algoritma pencarian *backtracking* sebagai suatu prosedur rekursif sebagai berikut :

```
ExtendRight:

ExtendRight (PartialWord , node N  in dawg , square)
=
    If  N  is a terminal node then
        LegalMove (PartialWord)

    for each edge E  out of N
        i f the letter 1 labeling edge E  is
        in our rack and
        1 is in the cross-check set of
        square then
        remove a tile from the rack
        let N' be the node reached by
        following edge E
        let next-square  be the square to
        the right of square
        ExtendRight (PartialWord - 1 , N',
        next-square )
        put the tile 1  back into the
        rack
    else
        let 1 be the letter occupying square
        if N  has an edge labeled by 1 that
        leads to some node N' then
            let next-square  be the square to
            the right of square
            ExtendRight (PartialWord . 1 ,N',
            next-square )
```

Sekarang kita sudahdapat menempatkan bagian-bagian kiri kemudian mengembangkannya ke kanan, berarti kita sudah memperoleh suatu algoritma untuk membangkitkan semua langkah yang dapat dilakukan. Dapat kita lihat bahwa algoritma yang digambarkan diatas membangkitkan setiap langkah mendatar tepat hanya satu kali.

4. KESIMPULAN

Algoritma *backtracking* merupakan algoritma yang cukup mangkus untuk menyelesaikan berbagai persoalan. Hal ini disebabkan karena pada prinsipnya, kita tidak perlu memeriksa semua kemungkinan solusi yang ada. Pencarian hanya mengarah pada solusi yang dipertimbangkan saja.

Karena algoritma ini cukup efektif, maka algoritma *backtracking* banyak diterapkan dalam berbagai program *game* dan persoalan yang berkaitan dengan bidang intelegensi buatan (*artificial inteligence*). Hasil analisis kemampuan algoritma *backtracking* dalam menyelesaikan persoalan permaianan *"Scrabble"* menunjukkan bahwa algoritma ini cukup efektif untuk mendapatkan solusi permainan tersebut.

Sistem kerja algoritma*backtracking* yang sistematis dan efisien dapat menjadi solusi yang efektif dan efisien untuk persoalan ini. Sebenarnya, sebagai sesuatu yang cukup baru, penyelesaian persoalan permainan *"Scrabble"* ini belum dapat dianalisis untuk berbagai algoritma. Namun melalui studi literatur dan analisis yang telah penulis lakukan, penulis berpendapat bahwa algoritma *backtracking* cukup efektif untuk menyelesaikan persoalan ini.

REFERENSI

[1] Munir, Rinaldi. 2005. *Strategi Algoritmik*. Teknik Informatika ITB : Bandung

[2] Horowitz, Ellis. 1979. *Fundametals of Computer Algorithms*. Pitman : Maryland

[3] *Backtracking* Algorithm. http://www.cse.ohiostate.edu/~gurari/course/cis680/cis680Ch19.html

[4] S. Baase, *Computer Algorithms: Introduction to Design and Analysis, 2nd edn*, Addison-Wesley, Reading, MA, 1988, pp.209–230.

ditentukan ada yang mengandung rumus berbeda untuk menghitung jumlah poin yang dibuat pada kata tersebut.

3. Penerapan Algoritma *Backtracking*

3.1 Definisi Algoritma *Backtracking*

Algoritma *backtracking* merupakan suatu algoritma yang berbasis pada DFS untuk mencari solusi persoalan lebih mangkus dan juga lebih cepat. Dengan algoritma *backtracking*, kita tidak perlu memeriksa semua kemungkinan solusi yang ada. Hanya pencarian yang mengarah ke solusi saja yang selalu dipertimbangkan. Akibatnya, waktu pencarian dapat dihemat Algoritma *backtracking* memiliki properti sebagai berikut :

- Solusi Persoalan, vektor dengan n-tuple
- Fungsi Pembangkit nilai *xk*, membengkitkan nilai *xk* yang merupakan vektor solusi
- Fungsi pembatas, menentukan apakah (x1, x2, …, xk) mengarah ke solusi. Jika ya, maka pembangkitan nilai untuk xk+1 dilanjutkan, tetapi jika tidak, maka (x1, x2, …, xk) dibuang dan tidak dipertimbangkan lagi dalam pencarian solusi.

3.2 Pencarian Solusi Menggunakan Algoritma *Backtracking*

Setiap kata mendatar yang ingin dibentuk harus mengandung paling tidak satu huruf yang sudah ditempatkan pada papan permainan. Kita akan menamakan kotak paling kiri yang baru diisi dan bercocokan dengan sebuah huruf yang sudah terdapat pada papan permainan, kotak *anchor* dari sebuah kata.

Kita menggunakan dua strategi sederhana dalam men*generate* pergerakan. Untuk tiap posisi, kita mengenerate semua pergerakan sbb.:

(1) Temukan semua kemungkinan "bagian kiri" kata dalam posisi tertentu. (Bagian kiri dari sebuah kata adalah huruf paling kiri dari *anchor* square)

(2) Untuk setiap bagian kiri di atas, temukan bagian kanan yang cocok. (Bagian kanan adalah semua huruf di sebelah kanan bagian kiri pada *anchor* square)

Bagian kiri akan mengandung huruf dari rak kita ataupun huruf yang ada di papan, tapi tidak keduanya.

Apabila *anchor* square kosong, kita harus mengisinya dengan bagian kiri dari rak; untuk ini, setelahnya kita mengisi ke kanan dari *anchor* untuk membuat kata tersebut. Apabila permulahan *anchor* square sudah terisi, dan kita bisa dengan mudah langsung mengisinya dengan huruf dari rak. Untuk menempakatkan bagian kiri bagian kiri bisa yang telah ada di papan atau semua dari papan. Pada kasus sebelumnya kita menghitung bagian kiri hanya melihat yang ada di sana. Kasus nantinya adalah tidak trivial: kita harus menemukan semua kemungkinan bagian kiri. Karena kita mendefinisikan *anchor* square adalah bagian paling kiri dari kecocokan, bagian kiri tidak bias berkembang untuk menutupi sebuah *anchor* square. Jadinya hal tersebut akan membatasi ukuran dari bagian kiri sebuah kata dari *anchor* square. Selanjutnya, semua *anchor* square adalah cross-checks tidak trivial. Jadinya, semua kotak yang terisi oleh bagian kiri mempunyai set trivial ceoss checks yang mengizinkan semua huruf ditempatkan di situ. Kita dapat mengenerate bagian kiri yang bias di tempatkna sebelum *anchor* square yang diberikan dengan melakukan pruned traversal..

Berikut adalah prosedur *backtracking* untuk penempatan pada bagian kiri. Program akan memanggil prosedur **ExtendRight** setiap dia menemukan bagian-bagian kiri.

```
LeftPart(PartialWord, node N  in dawg, limit)
=
ExtendRight (PartialWord, N, Anchorsquare)
i f  limit  > 0  then
       for each edge E  out of N
            i f the letter 1  labeling edge E
       is
         in our rack
         remove a tile labeled 1 from the rack
         let N' be the node reached by
         following edgeE
         Leftpart (PartialWord  . 1 , N ' ,
limit
            - 1 )
          put the tile 1  back into the rack
```

Untuk mengenerate semua pergerakan dari *Anchor* Square, dengan asumsi terdapat *k* kotak non-achored di sebelah kirinya, maka akan dipanggil :

```
Leftpart("", root of dawg, k)
```

Kita dapat mencoba melengkapi kata dengan menambahkan huruf ke kanan satu per satu secara traversal dengan root N. Proses traversal ini memiliki *constraints* huruf-huruf yang masih terdapat pada *rack*, huruf-huruf yang sudah ditempatkan pada sebelah kanan *anchor*, *cross-checks* yang relevan.

Dalam mengembangkan bagian kanan, kita akan menemukan bahwa sudah terdapat huruf-huruf yang sudah ditempatkan pada papan permainan. Ini tidak menghentikan proses pencarian, sebagai gerakan yang diperbolehkan mungkin akan menyertakan huruf-huruf yang sudah diletakkan diantara huruf-huruf yang baru diletakkan., kita dapat menyertakan huruf-huruf tersebut pada kata baru yang dibentuk jika memungkinkan.

Dengan menggunakan prosedur **`LegalMove`**, yang mengambil langkah-langkah yang diperbolehkan dan menyimpannya untuk sebagai bahan pertimbangan (sebuah prosedur **`LegalMove`** yang sederhana dapat menyimpan informasi gerakan dengan skor tertinggi dan menghapus yang lainnya). Kita dapat mengekspresikan

PEMANFAATAN ALGORITMA BACKTRACKING DALAM PROGRAM PERMAINAN "SCRABBLE"

Anatariani Virniawati

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung
Jl. Ganesa no. 10, Bandung
if14008@students.if.itb.ac.id

ABSTRAK

Seiring dengan perkembangan teknologi, fungsi komputer yang pada awalnya diciptakan untuk berbagai pekerjaan komputasi juga mengalami perkembangan untuk dapat menjalankan aplikasi game. *Game* menjadi sangat beragam jenisnya dan populer karena minat dari *user* sendiri. Sekarang *game* menjadi sangat digemari karena beragamnya jenis aplikasi yang ditawarkan. User bisa memilih aplikasi *game* yang disukai berdasarkan kriteria yang diinginkan, misalnya petualangan, strategi, dan lain-lain.

Pada makalah ini, penulis mencoba membahas tentang *"Scrabble"* sebuah game *Crossword Puzzle* yang memungkinkan user untuk mencoba memasukkan kata pada papan permainan yang tersedia danmenganalisa penerapan algoritma *backtracking* sebagai salah satu cara penyelesaian persoalan pada *game*
ini.

Sebuah algoritma *backtracking* yang efisien mampu mempercepat proses pada permainan *Scrabble,*. Efisiensi ini diperoleh dari pembuatan struktur data sebelum proses pencarian *backtracking* dimulai yang berfokus pada proses pencarian dan membuat setiap langkah proses pencarian cepat.

Kata kunci: algoritma *backtracking*, game, scrabble

1. PENDAHULUAN

Permainan atau *game* sudah merupakan kegiatan yang disukai oleh masyarakat di seluruh penjuru dunia. Salah satu bentuk *game* yang sekarang sedang populer tidak adalah *game* komputer. *Game* disini merupakan aplikasi yang cukup diminati sebagai ajang *refreshing*. Bahkan tidak jarang, aplikasi *game* menimbulkan kecanduan bagi *user* sehingga mengembangkannya sebagai hobi yang menantang.

"Scrabble". *Game* ini pada dasarnya merupakan suatu aplikasi *game* yang memungkinkan user menginput kata yang diambil dari kumpulan kata atau *database* yang tersedia dan mengisikannya secara tepat pada kotak yang telah tersedia. Pada makalah ini, analisa penulis berkenaan dengan proses pengisian kata-kata ke dalam kotak yang telah disediakan oleh komputer sebagai *player*. Dalam proses ini, algoritma yang dipergunakan adalah algoritma *backtracking* (runut balik) yang menangani ketidaksinambungan yang dapat terjadi jika memasukkan kata-kata yang salah.

2. *Scrabble*

Definasi Scrabble yang kami gunakan adalah Scrabble sebagai salah satu permainan di dunia yang menggunakan huruf-huruf dalam memainkannya dan menyusunnya dari atas ke bawah (vertikal) atau kiri ke kanan. (horizontal). Sebuah permainan yang menggunakan referensi kamus untuk pengecekan keabsahan barisan huruf yang membentuk sebuah kata tersebut. Pada setiap hurufnya terdapat poin atau niai yang berbeda-beda yang pada akhirnya poin pada setiap karakter dijumlahkan untuk mendapatkan nilai total dari kata tersebut. Tantangannya adalah dalam permainan Scrabble huruf-huruf atau karakter-karakter yang tersambung menjadi kata pertama dibutuhkan untuk kata yang kedua dengan sifat yang harus berbeda. Diamana kata yang kedua harus menggunakan salah satu dari karakter kata yang pertama. Namun dalam menyambungkannya ada peraturan lain seperti player tidak boleh menyambungkan karakter pertama dari kata kedua pada karakter terakhir dari kata pertama apabila kedua kata tersbut ingin disambungkan secara seri (horizontal dengan horizontal atau vertikal dengan vertikal). Dan juga player tidak boleh menyambungkan karakter terakhir dari kata kedua pada karakter pertama dari kata pertama (seri). Singkatnya Player dapat menyambungkannya apabila sifat (seri atau paralel) dari kedua kata tersebut berbeda. Tantangan yang kedua adalah player dapat menggunakan karakter yang mana saja dalam kata perta untuk disambungkan pada karakter yang mana saja pada kata yang kedua zsehingga kemunngkinan yang muncul sangat banyak sekali. Tantangan yang ketiga adalah dimensi yang digunakan dalam menyusun karakter-karakter tersbut tidak semua digunakan hanya sebagai kotsk untuk menyimpan karakter-karakter tersebut tetapi dalam kotak trertentu pada dimensi yang telah