Napredni algoritmi i strukture podataka

Predavači i asistenti:

prof.dr.sc. Damir Kalpić	: damir.kalpic@fer.hr
doc.dr.sc. Nikica Hlupić	: nikica.hlupic@fer.hr
Nikola Pavković, dipl.ing.	: nikola.pavkovic@irb.hr
Vedran Novaković, dipl.ing.	: venovako@fsb.hr
Mario Brčić, dipl.ing.	: mario.brcic@fer.hr

Administracija:

Zavod za primijenjeno računarstvo, III kat zgrada D

Tel: 6129-915 (gđa. Sonja Majstorović)

Obavijesti:

- web-stranica predmeta http://www.fer.hr/predmet/nasp
- vrata Zavoda za primijenjeno računarstvo (3. kat zgrade D)

Literatura:

- Adam Drozdek: Data Structures and Algorithms in C++, Thomson Course Technology 2005
- sve što je bilo preporučeno za ASP
  - Horowitz & Sahni: Fundamentals of Computer Algorithms, Pitman, London, 1995
  - Weiss: Data Structures and Algorithm Analysis in C, Addison Wesley, 1997
  - Sedgwick: Algorithms in C…, Addison-Wesley, 2001
  - Cormen, Leiserson & Rivest: Introduction to algorithms, 2/e,MIT Press, 2001
  - Donald E. Knuth, The Art of Computer Programming, Volumes 1–4, Addison-Wesley Professional
  - Wirth: Algorithms + Data Structures = Programs, Prentice-Hall, 1976
- sve što u naslovu ima “algorithm” ☺
- Internet i drugi izvori – oprezno!

Ocjenjivanje:

- elementi:
  - sudjelovanje u nastavi 5 %
  - domaće zadaće (?) 10 %
  - I međuispit 20 %
  - II međuispit 25 %
  - završni ispit 40 %

- za pozitivnu ocjenu (prolaz) treba ostvariti više od 50,00 % bodova.

Liste s preskakanjem (Skip Lists)

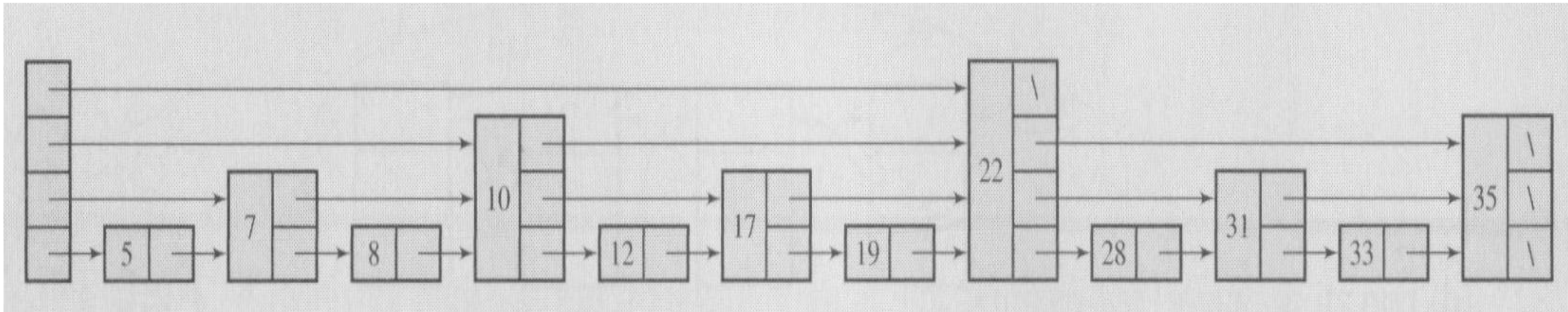
Pugh, William: "Skip lists: a probabilistic alternative to balanced trees“, Communications of the ACM 33, June 1990, pp. 668–676

Osnovni nedostatak lista: O(n) pretraživanje

Ograničavajuće svojstvo stabala: po prirodi hijerarhijske strukture, logički neprikladne za sve primjene

Skip liste: - ključne operacije $O(\log_2 n) \ldots O(n)$
- nema hijerarhije
- relativno jednostavno programiranje

Savršena (teorijska) struktura skip liste:



Stupanj (level) čvora = broj pokazivača

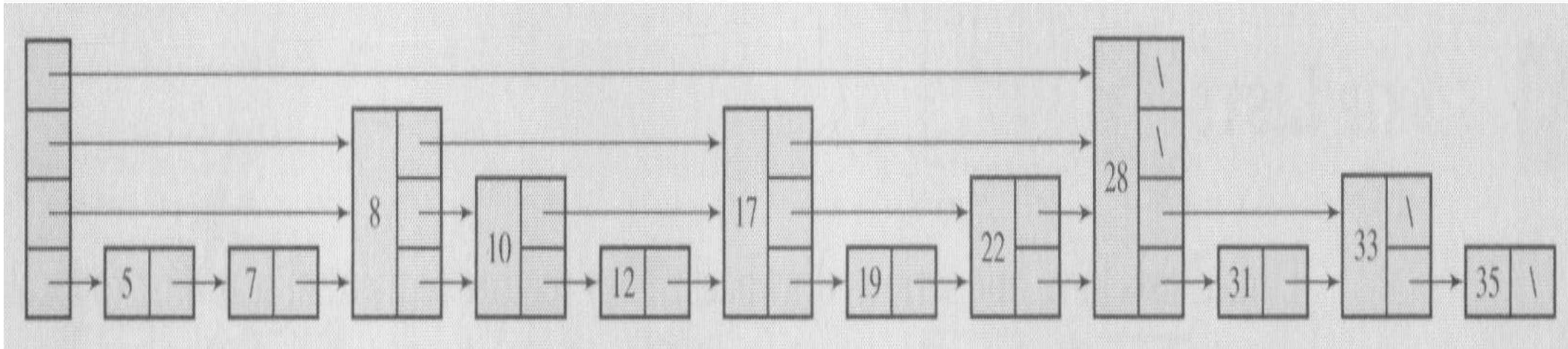
Održavanje savršene strukture je vrlo neučinkovito:

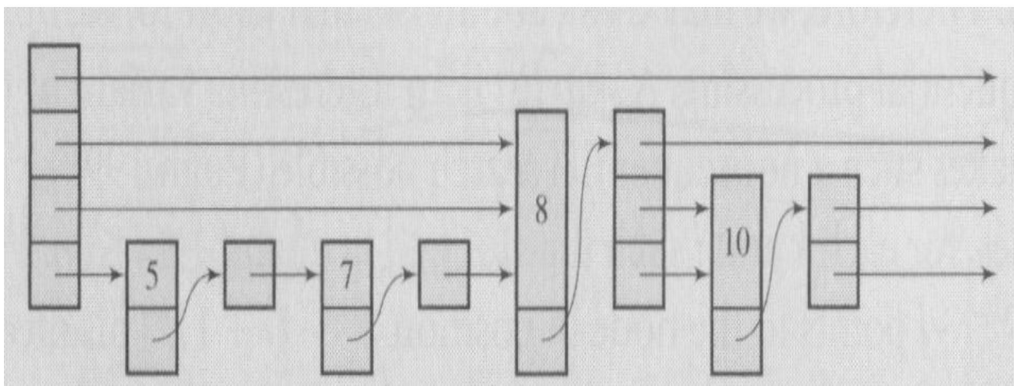
- nakon ubacivanja ili brisanja čvora treba restrukturirati sve čvorove iza njega (mijenjaju se stupnjevi čvorova, dakle i broj i odredišta pokazivača!)

U stvarnosti se odustaje od zahtjeva za pravilnim rasporedom čvorova i samo se *nastoji* postići pravilna razdioba njihovih stupnjeva.

“Nastoji” znači da se ni na tome ne inzistira bezuvjetno, nego se osigurava najveća vjerojatnost pravilne razdiobe stupnjeva čvorova u listi.

Uporabna struktura skip liste:





Brzina pristupa podatcima je u prosjeku sumjerljiva s brzinom u AVL ili RB stablu.

Uporabna struktura skip liste ovisi o dva čimbenika:

1. predviđenom kapacitetu n
   - to je najveći mogući broj elemenata u listi
2. vjerojatnosti pojedinih stupnjeva čvora
   - najčešće se zadaje samo vjerojatnost p "prelaska" čvora u višu razinu (razina=1 $\leftrightarrow$ vjerojatnost=1,
     razina=2 $\leftrightarrow$ vjerojatnost=p, ...,
     razina=k $\leftrightarrow$ vjerojatnost=p^{k-1})

Koji je stupanj liste potreban za smještaj n elemenata?

broj čvorova s barem jednim pokazivačem = broj elemenata u listi = n

broj čvorova s barem dva pokazivača = $n \cdot p$, ...

broj čvorova s barem k pokazivača = $n \cdot p^{k-1}$; $k = 1, 2, ..., h$

Stupanj liste h = stupanj najvišeg čvora

$$n \cdot p^{h-1} \geq 1 \Rightarrow \quad h \leq 1 + \log_p 1/n = 1 + \log_{1/p} n \quad ; \quad p < 1.$$

(uzeti floor(h) jer decimalni dio znači samo da za smještaj n elemenata trebamo i čvorove nižeg stupnja)

Ako se stupnjevi broje od nula, $h \rightarrow h+1 \quad \Rightarrow \quad h \leq \log_{1/p} n$.

<u>Primjer:</u> $p = ½$, $n = 12$

$$h \le 1 + \log_2 12 = 1 + 3,6 \quad \rightarrow \quad h = 4$$

s barem jednim pokazivačem: svi $= n$
s barem dva pokazivača: $½ \cdot n = 6$
s barem tri pokazivača: $(½)^2 \cdot n = 3$
s barem četiri pokazivača: $(½)^3 \cdot n = 3/2 \rightarrow 1$

Broj čvorova pojedinog stupnja?

najviših $= n \cdot p^{h-1} = 12/8 \rightarrow 1$
za jedan nižih $= n \cdot p^{h-1-1}$ – broj najviših $= 12/4 - 1 = 2$
za još jedan nižih $= n \cdot p^1$ – broj svih viših $= 12/2 - 3 = 3$
samo jedan pokazivač $= 12 - (1 + 2 + 3) = 6$

Određivanje stupnja novog čvora - osnovna ideja:

Teorija vjerojatnosti: ako je vjerojatnost uspješnog ishoda nekog pokusa p, vjerojatnost k uzastopnih uspješnih ishoda je p^k.

Ideja: ponavljamo pokus vjerojatnosti p sve dok završava uspješno, a broj uzastopnih uspjeha imat će razdiobu kakvu trebamo za stupnjeve čvorova.

```
int RandomLevel(p, maxListLevel)
{     int razina = 1;//razine se broje od =1
      while ((float) rand()/RAND_MAX < p)
           && (razina < maxListLevel))
           ++razina;
      return razina;    }
```

Zbog višekratnog izračunavanja slučajnog broja ovo nije učinkovito rješenje.

Učinkovitije određivanje stupnja novog čvora:

1. na temelju n i p izračunati h
2. odrediti broj n_k čvorova pojedinog stupnja (mora biti $\sum n_k = n$)
3. kapacitet liste n podijeliti u pretince (blokove) kapaciteta n_k
   - redni broj (stupanj) pretinca je jednak stupnju čvorova u njemu, a granice pretinaca određuju se iz n_k

   (npr. prvi pretinac ima granice $[1, n_1]$, drugi $[n_1+1, n_1+n_2]$, treći $[n_1+n_2+1, n_1+n_2+n_3]$ itd.)
4. izračunati slučajni broj x iz intervala $[1, n]$
5. stupanj čvora odgovara stupnju pretinca u koji "upada" x

Intervali se odrede na početku (jednom) pa se određivanje stupnja novog čvora svodi na izračunavanje jednog slučajnog broja i nekoliko usporedbi.

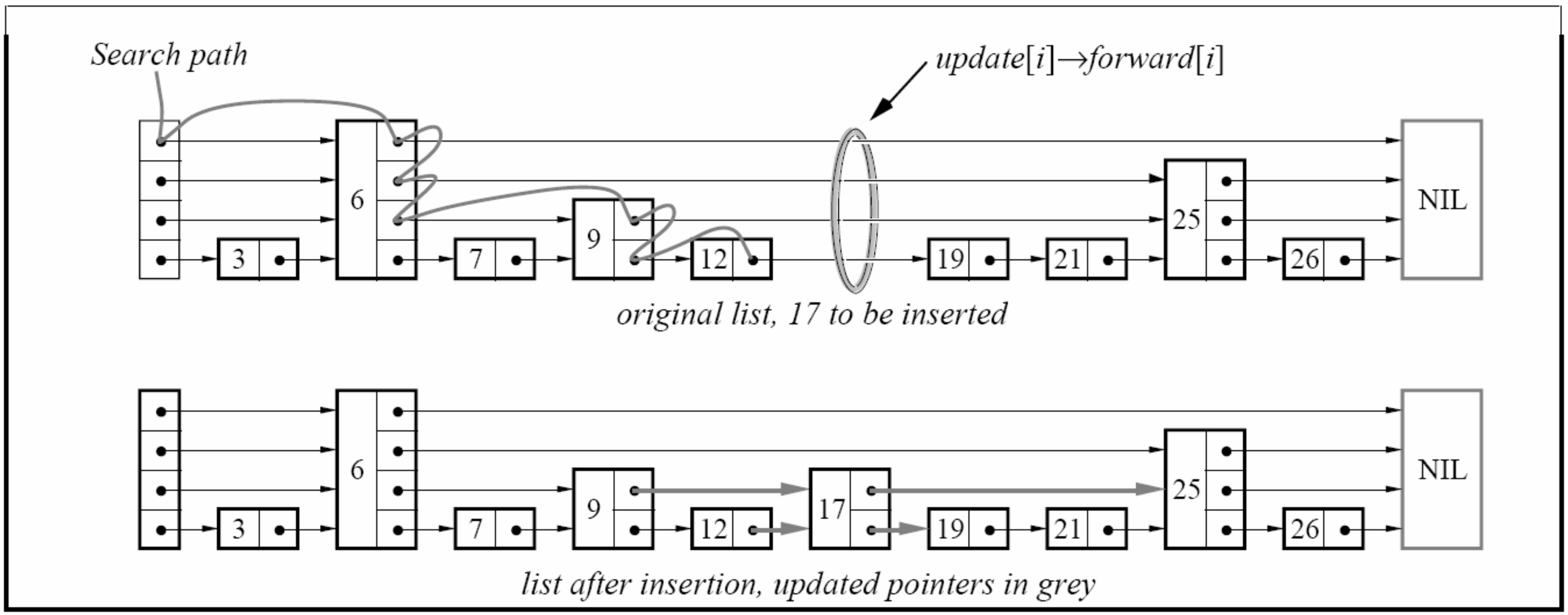
Primjer: p = ½, n = 12

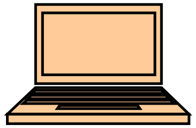
$$h \le 1 + \log_2 12 = 1 + 3{,}6 \quad \rightarrow \quad h = 4$$

stupanj	n_k	donja granica	gornja granica
1	6	1	6 (= n_1)
2	3	7 (= n_1+1)	9 (= n_1+ n_2)
3	2	10	11
4	1	12	12

Za niz slučajnih brojeva 5, 3, 11, 7 i 9 dodavali bi se čvorovi redom prvog, prvog, trećeg, drugog i drugog stupnja.

Pretraživanje i dodavanje čvora u skip-listu:





SkipList, SkipListObj

Pretraživanje skip-liste:

```
int ListSearch(SkipList* sl, int srkey)
{  //Vraća =1 ako je traženi element 'srkey' u listi 'sl', inače =0.
int i;
SkipNode* x = sl->header;   //'x' pokazuje čvor ispred onog
                            // čiji se ključ ispituje.
for(i = sl->ListLevel; i >= MinLevel; i--)
//'i' = razina; od najviše prema najnižoj...
{     while(x->next[i] != NULL
                  && x->next[i]->key < srkey)
            x = x->next[i];
      if (x->next[i]->key == srkey)
                return 1;
}
return 0;
}
```

Komentar: krenuti od najviše razine i pratiti listu na istoj razini sve dok se ne naiđe na traženi element, ključ veći od traženog (na toj razini nema traženog) ili NULL pokazivač (kraj liste na toj razini). Ako smo naišli na veći ključ ili kraj liste, spustiti se razinu niže i pregledati nju. Taj postupak ponavljati sve do dna liste.

Algoritam dodavanja čvora u skip-listu:

Dvije faze: - naći mjesto (bilježiti prethodnike na svim razinama!)
- dodati čvor

```
AddNode (list, key)
update[MaxLevel] = {NULL}   //prethodnici
//pretraživanje prilagođeno dodavanju i brisanju čvorova
x = list->head
for all levels i from ListLevel to MinLevel
{  while (x->next[i]-> NodeKey < key
              and x->next[i] != NULL)
        x = x->next[i]
   update[i] = x  }
if x->next[MinLevel] == NULL
   or x->next[MinLevel]->NodeKey != key
  //ako je uvjet ispunjen, takvog nema - dodati
  novog
```

Algoritam dodavanja čvora u skip-listu (nastavak):

```
//dodavanje novog čvora
level= new node level (call RandomLevel)
if level > ListLevel
    for all new levels i above ListLevel
        update[i] = list->head
    ListLevel = level
x = MakeNode(key, level)
for all levels i from MinLevel to level
    x->next[i] = update[i]->next[i]
    update[i]->next[i] = x
```

Algoritam uklanjanja čvora iz skip-liste:

Dvije faze: - naći (bilježiti prethodnike na svim razinama!)
- ukloniti i osloboditi memoriju

```
DelNode (list, key)
update[MaxLevel] = {NULL}
x = list->head
for all levels i from ListLevel to MinLevel
{  while (x->next[i]-> NodeKey < key
            and  x->next[i] != NULL)
         x = x->next[i]
   update[i] = x  }
node = x->next[MinLevel]
if node->NodeKey == key
  //u listi je - ukloniti ga
```

Algoritam uklanjanja čvora iz skip-liste (nastavak):

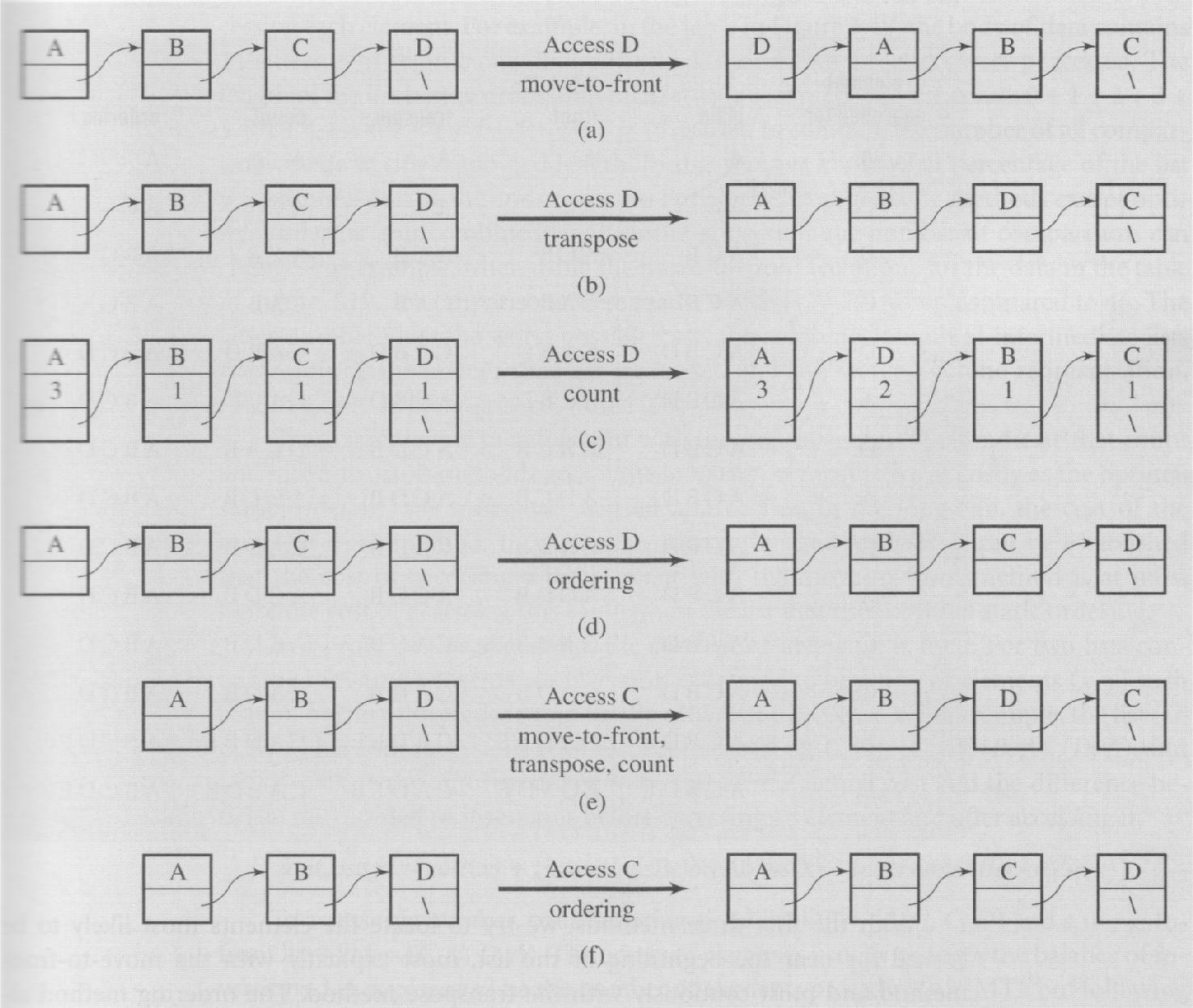
```
//uklanjanje čvora - preusmjeravanje prethodnika
for all levels i from MinLevel to ListLevel
   if update[i]->next[i] != node
         break
   update[i]->next[i] = node->next[i]
release memory occupied by node  //free(node)
//Po potrebi, zabilježiti promjenu visine liste.
while (list->head->next[list->ListLevel]
     == NULL
         and list->ListLevel > MinLevel)
     list->ListLevel--
```

Samoorganizirajuće liste (Self-Organizing Lists)

Pretraživanje “običnih” lista može se ubrzati i stalnim mijenjanjem poretka elemenata (*dinamical organizing*) u ovisnosti o raznim kriterijima. Na primjer:

1. *Move-to-front method*: nakon pristupa nekom elementu premjestiti ga na prvo mjesto
2. *Transpose method*: nakon pristupa nekom elementu zamijeniti mu mjesto s prethodnikom
3. *Count method*: elementi u poretku po broju pristupa
4. *Ordering method*: poredak po nekom kriteriju prirodnom za karakter elemenata (npr. po abecedi)

Primjer:



A
B
C
D
Access D
move-to-front
(a)
Access D
transpose
(b)
3
1
2
Access D
count
(c)
Access D
ordering
(d)
Access C
move-to-front,
transpose, count
(e)
Access C
ordering
(f)

U prve tri metode novi elementi se dodaju na kraj liste (slika *e*), dok se u četvrtoj ubaciju na mjesto određeno kriterijem poretka (slika *f*). U načelu sve su podjednako brze kad se primjenjuju u odgovarajućim sitacijama. Npr., ordering metoda može ustanoviti da traženog elementa uopće nema u listi i prekinuti pretraživanje, ali dodavanje novih je sporije nego u ostale tri.

Eksperimentalna analiza učinkovitosti tih metoda obično se temelji na odnosu stvarnog i najvećeg mogućeg broja usporedbi. Stvarni broj se dobiva brojanjem usporedbi tijekom testiranja, a najveći mogući zbrajanjem duljina liste prije svake potrage. Time se dobiva prosječni omjer pregledane i ukupne duljine liste tijekom cijelog testiranja.

Rijetko popunjene tablice (Sparse Tables)

Na primjer, kako pohraniti ocjene svih studenata iz svih predmeta na nekom fakultetu (ili cijelom sveučilištu!) u jednom semestru, gdje je ukupno P=300 ponuđenih predmeta i S=8000 studenata?

Prva pomisao - dvodimenzionalna tablica

[predmeti x studenti] s ocjenama u poljima.

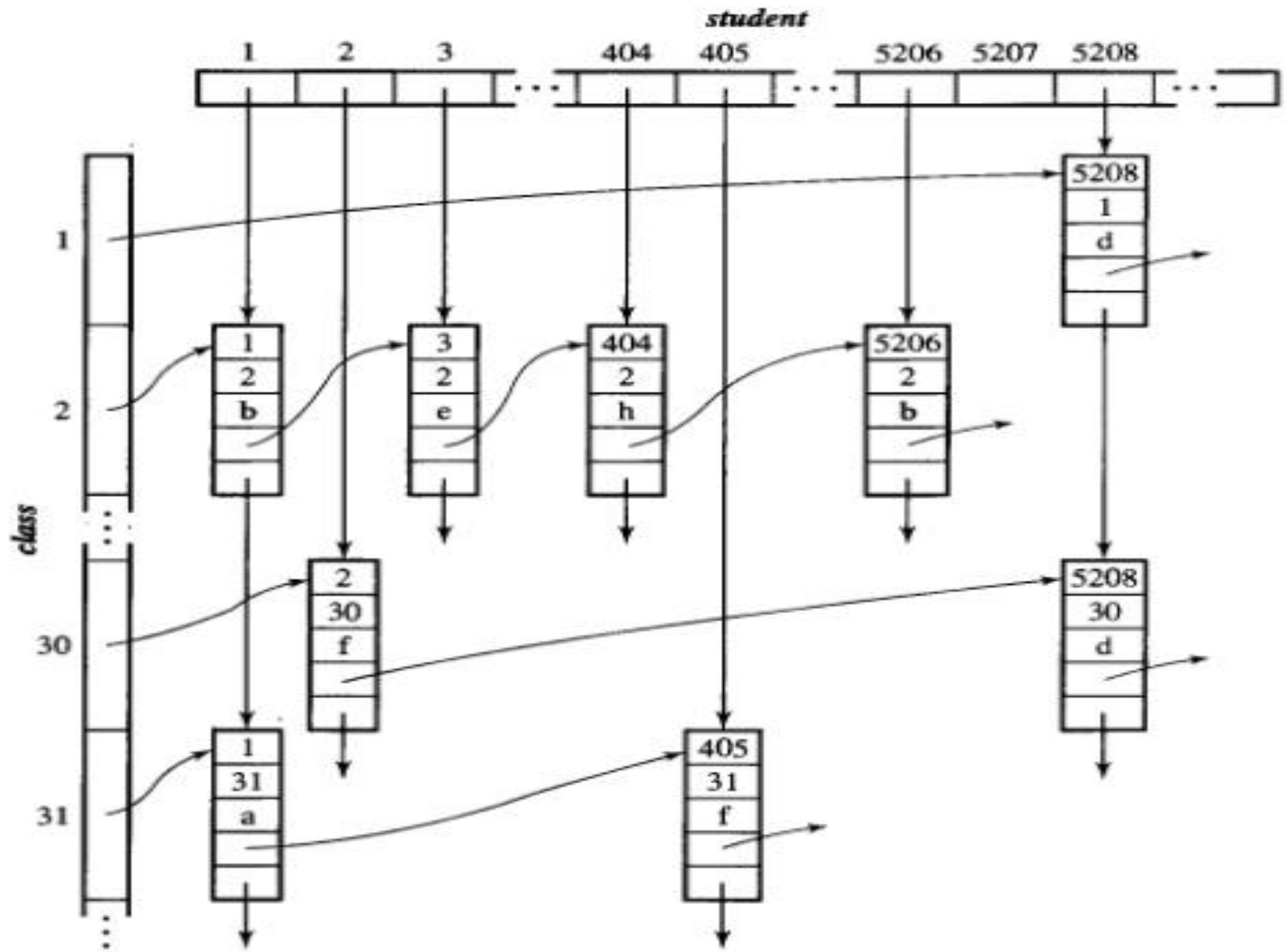
No, ako svaki student u prosjeku tijekom semestra položi PP=6 predmeta, popunjenost tablice bit će samo PP/P = 6/300 % = 2 %, znači “bacamo” 98 % rezervirane memorije.

Ako su ocjene podatak tipa char (1 byte), tablica zauzima ukupno $P \cdot S \cdot 1$ byte = $8000 \cdot 300$ byte = 2400000 byte $\approx$ 2,3 MB memorije.

Racionalnije je upotrijebiti dva jednodimenzionalna polja pokazivača Studenti i Predmeti, pri čemu je svaki element tih polja vrh (glava) liste pripadnih podataka. Na primjer, svaki element polja Studenti je vrh liste predmeta koje je taj student položio, dok su elementi polja Predmeti vrhovi lista studenata koji su položili određeni predmet.
Svaki element lista trebao bi sadržavati barem pet podataka (u zagradi su veličine tih podataka u byte):

- oznaku studenta (2)
- oznaku predmeta (2)
- ocjenu (1)
- pokazivač na sljedećeg studenta u listi (4)
- pokazivač na sljedeći predmet u listi (4)

Dakle, veličina podatka je 13 byte. Popunjenost te strukture je 100 %, a ukupna potrebna memorija 8000·6·13 byte = 624 000 byte ≈ 0,6 MB memorije.



Prednosti:

- racionalnije raspolaganje memorijom
- brzo pronalaženje svih podataka jedne skupine koji su u relaciji s jednim podatkom iz druge skupine (npr. svih studenata koji su položili neki predmet ili svih predmeta koje je položio neki student)

Nedostatci:

- sporiji pristup pojedinačnim podatcima; umjesto izravnog adresiranja u tablici, treba pretraživati listu
- čvorovi lista zauzimaju više memorije nego podatak u tablici pa takva struktura brzo prestaje biti svrhovita
  - kriterij: P·S·(veličina polja u tablici) > PP·S·(veličina čvora),

    dakle P·(veličina polja u tablici) > PP·(veličina čvora)

    $\Rightarrow$ u našem primjeru: PP < 300 · 1 / 13 $\approx$ 23