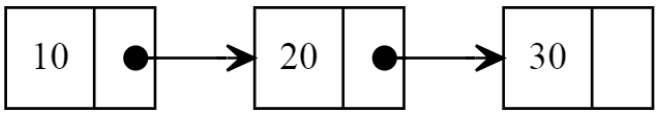
Roger Leonardo - 2440030071

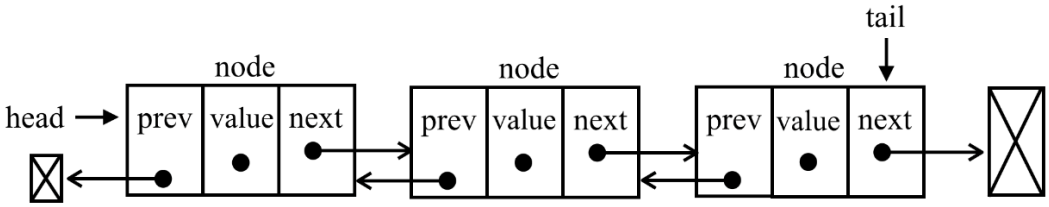
Linked List

1) Single linked list : setiap node mempunyai data dan pointer yang terus menuju ke node yang berikutnya

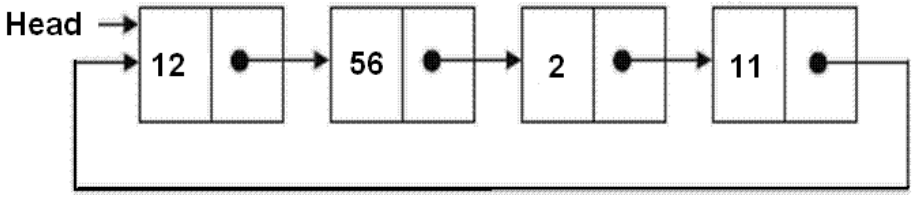
HEAD **NULL**



Double linked list : Setiap node mempunyai data dan pointer ke belakang (previous) maupun depan (next)



Circular linked list : Node terakhir dapat kembali lagi ke node awal



2)

Array	Linked list
Setiap elemen hanya berisi data saja.	Setiap elemen terdiri dari 2 bagian, data dan pointer address.
Elemen dicari berdasarkan indeks	Elemen dicari mulai dari head sampai elemen yang dicari ditemukan
Memori yang dibutuhkan lebih sedikit	Memori yang dibutuhkan lebih banyak
Ukuran dispesifikasi ketika melakukan deklarasi	Ukuran tidak perlu dispesifikasi
Penambahan dan pengurangan elemen kurang efektif	Penambahan dan pengurangan elemen lebih efektif
Memiliki ukuran yang pasti	Ukuran lebih dinamis/fleksibel
Elemen disimpan secara berurut	Elemen disimpan secara acak

3) Floyd's algorithm dapat digunakan untuk mengecek apakah ada looping di linked list atau tidak dan dilakukan dengan cara traverse linked list menggunakan dua pointer, yakni slow yang

bergerak satu demi satu langkah dan fast yang bergerak setiap dua Langkah, Ketika mereka bertemu maka ditemukan loop, jika tidak bertemu maka tidak ada loop
Pseudocode:
1. Inisialisasi dua pointer (fast & slow)
2. Terus loop selama tidak ada null
3. Atur slow ke node selanjutnya
4. Atur fast ke dua node selanjutnya
5. Jika kedua pointer bertemu, atur pointer slow ke head
6. Kedua pointer bergerak satu demi satu langkah sampai mereka bertemu lagi
7. Ketika sudah bertemu, maka kembalikan ke tempat awal mereka bertemu
8. Selain itu, jika bertemu null, return null

Stack & Queue

1)

Stack	Queue
Berdasarkan konsep LIFO dimana elemen yang dimasukkan terakhir dikeluarkan paling awal	Berdasarkan konsep FIFO dimana elemen yang dimasukkan pertama kali dikeluarkan paling awal
Insertion dan Deletion selalu dilakukan dari top (bagian terakhir dari list stack)	Insertion dilakukan dari rear(belakang) dan Deletion dilakukan dari front(depan)
Insertion disebut push dan Deletion disebut pop	Insertion disebut enqueue dan Deletion disebut dequeue
Memiliki 1 pointer	Memiliki 2 pointer
Implementasi lebih sederhana	Implementasi lebih kompleks

2)

Infix	Prefix	Postfix
A + B	+ A B	A B +
A + B * C	+ A * B C	A B C * +

Infix -> operasi matematika dengan susunan operator dan operand pada umumnya
Prefix-> operator akan melakukan operasi dengan 2 angka di belakangnya (kanan)
Postfix-> operator akan melakukan operasi dengan 2 angka di depannya (kiri)

Infix dalam stack

- Scan string dari kiri ke kanan dan mencari precedence operator tertinggi, kemudian melakukan operator melakukan operasi antara operator dengan kedua operand diantara operator tersebut

Prefix dalam stack

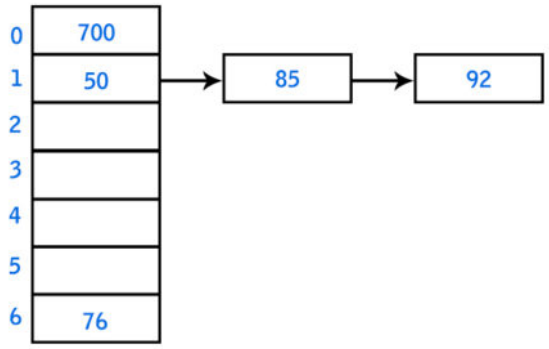
- Scan string dari kanan ke kiri, jika operand push ke dalam stack, jika operator, pop dua kali dua operand sebelumnya dan push hasil dari operasi operator pada dua operand yang telah dipop

Postfix dalam stack

- Scan string dari kiri ke kanan, jika operand push kedalam stack, jika operator, pop dua kali dua operand sebelumnya dan push hasil dari operasi operator pada dua operand yang telah dipop

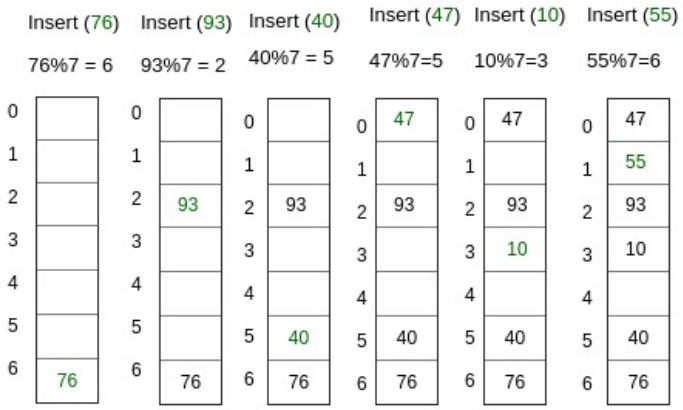
Hashing & Hash Tables

1) - Hash table adalah data structure yang digunakan untuk menyimpan pasangan kunci/nilai yang nantinya akan dimetakan oleh hash function.
   - Hash function digunakan untuk memetakan kunci-kunci besar ataupun bukan bilangan bulat ke indeks-indeks dalam range kecil
   - Collision adalah kondisi dimana terdapat lebih dari satu kunci yang menempati slot address yang sama
2) – Chaining



Contohnya linked list 1 memilii 3 slot(50,85,92), Ketika slot pertama sudah terisi, elemen berikutnya dapat dipindahkan ke slot berikutnya

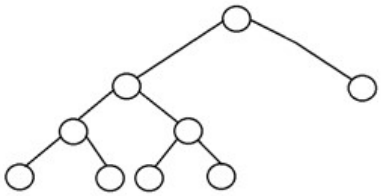
- Linear Probing: dilakukan dengan cara looping dari awal sampai menemukan slot yang kosong Ketika slot awal telah diisi oleh elemen lain



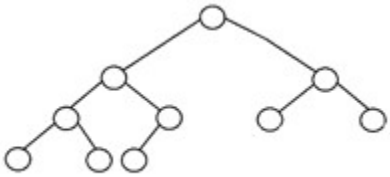
Dalam kasus diatas, hasil dari 40%7 adalah 5 sehingga ditempatkan di linked list ke 5, selanjutnya hasil dari 47%7 adalah 5 sehingga seharusnya ditempatkan di linked list ke 5 juga, tetapi karena linked list ke 5 telah terisi maka akan dicari slot berikutnya yang kosong untuk menempatkan 47

Binary Search Tree

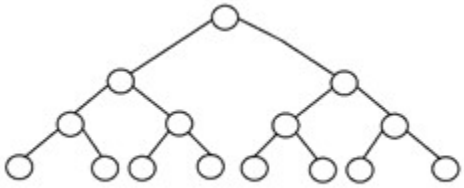
1) - Full Binary Tree: Jenis Binary tree yang mempunyai 0 atau 2 anak



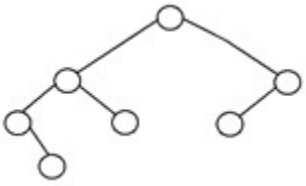
- Complete Binary Tree : suatu pohon biner yang kedalamannya sebesar n atau n-1 untuk beberapa n. Dan dalam penempatan simpulnya diutamakan yang sebelah kiri yang terpenuhi.



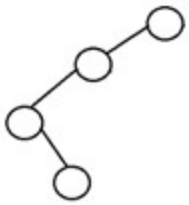
- Perfect Binary Tree : jika semua node internal memiliki tepat 2 anak, dan setiap node leaf/eksternal berada di level yang sama.



- Balanced Binary Tree: jika tinggi tree adalah O(logN) dimana N adalah jumlah node, perbedaan tinggi subtree kiri dan kanan maksimal 1 level



- Degenerate Binary Tree : jika setiap node internal memiliki tepat 1 anak dimana hal ini mirip seperti linked list



2) **Insertion 24->**
   - Cek apakah 24 lebih kecil atau lebih besar dari 27, dan ternyata lebih kecil sehingga menuju ke kiri
   - Cek apakah 24 lebih kecil atau lebih besar dari 14, dan ternyata lebih besar sehingga menuju ke kanan
   - Cek apakah 24 lebih kecil atau lebih besar dari 19, dan ternyata lebih besar sehingga menuju ke kanan

   Insertion 18->
   - Cek apakah 18 lebih kecil atau lebih besar dari 27, dan ternyata lebih kecil sehingga menuju ke kiri
   - Cek apakah 18 lebih kecil atau lebih besar dari 14, dan ternyata lebih besar sehingga menuju ke kanan
   - Cek apakah 18 lebih kecil atau lebih besar dari 19, dan ternyata lebih kecil sehingga menuju ke kiri

   Insertion 55->
   - Cek apakah 55 lebih kecil atau lebih besar dari 27, dan ternyata lebih besar sehingga menuju ke kanan
   - Cek apakah 55 lebih kecil atau lebih besar dari 35, dan ternyata lebih besar sehingga menuju ke kanan
   - Cek apakah 55 lebih kecil atau lebih besar dari 42, dan ternyata lebih besar sehingga menuju ke kanan

3) **Deletion 27->**
   - Ketika 27 dihapus, temukan angka terbesar dari subtree kiri
   - Ketika ditemukan, bawa/pindahkan ke atas sehingga angka tersebut menjadi main node yang baru

   Deletion 35->

   - Cek apakah 35 lebih kecil atau lebih besar dari 27, dan ternyata lebih besar sehingga menuju ke kanan
   - Ketika 35 ditemukan, kemudian dihapus
   - Setelah dihapus, 31 akan naik menggantikan 35 dan menjadi parent dari 42(diposisi kanan)

   Deletion 42 ->

- Cek apakah 42 lebih kecil atau lebih besar dari 27, dan ternyata lebih besar sehingga menuju ke kanan
- Cek apakah 42 lebih kecil atau lebih besar dari 35, dan ternyata lebih besar sehingga menuju ke kanan
- Ketika 42 ditemukan, kemudian dihapus
- Setelah dihapus, anak dari 35 berkurang menjadi 1 (31 di posisi kiri)