PENERAPAN ALGORITMA *BACKTRACKING* PADA PERMAINAN *WORD SEARCH PUZZLE*

Alvin Andhika Zulen (13507037)

Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung
Jalan Ganesha No. 10 Bandung, 40132
e-mail: alvin_az@students.itb.ac.id, if17037@students.if.itb.ac.id

ABSTRAK

Algoritma *backtracking* (Runut balik) adalah algoritma yang berbasis pada algoritma DFS (*Depth First Search*) untuk mencari solusi persoalan secara lebih mangkus. Algoritma ini merupakan perbaikan dari algoritma *brute force* yang memeriksa semua kemungkinan solusi yang ada. Dengan algoritma *backtracking*, kita tidak perlu memeriksa semua kemungkinan solusi yang ada. Hanya pencarian yang mengarah ke solusi yang akan dipertimbangkan. Simpul-simpul yang tidak mengarah ke solusi akan dipangkas. Akibatnya, waktu pencarian solusi dapat dihemat.

Saat ini, algoritma *backtracking* banyak diterapkan untuk program-program *games* (permainan). Salah satu permainan yang dapat diselesaikan dengan algoritma ini adalah permainan *Word Search Puzzle*. *Word Search Puzzle* merupakan permainan berbasis *puzzle* untuk mencari kata-kata yang disusun dalam bentuk *array* dua dimensi atau yang lebih dikenal dengan matriks. Kata-kata tersebut dapat disusun secara horizontal, vertikal maupun diagonal dan dapat ditulis dalam posisi terbalik maupun tidak. Dalam makalah ini akan dipaparkan penerapan algoritma *backtracking* untuk pencarian solusi bagi permainan ini.

Kata kunci: Word Search Puzzle, DFS, Backtracking, Simpul, Solusi, Pohon

1. PENDAHULUAN

Pada kehidupan kita sekarang ini, keberadaan *game* atau yang kita kenal dengan istilah permainan sangatlah penting. Permainan, baik tradisional maupun modern, dimainkan oleh semua kalangan sebagai pengisi waktu senggang, sekedar mencari hiburan di tengah kesibukan, atau bahkan sebagai sarana pendidikan.

Salah satu jenis permainan yang banyak diminati adalah permainan *puzzle*. Permainan *puzzle* yang berkembang sekarang ini semakin bervariasi. Salah satunya adalah permainan *Word search puzzle*. Permaianan ini merupakan permainan *puzzle* pencarian kata-kata tersembunyi yang disusun dalam bentuk *array* dua dimensi (matriks). Objektif dari permainan ini adalah menemukan semua kata yang tersembunyi di matriks permainan. Kata-kata tersebut dapat disusun secara horizontal, vertikal maupun diagonal dan dapat ditulis dalam posisi terbalik maupun tidak[1]. Permainan ini cukup menarik karena pemain diajak untuk teliti dalam mencari kata-kata diantara serangkaian huruf yang membentuk matriks. Contoh tampilan permainan ini dapat dilihat pada gambar berikut.

Gambar 1. Tampilan permainan *Word Search Puzzle*

Pada tampilan permainan *Word Search Puzzle* diatas, di bagian kanan terdapat daftar kata-kata yang harus dicari di antara huruf-huruf yang disusun di dalam matriks di bagian kiri. Jika pemain menemukan kata yang dimaksud, pemain menandai kata yang sudah ditemukan terebut.

Untuk mencari keseluruhan kata yang ada memang tidak mudah. Oleh karena itu, aplikasi permainan menyediakan sarana pencarian solusi secara otomatis. Algoritma *backtracking* akan diterapkan pada sarana pencarian solusi otomatis tersebut.

2. DASAR TEORI

Algoritma *backtracking* (Runut balik) adalah algoritma yang berbasis pada algoritma DFS (*Depth First Search*) untuk mencari solusi persoalan secara lebih mangkus. Algoritma ini merupakan perbaikan dari algoritma *brute force* yang memeriksa semua kemungkinan solusi yang ada. Dengan algoritma *backtracking*, kita tidak perlu memeriksa semua kemungkinan solusi yang ada. Hanya pencarian yang mengarah ke solusi yang akan dipertimbangkan. Simpul-simpul yang tidak mengarah ke solusi akan dipangkas.[2]

2.1 Properti Umum Algoritma *Backtracking*

Algoritma *backtracking* memiliki tiga properti utama dalam penerapannya, yaitu :

1) Solusi Persoalan

Solusi persoalan dinyatakan sebagai vektor dengan n-*tuple*.

$$\mathbf{X = (x_1, x_2, \ldots, x_n)} \quad (1)$$

2) Fungsi Pembangkit

Fungsi pembangkit digunakan untuk membangkitkan nilai x_i yang merupakan komponen vektor solusi. Fungsi pembangkit dinyatakan sebagai :

$$\mathbf{T(k)} \quad (2)$$

3) Fungsi Pembatas

Fungsi pembatas digunakan untuk menentukan apakah $(x_1, x_2, \ldots, x_k)$ mengarah ke solusi atau tidak. Jika ya, pembangkitan nilai dilanjutkan. Jika tidak, $(x_1, x_2, \ldots, x_k)$ dibuang dan tidak dipertimbangkan lagi dalam pencarian solusi. Fungsi pembangkit dinyatakan sebagai :

$$\mathbf{B(x_1, x_2, \ldots, x_k)} \quad (3)$$

2.2 Pengorganisasian Solusi

Semua kemungkinan solusi dari persoalan disebut ruang solusi (*solution space*). Ruang solusi diorganisasikan ke dalam struktur pohon. Tiap simpul pohon menyatakan status (*state*) persoalan, sedangkan sisi (cabang) dilabeli dengan nilai-nilai x_i. Lintasan dari akar ke daun menyatakan solusi yang mungkin. Seluruh lintasan dari akar ke daun membentuk ruang solusi. Pengorganisasian pohon ruang solusi diacu sebagai **pohon ruang status** (*state space tree*).

2.3 Prinsip Pencarian Solusi

Langkah-langkah pencarian solusi dengan algoritma *backtracking* adalah sebagai berikut[3] :

1) Solusi dicari dengan membentuk lintasan dari akar ke daun dengan mengikuti algoritma DFS. Simpul-simpul yang sudah dibangkitkan dinamakan **simpul hidup** (*live node*). Simpul hidup yang *sedang* diperluas dinamakan **simpul-E** (*Expand-node*). Simpul dinomori sesuai dengan urutan pembangkitannya.

2) Tiap kali simpul-E diperluas, lintasan yang dibangun bertambah panjang. Jika lintasan yang sedang dibentuk tidak mengarah ke solusi, maka simpul-E tersebut "dibunuh" oleh fungsi pembatas sehingga menjadi **simpul mati** (*dead node*).
Simpul yang sudah mati tidak akan pernah diperluas lagi.

3) Jika pembentukan lintasan berakhir dengan simpul mati, proses pencarian diteruskan dengan membangkitkan simpul anak yang lainnya. Jika tidak ada lagi simpul anak yang dapat dibangkitkan, pencarian solusi dilanjutkan dengan melakukan *backtracking* ke simpul hidup terdekat (simpul orangtua). Selanjutnya simpul ini menjadi simpul-E yang baru.

4) Pencarian dihentikan jika telah menemukan solusi atau tidak ada lagi simpul hidup untuk runut-balik.

2.4 Skema Umum Algoritma *Backtracking*

Algoritma *backtracking* bisa dikembangkan secara iteratif dan rekursif. Namun, untuk kasus ini penulis menggunakan algoritma yang bersifat iteratif. Skema umum algoritma *backtracking* versi iteratif adalah sebagai berikut :

```
procedure Backtracking(input n:integer)

Delarasi:
k : integer

Algoritma:
k←1

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

    k←k+1 {indeks anggota tupple berikutnya}

  else {x[1], x[2], …, x[k] tidak mengarah ke
        simpul solusi}
    k←k-1 {runut-balik ke anggota tupple
           sebelumnya}
  endif
endwhile
{k = 0}
```

3. ANALISIS PENYELESAIAN MASALAH

Untuk melakukan analisis terhadap penerapan algoritma *backtracking* dalam permainan *Word search puzzle*, penulis membuat sebuah program pencarian solusi otomatis pada permainan ini. Tampilan dari program ini dapat dilihat pada gambar berikut.

Gambar 2. Tampilan program pencarian solusi
Word Search Puzzle

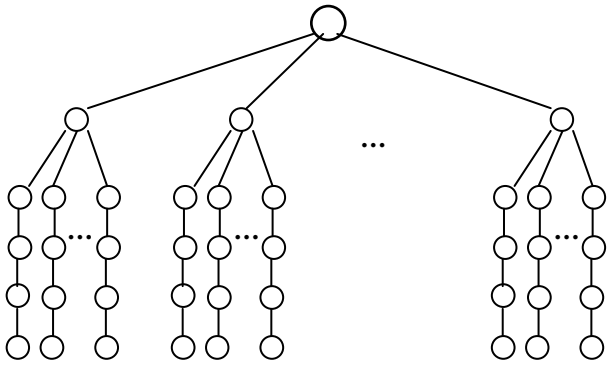
Untuk mempersempit ruang masalah, penulis mempergunakan matriks 5x5, dimana terdapat 25 huruf dalam matriks tersebut. Untuk mencari posisi suatu kata dalam papan permainan, pengguna mengetikkan kata yang ingin dicari di *textbox* yang tersedia dan menekan tombol **SOLVE**. Pencarian dilakukan hingga menemukan kata yang diinginkan (hanya mencari solusi pertama).

Dalam melakukan pencarian satu kata, pemain harus mencari melalui delapan jalur yang mungkin, yaitu :
1. horizontal ke kanan
2. horizontal ke kiri
3. vertikal ke atas
4. vertikal ke bawah
5. diagonal ke kiri atas
6. diagonal ke kiri bawah
7. diagonal ke kanan atas
8. diagonal ke kanan bawah

Jadi untuk pencarian huruf per huruf dari kata yang dicari, kita harus melihatnya dari delapan jalur tersebut.

Solusi persoalan dari permainan *word search puzzle* adalah vektor n-*tuple* dimana tiap komponennya adalah huruf pada posisi tertentu pada papan permainan. Untuk program yang dirancang, nilai n dibatasi dari 1 - 5 huruf. Ruang solusi dari persoalan ini adalah semua kemungkinan kata yang dapat diperoleh di papan permainan dengan memeriksa ke delapan jalur yang telah disebutkan diatas. Dengan kata lain, untuk matriks berukuran NxN, terdapat maksimal $8N^2$ kemungkinan solusi persoalan. Disebut maksimal karena tidak semua posisi pada papan permainan memiliki tepat delapan jalur pencarian, seperti huruf-huruf pada pinggir papan permainan.

Pohon ruang status statis pada program pencarian solusi ini dapat dilihat pada gambar berikut.



Gambar 3. Pohon ruang status statis pencarian solusi
Word Search Puzzle

Tiap simpul pohon (*state*) menyatakan kumpulan huruf yang telah ditemukan yang berada pada posisi-posisi tertentu. Sisi (cabang) menyatakan huruf selanjutnya pada posisi tertentu yang sedang diperiksa. Lintasan dari akar ke daun menyatakan solusi yang mungkin. Solusi dicari dengan membentuk lintasan dari akar ke daun dengan mengikuti algoritma DFS. Pada program pencarian solusi yang dibuat, pencarian dimulai dari huruf pada bagian kiri atas papan permainan dan simpul selanjutnya dibangkitkan sesuai algoritma DFS untuk kemudian diperiksa kecocokannya.

Langkah-langkah pencarian solusi pada program ini adalah sebagai berikut :

1) Simpul akar pada pohon ruang status merupakan inisialisasi dan menyatakan pencarian huruf awal dari kata yang ingin dicari di papan permainan. Pencarian posisi-posisi ini dilakukan dari bagian kiri atas hingga bagian kanan bawah papan permainan. Pencarian ini akan menghasilkan posisi-posisi pada papan permainan yang berisi huruf yang sama dengan huruf awal pada kata yang ingin dicari.

2) Jika pencarian pada langkah (1) tidak menemukan satupun posisi pada papan permainan yang berisi huruf awal dari kata yang dicari, berarti kata tersebut tidak ada di papan permainan.

3) Jika pencarian pada langkah (1) menemukan posisi yang tepat, pencarian huruf selanjutnya akan dimulai dari posisi tersebut.

4) Pembangkitan simpul-simpul dari posisi tersebut dilakukan dengan mengikuti algoritma DFS. Simpul dibangkitkan dengan urutan jalur dari atas → kanan-atas

→ kanan → kanan-bawah → bawah → kiri-bawah → kiri → kiri-atas.

5) Pembangkitan simpul selanjutnya dilakukan dengan berdasarkan pada fungsi pembatas. Terdapat tiga fungsi pembatas pada program ini, yaitu :

a) Jika pembangkitan suatu simpul telah memilih suatu jalur, pembangkitan simpul selanjutnya untuk lintasan tersebut dilakukan menurut jalur tersebut seperti dapat dilihat pada pohon ruang status.
Sebagai contoh, jika telah memilih jalur atas, maka pembangkitan simpul selanjutnya untuk lintasan tersebut hanya untuk jalur atas saja.

b) Jumlah huruf yang terdapat dalam kata yang ingin dicari. Dengan kata lain, jika ingin memeriksa suatu jalur, terlebih dahulu dilakukan pengecekan apakah pada jalur tersebut dapat diperoleh jumlah huruf yang sama dengan jumlah huruf pada kata yang ingin dicari.
Sebagai contoh, pada program yang dibuat, jika sedang berada di posisi kiri-atas pada papan permainan, jalur yang bisa dibangkitkan pada posisi ini hanya jalur kanan, kanan-bawah, dan bawah saja.

c) Kecocokan huruf pada suatu posisi di papan permainan dengan huruf pada kata yang ingin dicari.

6) Pencarian berhasil jika menemukan kata yang dicari pada pembangkitan simpul-simpul pada proses pencarian melalui suatu jalur (fungsi pembatas selalu mengembalikan nilai *true* pada pembangkitan simpul),

7) Jika lintasan yang sedang dibentuk tidak mengarah ke solusi (fungsi pembatas mengembalikan nilai *false*), simpul tersebut "dibunuh" sehingga menjadi simpul mati (*dead node*) dan tidak akan diperluas lagi.

8) Jika pembentukan lintasan berakhir dengan simpul mati, proses pencarian solusi dilanjutkan dengan melakukan *backtracking* ke simpul orangtua. Dalam hal ini, *backtracking* dilakukan hingga kembali ke simpul pada langkah (4) atau dengan kata lain simpul orangtua level 1 pada pohon ruang status (asumsi : simpul akar memiliki level 0). Selanjutnya simpul ini menjadi simpul-E yang baru dan simpul anak berikutnya dibangkitkan sesuai urutan priotitas jalur yang telah disebutkan sebelumnya.

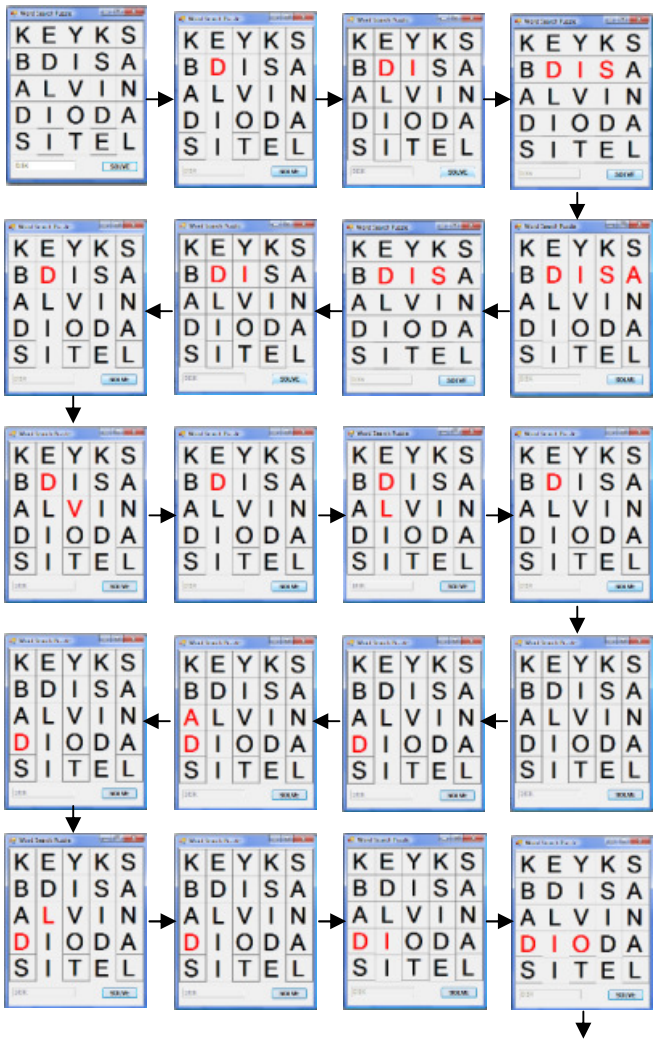
9) Jika simpul level 1 yang diperoleh dari langkah (8) tidak bisa diekspansi lagi (semua jalur telah diperiksa dan tidak ada yang memenuhi), pencarian solusi dilakukan dengan *backtracking* ke simpul akar. Dengan kata lain, proses dimulai kembali dari langkah (1) dengan mencari posisi lain dari huruf awal kata yang ingin dicari di papan permainan.

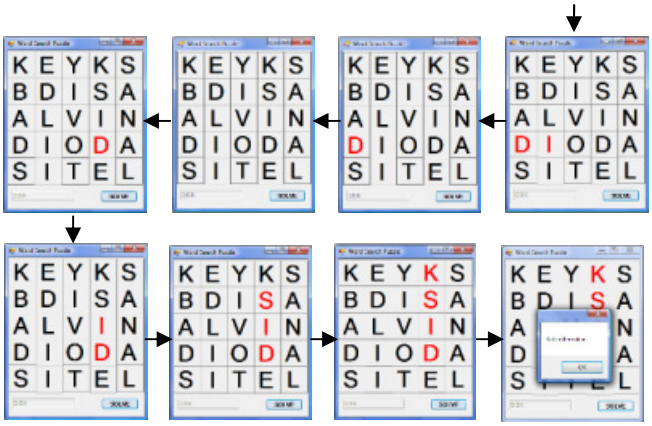
Algoritma ini merupakan perbaikan dari algoritma *brute force*. Pada algoritma *brute force,* semua kemungkinan solusi yang ada diperiksa atau dengan kata lain memeriksa semua lintasan yang mungkin tanpa menggunakan fungsi pembatas. Dengan algoritma *backtracking*, semua kemungkinan solusi yang ada tidak perlu diperiksa. Hanya pencarian yang mengarah ke solusi yang akan dipertimbangkan (memenuhi fungsi pembatas). Simpul-simpul yang tidak mengarah ke solusi akan dibunuh. Akibatnya, waktu pencarian solusi menjadi lebih sedikit.

Untuk menguji penyelesaian masalah ini, berikut akan ditampilkan *screenshot-screenshot* dari program pencarian solusi yang dibuat. Program diuji terhadap dua kasus, yaitu kasus dimana kata yang ingin dicari ada dan tidak ada di papan permainan.

1) Kasus 1 : Pencarian kata yang ada di papan permainan
Kata yang akan dicari : **DISK**

Screenshot program :

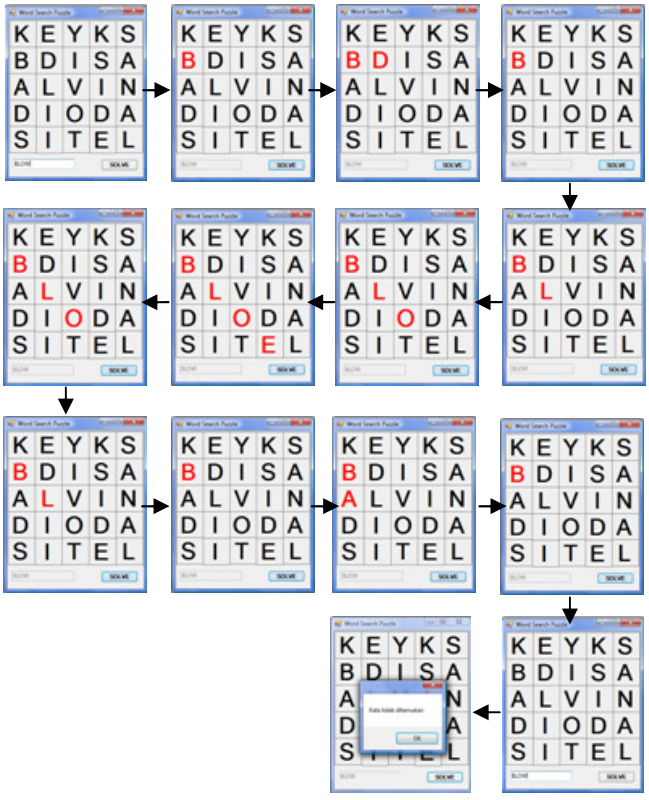




Gambar 4. *Screenshot* program pencarian solusi *Word Search Puzzle* untuk kasus 1

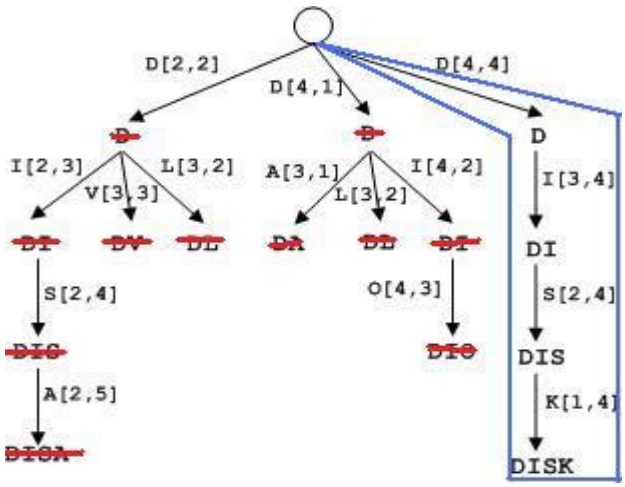
2) Kasus 2 : Pencarian kata yang tidak ada di papan permainan
Kata yang akan dicari : **BLOW**

Scrnshot program :



Gambar 5. *Screenshot* program pencarian solusi *Word Search Puzzle* untuk kasus 2

Untuk penggambaran pohon ruang status dinamis, simpul-simpul dikunjungi dengan mengikuti algoritma DFS. Nilai pada simpul menyatakan kumpulan huruf yang telah ditemukan yang berada pada posisi-posisi tertentu. Nilai pada sisi menyatakan huruf selanjutnya yang akan diperiksa pada posisi tertentu (asumsi : posisi pada papan permainan dimulai dari [1,1]). Simpul yang ditandai dengan garis merah merupakan simpul yang dibunuh oleh fungsi pembatas. Lintasan solusi dapat dilihat pada lintasan yang diblok dengan garis biru. Jika digambarkan, pohon ruang status dinamis untuk kasus 1 dapat dilihat pada gambar berikut.



Gambar 6. Pohon ruang status dinamis untuk kasus 1

Dari kedua kasus diatas, terlihat bahwa algoritma *backtracking* yang diterapkan pada program pencarian solusi permainan *Word search puzzle* jauh lebih baik dibandingkan algoritma *brute force* yang memeriksa semua kemungkinan solusi. Algoritma ini hanya membangkitkan dan mempertimbangkan simpul-simpul yang mengarah ke solusi persoalan dan membunuh simpul-simpul yang tidak mengarah ke solusi sehingga akan memangkas waktu pencarian solusi.

4. KESIMPULAN

Permainan *word search puzzle* ini merupakan suatu permainan tipe *puzzle* yang sangat sederhana. Namun ternyata permainan ini cukup menarik karena pemain diajak untuk teliti dalam mencari kata-kata diantara serangkaian huruf yang membentuk matriks.

.

Dalam makalah ini, dirancang sebuah program untuk pencarian solusi otomatis pada permainan ini dengan menggunakan algoritma *backtracking*. Algoritma yang berbasis algoritma DFS ini jauh lebih baik dari algoritma *brute force* yang memeriksa semua kemungkinan solusi yang ada. Algoritma ini hanya membangkitkan dan mempertimbangkan simpul-simpul yang mengarah ke solusi persoalan dan membunuh simpul-simpul yang tidak mengarah ke solusi sehingga akan memangkas waktu pencarian solusi.

REFERENSI

[1] http://en.wikipedia.org/wiki/Word_search
Tanggal akses : 19 Desember 2009 pukul 12:00

[2] http://en.wikipedia.org/wiki/Backtracking
Tanggal akses : 20 Desember 2009 pukul 13:10

[3] Munir, Rinaldi. "Diktat Kuliah IF3051 Strategi Algoritma".
Institut Teknologi Bandung. 2009