Hafta 4 – Ödev: Heap veri yapısı karmaşıklık analizi

Son teslim tarihi: 28 Nisan 2020, 23:59 *Bayram Çiçek - 160401002*

- *1. Heap yapısının diğer isimleri nelerdir?*

- Priority queues(kuyruk yapısı) - tree-based data structure(ağaç tabanlı yapı)

- *2. Heap veri yapısının iki özelliği nedir, nasıl avantaj sağlar?*

→ Structure Property'den:
complete tree → ağaç, soldan sağa olabildiğince dolu olmalıdır.
full complete tree → ağaç tamamen dolu olmalıdır.
(complete tree diziye atanır)

→ Heap-order Property'den:
Ebeveyn düğüm ile çocukları arasında bir kısıtlama vardır. Ebeveyn düğümler, çocuklarından ya büyük-eşit ya da küçük-eşit olmalıdır. (*Maxheap-Minheap*)

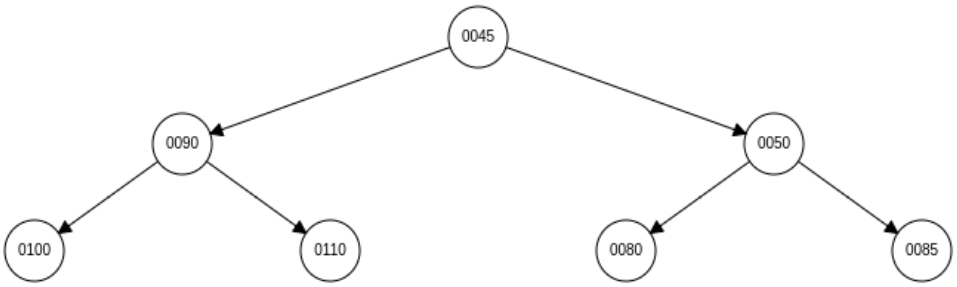
Dizideki herhangi bir elemanın ebeveyninin index'ini, index değerine bakarak(i/2) kolayca bulabiliriz. Ebeveynler her zaman çocuklarından daha küçük(veya büyük) olduğundan dizideki(ağaç yapısındaki) en küçük(veya büyük) elemanı *kök(root)* düğümde buluruz. Bu da bize kolaylık sağlar.

- *3. Maxheap yapısı ile Minheap yapısı arasında ne fark vardır?*

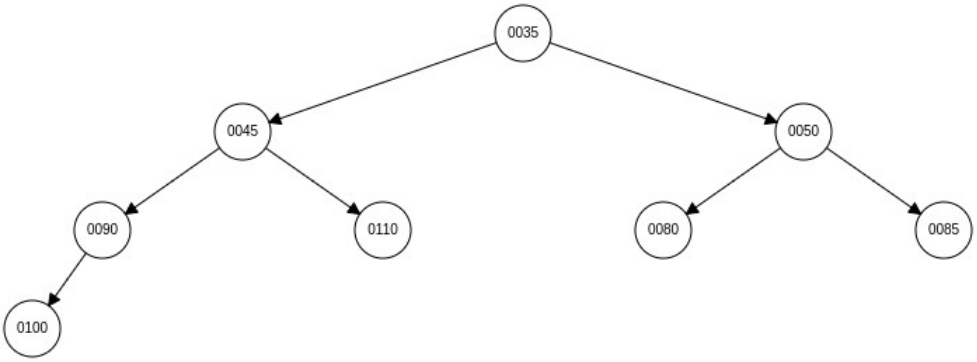
Maxheap yapısında her ebeveyn düğüm, çocuk düğümlerinden *büyük-eşit* olması gerekirken, *Minheap* yapısında ise *küçük-eşit* olması gerekir.

- *4. Video derste geçen linkler ile bir heap üzerinde insert ve remove işlemlerini uygulayıp karmaşıklığı yorumlayınız. Ödev raporuna ekleme/silme işlemi için şekil çizerek aktarınız.*

MinHeap yapısı doğrultusunda sırasıyla 100, 45, 50, 90, 110, 80, 85 eklenmiş olsun:

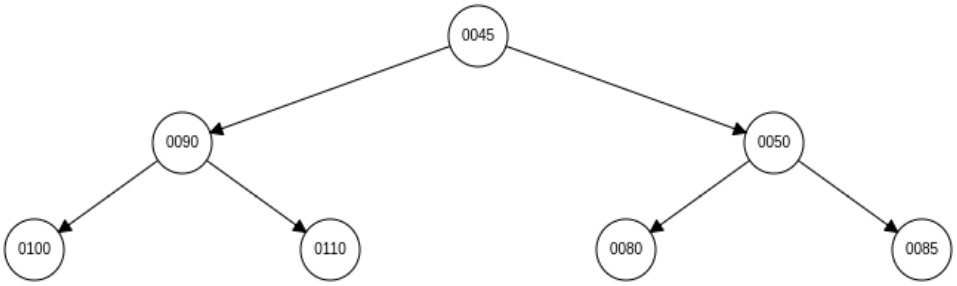


Ağaca 35 ekleyelim. Önce 100'ün soluna eklenecek sonra ebeveyni 100 ile karşılaştırılacaktır. 35, ebeveyninden küçük olduğundan yer değiştirecekler. Bu işlem bu şekilde devam edecek ve en sonunda 35, kök düğümünden de küçük olduğundan kök ile yer değiştirecek. Böylece 35 kök düğümü olacaktır:



Buradaki insertion işleminde eklenen elemanın en yukarı/root ile yer değiştirmesi wors case'e girer diyebiliriz.→ wors case O(logn) → logn tane karşılaştırma vardır.
Best case → ekleyeceğimiz eleman ağaçtaki tüm elemanlardan büyük ise soldan itibaren ilk boş bulduğu yere yerleşecek ve orada kalacaktır. Yani best case → O(1)

Remove işleminde ise en yukarıdaki 35'i kaldırırsak, önce en alt sağdaki 100, 35'in yerine atanacak. Böyle olunca ağacın yapısı bozulacağından 100, her seferinde çocuklarından en küçüğü ile yer değiştirecek. Önce 45 ile sonra 90 ile yer değiştirip duracak:



Toplam node sayısı n, yükseklik h ise → $n = (2^{h+1} - 1)$ olduğundan $h = \log(n)$ olur. Böylece hem insertion hem de remove işlemlerinde karmaşıklık yaklaşık O(logn) olacaktır.

- *5. Heap yapısı ile amortized cost arasında nasıl bir ilişki vardır?*

Heap yapısında karmaşıklığı O(1) veya O(logn) amaçlıyorsak yapı mutlaka dizi olmalıdır. Heap'in iç yapısının(internal structure) dizi olmasından dolayı amortized cost ile alakalıdır. Mesela dizi dolduğunda dizinin taşınması gerekir. Bu yüzden N operasyonda mutlaka en az 1 defa worst case'e girer. Bu yüzden Heap yapısının amortized cost ile ilişkisi vardır.