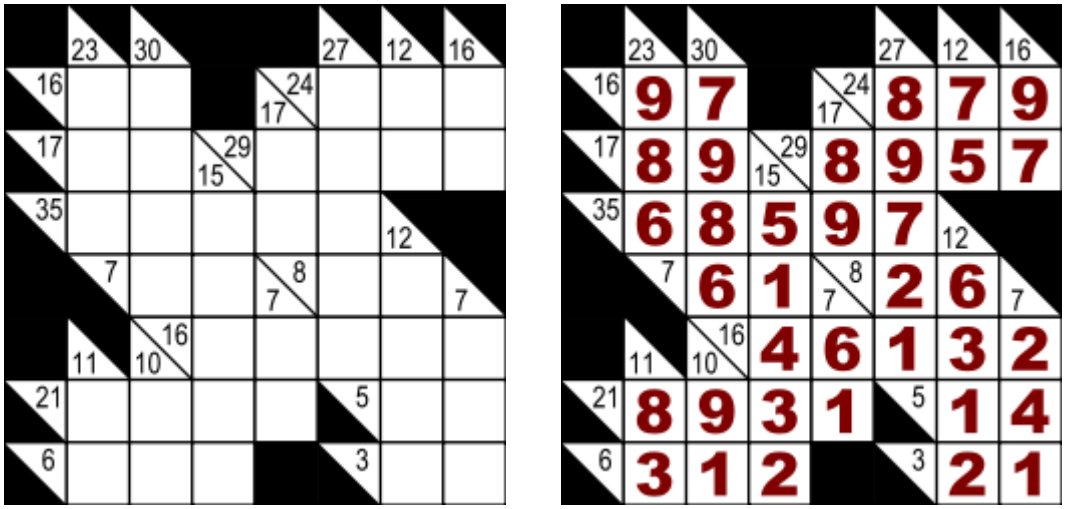
BAB 2
LANDASAN TEORI

2.1 *Puzzle* Kakuro

Kakuro adalah sebuah permainan *puzzle* yang bermula bernama *cross sums. Puzzle*, pertama dikeluarkan pada tahun 1966 oleh *Dell Megazine* di Amerika Serikat. Sepuluh tahun kemudian, *Dell Megazine* memperkenalkan *puzzle* sudoku pada dunia. Maki Kaji presiden dari *Nicoli Puzzle*, pada tahun 1980 membawa masuk *cross games* ke Jepang. Maki Kaji memberi nama *puzzle* buatannya adalah kasan kurosu (penjumlahan silang). Pada tahun 1986, *Nicoli* memberi nama dagang *game* dengan nama kakuro (kependekan dari kasan kurosu). September 2005 adalah titik penentunya, dimana *game puzzle* kakuro diperkenalkan ke barat oleh *The Guardian* dan *The Daily Mail* yang menerbitkan *puzzle kakuro* setiap harinya di Inggris dan setelah itu *game puzzle* kakuro ini menyebar keseluruh dunia [4].

2.1.1 Definisi *Puzzle* Kakuro

Puzzle kakuro adalah permainan *puzzle* dengan ukuran *grid* N x M dengan mempunyai dua jenis kotak yaitu kotak berwarna hitam dan kotak berwarna putih. Pada kedua jenis kotak tersebut mempunyai dua posisi yaitu horizontal dan vertikal. Kotak berwarna hitam digunakan sebagai tanda pembatas dan soal untuk menyelesaikan permainan *puzzle* kakuro, sedangkan kotak berwarna putih digunakan sebagai kotak pengisian jawaban. Pada baris paling atas dan kolom paling kiri pada *grid puzzle* kakuro, hanya berisi kotak berwarna hitam (dapat berfungsi sebagai pembatas atau soal). Aturan dalam pengisisan *puzzle* kakuro ini adalah dengan mengisi angka bilangan bulat dari 1 hingga 9 dengan tidak boleh terdapat angka yang sama pada satu lajur (baris maupun kolom). Nilai angka yang terdapat pada kotak pengisian jawaban yang sudah dijumlahkan harus bernilai sama dengan nilai angka pada kotak soal disetiap lajurnya [5]. Gambar 2.1 adalah sebagian contoh soal dari *puzzle* kakuro serta solusi penyelesaianya [6].

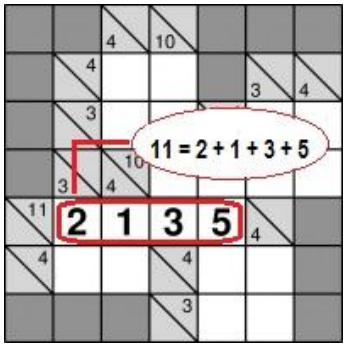


Gambar 2.1 Contoh soal dan solusi penyelesaiannya

2.1.2 Aturan Permainan *Puzzle* Kakuro

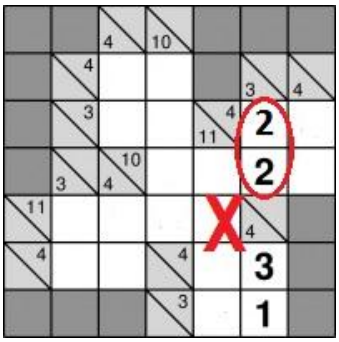
Puzzle kakuro adalah adalah sebuah permainan *puzzle* yang bersifat *puzzle* logika. *Puzzle* ini biasa disebut transliterasi matematis dari *puzzle* silang. Aturan dalam permainan *puzzle* kakuro ini adalah dengan mengisi setiap setiap kotak yang tersedia dengan angka bulat dari 1 hingga 9 dengan beberapa syarat dalam pengisiannya. Adapun syarat-syarat dalam pengisian kotak jawaban adalah sebagai berikut [5]:

1. Setiap kotak jawaban hanya boleh diisikan dengan bilangan bulat dari angka 1 hingga 9.
2. Setiap kotak jawaban yang berurutan dalam satu lajur (lajur yang dimaksud adalah deretan kotak yang berurtutan dalam satu baris atau kolom) tidak boleh memiliki angka yang sama.
3. Isi dari setiap lajur harus memiliki jumlah yang sama pada ujung kiri (baris) atau ujung atas (kolom) lajur tersebut.



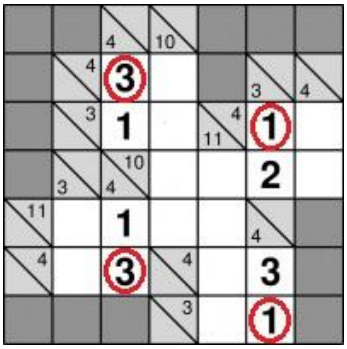
Gambar 2.2 Cara pengisisan dengan jumlah jawaban sesuai dengan soal

Pengisian kotak jawaban seperti pada gambar gambar 2.3 merupakan contoh yang salah karena terdapat dua angka atau lebih pada satu lajur.



Gambar 2.3 Nilai sama pada lajur

Sedangkan pada gambar 2.4 merupakan pengisian kotak jawaban yang benar karena terdapat pada lajur yang berbeda walaupun dalam kotak yang sama.



Gambar 2.4 Nilai sama pada satu lajur yang berbeda

2.2 Algoritma

Algoritma adalah salah satu cabang ilmu komputer yang membahas prosedur penyelesaian suatu permasalahan. Algoritma adalah urutan langkah-langkah dalam menentukan suatu masalah. Algoritma juga dapat didefinisikan dengan deretan langkah komputasi yang mentransformasikan masukan (*input*) menjadi keluaran (*output*). Dapat disimpulkan algoritma adalah serangkaian urutan langkah-langkah atau prosedur untuk menyelesaikan suatu masalah dengan memproses nilai keluaran.

Algoritma dapat ditulis dalam bentuk *pseudocode* ataupun *flowchart. Pseudocode* berasal dari kata *pseudo* yang artinya mirip atau menyerupai dan *code* berarti kode, sehingga *pseudocode* adalah kode yang menyerupai kode program sebenarnya. Sedangkan *flowchart* merupakan bentuk gambar atau diagram yang mempunyai aliran satu atau dua arah secara skuensial.

2.2.1 Definisi Algoritma

Algoritma adalah urutan langkah-langkah logis penyelesaian yang disusun secara matematis dan logis. 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 [7]:

1. Algoritma haruslah benar, artinya algoritma akan memberikan keluaran yang dikehendaki dari sejumlah masukan yang diberikan. Tidak peduli sebagus apapun algoritma, apabila menghasilkan keluaran yang bernilai salah maka algoritma tersebut bukanlah algoritma yang baik.
2. Seberapa baik algoritma yang diterapkan, artinya algoritma yang baik harus mampu memberikan hasil yang sedekat mungkin dengan nilai sebenarnya. Dalam hal ini, hal terpenting pada algoritma untuk menyelesaikan masalah yang memerlukan aproksimasi hasil (hasil yang berupa pendekatan).
3. Efisiensi algoritma, artinya efisiensi algoritma dapat ditinjau dari dua hal yaitu efisiensi waktu dan memori.

2.2.2 Sejarah Algortima

Kata algoritma berasal dari latinisasi nama seorang ahli matematika dari Uzbekistan Al Khawārizmi. Dahulu, orang hanya mengenal kata *algorism* yang berarti proses menghitung dengan angka arab. Para ahli bahasa berusaha menemukan asal kata *algorism,* dan akhirnya para ahli sejarah matematika berhasil menemukan asal kata yang berasal dari penulis buku arab yang terkenal yaitu Abu Ja'far Muhammad Ibnu Musa Al Khawārizmi. Al Khawārizmi dibaca orang barat menjadi *algorism.* Al Khawārizmi menulis kitab *Al Jabar Wal-Muqabala* yang artinya "Buku Pemograman Dan Pengurangan" (*The Book of Restoration And Reducation*). Perubahan kata dari *algorism* menjadi kata *algoritm* muncul dikarenakan kata *algorism* sering dikelirukan dengan *arithmetic,* sehingga akhiran *-sm* berubah menjadi *–thm.* Seiring perkembangan, 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 Algoritma *Backtracking*

Algoritma *backtracking* pertama kali diperkenalkan oleh D.H. Lehmer pada tahun 1950. Dalam perkembangannya, beberapa ahli seperti RJ Walker, Golomb, dan Baumert menyajikan uraian umum tentang algoritma *backtracking* dan penerapannya dalam berbagai persoalan dan aplikasi. Algoritma *backtracking* merupakan salah satu metode pemecahan masalah yang termasuk dalam strategi yang berbasis pencarian pada ruang status [8].

Algoritma *backtracking* bekerja secara rekursif dan melakukan pencarian solusi persoalan secara sistematis pada semua kemungkinan solusi yang ada. Algoritma ini berbasis *Depth-First Search* (DFS) dalam pencarian solusi dengan melakukan penelusuran suatu struktur berbentuk pohon berakar secara *preorder.* Proses ini dicirikan dengan ekspansi simpul terdalam lebih dahulu sampai tidak ditemukan lagi suksesor dari suatu simpul.

Mekanisme penyelesaian dengan menggunakan algoritma *backtracking* berprinsip pada metode rekursif. Untuk menyelesaikan keseluruhan masalah,

dibutuhkan sebuah solusi untuk permasalahan pertama kemudian permasalahan-permasalahan lainnya akan dicoba untuk diselesaikan secara rekursif berdasarkan solusi pertama. Apabila pada kemungkinan solusi yang dicari tidak ditemukan, maka dilakukan *backtrack* untuk menguji kemungkinan solusi selanjutnya. Proses *backtrack* akan selesai ketika tidak ada lagi solusi yang mungkin untuk diselesaikan pada permasalahan awal. Dalam penerapannya, algoritma *backtracking* mempunyai properti umum didalmnya yang harus dibutuhkan :

1. Solusi Persoalan.

   Solusi dinyatakan sebagai *vektor* dengan n-*tuple* :

   $X = (v, v_{2,} ..., v_n)$, $x_i \in s_i$

   Mungkin saja $s_i = s_2 \ldots = s_n$

   Contoh $s_i = \{0,1\}$, $x_i = 0$ atau 1

2. Fungsi Pembangkit nilai x_k.

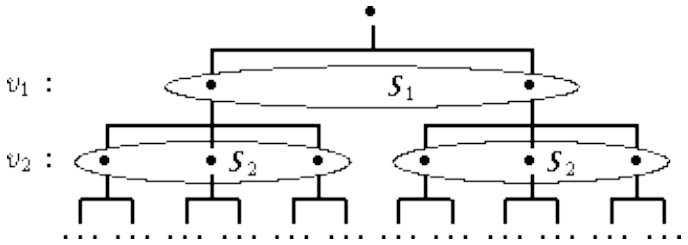
   Dinyatakan sebagai predikat : T(k)

   T(k)membangkitkan nilai untuk x_k yang merupakan komponen vektor solusi.

3. Fungsi Pembatas.

   Dinyatakan sebagai : B $(v_1, v_{2,} \ldots, v_k)$ mengarah ke solusi.

Jika *true*, maka pembangkit nilai untuk x_{k+1} dilanjutan, dan jika *false* maka $(x_1, x_{2,} \ldots, x_k)$ akan dibuang dan tidak dapat dipertimbangkan kembali dalam pencarian solusi. Penjelasan lain dapat dilihat melalui gambar berikut :



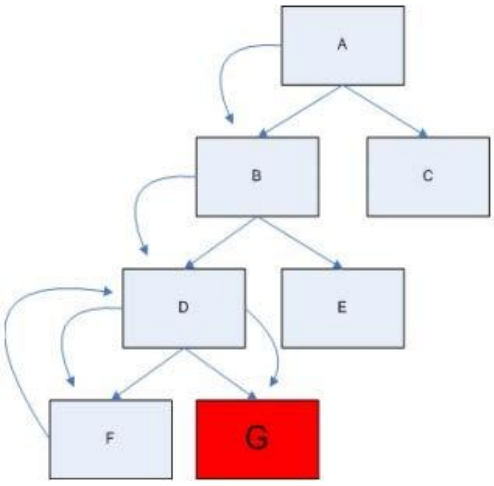
Gambar 2.1 *Tree* pencarian solusi

Solusi persoalan adalah kemungkinan solusi yang didapatkan dari permasalahan yang diberikan, sedangkan fungsi pembatas merupakan fungsi yang akan menentukan langkah selanjutnya berupa penerusan pencarian solusi ataupun melakukan *backtrack.*

2.3.1 Prinsip Algoritma *Backtracking*

Algoritma *backtracking* pada pencarian kemungkinan solusi dibuat dalam bentuk pohon ruang status dan algoritma akan menelusuri pohon tersebut secara *Depth-First Seacrh* (DFS) sampai ditemukan solusi yang layak. Langkah-langkah pencarian solusi pada algoritma *backtracking* adalah sebagai berikut [8]:

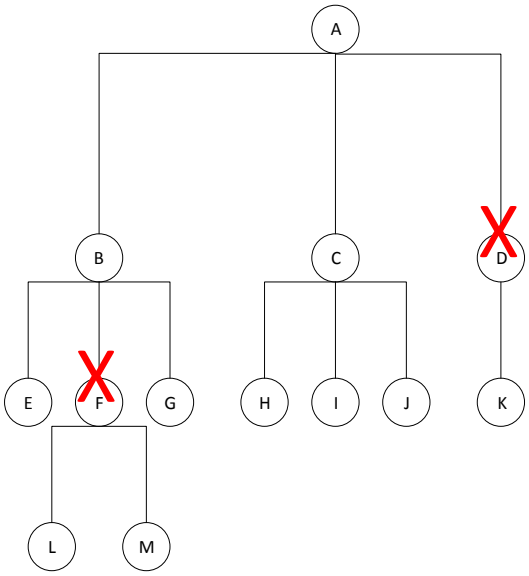
1. Solusi dicari dengan membentuk lintasan dari akar ke daun. Aturan yang dipakai adalah mengikuti metode pencarian *Depth-First Search* (DFS). Simpul yang telah dilahirkan dinamakan simpul hidup dan simpul hidup yang diperluas dinamakan simpul-E (*expand node*).
2. Jika lintasan yang diperoleh dari perluasan simpul-E tidak mengarah ke solusi, maka simpul itu akan menjadi simpul mati yang tidak dapat diperluas lagi.
3. Jika posisi terakhir ada disimpul mati, maka pencarian dilakukan dengan membangkitkan simpul anak yang lainnya. Apabila tidak ada simpul anak, maka dilakukan *backtracking* ke simpul induk.
4. Pencarian dihentikan jika telah menemukan solusi atau tidak ada simpul hidup yang dapat ditemukan.



Gambar 2.6 Contoh pohon ruang status algoritma *backtracking*

Salah satu fungsi yang menjadi ciri khas algoritma *backtracking* adalah fungsi pemangkasan *(pruning).* Jika tahap-tahap pencarian solusi suatu masalah dipresentasikan kedalam bentuk pohon solusi, maka proses pemangkasan akan

dilakukan terhadap simpul-simpul yang tidak mengarah kesolusi. Jika suatu simpul telah dipangkas, simpul-simpul yang menjadi anak dan turunan dari simpul tersebut secara langsung tidak akan diproses, karena pemangkasan sebuah simpul seperti membuang seluruh lintasan yang berada dibawah simpul tersebut.



Gambar 2.7 Proses pemangkasan pada ruang status *(pruning)*

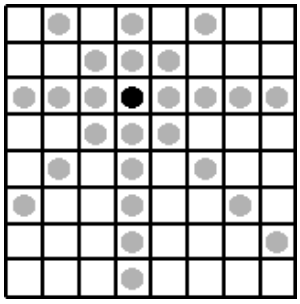
Berdasarkan gambar 2.7, simpul yang dipangkas adalah simpul D dan F. Akibat pemangkasan tersebut, simpul K yang merupakan anak simpul D serta simpul L dan M yang merupakan anak simpul F tidak akan diproses [2].

2.3.2 Kegunaan *Backtracking*

Penggunaan terbesar *backtrack* adalah untuk membuat *artificial intellegence* (AI) pada *puzzle games.* Dengan algoritma ini, program dapat menghasilkan pohon sampai dengan kedalaman tertentu dari *current status* dan memilih solusi yang akan membuat langkah-langkah pemain dalam melakukan tindakan untuk menghasilkan pohon solusi baru dengan jumlah pilihan langka terbanyak.

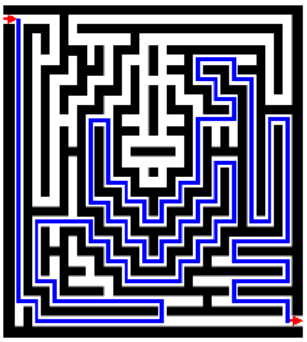
Beberapa kegunaan yang cukup terkenal dari algoritma *backtracking* pada suatu masalah statik adalah pemecahan masalah *N-Queen problem* dan *maze solver* [1]. *N-Queen problem* adalah bagaimana cara meletakan bidak *queen* catur sebanyak N buah pada papan catur atau pada papan ukuran (N x N) yang dapat

memangsa bidak lainnya dengan satu gerakan. Meskipun terdapat kemungkinan lebih dari satu solusi, tetapi pencarian semua solusi biasanya tidak terlalu diperlukan.



Gambar 2.8 Contoh *N-Queen problem*

Pada *maze solver* cara kerjanya adalah bagaimana mencari jalan keluar dari suatu *maze* (labirin). Pada *maze* yang sederhana dimana *field* yang dibentuk dapat dipresentasikan dalam bentuk *biner* dan pada setiap petak maksimal terdapat 4 kemungkinan yaitu atas, kanan, bawah, dan kiri. Untuk masalah ini biasanya solusi pertama yang ditemukan bukanlah solusi yang paling optimal, sehingga untuk mendapatkan hasil yang optimal dibutuhkan pencarian terhadap seluruh kemungkinan solusi yang ada.



Gambar 2.9 Contoh *maze solver*

2.4 Kecerdasan Buatan

Kecerdasan buatan merupakan sub-bidang ilmu komputer (*computer science*) yang khusus ditujukan untuk membuat perangkat lunak dan perangkat keras yang sepenuhnya bisa menirukan beberapa fungsi otak manusia, atau cabang ilmu komputer yang mempelajari otomatisasi tingkah laku cerdas

(*intelligent*). Kecerdasan buatan didasarkan pada teori suara (*sound theoretical*) dan prinsip-prinsip aplikasi dari bidangnya. Prinsip-prinsip ini meliputi struktur data yang digunakan dalam representasi pengetahuan algoritma yang diperlukan untuk mengaplikasikan pengetahuan serta bahasa dan teknik pemograman yang digunakan dalam mengimplementasikannya.

Beberapa para ahli, mendefinisikan kecerdasan buatan (*artificial intelligence*) secara berbeda-beda tergantung pada sudut pandangnya masing-masing. Para ahli ada yang mendefinisikan kecerdasan buatan (*artificial intelligence*) fokus kepada logika berpikir manusia saja, ada juga yang mendefinisikan kecerdasan buatan (*artificial intelligence*) secara lebih luas terhadap tingkah laku manusia. Stuart Russel dan Peter Norvig mengelompokan definisi kecerdasan buatan (*artificial intelligence*) yang diperoleh dari beberapa *textbook* berbeda kedalam empat kategori, yaitu [9]:

1. *Thinking humanly : The cognitive modeling approach*

   Pendekatan *thinking humanly* dilakukan dengan dua cara sebagai berikut :

   a. Melalui intropeksi dimana proses ini adalah menangkap pemikiran-pemikiran yang didapat pada saat berfikir.
   b. Melalui eksperimen-eksperimen psikologi.

2. *Acting humanly : The turing test approach*

   Pada tahun 1950, Alan Turing membuat sebuah eksperimen program penginterogasian untuk menguji kemampuan komputer pada manusia melalui *teletype* (komunikasi berbasis teks jarak jauh). Dalam hal ini, apabila *integrator* tidak dapat membedakan yang diinterogasi adalah manusia atau komputer maka program tersebut lolos dari *turing test. Turing test* sengaja menghindari interaksi fisik antara *integrator* dan komputer, karena simulasi fisik manusia tidak memerlukan intelejensia.

3. *Thinking rationally : The laws of thought approach*

   Terdapat dau masalah pada pendekatan ini, yaitu :

   a. Tidak mudahnya untuk membuat pengetahuan informal dan menyatakan pengetahuan tersebut kedalam *formal term* yang diperlukan oleh notasi

logika, khususnya ketika pengetahuan tersebut memiliki kepastian kurang dari 100%.

b. Terdapat perbedaan besar antara dapat memecahkan masalah "dalam prinsip" dan memecahkannya "dalam dunia nyata".

4. *Acting rationally : The rationally agent approach*

   Membuat inferensi yang logis merupakan bagian dari suatu *rational agent.* Hal ini disebabkan satu-satunya cara untuk melakukan aksi secara rasional adalah dengan menalar secara logis. Dengan menalar secara logis, maka bisa didapatkan kesimpulan bahwa aksi yang diberikan akan mencapai tujuan atau tidak. Apabila mencapai tujuan, maka *agent* dapat melakukan aksi berdasarkan kesimpulan tersebut.

Thinking humanly dan *acting humanly* adalah dua definisi dalam arti yang sangat luas. Hingga saat ini, pemikiran manusia yang diluar rasio, yakni refleks dan intuitif (perasaan), belum dapat ditirukan oleh komputer. Sedangkan definisi *thinking rationally* jauh lebih sempit dari *acting rationally* dengan pendekatan *rational agent.* Hal ini berdasarkan pemikiran bahwa komputer dapat melakukan penlaran secara logis dan juga dapat melakukan aksi secara rasional berdasarkan hasil penalaran.

2.5 *Unified Modeling Language* (UML)

Unified Modeling Language merupakan salah satu alat bantu yang dapat digunakan dalam bahasa pemograman yang berorientasi objek, saat ini UML akan mulai menjadi standar masa depan bagi industri pengembangan sistem/perangkat lunak yang berorientasi objek sebab pada dasarnya UML digunakan oleh banyak perusahaan raksasa seperti IBM, Microsoft, dan sebagainya. Berikut ini beberapa definisi UML [10]:

Unified Modeling Language merupakan metode pengembangan perangkat lunak (sistem informasi) dengan menggunakan metode grafis serta merupakan bahasa untuk visualisasi, spesifikasi, konstruksi serta dokumentasi.

1. *Unified Modeling Language* (UML) adalah bahasa yang telah menjadi standar untuk visualisasi, menetapkan, membangun dan mendokumentasikan arti suatu sistem perangkat lunak.
2. *Unified Modeling Language* (UML) dapat didefinisikan sebagai sebuah bahasa yang telah menjadi standar dalam industri untuk visualisasi, merancang dan mendokumentasikan sistem perangkat lunak.
3. *Unified Modeling Language* (UML) merupakan *standard modeling language* yang terdiri dari kumpulan-kumpulan diagram, dikembangkan untuk membantu para pengembang *system* dan *software* agar bisa menyelesaikan tugas-tugas seperti spesifikasi, visualisasi, desain arsitektur, konstruksi, simulasi dan testing, serta dokumentasi.

Berdasarkan beberapa pendapat yang dikemukakan diatas dapat ditarik kesimpulan bahwa *Unified Modeling Language* (UML) adalah sebuah bahasa yang berdasarkan grafik atau gambar untuk memvisualisasikan, menspesifikasikan, membangun, dan pendokumentasian dari sebuah sistem pengembangan perangkat lunak berbasis *Object Oriented Programming* (OOP).



Gambar 2.10 Logo *Unified Modeling Language* (UML)

2.5.1 Diagram *Unified Modeling Language* (UML)

Beberapa literatur menyebutkan bahwa UML menyediakan sembilan jenis diagram, yang lain menyebutkan delapan karena ada beberapa diagram yang digabung, misalnya diagram komunikasi, diagram urutan dan diagram pewaktuan digabung menjadi diagram interaksi. Namun demikian model-model itu dapat dikelompokkan berdasarkan sifatnya yaitu statis atau dinamis. Jenis diagram itu antara lain [10] :

1. Diagram Kelas (*Class Diagram*)

   Diagram kelas menggambarkan struktur sistem dari segi pendefinisian kelas-kelas yang akan dibuat untuk membangun sistem. Kelas memiliki apa yang disebut atribut dan metode atau operasi.

   a. Atribut merupakan variabel-variabel yang dimiliki oleh suatu kelas.
   b. Operasi atau metode adalah fungsi-fungsi yang dimiliki oleh suatu kelas.

   Diagram kelas dibuat agar pembuat program atau *programmer* membuat kelas-kelas sesuai rancangan di dalam diagram kelas agar antara dokumentasi perancangan dan perangkat lunak menjadi sinkron. Berikut adalah contoh dari diagram kelas.



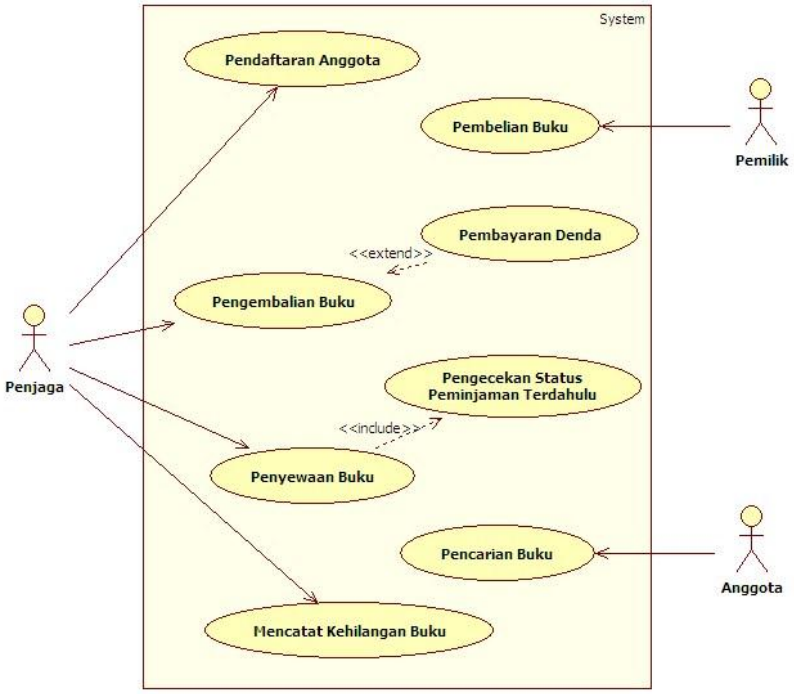
Gambar 2.11 Contoh *class diagram*

2. Diagram *Use Case*

   Diagram *use case* merupakan pemodelan untuk kelakuan *(behaviour)* sistem informasi yang akan dibuat. *Use case* mendeskripsikan sebuah interaksi antara satu atau lebih aktor dengan sistem informasi yang akan dibuat. Secara kasar, *use case* digunakan untuk mengetahui fungsi apa saja yang ada di dalam sebuah sistem informasi dan siapa saja yang berhak menggunakan fungsi-fungsi itu. Syarat penamaan pada *use case* adalah nama didefinisikan sesimpel mungkin dan dapat dipahami. Ada dua hal utama pada *use case* yaitu pendefinisian apa yang disebut aktor dan *use case.*

   a. Aktor merupakan orang, proses, atau sistem lain yang berinteraksi dengan sistem informasi yang akan dibuat diluar sistem informasi yang akan dibuat itu sendiri, jadi walaupun simbol dari aktor adalah gambar orang, tapi aktor belum tentu merupakan orang.

b. *Use case* merupakan fungsionalitas yang disediakan sistem sebagai unit-unit yang saling bertukar pesan antar unit atau aktor.



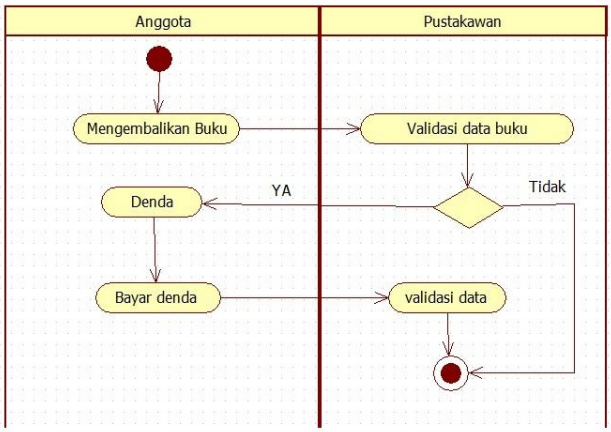
Gambar 2.12 Contoh *use case diagram*

3. Diagram Aktivitas (*Activity Diagram*)

Diagram aktivitas menggambarkan *workflow* (aliran kerja) atau aktivitas dari sebuah sistem atau proses bisnis atau menu yang ada pada perangkat lunak. Yang perlu diperhatikan disini adalah bahwa diagram aktivitas menggambarkan aktivitas sistem bukan apa yang dilakukan aktor, jadi aktivitas yang dapat dilakukan oleh sistem. Diagram aktivitas juga banyak digunakan untuk mendefinisikan hal-hal berikut :

a. Rancangan proses bisnis dimana setiap urutan aktivitas yang digambarkan merupakan proses bisnis sistem yang didefinisikan.
b. Urutan atau pengelompokkan tampilan dari *system* / *user interface* dimana setiap aktivitas dianggap memiliki sebuah rancangan antarmuka tampilan.
c. Rancangan pengujian dimana setiap aktivitas dianggap memerlukan sebuah pengujian yang perlu didefinisikan kasus ujinya.

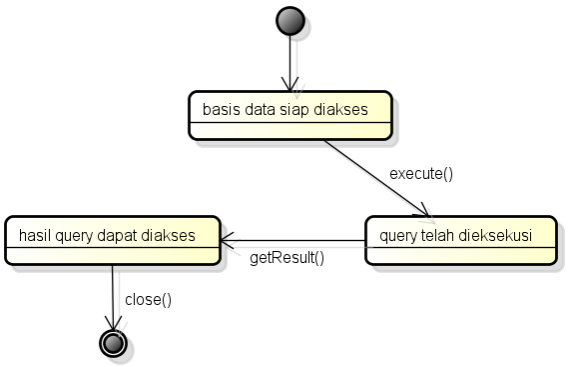
d. Rancangan menu yang ditampilkan pada perangkat lunak.



Gambar 2.13 Contoh *activity diagram*

4. Diagram *State Machine* (*Statechart Diagram*)

State machine diagram atau *statechart diagram* atau dalam bahasa Indonesia disebut diagram mesin status atau sering juga disebut diagram status digunakan untuk menggambarkan perubahan status atau transisi status dari sebuah mesin atau sistem atau objek. Jika diagram sekuen digunakan untuk interaksi antar objek maka diagram status digunakan untuk interaksi di dalam sebuah objek. Perubahan tersebut digambarkan dalam suatu graf berarah.

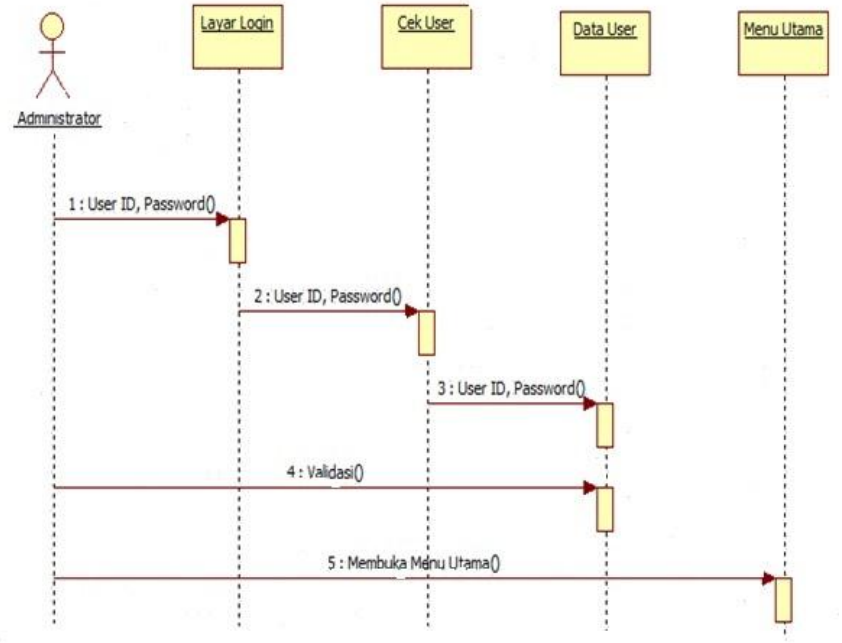


Gambar 2.14 Contoh *state chart diagram*

5. Diagram Sekuen (*Squence Diagram*)

Diagram sekuen menggambarkan kelakuan pada objek *use case* dengan mendeskripsikan waktu hidup objek dan *message* yang dikirimkan dan diterima antar objek. Oleh karena itu untuk menggambar diagram sekuen maka harus diketahui objek-objek yang terlibat dalam sebuah *use case* beserta

metode-metode yang dimiliki kelas yang diinstansiasi menjadi objek itu. Membuat diagram sekuen juga dibutuhkan untuk melihat skenario yang ada pada *use case*. Banyaknya diagram sekuen yang harus digambar adalah minimal sebanyak pendefinisian *use case* yang memiliki proses sendiri atau yang penting semua *use case* yang telah didefinisikan interaksi jalannya pesan sudah dicakup pada diagram sekuen sehingga semakin banyak *use case* yang didefinisikan maka diagram sekuen yang harus dibuat juga semakin banyak.



Gambar 2.15 Contoh *squence diagram*

2.6 *Java*

Java adalah salah satu bahasa pemrograman komputer yang berorientasi objek, yang diciptakan oleh satu tim dari perusahaan *Sun Microsystem*, perusahaan *workstation* UNIX (*Sparc*) yang cukup terkenal. *Java* diciptakan berdasarkan bahasa C++, dengan tujuan *platform independent* (dapat dijalankan pada berbagai jenis *hardware* tanpa kompilasi ulang), dengan slogan "*Write Once, Run Anywhere* (WORA), dan *Java* pada hakikatnya lebih sederhana dan memakai objek secara murni dibanding dengan bahasa pemrograman C++.

Asal usul *java* dimulai pada tahun 1991 ketika *Sun Microsystem* melakukan penelitian terhadap berbagai produk elektronika. James Gosling (kepala proyek penelitian) beserta Patrick Naughton ditugaskan untuk merancang perangkat lunak aplikasi yang independen, tidak bergantung pada jenis perangkat keras, agar bisa dipakai pada berbagai peralatan elektronik itu. James Gosling kemudian memakai bahasa pemrograman C++ untuk menulis beberapa aplikasi untuk peralatan mikro, namun ternyata dia mengalami banyak kesulitan. Setiap kali sebuah peralatan mikro menggunakan jenis *microchip* yang berbeda, program yang digunakannya harus dikompilasi ulang. Berdasarkan hal itu kemudian munculah ide dari James Gosling untuk menciptakan sebuah bahasa baru, yang ditulis berdasarkan C++ namun dengan beberapa perbaikan. Bahasa ini kemudian disebut *Oak* (nama pohon yang terlihat dari jendela ruangan kerja James Gosling), yang digunakan untuk membuat perangkat lunak cerdas bagi peralatan elektronika buatan *Sun Mirosystem.*

Pada tahun 1994, ketika Web mulai populer nama bahasa *Oak* kemudian diganti menjadi *Java* (salah satu sebabnya ternyata nama *Oak* telah terlebih dahulu dipakai untuk hal lain). Menurut gosip, Gosling bingung mencari nama untuk bahasa baru tersebut karena setiap nama yang dipikirkannya untuk nama bahasa pemrogramannya ternyata sudah ada yang memakai, sampai suatu ketika dia mampir di sebuah kafe untuk minum kopi, dan ia meminta "*Java"*. Timbullah idenya untuk memakai nama *Java*. Menurut Gosling nama ini cukup *cool*.

Java kemudian dipakai untuk membuat sebuah *browser* yang bernama *WebRunner*. Setelah berhasil maka browser ini kemudian dinamakan *HotJava* dengan simbol secangkir kopi panas.



Gambar 2.16 Logo *Java* [11]

2.7 *Greenfoot*

Greenfoot merupakan *software* pembelajaran pemrograman berorientasi objek, dengan berbasis pemrograman *java* yang memungkinkan kita untuk membuat game 2 dimensi (2D). Greenfoot diprakarsai oleh Michael Kolling pada tahun 2003. *Prototype* pertama dibangun oleh Michael Kolling dan Paul Henrikson pada tahun 2003-2004. Pembangunan dilanjutkan pada tahun 2005 oleh kelompok Blue J lain dari University of Kent dan Deakin University. *Greenfoot* versi 1.0 pertama dirilis pada tanggal 31 Mei 2006. Pada bulan Maret 2009, proyek *greenfoot* menjadi *software open source* dan berlisensi dibawah GNU GPL.

Untuk menjalankan *greenfoot* ini diharuskan sudah menginstalasi program *java* 5 (JDK 1.5) atau program *java* terbaru. Didalam *greenfoot* terdapat *project* yang dinamakan *scenario*. Dalam *project* tersebut, kita dapat membuat dua macam tipe *class* dari *library greenfoot*, yaitu *class world* dan *class actor*. *World* merupakan latar dari game yang akan dibuat. *Actor* adalah kelas yang berfungsi sebagai karakter pada game yang akan dibuat.



Gambar 2.17 Logo *Greenfoot* [12]