PENERAPAN DYNAMIC PROGRAMMING DALAM WORD WRAP

Wafdan Musa Nursakti (13507065)

Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung
Jalan Ganesha No. 10 Bandung, 40132
e-mail: zorvanion@gmail.com, if17065@students.if.itb.ac.id

ABSTRAK

Pemrograman dinamis (*dynamic programming*) adalah metode pemecahan masalah dengan cara menguraikan solusi menjadi sekumpulan langkah atau tahapan sedemikian sehingga solusi dari persoalan dapat dipandang dari serangkaian keputusan yang saling berkaitan.

Kata kunci: Word Wrap, Dynamic Programming, Ongkos, Optimal

1. PENDAHULUAN

Pada display teks, line wrap adalah suatu fitur untuk melanjutkan ke baris baru ketika suatu baris sudah penuh dengan tujuan agar suatu baris teks muat dalam satu window, sehingga teks dapat dibaca dari atas ke bawah tanpa perlu melakukan horizontal scrolling.

Word Wrap adalah fitur tambahan pada editor teks, pengolah kata, dan web browser untuk memecah baris di antara kata, bukan pada kata, kecuali jika satu kata panjangnya melebihi panjang baris.

2. PEMROGRAMAN DINAMIS

Pemrograman dinamis (*dynamic programming*) adalah metode pemecahan masalah dengan cara menguraikan solusi menjadi sekumpulan langkah atau tahapan sedemikian sehingga solusi dari persoalan dapat dipandang dari serangkaian keputusan yang saling berkaitan. Pada penyelesaian dengan metode ini:

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

2.1 Karakteristik Persoalan Program Dinamis

Program dinamis diterapkan pada persoalan yang memiliki karakteristik sebagai berikut:

1. Persoalan dapat dibagi menjadi beberapa tahap (stage), yang pada setiap tahap hanya diambil satu keputusan.
2. 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. Jumlah status bisa berhingga (finite) atau tidak berhingga (infinite).
3. Hasil dari keputusan yang diambil pada setiap tahap ditransformasikan dari status yang bersangkutan ke status berikutnya pada tahap berikutnya.
4. Ongkos (cost) pada suatu tahap meningkat secara teratur (steadily) dengan bertambahnya jumlah tahapan.
5. Ongkos pada suatu tahap bergantung pada ongkos tahap-tahap yang sudah berjalan dan ongkos pada tahap tersebut.
6. Keputusan terbaik pada suatu tahap bersifat independen terhadap keputusan yang dilakukan pada tahap sebelumnya.
7. Adanya hubungan rekursif yang mengidentifikasikan keputusan terbaik untuk setiap status pada tahap k memberikan keputusan terbaik untuk setiap status pada tahap k+1.
8. Prinsip optimalitas berlaku pada persoalan tersebut.

Dalam menyelesaikan persoalan dengan pemrograman dinamis, dapat digunakan dua pendekatan berbeda: maju atau mundur. Keduanya menghasilkan solusi optimum yang sama. Namun pengalaman menunjukkan bahwa penyelesaian dengan program dinamis mundur umumnya lebih mangkus.

Secara umum terdapat empat langkah yang dilakukan dalam mengembangkan algoritma program dinamis:

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

Solusi optimal yang dihasilkan oleh pemrograman dinamis dapat lebih dari satu buah.

3. ANALISIS PENYELESAIAN MASALAH

Persoalan word wrap termasuk permasalahan optimasi. Algoritma untuk dipergunakan tergantung pada aspek apa yang akan dioptimasi.

Jika aspek yang akan dioptimasi adalah minimnya jumlah baris (line) yang digunakan, maka algoritma yang tepat digunakan adalah algoritma greedy, dimana algoritma itu menempatkan sebanyak-banyaknya kata (*word*) pada suatu baris hingga baris itu tidak dapat menampung kata lagi. Kemudian baris baru dibuat untuk menampung sisa kata-kata selanjutnya sampai tidak ada lagi kata yang dimasukkan. Pseudocode algortima greedy ini adalah sebagai berikut:

```
SpaceLeft := LineWidth
for each Word in Text
  if Width(Word) > SpaceLeft
    insert line break before Word in Text
      SpaceLeft := LineWidth - Width(Word)
  else
      SpaceLeft := SpaceLeft - (Width(Word)
                   + SpaceWidth)
```

Namun algortima di atas tidak optimal jika aspek yang dioptimasi adalah mimimnya kekasaran (*raggedness*). Teks dengan kekasaran minim lebih nyaman dipandang mata. Secara kuantitatif, kekasaran berarti jumlah kuadrat (atau kubik) dari sisa spasi pada akhir keseluruhan baris pada teks. Sebagai contoh, terdapat teks:

```
aaa bb cc dddd
```

Jika lebar baris (*line width*) --yaitu jumlah maksimum karakter pada suatu baris-- adalah 6, dan fungsi ongkos (*cost*) didefinisikan sebagai kuadrat dari sisa spasi pada suatu baris, maka dengan algoritma greedy didapatkan hasil seperti ini:

```
------   lebar baris : 6
aaa bb   (s=0,o=0,q=0)
cc       (s=4,o=4,q=16)
ddddd    (s=1,o=1,q=1)

keterangan :
s = sisa spasi
o = ongkos
q = dikuadratkan
```

Dimana total ongkosnya adalah 17. Sedangkan hasil optimumnya adalah sebagai berikut:

```
------   lebar baris : 6
aaa      (s=3,o=3,q=9)
bb cc    (s=1,o=1,q=1)
ddddd    (s=1,o=1,q=1)

keterangan :
s = sisa spasi
o = ongkos
q = dikuadratkan
```

Dimana total ongkosnya adalah 9, lebih optimal.

Untuk menyelesaikan persoalan ini dengan pemrograman dinamis, pertama-tama kita didefinisikan dahulu ongkos c(i,j) yang merupakan ongkos suatu baris yang mengandung kata ke-i sampai kata ke-j.

Ongkos didefinisikan sebagai page_width dikurangi spasi_terpakai. Spasi_terpakai pada suatu baris yang mengandung kata ke-i sampai kata ke-j adalah jumlah spasi di antara kata-kata yang berurutan (j - i) ditambah jumlah panjang kata ke-i sampai ke-j.

$$\text{spasi\_terpakai} = j - i + \sum_{k=i}^{j} s_k \quad (1)$$

Sehingga ongkos pada suatu baris adalah:

$$c(i,j) = \left(\text{page\_width} - j + i - \sum_{k=i}^{j} s_k \right)^P \quad (2)$$

Dimana P adalah 2 atau 3. Jika c(*i,j*) bernilai negatif (dikarenakan jumlah karakter melebihi `page_width`), maka didefinisikan nilai c(*i,j*) = ∞, yang berarti cost-nya terlalu besar alias tidak mungkin. Ongkos untuk solusi optimal didefinisikan dengan fungsi rekurens:

$$f(j) = \begin{cases} c(i,j) & \text{jika } c(1,j) < \infty \\ \min_{1 \le k \le j} \left(f(k) + c(k+1, j) \right) & \text{jika } c(1,j) = \infty \end{cases} \quad (3)$$

Contoh kasus, didefinisikan `page_width` = 10. Dan rangkaian kata: `Halo apa kabar dunia`

Pada kasus ini, diasumsikan bahwa satu kata adalah string tanpa spasi. Dan di antara dua kata yang berurutan selalu terdapat satu spasi.

Pendekatan pemrograman dinamis yang digunakan adalah program dinamis maju, dimana keadaan awal adalah belum ada kata yang ditempatkan.

Langkah 1:
penempatan kata pertama, 'Halo' pada baris pertama. Cost baris pertama menjadi $6^3 = 216$.
f(k) = c(i,k)

Tabel 1 Langkah 1

k	f(k)	Layout
1	f(1) = c(1,1) = $(10 - 1 + 1 - 4)^3$ = 6^3 = 216	`Halo`

Langkah 2 :
penempatan kata 'apa' masih pada baris pertama, karena masih mencukupi. Cost baris pertama menjadi $2^3 = 8$.
f(k) = c(i,k)

Tabel 2 Langkah 2

k	f(k)	Layout
2	f(2) = c(1,2) = $(10 - 2 + 1 - (4+3))^3$ = 2^3 = 8	`Halo apa`

Langkah 3 :
Karena c(1,3) = ∞, maka $f(j) = \min_{1 \le k \le j} \left(f(k) + c(k+1, j) \right)$

Tabel 3 Langkah 3

q / k	c(q+1,k) + f(q)		f(k)	Layout
	2	1		
3	c(3,3)+f(2) $=(10–3+3–5)^3 + 2^3$ $= 133$	c(2,3)+f(1) $=(10-3+2-(3+5))^3 + 6^3$ $= 217$	133	`Halo apa kabar`

penempatan kata 'kabar' pada baris pertama tidaklah mungkin, karena `page_width` tidak mencukupi. Jika dipaksakan, cost baris pertama menjadi ∞, sehingga mesti dilakukan penyusunan ulang. Pada langkah ini terdapat dua pilihan:

1) Membiarkan 'Halo apa' pada baris pertama dan memindahkan 'kabar' ke baris kedua. Layout-nya seperti ini:
```
Halo apa
Kabar
```
2) Membiarkan 'Halo' pada baris pertama dan memindahkan 'apa kabar' ke baris kedua. Layout-nya seperti ini:
```
Halo
apa kabar
```

Karena cost pilihan (1) lebih kecil daripada cost pilihan (2), 133 < 217, maka pilihan (1) dipilih menjadi Layout pada tahap ini.

Langkah 4 :
Karena c(3,4) = ∞, maka $f(j) = \min_{1 \le k \le j} \left(f(k) + c(k+1, j) \right)$

Tabel 4 Langkah 4

q / k	c(q+1,k) + f(q)	f(k)	Layout
	2		
4	c(4,4)+f(3) $=(10-4+4-5)^3+133=342$	342	`Halo apa kabar dunia`

penempatan kata 'dunia' sedianya dilakukan pada baris kedua (pada baris pertama jelas tidak mungkin). Namun ternayata `page_width` tidak mencukupi sehingga mesti dilakukan penyusunan ulang. Karena pada langkah ini hanya terdapat satu pilihan: Membiarkan 'kabar' pada baris kedua dan memindahkan 'dunia' pada baris ketiga, maka pilihan inilah yang dipilih. Kenyataannya, baris terakhir pada umumnya tidak ikut terkena word wrap, sehingga berapapun cost baris terakhir tidak mempengaruhi layout secara keseluruhan.

Karena tidak ada lagi kata yang tersisa, maka Layout (susunan) yang optimal pada kasus ini adalah
```
Halo apa
kabar
dunia
```

Untuk memudahkan analisis, dibuatlah program dalam menyelesaikan contoh kasus word wrap.

Masalah yang diselesaikan program ini adalah bagaimana mencetak suatu teks paragraf pada layar secara rapi, dimana font yang digunakan adalah jenis fixed-length-font –seperti courier new. Masukan teksnya adalah berupa rangkaian sejumlah N kata (*word*) yang mengandung L1,L2,...,LN string (array of char) yang berurutan. Setiap baris pada layar dibatasi jumlah karakternya dengan '`page_width`'. Jika suatu baris mengandung kata-kata (*words*) dari i sampai j, maka banyaknya ruang kosong (*blanks*) yang tertinggal pada akhir baris adalah `Page_width-j+i-SUM{length(L[k]),k=1sampai j}`. Diasumsikan di antara dua kata yang berurutan selalu terdapat satu *blank*.

Tujuan kita adalah meminimumkan SUM dari kubik (perpangkatan tiga) dari ruang kosong pada akhir baris pada keseluruhan baris, kecuali pada baris terakhir. Sebagai contoh, jika terdapat *k* baris dengan $b_1, b_2, \ldots, b_k$ ruang kosong pada tiap akhir baris, maka kita akan meminimumkan
$(b_1)^3 + (b_2)^3 + \ldots + (b_{k-1})^3$.

Ide dari algoritma program ini adalah sebagai berikut:
Ketika kita akan menempatkan kata ke-i, secara umum kita mempunyai dua pilihan: menempatkan kata itu pada baris yang sekarang (*current line*) yang masih cukup memuat kata itu, atau membuat baris baru dan menempatkan kata itu pada awal baris baru. Perlu diperhatikan bahwa jika baris baru dibuat, maka susunan (*layout*) pada baris-baris sebelumnya dibuat tetap (*fixed*), dan cost-nya ditambahkan ke final cost, tidak peduli bagaimana kata-kata pada baris terakhir disusun.

Kompleksitas algoritma ini adalah $O(n^2)$, dimana n adalah banyaknya kata.

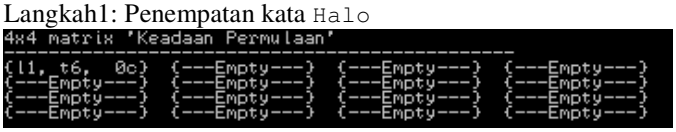
Program ini tidak menerima *input*. Teks dimasukkan secara *hardcode* pada *source* program.

Program menampilkan *output* berupa matriks yang menggambarkan langkah-langkah pengambilan keputusan. Setiap elemen matriks memberikan informasi:

1) Pada baris mana langkah tersebut dilakukan.
2) Jumlah spasi tersisa pada baris dimana langkah tersebut dilakukan.
3) Akumulasi ongkos mulai tahap paling awal hingga satu tahap sebelum tahap sekarang.

Screenshot program untuk kasus `Halo apa kabar dunia adalah` sebagai berikut:

Langkah1: Penempatan kata `Halo`



Gambar 1 Langkah 1

Elemen matriks[1,1] menggambarkan bahwa langkah 1 dilakukan pada baris 1, menyebabkan sisa spasi pada baris 1 tersebut menjadi 6, dan membutuhkan cost 0. Cost 0 karena teks yang masih muat di satu baris pertama dianggap sama dengan 'baris terakhir' dimana besar ongkos tidak berpengaruh pada layout.

Langkah 2: Penempatan kata `apa`

Gambar 2 Langkah 2

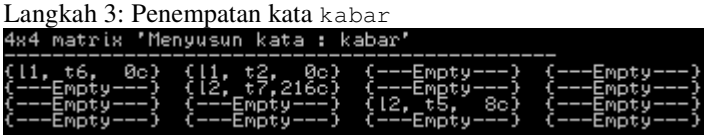
Elemen matriks[1,2] menggambarkan bahwa langkah kedua dilakukan pada baris 1, menyebabkan sisa spasi pada baris 1 tersebut menjadi 2, dan membutuhkan cost 0.
Hasil dari pilihan langkah ini adalah

```
Halo apa
```

Elemen matriks[2,2] menggambarkan keadaan jika langkah kedua dilakukan pada baris 2, maka akan menyebabkan sisa spasi pada baris 2 menjadi 7, dan cost menjadi pangkat tiga dari sisa spasi di baris 1 ($6^3 = 216$).
Hasil dari pilihan langkah ini adalah

```
Halo
apa
```

Langkah 3: Penempatan kata `kabar`



Gambar 3 Langkah 3 pilihan 1

Elemen matriks[3,3] menggambarkan pilihan langkah ketiga yang dilakukan jika langkah kedua yang dilakukan adalah yang diwakili elemen matriks[1,2]. Langkah ini dilakukan pada baris 2, menyebabkan sisa spasi pada baris 2 menjadi 5, dan cost menjadi pangkat tiga dari sisa spasi di baris 1 ($2^3 = 8$).
Hasil pilihan langkah ini adalah

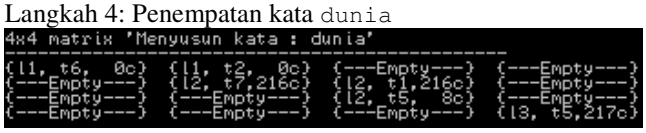
```
Halo apa
kabar
```

Gambar 4 Langkah 3 pilihan 2

Elemen matriks[3,2] menggambarkan pilihan langkah ketiga yang dilakukan jika langkah kedua yang dilakukan adalah yang diwakili elemen matriks[2,2]. Langkah ini dilakukan pada baris 2, menyebabkan sisa spasi pada baris 2 menjadi 1, dan cost menjadi pangkat tiga dari sisa spasi di baris 1 ($6^3 = 216$).
Hasil dari pilihan langkah ini adalah

```
Halo
apa kabar
```

Langkah 4: Penempatan kata `dunia`

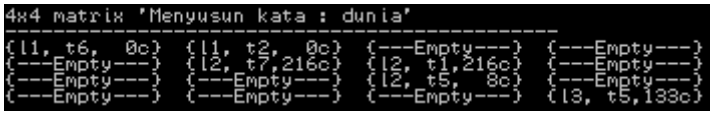


Gambar 5 Langkah 4 pilihan 1

Elemen matriks[4,4] di atas menggambarkan pilihan langkah keempat yang dilanjutkan dari pilihan langkah ketiga yang diwakili elemen matriks[3,2].

Elemen [4,4] menggambarkan bahwa lengkah ini dilakukan di baris 3, meyebabkan sisa spasi pada baris 3 menjadi 5, dan cost menjadi 217. 217 merupakan hasil penjumlahan cost seluruh baris selain baris terakhir ($6^3+1^3=217$).
Hasil dari pilihan langkah ini adalah

```
Halo
apa kabar
dunia
```



Gambar 6 Langkah 4 pilihan 2

Elemen matriks[4,4] di atas menggambarkan pilihan langkah keempat yang dilanjutkan dari pilihan langkah ketiga yang diwakili elemen matriks[3,3].

Elemen [4,4] menggambarkan bahwa lengkah ini dilakukan di baris 3, meyebabkan sisa spasi pada baris 3 menjadi 5, dan cost menjadi 133. 133 merupakan hasil penjumlahan cost seluruh baris selain baris terakhir ($2^3+5^3=133$).
Hasil dari pilihan langkah ini adalah

```
Halo apa
kabar
dunia
```

Hasil terakhir inilah yang dipilih sebagai hasil optimal, karena memiliki final cost yang lebih kecil (133) daripada pilihan sebelumnya (217).

4. KESIMPULAN

Pemrograman dinamis merupakan strategi pemecahan masalah yang sangat efektif untuk berbagai persoalan termasuk persoalan word wrap. Aspek kerapihan word wrap merupakan aspek yang dioptimasi dengan pemrograman dinamis. Dengan pemrograman dinamis, susunan kata-kata hasil word wrap terlihat lebih rapi dan nyaman dipandang.

REFERENSI

[1] Munir, Rinaldi. “Diktat Kuliah IF3051 Strategi Algoritma”. Institut Teknologi Bandung. 2009
[2]http://en.wikipedia.org/wiki/Word_wrap
[3]http://en.wikipedia.org/wiki/Dynamic_programming
[4]http://stackoverflow.com/questions/17586/best-word-wrap-algorithm
[5]http://www.catch22.net/tuts/neatpad/4
[6]http://okmij.org/ftp/packages/word_layout.cc