Penerapan *Graph Colouring* Untuk Merencanakan Jadwal

Hengky Budiman

NIM : 13505122

Program Studi Teknik Informatika, STEI
Institut Teknologi Bandung
Jl. Ganesha 10, Bandung
E-mail : if115122@students.if.itb.ac.id

Abstrak

Makalah ini membahas mengenai metode – metode mengenai cara membuat / merencanakan jadwal dengan menggunakan teknik mewarnai graf atau *graph colouring*. Pewarnaan graf di sini adalah pewarnaan simpul atau *vertex colouring* .

Pewarnaan simpul adalah teknik mewarnai simpul – simpul pada graf sehingga tidak ada simpul - simpul yang bertetangga, yaitu terhubung langsung dengan minimal sebuah sisi, memiliki warna yang sama. Biasanya hal ini juga dikaitkan dengan penggunaan warna yang seminimal mungkin.

Teknik pewarnaan graf merupakan salah satu subjek yang menarik dan terkenal dalam bidang graf. Teori – teori mengenainya telah banyak dikembangkan dan berbagai algoritma dengan kelebihan dan kelemahan masing – masing telah dibuat untuk menyelesaikannya. Aplikasi dari teknik ini juga telah banyak diterapkan di berbagai bidang, salah satunya adalah membuat jadwal. Perencanaan jadwal di sini khusunya diterapkan pada pekerjaan – pekerjaan atau hal – hal yang saling terkait, misalnya hal – hal yang berlangsung pada waktu yang sama, atau pekerjaan yang menggunakan sumber daya yang sama, dan sebagainya. Teknik pewarnaan graf akan membuat jadwal kerja yang dapat menghasilkan hasil yang maksimum dengan cara yang paling efisien.

Kata Kunci : graf, *graph colouring*, bilangan kromatik, *scheduling optimization*, *graph colouring algorithm*, aplikasi pewarnaan graph.

1. Pendahuluan

Graf adalah salah satu pokok bahasan Matematika Diskrit yang telah lama dikenal dan banyak diaplikasikan pada berbagai bidang. Secara umum, graf adalah pasangan himpunan (V,E) di mana V adalah himpunan tidak kosong dari simpul – simpul *(vertex* atau *node*) dan E adalah himpunan sisi (*edges* atau *arcs*) yang menghubungkan sepasang simpul pada graf tersebut.

$V= \{v_1,v_2,v_3,...,v_n\}$
$E= \{e_1,e_2,e_3,...,e_n\}$
Atau
$E= \{(v_1,v_2),(v_2,v_3),(v_3,v_4),...,(v_{n-1},v_n)\}$
Di mana $e=(v_i,v_j)$ yang artinya sisi yang menghubungkan simpul v_i dan v_j [6]

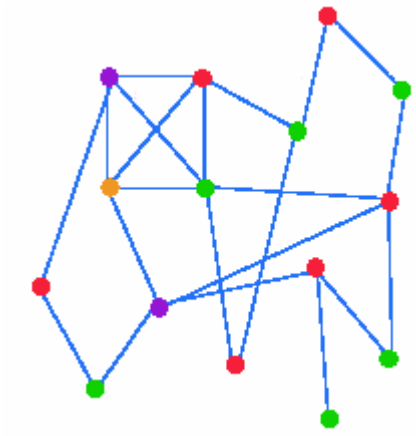
Kegunaan graf sangat banyak. Umumnya graf digunakan untuk memodelkan suatu masalah sehingga menjadi lebih mudah, yaitu dengan cara merepresentasikan objek – objek diskrit dan hubungan antara objek – objek tersebut. Contoh pemodelan suatu masalah dengan menggunakan graf dapat dilihat pada penggambaran rangkaian listrik, senyawa kimia, jaringan komunikasi, jaringan network komputer, analisis algoritma, peta, struktur hierarki sosial, dan lain – lain.

Salah satu topik yang menarik pada graf adalah masalah pewarnaan graf (*graph colouring*). Bidang ini memiliki sejarah yang sangat menarik dan teori – teorinya telah menimbulkan banyak perdebatan pada kalangan matematikawan.

Terdapat tiga macam pewarnaan graf, yaitu pewarnaan simpul (vertex colouring), pewarnaan sisi (edge colouring), dan pewarnaan wilayah (face colouring). Teknik yang dibahas pada makalah ini adalah pewarnaan simpul, teknik yang lain tidak dibahas karena teknik – teknik

tersebut hanyalah merupakan bentuk lain dari pewarnaan simpul dan dapat diubah kembali menjadi model pewarnaan simpul. Misalnya pewarnaan sisi adalah pewarnaan simpul dengan sisinya dianggap sebagai simpul dan pewarnaan wilayah adalah pewarnaan simpul dari graf dual planarnya

Masalah utama dalam pewarnaan simpul graf adalah bagaimana mewarnai semua simpul pada graf sehingga tidak ada simpul – simpul yang bertetangga, yaitu dihubungkan oleh minimal satu buah sisi, memiliki warna yang sama. Hal ini biasanya kemudian dikaitkan dengan penggunaan warna yang seminimum mungkin. Jumlah warna minimum yang dapat digunakan untuk mewarnai semua simpul disebut *bilangan kromatik* dari graf G, dan disimbolkan dengan X(G). Contohnya graf berikut :



$\chi(G) = 4$

Gambar 1.1
Pewarnaan Simpul Graph dengan $\chi(G) = 4$

Masalah pewarnaan graf diyakini pertama kali muncul sebagai masalah pewarnaan peta, di mana warna setiap daerah pada peta yang berbatasan dibuat berlainan sehingga mudah untuk dibedakan. Hal ini kemudian mengembangkan teorema – teorema menarik dan berujung pada teorema 4 warna, yang menyatakan : *"bilangan kromatik graf planar tidak lebih dari 4."* Teorema ini pertama kali muncul sebagai suatu perkiraan oleh Francis Guthrie, seorang mantan murid dari Augustus De Morgan, pada tahun 1852 dan akhirnya dibuktikan oleh matematikawan Amerika Kenneth Appel dan Wolfgang Haken. Pembuktian teorema ini menggunakan komputer dengan waktu yang melebihi 1000 jam. [7]

Gambar 1.2 Contoh Peta Dunia

Masalah pewarnaan graf memiliki banyak aplikasi di dalam bidang lain, misalnya membuat jadwal, alokasi *register* pada *compiler*, penentuan frekuensi untuk radio, pencocokan pola, dan lain – lain. Masalah ini bahkan telah berkembang luas menjadi suatu permainan yang sangat terkenal di kalangan masyarakat luas, yaitu *sudoku*, suatu permainan angka yang menarik. Aplikasi utama yang akan dibahas pada makalah ini adalah masalah penentuan jadwal.

2. Jenis dan Sifat Graf

Graf dapat digolongkan menjadi beberapa jenis graf yang spesifik. Jenis – jenis graf ini akan dijelaskan dengan singkat untuk membatasi graf dan menyamakan pengertian kita mengenai graf yang akan dibahas pada makalah ini. Beberapa jenis graf juga memiliki sifat – sifat penting yang dapat memudahkan kita untuk menyelesaikan masalah pewarnaan. Penulis juga akan memberikan beberapa terminologi mengenai graf yang banyak digunakan pada penulisan makalah ini.

2.1 Terminologi Dasar [6]

Berikut akan diberikan beberapa terminologi dasar penting yang akan banyak digunakan pada makalah ini, yaitu :

A. Bertentangga (*adjacent*)

Dua buah simpul pada graf tak berarah G dikatakan bertetangga bila keduanya terhubung langsung dengan sebuah sisi.

B. Bersisian (*incident*)

Suatu sisi e dikatakan bersisian dengan simpul v_i dan v_j jika e = (v_i, v_j) atau e menghubungkan simpul v_i dan v_j.

C. Derajat (*degree*)

Suatu simpul pada graf tak – berarah dikatakan berderajat n jika jumlah sisi yang bersisian dengan simpul tersebut berjumlah n.
Hal ini dilambangkan dengan d(v). Derajat maksimum simpul yang terdapat pada suatu graf dilambangkan dengan $\Delta(G)$.

D. Lintasan (*path*)

Lintasan adalah barisan berselang – seling simpul – simpul dan sisi – sisi yang berbentuk $v_0,e_1,v_1,e_2,...,v_{n-1},e_n,v_n$ yang menghubungkan simpul awal v_0 dan simpul tujuan v_n. Untuk graf sederhana, biasanya lintasan ditulis sebagai barisan simpul – simpulnya saja, yaitu $v_0,v_1,v_2,v_3,...,v_n$

E. Upagraf (*subgraph*)

Suatu gaf $G_1 = (V_1,E_1)$ merupakan upagraf dari graf G = (V,E) , bila V_1 merupakan himpunan bagian dari V dan E_1 merupakan himpunan bagian dari E.

F. Planar

Suatu graf G merupakan graf palanar bila graf tersebut dapat digambarkan pada bidang datar dengan sisi – sisi yang tidak saling memotong (bersilangan).

G. Upagraf merentang (Spanning Subgraph)

Upagraf G' = (V',E') dari graf G=(V,E) dikatakan upagraf merentang jika V'=V dan G' adalah himpunan bagian dari G.

2.2 Jenis – Jenis Graf [6]

Berikut ini, penulis akan menggolongkan graf berdasarkan jenis – jenisnya dan membatasi jenis graf yang dijelaskan pada makalah ini.

Berdasarkan adanya gelang atau sisi ganda pada suatu graf, maka graf dapat dibagi menjadi 2 macam, yaitu:

A. Graf sederhada (simple graph)
Adalah graf yang tidak mengandung gelang maupun sisi ganda.

B. Graf tak – sederhana (unsimple - graph)
Adalah graf yang mengandung sisi ganda atau gelang, ada 2 macam, yaitu graf ganda (multigraph) dan graf semu (pseudograph). Graf ganda adalah graf yang mengandung sisi ganda, sedangkan graf semu adalah graf yang mengandung gelang (termasuk bila mengandung sisi ganda sekalipun).

Pada makalah ini, yang dibahas adalah graf sederhana. Graf tak – sederhana tidak termasuk karena pada graf yang memiliki gelang, suatu simpul bertetangga dengan simpul itu sendiri sehingga tidak mungkin untuk mewarnainya.

Berdasarkan jumlah simpul pada suatu graf, maka graf dapat dibagi menjadi 2 macam, yaitu:

A. Graf berhingga (limited graph)

Adalah graf yang jumlah simpulnya terbatas.

B. Graf tak-berhingga (unlimited graph)

Adalah graf yang jumlah simpulnya tidak terbatas.

Pada makalah ini, yang dibahas adalah graf berhingga.

Berdasarkan orientasi arah pada sisinya, graf dapat dibagi menjadi 2 macam, yaitu:

A. Graf tak berarah (undirected graph)

Adalah graf yang sisinya tidak mempunyai orientasi arah. Urutan penulisan pasangan simpul pada sisinya tidak diperhatikan. Jadi $(v_i,v_j) = (v_j,v_i)$.

B. Graf berarah (directed graph)

Adalah graf yang sisinya mempunyai orientasi arah. Urutan penulisan pasangan simpul pada sisi graf diperhatikan. Jadi (v_i,v_j) tidak sama dengan (v_j,v_i). Kita biasanya menyebut sisi berarah dengan sebutan busur (arc).

Pada makalah ini, graf yang dibahas adalah graf tak-berarah, aplikasinya pada graf berarah akan sama saja selama graf berarah tersebut tidak memiliki loop, yaitu dengan cara menghilangkan arah pada busurnya sehingga menjadi graf tak berarah.

Berdasarkan keterhubungannya, graf dibagi menjadi dua, yaitu :

A. Graf terhubung (connected graph)

Suatu graf G dikatakan terhubung bila untuk setiap pasang simpul v_i dan v_j di dalam himpunan V terdapat lintasan dari v_i ke v_j.

B. Graf tak terhubung (*disconnected graph*)

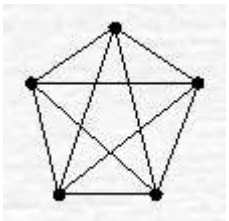
Suatu graf G dikatakan tidak terhubung bila tidak semua simpul pada graf tersebut dihubungkan oleh sebuah simpul. Kasus khusus dari graf jenis ini adalah graf kosong (*null graph*), di mana graf tersebut sama sekali tidak memiliki sisi. Graf kosong dengan jumlah simpul n dilambangkan dengan N_n.

Pada makalah ini, yang dibahas adalah graf terhubung. Aplikasi untuk graf tak terhubung tidak dibahas karena hal itu sama saja dengan menerapkan aplikasi yang sama pada komponen terhubungnya.

Berdasarkan sifat – sifatnya, graf sederhana dapat dibagi menjadi beberapa graf sederhana khusus, yaitu:

A. Graf lengkap (*complete graph*)

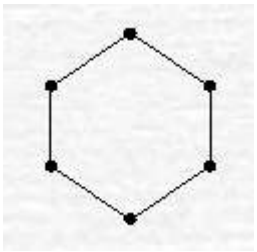
Adalah graf sederhana yang setiap simpulnya mempunyai sisi ke semua simpul lainnya. Graf lengkap dengan n buah simpul dilambangkan dengan K_n. Setiap simpul pada K_n berderajat n-1. Contohnya:



Gambar 2.1 Graf Lengkap K_5

B. Graf Lingkaran

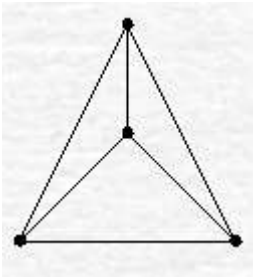
Graf lingkaran adala graf yang setiap simpulnya berderajat 2. Graf lingkaran dengan n buah simpul dinyatakan dengan C_n. Contohnya:



Gambar 2.2 Graf Lingkaran K_5

C. Graf Teratur

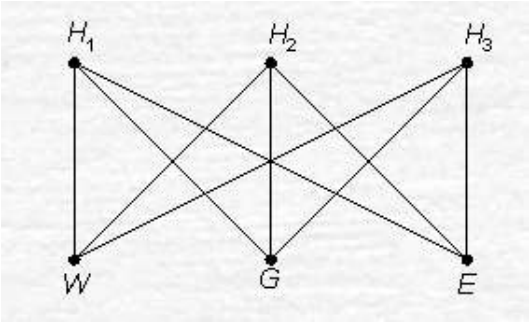
Graf teratur adalah graf yang setiap simpulnya mempunyai derajat yang sama. Graf lengkap K_n adalah graf teratur berderajat n-1 dan graf lingkaran C_n adalah graf teratur berderajat 2 Contohnya:



Gambar 2.3 Graf Teratur berderajat 3

D. Graf Bipartit

Graf G yang himpunan simpulnya dapat dipisah menjadi dua himpunan bagian V_1 dan V_2, sedemikian sehingga setiap sisi pada G menghubungkan sebuah simpul di V_1 ke sebuah simpul di V_2 disebut graf bipartit. Graf bipartite dilambangkan dengan $K_{m,n}$ dengan m adalah jumlah simpul V1 dan n adalah jumlah simpul V2. Contohnya :



Gambar 2.4 Graf Bipartit $K_{3,3}$

2.3 Sifat – Sifat Bilangan Kromatik Graf F

Berikut ini akan diberikan beberapa sifat graf. Sifat – sifat yang diberikan hanyalah sifat – sifat yang menyangkut bilangan kromatik (jumlah warna minimum untuk pewarnaan simpulnya), sesuai dengan tema yang dibahas pada makalah ini.

1. $\chi(G) = 1$ jika dan hanya jika G adalah graf kosong.
2. $\chi(G) \geq 3$ jika dan hanya jika G memiliki subgraph yang merupakan K3 atau C3
3. $\chi(G) \leq \Delta(G)+1$.
4. $\chi(G) \leq \Delta(G)$, kecuali jika G adalah graf lengkap atau graf lingkaran dengan jumlah simpul ganjil.

5. Untuk setiap graf planar, berlaku teorema 4 warna, yaitu $\chi(G) \leq 4$.
6. Bila G' adalah upagraph dari G, maka $\chi(G') \leq \chi(G)$.
7. Graf lengkap K_n memiliki $\chi(G)=n$.
8. Graf Lingkaran C_n memiliki $\chi(G)=2$ bila n genap dan $\chi(G)=3$ bila n ganjil.
9. Graf Teratur berderajat n selalu memiliki $\chi(G) \leq n+1$ sesuai sifat no 3 di atas.
10. Graf Bipartit selalu bisa diwarnai dengan 2 warna.
11. Graf yang berupa pohon selalu dapat diwarnai dengan 2 warna.

3. Aplikasi *Graph Colouring* untuk *Scheduling*

Sebagai pembukaan dari bagian ini, penulis akan memberikan sebuah contoh kasus yang menggambarkan manfaat metode pewarnaan graph dalam membuat jadwal.

Setiap semester, petugas kantor akademik MIT harus menentukan jadwal ujian. Hal ini tidaklah mudah, karena beberapa mahasiswa/i mengambil beberapa mata kuliah yang bisa bertabrakan jadwal ujiannya. Petugas tersebut ingin menghindari semua jadwal yang bertabrakan. Tentu saja, ia bisa membuat sebuah jadwal dengan setiap mata kuliah memiliki waktu ujian yang berlainan, tetapi hal ini mengakibatkan masa ujian tersebut akan sangat lama sekali dan bahkan mungkin lebih dari satu semester. Petugas tersebut tentu saja ingin membuat masa ujian sesingkat mungkin sehingga memudahkan semua pihak, bagaimana cara ia melakukannya? [3]

Di sinilah masalah pewarnaan graf memegang peranan yang penting. Kita dapat memodelkan masalah seperti di atas ke dalam model grafnya. Setiap mata kuliah dimodelkan dengan sebuah simpul pada graf tersebut. Kemudian setiap mata kuliah yang diikuti oleh setidaknya 1 orang yang sama akan memiliki hubungan yang direpresentasikan dalam bentuk sisi pada 2 simpul mata kuliah itu. Waktu atau jadwal sebuah mata kuliah berlangsung adalah warna yang diberikan pada simpul mata kuliah itu. Masalah yang harus dipecahkan berikutnya adalah bagaimana kita memberikan warna minimum kepada setiap simpul sehingga tidak ada simpul – simpul yang bertetangga memiliki warna yang sama. Hal ini merupakan masalah pewarnaan simpul graf yang telah kita singgung sebelumnya.

Contoh lain yang dapat dimodelkan dalam pewarnaan graf misalnya adalah masalah update software secara online. Contohnya pada perusahaan Akamai, versi baru dari sebuah software terus diluncurkan setiap beberapa hari ke 20.000 server yang dimilikinya. Tentu saja perusahaan tersebut tidak bisa mematikan seluruh servernya untuk mengupdatenya, ataupun mengupgrade satu per satu server tersebut karena hal tersebut akan memakan waktu yang terlalu lama. Dibutuhkan suatu cara yang efisien untuk menentukan jadwal upgrade setiap server.

Masih banyak contoh lain dari masalah pewarnaan graf, misalnya penentuan alokasi variabel pada register komputer, masalah pewarnaan peta, pemberian frekuensi radio kepada setiap stasiun radio, penentuan jadwal operasi kereta api, dan lain – lain.

Untuk masalah penentuan jadwal, masalah tersebut biasanya berhubungan dengan beberapa pekerjaan yang berhubungan seperti menggunakan sumber daya yang sama sehingga tidak bisa dilakukan pada waktu yang bersamaan.

Kita dapat menyelesaikan masalah ini dalam 3 langkah, yaitu:

1. Menggambar simpul – simpul graf

Simpul – simpul graf yang digambarkan haruslah mewakili pekerjaan yang akan dilakukan.

2. Menggambar sisi – sisi pada graf

Kita menggambarkan sisi – sisi pada setiap pasang simpul yang menggunakan sumber daya yang sama. Yang artinya kedua pekerjaan tersebut tidak bisa dilakukan pada waktu yang sama.

3. Mewarnai graf

Langkah terakhir yang harus kita lakukan adalah mewarnai simpul – simpul pada graf tersebut dengan warna yang minimum sehingga tidak ada simpul – simpul yang bertetangga memiliki warna yang sama.

Berikut ini diberikan sebuah contoh penyelesaian kasus mengenai penentuan jadwal.

Seorang penjaga kantor *baby sitter* mendapat titipan 7 orang anak, A,B,C,D,E,F,dan G.

Penjaga tersebut harus menentukan loker yang diberikan kepada setiap orangtua anak – anak tersebut sebagai tempat menyimpan peralatan mengasuhnya. Anak – anak tersebut tidak setiap waktu berada di kantor baby sitter tersebut, mereka datang dan pergi tergantung dari pesanan orang tua mereka. Karena loker di kantor tersebut terbatas dan harus digunakan sehemat mungkin, penjaga tersebut diminta untuk menentukan jadwal pemberian loker kepada setiap orang tua. Jadwal penitipan untuk setiap anak diberikan pada tabel berikut

Tabel 3.1 Jadwal Penitipan Bayi

.	A	B	C	D	E	F	G
7:00	*	-	-	*	*	-	-
8:00	*	*	*	-	-	-	-
9:00	*	-	*	*	-	*	-
10:00	*	-	*	-	-	*	*
11:00	*	-	-	-	-	*	*
12:00	*	-	-	-	*	-	-

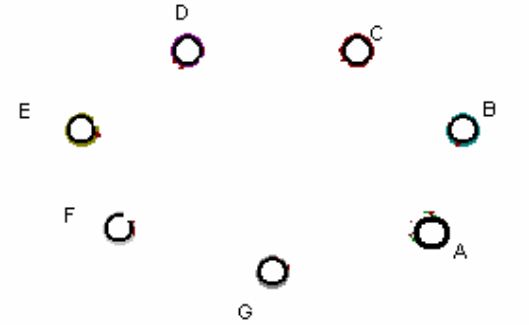
Tanda bintang (*) menggambarkan bahwa anak tersebut berada pada kantor pada waktu yang diberikan, sedangkan tanda strip (-) berarti bahwa anak tersebut tidak sedang dititipkan.

Tentukan bagaimana petugas tersebut harus menentukan loker mana yang diberikan kepada setiap orangtua. [5]

Kita akan menyelesaikan masalah di atas dalam beberapa langkah:

1. Menggambar Simpul graf

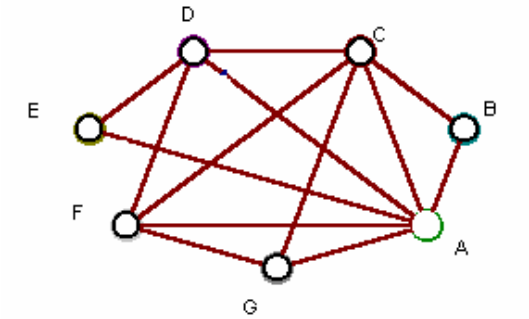
Graf yang digambarkan di sini harus menggambarkan kondisi di atas, pertama kita membuat tujuh buah simpul



Gambar 3.1 Simpul Yang Merepresentasikan Anak

2. Menggambar sisi graf

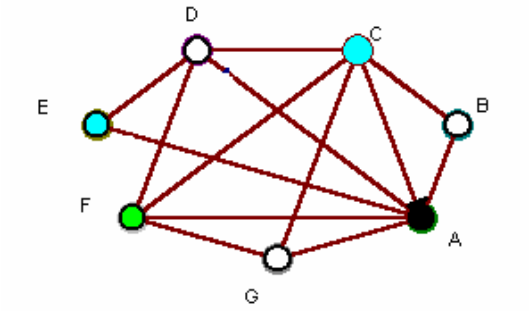
Kita menggambarkan sisi - sisi pada setiap pasang simpul bila anak – anak yang dilambangkan pada kedua simpul yang dihubungkan oleh sisi tersebut menggunakan petak (*slot*) waktu yang sama.



Gambar 3.2 Graf Yang Merepresentasikan Hubungan Jadwal Tiap Anak

3. Mewarnai Graf tersebut

Graf yang telah diwarnai akan terlihat sebagai berikut :



Gambar 3.3 Graf Yang Telah Diwarnai

Warna minimum yang bisa digunakan untuk mewarnai graf ini adalah 4, karenanya bilangan kromatik graf ini adalah 4. Bila yang tersedia hanya tiga buah warna, maka graf ini tidak akan bisa diwarnai sesuai dengan ketentuan. Dalam

hal ini, loker minimum yang tersedia harus berjumlah 4. C dan E diberikan loker 1. B, D, dan G deiberikan loker 2. F diberikan loker 3. A diberikan loker 4. Pemberian loker yang sama untuk 2 orang anak atau lebih dapat dilakukan karena mereka tidak pernah berada pada tempat yang sama dalam waktu yang sama.

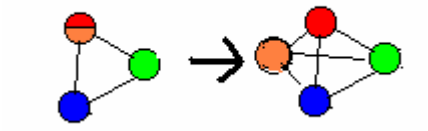
Pada contoh di atas, penulis tidak menjelaskan algoritma untuk mewarnai graf tersebut, hal ini akan dijelaskan pada bagian selanjutnya dalam makalah ini.

Selain masalah penjadwalan seperti di atas, biasanya terdapat variasi – variasi lain yang dapat diberikan. Berikut adalah beberapa variasi masalah penjadwalan. [4]

- Pewarnaan Banyak (*Multicoloring*)

Yaitu pewarnaan simpul graf di mana sebuah simpul terkadang diharuskan memiliki jumlah warna yang lebih dari 1. Contoh kasus yang menggambarkan keadaan ini adalah kasus membuat jadwal di mana tidak setiap pekerjaan memerlukan waktu yang sama. Sebuah pekerjaan mungkin membutuhkan jumlah waktu yang lebih banyak, misalnya 2 kali dibandingkan jumlah waktu yang dibutuhkan pekerjaan lain. Untuk memodelkannya, kita memberi 2 buah warna kepada pekerjaan ini.

Cara untuk memecahkan masalah pewarnaan banyak seperti ini sebenarnya sama saja dengan memecahkan pewarnaan biasa. Kita cukup menambahkan simpul – simpul baru yang memiliki tetangga – tetangga yang sama untuk sebuah simpul yang membutuhkan warna lebih.

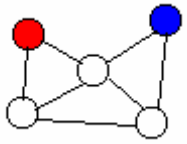


Gambar 3.4 Manipulasi Muticoloring

- Pewarnaan Awal (*Precoloring*)

Yaitu suatu jenis pewarnaan simpul di mana warna – warna simpul tertentu telah ditentukan dari awal. Kasus seperti ini bisa terjadi karena kita tidak selalu memiliki kontrol yang penuh akan semua pekerjaan yang ingin dilakukan. Beberapa pekerjaan mungkin memiliki waktunya sendiri yang tidak bisa diubah, misalnya pekerjaan *maintenance* yang biasanya memiliki waktu yang telah tetap.

Untuk memecahkan masalah pewarnaan ini, kita bisa menggunakan algoritma *Saturated Degree Ordering* atau *Incident Degree Oredering* yang akan dibahas pada bagian selanjutnya.

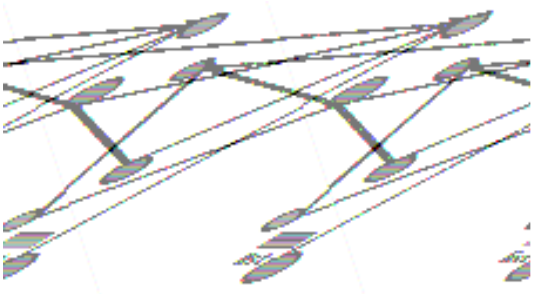


Gambar 3.5 *Precoloring Graph*

- Pewarnaan Daftar (*List Coloring*)

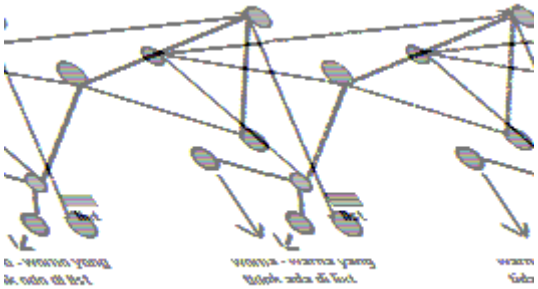
Merupakan masalah pewarnaan simpul graf di mana simpul – simpul tertentu memiliki *list of available colour,* yaitu daftar – daftar warna yang dapat diberikan kepada simpul tersebut. Tidak semua warna bisa diberikan kepada sebuah simpul, kita harus memilihnya dari daftar. Contoh kasus seperti ini terjadi misalnya ketika suatu pekerjaan hanya bisa dilakukan pada saat – saat tertentu, atau hanya bisa dilakukan oleh mesin – mesin tertentu.

Cara untuk memecahkan masalah pewarnaan seperti ini bisa dilakukan dengan memberikan simpul – simpul tetangga kepada simpul – simpul khusus yang memiliki daftar warna, simpul – simpul tetangga tersebut memiliki warna yang tidak ada di daftarnya. Dengan adanya simpul – simpul tetangga ini, simpul tersebut akan ”dipaksa” untuk hanya menggunakan warna – warna tertentu saja, yaitu warna – warna yang ada pada daftarnya. Contohnya dapat dilihat pada gambar berikut.



Gambar 3.6 *List Coloring Graph*

Dapat digambarkan dengan graf berikut



Gambar 3.7 Manipulasi *List Coloring Graph*

Terdapat Algoritma lain untuk menyelesaikan masalah pewarnaan seperti ini, tetapi hal itu tidak dibahas pada makalah ini.

- Pewarnaan Minimum (*Minimal Sum Coloring*)

Masalah pewarnaan ini adalah masalah pewarnaan yang telah dibahas di atas, yaitu pewarnaan simpul dengan jumlah warna yang minimum.

4. Algoritma Mewarnai Graf

Untuk graf – graf dengan sifat tertentu, kita bisa dengan muda menyebutkan bilangan kromatiknya dan mewarnainya, misalnya graf lengkap dan graf lingkaran. Tetapi umumnya graf yang kita hadapi adalah graf yang tidak memiliki keteraturan seperti itu. Karena itu, dibutuhkan algoritma yang dapat membantu kita menyelesaikan masalah pewarnaan graf.

Algoritma terbaik untuk menemukan bilangan kromatik dari suatu graf yang dikenal sampai saat ini memiliki kompleksitas waktu eksponensial untuk kasus terburuknya. Masalah ini termasuk ke dalam kelas NP Lengkap (NP Complete). Belum ada yang mampu memberikan solusi dengan kompleksitas waktu polinomial untuk kasus terburuknya, tapi juga tidak ada bukti bahwa algoritma seperti itu tidak ada. Dikatakan bahwa apabila salah satu masalah dalam kelas NP lengkap bisa diselesaikan dalam waktu polinomial, maka semua masalah di dalam kelas tersebut juga bisa diselesaikan dalam waktu polinomial. Sebaliknya bila ada bukti bahwa algoritma dengan kompleksitas algoritma polinomial tidak ada untuk salah satu masalah pada kelas NP lengkap, maka semua masalah pada kelas NP lengkap juga tidak bisa diselesaikan dalam kompleksitas waktu polinomial.

Beberapa algoritma yang telah banyak dikenal adalah: [1]

- *First Fit* (FF)

Algoritma ini adalah algoritma yang termudah dan tercepat. Prinsipnya adalah mewarnai setiap simpul graf dengan warna yang tidak akan diubah lagi. Meskipun algoritma ini sangat mudah untuk diimplementasikan dan juga sangat cepat. Namun, algoritma ini memiliki probablitas besar untuk menghasilkan jumlah warna yang melebihi bilangan kromatiknya.

Kompleksitas waktu asimtotik dari algoritma ini adalah O(n)

- *Largest Degree Odering* (LDO)

Algoritma ini merupakan algoritma yang prinsipnya berdasarkan pada nilai derajat dari setiap simpul. Simpul yang memiliki derajat yang lebih tinggi diwarnai lebih dulu. Algoritma ini memberikan hasil yang lebih baik daripada algoritma *first fit.*

Kompleksitas waktu asimtotik dari algoritma ini adalah $O(n^2)$.

- *Saturated Degree Ordering* (SDO)

Algotima ini berprinsipkan pada jumlah warna berlainan yang ada pada tetangga – tetangga dari sebuah simpul. Simpul yang bertetanggaan dengan simpul – simpul yang memiliki lebih banyak aneka warna akan diwarnai lebih dulu. Algoritma ini memberikan hasil yang lebih baik daripada algoritma LDO.

Kompleksitas waktu asimtotik dari algoritma ini adalah $O(n^3)$.

- *Incident Degree Ordering* (IDO)

Algoritma ini berprinsipkan pada jumlah simpul tetangga yang telah diwarnai dari suatu simpul. Simpul yang lebih banyak bertetanggaan dengan simpul yang telah diwarnai akan diwarnai lebih dulu. Algoritma ini merupakan modifikasi dari algoritma SDO. Algoritma ini dapat dieksekusi dalam waktu yang lebih cepat, tetapi hasilnya tidak sebaik algoritma SDO.

Kompleksitas waktu asimtotik dari algoritma ini adalah $O(n^2)$

Berikut ini adalah tabel yang menggambarkan jumlah warna yang dihasilkan dari setiap algortima. Kepadatan adalah perbandingan dari jumlah sisi (*vertex*) yang ada terhadap jumlah sisi dari graf lengkapnya.

Tabel 4.1 Perbandingan Efektifitas Algoritma

Jumlah Simpul	Kepadatan	FF	LDO	IDO	SDO
200	25%	20	18	18	17
200	50%	36	34	34	32
200	75%	58	55	56	53
1000	25%	64	62	63	58
1000	50%	127	123	126	116
1000	75%	217	212	214	204

Pada bagian ini, penulis akan menjelaskan mengenai algoritma yang dikembangkannya sendiri, algoritma ini adalah modifikasi serta gabungan dari algoritma *spanning tree* [8], *largest degree ordering* dan *saturated degree ordering*.

4.1. Algoritma Gabungan

Pada Algoritma ini, beberapa langkah yang harus dilakukan adalah:

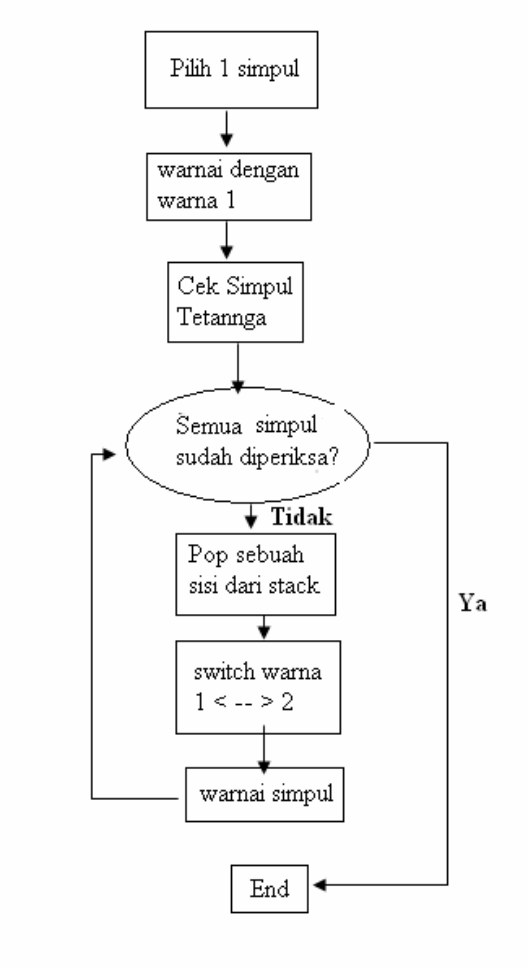
1. Membangun *spanning tree*

Untuk membangun *spanning tree* dari grafnya, pertama – tama kita memilih sebuah simpul pada graf, kemudian kita mewarnainya dengan warna 1 (W1). Berikutnya, kita perhatikan semua simpul yang bertetangga dengan simpul tersebut, ada 3 kasus:

a) Bila simpul tersebut sudah diwarnai, maka sisi yang menghubungkannya kita tandai sebagai sisi yang bermasalah dengan cara memasukkannya ke tabel sisi bermasalah. Bila sisi tersebut sudah ada dalam *stack* sisi yang belum diperiksa, maka sisi tersebut dihapus dari *stack*.
b) Bila simpul tersebut belum diwarnai, maka sisi yang menghubungkannya kita masukkan (*push*) ke dalam tumpukan sisi yang belum diperiksa (*unchecked edges stack*).

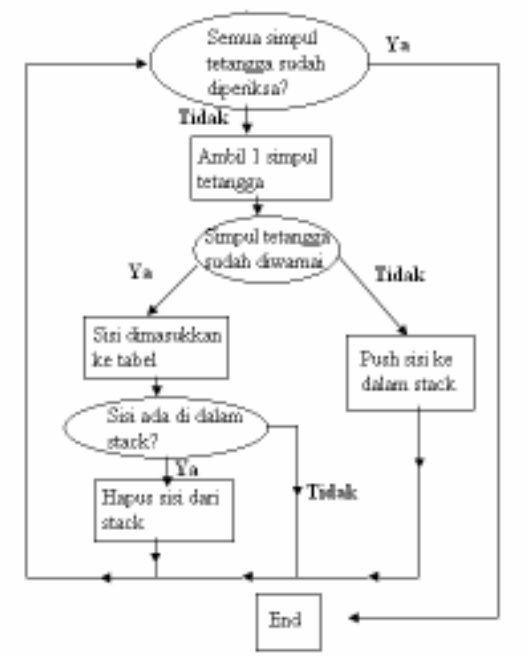
Setelah semua simpul tetangga diperiksa, kita ambil (*pop*) sebuah sisi dari *stack* sisi yang belum diperiksa. Kemudian kita warnai simpul tujuan yang dihubungkan oleh sisi itu dengan warna 2 (W2). Berikutnya, kita periksa lagi simpul – simpul yang bertetangga dengan simpul yang sedang kita warnai sekarang ini, kecuali simpul sebelumnya, seperti di atas. Kita pop lagi sebuah sisi dari stack sisi yang belum diperiksa, tetapi kali ini simpul tetangga tersebut diberi warna W1

Hal ini dilakukan terus menerus sehingga semua simpul telah diwarnai. Bagan *flowchart* dari langkah ini bias dilihat pada gambar berikut



Gambar 4.1 Sketsa *Flowchart* langkah 1

Pada akhir langkah ini, kita akan mendapati bahwa semua simpul tersebut diwarnai oleh 2 warna



Gambar 4.2 Sketsa *Flowchart* Pengecekan Simpul Tetangga

2. Menyelesaikan sisi – sisi yang bermasalah.

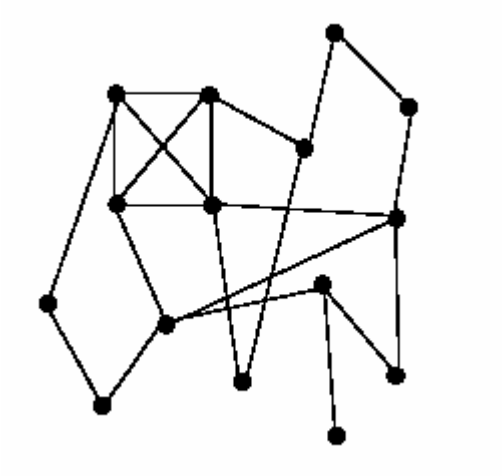
Pada langkah ini, kita akan memeriksa tabel sisi yang bermasalah dan menyelesaikannya. Kita ambil setiap sisi mulai dari indeks 1, 2, 3, .., n.
Untuk setiap sisi, yang harus kita lakukan adalah memeriksa apakah kedua simpul yang dihubungkan oleh sisi tersebut memiliki warna yang sama.

a) Bila sepasang simpul tersebut memiliki warna yang berbeda, maka sisi tersebut aman.
b) Bila sepasang simpul tersebut memiliki warna yang sama, maka salah satu dari simpul tersebut harus diubah warnanya. Penentuan simpul yang harus diubah warnanya ditentukan dari hal berikut:
   - Pilihalah simpul yang tidak perlu menambah warna baru bila simpul tersebut diubah warnanya, tetapi hanya menggunakan warna – warna yang sudah ada.
   - Bila poin pertama di atas sama- sama terpenuhi oleh kedua simpul atau sama-sama tidak terpenuhi oleh kedua simpul, maka pilihlah simpul yang memiliki lebih banyak sisi yang bermasalah. (modifikasi dari algoritma *Largest Degree Ordering*).
   - Bila jumlah sisi yang bermasalah sama untuk kedua simpul, maka pililah simpul yang memiliki tetangga dengan jumlah warna berbeda yang lebih banyak. (modifikasi dari algoritma *Saturated Degree Ordering*).
   - Bila kedua simpul berderajat sama, maka simpul yang dipilih boleh bebas.

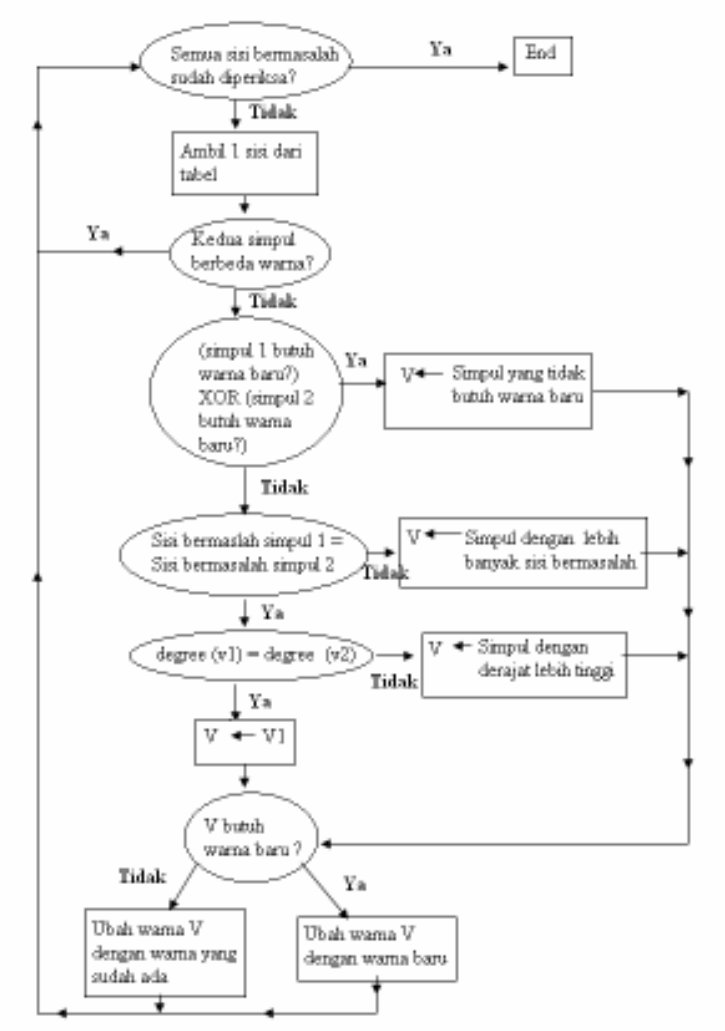
   Simpul tersebut kemudian diubah warnanya, dengan ketentuan sebagai berikut:
   - Pilihlah warna yang sudah ada, dan tentu saja tidak melanggar ketentuan bahwa warna tersebut tidak sama dengan simpul – simpul tetangganya yang lain.
   - Bila tidak ada warna yang seperti itu, maka kita harus menggunakan warna yang baru.

Sketsa *flowchart* dari langkah 2 ini diberikan di bawah.

Algoritma tersebut akan lebih jelas apabila kita praktekkan secara langsung. Contohnya akan kita praktekkan pada graf berikut:

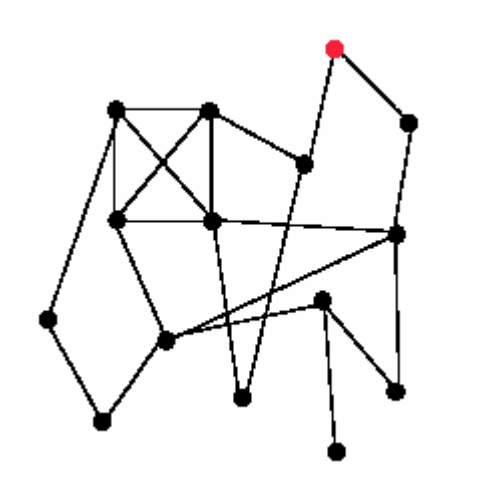


Gambar 4.3 Contoh Graph



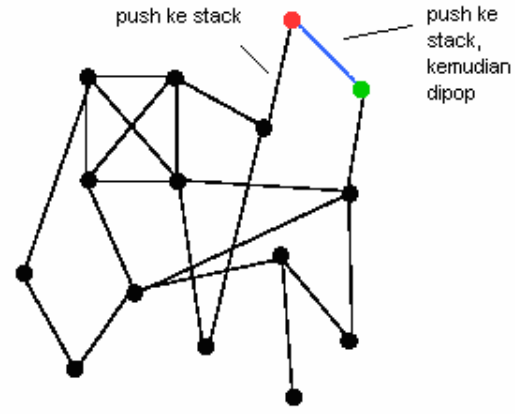
Gambar 4.4 Sketsa *Flowchart* Langkah 2

1. Pertama kita pilih sebuah simpul secara bebas dan kita beri warna merah



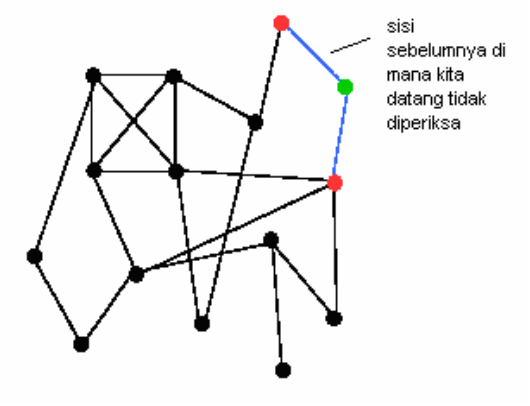
Gambar 4.5 Langkah 1

2. Kedua sisi yang bersisian denganya kita masukkan ke stack, kemudian kita pop sebuah sisi dari stack dan menelusurinya, simpul yang bersangkutan kita beri warna hijau.



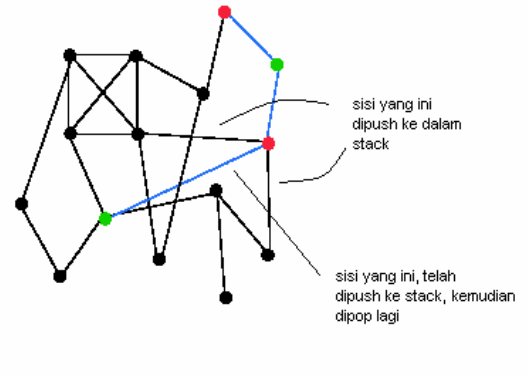
Gambar 4.6 Langkah 2

3.Hanya satu sisi yang harus diperiksa, warna yang diberikan menjadi merah lagi (berselang - seling)



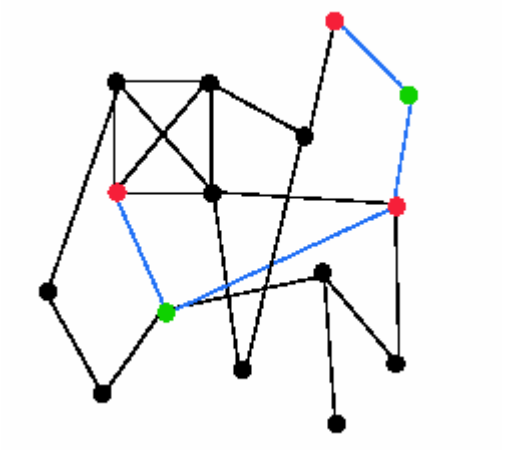
Gambar 4.7 Langkah 3

4. Ada 3 sisi yang harus diperiksa, ketiganya dipush ke stack, kemudian kita pop yang terakhir dan ditelusuri lagi.



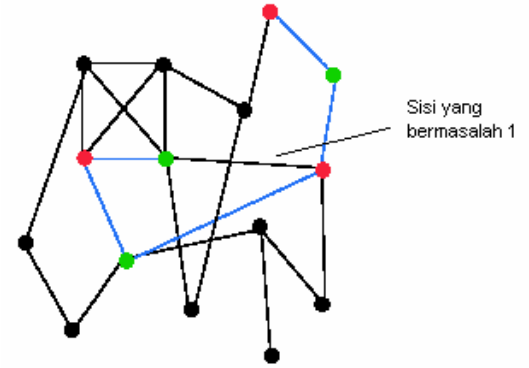
Gambar 4.8 Langkah 4

5. Ditelusuri lagi



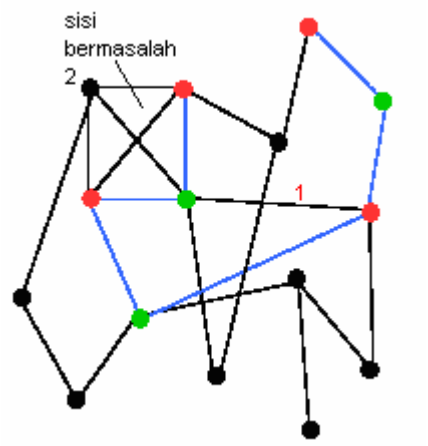
Gambar 4.9 Langkah 5

6. Ditemukan sebuah sisi yang bermasalah, sisi tersebut dimasukkan ke tabel. Karena sisi tersebut sebelumnya telah dimasukkan ke stack sisi yang belum diperiksa, maka sisi tersebut dihapus dari stack.



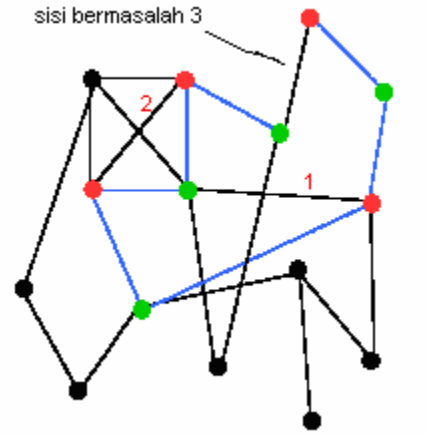
Gambar 4.10 Langkah 6

7. Ditemukan sisi bermasalah 2



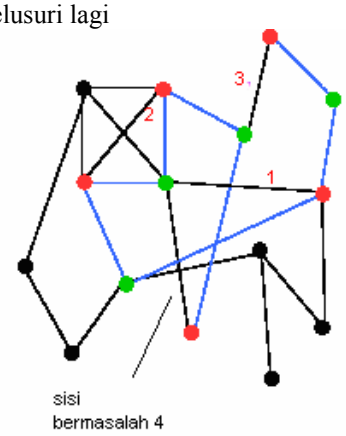
Gambar 4.11 Langkah 7

8. Ditelusuri lagi



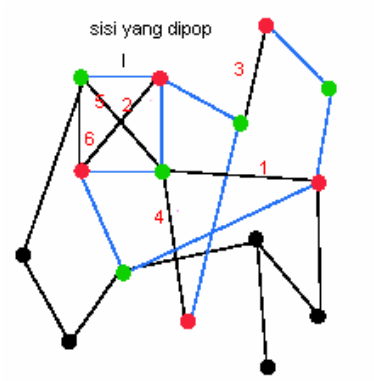
Gambar 4.12 Langkah 8

9. Ditelusuri lagi



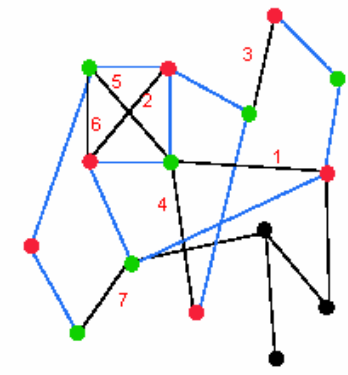
Gambar 4.13 Langkah 9

10. Pada bagian sebelumnya tidak ada masukan baru pada stack, sehingga ketika dipop sebuah sisi, maka sisi yang dipop itu adalah sisi terakhir yang dipush, yaitu sisi yang ditandai pada gambar, ditemukan 2 buah sisi bermasalah



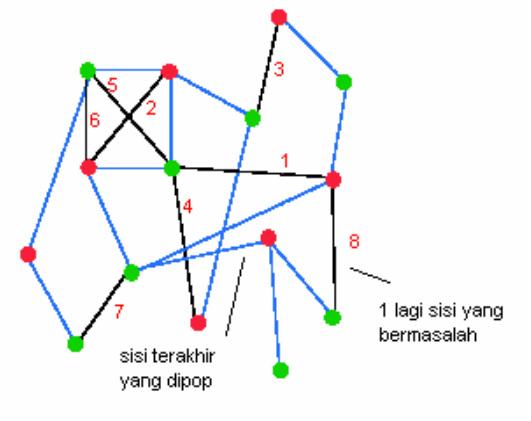
Gambar 4.14 Langkah 10

11. Maju 2 kali dan didapati sebuah sisi bermasalah



Gambar 4.15 Langkah 11

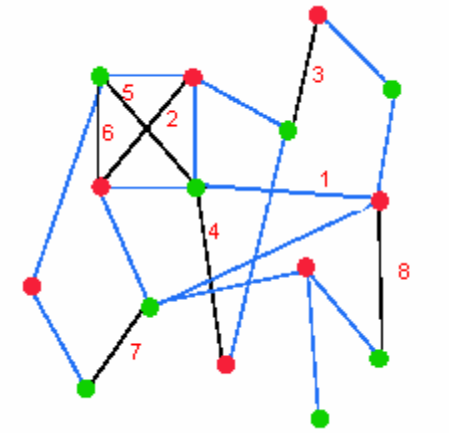
12. Karena langkah terakhir tidak memasukkan sisi baru pada stack, maka sisi yang dipop adalah sisi yang ditandai pada gambar. Maju sampai semua selesai dikerjakan. Pada tahap ini, stack menjadi kosong.



Gambar 4.16 Langkah 12

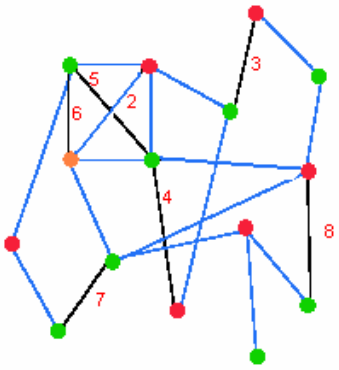
Sampai tahap ini, kita telah menyelesaikan proses membangun spanning tree, berikutnya kita akan menyelsaikan sisi – sisi yang bermasalah.

13. Pada sisi 1 tidak ada masalah karena pasangan simpulnya memiliki warna yang berbeda



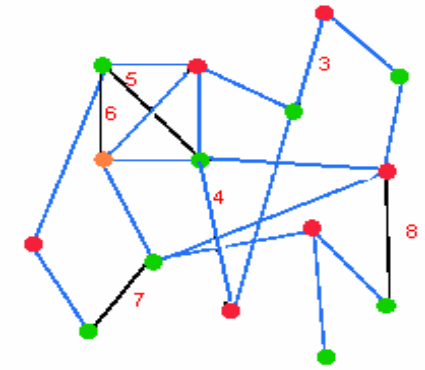
Gambar 4.17 Langkah 13

14. Pasangan simpul pada sisi 2 memiliki warna yang sama, karena itu kita mengubah warna salah satu simpulnya. Dalam kasus ini, kedua sisi sama – sama mengharuskan pengubahan warna dengan warna yang belum ada, karena itu kita mengubah warna simpul yang memiliki lebih banyak sisi bermasalah, warnanya kita ubah menjadi orange.



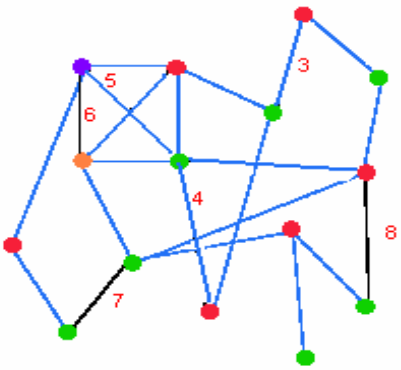
Gambar 4.18 Langkah 14

15. Pasangan simpul pada sisi 3 dan sisi 4 tidak ada masalah.



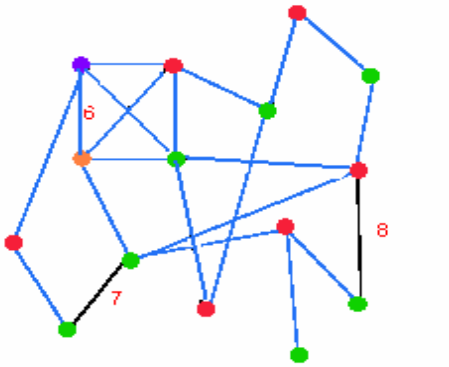
Gambar 4.19 Langkah 15

16. Pasangan simpul pada sisi 5 memiliki warna yang sama, karena itu kita mengubah warna salah satu simpulnya. Dalam kasus ini, simpul yang diubah warnanya adalah simpul yang memiliki sisi bermasalah lebih banyak. Karena warna merah dan orange tidak diperbolehkan, maka kita memberikan warna yang baru, yaitu warna ungu pada simpul tersebut.



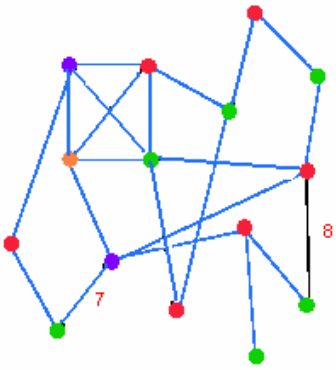
Gambar 4.20 Langkah 17

17. Pasangan simpul pada sisi 6 tidak ada masalah



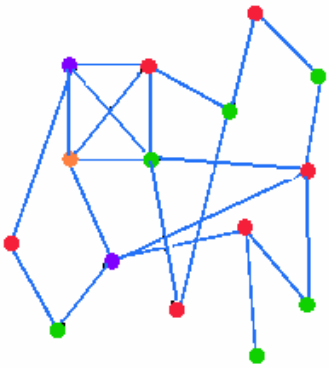
Gambar 4.21 Langkah 16

18. Pasangan simpul pada sisi 7 memiliki warna yang sama, karena kedua simpul sama – sama tidak mengharuskan penambahan warna baru dan memiliki sebuah sisi bermasalah, kita memilih simpul yang bertetangga dengan jumlah warna lebih banyak untuk diubah warnanya.



Gambar 4.22 Langkah 18

19. Tidak ada masalah pada pasangan simpul yang dihubungkan oleh sisi 8

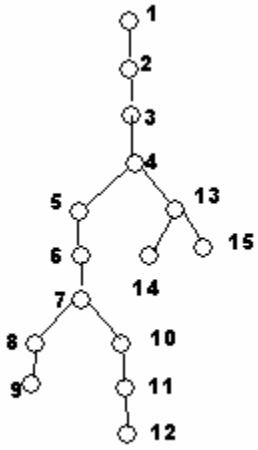


Gambar 4.23 Langkah 19

Selesailah proses pewarnaan graf ini, graf dalam contoh di atas memiliki bilangan kromatik 4, hal ini tidak mengherankan karena graf tersebut mengandung upagraf K_4

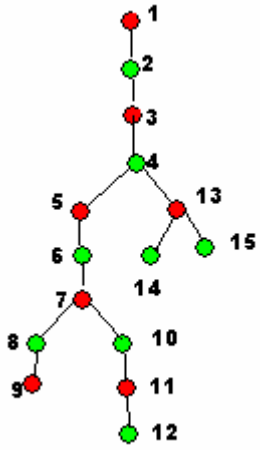
Analisis mengenai algoritma modifikasi ini dapat dijelaskan sebagai berikut:

- Ketika membangun *spanning tree*, kita menyederhanakan graf.



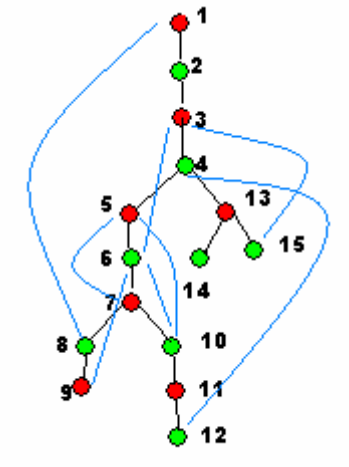
Gambar 4.24 Analisa Algoritma 1

- Sesuai sifatnya, pohon selalu dapat diwarnai dengan 2 warna.



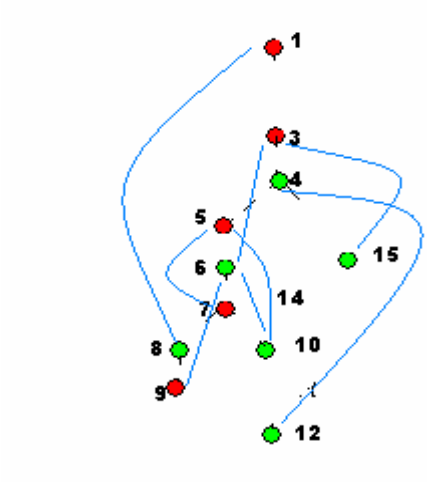
Gambar 4.25 Analisa Algoritma 2

- Namun, selain pohon ini, terdapat juga sisi – sisi yang masih bermasalah



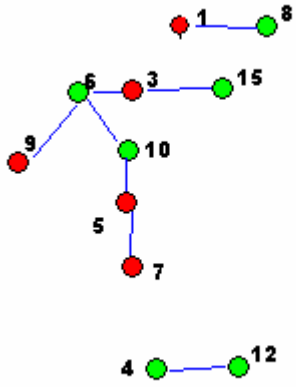
Gambar 4.26 Analisa Algoritma 3

- Atau bila simpul – simpul yang tidak berhubungan dengan sisi – sisi ini, dihilangkan, maka didapat upagraf sebagai berikut



Gambar 4.27 Analisa Algoritma 4

- Dan bila disederhanakan, upagraf menjadi seperti berikut



Gambar 4.28 Analisa Algoritma 5

Masalah berikutnya adalah menganalisis sisi – sisi yang bermasalah dan memprosesnya. Hal ini sebenarnya adalah masalah mewarnai upagraf yang lebih sederhana tersebut, tentu saja dengan tambahan syarat bahwa tetangga – tetangga sebenarnya dari graf tersebut tetap harus diperhatikan.

Karena itulah, algoritma ini menggabungkan metode dari algoritma LDO dan SDO. Algoritma LDO digunakan untuk memilih simpul yang diubah warnanya, yaitu ketika sebuah simpul dari upagraf memiliki lebih banyak sisi – sisi bermasalah. Algoritma SDO digunakan untuk memaksimalkan kinerja dari proses ini, yaitu ketika jumlah sisi – sisi yang bermasalah sama, maka algortima ini diterapkan dengan memilih simpul yang memiliki variasi warna tetangga lebih banyak.

Sekilas mungkin terlihat bahwa algoritma modifikasi ini menjadi lebih rumit, tetapi berdasarkan penilitian, algoritma – algoritma yang digabung akan memiliki efisiensi yang lebih tinggi. Contohnya adalah gabungan algoritma LDO dan SDO, algoritma ini berdasarkan percobaan mampu memberikan jumlah warna yang lebih efisien daripada masing – masing algoritma LDO atau SDO yang berdiri sendiri – sendiri.

5. Kesimpulan

Beberapa kesimpulan yang dapat ditarik dari penulisan makalah ini adalah:

a. Masalah pewarnaan graf merupakan masalah yang banyak digunakan untuk memodelkan masalah di berbagai bidang, salah satunya adalah pembuatan jadwal atau *scheduling*.
b. Beberapa jenis graf yang memiliki keteraturan dapat ditentukan bilangan kromatiknya secara langsung dengan menggunakan sifat – sifatnya.
c. Masalah penjadwalan atau *scheduling* memiliki beberapa variasi yang dapat dimodelkan ke dalam masalah pewarnaan yang berbeda pula.
d. Terdapat beberapa algoritma yang dapat digunakan untuk menyelesaikan masalah pewarnaan graf. Algoritma yang cepat biasanya hasilnya tidak efisien (memberikan warna yang lebih banyak), sedangkan algoritma yang efisien biasanya membutuhkan waktu eksekusi yang lebih lama.
e. Beberapa algoritma dapat dimodifikasi dan digabung untuk membentuk algoritma baru yang lebih efisien.

DAFTAR PUSTAKA

[1] Al-Omari, Hussein & Khair Eddin Sabri, New Graph Coloring Algorithms, www.scipub.org/fulltext/jms2/jms224739-741.pdf, 2006. Tanggal Akses : 2 January 2007, pukul 12.30 WIB.

[2] Klotz, Walter, Graph Coloring Algorithms, www.math.tu-clausthal.de/Arbeitsgruppen/Diskrete-Optimierung/publications/2002/gca.ps, 2000. Tanggal Akses : 29 Desember 2006, pukul 16.00 WIB.

[3] Leighton, Tom & Ronitt Rubinfeld, Graph Teory, http://theory.lcs.mit.edu/classes/6.042/fall06/lec6.pdf, 2006. Tanggal Akses : 29 Desember 2006, pukul 15.00 WIB,

[4] Marx, Daniel, Graph Coloring Problems And Their Applications In Scheduling, http://citeseer.ist.psu.edu/672702.html, 2006. Tanggal Akses : 29 Desember 2006, pukul 14.30 WIB.

[5] Mawata, Christopher, Graph Teory Lesson, http://oneweb.utc.edu/~Christopher-Mawata/petersen/answers/les8Ans.html, 2006. Tanggal Akses : 29 Desember 2006, pukul 14.15 WIB.

[6] Munir, Rinaldi, Diktat Kuliah IF 2153, Matematika Diskrit, Edisi Keempat, Program Studi Teknik Informatika, STEI, ITB, 2006.

[7] Rosen, Kenneth H., Descrete Mathematics and Its Applications, 4 th, McGraw-Hill Internasional, 1999.
[8] The Computer Action Team, Graph Colouring Algorithm, http://web.cecs.pdx.edu/~postj/graph/graph.html, 2005. Tanggal Akses : 29 Desember 2006, pukul 15.30 WIB.
[9] Wikipedia, Wikipedia – Free Encyclopedia, www.wikipedia.com, 2006. Tanggal akses : 27 Desember 2006, pukul 15.00 WIB.