BAB II

DASAR TEORI

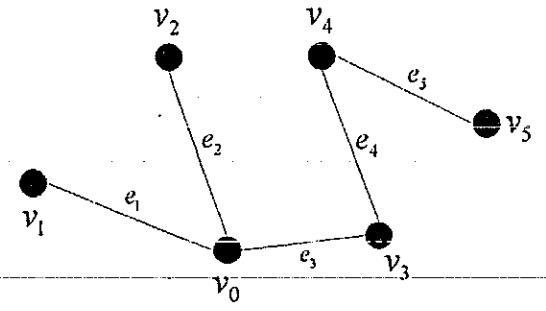
Pada Bab ini akan dibahas beberapa konsep dasar yang meliputi teori Graph, Pohon, Batas Atas dan Pengertian Algoritma Branch and Bound yang merupakan dasar dari penyelesaian masalah knapsack dengan Algoritma Branch and Bound.

2.1 Graph

Definisi 2.1.1 :

Suatu graph G didefinisikan sebagai himpunan sejumlah berhingga obyek yang disebut titik *(vertek, point, atau node)* dinyatakan dengan $V(G)$ dan suatu himpunan $E(G)$ yang merupakan himpunan pasangan elemen-elemen dari $V(G)$ yang disebut garis *(edges)*, dengan $E(G)$ boleh kosong. Dinotasikan $G=\{V,E\}$, dengan $V=\{v_1,v_2,...,v_n\}$ dan $E=\{e_1,e_2,...,e_n\}$, $e_i=(v_{i-1},v_i)$, $i=1,2,\ldots,n$.

Contoh 2.1.1 :



Gambar 2.1.1 Graph

Graph pada Gambar 2.1.1. terdiri dari himpunan vertex-vertex: :

$$V(G) = \{v_0, v_1, v_2, v_3, v_4, v_5\}$$

Dan himpunan edge

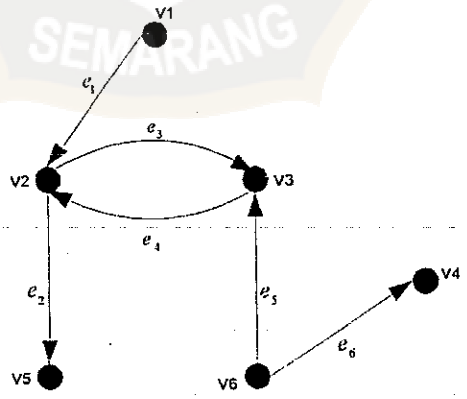
$$E(G) = \{e_1, e_2, e_3, e_4, e_5\}, e_1 = (v_0, v_1); e_2 = (v_0, v_2); e_3 = (v_0, v_3); e_4 = (v_3, v_4);$$

$$e_5 = (v_4, v_5)$$

Definisi 2.1.2 :

Suatu graph *G* disebut **graph tak berarah** *(undirected graph)* jika *E(G)* beranggotakan garis-garis yang tidak berarah. Sedangkan dikatakan **graph berarah** *(directed graph atau digraph)* jika garis-garis dalam graph *G* mempunyai arah. Graph berarah dinotasikan dengan *D = (V,E)* dan anggota-anggota *E(D)* disebut dengan busur *(arc)*

Contoh 2.1.2 :



Gambar 2.1.2 Graph berarah

Pada gambar 2.1.2 graph berarah mempunyai $V(D)=\{v_1,v_2,v_3,v_4,v_5,v_6\}$ dan $E(D)=\{(v_1,v_2),(v_2,v_3),(v_3,v_2),(v_2,v_5),(v_6,v_3),(v_6,v_4)\}$ atau $\{e_1,e_2,e_3,e_4,e_5,e_6\}$. Busur (v_2,v_3) dan (v_3,v_2) mempresentasikan dua garis yang mempunyai arah yang berbeda.

Definisi 2.1.3 :

Titik v dan w disebut sebagai titik akhir *(endpoint)* dari garis e, jika garis e menghubungkan titik v dan w.

Contoh 2.1.3 :

Pada graph Gambar 2.1.1 titik v_4 dan v_5 merupakan titik akhir garis e_5.

Definisi 2.1.4 :

Suatu garis e dikatakan *incident* dengan titik v, jika titik v merupakan titik akhir dari garis e.

Contoh 2.1.4 :

Garis e_4 dalam graph pada gambar 2.1.1 *incident* dengan titik v_3 dan titik v_4.

Definisi 2.1.5 :

Dua titik dikatakan *adjacent* jika ada garis yang secara langsung menghubungkan kedua titik tersebut, sedangkan dua garis dikatakan *adjacent* jika garis-garis tersebut *incident* pada titik yang sama.

Contoh 2.1.5

Graph pada Gambar 2.1.1, titik v_2 *adjacent* dengan titik v_0 membentuk garis e_2.

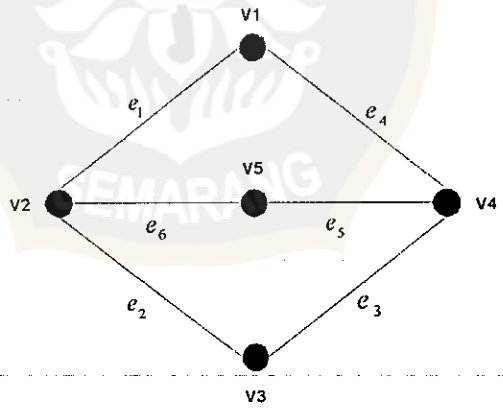
Definisi 2.1.6 :

Jalan *(walk)* dalam suatu graph G adalah barisan titik-titik dan garis-garis yang mempunyai bentuk

$$W : v_0, e_1, v_1, e_2, \ldots, v_{n-1}, e_n, v_n \quad (n \geq 0),$$

dimulai dan diakhiri oleh titik, sedemikian sehingga setiap garis *e incident* dengan titik *v*. Panjang suatu jalan dihitung berdasarkan jumlah garis dalam jalan tersebut.

Contoh 2.1.6 :



Gambar 2.1.3.Jalan dalam graph

Jalan dari graph pada gambar 2.1.3 adalah barisan $v_2, e_1, v_1, e_4, v_4, e_3, v_3, e_2, v_2, e_6, v_5, e_5, v_4$ yang mempunyai panjang 6

Definisi 2.1.7 :

Misalkan G adalah sebuah graph dan misalkan u dan v adalah titik-titik dalam G. Sebuah lintasan dari u hingga v dengan panjang n adalah barisan garis yang berbeda (satu sama lain) dari u sampai v dengan panjang n. Sebuah lintasan sederhana *(simple path)* dari u hingga v dengan panjang n adalah lintasan yang berbentuk :

$$(v_0, v_1, \ldots, v_n)$$

dengan $v_0 = u, v_n = v$ dan $v_0, v_1, \ldots, v_n$ saling berbeda.

Sirkuit *(circuit, cycle)* dalam suatu graph adalah suatu lintasan yang tertutup, dalam arti bahwa titik awal lintasan sama dengan titik akhir lintasan. Sirkuit sederhana adalah lintasan sederhana yang tertutup.

Contoh 2.1.7 :

Lintasan dalam graph Gambar 2.1.3 tersebut adalah $(v_1, v_2, v_5, v_4, v_3)$ atau (e_1, e_6, e_5, e_3) dan (e_1, e_2, e_3, e_4) atau $(v_1, v_2, v_3, v_4, v_1)$ merupakan suatu sirkuit.

Definisi 2.1.8 :

Suatu graph G disebut terhubung *(connected)* jika diberikan sebarang titik-titik u dan w yang berbeda, terdapat suatu lintasan dari u hingga w.

Definisi 2.1.9 :

Derajat *(degree)* dari titik v dinotasikan $\deg(v)$ adalah banyaknya garis *(edge)* yang incident titik v.

Contoh 2.1.9 :

Derajat dalam graph Gambar 2.1.1 diperoleh :

$$\deg(v_1) = \deg(v_2) = \deg(v_5) = 1$$

$$\deg(v_3) = \deg(v_4) = 2$$

$$\deg(v_0) = 3$$

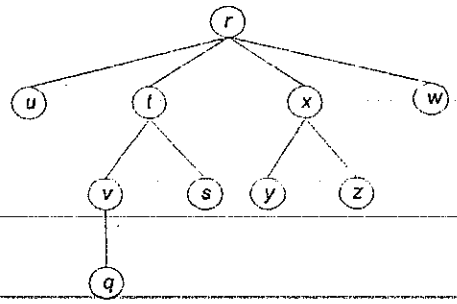
2.2 Pohon

Definisi 2.2.1 :

Pohon *(tree)* adalah suatu graph sederhana yang mempunyai sifat yaitu terdapat sebuah lintasan tunggal *(unique)* diantara setiap pasang titik.

Pohon berakar *(rooted tree)* adalah sebuah pohon yang mempunyai titik tertentu yang dijadikan sebagai akar *(root)*

Contoh 2.2.1 :



T

Gambar2.2.1.Pohon Berakar *(rooted tree)*

Teorema 2.2.1.1 :

Misalkan T adalah pohon dengan n titik. Berikut adalah ekuivalen :

1. T adalah terhubung dan tidak memuat sirkuit
2. T adalah terhubung dan memiliki $n-1$ garis
3. T tidak mempunyai sirkuit dan mempunyai $n-1$ garis

Bukti :

Andaikan T terhubung dan tidak memuat sirkuit. Akan dibuktikan bahwa T memiliki $n-1$ garis dengan induksi pada n. Jika $n=1$, maka T terdiri dari satu titik dan nol garis, jadi benar jika $n=1$. Sekarang andaikan hasilnya berlaku untuk graph yang terhubung dan tidak memuat sirkuit dengan n titik. Misalkan T graph yang terhubung dan tidak memuat sirkuit dengan $n+1$ titik. Pilih sebuah lintasan P dengan panjang maksimal. Karena T adalah tidak memuat sirkuit, P bukan sebuah sirkuit, maka P memuat sebuah titik u dengan derajat satu. Misalkan T^* adalah T dengan titik u dan sebuah garis incident terhadap u dihapuskan maka T^* adalah terhubung dan tidak memuat sirkuit. Karena T^* memuat n titik, dengan induksi T^* memuat $n-1$ garis. Oleh karena itu T memuat n garis. Dengan demikian telah ditunjukkan (1) maka berlaku (2).

Sekarang andaikan T terhubung dan memiliki $n-1$ garis. Harus ditunjukkan bahwa T tidak memuat sirkuit. Andaikan T memuat paling tidak satu sirkuit. Karena menghapuskan sebuah garis dari sebuah sirkuit tidak berarti *disconnect* (tidak terhubung) sebuah graph, maka garis dapat dihapuskan tetapi tidak berlaku untuk titik dalam sebuah sirkuit dalam T sampai graph yang dihasilkan T^* tidak

memuat sirkuit. Dapat digunakan hasil diatas (1) maka berlaku (2), untuk menyimpulkan bahwa T^* memiliki $n-1$ garis. Dengan demikian T memikili lebih dari *n-1* garis. Berarti kontradiksi. Oleh karena itu T tidak memuat sirkuit. Terbukti (2) maka berlaku (3).

Akhirnya, andaikan T tidak memuat sirkuit dan tidak memiliki *n-1* garis. Akan ditunjukkan bahwa T adalah sebuah pohon yaitu T memiliki lintasan tunggal diantara setiap pasangan titik yang tertentu.

Pertama-tama akan ditunjukkan bahwa T adalah terhubung. Andaikan dengan cara kontradiksi, T tidak terhubung. Misalkan

$$T_1, T_2, \ldots, T_k$$

adalah komponen-komponen dari T. Karena T tidak terhubung, $k > 1$. Andaikan T_i memiliki n_i titik. Tiap-tiap T_i adalah terhubung dan tidak memuat sirkuit, maka hasil pembuktian sebelumnya (1) maka berlaku (2) dapat digunakan untuk menyimpulkan bahwa T_i memiliki $n_i - 1$ garis. Sekarang

$$\begin{aligned} n-1 &= (n_1-1)+(n_2-1)\ldots+(n_k-1) && \text{(garis yang dihitung)} \\ &< (n_1+n_2+\ldots+n_k)-1 && \text{(karena k>1)} \\ &= n-1 && \text{(titik yang dihitung)} \end{aligned}$$

adalah tidak mungkin oleh karena itu T adalah terhubung.

Andaikan terdapat lintasan-lintasan berbeda P dan $(v_o, \ldots, v_n)$ dari a ke b dalam T. Maka P diikuti dengan $(v_n, \ldots, v_o)$ adalah sebuah sirkuit, yang berarti tidak mungkin. Jadi terdapat lintasan tunggal diantara setiap pasangan titik dalam

T. Oleh karena itu T adalah sebuah pohon. Dengan demikian terbukti bahwa (3) maka berlaku (1) dan semua kondisi ini adalah ekivalen.

Definisi 2.2.4 :

Misalkan T adalah pohon berakar dengan akar v_0. Andaikan x, y dan z adalah titik-titik dalam T dan $\{v_1, v_2, \ldots, v_n\}$ adalah suatu lintasan dalam T maka :

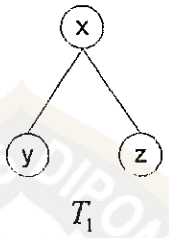
1. v_{n-1} adalah **orang tua** *(parent)* atau ayah dari v_n.
2. $v_{0,\ldots,}v_{n-1}$ adalah **pendahulu** *(ancestor)* dari v_n.
3. v_n adalah **anak** *(children)* dari v_{n-1}
4. Jika x adalah pendahulu y, maka y adalah **turunan** *(descendant)* dari x.
5. Jika x dan y anak dari z, maka x dan y **bersaudara** *(siblings)*
6. Jika x tidak memiliki anak, x adalah **titik ujung** *(terminal vertex)* atau **daun** *(leaf)*
7. Jika x bukan titik ujung, x adalah **titik dalam** *(internal vertex)* atau **titik cabang** *(branch vertex).*
8. Subgraph T yang terdiri dari x dan semua descendant-nya, dengan x dijadikan sebagai akar adalah **subpohon** *(subtree)* dari T yang berakar pada x.

Contoh 2.2.4 :

Dalam pohon berakar dari Gambar 2.2.1.diperoleh :

1. Orang tua atau ayah dari titik u adalah titik r.
2. Pendahulu dari titik s adalah titik t dan r
3. Anak dari titik r adalah titik-titik u, t, x dan w.

4. Turunan dari titik t adalah titik-titik v, s dan q.
5. Titik y dan z adalah bersaudara.
6. Titik ujung adalah titik-titik u, q, s, y, z, w.
7. Titik dalam adalah titik-titik r, t, x, v.
8. T_1 merupakan subpohon dari Gambar 2.2.1 yang berakar pada x.



T_1

Gambar 2.2.2 Sub Pohon

2.3. Batas Atas

Definisi 2.3.1. :

(Bartle,1982) Misalkan S himpunan bagian R. $u \in R$ dikatakan batas atas S jika $s \leq u, \forall s \in S$. Himpunan s dikatakan keatas jika himpunan tersebut mempunyai batas atas.

Contoh 2.3.1 :

$S = \{1,2,3,4,5\}$

Himpunan S adalah keatas

Bukti :

Akan dibuktikan bahwa himpunan S mempunyai batas atas

Misal diambil $5 \in R$ sebagai batas atas, karena 5 = 5, 4<5, 3<5, 2<5 dan 1<5 maka $S \leq 5$ untuk setiap $s \in S$.

Berdasarkan definisi diatas maka 5 adalah batas atas dari himpunan S sehingga himpunan S adalah keatas

2.4. Algoritma

2.4.1. Pengertian Algoritma

Algoritma berasal dari kata *algoris* dan *ritmis* yang diperkenalkan pertama kali oleh *Abu Ja'far Ibnu Musa Al Khawarismi* matematikawan Arab abad 19 dalam bukunya *Kitab Al-jab w'al muquabala.* Secara umum algoritma adalah suatu urutan dari barisan langkah-langkah yang digunakan untuk menyelesaikan suatu masalah.

Algoritma didefinisikan sebagai metode khusus yang terdiri dari serangkaian langkah yang terstruktur dan dituliskan secara sistematis yang akan dikerjakan untuk menyelesaikan suatu masalah dengan bantuan komputer. Hubungan antara masalah, algoritma, dan solusi digambarkan dalam bagan sebagai berikut :

Gambar 2.4.1. Hubungan antara Masalah, Algoritma dan Solusi

Ada beberapa sifat umum algoritma. Sifat-sifat tersebut digunakan untuk mempermudah dalam mengingat sebuah algoritma dideskripsikan. Sifat-sifat tersebut adalah :

1. *Input* (masukan)

   Suatu algoritma dapat mempunyai nol atau banyak input, yaitu suatu kuantitas yang diberikan nilai awal sebelum algoritma dimulai. Input diperoleh dari himpunan objek yang telah ditetapkan.

2. *Output* (keluaran)

   Dari setiap nilai input sebuah algoritma menghasilkan output (keluaran) dari himpunan yang dispesifikasikan. Nilai output adalah penyelesaian dari permasalahan. Suatu algoritma dapat mempunyai satu atau banyak output.

3. *Definiteness* (tertentu)

   Langkah-langkah sebuah algoritma harus didefinisikan dengan tepat, tindakan penyelesaian harus ditetapkan dengan tepat dan jelas untuk setiap kasus.

4. *Finiteness* (terbatas)

   Dengan langkah yang terbatas atau berakhir sebuah algoritma harus dapat menghasilkan output yang diinginkan setelah mencapai akhir langkah-langkah untuk sebarang himpunan input.

5. Efektif dan efisien

Algoritma dikatakan efektif jika algoritma tersebut menghasilkan solusi yang sesuai dengan masalah yang diselesaikan (tepat guna). Dan dikatakan efisien waktu proses algoritma tersebut relatif singkat dan penggunaan memorinya relatif kecil.

Algoritma dapat dinyatakan dalam beberapa bentuk dan cara yaitu sebagai berikut:

1. Bahasa tingkat rendah

Algoritma ditulis dalam bentuk bahasa sehari-hari.

2. *Pseudocode* (Kode semu)

Algoritma dinyatakan dengan perintah bahasa pemrograman tertentu dan ditambah dengan bahasa alamiah.

3. Bagan alir *(flowchart)*

Algoritma ditulis dalam bentuk diagram alir atau simbol.

2.4.2. Algoritma Branch and Bound

Algoritma Branch and Bound merupakan penyelesaikan masalah dengan membagi masalah menjadi submasalah yang lebih kecil sampai mendapat solusi secara optimal.

Pada penyelesaian masalah dengan menggunakan algoritma branch and bound berjalan dari satu solusi fisibel ke solusi fisibel lain sampai mendapatkan

solusi yang optimal. Ada empat elemen penting dalam penyelesaian algoritma branch and bound yaitu sebagai berikut :

1. Pembagian (*Partition*)

   Adalah membagi masalah ke bagian sub masalah sampai mendapat solusi fisibel. Strategi pemisahan semula dikemukakan oleh Land and Doig (1960) yang memisah s_k dengan k adalah titik, kedalam himpunan :

$$s_k \cap \{x | x_r = 0,1,....,u_r\}$$

   dimana u_r adalah batas atas dari x_r.

2. Batas (*bound*)

   Dalam penentuan batas maksimal dari solusi yang dicari, terdapat batas atas misalkan u_r dalam x_r untuk setiap r bilangan bulat, total bilangan dari nilai-nilai fisibel untuk variabel x_r dengan r adalah bilangan bulat $(r \in R)$. Selama struktur pohon yang mendasari tidak ada yang lebih dari pembagian berturut-turut dari himpunan

$$X(R) = \{x_j | 0 \le x_r \le U_r, r \in R\}$$

   dan selama masing-masing langkah pembagian individu mempunyai dua unsur tidak kosong dari $X(R)$.

3. Solusi tidak fisibel dari sub masalah

Pengabaian solusi yang tidak fisibel, dalam hal ini sub masalah yang tidak dilakukan pencabangan dari hasil pembagian masalah. Nilai yang fisibel akan terus digunakan sampai menemukan solusi optimal, sedangkan solusi yang tidak fisibel akan berhenti dengan kata lain tidak digunakan lagi.

4. Pencabangan (*branching*)

Dalam penyelesaian pada sub masalah dengan pencabangan dapat diketahui suatu keputusan yang akan digunakan untuk menentukan langkah selanjutnya.