatija struktura za čuvanje mape u spoljnoj memoriji
- B-stablo reda d je (a,b) stablo za $a = d/2$ i $b = d$



I/O složenost B-stabla

- B-stablo sa n čvorova ima I/O složenost $O(\log_B n)$ za pretragu i ažuriranje, i troši $O(n/B)$ blokova, gde je B veličina bloka