BAB II

MATERI PENUNJANG

2.1. Algoritma

Definisi 2.1.1.

Algoritma adalah himpunan berhingga instruksi.

Hubungan antara algoritma, masalah, dan solusi dapat digambarkan sebagai berikut :

Gambar 2.1. Diagram hubungan masalah, algoritma, dan solusi.

Proses dari masalah hingga terbentuk suatu algoritma disebut tahap pemecahan masalah, sedangkan tahap dari algoritma hingga terbentuk suatu solusi disebut dengan tahap implementasi. Solusi yang dimaksud adalah suatu program yang merupakan implementasi dari algoritma yang disusun.

Algoritma yang baik memiliki beberapa kriteria sebagai berikut :

1. Adanya output

Dalam menyelesaikan suatu permasalahan, algoritma harus memiliki output yang merupakan solusi dari masalah yang sedang diselesaikan.

2. Efektifitas dan Efisien

Suatu algoritma dikatakan efektif jika algoritma tersebut dapat menyelesaikan suatu solusi yang sesuai dengan masalah yang diselesaikan. Algoritma yang efisien adalah algoritma yang waktu prosesnya lebih singkat dan penggunaan memorinya relatif lebih sedikit.

3. Jumlah langkahnya berhingga

Barisan instruksi yang dibuat dalam suatu urutan tertentu, dimaksudkan agar masalah yang dihadapi dapat diselesaikan . Banyaknya instruksi atau langkah-langkah harus berhingga.

4. Berakhir

Proses didalam mencari penyelesaian suatu masalah harus berhenti atau berakhir. Hasil akhir yang didapat merupakan solusinya atau informasi tidak ditemukannya solusi.

5. Terstruktur

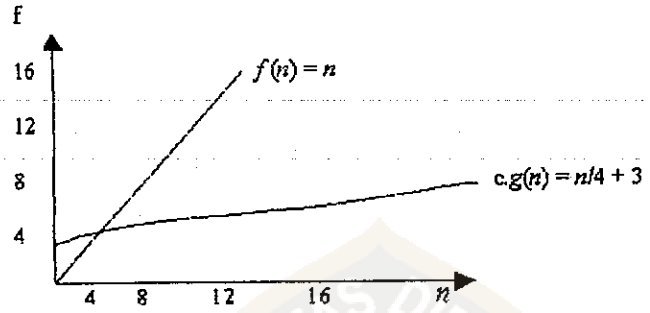
Urutan dari barisan langkah-langkah yang digunakan harus disusun sedemikian rupa agar proses penyelesaiannya tidak berbelit-belit, sehingga memungkinkan waktu prosesnya akan menjadi relatif lebih singkat.

2.2 Pengertian dan sifat – sifat dari notasi big O

Definisi 2.1.2

Misal $g : N \rightarrow R^*$. O(g) adalah himpunan fungsi $f : N \rightarrow R^*$ dikatakan bahwa $f(n) = O(g(n))$, dibaca '$f(n)$ adalah big O dari $g(n)$' jika dapat ditemukan bilangan $c \in R^+$ dan $n \in N$ sedemikian hingga $|f(n)| \leq c. |g(n)|$ untuk semua $n \geq n_0$.

Dengan N = {0,1,2,3,...}, R = himpunan bilangan riil, N^+ = {1,2,3,...}, R^+ = {himpunan bilangan riil positif}, dan $R^* = R^+ \cup \{0\}$.



Gambar 2.2. Fungsi $f(n) = O(g(n))$

Pada gambar 2.2. menunjukkan gambaran dari fungsi $f(n)$ dan $g(n)$ untuk $f(n) = O(g(n))$. Notasi O memberikan suatu batas atas untuk suatu fungsi pertumbuhan. Suatu fungsi $f(n)$ termasuk dalam himpunan $O(g(n))$ jika terdapat bilangan positif c dan n_o sedemikian hingga $f(n)$ berada dibawah c. $g(n)$ walaupun $f(n) > c.g(n)$ untuk $n > 4$. Penulisan $f(n) = O(g(n))$ berarti bahwa fungsi $f(n)$ merupakan anggota dari $O(g(n))$ atau $f(n) \in O(g(n))$.

Terdapat beberapa cara dalam menentukan notasi big O, diantaranya dengan menggunakan aturan limit dan teorema d'L Hospital.

(i). Penerapan aturan limit untuk menentukan notasi big O.

Aturan limit digunakan untuk menentukan kecepatan pertumbuhan dua fungsi $f(n) = O(g(n))$ atau $|f(n)| \leq c. |g(n)|$ jika $\lim_{n \to \infty} \frac{f(n)}{g(n)} = c$

untuk suatu c ∈ R. Jika limit dari rasio $f(n)$ terhadap $g(n)$ ada dan tidak sama dengan ∞ maka pertumbuhan fungsi $f(n)$ tidak lebih cepat dibandingkan fungsi $g(n)$. Jika limitnya adalah ∞, maka pertumbuhan fungsi $f(n)$ lebih cepat daripada $g(n)$.

(ii). Penerapan aturan d'L Hospital

Jika $\lim_{n\to\infty} f(n) = \lim_{n\to\infty} g(n) = \infty$, maka $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{f'(n)}{g'(n)}$

dengan syarat derivatif dari f' dan g' ada.

Contoh 2.1.

(i). Misalkan $g(n) = n^3/2$ dan $f(n) = 37\,n^2 + 120\,n + 17$. Akan ditunjukkan bahwa $f(n) = O(g(n))$ tetapi $g(n) \neq O(f(n))$.

(ii). Misalkan suatu $f(n) = n^2 \log n$ dan $g(n) = n^2$. Akan ditunjukkan bahwa $f(n) = O(g(n))$, tetapi $g(n) \neq O(f(n))$.

Analisa :

(i).- Untuk membuktikan $f(n) = O(g(n))$ digunakan limit dari rasio $f(n)$ terhadap $g(n)$.

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} (37n^2 + 120n + 17)/(n^3/2) = \lim_{n\to\infty} (74/n + 240/n^2 + 34/n^3)$$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$$

karena limit dari rasio $f(n)$ terhadap $g(n)$ ada, maka $f(n) = O(g(n))$.

- **Untuk membuktikan bahwa** $g(n) \neq O(f(n))$

$$\lim_{n\to\infty} \frac{g(n)}{f(n)} = \lim_{n\to\infty} (n^3/2)/(37n^2+120n+17) = \lim_{n\to\infty} n^3/(74n^2+240n+34)$$

$$\lim_{n\to\infty} \frac{g(n)}{f(n)} = \infty$$

karena limit dari rasio $g(n)$ terhadap $f(n)$ adalah ∞, maka $g(n) \neq O(f(n))$

(ii).- Untuk menentukan $f(n) = O(g(n))$ **digunakan aturan d'L Hospital**

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} (n\ {}^2\log n)/n^2 = \lim_{n\to\infty} ({}^2\log n)/n = \lim_{n\to\infty} \frac{f'(n)}{g'(n)} = \lim_{n\to\infty} \frac{f'(n)}{g'(n)}$$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{f'(n)}{g'(n)} = \lim_{n\to\infty} ({}^2\log e/n)/1 = \text{limit } ({}^2\log e)/n \text{ karena } f'(n) = ({}^2\log e)/n$$

maka $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$

karena limit dari rasio $f(n)$ terhadap $g(n)$ ada, maka $f(n) = O(g(n))$.

- **Untuk membuktikan** $g(n) \neq O(f(n))$ maka

$$\lim_{n\to\infty} \frac{g(n)}{f(n)} = \lim_{n\to\infty} n^2/(n\ {}^2\log n) = \lim_{n\to\infty} n/({}^2\log n) = \lim_{n\to\infty} \frac{g(n)}{f(n)} = \lim_{n\to\infty} \frac{g'(n)}{f'(n)}$$

$$\lim_{n\to\infty} \frac{g'(n)}{f'(n)} = \lim_{n\to\infty} 1/({}^2\log e/n) = \lim_{n\to\infty} n/({}^2\log e)$$

$$\lim_{n\to\infty} \frac{g(n)}{f(n)} = \infty$$

karena limit dari rasio $g(n)$ terhadap $f(n)$ adalah ∞, maka $g(n) \neq O(f(n))$.

Teorema 2.2.1

Misalkan

$$f(n) = a_p n^p + a_{p-1} n^{p-1} + \ldots + a_1 n + a_0, a_p \neq 0$$

Adalah fungsi polinomial derajat p, dan

$$g(n) = n^p$$

maka

$$f(n) = O(g(n))$$

Bukti :

$$\begin{aligned} |f(n)| &= |a_p n^p + a_{p-1} n^{p-1} + \ldots + a_1 n + a_0| \\ &\leq |a_p n^p| + |a_{p-1} n^{p-1}| + \ldots + |a_1 n| + |a_0| \\ &\leq |a_p| n^p + |a_{p-1}| n^{p-1} + \ldots + |a_1| n + |a_0| \\ &\leq |a_p| n^p + |a_{p-1}| n^p + \ldots + |a_1| n^p + |a_0| n^p \\ &= (|a_p| + |a_{p-1}| + \ldots + |a_1| + |a_0|) n^p \end{aligned}$$

Dipilih

$$c = |a_p| + |a_{p-1}| + \ldots + |a_1| + |a_0|$$

sehingga diperoleh :

$$|f(n)| \leq c. |g(n)|$$

Terbukti bahwa :

$$f(n) = O(g(n))$$

Teorema 2.2.2.

Misalkan S adalah himpunan semua fungsi dengan domain D, dengan

D : N → R*. Fungsi – fungsi

$$f_1(n), f_2(n), g_1(n), \text{dan } g_2(n)$$

adalah anggota-anggota S sedemikian sehingga

$$f_1(n) = O(g_1(n))$$

dan

$$f_2(n) = O(g_2(n))$$

maka berlaku :

a) $f_1(n) + f_2(n) = O(\text{maks}\{g_1(n), g_2(n)\})$

b) $f_1(n) \cdot f_2(n) = O(g_1(n) \cdot g_2(n))$

Bukti

a) Karena $f_1(n) = O(g_1(n))$ dan $f_2(n) = O(g_2(n))$ maka terdapat c_1 dan c_2, k_1 dan k_2 sedemikian sehingga jika $n \in D$ dan $n > k_1$, dan $n > k_2$, maka berlaku :

$$\begin{aligned} |f_1(n)| + |f_2(n)| &= f_1(n) + f_2(n) \\ &\le c_1 . |g_1(n)| + c_2 . |g_2(n)| \\ &\le c_1 . g_1(n) + c_2 . g_2(n) \\ &\le c_1 . \text{maks}\{g_1(n), g_2(n)\} + \\ &\quad c_2 . \text{maks}\{g_1(n), g_2(n)\} \\ &= (c_1 + c_2) \text{ maks}\{g_1(n), g_2(n)\} \\ &= c . \text{maks}\{g_1(n), g_2(n)\} \end{aligned}$$

maka terbukti bahwa $f_1(n) + f_2(n) = O(\text{maks}\{g_1(n), g_2(n)\})$

b) karena $f_1(n) = O(g_1(n))$ dan $f_2(n) = O(g_2(n))$ maka terdapat c_1, c_2, k_1, dan k_2 bilangan positif sedemikian sehingga

jika $n \in D$ dan $n > k_1$ maka,

$$|f_1(n)| \leq c \cdot g_1(n)$$

dan jika $n > k_2$ berlaku

$$|f_2(n)| \leq c \cdot g_2(n),$$

selanjutnya pilih $k = \text{maks}(k_1, k_2)$ dan $c = c_1 \cdot c_2$ maka

$$\begin{aligned} |(f_1 \cdot f_2)(n)| &= |f_1(n)| \cdot |f_2(n)| \\ &\leq c_1 |g_1(n)| \cdot c_2 |g_2(n)| \\ &= c |g_1(n) \cdot g_2(n)| \end{aligned}$$

Sehingga $f_1(n) \cdot f_2(n) = O(g_1(n) \cdot g_2(n))$, terbukti.

Contoh 2.2 :

Jika suatu fungsi $f(n) = 3n^3 + 2n^2$ merupakan fungsi dari waktu tempuh suatu algoritma maka big O nya adalah n^3, yang dinotasikan $f(n) = O(n^3)$

Analisa :

Berdasarkan definisi 2.1.2. bahwa jika $f(n) = O(n^3)$ maka akan terdapat dua konstanta bulat positif c dan n_0 yaitu

$$f(0) = 0 \quad n_0 = 0, \text{ dan } f(1) = 5 \quad c = 5$$

Sedemikian sehingga berlaku $3n^3 + 2n^2 \leq 5n^3$ untuk setiap $n \geq 0$

Jadi terbukti bahwa big O dari $f(n) = 3n^3 + 2n^2$ adalah n^3, yang dinotasikan $f(n) = O(n^3)$.

2.3. Waktu Tempuh (*Running Time*)

Pada suatu analisa algoritma untuk menentukan kecepatan prosesnya digunakan istilah running time dan dinotasikan dengan T(n). Proses suatu algoritma di dalam mencari solusi dari suatu masalah memerlukan waktu tertentu. Adapun hal-hal yang mempengaruhi waktu tempuh adalah

(i) Banyaknya Langkah

Makin banyak langkah atau instruksi yang digunakan makin lama running time yang dibutuhkan dalam proses tersebut.

(ii) Besar dan Jenis Input Data

Ukuran besar dan Jenis input data yang digunakan akan sangat berpengaruh pada proses penghitungan

(iii) Jenis Operasi

Running time juga dipengaruhi oleh jenis operasi yang digunakan. Jenis operasi tersebut meliputi operasi aritmatika, operasi logika.

(iv) Komputer dan Kompilator

Faktor ini di luar dari rancangan atau pembuatan algoritma yang efisien, dan lebih mengacu pada sistem komputer yang digunakan.

Dalam melakukan analisa algoritma terdapat tiga hal yang digunakan yaitu :

(i) **Worst Case Running Time**

Suatu keadaan 'terburuk' dari proses dalam suatu algoritma, sehingga waktu yang ditempuh oleh algoritma tersebut adalah waktu yang maksimum.

(iii) **Average Case Running Time**

Analisa waktu rata-rata yang digunakan untuk melaksanakan algoritma. Tetapi hal ini tidak sering digunakan, karena adanya kesulitan dalam menentukan suatu data yang secara umum dapat mewakili keadaan rata-rata data yang digunakan.

(iii) **Best Case Running Time**

Suatu keadaan 'terbaik' dari proses di dalam suatu algoritma sehingga waktu yang ditempuh oleh algoritma tersebut adalah waktu yang minimum.

Dalam pembahasan selanjutnya untuk menentukan running time suatu algoritma dipergunakan worst case runing time, hal ini berdasarkan pada beberapa alasan yaitu :

(i) Worst case running time merupakan waktu maksimal yang dibutuhkan suatu algoritma dalam menyelesaikan masalah.

(ii) Worst case running time dari suatu algoritma merupakan batas atas dari running time. Dengan demikian akan memberikan suatu jaminan bahwa algoritma yang digunakan mempunyai akhir proses.

(iii) Untuk beberapa algoritma, worst case running time merupakan kejadian yang sering terjadi. Sebagai contoh, dalam proses pencarian suatu data base untuk mendapatkan informasi. Worst case running time selalu terjadi ketika informasi yang dibutuhkan tidak berada dalam data base.

Suatu algoritma mempunyai ciri-ciri khusus yang mungkin berbeda satu sama lain, maka dibutuhkan kemampuan untuk mengidentifikasi algoritma secara signifikan,

walaupun demikian diperlukan suatu kerangka utama dalam menganalisa suatu algoritma sehingga memberikan kemudahan dalam melakukan analisa.

Dalam melakukan analisa algoritma digunakan suatu aturan umum yang dapat membuat lebih sederhana perhitungan dalam menentukan running time, yaitu perhitungan hanya dikenakan pada suatu loop atau iterasi dari algoritma. Aturan-aturan yang diperlukan dalam melakukan analisa algoritma adalah :

(i) Loop

Running time dari suatu loop adalah jumlah iterasi yang dilakukan untuk menyelesaikan statemen-statemen dalam loop.

(ii) Loop Berkalang

Jika dalam suatu algoritma tedapat loop berkalang yaitu loop yang berada dalam loop, maka yang menjadi pedoman adalah loop terdalam. Dan running time nya adalah hasil dari perkalian ukuran semua input pada loop yang diselesaikan.

(iii) Statemen Berurutan

Apabila terdapat statemen berurutan, maka penentuan running timenya menggunakan running time maksimal dari running time yang ada. (menurut teorema 2.2.2.).

(iv) Keadaan if – then – else

Untuk semua penggalan algoritma :

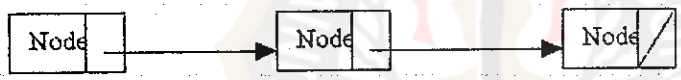
if (kondisi) then S_1 else S_2 . Running time dari keadaan if – then – else tidak akan melebihi dari running time test kondisi ditambah running time terbesar dari S_1 dan S_2.

2.4. Pointer

Suatu pointer adalah suatu peubah yang berisi alamat di memori dimana suatu data disimpan.

2.5. Senarai Berantai

Senarai berantai adalah suatu kumpulan elemen-elemen yang disebut node dengan setiap node memiliki medan khusus yang disebut pointer, yang menghubungkan node ke node dalam senarai.



Gambar 2.3. Senarai berantai

2.5.1. Operasi Pada Senarai Berantai

Beberapa operasi pada senarai berantai yang digunakan yaitu operasi menambah node dan menghapus node.

2.5.1.1. Operasi Menambah Node

Operasi menambah node dapat dipecah berdasarkan posisi node baru yang akan disisipkan, yaitu node baru selalu ditambahkan di belakang node terakhir, node baru selalu diletakkan sebagai node pertama, dan node baru menyisip diantara dua node yang sudah ada. Untuk jelasnya digunakan deklarasi pointer dan node seperti di bawah ini :

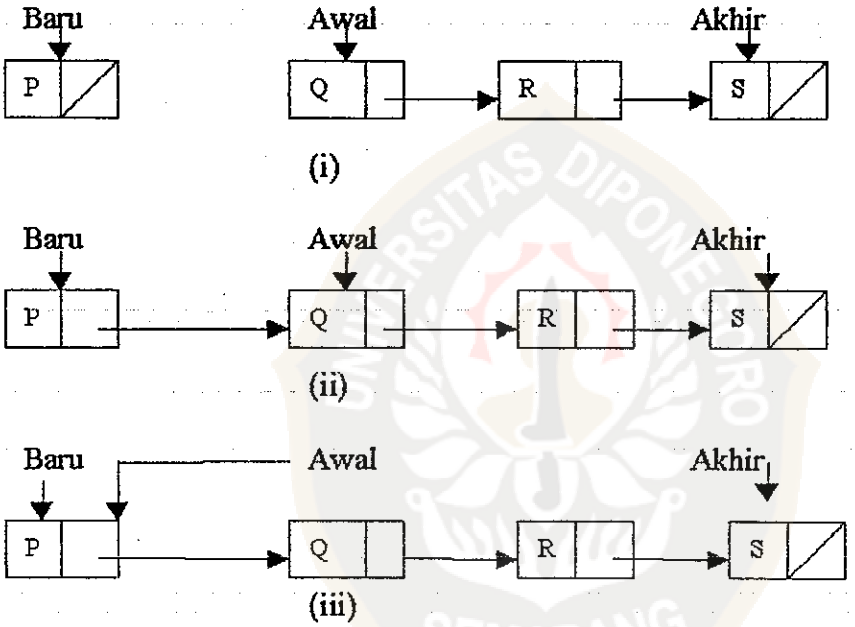
```
Type node = ^ Data;

Data = record
         Info    : char;
         Berikut : node;
```

```
Var  Elemen                : char;
     Awal, Akhir, Baru     : node;
```

a. Menambah node di Depan

Ilustrasi penambahannya disajikan sebagai berikut :

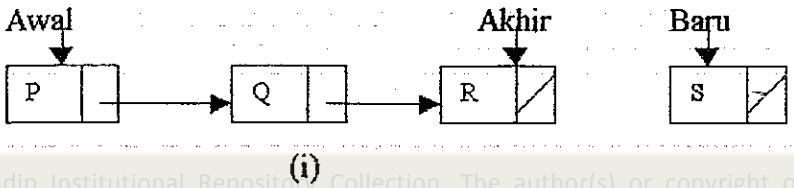


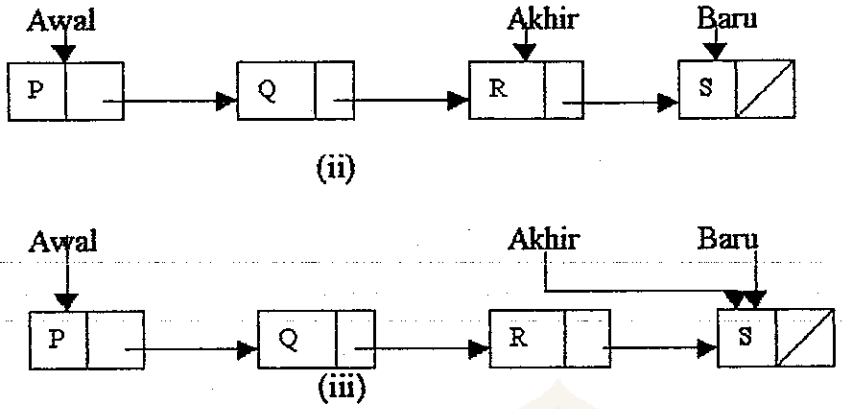
Gambar 2.4. Penambahan node di depan

Pada gambar 2.4. dapat dijelaskan sebagai berikut, (gambar 2.4. (ii)) pointer pada node yang ditunjuk oleh pointer Baru dibuat menunjuk lokasi yang ditunjuk pointer Awal. Kemudian pointer Awal dibuat menunjuk lokasi yang ditunjuk oleh pointer Baru (gambar 2.4. (iii)).

b. Menambah node di belakang

Ilustrasi penambahannya disajikan sebagai berikut:



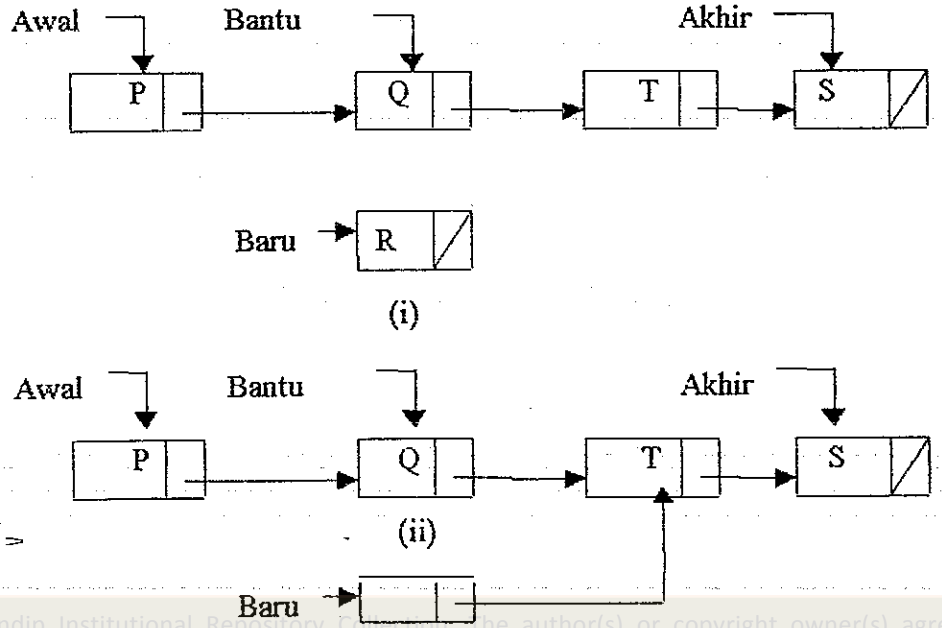


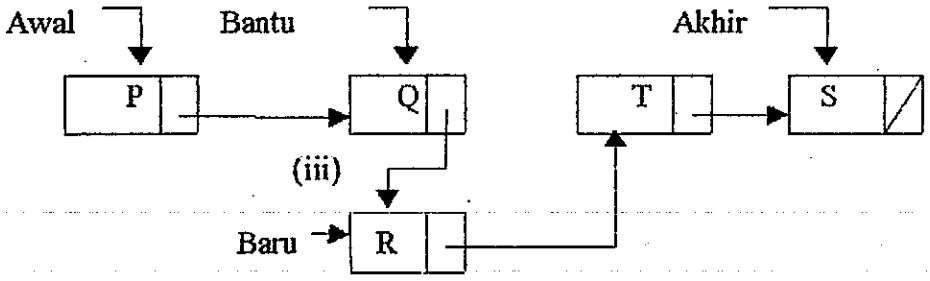
Gambar 2.5. Penambahan node di belakang

Pada gambar 2.5. dapat dijelaskan sebagai berikut, (gambar 2.5. (ii)) pointer pada node yang ditunjuk node Akhir dibuat menunjuk lokasi yang ditunjuk pointer Baru, kemudian pointer Akhir dibuat menunjuk lokasi yang ditunjuk pointer Baru (gambar 2.5. (iii)).

c. Menambah node di tengah

Ilustrasi penambahan node di tengah disajikan sebagai berikut :



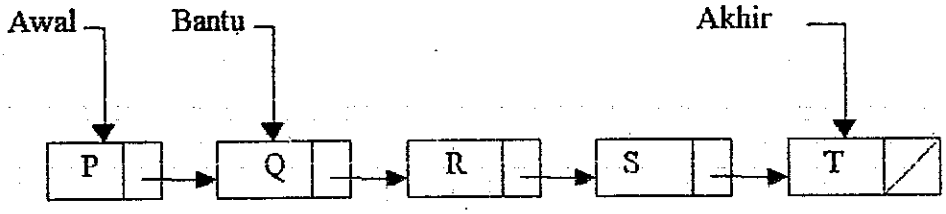


Gambar 2.6. Penambahan node di tengah

Pada gambar 2.6. dapat dijelaskan sebagai berikut, lebih dahulu ditentukan dimana node Baru akan ditambahkan, yaitu dengan menempatkan pointer Bantu pada node sebelum node yang akan ditambahkan node Baru. Kemudian pointer pada node yang ditunjuk oleh Baru dibuat menunjuk lokasi yang ditunjuk pointer pada node yang ditunjuk oleh Bantu (gambar 2.6. (ii)), selanjutnya pointer pada node yang ditunjuk oleh node Bantu dibuat menunjuk lokasi yang ditunjuk pointer Baru (gambar 2.6. (iii)).

2.5.1.2. Operasi menghapus node

Dalam operasi menghapus node ini, yaitu bahwa node yang bisa dihapus adalah node yang berada sesudah node yang ditunjuk oleh pointer (dalam gambar 2.7. adalah node yang berada di sebelah kanan node yang ditunjuk oleh suatu pointer), kecuali untuk node pertama. Dengan demikian kita tidak bisa menghapus node yang sudah ditunjuk oleh suatu pointer atau node sebelumnya. Untuk lebih jelasnya diberikan ilustrasi sebagai berikut :



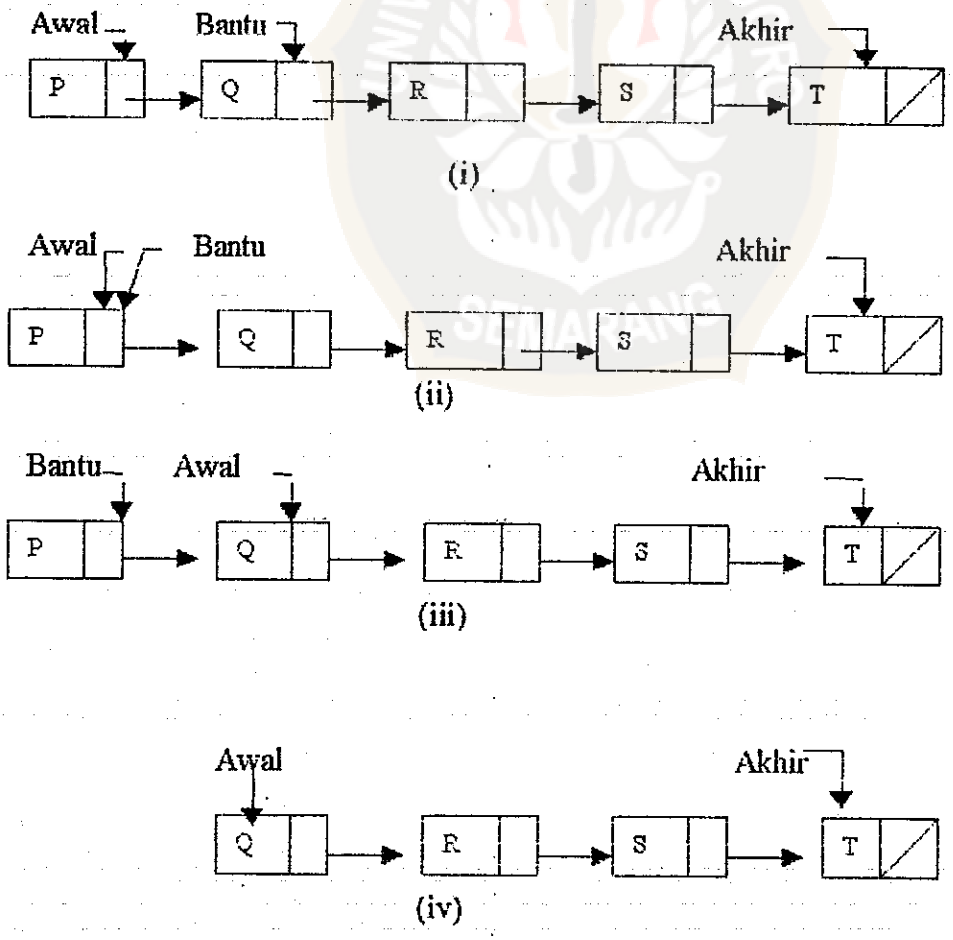
Gambar 2.7. Senarai Berantai

Pada gambar 2.7. terlihat bahwa senarai tersebut hanya dapat dihapus pada node yang berisi 'P' dan node yang berisi 'R' supaya senarai tetap dapat dipertahankan. Jika node 'Q' dihapus, maka senarai tersebut menjadi terputus. dan akan lebih sulit menyambung kembali node 'P' dan node 'R' . Demikian juga jika node 'T' dihapus, maka pointer pada node 'S' menjadi tidak menentu dan dapat menyebabkan terjadinya kesalahan.

Untuk menghapus node bisa dibagi dalam tiga penghapusan, menghapus node di depan, menghapus node yang di tengah dan node yang di akhir.

a. Menghapus node di depan

Ilustrasi penghapusan node di depan dapat disajikan sebagai berikut :

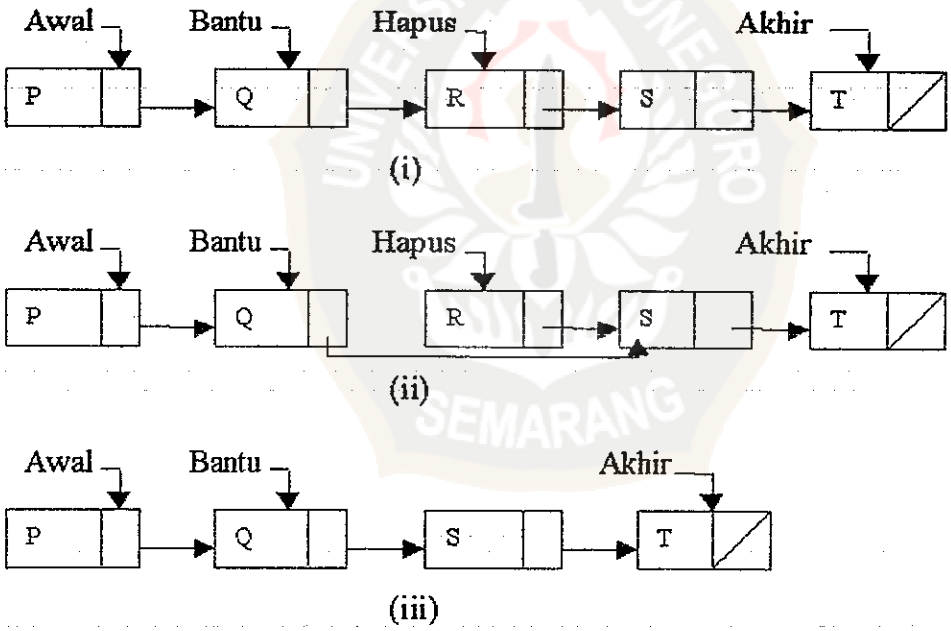


Gambar 2.8. Penghapusan node di didepan

Pada gambar 2.8. dapat dijelaskan sebagai berikut, (gambar 2.8. (ii)) pointer Bantu dibuat menunjuk lokasi yang ditunjuk pointer Awal. Kemudian pointer Awal di pindah ke node yang ditunjuk pointer pada node yang ditunjuk oleh pointer Bantu (gambar 2.8. (iii)), selanjutnya node yang ditunjuk oleh pointer Bantu di Dispose (gambar 2.8. (iv)).

b. Menghapus node di tengah

Ilustrasi penghapusan node di tengah dapat disajikan sebagai berikut :

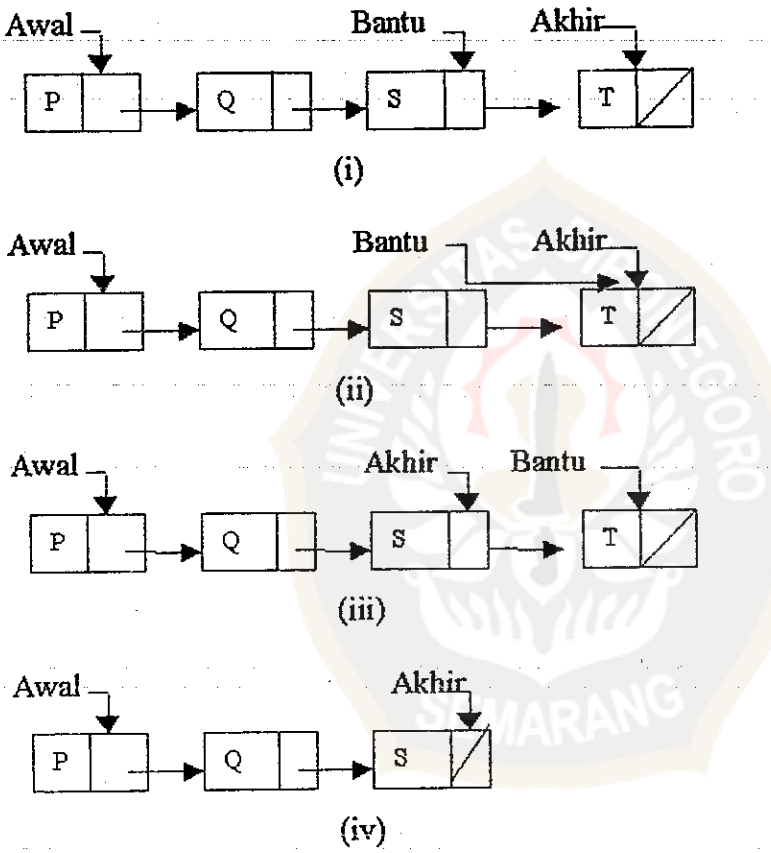


Gambar 2.9. Penghapusan node di tengah

Pada gambar 2.9. dapat dijelaskan sebagai berikut, (gambar 2.9. (i)) pointer Bantu diletakkan disebelah kiri node yang akan dihapus dengan node yang akan dihapus ditunjuk oleh pointer Hapus. Kemudian pointer pada node yang ditunjuk oleh Bantu ditunjukkan pada node yang ditunjuk oleh pointer pada node yang akan dihapus (gambar 2.9. (ii)), selanjutnya node yang ditunjuk oleh pointer Hapus di Dispose (gambar 2.9. (iii)).

c. Menghapus node terakhir

Ilustrasi penghapusan node terakhir dapat disajikan sebagai berikut :



Gambar 2.10. Penghapusan node terakhir

Pada gambar 2.10. dapat dijelaskan sebagai berikut, (gambar 2.10. (ii)) pointer Bantu dibuat menunjuk lokasi yang ditunjuk pointer Akhir. Kemudian pointer Akhir dipindah ke node yang menunjuk ke node yang ditunjuk oleh pointer Bantu (gambar 2.10. (iii)), selanjutnya node yang ditunjuk oleh pointer Bantu di Dispose (gambar 2.10. (iv)).