APLIKASI ALGORITMA *BACKTRACKING* DALAM PERMAINAN ANAGRAM

Muhammad Ikhsan Assaat – NIM 13505042

Program Studi Teknik Informatika
Sekolah Teknik Elektroni dan Informatika
Institut Teknologi Bandung
Jl. Ganesha 10, Bandung
E-mail : if15042@students.if.itb.ac.id

ABSTRAK

Anagram adalah salah satu jenis permainan tebak kata. Objektif permainan anagram adalah menebak kata dengan huruf-huruf yang telah diacak. Huruf yang tersedia harus dipakai sebanyak jumlahnya. Contoh yang sederhana, pemain diberikan empat huruf ”upla”, maka pemain harus menebak kata menggunakan huruf ’u’, ’p’, ’l’, ’a’; dengan tiap huruf tepat sebanyak jumlahnya. Sehingga kata-kata yang bisa dibentuk adalah ”lupa”, ”palu”, ”pula”, ”luap”, dan seterusnya. Kata yang ditebak harus valid, valid dalam arti kata tersebut termasuk dalam suatu database yang ditentukan, misalnya terdapat dalam Kamus Besar Bahasa Indonesia. Semakin banyak tebakan kata yang benar nilainya semakin baik. Makalah akan membahas simulasi permainan tersebut dalam bentuk program.

Dalam makalah ini, kita akan memberikan salah satu solusi cara mencari kata dalam database yang cukup besar. Kita akan mengaplikasikan algoritma *backtracking* dalam pencarian string. Namun kita akan menyederhanakan beberapa persoalan yang akan dijelaskan lebih lanjut.

Kata kunci: *Algoritma Pencarian String, Backtracking, Anagram.*

1. PENDAHULUAN

Anagram adalah salah satu jenis permainan tebak kata yang cukup populer. Objektif permainan anagram adalah menebak kata dengan huruf-huruf yang telah diacak sebanyak-banyaknya. Huruf yang tersedia harus dipakai sebanyak jumlahnya. Kata yang ditebak harus valid, valid dalam arti kata tersebut termasuk dalam suatu database yang ditentukan, misalnya terdapat dalam Kamus Besar Bahasa Indonesia. Semakin banyak tebakan yang benar maka nilainya semakin baik.

Program ini akan menyederhanaan beberapa persoalan. Penyederhanaan pertama adalah penyederhanaan database. Database tidak memasukkan kata berimbuhan, tetapi database hanya akan menangani kata dasar. Karena indeks (urutan) dalam database sangat berpengaruh dalam algoritma pencarian, sedangkan kata berimbuhan terletak di indeks kata dasarnya (seperti dalam KBBI). Penyederhanaan kedua adalah jumlah huruf dalam kata yang diberikan minimal empat dan maksimal delapan. Database disini juga tidak akan menampung kata lebih dari 100.000 untuk meminimalisasi pencarian.

Program permainan anagram akan melibatkan tiga elemen utama. Pertama adalah pemain, kedua adalah komputer dalam arti program itu sendiri, dan ketiga adalah database kata. Sebagai informasi, database tersebut telah terurut secara alfabet.

Program akan memilih secara acak satu kata dari database, lalu indeks huruf dari kata tersebut diacak. Langkah berikutnya adalah membangun *list* kata. *List* kata adalah kumpulan kata yang dapat disusun oleh huruf-huruf yang telah diacak dan terdapat di *database*. Setelah mendapatkan *list* kata tersebut, pemain menebak kata apa yang dapat dibangun. Selanjutnya program tinggal mencocokan apakah kata yang dimasukkan pemain terdapat di *list* kata atau tidak.

2. METODE

2.1 Algoritma *Bactracking*

Backtracking adalah algoritma berbasis DFS untuk mencari solusi persoalan. Backtracking merupakan algoritma perbaikan dari brute-force, yang secara sistematis hanya mencari solusi yang mungkin. Umumnya algoritma *Backtracking* bersifat rekursif, namun ada pula versi *backtracking* yang iteratif.

Langkah-langkah pencarian solusi adalah sebagai berikut, :

1. Solusi dicari dengan membentuk lintasan dari akar ke daun. Simpul yang dilahirkan bernama **simpul hidup**. Simpul hidup yang sedang diperluas dinamakan **simpul-E**.
2. Jika lintasan yang sedang dibentuk tidak mengarah ke solusi maka simpul-E dibunuh menjadi **simpul mati**. Fungsi yang digunakan untuk membunuh fungsi adalah **fungsi pembatas**.
3. Jika pembentukan **simpul-E** terakhir adalah simpul mati maka proses pencarian diteruskan dengan membangkitkan simpul anak yang lain. Bila tidak ada simpul anak lagi maka pencarian melakukan *backtracking* ke simpul orangtuanya.
4. Pencarian berakhir bila ditemukan solusi atau tidak ada lagi simpul yang hidup untuk *backtrack.*

Berikut adalah skema umum dari algoritma backtrack versi iteratif.

```
procedure Backtrack(input n: integer)
{mencari solusi persoalan versi iteratif;
Masukan : n, yaitu panjang vektor solusi
Keluaran : solusi x =(x[1],…, x[n])
}

Deklarasi:
  k : integer

Algoritma:
  k ← 1
  while k > 0 do
    if
    (x[k] belum dicoba sampai x[k]←T[(k))
    and (B(x[1], x[2], ..., x[n]) = true)
    then
       if((x[1],x[2],...,x[k])adalah
            lintasan akar ke daun)
        then
            CetakSolusi(x)
        endif

       k ← k+1
    else
       k ← k-1
    endif
  endwhile
  {k=0}
```

2.2 Algoritma Program Permainan

Program melakukan tiga hal utama, yaitu memilih kata dari database secara acak, membangun *list* kata valid dari huruf-huruf kata yang terpilih, dan terakhir mencocokan string masukan pemain dengan list yang telah dibuat. Namun dibutuhkan beberapa data-data tambahan dan fungsi khusus untuk membantu menyelesaikan permasalahan yang ada.

Data dan fungsi tambahan yang diperlukan adalah data indeks huruf. Dalam melakukan pencarian apakah string ada atau tidak dalam database, akan sulit bila setiap pencarian dilakukan dari awal. Sama seperti halnya menandai halaman awal tiap huruf buku tebal ensiklopedi dengan jelas agar mudah dicari. Sehingga tiap huruf mempunyai indeks jumlah kata yang diawalinya. Sebagai contoh, indeks kata yang diawali huruf ‘d’ yang pertama dalam database adalah 3.325, sehingga pencarian dimulai dari indeks angka tersebut.

Fungsi tambahan berikutnya adalah fungsi pembatas indeks, dengan masukan *array* of karakter, maka fungsi akan mengembalikan indeks terendah dan tertinggi kata yang masih bisa diawali oleh masukan karakter tersebut. Sebagai contoh, bila masukan fungsi tersebut adalah “da” maka fungsi akan mengembalikan indeks pertama dan terakhir kata yang diawali “da”. Bila tidak ada kata yang bisa dibentuk maka fungsi mengembalikan angka -99 untuk kedua keluaran. Untuk efektifitas pencarian,angka indeks, algoritma pencarian yang digunakan adalah *binary search* agar waktu pencarian setara untuk setiap huruf.

Skema umum langkah-langkah program adalah sebagai berikut :

2.2.1 Pencarian Kata dan Pengacakan

Program membangun bilangan acak dari angka 1 hingga [JumlahKataDatabase]. Program akan mengacak-cak indeks huruf dari kata tersebut. Program mengacak kata dari datebase, sehingga solusi yang dapat dibentuk minimal satu buah. Contoh, bila kata yang terpilih adalah “siang”, maka kata yang bisa dibentuk adalah “siang” itu sendiri, dan “asing”.

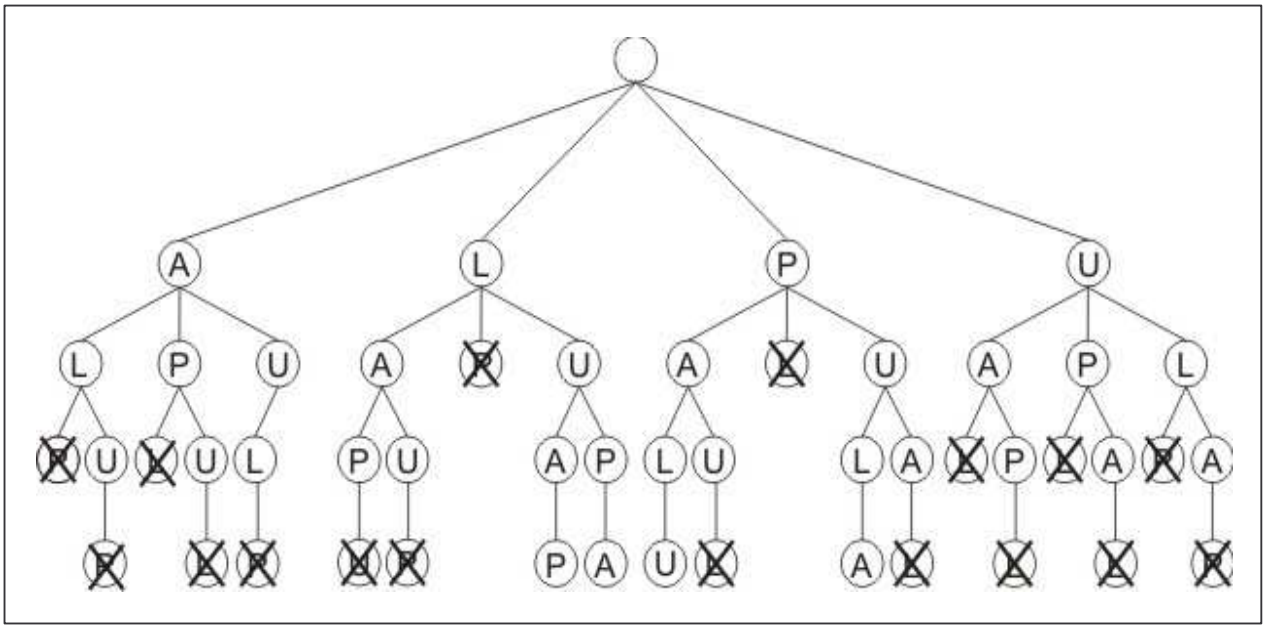
2.2.2 Pembangunan *List* of kata yang valid

Program akan membangun list kata yang mungkin dan mencocokannya dengan database. Kita telah menyederhanakan persoalan dengan membatasi jumlah huruf dalam kata yaitu minimal 4 huruf dan maksimal 8 huruf. ($4 \leq JumlahHuruf \leq 8$).

Namun bagaimana caranya membangun kata yang mungkin dan mencocokkannya dengan database? Mari kita hitung berapa kata yang terlibat untuk kasus terburuk dengan algoritma *brute force*. Misalkan jumlah huruf dari kata yang terpilih dari database sebanyak delapan, jumlah kata dalam database 100.000, huruf-huruf dari kata tersebut hanya dapat membangun kata itu sendiri. Maka kata yang dibangun sebanyak 8! dan tiap kata harus dicocokkan satu per satu. Bila proses pencocokan string memakan waktu 1 detik maka, untuk membangun *list* kata diperlukan waktu selama 8! x 100.000 = 4.032.000.000 detik. Sedangkan list hanya akan berisi satu kata, yaitu

kata yang diambil dari database. Sungguh waktu yang lama untuk memulai permainan.

Untuk itu kita akan gunakan algoritma backtracking sebagai penyempurnaan algoritma *brute force.* Kita sudah mempunyai data dan fungsi tambahan, yaitu data indeks huruf dan fungsi batas minimum maksimum. Maka kita akan membangun pohon solusi untuk membangun kata-kata yang valid. Sebagai contoh kasus, bila kata yang terpilih dari database adalah "LUPA". Maka kata yang dapat dibangun melalui pohon adalah sebagai berikut.



Gambar 1. Pohon yang dibangun untuk kata terpilih "lupa". Tanda X menandakan simpul dibunuh

Langkah-langkah dalam pembangunan pohon adalah sebagai berikut :

1. Simpul dihidupkan sesuai alfabet. Dimulai dari huruf 'A'. Maka pencarian langsung dimulai dari indeks huruf A, yaitu 1. Simpul yang dihidupkan pertama kali oleh simpul 'A' adalah simpul 'L'. Maka fungsi pembatas dipanggil dengan masukkan string "AL". Karena masih ada kata yang diawali oleh "AL" maka pencarian diteruskan dengan menghidupkan simpul 'P'.
2. Fungsi pembatas dipanggil kembali dengan masukan string "ALP". Namun fungsi mengembalikan -99 karena tidak ditemukan kata yang diawali "ALP". Simpul dibunuh, lalu *backtrack* ke simpul 'L'.
3. Hidupkan simpul 'U', lalu panggil fungsi pembatas dengan masukkan "ALU". Karena masih ada kata yang diawali "ALU" maka pencarian diteruskan.
4. Hidupkan simpul dengan huruf yang tersisa, cek dengan fungsi pembatas bila mengembalikan -99 bunuh simpul lalu *backtrack*, bila bukan -99 lanjutkan pencarian.
5. Bila string yang diperiksa fungsi pembatas sejumlah kata awal (dalam contoh ini 4) maka cek kondisinya. Solusi dianggap ditemukan bila memenuhi salah satu dari dua kondisi, kondisi pertama cek nilai batas minimum dan maksimum dari kembalian fungsi pembatas, bila sama maka solusi ditemukan dan dimasukkan dalam list. Kondisi kedua, bila batas minimum dan maksimum berbeda maka cek ke database apakah string masukkan tepat sama dengan kata dengan indeks batas minimum hasil fungsi pembatas. Bila tepat sama maka solusi dinyatakan ketemu dan dimasukkan kedalam list kata, bila tidak maka simpul dibunuh dan melanjutkan backtrack. Kondisi ini diperlukan untuk menangani kasus bila masukkan string "PUAL" ke fungsi pembatas. Fungsi tidak mengembalikan nilai -99 karena ada kata yang diawali "PUAL", contohnya "PUALAM" (batu). Setelah dicek nilai batas minimum dan maksimumnya berbeda, maka string masukkan di cek langsung ke database. Karena kata "PUAL" tidak diketemukan maka simpul dibunuh dan "PUAL" tidak dinyatakan sebagai solusi.
6. Pencarian dinyatakan berakhir bila seluruh pohon telah diiterasi.

Berikut adalah pseudo-codenya algoritma untuk pembangunan list.

```
procedure bangunList(input Kata : string,
output listKata : array of string)
```

```
{
membangun list dari huruf-huruf string
  masukkan
Masukan : string Kata,
Keluaran : array of string listKata
}

Deklarasi:
  k, min, max : integer
  strTemp : string
  idxList : integer
  dB : database string

Algoritma:
  idxList = 0
  strTemp =(huruf pertama kata dalam pohon)

  while (masih ada simpul yang bisa
        dihidupkan) do
    fungsiMinMax(strTemp, min, max)
    if (min != -99) then
       if (jumlah huruf strTemp = jumlah
          huruf Kata) then
          if (dB[min] = strTemp) then
              addList(strTemp)
          else
              (bunuh simpul, backtrack ke
simpul orangtua)
              strTemp -= (huruf terakhir)
          endif
       else
          (hidupkan simpul)
          strTemp += (huruf dari simpul
baru yang dihidupkan)
       endif
    else
       (bunuh simpul, backtrack ke simpul
orangtua)
       strTemp -= (huruf terakhir)
    endif
  endwhile
  {semua simpul mati}
```

2.2.3 Pencocokan masukan Pemain dengan *List*

Program akan mencocokkan string masukkan user dengan algoritma *brute force.* Kenapa menggunakan *brute force*? Karena pencocokan harus tepat sama dengan kata yang terdapat dalam list, satu huruf saja salah maka dinyatakan berbeda lalu dicocokkan dengan kata berikutnya yang terdapat dalam *list*. Bila tepat sama maka skor pemain ditambahkan dan kata tersebut *exclude* dari list agar tidak diperiksa kembali.

4. KESIMPULAN

Algoritma anagram lebih sulit dalam membangun kata-kata yang terdapat dalam database yang besar. Walaupun algoritma backtrack telah membuat pencarian brute force yang lebih efektif namun pencarian ke seluruh database tetap sulit. Diperlukan algoritma yang lebih mangkus daripada algoritma *brute force* dengan *backtracking*.

REFERENSI

[1] Munir, Rinaldi. (2007). Strategi Algoritmik. Program Studi Teknik Informatika, Institut Teknologi Bandung.

[2] Misra, Jayadev. (2002). Derivation of a Parallel String Matching.