Penerapan Program Dinamis untuk Perataan Teks

Jeremia Jason Lasiman - 13514021
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
jeremia_jason@yahoo.com

***Abstrak*—Perataan teks adalah sebuah fitur yang biasa dimiliki oleh tiap *word processor* dan pada umumnya selalu menggunakan algoritma *greedy*. Penggunaan algoritma *greedy* pada perataan teks memang normal, karena hasil yang dikeluarkan sama dengan saat pengguna mengetik secara langsung. Permasalahan muncul ketika hal yang sama diterapkan kepada perataan kiri-kanan (*justify*). Suatu baris dapat memiliki tingkat keburukan yang sangat tinggi sehingga ketika digunakan algoritma *greedy* hanya akan memperbesar spasi antar kata. Ini dapat menjadi tidak nyaman untuk dilihat karena dapat menimbulkan spasi yang sangat besar antar kata-katanya. Permasalahan perataan teks *justify* ini sebenarnya adalah permasalahan untuk pemecahan kata-kata tiap baris atau lebih dikenal sebagai *word warping / line breaking* yang cukup umum ada pada *text editor* manapun. Oleh sebab itu, banyak yang memikirkan untuk mengurangi tingkat keburukan baris. Salah satu caranya adalah dengan memamfaatkan algoritma program dinamis sehingga mencapai solusi yang optimum.**

Keywords—justify; perataan teks; word warping; line breaking; program dinamis, tingkat keburukan baris

I. PENDAHULUAN

Pengunaan *word processor* untuk menulis dokumen sudah tidak asing lagi pada zaman sekarang. Kebanyakan *word processor* yang digunakan tentu memiliki fitur perataan teks (*text alignment*). Perataan teks ini memudahkan pengguna untuk dapat membaca tulisan yang dibuat serta merapikan hasil ketikan dari penulis. Hal ini juga menjadi pembeda yang signifikan antara *word processor* dan *text editor* seperti Notepad ddan lainnya. Tetapi, terkadang hasil dari perataan teks ini kurang memuaskan seperti terdapat celah yang besar pada baris-baris tertentu.

Perataan teks yang ada pada beberapa *word processor* seperti Microsoft Word atau OpenOffice.org menggunakan algoritma *greedy* pada penggunaannya. Meskipun pada kebanyakan kasus cukup baik dan juga cepat, tetapi dapat menimbulkan celah. Hal ini dikarenakan pemilihan tiap kata yang dapat masuk pada tiap paragraf kurang optimal sehingga tingkat keburukan teks dalam satu baris menjadi tidak merata dan mungkin ada yang sangat buruk. Untuk itu, terdapat beberapa algoritma lain yang dapat melakukan optimasi untuk kasus seperti ini, diantaranya dengan menggunakan program dunamis ataupun dengan algoritma SMAWK (yang tidak dibahas pada algoritma ini).

Penggunaan program dinamis cocok untuk digunakan pada masalah ini karena memiliki sub-masalah yang dipakai berulang-ulang. Sub-masalah itu diantara lain adalah *word warping* atau disebut juga *line breaking* yaitu penentuan kata-kata yang ada pada satu baris. *Word warping* adalah fitur yang cukup umum ditemukan, bahkan pada *text editor* sekalipun. Masalah perataan teks sebenarnya kumpulan dari masalah *word warping.*

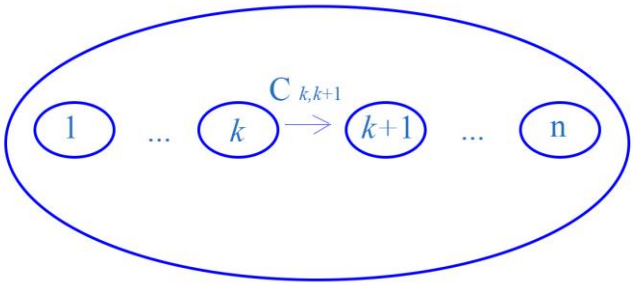
II. PROGRAM DINAMIS (DYNAMIC PROGRAMMING)

Program dinamis adalah metode pemecahan masala dengan cara menguraikan solusi menjadi sekumpulan tahapan (*stage*). Solusi dari persoalan dapat dipandang dari serangkaian keputusan yang saling berkaitan. Algoritma program dinamis ini sangat ampuh karena untuk tiap sub-masalah, hanya perlu diselesaikan sekali kemudian hasilnya disimpan untuk perhitungan selanjutnya.Terdapat beberapa karakteristik penyelesaian persoalan menggunakan program dinamis, yaitu [1]:

1. Terdapat sejumlah berhingga pilihan yang mungkin.
2. Solusi pada setiap tahap dibangun dari hasil solusi tahap sebelumnya.
3. Mengguanakan persyaratan optimasi dan kendala untuk membatasi sejumlah pilihan yang harus dipertimbangkan pada suatu tahap.

Pada program dinamis, rangkaian keputusan yang optimal dibuat dengan menggunakan **prinsip optimalitas.** Prinsip ini menyatakan jika solusi total optimal, maka bagian solusi sampai tahap ke-*k* juga optimal. Prinsip optimalitas berarti bahwa jika kita bekerja dari tahap *k* ke tahap *k* + 1, kita dapat menggunakan hasil optimal dari tahap *k* tanpa harus kembali ke tahap awal.

Ongkos pada tahap *k* + 1 = (ongkos yang dihasilkan pada tahap *k*) + (ongkos dari tahap *k* ke tahap *k* + 1)

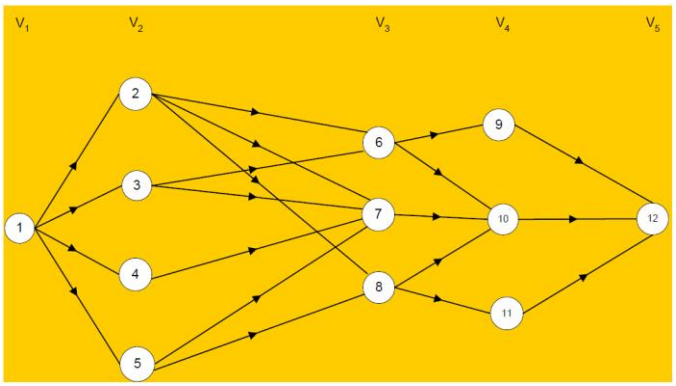


Gambar 1. Ongkos (*cost*) dari *k* ke *k* + `1

Program dinamis menjadi suatu algoritma yang handal dalam menangani optimasi suatu persoalan. Secara kasar, program dinamis dapat dianggap sebagai '*brute force* yang pintar' karena karakteristiknya yang mencoba setiap kemungkinan solusi permasalahan. Program dinamis lebih efisien meskipun mencari tiap kemungkinan solusi karena melakukan penyimpanan solusi optimum dari sub-masalah yang ada, sehingga tidak perlu memulai ulang dari awal, seperti *brute force*, tiap kali menemukan sub-masalah yang sama. Persoalan yang dapat diselesaikan oleh program dinamis memiliki karakteristik berupa [1]:

- Persoalan dapat dibagi menjadi beberapa tahap (*stage*), yang pada setiap tahap hanya diambil satu keputusan.
- Masing-masing tahap terdiri dari sejumlah status (*state*) yang berhubungan dengan tahap tersebut. Secara umum, status merupakan bermacam kemungkinan masukan yang ada pada tahap tersebut.
- Hasil dari keputusan yang diambil pada setiap tahap ditransformasikan dari status yang bersangkutan ke status berikutnya pada tahap berikutnya.
- Ongkos (*cost*) pada suatu tahap meningkat secara teratur (*steadily* dengan bertambahnya jumlah tahapan.
- Ongkos pada suatu tahap bergantung pada ongkos tahap-tahap yang sudah berjalan dan ongkos pada tahap tersebut.
- Keputusan terbaik pada suatu tahap bersifat independen terhadap keputusan yang dilakukan pada tahap sebelumnya.
- Adanya hubungan rekursif yang terbaik untuk setiap status pada tahap *k* memberikan keputusan terbaik untuk setiap status pada tahap *k* + 1.
- Prinsip optimalitas berlaku pada persoalan tersebut.

Contoh Persoalan yang dapat diselesaikan oleh program dinamis adalah graf multitahap seperti berikut.



Gambar 2. Contoh graf multitahap.

Terdapat dua pendekatan yang dapat digunakan dalam program dinamis, yaitu pendekatan maju (*forward / up-down*) dan pendekatan mundur (*backward / bottom- up*). Program dinamis maju bergerak dari tahap 1, 2, 3, sampai tahap ke *n*. Sedangkan program dinamis mundur bergerak dari tahap *n*, *n*-1, *n*-2, sampai tahap ke 1.

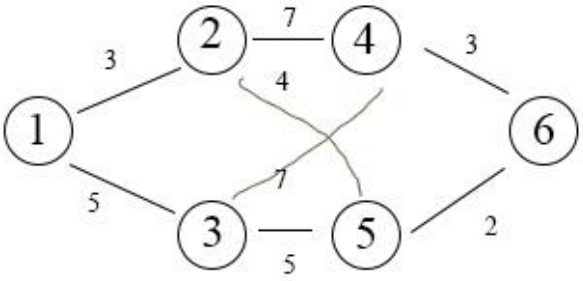
Langkah-langkah pengembangan algoritma program dinamis [1]:

1. Karakteristikan strtuktur solusi optimal.
2. Definisikan secara rekursif nilai solusi optimal.
3. Hitung nilai solusi optimal secara maju atau mundur.
4. Konstruksi solusi optimal.

Lebih mudahnya dapat dituliskan 5 langkah untuk melakukan program dinamis [2]:

1. Definisikan sub-masalah. Hitung jumlah sub-masalah
2. Mengira bagian dari solusinya. Definisikan berapa jumlah pilihan yang ada.
3. Hubungkan solusi dari sub-masalah.
4. Rekursif dan penyimpanan. Di sini juga menentukan untuk menggunakan program dinamis secara *top-down* atau *bottom-up*.
5. Selesaikan masalah awal yang merupakan sebuah sub-masalah atau kombinasi solusi dari sub-masalah.

Keunggulan dari algoritma ini adalah jalurnya yang sudah teratur dan juga terdapat kemampuan mengingat solusi optimum dari tiap sub-masalah. Mengingat solusi optimum untuk tiap masalah ini dapat dengan menggunakan larik, matriks, maupun beberapa hal lainnya yang mungkn lebih cocok untuk keperluan tertentu. Salah satu contoh penyimpanan yang cukup mudah dipahami adalah ketika mencari solusi optimum untuk *Travelling Salesman Problem.*



Gambar 3. Contoh persoalan TSP.

Contoh dari 1 ingn mencapai ke 6, maka dapat dibuat tabel (dalam representasi manual) untuk mencari dan menyimpan jalur yang telah dilewati. Maka akan didapatkan

$f_1(s) = c_{xls}$

S	Solusi optimum	
	$f_1(s)$	x_1
2	3	1
3	5	1

(x_k adalah nilai yang meminimumkan f_s)

S \ x2	$f_2(x_2,s) = c_{x2} + f_1(x_2)$		Solusi optimum	
	2	3	$f_2(s)$	x_2
4	10	12	10	2

Dan seterusnya sehingga mencapai 6.

Dapat dilihat pada kolom paling kanan tabel (solusi yang diambil) bahwa jalur yang diambil sejauh ini untuk mencapai 4 adalah 1-2-4 dan jika solusi optimum melalui 4, maka bagian awal yang diambil pasti melalui 1-2-4 juga.

III. Perataan Teks (Typographic/Text Alignment)

Terdapat empat perataan teks yang dasar ada, yaitu :

- Rata kiri (*flush left*)
- Rata kanan (*flush right*)
- Rata tengah (*centered*)
- Rata kiri-kanan (*justified*)

A. *Rata Kiri (Flush Left)*

Pada perataan teks yang rata kiri, setiap kata pada awal baris diratakan pada batas kertas sebelah kiri. Perataan teks seperti ini umumnya digunakan karena penulisan bahasa yang ditulis dan dibaca dari kiri ke kanan seperti bahasa Inggris dan Indonesia.

Contoh teks rata kiri :

> Ini adalah contoh teks rata kiri. Secara normal hasil dari rata kiri ini seperti tulisan tangan biasa dan pada akhir tiap baris mungkin mendapatkan jarak yang tidak rata sampai batas kanan kertas.

B. *Rata Kanan (Flush Right)*

Seperti pada teks rata kiri, perataan teks rata kanan artinya adalah tiap baris pada teks diratakan pada batas kertas sebelah kanan. Penulisan seperti ini umumnya dipakai oleh bahasa yang ditulis dan dibaca dari kanan ke kiri seperti bahasa Arab dan Ibrani.

> Ini adalah contoh teks rata kanan. Seperti dapat dilihat bahwa tiap baris akan rata di bagian kanan, sedangkan bagian kiri dapat memiliki jarak yang tidak rata sampai batas kiri kertas.

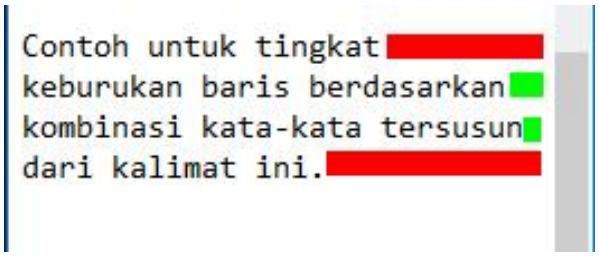
C. *Rata Tengah (Center)*

Rata tengah merupakan perataan teks yang teks dibuat mempunyai jarak yang sama ke batas kiri dan kanan dari kertas atau kolom sehingga tulisan tepat berada di tengah.

> Contoh teks rata tengah. Biasanya digunakan sebagai judul dari suatu tulisan. Biasanya digunakan sebagai judul suatu teks, menjadi fokus suatu tulisan, atau untuk penggunaan lainnya.

D. *Rata kiri-kanan (justified)*

Rata kiri-kanan atau lebih dikenal sebagai *justified* merupakan teknik perataan teks dimana pada suatu baris, teks akan selalu rata bagian kiri dan kanannya pada tepi kolom / kertas. Perataan teks ini sangat sering digunakan karena membuat tampilan lebih rapi dan memudahkan pembacaan. Tetapi, perataan *justified* ini juga memiliki kekurangan, yaitu spasi antar kata pada suatu baris dapat menjadi terlalu renggang karena tingkat keburukan suatu baris akan menjadi terdistribusi pada spasi tiap kata. Tingkat keburukan ini adalah suatu istilah dari jarak suatu baris ke tepi kolom yang tersisa.[3]



Gambar 4. Tingkat keburukan teks

Pada gambar ini ditunjukkan bahwa baris yang memiliki warna merah mempunyai tingkat keburukan yang tinggi, sedangkan warna hijau menunjukkan bahwa tingkat keburukannya tidak tinggi.

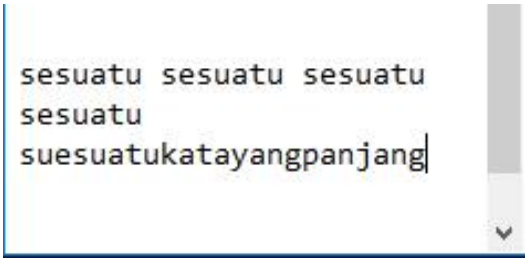
Contoh suatu teks menggunakan perataan *justified*. Dapat dilihat bahwa pada tiap baris memiliki jarak yang berbeda spasinya, namun rata kiri dan kanan. Berikut IniMembuatTeksMenjadi memilikiSpasi yang panjang karena terdapat kata yang tidak dipecahkan.

Pada beberapa *word processor* seperti Microsoft Word atau OpenOffice.org Writer, perataan *justified* ini mengunakan juga algoritma *greedy*. Pada dasarnya, algoritma *greedy* cukup baik karena prinsipnya yang mengambil sebanyak mungkin kata pada suatu baris dan hanya pindah baris ketika tidak cukup lagi. Algoritma *greedy* untuk rata kiri, rata kanan, rata tengah tidak terlalu terasa karena memang sudah terlihat ada sisi yang tidak rata, namun ketika menggunakan algoritma *greedy* untuk perataan *justified* akan terlihat jarak antar kata yang sangat mencolok. Ini disebabkan karena pilihan pemotongan baris yang tidak optimum dan pada akhirnya hanya mengatur jarak spasi antar kata.

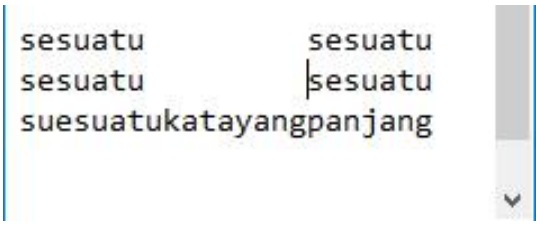
Beberapa cara untuk mengatasi ini adalah menggunakan pemenggalan kata, yang sekarang sudah beberapa aplikasi menyediakan secara otomatis (tidak dibahas pada makalah ini), atau mengatur ulang penyusunan kata-kata per baris. Karena itu lebih baik untuk menggunakan metode program dinamis sehingga hasil yang diberikan akan menjadi optimum.

IV. Word Warping / Line Breaking

Seperti yang sudah dibahas sebelumnya, permasalahan pada perataan *justify* sebenarnya adalah mengatur pemecahan baris di kata yang tepat. Permasalahan ini disebut sebagai *word warping*. Terdapat dua alternatif untuk masalah ini, yaitu dengan meminimumkan jumlah baris, atau minimumkan kekasaran (tingkat keburukan baris). Algoritma *greedy* bertujuan untuk meminimumkan jumlah baris dan efektifitas waktu. Program dinamis dan algoritma SMAWK meminimumkan kekasaran. Program dinamis ini memiliki kompleksitas waktu $O(n^2)$ sedangkan algoritma SMAWK memiliki kompleksitas waktu $O(n)$[5].



Gambar 5. Tingkat keburukan teks yang biasa ditemukan menggunakan algoritma *greedy*



Gambar 6. Tingkat kebuukan teks berkurang seletah optimasi menggunakan program dinamis

Meminimumkan jumlah baris dapat menimbulkan baris yang buruk. Ini lah masalah yang coba dipecahkan oleh pendekatan untuk minimumkan kekasaran baris. Terdapat beberapa algoritma yang dapat digunakan untuk meminimumkan kekasaran baris, diantaranya adalah :

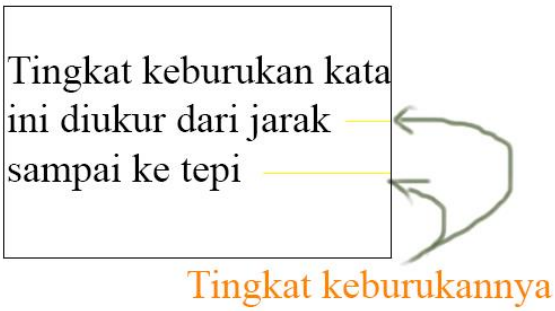
Nama Algoritma	Kompleksitas
Brute Force	$O(2^n)$
Dynamic Programming (tanpa ada asumsi tambahan)	$O(n^2)$
Shortest Path (algoritma program dinamis yang membatasi ketika sudah melewati batas lebar kolom akan tidak dihitung)	O(n * width)
SMAWK	O(n)
Divide & Conquer	O(n * log n)

Tabel 1. Perbandingan kompleksitas berbagai algoritma dalam perataan teks[5]

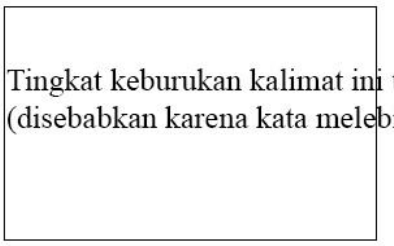
V. Implementasi Program Dinamis Untuk Masalah Text Justification

Pertama dalam menyelesaikan masalah perataan teks secara rata kiri-kanan, perlu didefinisikan dahulu tujuan yang ingin dicapai. Juga harus ada penentu keberhasilannya. Dalam hal ini, digunakan pengukuran berupa tingkat keburukan barisan kata. Misalkan tingkat keburukan menjadi lebar kolom dikurangi total panjang kata atau tingkat keburukan didefinisikan menjadi tak hingga ketika total panjang kata melebihi lebar kolom. Tingkat keburukan ini didefinisikan sebagai :

Tingkat keburukan (*i*,*j*) untuk baris kata [i : j]



Gambar 7. Cara menghitung tingkat keburukan teks



Gambar 8. Baris yang sudah melewati batas kolom dalam perhitungkan memiliki tingkat keburukan teks tak hingga

Tujuan dari penggunaan algoritma program dinamis untuk masalah ini adalah meminimalkan tingkat keburukan total dari suatu tulisan. Dilakukan dengan cara membagi kata pada tempat yang mungkin berbeda dengan pada awal diketik, seperti pada gambar 4.2 dimana kata 'sesuatu' yang ke 3 dipindahkan ke baris selanjutnya untuk mengurangi tingkat keburukan dari baris 2 sehingga didapatkan seperti gambar 4.3.

Menyelesaikan masalah ini menggunakan program dinamis berarti perlu mendefinisikan apa sub-masalahnya. Seperti yang sudah dibahas sebelumnya bahwa permasalahan perataan teks secara *justfy* adalah kumpulan dari masalah *word warping / line breaking*. Kemudian perlu dilihat apa saja kemungkinan solusinya. Kemungkinan solusinya adalah dengan menyelesaikan satu masalah *word warping* ini, yaitu permasalahan letak pemenggalan kata dalam satu baris. Dengan menyelesaikan masalah *word warping*, dapat disimpulkan bahwa permasalahan secara utuh dapat diselesaikan. Masalah ini didefinisikan sebagai pencarian tingkat keburukan minimum untuk bagian belakang kata [*i* :] pada suatu baris.

Setelah mendapatkan apa tujuan dan masalah yang ada, maka perlu didefinisikan fungsi rekursif yang akan digunakan. Perlu diperhatikan untuk cara penghubungan antar sub-masalah yang ada dengan masalah utama. Dalam permasalahan ini, sub-masalah yang ada dan juga masalah utama pada dasarnya adalah sama sehingga langusng terlihat keterhubungannya. Selain itu, perlu diperhatikan juga dengan pendekatan apa masalah ini diselesaikan. Pada tahap ini dipilih *bottom-up* sebagai pendekatan untuk menyelesaikan masalah. Hasil rekursif yang dihasilkan, yaitu [2]:

- Solusi pada baris terakhir adalah nol. Karena diasumsikan untuk baris terakhir adalah baris kosong.

  F[n] = 0

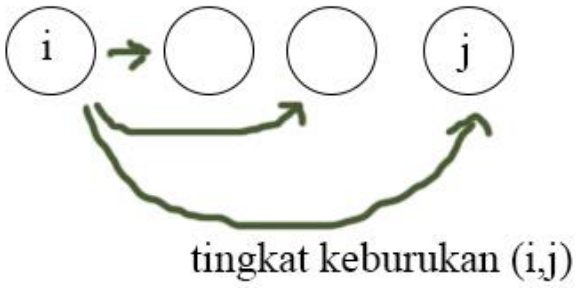
- Solusi selain baris terakhir adalah minimum dari solusi baris bawahnya ditambah dengan tingkat keburukan baris ini selama baris masih belum melebihi batas kolom.

  F[i] = min (tingkat keburukan (i,j), + F[j])

  (untuk j dalam jarak i+1 sampai n+1)

  Waktu yang diperlukan menyelesaikan satu masalah adalah $O(n)$.

  Waktu yang diperlukan untuk mencapai solusi total adalah $O(n^2)$.

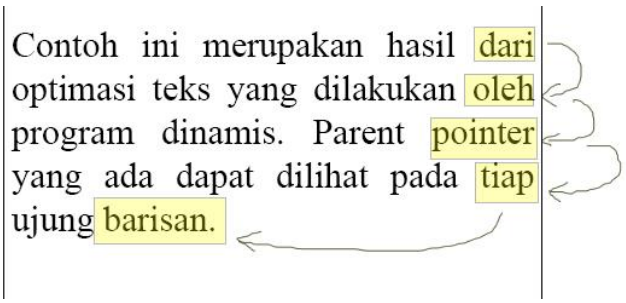


Gambar 8. Tingkat keburukan dari i ke j

Solusi total dari semua ini adalah solusi ketika sudah mencapai baris paling awal, yaitu baris nol. Namun, tidak cukup hanya dengan mencari solusi pada baris pertama, karena setelah itu hanya mendapat nilai hasil minimum yang mungkin dari tingkat keburukan yang mungkin. Karena itulah perlu ada penyimpanan jalur yang telah dilewati.

Penyimpanan jalur yang telah dilewati dapat menggunakan pointer kepada sebelumnya yang disebut *parent pointer*. *Parent pointer* menunjukkan solusi optimum sebelumnya yang telah dipilih untuk mencapai solusi optimum sekarang. *Parent pointer* ini sangat berguna untuk dapat dengan cepat mencari jejak jalur yang dilewati dari awal sampai akhir untuk mencapai solusi optium dari masalah yang diberikan.

Parent pointer yang dimaksud di sini mirip dengan penggunaan sistem tabel untuk TSP yang ada pada bab pembahasan program dinamis, yaitu ketika ditemukan solusi optimum, dicatat nilai minimumnya dan juga jalur yang memberi nilai minimum. Hal demikian dapat juga digunakan sebagai *parent pointer* ini. Kata terakhir pada baris perama akan menunjukkan letak kata selanjutnya harus berhenti yang merupakan solusi optimum. Kemudian akan terus dilanjutkan sampai beris terakhir ketika tidak ada lagi yang ditunjuk.



Gambar 9. Contoh *parent pointer* jika dapat dilhat langsung

VI. ANALISIS HASIL

Dari hasil pembahasan ini, dapat disimpulkan bahwa penggunaan algoritma program dinamis akan mengurangi tingkat keburukan pada teks, tetapi memakan waktu lebih lama dari algoritma *greedy* karena kompleksitasnya yang berupa $O(n^2)$. Tetapi untuk tulisan yang tidak terlalu banyak (kurang dari 5000 kata) masih dapat dieksekusi kurang dari satu detik

sehingga sebenarnya cukup baik jika hanya untuk merapikan tulisan.

VII. Kesimpulan dan Saran

Program dinamis dapat dikatakan sebagai 'algoritma *brute force* yang pintar' karena sifatnya yang mencoba semua kemungkinan solusi pada tiap sub-masalah. Namun, algoritma ini jauh lebih berkekuatan karena memiliki struktur yang rapi dalam pembuatannya dan juga menggunakan sistem penyimpanan nilai optimum untuk tiap sub-masalah yang ada. Algoritma ini sangat baik digunakan untuk optimasi suatu permasalahan terutama permasalahan yang memiliki sub-masalah yang banyak dan saling berkait satu dengan lainnya. Program dinamis secara sederhana sudah cukup untuk menyelesaikan masalah optimasi, namun dapat lebih dikembangkan lagi dengan ditambahkan beberapa asumsi semantik untuk mengurangi kompleksitasnya. Seperti pada perataan teks ini, algoritma *shortest path* yang dimaksud di sini adalah program dinamis yang sudah ditambahkan semantik sehingga kompleksitasnya berkurang.

Perataan teks pada dasarnya terdapat empat macam, yaitu rata kiri (flush), rata kanan (*flush right*), rata tengah (*centered*), rata kiri-kanan (*justified*). Perataan teks ini pada umumnya menggunakan algoritma *greedy* yang hasilnya sama persis dengan hasil ketikan langusng tanpa ada proses. Algoritmnya *greedy* juga diterapkan pada *justified* meskipun terlihat tidak optimum. Dengan penggunaan algoritma *greedy*, maka *word processor* hanya perlu untuk mengatur jarak antar kata dan tidak ada perubahan susunan kata di barisan karena hasil yang diberikan sudah sama dengan hasil mengetik langusng. Inilah yang dicoba diselesaikan oleh program dinamis, yaitu untuk mengurangi tingkat keburukan teks (karena spasi antar kata yang terlalu jauh dan tidak rata). Program dinamis mampu untuk mencari susunan kata per baris yang terbaik sehingga tingkat keburukan teks yang didapait menjadi minimal. Pemanfaatan algoritma program dinamis ini dengan cara menyelesaikan sub-masalah pada perataan teks, yaitu *word warping* dengan menghitung tingkat keburukan teks minimum pada suatu paragraf. Selain dhitung juga dicatat jalur yang telah dilalui menggunakan *parent pointer*. Program dinamis ini menggunakan pendekatan *bottom-up* dan akan selesai ketika perhitungan sudah mendapatkan solusi untuk baris pertama dari paragaraf yang diproses.

Ucapan Terima Kasih

Puji syukur kepada Tuhan Yang Maha Esa karena atas berkat-Nya telah memampukan penulis untuk menyelesaikan makalah ini dengan baik dan telap waktu. Penulis juga menyampaikan terima kasih kepada Bapak Rinaldi Munir dan Ibu Nur Ulva Maulidevi yang telah mengajarkan dasar-dasar teori yang diperlukan untuk Penulis dapat menyelesaikan makalah ini.

Referensi

[1]] Munir, Rinaldi. Strategi Algoritma. 2013. Bandung : Penerbit ITB

[2] Demaine, Erik Dynamic Programming II : Text Justification, Black Jack. http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-006-introduction-to-algorithms-fall-2011/lecture-videos/lecture-20-dynamic-programming-ii-text-justification-blackjack/#vid_playlist. 14.15 06/05/2016.

[3] Sim, Alex Xandra Albert. Dynamic Programming. https://bertzzie.com/knowledge/analisis-algoritma/DynamicProgramming.html#contoh-aplikasi-dynamic-programming-text-justification. 16.14 06/05/2016

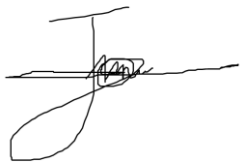
[4] D. E. Knuth, M. F. Plass. Breaking Paragraphs into Lines. Software--Practice and Experience 11, 1981.

[5] http://xxyxyz.org/line-breaking/ 09.45 07/05/2016.

Pernyataan

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 08 Mei 2016



Jeremia Jason Lasiman