BAB II

LANDASAN TEORI

Bab ini menjelaskan dasar teori yang digunakan dalam pengembangan perangkat lunak, dasar teori tersebut terdiri dari algoritma genetika. Pada bagian algoritma genetika dibahas mengenai representasi kromosom, seleksi kromosom induk, rekombinasi kromosom, mutasi kromosom, regenerasi populasi, parameter algoritma genetika.

2.1 Algoritma Genetika

Algoritma genetika ditemukan di Universitas Michigan, Amerika Serikat oleh John Holland (1975) melalui sebuah penelitian dan dipopulerkan oleh salah satu muridnya David Goldberg. Algoritma genetika adalah algoritma yang memanfaatkan proses seleksi alamiah yang dikenal dengan proses evolusi. Dalam proses evolusi, individu secara terus menerus mengalami perubahan gen untuk menyesuaikan diri dengan lingkungan hidupnya. ”Hanya individu-individu yang kuat yang mampu bertahan”. Proses seleksi alamiah ini melibatkan perubahan gen yang terjadi pada individu melalui proses perkembangbiakan. Dalam algoritma genetika ini, proses perkembangbiakan menjadi proses dasar yang menjadi perhatian utama, dengan dasar berpikir: ”Bagaimana mendapatkan keturunan yang lebih baik”.

Sebuah solusi yang dibangkitkan dalam algoritma genetika disebut sebagai kromosom, sedangkan kumpulan kromosom-kromosom tersebut disebut sebagai

populasi. Sebuah kromosom dibentuk dari komponen-komponen penyusun yang disebut sebagai gen dan nilainya dapat berupa bilangan numerik, biner, simbol ataupun karakter tergantung dari permasalahan yang ingin diselesaikan. Kromosom-kromosom tersebut akan berevolusi secara berkelanjutan yang disebut dengan generasi.

Dalam tiap generasi Kromosom-kromosom tersebut dievaluasi tingkat keberhasilan nilai solusinya terhadap masalah yang ingin diselesaikan (fungsi objektif) menggunakan ukuran yang disebut dengan *fitness*. Untuk memilih kromosom yang tetap dipertahankan untuk generasi selanjutnya dilakukan proses yang disebut dengan seleksi. Proses seleksi kromosom menggunakan konsep aturan evolusi yaitu kromosom yang mempunyai nilai *fitness* tinggi akan memiliki peluang lebih besar untuk terpilih lagi pada generasi selanjutnya.

Kromosom-kromosom baru yang disebut dengan *offspring*, dibentuk dengan cara melakukan perkawinan antar kromosom-kromosom dalam satu generasi yang disebut sebagai proses *crossover*. Jumlah kromosom dalam populasi yang mengalami *crossover* ditentukan oleh paramater yang disebut dengan *crossover rate*. Mekanisme perubahan susunan unsur penyusun mahkluk hidup akibat adanya faktor alam yang disebut dengan mutasi direpresentasikan sebagai proses berubahnya satu atau lebih nilai gen dalam kromosom dengan suatu nilai acak. Jumlah gen dalam populasi yang mengalami mutasi ditentukan oleh parameter yang dinamakan *mutation rate*. Setelah beberapa generasi akan dihasilkan Kromosom-kromosom yang nilai gen-gennya *konver*gen ke suatu nilai

tertentu yang merupakan solusi terbaik yang dihasilkan oleh algoritma genetika terhadap permasalahan yang ingin diselesaikan.

Ada banyak metoda atau teknik yang dapat digunakan ketika menerapkan algoritma genetika. Meskipun demikian, algoritma ini memiliki suatu struktur algoritma yang umum digunakan pada setiap implementasi.

1. Buat populasi random yang terdiri dari sejumlah n kromosom solusi.
2. Lakukan evaluasi nilai mutu setiap kromosom dalam populasi. Jika sudah memenuhi syarat tertentu proses dihentikan (terdapat suatu individu yang telah memiliki nilai *fitness* tertentu yang diharapkan atau evolusi telah mencapai suatu generasi maksimum yang diizinkan atau dalam beberapa generasi tertentu tidak ada peningkatan nilai *fitness* yang diharapkan), lanjutkan ke tahap 3 jika belum mendapatkan solusi yang diinginkan
3. Buat populasi baru melalui tahap-tahap sebagai berikut :
    a. Pilih 2 kromosom dari sebuah populasi, semakin baik nilai mutunya semakin besar kemungkinan kromosom tersebut terpilih.
    b. Lakukan operasi persilangan terhadap 2 kromosom yang terpilih. Hasil dari persilangan tersebut menjadi sebuah kromosom baru.
    c. Lakukan operasi mutasi terhadap kromosom baru tersebut.
    d. Masukan kromosom baru kedalam populasi.
4. Populasi baru menggantikan populasi lama dalam proses iterasi.
5. Kembali ke tahap 2.

2.2 Representasi Kromosom

Hal paling mendasar dalam algoritma genetika adalah penyandian kromosom. Penyandian kromosom adalah suatu teknik untuk menyatakan populasi awal sebagai kandidat solusi suatu masalah ke dalam suatu kromosom. Sebuah kromosom harus mengandung atau mewakili informasi mengenai solusi dari masalah yang akan diselesaikan .

2.3 Representasi Biner

Representasi biner adalah representasi paling sederhana dan paling umum. Pada representasi biner ini, setiap gen hanya bisa bernilai 0 atau 1. Representasi biner sangat mudah diimplementasikan dan memerlukan operator-operator rekombinasi dan mutasi yang sangat sederhana. Tetapi representasi biner ini biasanya digunakan untuk masalah-masalah yang sangat sederhana (tidak terlalu kompleks).

Kromosom X	1 0 1 1 0 0 0 1 1 1 0 0 0 1 1 1 0 0 0 1 1 0
Kromosom Y	0 0 1 1 0 1 0 1 0 1 1 1 1 1 0 0 0 0 0 0 0 0

Gambar 2.1 Kromosom Menggunakan Representasi Biner

2.4 Representasi Integer

Pada representasi ini, setiap gen bisa bernilai bilangan bulat (*integer*). Bilangan bulat dapat merepresentasikan jenis, atau kuantitas objek.

Kromosom X	3	2	4	7	5	11	9	27	1
Kromosom Y	5	9	1	4	27	6	7	14	2

Gambar 2.2 Kromosom Menggunakan Representasi Integer

2.5 Representasi Real

Permasalahan praktis di dunia nyata mungkin saja membutuhkan tingkat ketelitian sangat tinggi. Jika representasi biner maupun *integer* tidak bisa mencapai ketelitian yang diinginkan, kita bisa menggunakan representasi *real*.

Pada representasi biner, setiap gen hanya bernilai 0 atau 1. Pada representasi *integer*, setiap gen bisa bernilai bilangan bulat dalam interval [0,9]. Sedangkan pada representasi *real* setiap gen bisa bernilai real dalam interval [0,1].

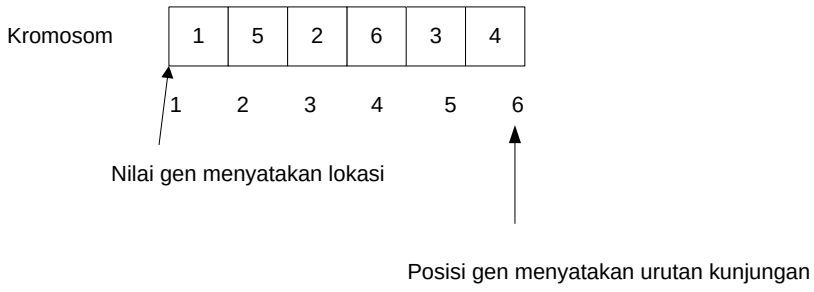
Kromosom X	0,234	0,456	1,000	6,732	5,543
Kromosom Y	1,234	2,000	0,789	0,987	1,000

Gambar 2.3 Kromosom Menggunakan Representasi Real

2.6 Representasi Permutasi

Untuk masalah tertentu, kita mungkin saja tidak bisa menggunakan representasi biner, *integer*, maupun *real*. Misalkan, masalah *Travelling Salesman Problem* (TSP). Pada TSP, masalahnya adalah mencari "urutan". Bagaimana menemukan urutan kunjungan lokasi (satu lokasi hanya dikunjungi satu kali) yang total nilainya paling optimal (bisa minimal atau maksimal bergantung tujuannya). "Nilai" disini bisa berupa jarak, kenyamanan, biaya, dan sebagainya. Bagi seorang kurir, misalnya tujuannya adalah menemukan urutan lokasi pengantaran paket yang total jaraknya paling minimal. Bagi seorang wisatawan, tujuannya bisa berupa urutan lokasi wisata yang tingkat kenyamanan jalannya paling maksimal.

Yang harus diperhatikan saat membangun kromosom dengan representasi permutasi adalah "satu kromosom harus menyatakan satu solusi". Dua kata kunci lain yang juga perlu kita perhatikan adalah "posisi" dan "nilai" gen. Posisi gen (indeks pada kromosom) bisa digunakan untuk menyatakan urutan kunjungan lokasi.



Gambar 2.4 Kromosom Menggunakan Representasi Permutasi

2.7 Seleksi Kromosom Induk

Seleksi adalah suatu operator algoritma genetika yang berfungsi memilih individu-individu yang akan dijadikan induk pada proses rekombinasi ataupun yang akan tetap bertahan pada generasi berikutnya. Ada beberapa metoda seleksi yang umum digunakan dalam algortima genetika, diantaranya:

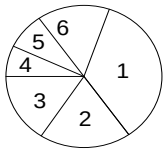
1. *Roulette Wheel Selection*

   Metode ini sama dengan metode yang diperlukan pada permainan *roulette*, dimana setiap angka dinyatakan dalam suatu area sektor dalam lingkaran. Kemungkinan suatu sektor terpilih sangatlah tergantung kepada area atau luas dari sektor tersebut. Dengan demikian, semakin luas sektor tersebut maka semakin berpeluang ia dipilih untuk dikawinkan ataupun bertahan hidup pada generasi selanjutnya. Hal ini sesuai dengan teori Darwin yang menyatakan bahwa individu yang lebih kuat atau lebih baik dapat lebih bertahan dan menghasilkan keturunan dalam proses evolusi.

   Pada permainan *roulette* semua sektor memiliki luas yang sama sehingga kemungkinan yang dimiliki oleh setiap sektor adalah sama. Pada metode algortima genetika ini, luas sektor akan sebanding dengan nilai *fitness* dari masing-masing kromosom.

   Seleksi dilakukan dengan cara memutar *roda roulette* sebanyak jumlah kromosom yang berada dalam populasi. Setiap kromosom yang terpilih sebagai orang tua dipindahkan dari populasi lama ke suatu tempat tertentu yang disebut sebagai *matting pool*. Di dalam *matting pool*, posisi – posisi

kromosom orang tua diacak. Selanjutnya, dua kromosom orang tua yang posisinya berurutan direkombinasikan untuk menghasilkan dua anak .



Gambar 2.5 Kromosom Dengan *Fitness* Lebih Besar Menempati Ruang Lebih Besar Dalam *Roda Roulette*

Gambar II.5 dapat dijadikan ilustrasi dalam memahami cara kerja metode seleksi *roulette wheel*. Dapat dilihat kromosom 1 memiliki nilai mutu yang paling besar karena luas bagian roda yang dimilikinya terbesar. Ada beberapa tahap dalam penggunaan metode ini, seperti dijelaskan melalui contoh di bawah ini.

Sebuah populasi terdiri dari 6 kromosom : A, B, C, D, E, dan F Nilai mutu dari masing-masing kromosom yaitu 40, 17, 20, 5, dan 8.

Akan dilakukan proses pemilihan sebuah kromosom melalui tahap-tahap berikut.

i. Hitung total *fitness* (F) seluruh kromosom dalam populasi

$$Tf = f(vk)\ 1 + f(vk)2..+f(vk)n \quad \text{...(2.1)}$$

Ket :

Tf : total fitness

$$Tf=40 + 17 + 20 + 5 + 10 + 8 = 100$$

ii. hitung probabilitas seleksi (pk) setiap kromosom

$$Pk(i)=\sum_{i=1}^{n} (f(vk)i/tf) \quad \text{...(2.2)}$$

Ket:

Pk: probabilitas seleksi

F(vk): nilai fitness

Tf: total fitness

Pk1= 40/100 Pk2=17/100 Pk3=20/100 Pk4=5/100 Pk5=10/100 Pk6=8/100

Pk1= 0,40 Pk2= 0,17 Pk3= 0,20 Pk4=0,05 Pk5=0,10 Pk6=0,08

iii. Hitung probabilitas kumulatif (qk) setiap kromosom

$$qk(j)=\sum_{i=1;j=1}^{n} qkj+pki \qquad \ldots(2.3)$$

Ket :
qk: probabilitas kumulatif
pk: probabiltas seleksi

qk1=0,40 qk2=0,57 qk3=0,77 qk4=0,82 qk5=0,92 qk6=1,00

iv. Buat 1 bilangan *random* acak untuk memilih satu kromosom induk
0,79 : kromosom D terpilih karena qk > 0,79

2. *Truncation-Random Selection*

Metoda ini lebih mudah diterapkan jika dibandingkan dengan metoda seleksi *roulette wheel*. Pemilihan kromosom dilakukan secara acak tetapi tidak semua kromosom mendapatkan kesempatan tersebut, hanya kromosom-kromosom terbaik saja yang berpeluang.

3. *Elitism Method*

Metode ini bertujuan mencegah hilangnya individu terbaik pada generasi berikutnya. Cara kerja elitism method adalah duplikasi n individu terbaik dari populasi saat ini ke generasi beriktunya. Metode ini menunjukan kinerja yang sangat baik pada algoritma genetika dalam mencari solusi terbaik.

4. Steady-state reproduction. Pada metode ini sejumlah *fitness parents* yang terburuk digantikan dengan sejumlah individu baru (*offspring*).

2.8 Rekombinasi Kromosom (*Crossover*)

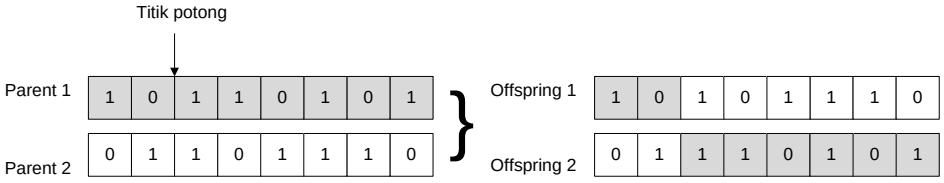
Persilangan dilakukan terhadap dua kromosom induk terpilih dengan tujuan untuk menciptakan kromosom anak baru. Tetapi, operasi rekombinasi tidak selamanya berhasil. Hal ini menirukan apa yang terjadi di dunia nyata bahwa tidak semua pasangan orang tua bisa memiliki anak.

2.9 Rekombinasi untuk Representasi Biner

Untuk representasi biner rekombinasi bisa dilakukan menggunakan tiga metode, yaitu: rekombinasi satu titik (*1-point crossover*), rekombinasi banyak titik (*n-point crossover*) dan rekombinasi seragam (*uniform crossover*).

1. Rekombinasi satu titik (*1-point crossover*)

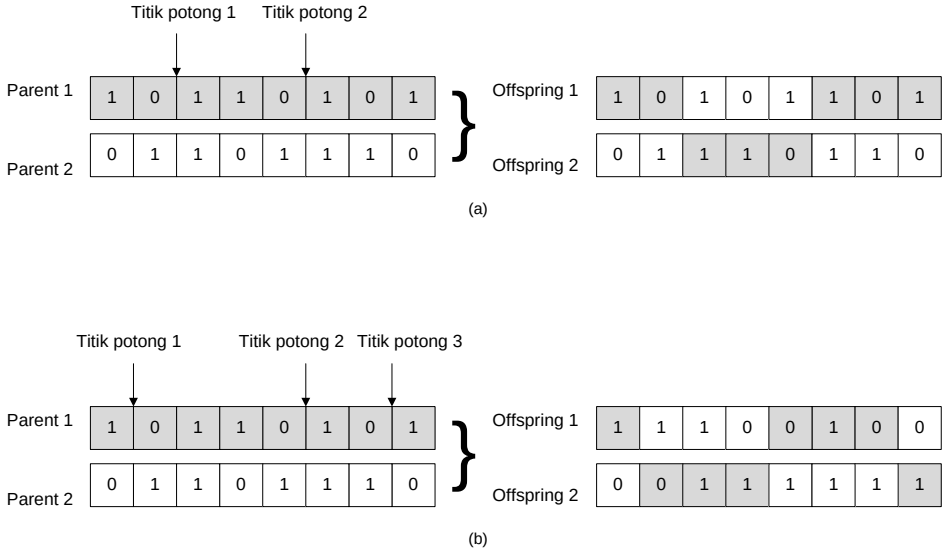
   Caranya, pilih titik potong secara acak pada posisi 1 hingga G-1, dalam hal ini G adalah jumlah gen dalam kromosom. Misalnya, terpilih titik potongnya adalah titik T. Selanjutnya, buat anak 1 dengan gen 1 sampai T diambil dari orang tua 1 dan gen T+1 sampai G diambil dari orang tua 2. Sebaliknya, anak 2 mendapatkan gen 1 sampai T dari orang tua 2 dan gen T+1 sampai G dari orang tua 1. Perhatikan gambar berikut.



Gambar 2.6 Rekombinasi Satu Titik Potong Pada Posisi 2

2. Rekombinasi banyak titik (*n-point crossover*).

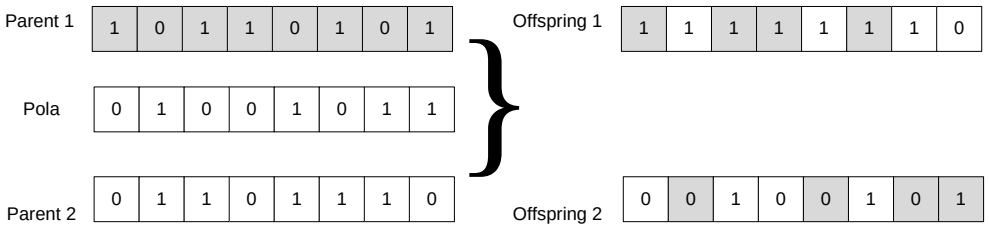
   Caranya, pilih beberapa titik potong secara acak pada posisi antara 1 hingga G-1, dalam hal ini G adalah jumlah gen dalam kromosom. Selanjutnya, pewarisan gen-gen dilakukan secara menyilang (zigzag) pada posisi – posisi tersebut. Perhatikan gambar berikut.



Gambar 2.7 Rekombinasi dua titik potong (a) dan tiga titik potong (b)

3. Rekombinasi seragam

   Rekombinasi ini bisa dilakukan dengan cara pelemparan mata uang logam. Lempar koin satu kali untuk setiap gen pada kromosom orang tua. Misalkan, jika lemparan koin menghasilkan gambar “garuda”, berarti anak 1 mendapatkan gen dari orang tua 1. Sebaliknya, jika lemparan koin menghasilkan gambar ”bunga ”, berarti anak 1 mendapatkan gen dari orang tua 2. Jika terdapat G gen berarti pelemparan koin dilakukan sebanyak G kali. Pelemparan koin sebanyak G kali bisa direpresentasikan sebagai suatu pola.

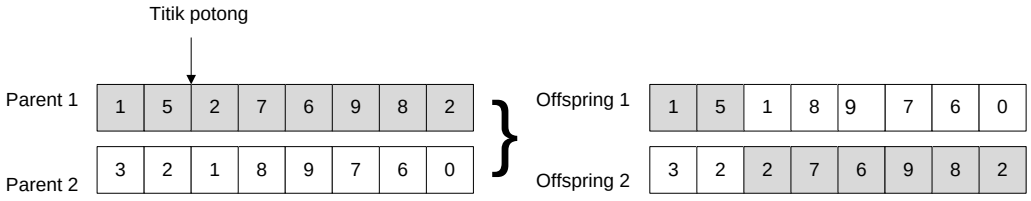


Gambar 2.8 Rekombinasi seragam pada representasi biner

2.10 Rekombinasi untuk Representasi Integer

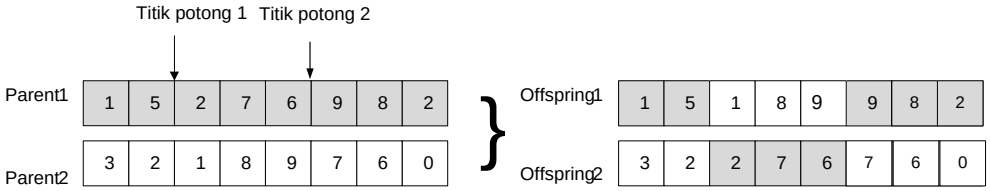
Untuk representasi *integer*, rekombinasi bisa dilakukan menggunakan tiga metode seperti pada representasi biner.

1. Rekombinasi satu titik (*1-point crossover*)



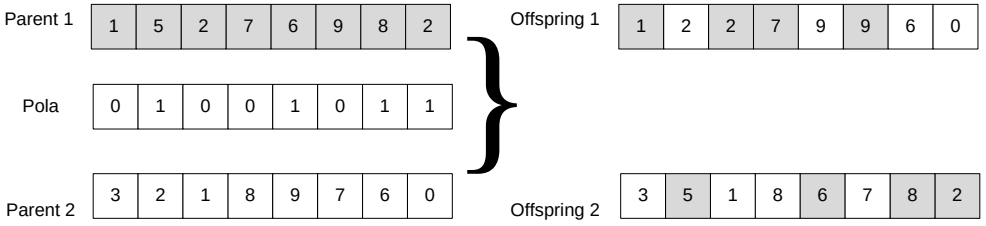
Gambar 2.9 Rekombinasi Satu Titik

2. Rekombinasi banyak titik (*n point crossover*)



Gambar 2.10 Rekombinasi Banyak Titik

3. Rekombinasi seragam

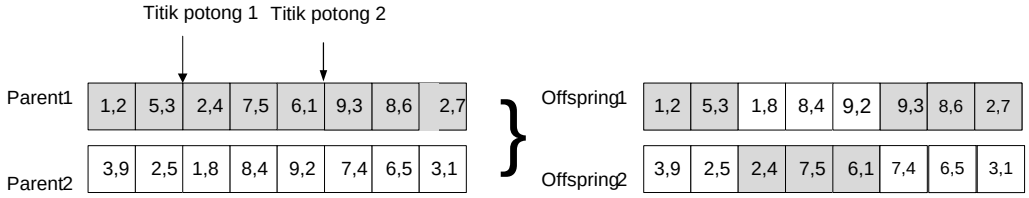


Gambar 2.11 Rekombinasi Seragam Pada Representasi Integer

2.11 Rekombinasi untuk Representasi Real

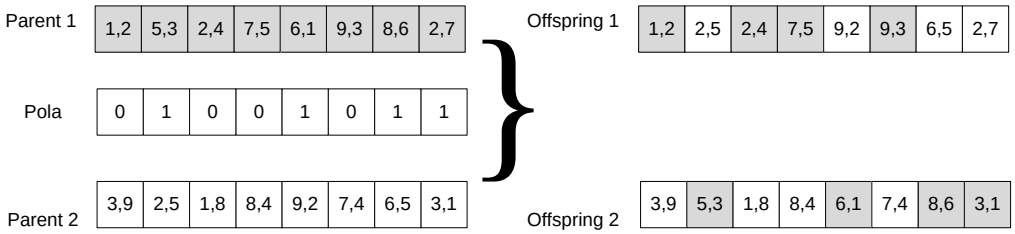
Untuk representasi *real*, rekombinasi bisa dilakukan dengan cara rekombinasi banyak titik atau rekombinasi seragam.

1. Rekombinasi banyak titik



Gambar 2.12 Rekombinasi Banyak Titik Pada Representasi Real

2. Rekombinasi seragam



Gambar 2.13 Rekombinasi Seragam Pada Representasi Real

2.12 Rekombinasi untuk Representasi Permutasi

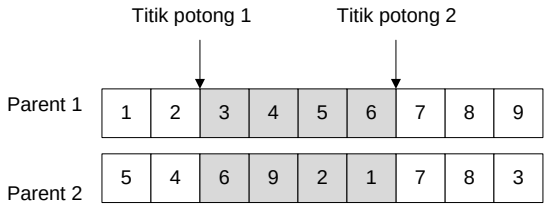
Representasi permutasi banyak digunakan dalam masalah penjadwalan seperti *traveling salesman problem* (TSP), *Job shop*, dll. Pada masalah TSP, representasi permutasi memanfaatkan posisi gen sebagai urutan kota. Hal ini menyebabkan metode-metode rekombinasi diatas tidak bisa digunakan untuk representasi ini.

Metode-metode tersebut tidak dapat digunakan untuk representasi permutasi karena ada kemungkinan anak-anak yang dihasilkan memiliki gen-gen yang tidak *valid* (dalam satu kromosom terdapat dua gen yang sama). Misalkan, ada dua gen yang bernilai sama dalam suatu kromosom yang berarti ada satu lokasi yang

dikunjungi dua kali. Padahal pada masalah TSP ada batasan bahwa setiap lokasi hanya boleh dikunjungi sebanyak satu kali.

Untuk menghasilkan anak-anak yang *valid* terdapat suatu metode rekombinasi, yaitu dengan menggunakan *partially mapped crossover.* Pada *partially mapped crossover* (PMX), rekombinasi dilakukan dengan cara mewariskan sebagian gen orang tua secara searah dan sebagian lainnya secara menyilang. Pewarisan sebagian gen secara menyilang dilakukan dengan memanfaatkan posisi-posisi gen kedua orang tua yang memiliki nilai sama untuk dilakukan pemetaan. Karena pewarisan gen yang menggunakan pemetaan dilakukan hanya pada sebagian gen, maka metode rekombinasi ini dinamakan *partially mapped crossover*. Gambar dibawah ini mengilustrasikan rekombinasi dengan metode *partially mapped crossover.*

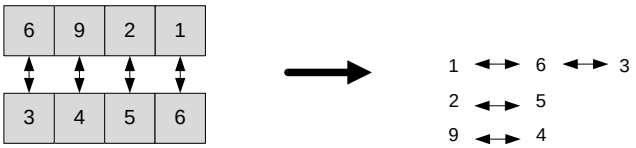
1. Pilih **segmen** kromosom dari kedua orangtua secara acak dengan cara membangkitkan dua titik, titik potong satu TP1 dan titik potong dua TP2.



2. Tukar **segmen** kedua orangtua untuk membuat *proto-child.*



3. Lakukan *mapping relationship.*



4. Buat offspring yang valid dengan *mapping relationship* dengan cara :



Gambar II.14 Ilustrasi rekombinasi PMX

a. Untuk membangun *offspring* 1

Langkah 1 :

Untuk gen-gen yang terdapat di dalam **segmen** pada *protochild*-1 tidak perlu dilakukan pengecekan dan pemetaan.

Langkah 2 :

Lihat satu persatu gen – gen sebelum **segmen** pada *protochild*-1 (sebelum Tp1). Dimulai dari gen pertama dengan nilai 1, periksa apakah di dalam **segmen** pada *protochild*-1 terdapat gen bernilai 1. Jika pada **segmen** *protochild*-1 terdapat gen bernilai 1 maka, lakukan pemetaan. Dalam hal ini gen bernilai 1 dipetakan menjadi gen bernilai 6 (lihat langkah 3, lakukan *mapping relationship*). Setelah itu periksa kembali apakah di dalam **segmen** pada *protochild*-1 terdapat gen bernilai 6. Jika pada **segmen** *protochild*-1 terdapat gen bernilai 6 maka, lakukan pemetaan. Gen bernilai 6 tersebut dipetakan menjadi gen bernilai 3 (lihat langkah 3,

lakukan *mapping relationship*). Cek kembali apakah gen bernilai 3 terdapat di dalam **segmen** pada *protochild*-1. Hasilnya, gen bernilai 3 tidak ada di dalam **segmen** pada *protochild*-1, sehingga gen bernilai 3 tersebut dimasukan ke dalam gen pertama pada *offspring* 1.

Langkah 3 :

Periksa gen ke-dua yang bernilai 2 sebelum **segmen** pada *protochild*-1, lalu periksa apakah di dalam **segmen** pada *protochild*-1 terdapat gen bernilai 2. Jika pada **segmen** *protochild*-1 terdapat gen bernilai 2 maka, lakukan pemetaan. Gen bernilai 2 tersebut dipetakan menjadi gen bernilai 5 (lihat langkah 3, lakukan *mapping relationship*). Periksa kembali apakah gen bernilai 5 terdapat di dalam **segmen** pada *protochild*-1. Hasilnya, gen bernilai 5 tidak terdapat di dalam **segmen** *protochild*-1, sehingga gen bernilai 5 tersebut dimasukan ke dalam gen ke-dua pada *offspring* 1.

Langkah 4 :

Lihat satu persatu gen – gen sesudah **segmen** pada *proto-child*-1 (sesudah Tp2). Dimulai dari gen ke-tujuh dengan nilai 7, periksa apakah di dalam **segmen** pada *proto-child*-1 terdapat gen bernilai 7. Hasilnya, gen bernilai 7 tidak terdapat di dalam **segmen** *protochild*-1, sehingga gen bernilai 7 tersebut dimasukan ke dalam gen ke-tujuh pada *offspring* 1.

Langkah 5:

Dimulai dari gen ke-delapan dengan nilai 8, periksa apakah di dalam **segmen** pada *proto-child*-1 terdapat gen bernilai 8. Hasilnya, gen bernilai 8 tidak terdapat di dalam **segmen** *protochild*-1, sehingga gen bernilai 8 tersebut dimasukan ke dalam gen ke-delapan pada *offspring* 1.

Langkah 6 :

Periksa gen ke-sembilan yang bernilai 9 pada *protochild*-1, periksa apakah di dalam **segmen** pada *protochild*-1 terdapat gen bernilai 9. Jika pada **segmen** *protochild*-1 terdapat gen bernilai 9 maka, lakukan pemetaan. Gen bernilai 9 tersebut dipetakan menjadi gen bernilai 4 (lihat langkah 3). Periksa kembali apakah gen bernilai 4 terdapat di dalam **segmen** pada *protochild*-1. Hasilnya, gen bernilai 4 tidak terdapat di dalam **segmen** *protochild*-1, sehingga gen bernilai 4 tersebut dimasukan ke dalam gen ke-sembilan pada *offspring* 1

b. Untuk membangun *offspring* 2

Lakukan langkah yang sama untuk pembuatan *offspring*-2 seperti langkah pembuatan *offspring*-1

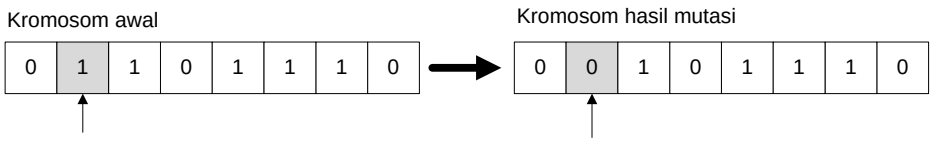
2.13 Mutasi Kromosom

Setelah tahap rekombinasi terhadap semua pasangan kromosom pada *matting pool* yang menghasilkan N (ukuran populasi) kromosom, maka GA menjalankan operator mutasi.

Ada satu hal di dunia nyata yang diadopsi oleh GA. Hal ini berkaitan dengan probabilitas terjadinya mutasi yang sangat kecil. Hal ini diadopsi GA dengan adanya variabel probabilitas mutasi yang dilambangkan dengan P*m* dan nilainya diset dengan sangat kecil. Biasanya P*m* berada dalam interval antara 1 dibagi jumlah gen dalam semua kromosom dalam populasi sampai dengan 1 dibagi julamh gen dalam satu kromosom.

2.14 Mutasi untuk Representasi Biner

Mutasi dilakukan secara sederhana dengan cara membalik nilai binernya. Gen yang bernilai biner 1 dimutasi menjadi 0. Sedangkan gen yang bernilai 0 dimutasi menjadi 1. Perhatikan ilustrasi berikut ini.



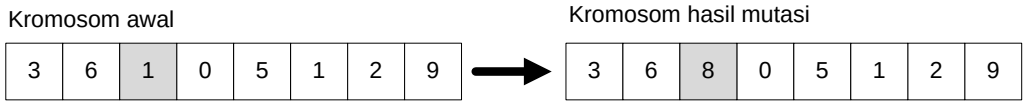
Gambar 2.15 Mutasi Untuk Representasi Biner

2.15 Mutasi untuk Representasi Integer

Mutasi gen untuk representasi integer bisa dilakukan dengan beberapa cara, diantaranya adalah:

1. Membalik nilai *integer*

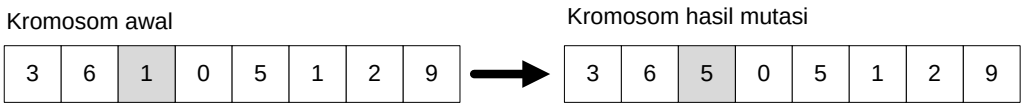
   Cara ini merupakan peluasan dari mutasi pada representasi biner. Sebagai contoh, jika nilai gen berada dalam interval $[0, 9]$, maka gen bernilai 0 dibalik menjadi 9, gen bernilai 1 dibalik menjadi 8, dan seterusnya. Perhatikan ilustrasi berikut.



Gambar 2.16 Mutasi Untuk Representasi Integer Menggunakan Pembalik Nilai Integer

2. Pemilihan nilai secara acak

   Suatu gen yang terpilih untuk dimutasi nilainya diganti dengan gen baru yang dibangkitkan secara acak dalam interval nilai-nilai gen yang diizinkan. Misalkan, jika nilai-nilai gen dalan interval $[0, 9]$, maka gen baru yang dibangkitkan secara acak juga berada dalam interval $[0, 9]$. Nilai gen baru yang dihasilkan bisa saja dibatasi dengan aturan harus berbeda dengan nilai gen lama [SUY08]. Perhatikan ilustrasi berikut.



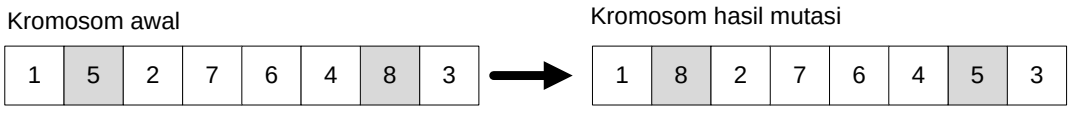
Gambar 2.17 Mutasi untuk representasi integer menggunakan pemilihan nilai secara acak.

2.16 Mutasi untuk Representasi Permutasi

Mutasi pada representasi permutasi harus menghasilkan kromosom yang *valid*. Sehingga, proses mutasi dilakukan dengan suatu cara tertentu yang menjamin kromosom hasil mutasi tetap *valid.* Ada banyak cara yang bisa digunakan, diantaranya adalah:

1. Mutasi pertukaran (*Swap Mutation*)

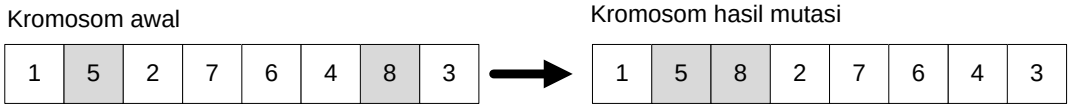
   Pertama, pilih dua posisi gen secara acak. Kemudian tukarakan gen pada kedua posisi tersebut. Perhatikan gambar berikut.



Gambar 2.18 Mutasi untuk representasi permutasi (*Swap mutation*)

2. Mutasi penyisipan (*Insert Mutation*)

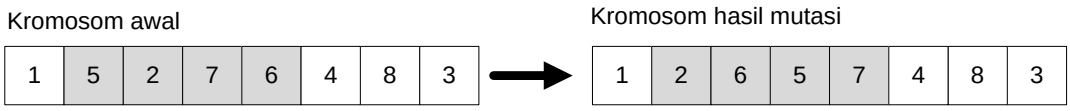
   Pertama, pilih dua posisi gen secara acak. Misalkan posisi P1 dan P2. Kemudian sisipkan gen posisi P2 setelah posisi P1. Perhatikan gambar berikut.



Gambar 2.19 Mutasi untuk representasi permutasi (insert *mutation*)

3. Mutasi pengacakan (*Scramble Mutation*)

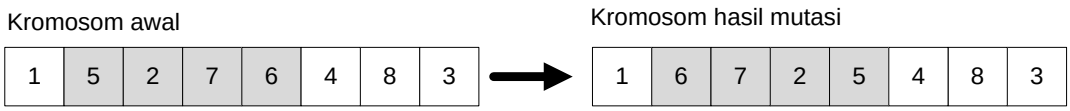
   Pertama, suatu segmen kromosom dengan memilih dua titik secara acak. Misalkan posisi T1 dan T2. Kemudian lakukan pengacakan posisi pada semua gen yang berada dalam segmen tersebut. Perhatikan gambar berikut.



Gambar 2.20 Mutasi Untuk Representasi Permutasi (*Scramble Mutation*)

4. Mutasi pembalikan (*Invertion Mutation*)

   Pertama, suatu segmen kromosom dengan memilih dua titik potong secara acak. Misalakan posisi T1 dan T2. Kemudian lakukan pembalikan posisi semua gen yang berada dalam segmen tersebut. Perhatikan gambar berikut.



Gambar 2.21 Mutasi Untuk Representasi Permutasi (*Inversion Mutation*)

2.17 Regenerasi Populasi

Regenerasi bertujuan untuk menghasilkan generasi populasi baru yang menggantikan generasi populasi sebelumnya dalam proses evolusi. Ada dua cara yang biasanya dilakukan, yaitu:

1. Kromosom – kromosom anak yang dihasilkan oleh persilangan dan mutasi, langsung menggantikan kromosom-kromosom terjelek dalam populasi.
2. Populasi baru dibentuk dari kromosom-kromosom yang dipilih dari gabungan kromosom-kromosom induk dan kromosom-kromosom anak. Proses pemilihannya dapat dilakukan dengan menggunakan metode seleksi *roulette whell* atau dipilih secara acak. Cara regenerasi ini biasanya juga menerapkan metode *elitis* untuk menghindari hilangnya kromosom terbaik akibat tidak terpilih saat regenerasi.

2.18 Parameter Algoritma Genetika

Ada 3 parameter yang umum digunakan dalam algoritma genetika, yaitu:

1. Ukuran Populasi

   Ukuran populasi menyatakan jumah kromosom yang ada dalam setiap generasi populasi. Dari generasi ke generasi, ukuran populasi harus dipertahankan tetap. Semakin besar ukuran populasi semakin besar pula domain pencarian yang berpotensi memberikan solusi yang diinginkan. Sisi buruk dalam hal tersebut antara lain adalah bahwa proses pencarian terjebak ke dalam kesia-siaan waku dan tenaga komputasi karena ada kemungkinan solusi teroptimal tidak berada dalam jelajah perluasan pencarian tersebut.

2. Probabilitas persilangan

   Parameter ini menyatakan perbandingan antara jumlah kromosom anak hasil persilangan dengan jumlah kromosom induk. Ukuran probabilitas persilangan yang besar akan memperbesar juga peluang dihasilkannya kromosom anak

yang memiliki nilai mutu yang lebih baik, tetapi jika terlalu besar akan memperlambat konvergensi proses pencarian.

3. Probabilitas mutasi

   Parameter ini menyatakan perbandingan antara jumlah gen kromosom anak yang termutasi dengan jumlah seluruh gen dari seluruh kromosom anak. Probabilitas mutasi yang terlau besar juga akan memperlambat konvergensi proses pencarian.

2.19 *Object Oriented Analysis and Design* (OOAD)

Object oriented analysis adalah metode analisis yang memeriksa keperluan (*requirements*) dari sudut pandang kelas-kelas dan objek-objek yang ditemui dalam ruang lingkup permasalahan [ASU[1]].

Object oriented design adalah metode untuk mengarahkan arsitektur *software* yang didasarkan pada manipulasi objek-objek sistem atau subsistem.

2.20 Objek

Objek (*object*) adalah benda, secara fisik ataupun konseptual. *Hardware, software,* dokumen dan manusia adalah beberapa contoh dari objek. Sebuah objek memiliki keadaan sesaat (*state*) dan prilaku (*behavior*) [ASU[1]].

State adalah kondisi objek yang menggambarkan objek tersebut, sedangkan *behaviour* adalah suatu definisi tindakan dan reaksi suatu objek.

2.21 Kelas

Kelas (*class*) adalah definisi umum untuk himpunan objek sejenis. Kelas menetapkan spesifikasi prilaku (*behavior*) dan atribut objek-objek tersebut. *Class*

adalah abstraksi dari entitas dalam dunia nyata. Objek adalah contoh (*instance*) dari sebuah kelas [ASU[1]].

2.22 UML (*Unified Modeling Language*)

UML merupakan model yang dapat digunakan secara luas dalam pemodelan bisnis, pemodelan perangkat lunak dari semua fase pembentukan dan semua tipe sistem, dan pemodelan secara umum dari berbagai pembentukan / konstruksi yang memiliki dua perilaku yaitu baik statis maupun dinamis [SOF[10]].

2.23 Sejarah UML

UML dimulai secara resmi pada oktober 1994, ketika Rumbaugh bergabung dengan Booch pada Relational Software Corporation. Proyek ini memfokuskan pada penyatuan metode Booch dan OMT. UML versi 0.8 merupakan metode penyatuan yang dirilis pada bulan Oktober 1995. Dalam waktu yang sama, Jacobson bergabung dengan Relational dan cakupan dari UML semakin luas sampai diluar perusahaan OOSE. Dokumentasi UML versi 0.9 akhirnya dirilis pada bulan Juni 1996. Meskipun pada tahun 1996 ini melihat dan menerima feedback dari komunitas. Dalam waktu tersebut, menjadi lebih jelas bahwa beberapa organisasi perangkat lunak melihat UML sebagai strategi dari bisnisnya. Kemudian dibangunlah UML Consortium dengan beberapa organisasi yang akan menyumbangkan sumber dayanya untuk bekerja, mengembangkan, dan melengkapi UML [SOF[10]].

Di sini beberapa perusahaan yang berkontribusi pada UML 1.0, diantaranya Digital Equipment Corporation, Hewlett-Packard, I-Logix, Intellicorp, IBM, ICON Computing, MCI Systemhouse, Microsoft, Oracle, Relational, Texas Instruments dan Unisys. Dari kolaborasi ini dihasilkan UML 1.0 yang merupakan bahasa pemodelan yang ditetapkan secara baik, kuat, dan cocok untuk lingkungan masalah yang luas. UML 1.0 ditawarkan menjadi standarisasi dari Object Management Group (OMG). Dan pada Januari 1997 dijadikan sebagai standar bahasa pemodelan.

Antara Januari–Juli 1997 gabungan group tersebut memperluas kontribusinya sebagai hasil respon dari OMG dengan memasukkan Adersen Consulting, Ericsson, ObjectTimeLimeted, Platinum Technology, Ptech, Reich Technologies, Softeam, Sterling Software dan Taskon. Revisi dari versi UML (versi 1.1) ditawarkan kepada OMG sebagai standarisasi pada bulan Juli 1997. Dan pada bulan September 1997, versi ini dierima oleh OMG Analysis dan Design Task Force (ADTF) dan OMG ArchitectureBoard. Dan Akhirnya pada Juli 1997 UML versi 1.1 menjadi standarisasi.

Pemeliharaan UML terus dipegang oleh OMG Revision Task Force (RTF) yang dipimpin oleh Cris Kobryn. RTP merilis editorial dari UML 1.2 pada Juni 1998. Dan pada tahun 1998 RTF juga merilis UML 1.3 disertai dengan user guide dan memberikan technical cleanup.

2.24 Tujuan Penggunaan UML

Memodelkan suatu sistem (bukan hanya perangkat lunak) yang menggunakan konsep berorientasi *object*. Dan menciptakan suatu bahasa pemodelan yang dapat digunakan baik oleh manusia maupun mesin [SOF[10]].

2.25 Gambaran dari UML

UML dapat dimanfaatkan untuk berbagai macam kebutuhan, diantaranya untuk melakukan pemodelan, visualisasi, spesifikasi, konstruksi dan dokumentasi.

2.26 UML sebagai Bahasa Pemodelan

UML merupakan bahasa pemodelan yang memiliki pembendaharaan kata dan cara untuk mempresentasikan secara fokus pada konseptual dan fisik dari suatu sistem. Contoh untuk sistem *software* yang *intensive* membutuhkan bahasa yang menunjukkan pandangan yang berbeda dari arsitektur sistem, ini sama seperti menyusun/mengembangkan *software development life cycle*. Dengan UML akan memberitahukan kita bagaimana untuk membuat dan membaca bentuk model yang baik, tetapi UML tidak dapat memberitahukan model apa yang akan dibangun dan kapan akan membangun model tersebut. Ini merupakan aturan dalam *software development process*.

2.27 UML sebagai bahasa Visualizing

UML tidak hanya merupakan rangkaian simbol grafikal, cukup dengan tiap simbol pada notasi UML merupakan penetapan semantik yang baik. Dengan cara ini, satu pengembang dapat menulis model UML dan pengembang lain atau perangkat yang sama lainnya dapat mengartikan bahwa model tersebut tidak

ambigu. Hal ini akan mengurangi error yang terjadi karena perbedaan bahasa dalam komunikasi model konseptual dengan model lainnya. UML menggambarkan model yang dapat dimengerti dan dipresentasikan ke dalam model tekstual bahasa pemograman.

Dengan model UML maka kita dapat memodelkan suatu sistem web tersebut dan direpresentasikan ke bahasa pemrograman. UML merupakan suatu model eksplisit yang menggambarkan komunikasi informasi pada sistem. Sehingga kita tidak kehilangan informasi *code* implementasi yang hilang dikarenakan *developer* memotong *coding* dari implementasi.

2.28 UML sebagai bahasa Specifying

Maksudnya membangun model yang sesuai, tidak ambigu dan lengkap. Pada faktanya UML menunjukan semua spesifikasi keputusan analisis, desain dan implementasi yang penting yang harus dibuat pada saat pengembangan dan penyebaran dari sistem *software* intensif.

2.29 UML sebagai bahasa Constructing

UML bukan bahasa pemograman visual, tetapi model UML dapat dikoneksikan secara langsung pada bahasa pemograman visual. Maksudnya membangun model yang dapat dipetakan ke bahasa pemograman seperti java, C++, VB atau tabel pada *database relational* atau penyimpanan tetap pada *database* berorientasi *object*.

2.30 UML sebagai bahasa Documenting

Maksudnya UML menunjukan dokumentasi dari arsitektur sistem dan detail dari semuanya. UML hanya memberikan bahasa untuk memperlihatkan permintaan dan untuk tes. UML menyediakan bahasa untuk memodelkan aktifitas dari perencanaan project dan manajemen pelepasan (*release management*).

2.31 Bagian-Bagian dari UML

Bagian-bagian utama dari UML adalah gambar, diagram, model element, dan *general mechanism.*

2.32 View

View digunakan untuk melihat sistem yang dimodelkan dari beberapa aspek yang berbeda. View bukan melihat grafik, tapi merupakan suatu abstraksi yang berisi sejumlah diagram. Beberapa jenis view dalam UML antara lain: use case view, logical view, component view, concurrency view, dan deployment view.

2.33 Use case view

Mendeskripsikan fungsionalitas sistem yang seharusnya dilakukan sesuai yang diinginkan *external actors*. *Actor* yang berinteraksi dengan sistem dapat berupa user atau sistem lainnya.

View ini digambarkan dalam *use case diagrams* dan kadang-kadang dengan *activity diagrams*. View ini digunakan terutama untuk pelanggan, perancang (*designer*), pengembang (*developer*), dan penguji sistem (*tester*).

2.34 Logical view

Mendeskripsikan bagaimana fungsionalitas dari sistem, struktur statis (*class*, *object*, dan *relationship*) dan kolaborasi dinamis yang terjadi ketika *object* mengirim pesan ke *object* lain dalam suatu fungsi tertentu. View ini digambarkan dalam *class diagrams* untuk struktur statis dan dalam *state*, *sequence*, *collaboration*, dan *activity diagram* untuk model dinamisnya. View ini digunakan untuk perancang (*designer*) dan pengembang (*developer*).

2.35 Component view

Mendeskripsikan implementasi dan ketergantungan modul. Komponen yang merupakan tipe lainnya dari *code module* diperlihatkan dengan struktur dan ketergantungannya juga alokasi sumber daya komponen dan informasi administrative lainnya. View ini digambarkan dalam component view dan digunakan untuk pengembang (*developer*).

2.36Concurrency view

Membagi sistem ke dalam proses dan prosesor. View ini digambarkan dalam diagram dinamis (*state*, *sequence*, *collaboration*, dan *activity diagrams*) dan diagram implementasi (*component* dan *deployment diagrams*) serta digunakan untuk pengembang (*developer*), pengintegrasi (*integrator*), dan penguji (*tester*).

2.37 Deployment view

Mendeskripsikan fisik dari sistem seperti komputer dan perangkat (*nodes*) dan bagaimana hubungannya dengan lainnya. View ini digambarkan dalam

deployment diagrams dan digunakan untuk pengembang (*developer*), pengintegrasi (*integrator*), dan penguji (*tester*).

2.38 Diagram

Diagram berbentuk grafik yang menunjukkan simbol elemen model yang disusun untuk mengilustrasikan bagian atau aspek tertentu dari sistem. Sebuah diagram merupakan bagian dari suatu gambar tertentu dan ketika digambarkan biasanya dialokasikan untuk gambar tertentu. Adapun jenis diagram antara lain :

2.39Use Case Diagram

Menggambarkan sejumlah actors dan hubungannya ke *usecase* yang diberikan oleh sistem. *Usecase* adalah deskripsi fungsi yang disediakan oleh sistem dalam bentuk teks sebagai dokumentasi dari use case symbol namun dapat juga dilakukan dalam *activity diagrams*. Use case digambarkan hanya yang dilihat dari luar oleh *actor* (keadaan lingkungan sistem yang dilihat user) dan bukan bagaimana fungsi yang ada di dalam sistem.

2.40 Class Diagram

Menggambarkan struktur statis *class* di dalam sistem. *Class* merepresentasikan sesuatu yang ditangani oleh sistem.

Class dapat berhubungan dengan yang lain melalui berbagai cara: *associated* (terhubung satu sama lain), *dependent* (satu class tergantung/menggunakan class yang lain), *specialed* (satu class merupakan spesialisasi dari class lainnya), atau *package* (grup bersama sebagai satu unit). Sebuah sistem biasanya mempunyai beberapa *class diagram*.

2.41 State Diagram

Menggambarkan semua *state* (kondisi) yang dimiliki oleh suatu *object* dari suatu *class* dan keadaan yang menyebabkan *state* berubah. Kejadian dapat berupa *object* lain yang mengirim pesan. *State class* tidak digambarkan untuk semua *class*, hanya yang mempunyai sejumlah *state* yang terdefinisi dengan baik dan kondisi *class* berubah oleh *state* yang berbeda.

2.42 Sequence Diagram

Menggambarkan kolaborasi dinamis antara sejumlah *object*. Kegunaanya untuk menunjukkan rangkaian pesan yang dikirim antara *object* juga interaksi antara *object*, sesuatu yang terjadi pada titik tertentu dalam eksekusi sistem [SOF[10]].

2.43 Collaboration Diagram

Menggambarkan kolaborasi dinamis seperti *sequence diagrams*. Dalam menunjukkan pertukaran pesan, *collaboration diagrams* menggambarkan *object* dan hubungannya (mengacu ke konteks). Jika penekannya pada waktu atau urutan gunakan *sequence diagrams*, tapi jika penekanannya pada konteks gunakan *collaboration diagram*.

2.44 Activity Diagram

Menggambarkan rangkaian aliran dari aktivitas, digunakan untuk mendeskripsikan aktifitas yang dibentuk dalam suatu operasi sehingga dapat juga digunakan untuk aktifitas lainnya seperti *use case* atau interaksi.

2.45 Component Diagram

Menggambarkan struktur fisik kode dari komponent. Komponent dapat berupa *source code*, komponent *biner*, atau *executable component*. Sebuah komponent berisi informasi tentang *logic class* atau *class* yang diimplementasikan sehingga membuat pemetaan dari *logical view* ke *component view*.

2.46 Deployment Diagram

Menggambarkan arsitektur fisik dari perangkat keras dan perangkat lunak sistem, menunjukkan hubungan komputer dengan perangkat (*nodes*) satu sama lain dan jenis hubungannya. Di dalam *nodes*, *executeable component* dan *object* yang dialokasikan untuk memperlihatkan unit perangkat lunak yang dieksekusi oleh node tertentu dan ketergantungan komponen.

2.47 Area Penggunaan UML

UML digunakan paling efektif pada domain seperti :

1. Sistem Informasi Perusahaan
2. Sistem Perbankan dan Perekonomian
3. Bidang Telekomunikasi
4. Bidang Transportasi
5. Bidang Penerbangan
6. Bidang Perdagangan
7. Bidang Pelayanan Elekronik
8. Bidang Pengetahuan

9. Bidang Pelayanan Berbasis Web Terdistribusi

Namun UML tidak terbatas untuk pemodelan *software*. Pada faktanya UML banyak untuk memodelkan sistem non *software* seperti:

1. Aliran kerja pada sistem perundangan.
2. Struktur dan kelakuan dari Sistem Kepedulian Kesehatan Pasien
3. Desain *hardware* dll.

2.48 Rational Rose

Rational Rose adalah perangkat lunak yang memiliki perangkat-perangkat pemodelan secara visual untuk membangun solusi dalam rekayasa perangkat lunak dan pemodelan bisnis.

Rational Rose juga dapat didefinsikan sebagai tools pemodelan visual untuk pengembangan sistem berbasis objek yang handal untuk digunakan sebagai bantuan bagi para pengembang dalam melakukan analisis dan perancangan sistem. *Rational rose* mendukung permodelan bisnis yang membantu para pengembang memahami sistem secara komprehensif. Ia juga membantu analisis sistem dengan cara pengembang membuat diagram use case untuk melihat fungsionalitas sistem secara keseluruhan sesuai dengan harapan dan keinginan pengguna. Kemudian, ia juga menuntut pengembang untuk mengembangkan *Interaction Diagram* untuk melihat bagaimana objek-objek saling bekerjasama dalam menyediakan fungsionalitas yang diperlukan.

Dalam *Rational rose*, pemodelan adalah cara melihat sistem dari berbagai sudut pandang. Ia mencakup semua diagram yang dikenal dalam UML, actor-aktor yang terlibat dalam sistem, use-case, objek-objek, kelas-kelas, komponen-komponen, serta simpul-simpul penyebaran. Model juga mendeskripsikan rincian yang diperlukan sistem dan bagaimana ia akan bekerja, sehingga para pengembang dapat menggunakan model itu sebagai blue print untuk sistem yang akan dikembangkan.

2.49 Fitur Rational Rose

Bahasa yang digunakan adalah bahasa pemodelan standar yaitu UML.

1. Rational Rose mendukung round-trip engineering sehingga kita dapat men-*generate* model kedalam kode (Java, C++, C#, Visual Basic, dan sebagainya) dan melakukan reverse engineering untuk menampilkan arsitektur sistem dari kode yang ada.
2. Model dan kode senantiasa singkron.
3. Mudah dalam memperbaiki perangkat lunak, karena kita dapat menggambarkan kembali arsitektur perangkat lunak tersebut dalam UML.
4. Tersedia Rational Rose untuk berbagai platform.
5. Memiliki Rose Web Publisher sehingga suatu tim dapat mengkomunikasikan model dan spesifikasinya dalam web browser.
6. Mendukung rekayasa perangkat lunak untuk sistem *client server*.

Jika mulai mengembangkan database dengan SQL Server 2005, Anda tentu perlu mengenal dengan baik komponen-komponen yang dimiliki SQL Server 2005,

aneka *feature* pengembangan baru yang ditambahkan, serta alat-alat pengembangan (development tools) yang disediakan. Itulah cara jitu dalam pengembangan database (database developtment) pada SQL server 2005.

Setelah menyoroti berbagai sisi administrasi database SQL server 2005, kini kita mencoba mengupas kiat-kiat jitu (the best practices) dalam pengembangan database. Kita mulai dengan mengenal SQL Server 2005 dari sisi kepentingan seorang pengembang (developer).

2.50 Komponen SQL Server 2005

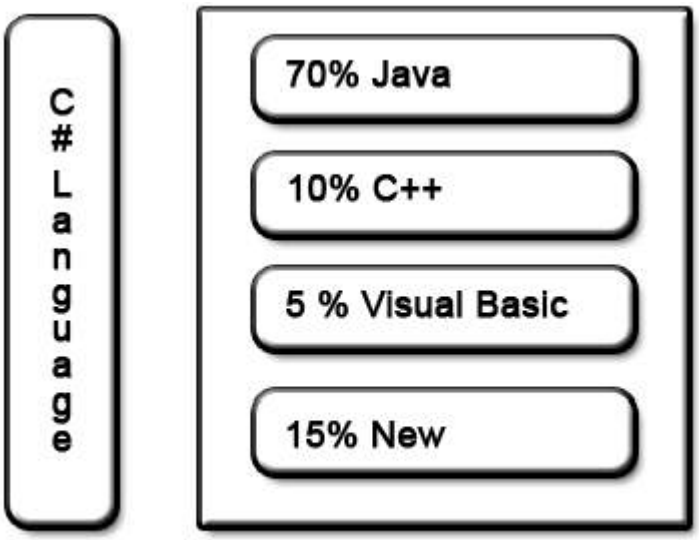
Tentu saja komponen-komponen SQL Server 2005 tidak berbeda, baik untuk kepentingan administrasi, maupun kepentingan pengembangan. Namun anda perlu selalu mengingatnya karena komponen-komponen inilah yang anda perlukan dalam membuat solusi *DATABASE*. Komponen-komponen ini adalah :

1. Relational Database Engine : komponen utama atau jantung SQL Server 2005.
2. Analysis Services : Basis dari solusi intelijen bisnis yang ampuh (*powerful*), dan mendukung aplikasi-aplikasi OLAP (online analytical *processing*), serta data minning.
3. *Data Transformation Service* (DTS): sebuah mesin untuk membuat solusi ekspor dan impor data, serta untuk mentransformasi data ketika data tersebut ditransfer.
4. Notification Services: sebuah framework untuk solusi dimana pelanggan akan dikirimi notifikasi ketika sebuah event muncul.
5. Reporting Services: service yang akan mengambil data dari SQL Server, dan menghasilkan laporan-laporan.

6. Service broker: sebuah mekanisme antrian yang akan menangani komunikasi berbasis pesan diantara service.
7. Native HTTP Support: dukungan yang memungkinkan SQL server 2005 yang (jika diinstall pada Windows Server 2003) akan merespon request terhadap HTTP endpoint, sehingga memungkinkan pembangunan sebuah web service untuk SQL Server tanpa menggunakan IIS.
8. SQL server Agent : akan mengotomatiskan perawatan database dan mengatur task, event dan alert.
9. .NET CLR (Common Language Runtime): akan memungkinkan pembuatan solusi menggunakan managed code yang ditulis dalam salah satu bahasa .NET.
10. Replication: serangkaian teknologi untuk menjalin dan mendistribusikan data dan obyek database dari sebuah database ke database lain, dan melakukan sinkronisasi untuk menjaga konsistensinya.
11. Full-Text Search: memungkinkan pengindeksan yang cepat dan flexibel untuk query berbasis kata kunci (terhadap data teks yang disimpan dalam database).

2.51 Pengertian C#

C# adalah bahasa pemograman berorientasi objek yang dikembangkan oleh microsoft sebagian dari . net dan sekarang sudah di akui sebagai standar oleh ECMA dan ISO komposisi bahsa C# adalah sebagai berikut



Gambar 2.22 Komposisi Bahasa C#

2.52 Tinjuan Tempat Penelitian

2.52.1 Sejarah Singkat Perusahaan

Peraturan daerah kota Bandung nomor 13 tahun 2007 tentang pembentukan dan susunan organisasi dinas daerah kota Bandung

Menimbangkan:

1) bahwa dengan telah ditetapkannya Peraturan Pemerintah Nomor 38 Tahun 2007 tentang Pembagian Urusan Pemerintahan Antara Pemerintah, Pemerintahan Daerah Provinsi, dan Pemerintahan Daerah Kabupaten/Kota dan Peraturan Pemerintah Nomor 41 Tahun 2007 tentang Organisasi Perangkat Daerah, maka dipandang perlu melakukan penyesuaian organisasi perangkat daerah;

2) bahwa dalam rangka penyesuaian peraturan terhadap perundang-undangan sebagaimana pada huruf a dan sekaligus sebagai pelaksanaan reformasi birokrasi di Lingkungan Pemerintah Kota Bandung serta upaya mendukung peningkatan pelayanan publik maka perlu dilakukan penyesuaian susunan organisasi Dinas Daerah Kota Bandung;

3) bahwa berdasarkan pertimbangan sebagaimana dimaksud dalam huruf a dan b, maka perlu membentuk Peraturan Daerah Kota Bandung tentang Pembentukan dan Susunan Organisasi Dinas Daerah Kota Bandung;

Mengingat :

Peraturan Daerah Kota Bandung Nomor 06 Tahun 2006 tentang Pemekaran dan Pembentukan Wilayah Kerja Kecamatan dan Kelurahan di Lingkungan Pemerintah Kota Bandung (Lembaran Daerah Tahun 2006 Nomor 06);

Pembentukan :

Dinas Kebudayaan dan Pariwisata Kota Bandung.

2.52.2 Tugas Pokok

1) Dinas Kebudayaan dan Pariwisata mempunyai tugas pokok melaksanakan sebagian urusan Daerah di bidang kebudayaan dan pariwisata.

2) Untuk melaksanakan tugas pokok sebagaimana dimaksud pada ayat (1) Pasal ini, Dinas Kebudayaan dan Pariwisata mempunyai fungsi :

a. perumusan kebijakan teknis bidang kebudayaan dan pariwisata;

b. penyelenggaraan sebagian urusan pemerintahan dan pelayanan umum di bidang kebudayaan dan pariwisata;

c. pembinaan dan pelaksanaan tugas di bidang kebudayaan dan pariwisata yang meliputi kebudayaan dan kesenian, sarana wisata, objek wisata dan pemasaran wisata;

d. pelaksanaan pelayanan teknis ketatausahaan Dinas;

e. pelaksanaan tugas lain yang diberikan oleh Walikota sesuai dengan tugas dan fungsinya.

2.52.3 Struktur Organisasi

dalam strukur organisasi dinas kebudayaan dan pariwisata kota Bandung yang di bentuk di tahun 2007 terdiri dari:

a. Kepala Dinas;

b. Sekretariat, membawahkan :

1. Sub Bagian Umum dan Kepegawaian;

2. Sub Bagian Keuangan dan Program.

c. Bidang Kebudayaan dan Kesenian, membawahkan :

1. Seksi Kebudayaan;

2. Seksi Kesenian.

d. Bidang Sarana Wisata, membawahkan :

1. Seksi Pembinaan Sarana Wisata;

2. Seksi Pengembangan Sarana Wisata.

e. Bidang Objek Wisata, membawahkan :

1. Seksi Pembinaan Objek Wisata;

2. Seksi Pengembangan Objek Wisata.

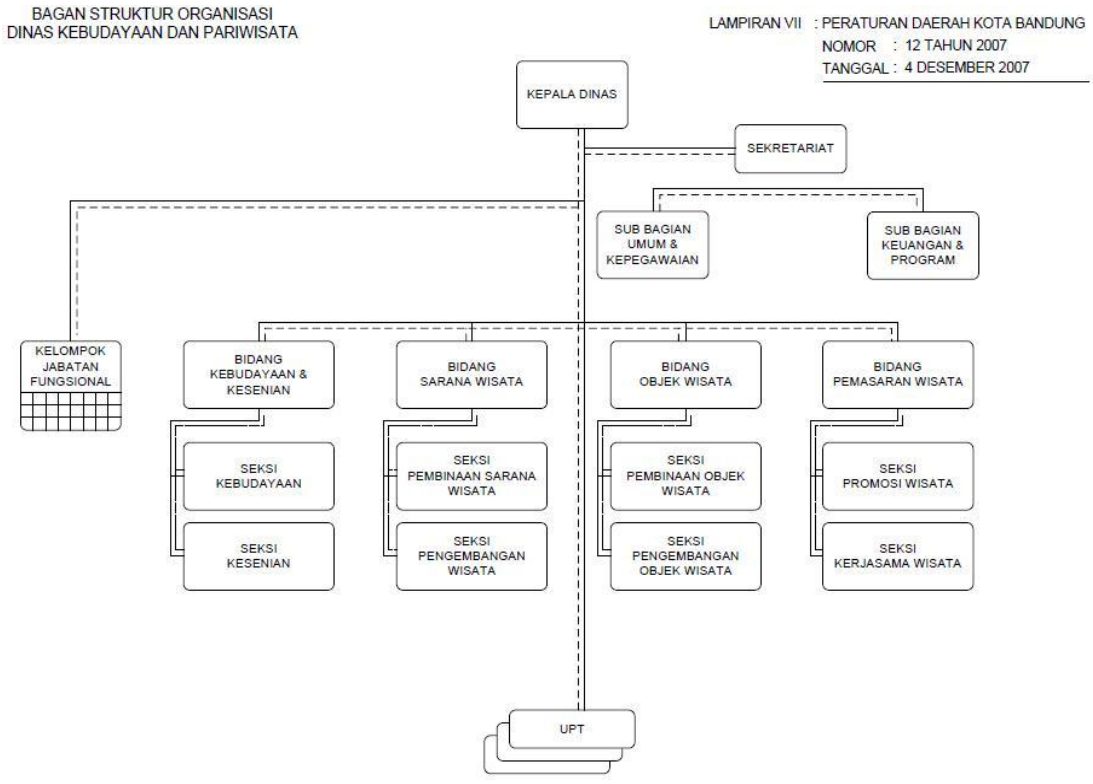
f. Bidang Pemasaran Pariwisata, membawahkan :

1. Seksi Promosi Pariwisata;

2. Seksi Kerjasama Wisata.

g. Unit Pelaksana Teknis Dinas;

h. Kelompok Jabatan Fungsional.



Gambar 2.23 Struktur Organisasi Dinas Kebudayaan Dan Pariwisata Kota Bandung

2.52.4 visi dan misi

visi dinas pariwisata dan kebudayaan kota bandung :

kota bandung sebagai kota tujuan wisata dan kota idaman

Misi Dinas Pariwisata dan Kebudayaan Kota Bandung :

1. Mendorong perwujudan Kota Bandung sebagai Kota Wisata dan Kota Seni Budaya;
2. Mendorong terwujudnya kondisi lingkungan yang kondusif untuk pengembangan kepariwisataan dan kebudayaan, serta pengembangan investasi;
3. Mendorong perwujudan potensi daerah dan masyarakat dengan memperkuat identiras kelokalan;
4. Meningkatkan Kualitas pelayanan dan kualitas aparatur;
5. Mendorong pengembangan kemitraan;
6. Mendorong promosi kepariwisataan kota bandung;
7. Mendorong peningkatan pemanfaatan iptek dan litbang