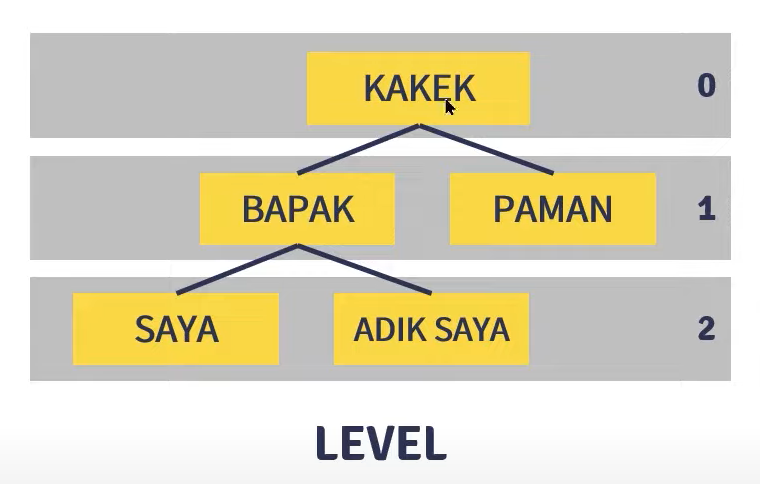
Tree

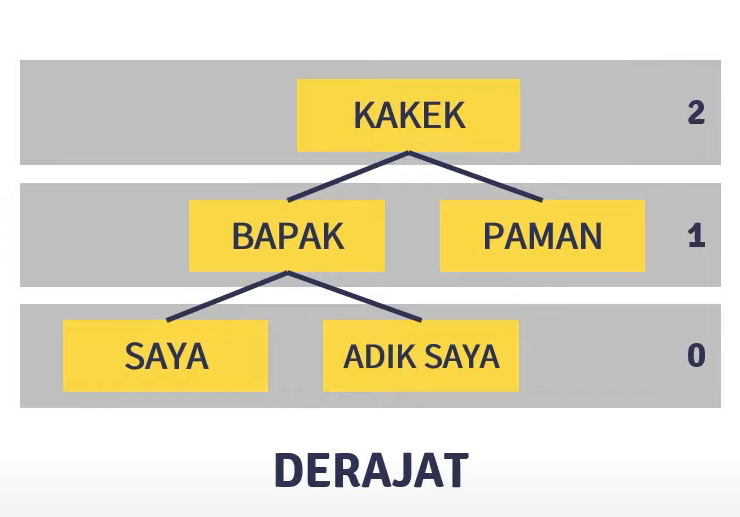
Konsep Tree

* Struktur data yang terdiri dari akar dan simpul-simpul yang berada dibawahnya
* Struktur data yang menunjukan hubungan bertingkat(memiliki hirarki)
* Merupakan struktur data yang tidak linier yang digunakan untuk mempresentasikan data yang bersifat hirarki(urutan/tingkat/kedudukan) antar elemn-elemennya

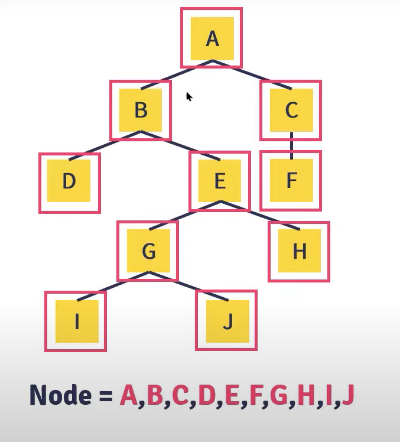
Level dan Derajat Tree



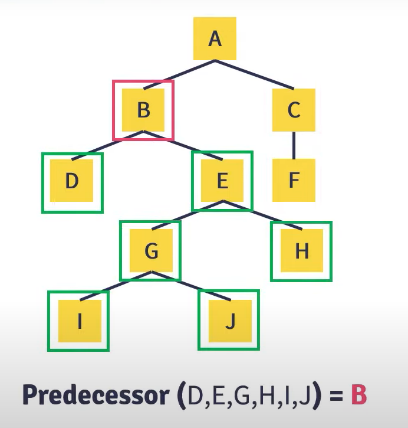
* Level dibaca dari data yang pertama kali ada atau hadir
* Kalau level baca nya ada atas



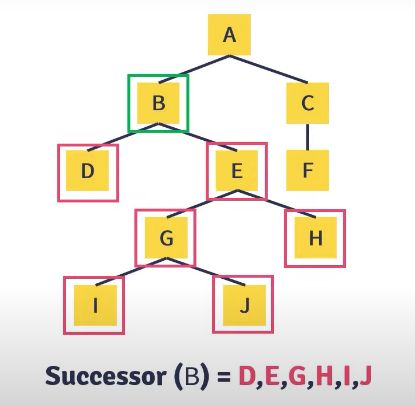
* Derajat dibaca dari bawah ini menunjukan bahwa yang diatas memiliki derajat nya lebih tinggi

Istilah dan Hubungan Kompenen Tree

**Node (Simpul)** : adalah simpul dari masing masing data dari suatu tree

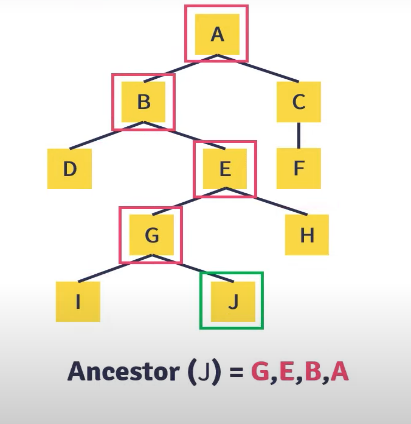
**Predecessor (Pendahulu)** : adalah node yang berada diatas node tertentu

Seperti contoh di samping saat kita tanya Predecessor dari (D,E,G,H,I,J) adalah B

**Successor (Penerus) dan Subtree** : adalah node yang berada dibawah node tertentu

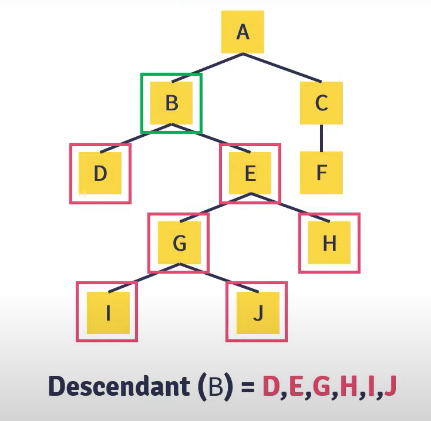
Kebalikannya dari **Prodecessor**

Misalnya kita tanya **Successor** dari B adalah D,E,G,H,I,J

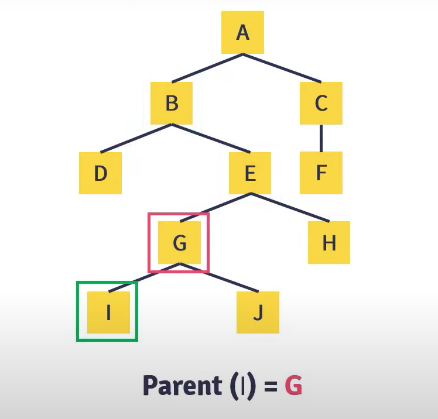


**Ancestor (Leluhur)** : Seluruh node yang terletak sebelum node tertentu dan terletak pada jalur yang sama

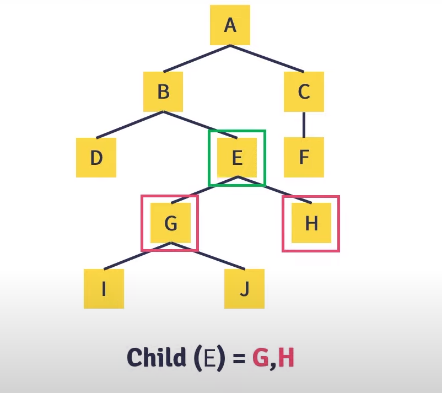
Misalnya ditanyakan **Ancestor** dari J adalah G,E,B,A

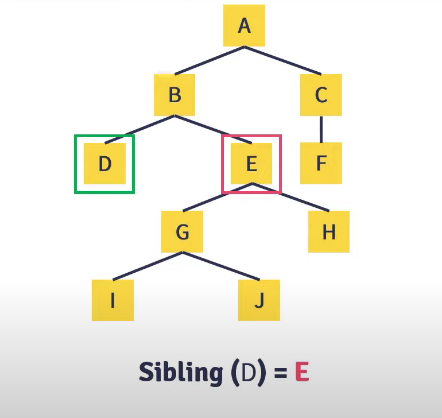
**Descendant ( Keturunan )** : Seluruh node yang terletak setelah node tertentu dan terletak pada jalur yang sama

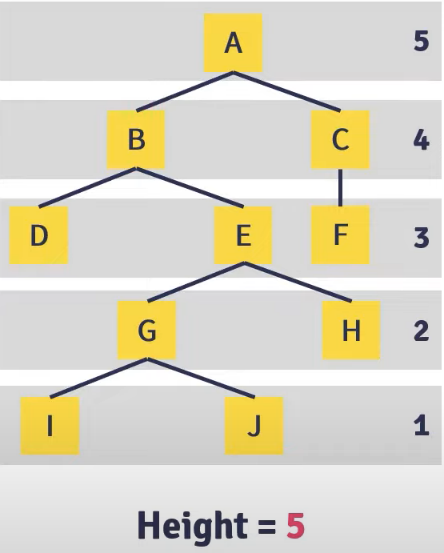
**Descendant** ini sama dengan **Successor atau subtree** tidak ada beda nya yang membedakan hanya istilahnya saja

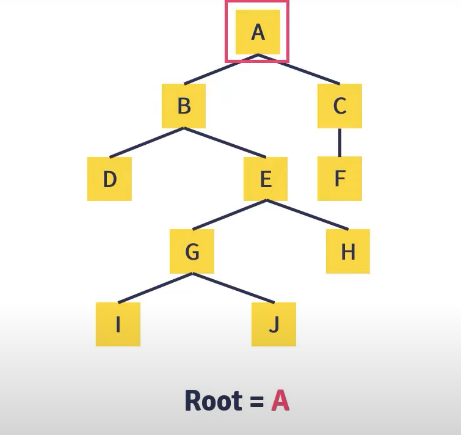
**Parent ( Orang Tua )** : Predecessor (Pendahulu) satu level diatas suatu node

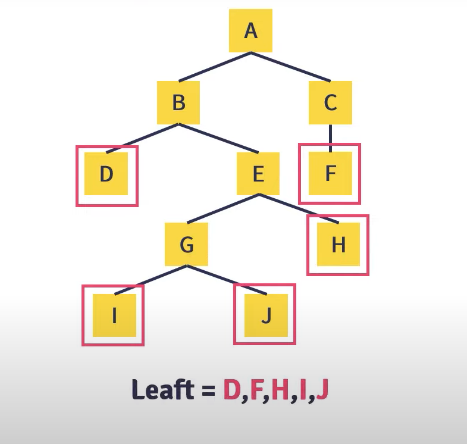
Misal ada yang tanya paret dari I adalah G

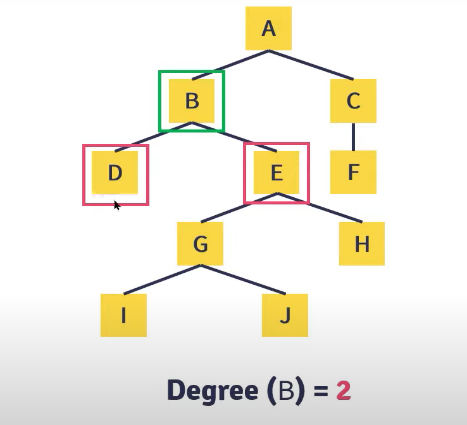
**Child** : adalah successor (penerus ) satu level dibawah suatu node

**Sibling (Saudara) :**  adalah node -node yang memiliki parent yang sama

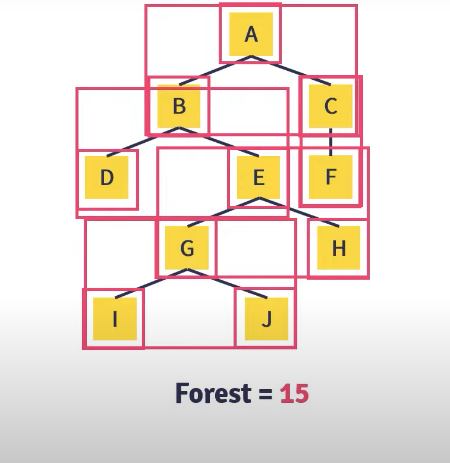
**Height :** Banyak nya tingkatan dalam suatu tree

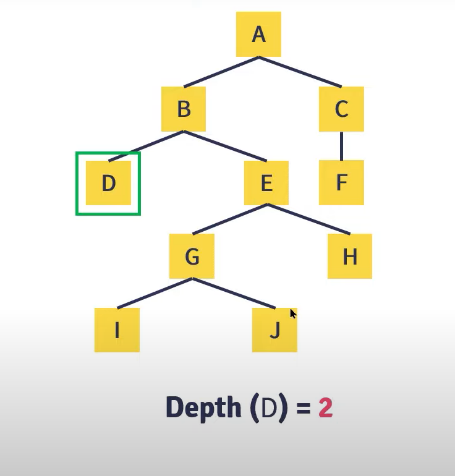
**Root(akar):**Adalah node khusu yang tidak memiliki predecessor(pendahulu)

**Leaf (daun)**: adalah node dalam tree yang tidak memiliki successor (penerus)



**Degree :**  adalah banyaknya child(anak) dalam suatu node

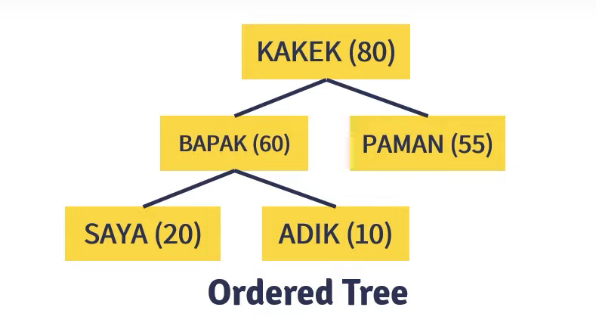
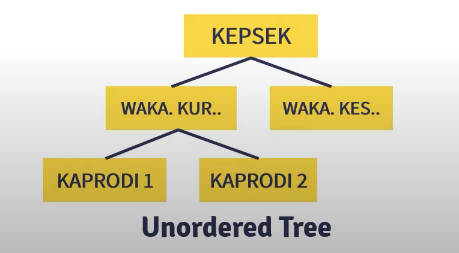
**Forest(Hutan) :**adlah kumpulan dari tree

**Depth (Kedalaman):** hasil tingkat node meksimum dikurangi satu atau lebih gampangnya level dari node **X**

Definisi Tree

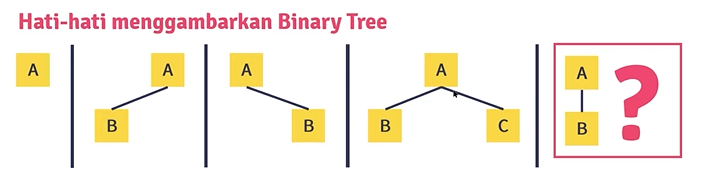
* Sebuah tree didefinisikan sebagai struktur yang dibentuk secara rekursif oleh aturan berikut:
  + Sebuah node adalah sebuah tree.Node satu satunya pada tree ini berfungsi sebagai root maupun leaf
  + Dari K buah tree T1 sampai Tk dan masing masing memiliki root n1 sampai nk
  + Jika node n adalah parent dari n1 sampai nk akan diperoleh sebuah tree baru T yang memiliki root n.Dalam kondisi ini,tree T1 sampai Tk menjadi subtree dari tree T
  + T = Tree
  + N = Node
  + K = banyak

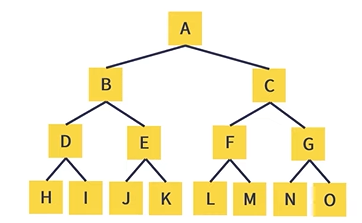
Ordered & Unordered Tree

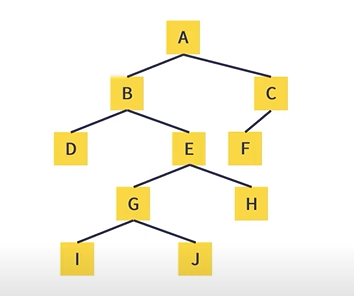
* Ordered Tree
  + Antar sibling (saudara) terdapat urutan “usia”
  + Node yang paling kiri berusia paling tua (Sulung),sedangkan node yang paling kanan berusia paling muda (bungsu)
  + Posisi node diatur atas urutan tertentu
  + Contoh :silsilah keluarga
* Unordered tree
  + Antara sibling (saudara) tidak terdapat urutan tertentu
  + Contoh struktur organisasi

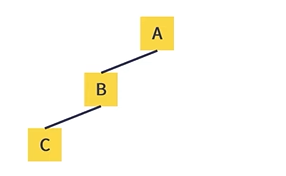
Konsep Binary Tree

* Binary adalah tree dengan syarat bahwa tiap node hanya boleh memiliki maksimal dua subtree dan kedua subtree harus terpisah
  + Binary tree boleh tidak memiliki child (anank) ataupun subtree
  + Boleh hanya memiliki subtree sebelah kiri (left subtree)
  + Boleh hanya memiliki subtree sebelah kanan (rught subtree)
  + Boleh hanya memiliki subtree sebelah kiri (left subtree) dan kanan (right subtree)
  + Tidak boleh lebih dari 2 child

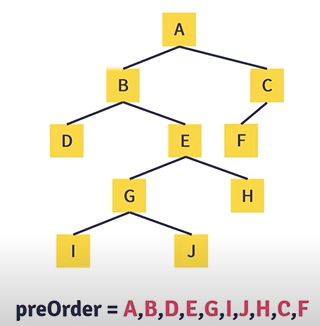


* Jenis Jenis binary tree
  + Full Binary Tree adalah binary tree yang tiap node nya (kecuali leaf) memiliki dua child tiap subtree mempunyai Panjang patch yang sama



* + Complete Binary Tree adalah binary tree yang mirip dengan full binary tree, namun setiap subtree boleh memiliki Panjang patch yang berbeda
  + Skewed Binary Tree adalah binary tree yang semua node nya (kecuali left) hanya memiliki satu child

Definisi Tree Tranversal

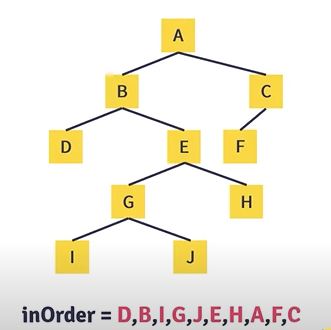
* Teknik menyusuir node dalam sebuah tree secara sistematis,sehingga semua node dapat dan hanya satu kali saja dikunjungi
* Ada tig acara transversal
  + preOrder

Cara nya :

1. Kunjungi rootnya

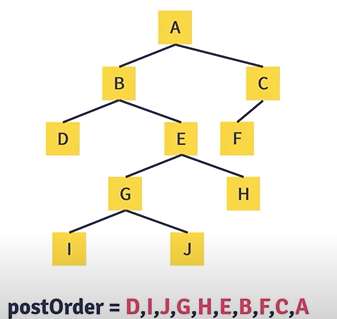
2. Telusuri subtree nya

3. Telusuri subtree kanan



* + inOrder

Caranya :

1. Telusuri subtree kiri
2. Kunjungi root nya
3. Telusuri subtree kanan
   * postOrder

Caranya :

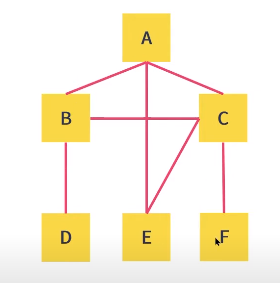
1. Telusuri subtree kiri
2. Telusuri subtree kanan
3. Kunjungi root nya

* Untuk tree atau node yang kosong,tranversal tidak perlu dilakukan

Operasi pada Tree

* **Create** digunakan untuk membentuk binary tree baru yang masih kosong
* **Clear**  digunakan untuk mengosongkan binary tree yang sudah ada atau menhapus semua node pada binary tree
* **Empety**  digunakan untuk memeriksa apakah binary tree masih kosong atau tidak
* **Insert**  digunakan untuk memeasukan sebuah node kedalam tree
* **Find** digunakan untuk mencari root,paret,leaf,child,atau rught child dari suatu node dengan syarat tree tidak boleh kosong
* **Update** digunakan untuk mengubah isi dari node yang ditunjukan oleh pointer current dengan syarat tree tidak boleh ksoong
* **Retrieve** digunakan untuk mengetahui isi dari node yang ditunjukan pointer current dengan syarat tree tidak boleh kosong
* **Delet Sub**  digunakan untuk menghapus sebuah subtree (node berserta seluruh descendant-nya) yang ditunjuk pointer current dengan syarat tree tidak boleh kosong
* **Charateristic** digunakan untuk mengetahui karakteristik dari suatu tree.Yakni size,height,serta average length-nya
* **Tranverse** digunakan untuk mengunjungi seluruh node-node pada tree dengan cara tranversal

Graph

Konsep Graph

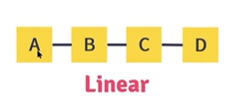
* Konsep struktur data yang terdiri dari node (vertex atau vertices) dan garis penghubung(arc atau edge)
* Vertex disimbolkan dengan V dan edge disimbolkan dengan E
* Keterhubungan graph dapat membentuk relasi One to One,One to Many,Many to One dan Many to Many
* Contoh : Informasi topologi jaringan,keterhubungan antar kota kota

Contoh Pengunaan Graph

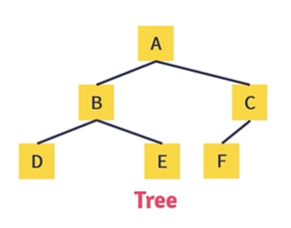
* Menggambar jaringan dan peta jalan,jalan kereta api,lintasan pesawat,system perpipaan,saluran telepon,koneksi elektrik,ketergantungan diantara task pada system manufaktur
* Terdapat banyak hasil dan struktur penting yang iddapatkan dari perhitungan dangan graph

Masalah yang bisa diselesaikan dengan Graph

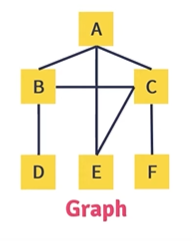
* Shortest Path Problem (Masalah Patch Minimum) :Mencari route dengan jarak terpendek dalam suatu jaringan transportasi
* Maximum Flow Problem ( maslah Aliran Maksimum) : Manghitung volume aliran BBM dari suatu reservoir ke suatu titik tujuan melalui jaringan pipa
* Graph Searching Problem ( Masalah Pencarian dalm Graph) : mencari langkah langkah terbaik de;am program permainan catur computer
* Topologi Ordering Problem (Masalah Pengurutan Topologi): menentukan urutan pengambilan mata mata kuliah yang saling berkaitan dalam hubunga prasyarat (prerequisite)
* Task Network Problem (Masalah Jringan Tugas):membuat penjadwalan perngerjaan suatu proyek yang memungkinkan waktu peyelesian tersingkat
* Minimum Spanning Tree Problem ( Masalah Pnecarian Pohon Rentag Minumum) : mencari reb=ntang kabel listrik yang totalnya adalah minimal untuk menghubungkan sejumlah kota
* Traveling Salesperson Problem : tukang pos mencari lintasan terpendek melalui semua alamat penerima pos tanpa harus mendatangi suatu empat lebih dari satu kali
* Four-color Problem : dalam menggambar peta,memberikan warna berbeda pada setiap provinsi yang saling bersebelahan

Sturktur Data linier vs Tree vs Graph

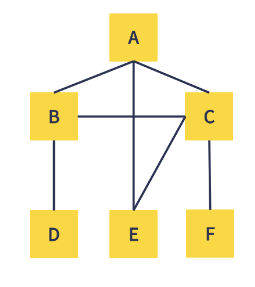
* Linier Keterhubgungan sekuantial (terurut) antara entitas data

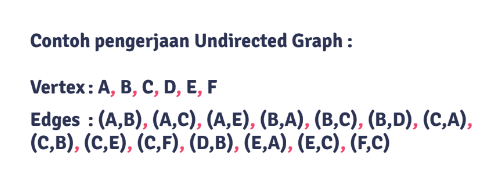
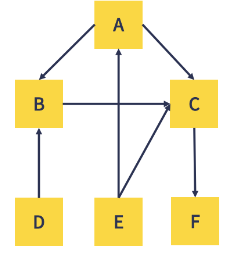
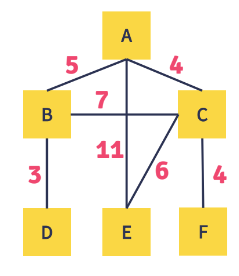
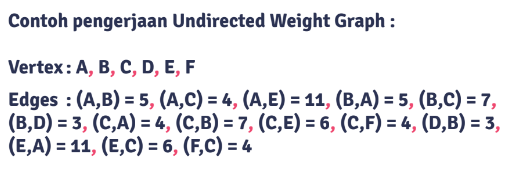
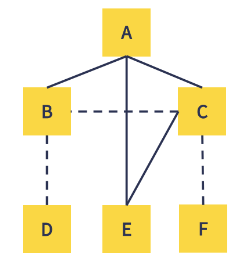


* Tree keterhubungan hirarkis



* Graph keterhubungan tak terbatas antara entitas data

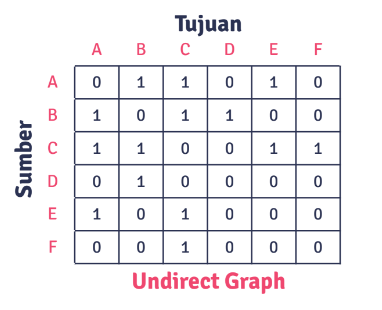
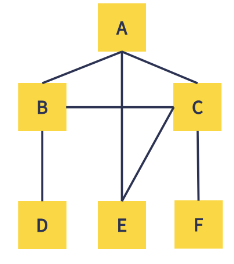
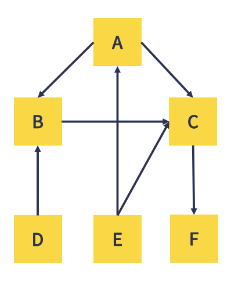
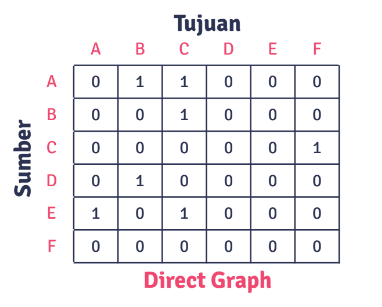
Kategori Graph

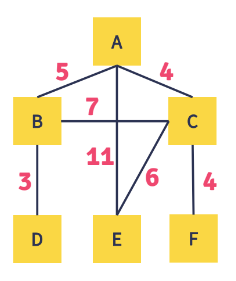
* Undirected Graph (Undigraph)
  + Edge tidak ada penugasan arah dari satu vertex ke vertex yang lain
  + Digunakn untuk mempresentasikan relasi One-to-One
  + Jika A dan B adalah vertex maka edge dapat mempresentasikan sebagai (A,B) dan (B,A)
* Directed Graph (Digraph)
  + Edge ada penugasan arah dari suatu vertex ke vertex yang lain
  + Edges antara Vertex diwakilkan dengan kurung sudut
  + Jika vertex A sebagai sumber dan menuju ke arah vertex B sebagai tujuan,maka edge dapat mempresentasikan sebagai <A,B>
* Weighted Graph
  + Edge diberikan suatu nilai sebagai weight dari satu vertex ke vertex yang lain
  + Dapat direpresentasikan oleh Undirected atau Directed Graph
  + Weight dapat direpresentasikan sebagai jarak,lama waktu
* Graph Paths
  + Path adalah urutan vertice yang dihunungkan oleh edges atau adjencent vertices

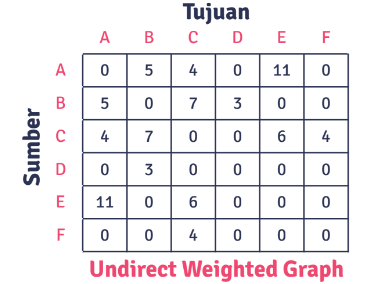
Operasi pada Graph

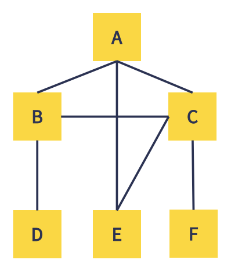
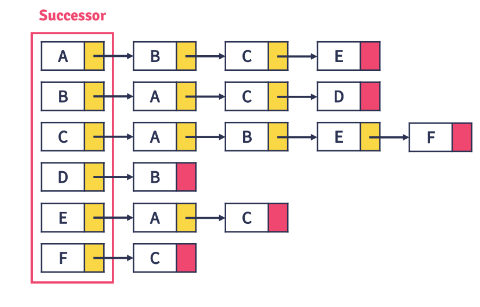
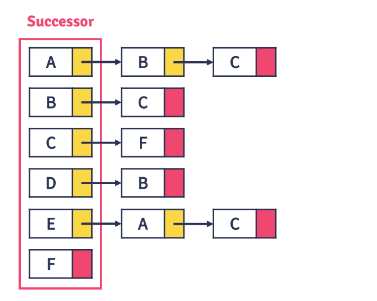
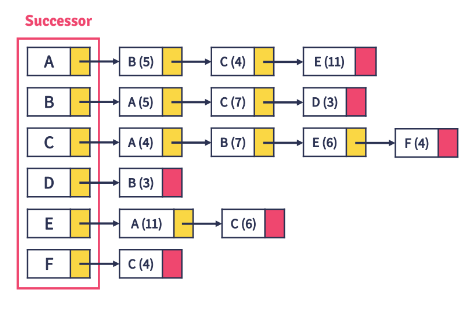
* Add Vertex : menambhkan vertex bar uke dalam graph
* Delete Vertex menhapus vertex tertentu disuatu graph
* Add Edge : menambahkan edge baru yang menghubungkan antar vertex disuatu graph
* Delete Edge : menhapus suatu edge yang menghubungkan antara vertex disuatu graph
* Search Path : mencari panjan paths dari suatu vertex ke vertex yang lain
* Transversing : penelusuran dari vertex lainnya dengan penghubung edge

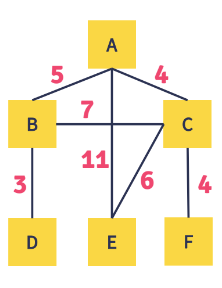
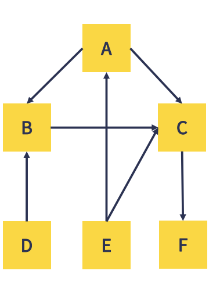
Konsep Implementasi Graph

* Adjacency Matrices (array Based) : Implemntasi menggunkana array matriks
  + Representasi dengan array 2 dimensi atau matriks
  + Pada Undirect & Direct Graph ) berarti tidak ada relasi edge,sedangkan nilai 1 ada relasi edge
  + Pada weighted Graph nilainya diisi nilai beb an edge nya





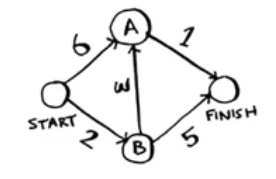
* Adjancency List(Linked List)
  + Representasinya dengan linked list
  + Node asal akan disebut Successor
  + Isi dari Successor akan disi list node yang berkaitan,node terakhir akan disi pada nextnya
  + Implementasinya harus paham PBO (Pemrograman Berbasis Objek)

Konsep Algortima Djikstra

* Kumpulan node (simpul) yang membentuk graf  berbobot dan berarah yang dimulai dari simpul awal sampai simpul tujuan dalam mencari rute terpendek
* Djikstra Algoritma adalah algortima greedy yang dipakai dalam memecahkan permasalahan jarak terpendek untuk sebuah graf berarah dengan bobot sisi(edge) yang bernilai tak negative
* Djikstra Algoritma hanya berkerja/digunakan pada Directed Acyclic Graph atau biasa disingkat DAG

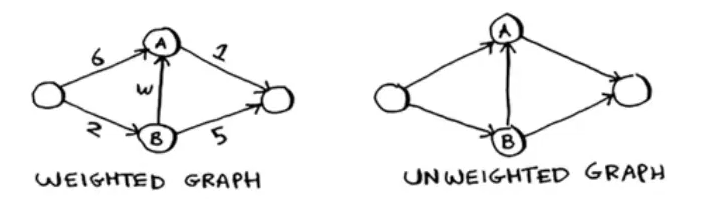
Cara Kerja Djikstra Algoritma

* Disni kita mempunyai 4 node : star node,A node,B node,dan finish node

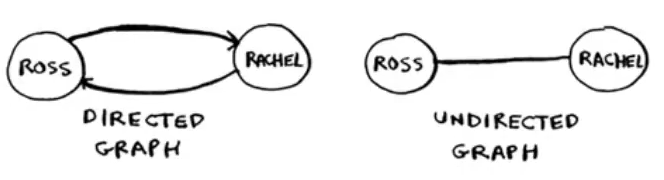
* Langkah pertama pilih antara node A dan B,kita compare mana yang lebih kecil
  + Karena kita belum tau berapa yang dibutuhkan untuk ke  node finish maka menuju node finish kita beri infinity
  + Disini kita tau bahwa node B lebih dekat dengan finish setelah kita compare dengan node A
* Langkah kedua seperti langkah diatas kita telah memilih node B sebagai node terkecil dibandingkan denga A
  + Di node B kita melakukan update terhadap semua neighboors  dari node B,neighboors dari node B adalah node A dengan jumlah 3,sedangkan dari star menuju B adalah 2 berarti dari node star sampai node A melalui node B memiliki jumlah 5, dari data sebelumnya bahwa dari node star sampai A adalah 6 berarti bahwa data yang kita dapatkan barusan lebih baik dari data sebelunya
  + Ada satau neighboors dari node B yaitu node finish,pada data sebelumnya kita belum tau dari node star menuju node finish perlu jumlah berapa atau tadi kita kasih nila infinity,tetapi disini kita sudah tau bahwa nilainya 7 dari jalur star,B,finish
* Langka ke tiga adalah kita  mengulangi lagi langkah pertama
  + Disini kita mengcompre lagi dari node  A yang melalui node B dengan node finish,didapatkan bahwa node A lebih kecil dari node finish maka kita pilih yang node A
  + Setelah kiita memilih nnode A neigboors dari node A adalah node finish dengan biaya 1
  + Dari data sebelumnya kita tau bahwa node A adalah 5 dan dari node A sampai finish adalah 1,jadi total nya adalah 6,sedangkan dari data yang tadi kita dapatkan bahwa untuk sampai ke node finish membutuhkan 7 artinya data yang baru saja kita dapat lebih baik dari data sebelumnya

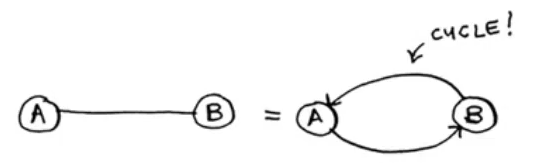
Directed Acyclic Graph

* DAG adalah graph berarah yang tidak memiiki cycle/loop didalamnya
* Kompenen DAG adalah root dan leaf
* Root adalah simpul tanpa sisi masukan
* Leaf adalah simpul tanpa sisi keluar



* Weighted Graph adalah graph yang memiliki bobot
* Unweighted Graph adalah graph yang tidak memiliki bobot
* Untuk mencari jalur terpendek pada weighted graph kita menggunakan Djikstra Algoritma
* Sedankan pada unweighted graph kita menggunakan BFS



* Directed Graph adalah graph yang memiliki anak panah
* Sedangan undirected graph adalah graph yang tidak memiliki anak panah
* Undirected grah sama dengan cycle graph yang artinya a bisa ke b dan b bisa ke a
* Seperti gambar dibawah

Recap

* Djikstra algoritma hanya berkerja pada positive weights
* Jika kita  menemui negative weights kitab isa menggunakan Bellman-Ford algortima
* BFS digunaan untuk mencari shortest path pada unweighted graph
* Djikstra algoritma digunakan untuk mencari shortest path pada weighted graph

Konsep Algoritma Greddy

* Algoritma greedy adalah pendekatan dalam pemrograma yang memilih langkah terbaik pada setiap langkah dalam harapan bahwa itu akn mengarah pada solusi global terbaik.
* Pada setiap langkah,algoritma greedy memilih opsi yang tampaknya paling menguntungkan pada saat itu,tanpamempertimbangkan implikasi jangka Panjang atau memeriksa solusi alternatif

Masalah yang dapat diselesaikan dengan Algoritma Greedy

* Masalah pemilihan Aktivitas (Activity Selction Problem)
* Masalah Pemotongan Batang (Cutting Stock Problem)
* Masalah Koin Minimum (Minimum Coin Change Problem)
* Masalah Pencocokan Interval (interval Scheduling Problem)
* Masalah Pohon Merentang Minimum (Minimum Spaning Tree Problem)
* Masalah Ransel (Knapsack Problem)
* Masalah Pemilihan Jalur Terpendek (Shortest Path Problem)
* Masalah Penjadwalan Tugas (Task Scheduling Problem)

Klasifikasi Algoritma Greedy

* Greedy Terikat (Bound Greedy)
  + Mempertimbangkan Batasan atau kriteria tertentu dalam pengambilan keputusan greedy
  + Memilih langkah terbaik yang memenuhi Batasan yang ditetapkan
  + Pilihan yang dibuat tidak melampaui batasan atau kriteria yang telah ditentukan sebelumnya
* Greedy Tidak Terikat (Unbound Greedy)
  + Tidak mempertimbangkan Batasan atau kriteria tertentu dalam pengambilan keputusan serakah
  + Memilih langkah terbaik berdasarkan factor factor local tanpa mempertimbangkan Batasan global atau kriteria yang ditetapkan sebelumnya
* Jenis Algoritma Greedy
  + Greedy Terikat (Bound Greedy)
    - Algoritma Prim
    - Algoritma Kruskal
    - Algoritma Set Cover
    - Algoritma Job Scheduling
    - Algoritma Huffman Coding
  + Greedy Tidak Terikat (Unbound Greedy)
    - Algoritma Djikstra
    - Algoritma Knapsack Fractional
* Kesimpulan
  + Algoritma Greedy merupakan algoritma yang sangat berguna dalam menyelesaikan masalah optimasi
  + Algoritma ini berkerja dengan memilih opsi terbaik pada setiap tahap,sehingga mencapai solusi terbaik pada akhirnya

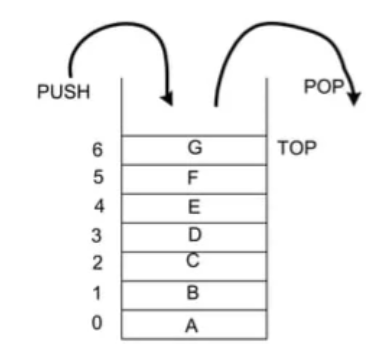
* **Knapsack Problem**
  + Diterjemahkan menjadi ransel/karung
  + Permasalahan Pengelola Ruang
  + Ransel sebagai ruang penyimpanan barang
  + Spesifikasi Barang : Berat,Kegunaan,Ukuran
  + Spesifikasi Ransel : Kapasitas/Daya Tampung

* **Penyelesian Knapsack dengan Greedy**
  + Fokus pada 3 kondisi Prioritas
    - Profit lebih besar
    - Berat lebih kecil
    - Perbandingan Profit/Berat lebih besar
  + Ketiga di atas disebut Optimal Solution
* **Case Scheduling**
  + Misalkan kita mejadwalkan suatu kelas maka langkah yang harus dilakukan adalah
    - Cari kelas berakhiran paling awal
    - Cari kelas setelah kelas pertama berakhir lalu pilih lagi kelas yang paling luar
* **Set Covering Problem**
  + Misalkan kita mempunyai program radio langkah yang harus kita lakukan adalah
    - Memilih stasiun radio dengan jangkuan paling luas tidak masalah jika stasion ini telah di jangkau sebelumnya
    - Ulangi langka ini sampai stasion tercover keseluruhan
    - Ini biasa disebut dengan Algoritma Approximmation
      * Tidak mencari solusi paling optimal tapi mendekati optimal
      * Approximmation banyak digunakan ketika menyelesaikan problem yang membutuhkan waktu yang sangat lama

Stack dan Queue

Stack

* Stack adalah struktur data dinamis yang mengikuti prinsip Last in First Out(Lifo) artinya item yang terakhir disertakan dalam stack menjadi item pertama yang dihilangkandari stack
* Penyimpanan data yang diakses adalah paing akhir yang disebut top of stack
* Untuk memasukan elemen ke dalam tumpukan disebut Push,dan penghapusan elemn dari tumpukan disebut Pop.Di Stack selalu melacak elemen terakhir yang ada dalam daftar dengan penunjuk bernama Top
* Representasi diagram stack :

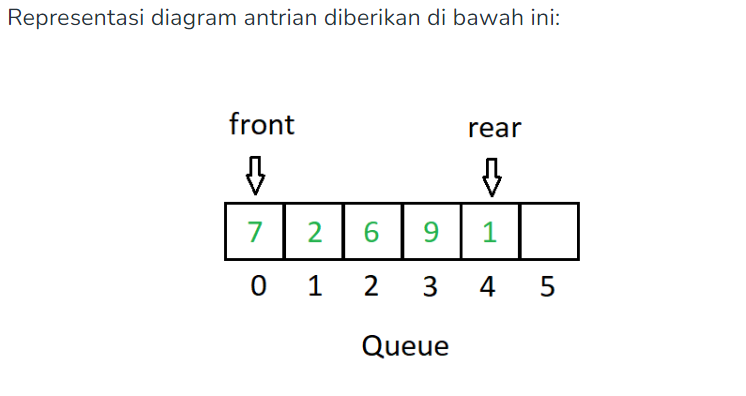


* Sifat sifat Stack

1. Struktur data dinamis
2. Tidak memiliki ukuran tetap
3. Tidak menggunkan kapasitas memori tetap
4. Ukuran Stack berubah dengan tiap operasi push dan pop

Queue

* Struktur data yang mengikuti pola Firt in First out (FiFo) artinya element pertama yang ditambahkan ke dalam queue akan mejadi element pertama yang akan di hapus
* Penyisipan kedalam queue disebut operasi enqueue dan penhapusan suatu elemn disebut dequeue.
* Dalam queue selalu mempeertahankan dua penunjuk,satu menunjukan ke elemen yang disisipkan di elemen pertama dan masih ada dalam daftar dengan penunjuk depan and penunjuk kedua menunjukan ke elemen yang disisipkan terakhir dengan penunjuk back



* Penerapan Stack
  + Beberapa CPU memiliki seluruh Bahasa rakitannya berdasarkan konsep melakukan operasi pada register yang disimpan dalm stack
  + Struktur Stack digunakan dalam system run-time C++
* Penerapan Queue
  + Struktur data queue diimpelmentasikan dalam instrukai mikro perangkat kereas di dalam CPU
  + Struktur queue digunakan di sebagain besar system operasi

Hashing

* Hashing adalah transformasi aritmatika sebuah string dari karakter menjadi nilai yang merepresentasikan string aslinya
* Menurut bahasanya,hash berarti memenggal dan kemudian menggabungkan
* Hashing digunakan sebagai metode untuk menyimpan data dalam sebuah larik(array) agar penyimpanan data,pencarian data,penambahan data,dan pengahpusan data dapat diilakukan dengan cepat
* Hashing digunakan untuk mengindex dan mendapatkan kembali key di database (hash table),karena lebih cepat untuk mengambil key yang sudah dihash daripada mencarinya menggunakan original value
* Hashing juga dikenal sebagai konsep pendistribusian key dalam sebuah array (Hash Table) menggunakan fungsi yang sudah diketahui sebelumnya (Hash Function)

Hash Table

* Hash table adalah array dengan sel sel yang ukurannya telah ditentukan dan dapat berisi data atau ke yang berkesesuian dengan data
* Hash table adalah sebuah array untuk menyipan original string.Index dari hash table adalah hashed key.
* Besarnya hash table lebih kecil dari jumlah key yang ada ,sehungga memungkinkan ada string yang memiliki hashed key yang sama.

Hash Function

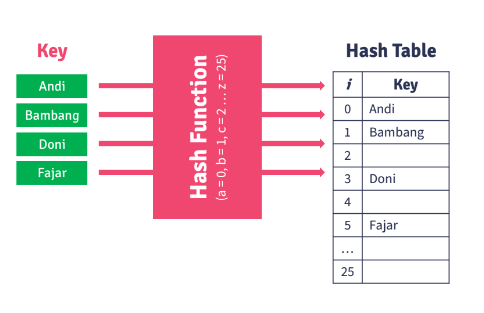
* Hash Function adalah cara yang kita buat untuk men-tranformasi artimetika sebuah string dari karakter menjadi nilai yang merepresentasikan string aslinya
* Secara teori,kompleksitas wakatu (T(n)) dari fungsi hash yang ideal adalah algoritma konstan 0(1).Untuk mencapai itu setiap record membutuhkan suatu kunci yang unik

Notes

* Salah satu kegunaan hashing adalah untuk mengamankan tulisan yang biasanya diterapkan untuk menyembuyikan password asli
* Teknik dan cara hashing semakin berkembang seiring dengan kemanan data yang terus bisa diretas

Cara-cara Hash Function

* Untuk dasarnya,biasnya mengambil dari satu karakter awal pada string original dan diubah menjadi angka index yang nantinya disimpan le hash table.



* Banya cara untuk melakukan hash sebuah string menjadi hashed key
  + Mid-Sqeare
    - Pangkat key,dan ambil bit pada bagian tengah dari hasil pnagkat untuk dijadikan hash-key
    - Jika key berupa string maka ubah menjadi number dengan cara:

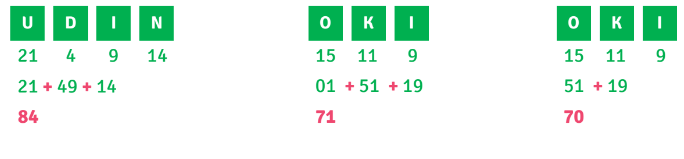


* Division
  + Lakukan pembagian pada key dengan operator modulus (sesuai dengan jumlah hash table)
  + Jika key berupa string maka ubah menjadi nuber dengan cara :



* Folding
  + Hash Function folding akan men-transformasi string ke hashed key dengan 2 langkah

1. Bagi key mnejadi bagian bagian yang memiliki jumlah digit sama
2. Jumlahkan setiap bagian





* Digit Extraction
  + Mendapatkan hashed key dengan cara mengambil digit – digit tertentu dari sebuah key



* Rotating Hash
  + Mendapatkan hashed key dengan cara membalikan urutan dari key



* Truncation
  + Mendapatkan hashed key dengan cara memenggal key sebanyak K (tidak boleh dilewati) dari n digit,dimana K<n



Hash Function yang baik

* Hash Function harus memiliki sifat berikut:
  + Mudah dihitung
  + Due key yang berbeda alan dipetakan pada dua sel yang berbeda [ada array.Tapi secara umum tidak bisa berlaku karena bisa jadi dua key yang berbeda mempunyai hasil hashed key yang sama (tabrakan).
  + Membagi key secara rata pada seluruh sel

* Sebuah hash function sederhana adalah menggunakan fungsi modulus (sisa bagi) dengan bilangan prima
* Dapat menggunakan manipulasi digit dengan kompleksitas rendah dan distribusi key rata

Memilih Hash Function

* Sebuah hash function yang bagus memiliki 2 kriteria

1. Mudah dan cepat dihitung
2. Harus meminimalka juga collosions (tabrakan) yang terjadi

Collision

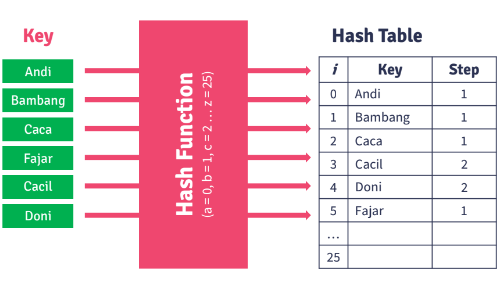
* Dikatakan terjadi collision (tabrakan) jika dua buah key dipetakan pada sebuah sel yang sama
* Collision bisa terjadi saat melakkan insertion (penambahan data)
* Penyelesaian bila terjadi collision (tabrakan) disebut Collision Resolution
* Dibutuhkan prodesur tambahan untuk mengatasi terjadinya collision (tabrakan)

Collision Resolutin

* Ada dua strategi umum dalam melakukan Collision Resolution:

1. Closed Hashing – Linear Probing

* Idenya adalah mencari alternatif sel lain pada hash table



1. Open Hashing (Chaining)

* Idenya meletakan key pada table menggunakan linked list

