**Tires**

**Tires** janë struktur e të dhënave që përdoret për search. Ekzistojn dy lloje të tries **R-way Tries** dhe versioni i vçantë i saj që quhet **Ternary search tries** (TST). Parim i punës në tries është impelementimi i Symbol Tables me të dhëna të tipit String.

Symbol Tables API

StringST() -Krijon nje Symbol table te zbrazet

void put(String key, Value val) -Metoda për shtim të elementet ne ST

value get(String key) -Kthen Value që i shoqërohet key të cilin e kërkojmë

void delete(String key) -E bën null Value të çelsit të cilin dëshirojm ta fshijmë

**Metodat më të rëndësishme të Tires :** Metoda **put**(String key, Value val) kjo metod mundëson të shtojmë elemente në tries, si parametra pranon çelsin që zakonisht është String , dhe value i cili mund të jet vler e qfar do tipi por ne e përdorim value të tipit int.Metoda **get**(String key) kjo metod përoret për të gjetur(marrur) Stringje nga tries , si parametër pranon çelsin(String-un) dhe nëse e gjen e kthen vlerën (value) të shoqëruar me të .Metoda **delete**(String key) kjo metod përdoret për të fshir elemente nga tires , si parametër pranon çelsin zakonisht String, dhe nëse e gjen at çelës vlerën e cila është e shoqëruar me at çels e kthen në null.

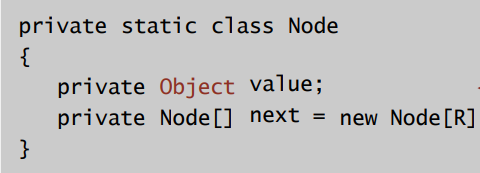
Tires në memorije paraqiten si një grumbull i Stringjeve ose ngjyrse koleksione i Stringjeve. Kleksion nga i cili ne mund të marrin Stingje të caktuara në qoftëse ato gjinden aty.

Tries si fjal rrjedh nga retrivel që do të thotë kthim i diqkaje.

Një ndër aplikimet e tries në jetën reale është rasti i contact lista të telefonit.

**R-way Tries :** Karakteret ruhen në nyje(nodes) ,secili node ka nga **R**(Radix) fëmijë pra nga një node për gjdo karakter të mundshëm. Fëmijët e nodes by default kanë vlerë null e cila më pas mbishkruhet pas shtimit të karaktereve të caktuara në tire. Gjdo Trie ka një **root** , i cili as nuk ka vlerë e as nuk ka çelës, rooti në memorie paraqitet si një flag për të treguar se nga ku të filloj kërkimi nëpër **Tire.** Edhe rooti i ka **R** fëmijë.

Janë dy koncepte kryesore që lidhen direkt me metodën **get** : **1**.Kur përdorim metodën get dhe nëse kjo metod arrin të na kthej vlerën e Stringut të cilin ja kemi dhënë si parametër e quajm **Search Hit. 2**.Në rast se metoda get na ka thyer një vler **null** themi se kemi të bëjm me **Search Miss,** kemi dy raste të search miss **:**  1.Kur e gjen Stringun por aj string e ka value null , dhe 2.Kur nuk e gjen String-un fare.

**Node** është një klas private statike e cila përmban një objekt të qfardoshëm dhe një array me nodes të gjatësisë **R**. 

Ekzistojnë tri metoda kryesore që përdoren për të krahasuar Stringje : compareTo, equals dhe charAt. Tries e përdorin metodën **charAt.**

**R-Way Tries Performanca** : Për Search Hit kompleksiteti kohor është **N** ndërsa për Search Miss është më i shpejt kjo për shkak se gjasat të mos përputhen Stringjet janë më të mëdha, gjithashtu kjo mos përputhje mund të ndodhë që në karakterin e par , mirpo zakonish kompleksiteti kohor i search Miss është **N/2** sublinar dhe max mund të jet **R**. Për sa i përket hapsirës memoruese R-way Tires nuk janë të preferueshme për shkak të asaj që përmbajn një numër shumë të madh të null nodes.

Për të bërë delete përdorim metodën **delete** siç u përmend edhe më lart , pra duke i vendosur vlerën (value) të Stringut në null .Nëse në tries hasim në një node (link) i cili ka vlerën null dhe gjithashtu nivele më poshtë ka fëmij të cilët deri tek i fundit e kan vlerën null , atëher kto nodes duhet fshirë , kët pun zakonisht e bën garbage collector në java.

**Ternary Search Tires (TST) :** Është përafërsisht i njëjti me R-wat Tries por kanë dy dallime esenciale : **1.**Secili Node në TST ka vetëm nga tre fëmijë për dallim nga r-way ku gjdo node kishte nga R fëmijë. **2.**TST për dallim nga R-way nuk ka **root**.

Gjdo Node me fëmijët e majt dhe të djathtë paraqesin një index në TST , ndërsa fëmijët e mesit e rrisin kët index për një.

TST i menjanon në mënyr të madhe null linkat, kjo bën që në aspektin e memories të jen më të mira se r-way , ndërsa në aspektin e kompleksitetit kohor janë njësoj.

**TST vs Hashing :** TST janë më të shpejta sesa hashing e sidomo ne search miss, TST janë më fleksibile se R&B Trees, TST për dallim nga Hashing i rradhit elementet përshkak se implementohen me ST, hashing janë më të sigurta sesa TST.