Podatkovne strukture in algoritmi

Andrej Brodnik
UP FAMNIT

Slovar

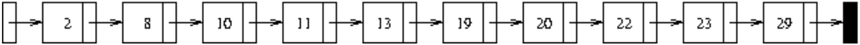
preskočni seznam

Seznami

Najpreprostejša oblika izvedbe slovarja je seznam, oziroma urejen seznam:

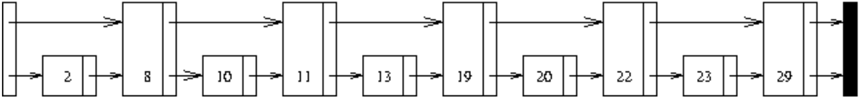
$$(2, 8, 10, 11, 13, 19, 20, 22, 23, 29)$$

Čas iskanja je sorazmeren dolžini seznama, oziroma, v najslabšem primeru moramo narediti n primerjav.
Zaradi poenostavitve prikaza, lahko seznam narišemu tudi takole:



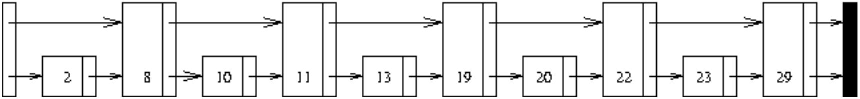
Seznami – 2

Iskanje lahko še enkrat pospešimo, če vsak drug element vsebuje referenco na element, ki je v seznamu dve mesti naprej:



Seznami – 2

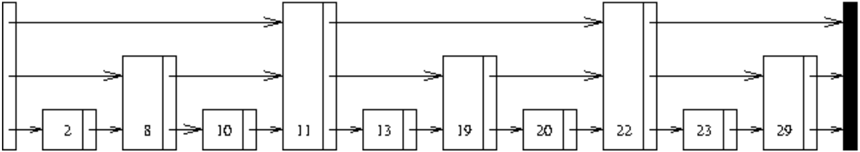
Iskanje lahko še enkrat pospešimo, če vsak drug element vsebuje referenco na element, ki je v seznamu dve mesti naprej:



V tem primeru naredimo samo še največ $\lceil n/2 \rceil + 1$ primerjavo.

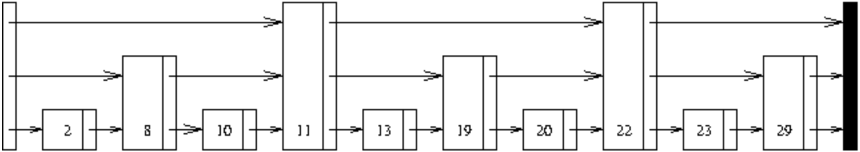
Seznami – 4

Zgodba se nadaljuje – iskanje lahko še enkrat pospešimo, če vsak četrti element vsebuje referenco na element, ki je v seznamu štiri mesta naprej:



Seznami – 4

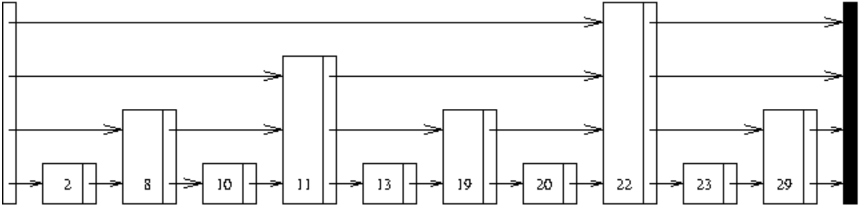
Zgodba se nadaljuje – iskanje lahko še enkrat pospešimo, če vsak četrti element vsebuje referenco na element, ki je v seznamu štiri mesta naprej:



V tem primeru naredimo samo še največ $\lceil n/4 \rceil + 2$ primerjavo.

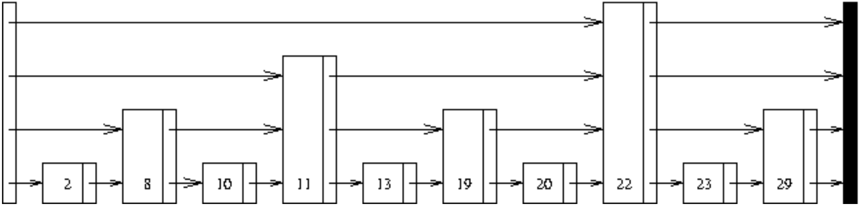
Seznami – k

V splošnem lahko element vsebuje referenco na podseznam (element), ki je $k = 2^i$ mest naprej. Koliko je lahko največ i?



Seznami – k

V splošnem lahko element vsebuje referenco na podseznam (element), ki je $k = 2^i$ mest naprej. Koliko je lahko največ i?



Največ $i = \lg n$ in v tem primeru naredimo največ

$$\lceil n/(2^i) \rceil + i = O(\log n)$$

primerjavo. Iskanje je učinkovito!

Preskočni seznami

DEFINICIJA: Element nivoja l vsebuje l referenc, ki jih označimo z $i \leq k$.

Preskočni seznami

DEFINICIJA: Element nivoja l vsebuje l referenc, ki jih označimo z $i \leq k$. Kaj pa vstavljanje? Struktura je zelo stroga, saj je bi vstavljanje elementa na določeno mesto (eno mesto pred polovico v seznamu dolžine $n = 2^m$) povzročilo tudi $O(n)$ sprememb. Kako?

Preskočni seznami

DEFINICIJA: Element nivoja l vsebuje l referenc, ki jih označimo z $i \leq k$.
Kaj pa vstavljanje? Struktura je zelo stroga, saj je bi vstavljanje elementa na določeno mesto (eno mesto pred polovico v seznamu dolžine $n = 2^m$) povzročilo tudi $O(n)$ sprememb. Kako?
V resnici si želimo:

pogostnost: imeti približno $n/2^l$ elementov nivoja l, kjer $l \leq \lceil \lg n \rceil$; in

porazdeljenost: da so elementi nivoja l enakomerno porazdeljena po seznamu.

Preskočni seznami

Definicija: Element nivoja l vsebuje l referenc, ki jih označimo z $i \leq k$.
Kaj pa vstavljanje? Struktura je zelo stroga, saj je bi vstavljanje elementa na določeno mesto (eno mesto pred polovico v seznamu dolžine $n = 2^m$) povzročilo tudi $O(n)$ sprememb. Kako?
V resnici si želimo:

pogostnost: imeti približno $n/2^l$ elementov nivoja l, kjer $l \leq \lceil \lg n \rceil$; in
porazdeljenost: da so elementi nivoja l enakomerno porazdeljena po seznamu.

Velja torej:

Preskočni seznami

DEFINICIJA: Element nivoja l vsebuje l referenc, ki jih označimo z $i \leq k$.
Kaj pa vstavljanje? Struktura je zelo stroga, saj je bi vstavljanje elementa na določeno mesto (eno mesto pred polovico v seznamu dolžine $n = 2^m$) povzročilo tudi $O(n)$ sprememb. Kako?
V resnici si želimo:

pogostnost: imeti približno $n/2^l$ elementov nivoja l, kjer $l \leq \lceil \lg n \rceil$; in

porazdeljenost: da so elementi nivoja l enakomerno porazdeljena po seznamu.

Velja torej:
INVARIANCA: Pri elementu nivoja l je referenca $i \leq l$ referenca na element, ki ima vsaj i nivojev.

Preskočni seznami

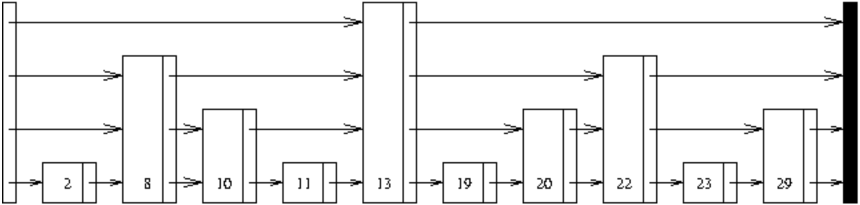
DEFINICIJA: Element nivoja l vsebuje l referenc, ki jih označimo z $i \leq k$.
Kaj pa vstavljanje? Struktura je zelo stroga, saj je bi vstavljanje elementa na določeno mesto (eno mesto pred polovico v seznamu dolžine $n = 2^m$) povzročilo tudi $O(n)$ sprememb. Kako?
V resnici si želimo:

pogostnost: imeti približno $n/2^l$ elementov nivoja l, kjer $l \leq \lceil \lg n \rceil$; in

porazdeljenost: da so elementi nivoja l enakomerno porazdeljena po seznamu.

Velja torej:
INVARIANCA: Pri elementu nivoja l je referenca $i \leq l$ referenca na element, ki ima vsaj i nivojev.
POSLEDICA: Za element nivoja l obstaja natančno en element za vsak $i \leq l$, ki ima referenco nivoja i nanj (lahko je glava celotnega preskočnega seznama).
Ali ima lahko isti element reference različnih nivojev na nek element?

Preskočni seznam – osnutek razreda




```
public class SkipList {
  Elt head; // pri glavi preskočnega seznama je glava prazna
  int level;
  SkipList[] tail; // teh je level
  ...
}
```

Iskanje

Ostaja nespremenjeno:

```
public Elt Find(int key, int atLevel) {
  if (head.key == key) return head;
  while ((atLevel > 0) && (tail[atLevel].Key > key))
    atLevel--;
  if (atLevel < 0) return NULL;
  return tail[atLevel].Find(key, atLevel);
}
```

Kako izgleda nerekurzivna metoda?

Kaj pa časovna zahtevnost? Odvisna je od pogostnosti in porazdeljenosti nivojev med elementi. Če so nivoji približno enako pogosti in enakomerno porazdeljeni, potem je $O(\log n)$.

Vstavljanje

Očitno moramo najprej najti pravo mesto v seznamu, kamor bomo vstavili novi element. In potem?
Na katerem nivoju naj bo vstavljeni element? Ohraniti moramo invarianco in obe lastnosti (pogostnost in porazdeljenost). Kako?

- Kako izgleda drevo pri vstavljanju, ko so podatki urejeni?
- Kako izgleda drevo pri vstavljanju, ko so podatki naključni?

Na pomoč nam priskoči naključnost

Vstavljanje

Očitno moramo najprej najti pravo mesto v seznamu, kamor bomo vstavili novi element. In potem?
Na katerem nivoju naj bo vstavljeni element? Ohraniti moramo invarianco in obe lastnosti (pogostnost in porazdeljenost). Kako?

- Kako izgleda drevo pri vstavljanju, ko so podatki urejeni?
- Kako izgleda drevo pri vstavljanju, ko so podatki naključni?

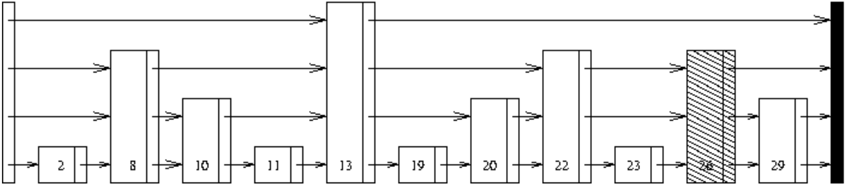
Na pomoč nam priskoči naključnost – v povprečju je naključno življenje znosno v redu.

Vstavljanje – ideja

- Ko vstavimo nov element v seznam se odločimo za njegovo višino nekako naključno: verjetnost, da bo nivoja 1 naj bo $1/2$, da bo nivoja 2 naj bo $1/4$ in da bo nivoja l naj bo $1/2^l$.
- To dosežemo tako, da mečemo kovanec toliko časa, da pade cifra ter število metov pomeni nivo elementa.

Vstavljanje – primer

V preskočni seznam s prosojnice 15 vstavimo element 26. Recimo, da vržemo tretjič cifro. Potem dobimo nov preskočni seznam:



Vstavljanje – ideja metode

Kako zagotoviti še (strukturno) invarianco?

> *Pri postopku vstavljanja, ko iščemo mesto, kamor bomo vstavili novi element, si vsakič, ko se spustimo za nivo, zapomnimo element, pri katerem smo se spustili.*

> POZOR: ko vstavljamo element, se vedno spustimo do nivoja 1 (glej metodo za iskanje).

Če je vstavljeni element nivoja l, potem vsi predhodniki do nivoja l morajo imeti referenco nanj.

Vstavljanje – ideja metode

```
public void SLInsert(int key, int atLevel, SkipList[] refsOn) {
  if (head.key == key) return; // že vstavljen
  while ((atLevel > 0) && (tail[atLevel].Key > key)) {
    refsOn[atLevel]= this;
    atLevel--;
  }
  if (atLevel < 0) ...; // tukaj vstavimo
  tail[atLevel].SLInsert(key, atLevel, refsOn);
}
```

Vstavljanje – ideja metode

```
public void SLInsert(int key, int atLevel, SkipList[] refsOn) {
  if (head.key == key) return; // že vstavljen
  while ((atLevel > 0) && (tail[atLevel].Key > key)) {
    refsOn[atLevel]= this;
    atLevel--;
  }
  if (atLevel < 0) ...; // tukaj vstavimo
  tail[atLevel].SLInsert(key, atLevel, refsOn);
}
```

Časovna zahtevnost je:

- za to, da najdemo mesto, kamor se vstavi element $O(\log n)$ (glej razlago pri iskanju) in
- za samo vstavljanje tudi $O(\log n)$, saj popravljamo največ dva krat toliko referenc.

Skupno je časovna zahtevnost vstavljanja $O(\log n)$.

Vstavljanje – ideja metode

```
public void SLInsert(int key, int atLevel, SkipList[] refsOn) {
  if (head.key == key) return; // že vstavljen
  while ((atLevel > 0) && (tail[atLevel].Key > key)) {
    refsOn[atLevel]= this;
    atLevel--;
  }
  if (atLevel < 0) ...; // tukaj vstavimo
  tail[atLevel].SLInsert(key, atLevel, refsOn);
}
```

Časovna zahtevnost je:

- za to, da najdemo mesto, kamor se vstavi element $O(\log n)$ (glej razlago pri iskanju) in
- za samo vstavljanje tudi $O(\log n)$, saj popravljamo največ dva krat toliko referenc.

Skupno je časovna zahtevnost vstavljanja $O(\log n)$.
Kaj pa prostorska?

Iskanje – ponovno

Kaj pa se zgodi sedaj s časovno zahtevnostjo iskanja? Porazdeljenost in pogostnost nivojev elementov se lahko poruši.

Iskanje – ponovno

Kaj pa se zgodi sedaj s časovno zahtevnostjo iskanja? Porazdeljenost in pogostnost nivojev elementov se lahko poruši.
Če je kovanec pravičen, potem se obe lastnosti ohranjata in tedaj je *pričakovana* vrednost časovna zahtevnost iskanja (ter tudi posledično vstavljanja) za katerikoli nabor elementov $O(\log n)$.

Iskanje – ponovno

Kaj pa se zgodi sedaj s časovno zahtevnostjo iskanja? Porazdeljenost in pogostnost nivojev elementov se lahko poruši.
Če je kovanec pravičen, potem se obe lastnosti ohranjata in tedaj je *pričakovana* vrednost časovna zahtevnost iskanja (ter tudi posledično vstavljanja) za katerikoli nabor elementov $O(\log n)$.
Seveda, najslabša vrednost je še vedno $O(n)$.

Iskanje – ponovno

Kaj pa se zgodi sedaj s časovno zahtevnostjo iskanja? Porazdeljenost in pogostnost nivojev elementov se lahko poruši.
Če je kovanec pravičen, potem se obe lastnosti ohranjata in tedaj je *pričakovana* vrednost časovna zahtevnost iskanja (ter tudi posledično vstavljanja) za katerikoli nabor elementov $O(\log n)$.
Seveda, najslabša vrednost je še vedno $O(n)$. Kdaj nastopi?

Iskanje – ponovno

Kaj pa se zgodi sedaj s časovno zahtevnostjo iskanja? Porazdeljenost in pogostnost nivojev elementov se lahko poruši.
Če je kovanec pravičen, potem se obe lastnosti ohranjata in tedaj je *pričakovana* vrednost časovna zahtevnost iskanja (ter tudi posledično vstavljanja) za katerikoli nabor elementov $O(\log n)$.
Seveda, najslabša vrednost je še vedno $O(n)$. Kdaj nastopi? Dva preprosta (trivialna) primera.

Iskanje – ponovno

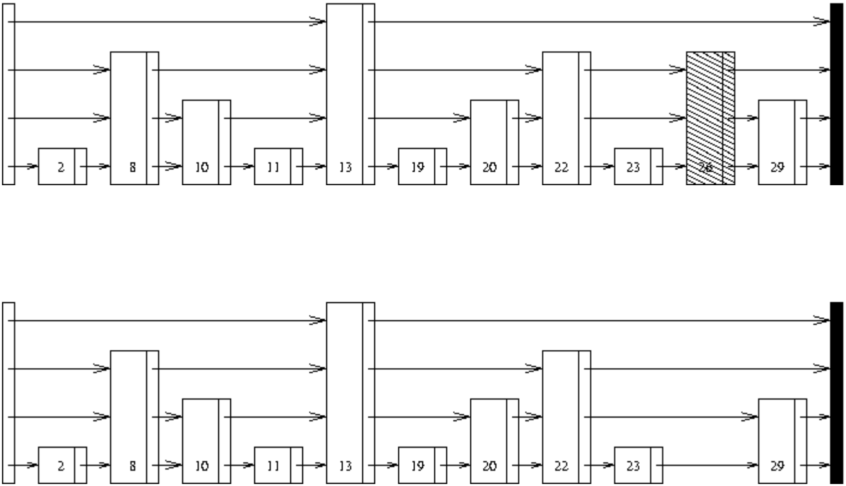
Kaj pa se zgodi sedaj s časovno zahtevnostjo iskanja? Porazdeljenost in pogostnost nivojev elementov se lahko poruši.
Če je kovanec pravičen, potem se obe lastnosti ohranjata in tedaj je *pričakovana* vrednost časovna zahtevnost iskanja (ter tudi posledično vstavljanja) za katerikoli nabor elementov $O(\log n)$.
Seveda, najslabša vrednost je še vedno $O(n)$. Kdaj nastopi? Dva preprosta (trivialna) primera. Ali je odvisna od elementov, ki jih vstavljamo?

Iskanje – ponovno

Kaj pa se zgodi sedaj s časovno zahtevnostjo iskanja? Porazdeljenost in pogostnost nivojev elementov se lahko poruši.
Če je kovanec pravičen, potem se obe lastnosti ohranjata in tedaj je *pričakovana* vrednost časovna zahtevnost iskanja (ter tudi posledično vstavljanja) za katerikoli nabor elementov $O(\log n)$.
Seveda, najslabša vrednost je še vedno $O(n)$. Kdaj nastopi? Dva preprosta (trivialna) primera. Ali je odvisna od elementov, ki jih vstavljamo?

Razmislite in/ali poskusite.

Brisanje – primer



Brisanje – ideja metode

Podobno kot pri vstavljanju – zagotoviti moramo strukturno invarianco:

> *Pri postopku brisanja, ko iščemo element, ki ga bomo izbrisali, si vsakič, ko se spustimo za nivo, zapomnimo element, pri katerem smo se spustili.*

RAZLIKA: Lahko ga najdemo, predno pridemo do nivoja 1.
Kaj pa sedaj pričakovana časovna zahtevnost?
Kakšna pa je prostorska zahtevnost?

Zahtevnost

	`Find`	`Insert`	`Delete`
seznam	$O(n)$	$O(1)$	$O(n)$
urejen seznam	$O(n)$	$O(n)$	$O(n)$
binarno drevo	$O(n)$	$O(n)$	$O(n)$
AVL drevo	$O(\lg n)$	$O(\lg n)$	$O(\lg n)$
B drevo	$O(\log_b n)$	$O(\log_b n)$	$O(\log_b n)$
RB-drevo	$O(\lg n)$	$O(\lg n)$	$O(\lg n)$
preskočna vrsta	$O(\log n)$	$O(\log n)$	$O(\log n)$

Zahtevnost

	`Find`	`Insert`	`Delete`
seznam	$O(n)$	$O(1)$	$O(n)$
urejen seznam	$O(n)$	$O(n)$	$O(n)$
binarno drevo	$O(n)$	$O(n)$	$O(n)$
AVL drevo	$O(\lg n)$	$O(\lg n)$	$O(\lg n)$
B drevo	$O(\log_b n)$	$O(\log_b n)$	$O(\log_b n)$
RB-drevo	$O(\lg n)$	$O(\lg n)$	$O(\lg n)$
preskočna vrsta	$O(\log n)$	$O(\log n)$	$O(\log n)$

- Kakšen je čas pri preskočni vrsti: največji, najmanjši, povprečni, pričakovan?
- Dâ se narediti največji (Munro, Papadakis).
- Lahko naključnost uporabimo pri drevesih? Kaj bi se poenostavilo?