**Pengertian Desain Pattern**

***Design Pattern*** adalah sebuah solusi untuk menyelesaikan masalah umum yang sering di temui dan dihadapi, yang membentuk sebuah pola (*pattern*) tertentu dalam membuat sebuah program. ***Pattern*** adalah bagaimana mendesain kelas dan bagaimana interaksi yang terjadi antar kelas tersebut sehingga kelas yang kita bangun bisa lebih elegan dan reusable.  
Ada beberapa  pattern yang populer saat ini yang diciptakan oleh GoF (Gang of Four) yaitu Abstract Factory, Builder, Factory Method, Prototype, Singleton, Adapter, Bridge, Composite, Decorator, Facade, Flyweight, Proxy, Chain of Responbility, Command, Interpreter, Iterator, Mediator, Mementto, Observer, State, Strategy, Template Method, Visitor.

Secara singkat, design pattern adalah sebuah solusi untuk mengulang masalah design. Solusi ini telah dikembangkan, dikompile, dan disaring oleh programmer yang sebelumnya telah  menghadapi masalah ini. Sebuah *design pattern* adalah salah satu jenis petunjuk yang menggunakan tipe "Jika Anda memiliki masalah ini, maka lakukan ini." Hal tersebut serupa dengan sebuah algoritma : sebagai contoh, jika Anda ingin melakukan sebuah operasi pencarian pada sebuah daftar, Anda tidak perlu menyelesaikan dengan solusi Anda sendiri, terdapat beberapa algoritma yang  menyediakan  fungsi sorting ini seperti pencarian biner. Perbedaan antara sebuah Design pattern dan sebuah  algoritma adalah bahwa algoritma fokus pada implementasi solusi, sebuah algoritma biasanya mendaftar dari atas ke bawah menuju ke tahap akhir dari action-action yang terurut dimana yang dibutuhkan untuk menghasilkan sebuah solusi. Sebuah pattern memusatkan untuk menghasilkan beberapa Design.

Hal tersebut memberitahu kepada pengembang apa yang harus diperbuat sebagai lawan dari bagaimana cara melakukannya. Mungkin tidak terlihat langsung hasilnya, tetapi kita telah siap menggunakan beberapa Design pattern dalam pembahasan Kita sebelumnya. Satu, Kita telah belajar secara detail pattern dari Model-View-Controller : diberikan sebuah masalah interaksi gabungan yang begitu kompleks antara presentasi dan layer business, Hal itu menentukan pemisahan kode menjadi tiga layer terpisah seperti interaksi-interaksi mereka. Ingat, pattern itu sendiri tidak mendikte implementasi, hanya dijelaskan sebuah solusi Design. Kita harus melaksanakan pattern MVC manapun untuk untuk menggunakan framework third-party.

**Design pattern dapat dibagi menjadi 3 kelompok yaitu:**

**1.        Creational design pattern**

Pattern ini hanya berbicara tentang bagaimana meng-instance sebuah kelas. Secara teori memang mudah untuk membuat object tetapi cara yang mudah tersebut belum tentu efektif. Adapun jenis-jenis dari creational design pattern adalah *Abstract Factory, Builder, Factory Method, Object Pool, Prototype,Singleton*. Dari sekian banyak jenis-jenis tersebut ada yang memiliki fungsi yang sama dan saling melengkapi.

**2.        Structural design patterns**

Tujuan pattern ini untuk membuat class composite atau object composite. Tujuan utamanya adalam mempermudah dalam mendefinisi kan cara untuk meng-compose object untuk fungsionalitas yang baru. Adapun contoh dari pattern ini adalah : Adapter pattern, bridge pattern, composite pattern, Decorator, Facede, Flyweight, Proxy.

**3.        Behavioral design patterns**

      Pattern ini adalah bertujuan untuk menangani kominikasi antar kelas atau object. Adapun contoh dari      pattern ini adalah : Chain of responsibility, Command, Interpreter, Iterator, Mediator, Observer, strategy.

**ALGORITMA**

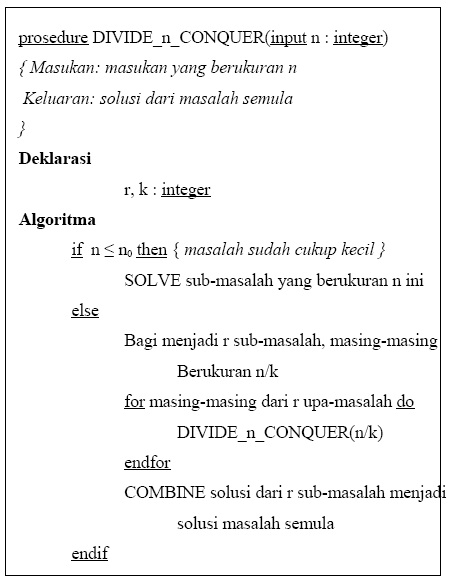
**Pengertian**

**Algoritma Divide** and Conquer merupakan algoritma yang sangat populer di dunia Ilmu Komputer. Divide and Conquer merupakan algoritma yang berprinsip memecah-mecah permasalahan yang terlalu besar menjadi beberapa bagian kecil sehingga lebih mudah untuk diselesaikan. Langkah-langkah umum algoritma Divide and Conquer :

* Divide : Membagi masalah menjadi beberapa upa-masalah yang memiliki kemiripan dengan masalah semula namun berukuran lebih kecil ( idealnya berukuran hampir sama ).
* Conquer : Memecahkan ( menyelesaikan ) masing-masing upa-masalah ( secara rekursif ).
* Combine : Menggabungkan solusi masing-masing upa-masalah sehingga  membentuk solusi masalah semula.

Objek masalah yang di bagi adalah masukan (input) atau instances yang berukuran n: tabel (larik), matriks, dan sebagainya, bergantung pada masalahnya. Tiap-tiap upa-masalah mempunyai karakteristik yang sama (the same type) dengan karakteristik masalah asal, sehingga metode Divide and Conquer lebih natural diungkapkan dalam skema rekursif. Sesuai dengan karakteristik pembagian dan pemecahan masalah tersebut, maka algoritma ini dapat berjalan baik pada persoalan yang bertipe rekursif (perulangan dengan memanggil dirinya sendiri). Dengan demikian, algoritma ini dapat diimplementasikan dengan cara iteratif ( perulangan biasa ), karena pada prinsipnya iteratif hampir sama dengan rekursif. Salah satu penggunaan algoritma ini yang paling populer adalah dalam hal pengolahan data yang bertipe array ( elemen larik ). Mengapa ? Karena pengolahan array pada umumnya selalu menggunakan prinsip rekursif atau iteratif. Penggunaan secara spesifik adalah untuk mencari nilai minimal dan maksimal serta untuk mengurutkan elemen array. Dalam hal pengurutan ini ada empat macam algoritma pengurutan yang berdasar pada algoritma Divide and Conquer, yaitu merge sort, insert sort, quick sort, dan selection sort. Merge sort dan Quick sort mempunyai kompleksitas algoritma O(n ²log n). Hal ini lebih baik jika dibandingkan dengan pengurutan biasa dengan menggunakan algoritma brute force.

**Skema Umum Algoritma Divide and Conquer :**



***Brute force***adalah sebuah pendekatan yang lempang (*straightforward*) untuk memecahkan suatu masalah, biasanya didasarkan pada pernyataan masalah (problem statement) dan definisi konsep yang dilibatkan.

**Algoritma *brute force***memecahkan masalah dengan sangat sederhana, langsung dan dengan cara yang jelas (*obvious way*).

contoh

* Menghitung an (a > 0, n adalah bilangan bulat tak-negatif)

an = a × a × … × a (n kali) , jika n > 0

= 1 , jika n = 0

**lgoritma Boyer-Moore** adalah salah satu [algoritma pencarian string](http://id.wikipedia.org/wiki/Algoritma_pencarian_string" \o "Algoritma pencarian string), dipublikasikan oleh [Robert S. Boyer](http://id.wikipedia.org/w/index.php?title=Robert_Boyer&action=edit&redlink=1), dan [J. Strother Moore](http://id.wikipedia.org/w/index.php?title=J._Strother_Moore&action=edit&redlink=1" \o "J. Strother Moore (halaman belum tersedia)) pada tahun 1977.

Algoritma ini dianggap sebagai algoritma yang paling efisien pada aplikasi umum.[1]Tidak seperti algoritma pencarian string yang ditemukan sebelumnya, algoritma Boyer-Moore mulai mencocokkan karakter dari sebelah kanan pattern. Ide dibalik algoritma ini adalah bahwa dengan memulai pencocokkan karakter dari kanan, dan bukan dari kiri, maka akan lebih banyak informasi yang didapat.[[2]](http://id.wikipedia.org/wiki/Algoritma_Boyer-Moore" \l "cite_note-2)

Misalnya ada sebuah usaha pencocokan yang terjadi pada teks[i..i+n-1], dan anggap ketidakcocokan pertama terjadi di antara teks[i+j]dan pattern[j], dengan 0 < j < n. Berarti, teks[i+j+1..i+n-1] = pattern[j+1..n-1]dan  a=teks[i+j]tidak sama dengan b=pattern[j] . Jika uadalah akhiran dari pattern sebelum bdan vadalah sebuah awalan dari pattern, maka penggeseran-penggeseran yang mungkin adalah:

1. Penggeseran good-suffix yang terdiri dari mensejajarkan potongan teks[i+j+1..i+n-1] = pattern[j+1..n-1]dengan kemunculannya paling kanan di pattern yang didahului oleh karakter yang berbeda dengan pattern[j]. Jika tidak ada potongan seperti itu, maka algoritma akan mensejajarkan akhiran vdari teks[i+j+1..i+n-1]dengan awalan dari pattern yang sama.
2. Penggeseran bad-character yang terdiri dari mensejajarkan teks[i+j]dengan kemunculan paling kanan karakter tersebut di pattern. Bila karakter tersebut tidak ada di pattern, maka pattern akan disejajarkan dengan teks[i+n+1].

Secara sistematis, langkah-langkah yang dilakukan algoritma Boyer-Moore pada saat mencocokkan string adalah:

1. Algoritma Boyer-Moore mulai mencocokkan pattern pada awal teks.
2. Dari kanan ke kiri, algoritma ini akan mencocokkan karakter per karakter pattern dengan karakter di teks yang bersesuaian, sampai salah satu kondisi berikut dipenuhi:
   1. Karakter di pattern dan di teks yang dibandingkan tidak cocok (mismatch).
   2. Semua karakter di pattern cocok. Kemudian algoritma akan memberitahukan penemuan di posisi ini.
3. Algoritma kemudian menggeser pattern dengan memaksimalkan nilai penggeseran good-suffix dan penggeseran bad-character, lalu mengulangi langkah 2 sampai pattern berada di ujung teks.

Berikut adalah pseudocode algoritma Boyer-Moore pada fase pra-pencarian:

procedure preBmBc(

input P : array[0..n-1] of char,

input n : integer,

input/output bmBc : array[0..n-1] of integer

)

Deklarasi:

i: integer

Algoritma:

for (i := 0 to ASIZE-1)

bmBc[i] := m;

endfor

for (i := 0 to m - 2)

bmBc[P[i]] := m - i - 1;

endfor

procedure preSuffixes(

input P : array[0..n-1] of char,

input n : integer,

input/output suff : array[0..n-1] of integer

)

Deklarasi:

f, g, i: integer

Algoritma:

suff[n - 1] := n;

g := n - 1;

for (i := n - 2 downto 0) {

if (i > g and (suff[i + n - 1 - f] < i - g))

suff[i] := suff[i + n - 1 - f];

else

if (i < g)

g := i;

endif

f := i;

while (g >= 0 and P[g] = P[g + n - 1 - f])

--g;

endwhile

suff[i] = f - g;

endif

endfor

procedure preBmGs(

input P : array[0..n-1] of char,

input n : integer,

input/output bmBc : array[0..n-1] of integer

)

Deklarasi:

i, j: integer

suff: array [0..RuangAlpabet] of integer

preSuffixes(x, n, suff);

for (i := 0 to m-1)

bmGs[i] := n

endfor

j := 0

for (i := n - 1 downto 0)

if (suff[i] = i + 1)

for (j:=j to n - 2 - i)

if (bmGs[j] = n)

bmGs[j] := n - 1 - i

endif

endfor

endif

endfor

for (i = 0 to n - 2)

bmGs[n - 1 - suff[i]] := n - 1 - i;

endfor

Dan berikut adalah pseudocode algoritma Boyer-Moore pada fase pencarian:

procedure BoyerMooreSearch(

input m, n : integer,

input P : array[0..n-1] of char,

input T : array[0..m-1] of char,

output ketemu : array[0..m-1] of boolean

)

Deklarasi:

i, j, shift, bmBcShift, bmGsShift: integer

BmBc : array[0..255] of interger

BmGs : array[0..n-1] of interger

Algoritma:

preBmBc(n, P, BmBc)

preBmGs(n, P, BmGs)

i:=0

while (i<= m-n) do

j:=n-1

while (j >=0 n and T[i+j] = P[j]) do

j:=j-1

endwhile

if(j < 0) then

ketemu[i]:=true;

endif

bmBcShift:= BmBc[chartoint(T[i+j])]-n+j+1

bmGsShift:= BmGs[j]

shift:= max(bmBcShift, bmGsShift)

i:= i+shift

**Algoritma *Graff*** adalah cabang ilmu yang mempelajari sifat-sifat graf. Secara informal, suatu graf adalah himpunan benda-benda yang disebut verteks (atau node) yang terhubung oleh edge-edge (atau arc). Biasanya graf digambarkan sebagai kumpulan titik-titik (melambangkan verteks) yang dihubungkan oleh garis-garis (melambangkan edge).  
Banyak sekali struktur yang bisa direpresentasikan dengan graf, dan banyak masalah yang bisa diselesaikan dengan bantuan graf. Jaringan persahabatan pada Friendster bisa direpresentasikan dengan graf: verteks-verteksnya adalah para pemakai Friendster dan ada edge antara A dan B jika dan hanya jika A berteman dengan B. Perkembangan algoritma untuk menangani graf akan berdampak besar bagi ilmu komputer.  
Sebuah struktur graf bisa dikembangkan dengan memberi bobot pada tiap edge. Graf berbobot dapat digunakan untuk melambangkan banyak konsep berbeda. Sebagai contoh jika suatu graf melambangkan jaringan jalan maka bobotnya bisa berarti panjang jalan maupun batas kecepatan tertinggi pada jalan tertentu. Ekstensi lain pada graf adalah dengan membuat edgenya berarah, yang secara teknis disebut graf berarah atau digraf (directed graph). Digraf dengan edge berbobot disebut jaringan.  
Jaringan banyak digunakan pada cabang praktis teori graf yaitu analisis jaringan. Perlu dicatat bahwa pada analisis jaringan, definisi kata “jaringan” bisa berbeda, dan sering berarti graf sederhana (tanpa bobot dan arah).  
Suatu graph G dapat dinyatakan sebagai G = . Graph G terdiri atas himpunan V yang berisikan verteks/node pada graph tersebut dan himpunan dari E yang berisi edge pada graph tersebut. Himpunan E dinyatakan sebagai pasangan dari verteks yang ada dalam V. Sebagai contoh definisi dari graf pada gambar diatas adalah : V = {1,2,3,4,5,6} dan E = {(1,2),(1,5),(2,3),(3,4),(4,5),(5,2),(4,6)}

Gambar dengan node yang sama dengan yang diatas, tapi merupakan digraf.  
Pada digraf maka pasangan-pasangan ini merupakan pasangan terurut. Untuk menyatakan digraf (gambar kedua yang menggunakan tanda panah) kita dapat menggunakan himpunan edge sebagai berikut :  
E = { , , , , , , }  
Dalam himpunan edge untuk digraf, urutan pasangan verteks menentukan arah dari edge tersebut.  
Dalam teori graf, formalisasi ini untuk memudahkan ketika nanti harus membahas terminologi selanjutnya yang berhubungan dengan graph. Beberapa terminologi berhubungan dengan teori graf :  
• Degree atau derajat dari suatu node, jumlah edge yang dimulai atau berakhir pada node tersebut. Node 5 berderajat 3. Node 1 berderajat 2.  
• Path suatu jalur yang ada pada graph, misalnya antara 1 dan 6 ada path  
• Cycle siklus ? path yang kembali melalui titik asal 2 kembali ke 2.  
• Tree merupakan salah satu jenis graf yang tidak mengandung cycle. Jika edge f dan a dalam digraf diatas dihilangkan, digraf tersebut menjadi sebuah tree. Jumlah edge dalam suatu tree adalah nV – 1. Dimana nV adalah jumlah vertex