

Mengapa Harus Perl?

Kontribusi: Taufik Abidin
Sunday, 08 October 2006

Perl atau Practical Extraction and Report Language dirancang oleh Larry Wall dipertengahan tahun 1980-an ketika ia ingin membuat bug-reporting system namun bahasa pemrograman yang tersedia, seperti awk, tidak dapat memenuhi keinginannya. Larry bisa saja waktu itu menggunakan bahasa pemrograman seperti C atau C++, tetapi karena dia menginginkan agar system-nya itu bisa dijalankan melalui shell ala awk, akhirnya dia memutuskan untuk merancang bahasa baru yaitu Perl. Secara umum, Perl bisa dikategorikan berada diantara bahasa pemrograman tingkat tinggi (high-level programming language) seperti pemrograman shell dan bahasa pemrograman tingkat rendah (low-level programming language) seperti C, C++ atau assembly. Program yang ditulis dengan bahasa pemrograman tingkat tinggi biasanya lambat dan sangat tergantung pada command yang disediakan oleh operating system. Sebaliknya, program yang ditulis dengan bahasa pemrograman tingkat rendah cenderung lebih cepat, namun lebih sulit dalam menuliskannya, walau hampir semua hal dapat ditulis dengan bahasa pemrograman level ini. Perl dirancang untuk menjembatani ketimpangan diantara kedua level bahasa pemrograman di atas. Sejak pertama sekali diperkenalkan sampai dengan saat ini, Perl berkembang dengan sangat cepat. Perl versi 5 yang cukup populer dan stabil, tidak lama lagi akan digantikan dengan Perl versi 6. Versi teranyar ini tentu saja memiliki fitur-fitur yang lebih baik dari versi sebelumnya. Informasi yang lebih lengkap tentang Perl versi terbaru ini dapat anda peroleh di <http://dev.perl.org/perl6> dan <http://www.pugscode.org>.

Sebagian Keunggulan Perl Setiap bahasa pemrograman pasti memiliki kelebihan dan kelemahan, karena keunggulan suatu bahasa pemrograman sangat tergantung pada persoalan apa yang ingin diselesaikan. Perl sangat cocok digunakan untuk menyelesaikan persoalan yang berkaitan dengan teks. Lebih kurang 90% dari kemampuan Perl memang dioptimalisasikan untuk menyelesaikan persoalan-persoalan yang berhubungan dengan teks, sementara sisanya, 10%, untuk menyelesaikan persoalan-persoalan diluar teks [1]. Bagi para peneliti di bidang bioinformatika, Perl merupakan tool utama dalam melakukan pattern matching dari gen dalam DNA. Sebagaimana kita ketahui, gen merupakan rangkaian nucleotide yang secara formal disimbolkan menggunakan gabungan karakter A (adenine), T (thymine), C (cytosine), dan G (guanine), seperti yang digambarkan pada gambar berikut: Gambar 1. Ilustrasi DNA [2]. Untuk mengetahui apakah terdapat gen tertentu dalam sebuah DNA, tentunya kita harus mencari pattern (yang merupakan kumpulan dari karakter-karakter A, T, C, G) di dalam DNA tersebut. Sebagai gambaran, perhatikan potongan program berikut ini:

```
1 my $sequence = 'CTAATGGA'; 2 my $pattern = 'ATG'; 3 if($sequence =~ /$pattern/){ 4 "Pattern $pattern ditemukan dalam sequence $sequence\n"; 5 } 6 else{ 7 print "Pattern $pattern tidak ditemukan dalam sequence $sequence\n"; 8 }
```

Pada baris ke 1 dan ke 2 di atas, sebuah skalar \$sequence dengan nilai CTAATGGA dan skalar \$pattern dengan nilai ATG dideklarasikan. Pengecekan apakah pattern ATG terdapat dalam sequence CTAATGGA dapat dengan mudah dilakukan menggunakan fasilitas pattern matching. Menggunakan binding operator, =~, seperti yang dilakukan pada baris ke 3 di atas, Perl memeriksa dengan sangat efisien apakah dalam skalar \$sequence terdapat pattern ATG. Untuk contoh di atas, pattern ATG ada dalam sequence CTAATGGA, sehingga pernyataan "Pattern ATG ditemukan dalam sequence CTAATGGA" akan ditampilkan dilayar monitor.

Sebenarnya, untuk contoh kasus di atas, bahasa C atau bahasa pemrograman lainnya juga dapat digunakan. Namun akan lebih rumit dan panjang. Bila ditulis dengan bahasa C, sudah jelas kita harus berurusan dengan pointer dan iterasi yang memeriksa satu per satu apakah setelah karakter A dua karakter selanjutnya adalah karakter T dan G.

Bagaimana pula bila kita ingin menggantikan semua karakter A dalam sebuah sequence DNA menjadi karakter T, karakter C menjadi G, karakter G menjadi C, dan karakter T menjadi A? Dengan bahasa C, salah satu solusinya adalah seperti program sederhana berikut ini:

```
1 char sequence[32] = 'CTAATGGA'; 2 char *p = sequence; 3 for (; *p != '\0'; p++){ 4 if (*p == 'A') 5 *p = 'T'; 6 else if (*p == 'C') 7 *p = 'G'; 8 else if (*p == 'G') 9 *p = 'C'; 10
```

Lumayan, tidak begitu sulit juga. Namun yang perlu kita pahami disini adalah, untuk kasus-kasus manipulasi teks seperti kedua contoh di atas, menggunakan Perl jauh lebih mudah. Fungsi tr atau translate dari Perl dapat mengubah setiap karakter yang dimaksud dengan sangat mudah.

```
tr/<pola awal>/<pola baru>/
```

Fungsi tr menerjemah setiap karakter yang ditemukan di <pola awal> menjadi karakter yang bersesuaian posisinya di bagian <pola baru>. Jadi untuk contoh kasus kedua ini, hanya dengan satu pernyataan saja, perubahan karakter dapat diselesaikan.

```
$sequence =~ tr/ACGT/TGCA/;
```

Bila pada awalnya sequence DNA adalah CTAATGGA, maka dengan pernyataan di atas, sequence DNA berubah menjadi GATTACCT.

Masih banyak lagi keunggulan-keunggulan Perl, yang tentunya tidak mungkin kita bahas semuanya dalam tulisan ini.

Namun penulis ingin menyampaikan bahwa begitu powerful-nya kemampuan Perl dalam memanipulasi teks dan file, termasuk dalam membuat cgi (common gateway interface). Selain itu, Perl juga merupakan tool yang sangat tepat untuk merancang prototype suatu algoritma. Benar memang kalau Perl bukanlah tool untuk semua hal. Perl bukanlah bahasa pemrograman yang tepat untuk merancang kernel, tidak pula untuk merancang program client-server yang membutuhkan penanganan thread dan memori poll yang efisien.

Kelemahan Perl

Perl dijalankan menggunakan interpreter. Interpreter Perl disebut perl (menggunakan huruf kecil semua). Perl tidak memiliki compiler yang dapat mengubah source code menjadi bahasa mesin (binary code). Program yang ditulis dengan Perl dapat dilihat dengan mudah oleh orang lain yang menjalankan program tersebut. Selain itu, tidaklah efisien menulis program yang sangat panjang (mungkin lebih dari 10,000 baris) menggunakan Perl bila program yang ditulis tersebut dijalankan berulang-ulang sebanyak ratusan atau bahkan ribuan kali per menit. Hal ini disebabkan karena setiap program tersebut dijalankan, interpreter perl harus menerjemahkannya ke dalam bytecode yang dipahami oleh compiler internal perl.

Penutup

Menggunakan tool yang tepat merupakan kunci sukses dalam menyelesaikan suatu persoalan. Agaknya pernyataan ini juga sangat relevan bagi programmer di dalam memilih bahasa pemrograman untuk menyelesaikan suatu masalah. Industri-industri IT yang bergerak dalam bidang searching engine menggunakan Perl sebagai tool penunjang dalam mengembangkan prototype suatu algoritma karena Perl memang sangat efisien dalam memanipulasi teks, bukan karena Perl itu gratis dengan lisensi GNU-nya. Demikian pula, alasan peneliti di bidang bioinformatika menggunakan Perl untuk mencari pattern yang bersesuaian di dalam sequence DNA adalah karena Perl sangat powerful dengan pattern matching-nya. Dus, bila anda sedang mengembangkan prototype suatu algoritma yang banyak berhubungan dengan teks tetapi anda tidak menggunakan Perl, berarti anda belum menggunakan tool yang tepat. Tidak percaya? Coba Perl!

Referensi 1. R. L. Schwartz, T. Phoenix, dan B. D. Foy, "Learning Perl", edisi ke 4, O'Reilly Media, Inc., California, 2005.
2. U.S. Department of Energy Human Genome Program, <http://www.ornl.gov/hgmis>.