PENGGUNAAN *EXHAUSTIVE SEARCH* SEBAGAI SOLUSI PERMAINAN SCRAMBLE

Mohammad Dimas (13507059)

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
Jalan Ganesha nomor 10
e-mail: if17059@students.if.itb.ac.id

ABSTRAK

***Exhaustive search* adalah teknik pencarian solusi secara *brute force* (lempang / *straight forward)* untuk masalah yang melibatkan pencarian elemen dengan sifat khusus, biasanya di antara objek-objek kombinatorik seperti permutasi, kombinasi atau himpunan bagian dari sebuah himpunan.**

Permainan scramble adalah permainan yang melibatkan papan dan huruf, pada papan berukuran 4x4 atau lebih diletakkan huruf-huruf alfabet secara acak, pemain diharuskan menemukan kata-kata yang terdapat dalam kamus bahasa yang digunakan dalam permainan, misalkan Bahasa Inggris. Kata yang dimaksud adalah kata dengan panjang minimal 3 huruf dengan huruf-huruf pada papan sebagai huruf penyusun. Dalam menyusun kata yang dimaksud, huruf-huruf penyusunnya harus saling bertetangga. Setiap kata yang ditemukan memiliki nilai berdasarkan panjang dan huruf yang dikandungnya. Untuk memenangkan permainan, pemain harus memperoleh kata-kata sebanyak mungkin dengan waktu yang disediakan.

Dalam makalah ini penulis bermaksud membahas suatu metode untuk menemukan solusi dari permainan ini, solusi yang didapat diharapkan dapat membantu pemain untuk memperoleh nilai setinggi-tingginya. Solusi yang diberikan berupa menghasilkan seluruh kata-kata yang dapat dibentuk berdasarkan huruf yang dibangkitkan oleh permainan, dan memberikan posisi huruf-huruf penyusun kata tersebut pada papan. Dalam menemukan solusi tersebut penulis akan mengimplentasikan *Exhaustive Search* sebagai bagian dari algoritma *bruteforce*.

Kata kunci: Scramble, Solusi, Exhaustive Search, brute force,.

1. PENDAHULUAN

1.1 PERATURAN PERMAINAN

Permainan *scramble* dapat dimainkan oleh satu orang ataupun lebih, namun demikian berapapun jumlah orang yang memainkannya, inti dari permainan ini tetap sama, yaitu setiap pemain harus memperoleh nilai setinggi-tingginya.

Gambar 1 tampilan permainan *scramble*

Dalam permainan ini akan terdapat papan berukuran 4x4 atau lebih dengan huruf acak di atasnya. Untuk memperoleh nilai pemain harus menemukan kata yang terdapat di kamus permainan(misalnya bahasa inggris) di dalam papan tersebut. Kata yang valid ialah kata yang panjangnya lebih dari 3 huruf dan kata tersebut terdaftar didalam kamus. Selain itu huruf-huruf penyusun kata harus membentuk suatu jalur. Misalkan kata yang dimaksud adalah *Scramble*, maka huruf s harus bertetangga dengan huruf c, huruf c bertetangga dengan huruf r dan begitu seterusnya hingga seleruh huruf dalam kata. Untuk setiap huruf yang ada di dalam papan hanya dapat dipakai satu kali dalam pembentukan suatu kata, misalkan kata yang dimaksud *doll*, maka l pertama dan l

ke-dua adalah dua l yang berbeda. Besarnya nilai yang didapat dipengaruhi oleh panjang kata dan huruf yang terdapat di dalam kata tersebut, semakin panjang kata maka semakin besar nilainya, dan adanya huruf-huruf tertentu seperti q atau z akan memberikan suatu nilai tambah.

Jika permainan ini dimainkan oleh satu orang, maka tujuan permainan ialah menciptakan rekor personal yang baru, rekor personal yang baru akan tercipta ketika rekor sebelumnya telah dilewati dengan cara mendapatkan nilai yang lebih tinggi, sedangkan jika dimainkan lebih dari satu orang, maka tiap pemain akan bertanding untuk memperoleh nilai paling tinggi.

1.2 SKEMA UMUM ALGORITMA *EXHAUSTIVE SEARCH*

Exhaustive Search adalah bentuk algoritma dasar yang dipergunakan untuk pencarian. Exhaustive search merupakan teknik pencarian solusi seccara brute force untuk masalah yang melibatkan pencarian elemen dengan sifat khusus, biasanya di antara objek-objek kombinatorik seperti permutasi, kombinasi, atau himpunan bagian dari semua himpunan. Algoritma tersebut mengiterasi seluruh kemungkinan solusi yang mungkin ada pada suatu masalah yang diberikan, memeriksa apakah solusi telah ditemukan, dan melanjutkan pencarian hingga solusi ditemukan dan mengembalikan nilai solusi tersebut.

Metode exhaustive search dapat dirumuskan langkah-langkahnya sebagai berikut

1. Enumerasi (list) setiap solusi yang mungkin dengan cara yang sistematis
2. Evaluasi setiap kemungkinan solusi satu per ssatu, mungkin saja beberapa kemungkinan solusi yang tidak layak dikeluarkan, dan simpan solusi terbaik yang ditemukan sampai sejauh ini
3. Bila solusi ditemukan, nilai solusi tersebut dikembalikan

```
procedure search(input sol: solusi,
depth: int)
{
   Menuliskan solusi dari hasil
pencarian yang ditemukan dengan teknik
pencarian exhaustive search. Masukan
sol:solusi dan depth:int.
}

Deklarasi
solgenerated : solusi


function generatesolution() → solusi
{
  Mengembalikan kandidat solusi yang
mungkin
}

function issolution(sol : solusi) →
boolean
{
  Memeriksa apakah solusi yang
diberikan adalah solusi yang benar atau
tidak.
true jika sol adalah solusi;
false jika sol bukan solusi.
}

procedure printsolution(input sol:
solusi)
{
  Menuliskan solusi dari masukan ke
layar. Masukan sol: solusi.
}

Algoritma
if (issolution(sol))
  printsolution(sol)
else
{
  solgenerated ←
     generatesolution()
  search(solgenerated, depth+1)
}
```

2. RUANG LINGKUP MASALAH

Ruang lingkup pada makalah ini penulis batasi pada tahapan yang dibutuhkan untuk menyelesaikan permainan *scramble* dan memperoleh nilai yang setinggi-tingginya.

Terdapat dua tahapan yang dibutuhkan untuk menyelesaikan permainan ini, yaitu:

1. Menemukan kata-kata yang ada di dalam papan
   Kata-kata yang mungkin ada di dapan dapat diperoleh dengan cara merangkai huruf-huruf yang ada dipapan, huruf-huruf tersebut dirangkai sedemikian hingga seluruh kombinasi yang mungkin diperoleh. Dari rangkaian kata yang diperoleh dibandingkan dengan kata yang ada di dalam kamus, jika kata hasil rangkaian dan kata yang ada di kamus cocok, berarti kata tersebut ialah kata yang dimaksud.

2. Menentukan huruf-huruf penyusun beserta posisi huruf tersebut di atas papan
   Setelah memperoleh kata-kata yang dapat dibentuk dari huruf-huruf yang diberikan, pemain harus menginput kata tersebut ke dalam permainan sehingga ia dapat memperoleh nilai. Untuk melakukan hal tersebut pemain harus mengetahui jalur pembentukan kata yang dimaksud dengan cara mengetahui posisi huruf-huruf penyusunnya di papan.

3 MENEMUKAN KATA-KATA YANG ADA DI DALAM PAPAN

3.1 Representasi Masalah

Untuk merepresentasikan masalah ini kamus bahasa yang digunakan disimpan dalam bentuk *array of list of strig*. *Array* dipergunakan untuk mengoptimasi proses pencarian, tiap kata dengan panjang yang berbeda-beda disimpan pada *array* yang berbeda pula, sehingga *list* dalam suatu *array* selalu menyimpan kata-kata dengan panjang yang sama.

Huruf-huruf dalam papan direpresentasikan dalam bentuk kelas baru, yaitu kelas node, di dalam kelas tersebut disimpan informasi huruf berupa *string* dan informasi tetangga huruf tersebut berupa *list of node.* Informasi huruf yang disimpan oleh *node* disimpan dalam bentuk *string* dikarenakan dalam permainan *scramble* ini terdapat huruf yang selalu dipasangkan dengan huruf lain dalam satu area, contohnya Q yang selalu digabung dengan u. Kombinasi huruf-huruf yang dicari direpresentasikan dalam bentuk *string*. String yang akan diproses hanyalah string dengan panjang antara 3 hingga 10 karakter, pembatasan tersebut didasarkan atas peraturan permainan yang hanya menerima kata dengan panjang lebih dari 3 karakter dan juga usaha untuk mengoptimasi pencarian dengan membatasi panjang karakter pada kata hingga 10 karakter. Hasil pencarian direpresentasikan dalam bentuk *list of string*.

3.2 Inisiasi

Pada tahap ini dilakukan inisasi berupa menyiapkan seluruh kebutuhan program sehingga proses pencarian dapat dilakukan. Persiapan yang dilakukan diantara lain menyiapkan kamus dan menyiapkan kelas *node*.

Kamus disiapkan dengan cara membaca *file* eksternal dan kemudian memasukkannya kedalam struktur internal berupa *array of list of string*, kata-kata di dalam kamus dipisah-pisahkan sesuai dengan panjangnya dengan tujuan optimasi. Berikut adalah *pseudocode*-nya:

```
function LoadAllWords()→array of list
of string
{
   Fungsi ini akan mengembalikan array
of list of string yang merupakan hasil
konversi dari bentuk list of string,
konversi dilakukan dengan memisah-
misahkan berdasarkan panjang kata
}

Deklarasi
i: integer

AllWord: list of string

Temp: array of list of string

function getWord()→list of string
{
   Fungsi ini akan mengembalikan list
of string yang merupakan hasil
pembacaan file eksternal
}

Algoritma
AllWord←getWord()
Iterate i 0..Allword.Count
   if AllWord.ElemenAt(i).length<10
then
   Temp AllWord.ElemenAt(i).length .Add(AllWord.El
emenAt(i))
return Temp
```

Inisiasi berikutnya yaitu menyiapkan kelas node dengan cara menghubungkan tiap node dengan tetangga-tetangganya. Berikut adalah *pseudocode*-nya:

```
function LinkBoard(size:integer,
board:array of array of string)→array
of array Node
{
   Fungsi ini akan mengembalikan node-
node yang akan siap dipakai, yaitu node
dengan seluruh atributnya telah
terinisialisasi. Masukan fungsi ini
berupa size yang bertipe integer yang
merepresentasikan ukuran papan dan juga
board yang merupakan array dua dimensi
yang merepresentasikan huruf-huruf yang
dibangkitkan permainan
}

Deklarasi
i: integer
j: integer
```

```
temp: array of array of Node

procedure AddSibbling(input
i,j:integer, input/output board:array
of array of Node)
{
   Prosedur ini menginisialisasi nilai
Sibbling atau tetangga node pada posisi
i,j. Sibbling atau Tetangga maksudnya
node lain pada array yang mengelilingi
node yang dimaksud, Sibbling
direperesentasikan dalam bentuk list of
node
}

Algoritma
Iterate i 0..size
   Iterate j 0..size
       Temp[i,j].Letter←board[i,j]

Iterate i 0..size
   Iterate j 0..size
       AddSibbling(i,j,temp)

return temp
```

3.3 Pencarian

Tahap ini merupakan inti dari tahap pertama, pada tahap ini dilakukan pencarian kata-kata yang dapat dibentuk dengan huruf-huruf yang dibangkitkan oleh permainan. Pencarian ini dilakukan dengan menggunakan exhaustive search. Inti dari pencarian ini yaitu mencocokan seluruh kombinasi huruf-huruf yang terdapat di papan, yaitu huruf yang dibangkitkan oleh permainan, dengan kata yang ada di kamus. Proses pengkombinasian dilakukan dengan mengiterasi seluruh huruf yang ada di papan, dan untuk setiap huruf tersebut digabungkan dengan huruf tetangganya, hal tersebut dilakukan secara rekursif hingga panjang kombinasi mencapai 10 huruf atau seluruh tetangga telah dikombinasikan.

Dikarenakan tiap *Node* hanya dapat dipakai satu kali dalam penyusunan suatu kata, maka diperlukan penyimpanan yang berfungsi mencatat *node-node* mana saja yang telah dipakai, bentuk penyimpanan yang digunakan berupa *list*.

Berikut adalah pseudocodenya:

```
procedure findWords(input
currWord:string, currNode:Node,
usedNode:list of Node, output
foundWord:list of string)
{
   Prosedur ini berfungsi untuk mencari
seluruh kata-kata yang dapat dibentuk
berdasarkan huruf-huruf yang
dibangkitkan oleh permainan
}

Deklarasi
Length: integer
temp: list of Node
AllWord: array of list of
    string←LoadAllWords()

Algoritma
Length←currWord.length
if(Length>=3)then
   if(AllWord[Length].contains(currWord))th
    en
      foundWord.add(currWord)

if(Length>9)then
   return

usedNode.add(currNode)
iterate i 0..currNode.Sibbling.count
   if(!usedNode.contains(currNode.Sibbl
     ing.elemenAt(i)))then
         temp←usedWord //CCtor
         findWords(currWord+
          currNode.Sibbling.elemenAt(i)
          .Letter,
          currNode.Sibbling.elemenAt(i)
          , temp, foundWords)
```

4 MENENTUKAN HURUF-HURUF PENYUSUN BESERTA POSISI HURUF TERSEBUT DI ATAS PAPAN

4.1 Representasi Masalah

Pada bagian ini, terdapat 3 hal utama yang perlu direpresentasikan di dalam algoritma, yaitu kata yang akan dicari huruf penyusunnya, huruf-huruf yang dibangkitkan, dan posisi huruf-huruf penyusun kata tersebut.

Kata yang akan dicari huruf-huruf penyusun beserta posisi huruf-hurufnya direpresentasikan dalam bentuk *string*. Huruf-huruf yang dibangkitkan dalam permainan direpresentasikan dalam bentuk *array* dua dimensi dari *textbox*. Sedangkan posisi huruf-huruf penyusunnya direpresentasikan dalam bentuk *list of point.*

4.2 Inisiasi

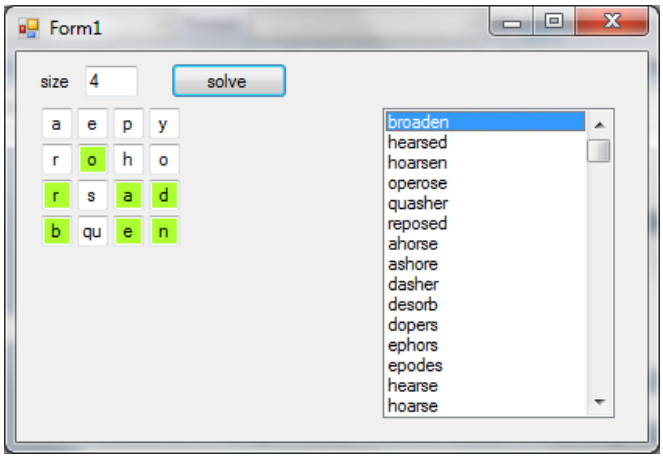
Pada tahap ini terdapat satu hal yang harus dilakukan sebelum pencarian dapat dilakukan, yaitu memastikan kata yang dicari posisi huruf-huruf penyusunnya memang kata yang dapat dibentuk dengan huruf-huruf yang dibangkitkan permainan. Hal tersebut dapat terlaksana dengan sendirinya jika kata yang dicari ialah hasil dari tahap pertama, yaitu "menemukan kata-kata yang ada di dalam papan" pada bab 3.

4.3 Pencarian

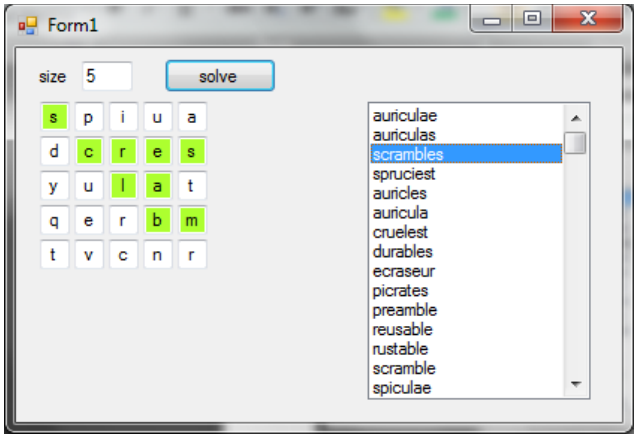
Bagian ini merupakan bagian terpenting dalam tahapan ini, disini akan dihasilkan posisi huruf-huruf penyusun suatu kata pada papan. Pencarian dilakukan dengan cara rekursif, serupa dengan pencarian pada bab 3.3. Pertama-tama dilakukan iterasi terhadap seluruh huruf pada papan, disetiap iterasi dilakukan pengkombinasian huruf yang sedang ditunjuk dengan tetangga-tetangganya secara rekursif, rekursif akan dihentikan jika panjang kombinasi huruf melewati panjang kata yang dimaksud, kombinasi huruf yang didapat bukan merupakan *substring* kata yang dimaksud, dan kombinasi huruf yang didapat telah sama dengan kata yang dimaksud. Berikut adalah *pseudocode*-nya:

```
Procedure findPath(input
targetWord:string, currWord:string,
currPoint:point, output path:list of
point)
{
   Prosedur ini akan menghasilkan list
of point yang berisi posisi huruf-huruf
penyusun kata yang dimaksud yaitu
targetWord. Setelah mengetahui posisi
huruf-hurufnya tindakan selanjutnya
yaitu meng-highlight textbox tempat
huruf yang bersangkutan berada.
}

Deklarasi
Length: integer
i: integer

sibbling: list of point
temp: list of point

function getSibbling(p:point)→list of
point
{
   Fungsi ini mengembalikan point-point
yang berada disekitar point masukan
}

function wordAt(p:point)→string
{
   Fungsi ini mengembalikan huruf pada
papan dengan posisi p
}

Algoritma
if currWord.length > targetWord.length
then
   return
else if
targetWord.Substring(0,currWord.length)
.CompareTo(currWord)!=0 then
   return
else if
targerWord.CompareTo(currWord)==0 then
   path.Add(currPoint)
   iterate i 0..path.count
      //highlight textbox pada posisi
path.elemenAt(i).x dan
path.elemenAt(i).y
   return

path.Add(currPoint)
sibbling←getSibbling(currPoint)
iterate i 0..temp.count
   if(!path.contain(sibbling.elemenAt(i
))) then
      temp←path//CCTOR
      findpath(targetWord,
currWord+wordAt(sibbling.elemenAt(i)),s
ibbling.elemenAt(i),temp)
```

5 CONTOH SOLUSI



Gambar 2 *screenshot* program dengan papan 4x4



Gambar 3 *screenshot* program dengan papan 5x5

6 KESIMPULAN

Kesimpulan yang dapat diambil yaitu:

1. *Exhaustive search* merupakan metode yang relatif sederhana, namun untuk masalah pencarian, metode ini merupakan metode yang cukup *powerful* dalam artian, prosesnya di dalam program tidak terlalu berat dan proses pembuatannya yang mudah dan dapat relatif dapat dimengerti.
2. Dengan memisahkan proses pencarian solusi menjadi dua tahap, membuat program berjalan lebih ringan dikarenakan proses pencarian posisi huruf penyusun hanya dilakukan terhadap kata yang diinginkan saja.

REFERENSI

[1] Munir, Rinaldi, "Diktat Kuliah IF3051 Strategi Algoritmik", Program Studi Teknik Informatika ITB, 2009.
[2] *http://www.codeproject.com/KB/cs/WordScramble.aspx.* Diakses pada 24 Desember 2009 pukul 13.00
[3] *http://www.algorithmist.com/index.php/Exhaustive_Search.* Diakses pada 31 Desember 2009 pukul 8.06