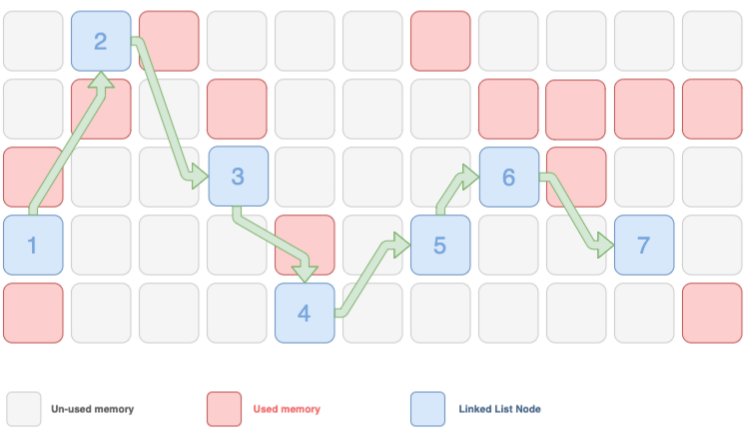
**Linked List (Bağlı Listeler)**

**Ahmet kürkçü 200312027**

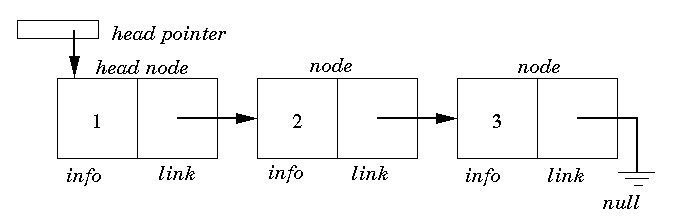
**LINKED LIST KAVRAMI**

Linked list, yazılım dünyasında önemli yeri olan veri yapılarından biridir. Temel mantık olarak, linked list’te her bir eleman(node) tutması gereken dataya ilaveten kendinden bir sonra gelecek olan node’un adresini tutar. Dolayısıyla her linked list node’u kendisinden bir sonraki node’un yerini bilir. Bu da linked listlerin bellekte sıralanış şekli açısından bize esneklik sağlar. Örneğin, bir array tanımladığınızda arrayın boyutunu başlangıçta ***int[] myArray = new int[10];*** şeklinde statik olarak vermeniz gerekir. Bu da size bellekte sıralı olarak arka arkaya ayrılmış 10 intlik bir alan rezerve etmek etmektir. Kullansanız da kullanmasanız da bu alan sizin için ayrılır. Hatta bellek alanları ardışık rezerve edildiği için pointer ile ilk index adresini alıp diğer elemanlara, ilk adrese bellekte 4er bytelık değerler ekleyerek ulaşabilirsiniz de. Ancak linked list’te böyle bir zorunluluk yok. Linked listin her node’u bir diğerinin adresini bildiğinden, node’lar bellekte bambaşka boş buldukları alanlara yerleşebilirler. Dolayısıyla linked listin eleman sayısı da dinamik olarak artabilir. Madem öyle linked list’in ortalarındaki elemanlarına nasıl ulaşıyoruz gibi bir soru kafanıza takılabilir. 1. elemandan başlayıp traverse ederek.



**2. LINKED LIST’IN YAPISI**

## **2.1 Node kavramı**

Yazının başında da bahsettiğim gibi, linked listin her bir elemanına node denir. Linked list isminden de anlaşılacağı üzere birbirine bağlı node’lardan oluşur. Node’lar temelde (simple one way node için) iki bilgiyi tutar. Bunlar;**1.** Tutulacak olan veri -> Data  
 2. Kendinden bir sonraki node’ın adresi -> Next

Genelde bizim linked liste dair tek bildiğimiz şey ilk elemanın yani head’in adresi ve uygulamaya göre tailin yani son node’un adresidir. Bazı uygulamalarda son nodeun adresinin tail olarak bir değişkende tutulmadığını görebilirsiniz. Böyle durumlarda son node’a ulaşmanın maliyeti O(n) olacaktır. Tutulduğu durumlarda ise O (1). Diğer elemanların yerlerini ise hiç bir şekilde bilemeyiz. E o zaman diğer elemanlara nasıl ulaşacağız. Tabiki head üzerinden… Aynı kervan saraylarda dolaşırcasına, adresini bildiğimiz ilk node’u alır, onun nexti üzerinden diğer node’a oradan diğer node ilerleyerek aradığımız elemanı buluruz. Şanslı isek aradığımız eleman ilk node’da çıkar ve linked list için search işlemi best case’de O (1) olurken, en kötü ihtimalle aradığımız eleman listede olmaz ve worst case’imiz O (n) olur.

ilk eleman : head  
İkinci eleman: head.next  
Üçüncü eleman: head.next.next……

Son eleman (tail) : head.next.next.next… = tail –> tail.next => null

Şeklinde tüm elemanlar üzerinde dolaşabiliriz.

Peki linkedin list kullanmanın avantajlarını nerede görürüz.

1. Dinamic memory yerleşme yapısı sayesinde daha az memory israfı.
2. Dinamik boyutlu olarak kullanılabilmesi.
3. Başa ve sona ekleme/çıkarma işlemlerinin O (1) komplexitysi ile yapılabilmesi.

Peki linked list kullanmanın dezavantajları nelerdir.

1. Random access’e izin vermemesi.
2. Her bir node’un datanın dışında, kendinden bir sonra gelecek node’un adresini de tutmasının getirdiği extra memory yükü.

**3. LINKED LIST IMPLEMENTASYONU**

Linked list farklı şekillerde implemente edilebilir. Genelde kullandığımız dinamik bellek yapısında olan, her bir nodeun bellekte boş bulunduğu yere yerleşmesi şeklinde implemente edilebileceği gibi, dizi kullanılarak da linkedlist implemente edilebilir. Dizi kullanılırsa, tüm kayıtlar için ardışık bellek alanı kullanılır. Ancak normal dizi uygulamalarında olduğu gibi (i+1). Kayıt hemen i. kayıttan sonra gelmeyebilir. Kayıtlar arasında indis değişkenleri üzerinden bağlantı olacaktır.Yani sonuç olarak linked list’i

* Dinamik memory ile
* Dizi ile implementasyonu bulunmakta olup aşağıdaki gibi farklı linked list tipleri mevcuttur.
* ***Tek yönlü linked list***
* ***Double linked list :***Linked list üzerinde headten başlanıp tail’e kadar gidilebildiği gibi geriye doğru da travers edilebilir. Bu linked listin her bir nodeunun ilerisindeki nodeun yani nextinin adresini tutmasının yanında gerisindeki nodeun da adresini tutmasıyla sağlanır. Bu da her node için extra bir adreslik bellek maliyeti getirir.
* ***Circular linked list :*** Bu tip linked listin üzerinde traverse yaparken baştan sona gidildiğinde tekrardan başa dönülebilen dairesel bir yapıdadır.
* ***Hybrid linked list :*** Farklı tip linked listlerin bir araya getirilmesi ile oluşturulmuş linked list tipidir.

***3.1. Linked List Operasyonları***

Linked list veri yapısında bazı temel operasyonlar söz konusudur. Bunlar;

– Listenin başına eleman ekleme (head’a)  
– Listenin başından eleman silme (head’ten)  
– Listenin sonuna eleman ekleme (tail’a)  
– Listenin sonundan eleman silme (tail’dan)  
– Listeleme  
– Arama

***– Listenin başına eleman ekleme (head kısmına)***

Eski head normal bir ara eleman olur. Yeni eklenen ise head olur.

– Listenin başında eleman silme (head’i silme) : eski head silinir. Onun bir önündeki head olur.

***– Listenin başına eleman silme***

Böyle bir durumda listenin başındaki head nodeunun liste ile olan bağlantısı silinir. Yani nextine null atanır. next’de yeni head olarak atanır.