Pencocokan Poligon Menggunakan Algoritma Pencocokan String

Wiwit Rifa'i 13513073[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]wiwitrifai@students.itb.ac.id

***Abstrak*—Algoritma pencocokan string tidak hanya bisa digunakan untuk mencocokkan string teks. Akan tetapi algoritma pencocokan string memiliki banyak kegunaan lainnya seperti untuk mencocokkan poligon. Poligon adalah bentuk yang paling sederhana yang bisa menggambarkan benda-benda di sekitar kita. Oleh karena itu, membandingkan poligon merupakan hal mendasar yang harus bisa dilakukan oleh komputer pintar yang mampu mendeteksi benda berdasarkan gambar. Dalam makalah ini akan dibahas mengenai salah satu cara untuk membandingkan poligon menggunakan algoritma pencocokan string.**

***Kata Kunci*—String Matching, Poligon, Knuth-Morris-Pratt, Boyer-Moore.**

I. PENDAHULUAN

Setiap benda di sekitar kita memiliki bentuk yang unik. Terdapat berbagai macam bentuk benda yang sering kita temui. Bentuk-bentuk tersebut bisa berupa ruang tiga dimensi ataupun dua dimensi. Bentuk benda sering digunakan untuk membandingkan benda satu dengan benda lainnya. Misalnya kita sering menyebut macan sebagai kucing besar karena bentuknya yang sangat menyerupai bentuk kucing namun hanya berbeda ukurannya. Selain itu, pencocokan bentuk ini digunakan pada sejenis komputer pintar yang bisa mendeteksi benda dengan mencocokkan bentuk benda tersebut terhadap sesuatu pola. Salah satu bentuk sederhana yang bisa digunakan mendeteksi suatu benda adalah bentuk poligon.

Poligon atau segi banyak adalah salah satu bentuk bangun geometri pada bidang datar. Poligon tersusun atas sejumlah titik dan segmen garis atau sisi yang berurutan dan siklis. Poligon setidaknya memiliki 3 buah titik dan 3 buah sisi. Dengan bentuk poligon kita bisa membuat berbagai macam gambar yang bisa merepresentasikan benda-benda disekitar kita. Bahkan bentuk lingkaran bisa didekati dengan poligon reguler yang memiliki banyak sisinya mendekati jumlah tak hingga.

Oleh karena itu, mencocokkan dua buah poligon merupakan hal yang dasar yang harus bisa dilakukan ketika kita ingin membuat komputer bisa mendeteksi benda-benda disekitarnya. Meskipun sudah banyak berbagai macam algoritma untuk mendeteksi bentuk yang lebih canggih seperti *Face Recognition*, akan tetapi mendeteksi bentuk dari poligon merupakan hal dasar cukup penting.

II. DASAR TEORI

Algoritma pencocokan string atau *string matching* adalah algoritma yang melakukan pencocokan 2 buah string yaitu sebuah string *pattern* dan string *text* untuk mengetahui apakah string *pattern* muncul pada string *text*.

Pencocokan string ini tidak hanya digunakan untuk mencocokan teks atau sekumpulan karakter, namun bisa juga digunakan untuk mencocokkan sekumpulan angka. Bahkan dalam makalah ini penulis mencoba memanfaatkan algortima pencocokan string ini untuk pencocokan poligon.

Algoritma pencocokan string dapat dikelompokan menjadi dua kategori, yaitu *Exact String Matching* dan *Approximate String Matching*. Algoritma *Exact String Matching* mencocokan dengan tepat sesuai *pattern*. Sedangkan Algoritma *Approximate String Matching* melakukan pencocokan string dengan perkiraan. Ada beberapa contoh algoritma *Exact String Matching* yang bisa digunakan seperti algoritma *Knuth-Morris-Pratt* dan *Boyer-Moore*.

A. Algoritma Knuth-Morris-Pratt

Algoritma ini mencocokan *pattern* dari kiri ke kanan. Misalkan kita akan mencocokan 2 buah string, yaitu string *pattern* P dengan panjang string m dan string teks T dengan panjang string n. Mula-mula kita *pattern* P dicocokkan pada awal teks T. Pencocokan *pattern* dilakukan dari kiri ke kanan dengan membandingkan setiap karakter yang saling bersesuaian. Jika semua karakter yang dibandingkan cocok maka pencocokan selesai.

Namun jika terdapat ketidakcocokan pada suatu karakter maka *pattern* perlu dilakukan pergeseran ke kanan untuk mencari substring dari teks T yang cocok dengan *pattern*. Misalkan indeks terakhir yang gagal adalah indeks ke-X. Agar pergeseran optimal maka perlu diketahui panjang prefiks dan suffiks terbesar dari string P[1] sampai P[X-1] dengan prefiks dan suffiks tersebut saling cocok. Kemudian pergeseran *pattern* dilakukan sedemikian sehingga posisi prefiks tersebut sekarang bersesuaian dengan posisi suffiks tersebut. Hal tersebut terus dilakukan sampai diketemukan substring teks yang cocok dengan *pattern* atau semua teks telah dibandingkan dengan *pattern*.

Agar pergeseran lebih mudah maka sebelum menjalankan algoritma KMP, perlu dilakukan perhitungan awal mengenai prefiks dan suffiks terpanjang seperti pada

penjelasan sebelumnya. Hasil perhitungan ini biasa disebut sebagai fungsi pinggiran (*border function*).

Kompleksitas untuk menghitung fungsi pinggiran adalah O(m) dengan m menyatakan panjang string *pattern*. Dan kompleksitas waktu untuk melakukan pencarian pada algoritma ini adalah O(n) dengan n menyatakan panjang string teks. Oleh karena itu kompleksitas total algoritma *Knuth-Morris-Pratt* adalah O(m+n). Keuntungan algoritma KMP adalah algoritma ini tidak pernah mundur pada string teks. Namun algoritma ini kurang bagus ketika jumlah alphabetnya lebih banyak.

B. *Algoritma Boyer-Moore*

Algoritma *Boyer-Moore* mencocokkan string dengan bergerak dari kanan ke kiri. Algoritma ini bekerja berdasarkan dua buah teknik yaitu teknik *looking-glass* dan teknik *character-jump*. Teknik *looking-glass* artinya pencocokan *pattern* pada teks dilakukan dengan mencocokkan dari karakter akhir *pattern* bergerak mundur ke awal *pattern*. Dan teknik *character-jump* maksudnya adalah ketika terjadi ketidakcocokan antara karakter pada *pattern* dengan teks yang bersesuaian maka string *pattern* akan lompat ke depan dengan panjang tertentu. Panjang lompatan ini bergantung pada kemunculan terakhir dari karakter pada teks yang mengalami ketidakcocokan tersebut sebut saja karakter tersebut sebagai X. Panjang lompatan ini akan dijelaskan pada 3 buah kasus berikut.

1) Jika *pattern* mengandung X pada posisi tertentu maka *pattern* akan lompat/geser ke kanan sehingga kemunculan X pada *pattern* bersesuaian dengan karakter X pada teks.
2) Jika *pattern* berisi X pada posisi tertentu tetapi tidak dimungkinkan untuk bergeser ke kanan maka *pattern* akan digeser sebanyak 1 karakter.
3) Jika kedua kasus tersebut tidak terjadi maka geser *pattern* ke kanan sebanyak panjang *pattern* tersebut.

Agar lebih mudah mengetahui kemunculan terakhir dari karakter-karakter yang muncul pada teks maka sebelum algortima *Boyer-Moore* dijalankan maka perlu dicari terlebih dahulu kemunculan terakhir pada *pattern* untuk setiap karakter-karakter yang muncul pada teks. Hasil dari perhitungan ini biasa disebut sebagai fungsi *last occurance*.

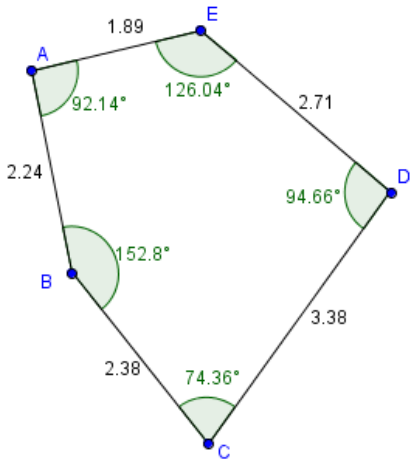
Pada kasus terburuk, kompleksitas waktu algoritma ini bisa mencapai O(n*m + A) dengan n adalah panjang string teks, m adalah panjang string *pattern*, dan A adalah ukuran alfabet. Algoritma *Boyer-Moore* cenderung akan lebih cepat jika banyaknya jumlah alfabetnya lebih besar dan akan lebih lambat jika jumlah alfabetnya kecil.

III. Analisis Masalah

A. *Representasi Poligon Agar Bisa Dilakukan String Matching*

Sejauh ini untuk melakukan pencocokan string kita menggunakan 2 buah teks atau kumpulan karakter. Selain itu algoritma pencocokan string juga bisa digunakan untuk pencocokan kumpulan objek-objek lain yang bisa dibandingkan seperti pencocokan pada kumpulan bilangan. Sehingga agar kita bisa menggunakan pencocokan string untuk mencocokkan poligon, kita perlu merepresentasikan poligon sebagai kumpulan objek yang dapat dibandingkan dengan cukup mudah.

Representasi yang akan dipilih adalah merepresentasikan poligon sebagai kumpulan bilangan. Suatu poligon bisa diidentifikasi dengan sisi-sisi dan sudut yang diapit oleh sisi-sisi yang berhubungan. Sehingga kita bisa kita bisa merepresentasikan poligon sebagai kumpulan bilangan yang merupakan panjang tiap sisi atau sudut dari dua sisi yang saling terhubung secara berselang-seling dan sesuai urutannya.



Gambar 1 : Contoh poligon

Sebagai contoh kita gunakan poligon yang ada pada gambar 1 yang merupakan sebuah pentagon. Poligon tersebut bisa direpresentasikan sebagai kumpulan bilangan { 2.24, 152.8º, 2.38, 74.36º, 3.38, 94.66º, 2.71, 126.04º, 1.89, 92.14º }. Karena terdapat dua macam sisi sudut maka diperlukan suatu standar dalam pengambilan sudut sehingga semua sudut diambil secara menjadi seragam. Berikut ini adalah fungsi-fungsi yang bisa digunakan untuk menghitung sudut tersebut (dalam bahasa C++).

```
// cross Products
double cross(Point p, Point q, Point r) {
  return (r.x-q.x)*(p.y-q.y) - (r.y-
q.y)*(p.x-q.x);
}
// dot product
double dot(Point p, Point q, Point r) {
  return (p.x-q.x)*(r.x-q.x) + (p.y-
q.y)*(r.y-q.y);
}
// distance
double distance(Point p, Point q) {
  return sqrt((p.x-q.x)*(p.x-q.x) + (p.y-
q.y)*(p.y-q.y));
}
//menghitung sudut dari sisi pq dan qr
double angle(Point p, Point q, Point r) {
  double sudut = acos(dot(p,q,r)/
(distance(p,q)*distance(q,r)));
  sudut = 180.0 * sudut / PI;
```

```
  if(cross(p,q,r)<0) {
    sudut = 360.0 - sudut;
  }
  return sudut;
}
```

Selain kita bisa memilih sudut bagian mana yang akan diambil, kita juga bisa memilih arah jalur mana yang akan digunakan yaitu jalur yang searah jarum jam atau yang berlawanan arah jarum jam. Jadi terdapat 4 kemungkinan representasi dari suatu poligon. Sehingga dalam mencocokkan poligon maka untuk salah satu poligon kita harus mencoba keempat kemungkinan tersebut dan poligon lainnya hanya memilih satu representasi saja dengan panjangnya menjadi 2 kalinya.

Dan untuk mencocokkan poligon tentu kita perlu menentukan batasan-batasan yang harus dipenuhi agar bisa dikatakan bahwa dua buah poligon itu sama atau cocok. Dalam makalah ini dua buah poligon akan dianggap sama jika poligon yang satu merupakan hasil dari translasi, rotasi, ataupun refleksi dari poligon yang satu lagi.

Ketika suatu poligon dilakukan translasi maka representasi poligon sebagai kumpulan bilangan seperti pada subbab sebelumnya tidak akan mengalami perubahan. Sedangkan ketika poligon dirotasikan maka representasi tadi juga tidak akan berubah. Namun jika poligon direfleksikan/dicerminkan maka perubahannya tetap akan menjadi salah satu representasi poligon dari keempat kemungkinan yang ada.

B. *Implementasi Knuth-Morris-Pratt untuk Pencocokan Poligon*

Karena dalam pencocokan ini yang dibandingkan adalah berupa bilangan riil maka ketika membandingkan antar bilangan riil, kita perlu memberikan batas maksimum selisih dari kedua bilangan tersebut sehingga dua bilangan riil bisa dikatakan cocok. Hal ini perlu dilakukan karena operasi bilangan riil pada komputer akan memberikan sejumlah galat pada perhitungan sebab kemampuan komputer untuk menyimpan bilangan riil terbatas.

Berikut ini adalah potongan kode sumber dari implementasi algoritma *Knuth-Morris-Pratt* untuk pencocokan Poligon.

```
// Nilai Galat maksimum
const double EPS = 1e-3;

// Prosedur KMP, akan bernilai true jika
pattern cocok dengan text
bool kmp(vector<double> pattern,
vector<double> text) {
   // menghitung border function dari
pattern
   vector<int> b;
   b.assign(pattern.size(), 0);
  int i = 1, j = 0;
  while(i<pattern.size()) {
    if(fabs(pattern[i]-pattern[j]) < EPS)
{
      b[i++] = ++j;
    }
    else if(j > 0){
      j = b[j-1];
    }
    else {
      b[i++] = 0;
    }
  }
  // Algoritma pencocokan string
  i = j = 0;
  while(i<text.size()) {
    if(fabs(text[i]-pattern[j]) < EPS) {
      i++;
      j++;
      if(j == pattern.size())
        return true;
    }
    else if(j > 0) {
      j = b[j-1];
    }
    else
      i++;
  }
  return false;
}
```

Algoritma *Knuth-Morris-Pratt* secara teori memang kurang cocok untuk jumlah alfabet yang banyak. Terlebih lagi dalam bilangan riil terdapat tak hingga banyaknya nilai yang mungkin untuk dipakai. Namun pada prakteknya algoritma tetap mampu berjalan dengan baik dan cukup cepat karena seperti yang kita ketahui bahwa kompleksitas algoritma *Knuth-Morris-Pratt* ini adalah O(n+m).

C. *Implementasi Boyer-Moore untuk Pencocokan Poligon*

Implementasi algoritma *Boyer-Moore* untuk pencocokan poligon akan sedikit lebih sulit. Hal ini dikarenakan elemen yang dibandingkan adalah bilangan riil dengan menggunakan galat tertentu. Sehingga kita tidak bisa menggunakan *array* biasa untuk menyimpan fungsi *last occurance*. Namun implementasinya akan sedikit berbeda dari biasanya, yaitu *array* terurut dari dua buah bilangan. Bilangan pertama adalah bilangan yang biasanya akan muncul dalam *pattern*. Dan bilangan kedua adalah bilangan yang menunjukkan indeks last occurance dari bilangan pertama tadi pada *pattern*. Untuk mendapatkan indeks las occurance tersebut tidak bisa dilakukan dengan O(1). Namun harus menggunakan *binary search* yang memiliki kompleksitas waktu $O(^2\log(n))$.

Berikut ini adalah potongan kode sumber dalam bahasa C++ yang mengimplementasikan algoritma Boyer-Moore untuk pencocokan Poligon.

```
// last occurance untuk algoritma boyer-
moore
vector< pair< double, int > > lastOccur;
// fungsi untuk mendapatkan last
occurance dengan keompeksitas waktu
O(2_log(n))
int getLastOccur(double x) {
  int low, up, mid;
  low = 0; up = lastOccur.size()-1;
```

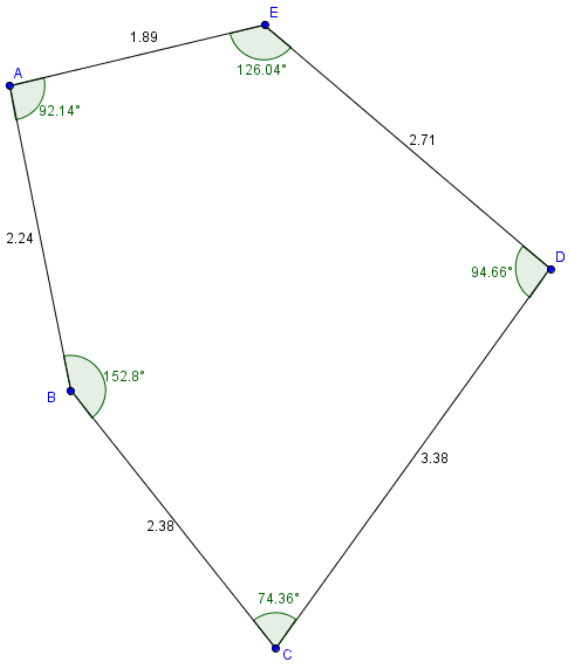
```
  while(low + 1 < up) {
    mid = (low+up)/2;
    if(fabs(lastOccur[mid].first-x) <
EPS) {
      return lastOccur[mid].second;
    }
    else if(lastOccur[mid].first < x)
      low = mid;
    else
      up = mid;
  }
  if(fabs(lastOccur[low].first-x) < EPS)
    return lastOccur[low].second;
  else if(fabs(lastOccur[up].first-x) <
EPS)
    return lastOccur[up].second;
  else
    return -1;
}
bool boyer_moore(vector<double> pattern,
vector<double> text) {
  // Menghitung last occurance
  lastOccur.clear();
  int i, j;
  for(i = 0; i< pattern.size(); i++)

lastOccur.push_back(make_pair(pattern[i],
i));
  sort(lastOccur.begin(),
lastOccur.end());
  i = j = pattern.size()-1;
  while(i < text.size()) {
    if(fabs(text[i]-pattern[j]) < EPS) {
      if(j == 0)
        return true;
      i--;
      j--;
    }
    else {
      int lastOcc =
getLastOccur(pattern[j]);
      if(lastOcc >= 0) {
        if(lastOcc < j) // Case 1
          i += pattern.size() - lastOcc -
1;
        else // Case 2
          i += pattern.size() - j;
      }
      else // Case 3
        i += pattern.size();
      j = pattern.size()-1;
    }
  }
  return false;
}
```

Algoritma Boyer-Moore ini secara teori akan berjalan baik pada pencocokan poligon ini, karena elemen yang digunakan untuk membandingkan adalah berupa bilangan riil. Dan karena bilangan riil terdapat tak terhingga maka banyaknya "alfabet" yang terdapat pada pattern pun juga sangat banyak sehingga performa algoritma ini akan bekerja dengan baik. Hanya saja dalam pencocokan poligon ini untuk mendapatkan suatu fungsi last occurance diperlukan kompleksitas waktu lebih lama yaitu $O(^2\log(n))$.

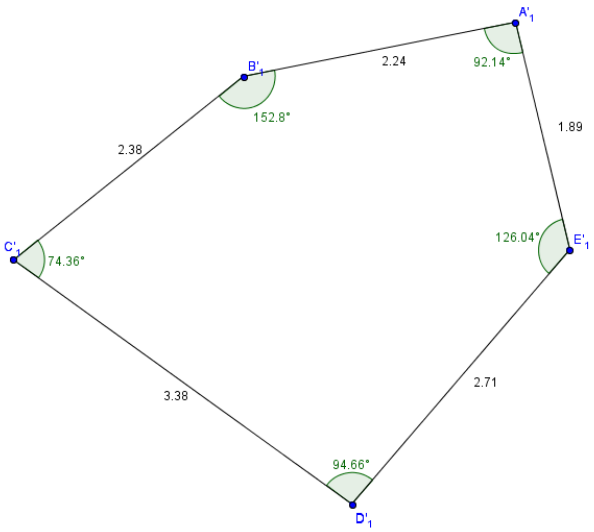
D. Pengujian

Dalam percobaan ini, suatu poligon akan didefiniskan oleh barisan titik-titik dimana titik-titik yang bersebelahan akan membentuk suatu sisi serta titik pertama dan terakhir juga akan membentuk suatu sisi.

1. Kasus Uji I



Gambar 2 : Poligon pertama pada kasus uji I



Gambar 3 : Poligon kedua pada kasus uji II

Poligon pertama (pada gambar 2) = {(2.4, 5.14), (2.84, 2.94), (4.32, 1.08), (6.3, 3.82), (4.24, 5.58),)}.

Poligon kedua (pada gambar 3) = {(12.52, 5.65), (10.76, 3.59), (8.02, 5.57), (9.88, 7.05), (12.08, 7.49)}.

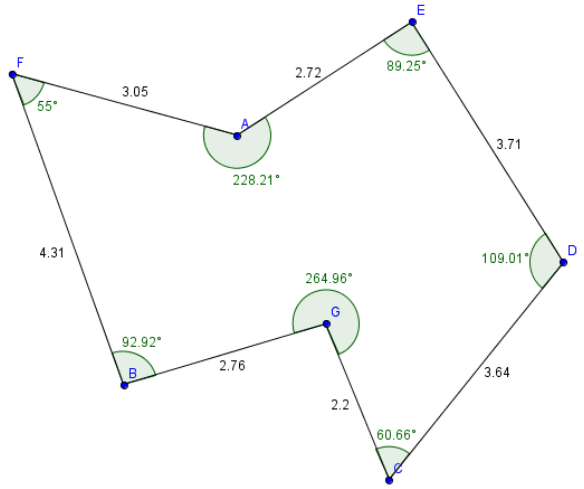
Poligon kedua merupakan poligon yang dibuat dengan mentranslasikan poligon pertama dan kemudian memutarnya 90^o searah jarum jam. Sehingga seharusnya kedua poligon tersebut adalah identik atau sama.

Dan setelah mengeksekusi program yang telah dibuat maka diperoleh hasil sebagai berikut.

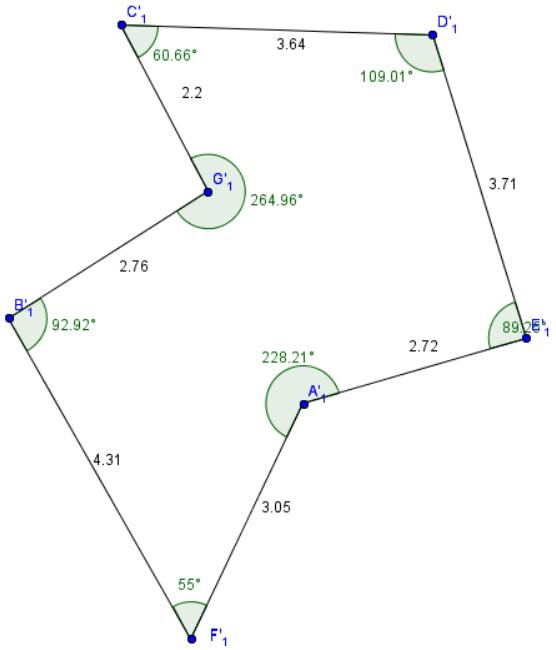
```
E:\TUGAS\Makalah Stima>g++ a.cpp -o e

E:\TUGAS\Makalah Stima>e < in.txt
Poligon I : { (2.4, 5.14),(2.84, 2.94),(4.32, 1.08),(6.3, 3.82),(4.24, 5.58)}
Poligon II : { (12.52, 5.65),(10.76, 3.59),(8.02, 5.57),(9.88, 7.05),(12.08, 7.4
9)}

*** Allgoritma KMP ***
Kedua poligon masukan sama
Jumlah proses membandingkan : 60
Waktu eksekusi = 0 ms

*** Allgoritma Boyer-Moore ***
Kedua poligon masukan sama
Jumlah proses membandingkan : 39
Waktu eksekusi = 0 ms

E:\TUGAS\Makalah Stima>
```

Gambar 3 : Hasil Kasus Uji I

2. Kasus Uji II



Gambar 4 : Poligon pertama pada kasus uji II



Gambar 5 : Poligon kedua pada kasus uji II

Poligon pertama = { (5.32, 6.18),(7.6, 7.66),(9.58, 4.52),(7.3, 1.68),(6.48, 3.72),(3.84, 2.92),(2.38, 6.98)}

Poligon kedua = { (11.55, -7.38),(14.16, -6.62),(13.07, -3.07),(9.43, -2.95),(10.44, -4.9),(8.12, -6.38),(10.25, -10.13)}

Poligon kedua merupakan poligon yang dibuat dari hasil refleksi atau pencerminan poligon pertama terhadap suatu garis lurus. Sehingga seharusnya kedua poligon di atas dianggap sama atau identik. Dan setelah program dieksekusi ternyata hasilnya sesuai.

```
E:\TUGAS\Makalah Stima>e < in.txt
Poligon I : { (5.32, 6.18),(7.6, 7.66),(9.58, 4.52),(7.3, 1.68),(6.48, 3.72),(3.
84, 2.92),(2.38, 6.98)}
Poligon II : { (11.55, -7.38),(14.16, -6.62),(13.07, -3.07),(9.43, -2.95),(10.44
, -4.9),(8.12, -6.38),(10.25, -10.13)}

*** Allgoritma KMP ***
Kedua poligon masukan sama
Jumlah proses membandingkan : 54
Waktu eksekusi = 0 ms

*** Allgoritma Boyer-Moore ***
Kedua poligon masukan sama
Jumlah proses membandingkan : 29
Waktu eksekusi = 0 ms

E:\TUGAS\Makalah Stima>_
```

Gambar 6 : Hasil Kasus Uji II

V. KESIMPULAN

Algoritma *String Matching* seperti *Knuth-Morris-Pratt* dan *Boyer-Moore* tidak hanya digunakan untuk membandingkan huruf ataupun karakter. Tetapi juga bisa digunakan untuk mencocokkan dua buah poligon seperti yang sudah dijelaskan dalam makalah ini. Pencocokan poligon ini bisa dikembangkan lagi dan digunakan untuk mencocokkan bentuk-bentuk benda yang lebih kompleks.

VII. PENUTUP

Ucapan terima kasih penulis berikan kepada Tuhan Yang Maha Esa, Institut Teknologi Bandung dan kedua dosen mata kuliah IF2211-Strategi Algoritma yaitu Bapak Rinaldi Munir dan Ibu Ulfa Maulidevi. Diharapkan makalah ini dapat memberi manfaat bagi pembaca dan menginspirasi untuk dibuat pengembangannya yang lebih lanjut.

REFERENCES

[1] Munir, Rinaldi, 2009. "Diktat Kuliah IF2211 Strategi Algoritma," Program Studi Teknik Informatika STEI ITB

[2] Weisstein, Eric W. "Polygon." From MathWorld--A Wolfram Web Resource. http://mathworld.wolfram.com/Polygon.html diakses pada 5 Mei 2015 pukul 13.13 WIB.

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, 5 Mei 2015

Wiwit Rifa'i 13513073