Penerapan Dynamic Programming dalam Word Segmentation

Sonny Theo Tumbur / 13510027
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13510027@std.stei.itb.ac.id

***Abstract*—dalam komunikasi verbal, kita mengenal adanya entitas kata, kalimat, paragraf, dll. Pada penerapannya dalam analisis komputasi, bagaimana komputer memproses kata dan kalimat tidaklah semudah yang terjadi pada manusia. Kata demi kata kata, kalimat demi kalimat akan melalui proses pemisahan antarentitas kata, kalimat dan lain-lain. Dalam proses segmentasi, ada banyak pendekatan algoritma yang dapat diterapkan, misalnya *bruteforce*, *greedy*, *decrease and conquer*, dll. Penulis akan melakukan analisis pada *word segmentation* dengan menggunakan pendekatan pemrograman dinamis (*dynamic programming*).**

***Index Terms*—*dynamic programming*, optimalisasi, *word segmentation*.**

I. PENDAHULUAN

Kumpulan kalimat tertulis (*written text*) manusia, baik dalam bahasa Indonesia, bahasa Inggris ataupun bahasa lainnya, seringkali sulit untuk diproses oleh sistem terkomputerisasi, yang salah satu penyebab terbesarnya adalah kegagalan proses segmentasi. Proses segmentasi (*segmentation process*) dapat digolongkan menjadi tiga golongan secara garis besar : *word segmentation*, *sentence segmentation*, dan *text segmentation*. Segmentasi kata (*word segmentation*) melakukan pembagian string menjadi komponen dalam satuan kata. Dalam bahasa Inggris, bahasa Indonesia, bahasa Latin, karakter spasi adalah karakter yang umum digunakan untuk membatasi antara kata yang satu dengan kata yang lain. Namun, ada juga bahasa yang tidak menggunakan karakter spasi sebagai pemisah antarkatanya seperti bahasa Cina, Jepang, dimana bukan kata yang mengalami pemisahan melainkan pemisahan diterapkan pada satuan kalimat. Hal-hal seperti ini diselesaikan dengan metode *word splitting*. Masalah berikutnya yang kita hadapi adalah pemisahan kalimat. Dalam bahasa Indonesia dan bahasa Inggris, pemisah antarkalimat yang digunakan adalah karakter titik "." (*full stop*). Meskipun demikian, tetap saja ada masalah yang tidak memenuhi aturan tersebut. Contohnya, karakter titik pada kalimat "*Mr. Smith went to the shops in Jones Street.*" Jelas tidak sama artinya dengan sebuah pemisah antarkalimat. Hal-hal seperti ini yang menjadi pertanyaan *sentence segmentation*. Dalam hal pengolahan teks (kumpulan kalimat yang relatif berukuran besar), permasalahan yang umum ditemui adalah identifikasi topik dan segmentasi teks (*text segmentation*). Segmentasi teks digunakan untuk mengatasi masalah-masalah seperti *text summatizing*, dan sejenisnya.

II. DASAR TEORI

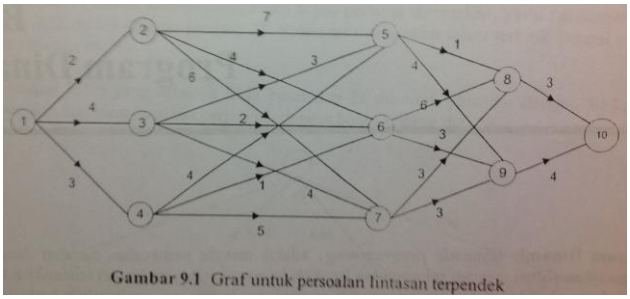
Dynamic Programming

Pemrograman dinamis merupakan salah satu teknis desain algoritma yang relatif baru dibandingkan teknis desain algoritma lainnya.Teknik ini pertama kali ditemukan oleh matematikan Amerika Serikat bernama Richard Bellman. Teknik pemrograman dinamis ini digunakan untuk memecahkan masalah dengan lebih dari satu submasalah yang saling bertindihan satu dengan yang lain. Pemrograman dinamis membagi suatu masalah ke dalam langkah-langkah kecil dan secara langsung melakukan penyelesaian pada tiap langkah tersebut.

Sebagian besar persoalan yang diselesaikan dengan pemrograman dinamis merupakan persoalan terkait proses optimalisasi, minimisasi, maksimalisasi, dan sejenisnya. Berhubungan dengan hal tersebut, Richard Bellman membuat suatu konsep yang dikenal dengan Prinsip Optimalitas (*Principle of Optimality*). Berikut adalah bunyi prinsip tersebut (Levitin: 284) :

> *"Optimal solution to any instance of an optimization problem is composed of optimal solutions to its subinstances."*

Berikut ini adalah salah satu contoh soal menemukan lintasan terpendek yang diselesaikan menggunakan prinsip dynamic programming. Pada persoalan ini, kita ingin menemukan lintasan terpendek dari simpul 1 menuju ke simpul 10.



Gambar 1

Mari kita coba mulai dari simpul 1, maka kemungkinan simpul tujuan berikutnya adalah 2, 3, dan 4. Menurut metode greedy, langkah berikutnya yang kita lakukan adalah memilih simpul 2, karena bobotnya yang paling kecil dibandingkan sisi lainnya. Namun, jika kita teruskan dengan cara ini, lintasan terpendek yang dihasilkan dari simpul 1 ke simpul 10 adalah sebesar 12. Akan tetapi, misalkan kita berkorban satu langkah, maka lintasan terpendek yang dihasilkan bisa mencapai 11.

Salah satu cara untuk menyelesaikan persoalan ini adalah dengan melakukan enumerasi semua rangkaian keputusan yang mungkin dan memilih keputusan yang terbaik. Cara ini kita kenal sebagai metode exhoustive search. Akan tetapi, dengan program dinamis kita dapat secara drastis mengurangi pengenumerasian rangkaian keputusan yang tidak mengarah ke solusi optimal.

Berikut ini adalah beberapa karakteristik persoalan yang dapat diselesaikan dengan program dinamis.

Persoalan dapat dibagi menjadi beberapa tahapan (stage), yang pada tiap tahapnya hanya diambil satu keputusan.

a. Masing-masing tahapan terdiri dari sejumlah status yang berhubungan dengan tahap tersebut. Secara umum, status merupakan bermacam kemungkinan masukan yang ada pada tahap tersebut. Jumlah status bisa berhingga atau tidak berhingga.
b. Hasil keputusan yang diambil pada setiap tahap ditransformasikan dari status yang bersangkutan ke status berikutnya pada tahap selanjutnya.
c. Ongkos pada suatu tahap meningkat secara teratur dengan bertambahnya jumlah tahapan.
d. Ongkos pada suatu tahap bergantung pada ongkos tahap-tahap yang sudah berjalan dan ongkos pada tahap sebelumnya.
e. Keputusan terbaik pada suatu tahap bersifat independen terhadap keputusan yang dilakukan pada tahap sebelumnya.
f. Adanya hubungan rekursif yang mengidentifikasikan keputusan terbaik untuk setiap status pada tahap *k* memberikan keputusan terbaik untuk setiap status pada tahap *k+1*.
g. Prinsip optimalitas berlaku pada persoalan tersebut.

III. ANALISIS IMPLEMENTASI

Pertama-tama, kita ketahui bahwa untuk sebuah string dengan *n* karakter, kita memiliki sebesar 2^{n-1} segmentasi yang bisa dibangun. Contohnya pada string "wordsegmentation" dengan garis vertikal sebagai pemisah antarhuruf di dalamnya, "word|seg|men|ta|tion". Jumlah maksimum dari garis vertikal yang mungkin ditempatkan pada string tersebut adalah *n-1*. Dengan begitu, kita dapat menghitung jumlah segmentasi yang mungkin dibangun dengan menggunakan bantuan keberadaan jumlah garis vertikal ini. Kita misalkan garis vertikal ini sebagai suatu deretan angka dalam bilangan biner, lalu deretan biner dengan jumlah *n-1* otomatis akan menghasilkan 2^{n-1} segmentasi yang valid. Pada dasarnya, kita akan melakukan pengujian terhadap tiap bagian dari segmentasi dengan berdasar pada kamus (*dictionary*) yang berisi data acuan kata-kata valid untuk bahasa tertentu. Misalnya string "lovenails", kita memiliki segmentasi berupa "love|snails", "loves|nails", dan lain sebagainya. Substring "love", "snails", "loves", dan "nails" kita anggap sebagai suatu kata tertentu, karena kata tersebut merupakan kata yang valid dalam bahasa Inggris. Namun demikian, kamus (*dictionary*) yang kita kenal secara umum belum cukup untuk menyelesaikan berbagai jenis masalah *word segmentation* ini. Contohnya, string www.bbcamerica.com, dalam kamus bahasa Inggris (*American English / British English*), tidak ditemukan kata dengan awalah "bb", dengan demikian terbukti bahwa kamus tidak bisa kita jadikan acuan satu-satunya dalam menentukan kevalidan suatu substring. Solusi yang dapat kita berikan yaitu membuat kamus tambahan mengenai segmentasi unik untuk memastikan ketepatan dalam menentukan kevalidan suatu segmentasi.

Dalam proses implementasinya, pertama kita buat fungsi 'splitPairs' yang menerima input string *s* dan keluarannya adalah seluruh pasangan split (*u*,*v*) dimana *s* = *uv*. Berikut adalah ilustrasi implementasinya

```
def splitPairs(word):
    return [(word[:i+1], word[i+1:]) for i in range(len(word)
```

Gambar 2

Untuk masukan string "hello", akan menjadi

```
>>> splitPairs("hello")
[('h', 'ello'), ('he', 'llo'), ('hel', 'lo'),
 ('hell', 'o'), ('hello', '')]
```

Gambar 3

Berikutnya kita akan membuat suatu fungsi "wordSeqFitness" yang mana melakukan perhitungan terhadap kata-kata untuk menentukan kebenaran dari segmentasi tersebut. Berikut adalah implementasinya :

```
def segment(word):
    if not word: return []
    allSegmentations = [[first] + segment(rest)
                        for (first, rest) in splitPairs(word)
    return max(allSegmentations, key = wordSeqFitness)
```

Gambar 4

Pada dasarnya yang kita lakukan adalah menginduksi berdasarkan panjangnya string yang akan disegmentasi. Misalnya kita telah mendapatkan segmentasi yang optimal untuk substring dimana hanya karakter pertama yang tidak termasuk ke dalam substring tersebut. Selanjutnya, tidaklah sulit untuk menetukan segmentasi yang optimal untuk keseluruhan string (termasuk karakter pertama dalam string). Dalam hal ini, kita menggunakan algoritma rekursif dalam melakukan induksi pada tiap tahapan masalah.

Masalah berikutnya yang kita hadapi adalah bagaimana menentukan nilai kebenaran / kevalidan dari suatu substring. Seperti yang telah diterangkan di atas, bahwa tidaklah mungkin bagi kita untuk hanya menggunakan kamus biasa dalam menyusun library untuk menentukan kevalidan suatu substring. Oleh karena itu, kita akan menggunakan *probabilistic model* dalam bahasa Inggris yang disediakan Google. Ada dua macam pemodelan utama yang disediakan Google. Model pertama menghitung frekuensi dari substring berdasarkan kata-kata yang diakses secara random dari internet. Model ini membentuk pola *n-grams*, deretan *n* tokens. Model kedua secara spesifik berdasar pada data yang di dapat dari buku-buku yang mengalami proses digitalisasi / *scanned*. Sama halnya dengan model pertama, kata-kata yang didapat juga disusun dalam bentuk *n-grams*. Namun, kedua model ini tetap memiliki celah kesalahan, misalnya untuk kata "lol" dan "LotR" yang umum digunakan di internet, kemungkinan kecil terdapat di dalam buku.

Berikut ini adalah beberapa baris data yang disusun oleh Peter Norvig, Profesor Stanford University, dalam bentuk *word*<*tab*>$_n$.

```
the     23135851162
of      13151942776
and     12997637966
to      12136980858
a       9081174698
in      8469404971
for     5933321709
is      4705743816
on      3750423199
that    3400031103
by      3350048871
this    3228469771
with    3183110675
i       3086225277
you     2996181025
```

Gambar 5

dan beberapa baris terakhir di antaranya :

```
goolld  12711
goolh   12711
goolgee 12711
googook 12711
googllr 12711
googlal 12711
googgoo 12711
googgol 12711
goofel  12711
gooek   12711
gooddg  12711
gooblle 12711
gollgo  12711
golgw   12711
```

Gambar 6

Implementasi

Probabilita Bayes yang digunakan pada model ini akan berujung pada pembentukan kelas-kelas. Seperti program *object-oriented* pada umumnya, kita akan melakukan input output file, melakukan organisasi *word counts*, dan lain sebagainya. Berikutnya kita akan menyusun suatu struktur *dictionary* seperti contoh di bawah ini:

```
class OneGramDist(dict):
```

Dengan keberadaan kamus ini, kita ke depannya dapat menganggap instantiasi dari kelas selayaknya sebuah kamus. Langkah selanjutnya, kita akan melakukan inisialisasi terlebih dahulu, lalu memanggil method yang akan mengeluarkan nilai probabilita dari kata masukan pengguna sebelumnya. Berikut ilustrasi implementasinya:

```
    def __init__(self):
        self.gramCount = 0
        for line in open('one-grams.txt'):
            (word, count) = line[:-1].split('\t')
            self[word] = int(count)
            self.gramCount += self[word]

    def __call__(self, word):
        if word in self:
            return float(self[word]) / self.gramCount
        else:
            return 1.0 / self.gramCount
```

Gambar 7

Pada proses inisialisasi, setelah melakukan input string, program akan melakukan perhitungan terhadap string masukan. Berikutnya, string masukan tersebut akan diasosiasikan dengan stuktur kamus/*dictionary* yang telah dibuat sebelumnya, lalu objek tersebut akan melalui proses perhitungan menggunakan variabel "*gramCount*". Lalu, untuk menggabungkan dengan kode segmentasi sebelumnya, kita akan melakukan implementasi untuk fungsi "wordSeqFitness" sebagai berikut:

```
singleWordProb = OneGramDist()
def wordSeqFitness(words):
    return functools.reduce(lambda x,y: x+y,
      (math.log10(singleWordProb(w)) for w in words))
```

Gambar 8

Implementasi di atas membutuhkan proses *import* library "math" dan "functools".

.

IV. KESIMPULAN

a. Persoalan *segmentation process*, seperti *word segmentation*, *sentence segmentation*, dan *text segmentation*, seluruhnya dapat diselesaikan dengan menggunakan program dinamis.
b. Program dinamis merupakan salah satu dari algoritma terbaik dalam memecahkan banyak persoalan, khusunya dalam menyelesaikan persoalan word segmentation.

REFERENSI

[1] Anany Levitin, *Introduction the Design and Analysis of Algorithm 3rd edition.* Pearson. June 2011.
[2] Jeremykun.wordpress.com/2012/01/15/word-segmentation/
[3] Munir, Rinaldi. "Dikat Kuliah IF3051 Strategi Algoritma". Institut Teknologi Bandung. 2009.

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, 16 Desember 2012

Ttd

Sonny Theo Tumbur / 13510027