Penerapan Algoritma Pattern Matching dalam Game Typer Shark Deluxe

Aditya Pratama - 13515103
Program Studi Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13515103@std.stei.itb.ac.id

***Abstract*—Di dalam dunia pemrogaman terdapat istilah algoritma. Algoritma merupakan cara atau langkah yang digunakan untuk menyelesaikan suatu masalah secara terstruktur dan efisien. Salah satu jenis algoritma yang zaman sekarang ini cukup terkenal dan banyak dipakai adalah pattern matching. Algortima pattern matching digunakan untuk melihat apakah ada kecocokan antara string yang diberikan / yang akan dicocokan (pattern) dengan string yang tersedia / string text. Salah satu penggunaan algoritma ini adalah pada game Typer Shark Deluxe. Dalam makalah ini akan dijelaskan mengenai implementasi algoritma pattern matching dalam game Typer Shark Deluxe.**

Keywords—pattern matching, game

I. PENDAHULUAN

Di zaman sekarang ini, rutinitas dengan kesibukan yang padat tidaklah asing lagi bagi masyarakat. Dari kesibukan tersebut, rasa jenuh yang dirasakan masyarakat dapat menyebabkan kinerja aktivitas menjadi turun. Karena itu, perlulah diisi dengan kegiatan yang menghilangkan rasa jenuh seperti menonton, makan, mendengar lagu, dan lain – lain. Kegiatan yang tidak kalah sering dilakukan saat menghilangkan rasa jenuh atau mengisi waktu luang adalah bermain. Ada banyak jenis tipe permainan, mulai dari permainan fisik, permainan mobile, permainan PC, dan lain – lain. Dalam makalah ini akan dibahas permainan yang akan dibahas adalah permainan yang berbasis PC.

Permainan ini memiliki nama Typer Shark Deluxe, sebuah permainan yang dibuat oleh developer bernama PopCap Games. Tujuan dari game ini adalah bertahan hidup di dalam laut dari predator – predator yang ada. Cara bertahan hidup pemain adalah dengan memasukan string atau *pattern* yang sesuai dengan teks string pada badan predator.

Gambar 1: Type Shark Deluxe

Sumber: (http://cdn.akamai.steamstatic.com/steam/apps/3450/header.jpg?t=1447350888)

Salah satu algortima yang telah dipelajari pada mata kuliah IF2211 Strategi Algoritma adalah Algortima Pattern Matching. Dalam makalah ini akan ditunjukan penerapan algoritma tersebut dalam permainan Typer Shark Deluxe, yaitu dengan mencocokan string yang telah dimasukan pemain dengan string yang ada pada badan predator.

II. DASAR TEORI

A. *Algoritma Pattern Matching*

Pattern matching adalah sebuah proses pencarian string pada suatu teks. Ada 2 elemen dalam persoalan pencarian string, yaitu :

- Teks, merupakan string yang memiliki panjang n karakter.
- *Pattern*, merupakan string yang memiliki panjang m karakter, yang akan dicari keberadaannya pada teks (m < n)

Pada saat pencocokan, akan dicari lokasi pertama dimana *pattern* terletak pada teks. Ada beberapa metode yang digunakan dalam pattern matching, yaitu algoritma Brute Force, algoritma Knuth-Morris-Pratt (KMP), dan algoritma Boyer Moore. [1]

1. Algoritma Brute Force

Algoritma Brute Force adalah algoritma yang mengecek semua kemungkinan yang ada tanpa memperdulikan efisiensi, karena itu algoritma ini sering disebut sebagai algoritma naif. Algoritma Brute Force ini memiliki langkah – langkah pengerjaan sebagai berikut :

a. Sejajarkan P[i] (*pattern*) dengan T[j] (teks) dimana i = j = 1.

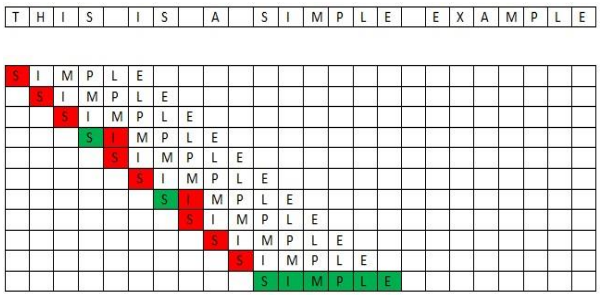
b. Pencocokan pattern akan bergerak dari kiri ke kanan dengan mencocokan setiap karakter dari pattern yaitu P[i] dengan T[j], kemudian P[i + 1] dengan T[j + 1], dst sampai :

   i. pattern sudah ditemukan didalam teks, atau

   ii. terdapat perbedaan karakter saat pencocokan atau mismatch

c. Apabila terjadi mismatch, maka ulangi langkah 2 dengan i = 1, dan j = j + 1

d. Apabila teks sudah habis dan *pattern* masi belum ditemukan di dalam teks, maka akan membalikan atau return nilai yang melambangkan *pattern not found*.



Gambar 2: Contoh proses Algoritma Pattern String dengan Brute Force

Sumber: (http://1.bp.blogspot.com/-YJDalyxz6XY/UFCYnd2_nBI/AAAAAAAAAB4/uewJpXgs9Mc/s1600/Brute+Force.jpg)

Berikut adalah penerapan algoritma Brute Force dalam bahasa Java :

```
public static int brute(String text,String pattern)
{ int n = text.length();    // n is length of text
  int m = pattern.length(); // m is length of pattern
  int j;
  for(int i=0; i <= (n-m); i++) {
    j = 0;
    while ((j < m) && (text.charAt(i+j)== pattern.charAt(j)) ) {
         j++;
   }
    if (j == m)
         return i;   // match at i
  }
  return -1;   // no match
} // end of brute()
```

Gambar 3: Algoritma Brute Force Pattern Matching dalam Bahasa Java

Sumber: [1]

Kompleksitas waktu yang dihasilkan oleh Algoritma Brute Force berbeda – beda tergantung dari kasus yang ada. Pada kasus terburuk, kompleksitas waktunya adalah O(mn) dengan jumlah pencocokannya atau perbandingan sebanyak m(n - m + 1). Kasus tersebut dapat terjadi apabila semua karakter pada *pattern* selalu cocok dengan teks kecuali pada karakter akhir *pattern*, sehingga terdapat banyak pencocokan atau perbandingan antara *pattern* dengan teks. Contohnnya ketika :

Teks : aaaaaaaaaaaaaaaaaaaaaaaah

Pattern : aaah

Pada kasus terbaik, kompleksitas waktu yang dihasilkan adalah O(n), terjadi apabila karakter pertama *pattern* tidak pernah sama dengan karakter teks yang dicocokan, kecuali ketika posisi j berada pada karakter ke m – n + 1, Contoh :

Teks : String ini berakhir dengan zzz

Pattern : zzz

Pada kasus biasa, kompleksitas waktunya adalah O(m + n). Pencocokan string dengan Algoritma Brute Force biasanya cocok pada kalimat yang memiliki karakter yang bervariasi seperti pada teks yang memiliki banyak alfabet yang bervariasi. [1]

2. Algoritma Knuth-Morris-Pratt

Algoritma *pattern matching* dengan KMP memiliki sifat yang sama dengan Brute Force yaitu menelusuri karakter dari kiri ke kanan, tetapi Algoritma KMP memiliki cara yang lebih efisien dibanding dengan Algoritma Brute Force. Prinsip pada algoritma KMP adalah menentukan jumlah pergeseran *pattern* saat mencocokan karakter dengan teks yang nantinya ada proses tambahan saat mencocokan karakter yaitu dengan memakai fungsi pinggiran. [1]

Sebelum kita melakukan perbandingan antar karakter *pattern* dengan teks, algoritma ini akan menghitung fungis pinggiran tiap – tiap karakter pada *pattern* terlebih dahulu. Fungsi pinggiran nantinya akan membalikan atau *return* sebuah angka yang

merepresentasikan ukuran dari prefik terbesar *pattern* yang juga merupakan suffix dari *pattern*, guna dari fungsi pinggiran adalah untuk menghindari pengulangan pencocokan atau perbandingan prefix yang ada pada suffix. Berikut merupakan algoritma untuk menghitung fungsi pinggiran dalam Bahasa Java :

```
public static int[] computeFail(String pattern)
{
  int fail[] = new int[pattern.length()];
  fail[0] = 0;

  int m = pattern.length();
  int j = 0;
  int i = 1;
  while (i < m) {
    if  (pattern.charAt(j) ==
         pattern.charAt(i)){  //j+1 chars match
      fail[i] = j + 1;
      i++;
      j++;
    }
    else if (j > 0) // j follows matching prefix
      j = fail[j-1];
    else {      // no match
      fail[i] = 0;
      i++;
    }
  }
    return fail;
} // end of computeFail()
```

Gambar 4: Fungsi Pinggiran pada Algoritma KMP

Sumber: [1]

Setelah membentuk fungsi pinggiran, barulah algoritma KMP mulai dijalankan, langkah – langkahnya adalah sebagai berikut :

a. Sejajarkan P[i] (*pattern*) dengan T[j] (teks) dimana i = j = 1.

b. Pencocokan pattern akan bergerak dari kiri ke kanan dengan mencocokan setiap karakter dari pattern yaitu P[i] dengan T[j], kemudian P[i + 1] dengan T[j + 1], dst sampai :

   i. pattern sudah ditemukan didalam teks, atau

   ii. terdapat perbedaan karakter saat pencocokan atau mismatch

c. Apabila menemui mismatch, *pattern* akan melakukan pergeseran sebanyak indeks karakter terakhir yang cocok dikurangi dengan fungsi pinggirannya karakter terakhir yang cocok tersebut. Jika mismatch yang terjadi adalah pada P[1], maka *pattern* akan geser ke kanan sebanyak 1 karakter.

d. Apabila teks sudah habis dan *pattern* masi belum ditemukan di dalam teks, maka akan membalikan atau return nilai yang melambangkan *pattern not found*.

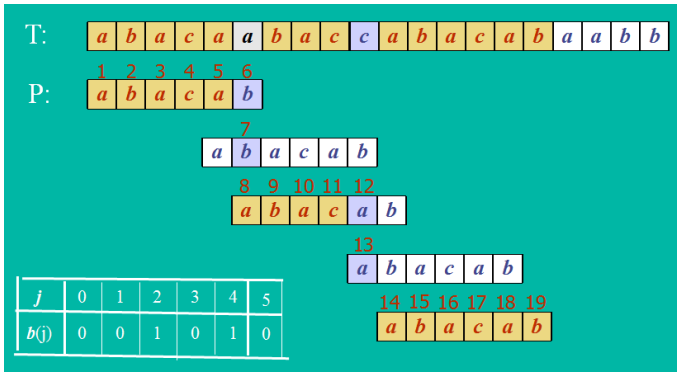
Berikut adalah Algoritma KMP dalam Bahasa Java :

```
public static int kmpMatch(String text, String pattern)
{
  int n = text.length();
  int m = pattern.length();

  int fail[] = computeFail(pattern);

  int i=0;
  int j=0;
  while (i < n) {
    if (pattern.charAt(j) == text.charAt(i)) {
      if (j == m - 1)
        return i - m + 1; // match
      i++;
      j++;
    }
    else if (j > 0)
       j = fail[j-1];
    else
      i++;
  }
  return -1; // no match
} // end of kmpMatch()
```

Gambar 5: Algoritma KMP dalam Bahasa Java

Sumber: [1]

Berikut adalah contoh kasus *pattern matching* yang diselesaikan dengan Algoritma KMP :



Gambar 6: Contoh kasus yang diselesaikan dengan Algoritma KMP

Sumber: [1]

Kompleksitas waktu yang dihasilkan oleh Algoritma KMP adalah O(m + n) dengan O(m) untuk mencari fungsi pinggiran dan O(n) untuk pencocokan *pattern* dengan teks. Algoritma KMP bagus digunakan ketika variasi karakter pada teks tidak terlalu banyak seperti pencarian sub-DNA pada keseluruhan DNA, tetapi ketika variasi karakter pada teks bertambah, waktu kerja KMP pun semakin lama dan menjadi tidak efisien. [1]

3. Algoritma Boyer Moore

Algoritma Boyer Moore merupakan algoritma yang dinilai cukup cepat untuk *pattern matching*. Algoritma ini terdiri dari 2 teknik, yaitu yang pertama teknik

looking-glass, yang memiliki arti bahwa pencocokan *pattern* dengan teks dimulai dari karakter *pattern* akhir ke karakter yang depan atau dapat bilang dari P[m] ke P[1]. Teknik kedua pada Algoritma Boyer Moore adalah *character-jump*. Teknik ini adalah teknik dalam pergeseran saat pencocokan string dan terbagi menjadi 3 macam kasus, yaitu :

a. Kasus pertama ketika P[i] terjadi mismatch dengan T[i], tetapi T[i] ada pada sebelah kiri P[i], sehingga dilakukan pergeseran sehingga P[i] dan T[i] menjadi sejajar dan tidak lagi mismatch.

b. Kasus kedua adalah ketika P[i] terjadi mismatch dengan T[i], tetapi T[i] tidak ada pada sebelah kiri P[i], melainkan berada pada sebelah kanan P[i], sehingga dilakukan pergeseran sebanyak ke kanan sebanyak 1 kali / 1 karakter.

c. Kasus ketiga adalah ketika kasus yang terjadi bukanlah kasus pertama dan kasus kedua yaitu ketika P[i] terjadi mismatch dengan T[i], dan T[i] tidak ada pada sebelah kiri maupun kanan P[i], sehingga dilakukan pergeseran dengan posisi indeks awal *pattern* (P[0]) menjadi indeks ke i + 1 pada T

Berikut merupakan Algoritma Boyer Moore dalam Bahasa Java :

```
public static int bmMatch(String text, String pattern)
{
  int last[] = buildLast(pattern);
  int n = text.length();
  int m = pattern.length();
  int i = m-1;

  if (i > n-1)
   return -1;  // no match if pattern is
               // longer than text
  int j = m-1;
     do {
       if (pattern.charAt(j) == text.charAt(i))
         if (j == 0)
           return i; // match
         else { // looking-glass technique
           i--;
           j--;
         }
       else { // character jump technique
          int lo = last[text.charAt(i)];  //last occ
          i = i + m - Math.min(j, 1+lo);
          j = m - 1;
       }
     } while (i <= n-1);

     return -1; // no match
  }// end of bmMatch()
```

Gambar 7: Algoritma Boyer Moore dalam Bahasa Java

Sumber: [1]

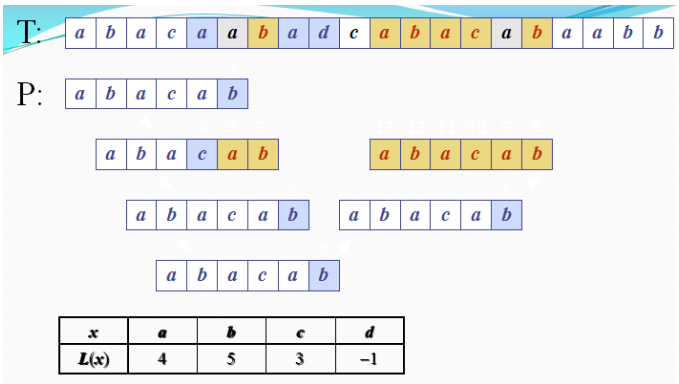
Di sebelumnya, disebutkan bahwa dalam menentukan pergeseran ketika harus tau akan posisi T[i] dan P[i], apakah T[i] yang mismatch berada di sebelah kiri P[i] atau berada disebelah kanan P[i] atau bahkan tidak ada di keduanya. Oleh karena itu, dibutuhkan suatu fungsi untuk menghitung letak dari P[i]. Fungsi tersebut dinamakan sebagai Fungsi Last Occurrence, yaitu fungsi yang menghitung letak atau posisi terakhir indeks dari karakter – karakter yang ada pada *pattern* dan akan mengembalikan -1 apabila tidak ada pada *pattern*. Berikut adalah Algoritma Fungsi Last Occurrence dalam Bahasa Java :

```
public static int[] buildLast(String pattern)
/* Return array storing index of last
  occurrence of each ASCII char in pattern. */
{
  int last[] = new int[128]; // ASCII char set

  for(int i=0; i < 128; i++)
    last[i] = -1; // initialize array

  for (int i = 0; i < pattern.length(); i++)
    last[pattern.charAt(i)] = i;

  return last;
} // end of buildLast()
```

Gambar 8: Algoritma Fungsi Last Occurrence dalam Bahasa Java

Sumber: [1]

Berikut adalah contoh kasus *pattern matching* yang diselesaikan dengan Algoritma Boyer Moore :



x	a	b	c	d
L(x)	4	5	3	-1

Gambar 9: Contoh kasus yang diselesaikan dengan Algoritma Boyer Moore

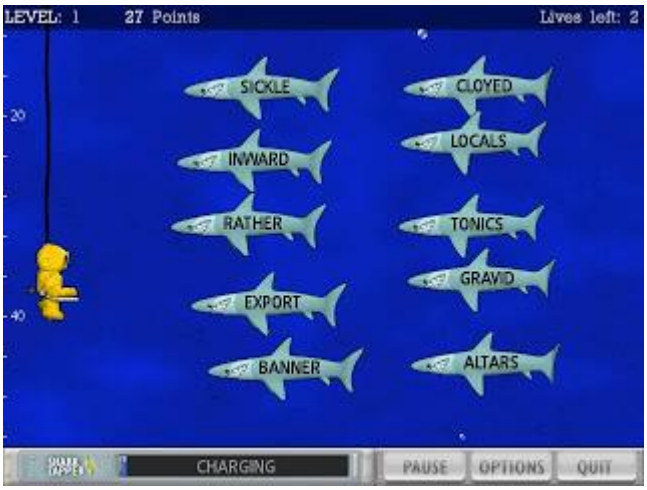
Sumber: [1]

Kompleksitas waktu terburuk pada Algoritma Boyer Moore adalan O(nm + A) dengan A merupakan jumlah alphabet yang ada. Algoritma Boyer Moore bagus digunakan untuk *pattern matching* pada kalimat – kalimat atau artike yang terdapat banyak variasi karakter dan lebih cepat dari Algoritma Brute Force, tetapi jelek ketika digunakan untuk pencarian seperti pencocokan binary. [1]

III. PENERAPAN ALGORITMA PATTERN MATCHING

Algoritma *Pattern Matching* akan terpakai ketika pemain berusaha membunuh predator seperti hiu, piranha, dan lain – lain di dalam laut. Proses *pattern matching* akan terus dilakukan sampai panjang *pattern* sama dengan panjang teks

pada badan hiu. Berikut adalah contoh permainan dari Typer Shark Deluxe :



Gambar 9: Contoh permainan dari Type Shark Deluxe

Sumber:

(http://4.bp.blogspot.com/-tNO_8AKTfiI/UKYN4kWHx0I/AAAAAAAAAZc/QSea9W2iZ-A/s320/download-typer-shark-deluxe-full-version-free.jpg)

Misalkan pada teks string "SICKLE" :

- Pertama kita harus memasukan karakter yang sesuai dengan teks string, asumsikan karakter yang kita masukan pertama kali benar yaitu karakter S, berarti nantinya program mengecek apakah S berada teks string atau tidak, jika ya maka akan meminta inputan lebih lanjut ke user, apabila user memberi inputan I, maka akan melakukan *pattern matching* "SI" pada teks string, apabila user memberi inputan A, maka akan melakukan *pattern matching* "SA" pada teks string, begitu seterusnya sampai panjang karakter *pattern* sama dengan panjang karakter teks string pada badan predator dan *pattern*-nya match dengan teks string.
- Apabila terjadi mismatch pada teks string maka akan muncul pesan error mismatch, dan akan meminta user untuk memberi masukan ulang sampai *pattern* yang ada benar atau cocok dengan teks string.

Gambar 10: Contoh ketika mismatch karakter 'M'

Sumber:

(https://img.youtube.com/vi/4Hr6drzeFgs/0.jpg)

- Pada saat pencocokan string, sayangnya Algoritma Boyer Moore tidak dapat dipakai, dikarenakan *pattern matching* pada game ini memiliki 2 kondisi yaitu :
  - Selalu dicocokan dari kiri ke kanan
  - Apabila terjadi mismatch, tidak akan ditelusuri lagi, langsung mengirim pesan error mismatch
- Ketika predator sudah mati, maka pemain akan mendapatkan score dan bisa berlanjut ke teks string yang ada pada predator selanjutnya.

Gambar 11: Contoh ketika predator dibunuh

Sumber:

(http://cdn.download-free-games.com/cf/images/nfe/screens/typershark_deluxe_2_m.jpg)

IV. KESIMPULAN

Algoritma *Pattern Matching* merupakan algoritma yang cukup banyak digunakan untuk menyelesaikan berbagai macam persoalan, salah satunya adalah dalam game Typer Shark Deluxe. Dalam game tersebut digunakan Algoritma *Pattern Matching* untuk membunuh predator – predator yang ada dengan memberi masukan *pattern* yang sesuai dengan teks string dari predator. Dengan ini pemain dapat menikmati game tersebut baik secara sekedar mengisi waktu luang, menghilangkan kejenuhan, ataupun untuk edukasi.

UCAPAN TERIMAH KASIH

Pertama – tama saya mengucapkan terima kasih kepada Tuhan Yang Maha Esa yang telah melimpahkan rahmat dan hidayah-Nya sehingga makalah Strategi Algoritma ini dapat diselesaikan tepat pada waktunya. Selain itu, saya juga ingin berterima kasih kepada orang tua dan rekan – rekan saya yang telah memberikan semangat dan dorongan sehingga saya dapat menempuh pendidikan sampai saat ini. Tak lupa saya juga ingin mengucapkan terima kasih kepada Ibu Masayu Leylia Khodra yang telah mengajarkan mata kuliah IF2211 Strategi Algoritma pada semester 2 tahun 2016/2017 ini sehingga dengan dengan ilmu yang telah diajarkan, saya dapat membuat dan menyelesaikan makalah ini.

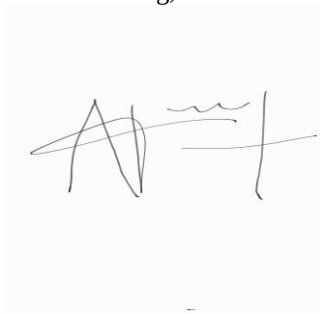
REFERENCES

[1] Munir, Rinaldi. 2004. *Bahan Kuliah ke-3 IF2251 Strategi Algoritmik Algoritma Greedy*. Bandung: Institut Teknologi Bandung.

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, 19 Mei 2017



Aditya Pratama, 13515103