**OPTİMAL BİNARY SEARCH TREE**

**Optimal binary search tree nedir?**

Optimal ikili arama ağacı, ağacın maliyetinin minimum olması için düğümlerin seviyelerde düzenlediği ikili bir arama ağacıdır. Optimal binary search tree, gelişmiş ağaçlardan birisidir.

Binary search tree'nin araştırmasının maliyetini nasıl azaltacağınıza odaklanır. En düşük yüksekliğe sahip olmayabilir.

Ağaca yerleştirilecek olan düğümler sıralı bir şekilde keys[0,1,…,n-1] dizisi içerisinde verilir. Bu düğümlerin tekrar etme sayısı ise freq[0,1,…,n-1] dizisi şeklinde verilir. Freq dizisi içerisindeki elemanlar keys dizisi içerisindeki elemanların arama sayısını ifade eder.

Tüm anahtarlardan ayrı ayrı ikili arama ağacı oluşturarak tüm aramalarda en düşük maliyetli olan bulunur.

Örnek

Giriş: keys[] = {10, 12}, freq[] = {34, 50}

İkili arama ağacı için iki seçenek var.

10 12

\ /

12 10

Ağaç I Ağaç II

10 ve 20 için arama frekansı sırasıyla 34 ve 50dir.

Ağaç I’in maliyeti: 34\*1 + 50\*2 = 134

Ağaç II’nin maliyeti: 50\*1 + 34\*2 = 118

**Optimal binary search tree’nin dinamik programlama yaklaşımıyla adım adım çalıştırılması**

* Her eleman için ayrı (keys ki…kj) optimal binary search tree bulma algoritması.
* Olabilecek her bir kök yani kr için r  i≤r≤j şartını sağla.
* ki,…,kr−1 için optimal alt ağaç yap.
* kr+1,…,kj için optimal alt ağaç yap.
* En iyi toplamı veren kökü seç.
* Formül: e(i,j)= ki…kj için optimal ağacın beklenen karşılaştırma sayısı

C:\Users\xman\Desktop\Ekran Alısdntısı.JPG

* C:\Users\xman\Desktop\sdsdf.JPG Bu formül ki…kj bir düğümün alt ağacıysa maliyeti arttırır.
* Algoritmayı aşağıdan yukarı çalıştır ve çözümü hatırla.

**Optimal binary search pseudo kod:**

Optimal alt ağacın en iyi maliyetini ve kök düğümünü kaydetmek için xi ile Xj tarafından oluşturulan optimal alt ağacın en iyi arama maliyeti için maliyet[n+1][n+1] için iki matrise ihtiyaç duyulmaktadır. Anahtar düğümler Xi ila Xj tarafından oluşturulan optimal alt ağacın kök düğümü için [n+1][n+1].

1. for low←n to n+1
2. for high←low+1 to 0
3. for r←low to high
4. rcost←cost[low][r]+cost[r+1][high]
5. For j←low to high rcost←rcost+p[j]
6. İf rcost<bestcost
7. bestcost←rcost
8. bestroot←root
9. Cost[low][high] ←bestcost
10. root[low][high] ←bestroot

Bu algoritmanın zaman karmaşıklığı;

C:\Users\xman\Desktop\sdfsfd.JPG