Penerapan Algoritma BFS dan DFS dalam permainan Wordament

Yanuar Aristya Edy Putra / 13511039
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
yanuararistya@gmail.com

Makalah ini menjabarkan dan menjelaskan bagaimana penggunaan algoritma Breadth-First Search (BFS) dan Depth-First-Search (DFS) dalam permainan Wordament. Wordament adalah suatu permainan dimana pemain menyambungkan huruf sehingga menjadi suatu kata dalam bahasa inggris, setiap kata akan memberikan nilai. Tujuan dari permainan wordament adalah mendapatkan nilai sebanyak-banyaknya. Pada makalah ini pertama kali akan dijelaskan lebih lanjut mengenai wordament, selanjutnya akan dijelaskan mengenai dasar teori BFS dan DFS. Dasar teori yang telah dijelaskan akan menjadi dasar implementasi dalam penggunaan BFS dan DFS untuk mencari kata-kata yang ada pada suatu stage dalam permainan wordament.

BFS, DFS, Wordament.

I. PENDAHULUAN

Wordament merupakan suatu permainan dimana pemain mencari kata. Cara untuk mendapatkan suatu kata adalah dengan cara menyambungkan huruf-huruf yang ada, huruf yang bisa disambung hanya huruf yang bersebelahan. Bersebelahan maksutnya adalah huruf yang ada di sebelah kiri, kanan, atas, bawah, ataupun secara diagonal. Setiap kata yang didapatkan akan memberikan nilai, tujuan dari permainan ini adalah mengumpulkan nilai sebanyak-banyaknya dalam jangka waktu dua menit.

Gambar 1. Screenshot permainan Wordament

Suatu stage wordament terdiri dari 16 kotak, setiap kotak biasanya terdiri dari satu atau dua huruf. Pada pojok kanan atas terdapat sisa waktu yang tersisa pada stage tersebut. Pada bagian kiri bawah terlihat kata apa saja yang sudah kita dapatkan.

Wordament merupakan permainan dengan platform mobile. Input dilakukan secara touchscreen.

Gambar 2.Proses mendapatkan kata (kiri ke kanan)

Dimulai dengan menekan kotak “M”, selanjutnya tahan dan *drag* sampai ke kotak “I”, lalu *drag* lagi ke “D”. Setelah sampai ke “D”, lepaskan jari dari screen untuk mendapatkan kata “mid”. Kotak yang sudah dilewati tidak dapat dilewati lagi. Misalnya setelah “mid” pemain *drag* ke kotak “M” tidak akan menjadi “midm”. Ini merupakan cara bermain wordament.

Gambar 3. Tampilan setelah selesai stage

Setelah selesai suatu stage, pemain dapat melihat bagaimana performa mereka pada stage tersebut, seperti total nilai yang didapat, total kata yang ditemukan, dan lain lain. Pemain dapat pula melihat berbagai kata yang terdapat pada stage tersebut.

Setelah menampilkan performa, pemain juga dapat melihat peringkat berapa mereka diantara pemain lainnya di seluruh dunia. Pemain diurutkan berdasarkan total nilai yang mereka dapatkan.

Setelah mempelajari strategi algoritma, penulis tertarik untuk membuat suatu *solver* untuk menemukan kata yang ada pada suatu stage dengan menggunakan suatu strategi algoritma. Strategi yang akan dibahas pada makalah ini adalah BFS, dan DFS.

II. DASAR TEORI

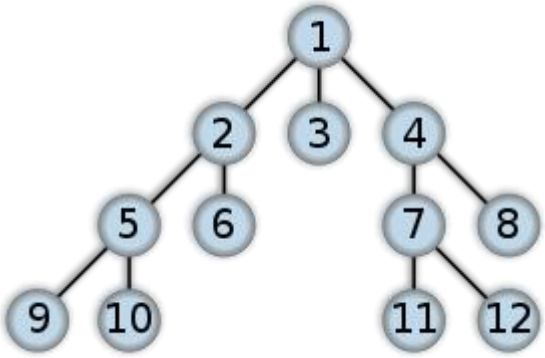
A. *BREADTH-FIRST SEARCH*

Breadth-First Search (BFS) merupakan suatu strategi pencarian graf. Penelusuran BFS dimulai dari akar, kemudian mengecek setiap simpul tetangga yang belum dikunjungi, sampai semua simpul sudah dikunjungi atau objektif pencarian tercapai. BFS menggunakan struktur data *queue* untuk menyimpan simpul-simpul yang akan dikunjungi.

Pencarian solusi dengan BFS pada pohon ruang status dapat digambarkan dengan algoritma berikut ini.

1. Masukkan simpul akar ke antrian *Q*. Jika akar merupakan solusi, stop.
2. Jika *Q* kosong, stop.
3. Ambil simpul *s* dari kepala (*head)* antrian, bangkitkan semua anaknya dan masukkan ke **belakang** antrian *Q*.
4. Jika *s* merupakan simpul solusi, stop, jika tidak, kembali ke langkah 2

Berikut ilustrasi urutan pencarian BFS.



Gambar 4. Urutan pencarian BFS

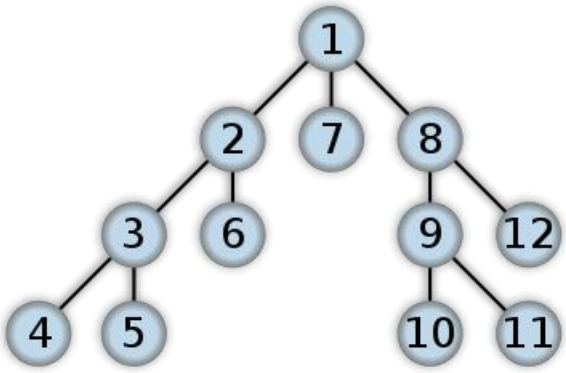
B. *DEPTH-FIRST SEARCH*

Depth-First Search (DFS) juga merupakan strategi pencarian graf. Perbedaannya dengan BFS adalah, DFS melakukan pencarian dari akar, lalu melanjutkan ke anaknya terlebih dahulu. Saat sudah mencapai simpul daun, dan masih belum ditemukan solusi, DFS akan melakukan backtrack ke simpul sebelumnya, dan akan melanjutkan ke simpul anak lainnya yang belum dikunjungi. DFS menggunakan struktur data *stack*.

Pencarian solusi dengan DFS pada pohon ruang status dapat digambarkan dengan algoritma berikut ini.

1. Masukkan simpul akar ke antrian *S*, jika akar merupakan solusi, stop.
2. Jika *S* kosong, stop.
3. Ambil simpul *v* dari kepala antrian *S*, bangkitkan semua anaknya, masukkan ke **depan** antrian *S*.
4. Jika v solusi, stop, jika tidak, kembali ke langkah 2.

Berikut ilustrasi urutan pencarian DFS.



Gambar 5. Urutan pencarian DFS

III. IMPLEMENTASI

Pada bagian ini akan dibahas bagaimana implementasi strategi BFS dan DFS pada permainan wordament untuk mencari kata yang ada pada suatu stage.

Untuk melakukan implementasi algoritma BFS dan DFS akan digunakan bahasa C++.

Pertama, kita harus mendefinisikan bagaimana bentuk dari suatu *state* (simpul). Pada kasus kali ini, suatu state didefinisikan dengan suatu *string* yang menyimpan suatu kata, suatu matriks *boolean* yang merepresentasikan kotak mana saja yang sudah dilewati. Matriks *boolean* digunakan untuk melakukan validasi apakah suatu kotak sudah dilewati atau belum. *True* menandakan sudah dikunjungi, sedangkan *false* menandakan bahwa kotak tersebut belum dikunjungi. *State* juga menyimpan informasi dimana posisi kotak terakhir. Posisi tersebut didefinisikan dengan dua nilai integer yang merepresentasikan baris dan kolom keberapa.

Suatu stage di wordament direpresentasikan dengan matriks *string. String* digunakan karena ada kasus dimana suatu kotak berisi lebih dari satu huruf.

Simpul yang dibangkitkan merupakan hasil perjalanan dari suatu *state*. Suatu state dapat berjalan ke delapan arah, yaitu kanan, kanan bawah, bawah, kiri bawah, kiri, kiri atas, atas, kanan atas, dan atas. Urutan jalan pencarian juga dalam urutan yang sama seperti yang dituliskan diatas.

Suatu perjalanan akan merubah *state*. Kata yang disimpan di *state* tersebut akan ditambahkan dengan huruf

di kotak yang akan dikunjungi. Matriks *boolean* juga akan dirubah, nilai *boolean* di posisi kotak yang akan dikunjungi akan dijadikan *true*. Posisi kotak terakhir juga akan dirubah menjadi posisi kotak yang akan dikunjungi.

Suatu kata valid jika terdapat dalam kamus yang didefinisikan. Kamus berbentuk suatu *map*<*string, boolean*>. Map digunakan karena mudah untuk melakukan pengecekan apakah suatu elemen berada di dalam map atau tidak, hal ini disebabkan map akan mengembalikan value *false* sebagai default jika *key* yang dicari tidak ditemukan di dalam map. Setiap entri akan memiliki nilai *true* sebagai nilai boolean nya.

A. *IMPLEMENTASI BFS*

Pseudo-code dari algoritma BFS yang digunakan:

```
procedure BFS(input/output array of string V)
state currentState
state newState
queue of state states
initiateQueue(states)

while states not empty do
 currentState=states.pop()
 foreach moveDirection do
  if move is valid do
   newState=state.move(moveDirection)
   states.push(newState)
  endif
  if state is solution do
    V.push(state.word)
  endif
 endfor
```

Proses pencarian kata dengan BFS dalam suatu stage di wordament dijabarkan lebih lanjut dalam algoritma berikut ini.

1. Inisialisasi antrian *Q* dengan memasukkan semua *state* awal. Setiap *state* awal merupakan setiap satu kotak yang unik.
2. Jika *Q* kosong, stop.
3. Ambil simpul *s* dari kepala antrian, bangkitkan semua anak yang valid. Simpul anak yang valid merupakan kotak yang dapat dituju dari kotak sebelumnya dan kotak yang dituju tersebut masih belum dilewati sebelumnya.
4. Untuk setiap anak yang dibangkitkan, update matriks *boolean* dan *string* dari setiap anak sebagaimana *state* dari simpul sebelumnya dan kearah mana *state* sebelumnya berjalan.
5. Masukkan semua anak yang dibangkitkan ke **belakang** antrian *Q*.
6. Jika *s* merupakan solusi, masukkan *s* ke himpunan solusi *V*. Suatu simpul merupakan solusi jika panjang dari kata di *s* lebih dari tiga huruf, dan *s* terdapat dalam kamus.
7. Kembali ke tahap 2.

B. *IMPLEMENTASI DFS*

Pseudo-code dari algoritma DFS yang digunakan:

```
procedure DFS(input/output stack of state states, input/output array of string V)
state currentState
state newState

foreach moveDirection do
 currentState=states.pop()
  if move is valid do
   newState=state.move(moveDirection)
   states.push(newState)

  if state is solution do
    V.push(state.word)
  endif

   DFS(states)
  endif
endfor
```

Simpul akar didefinisikan sebagai *initial state*, dimana *initial state* merupakan simpul yang berisikan satu atau dua huruf di satu kotak saja. Karena itu, kita butuh untuk memanggil fungsi pencarian DFS sebanyak jumlah *initial state* yang ada, yaitu sebanyak 16 kali (karena ada 4x4 kotak).

```
procedure executeDFS()
stack of state S
array of string V
foreach initial state do
 S.push(initial state)
 DFS(S,V)
endfor
```

Pencarian kata dengan DFS menggunakan fungsi rekursif. Tidak banyak perbedaannya dibandingkan BFS, berikut algoritmanya.

1. Inisialisasi antrian *Q* dengan memasukkan semua *state* awal. *Q* merupakan struktur data *stack*.
2. Jika *Q* kosong, stop.
3. Ambil simpul *s* dari kepala antrian, bangkitkan semua anak yang valid..
4. *Update state* dari setiap anak yang dibangkitkan.
5. Masukkan semua anak yang dibangkitkan ke **depan** antrian *Q*.
6. Jika *s* merupakan solusi, masukkan *s* ke himpunan solusi *V*.
7. Kembali ke tahap 2.

Setelah melaksanakan algoritma, akan ditampilkan ke layar semua kata yang ditemukan. Waktu eksekusi juga dihitung, lalu ditampilkan.

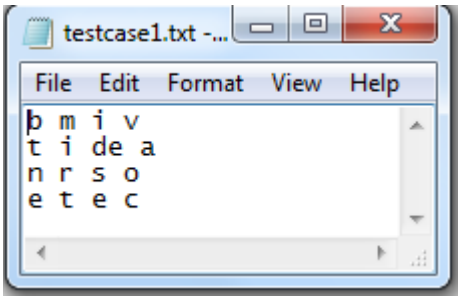
IV. HASIL EKSPERIMEN

Penulis sudah membuat program untuk mencari solusi dari suatu stage dalam permainan wordament dengan menggunakan strategi BFS dan DFS. Program menerima input 16 kotak, dengan satu kotak merepresentasikan satu atau dua huruf, dan memberikan output keluaran semua kata yang dapat ditemukan dari stage tersebut. Program akan mencari kata yang ada dari suatu kamus yang berisi sekitar 50 ribu kata yang penulis temukan dari suatu sumber di internet.

Hasil dari wordament kemungkinan besar masih banyak yang tidak ditemukan. Hal ini karena kamus yang digunakan oleh permainan wordament berbeda dari kamus yang digunakan program.

1. Testcase 1

Input:



Gambar 6. Input testcase 1

Output BFS:

0.ode 1.bide 2.bin 3.bit 4.idea
5.ides 6.idem 7.via 8.vim
9.tide 10.tin 11.ire 12.aide
13.aim 14.net 15.nit 16.nib
17.ride 18.rib 19.rim 20.sec
21.set 22.side 23.sir 24.sin
25.sit 26.odes 27.ere 28.ten
29.est 30.code 31.bides 32.mist
33.mire 34.mint 35.mine 36.ideas
37.vase 38.vast 39.video 40.tides
41.tire 42.tint 43.trim 44.deist
45.aside 46.aides 47.aider 48.nets
49.rest 50.rent 51.rides 52.rise
53.sider 54.sire 55.sine 56.oast
57.erst 58.tern 59.tent 60.cost
61.codes 62.coder 63.miser 64.mires
65.mints 66.miner 67.mitre 68.videos
69.tires 70.tints 71.inter 72.inert
73.desert 74.desire 75.demise 76.demist
77.aster 78.aiders 79.recode 80.reside
81.resin 82.resit 83.rents 84.stern
85.stride 86.siren 87.enter 88.terse
89.ester 90.eosin 91.coders 92.coast
93.mister 94.miners 95.mitres 96.vaster
97.astern 98.astride 99.recodes 100.stereo
101.sinter 102.enters 103.entire 104.cosine
105.codeine 106.entires 107.coaster
time: 61.929 seconds

Tabel 1. Hasil Testcase 1

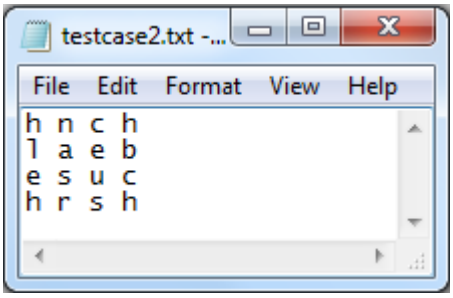
Output DFS:

0.bide 1.bides 2.bin 3.bit 4.miser
5.mist 6.mister 7.mire 8.mires
9.mint 10.mints 11.mine 12.miner
13.miners 14.mitre 15.mitres 16.idea
17.ideas 18.ides 19.idem 20.vase
21.vast 22.vaster 23.via 24.video
25.videos 26.vim 27.tide 28.tides
29.tire 30.tires 31.tin 32.tint
33.tints 34.trim 35.ire 36.inter
37.inert 38.desert 39.desire 40.deist
41.demise 42.demist 43.aster 44.astern
45.astride 46.aside 47.aide 48.aides
49.aider 50.aiders 51.aim 52.net
53.nets 54.nit 55.nib 56.recode
57.recodes 58.rest 59.reside 60.resin
61.resit 62.rent 63.rents 64.ride
65.rides 66.rise 67.rib 68.rim
69.sec 70.set 71.stern 72.stereo
73.stride 74.side 75.sider 76.sir
77.sire 78.siren 79.sin 80.sinter
81.sine 82.sit 83.ode 84.odes
85.oast 86.enter 87.enters 88.entire
89.entires 90.erst 91.ere 92.tern
93.ten 94.tent 95.terse 96.est
97.ester 98.eosin 99.cost 100.cosine
101.code 102.codes 103.coder 104.coders
105.codeine 106.coast 107.coaster
time: 65.86 seconds

Tabel 2. Hasil DFS testcase 1

2. Testcase 2

Input:



Gambar 7. Input testcase 2

Output BFS:

0.has 1.can 2.hen 3.las 4.lea
5.ass 6.ash 7.ale 8.ace
9.bus 10.bur 11.ben 12.els
13.sue 14.sub 15.she 16.sea
17.san 18.sac 19.sec 20.sen
21.use 22.cur 23.cue 24.cub
25.her 26.rue 27.rub 28.hue
29.hub 30.hash 31.hale 32.cess
33.cash 34.case 35.cane 36.heal
37.lass 38.lash 39.lase 40.lane
41.lace 42.less 43.lean 44.ales
45.ache 46.aces 47.acne 48.esau

49.ease 50.each 51.bush 52.burs
53.beau 54.bean 55.elan 56.else
57.such 58.sure 59.seal 60.sale
61.sane 62.urea 63.user 64.cuss
65.curs 66.cure 67.cues 68.cube
69.hers 70.rhea 71.real 72.rush
73.ruse 74.rues 75.sues 76.hush
77.hues 78.chess 79.cease 80.cause
81.canes 82.heals 83.laser 84.lance
85.lanes 86.laces 87.leash 88.lease
89.leach 90.aches 91.easel 92.beaus
93.beach 94.bench 95.shrub 96.sauce
97.usher 98.users 99.cures 100.curse
101.cubes 102.resea 103.reals 104.reach
105.healer 106.lasers 107.lances 108.laches
109.assure 110.encase 111.bushel 112.seance
113.secure 114.sealer 115.ushers 116.rhesus
117.healers 118.lashers 119.leaches 120.because
121.sealers 122.reaches 123.realness
sstime: 62.293 seconds

Tabel 3. Hasil BFS testcase 2

Output DFS:

0.has 1.hash 2.hale 3.chess 4.cess
5.cease 6.cause 7.cash 8.case
9.can 10.cane 11.canes 12.heal
13.heals 14.healer 15.healers 16.hen
17.las 18.lass 19.lash 20.lashers
21.lase 22.laser 23.lasers 24.lance
25.lances 26.lane 27.lanes 28.laches
29.lace 30.laces 31.less 32.lea
33.leash 34.lease 35.lean 36.leach
37.leaches 38.ass 39.assure 40.ash
41.ale 42.ales 43.ache 44.aches
45.ace 46.aces 47.acne 48.esau
49.ease 50.easel 51.each 52.encase
53.bus 54.bush 55.bur 56.burs
57.bushel 58.beau 59.beaus 60.bean
61.ben 62.because 63.elan 64.els
65.else 66.such 67.sure 68.sue
69.sub 70.shrub 71.she 72.sea
73.seal 74.seance 75.sauce 76.sale
77.san 78.sane 79.sac 80.sec
81.secure 82.sealer 83.sealers 84.sen
85.urea 86.usher 87.ushers 88.use
89.user 90.users 91.cuss 92.cur
93.curs 94.cure 95.cures 96.curse
97.cue 98.cues 99.cub 100.cube
101.cubes 102.her 103.hers 104.rhesus
105.rhea 106.resea 107.real 108.reals
109.realness 110.reach 111.reaches 112.rush
113.ruse 114.rue 115.rues 116.rub
117.sues 118.hush 119.hue 120.hues 121.hub
time: 63.953 seconds

Tabel 4. Hasil DFS testcase 2

V. KESIMPULAN

BFS dan DFS dapat digunakan untuk mencari kombinasi kata yang ada pada suatu stage di wordament. Dari waktu eksekusi, perbedaan dari DFS dan BFS tidak terlalu signifikan. Tetapi DFS lebih cocok digunakan pada kasus dimana terdapat suatu event untuk mencari *long word*. Sedangkan, BFS akan lebih cepat memunculkan hasil dari kata yang pendek.

Dari hasil eksperimen, terdapat perbedaan kata yang ditemukan dari BFS dan DFS. BFS menemukan 123 kata, sedangkan DFS menemukan 121 kata. Hal ini mungkin disebabkan suatu *bug* yang masih belum dapat ditemukan solusinya karena keterbatasan waktu.

Dari hasil eksperimen juga dapat dilihat bahwa BFS relatif lebih cepat dalam melakukan pencarian dibandingkan DFS.

REFERENCES

[1] Munir, Rinaldi, Diktat Kuliah IF3051Strategi Algoritma, Penerbit Informatika : Bandung, 2009
[2] Game Wordament
[3] http://www.manythings.org/vocabulary/lists/l/
[4] http://wordament.com/how-to-play-wordament/
[5] http://www.cse.ohio-state.edu/~gurari/course/cis680/cis680Ch14.html

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, 19 Desember 2013

Yanuar Aristya Edy Putra - 13511039