APLIKASI PROGRAM DINAMIS DALAM ALGORITMA COCKE-YOUNGER-KASAMI DAN ALGORITMA NEEDLEMAN-WUNSCH

Marselina Tando

Program Studi Teknik Informatika
Sekolah TeknikElektro dan Informatika, Institut Teknologi bandung
Jalan Ganesha 10, Bandung
e-mail: if16100@students.if.itb.ac.id

ABSTRAK

Makalah ini akan membahas aplikasi program dinamis dalam beberapa algoritma, yaitu algoritma CYK dan algortma Needleman-Wunsch. Keduanya merupakan algoritma yang digunakan secara spesifik dalam bidang terapan khusus. CYK digunakan dalam tata bahasa bebas konteks, terutama dalam embentuk pohon *parser*, untuk mengenali dan menguraikan kalimat menjadi elemen-elemen yang lebih kecil. Needleman-Wunsch sendiri digunakan dalam bidang bioinformatika, yaitu dalam penguurutan protein dan nukleotida (*sequence analysis*). Program dinamis sendiri akan dibahas pada bagian pertama sebagai dasar pengenalan untuk algoritma-algoritma implementasinya. Persoalan optimasi yang berhasil dipecahkan dengan baik oleh program dinamis ini menjadi daya tarik tersendiri dalam penggunaannya. Program dinamis sendiri memilki beberpa kesamaan dengan algoritma *greedy*, hanya saja program dinamis mampu menghasilkan serangkaian keputusan yang lebih ternilai optimasinya dibandingkan dengan algoritma *greedy*, karena program dinamis menjadikan pengambilan keputusan yang satu dengan yang lainnya saling berkaitan.

Kata kunci: Program dinamis, Algoritma Needleman-Wunsch, Algoritma CYK.

1. PENDAHULUAN

Dalam ilmu matematika atau komputer, program dinamis merupakan sebuah metode untuk menyelesaikan masalah dengan menggunakan properti berupa submasalah overlapping dan substruktur optimal yang menggunakan waktu yang lebih singkat jika dibandingkan dengan metode lainnya.

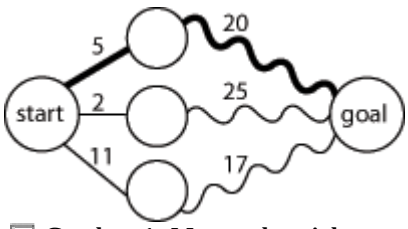
Istilah ini pertama kali digunakan pada tahun 1940-an oleh Richard Bellman untuk menggambarkan proses penyelesaian masalah di mana keputusan terbaik dibuat secara berurutan. Pada tahun 1953, ia menyempurnakan deskripsi tersebut menjadi lebih modern. Kontribusi Bellman dihargai dengan menggunakan istilah persamaan Bellman, sebuah hasil pokok program dinamis yang menyatakan kembali optimasi masalah dalam bentuk rekursif.

Pada saat ini, program dinamis telah diimplementasikan dalam algoritma-algoritma optimasi yang dekat dengan kehidupan kitasehari-hari. Optimasi masalah yang dilakukan dalam program dinamis ini memberikan solusi yang tidak hanya terbaik secara sekunsial saja (seperti pada algoritma *greedy*), tetapi juga secara keseluruhan.

2. PEMBAHASAN

2.1 Program Dinamis

Istilah “program ” dalam program dinamis ini sendiri tidak hanya diartikan sebagai program komputer, tetapi menunjuk kepada arti yang lebih luas, yaitu sebagai rencana optimal untuk aksi yang dilakukan. Dengan kata lain, menemukan rencana kegiatan optimal yang dapat diterima.



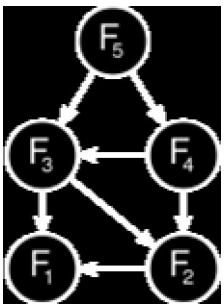
Gambar 1. Menemukan jalur terpendek dalam sebuah graf dengan menggunakan substruktur optimal; garis lurus menggambarkan ujung tunggal; garis berelomabng menunjukkan jalur terpendek antara 2 simpul yanng dihubungkan (simpul-simpul lain dalam gambar tidak diperlihatkan); garis tebal merupakan jalur total terpendek dari start ke goal/tujuan .

Substruktur optimal berarti bahwa solusi optimal dari submasalah dapat digunakan untuk menemukan solusi

optimal dari masalah secara global. Contohnya adalah jalur terpendek k simpul tujuan dari sebuah simpul dapat ditemukan pertama kali dengan mencari jalur terpendek ke simpul tujuan dari seluruh simpul yang bersisian, kemudian menggunakan hasil ini untuk memilih jalan terbaik, seperti ditunjukkan pada Gambar 1. Secara umum, kita dapat menyelesaikan masalah dengan substruktur optimal dengan menggunakan 3 tahap pemrosesan:

1. Pecahkan masalah menjadi submasalah yang lebih kecil.
2. Selesaikan maslah tersebuat secara optimal dengan menggunakan 3 tahapan ini secara rekursif.
3. Gunakan solusi-solusi optimal tersebut untuk membangun sebuah solusi optimal untuk masalah awal.

Submasalah sendiri diselesaikan dengan membaginya menjadi subsubmasalah, dan seterusnya, hingga kita sampai pada sebuah kasus sederhana yang dapat diselesaikan dalam waktu konstan.



Gambar 2. Graf submasalah untuk deret Fibonacci. Kenyataan bahwa gambar tersebut bukan merupakan sebuah pohon melainkan sebuah DAG menunjkkan submasalah yang mengalami kasus overlapping (beririsan).

Jika dkatakan bahwa sebuah masalah memiliki submasalah yang overlapping, berarti bahwa submasalah tersebut digunakan untuk menyelesaikan lebih banyaka masalah besar lainnya. Sebagai contoh, dalam deret Fibonacci, $F_3 = F_1 + F_2$ and $F_4 = F_2 + F_3$ — memroses tiap angka melibatkan pemrosesan F_2. Karena F_3 and F_4 dibutuhkan untuk memroses F_5, pendekatan yang sederhana dalam memroses F_5 bisa berakhir dengan memroses F_2 dua kali atau lebih. Hal ini berlaku setiap kali ada subproblem yang overlapping : pendekatan yang sederhana tadi dapat menghabiskan waktu untuk memroses kembali solusi optimal ke submasalah yang telah diselesaikan.

Untuk mencegah terjadinya hal tersebut, kta menyimpan solusi dai masalah yang telah kita selesaikan sebelumnya.Kemudian, jika kita harus menyelesaikan masalah yang sama di kemuadian waktu, kita dapat mengambil dan menggunakan kembali solusi yang telah disimpan tersebut. Pendekatan ini disebut memoisasi (bukan memorisasi, walaupun istilah ini juga cocok dengan konteks yang dimaksudkan). Jika kita yakin tidak akan menggunakan solusi tertentu lagi, kita dapat menyimpannya ke temapat lain. Dalam beberapa kasus, kita bahkan dapat memroses solusi dari submasalah yang kita tahu akan kita butuhkan di kemudian hari.

Program dinamis menggunakan salah satu dari 2 pendekatan di bawah ini.

1. Pendekatan maju (*top-down*): Masalah dipecah menjadi submasalah. Submasalah ini diselesaikan dan solusinya disimpan untuk menjaga kemungkinan adanya kasus serupa yang perlu diselesaikan. Proses rekursi dan memoisasi dikombinasikan bersama.
2. Pendekatan mundur (*bottom-up*): Seluruh submasalah yang mungkin dibutuhkan diselesaikan dan kemudian digunakan untuk membangun solusi dari masalah yang lebih besar. Pendekatan ini sedikit lebih baik dalam hal stack space dan jumlah fungsi yang dipanggil, tapi terkadang tidak intuitif untuk memikirkan semua submasalah yang dibutuhkan untuk menyelesaikan masalah yang diberikan.

2.2 Algoritma CYK

Algoritma Cocke-Younger-Kasami (CYK)**,** yang juga dikenal dengan istilah CKY menentukan apakah sebuah string dapat dibentuk dari sebuah tata bahasa bebas konteks yang diberikan, dan jika dapat, menentukan bagaimana string tersebut dapat dibentuk. Hal ini dikenal sebagai proses parsing string. Algoritma ini merupakan sebuah contoh dari program dinamis.

Versi standar dari CYK mengenali bahasa yang didefinisikan oleh tata bahasa bebas konteks yang dituliskan dalam bentuk Chomsky normal (*Chomsky normal form*- CNF). Karena sembarang tata bahasa bebas konteks daat dikonversikan ke bentuk CNF tanpa memerlukan banyak kesulitan, CYK dapat digunakan untuk mengenali sembarang bahasa bebas konteks. CYK juga memungkinkan untuk memperluas algoritmanya sendiri dan menggunakannya untuk menangani beberapa tata bahasa bebas konteks yang tidak dituliskan dalam CNF; hal ini bertujuan untuk meningkatkan performansi, walaupun membuat algoritma ini sendiri menjadi sulit untuk dipahami secara sekilas.

Kasus kompleksitas asimptotik terburuk dari CYK adalah $\underline{\Theta}(n^3)$, di mana *n* merupakan panjang dari string yang di-parse. Hal ini menguatkan algoritma ini sebagai algiritma paling efisien dalam mengenali sembarang tata bahasa bebas konteks. Bagaimanapun, ada pula algoritma lain yang akan menunjukkan performansi yang lebih baik untuk beberapa bahasa bebas konteks tertentu.

Algoritma CYK merupakan sebuah algoritma program dinamis yang bergerak mundur dan hal ini secara teoretis sangat penting, karena dapat digunakan untuk membuktikan secara konstriktif bahwa masalah keanggotaan datam bahasa bebas konteks bersifat *decidable* (dapat diputuskan)

Algoritma CYK untuk masalah keanggotaan adalah sebagai berikut.

```
input string consist of n letters, a1
... an.
grammar contain r terminal and
nonterminal symbols R1 ... Rr.
This grammar contains the subset Rs
which is the set of start symbols.
P[n,n,r] be an array of booleans.
Initialize all elements of P to false.
For each i = 1 to n
  For each unit production Rj -> ai,
set P[i,1,j] = true.
For each i = 2 to n -- Panjang of span
  For each j = 1 to n-i+1 -- Start of
span
    For each k = 1 to i-1 -- Partition
of span
      For each production RA -> RB RC
        If P[j,k,B] and P[j+k,i-k,C]
then set P[j,i,A] = true
If any of P[1,n,x] is true (x is
iterated over the set s, where s are
all the indices for Rs)
  Then string is member of language
  Else string is not member of language
```

Secara informal, algoritma ini menganggap setiap subderet yang mungkin dari sederatan kata dan himpunan P[i,j,k] bernilai benar jika subdere dari kata yang dimulai dari i dari panjang j dapat dibentuk dari R_k. Setelah algoritma ini menyelesaikan subderet dengan panjang 1, algoritma melanjutkan dengan subderet dengan panjang 2, an seterusnya. Untuk subderet dengan panjang 2 atau lebih, algoritma mengasumsikan tiap partisi yang mungkin dari subderet menjadi 2 bagian, kemudian memeriksa apakah ada produksi $P \rightarrow Q\ R$ di mana Q cocok dengan bagian pertama dan apakah R cocok dengan bagian kedua.Jika demikan, algoritma megeluarkan hasil dan menyimpan bahwa P cocok dengan seluruh subderet. Pada saat proses ini selesai, kalimat dikenali oleg tata bahasa jika subderet yang mengandung keseluruhan kalimat cocok dengan simbol mulai.

S						
	VP					
S						
	VP			PP		
S		NP			NP	
NP	V, VP	Det.	N	P	Det	N
She	eats	A	fish	with	A	Fork

Tabel 1 CYK

Merupakan hal yang sangan sederhana untuk memperluas algoritma di atas untuk tidak hanya menentukan apakah sebuah kalimat termasuk dalam sebuah bahasa, tapi juga untuk membangun sebuah pohon parse , dengan menyimpan simpul pohon parse sebagai elemen larik daripada menyimpannya sebagai boolean. Karena tata bahasa yang dikenali dapat menjadi ambigu, list simpul perlu disimpan (kecuali jika ada yang hanya ingin memilih salah satu pohon parse yang mungkin); hasil akhirnya adalah timbulnya hutan atau kumpulan yang terdiri dari pohon-pohon parse. Formulasi lain menggunakan tabel kedua B[n,n,r] ang dikenal dengan *backpointers*.

Algoritma CYK juga mungkin diperluas untuk melakukan parse pada string dengan menggunakan tata bahasa bebas konteks stokastik dan berbobot. Bobot (kemungkinan) kemudian disimpan dalam tabel P, sehingga P[i,j,A] akan mengandung bobot minimum dan (kemugkinan maksimum) substring dari i ke j dapat diturunkan dari A. Perluasan lebih jauh dari algoritma memugkinkan seluruh parse string diurutkan dari bobot terendah ke bobot tertinggi (kemungkinan tertinggi ke kemungkinan terendah).

2.3 Algoritma Needleman-Wunsch

Algoritma Needleman–Wunsch memberikan hasil berupa sebuah susunan global dari 2 deret (dalam makalah ini disebut A and B). Algoritma ini secara umum digunakan dalam bioinformatika untuk menyusun deret protein dan nukleotida. Algoritma ini pertama kali diperkenalkan oleh Saul Needleman dan Christian Wunsch pada tahun 1970

dalam makalah mereka yang berjudul *A general method applicable to the search for similarities in the amino acid sequence of two proteins*, J Mol Biol. **48**(3):443-53. Algoritma ini merupakan contoh lain program dinamis, dan merupakan aplikasi pertama program dinamis dalam perbandingan deret/rangkaian biologis.

Nilai untuk karakter yang disusun dalam sebuah rangkaian dispesifikasikan dalam sebuah matriks kesamaan. Dalam konteks ini, $S(i,j)$ merupakan kesamaan antara karakter i dan j. Matriks ini juga menggunakan penatli gap yang linear, yang disebut d.

Sebagai contoh, jika matriks kesamaannya adalah sebagai berikut.

-	**A**	**G**	**C**	**T**
A	10	-1	-3	-4
G	-1	7	-5	-3
C	-3	-5	9	0
T	-4	-3	0	8

Maka konfigurasinya adalah sebagai berikut:

```
AGACTAGTTAC
CGA---GACGT
```

Dengan penalti gap -5, akan memiliki nilai sebagai berikut.

$$S(A,C)+S(G,G)+S(A,A)+3\times d+S(G,G)+S(T,A)+S(T,C)+S(A,G)+S(C,T)$$

$$= -3+7+10-3\times 5+7+-4+0+-1+0=1$$

menemukan penempatan dengan nilai tertinggi, larik 2 dimensi (atau matriks) dialokasikan. Matriks ini biasa disbut matriks F, dan entri ke (i,j) dilambangkan dengan F_{ij} Ada 1 kolom untuk masing-masing karakter dala deret A, dan 1 baris untuk tiap karakter dalam deret B. Jika kita menyusun deret dengan ukuran n dan m, waktu kerja algoritma adalah O(nm) dan jumlah memori yng digunakan adalah O(nm). (Bagaimanapun, ada versi modifikasi dari algoritma ini yang hanya menggnakan ruang O(m + n), tapi berefek pada penggunaan waktu yang lebih panjang. Modifikasi ini sebenarnya merupakan teknik umum yang telah diaplikasikan pada banyak algoritma program dinamis; etode ini pertama kali diperkenalkan dalam algoritma Hirschberg's untuk menyelesaikan masalah subderet umum terpanjang.

Sementara algoritma diproses, F_{ij} akan diisi dengan nilai optimal dari konfigurasi dari karakter pertama dalam A dan karakter j pertama dalam B. Prinsip optimalisasi kemudian diapliksikan sebagai berikut.

```
  Basis:
  F0j = d * j
  Fi0 = d * i
  Rekursi, berdasarkan pada prinsip
optimalisasi:
  Fij = max(Fi-1,j-1 + S(Ai,Bj),Fi,j-1 +
d,Fi-1,j + d)
```

Pseudo-code algoritma untuk compute matriks F adalah sebagai berikut (indeks larik dan deretdimulai dari 0).

```
  for i=0 to panjang(A)-1
    F(i,0) <- d*i
  for j=0 to panjang(B)-1
    F(0,j) <- d*j
  for i=1 to panjang(A)
    for j = 1 to panjang(B)
    {
      Pilihan1 <- F(i-1,j-1) + S(A(i),
B(j))
      Pilihan2 <- F(i-1, j) + d
      Pilihan3 <- F(i, j-1) + d
      F(i,j) <- max(Pilihan1, Pilihan2,
Pilihan3)
    }
```

Pada saat matrik F diproses, ingatlah bahwa sudut kanan bawah dari matriks berisi nilai maksimum untuk sembarang konfigurasi. Untuk memroses konfigurasi yang sebenarnya memberikan nilai ini, Anda dapat memulai dari sel kanan bawah, dan membandingkan nilainya dengan 3 kemungkinan sumber (Pilihan1, Pilihan2, dan Pilihan3 di atas) untuk melihat darimana nilai tersebut sebenarnya berasal. Jika Pilihan1, maka A(i) dan B(i) telah tersusun, jika Pilihan2, maka A(i) telah tersusun dengan sebuah gap, dan jika Pilihan3, maka B(i) telah tersusun dengan sebuah gap.

```
  KonfigurasiA <- ""
  KonfigurasiB <- ""
  i <- panjang(A)
  j <- panjang(B)
  while (i > 0 AND j > 0)
  {
    Nilai <- F(i,j)
    NilaiDiag <- F(i - 1, j - 1)
    NilaiUp <- F(i, j - 1)
    NilaiLeft <- F(i - 1, j)
```

```
    if (Nilai == NilaiDiag + S(A(i),
B(j)))
    {
      KonfigurasiA <- A(i-1) +
KonfigurasiA
      KonfigurasiB <- B(j-1) +
KonfigurasiB
      i <- i - 1
      j <- j - 1
    }
    else if (Nilai == NilaiLeft + d)
    {
      KonfigurasiA <- A(i-1) +
KonfigurasiA
      KonfigurasiB <- "-" +
KonfigurasiB
      i <- i - 1
    }
    otherwise (Nilai == NilaiUp + d)
    {
      KonfigurasiA <- "-" +
KonfigurasiA
      KonfigurasiB <- B(j-1) +
KonfigurasiB
      j <- j - 1
    }
  }
  while (i > 0)
  {
    KonfigurasiA <- A(i-1) +
KonfigurasiA
    KonfigurasiB <- "-" + KonfigurasiB
    i <- i - 1
  }
  while (j > 0)
  {
    KonfigurasiA <- "-" + KonfigurasiA
    KonfigurasiB <- B(j-1) +
KonfigurasiB
    j <- j - 1
  }
```

III. KESIMPULAN

Setelah melakukan kajian mengenai topik ini, dapat disimpulkan bahwa program dinamis merupakan sebuah metode yang sangat mangkus dan sangkil dalam memecahkan persoalan-persoalan optimasi yang banyak ditemui pada saat ini. Metode pencarian solusi yang saling berkaitan satu sama lain menjadi kunci keberhasilan metode ini dalam menemukan serangkaian keputusan yang benar-benar optimum.

REFERENSI

[1] Rinaldi Munir, "Diktat Kuliah Strategi Algoritmik", Program Studi Teknik Inofrmatika ITB, 2007.
[2] http://en.wikipedia.org/wiki/Dynamic programming, diakses pada tanggal 19 Mei 2008 pukul 14:00.
[2] http://en.wikipedia.org/wiki/CYK, diakses pada tanggal 19 Mei 2008 pukul 15:00.
[2] http://en.wikipedia.org/wiki/Needleman-Wunsch, diakses pada tanggal 20 Mei 2008 pukul 16:00.