Skip liste

Kao što smo već pomenuli, balansirana uređena binarna drveta se često koriste za realizaciju uređenih rečnika. Algoritmi za rad sa balansiranim drvetima tokom izvršavanja operacija umetanja i brisanja elemenata menjaju njihovu strukturu kako bi se održao uslov balansiranosti i osigurale dobre performanse.

Skip lista (brza lista ili lista sa prečicama) je probabilistička alternativa balansiranim drvetima. Ona predstavlja varijantu uređene povezane liste, međutim za razliku od klasične jednostruko povezane liste, skip lista sa velikom verovatnoćom ima dobru *očekivanu* složenost operacija pretrage, umetanja i brisanja elemenata. Interesantno je da se očekivana složenost ovde ne odnosi na raspodelu ključeva na ulazu, već na generator slučajnih brojeva koji se koristi prilikom umetanja novog elementa u skip listu.

Skip liste su randomizovana (probabilistička) struktura podataka: isti niz operacija umetanja i brisanja može da proizvede različite strukture. Ovo je zato što skip liste prilikom izgradnje strukture podataka za generisanje slučajnih brojeva koriste tehniku poznatu kao *nasumično bacanje novčića*. Tehnika nasumičnog bacanja novčića je ono što skip listama daje probabilističku prirodu i što omogućava dobru očekivanu složenost. Šta više u praksi se pokazuje da su skip liste jedna od najefikasnijih struktura podataka za implementaciju uređenih rečnika.

Razmotrimo običnu povezanu listu koja sadrži elemente *u rastućem poretku ključeva*. Naime, ključevi se koriste za uređenje elemenata u listi, dok vrednosti elemenata mogu biti proizvoljne. Primetimo da i pored sortiranosti ključeva nije moguće vršiti binarnu pretragu nad njom jer nemamo mogućnost indeksnog pristupa. Stoga je prilikom pretrage liste potrebno u najgorem slučaju pregledati svaki njen element (slika 1a). Ukoliko lista čuva elemente u sortiranom redosledu ključeva i ako bi svaki drugi element liste imao i pokazivač na element dva mesta ispred njega, onda bi maksimalni broj elemenata liste koje treba ispitati bio $\lceil n/2 \rceil + 1$ (gde je sa n označena ukupna dužina liste). Prilikom pretraživanja liste koriste se „proređeni“ pokazivači dok se ne naiđe na element sa vrednošću ključa većom od tražene: tada se pretraga nastavlja putem osnovnih pokazivača (slika 1b). Opišimo postupak pretrage u slučaju kada lista čuva pet elemenata u sortiranom poretku, a mi tražimo element koji se nalazi na četvrtom mestu u listi. Najpre bismo ispitali prvi element liste, zaključili da je vrednost elementa koji tražimo veća od njegove vrednosti ključa i pokazivačem koji preskače po dva uzastopna elementa liste ispitali njen treći element. Pošto je i njegova vrednost ključa manja od tražene, kao naredni element ispitali bismo njen peti element. Međutim, vrednost ključa petog elementa liste je veća od tražene, te bismo iz trećeg elementa liste idući pokazivačem koji povezuje susedne elemente liste ispitali njen četvrti element i zaključili da se tražena vrednost nalazi na tom mestu. Dakle ukupno bismo analizirali $\lceil 5/2 \rceil + 1 = 4$ elementa liste.

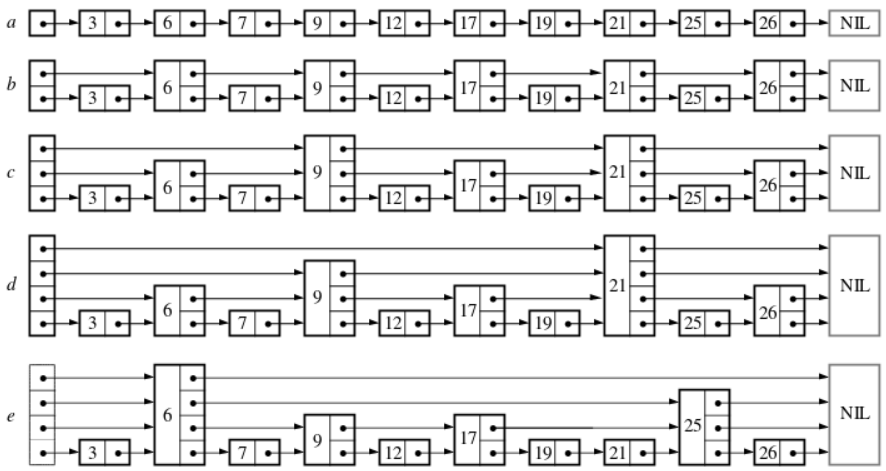
Ako bi svaki četvrti element liste pored navedenih pokazivača imao i pokazivač ka elementu liste četiri mesta ispred njega, maksimalni ukupan broj elemenata liste koje bi trebalo analizirati bio bi $\lceil n/4 \rceil + 2$ (slika 1c). Ako bi svaki 2^i-ti čvor u listi imao pokazivače na čvorove $2^0, 2^1, \ldots, 2^i$ mesta ispred njega u listi, maksimalni broj elemenata koje bi trebalo analizirati bi se smanjio na $O(\log_2 n)$ (slika 1d).

Razmotrimo koliki bi bio ukupan broj pokazivača u ovakvoj listi. Pretpostavimo da lista sadrži n čvorova. Svih n čvorova ima pokazivač na sledeći čvor, $n/2$ čvorova ima dodatni pokazivač ka čvoru dva mesta ispred njega, $n/4$ čvorova ima pokazivač ka čvoru četiri mesta ispred njega, itd. Stoga je ukupan broj pokazivača jednak:

$$n + \frac{n}{2} + \frac{n}{4} + \frac{n}{8} + \ldots + \frac{n}{2^{\log_2 n}} = n \sum_{i=0}^{\log n} \frac{1}{2^i} < 2n$$

Zaključujemo da bi ukupan broj pokazivača u ovakvoj listi bio dvostruko veći od broja pokazivača u klasičnoj jednostruko povezanoj listi.

U ovako definisanoj strukturi podataka bi se operacija pretrage izvodila veoma efikasno, međutim, operacije brisanja i umetanja elemenata bi bile potpuno nepraktične. Naime, nakon brisanja ili umetanja elementa, potencijalno bismo morali da za veliki broj čvorova ažuriramo njihove nivoe, tj. broj pokazivača koji iz njih vodi, kao i da pravilno povežemo čvorove pokazivačima. Poželjno je da prilikom umetanja i brisanja elemenata iz strukture izmene budu *lokalne*. To možemo postići ukoliko ne fiksiramo unapred koje pozicije u listi imaju koliki broj pokazivača.



Slika 1: Povezana lista sa dodatnim pokazivačima.

Nazovimo čvor koji ima k pokazivača *čvorom nivoa k*. Ukoliko bi svaki 2^i-ti čvor imao pokazivače na čvorove $2^j, 0 \leq j < i$ mesta ispred njega, onda bi nivoi čvorova imali sledeću raspodelu: 1/2 čvorova bilo bi nivoa 1, 1/4 čvorova nivoa 2, 1/8 njih nivoa 3, itd. U idealnom slučaju lista na svakom narednom nivou sadrži tačno pola elemenata sa prethodnog nivoa. Međutim, kao što smo već napomenuli, ovakav način organizacije nije pogodan za operacije umetanja i brisanja, te se ovaj zahtev mora relaksirati tako da je *očekivani* broj elemenata na narednom nivou polovina broja elemenata sa prethodnog.

Šta bi se desilo ako bi nivoi čvorova bili birani na slučajan način, ali u istoj proporciji (slika 1e)? Čvorov i-ti pokazivač bi onda, umesto da pokazuje na čvor 2^{i-1} mesta ispred njega, pokazivao na naredni čvor nivoa i ili većeg. U ovakvoj organizaciji liste operacije umetanja i brisanja elementa bi zahtevale samo lokalne izmene. Pritom, nivo čvora izabran na slučajan način prilikom kreiranja ne bi morao nikada da se menja. Ovako opisanu strukturu podataka nazivamo *skip listom*.

U skip listi, elementi su organizovani po nivoima, tako da svaki naredni nivo ima manji broj elemenata od prethodnog nivoa. Krajnji donji nivo predstavlja regularnu povezanu listu, dok nivoi iznad njega sadrže pokazivače koji omogućavaju brži prelaz između elemenata koji su međusobno udaljeni na donjem nivou. Ideja je omogućiti brz prelazak do željenog elementa. Otud i potiče ime strukture jer viši nivoi liste omogućavaju da se „preskoči“ određen broj elemenata liste. Skip liste je 1989. godine izumeo Vilijem Pu (William Pugh).

Pokazaćemo da je u skip listi očekivani broj čvorova nivoa 1 jednak $n/2$, nivoa 2 je $n/4$, nivoa 3 je $n/8, \ldots$ nivoa $\log n$ je 1. Stoga je očekivani broj nivoa skip liste jednak $O(\log n)$.

Operacije sa skip listom

Razmotrimo kako se nad skip listom izvode osnovne operacije: pretraga, umetanje i brisanje elementa.

Svaki element skip liste predstavljen je čvorom čiji se nivo bira na slučajan način prilikom umetanja u listu. Čvor nivoa i ima i pokazivača, sa indeksima redom od 1 do i. Maksimalni dozvoljeni nivo čvora je zadat konstantom `MaxNivo`. *Nivo liste* je maksimalni nivo čvora u listi (ili 1 ako je lista prazna). Glava liste (početni element, koji postoji i u praznoj listi) ima pokazivače na nivoima od 1 do `MaxNivo`. Pokazivači glave liste na nivoima većim od trenutno maksimalnog nivoa liste pokazuju na poslednji element liste, tj. na element `NIL`.

Inicijalizacija skip liste Inicijalizacija skip liste se sastoji od dva koraka:

- alocira se element `NIL` i pridružuje mu se vrednost veća od svih dozvoljenih vrednosti za ključ (kako bi uvek bio poslednji),
- inicijalizuje se nova lista čiji je nivo jednak 1, a kod koje svi pokazivači glave liste pokazuju na element `NIL`.

Dakle, skip lista sadrži dva sentinel čvora: glavu liste (kojoj se pridružuju minimalne vrednosti ključeva) i rep liste, tj. element `NIL`, koji ima maksimalnu pridruženu vrednost ključa.

Traženje elementa u skip listi Traženje elementa u skip listi sa ključem jednakim zadatom broju k počinje na najvišem nivou liste. Prelazi se kroz elemente liste na tom nivou sve dok se ne naiđe na čvor čija je vrednost ključa veća ili jednaka k. U prethodnom čvoru na tom nivou spuštamo se na nivo za 1 niži od tekućeg. Postupak se ponavlja na tom i svim nižim nivoima i završava se na najnižem, osnovnom nivou. Ako se na osnovnom nivou naiđe na čvor sa ključem jednakim k, pretraga je uspešno završena, inače se zaključuje da se element sa datom vrednošću ključa ne nalazi u listi.

Pod pretpostavkom da svaki element skip liste ima polja *kljuc*, *vrednost* i niz pokazivača *naredni*, a da lista ima član *nivo*, postupak traženja elementa skip liste L sa vrednošću ključa k može se opisati narednim kodom.

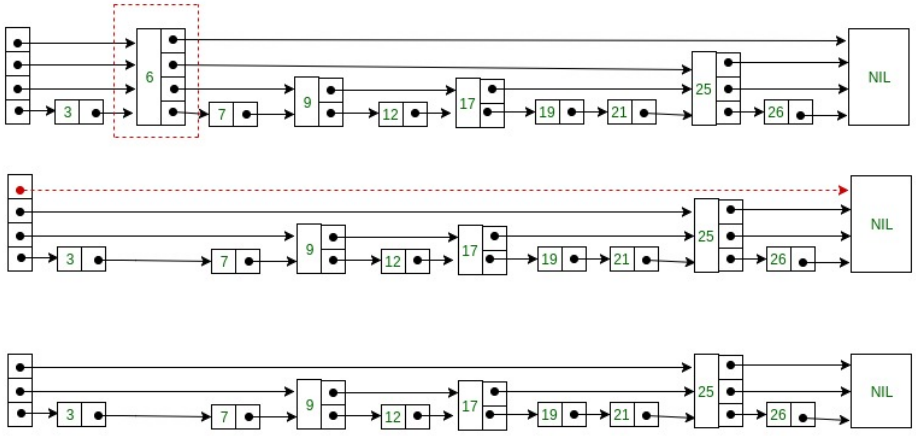
Algoritam Search(L, k);
Ulaz: L (skip lista) i k (ključ elementa koji se traži).
Izlaz: vrednost elementa sa datim ključem ili *neuspeh* ako se element ne nalazi u listi.
begin
    $tekuci := L \rightarrow glava$
    for $i := L \rightarrow nivo$ **downto** 1 **do**
        while $tekuci \rightarrow naredni[i] \rightarrow kljuc < k$ **do**
            $tekuci := tekuci \rightarrow naredni[i]$
    $tekuci := tekuci \rightarrow naredni[1]$
    if $tekuci \rightarrow kljuc = k$ **then return** $tekuci \rightarrow vrednost$
    else return *neuspeh*
end

Na primer, ako se u skip listi prikazanoj na slici 1e traži čvor sa vrednošću ključa 12, polazi se od čvora 6 i nivoa 4, pa se nastavlja sa istim čvorom na nivou 3 i nivou 2, dolazi se do čvora 9 na nivou 2, a zatim spušta na nivo 1 u istom čvoru. Pošto sused čvora 9 na nivou 1 ima vrednost ključa koja nije manja od k, iskače se iz petlje, pri čemu u tom trenutku promenljiva *tekuci* ukazuje na čvor 9 na nivou 1. Naredno premeštanje vodi nas u čvor sa traženom vrednošću ključa 12.

Brisanje elementa iz skip liste Da bismo obrisali element iz skip liste, potrebno je da pretražimo skip listu, nađemo čvor koji treba obrisati i da izvršimo prevezivanje pokazivače. Prilikom potrage za elementom koji treba obrisati podaci neophodni za brisanje pamte se u pomoćnom vektoru *prethodni* dužine jednake broju nivoa liste. Preciznije, *prethodni*$[i]$ treba da sadrži pokazivač na najdesniji čvor nivoa i ili višeg koji se nalazi levo od elementa koji treba obrisati.

Na primer, ako iz skip liste sa slike 2 želimo da obrišemo čvor sa vrednošću

ključa 6, prvo pronalazimo taj čvor. On je nivoa 4. Vrednosti *prethodni*[4], *prethodni*[3] i *prethodni*[2] pokazuju na glavu liste, a *prethodni*[1] na čvor sa vrednošću ključa 3. Čvor sa vrednošću ključa 6 briše se tako što se prevezuju pokazivači na nivoima manjim ili jednakim od njegovog nivoa na sledeći način: elementi niza *prethodni* treba da pokazuju na čvorove na koje su pokazivali redom pokazivači istog nivoa obrisanog čvora: u ovom slučaju na `NIL`, 25, 9 i 7.



Slika 2: Ilustracija brisanja elementa sa vrednošću 6 iz skip liste.

Nakon svakog brisanja, vrši se provera da li je obrisani čvor bio maksimalnog nivoa i, ako jeste, da li je jedini čvor sa maksimalnim nivoom u listi; ako je to slučaj (kao u primeru ilustrovanom na slici 2), smanjuje se nivo liste. To možemo uraditi tako što proveravamo da li nakon brisanja elementa liste pokazivač glave liste na trenutnom nivou liste pokazuje na `NIL`. Sve dok je to slučaj, nivo liste smanjuje se za 1.

Umetanje elementa u skip listu Da bismo umetnuli čvor u skip listu, kao i kod brisanja, potrebno je da pretražimo listu kako bismo pronašli mesto na koje treba umetnuti novi čvor, a zatim i da prevežemo pokazivače na odgovarajući način (slika 3). Za prevezivanje pokazivača nam je opet potreban vektor *prethodni* koji, kao i u slučaju brisanja elementa iz liste, na poziciji i sadrži pokazivač na najdesniji čvor nivoa i ili višeg koji se nalazi levo od pozicije na kojoj treba izvršiti umetanje elementa.

Razmotrimo skip listu ilustrovanu na slici 3 i operaciju umetanja elementa sa vrednošću ključa 17. U ovom slučaju *prethodni*[4] i *prethodni*[3] pokazuju na čvor sa vrednošću 6, *prethodni*[2] na čvor sa vrednošću 9, a *prethodni*[1] na 12. Nivo novog čvora bira se na slučajan način: u primeru na slici on je 2. Nakon toga vrši se prevezivanje pokazivača na nivoima manjim ili jednakim nivou čvora.

Ako u skip listi već postoji element sa datom vrednošću ključa, onda se samo ažurira vrednost elementa sa datim ključem.

```
Algoritam Delete(L, k);
Ulaz: L (skip lista) i k (ključ elementa koji treba obrisati).
begin
    tekuci := L → glava
    for i := L → nivo downto 1 do
        while tekuci → naredni[i] → kljuc < k do
            tekuci := tekuci → naredni[i]
        prethodni[i] := tekuci
    tekuci := tekuci → naredni[1]
    if tekuci → kljuc = k then
        for i := 1 to L → nivo do
            if prethodni[i] → naredni[i] ≠ tekuci then break {premašili smo nivo čvora tekuci}
            prethodni[i] → naredni[i] := tekuci → naredni[i]
        free(tekuci)
        while L → nivo > 1 and L → glava → naredni[L → nivo] = NULL do
            L → nivo := L → nivo − 1
end
```

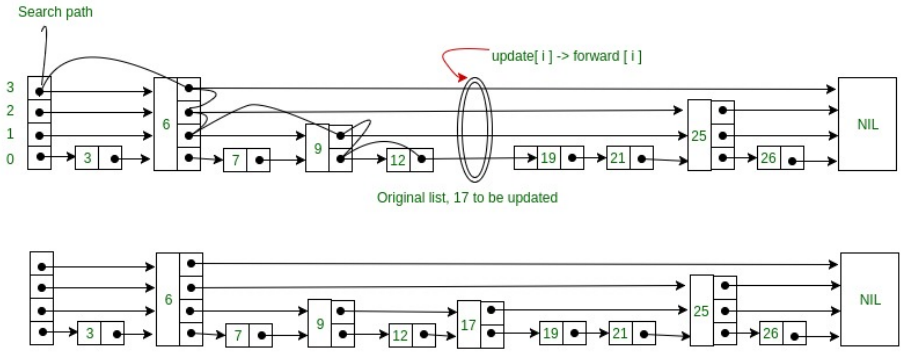
Ukoliko umetanje generiše čvor većeg nivoa nego što je bio prethodni nivo liste, ažuriramo nivo skip liste i inicijalizujemo nove elemente niza *prethodni*.

Biranje nivoa čvora na slučajan način Postavlja se pitanje kako na slučajan način izabrati nivo novog čvora. Pretpostavimo da želimo da procenat p čvorova nivoa i ima pokazivače nivoa $i+1$ (najčešće se bira $p = 1/2$ i mi ćemo to pretpostaviti u daljem tekstu). Da bi se odredio nivo novog čvora koristi se generator slučajnih brojeva, koji na izlazu daje slučajni broj iz intervala $[0, 1)$ sa ravnomernom raspodelom verovatnoća. Svaki put kada se na izlazu iz generatora dobije broj manji od p (što se dešava sa verovatnoćom jednakom upravo p), nivo se povećava za 1, sem ako već nije dostignut maksimalni dozvoljeni nivo liste.[1]

Može se desiti da u skip listi koja sadrži 16 elemenata generisanih na ovaj način dobijemo 9 elemenata nivoa 1, 3 elementa nivoa 2, 3 elementa nivoa 3 i jedan element nivoa 14. Ako bismo pretragu vršili korišćenjem prethodno navedenog algoritma, imali bismo puno praznog hoda. Postavlja se pitanje na kom nivou bi bilo pogodnije započeti pretragu elementa? Sprovedena analiza sugeriše da bi idealno bilo započeti pretragu na nivou $L(n)$ na kome očekujemo $1/p = 2$ čvora. Ovo se dešava za nivo $L(n) = \log_{1/p} n = \log_2 n$. Dakle, umesto da krećemo od nivoa liste, u algoritmu je poželjnije pretragu započeti od nivoa $L(n)$.

Primetimo da ni za jednu od navedenih operacija nije potrebno da u samom

[1] Primetimo da ova funkcija nije konstantne vremenske složenosti. Naime, ovakav način generisanja nivoa, koji prati ideje predstavljene u originalnom radu o skip listama, nekada može biti neefikasan jer podrazumeva potencijalno veći broj poziva funkcije *random*(), doduše dva poziva u proseku.



Slika 3: Ilustracija dva koraka od kojih se sastoji operacija umetanja elementa sa vrednošću ključa 17 u datu skip listu.

Algoritam Insert(L, k, v);
Ulaz: L (skip lista), k (ključ elementa koji se dodaje) i v (vrednost elementa koji se dodaje).
begin
    $tekuci := L \rightarrow glava$
    for $i := L \rightarrow nivo$ **downto** 1 **do**
        while $tekuci \rightarrow naredni[i] \rightarrow kljuc < k$ **do**
            $tekuci := tekuci \rightarrow naredni[i]$
        $prethodni[i] := tekuci$
    $tekuci := tekuci \rightarrow naredni[1]$
    if $tekuci \rightarrow kljuc = k$ **then** {već postoji ta vrednost ključa, samo ažuriramo vrednost}
        $tekuci \rightarrow vrednost := v$
    else
        $nivo := SlucajniNivo()$ { na slučajan način bira se nivo novog čvora }
        if $nivo > L \rightarrow nivo$ **then** { ako je nivo novog čvora veći od tekućeg nivoa liste}
            for $i := L \rightarrow nivo + 1$ **to** $nivo$ **do**
                $prethodni[i] := L \rightarrow glava$
            $L \rightarrow nivo := nivo$
        $tekuci := makeNode(nivo, k, v)$
        for $i := 1$ **to** $nivo$ **do**
            $tekuci \rightarrow naredni[i] := prethodni[i] \rightarrow naredni[i]$
            $prethodni[i] \rightarrow naredni[i] := tekuci$
end

čvoru čuvamo informaciju o njegovom nivou.

Složenost osnovnih operacija sa skip listom Vremenima potrebnim za izvršavanje operacija pretrage, umetanja i brisanja dominira vreme potrebno za

```
Algoritam SlucajniNivo();
Izlaz: nivo čvora.
begin
    nivo := 1
    while random() < p and nivo < MaxNivo do
        nivo := nivo + 1
    return nivo
end
```

nalaženje odgovarajućeg elementa u skip listi, pri čemu kod operacija umetanja i brisanja elementa postoji dodatna cena proporcionalna nivou čvora koji se umeće ili briše. Vreme potrebno za pronalaženje elementa je proporcionalno dužini puta pretrage. Podsetimo se da je očekivani broj nivoa skip liste $O(\log n)$, ali je i dalje potrebno proveriti da je broj koraka na svakom nivou mali.

Pokazuje se da je lakše analizirati dužinu puta pretrage, ako ga posmatramo unazad, počev od čvora sa vrednošću ključa koji smo tražili, napredujući ka višim nivoima i ulevo ka glavi liste.

Označimo sa $C(k)$ *očekivanu* cenu (tj. dužinu) puta ako smo na k-tom nivou računato odozgo. Primetimo da se pri pretrazi penjemo naviše uvek kada možemo jer se u neki čvor uvek dolazi preko njegovog najvišeg nivoa; samo ukoliko ne možemo da se krećemo naviše idemo ulevo. Prilikom procene očekivane dužine puta $C(k)$ pretpostavićemo, radi jednostavnosti, da je skip lista beskonačne dužine.

- Sa verovatnoćom $p = 1/2$ u čvor gde se sada nalazimo dospeli smo iz istog čvora sa nivoa za jedan više (drugim rečima, ako postoji viši nivo čvora, što se dešava sa verovatnoćom p, došli smo iz njega), a tome je prethodio put čija je očekivana dužina $C(k-1)$.
- Sa verovatnoćom $1 - p = 1/2$ u čvor gde se sada nalazimo dospeli smo iz čvora sa istog nivoa, čemu je prethodio put čija je očekivana dužina $C(k)$[2].

Prema tome, vrednosti $C(k)$ zadovoljavaju rekurentnu jednačinu:

$$C(k) = 1/2(1 + C(k)) + 1/2(1 + C(k-1))$$

Sređivanjem ovog izraza dobijamo:

$$C(k) = 2 + C(k-1)$$

[2]Primetimo da ovde vrednost $C(k)$ definišemo preko same te vrednosti. Ipak, ovo nije cirkularna definicija jer je naredna pojava vrednosti $C(k)$ uvek data sa manjom verovatnoćom, pa postupak konvergira.

odnosno očekivani broj koraka na svakom nivou je 2. Daljim raspisivanjem izraza za $C(k)$, pod pretpostavkom da je $C(0) = const$ dobijamo $C(k) = 2k$, gde je k redni broj nivoa sa koga pretraga kreće, računato odozgo.

S obzirom na to da skip lista nije beskonačna, u nekom momentu pri kretanju ulevo naići ćemo na glavu liste iz koje više nisu moguće kretnje ulevo već samo naviše. Ako sa $L(n) = \log_2 n$ označimo očekivani nivo skip liste dužine n, maksimalna vrednost za k je $L(n) - 1$. Dakle, očekivano vreme traženja je $C(k) = O(\log n)$. Povratak na najviši nivo liste nas ne mora direktno dovesti do glave skip liste, ali je očekivani broj preostalih koraka mali (jer se na tom nivou očekuje konstantan broj čvorova) te ne utiče na ukupnu složenost.

Poređenje skip listi i balansiranih uređenih drveta Iako u najgorem slučaju skip liste imaju loše performanse, dobra strana skip listi je to što ni jedan ulaz ne odgovara konzistentno najgorem slučaju (slično algoritmu brzog sortiranja kada se pivot bira na slučajan način). Štaviše, malo je verovatno da skip lista bude značajno nebalansirana. Stoga, skip lista ima svojstvo balansiranosti nalik uređenim binarnim drvetima izgrađenim slučajnim umetanjima elemenata, ali ne zahteva da umetanja budu slučajna. Ipak, napomenimo da se kod skip listi sa veoma malom verovatnoćom mogu desiti degenerisani slučajevi: da postane obična povezana lista ili pak da svaki čvor bude istog maksimalnog nivoa.

Skip liste ne zahtevaju da se uz čvorove liste čuvaju informacije o balansiranosti (kao kod balansiranih drveta), ali zato zahtevaju čuvanje dodatnih pokazivača – u proseku dva pokazivača po čvoru.

Za mnoge primene skip liste su prirodnija reprezentacija od drveta i vode jednostavnijim algoritmima. Dosta su jednostavnije i za implementaciju od balansiranih binarnih drveta. Takođe, prilikom operacija umetanja i brisanja sve izmene su lokalne, sa izuzetkom promene maksimalnog nivoa liste, te su pogodnije za paralelizaciju i u situacijama kada je potrebno omogućiti konkurentni pristup strukturi podataka. Npr. u crveno-crnom drvetu je nakon umetanja novog elementa potrebno rebalansirati potencijalno čitavo drvo i dok ažuriranje traje ne bi bio moguć pristup drugim elementima strukture. Nasuprot ovome, umetanje u skip listu utiče samo na susedne čvorove liste, te bi tokom ovih izmena velikom delu liste bio moguć paralelni pristup.

Kod skip liste, kao i kod klasične povezane liste, susedni elementi uglavnom neće biti u istom regionu u memoriji, što znači da skip lista ne može da iskoristi prednosti keširanja.

Primene skip listi Skip liste se koriste za distribuirane primene, npr. za implementaciju reda sa prioritetom koji dobro radi u višenitnom okruženju jer ne zahtevaju zaključavanje čitave strukture podataka dok se izvodi pisanje, već samo zaključavanje čvorova susednih čvoru nad kojim radi. One se mogu koristiti i za implementaciju struktura podataka potrebnih za grafovske algoritme, poput algoritama za određivanje najkraćih puteva. Mogu se koristiti i za indeksiranje u

bazama podataka, kao i za sortiranje velikih skupova podataka. Skip liste imaju primenu i u geografskim informacionim sistemima za indeksiranje i pretragu geografskih podataka, poput mapa i satelitskih slika. Koriste se i za efikasna statistička izračunavanja, npr. za računanje pokretne medijane (eng. running median).

Zadaci za vežbu

1. Kolika je očekivana složenost operacije pretraživanja elemenata u skip listi, a kolika složenost u najgorem slučaju?
2. Simulirati umetanje elementa sa vrednošću ključa 20 u skip listu sa slike 1 ako je nivo novog čvora 4. Na koje čvorove pokazuju elementi niza *prethodni*?
3. Simulirati brisanje elementa sa vrednošću ključa 25 iz skip liste sa slike 1. Na koje čvorove pokazuju elementi niza *prethodni*?
4. U kakvom su odnosu vrednosti ključeva čvorova na koje pokazuju pokazivači *prethodni*[1], *prethodni*[2], *prethodni*[3], ... pre umetanja elementa?
5. Neka skip lista sadrži n elemenata. Dokazati da je očekivani ukupan broj pokazivača koje lista koristi $O(n)$.
6. Koliko je unapređenje vremenske složenosti operacija umetanja i brisanja postignuto prelaskom sa jednostruko povezane liste na skip listu?
   a. sa $O(n)$ na $O(\log n)$
   b. sa $O(n)$ na $O(1)$
   c. sa $O(n)$ na $O(n^2)$
   d. nije postignuto nikakvo unapređenje
7. Pretpostavimo da imamo dve skip liste na raspolaganju: skip listu A koja sadrži m elemenata i skip listu B koja sadrži n elemenata. Opisati algoritam za spajanje ove dve liste u jedinstvenu skip listu koja sadrži $m+n$ elemenata. Ne pretpostavljati da su svi ključevi jedne liste manji od svih ključeva druge liste. Očekivana vremenska složenost algoritma treba da bude $O(m+n)$.
8. Neka je zadata skip lista od n elemenata, gde je n dovoljno veliko. Koliki je očekivani broj posećenih čvorova na nivou 4 prilikom operacije pretrage?
   a. nijedan
   b. $O(1)$
   c. $O(\log n)$
   d. $O(n/2^4)$
   e. svi

9. Koje su prednosti skip liste u odnosu na balansirana uređena binarna drveta?

10. Ukoliko bi skip lista imala samo dva nivoa, koliko bi elemenata liste trebalo da ima dva pokazivača i koji bi to elementi bili?