BAB IV

ALGORITMA PENGENALAN *LINE* DIGRAPH DAN GRAF ASALNYA.

Pada bab ini akan dibahas mengenai cara untuk mengenali suatu *line digraph* berdasarkan sifat – sifatnya dengan menggunakan suatu algoritma serta contoh penerapannya pada graf DNA. Akan diberikan suatu algoritma pengenalan *adjoint* [**Sys82**]. Algoritma ini mengenali apakah suatu graf merupakan *adjoint* atau bukan. Karena *line digraph* adalah *adjoint*, maka algoritma ini juga dapat digunakan untuk mengenali suatu *line digraph*. Pengenalan ini memanfaatkan pelabelan pada suatu graf, jika pelabelan berhasil maka graf tersebut adalah *adjoint*. Jika tidak berhasil maka graf tersebut bukan *adjoint*. Dari Teorema 5 pada bab sebelumnya, suatu graf merupakan *line digraph* jika suatu graf termasuk ke dalam kelas L_2^{∞} yang artinya jika suatu graf dapat dilabel dengan pelabelan-$(2,\infty)$ maka graf tersebut adalah suatu *line digraph* dan pasti memiliki graf lain sebagai graf asalnya. Dengan algoritma tersebut *line digraph* dapat dikenali apabila pelabelan yang diberikan memenuhi Definisi 3.1.

Oleh karena itu algoritma yang akan dibangun merupakan algoritma yang menghasilkan suatu pelabelan pada simpul dengan elemen labelnya (*tail,head*) dan jika ada busur yang menghubungkan simpul *u* ke simpul *v* maka nilai *head*(*u*) sama dengan *tail*(*v*).

Algoritma Pengenalan *Adjoint*

1. **begin**
2. **for** setiap $u \in H$ **do**
3. **begin**
4. tail(*u*) = 0;
5. head(*u*) = *n* + 1;
6. **end**
7. **set** *p* = 0;
8. **for** setiap $u \in H$ **do**
9. **if** $N^+(u) \neq \varnothing$ **then**
10. **begin**
11. k = tail(*w*); untuk suatu verteks *w* di $N^+(u)$
12. **if** k = 0 **then**
13. *p* = *p* + 1;
14. head(*u*) = *p*;
15. **else**
16. head(*u*) = *k*;
17. **for** setiap $w \in N^+(u)$ **do**
18. **if** tail(*w*) $\neq k$ **then** STOP
19. **else**
20. **if** k = 0 **then** tail(*w*) = *p*;
21. **end**
22. *H* adalah *adjoint*
23. **end**

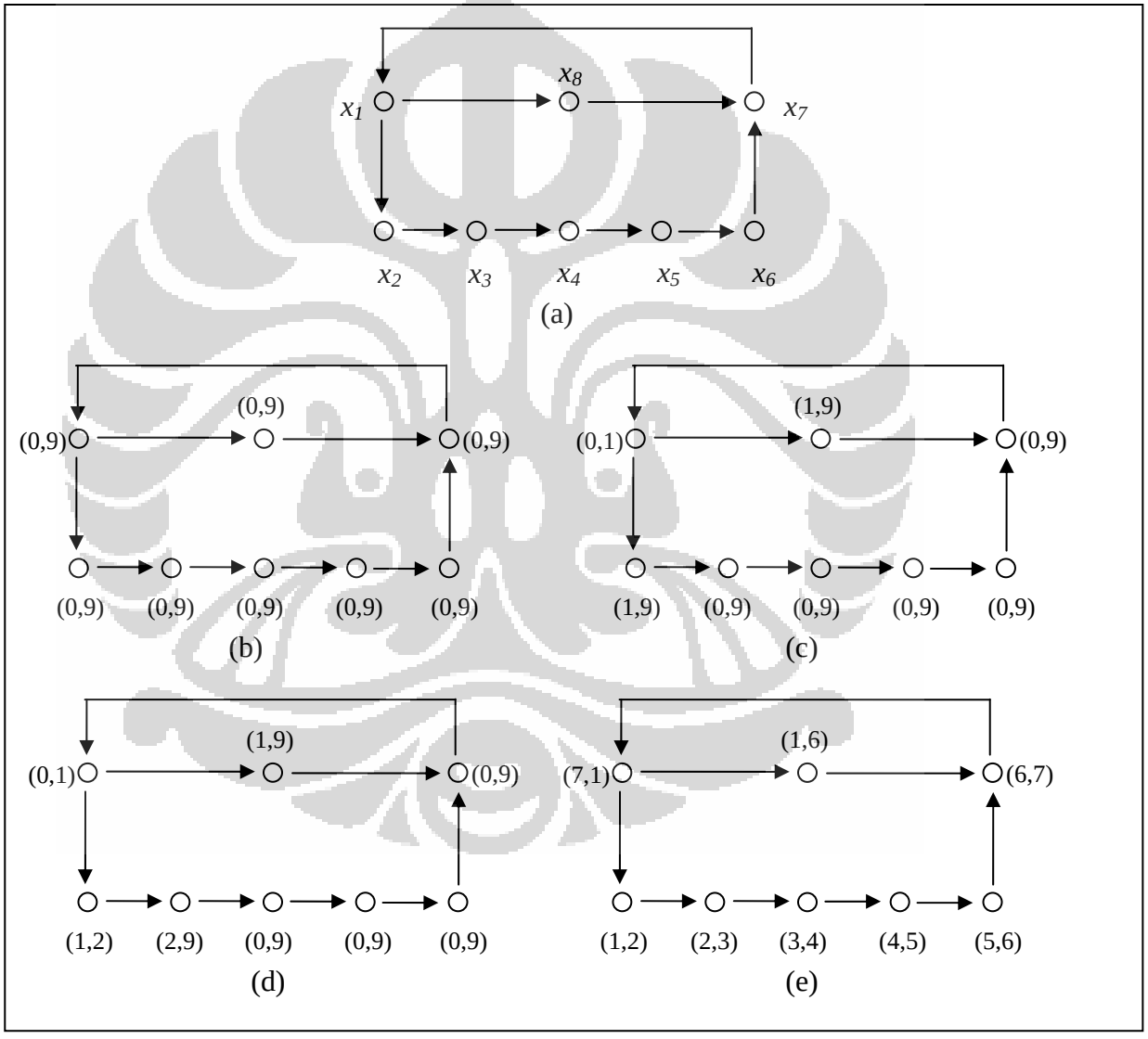
Gambar 4.1 Algoritma pengenalan *adjoint*

Pada Gambar 4.1 diberikan algoritma yang memberikan label dengan panjang 2 pada suatu graf *H* sehingga dari label yang dihasilkan dapat dibangun suatu graf asal dari graf *H*. Algoritma ini dapat digunakan untuk memberi label pada sembarang graf, namun pelabelan belum tentu berhasil dilakukan jika graf yang dilabel bukan graf-1. Jika pemberian label dengan algoritma ini berhasil, masih terdapat dua kemungkinan, yaitu semua label berbeda atau ada simpul – simpul yang labelnya sama. Jika setiap labelnya

berbeda *adjoint* tersebut adalah suatu *line digraph* sedangkan jika ada simpul – simpul yang labelnya sama maka akan terbentuk busur sejajar pada graf asal.

Pada algoritma ini, label untuk setiap simpul terdiri dari pasangan terurut (*tail*,*head*). Pertama dilakukan inisialisasi nilai awal (label awal) untuk tiap simpul pada graf *H* yang diberikan seperti yang dilakukan pada baris 2 – 6 dengan memberikan setiap *tail* bernilai nol dan *head* bernilai $n + 1$ (n menyatakan banyaknya simpul pada graf). Tahap berikutnya masuk ke tahap iterasi (baris 8 -21), dengan iterasi dilakukan sebanyak jumlah simpul pada graf. Pada tahap ini, nilai label akan berubah sehingga jika ada busur dari simpul x ke y maka nilai $head(x) = tail(y)$. Perubahannya dilakukan dengan cara menyesuaikan nilai label *head* dengan dari sembarang simpul dengan nilai *tail* dari setiap lingkungan kanannya. Label (*tail,head*) dari setiap simpul menyatakan adanya busur dari simpul *tail* ke *head* pada graf asal. Algoritma ini akan berhenti pada 2 kondisi. Pertama, jika terdapat nilai *tail* dari salah satu lingkungan kanan suatu simpul tidak bernilai nol dan juga tidak sama dengan nilai *head* simpul tersebut maka graf tersebut bukanlah suatu *adjoint*. Kedua, jika setiap simpul telah berhasil dilabel maka graf tersebut merupakan *adjoint* karena untuk setiap x dan y yang terhubung satu komponen label paling kanan dari simpul x sama dengan satu komponen label dari kiri simpul y. Jika setiap label yang diperoleh berbeda maka *adjoint* tersebut adalah suatu *line digraph*.

Untuk lebih jelasnya perhatikan contoh graf *H* seperti pada Gambar 4.2(a). Akan ditunjukan bahwa graf *H* tersebut adalah sebuah *line digraph*. Sesuai dengan algoritma di atas, sebagai inisiasi label awal pada simpul diberikan label (0,*n*+1) yakni (0,9) (Gambar 4.2(b)) dan *p* = 0.



Gambar 4.2 Graf *H* dan pelabelan graf *H*

Kemudian pada iterasi pertama pilih x_1 sebagai *u* (pemilihannya boleh bebas). Maka $N^+(x_1) = \{x_2, x_8\} \neq \varnothing$. Lakukan tahapan pada baris 11, *k* =

tail(x_2) = 0 (boleh bebas memilih simpul juga). Karena $k = 0$ maka $p = p + 1 = 1$ dan kita ubah *head*(x_1) = $p = 1$ (masuk ke dalam kondisi pemilihan baris 12 sampai 16), sehingga simpul x_1 memiliki label (0,1). Lalu ubah *tail* semua simpul yang termasuk dalam $N^+(x_1)$ dengan *p* jika nilai *tail* simpul tersebut adalah 0 (tahapan pada baris ke 17 sampai 20). Sehingga label untuk x_2 adalah (1,9) dan untuk x_8 juga (1,9) (Gambar 4.2(c)). Lanjut ke-iterasi kedua, misalkan dipilih simpul x_2 sebagai *u*. Maka $N^+(x_2) = \{x_3\} \neq \varnothing$ dan k = *tail*(x_2) = 0. Karena $k = 0$ maka $p = p + 1 = 2$ dan kita ubah *head*(x_2) = $p = 2$. sehingga simpul x_1 memiliki label (1,2). Lalu ubah *tail* semua simpul yang termasuk dalam $N^+(x_2)$ dengan *p* jika nilai *tail* simpul tersebut adalah 0 (Gambar 4.2(d)). Sehingga label untuk x_3 adalah (1,2). Begitu seterusnya sampai kedelapan simpul tersebut diproses menghasilkan Gambar 4.2(e). Karena graf *H* dapat dilabel dengan panjang dua dan tidak ada label yang sama maka graf *H* adalah suatu *line digraph*.

Pada Tabel 4.1 diberikan perhitungan kompleksitas dari algoritma pada Gambar 4.1. Kompleksitas dari algoritma ini dapat ditulis sebagai

$$T(n) = t_1(n) + t_2(n) + t_3(n*m) = (t_1 + t_2 + t_3)(n*m) = O(n)$$

Dengan *m* adalah banyaknya simpul yang berada pada lingkungan kanan dari simpul *u*, karena graf DNA bukanlah graf yang padat

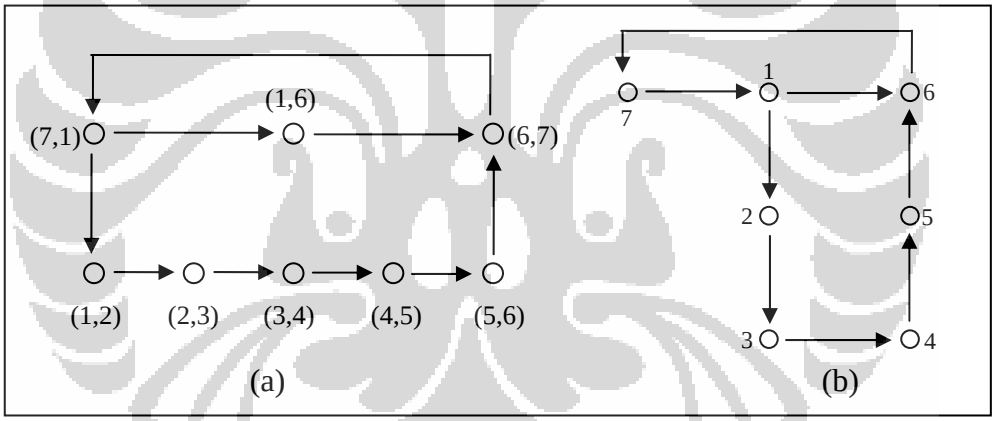
(keterhubungan antar simpul tidak banyak) maka *m* dapat dianggap konstan sehingga dapat disimpulkan algoritma pengenalan *adjoint* memberikan kompleksitas linear.

Tabel 4.1
Perhitungan Kompleksitas Algoritma Pengenalan *Adjoint*

Algoritma	Waktu	Jumlah
1. **begin**		
2. **for** setiap $u \in H$ **do**	t_1	n
3. **begin**		
4. tail(*u*) = 0;		
5. head(*u*) = n + 1;		
6. **end**		
7. **set** *p* = 0;		
8. **for** setiap $u \in H$ **do**	t_2	n
9. **if** $N^+(u) \neq \varnothing$ **then**		
10. **begin**		
11. k = tail(*w*); untuk suatu verteks *w* di $N^+(u)$		
12. **if** k = 0 **then**		
13. *p* = *p* + 1;		
14. head(*u*) = *p*;		
15. **else**		
16. head(*u*) = *k*;		
17. **for** setiap $w \in N^+(u)$ **do**	t_3	n*m
18. **if** tail(*w*) $\neq k$ **then** STOP		
19. **else**		
20. **if** k = 0 **then** tail(*w*) = *p*;		
21. **end**		
22. *H* adalah *line digraph*		
23. **end**		

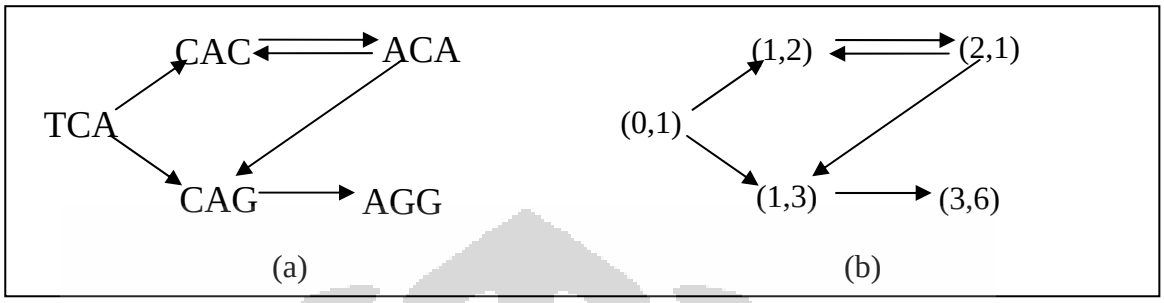
Setelah mendapatkan label dari tiap simpul pada graf *H*(*V,U*), dapat dibentuk graf asalnya, sebut graf *G*(*X,V*). Secara umum graf asal yang dihasilkan dari algoritma ini adalah graf yang memiliki busur – busur himpunan pasangan terurut label dari setiap simpul di *adjoint*.

Untuk graf pada Gambar 4.3(a) yang telah diberi label, graf asal yang terbentuk adalah graf yang memiliki busur – busur himpunan pasangan terurut label dari setiap simpul pada Gambar 4.3 (a) dan memiliki simpul sebanyak elemen label yang berbeda. Jadi pada graf *G* akan ada 7 simpul dan busur – busur {(7,1),(1,6),(1,2),(2,3),(3,4),(4,5),(5,6),(6,7)}. Sehingga akan diperoleh graf *G* (Gambar 4.3(b)) sebagai graf asal dari graf pada Gambar 4.3(a).



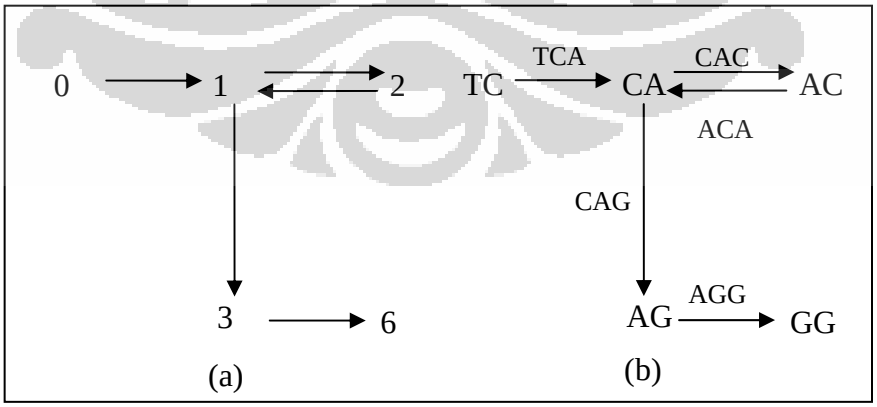
Gambar 4.3 (a) Graf *H*, (b) Graf *G* sebagai graf asal dari *H*

Algoritma pengenalan *adjoint* dan pembentukan graf asal dapat diterapkan pada graf DNA. Misalkan pada contoh graf DNA di Bab II sebelumnya. Karena setiap oligonukleotida merupakan simpul maka setiap oligonuklotida akan mendapat label dengan panjang dua. Dengan algoritma pengenalan *adjoint*, graf tersebut memiliki label yang berbeda sehingga merupakan suatu *line digraph* dan dihasilkan label seperti pada Gambar 4.4(b).



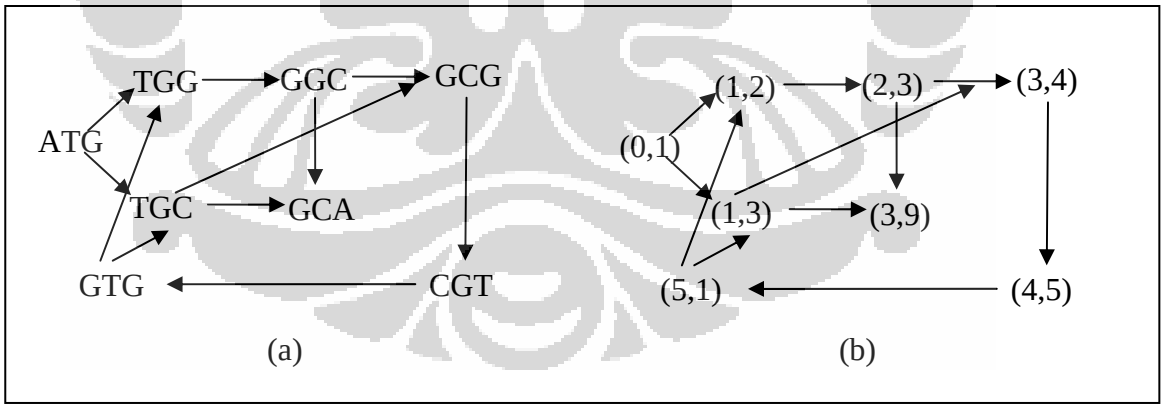
Gambar 4.4 Graf DNA

Komponen 0 akan merepresentasikan nukleotida TC, komponen 1 merepresentasikan nukloetida CA, komponen 2 merepresentasikan AC, komponen 3 merepresentasikan AG dan komponen 6 merepresentasikan GG. Graf asal dari graf DNA pada Gambar 4.4(a) akan seperti pada Gambar 4.5(a). Jika kita ganti label simpulnya dengan nukleotida yang berkorespondensi akan diperoleh Gambar 4.5(b).



Gambar 4.5 Graf asal dari graf DNA pada Gambar 4.3

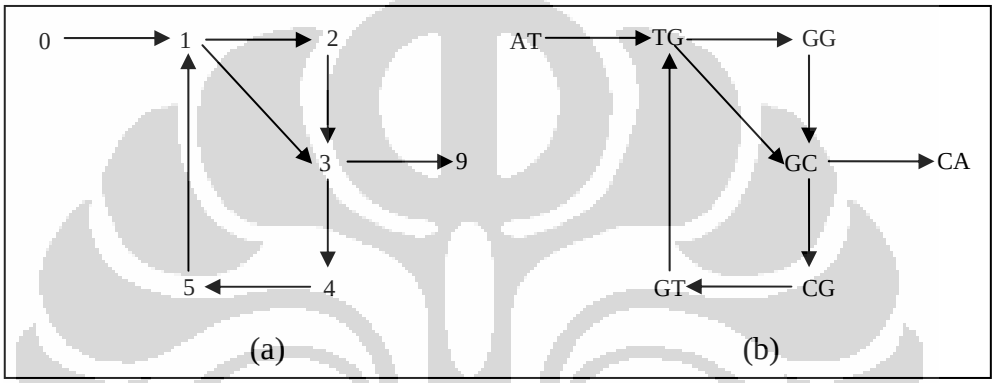
Lintasan Hamilton pada suatu graf belum tentu tunggal. Sebagai contoh diberikan spektrum {ATG, TCG, TGC, CTG, GGC, GCA, GCG, CGT} yang diperoleh pada tahapan biokimia. Graf DNA yang terbentuk akan seperti pada Gambar 4.6(a). Pada graf DNA ini terdapat dua lintasan Hamilton yang berbeda yaitu ATG – TGG – GGC – GCG – CGT – GTG – TGC – GCA dan lintasan ATG – TGC – GCG – CGT – GTG – TGG – GGC – GCA. Lintasan Hamilton yang pertama menghasilan barisan ATGGCGTGCA sedangkan lintasan Hamilton yang kedua menghasilkan barisan ATGCGTGGCA. Barisan DNA yang diperoleh juga berbeda. Jika graf tersebut dilabel dengan algoritma pengenalan *adjoint* akan menghasilkan Gambar 4.6(b).



Gambar 4.6 Graf DNA dengan 2 Lintasan Hamilton

Graf asal dari graf tersebut ada pada Gambar 4.7 yang juga akan menghasilkan lintasan Euler yang tidak tunggal yakni lintasan ATG – TGG – GGC – GCG – CGT – GTG – TGC – GCA dan lintasan ATG – TGC – GCG – CGT – GTG – TGG – GGC – GCA. Lintasan Euler yang pertama

menghasilkan barisan ATGGCGTGCA sedangkan lintasan Euler yang kedua menghasilkan barisan ATGCGTGGCA. Barisan DNA yang diperoleh juga berbeda. Sehingga belum tentu barisan yang dihasilkan dengan menelusuri lintasan Hamilton atau Euler tunggal.



Gambar 4.7 Graf asal dari graf DNA pada Gambar 4.6 dengan 2 Lintasan Euler