Skip List veri yapısında Seviye Optimizasyonu

[1]Mustafa AKSU, [2]Ali KARCI ve [1]Şaban YILMAZ
[*1]Kahramanmaraş Meslek Yüksek Okulu, Sütçü İmam Üniversitesi, Kahramanmaraş, Türkiye
[2]Mühendislik Fakültesi, Bilgisayar Mühendisliği Bölümü, İnönü Üniversitesi, Malatya, Türkiye

Özet

Skip list, bağlı listelere alternatif olarak Pugh tarafından ortaya atılan bir veri yapısıdır. Skip list veri yapısında bağlı listeler kullanılır. Düğümler, birbirine bağlı listeler halinde farklı seviyelere pramit şeklinde yerleştirilir. Böylece eleman arama, ekleme, silme gibi işlemler kolayca(O(lg N)) yapılabilir. Bununla birlikte, bu veri yapısında seviyelere düğüm ekleme işleminde olması gerekenden çok yüksek seviyeler üretilebilmektedir. Bu makalede rastgele seviye üretme problemi ele alınarak çözüm önerilmiştir. Skip list veri yapısındaki seviye üretme problemi toplam düğüm(eleman) sayısı(***nodecount***) bulunarak çözülmüştür. Böylece skip list için en ideal seviye ***level=log$_2$(nodecount)*** formülü ile oluşturulmuştur.

Anahtarlar: Skip list, rastgele seviye, seviye optimizasyonu, veri yapısı.

1. Giriş

Veri yapıları bilgisayar bilimlerinin bir çok alanında doğrudan veya dolaylı yollardan kullanılmaktadır. Bir çok problemin çözümünde değişik veri yapıları kullanılmaktadır. Zaman zaman mevcut veri yapılarının ihtiyaçlara cevap verememesinden veya işlem, zaman, donanım gibi kısıtlamalardan dolayı yeni veri yapılarına ihtiyaç duyulmuştur. Bazan dinamik , bazan statik bir yapıya ihtiyaç olduğundan farklı veri yapıları ortaya çıkmıştır.Bir problemin çözümü için geliştirilen algoritmanın verimliliği düşük olabilir yada algoritma yetersiz kalabilir. Bütün bunlar göz önünde bulundurulduğunda yeni veri yapıları ve algoritmalar ortaya çıkmaya devam edecektir.

Skip List PUGH[1,2,3] tarafından ortaya atılan dinamik bir veri yapısıdır. Skip list veri yapısında bağlı listeler kullanılır. Bağlı listeler pramit şeklinde farklı seviyelere yerleştirilir. Skip List veri yapısındaki herbir liste kendi üzerindeki listeleri kapsar. N elemanlı bir Skip list veri yapısında arama, ekleme, silme işlemleri için zaman karmaşıklığı O(lg N)' dir. Dizilerde ve bağlı listelerde bu işlemler O(N) zaman karmaşıklığına sahipken skip list veri yapısında O(lgN) olması büyük bir avantajdır.

Skip list veri yapısı algoritmalarının analizi ve iyileştirilmesi adına 1990'dan günümüze bazı çalışmalar [6,7,8,9,10,11,12,13,14] yapılmıştır.

Bütün bunların yanında skip list veri yapısı için geliştirilen algoritmalarda bazı problemler vardır. Bu problemler skip list veri yapısı işlemlerinde (arama, ekleme, silme) performansı düşürmektedir.

Bizim çalışmamızda aşağıdaki problemler ele alınıp çözüm önerileri sunulmuştur.
Bu problemler şunlardır:

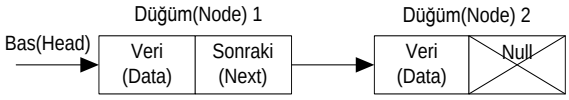
a. Yapıya bir düğüm eklerken eklenecek bu düğüm için seviye belirlemede kullanılan PUGH'un randomlevel algoritması tutarsız seviye üretmektedir.
b. Yapıya düğüm eklendikten sonra Skip list olması gereken seviyeden çok yüksek olabilmektedir.

2. Materyal ve Metod

Bilgisayar bilimlerinde listeler yararlı ve yaygın olarak kullanılan veri yapılarındandır. Programlama açısından liste, aralarında doğrusal ilişki olan veriler topluluğu olarak görülebilir. Veri yapılarında değişik biçimlerde listeler kullanılmakta ve üzerlerinde değişik işlemler yapılmaktadır [4]. Bağlı listeler, çift yönlü bağlı listeler, çevrimsel bağlı listeler, skip list bu veri yapılarından bazılarıdır.

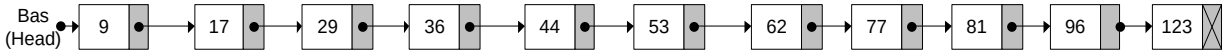
2.1. Bağlı Listeler

Bağlı listeler, bir grup düğümün(9, 17, 29, 36, 44, 53, 62, 77, 81, 96, 123) bir araya getirilip birbirlerine bağlanmasıyla oluşturulan bir veri yapısıdır. Bu düğümler sıralı bir şekilde birbirlerine bağlanırsa Şekil 2'deki yapı elde edilir. Bu sıralı yapı herhangi bir pozisyona düğüm ekleme, silme gibi işlemlerde etkin bir şekilde kullanılır. Bağlı listelere erişim için bir başlangıç düğümünü(head) ihtiyaç vardır. Her düğüm veri(ler) ve bir sonraki düğüme bağlantı(next) bileşenlerinden oluşur. Ayrıca bağlı listenin bittiğini ifade eden bir sonlandırıcıya(null) ihtiyaç vardır (Şekil 1).



Şekil 1. Bağlı Liste düğüm Yapısı

Bağlı listeler en basit ve en çok kullanılan veri yapılarındandır. Ayrıca diğer bağlı liste çeşitleri, yığıt(stack), kuyruk(queue), skip list gibi veri yapıları da temelde bağlı listelerden oluşur [5].



Şekil 2. Bağlı Liste

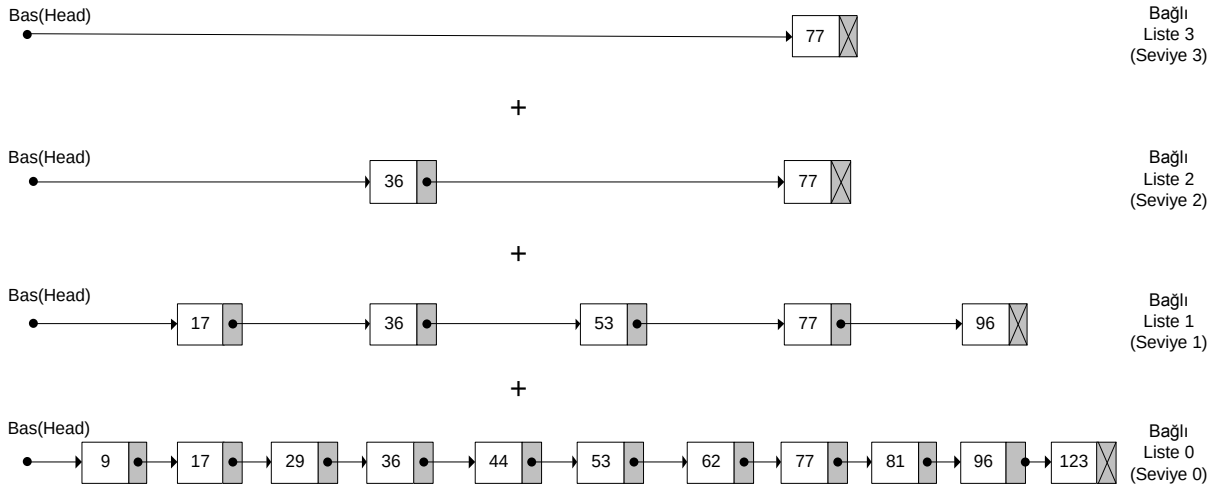
2.2. Skip listin Oluşturulması

Skip List veri yapısında, bağlı liste veri yapısında olduğu gibi listenin her bir elemanı düğüm olarak kabul edilir ve her bir eleman bir anahtar ve bir değer ile ifade edilir.

Skip list very yapısında bağlı listeler kullanılır ve bağlı liste elemanları sıralı olarak değişik levellere yerleştirilerek arama işlemlerinde kolaylık sağlanması amaçlanır. Bağlı listeler üst üste birbirinin fihristi olacak şekilde yerleştirilip birbirlerine bağlanırsa Skip List veri yapısı ortaya çıkar (Şekil 3, Şekil 4).

{9, 17, 29, 36, 44, 53, 62, 77, 81, 96, 123} düğümlerinden oluşan bir Skip Listin bağlı listelerden oluşturulması Şekil 3 ve Şekil 4' te görülmektedir. İlk olarak tüm düğümler Seviye 0'da yer alır ve sol taraftan başlanarak her 2^i. düğüm(i=0,..,MaxLevel(15/31)) atlanarak üste doğru her seviyeyi temsil eden işaretçiler oluşturulur.

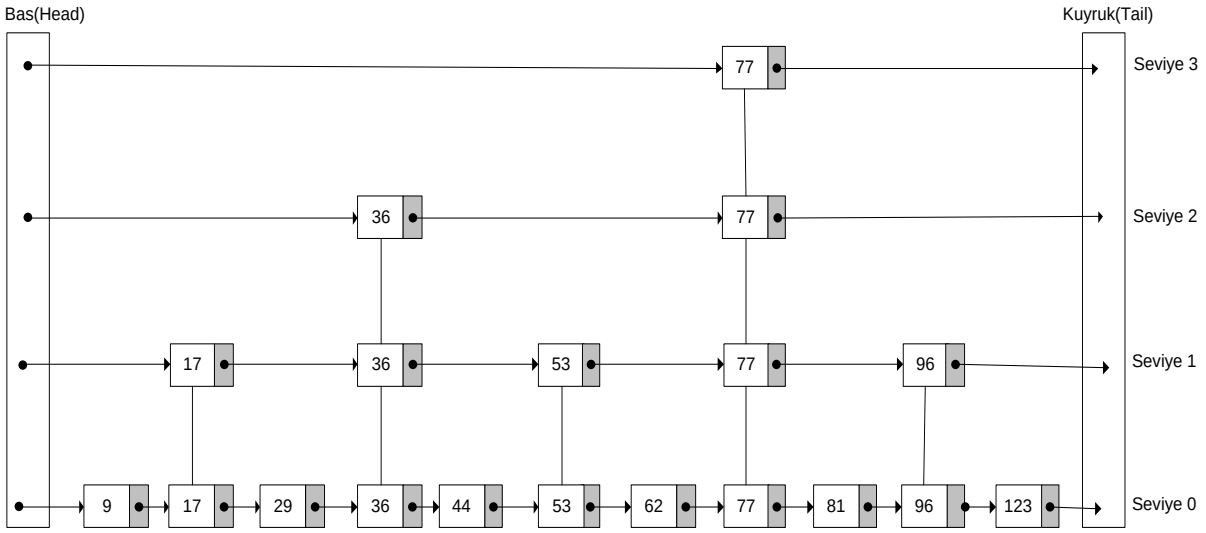
Bağlı liste$_0$ $\supseteq$ Bağlı liste$_1$ $\supseteq$ .....$\supseteq$ Bağlı liste$_l$. Bağlı liste$_0$, Skip List veri yapısında en alt seviyedeki bağlı liste olup tüm düğümleri kapsar. Alttan üste doğru her liste bir altındaki listenin fihristi şeklinde sıralanır.



Şekil 3. Bağlı listelerden Skip Listin oluşturulması

Bağlı listelerde yapılan işlemler O(N) karmaşıklığında yapılmakta iken skip list veri yapısı sayesinde bu karmaşıklık azaltılmıştır.

Skip list veri yapısı N tane sıralı düğümden oluşsun. Bağlı liste$_0$, bu N tane sıralı düğümlerin tamamından oluşur (Şekil 4-Seviye 0).



Şekil 4 - Skip List

Skip list veri yapısı ekleme, silme, arama işlemleri için kulllanılabilmektedir. Arama algoritmasında aranacak düğüm üst seviyeden başlanarak aşağı seviyelere doğru aranır. Ekleme işleminde önce eklenecek düğüm aranır bulunamamışsa rastgele belirlenen seviyeden itibaren eşleşen konuma yeni değer eklenir, işaretçiler ve listeler güncellenir. İşlem düğümün ekleneceği diğer seviyeler için tekrarlanır. Silme işleminde ise en üst seviyeden başlanarak alt seviyelere doğru arama gerçekleştirilir silinecek düğüm bulununca silinir, işaretçiler ve listeler güncellenir. Aynı işlemler düğümün olduğu diğer seviyeler için tekrarlanır.

Skip list veri yapısına düğüm ekleme, silme ve arama ile ilgili daha detaylı bilgi ve algoritmalar PUGH[1,2,3]'un makalesinde mevcuttur.

Bağlı ve sıralı listeler kullanıldığında skip list algoritmalarının(arama, ekleme, silme) zaman karmaşıklığı O(lg N) olmakta, bu da diğer algoritmalar ile karşılaştırıldığında önemli bir zaman farkı avantajı oluşturmaktadır.

PUGH'un Skip list veri yapısı için geliştirmiş olduğu algoritmalarda bazı problemler vardır. Bu problemler skip list işlemlerinde (arama, ekleme, silme) performansı düşürmektedir.

Bu problemlerden bazıları:

- Yapıya bir düğüm eklerken eklenecek bu düğüm için seviye belirlemede kullanılan randomlevel algoritması tutarsız seviye üretmektedir.
- Yapıya birçok düğüm eklendikten sonra Skip list olması gereken seviyeden çok yüksek olabilmektedir.

İlerleyen bölümlerde bu sorunlar ele alınıp bunlara çözüm önerileri üzerinde durulacaktır.

2.3. Skip List için seviye(level) oluşturma

Skip List veri yapısı inşa edilirken, düğümler rastgele belirlenen seviyelere yerleştirilir. PUGH'un geliştirdiği randomlevel algoritması (Algoritma 2) düğüm eklerken seviye(level) oluşturmak için 1..MaxLevel(15/31) arası rastgele bir değer üretir. Buda Skip List veri yapısında 2-3 düğüm varken bile level olarak yüksek değerler oluşturabilmektedir (Şekil 5.b). Bu istenmeyen bir durumdur. Halbuki 2 veya 3 düğüm varken seviye level=$\log_2$(N=3)'den 2 olmalıdır. Bu şekilde seviye hesaplamak için Skip Listteki toplam düğüm sayısının bilinmesi gerekir. Bunun için düğüm sayısını saklayacak bir değişken (nodecount) tanımlanıp skip list oluşturulurken başlangıç değeri 0 alınmalıdır. Bu değişken(nodecount) Skip list yapısına her düğüm eklemede 1 arttırılıp her düğüm silme işleminde 1 azaltılarak güncel tutulmalıdır. Bu bilgiler doğrultusunda randomlevel algoritması oluşturulmalıdır.

İdeal Skip listte N(nodecount) sayıda düğüm varsa bu yapı için olması gereken en yüksek seviye düğüm sayısının 2 tabanında logaritması alınarak bulunur. Skip listin o anki olması gereken seviyesi ***level = $\log_2$(N=nodecount)*** ile bulunur.

PUGH'un ***randomLevel*** algoritması şöyledir;

Algoritma 2

```
randomLevel()
  lvl := 1
   -- random() that returns
   --a random value in [0...1)
while random()< p and lvl < MaxLevel do
    lvl := lvl + 1
return lvl
```

Şekil 5'de {9,17,29} düğümlerinden oluşan iki ayrı Skip List yapısı görülmektedir. Şekil 5.a'da olması gereken ideal seviye level=$\log_2$(N=3)'den 2 olmaktadır. Şekil 5.b'de ise istenmeyen (0..15/31) seviyede gerçekleşen Skip list görülmektedir.

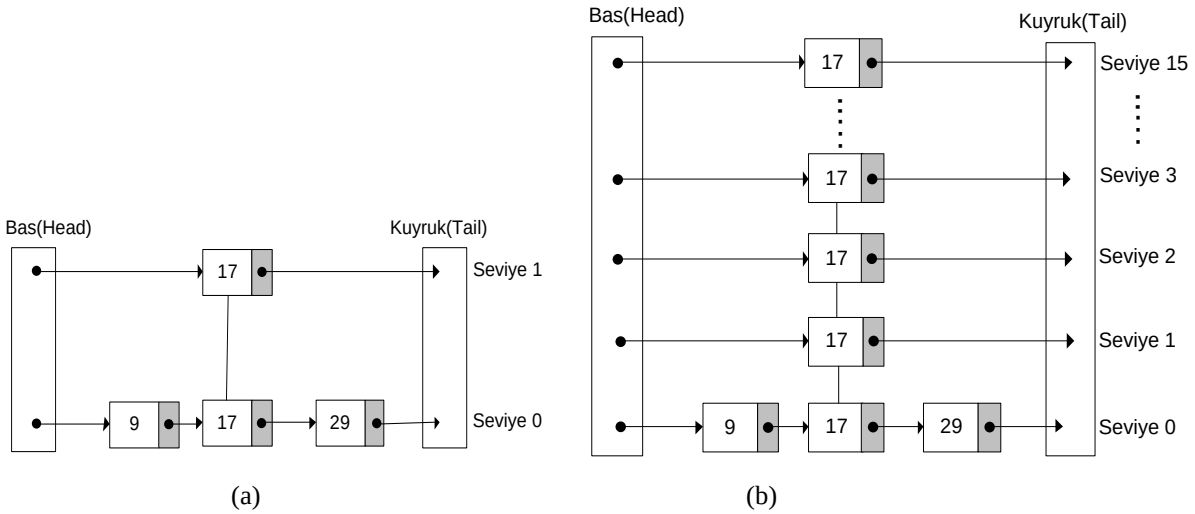


Figure 5. (a) İdeal seviyede Skip List, (b) Gerçekleşebilecek Skip List (PUGH)

Bizim geliştirmiş olduğumuz ***randomlevel*** algoritması (Algoritma 3) ise nodecount değerini esas alarak seviye(level) üreten bir algoritmadır.

Algoritma 3

```
randomLevel(nodecount)
      int  lvl,level;
  if ( nodecount=0 or head->level=0 ) then   {Skip list boşsa level=0 al}
      level := 0;
  else
     {
       lvl := log2(nodecount);   {nodecount değerinden level hesaplama}
     if  ( lvl ≠ head->level ) then
       level:=lvl
     else
       level := random() % (head->level+1);   { Mod alma işlemi }
     }
  return level;
```

Bizim algoritmamızda düğüm sayısına (nodecount) bağlı seviye üretildiği için çok yüksek seviyeler üretilmez. Yapı düzenli oluşacağından arama, ekleme, silme işlemlerinde zaman kaybı olmaz. *Yani* ***level = log$_2$(nodecount)*** ifadesinin sonucu olan ***level*** değeri skip listin o anki olması gereken ideal seviyesidir.

İlk olarak geliştirdiğimiz uygulama ile 17 düğümden oluşan bir skip list yapısı oluşturduk. Sonra **randomlevel** algoritmamızla rastgele seviyeler üretip bu seviyelere rastgele 16 yeni düğüm ekledik. Sonuç olarak bu 16 yeni düğümün seviyelere dağılımı Tablo 1'de görülmektedir.

Tablo 1. Rastgele eklenen 16 düğümün seviyelere dağılımı

Seviyeler(Levels)	0	1	2	3	4
Düğüm Sayısı	4	3	4	3	2

Aynı işlemleri 72 düğümden oluşan bir skip list yapısı için tekrarladık. Sonra randomlevel algoritmamızla rastgele seviyeler üreterek bu seviyelere rastgele 25 yeni düğüm ekledik. Sonuç olarak bu 25 yeni düğümün seviyelere dağılımı Tablo 2'de görülmektedir.

Tablo 2. Rastgele eklenen 25 düğümün seviyelere dağılımı

Seviyeler(Levels)	0	1	2	3	4	5	6
Düğüm Sayısı	4	3	4	2	4	5	3

Tablo 1 ve Tablo 2 incelendiğinde bizim algoritmamızla elde edilen seviyeler idealdir. Ayrıca rastgele üretilen seviyelere düğümlerin dağılımı da düzenlidir. Skip list için olması gereken ideal seviye korunmuştur. PUGH'un kendi ifadesine göre geliştirdiği olasılıksal **randomlevel** algoritması (Algoritma 2) 15 düğüm varken bile 14 seviyeli bir yapı oluşturabilmektedir. Bu kötü bir durumdur. Oysa 15 düğümlü bir skip list veri yapısı 4 seviyeden fazla olmamalıdır. Biz bu problemi düğüm sayısından(nodecount) seviye üreterek (Algoritma 3) çözdük.

2.4. Düğüm Ekleme

Ekleme algoritmasında üst seviyeden en alt seviyeye doğru eklenecek konum bulunduktan sonra ekleme gerçekleşir. Burada en önemli problem eklenecek düğümün seviyesidir. PUGH 1..15/31(MaxLevel) arası rastgele bir sayı üretip onu seviye alarak düğümü o seviyeden itibaren tüm alt seviyelere doğru eklemektedir. 1..15/31(MaxLevel) arası rastgele bir sayı üretilmesi sonucu skip list yapısında 2-3 düğüm varken bile seviye olarak yüksek değerler(Figure 5.b) oluşturabilmektedir. Bu istenmeyen bir durumdur. Halbuki 2 veya 3 düğüm varken seviye level=$\log_2$(N=3)'den 2 olmalıdır(Figure 5.a). Bunun için Skip listteki toplam düğüm sayısının bilinmesi gerekir. Bu bilgiler doğrultusunda ekleme algoritması düğüm sayısını güncel tutacak şekilde oluşturulmalıdır.

PUGH'un ***insert*** algoritmasına ***nodecount*** diye bir değişken eklenerek ***insert*** algoritması yeniden düzenlenmiştir (Algoritma 4). Algoritmanın bu halinde Skip list yapısına eleman eklenirken her düğüm eklemede ***nodecount*** değeri 1 artırılmaktadır. Bu sayede skip listteki toplam düğüm sayısı bulunmaktadır. Nodecount parametresinin o anki değeri skip listin $Seviye_0$ düzeyindeki toplam düğüm sayısını verir.

Algoritma 4

```
Insert(list, searchkey, newvalue, nodecount)
  local update[1..MaxLevel];
  temp := list→header;
   for i := list→level downto 1 do
      while temp→next[i]→key < searchkey do
         temp := temp →next[i]
      end while
      update[i] := temp
   end for
   temp := temp →next[1];
   if temp →key = searchkey then
      temp →value := newvalue
   else
      newlevel := randomlevel(nodecount);
      if newlevel > list→level then
          update[i] := list→header
          list→level := newlevel
      end if;
```

```
    temp:=MakeNode(newlevel,searchkey,newvalue);
     for i := 1 to newlevel do
        temp →next[i] := update[i]→next[i];
        update[i]→next[i] := temp
     end for
    nodecount:=nodecount+1;
  end if-else
  return nodecount
```

2.5. Düğüm Silme

Skip list veri yapısından bir düğümü silmek için önce arama algoritması kullanılarak silinecek düğüm bulunmalıdır. Düğüm bulunduktan sonra bulunduğu tüm seviyelerden silinir. İşaretçiler(pointer) güncellenir.

PUGH'un **Delete** algoritmasını skip list yapısındaki düğüm sayısını güncel tutacak şekilde düzenlemek gerekmektedir. Yani skip list yapısından düğüm silerken her düğüm silme işleminden sonra **nodecount** değeri 1 azaltılmalıdır. Bu sayede skip listteki toplam düğüm sayısı güncel tutulmaktadır. PUGH'un **Delete** algoritması bunlar dikkate alınarak yeniden düzenlenmiştir(Algoritma 5).

Algoritma 5

```
Delete(list,  searchkey, nodecount)
local update[1..MaxLevel];
temp:= list→header;
   for i := list→level downto 1 do
      while temp→next[i]→key < searchkey do
         temp := temp→next[i]
      end while;
      update[i] := temp
   end for;
 temp := temp→next[1];
  if temp→key = searchkey then
      for i := 1 to list→level do
         if update[i]→next[i] ≠ temp then
            break
         end if;
         update[i]→next[i] := temp→next[i]
      end for;
      free(temp);
      while list→level > 1 and
          list→header→next[list→level] = NIL do
         list→level := list→level – 1
      end while
    nodecount:=nodecount-1;
  end if;
  return nodecount
```

3. Tartışma ve Sonuç

Skip list veri yapısı, bağlı liste veri yapısındaki arama, ekleme, silme işlemlerinin O(N) olan zaman karmaşıklığını O(lg N) zaman karmaşıklığına indirmektedir. Bundan dolayı üzerinde durulması, çalışılması ve araştırılması gereken bir veri yapısıdır.

Skip List veri yapısı PUGH tarafından ortaya atılan ve bağlı listelerin kullanıldığı, arama işleminde çok hızlı bir veri yapısıdır. Buna karşılık düğüm ekleme, seviye oluşturma, silme işlemlerinde daha önce geliştirilen algoritmalarda bazı problemler vardır. Skip list veri yapısından iyi sonuçlar elde etmek için bu algoritmaların iyileştirilmesi gerekmektedir.

Bizim çalışmamızda öncelikle PUGH'un **Insert** ve **Delete** algoritmalarında iyileştirme yapılmıştır. Böylece skip list veri yapısındaki toplam düğüm sayısı(nodecount) bilinir hale gelmiştir.

Yine bu çalışmamızda, skip list veri yapısında tutarsız seviye üretme sorununa çözüm bulunmuştur. Bu sorun, skip list veri yapısındaki toplam düğüm sayısından ***level=log$_2$(nodecount)*** formülü ile ideal seviye üretilerek çözülmüştür. Böylece skip list oluşturulurken veya düğüm eklenirken çok yüksek seviyelerin oluşturulması sorunu ortadan kalkmıştır. Bizim yaklaşımımızla Skip list veri yapısındaki seviyeler ideal hale gelmiştir.

Kaynaklar

[1] Pugh W, Skip Lists: A Probabilistic Alternative to Balanced Trees, Communications of the ACM. 1990;33(6);668–676
[2] Pugh W, A Skip List Cookbook, Dept. of Computer Science, University of Maryland, College Park, 1989, UMIACS–TR–89–72.1
[3] Pugh W, Concurrent Maintenance of Skip Lists, Dept. of Computer Science, University of Maryland, College Park, 1989, TR–2222
[4] Aybars U, Data Structures Lecture Notes, Department of Computer Engineering, Ege University, 2005.
[5] Wikipedia, the free encyclopedia,
http://en.wikipedia.org/wiki/Linked_list#Linear_and_circular_lists. 2012
[6] Herlihy M, Lev Y, Luchangco V, Shavit N, A Simple Optimistic Skiplist Algorithm, SIROCCO 2007;124-138
[7] Colvin R, Groves L, Luchangco V, and Moir M, Formal verification of a lazy concurrent list-based set. In Proceedings of Computer-Aided Verification (Aug. 2006)
[8] Vyukov D., Concurrent Skip List, July 12, 2010
[9] Kirschenhofer P, Martinez C, Prodinger H, Analysis of an optimized search algorithm for skip lists, Theoretical Computer Science 1995;144;199-220
[10] Devroye L, A limit theory for random skip lists, Annals of Applied Probability, 1992;2(3);597–609.
[11] Kirschenhofer P, Prodinger H, The path length of random skip lists, Acta Informatica, 31 1994;8;775–792
[12] Papadakis T, Skip lists and probabilistic analysis of algorithms, Ph.D. Thesis, University of Waterloo, 1993; Tech. Report CS-93-28.
[13] Papadakis T, Munro J.I, Poblete P.V, Average search and update costs in skip lists, BIT 1992; 32;316–332.
[14] Poblete P.V, Munro J.I, Papadakis T, The binomial transform and the analysis of skip lists, Theoretical Computer Science 2006;352;136 – 158