BAB II
TEORI PENUNJANG

2.1 GRAPH

Definisi 2.1.1.:
Suatu graph $G = \{V,E\}$ didefinisikan sebagai himpunan tidak kosong dan berhingga titik – titik $V$ beserta himpunan garis – garis $E$, dan $E$ boleh merupakan himpunan kosong.

Definisi 2.1.2.: 
Walk pada suatu graph adalah suatu barisan titik dan garis yang bergantian dengan setiap garis incident dengan dua titik terdekat pada barisan itu, dimulai dan diakhiri di suatu titik.
Path adalah suatu walk yang titik dan garisnya tidak berulang.
Cycle adalah suatu path tertutup yang diawali dan diakhiri oleh titik yang sama.

Definisi 2.1.3.:
Subgraph dari $G$ adalah graph yang semua garis dan titiknya merupakan anggota himpunan titik dan garis pada $G$, dan spanning subgraph adalah subgraph yang berisi semua titik anggota $G$. 
Definisi 2.1.4. :
Graph terhubung adalah suatu graph yang semua titiknya dihubungkan oleh sebuah path.

Definisi 2.1.5. :
Graph bipartite adalah suatu graph yang mempunyai himpunan titik – titik V yang dipisahkan menjadi dua himpunan, yaitu $V_A$ dan $V_B$, sedemikian sehingga jika ada garis e dalam G maka e pasti menghubungkan titik di $V_A$ dengan titik di $V_B$.

Definisi 2.1.6. :
Pohon adalah suatu graph terhubung yang tidak mempunyai cycle.

2.2 MATCHING MAKSIMUM PADA GRAPH BIPARTITE

Definisi 2.2.1. :
Matching adalah suatu subgraph yang dihasilkan oleh himpunan pasangan garis yang tidak adjacent, dan tiap titik mempunyai paling banyak satu degree. Matching maksimum adalah suatu matching yang mempunyai cardinality maksimal dalam graph G.
Definisi 2.2.2. :
Misalkan M adalah sebuah matching dalam graph G. Garis matched adalah garis G yang terdapat dalam M, sedangkan garis unmatched adalah garis G yang tidak terdapat dalam M. Titik matched adalah titik pada G yang incident dengan garis pada M, sedangkan jika tidak incident disebut titik unmatched. Titik bebas adalah titik yang tidak berada dalam matching.

Definisi 2.2.3. :
Path alternating dari G adalah path yang mempunyai garis mathed dan garis unmatched dalam M. Path perluasan adalah path alternating yang diawali dan diakhiri oleh titik bebas


Teorema 2.2.1. :
Matching M dalam G adalah matching maksimum jika dan hanya jika tidak terdapat path perluasan L yang berkenaan pada M dalam G.
Bukti :

⇒ Jika M adalah matching maksimum dalam G maka tidak terdapat path perluasan L yang berkenaan pada M dalam G.


⇐ Jika tidak terdapat path perluasan L yang berkenaan pada M dalam G maka M adalah matching maksimum. Misalkan M_1 adalah matching dalam G dan tidak terdapat path perluasan L yang berkenaan pada M_1. Misalkan M_2 adalah matching maksimum dalam G, sehingga tidak terdapat path perluasan L dalam G yang berkenaan pada M_2. Ambil H adalah spanning subgraph G dengan \( E(H) = (M_1 - M_2) \cup (M_2 - M_1) \), dan H_1 komponen H. Karena tiap titik pada H incident dengan paling banyak satu garis pada M_1 dan M_2 maka tiap komponen H adalah path atau cycle. Karena tidak terdapat path perluasan dalam M_1 atau M_2 maka H_1 merupakan path trivial dan H_1 adalah titik terasing. Karena pada sebuah matching tiap garis tidak adjacent maka cycle dan path alternating dalam H mempunyai garis di dalam M_1 dan M_2 secara bergantian. Oleh
karena itu tiap cycle dan path alternating tersebut genap. Sehingga 
\[ |M_i| = |M_2| \] dan \( M_1 \) adalah matching maksimum.

**Definisi 2.2.4.** :

Misalkan suatu graph \( G \) dengan himpunan bagian tidak kosong dan disjoint \( V_1 \) dan \( V_2 \) dari \( V(G) \). \( V_1 \) matched ke \( V_2 \) jika terdapat matching \( M \) dalam \( G \) sedemikian sehingga tiap garis pada \( M \) incident dengan suatu titik \( V_1 \) dan suatu titik \( V_2 \), dan tiap titik \( V_1 \) atau \( V_2 \) incident dengan sebuah garis pada \( M \). Jika \( M \subseteq M^* \), dengan \( M^* \) adalah matching dalam \( G \), maka \( V_1 \) matched under \( M^* \) ke \( V_2 \).

**Definisi 2.2.5.** :

\( V \) adalah himpunan titik pada graph \( G \), persekitaran \( N(V) \) adalah himpunan semua titik \( G \) yang adjacent dengan paling sedikit satu anggota \( V \). \( V \) dikatakan nondeficient jika \( |N(C)| \geq |C| \) untuk setiap \( C \subseteq V \) yang tidak kosong.

**Teorema 2.2.2.** :

\( G \) adalah graph bipartite dengan himpunan titik \( V_1 \) dan \( V_2 \). Himpunan \( V_1 \) dapat matched ke himpunan bagian \( V_2 \) jika dan hanya jika \( V_1 \) nondeficient.
Bukti:

$\Rightarrow V_1$ dapat matched ke himpunan bagian $V_2$, maka $V_1$ nondeficient.

Misalkan $V_1$ matched under $M^*$ ke himpunan bagian $V_2$, maka tiap himpunan bagian $C$ dari $V_1$ yang tidak kosong dapat matched under $M^*$ ke suatu himpunan bagian $V_2$. Sehingga $|N(C)| \geq |C|$ dan $V_1$ nondeficient.

$\Leftarrow$ Jika $V_1$ nondeficient maka $V_1$ dapat matched ke himpunan bagian $V_2$.

Andaikan negasi pernyataan di atas benar sehingga $V_1$ nondeficient dan $V_1$ tidak dapat matched ke himpunan bagian $V_2$. Misalkan $M$ adalah matching maksimum dalam $G$. Terdapat titik $v \in V_1$ yang tidak incident dengan garis pada $M$. Misalkan $C$ adalah himpunan semua titik $G$ yang terhubung ke $v$ oleh sebuah path alternating. Karena $M$ adalah matching maksimum maka menurut teorema 2.2.1 $v$ adalah titik bebas dalam $C$. Misalkan $D_1 = C \cap V_1$ dan $D_2 = C \cap V_2$. Menurut definisi $C$ dan kenyataan bahwa tidak terdapat titik pada $C - \{v\}$ yang unmatched, maka $D_1 - \{v\}$ matched under $M$ ke $D_2$. Oleh karena itu $|D_2| = |D_1| - 1$ dan $D_2 \subseteq N(D_1)$. Untuk setiap titik $d \in N(D_1)$, graph $G$ berisi sebuah path alternating $v-d$, sehingga $N(D_1) \subseteq D_2$. Oleh karena itu $N(D_1) = D_2$ dan $|N(D_1)| = |D_2| = |D_1| - 1 < |D_1|$. Hal ini kontradiksi.

Jadi $V_1$ dapat matched ke himpunan bagian $V_2$. 
Algoritma 2.2.1. : {Algoritma matching maksimum pada graph bipartite}

Langkah 1 : i = 1

n = jumlah titik dalam V

M = matching awal

Langkah 2 : Apakah i ≤ n ?

2.1. Jika ya → lanjutkan ke langkah 3

2.2. Jika tidak → berhenti

Langkah 3 : [Menyelidiki titik bebas \( v_i \in V \)] Apakah \( v_i \) adalah titik matched ?

3.1. Jika ya → i = i+1, kembali ke langkah 2

3.2. Jika tidak → bentuk pohon alternating dengan \( v_i \) sebagai akar.

Langkah 4 : [Pohon alternating dengan akar \( v_i \)]

4.1. Hubungkan \( v_i \) dengan \( u_j \in U \) yang adjacent dengan \( v_i \) sampai konstruksi lengkap.

4.2. Apakah ditemukan path perluasan ?

4.2.1. Jika ya → lanjutkan ke langkah 5

4.2.2. Jika tidak → i = i+1, kembali ke langkah 2

Langkah 5 : Perluas M sekitas path perluasan untuk menghasilkan matching \( M' \), dan \( M = M' \), i = i+1, kembali ke langkah 2.
2.3 ALGORITMA BACKTRACKING

Kata algoritma diambil dari nama seorang penulis Persia, yaitu Abu Ja’far Mohammed ibn Musa al Khowarizmi, yang menulis sebuah buku tentang matematika. Menurut kamus Webster, algoritma adalah metode khusus untuk menyelesaikan suatu persoalan. Sedangkan menurut ilmu komputer algoritma merupakan metode yang dapat digunakan oleh komputer untuk menyelesaikan suatu peroalan.

Algoritma tersusun atas himpunan berhingga langkah – langkah, tiap langkah membutuhkan satu atau lebih operasi.

Algoritma backtracking dimulai dari node akar, kemudian dibuat tingkat yang berikutnya dengan cara mencari semua kemungkinan node pada node yang ada di bawahnya. Jika penyelidikan tidak menemukan tujuan yang diharapkan, maka penyelidikan akan mundur ke node di atasnya yang mempunyai kemungkinan untuk menjadi path menuju node tujuan.

Algoritma untuk menemukan pohon pada sebuah graph yang mempunyai n node dengan menggunakan aturan pencarian kedalaman pertama merupakan salah satu contoh algoritma backtracking. Algoritma tersebut adalah sebagai berikut:

Algoritma 2.3.1.:

Langkah 1 : u = v₁, v₁ sebagai node akar. Misalkan T adalah pohon yang berisi v₁.
Langkah 2 : Pilih node \( v_i \), dengan i minimum. Apakah terdapat garis yang menghubungkan \( u \) dengan \( v_i \) ?

2.1. Jika ya → Apakah penambahan garis tersebut pada \( T \) membentuk sebuah cycle ?

2.1.1. Jika ya → lanjutkan ke langkah 3

2.1.2. Jika tidak → tambahkan garis tersebut pada \( T \) dan \( u = v_i \), kembali ke langkah 2.

2.2. Jika tidak → lanjutkan ke langkah 4

Langkah 3 : Apakah \( u = v_i \) ?

3.1. Jika ya → berhenti.

3.2. Jika tidak → lanjutkan ke langkah 4.

Langkah 4 : [ backtrack ] Misal \( x \) adalah parent dari \( u \) ( di dalam \( T \) ), \( u = x \). Kembali ke langkah 2.

2.4 BAHASA PEMROGRAMAN PASCAL


Bahasa pemrograman Pascal adalah suatu bahasa pemrograman yang terstruktur, mudah dipelajari, dan urutan logikanya jelas dan sederhana.
Prinsip utama dari pemrograman terstruktur adalah jika suatu proses eksekusi telah mencapai baris tertentu maka proses eksekusi selanjutnya tidak boleh melompat ke baris sebelumnya, kecuali untuk proses berulang.


Pada awalnya Pascal ditujukan sebagai alat mengajar konsep pemrograman, namun pada akhirnya digunakan secara luas di dalam berbagai bidang kehidupan, termasuk bidang ekonomi dan industri. Meskipun Pascal mudah dipelajari, tetapi mengandung cakupan konsep pemrograman yang luas. Adapun syntax program Pascal adalah sebagai berikut:

```
PROGRAM nama_program

Deklarasi

BEGIN

Statement

END.
```

Sebuah program Pascal diawali dengan kata `PROGRAM` yang diikuti dengan nama program tersebut. Bagian deklarasi digunakan untuk
mendeklarasikan variable, type data, dan constanta yang digunakan dalam
program tersebut. Kata **BEGIN** mengawali instruksi atau perintah yang
akan dijalankan oleh komputer. Akhir program ditandai dengan **END**.

Pascal mempunyai empat type data yang digunakan untuk memproses
data, yaitu:

1. **Integer**
   Integer merupakan sebuah bilangan positif atau negatif yang tidak
   mengandung koma (,). Integer dimulai dari -32767 sampai 32767.

2. **Real**
   Bilangan real merupakan bilangan desimal. Real mempunyai bagian
   integer dan bagian fractional yang diantaranya dibatasi oleh tanda koma

3. **Character**
   Type data Character merupakan data yang terdiri atas satu karakter, bisa
   berupa huruf, angka, atau simbol.

4. **Boolean**
   Boolean adalah type data yang hanya mempunyai dua nilai yaitu True
   (benar) dan False (salah). Type ini digunakan dalam tes kondisi untuk
   membantu komputer membuat keputusan.

Selain type data, bagian lain dari Pascal adalah **Variable**. Penggunaan
variable harus dideklarasikan terlebih dahulu pada bagian deklarasi. Hal
lain yang penting di dalam Pascal adalah penggunaan **Constanta** yang juga
dideklarasikan pada bagian deklarasi.
Pascal mempunyai beberapa kata yang mempunyai arti khusus terhadap Pascal dan tidak dapat digunakan sebagai identifier. Kata-kata tersebut disebut *Reserved Word*.

2.5 **REKURSI**


a. Faktorial \( n! \)
   1. \( 0! = 1 \)
   2. Jika \( n > 0 \), maka \( n! = n \cdot (n-1) \)

b. Deret Fibonacci
   1. \( \text{fibo}(1) = 1 \)
   2. \( \text{fibo}(2) = 1 \)
   3. \( \text{fibo}(n) = \text{fibo}(n-1) + \text{fibo}(n-2) \)

Pada proses rekursi, komputasi bilangan yang tidak berhingga dapat dinyatakan dengan program rekursi berhingga, meskipun di dalam
tidak tertulis statemen perulangan secara eksplisit. Program rekursi dapat
dinyatakan sebagai komposisi \( \varphi \) dari statemen dasar \( S_i \) (tidak berisi \( F \)) dan
\( F \) itu sendiri, dapat dinyatakan sebagai berikut:

\[ F = \varphi [S_i, F] \]  

2.5.1

Alat yang diperlukan dalam menyatakan program rekursi adalah prosedur
atau subroutine. Program rekursi dapat dibagi menjadi dua bagian, yaitu:

a. Rekursi langsung, jika prosedur \( F \) berisi acuan eksplisit ke
dirinya sendiri.

b. Rekursi tidak langsung, jika prosedur \( F \) berisi acuan ke
prosedur lain.

Di dalam sebuah prosedur rekursi dinyatakan sejumlah obyek lokal,
yaitu sekumpulan peubah, konstanta, tipe, dan prosedur yang didefinisikan
secara lokal ke prosedur ini dan tidak dikenal di luar prosedur tersebut.
Setiap kali prosedur diaktifkan secara rekursi maka himpunan peubah
lokal yang baru akan dibentuk, sehingga meskipun mereka mempunyai
nama yang sama dengan himpunan peubah lokal dari pemanggilan
sebelumnya tetapi nilai mereka berbeda. Konflik dalam penggunaan nama
peubah dapat dihindari dengan menggunakan aturan skope peubah, yaitu
pengenal selalu mengacu kepada himpunan peubah baru yang dibentuk.
Aturan tersebut juga berlaku untuk parameter prosedur yang sesuai dengan
definisinya terbatas pada prosedur yang dimaksud.
Pada prosedur rekursif juga terdapat kemungkinan terjadi komputasi yang tak pernah berhenti, sehingga diperlukan adanya kondisi atau syarat untuk menghentikan proses eksekusi program. Syarat dasarnya adalah pemanggilan rekursif ke prosedur F harus memperhatikan B, yang pada suatu saat harus bernilai salah. Algoritma rekursif tersebut dapat dinyatakan sebagai berikut:

\[ F \equiv \text{if } B \text{ then } \emptyset [S_n, F] \quad 2.5.2 \]

atau

\[ F \equiv \emptyset [S_n, \text{if } B \text{ then } F] \quad 2.5.3 \]

Teknik dasar yang biasa digunakan untuk menghentikan proses perulangan adalah dengan mendefinisikan fungsi \( f(x) \) sedemikian sehingga \( f(x) \) akan berkurang nilainya pada setiap perulangan. Oleh karena itu, penghentian proses rekursif dapat dilakukan dengan memperlihatkan bahwa setiap eksekusi \( F \) akan mengurangi nilai \( f(x) \). Salah satu cara untuk melakukan penghentian adalah menggunakan nilai parameter, misalkan \( n \), di dalam \( F \) dan secara rekursif memanggil \( F \) dengan \( n - 1 \) sebagai nilai parameter. Penggantian \( B \) dengan \( n>0 \) akan menjamin bahwa proses rekursif akan berhenti. Kondisi tersebut dapat dinyatakan sebagai berikut:

\[ F(n) \equiv \text{if } n>0 \text{ then } \emptyset [S_n, F(n-1)] \quad 2.5.4 \]

\[ F(n) \equiv \emptyset [S_n, \text{if } n>0 \text{ then } F(n-1)] \quad 2.5.5 \]