

Aplikasi Pemrograman Dinamis Dalam Bioinformatika

Arief Wicaksono

awicaksono@dnet.net.id

Lisensi Dokumen:

Copyright ©2003-2006 IlmuKomputer.Com

Seluruh dokumen di IlmuKomputer.Com dapat digunakan, dimodifikasi dan disebarkan secara bebas untuk tujuan bukan komersial (nonprofit), dengan syarat tidak menghapus atau merubah atribut penulis dan pernyataan copyright yang disertakan dalam setiap dokumen. Tidak diperbolehkan melakukan penulisan ulang, kecuali mendapatkan ijin terlebih dahulu dari IlmuKomputer.Com.

Abstrak:

Pemrograman dinamis (*dynamic programming*) merupakan salah satu metode penyelesaian masalah yang terkenal, aplikasinya sangat luas, seperti dalam bidang, dalam bidang *bioinformatika*, pemrograman dinamis dapat diterapkan dalam lingkup *sequence alignment*, yaitu penyusunan dua untai atau lebih sekuen untuk melihat adanya suatu kesamaan tertentu. Dalam tulisan ini akan diketengahkan mengenai teori pemrograman dinamis, juga aplikasi dan kegunaannya di dalam *global sequence alignment* yang menggunakan algoritma Needleman Wunsch

Keywords: Pemrograman dinamis, optimalisasi, *sequence alignment*, *global sequence alignment*, Needleman-Wunsch

1. PENDAHULUAN

Istilah pemrograman dinamis (*dynamic programming*) tentunya pernah kita dengar, secara literatur, istilah ini banyak kita temui pada buku-buku algoritma komputer, riset operasional, ataupun buku yang berhubungan dengan matematika terapan.

Sebagian orang juga mengenal Pemrograman dinamis sebagai suatu metode yang berkaitan dengan prinsip optimalisasi dan dapat diaplikasikan pada bidang industri, perbankan, hingga perencanaan *network* dan aplikasi perjalanan luar angkasa.

Aplikasinya yang luas menjadikan pembahasan mengenai pemrograman dinamis menjadi suatu topik yang umum dan menarik.

Untuk mencoba mengetahui lebih lanjut tentang Pemrograman dinamis, pada artikel ini akan dibahas tentang dasar-dasar teori pemrograman dinamis secara singkat, juga dengan sejarah singkat dari asal mula istilah pemrograman dinamis.

Bioinformatika, sebagai bidang multidisiplin yang berkembang dan banyak menerapkan metode komputasi, tentunya menjadi bidang yang cukup terbuka terhadap suatu metode seperti metode pemrograman dinamis, lebih lanjut pada tulisan ini diterapkan aplikasi pemrograman dinamis sebagai metode optimalisasi dalam pemecahan masalah, dan dicontohkan pada masalah bilangan Fibonacci dan graf multistap, pada akhirnya, sebagai

penekanan dalam tulisan ini, adalah aplikasi pemrograman dinamis dalam bidang *bioinformatika*, khususnya pada algoritma Needleman-Wunsch untuk *global sequence alignment*.

2. PEMROGRAMAN DINAMIS

Istilah Pemrograman Dinamis [1], pertama kali diperkenalkan pada era 1950-an oleh Richard Bellman seorang professor di universitas Princeton dan juga bekerja pada RAND corporation, perlu diketahui bahwa RAND corporation pada era itu merupakan suatu perusahaan yang dibentuk untuk menawarkan analisis dan riset untuk angkatan bersenjata Amerika Serikat [2].

Pada saat itu, Bellman bekerja di bidang pengambilan keputusan multi tahap (*multistage desicion process*) dan mengerjakan beberapa metode matematis, beberapa tahun kemudian setelah Bellman berada di RAND, lahirlah istilah Pemrograman dinamis. Istilah ini tidak secara langsung berhubungan dengan pemrograman, melainkan digunakan sebagai judul project yang kemudian yang diusulkan RAND coorporation pada Angkatan Udara Amerika Serikat. Selanjutnya, pada penerapannya pemrograman dinamis banyak digunakan pada proses optimalisasi masalah.

Untuk mengartikannya, pemrograman dinamis dapat didefinisikan sebagai disain algoritma yang dapat digunakan, apabila dalam pencarian solusi untuk suatu problem matematis, langkah-langkahnya dapat dilihat dan diselesaikan, sebagai suatu urutan bertahap dari penyelesaian problem yang lebih kecil (sub problem) ke problem yang lebih besar..

Pemrograman dinamis mempunyai tiga komponen utama yaitu :

1. Solusi dari masalah pemrograman dinamis dapat diformulasikan secara rekursif.
2. Terdapat tahap dimana pencarian solusi dilakukan pada suatu subproblem sebelum pencarian solusi dilakukan pada problem yang lebih besar

3. penyimpanan hasil dari pencarian solusi subproblem dapat digunakan untuk mencari solusi dari problem yang lebih besar.

Untuk dapat memahaminya, lebih lanjut pemrograman dinamis dapat diaplikasikan pada pencarian barisan Fibonnaci dan pencarian jalur terpendek dalam graf multistage (multistage graph)

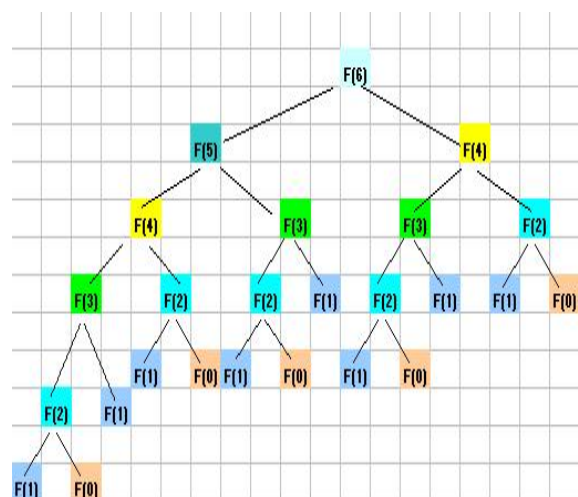
Barisan Fibonacci

Barisan Fibonacci merupakan salah satu barisan yang terkenal, untuk Barisan fibonacci ke n , dimana $n = 2, 3, 4, \dots$ didefinisikan sebagai

$$F(n) = F(n-1) + F(n-2)$$

dengan $F(0) = 0$ dan $F(1) = 1$.

Untuk mencari nilai n tertentu, misalnya untuk $n = 6$, nilai $F(6)$ dapat dibentuk diagram seperti pada gambar dibawah.



Gambar 1: penjabaran barisan Fibonacci untuk nilai $n=6$

Yang berada di paling atas pada gambar 1 adalah nilai $F(6)$, dan yang paling bawah adalah $F(0)$ dan $F(1)$, dimana kedua nilai ini diketahui, sehingga $F(6)$ dapat dijabarkan sebagai :

$$\begin{aligned} F(6) = & F(1) + F(0) + F(1) + F(1) + F(0) + \\ & F(1) + F(0) + F(1) + F(1) + F(0) + \\ & F(1) + F(1) + F(0) \end{aligned}$$

Dari Penjabaran ini, pencarian nilai $F(6)$ menjadi cukup sulit. Cara lain untuk memecahkan masalah ini adalah dengan

menjabarkan $F(6)$ menjadi beberapa persamaan seperti :

$$\begin{aligned} F(2) &= F(1) + F(0) \\ F(3) &= F(2) + F(1) \\ F(4) &= F(3) + F(2) \\ F(5) &= F(4) + F(3) \\ F(6) &= F(5) + F(4) \end{aligned}$$

Atau dengan kata lain untuk mencari nilai $F(6)$ diperlukan mencari nilai sub problem terlebih dahulu, dalam masalah ini sub problemnya adalah nilai $F(1)$ dan $F(0)$, selanjutnya nilai $F(n)$ dicari untuk nilai n yang semakin besar hingga $n = 6$. Pemecahan masalah dengan metode ini merupakan prinsip dalam pemrograman dinamis

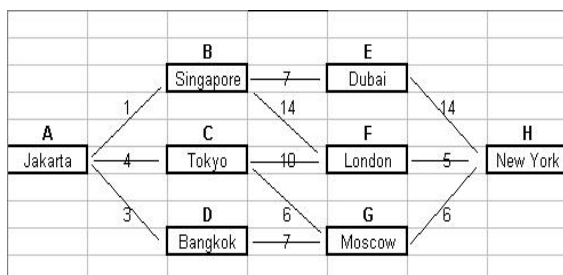
Dalam Bentuk pseudo code algoritma ini dapat dituliskan sebagai

```
F(0)=0, F(1)=1
FOR I = 2 TO 6
  F(I)=F(I-1)+F(I-2)
NEXT I
```

Denagn cara ini pencarian nilai $F(6)$ dapat diselesaikan secara bertahap dan terstruktur.

Jalur Terpendek dalam Graf Bertahap

Pemrograman dinamis juga dapat diaplikasikan pada masalah graf multi tahap (*multi stage graph*), sebagai contoh gambar 2 yang menggambarkan jarak tempuh penerbangan di beberapa kota dunia. Tiap vertek yang menghubungkan dua titik di Gambar(2) merepresentasikan panjangnya jarak tempuh antar dua kota .



Gambar 2 : Kasus multistage graph

Dimisalkan terdapat persoalan untuk mencari jarak tempuh terpendek antara Jakarta ke New

York, lewat manakah dan berapa jaraknya ?, kita dapat menjawab persoalan ini dengan menghitung setiap kemungkinan melewati jalur-jalur yang ada, dan kemudian mencari jarak yang paling minimum, untuk pekerjaan ini. Tetapi kembali lagi masalah yang timbul adalah perlunya mengkombinasikan semua jalur untuk dapat menghasilkan jawaban.

Dengan menggunakan pemrograman dinamis penyelesaian masalahnya dapat dibentuk agak lain, pada awalnya untuk gambar 2 dapat dibentuk persamaan berikut

$$R(A-H) = \text{Min} \{ 1 + R(B-H), 4 + R(C-H), 3 + R(D-H) \}$$

$R(I-J)$ adalah jarak dari I ke J , untuk dapat menghitung persamaan diatas perlu diketahui nilai $R(B-H)$, $R(C-H)$, dan $R(D-H)$.

Kembali lagi, kita dihadapkan pada pencarian subproblem, dalam masalah ini subproblem adalah $R(E-H)$, $R(F-H)$ dan $R(G-H)$,

Dapat dicari lebih lanjut :

$$\begin{aligned} R(B-H) &= \text{Min} \{ 7 + R(E-H), 14 + R(F-H) \} \\ &= \text{Min} \{ 7 + 14, 14 + 5 \} \\ &= 19 \dots\dots \text{Jalur F-H} \end{aligned}$$

$$\begin{aligned} R(C-H) &= \text{Min} \{ 10 + R(F-H), 6 + R(G-H) \} \\ &= \text{Min} \{ 10 + 5, 6 + 6 \} \\ &= 12 \dots\dots \text{Jalur G-H} \end{aligned}$$

$$\begin{aligned} R(D-H) &= \text{Min} \{ 7 + R(G-H) \} \\ &= \text{Min} \{ 7 + 6 \} \\ &= 13 \dots\dots \text{Jalur G-H} \end{aligned}$$

Dan Jarak terpendek antara Jakarta – New York adalah

$$\begin{aligned} R(A-H) &= \text{Min} \{ 1+19, 4+12, 3+13 \} \\ &= 16 \dots\dots \text{Jalur C-H dan D-H} \end{aligned}$$

Untuk mengetahui Jalur mana saja yang ditempuh dari Jakarta menuju New York, maka setiap perhitungan diperlukan pencatatan untuk mengingat jalur manakah yang dilewati, yaitu pada nilai-nilai yang minimal,

Untuk melewati jalur A-H, dari perhitungan, jarak terpendeknya adalah melewati C-H atau D-H, dimana kedua jalur ini memiliki jarak yang sama yaitu enam belas, Untuk Melewati C-H jalur terpendeknya adalah G-H, dan untuk D-H jalur terpendeknya hanyalah satu yaitu melalui G-H, sehingga Jalur Terpendek dari A ke H dapat direkonstruksi menjadi A-C-G-H atau A-D-G-H.

Pemrograman Dinamis dan Prinsip Optimalitas

Suatu properti yang dimiliki oleh pemrograman dinamis disebut dengan prinsip optimalitas (*principles of optimality*) [3]

,yaitu: apabila di dalam pemecahan masalah terdapat langkah $D_1, D_2, D_3, \dots, D_n$, dan langkah ini optimal, maka, k keputusan terakhir, dimana $1 < k < n$, adalah suatu keputusan yang juga optimal.

Atau dalam penerapannya dalam masalah pencarian jalur terpendek dari i ke j , dapat dikatakan, apabila $i, R_1, R_2, R_3, \dots, j$ adalah jalur terpendek dari i ke j , maka R_1, R_2, R_3, \dots, j adalah juga jalur terpendek dari R_1 ke j .

3. APLIKASI DALAM BIOINFORMATIKA

Dalam *bioinformatika*, analisis sekuen DNA merupakan suatu tahap yang sering dilakukan. Sekuen DNA yang terdiri dari kombinasi basa ATCG, dapat dikomparasi dengan sekuen lain. Pengkomparasian ini dikenal dengan istilah *sequence alignment*, secara garis besar *alignment* ini juga dapat dilakukan untuk protein.

Sequence Alignment

Dikenal tiga tipe dari *sequence alignment* [4]. Yaitu *Pairwise Alignment* yang terdiri dari dua aplikasi: *global alignment* dimana semua karakter dalam suatu sekuen dibandingkan, dan *local alignment* yang membandingkan suatu subset dari sekuen saja, keduanya baik *local alignment* maupun *global alignment* memakai suatu metode tertentu sehingga terjadi kemiripan

terbaik antara sekuen dengan sekuen lain yang ditujukan untuk melihat homologi suatu sekuen. Hal ini berguna untuk mengidentifikasi sekuen dari struktur yang belum diketahui atau untuk penyelidikan evolusi, yaitu penyelidikan apakah suatu sekuen berasal dari keturunan yang sama.

Yang kedua adalah *structural alignment* yang digunakan pada protein dengan berdasarkan dari lipatannya (secara 3D), dan ketiga yaitu *multiple Alignment* yang mengkomparasi lebih dari dua sekuen.

Pada Gambar 8, terdapat contoh *alignment* antara dua sekuen, yang menandakan *gap* (_), *match* (|) dan *misMatch* (*). Pertanyaannya adalah bagaimanakah kita dapat menentukan *alignment* sehingga yang terjadi adalah konfigurasi yang optimal?, untuk *global alignment* dikenal algoritma Needleman Wunsch, untuk *local* dikenal Algoritma Smith Waterman [5] dan BLAST (Basic Local Alignment Search Tool).

t	c	c	t	c	g	c	c	t	c	g	c	c	a	t	c	a	t	-	-	-	c	a	a	c	c	c	c	a	a	a	g	t	
t	c	c	t	c	g	c	a	t	c	t	g	c	a	a	t	c	a	t	g	g	g	c	a	a	c	c	c	c	a	a	a	g	t

Gambar 8: contoh *alignment*

Pada *pairwise alignment* antara sekuen X dan Y, komparasi dua sekuen akan menghasilkan hubungan seperti: string di X akan ter-align dengan string di Y dengan hubungan cocok (*match*) atau tidak cocok (*mismatch*), string di X akan ter-align dengan *gap* di Y, atau string di Y akan ter-align dengan *gap* di X. tentunya akan terdapat beberapa kemungkinan *alignment*. Tentunya yang paling baik adalah yang menghasilkan skor yang tertinggi atau secara logika lebih banyak *match* dari pada *mismatch* dan lebih sedikit *gap*.

Untuk mempersempit topik, dalam tulisan ini, hanya dibahas tentang *pairwise alignment* pada *global sequence alignment*.

Algoritma Needleman Wunsch

Untuk *global alignment*, misalkan sekuen $X=\{ATCGAGACA\}$ (10 String) dan sekuen $Y=\{TACAG\}$ (10 String). Apabila aturan Penilaian untuk *alignment* adalah (+1) apabila kedua string *match*, (0) apabila kedua string *mismatch* dan (-1) apabila salah satu string ter-align dengan *gap*, maka *alignment* untuk kedua sekuen adalah

```

AGACTGGAC-T
*|*|**|||
CG-GTCCACAT

```

Atau,

```

AGACTGGAC-T
*|*|**|||
CGG-TCACAT

```

Terlihat diatas, terjadi *match*, *mismatch* dan string yang ter-align dengan *gap*. Tanda bintang menyatakan kedua sting tidak-match (*mismatch*), garis vertikal apabila kedua string *match* dan garis horizontal yang merupakan *gap*. Skor *alignment* untuk (gambar 4) dan (gambar 5) adalah $(+1)(5) + (0)(4) + (-1)(2) = 3$

Untuk menghasilkan konfigurasi *alignment* dengan skor tertinggi, maka diperlukan suatu metode yang dapat menentukan konfigurasi yang optimal. Pada tahun 1970, Needleman dan Wunsch mengerjakan aplikasi penggunaan pemrograman dinamis untuk memecahkan *alignment* pada protein [5], tentunya hal ini dapat juga diaplikasikan pada sekuen DNA.

Idenya adalah dengan membentuk Matrik kesamaan (*similarity Matrix*) dimana komparasi antar string dilakukan, dimisalkan untuk sekuen $X=\{T,A,T,A,G\}$ dan sekuen $Y=\{T,A,C,A\}$. dilakukan dengan mengkomparasi problem terkecil yaitu T dengan T, TA dengan T, menuju ke problem terbesar yaitu melakukan *alignment* TATAG dengan TACA.

Untuk merepresentasikan hal ini maka dibentuklah Matrik *Mat* yang berdimensi $Mat[panjang\ sekuen\ X + 1][panjang\ sekuen\ Y + 1]$.

		j=0	j=1	j=2	j=3	j=4
			T	A	C	A
i=0						
i=1	T					
i=2	A					
i=3	T					
i=4	A					
i=5	G					

Gambar 6: Inisialisasi $Mat(i,j)$

Isi dari $Mat(i,j)$ adalah skor yang optimal dari alignmet antara X_1, \dots, X_i dengan Y_1, \dots, Y_j , sedangkan $Mat(0,0)$ adalah alignmet antara " " dengan " " atau dengan kata lain, komparasi antara string kosong dengan string kosong, sebagai pengecualian bernilai 0 ($Mat(0,0)=0$). Selanjutnya dapat dilakukan pengisian Matrik, sebagai contoh:

- $Mat(0,1)$ adalah align " " dengan "T" sehingga bernilai -1
- $Mat(0,2)$ adalah align " " with "TA" sehingga bernilai -2
- $Mat(1,0)$ adalah align "T" with " " sehingga bernilai -1
- $Mat(2,0)$ adalah align "TA" with " " sehingga bernilai -2

Selanjutnya kita akan mengisi nilai $Mat(1,1)$, yaitu skor komparasi T dengan T, untuk hal ini diperlukan pencarian sub problem. Dimanakah sub problem dari masalah ini?

Untuk komparasi T dengan T, dengan memperhitungkan nilai komparasi *gap* (_) dengan *gap*, terdapat tiga kandidat yaitu

1	Skor	-	+	nilai match / notmatch	T
		-			T
2	Skor	-	+	nilai alignment dengan gap	T
		T			-
3	Skor	T	+	nilai alignment dengan gap	-
		-			T

ketiga nilai ini, tidak lain adalah tiga nilai Matrik yang telah diketahui sebelumnya yaitu di kotak

atas $Mat(i,j)$ atau $Mat(i-1,j)$, kotak kiri atau $Mat(i,j-1)$ dan kotak diagonal atas atau $Mat(i-1,j-1)$ sehingga hubungan antar nilai Matrik dapat digambarkan sebagai

		j=0	j=1	j=2
			T	A
i=0		0	-1	-2
i=1	T	-1		
i=2	A	-2		

Gambar 8: Pengisian matrik $Mat(i,j)$, yang menunjukkan asal sub problem

dari ketiga nilai tersebut, kandidat ke satu atau $Mat(i-1,j-1)$ menghasilkan nilai $(0 + 1 = 1)$, kandidat ke dua atau $Mat(i-1,j)$ dan kandidat ke tiga atau $Mat(i,j-1)$, keduanya sama-sama menghasilkan nilai $(-1 + -1 = -2)$, sehingga nilai yang paling maksimum adalah kandidat ke satu, kemudian nilai ini akan mengisi $Mat(1,1)$.

Nilai-nilai lain dari $Mat(i,j)$ selanjutnya dapat diisi, dan kesemuanya dapat diformulasikan dalam hubungan :

$$\begin{aligned}
 &Mat(0,0) = 0 \\
 &Mat(0,j) = j * gapscore, j > 0 \\
 &Mat(i, 0) = i * gapscore, i > 0 \\
 &\text{untuk } 0 < i < \text{panjang string X}, 0 < j < \text{panjang string Y} \\
 &Mat(i,j) = \text{Max} \begin{cases} Mat(i-1,j-1) + score \text{ match/notmatch} \\ Mat(i,j-1) + gapscore \\ Mat(i-1,j) + gapscore \end{cases}
 \end{aligned}$$

Kotak 1: Formulasi pengisian $Mat(i,j)$

Dari langkah-langkah pengisian Matrik diperlukan penyimpanan ke dalam memori untuk mengingat dari manakah nilai $Mat(i,j)$ berasal, apakah dari operasi $Mat(i-1,j-1)$, $Mat(i-1,j)$ atau $Mat(i,j-1)$.

Setelah semua nilai $Mat(i,j)$ di isi, selanjutnya kita melakukan langkah pembalikan (*traceback*) untuk menentukan posisi *alignment*

		j=0	j=1	j=2	j=3	j=4
			T	A	C	A
i=0		0	-1	-2	-3	-4
i=1	T	-1	1	0	-1	-2
i=2	A	-2	0	2	1	0
i=3	T	-3	-1	1	2	1
i=4	A	-4	-2	0	1	3
i=5	G	-5	-3	-1	0	2

Gambar 9: $Mat(i,j)$ yang terisi penuh dan garis putus-putus yang menunjukkan *traceback*-nya

Dari gambar 9 terlihat nilai $Mat(i,j)$ dan tanda panah yang menunjukkan asal operasi dari $Mat(i,j)$, sebagai contoh $Mat(1,2)$ berasal dari operasi $Mat(1,1)$. Terdapat juga $Mat(4,2)$ yang berasal dari operasi $Mat(3,1)$ dan $Mat(3,2)$ (keduanya menghasilkan nilai yang sama), karena $Mat(3,1) + 1 = Mat(3,2) + (-1)$.

Untuk *traceback*, kita akan berangkat dari masalah terbesar di $Mat(i,j)$ dan menuju ke $Mat(i,j)$ yang memberikan kontribusi operasi , pada kasus ini masalah terbesar adalah *alignment* antara TATAG dengan TACA, yang berada pada $Mat(5,4)$. Dari $Mat(5,4)$ kita akan menarik garis ke $Mat(4,4)$, kemudian ke $Mat(3,3)$, $Mat(2,2)$, $Mat(1,1)$ dan terakhir $Mat(0,0)$. Pada gambar 9, *traceback* ditunjukkan dengan garis putus-putus.

Dari kotak 1 juga diketahui bahwa untuk garis vertikal, $Mat(i,j)$ merupakan hasil operasi dari $Mat(i-1,j)$ yang berarti *alignment* antara string $X[i]$ dengan *gap*, garis horizontal yang berarti $Mat(i,j)$ merupakan hasil operasi dari $Mat(i,j-1)$ adalah *alignment* antara *gap* dengan string $Y[j]$, dan garis diagonal yang merupakan hasil operasi dari $Mat(i-1,j-1)$ adalah *alignment* antara string $X[i]$ dengan $Y[j]$.

Melalui langkah langkah diatas maka alignmen yang optimal dari $X=\{TATAG\}$ dan $Y=\{TACA\}$ mempunyai bentuk

TATAG
||*|
TACA-

Contoh lain adalah *alignment* $X=\{ACTCA\}$ dengan $Y=\{ACAGTAG\}$, yang menarik dari contoh ini adalah adanya lebih dari satu *traceback*. Papat dilihat dari $Mat(i,j)$ di gambar 11

		A	C	A	G	T	A	G
	0	-1	-2	-3	-4	-5	-6	-7
A	-1	2	1	0	-1	-2	-3	-4
C	-2	1	4	3	2	1	0	-1
T	-3	0	3	4	3	4	3	2
C	-4	-1	2	3	4	3	4	3
A	-5	-2	1	4	3	4	5	4

Gambar 11: $Mat(i,j)$ yang menunjukkan lebih dari satu *traceback*

pada gambar 11 terdapat lima hasil *traceback* dapat dilihat. Kelimanya menghasilkan skor yang sama, yaitu empat, alignment-nya dapat dilihat di gambar 12. Dalam aplikasinya di pemrograman tentunya hasil lebih alignmnet optimal yang berarti dari satu *traceback* harus kita antisipasi

```
AC--TCA
||**
ACAGTAG
alignment no 1,score = 4
AC--TCA-
|||
ACAGT-AG
alignment no 2,score = 4
AC-TCA-
||**|
ACAGTAG
alignment no 3,score = 4
ACT-CA-
||* *|
ACAGTAG
alignment no 4,score = 4
ACTC-A-
||**|
ACAGTAG
alignment no 5,score = 4
```

Gambar 12: Hasil *alignment* yang menunjukkan skor yang sama

Dari contoh-contoh diatas terlihat fungsi pemrograman dinamis dalam aplikasi bioinformatika yaitu pengerjaan *sequence alignment*. Pemrograman singkat untuk algoritma Needleman Wunsch dapat dilakukan untuk panjang sekuen yang pendek. Langkah langkah pemrogramannya adalah (untuk program keseluruhan dapat dilihat di lampiran program).

1. Input sekuen, Sekuen X dan Y.
2. input aturan skor, untuk *match*, *misMatch* dan *gap*.
3. Inisialisasi $Mat(i,j)$, pembuatan Matrik berukuran $Mat [panjang\ sekuen\ X + 1][panjang\ sekuen\ Y + 1]$.
4. Pengisian $Mat(i,j)$, mulai dari indeks i kemudian j.
5. Penyimpanan kode *traceback* dalam suatu Matrik, dimisalkan untuk $Mat(i,j)$ berasal operasi dari diagonal $Mat (i-1,j-1)$ diberi kode tiga, dari kiri kode satu dan seterusnya.
6. Penelusuran *traceback* dimulai dari elemen $Mat(m,n)$ dimana m adalah panjang sekuen X + 1 dan n adalah panjang sekuen Y + 1.
7. Penghitungan skor menjumlahkan antara skor *match*, *mismatch* dan *gap*.

4. KESIMPULAN

Pada aplikasi pemrograman dinamis, terlihat suatu motif yang sama yaitu mencari suatu sub problem yang kemudian diselesaikan untuk menjawab suatu problem yang lebih kompleks. Pada contoh dasar seperti pada baris fibonacci dan graf multistap. Aplikasinya di bidang *bioinformatika* yaitu pada Global sekuen *alignment* diterapkan pada Algoritma Needleman Wunsch, algoritma ini berjalan dengan membuat suatu Matrik skor dan *traceback*.

Pada penelusuran kembali dengan *traceback*, dimungkinkan ditemukan lebih dari satu *traceback*, yang berarti terdapat lebih dari satu

pola *alignment* tetapi dengan skor yang sama.

5. REFERENSI

- [1] S Dreyfus, Operations Research journal, Vol. 50, No. 1, January–February 2002, pp. 48–51
- [2] <http://en.wikipedia.org/wiki/RAND>
- [3] Bellman R, Dynamic Programming Princeton University Press, 1957.
- [4] C Gibas, P Jambeck, Developing Bioinformatics Computer Skills, O'Reilly Publishing, 2001.
- [5] S Needleman, C Wunsch, A general method applicable to the search for similarities in the amino acid sequence of two proteins, J Mol Biol. 48(3):443-53, 1970
- [6] Temple F. Smith and Michael S. Waterman, "Identification of Common Molecular Subsequences", Journal of Molecular Biology, 147:195-197, 1981

Lampiran Program C

```
/* Global sequence Alignment, Needleman & Wunsch
Algorithhm */
/* Arief Wicaksono */
/* @2004 - 2006 */

# include <stdio.h>
# include <stdlib.h>
# define N 24 /* Length Nucleotide A */
# define M 20 /* Length Nucleotide B */

int main()
{
char A[N]="AGGTCGTGACAGTACGTAGCAGTA";
/*Nucleotide A */
char B[M]="CGTCTAACGTACGAGCCGTA";
/*Nucleotide B */
char G[1]="-"; /*Gap*/
char Af[N+M],Bf[N+M]; /*Matched sequence*/

/*scoring rule */
int match=1,notMatch=0;
int gap=-1;

int Mat[N+1][M+1];
int path[N+1][M+1];
```

```
int isplit,jsplit,valsplit;
int Matchscore,gapscore,notMatchscore;
int r[3];
int i,j,k,ii;
int score;
int large,tag;

int t,sign,ux,vy;
int branch,num;

printf("string A: ");
for (i=0;i<N;i++){
    printf("%c",A[i]);
}
printf("\n");
printf("string B: ");
for (i=0;i<M;i++){
    printf("%c",B[i]);
}
printf("\n\n");

/*Dynamic programming Matrix Mat[][] */
/*Traceback Matrix path[][] */
/* _1 _2 |3 _\4 _|5 _\6 _\|7 */

Mat[0][0]=0;
for (i=1;i<N+1;i++){
    Mat[i][0]=i*gap;
    path[i][0]=3;
}

for (j=1;j<M+1;j++){
    Mat[0][j]=j*gap;
    path[0][j]=1;
}
path[0][0]=0;

for (i=1;i<N+1;i++){
    for (j=1;j<M+1;j++){
        score=notMatch;

        if(A[i-1]==B[j-1])
            score=match;
        r[0]=Mat[i][j-1]+gap;
        r[1]=Mat[i-1][j-1]+score;
        r[2]=Mat[i-1][j]+gap;

        if(r[0]==r[1] && r[2]<r[1])
            path[i][j]=4;
```



```

else if(r[0]==r[2] && r[1]<r[2])
    path[i][j]=5;

else if(r[1]==r[2] && r[0]<r[1])
    path[i][j]=6;

else if(r[0]==r[1] && r[1]==r[2])
    path[i][j]=7;

else{
    large=r[0];
    tag=0;
    for(k=1;k<3;k++){
        if(r[k]>large){
            large=r[k];
            tag=k;
        }
    }
    path[i][j]=tag+1;
}

large=r[0];

for(k=1;k<3;k++){
    if(r[k]>=large){
        large=r[k];
    }
}
Mat[i][j]=large;
}
}

for(i=0;i<N+1;i++){
    for(j=0;j<M+1;j++){
        printf("%3.1d ",Mat[i][j]);
    }
    printf ("\n");
}
printf("\n");

for(i=0;i<N+1;i++){
    for(j=0;j<M+1;j++){
        printf("%d ",path[i][j]);
    }
    printf ("\n");
}

/* constructing Matched string */

/* with traceback Matrix */
/* maybe there is more than one optimal result*/

num=0;
do{
    t=0;
    i=N;j=M;
    ux=N-1;vy=M-1;
    branch=0;

    while(i>=1 || j>=1){
        sign=path[i][j];/*printf("sign %d [%c][%c]
i=%d j=%d \n",sign,A[ux],B[vy],i,j);*/

        switch(sign){
            case 1: j--;
                Af[t]=G[0]; Bf[t]=B[vy];vy--;
                break;

            case 2: i--;j--;
                Af[t]=A[ux]; Bf[t]=B[vy];ux--;vy--;
                break;

            case 3: i--;
                Af[t]=A[ux]; Bf[t]=G[0];ux--;
                break;

            case 4:
                Af[t]=A[ux]; Bf[t]=B[vy];ux--;vy--;
                isplit=i;jsplit=j;valsplit=path[i][j];
                i--;j--;branch=1;
                break;

            case 5:
                Af[t]=A[ux]; Bf[t]=G[0];ux--;
                isplit=i;jsplit=j;valsplit=path[i][j];
                i--;branch=1;
                break;

            case 6:
                Af[t]=A[ux]; Bf[t]=G[0];ux--;
                isplit=i;jsplit=j;valsplit=path[i][j];
                i--;branch=1;
                break;

            case 7:
                Af[t]=A[ux]; Bf[t]=G[0];ux--;
                isplit=i;jsplit=j;valsplit=path[i][j];
                i--;branch=1;
                break;
        }
    }
}

```

```

        default:
        break;
    }
    t++;
}

if(branch==1){
    switch (valsplitted){
        case 4: path[isplitted][jsplitted]=1;
        break;
        case 5: path[isplitted][jsplitted]=1;

        break;
        case 6: path[isplitted][jsplitted]=2;
        break;
        case 7: path[isplitted][jsplitted]=4;

        break;
        default:
        break;
    }
}

for(ii=t-1;ii>=0;ii--){
    printf("%c",Af[ii]);
}

printf("\n");
Matchscore=0;gapscore=0;notMatchscore=0;
for(ii=t-1;ii>=0;ii--){
    if(Af[ii]==Bf[ii]){
        printf("|");Matchscore+=match;
    }
    else if( (Af[ii]!=G[0] && Bf[ii]==G[0]) ||
(Af[ii]==G[0] && Bf[ii]!=G[0]) ){
        printf(" ");gapscore+=gap;
    }
    else{
        printf("*");notMatchscore+=notMatch;
    }
}
printf("\n");
for(ii=t-1;ii>=0;ii--){
    printf("%c",Bf[ii]);
}
printf("\n");
printf("alignment no %d,",num+1);
printf("score = %d
\n\n",Matchscore+gapscore+notMatchscore);
num++;
}while(branch==1);

```

```

return 0;
}

```

Keluaran Program

```

string A: AGGTCGTGACAGTACGTAGCAGTA
string B: CGTCTAACGTACGAGCCGTA

```

```

AGGTCGTGACAGTACGTAGCAGTA
*| || |*|| |||| |||*|||
CG-TC-TAAC-GTACG-AGCCGTA
alignment no 1,score = 13

```

```

AGGTCGTGACAGTACGTAGCAGTA
* ||| |*|| |||| |||*|||
C-GTC-TAAC-GTACG-AGCCGTA
alignment no 2,score = 13

```

```

AGGTCGTGACAGTACGTAGCAGTA
*||| |*|| |||| |||*|||
-CGTC-TAAC-GTACG-AGCCGTA
alignment no 3,score = 13

```

BIOGRAFI PENULIS



Arief Wicaksono. Lahir di Jakarta 21 September 1979. Menamatkan sekolah menengah umum di SMU 90, Jakarta pada tahun 1997. Menyelesaikan program S1 pada jurusan Fisika Teori di Universitas Indonesia Pada tahun 2003. Berminat pada bidang pemrosesan paralel, komputasi, *bioinformatika* dan elektronik, juga mempunyai hobi bertanam sayuran organik, sampai saat ini bekerja di perusahaan swasta sebagai programmer di platform open source, penulis dapat dihubungi di alamat email awicaksono@dn.net.id.