APLIKASI PROGRAM DINAMIS DALAM STRING ALIGNMENT UNTUK MELAKUKAN PENCOCOKAN DNA

Edria Albert Varian W – NIM 13507031

Program Studi Teknik Informatika Institut Teknologi Bandung
Jalan Ganesha no 10 Bandung Indonesia 40132
e-mail: if17031@students.if.itb.ac.id

ABSTRAK

Program Dinamis adalah metode pemecahan masalah dengan cara menguraikan solusi menjadi sekumpulan langkah (step) atau tahapan (stage) sedemikian sehingga solusi dari persoalan dapat dipandang dari serangkaian keputusan yang saling berkaitan. Program dinamis ini salah satunya digunakan dalam algoritma *string alignment* yang digunakan dalam biologi molekuler untuk melakukan pencocokan DNA.

Dalam ilmu biologi molekuler sering kali ditemukan sample-sample DNA dari organisme yang diteliti. DNA tersebut akan diteliti untuk dicari fungsinya ataupun mencari tahu apakah organisme ini memiliki kekerabatan dengan spesies lain. Sebelum melakukan penelitian sample DNA tersebut dapat dibandingkan dengan sample DNA lain yang sudah pernah diteliti dan berada di database. Namun sample tersebut tidak diharuskan untuk 100% sama untuk dikategorikan cocok dan memiliki hubungan dengan DNA yang sudah pernah diteliti. Untuk itu algoritma string alignment diperlukan untuk mendapatkan kombinasi yang menghasilkan nilai kecocokan tertinggi. Setelah itu peneliti bisa menyimpulkan hubungan antara DNA sampel dan DNA di database berdasarkan nilai kecocokan tersebut.

Kata kunci: Program Dinamis, String Alignment, Biologi Molekuler, Pencocokan DNA.

1. PENDAHULUAN

1.1 Program Dinamis

Dalam Program Dinamis, sebuah persoalan dapat diselesaikan dengan metode yang memiliki ciri:
1. Terdapat sejumlah berhingga pilihan yang mungkin
2. Solusi pada setiap tahap dibangun dari hasil solusi tahap sebelumnya
3. Menggunakan persyaratan optimasi dan kendala untuk membatasi sejumlah pilihan yang harus dipertimbangkan setiap tahap
4. Jika solusi total optimal, maka bagian solusi sampai tahap ke-k juga optimal

Algoritma yang membentuk solusi secara bertahap selain Program Dinamis adalah algoritma Greedy, namun hal yang membedakan dengan Program Dinamis adalah dalam Program Dinamis rangkaian keputusan yang pernah dihasilkan tidak hanya satu buah seperti layaknya Greedy.

1.2 Biologi Molekuler

Sebagian besar subjek dari biologi molekuler adalah yang berhubungan dengan analisis dan perbandingan dari materi genetik (DNA, RNA, protein, kromosom, gen, dll) dari organisme yang berbeda-beda. Untuk melakukan hal ini diperlukan pengumpulan, pemerosesan dan penyimpanan informasi dalam jumlah yang besar. Untungnya, banyak jenis dari data biomolekular memiliki struktur data yang mirip, contohnya struktur primer dari protein adalah rantai satu dimensi dari residu asam amino dan untaiannya dapat dituliskan sebagai *string* dengan panjang tertentu. Struktur yang sama ditemukan juga pada rangkaian dalam DNA, yang biasa disebut rangkaian genetik. Rangkaian genetik merupakan sebuah *string* yang dibentuk dari empat buah alfabet {A,G,T,C}. Keempat alfabet ini mewakili struktur penyusun DNA yang terdiri dari purin (Adenin an Guanin) dan pirimidin (Timin dan Sitosin).

Gen merupakan rangkaian genetik yang berisi informasi-informasi yang dibutuhkan untuk menyusun sebuah protein. Pada mahluk hidup, kumpulan gen-gen akan bergabung menjadi genom. Setiap kali sebuah sel dibentuk oleh mahluk hidup, sel tersebut akan menerima salinan dari genom sel lain. Proses pensalinan ini sering-kali tidak berlangsung sempurna untuk seluruh gen yang terkandung didalamnya, sehingga sering terjadi perubahan berupa pertukaran antara basis yang satu dengan yang lain atau penghapusan dari sub-*string*. Proses ini biasa disebut mutasi dan jumlah perubahan yang terjadi biasa digolongkan dalam poin mutasi. Pada gen yang sama dari organisme yang memiliki hubungan kekerabatan yang dekat, poin mutasinya akan sangat kecil.

1.3 *String Alignment*

Alignment digunakan untuk merepresentasikan relasi evolusi dari dua buah protein atau DNA yang dibandingkan. Misalnya dalam suatu penelitian suatu organisme didapat suatu rangkaian genetik AACAGTTACC. Dengan rangkaian genetik ini dapat dilakukan suatu penelitian untuk melihat peran dari gen tersebut, tetapi ada kemungkinan bahwa gen tersebut merupakan varian dari gen yang sudah diketahui dari penelitian sebelumnya. Oleh karena itu dapat dilakukan pengecekan terlebih dahulu kepada gen-gen yang telah diteliti. Untuk membandingkan rangkaian genetik dibutuhkan suatu metode tertentu yang bisa memastikan bahwa dua rangkaian genetik tersebut cukup mirip dan disimpulkan bahwa gen tersebut memiliki fungsi yang mirip.

Dengan metode string alignment dua gen ini akan dibandingkan dan didapatkan suatu nilai perbedaan (edit distance), konsep ini pertama kali diperkenalkan oleh Levenshtein pada konteks teori pemrograman. Dalam melakukan alignment ini diperbolehkan untuk menyisipkan suatu *gap* di antara string yang dibandingkan. Pada setiap perbandingan karakter terdapat suatu standar nilai dalam penentuan nilai perbedaan dari pencocokan tersebut. Standar nilai tersebut terdiri dari nilai gap, nilai cocok dan nilai tidak cocok. Penentuan nilai ini tidak memiliki standar baku sehingga sangat bergantung pada standar dari peneliti. Dalam algoritma string alignment standard ini biasa disebut *Scoring Matrix*.

2. METODE

Pada *string alignment* ini algoritma program dinamis digunakan untuk menghitung nilai dan *alignment* optimal untuk setiap sub-*string*. pertama-tama kita harus menghitung *alignment* kombinasi yang optimal untuk masing-masing sub-*string* dan menyimpan nilainya pada sebuah matriks. Untuk dua *string*, misalnya s dengan panjang m merupakan *string* pertama dan t dengan panjang n merupakan *string* kedua, D[I,j] merupakan nilai terbaik dari kombinasi sub-*string* s[1..j] dan t[1..i].

Untuk *alignment* global, dicari nilai terbaik dan *alignment* terbaik dari kedua *string*, untuk *alignment* lokalnya dicari kemungkinan sub-*string* yang akan memberikan nilai optimum. Pada *alignment* global yang tidak memiliki *gap* di akhir string, kita menghitung nilai terbaik dari *alignment* dua string tanpa *scoring* yang diakibatkan oleh *gap*. Tiga pendekatan ini berbeda dalam segi kondisi awal, metode untuk penghitungan nilai dan kondisi akhir proses.

Umumnya pada algoritma program dinamis terdapat dua fase, fase maju dan fase mundur. Pada fase maju, dihitung nilai optimal dari setiap sub-masalah. Pada fase mundur, dibentuk solusi yang memberikan nilai maksimal. Untuk algoritma *string alignment* ini dalam fase maju kita akan menggunakan operasi rekursif untuk menghitung setiap nilai D[i,j] dan fase mundurnya akan melakukan penelusuran ulang untuk menemukan *alignment* optimal.

Operasi rekursif akan menghasilkan hubungan antara D[i,j] dan nilai dari D yang lebih kecil dari di i dan j. Jika tidak ada nilai yang lebih kecil dari di i dan j maka nilai dari D[i,j] harus diinisiasi pada saat kondisi awal. Kondisi awal digunakan untuk menghitung nilai di baris dan kolom pertama pada saat belum ada nilai pada elemen matriks yang diatas dan dikanannya. Setelah kondisi awal ini diinisiasi barulah proses rekursif dapat dijalankan. Algoritma rekursifnya akan berupa

```
D[i,j] = max{D[i-1,j-1]+sim_mat[s[j],t[i]],
             D[i-1,j] + gap_score,
             D[i,j-1] + gap_score }
```

dengan lain, jika terdapat *alignment* optimal hingga pada D[i-1,j-1] maka aka nada tiga kemungkinan yang akan terjadi

1. Karakter pada s[i] dan t[j] cocok
2. Gap diberikan di t atau
3. Gap diberikan di s

Tidak diperbolehkan untuk menambahkan gap pada kedua *string* pada indeks yang sama. Maksimum dari ketiga nilai ini akan dipilih sebagai nilai optimal dan ditulis dalam elemen matriks D[i,j].

Fase kedua digunakan untuk membentuk *alignment* optimal dengan melakukan penelusuran melalui jalur manapun dari D[m,n] ke D[0,0] yang mengarah ke nilai terbaik. Oleh karena itu mungkin terdapat beberapa kemungkinan dari nilai optimal di matriks D[i,j].

2.1 Inisialisasi

Untuk selanjutnya akan dicontohkan proses *string alignment* pada dua buah *string* DNA yaitu, ACGGTAG dan CCTAAG. Pertama-tama kedua *string* ini disimpan dalam dua variable s dan t dan panjang dari masing-masing string disimpan dalam m dan n.

Matriks D dengan dimensi (m+1)(n+1)dibuat untuk menyimpan nilai optimal dinamis dari setiap pasangan sub-*string*. Pada kondisi awal matriks D akan berupa

		A	C	G	G	T	A	C
	0							
C								
C								
T								
A								
A								
G								

2.2 Membuat *Scoring Matrix*

Scoring Matrix ini dibuat sesuai kebutuhan peneliti tetapi sebaiknya penilaiannya berdasarkan dasar-dasar pada kondisi yang sebenarnya. Karena objek dari *string alignment* ini adalah DNA maka *scoring matrix* ini dibuat berdasarkan prinsip-prinsip dalam pencocokan DNA yaitu: tipe purin secara kimiawi memiliki kemiripan antara Adenin dan Guanin begitu juga pada Timin dan Sitosin. Pada kasus cocok diberi nilai 2, bila purin dibandingkan dengan purin atau pirimidin dengan pirimidin diberi nilai 1, dan nilai -1 untuk purin dengan pirimidin.

	A	C	G	T
A	2	-1	1	-1
C	-1	2	-1	1
G	1	-1	2	-1
T	-1	1	-1	2

Matiks sim_mat, Hasil dari *Scoring Matrix*

Pada kemungkinan perbandingan karakter dengan *gap* diberikan nilai -2 (*gap_score*).

2.3 Inisialisasi Perhitungan

Elemen pertama dari matriks yang diisi adalah D[1,1] dan diisi dengan nilai 0, karena tidak ada aturan penalti nilai untuk *alignment* karakter kosong dengan karakter kosong. Lalu gunakan nilai dari D[i-1,j-1], D[i,j-1], dan D[i-1,j] untuk menentukan nilai dari D[i,j]. Kolom dan Baris pertama dari matriks D merupakan nilai dari *alignment* sub-*string* s[1..j] dan t[1..i] dengan *string gap* yang memiliki panjang j atau i.

Relasi rekursif pada kolom pertama dan baris pertama adalah,

D[1,j] = D[1,j-1] + gap_score;

D[i,1] = D[i-1,1] + gap_score;

Relasi ini dipakai dalam kalang perhitungan untuk mengisi nilai kolom dan baris pertama

```
for j to n do
  D[1,j+1] ← gap_score*j;
endfor;
for i to m do
  D[i+1,1] ← gap_score*i;
endfor;
```

Sehingga matriks D sekarang akan berisi

```
        A   C   G   G   T   A   C
    0  -2  -4  -6  -8 -10 -12 -14
C  -2   0   0   0   0   0   0   0
C  -4   0   0   0   0   0   0   0
T  -6   0   0   0   0   0   0   0
A  -8   0   0   0   0   0   0   0
A -10   0   0   0   0   0   0   0
G -12   0   0   0   0   0   0   0
```

2.4 Menghitung semua D[i,j]

Seperti yang sudah dijelkaskan sebelumnya bahwa prinsip rekursif untuk menyelesaikan sisa dari tabel D adalah

```
D[i,j] = max { D[i-1,j-1]+
sim_mat[s[j],t[i]], D[i-1,j] +
gap_score, D[i,j-1] + gap_score };
```

1. Apabila nilai tertinggi berasal dari element D[i-1,j-1] maka karakter pada s[i] dan t[j] cocok.
2. Apabila nilai tertinggi berasal dari elemen D[i-1,j] maka *gap* pada *string* s akan bertemu karakter pada *string* t.
3. Apabila nilai tertinggi berasal dari elemen D[I,j-1] maka *gap* pada *string* akan bertemu karakter pada *string* s.

Ulangi hingga i dan j dan hitung nilai dari ketiga kemungkinan dan simpan nilai maksimumnya pada D[i,j]. Setelah semua elemen terisi, maka nilai dari *alignment* global akan berada pada matriks D[m+1,n+1].

```
for i from 2 to m + 1 do
  for j from 2 to n + 1 do
    match ← D[i-1,j-1] +
sim_mat[BToInt(s[j-1]),BToInt(t[i-1])];
    gaps ← D[i,j-1] + gap_score;
    gapt ← D[i-1,j] + gap_score;
    D[i,j] ← max(match,gaps,gapt);
  endfor;
endfor;
score := D[m+1,n+1];
```

maka kondisi matriks D akan menjadi,

```
   0  -2  -4  -6  -8 -10 -12 -14
  -2  -1   0  -2  -4  -6  -8 -10
  -4  -3   1  -1  -3  -3  -5  -7
  -6  -5  -1   0  -2  -1  -3  -5
  -8  -4  -3   0   1  -1   1  -1
 -10  -6  -5  -2   1   0   1   2
 -12  -8  -7  -3   0   0   1   3

score = 3
```

2.5 Membentuk *Alignment*

Untuk membentuk *alignment* ini kita harus melakukan penyelusuran ulang untuk membentuk pola dari posisi D[m+1,n+1] ke D[1,1] yang mengarah ke nilai terbesar. Mulai dari matriks elemen D[m+1,n+1] yang berisi nilai terbesar untuk *alignment* global, inisiasi variable bernana s_aln dan t_aln dengan *string* kosong untuk nantinya di-konkat-kan dengan karakter yang kita dapatkan selama

proses penelusuran balik tersebut. Karakter yang didapatkan tersebut ditambahkan di depan isi variable karena kita memulai penelusuran dari belakang.

Dalam penelusuran ini mungkin tidak akan hanya terdapat satu jalur yang unik yang berarti terdapat beberapa kemungkinan yang menghasilkan nilai tertinggi.

Untuk melakukan penelusuran yang mengarah kepada nilai yang optimal, kita mulai dari akhir, elemen matriks D[m,n] dan ulangi perhitungan kearah awal untuk menemukan titik awalnya. Dengan kata lain dari elemen D[i,j] kita menghitung nilai pada elemen D[i-1,j-1], D[i-1,j] dan D[I,j-1]. Dengan memilih nilai terbesar, misalnya jika nilai D[i-1,j-1] yang dipilih, lalu kita tambahkan karakter selanjutnya dari s dan t ke s_aln dan t_aln. Jika D[i,j-1] yang dipilih maka kita menambahkan *gap* ke t_aln dan karakter selanjutnya dari *string* s ke s_aln. Jika D[i-1,j] yang dipilih, maka kita menambahkan *gap* ke s_aln dan karakter selanjutnya dari *string* t ke t_aln. Dalam algoritma ini kita membuat prioritas untuk memilih jalur diagonal terlebih dahulu, lalu menambahkan *gap* ke t pada prioritas kedua, dan menambahkan *gap* ke s pada prioritas ketiga. Prioritas ini ditentukan oleh peneliti.

```
while i > 1 and j > 1 do
  if D[i,j] - sim_mat[BToInt(s[j-
1]),BToInt(t[i-1])] == D[i-1,j-1] then
    t_aln ← t[i-1].t_aln;
    s_aln ← s[j-1].s_aln;
    i ← i-1;
    j ← j-1;
  elseif D[i,j] - gap_score == D[i,j-1]
then
    s_aln ← s[j-1].s_aln;
    t_aln ← '_'.t_aln;
    j ← j-1;
  elseif D[i,j] - gap_score = D[i-1,j]
then
    s_aln ← '_'.s_aln;
    t_aln ← t[i-1].t_aln;
    i ← i-1;
  else
    error('should not happen');
  endif;
endwhile;
```

Pada akhir dari proses i dan j akan bernilai 1. Jika belum, akhiri dengan menambahkan *gap* pada sisa *string* hingga i dan j bernilai 1.

```
if j > 1 then
  while j > 1 do
    s_aln ← s[j-1].s_aln;
    t_aln ← '_'.t_aln;
    j ← j-1;
  endwhile;
elseif i > 1 then
  while i > 1 do
    s_aln ← '_'.s_aln;
    t_aln ← t[i-1].t_aln;
    i ← i-1;
  endwhile;
endif;
```

Pada akhir dari proses ini s_aln dan t_aln akan bernilai

s_aln	A	C	G	G	T	A	G
t_aln	C	C	T	A	_	A	G
score	-1	2	-1	1	-2	2	2

Sehingga dari alignment ini didapatkan nilai *alignment* global 3. Nilai *alignment* global ini akan berbanding terbalik dengan poin mutasi, yang artinya semakin besar nilai dari *alignment* global yang didapatkan maka semakin banyak kesamaa dari dua *string* gen yang dibandingkan yang artinya poin mutasinya akan semakin kecil.

3. KESIMPULAN

String alignment merupakan suatu algoritma untuk melakukan penyusunan yang optimal pada suatu string sehingga bila dibandingkan dengan *string* lain akan ditemukan tingkat kecocokan yang paling maksimum. *String alignment* pada penggunaannya dalam pencocokan DNA cukup pas untuk menerapkan konsep pemerograman dinamis. *String alignment* memiliki serangkaian tahap yang terpisah namun saling berhubungan. Oleh karena itu tahap demi tahap dapat diselesaikan secara terpisah.

Algoritma *string alignment* cukup untuk menyelesaikan masalah yang terdapat dalam pencocokan DNA dimana sample DNA dapat direkayasa dengan memberikan *gap* diantara karakter untuk mendapatkan nilai kecocokan antara sample-sample yang dicocokan.

REFERENSI

[1] Functional Programming Department of Computer Science Lund Institute of Technology http://www.cs.lth.se/EDA120/assignment3/ Tanggal akses : 31 Desember 2009 pukul 15.00

[2] Munir, Rinaldi, *Matematika Diskrit*, Program Studi Informatika, Institut Teknologi Bandung. 2007.

[3] Global Sequence Alignment http://www.cs.princeton.edu/courses/archive/spr05/cos126/assignments/sequence.html Tanggal akses : 31 Desember 2009 pukul 15.30

[4] String Alignment using Dynamic Programming http://www.biorecipes.com/DynProgBasic/code.html Tanggal akses : 31 Desember 2009 pukul 15.30

[5] String Alignment http://www.inf.ethz.ch/personal/cannaroz/courses/compbio/week2/week2/week2.html Tanggal akses : 31 Desember 2009 pukul 15.30