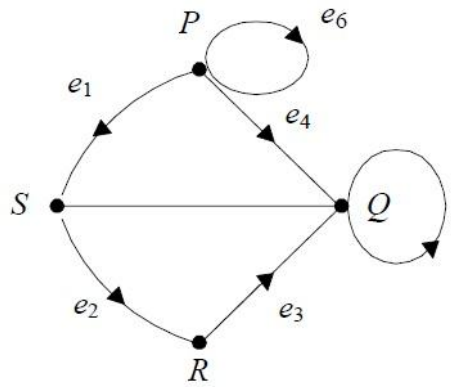
II. LANDASAN TEORI

Ide Leonard Euler di tahun 1736 untuk menyelesaikan masalah jembatan Konisberg yang kemudian menghasilkan konsep graf *Eulerian* merupakan awal dari lahirnya teori graf. Euler mengilustrasikan permasalahan tersebut dalam bentuk sketsa titik dan garis yang masing – masing merupakan representasi dari daratan dan jembatan, dimana daratan dinyatakan dengan titik atau *vertex* dan jembatan dinyatakan dengan garis atau *edge*. Ilustrasi dari Euler ini melahirkan suatu konsep yang disebut dengan graf. Berdasarkan ilustrasi yang dilakukan oleh Euler jadi, graf merupakan kumpulan objek berhingga dari titik dan garis, dimana titik – titik tersebut dapat terhubung ataupun tidak, dihubungkan oleh garis yang berarah maupun tidak berarah, dan dapat pula terhubung dengan dirinya sendiri.

Menurut Deo (1989), suatu graf G = (V,E) terdiri dari himpunan objek $V = \{v_1, v_2, \dots\}$ yang disebut *vertex* (titik) yang tidak kosong, dan himpunan lain $E = \{e_1, e_2, \dots\}$ yang unsur-unsurnya disebut *edge* (garis) yang boleh kosong, sehingga setiap *edge* e_k diidentifikasikan sebagai pasangan bukan berurutan (v_i, v_j) dari *vertex*. Representasi paling umum dari graf adalah dengan cara diagram, di mana *vertex* direpresentasikan sebagai titik dan setiap *edge* sebagai garis yang menghubungkan *vertex*. Graf yang terhubung dengan *edge* berarah

disebut dengan *directed graph* atau graf berarah, sedangkan graf yang terhubung dengan *edge* tanpa arah disebut dengan *undirected graph* atau graf tidak berarah.

Lipschutz and Lipson (2002) mengatakan suatu graf berarah (*digraph*) G terdiri dari sebuah himpunan V(G) yang elemen – elemennya disebut titik (*vertex*) dan suatu koleksi E(G) dari pasangan – pasangan *vertex* terurut yang disebut garis (*edge*) berarah (*arc*). Graf berarah G dikatakan berhingga jika himpunan V dari *vertex* – *vertex* berhingga. Contoh graf berarah dapat dilihat pada Gambar 2.1.



Gambar 2.1. contoh graf berarah (*digraph*).

Setiap *vertex* pada graf dapat dihubungkan ke setiap *vertex* lainnya atau tidak dihubungkan sama sekali. Karena itu, masing – masing *vertex* akan mempunyai sejumlah *edge* tertentu yang menempel pada *vertex* tersebut. Definisi berikut adalah definisi tentang *degree*.

Definisi 2.1 *Degree*

Derajat (*degree*) adalah jumlah *edge* yang menempel pada suatu *vertex* v_i, dengan *loop* dihitung dua kali dan ditulis dengan $d(v_i)$ (Deo, 1989) .

Degree pada graf berarah (*digraph*) memiliki perbedaan dengan *degree* pada graf tidak berarah. Definisi berikut adalah tentang *Degree* pada graf berarah (*digraph*).

Definisi 2.2 Derajat Keluar (*outdeg*) dan Derajat Masuk (*indeg*)

Suatu *edge* berarah $e = (u, v)$ dikatakan mulai pada titik awal *u* dan berakhir dititik akhir *v*. Derajat keluar dari *v*, ditulis dengan *outdeg(v)* adalah jumlah *edge* yang keluar dari *v*, dan derajat masuk dari *v*, ditulis dengan *indeg (v)* adalah jumlah *edge* yang menuju *v*. Pada Gambar 2.1 *indeg* (R) adalah 1 dan *outdeg* (R) adalah 1 (Lipschutz and Lipson, 2002).

Pada pernyataan sebelumnya, *vertex* - *vertex* suatu graf dapat terhubung ataupun tidak terhubung. Definisi berikut adalah tentang keterhubungan *vertex* oleh suatu *edge* pada suatu graf.

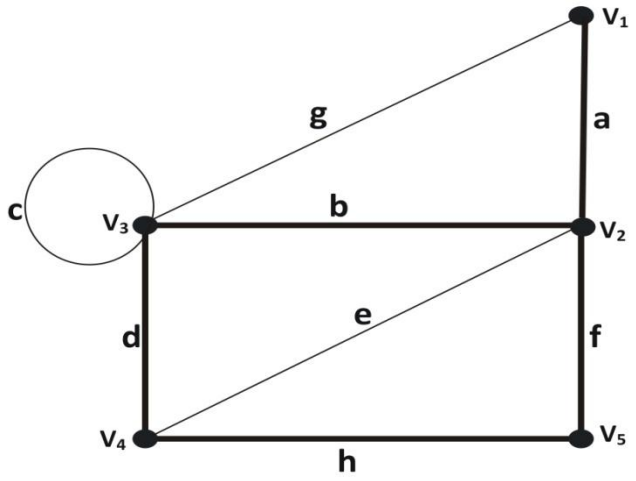
Definisi 2.3 Bertetangga (*Adjacent*) dan Menempel (*Incidence*)

Vertex v_i dan *edge* e_j dikatakan *incident* dengan satu sama lain jika *vertex* v_i adalah suatu *vertex* ujung dari *edge* e_j. Dua *edge* yang tidak paralel dikatakan *adjacent* jika kedua edges tersebut *incident* pada satu *vertex* dan dua vertex dikatakan *adjacent* jika kedua vertex tersebut adalah *vertex* akhir dari *edge* yang sama. *Isolated vertex* merupakan *vertex* yang tidak terhubung dengan *vertex* lainnya pada suatu graf dan mempunyai *degree* bernilai 0 (Deo, 1989).

Dengan adanya *adjacent* dan *incidence*, maka suatu graf dapat membentuk pola dari himpunan *adjacent* dan *incidence*. Terdapat beberapa istilah dalam teori, antara lain *walk*, *trail*, *path* dan *cycle*. Definisi dari *walk* dan *trail* diberikan sebagai berikut.

Definisi 2.4 *Walk* dan *Trail*

Walk adalah barisan berhingga dari titik (*vertex*) dan garis (*edge*), dimulai dan diakhiri dengan *vertex*, sedemikian sehingga setiap *edge* menempel dengan *vertex* sebelum dan sesudahnya. *Walk* yang berawal dan berakhir pada titik yang sama disebut *walk* tertutup (Deo, 1989).



Gambar 2.2. Contoh *walk* pada suatu graf.

Garis yang tebal pada Gambar 2.2 merupakan salah satu *walk* yaitu $v_1\ a\ v_2\ b\ v_3\ d\ v_4\ h\ v_5\ f\ v_2$. Sedangkan *trail* adalah suatu *walk* yang melewati garis (*edge*) yang berbeda. Pada Gambar 2.2 salah satu *trail*nya adalah $v_1\ a\ v_2\ b\ v_3\ d\ v_4\ h\ v_5$ (Deo, 1989).

Selain terbentuk karena pola dari himpunan *adjacent* dan *incidence, trail* yang terdapat pada *digraph* dapat membentuk *Eulerian digraph*. Definisi berikut adalah tentang *Eulerian digraph*.

Definisi 2.5 *Eulerian Digraph*

Graf berarah terhubung D adalah *Eulerian* jika terdapat *trail* tertutup yang semua *arc*nya ada di D (Wilson and Watkins, 1990).

Berikut ini adalah Teorema yang menghubungkan antara Graf *Eulerian* dengan derajat suatu *vertex*.

Teorema 1 (Wilson and Watkins, 1990)

Graf berarah G adalah graf berarah *Eulerian* jika dan hanya jika G terhubung dan derajat masuk sama dengan derajat keluar pada setiap *vertex*nya.

Bukti :

(⇒) Akan ditunjukkan Jika G *Eulerian*, maka setiap *vertex* dalam G memiliki derajat masuk yang sama dengan derajat keluar.

Misalkan G graf berarah *Eulerian*.

Dari Definisi 2.5 jelas bahwa G terhubung.

Ambil *walk* berarah tertutup $W_1 = v_1 a_1 \dots a_{n-1} v_n$ dengan $v_1 = v_n$.

Hapus semua derajat masuk dan derajat keluar dari setiap *vertex* di W_1.

Akibatnya, semua *arc* akan terhapus juga, sehingga diperoleh :

$$indeg(v) = outdeg(v) = 0$$

Sehingga terbukti bahwa : $indeg(v) = outdeg(v)$.

(⇐) Akan ditunjukkan Jika setiap *vertex* dalam G memiliki derajat masuk yang sama dengan derajat keluar, maka G adalah *Eulerian*.

Misalkan G terhubung, dengan derajat masuk sama dengan derajat keluar pada setiap *vertex*.

Jika G memiliki satu *vertex*, jelas G *Eulerian*.

Oleh karena itu, diasumsikan G memiliki *vertex* lebih dari satu.

Misalkan $v_1 v_2$ ... v_n barisan *vertex* pada suatu *walk* berarah di G yang terpanjang.

Setiap vertex memiliki derajat masuk dan derajat keluar yang sama kecuali di v_1 dan v_n.

Andaikan $v_1 \neq v_n$ maka $indeg(v_1) \neq outdeg(v_n)$

Hal ini menimbulkan kontradiksi. Akibatnya $v_1 = v_n$.

Akan ditunjukkan bahwa *walk* berarah tertutup W_1 di G memuat semua *arc* di G.

Andaikan tidak, maka akan terdapat *walk* berarah lain W_2 yang bukan *walk* berarah terpanjang $v_1 v_2$ ... v_n.

Karena G terhubung maka akan terdapat *arc* a dari W_2 yang *incident* dengan salah satu *vertex* di W_1, misalkan v_i dengan $i = 1,2, \dots, n$.

Jika v_i sebagai *vertex* awal dari a, walk berarah terpanjang dimulai dari v_i ke v_i ditambah dengan *arc* a. Hal ini kontradiksi dengan W_1 yang merupakan *walk* berarah tertutup.

Jika v_i sebagai *vertex* akhir dari a, *walk* berarah terpanjang dimulai dari *arc* a ditambah dengan v_i ke v_i. Hal ini kontradiksi dengan W_1 yang merupakan *walk* berarah tertutup.

Dari penjelasan sebelumnya, terlihat bahwa jika v_i dihapus maka *arc* yang *incident* dengannya juga terhapus.

Akibatnya, W_1 bukan lagi *walk* berarah terpanjang. Dengan demikian W_1 merupakan *walk* berarah tertutup yang memuat setiap *arc* D tepat satu kali.

Jadi, G merupakan graf berarah *Eulerian*. ■

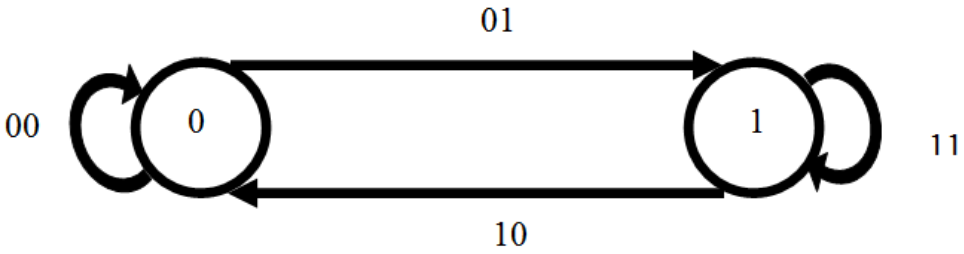
Eulerian yang tertutup disebut *Eulerian tour* (Misra, 2001).

Terdapat beberapa jenis graf, antara lain graf lengkap, graf sederhana, reguler, graf bipartite, graf terhubung, dan graf *deBruijn*. Graf deBruijn merupakan contoh dari *Eulerian tour*. Definisi dari graf *deBruijn* diberikan sebagai berikut.

Definisi 2.6 Graf *DeBruijn*

Graf deBruijn adalah suatu graf berarah $G_{a,n}$, $a \geq 2, n \geq 1$ yang dibentuk dari setiap bilangan bulat positif n dan a, yang berisi a^{n-1} *vertex* yang diberi label dengan *bitstring* panjangnya $n-1$, dan a^n *arc* yang diberi label dengan *bitstring* panjangnya n. *Arc* dari *vertex* $v_1 = b_1 b_2 \ldots b_{n-1}$ ke *vertex* $v_2 = b_2 b_3 \ldots b_n$ diberi label *vertex* $v_3 = b_1 b_2 \ldots b_n$ (Gross dan Yellen, 1999).

Contoh salah satu graf deBruijn dapat dilihat pada Gambar 2.3.



Gambar 2.3. Graf deBruijn $G_{2,2}$.

Dari Gambar 2.3, dapat dilihat bahwa sesuai dengan Definisi 2.6 bahwa graf deBruijn $G_{2,2}$ memiliki $a = 2, n = 2$ yang berisi $2^{2-1} = 2$ *vertex* yang diberi label dengan *bitstring* yang panjangnya $2 - 1 = 1$, dan $2^2 = 4$ *arc* yang diberi label dengan *bitstring* panjangnya 2. *Arc* dari *vertex* $v_1 = 0$ ke *vertex* $v_2 = 1$ diberi label *vertex* $v_3 = 01$, dan seterusnya.

Eulerian tour adalah dasar yang digunakan untuk mengembangkan graf *deBruijn* menjadi sebuah barisan *deBruijn*. Secara singkat, barisan *deBruijn* merupakan bagian dari graf *deBruijn* yang membentuk *Eulerian tour.*

Definisi 2.7 Barisan *DeBruijn*

Barisan deBruijn merupakan barisan yang dibentuk dari *string* berhingga dengan sifat – sifat tertentu, sedangkan *string* adalah barisan berhingga (*finite*) simbol – simbol. Sebagai contoh, jika a, b, dan c adalah tiga buah simbol maka abc adalah sebuah *string* yang dibangun dari ketiga simbol tersebut. *Substring* dari *string w* adalah *string* yang dihasilkan dari *string w* dengan menghilangkan nol atau lebih simbol – simbol paling depan dan atau simbol – simbol paling belakang dari *string w* tersebut.

Suatu *string* dengan panjang 2^n disebut barisan deBruijn $(2, n)$ jika setiap *string* dengan panjang n hadir tepat satu kali sebagai *substring* dari 2^n. Contoh *string* yang dibentuk dari alfabet berukuran 2 yaitu $\{0,1\}$ adalah barisan deBruijn $(2, n)$. Untuk kasus $n = 1,2,3,4$ barisan *deBruijn*nya sebagai berikut : 01, 0110, 01110100, 0000100110101111 (Gross dan Yellen, 2006).

Proses pembentukan barisan *deBruijn* adalah sebagai berikut :

Misalkan suatu *Eulerian Tour* dalam sebuah graf *deBruijn* terbentuk oleh beberapa *Arc* (*Edge* berarah) yaitu (abc, bcd, cde, def, efg, fgh). Maka, barisan *deBruijn*nya adalah abcdefgh (Gross dan Yellen, 2006).

Graf *deBruijn* banyak digunakan untuk menyelesaikan beberapa masalah dalam kehidupan nyata dengan menggunakan beberapa aplikasi. Salah satu aplikasi yang biasa digunakan adalah aplikasi *genetika* (*genetics application*) untuk memecahkan masalah seperti kemacetan dan berbagai masalah genetika lainnya. Algoritma genetika merupakan suatu bagian dari aplikasi genetika yang paling sering digunakan untuk menyelesaikan masalah *genetics application*.

Definisi 2.8 Algoritma Genetika

Algoritma genetika adalah algoritma yang dikembangkan dari proses pencarian solusi menggunakan pencarian acak, ini terlihat pada proses pembangkitan populasi awal yang menyatakan sekumpulan solusi yang dipilih secara acak (Goldberg, 1989).

Algoritma genetika memiliki beberapa istilah penting yang tidak dapat dipisahkan dari aplikasi genetika. Beberapa istilah tersebut adalah *genotype*, kromosom, populasi, nilai *fitness* dan individu.

Definisi 2.9 *Genotype* (*gen*), Kromosom, Populasi, Nilai *Fitness,* Individu

Genotype (gen) adalah suatu nilai yang menyatakan satuan dasar yang membentuk suatu arti tertentu dalam satu kesatuan gen yang dinamakan kromosom. Kromosom adalah gabungan *gen* – *gen* yang membentuk nilai tertentu.

Populasi merupakan sekumpulan individu dengan ciri-ciri yang sama (spesies) yang hidup menempati ruang yang sama pada waktu tertentu (Goldberg, 1989).

Nilai Fitness menyatakan seberapa baik nilai dari suatu individu atau solusi yang didapatkan. Individu menyatakan satu nilai atau keadaan yang menyatakan salah satu solusi yang mungkin dari permasalahan yang diangkat. Individu dinyatakan dalam 8 *gen biner* dengan batas 0 sampai dengan 1, yang berarti 1 *bit* setara dengan 2^{-8} yang disebut dengan nilai X. Sebagai contoh 10001001 = (128+8+1)/256 = 0.5352 (Goldberg, 1989). Aplikasi genetika bertujuan untuk mencari nilai *fitness* terbaik dari suatu populasi. Nilai *fitness* didapatkan dengan menggunakan fungsi *fitness*.

Definisi 2.10 Fungsi *Fitness*

Fungsi *fitness* adalah fungsi yang digunakan untuk mencari suatu nilai *fitness*. Fungsi *fitness* mempunyai daerah penyelesaian lebih besar dari nol (> 0) dan lebih kecil dari satu (< 1) untuk daerah asal yang lebih besar dari nol (> 0) dan lebih kecil dari satu (< 1), serta daerah penyelesaian lebih besar dari satu (> 1) atau lebih kecil dari nol (< 0) untuk daerah asal yang lainnya. Jika kita misalkan fungsi *fitness* sebagai $f(x)$, maka dalam bahasa matematika dapat dituliskan sebagai berikut :

$$f(x) = \begin{cases} 0 < f(x) < 1; \text{untuk } 0 < x < 1 \\ f(x) < 0 \text{ atau } f(x) > 1; \text{lainnya} \end{cases}$$

Beberapa contoh fungsi - fungsi *fitness* yaitu $f(x) = e^{-2x}\cos(3x)$, $f(x) = e^{-x}\cos(3x)$, $f(x) = e^{-x}\cos(2x)$.

Untuk mendapatkan nilai *fitness* terbaik dari suatu populasi, maka digunakan pengkombinasian individu. Pengkombinasian individu yang paling sering digunakan adalah *cross-over* (perkawinan silang).

Definisi 2.11 *Cross-Over* (Perkawinan Silang)

Cross-over (perkawinan silang) merupakan proses mengkombinasikan dua individu untuk memperoleh individu – individu baru yang diharapkan mempunyai *fitness* yang lebih baik. Sebagai contoh *cross-over* (perkawinan silang) adalah sebagai berikut:

Misalkan kita ambil sebarang induk yaitu Induk 1: 0 0 1 1 1 0 0 1 dan induk 2: 1 0 0 1 1 0 1 0, maka hasil dari *cross-over* (perkawinan silang) adalah anak 1: 0 0 1 1 1 0 1 1 dan anak 2: 1 0 0 1 1 0 0 0.