BAB II

LANDASAN TEORI

2.1 Konsep Dasar Algoritma

2.1.1 Sejarah Algoritma

Dilihat dari asal usul kata, kata algoritma sendiri mempunyai sejarah yang aneh. Kata ini tidak muncul dalam kamus *Webster* sampai akhir tahun 1957. Orang hanya menemukan kata *algorism* yang berarti proses menghitung dengan angka Arab [KNU73]. Seseorang dikatakan *algorist* jika menggunakan angka Arab. Para ahli berusaha menemukan asal kata *algorism* ini namun hasilnya kurang memuaskan. Akhirnya para ahli sejarah matematika menemukan asal mula kata tersebut. Kata *algorism* berasal dari nama penulis buku Arab yang terkenal, yaitu Abu Ja'far Muhammad ibnu Musa al-Khuwarizmi (al-Khuwarizmi dibaca orang Barat menjadi *algorism*).

Al-Khuwarizmi menulis buku yang berjudul *Kitab al jabar wal-muqabala*, yang artinya "Buku pemugaran dan pengurangan" (*The book of restoration and reduction*). Dari judul buku itu diperoleh juga akar kata "aljabar" (*algebra*). Perubahan dari kata *algorism* menjadi *alghorithm* 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 aslinya. Dalam bahasa Indonesia, kata *alghorism* diserap menjadi algoritma.

2.1.2 Definisi Algoritma

Menurut Rinaldi Munir (2005 : 176) *"Algoritma adalah urutan logis langkah-langkah penyeleseian masalah yang disusun secara sistematis"*. Alur pemikiran dalam menyelesaikan suatu pekerjaan yang dituangkan secara tertulis. Yang ditekankan pertama adalah alur pikiran, sehingga algoritma seseorang dapat juga berbeda dari algoritma orang lain. Sedangkan penekanan kedua adalah tertulis, yang artinya dapat berupa kalimat, gambar, atau tabel tertentu.

Algoritma dapat dituliskan dalam berbagai notasi, misalnya dalam notasi kalimat-kalimat deskriptif. Dengan notasi kalimat deskriptif, deskripsi setiap langkah dijelaskan dengan bahasa sehari-hari secara jelas. Setiap langkah biasanya diawali dengan kata kerja seperti 'baca', 'hitung', 'masukan', 'bagi', 'ganti', dan sebagainya. Sedangkan pernyataan bersyarat dinyatakan dengan 'jika','maka', dan sebagainya.

2.2 Konsep Dasar Graf

2.2.1 Sejarah Graf

Menurut catatan sejarah, masalah jembatan *Konigsberg* adalah masalah yang pertama kali menggunakan graf (tahun 1736). Di kota *konigsberg* (sebelah timur negara bagian Prussia, Jerman), sekarang bernama kota *Kaliningrad*, terdapat sungai Pregal yang mengalir mengintari pulau *Kneiphof* lalu bercabang menjadi dua buah anak sungai. Ada tujuh buah jembatan yang menghubungkan daratan yang dibelah oleh sungai tersebut. Masalah jembatan *Konigsberg* adalah : apakah mungkin melalui ketujuh buah jembatan itu masing-masing tepat satu kali,

dan kembali lagi ke tempat semula. Sebagian penduduk kota tersebut sepakat bahwa memang tidak mungkin melalui setiap jembatan itu hanya sekali dan kembali lagi ke tempat asal mula keberangkatan, tetapi mereka tidak dapat menjelaskan mengapa demikian jawabannya, kecuali dengan cara coba-coba. Tahun 1736, seorang matematikawan Swiss, L.Euler, adalah orang pertama yang berhasil menemukan jawaban masalah itu dengan pembuktian yang sederhana. Ia memodelkan masalah ini kedalam graf. Daratan (titik-titik yang dihubungkan oleh jembatan) dinyatakannya sebagai titik (noktah) yang disebut simpul (vertex) dan jembatan dinyatakan sebagai garis yang disebut sisi (edge). Setiap titik diberi label huruf A, B, C, dan D. Graf yang dibuat oleh Euler diperlihatkan pada gambar berikut :

Jawaban yang dikemukakan oleh Euler adalah : orang tidak mungkin melalui ketujuh jembatan itu masing-masing satu kali dan kembali lagi ke tempat asal keberangkatan jika derajat setiap simpul tidak seluruhnya genap. Yang dimaksud dengan derajat adalah banyaknya garis yang bersisian dengan noktah. Sebagai contoh, simpul C memiliki 3 derajat karena ada tiga buah garis yang bersisian dengannya, simpul B dan D juga berderajat dua, sedangkan simpul A berderajat 5. Karena tidak semua simpul berderajat genap, maka tidak mungkin dilakukan perjalanan berupa sirkuit (yang dinamakan sirkuit Euler) pada graf tersebut.

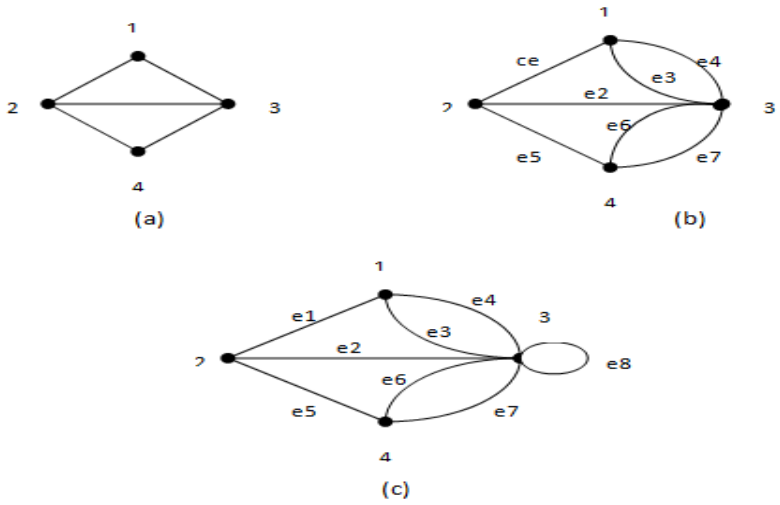
2.2.2 Definisi Graf

"Graf secara umum bisa didefinisikan sebagai kumpulan titik (nodes atau vertices) dan garis (arcs atau edges)" (P.Insap Santosa, 2004 : 497). Karena garis selalu diawali dari suatu titik dan diakhiri pada titik yang lain, maka garis bisa dituliskan sebagai pasangan antara dua titik.

Graf *G* didefinisikan sebagai pasangan himpunan *(V,E)*, yang dalam hal ini : *V* = himpunan berhingga dan tidak kosong dari simpul-simpul (*vertices* atau *node*) = { $v_1, v_2, ..., v_n$ dan *E* = himpunan sisi (*edges* atau *arcs*) yang menghubungkan sepasang simpul = { $e_1, e_2, ..., e_n$ } atau dapat ditulis singkat notasi *G = (V, E .)*. Simpul pada graf dapat dinomori dengan huruf, seperti *v, w,...,*dengan bilangan asli 1, 2, 3,…, atau gabungan keduanya. Sedangkan sisi yang menghubungkan simpul v_i dengan simpul v_j dinyatakan dengan pasangan (v_i, v_j) atau dengan lambang $e_1, e_2, ...$ Dengan kata lain, jika *e* adalah sisi yang menghubungkan simpul v_i dengan simpul v_j, maka e dapat ditulis sebagai :

$$e = (v_i, v_j)$$

Berikut adalah contoh gambar graf :



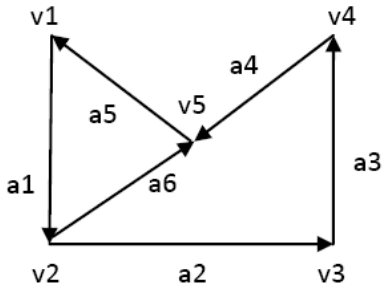
Gambar 2.1 Graf sederhana (a), Graf Ganda (b), Graf semu (c)

Sisi pada graf dapat mempunyai orientasi arah. Berdasarkan orientasi arah pada sisi, maka secara umum graf dibedakan atas 2 jenis yaitu :

1. Graf Berarah
2. Graf Tidak Berarah

2.2.3 Graf berarah

Graf yang setiap sisinya diberikan orientasi arah disebut sebagai graf berarah *(directed graf)*. Menyebut sisi berarah lebih sering dengan sebutan busur (*arc*). Pada graf berarah, (v_j, v_k) dan (v_k, v_j) menyatakan dua buah busur yang berbeda, dengan kata lain (v_j, v_k) tidak sama dngan (v_k, v_j). Untuk busur (v_j, v_k), simpul v_j dinamakan simpul asal (*initial vertex*) dan simpul v_k dinamakan simpul terminal (*terminal vertex*). Pada gambar 2.3 adalah contoh graf berarah. Graf berarah sering dipakai untuk menggambarkan aliran proses, peta lalu lintas suatu kota (jalan searah atau dua arah) dan sebagainya.

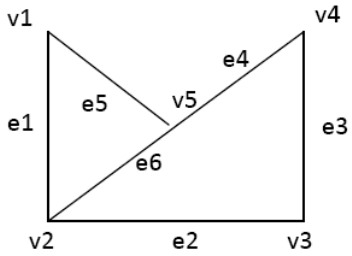


Gambar 2.2 Graf Berarah

Digraf pada gambar 2.3 menunjukan graf berarah dengan himpunan simpul V(G) = $\{v1, v2, v3, v4, v5\}$ dan himpunan busur A(G) = $\{a1, a2, a3, a4, a5, a6\}$ yaitu pasangan terurut dari $\{(v1, v2), (v2, v3), (v3, v4), (v4, v5), (v5, v1), (v2, v5)\}$.

2.2.4 Graf tak berarah (*undirect graph*)

Graf yang sisinya tidak mempunyai orientasi arah disebut graf tak berarah. Pada graf tak berarah, urutan pasangan simpul yang dihubungkan oleh sisi yang sama. Graf pada gambar 2.3 adalah graf tak berarah.

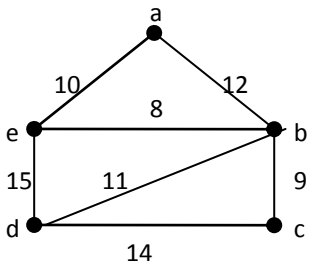


Gambar 2.3 Graf Tidak Berarah

Graf pada gambar 2.3 menunjukan graf tidak berarah dengan himpunan simpul, V(G) = $\{v1, v2, v3, v4, v5\}$ dan himpunan sisi E(G) = $\{e1, e2, e3, e4, e5, e6\}$ yaitu pasangan terurut dari $\{(v1, v2), (v2, v3), (v3, v4), (v4, v5), (v5, v1), (v5, v2)\}$.

2.2.5 Graf Berbobot

Graf berbobot (*weighted graph*) adalah graf yang setiap sisinya diberi sebuah nilai (bobot). Bobot tiap sisi dapat menyatakan jarak antara dua buah kota atau panjang suatu jalan pada sebuah kota, biaya perjalanan antara dua buah kota, ongkos produksi, dan sebagainya. Gambar 2.4 adalah contoh graf berbobot dan tidak berarah.



Gambar 2.4 Graf Berbobot

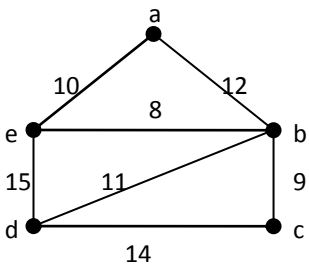
2.2.6 Representasi Graf

Pemrosesan graf dengan program komputer, memerlukan representasi graf dalam memori. Ada beberapa representasi graf yang mungkin untuk dilakukan, yaitu :

1. Matriks Ketetanggaan *(Adjacency)*

   Matriks ketetanggaan adalah representasi graf yang paling umum digunakan. Misalkan G = (V, E) adalah graf dengan n simpul, n ≥ 1. Matriks ketetanggaan G adalah matriks yang berukuran n x n. Bila matriks tersebut dinamakan matriks A = [a_{ij}] maka bernilai 1 jika simpul i dan j bertetanggaan dan bernilai 0 jika simpul i dan simpul j tida bertetangga. Matriks ketetanggaan untuk graf sederhana dan tidak berarah selalu simetris dan diagonal utamanya selalu bernilai 0 (nol)

karena tidak ada sisi gelang (*loop*). Jumlah elemen matriks ketetanggaan untuk graf dengan n simpul adalah n^2, jika tiap elemen membutuhkan ruang memori sebesar p maka ruang memori yang dibutuhkan seluruhnya adalah pn^2. Pada matriks ketetanggaan untuk graf tak berarah sederhana simetris, cukup dengan menyimpan elemen segitiga atas saja, karena matriksnya simetris, sehingga ruang memori akan dapat dihemat sebesar $pn^2/2$. Pada graf berbobot, a_{ij} menyatakan tiap sisi yang menghubungkan simpul i dengan simpul j. Gambar 2.7 merupakan contoh graf berbobot beserta matriks ketetanggaannya.



$$\begin{array}{c} \\ a \\ b \\ c \\ d \\ e \end{array} \begin{array}{c} \begin{array}{ccccc} a & b & c & d & e \end{array} \\ \begin{bmatrix} \infty & 12 & \infty & \infty & 10 \\ 12 & \infty & 9 & 11 & 8 \\ \infty & 9 & \infty & 14 & \infty \\ \infty & 11 & 14 & \infty & 15 \\ 10 & 8 & \infty & 15 & \infty \end{bmatrix} \end{array}$$

Gambar 2.5 Matriks Ketetanggaan untuk Graf

2.3 Definisi Lintasan (*Path*)

Lintasan adalah hubungan antara titik dalam sebuah graf. Lintasan yang berawal dan berakhir pada simpul yang sama disebut lintasan tertutup (*closed path*), sedangkan lintasan yang tidak berawal dan berakhir pada simpul yang sama disebut lintasan terbuka (*open path*).

2.3.1 Lintasan Terpendek (Shortest Path)

Persoalan mencari lintasan terpendek di dalam graf merupakan salah satu persoalan optimasi. Graf yang digunakan dalam pencarian lintasan terpendek adalah graf berbobot (*weighted graph*), yaitu graf yang setiap sisinya diberikan suatu nilai atau bobot. Bobot pada sisi graf dapat menyatakan jarak antar kota, waktu pengiriman pesan, ongkos pembangunan, dan sebagainya. Asumsi yang digunakan disini adalah bahwa semua bobot bernilai positif. Kata "terpendek" jangan selalu diartikan secara fisik sebagai panjang minimum, sebab kata "terpendek" berbeda-beda maknanya tergantung pada tipikal persoalan yang akan diseleseikan. Namun secara umum "terpendek" berarti meminimisasi bobot pada suatu lintasan di dalam graf.

Ada beberapa macam persoalan lintasan terpendek, antara lain :

a. Lintasan terpendek antara dua buah simpul tertentu.
b. Lintasan terpendek antara semua pasangan simpul.
c. Lintasan terpendek dari simpul tertentu ke semua simpul yang lain.
d. Lintasan terpendek antara dua buah simpul yang melalui beberapa simpul tertentu.

2.4 Algoritma *Greedy*

Algoritma *Greedy* adalah algoritma yang memecahkan masalah langkah demi langkah dan merupakan salah satu metode dalam masalah optimasi. Prinsip dari algoritma greedy adalah "*take what you can get now"* yaitu mengambil pilihan yang terbaik yang dapat diperoleh pada saat itu tanpa memperhatikan

konsekuensi kedepan. Algoritma *greedy* membentuk solusi langkah per langkah sebagai berikut:

1. Terdapat banyak pilihan yang perlu diekspolarasi pada setiap langkah solusi. Oleh karena itu, pada setiap langkah harus dibuat keputusan yang terbaik dalam menentukan pilhan. Keputusan yang telah diambil pada suatu langkah tidak dapat diubah lagi pada langkah selanjutnya.
2. Pendekatan yang digunakan di dalam algoritma greedy adalah membuat pilihan yang terlihat memberikan perolehan terbaik, yaitu dengan membuat pilihan optimum lokal pada setiap langkah dan diharapkan akan mendapatkan solusi optimum global.

Algoritma greedy didasarkan pada pemindahan *edge* per *edge* dan pada setiap langkah yang diambil tidak memikirkan konsekuensi ke depan, *greedy* tidak beroperasi secara menyeluruh terhadap semua alternatif solusi yang ada serta sebagian masalah *greedy* tidak selalu berhasil memberikan solusi yang benar-benar oprimum tapi pasti memberikan solusi yang mendekati nilai optimum.

Algoritma greedy disusun oleh elemen-elemen sebagai berikut :

1. Himpunan Kandidat

   Himpunan ini berisi elemen-elemen yang memiliki peluang pembentuk solusi.

2. Himpunan Solusi

   Himpunan ini berisi kandidat-kandidat yang terpilih sebagai solusi persoalan. Elemennya terdiri dari elemen dalam himpunan kandidat,

namun tidak semuanya dengan kata lain himpunan solusi ini adalah bagian dari himpunan kandidat.

3. Fungsi seleksi

   Fungsi yang pada setiap langkah memilih kandidat yang paling mungkin untuk menghasilkan solusi optimal. Kandidat yang sudah dipilih pada suatu langkah tidak pernah dipertimbangkan lagi pada langkah selanjutnya.

4. Fungsi kelayakan

   Fungsi yang memeriksa apakah suatu kandidat yang telah dipilih (diseleksi) dapat memberikan solusi yang layak.

5. Fungsi obyektif

   Fungsi yang memaksimumkan atau meminimumkan nilai solusi. Tujuannya adalah memilih satu saja solusi terbaik dari masing-masing anggota himpunan solusi.

2.5 Algoritma *Floyd*

Algoritma *Floyd* pertama kali diperkenalkan oleh *Robert Floyd* pada tahun 1962. Algoritma ini adalah salah satu metode untuk mencari lintasan terpendek dalam graf berbobot (dengan bobot sisi negatif atau positif). Algoritma ini juga merupakan salah satu varian dari pemrograman dinamis yaitu suatu metode yang melakukan pemecahan masalah dengan memandang solusi yang akan diperoleh sebagai suatu keputusan yang saling terkait. Artinya solusi-solusi tersebut

dibentuk dari solusi yang berasal dari tahap sebelumnya dan ada kemungkinan solusi lebih dari satu.

Algoritma *Floyd* membandingkan semua kemungkinan lintasan pada graf untuk setiap sisi dari semua simpul. Algoritma ini mampu mengerjakan proses perbandingkan ini sebanyak (*V*3) kali (bandingkan dengan kemungkinan jumlah sisi sebanyak (*V2*) (kuadrat jumlah simpul) pada graf, dan setiap kombinasi sisi diujikan). Hal tersebut bisa terjadi karena adanya perkiraan pengambilan keputusan (pemilihan lintasan terpendek) pada setiap tahap antara dua simpul, hingga perkiraan tersebut diketahui sebagai nilai optimal.

2.8 Flowchart

Flowchart adalah penggambaran secara grafik dari langkah-langkah dan urut-urutan prosedur dari suatu program. *Flowchart* menolong analis dan programmer untuk memecahkan masalah kedalam segmen-segmen yang lebih kecil dan menolong dalam menganalisis alternatif-alternatif lain dalam pengoperasian. Flowchart biasanya mempermudah penyelesaian suatu masalah khususnya masalah yang perlu dipelajari dan dievaluasi lebih lanjut.

Tahapan-tahapan dalam membuat *flowchart*, ada beberapa petunjuk yang harus diperhatikan, seperti :

1. Flowchart digambarkan dari halaman atas ke bawah dan dari kiri ke kanan.
2. Aktivitas yang digambarkan harus didefinisikan secara hati-hati dan definisi ini harus dapat dimengerti oleh pembacanya.
3. Kapan aktivitas dimulai dan berakhir harus ditentukan secara jelas.

4. Setiap langkah dari aktivitas harus diuraikan dengan menggunakan deskripsi kata kerja, misalkan menghitung keuntungan penjualan.
5. Setiap langkah dari aktivitas harus berada pada urutan yang benar.
6. Lingkup dan range dari aktifitas yang sedang digambarkan harus ditelusuri dengan hati-hati. Percabangan-percabangan yang memotong aktivitas yang sedang digambarkan tidak perlu digambarkan pada flowchart yang sama. Simbol konektor harus digunakan dan percabangannya diletakan pada halaman yang terpisah atau hilangkan seluruhnya bila percabangannya tidak berkaitan dengan sistem.
7. Gunakan simbol-simbol flowchart yang standar.

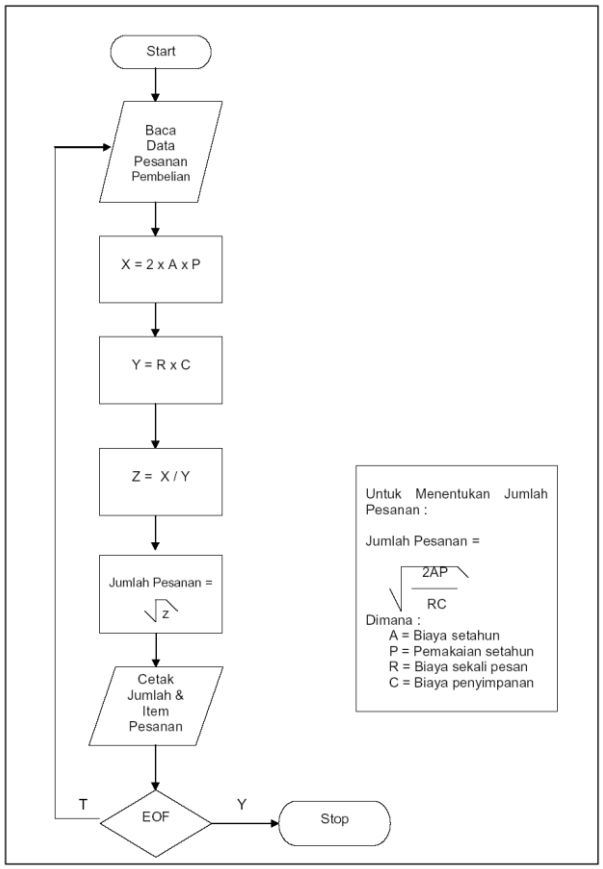
2.6.1 Jenis-jenis Flowchart :

Flowchart terbagi atas lima jenis, yaitu :

- Flowchart Sistem (*System Flowchart*)
- Flowchart Paperwork / Flowchart Dokumen (*Document Flowchart*)
- Flowchart Skematik (*Schematic Flowchart*)
- Flowchart Program (*Program Flowchart*)
- Flowchart Proses (*Process Flowchart*)

Dalam hal ini penulis menggunakan Flowchart Program (*Program Flowchart*) untuk mengetahui alur simulasi jalur terpendek. Flowchart Program dihasilkan dari Flowchart Sistem. Flowchart Program merupakan keterangan yang lebih rinci tentang bagaimana setiap langkah program atau prosedur sesungguhnya dilaksanakan. Flowchart ini menunjukkan setiap langkah program atau prosedur dalam urutan yang tepat saat terjadi. Programmer menggunakan

flowchart program untuk menggambarkan urutan instruksi dari program komputer. Analis Sistem menggunakan flowchart program untuk menggambarkan urutan tugas-tugas pekerjaan dalam suatu prosedur atau operasi.



Gambar 2.6 Contoh *Flowchart* Program.

2.7 JavaScript

JavaScript adalah bahasa pemrograman berbasis prototipe yang berjalan di sisi klien. Jika berbicara dalam konteks web, sederhananya kita dapat memahami *JavaScript* sebagai bahasa pemrogrman yang berjalan khusus untuk di browser atau halaman web agar halaman web menjadi lebih hidup. Jika dilihat dari suku katanya, *JavaScript* terdiri dari dua kata, yaitu *Java* dan *Script*. *Java* adalah bahasa pemrograman yang berorientasi objek, sedangkan *Script* adalah

serangkaian instruksi program. *JavaScript* adalah bahasa *script* (bahasa pemrograman yang dapat memegang kontrol aplikasi) yang berbasis pada bahasa pemrograman Java. Namun *JavaScript* bukanlah bagian dari teknologi *Java* dari Sun. *Java* merupakan bahasa pemrograman berorientasi objek murni, sedangkan *JavaScript* digunakan secara prosedural.

Cara kerja JavaScript adalah mengakses elemen pada HTML dan membuat aksi jika elemen-elemen HTML itu mengalami perubahan,misalnya berubahnya warna halaman web begitu sebuah tombol di klik. *JavaScript* membuat sebuah halaman web menjadi lebih dinamis. *JavaScript* yang digunakan pada halaman web merupakan client side scripting yang berarti bahwa web browser mengidentifikasi dan menjalankan skrip program yang disisipkan dalam dokumen web (yang diterima dari server),dan mungkin memperbarui tampilan halaman di komputer pemakai (*user*) tanpa mengirimkan permintaan (*request*) baru kepada web server. Karena *JavaScript* pada halaman web adalah client side scripting, maka kode *JavaScript* yang ditulis satu file dengan dokumen HTML akan ditampilkan ketika sebuah halaman web dilihat source code-nya. Secara fungsional, *JavaScript* digunakan untuk menyediakan akses skrip pada objek yang dibenamkan (*embedded*).

Contoh sederhana dari penggunaan *JavaScript* adalah membuka halaman pop up, fungsi validasi pada form sebelum data dikirimkan ke server, merubah gambar kursor ketika melewati objek tertentu, dan lain-lain. Yang harus diperhatikan dalam pengelolaan pemrograman *JavaScript* diantaranya *JavaScript* adalah bahasa pemrograman yang *case sensitive*, yang artinya *JavaScript* membedakan

huruf kecil dan huruf besar. Hal ini sama seperti bahasa pemrograman Turbo C atau C++ dimana huruf "A" tidak sama dengan huruf "a".

JavaScript bekerja pada sisi browser, artinya untuk menampilkan halaman web, user menuliskan alamat web di *address bar url*. Setelah itu, *browser* mengambil file HTML (dengan file *JavaScript* yang melekat padanya jika memang ada) ke server yang beralamat di URL yang diketikkan oleh user. Selesai file diambil, file ditambilkan pada browser. Setelah file *Java Script* berada pada *browser*, barulah skrip *JavaScript* tersebut bekerja. Efek dari *JavaScript* yang bekerja pada sisi browser ini, *Java Script* dapat merespon perintah *user* dengan cepat, dan membuat halaman web menjadi lebih responsif. *Java Script* melakukan apa yang tidak bisa dilakukan oleh HTML, PHP, dan CSS dalam menangani hal-hal yang membutuhkan respon cepat terhadap aksi dari *user*. Misalnya, fungsi validasi pada *form*. Ketika *user* mengisi sebuah *form* yang divalidasi menggunakan *JavaScript*, *user* mengetikkan data lalu mengklik *submit*. Sebelum data dikirimkan ke server, data akan dicek terlebih dahulu pada *browser* menggunakan fungsi *JavaScript* yang ada pada halaman web. Sehingga jika memang data yang diisikan tidak valid, daripada membuang-buang waktu dengan mengirim data ke *server* baru divalidasi di *server*, lalu server mengirimkan respon balik mengenai ketidak validan input data *user*, lebih baik pengecekan *validasi* data *form* dilakukan secara lokal di browser menggunakan fungsi *JavaScript*.

2.5.1 Kelebihan dan Kekurangan *JavaScript*

Kelebihan dari bahasa pemrograman JavaScript adalah sebagai berikut :

- Ukuran file kecil Skrip dari *JavaScript* memiliki ukuran yang kecil sehingga ketika web yang memiliki *JavaScript* ditampilkan di browser, maka akses tampilannya akan lebih cepat dibandingkan ketika browser membuka suatu web yang memiliki skrip *Java*. Hal ini juga sangat berkaitan dengan daya kerja server. Semakin kecil space suatu web yang disimpan dalam suatu server, maka daya kerja server ketika di-browsing oleh user di internet akan tidak terlalu berat. Selain itu sifat *JavaScript* Client Side yang tidak perlu lagi ditangani oleh server ketika browser memanggil web dari sebuah server.
- Mudah untuk dipelajari *JavaScript* merupakan bahasa semi pemrograman yang merupakan gabungan antara bahasa pemrograman *Java* dengan bahasa kode *HTML* sehingga disebut juga bahasa *hybrid*. Walaupun *JavaScript* merupakan turunan dari *Java*, namun *JavaScript* tidak memiliki aturan serumit *Java*.
- Terbuka, *JavaScript* tidak terikat oleh hardware maupun software tertentu, bahkan sistem operasi seperti *windows* maupun *unix*. Karena ia bersifat terbuka, maka ia dapat dibuat maupun dibaca di semua jenis komputer.

Sedangkan kekurangan *JavaScript* diantaranya adalah :

- Skrip tidak terenkripsi

  Karena JavaScript bersifat *client side* , maka skrip yang kita buat di *text* editor dan telah dijadikan web di *server*, ketika user me-*request* web dari *server* tersebut, maka sintaks *JavaScript* akan langsung ditampilkan di browser. *User* bisa melihatnya dan menirunya dari *source* -nya.

- Kemampuan terbatas

  Walaupun *JavaScript* mampu membuat bentuk web menjadi interaktif dan dinamis, namun *JavaScript* tidak mampu membuat program aplikasi sendiri seperti *Java*.

- Keterbatasan objek

  JavaScript tidak mampu membuat kelas-kelas yang bisa menampung objek-objek tambahan seperti *Java*, karena *JavaScript* telah memiliki objek yang built in pada struktur bahasanya.

2.8 Perangkat Lunak Pendukung

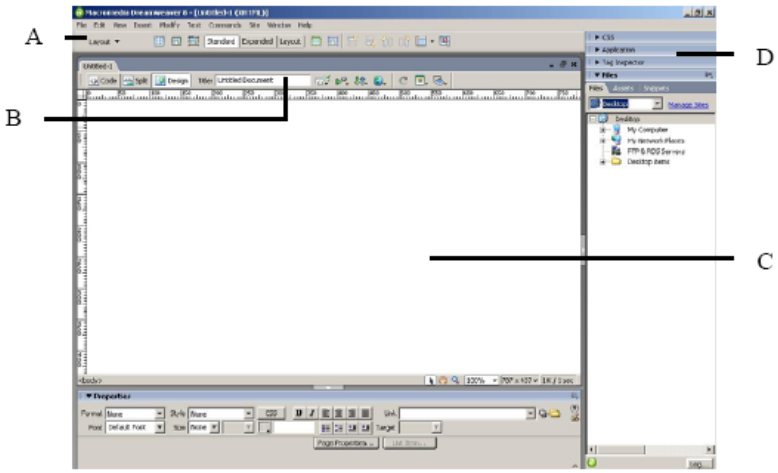
Untuk perangkat lunak pendukung penulis mengembangkannya melalui *Macromedia Dreamwaver*. *Dreamweaver* adalah *software* yang digunakan oleh *Web* desainer ataupun *Web programme*r untuk mengembangkan situs *web*. Ruang kerja, fasilitas dan kemampuan *Dreamweaver* mampu meningkatkan produktivitas dan efektivitas dalam mendesain ataupun membangun situs *web*. *Dreamweaver* juga dilengkapi dengan fasilitas yang cukup lengkap. Maka dari itu aplikasi yang akan dibuat ini menggunakan *Macromedia Dreamweaver*.

2.8.1 Ruang Kerja Dreamwaver

Pada saat pertama kali anda memulai *Dreamweaver*, sebuah kotak dialog *Workspace* akan muncul, memberikan penawaran kepada anda untuk memilih tampilan ruang kerja yang ingin anda gunakan. *Dreamweaver Workspace* adalah ruang kerja yang semua jendela dokumen dan panel integrasi pada satu jendela besar, dengan panel *group* yang berada disisi kanan. Pilihan ini dipakai oleh

banyak pengguna. *Dreamweaver Workspace* adalah tampilan lama seperti pada *Dreamweaver*.

Setiap dokumen akan berada pada jendela yang terpisah. Ruang kerja pada Draemweaver 8 memiliki komponen yang memberikan fasilitas dan ruang untuk menuangkan kreasi dalam bekerja, seperti yang terlihat pada Gambar 2.5. Komponen yang disediakan oleh *Dreamweaver* antara lain adalah *Insert bar*, *Document toolbar*, *Document window*, *Panel groups*, *Site panel*, *Property inspector*. Perhatikan **Gambar 2.7** dibawah ini. Berikut bagian-bagian yang terdapat pada jendela kerja dari Dreamweaver 8.



Gambar 2.7 Jendela Kerja *Dreamwaver*

A. Insert Bar

Insert Bar merupakan kumpulan menu yang digunakan untuk memasukkan sebuah objek atau fungsi lainnya ke dalam jendela dokumen. Contoh: *Image*, *Layer*, *Tabel* dan lain-lain. *Insert Bar* juga mempunyai tujuh menu didalamnya yaitu *Common*, *Layout*, *Forms*, *Text*, *HTML*, *Application* dan *Flash Elements.* Untuk menampilkan dan menggunakan

menu yang terdapat pada *Insert Bar*, kita dapat meng-klik tombol *drop down* yang berada di sebelah kiri *Insert Bar*. Perhatikan **Gambar 2.8** .



Gambar 2.8 *Menu Insert Bar*

B. Document Toolbar

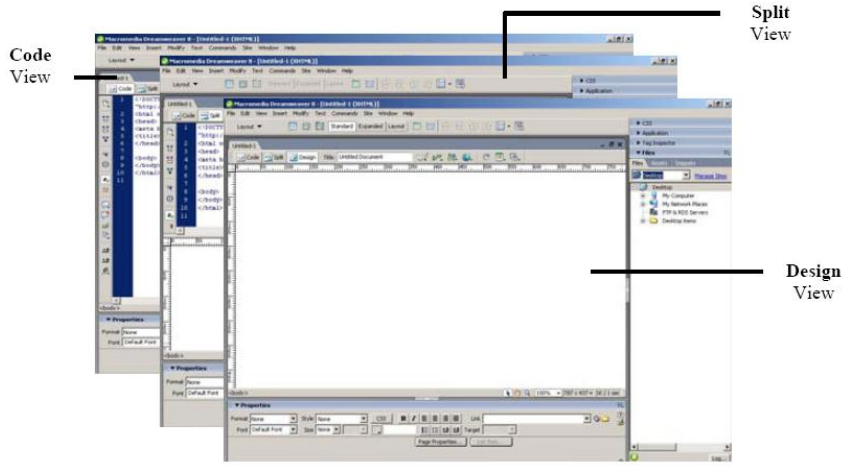
Document Toolbar digunakan sebagai penempatan *file-file* yang telah dibuka dan sekaligus untuk menampilkan nama dari *file* tersebut. *Document Toolbar* juga mempunyai tiga buah *tab* yang dapat membantu Anda mendesain *web* dengan mengubah tampilan dari jendela dokumen, diantaranya *Code, Split, Design*.



Gambar 2.9 Tampilan *Document Toolbar*

- ***Code*** : merupakan kumpulan kode program yang telah disusun, dan berfungsi untuk mengedit listing dari program.
- ***Split*** : digunakan untuk menampilkan *tab* dari *code view* dan *design view* secara bersamaan.
- ***Design***: digunakan sebagai tempat untuk meletakkan objek-objek ke dalam jendela dokumen, seperti Tabel, *Form*, dan lainnya.

Pada gambar 2.10 merupakan tampilan dari *tab-tab code, split, view*.



Gambar 2.10 Tampilan *Tab-tab View, Split dan Code*

C. Document Window

Document Window atau biasa disebut dengan jendela dokumen, merupakan tempat untuk menampilkan objek-objek atau kode program dari *tab code, split, dan design*.

D. Panel Groups

Panel Group berfungsi untuk mengatur halaman *web* yang telah Anda buat. Contoh : *Panel File*, berfungsi untuk mencari dan mengetahui letak halaman *file-fi*le yang telah di buat.

E. Property Inspector

Property Inspector berfungsi untuk mengetahui atau mengubah property dari sebuah objek. Contoh : untuk penulisan sebuah teks, kita dapat mengubah *font, color, dan size* dari teks tersebut dengan menggunakan *Property Inspector*.



Gambar 2.11 Tampilan *Property Inspector*

2.8.2 Komponen Ruang Kerja *Dreamwaver*

Di dalam ruang kerja *Dreamweaver* terdapat beberapa komponen utama yang membentuk suatu ruang kerja. Berikut ini keterangan singkat tentang menu yang ada pada *Dreamweaver*. Menu berisi *submenu* yang biasanya ditampilkan sebagai *dropdown* menu yang akan muncul ketika anda menekan tombol menu utama.



Gambar 2.12 Menu Pada *Dreamwaver*

Di dalam sistem menu *Dreamweaver* terdapat beberapa komponen yang dapat digunakan, yaitu :

1. ***File Menu***

   Berisi menu standar seperti *New*, *Open*, *Save*, *Cut*, *Copy*, dan *Paste*. *File menu* juga berisi beberapa perintah tambahan, seperti *Preview in Browser* dan *Print Code*.

2. ***Edit Menu***

Berisi perintah seleksi dan pencarian, seperti *Select Parent Tag* dan *Find and Replace*, dan menyediakan akses ke *Keyboard Shortcut Editor* dan *Tag Library Editor*.

3. View Menu

Mengijinkan anda melihat tampilan dokumen dengan beberapa cara (seperti *Design view* dan *Code view*) dan untuk menampilkan atau menyembunyikan beberapa elemen pada dokumen dan peralatan yang dimiliki *Dreamweaver*.

4. Insert Menu

Menyediakan alternatif untuk memasukkan objek pada dokumen.

5. Modify Menu

Untuk mengubah properti elemen halaman *web* atau item lainnya yang dipilih. Menggunakan menu ini dapat mengedit *tag*, mengubah *table* dan elemen tabel dan menampilkan beberapa tindakan yang dapat dilakukkan untuk *item* pada *library* dan *template*.

6. Text Menu

Menyediakan perintah yang dibutuhkan untuk melakukan pemformatan pada teks.

7. Commands Menu

Menyediakan akses untuk beberapa macam perintah, termasuk untuk membuat album foto dengan cepat dan melakukan optimasi gambar dengan menggunakan program lain.

8. Site Menu

Menyediakan item menu untuk membuat, membuka, mengedit site.

9. Window Menu

Menyediakan akses kesemua *panel*, *inspectors*, dan jendela pada *Dreamweaver*.

10. Help Menu

Menyediakan akses ke dokumentasi yang dimiliki oleh *Dreamweaver*, termasuk cara penggunaan *Dreamweaver*, membuat ekstensi dan material lainnya yang dapat membantu anda menggunakan *Dreamweaver* dengan lebih baik.