BAB II

TINJAUAN PUSTAKA

2.1. Pengertian Undang-undang

Undang-undang adalah salah satu macam hukum formal, Undang-undang yaitu suatu peraturan Negara yang mempunyai kekuatan hukum yang mengikat diadakan dan dipelihara oleh penguasa Negara. Undang-undang dalam arti formil, yaitu setiap keputusan pemerintah yang merupakan UU karena cara pembuatanya(misalnya,dibuat oleh pemerintah bersama-sama dengan parlemen). Sedangkan dalam arti material, yaitu setiap keputusan pemerintah yang menurut isinya mengikat setiap penduduk.

Syarat berlakunya Undang-undang ialah diundangkanya dalam Lembaran Negara (LN) terlebih dahulu oleh Mentri/Sekertaris Negara atau lebih tepatnya oleh Menkuhkam. Tujuanya agar setiap orang dapat mengetahui UU tersebut. Undang-undang tidak berlaku lagi jika kondisinya :

a. Jangka waktu berlakunya telah ditentukan UU itu sudah lampau.

b. Keadaaan atau hal untuk mana UU itu diadakan sudah tidak ada lagi.

c. UU itu dengan tegas dicabut oleh intansi yang membuat atau intansi yang lebih tinggi.

d. Telah ada UU yang baru yang isinya bertentangan atau berlainan dengan UU yang dulu berlaku.

Pada Undang-undang No.22 Tahun 2009 tentang Lalu Lintas dan Angkutan Jalan ini memiliki total 326 Pasal dan 22 BAB yang masing-masing merangkum sesuai kategori.

2.1.1. Undang-undang Lalu lintas

Undang-undang Lalu lintas adalah suatu sistem peraturan atau norma yang dikhususkan dibuat untuk mengatur aktivitas Lalu lintas. Mengutip dari UU No 22 tahun 2009 Pasal 3 tujuan dibuat atau diselenggarakanya Undang-undang Lalu lintas adalah :

a. Terwujudnya Pelayanan Lalu lintas dan angkutan jalan yang aman, selamat, tertib, lancar , dan terpadu dengan moda angkutan lain untuk mendorong perekonomian nasional, memajukan kesejahteraan umum, memperkukuh persatuan dan kesatuan bangsa, serta mampu menjungjung tinggi martabat bangsa;

b. Terwujudnya etika berlalu lintas dan budaya bangsa; dan

c. Terwujudnya pengegakan hukum dan kepastian hukum bagi masyarakat.

2.2. Pengertian Lalu lintas

Lalu lintas di dalam Undang-undang No 22 tahun 2009 didefinisikan sebagai gerak Kendaraan dan orang di Ruang Lalu Lintas Jalan, sedang yang dimaksud dengan Ruang Lalu Lintas Jalan adalah prasarana yang diperuntukkan bagi gerak pindah Kendaraan, orang, dan/atau barang yang berupa Jalan dan fasilitas pendukung. Permasalahan lalu lintas di kota – kota besar seperti Bandung cukup memprihatinkan. Pertumbuhan jumlah penduduk setiap tahunnya, secara tidak langsung membuat permintaan akan kebutuhan transportasi meningkat.

Ada tiga komponen terjadinya Lalu lintas yaitu manusia sebagai pengguna, kendaraan dan jalan yang saling berinteraksi dalam pergerakan kendaraan yang memenuhi persyaratan kelayakan dikemudikan oleh pengemudi mengikuti aturan Lalu lintas yang ditetapkan berdasarkan peraturan perundangan yang menyangkut Lalu lintas dan angkutan jalan melalui jalan yang memenuhi persyaratan geometric.

2.3. Konsep Dasar Algoritma

Algoritma merupakan urutan langkah-langkah dalam menentukan suatu masalah. Algoritma juga dapat didefinisikan dengan deretan langkah komputasi yang mentransformasikan masukan menjadi keluaran.

2.3.1. Sejarah Algoritma

Ditinjau dari asal-usul katanya, Algoritma mempunyai sejarah yang unik. Orang hanya menemukan kata *algorism* yang berarti proses menghitung dengan angka arab. Seseorang dikatakan *algoririst* jika orang tersebut menghitung menggunakan angka arab. Para ahli bahasa berusaha menemukan asal kata ini, namun hasilnya kurang memuaskan.

Akhirnya para ahli sejarah matematika menemukan asal kata tersebut yang berasal dari nama penulis buku arab yang terkenal yaituh Abu Ja'far Muhammad Ibnu MusaAl-Khuwarizmi. Al-Khuwarizmi dibaca orang barat menjadi Algorism. Al-Khuwarizmi menulis buku yang berjudul Kitab Al Jabar WalMuqabala yang artinya "Buku pemugaran dan pengurangan" (*The book of restoration and reduction).* Dari judul buku tersebut diperoleh akar kata"*Aljabar*" (*Algebra).*

Perubahan dari kata *algorism* menjadi *algorithm* muncul karena kata *algorism* sering dikelirukan dengan *arithmetic,* sehingga akhiran –sm berubah menjadi –thm. Karena perhitungan dengan angka Arab sudah menjadi hal yang biasa, maka lambat laun kata *algorithm* berangsur-angsur dipakai sebagai metode perhitungan (komputasi) secara umum, sehingga kehilangan makna kata aslinya. Dalam bahasa Indonesia, kata *algorithm* diserap menjadi algoritma

2.3.2. Definisi Algoritma

Algoritma adalah urutan langkah-langkah logis penyelesaian masalah yang disusun secara sistematis dan logis. Kata logis merupakan kata kunci dalam algoritma. Langkah-langkah dalam algoritma harus logis dan harus dapat ditentukan bernilai salah atau benar. Dalam beberapa konteks, algoritma adalah spesifikasi urutan langkah untuk melakukan pekerjaan tertentu.

Pertimbangan dalam pemilihan algoritma adalah haruslah benar. Artinya algortima akan memberikan keluaran yang dikehendaki dari sejumlah masukan yang diberikan. Tidak peduli serumit apapun algoritma, jika memberikan keluaran yang salah, pastilah algoritma tersebut bukanlah algoritma yang baik.

Pertimbangan lain yang harus diperhatikan adalah seberapa baik hasil yang dicapai oleh algoritma tersebut. Hal ini penting terutama pada algoritma untuk menyelesaikan masalah yang memerlukan aproksimasi hasil. Algoritma yang baik harus mampu memeberikan hasil yang sedekat mungkin dengan nilai yang sebenarnya.

Selanjutnya adalah efisiensi algoritma. Efisiensi algoritma dapat ditinjau dari 2 hal yaitu efisiensi waktu dan memori. Meskipun algoritma memberikan keluaran yang benar, tetapi jika harus menunggu lama untuk mendapatkan keluaranya, algoritma tersebut bukanlah algoritma yang baik, setiap orang menginginkan keluaran yang cepat. Dalam kenyataanya, setiap orang bisa membuat algoritma yang berbeda untuk menyelesaikan suatu permasalahan, walaupun terjadi perbedaan dalam menyusun algoritma, tentunya diharapkan keluaran yang sama.

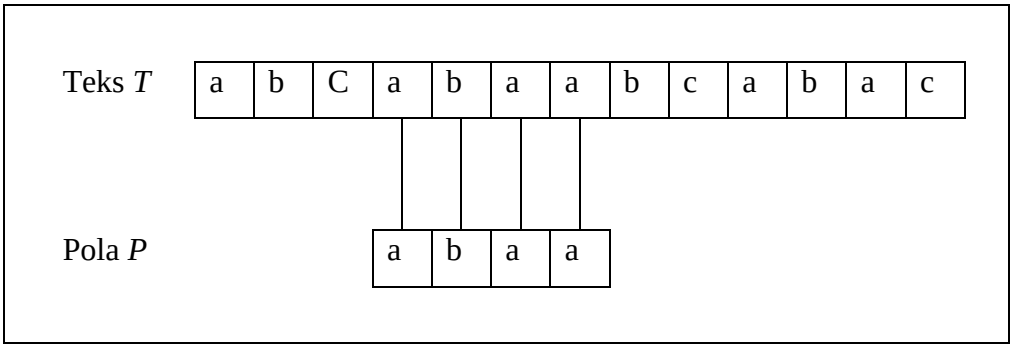
2.4. *String Maching*

Pola-pola khusus di dalam sebuah teks merupakan salah satu masalah yang sering timbul pada program-program pebaikan teks. Secara khusus, teks adalah sebuah dokumen yang sedang diperbaiki, dan pola pencarian adalah dunia khusus yang disediakan oleh pengguna. Algoritma-algoritma yang efisien untuk kasus tersebut sangat membantu pada daya respon program perbaikan teks. Salah satu contohnya, menggunakan algoritma pencocokan kata pada pola khusus rentetan DNA.

Berikut perumusan kasus pencocokan kata (***string-matching)***. Diasumsikan teks adalah sebuah susunan T[1..n] dengan panjang *n* dan memiliki susunan *P*[1..m] dengan panjang *m.* Lalu, elemen *P* dan *T* merupakan karakter-karakter yang digambarkan dari $\sum$ abjad terbatas. Contoh, diketahui $\sum = \{0,1\}$

atau $\sum$ = {a,b,…,z}. Sedangkan karakter susunan *P* and *T* dapat didefinisikan sebagai kata(***string***) pada banyak karakter.

Pola yang dimaksud adalah dimana ***P muncul dengan shift s*** dalam text T (atau , ekuivalen , pola ***P muncul pada permulaan di posisi s + 1*** pada teks *T*) Jika $0 \leq s \leq n - m$ dan $T[s+1..s+m] = P[1..m]$ (itu adalah, jika $T[s + j] = P[j]$, for $1 \leq j \leq m$). Jika *P* muncul dengan shift s di *T*, maka disebut s sebuah shift yang sah. Kasus pencocokan kata (*string-macthing*) merupakan masalah dari menemukan semua shift yang sah dengan pola pemberian P yang muncul di teks pemberian T.



Gambar 2.1 Kasus Pencocokan kata.

Tujuan pencocokan kata adalah untuk menemukan banyak kejadian pada pola *P* = abaa di teks *T* =*abcabaabcabac.* Pola tersebut muncul hanya sekali di teks, seperti shift s = 3. Shift s = 3 tersebut dikatakan sebuah shift yang sah. Tiap karakter pada pola dihubungkan dengan garis vertical ke pencocokan karakter pada teks, dan semua karakter yang sesuai ditampilkan bayangannya.

Walaupun memiliki kasus terburuk pada jalannya waktu $O((n - m + 1)m)$, Algoritma pencocokan kata bekerja sangat baik dengan rata-rata kecepatan dan kepraktisannya.

2.5. Algoritma Boyer-Moore

Algoritma Boyer-Moore adalah algoritma yang ditemukan oleh Bob Boyer dan J.Stroher Moore pada tahun 1977. Algortima Boyer-Moore akan menyimpan informasi pergeseran untuk melakukan pencarian *string*. Algoritma Boyer-Moore termasuk algoritma *string matching* yang paling efisien dibandingkan algoritma-algoritma *string matching* lainya. Karena sifatnya yang efisien, banayak

dikembangkan algoritma string matching dengan bertumpu pada konsep algoritma Boyer-Moore, beberapa diantaranya adalah Turbo BM dan algoritma *Quick Search*.

Algoritma Boyer-Moore menggunakan metode pencocokan *string* dari kanan ke kiri yaitu memindai karakter *pattern* dari kanan ke kiri mulai dari karakter paling kanan. Algoritma Boyer-Moore menggunakan dua fungsi *shift* yaitu *good shufix shift* (BmGs) dan *bad-caracter shift* (BmBc) untuk mengambil langkah berikutnya setelah terjadi ketidakcocokan antara karakter *pattern* dan karakter *teks* yang dicocokan.

Algoritma Boyer-Moore menggunakan dua buah tabel untuk mengolah informasi saat terjadi kegagalan pencocokan *string.* Table pertama akan menghitung banyaknya pergeseran yang harus dilakukan berdasarkan identitas karakter yang menyeabkan kegagalan pencocokan *string.* Jumlah pergeseran yang terdapat pada tabel pertama ini disebut *bad character shitf.* Tabel kedua juga akan menghitung banyaknya pergeseran yang harus dilakukan, tetapi berdasarkan jumlah karakter yang berhasil dicocokan sebelum pencocokan *string* tersebut gagal. Jumlah pergeseran yang dihitung oleh table kedua ini disebut dengan istilah *good suffix shift.*

Diasumsikan 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 u adalah akhiran dari *pattern* sebelum b dan v adalah 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 v dari 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:
   a. Karakter di *pattern* dan di teks yang dibandingkan tidak cocok (*mismatch*).
   b. 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.

Algoritma Boyer-Moore menggunakan dua buah tabel (BmGs & BmBc) untuk mengolah informasi saat terjadi kegagalan pencocokan string. Tabel pertama akan menghitung banyaknya pergeseran yang harus dilakukan berdasarkan identitas karakter yang menyebabkan kegagalan pencocokan string. Jumlah pergeseran yang terdapat pada tabel pertama ini disebut *bad character shitf*. Tabel kedua juga akan menghitung banyaknya pergeseran yang harus dilakukan, tetapi berdasarkan jumlah karakter yang berhasil dicocokan sebelum pencocokan string tersebut gagal. Jumlah pergeseran yang dihitung oleh tabel kedua ini disebut dengan istilah *good suffix shift*. Berikut ini contoh penjelasan proses pencocokan algoritma Boyer-Moore menggunakan gambar berikut :

1. Tahap pertama menentukan *pattern* atau kata yang akan dicari, kemudian membuat tabel ***bad-character shift.***

   Contoh *pattern :* MAKANAN

BmBc							
Index	0	1	2	3	4	5	6
Char	M	A	K	A	N	A	N
OH value	?	?	?	?	?	?	?

Gambar 2.2 Pencacahan character untuk BmBc

Sehingga nanti akan mendapatkan hasil seperti pada Tabel 2.1 yang untuk seterus nya akan di isi sesuai dengan nilai yang didapat dari hasil perhitungan.

BmBc	
Char	?
OH value	?

Gambar 2.3 Kondisi awal BmBc

BmBc = *Boyer-Moore Bad Character* **OH =** *Occurrence Heuristic*

2. Lakukan pencacahan disetiap karakter pada *pattern* seperti yang dilakukan pada tabel pertama. Lalu proses selanjutnya sebagai berikut:

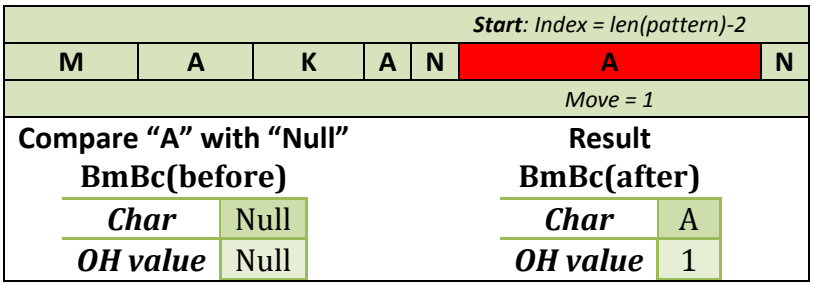
   1) Pencacahan dimulai dari karakter ke-2 paling kanan (index-Lenght(pattern) -2) dari pattern.

   2) Bandingkan setiap karakter yang dicacah terhadap BmBc, jika karakter yang dicacah tidak ditemukan dalam tabel tersebut, maka tambahkan karakter tersebut ke dalam BmBc dan OH *value* sama dengan jumlah pergeseran karakter yang dilakukan sebelumnya.

3) Lakukan langkah ke 2 kembali, dengan melakukan perpindahan 1 karakter ke kiri hingga mencapai karakter paling kiri (index=0) secara terus menerus.

4) Jika index=0, Cacah karakter paling kanan (index= Len(pattern) – 1) lalu bandingkan terhadap BmBc, lakukan hal yang sama jika karakter ditemukan sesuai dengan point 2.

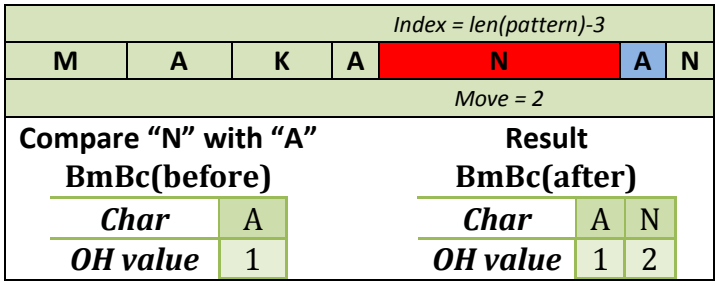
Process #1



Gambar 2.4 Proses 1 pencarian nilai OH pada BmBc.

Sesuai dengan point ke-1 proses yang paling awal yaitu melakukan pencacahan dari karakter ke-2 paling kanan (index-Lenght(pattern) -2) dari pattern, maka didapatkan character A dilanjutkan dengan point ke-2 dengan membandingkan karakter yang dicacah, sehubungan langkah pertama maka nilai move (pergeseran) ke satu maka masukan nilai 1 utuk character A pada tabel 2.3. BmBc.

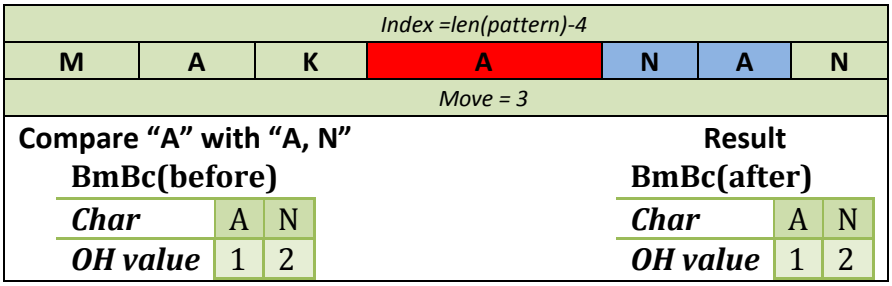
Process #2



Gambar 2.5 Proses 2 pencarian nilai OH pada BmBc

Melanjutkan proses pertama pada langkah ke-2 ini seperti yang telah dijelaskan pada point ke 3 yaitu Lakukan langkah ke 2 kembali sehubungan karakter M belum memiliki nilai maka masukan M dengan nilai OH 2,dilanjutkan dengan melakukan perpindahan 1 karakter ke kiri hingga mencapai karakter paling kiri (index=0) secara terus menerus.

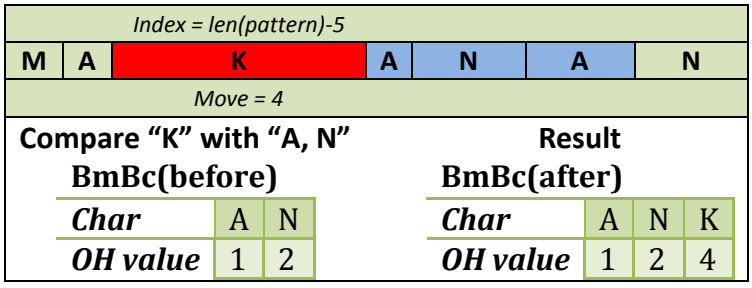
Process #3



Gambar 2.6 Proses 3 pencarian nilai OH pada BmBc

Melanjutkan proses ke-2 pada proses ke-3 ini seperti yang telah dijelaskan pada point ke 3 yaitu Lakukan langkah ke 2 kembali sehubungan karakter A sudah memiliki nilai OH maka dilanjutkan dengan melakukan perpindahan 1 karakter ke kiri hingga mencapai karakter paling kiri (index=0) secara terus menerus.

Process #4



Gambar 2.7 Proses 4 pencarian nilai OH pada BmBc

Melanjutkan proses ke-3 pada langkah ke-4 ini seperti yang telah dijelaskan pada point ke 3 yaitu Lakukan langkah ke 2 kembali sehubungan karakter N belum memiliki nilai maka masukan M dengan nilai OH 4,dilanjutkan dengan melakukan perpindahan 1 karakter ke kiri hingga mencapai karakter paling kiri (index=0) secara terus menerus.

Process #5

Index = len(pattern)-6

M	A	K	A	N	A	N

Move = 5

Compare "A" with "A, N, K"

BmBc(before)

Char	A	N	K
OH value	1	2	4

Result

BmBc(after)

Char	A	N	K
OH value	1	2	4

Gambar 2.8 Proses 5 pencarian nilai OH pada BmBc

Melanjutkan proses ke-4 pada proses ke-5 ini seperti yang telah dijelaskan pada point ke 3 yaitu Lakukan langkah ke 2 kembali sehubungan karakter A sudah memiliki nilai OH maka dilanjutkan dengan melakukan perpindahan 1 karakter ke kiri hingga mencapai karakter paling kiri (index=0) secara terus menerus.

Process #6

Index = len(pattern)-1

M	A	N	A	M	A	N

Move = 6

Compare "M" with "A, N, K"

BmBc(before)

Char	A	N	K
OH value	1	2	4

Result

BmBc(after)

Char	A	N	K	M
OH value	1	2	4	6

Gambar 2.9 Proses 6 pencarian nilai OH pada BmBc

Melanjutkan proses ke-5 pada proses ke-6 ini seperti yang telah dijelaskan pada point ke 3 yaitu Lakukan langkah ke 2 kembali sehubungan karakter M sudah memiliki nilai OH maka dilanjutkan dengan melakukan perpindahan 1 karakter ke kiri hingga mencapai karakter paling kiri (index=0) secara terus menerus.

Process #7

						***END**: Index = (len(pattern)-1)*
M	**A**	**K**	**A**	**N**	**A**	**N**
						Move = 7

Compare "N" with "A, M, N"

BmBc(before)

Char	A	N	K	M
OH value	1	2	4	6

Result

BmBc(after)

Char	A	N	K	M
OH value	1	2	4	6

Gambar 2.10 Proses 7 pencarian nilai OH pada BmBc

Melanjutkan proses ke-6 pada proses ke-7 ini merupakan proses terakhir berhubung pada proses ke-6 telah mencapai index=0 maka lakukan point 4 yaitu Jika index=0, Cacah karakter paling kanan (index= Len(pattern) – 1) lalu bandingkan terhadap BmBc, lakukan hal yang sama jika karakter ditemukan sesuai dengan point 2.

Setelah selesai melakukan pecacahan dan pembandingan karakter untuk mencari nilai OH padal tabel BmBc maka didapatkan nilai dari hasil perhitungan tersebut seperti yang tertera pada Gambar 2.10 berikut ini :

Result

BmBc

Char	A	N	K	M
OH value	1	2	4	6

Gambar 2.11 Hasil akhir nilai BmBc

3. Setelah mendapatkan hasil akhir pada tabel BmBc maka dilanjutkan membuat tabel akhiran / *suffix* dari *pattern.*

Pattern: MAKANAN (kanan ke kiri)			
Idx	**Prefix**	**Suffix**	**Len**
6	N	MAKANA	6
5	AN	MAKAN	5
4	NAN	MAKA	4
3	ANAN	MAK	3
2	KANAN	MA	2
1	AKANAN	M	1
0	MAKANAN	*NULL*	0

Pattern: MAKANAN (kiri ke kanan)			
Idx	**Prefix**	**Suffix**	**Len**
0	M	AKANAN	6
1	MA	KANAN	5
2	MAK	ANAN	4
3	MAKA	NAN	3
4	MAKAN	AN	2
5	MAKANA	N	1
6	MAKANAN	*NULL*	0

Gambar 2.12 *Suffix* kanan ke kiri & *Suffix* kiri ke kanan

Pembuatan tabel *suffix* dari kanan ke kiri berfungsi untuk dijadikan *suffix* comparator pada Tabel *Suffix* sedangkan pembuatan *suffix* dari kiri ke kanan berfungsi sebagai *suffix* yang nanti akan dibandingkan dan pencarian nilai MH.

		Suffix							
Index		0	1	2	3	4	5	6	
Prefix		**M**	**A**	**K**	**A**	**N**	**A**	**N**	***Move***
Suffix		**AKANAN**	**KANAN**	**ANAN**	**NAN**	**AN**	**N**	*null*	
Suffix comparator	MAKANA								**1**
	MAKAN								**2**
	MAKA								**3**
	MAK								**4**
	MA								**5**
	M								**6**
	null								**7**
MH value		**?**	**?**	**?**	**?**	**?**	**?**	**?**	

Gambar 2.13 Suffix lengkap beserta suffix comparator

4. Kemudian lakukan pemecahan akhiran pada *suffix* terhadap Suffix comparator Tabel. Berikan nilai MH pada tabel BmGs dengan *moving long* yang berkesesuaian.

    1) Cacah *suffix* pada masing-masing index terhadap seluruh *suffix comparator*

    2) Jika *length* kedua *suffix* tidak sama panjang, potong salah satu suffix yang memiliki length terbesar, besaran pemotongan suffix

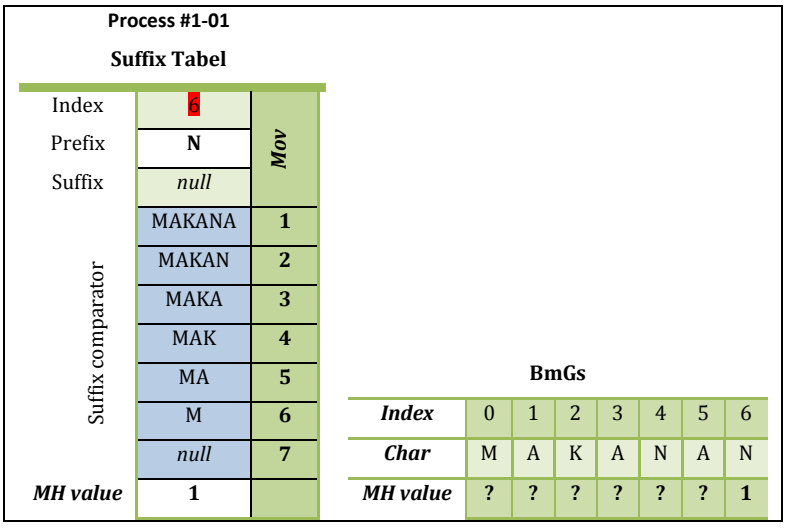
sesuai dengan nilai length terkecil kearah paling kanan karakter. Jika ditemukan kecocokan antar suffix, bandingkan prefix kedua suffix jika masing-masing suffix memiliki prefix. Jika prefix yang diperbandingkan adalah sama, maka kecocokan tidak dapat diterima.

3) Lakukan point 2 hingga index suffix mencapai 0.

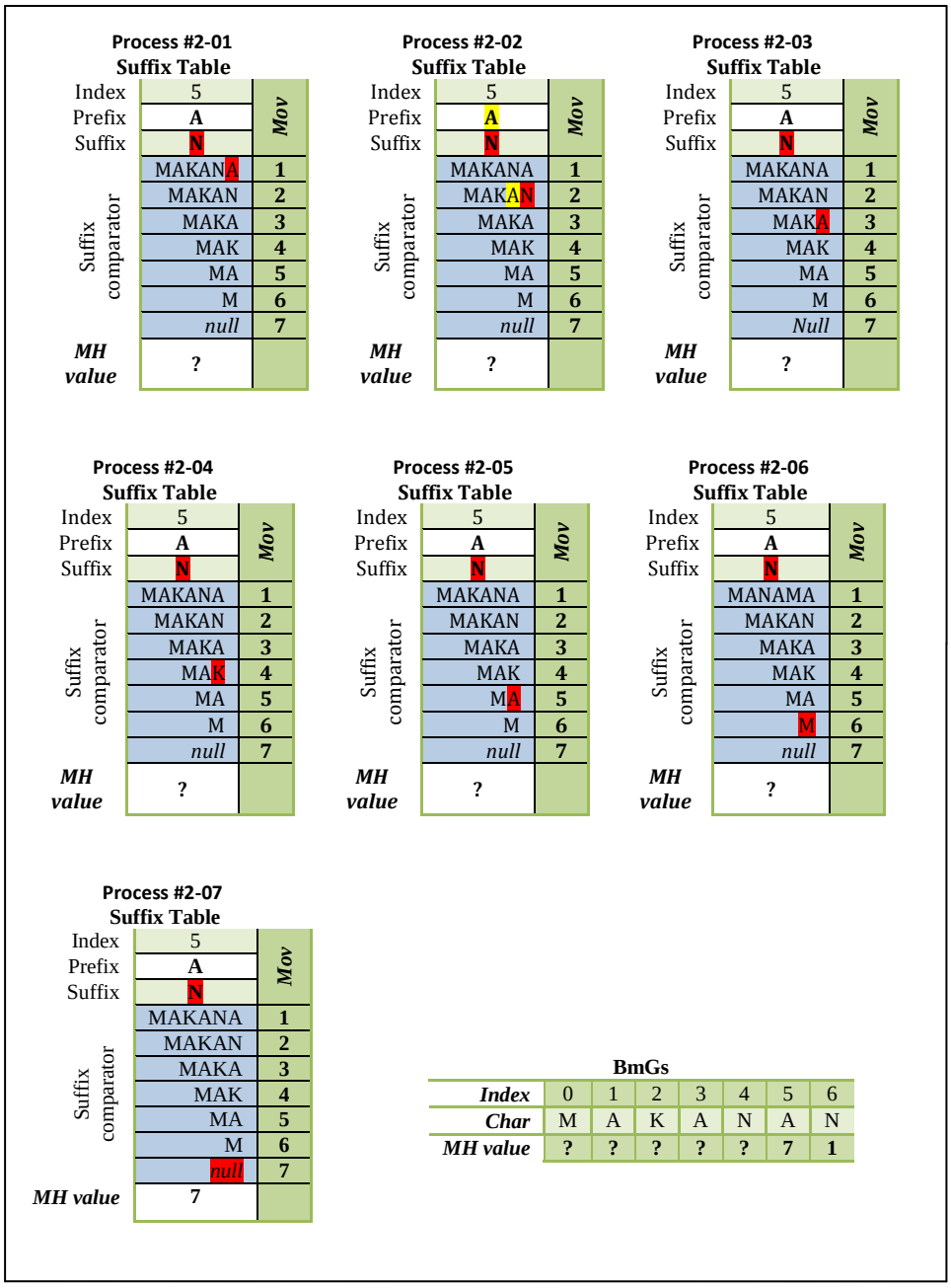
4) Untuk index suffix terbesar, secara default akan diberikan nilai 1 pada MH Value, sedangkan yang lainnya diberikan nilai sesuai dengan nilai move yang diraih.

BmGs							
Index	0	1	2	3	4	5	6
Char	M	A	K	A	N	A	N
MH value	**?**	**?**	**?**	**?**	**?**	**?**	**?**

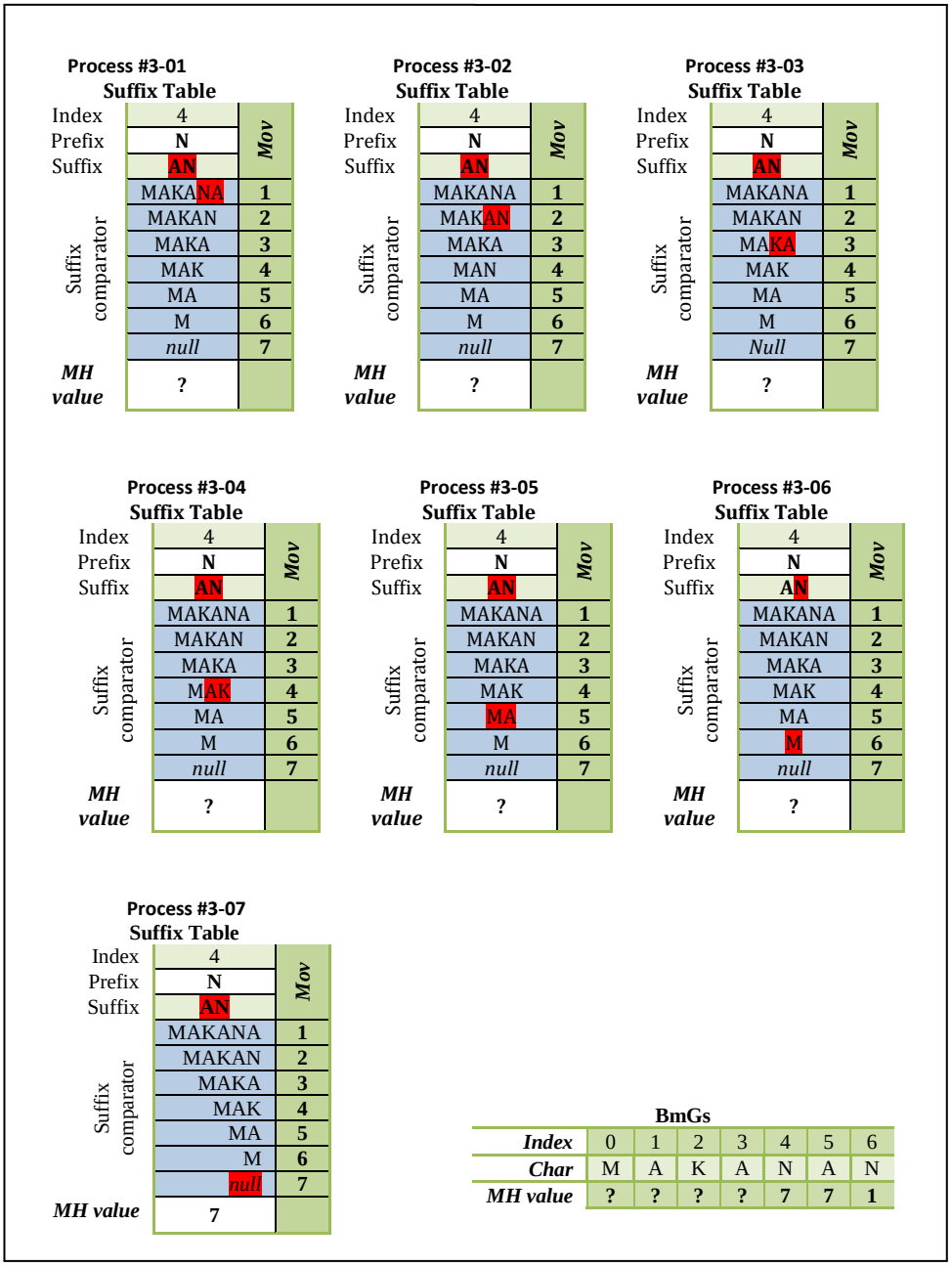
Gambar 2.14 Tabel BmGs awal



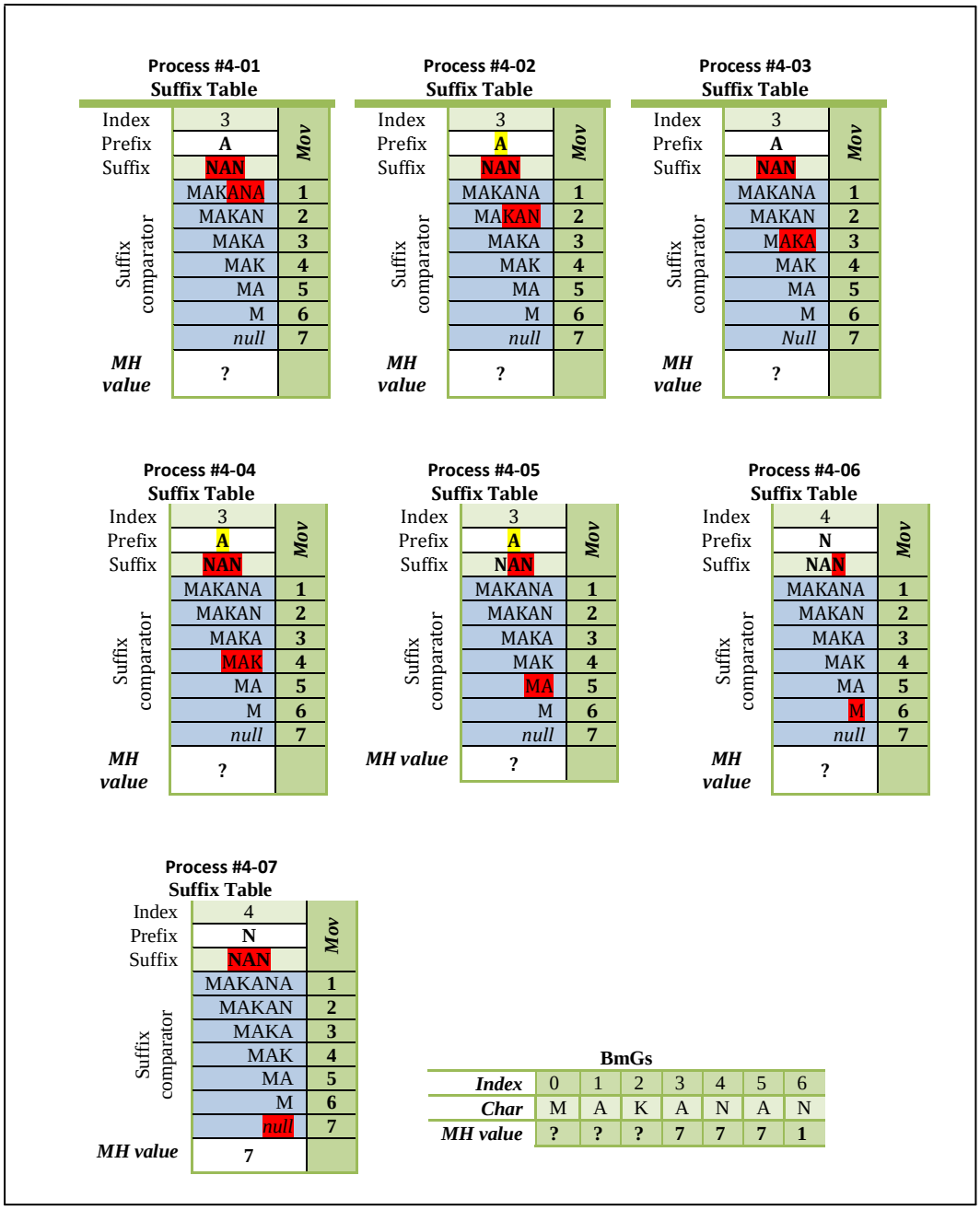
Gambar 2.15 Proses 1 PreBmGs



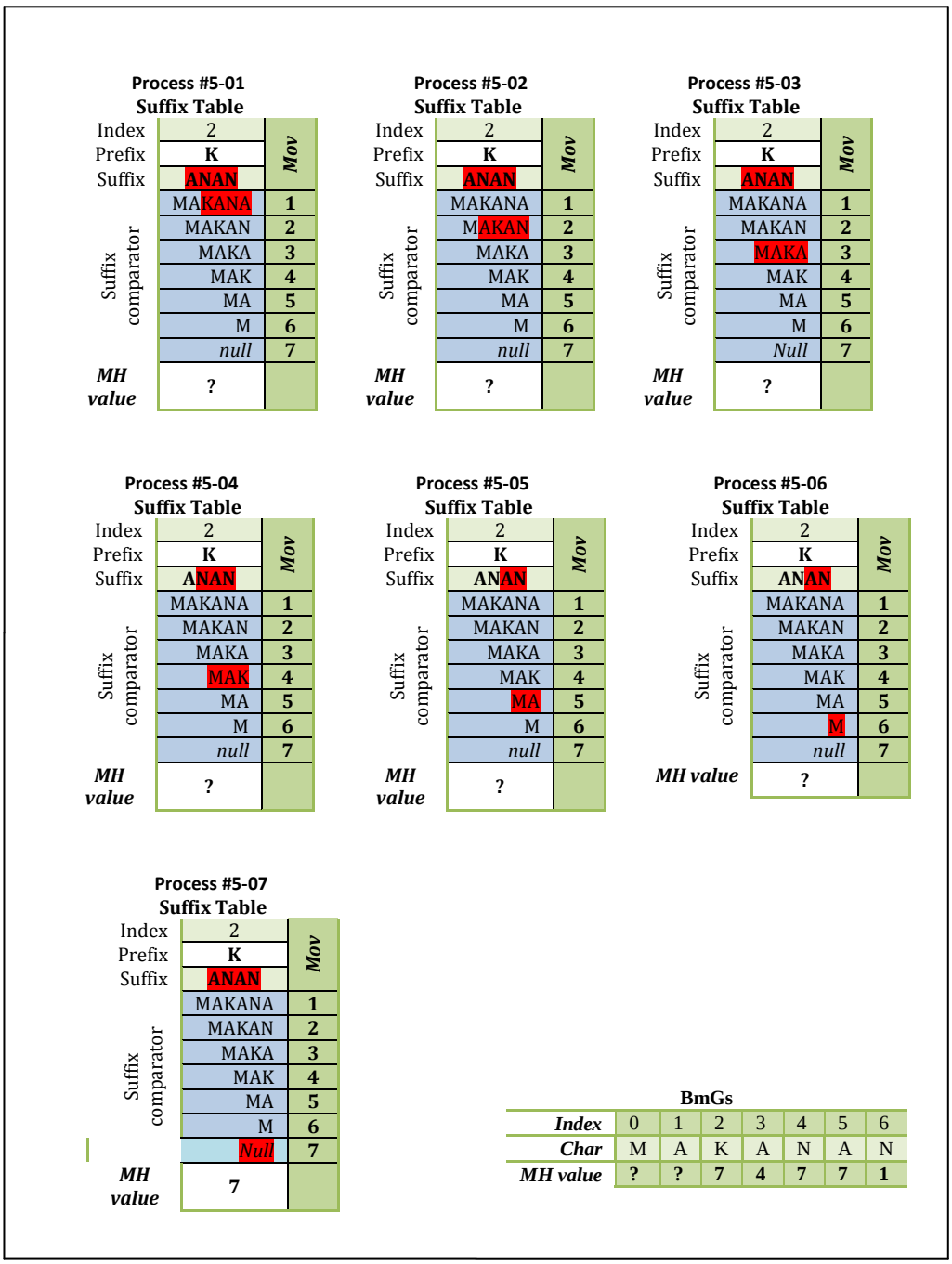
Gambar 2.16 Proses 2 PreBmGs



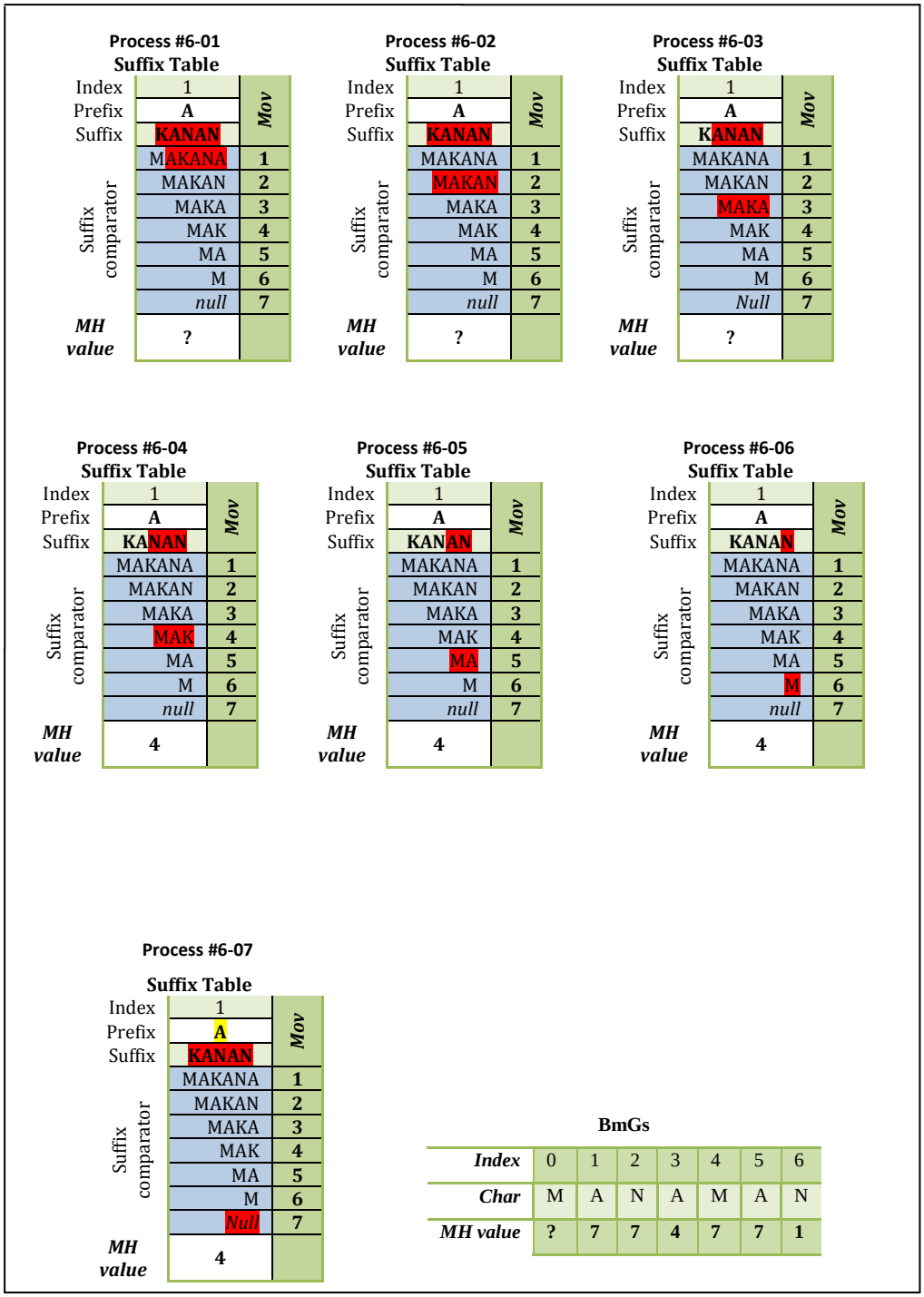
Gambar 2.17 Proses 3 PreBmGs



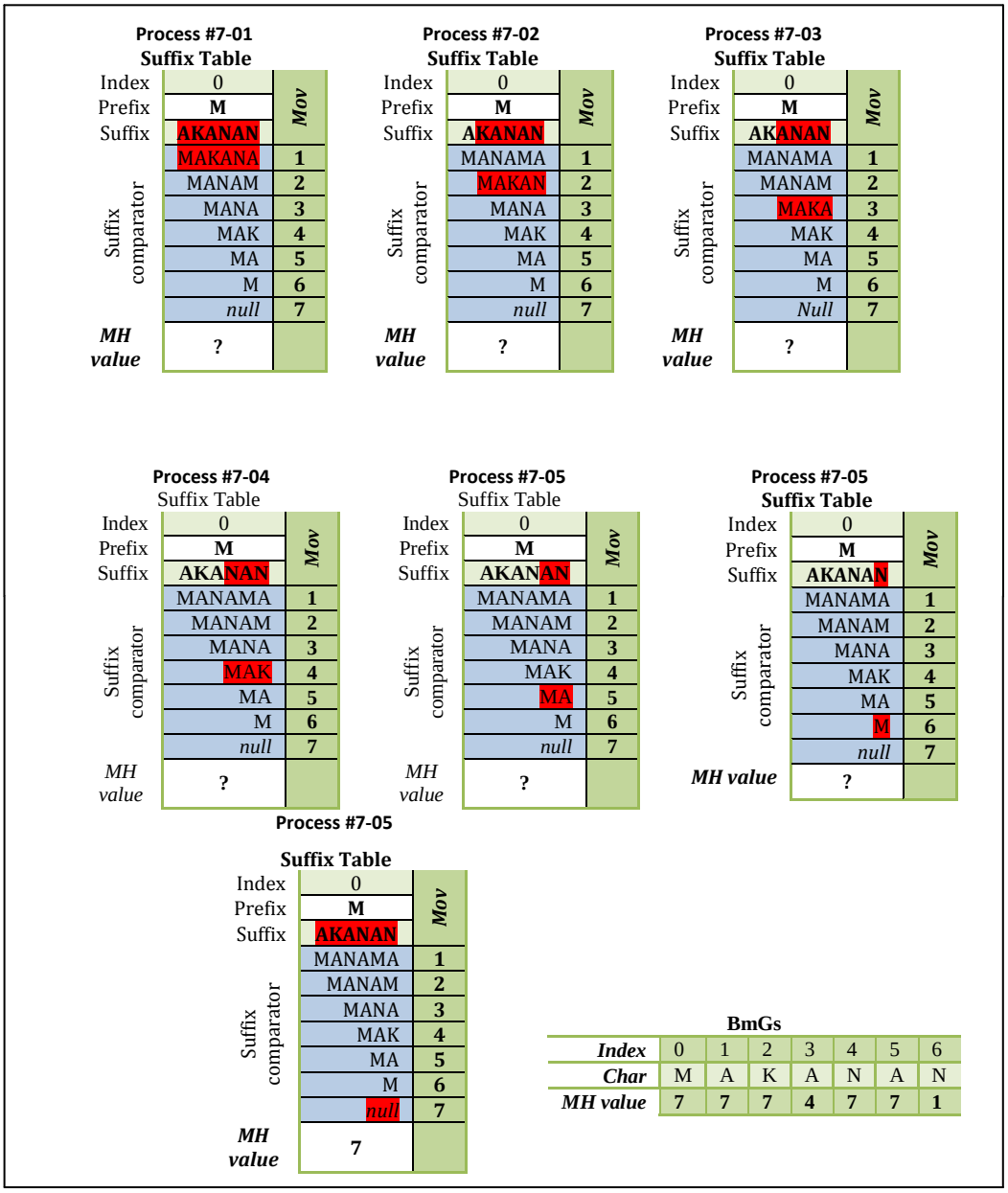
Gambar 2.18 Proses 4 PreBmGs



Gambar 2.19 Proses 5 PreBmGs



Gambar 2.20 Proses 6 PreBmGs



Gambar 2.21 Proses 7 PreBmGs

5. Kemudian lakukan pencarian pada teks dengan menggunakan OH dan MH yang telah dicari sebelumnya dengan aturan sebagai berikut :

   1) Jika karakter yang dituju tidak terdapat pada BmBc, maka OH=Length(pattern).

   2) Cari selisih antara OH dengan jumlah karakter yang telah Cocok.

3) Bandingkan OH dan MH, ambil nilai Max sebagai keputusan pengambilan Shift.

BmBc & BmGs							
Index	0	1	2	3	4	5	6
Char	M	A	K	A	N	A	N
OH	**6**	**1**	**4**	**1**	**2**	**1**	**2**
MH	**7**	**7**	**7**	**4**	**7**	**7**	**1**

Gambar 2.22. Tabel BmBc & BmGs hasil akhir beserta nilai

Sample text: ***CONTOH PENCARIAN MAKANAN***

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**
C	O	N	T	O	H		P	E	N	C	A	R	I	A	N		M	A	K	A	N	A	N	
M	A	K	A	N	A	N																		
0	**1**	**2**	**3**	**4**	**5**	**6**																		

Gambar 2.23 Contoh implementasi pencarian pada teks proses ke-1

Pattern[6] ≠ Text[6],

Macth char = 0

Maka OH=spasi=7-0, MH=idx[6]=1, dengan begitu pattern digeser 7 karakter

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
C	O	N	T	O	H		P	E	N	C	A	R	I	A	N		M	A	K	A	N	A	N	
							M	A	K	A	N	A	N											
							0	**1**	**2**	**3**	**4**	**5**	**6**											

Gambar 2.24 Contoh implementasi pencarian pada teks proses ke-2

Pattern[6] ≠ Text[13],

Macth char = 0

Maka OH=I=7-0, MH=idx[6]=1, dengan begitu pattern digeser 7 karakter.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
C	O	N	T	O	H		P	E	N	C	A	R	I	A	N		M	A	K	A	N	A	N	
														M	A	K	A	N	A	N				
														0	**1**	**2**	**3**	**4**	**5**	**6**				

Gambar 2.25 Contoh implementasi pencarian pada teks proses ke-3

Pattern[6] ≠ Text[20],

Macth char = 0

Maka OH=A=1-0, MH=idx[6]=1, dengan begitu pattern digeser 1 karakter.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
C	O	N	T	O	H		P	E	N	C	A	R	I	A	N		M	A	K	A	N	A	N	
															M	A	K	A	N	A	N			
															0	**1**	**2**	**3**	**4**	**5**	**6**			

Gambar 2.26 Contoh implementasi pencarian pada teks proses ke-4

Pattern[6] ≠ Text[21],

Macth char = 0

Maka OH=M=2-0, MH=idx[6]=1, dengan begitu pattern digeser 2 karakter.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
C	O	N	T	O	H		P	E	N	C	A	R	I	A	N		M	A	K	A	N	A	N	
																	M	A	K	A	N	A	N	
																	0	**1**	**2**	**3**	**4**	**5**	**6**	

Gambar 2.27 Contoh implementasi pencarian pada teks proses ke-5

Pattern[0..6] ≠ Text[17..21],

Macth char = 7

Maka dengan begitu pattern digeser 7 karakter, dan seterusnya sampai akhir teks.

2.6. Android

Android adalah sistem operasi untuk telepon seluler yang berbasis Linux. Android menyediakan platform terbuka bagi para pengembang untuk membuat aplikasi mereka sendiri untuk digunakan oleh bermacam peranti bergerak. Pada mulanya, Google Inc. membeli Android Inc., pendatang baru yang membuat peranti lunak untuk ponsel.

Kemudian untuk mengembangkan Android, dibentuklah Open Hanset Alliance, konsorium dari 34 perusahaan peranti lunak dan telekomunikasi,

termasuk Google, HTC, Intel, Motorola, Qualcom, T-Mobile dan Nvidia. Pada saat pelirisan perdana Android, 5 November 2007, Android bersama Open Handset Alliance menyatakan mendukung pengembangan standar terbuka pada perangkat seluler.

Di lain pihak, Google meliris kode-kode Android dibawah lisensi Apache, sebuah lisensi perangkat lunak dan standar terbuka perangkat seluler. Di dunia ini terdapat dua jenis distributor sistem operasi Android. Pertama yang mendapat dukungan penuh dari Google atau Google Mail Service (GMS) dan kedua adalah yang benar – benar bebas distribusinya tanpa dukungan langsung Google atau dikenal sebagai Open Handset Distribution (OHD).

2.6.1. Sejarah Android

Pada Juli 2000, Google bekerjasama dengan Android Inc., perusahaan yang berada di Palo Alto, California Amerika Serikat. Para pendiri Android Inc., bekerja pada Google, di antaranya Andy Rubin, Rich Miner, Nick Sears dan Chris White. Saat itu banyak yang menganggap fungsi Android Inc., hanyalah sebagai perangkat lunak pada telepon seluler. Sejak saat muncul rumor bahwa Google hendak memasuki pasar telepon seluler. Di perusahaan Google, tim yang dipimpin Rubin bertugas mengembangkan program perangkat seluler yang didukung oleh kernel Linux. Hal ini menunjukan indikasi bahwa Google sedang bersiap menghadapi persaingan dalam pasar telepon seluler. Versi android terbaru yaitu versi 3.0. Pada tahun 2005 Google mengakusisi Android Inc., yang pada saat itu di motori oleh Andy Rubin, Rich Miner, Nick Sears dan Crish White. Yang kemudian pada tahun itu juga memulai membangun platform Android secara lebih intensif. Kemudian pada tanggal 12 November 2007 Google bersama Open Handset Aliance (OHA) yaitu konsorium perangkat lunak mobile terbuka meliris Google Android SDK, setelah mengumumkanya seminggu sebelumnya. Dan sambutan sangat luar biasa, hamper semua media berita tentang IT dan Programing memberitakan tentang dirilisnya Android SDK (*Software Deplovment Kit).* Ini dikarenakan ide dari flatfrom Android sangat menarik untuk develover dan programmer di seluruh penjuru dunia. Di sisi lain, produsen

pembuat handset ponsel juga segera berlomba-lomba membuat handset dengan platform Android ini.

Google bersama dengan OHA merilis paket software SDK yang lengkat untuk mengembangkan aplikasi pada perangkat mobile, yaitu : Sistem Operasi Middleware dan Aplikasi utama untuk perangkat mobile. Sebagai Programer atau Developer kita bisa melakukan segalanya, mulai dari membuat aplikasi pengiriman SMS hanya dengan dua baris kode, hingga mengganti even pada Home Screen perangkat Android. Selain itu, bahkan dengan mudah kita bisa mebuat dan mengkostumasi Sistem Operasinya atau mengganti semua aplikasi default dari Google.

Semua aplikasi yang dibuat untuk android akan memiliki akses setara dalam mengakses seluruh kemampuan handset, tanpa membedakan apakah itu merupakan aplikasi inti atau aplikasi pihak ketika. Dalam kata lain dengan platform Android ini, Programer atau Developer secara penuh akan bisa mengkostumasi perangkat androidnya. Android built in pada LInuk kernel (Opern Linux Kernel), dengan sebuah mesin virtual yang telah didesain dan untuk mengoptimalkan penggunaan sumberdaya memori dan hardware pada lingkungan perangkat mobile (*Mobile Environment).* Dalvik adalah nama dari Android Virtual Mesin, yang merupakan interpreter virtual mesin yang akan mengeksekusi file dalam format Dalvik Executable (*.dex). Sebuah format yang dirancang untuk ruang penyimpanan yang efisien dan eksekusi memori yang terpetakan (*memory-mappable execution*).

Dalvik Virtual Mesin (Dalvik VM) berbasis sergister dan dapat mengeksekusi kelasi (class) yang telah terkompilasi pada complier bahasa java, kemudian di transformasikan kedalam native format dengan menggunakan Tool “dx” yang telah di integrasi. Kita mungkin telah mengenal JavaVM (Java Virtual Mesin), yang saat ini bisa ditemukan pada setiap komputer desktop. Berbeda dengan DalvikVM, JavaVM berbasis stack (Stack-based Virtual Machine), DalvikVM memiliki keunggulan dengan menggunakan Registred Base, ini karena

prosesore perangkat genggam telah dioptimalisasi untuk mengeksekusi berbasis register. Android saat ini tidak hanya berjalan pada handphone, beberapa vendor menananmkan Android pada tablet, Internet Tablet, E-Book Reader, Laptop dan gadget lainya. Adapun beberapa versi dari android yang telah diluncurkan diantaranya :

1. Android versi 1.1

   Pada 9 Maret 2009, Google meliris Android versi 1.1. Android versi ini dilengkapi dengan pembaruan estetis pada aplikasi, jam alarm, voice search (pencarian suara), pengiriman pesan Gmail dan pemberitahuan email.

2. Android versi 1.5 (Cupcake)

   Pada pertengahan Mei 2009, Google kembali meliris telepon seluler dengan menggunakan Android dan SDK (*Software Development Kit)* dengan versi 1.5 (Cupcake). Terdapat beberapa pembaruan termasuk juga penambahan beberapa fitur dalam seluler versi ini yakni kemampuan merekam dan meononton video dengan modus kamera, mengunggah video ke Youtube dan gambar ke Picasa langsung dari telepon, dukungan Bluetooth A2DP, kemampuan terhubung secara otomatis ke *headset* Bluetooth, naimasi layar dan *keyboard* pada layar yang dapat disesuaikan dengan sistem.

3. Android versi 1.6 (Donut)

   Donut (versi 1.6) diliris pada September dengan menampilkan proses pencarian yang lebih baik disbanding sebelumnya, penggunaan batrai indicator dan control applet VPN. Fitur lainya adalah galeri yang memungkinkan pengguna untuk memilih foto yang akan dihapus : kamera camcorder dan galeri yang diintegrasikan ; CDMA / EVDO, 802.1x, VPN, *Gestrures*, dan *Text-to-speech* engine; kemampuan dial kontak; teknologi

text to change speech (tidak tersedia pada semua ponsel; pengadaan resolusi VWGA.

4. Android versi 2.0/2.1 (Éclair)

   Pada 3 Desember 2009 kembali dilucurkan ponsel Android dengan versi 2.0/2.1 (Éclair), perubahan yang dilakukan adalah pengoptimalan hardware, peningkatan Google Maps 3.1.2, perubahan UI dengan browser baru dan dukungan HTML5, daftar kontak yang baru, dukungan *flash* untuk kamera 3,2 MP, digital Zoom dan Bluetooth 2.1. Untuk bergerak cepat dalam persaingan perangkat generasi berikut, Google melakukan investasi dengan mengadakan komptetisi aplikasi mobile terbaik (*killer apps* - aplikasi unggulan). Kompetisi ini berhadiah $25,000 bagi setiap pengembang aplikasi terpilih. Kompetisi diadakan selama dua tahap yang tiap tahap nya dipilih 50 aplikasi terbaik. Dengan semakin berkembangnya dan semakin bertambahnya jumlah handset Android, semakin banyak pihak ketiga yang berminat untuk menyalurkan aplikasi mereka pada sistem operasi Android. Aplikasi terkenal yang diubah ke dalam sistem operasi Android adalah Shazam, Backgrounds dan WheaterBug. Sistem operasi Android dalam situs Internet juga dianggap penting untuk menciptakan aplikasi Android asli, contohnya oleh MySpace dan Facebook.

5. Android versi 2.2 (Froyo: Frozen Yoghurt)

   Pada 20 Mei 2010, Android versi 2.2 (Froyo) diluncurkan. Perubahan-perubahan umumnya terhadap veris-veris sebelumnya antara lain dukungan Adobe Flash 1.1 , kecepatan kinerja dan aplikasi 2 sampai 5 kali lebih cepat, integrasi V8 JavaScript engine yang dipakai Google Chrome yang mempercepat kemampuan rendering pada browser, pemasangan aplikasi dalam SD Card, kemampuan WiFi Hotspot portable dan kemampuan auto update dalam aplikasi Android Market (Google Play).

6. Android versi 2.3 (Gingerbread)

   Pada 6 Desember 2010, Android versi 2.3 (Gingerbread) diluncurkan. Perubahan-perubahan umum yang didapat dari Android versi ini antara lain peningkatan kemampuan permainan (*gaming*), peningkatan fungsi *copy paste*, layar antar muka (*User Interface*) didesain ulang, dukungan format video VP8 dan WebM, efek audio baru (*reverb, equalization, handphone virtualization* dan *bass boost*), dukungan kemampuan *Near Field Communication* (NFC) dan dukungan jumlah kamera yang lebih dari satu.

2.7. Eclipse

2.7.1. Sejarah Eclipse

Eclipse awalnya dikembangkan oleh IBM untuk menggantikan perangkat lunak IBM Visual Age for Java 4.0. Produk ini diluncurkan oleh IBM pada tanggal 5 November 2001, yang menginvestasikan sebanyak US$ 40 juta untuk pengembangannya. Semenjak itu konsursium Eclipse Foundation mengambil alih untuk pengembangan Eclipse lebih lanjut dan pengaturan organisasinya.

2.7.2. Arsitektur Eclipse

Sejak versi 3.0, Eclipse pada dasarnya merupakan sebuah kernel, yang mengangkat plug-in. Apa yang dapat digunakan di dalam Eclipse sebenarnya adalah fungsi dari plug-in yang sudah diinstal. Ini merupakan basis dari Eclipse yang dinamakan Rich Client Platform (RCP). Berikut ini adalah komponen yang membentuk RCP:

1. Core platform
2. OSGi
3. SWT (Standard Widget Toolkit)
4. JFace
5. Eclipse Workbench

Secara standar Eclipse selalu dilengkapi dengan JDT (Java Development Tools), plug-in yang membuat Eclipse kompatibel untuk mengembangkan program Java, dan PDE (Plug-in Development Environment) untuk

mengembangkan plug-in baru. Eclipse beserta plug-in-nya diimplementasikan dalam bahasa pemrograman Java.

Konsep Eclipse adalah IDE yang terbuka (open), mudah diperluas (extensible) untuk apa saja, dan tidak untuk sesuatu yang spesifik. Jadi, Eclipse tidak saja untuk mengembangkan program Java, akan tetapi dapat digunakan untuk berbagai macam keperluan, cukup dengan menginstal plug-in yang dibutuhkan. Apabila ingin mengembangkan program C/C++ terdapat plug-in CDT (C/C++ Development Tools). Selain itu, pengembangan secara visual bukan hal yang tidak mungkin oleh Eclipse, plug-in UML2 tersedia untuk membuat diagram UML. Dengan menggunakan PDE setiap orang bisa membuat plug-in sesuai dengan keinginannya.

2.7.3. Perkembangan

Sejak tahun 2006, Eclipse Foundation mengkoordinasikan peluncuran Eclipse secara rutin dan simultan yang dikenal dengan nama Simultaneous Release. Setiap versi peluncuran terdiri dari Eclipse Platform dan juga sejumlah proyek yang terlibat dalam proyek Eclipse. Tujuan dari sistem ini adalah untuk menyediakan distribusi Eclipse dengan fitur-fitur dan versi yang terstandarisasi. Hal ini juga dimaksudkan untuk mempermudah deployment dan maintenance untuk sistem enterprise. Adapun versi eclipse yang telah diluncurkan yaitu :

1. Eclipse 3.0
2. Eclipse 3.1
3. Callisto
4. Europa
5. Ganymede
6. Galileo
7. Helios
8. Juno

2.7.4. *Plugin* Android *Development Tools* (ADT)

ADT adalah plugin untuk *Eclipse Intergrated Development Environment* (IDE) yang dirancang untuk memberikan lingkungan yang terpadu di mana untuk membangun aplikasi Android. ADT memperluas kemampuan Eclipse untuk membiarkan para developer lebih cepat dalam membuat proyek baru Android, membuat aplikasi UI, menambahkan komponen berdasarkan Android Framework API, debug aplikasi dalam pengunaan Android SDK, dan membuat file APK untuk mendistribusikan aplikasi.

Mengembangkan aplikasi di Eclipse dengan ADT sangat dianjurkan dan merupakan cara tercepat untuk memulai membuat aplikasi android, karena banyak kemudahan-kemudahan sebagai tools yang terintegrasi seperti, custom XML editor, dan debug panel ouput. Selain itu ADT memberikan dorongan luar biasa dalam mengembangkan aplikasi Android.

2.8. Java

Java adalah bahasa pemrograman yang dapat dijalankan di berbagai komputer termasuk telepon genggam. Dikembangkan oleh Sun Microsystems dan diterbitkan tahun 1995. Java tidak boleh disalahpahami sebagai JavaScript. JavaScript adalah bahasa *scripting* yang digunakan oleh *web browser*.

Java adalah bahasa pemrograman yang disusun oleh James Gosling yang dibantu oleh rekan-rekannya seperti Patrick Naugton, Chris Warth, Ed Frank, dan Mike Sheridan di suatu perusahaan perangkat lunak yang bernama Sun Microsystem, pada tahun 1991.

Bahasa pemrograman ini mula-mula diinisialisasi dengan nama“oak”, namun pada tahun 1995 diganti namanya menjadi “Java”. Alasan utama pembentukan bahasa java adalah untuk membuat aplikasi-aplikasi yang dapat diletakkan diberbagai macam perangkat elektronik, seperti *microwave oven* dan *remote control*, sehingga Java harus bersifat *portable* atau yang sering disebut dengan *platform independent* (tidak tergantung pada *platform*). Itulah yang menyebabkan dalam dunia pemrograman Java, dikenal adanya istilah ‘*write once, run everywhere*’, yang berarti kode program hanya ditulis sekali, namun dapat

dijalankan dibawah *platform* manapun, tanpa harus melakukan perubahan kode program.

2.8.1. Sejarah Java

Pada 1991, sekelompok insinyur Sun dipimpin oleh Patrick Naughton dan James Gosling ingin merancang bahasa komputer untuk perangkat konsumer seperti *cable TV Box*. Karena perangkat tersebut tidak memiliki banyak memori, bahasa harus berukuran kecil dan mengandung kode yang liat. Juga karena manufaktur – manufaktur berbeda memilih *processor* yang berbeda pula, maka bahasa harus bebas dari manufaktur manapun. Proyek diberi nama kode ”Green”. Bahasa pemrograman Java pertama lahir dari The Green Project, yang berjalan selama 18 bulan, dari awal tahun 1991 hingga musim panas 1992.Salah satu hasil proyek ini adalah maskot *Duke* yang dibuat oleh Joe Palrang.

Kebutuhan untuk fleksibilitas, kecil, liat dan kode yang netral terhadap *platform* mengantar tim mempelajari implementasi Pascal yang pernah dicoba. Niklaus Wirth, pencipta bahasa Pascal telah merancang bahasa portabel yang menghasilkan *intermediate code* untuk mesin hipotesis. Mesin ini sering disebut dengan mesin maya *(virtual machine)*. Kode ini kemudian dapat digunakan di sembarang mesin yang memiliki *interpreter*. Proyek Green menggunakan mesin maya untuk mengatasi isu utama tentang netral terhadap arsitektur mesin.

Karena orang – orang di proyek Green berbasis C++ dan bukan Pascal maka kebanyakan sintaks diambil dari C++, serta mengadopsi orientasi objek dan bukan prosedural. Mulanya bahasa yang diciptakan diberi nama ”Oak” oleh James Gosling yang mendapat inspirasi dari sebuah pohon yang berada pada seberang kantornya, namun dikarenakan nama Oak sendiri merupakan nama bahasa pemrograman yang telah ada sebelumnya, kemudian SUN menggantinya dengan JAVA. Nama JAVA sendiri terinspirasi pada saat mereka sedang menikmati secangkir kopi di sebuah kedai kopi yang kemudian dengan tidak sengaja salah satu dari mereka menyebutkan kata JAVA yang mengandung arti asal bijih kopi. Akhirnya mereka sepakat untuk memberikan nama bahasa pemrograman tersebut dengan nama Java.

Pertemuan proyek berlangsung di sebuah gedung perkantoran *Sand Hill Road* di Menlo Park. Sekitar musim panas 1992 proyek ini ditutup dengan menghasilkan sebuah program *Java Oak* pertama, yang ditujukan sebagai pengendali sebuah peralatan jarak jauh yang sangat cerdas dengan teknologi layar sentuh (*touch screen*), seperti pada PDA sekarang ini. Produk pertama teknologi baru ini dinamai "*7" (*Star Seven*).

Dikarenakan pasar masih belum tertarik dengan produk konsumer cerdas maka proyek Green harus menemukan pasar lain dari teknologi yang diciptakan. Pada saat yang sama, implementasi WWW dan Internet sedang mengalami perkembangan pesat. Di lain pihak, anggota dari proyek Green juga menyadari bahwa Java dapat digunakan pada pemrograman internet, sehingga penerapan selanjutnya mengarah menjadi teknologi yang berperan di web.

Sayang terjadi perpecahan di antara mereka suatu hari pada pukul 04.00 di sebuah ruangan hotel Sheraton Palace. Tiga dari pimpinan utama proyek, Eric Schmidt dan George Paolini dari Sun Microsystems bersama Marc Andreessen, membentuk Netscape.

Nama Oak, diambil dari pohon oak yang tumbuh di depan jendela ruangan kerja "bapak java", James Gosling. Nama Oak ini tidak dipakai untuk versi release Java karena sebuah perangkat lunak sudah terdaftar dengan merek dagang tersebut, sehingga diambil nama penggantinya menjadi "Java". Nama ini diambil dari kopi murni yang digiling langsung dari biji (kopi tubruk) kesukaan Gosling. Konon kopi ini berasal dari Pulau Jawa. Jadi nama bahasa pemrograman Java tidak lain berasal dari kata Jawa (bahasa Inggris untuk Jawa adalah Java).

Versi awal Java ditahun 1996 sudah merupakan versi release sehingga dinamakan Java Versi 1.0. Java versi ini menyertakan banyak paket standar awal yang terus dikembangkan pada versi selanjutnya :

1. Java.lang : Peruntukan kelas elemen-elemen dasar.
2. Java.io : Peruntukan kelas input dan output, termasuk penggunaan berkas.
3. Java.util : Peruntukan kelas pelengkap seperti kelas struktur data dan kelas kelas penanggalan.

4. Java.net : Peruntukan kelas TCP/IP, yang memungkinkan berkomunikasi dengan komputer lain menggunakan jaringan TCP/IP.
5. Java.awt : Kelas dasar untuk aplikasi antarmuka dengan pengguna (GUI).
6. Java.applet : Kelas dasar aplikasi antar muka untuk diterapkan pada penjelajah web.

2.9. *Object Oriented Programing* (OOP)

Object Oriented Programing atau dikenal dengan pemograman berorientasi object merupakan paradigma pemograman yang berorientasikan kepada objek. Semua data dan fungsi dalam paradigma ini dibungkus ke dalam kelas-kelas atau objek-objek.

Dengan menggunakan OOP maka dalam melakukan pemecahan suatu masalah tidak melihat bagaimana cara menyelesaikan masalah tersebut (terstruktur) tetapi objek-objek apa yang dapat melakukan pemecahan masalah tersebut. Sebagai contoh anggap dalam sebuah departemen yang mememiliki manajer, sekrtaris , petugas administrasi data dan lainnya. Misalkan manajer tersebut ingin memperoleh data dari bagian administrasi maka manager tersebut tidak harus mengambilnya langsung tetapi dapat menyruh petugas bagian administrasi untuk mengambilnya. Pada kasus tersebut seorang manajer tidak harus mengetahui bagaimana cara mengambil data tersebut tetapi manajer bisa mendapatkan data tersebut melaui objek petugas administrasi. Jadi untuk menyelesaikan suatu masalah dengan kolaborasi antar objek-objek yang ada karena setiap objek memiliki deskripsi tugasnya sendiri.

Pemograman orientasi objek menekankan konsep berikut :

1. *Class* – kumpulan atas definisi data dan fungsi-fungsi dalam suatu unit untuk suatu tujuan tertentu. Sebagai contoh ‘*class of dog*’ adalah suatu unit yang terdiri atas definisi-definisi data dan fungsi-fungsi yang menunjuk pada berbagai macam perilaku/turunan dari anjing. Sebuah *class* adalah dasar dari modularitas dan struktur dalam pemograman berorientasi objek. Sebuah *class* secara tipikal sebaiknya dapat dikenali oleh seorang non-

programer sekalipun terkait dalam sebuah domain permasalahan yang ada dan ide yang terdapat dalam sebuah *class* sebaiknya (*relative*) bersifat mandiri dan independen (sebagaimana kode tersebut digunakan jika tidak menggunakan OOP). Dengan modularitas struktur dari sebuah program akan terkait dengan aspek-aspek dalam masalah yang akan diselesaikan melalui program tersebut. Cara seperti ini akan menyederhanakan pemetaan dari masalah ke sebuah program ataupun sebaliknya.

2. Objek (*Object*) – membungkus data dan fungsi bersama menjadi suatu unit dalam sebuah program komputer. Objek merupakan dasar dari modularitas dan struktur dalam sebuah program komputer berorientasi objek.

3. Abstraksi (*Abstract*) – Kemampuan sebuah program untuk melewati aspek informasi yang diprosesnya, yaitu kemampuan untuk memfokuskan pada inti. Setiap objek dalam sistem melayani sebagai model dari "pelaku" abstrak yang dapat melakukan kerja, laporan dan perubahan keadaaanya dan berkomunikasi dengan objek lainya dalam sistem, tanpa mengungkapkan bagaimana kelebihan ini diterapkan. Proses, fungsi atau metode dapat juga dibuat abstrak dan beberapa teknik digunakan untuk mengembangkan sebuah pengabstrakan.

4. Engkapsulasi (*Encapsulation*) – Memastikan pengguna sebuah objek tidak dapat mengganti keadaaan dalam dari sebuah objek dengan cara yang tidak layak; hanya metode dalam objek tersebut yang diberi ijin untuk mengakses keadaaanya. Setiap objek mengakses *interface* yang menyebutkan bagaimana objek lainya dapa berinteraksi dengannya. Objek lainya tidak akan mengetahui dan tergantung kepada representasi dalam objek tersebut.

5. Polimorfime (*Polimorfism*) melalui pengiriman pesan. Tidak bergantung kepada pemanggilan subrutin orientasi objek dapat mengirim pesan; metode tertentu yang berhubungan dengan sebuah pengriman pesan tergantung kepada objek tertentu dimana pesan tersebut dikirim.

Contohnya, bila sebuah burung menerima pesan “gerak cepat”, dia akan menggerakan sayapnya dan terbang. Bila seekor singa menerima pesan yang sama, dia akan menggerakan kakinya dan berlari. Keduanya menjawab sebuah pesan yang sama namun yang sesuai dengan kemampuan hewan tersebut. Ini disebut polimorfisme karena sebuah variabel tunggal dalam program dapat memegang berbagai jenis objek yang berbeda selagi program berjalan dan teks program yang sama dapat memanggil beberapa metode yang berbeda di saat yang berbeda dalam pemanggilan yang sama. Hal ini berlawanan dengan bahasa fungsional yang mencapai polimorfisme melaui penggunaan fungsi kelas-pertama.

6. Inheritas (*inheritance*) – Mengatur polismorfisme dan engkapsulasi dengan mengijinkan objek didefinisikan dan diciptakan dengan jenis khusus dari objek yang sudah ada – objek-objek ini dapat membagi (memperluas) perilaku mereka tanpa harus mengimplementasikan ulang perilaku tersebut (bahasa berbasis-objek tidak selalu memlikik inheritas).

2.10. *Unified Modeling Language* (UML)

UML adalah bahasa spesifikasi standar untuk mendokumentasikan, menspesifikasikan dan membangun sistem perangkat lunak. UML adalah himpunan struktur dan teknik untuk pemodelan desain program berorientasi objek (OOP) serta aplikasinya. UML adalah metodologi untuk mengembangkan sistem OOP dan sekelompok perangkat untuk mendukung pengembangan sistem tersebut. UML mulai diperkenalkan oleh Object Management Group, sebuah organisasi yang telah mengembangkan model, teknologi dan standar OOP sejak tahun 1980-an. Sekarang UML sudah mulai banyak digunakan oleh para praktisi OOP. UML merupakan dasar bagi perangkat (*tool*) desain berorientasi objek dari IBM.

UML adalah suatau bahasa yang digunakan untuk menentukan, menvisualisasikan, membangun dan mendokumentasikan suatu sistem informasi. UML dikembangkan sebagai suatu alat untuk analisis dan desain berorientasi

objek oleh Grady Booch, Jim Rumbaugh dan Ivar Jacobson. Namun demikian UML dapat digunakan memahami dan mendokumentasikan setiap sistem informasi. Penggunaan UML dalam industry terus meningkat. Ini merupakan standar terbuka yang menjadikannya sebagai bahasa pemodelan yang umum dalam industry peranti lunak dan pengembangan sistem.

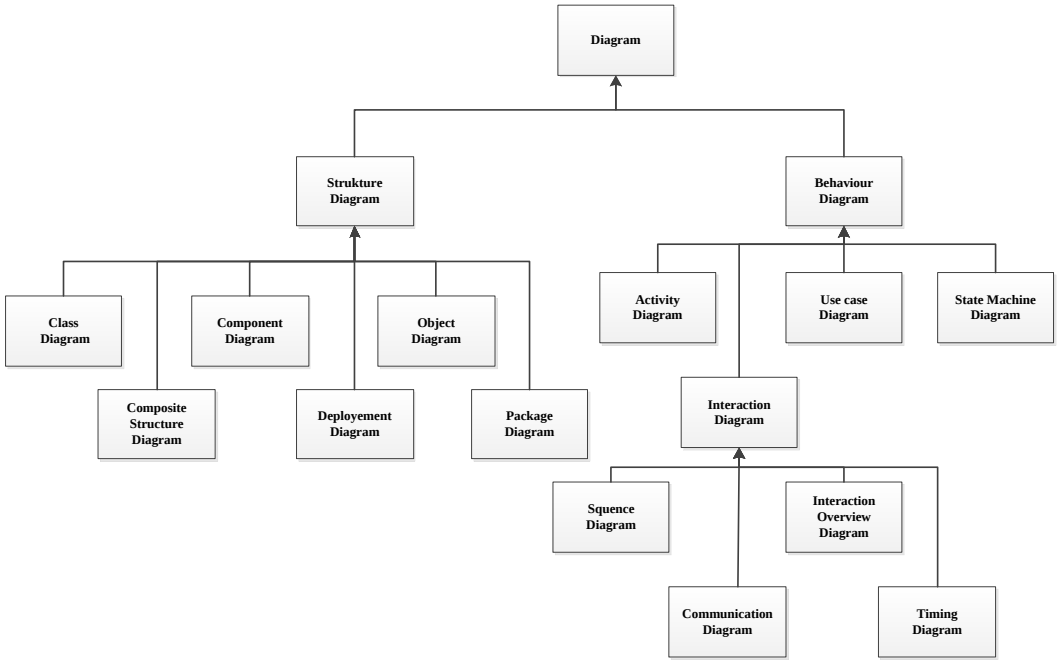
2.10.1. Sejarah UML

Sampai era tahun 1990 puluhan metodologi pemodelan berorientasi objek telah bermunculan di dunia. Diantaranya adalah : metodologi booch, metodologi coad, metodologi OOSE, metodologi OMT, metodologi shlaer-mellor, metodologi wirfs-brock, dsb. Masa itu terkenal dengan masa perang metodologi (method war) dalam pendesaainan berorientasi objek. Masing-masing metodologi membawa notasi sendiri-sendiri yang mengakibatkan timbul maslah baru apabila bekerjasama dengan kelompok/perusahaan lain yang menggunakan metodologi yang berlainan.

Dimulai pada bulan Oktober 1994 Booch, Rumbaugh dan Jacobson yang merupakan tiga tokoh yang boleh dikatakan metodologinya banyak digunakan mempelopori usaha untuk penyatuan metodologi pendesainan berorientasi objek. Pada tahun 1995 direlease draft pertama UML (versi 0.8). Sejak tahun 1996 pengembangan tersebut dikoordinasikan oleh Object Managemen Group [OMG].

2.10.2. Diagram UML (versi 2.0)

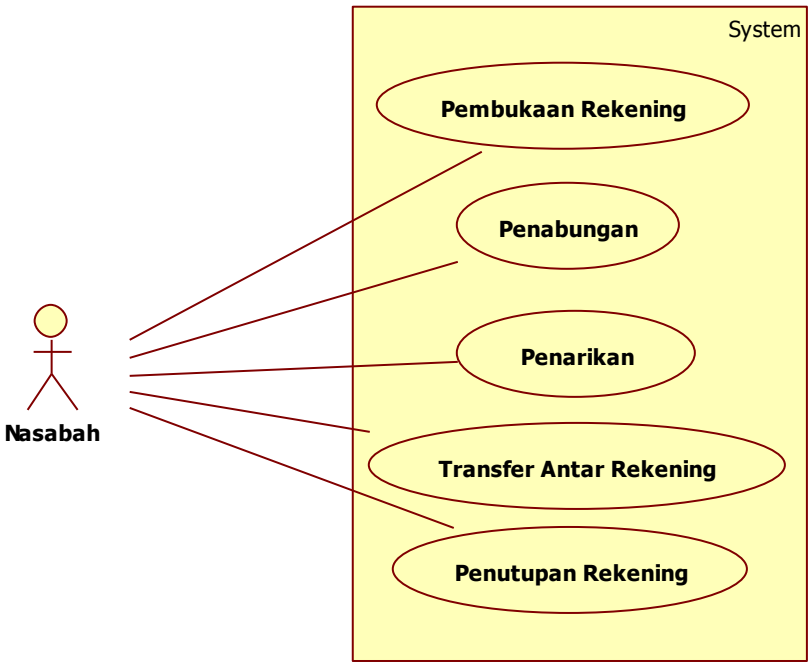
UML merupakan sintak umum untuk membuat model logika dari suatu sistem dan digunakan untuk menggambarkan sistem agar dapat dipahami selama fase analisis dan desain. UML biasanya disajikan dalam bentuk diagram / gambar yang meliputi *class* beserta atribut dan operasinya, serta hubungan antara *class* yang meliputi *inheritance, association* dan komposisi. UML terdiri dari banyak diagram secara garis besar diagram yang terdapat pada UML dapat diperlihatkan pada gambar 2.28 berikut ini :



Gambar 2.28 Diagram-diagram pada UML versi 2.0

2.10.2.1 *Use Case* Diagram

Use Case Diagram menggambarkan fungsionalitas yang diharapkan dari sebuah sistem. Yang ditekankan adalah "apa" yang diperbuat sistem dan bukan "bagaimana". Sebuah *use case* merepresentasikan sebuah interaksi antara actor dengan sistem. *Use case* merupakan sebuah pekerjaan tertentu, misalnya *login* ke sistem, meng-*create* sebuah daftar belanja dan sebgainya. Seorang /sebuah actor adalah sebuah entitas manusia atau mesin yang berinteraksi dengan sistem melakukan pekerjaan-pekerjaan tertentu. Sebuah *use case* dapat meng-*include* fungsionalitas *use case* lain sebgai bagian dari proses dalam dirinya. Secara umum diasumsikan bahwa *use case* yang di-*include* akan dipanggil setiapkali *use case* yang meng-*include* diekssekusi secara normal. Sebuah *use case* dapat di-*include* oleh lebih dari satu *use case* lain, sehinga duplikasi funsionalitas dapat dihindari dengan cara menarik keluar fungsionalitas yang *common.* Sebuah *use case* juga dapat meng-*extend use case* lain dengan *behavior*-nya sendiri. Sementara hubungan generalisasi anatar *use case* menunjukan bahwa *use case* yang satu merupakan spesialisasi dari yang lain. Penggambaran *Use Case diagram* dapat di lihat dalam gambar 2.29 berikut.



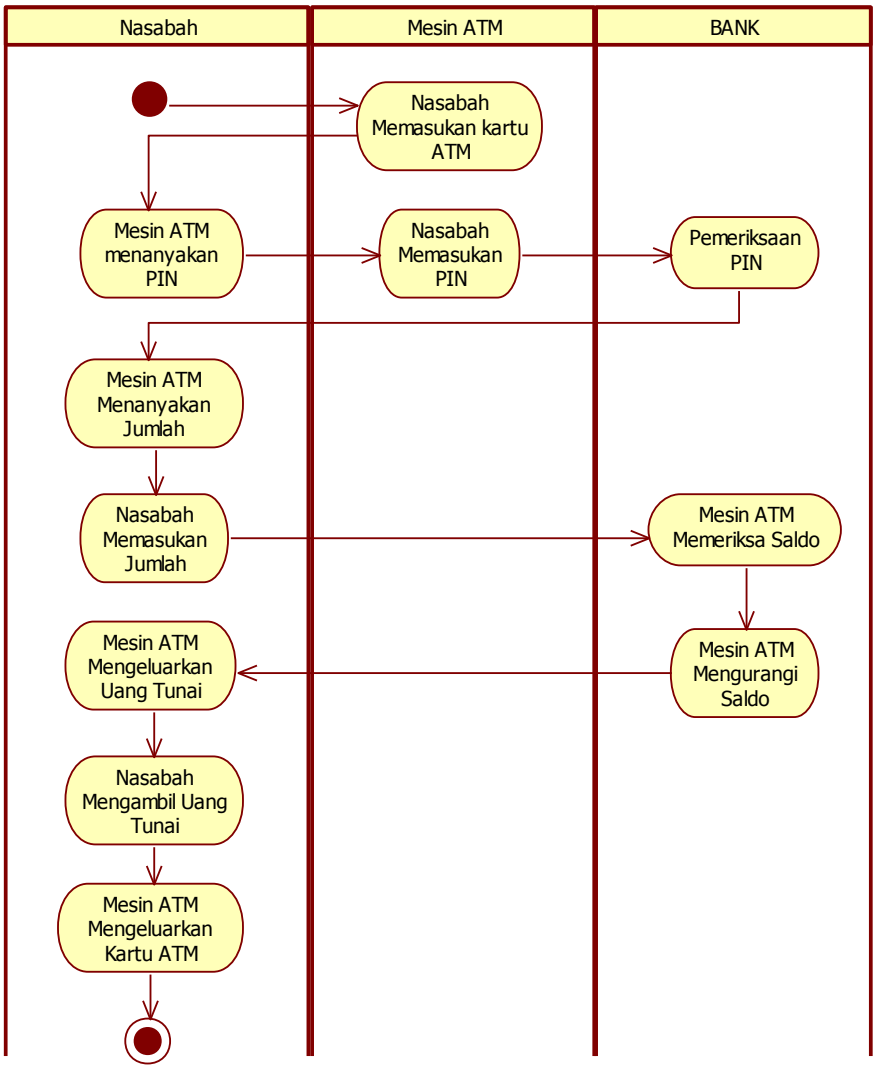
Gambar 2.29 contoh use case diagram

2.10.2.2. *Activity* Diagram

Activity diagram menggambarkan berbagai alir aktivitas dalam sistem yang sedang dirancang, bagaimana masing-masing alir berawal, *decision* yang mungkin terjadi dan bagaimana mereka berakhir. *Activity diagram* juga dapat menggambarkan proses pararel yang mungkin terjadi pada beberapa eksekusi. *Activity diagram* merupakan *state diagram* khusus, dimana sebagian besar *state* sebelumnya (*internal processing).* Oleh karena itu *activity diagram* tidak menggambarkan *behavior internal* sebuah sistem (interaksi antar subsistem) secara eksak, tetapi lebih menggambarkan proses-proses dan jalur-jalur aktivitas dari level atas secara umum.

Sebuah aktivitas dapat direalisasikan oleh satu *use case* atau lebih. Aktivitas menggambarkan proses yang berjalan, sementara *use case* menggambarkan bagaimana actor menggunakan sistem untuk melakukan aktivitas. *Decision* digunakan untuk menggambarkan behavior pada kondisi terntu. Untuk mengilustrasikan proses-proses pararel (*fork* dan *join*) digunakan

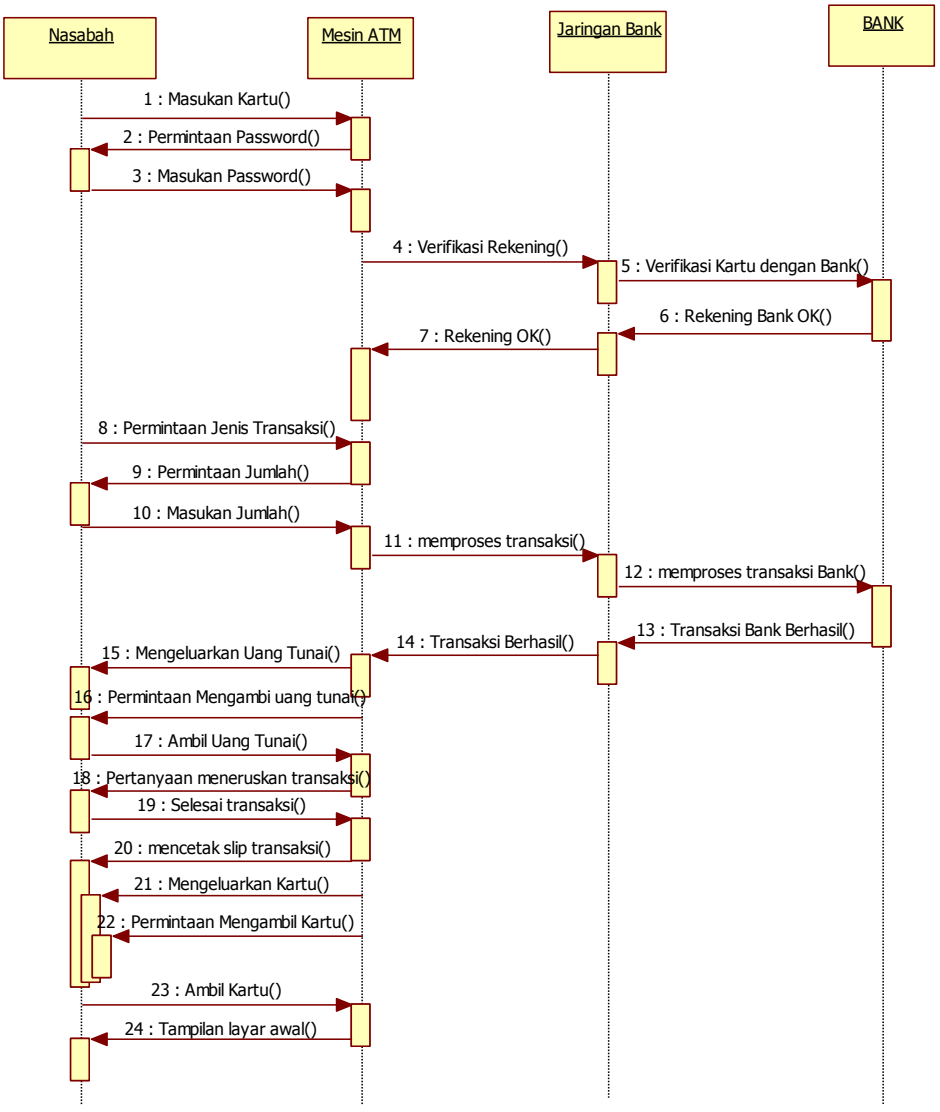
titik singkronisasi yang dapat berupa titik, garis horizontal atau vertical. *Activity diagram* dapat dibagi menjadi beberapa *object swimlane* untuk menggambarkan objek dimana yang bertanggung jawab untuk aktivitas tertentu. Contoh Activity diagram dapat dilihat pada gambar 2.30 berikut ini.



Gambar 2.30 contoh activity diagram

2.10.2.3. *Squence* Diagram

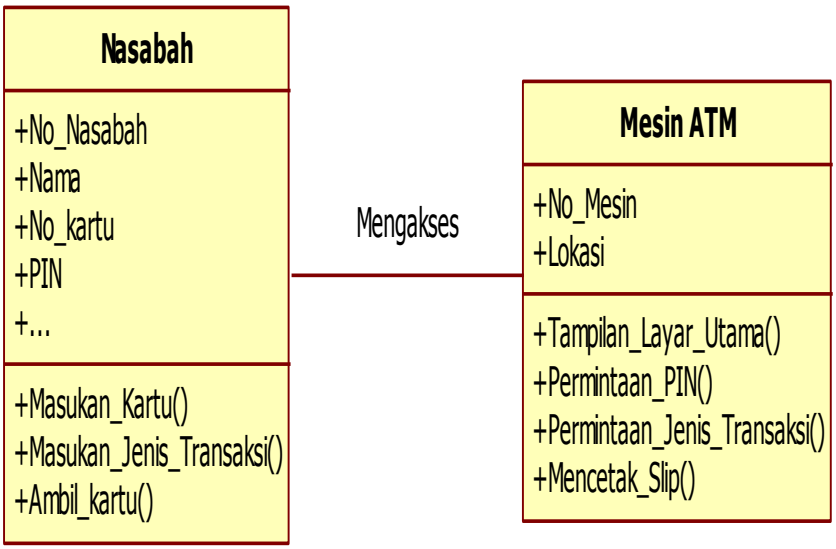
Sequence diagram menjelaskan secara detail urutan proses yang dilakukan dalam sistem untuk mencapai tujuan dari use case: interaksi yang terjadi antar *class*, operasi apa saja yang terlibat , urutan antar operasi dan informasi yang diperlukan oleh masing-masing operasi. Contoh sequence diagram dapat dilihat pada gambar 2.3 berikut ini.



Gambar 2.31 Contoh sequence diagram

2.10.2.4. *Class* Diagram

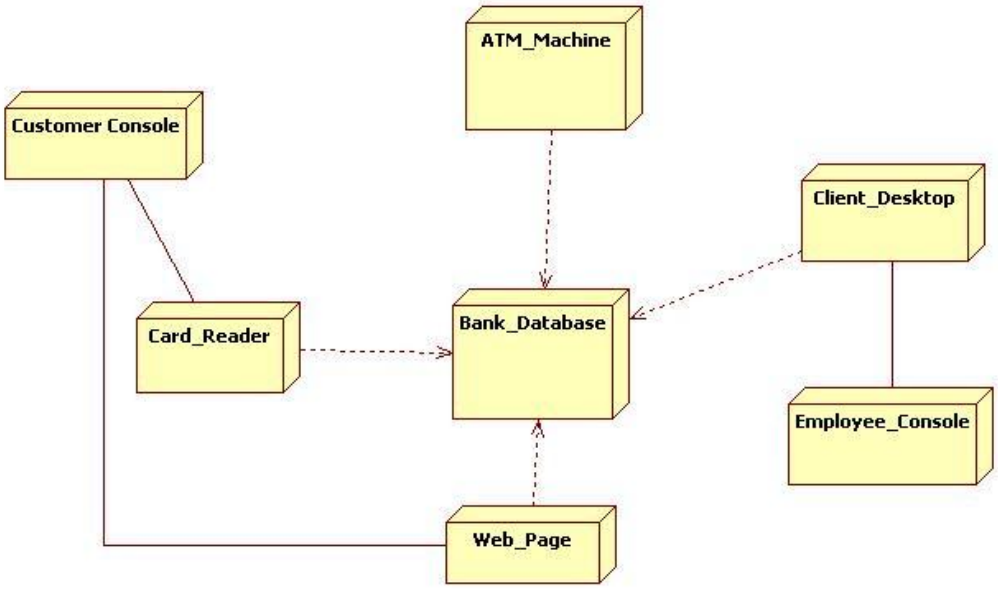
Class diagram merupakan diagram yang selalu ada di pemodelan sistem berorientasi objek. Class diagram menunjukan hubungan antar class dalam sistem yang sedang dibangun dan bagaimana mereka saling berkolaborasi untuk mencapai suatu tujuan. Penggambaran class diagram dapat dilihat dalam gambar 2.31 dibawah ini.



Gambar 2.32 Contoh class diagram

2.10.2.5. *Deployment* Diagram

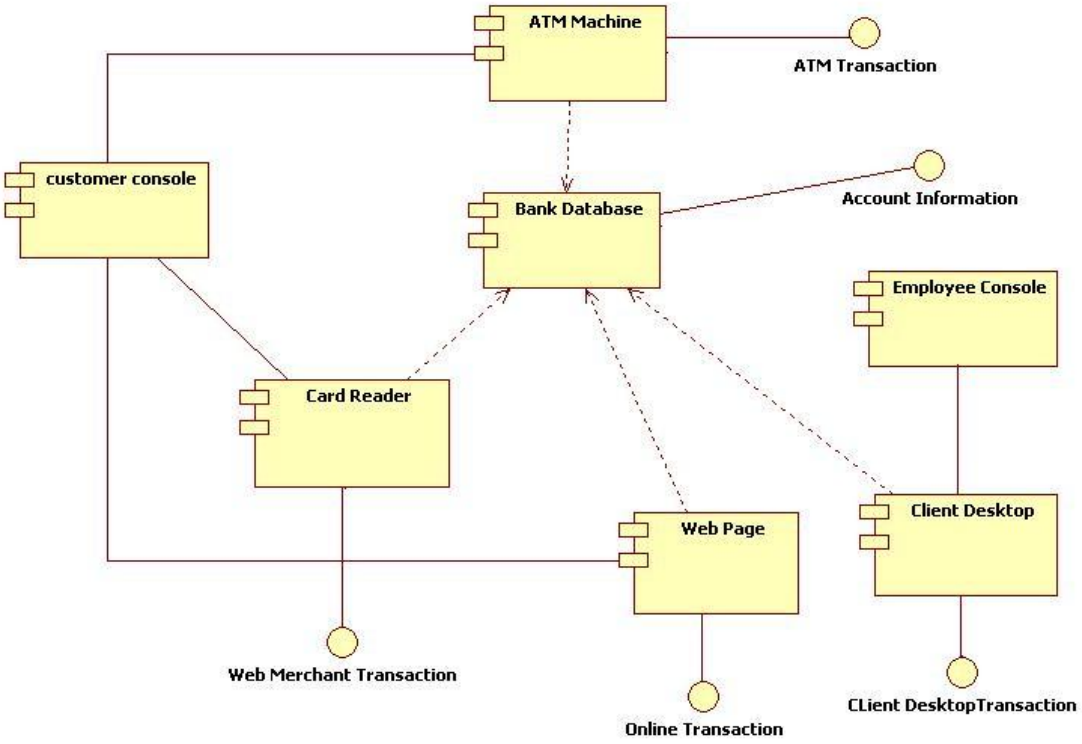
Deployment diagram menunjukan tata letak sebuah sistem secara fisik, memperlihatkan bagian-bagian *hardware*. Bagian utama *hardaware*/perangkat keras adalah *node;* yaitu nama umum untuk sebuah jenis sumber komputasi. Prosesor adalah node yang bisa mengeksekusi sebuah komponen, sedangkan *device* tidak. *Device* adalah perangkat keras (seperti printer atau monitor) tipikalnya menjadi interface dengan dunia luar. Contoh lainnya pada mesin ATM terdapat banyak subsistem yang dapat dijalankan pada peralatan fisik yang terpisah. Peralatan fisik tersebut disebut node yang dihubungkan satu sama lain dengan menggunakan garis yang disebut path.



Gambar 2.33 Contoh deployment diagram

2.10.2.6. *Component* Diagram

Component diagram menggambarkan struktur dan hubungan antar komponen piranti lunak, termasuk ketergantungan (*dependency*) di antaranya. Komponen piranti lunak adalah modul berisi code, baik berisi source code maupun binary code, baik library maupun executable, baik yang muncul pada compile time, link time, maupun run time. Umumnya komponen terbentuk dari beberapa class dan/atau package, tapi dapat juga dari komponen-komponen yang lebih kecil. Komponen dapat juga berupa interface, yaitu kumpulan layanan yang disediakan sebuah komponen untuk komponen lain.

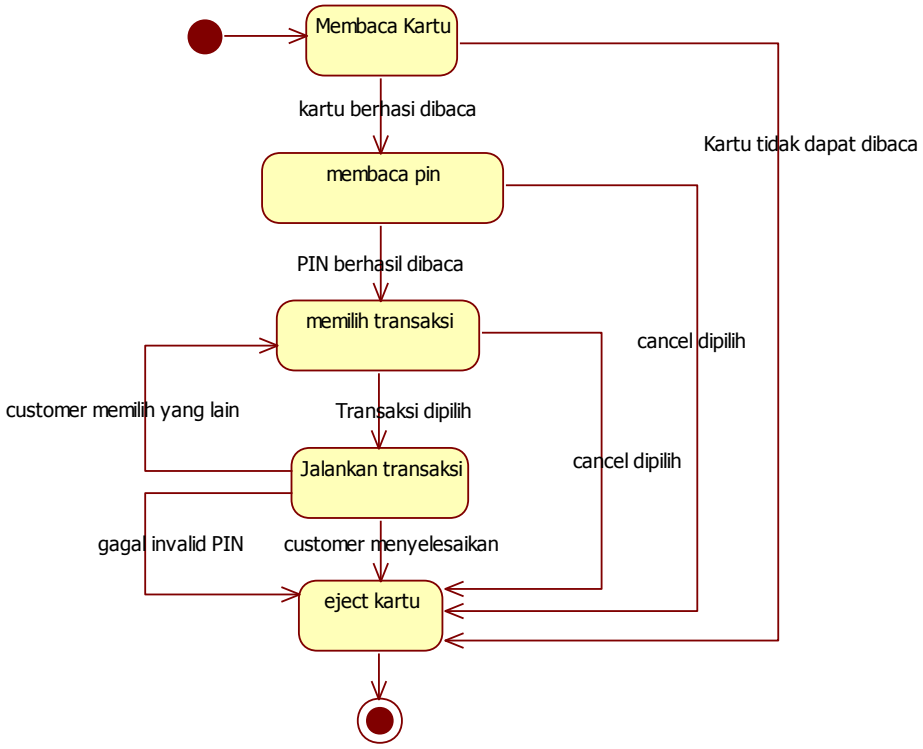


Gambar 2.34 Contoh Component diagram

2.10.2.7. *Statechart* Diagram

Statechart diagram menggambarkan transisi dan perubahan keadaan (dari satu state ke state lainnya) suatu objek pada sistem sebagai akibat dari stimuli yang diterima. Pada umumnya statechart diagram menggambarkan class tertentu (satu class dapat memiliki lebih dari satu statechart diagram).

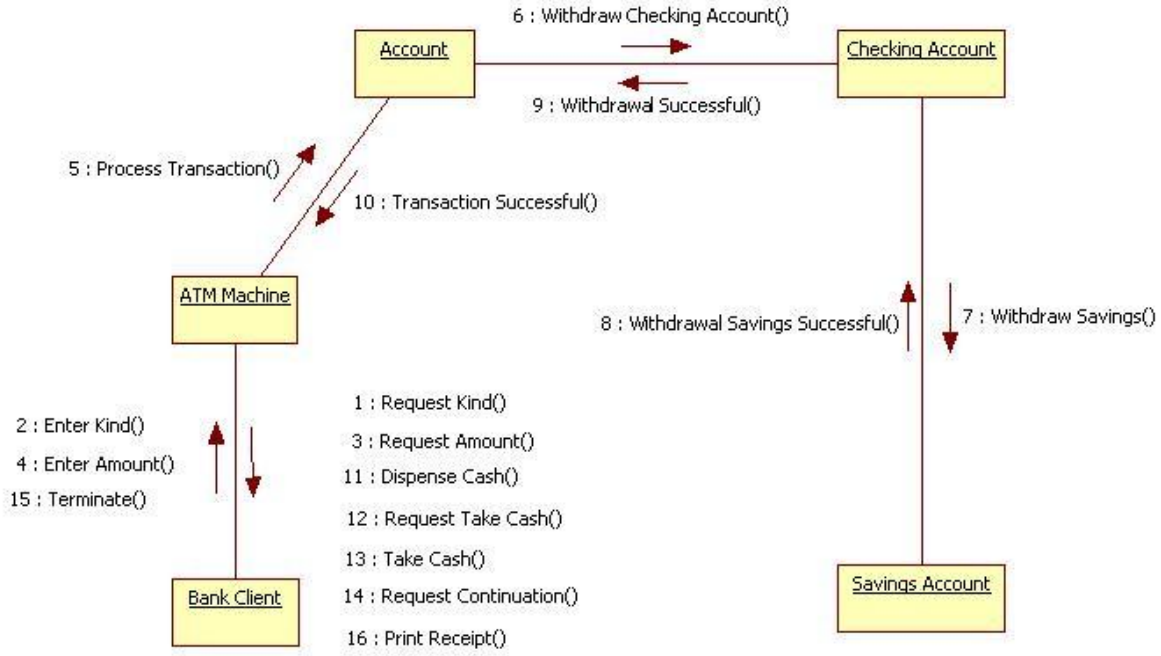
Dalam UML, state digambarkan berbentuk segiempat dengan sudut membulat dan memiliki nama sesuai kondisinya saat itu. Transisi antar state umumnya memiliki kondisi guard yang merupakan syarat terjadinya transisi yang bersangkutan, dituliskan dalam kurung siku. Action yang dilakukan sebagai akibat dari event tertentu dituliskan dengan diawali garis miring. Titik awal dan akhir digambarkan berbentuk lingkaran berwarna penuh dan berwarna setengah.



Gambar 2.35 Contoh Statechart diagram

3.2.2.1. *Collaboration* Diagram

Collaboration diagram juga menggambarkan interaksi antar objek seperti sequence diagram, tetapi lebih menekankan pada peran masing-masing objek dan bukan pada waktu penyampaian *message*. Setiap *message* memiliki *sequence number*, di mana message dari level tertinggi memiliki nomor 1. *Messages* dari level yang sama memiliki *prefiks* yang sama.



Gambar 2.36 Contoh Collaboration diagram