Penerapan Algoritma Depth-First Search pada Penyelesaian Teka Teki Silang

Muh. Habibi Haidir / 13516085
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
muhhabibih@gmail.com, *13516085@std.stei.itb.ac.id*

***Abstract*—Teka teki silang atau crossword puzzle merupakan permainan teka-teki dimana disediakan daftar kata-kata dan matriks yang selnya terdiri dari sel hitam dan putih. Tujuan dari permainan ini adalah mengisi sel-sel putih tersebut dengan seluruh kata yang disediaan sesuai dengan kategori tiap sel, dimana 1 sel hanya diisi oleh 1 digit. Strategi algoritma adalah salah satu mata kuliah informatika. Salah satu strategi algoritma yang dipelajari adalah algoritma Depth-First Search (DFS). Makalah ini akan membahas tentang penyelesaian teka teki silang menggunakan pendekatan algoritma DFS. Dijelaskan pula tentang kompleksitas waktu yang dibutuhkan dan perbandingannya dengan algoritma brute force.**

Keywords— Teka-teki silang, Depth-First Search, kompleksitas.

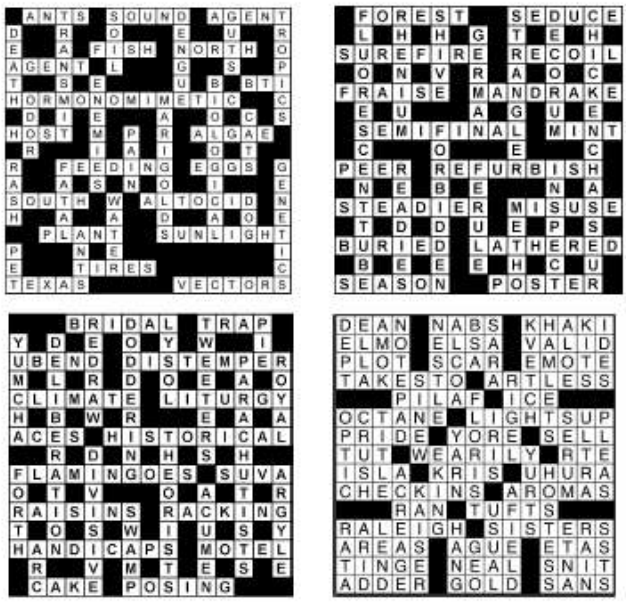
I. PENDAHULUAN

Manusia adalah makhluk yang memiliki banyak aktivitas, seperti bekerja, belajar, mengerjakan tugas besar maupun tugas kecil, dan juga pekerjaan lainnya. Semua aktivitas tersebut dapat membuat manusia menjadi lelah, jenuh, dan juga penat. Untuk melepaskan kejenuhan tersebut, dapat dilakukan beberapa aktivitas seperti bermain permainan maupun mengerjakan teka-teki. Salah satu teka-teki yang biasa dikerjakan adalah teka-teki silang.

Pada teka-teki silang, disediakan matriks yang selnya terdiri dari sel hitam dan juga sel putih.Sel hitam tidak dapat diisi oleh huruf, sedangkan sel putih harus diisi oleh huruf. Terdapat juga beberapa kata yang akan mengisi sel-sel putih tersebut. Penyelesaian teka-teki ini biasanya membutuhkan percobaan *(trial and error)*. Terdapat 2 kategori kata pada teka-teki silang, mendatar dan juga menurun. Setiap kata dikelompokkan berdasarkan kategorinya, menurun maupun mendatar. Tujuan dari permainan ini adalah mengisi semua sel putih pada matriks dengan kata-kata yang telah disediakan. Satu sel putih hanya boleh diisi oleh 1 huruf saja. Teka-teki ini biasanya terdapat pada majalah disamping teka-teki lain seperti mencari kata, teka-teki mengisi angka, Sudoku, ataupun teka-teki logika lainnya.

Terdapat beberapa algoritma yang dapat menyelesaikan teka-teki silang. Algoritma yang sudah dipelajari dalam mata kuliah strategi algoritma dan dapat menyelesaikan teka-teki silang adalah algoritma *brute force*, *depth-first search*, dan juga runut balik.Pada algoritma brute force, akan dilakukan pengisian setiap sel putih dengan daftar kata sesuai dengan kategorinya, dan mencoba semua kemungkinan dari daftar kata tersebut. Algoritma ini memiliki kelebihan dan kekurangan. Kelebihan dari algoritma bruteforce adalah mudah untuk diimplementasikan. Kekurangan dari algoritma bruteforce adalah memiliki kompleksitas yang besar.

Dalam makalah ini, penulis akan membahas tentang penyelesaian teka-teki silang menggunakan algoritma depth-first search dan juga kompleksitas algoritma tersebut.



Gambar 1.1 Ilustrasi Teka-Teki Silang
Sumber :
http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2017-2018/stima17-18.html

II. TEORI DASAR

A. Algoritma Traversal Graf

Algoritma traversal graf adalah algoritma yang mengunjungi simpul pada graf dengan cara sistematik. Terdapat 2 jenis dari algoritma traversal graf sesuai dengan cara mengunjungi simpulnya, yaitu:

1. Pencarian melebar (*breadth-first search/BFS)*
2. Pencarian Mendalam (*Depth-first search/DFS)*.

Algoritma traversal graf juga dapat dibedakan sesuai dengan informasinya, yaitu:

1. Tanpa informasi (*uninformed/blind search)*
   - Tidak ada informasi tambahan
   - Contoh : DFS, BFS, *Depth Limited Search, Iterative Deepening Search, Uniform Cost Search*
2. Dengan informasi (*informed Search)*
   - Pencarian berbasis heuristic
   - Mengetahui non-goal state "lebih menjanjikan" daripada yang lain
   - Contoh: *Best First Search, A**

B. Sejarah Algoritma DFS

Versi pertama dari algoritma DFS ditemukan pada abad ke-19 oleh Matematikawan Perancis, Charles Pierre Tremaux sebagai strategi untuk menyelesaikan labirin. Algoritma tersebut disebut dengan *Tremaux's Algorithm.*

Algoritma Tremaux adalah metode efisien untuk mencari jalan keluar dari sebuah labirin yang membutuhkan penggambaran garis pada lantai untuk menanjadi sebuah jalur, dan dijamin untuk bekerja pada semua labirin yang mempunyai jalan keluar yang benar. Jalur yang belum pernah dikuncungi, Algoritma tersebut digunakan dalam beberapa ratus tahun sebagai Algoritma DFS.

Algoritma DFS untuk sebuah graph sama seperti algoritma *Depth-First Traversal* untuk sebuah tree. Perbedaannya adalah, tidak seperti tree, graph mungkin mengandung cycle, jadi dapat kembali pada node yang awal lagi. Untuk menghindari kondisi tersebut, perlu digunakan suatu metode untuk menandai path-path yang sudah dikunjungi.

C. Properti Algoritma DFS

Waktu dan ruang analisis dari algoritma DFS berbeda sesuai dengan dimana algoritma tersebut diaplikasikan. Dalam teori ilmu komuter, DFS biasanya digunakan untuk mengunjungi semua simpul graph, dan membutuhkan waktu $O(|V| + |E|)$, sesuai dengan ukuran dari graph. Pada teori ilmu komputer, DFS juga menggunakan ruang $O(|V|)$ pada kasus terburuknya untuk menyimpan stack dari setiap simpul pada pencarian jalur saat ini serta set simpul yang sudah dikunjungi. Dengan demikian, batas waktu dan ruang sama seperti algoritma BFS dan pemilihan pada kedua algoritma ini bukan bergantung pada kompleksitas yang dibutuhkan oleh kedua algoritma tetapi lebih kepada urutan simpul hasil dari dua algoritma tersebut.

D. Pengorganisasian Solusi

Semua kemungkinan solusi dari persoalan disebut dengan ruang solusi. Secara formal dapat dinyatakan, bahwa jika xi ∈ S, maka

$$S_1 \times S_2 \times \ldots \times S_n$$

disebut ruang solusi. Jumlah anggota di dalam ruang solusi adalah $|S_1| \cdot |S_2| \cdot \ldots \cdot |S_n|$.

Penyelesaian secara exhaustive search atau brute force adalah dengan menguji setiap kemungkinan solusi yang mungkin untuk dapat menyelesaikan suatu permasalahan. Solusi yang memenuhi atau dapat menyelesaikan permasalahan disebut solusi layak.

Algoritma DFS memperbaiki pencarian solusi secara exhaustive search dengan mencari solusi secara sistematis. Ruang solusi diorganisasikan ke dalam struktur pohon. Pencarian solusi dilakukan dengan mengunjungi (traversal) simpul – simpul di dalam pohon. Pohon yang ditelusuri adalah pohon dinamis (pohon yang dibangun selama pencarian solusi berlangsung). Pohon dinamis menyatakan status – status persoalan pada saat pencarian solusi berlangsung.

E. Prinsip Pencarian Solusi dengan Metode DFS

Akan ditinjau pencarian solusi pada pohon ruang status yang dibangun secara dinamis. Langkah – langkah pencarian solusi pada algoritma DFS adalah sebagai berikut :

1. Solusi dicari dengan membentuk lintasan dari akar ke daun. Simpul – simpul yang sudah dilahirkan dinamakan simpul hidup. Simpul hidup yang diperluas dinamakan simpul-E. Simpul dinomori dari atas ke bawah sesuai urutan kelahiran dari simpul tersebut.
2. Tiap kali simpul-E diperluas, lintasan yang dibangun olehnya bertambah panjang. Jika lintasan yang sedang dibentuk tidak mengarah pada solusi yang diinginkan, maka simpul-E tersebut di-set menjadi simpul mati. Fungsi yang digunakan simpul mati adalah dengan menerapkan fungsi pembatas. Simpul mati tidak akan pernah diperluas lagi.
3. Jika pembentukan lintasan berakhir dengan simpul mati, maka proses pencarian diteruskan dengan membangkitkan simpul anak yang lainnya. Bila tidak ada lagi simpul anak yang dapat mengarah pada solusi yang diinginkan atau tidak dapat dibangkitkan, maka pencarian solusi dilanjutkan dengan melakukan runut-balik ke simpul hidup terdekat (simpul sebelum simpul kita berada sekarang). Selanjutnya simpul ini menjadi simpul-E yang baru. Lintasan baru dibangun bila kita telah menemukan solusi atau tidak ada lagi simpul hidup untuk runut-balik.
4. Pencarian dihentikan bila kita telah menemukan solusi atau tidak ada lagi simpul hidup untuk runut balik.

F. Skema Umum Algoritma DFS

Dibawah ini dijasikan skema umum algoritma DFS dalam dua versi, yaitu versi rekursif (menggunakan pemanggilan prosedur secara berkala) dan versi iterative (menggunakan stack). Skema rekusif lebih tepat digunakan karena algoritma DFS lebih alami

dinyatakan dalam bentuk rekursi. Algoritma di bawah ini akan menghasilkan semua solusi:

1. Versi Rekursif

```
procedure dfs (input pos:integer)
{Mencari semua solusi persoalan dengan metode dfs dengan skema rekursif.
I.S : pos, yaitu posisi atau indeks untuk vector yang sekarang ingin di check
F.S : mengembalikan solusi dalam bentuk yang diinginkan}

If (kondisi sudah memenuhi) {
  Kembalikan solusi dalam bentuk yang diinginkan
}
For (tiap indeks dalam ruang solusi) {
  If (dapat menghasilkan solusi) {
    Masukkan dalam solusi
    Dfs(posisi++) //dilakukan rekursif
  }
}
```

Pemanggilan prosedur pertama kali dfs(0) atau dfs(1) bergantung dengan indeks awal.

2. Versi Iteratif

Menggunakan bantuan standard library Stack. Logika algoritma dfs versi iterative sama dengan logika algoritma dfs versi rekursif.

```
procedure dfs (input pos:integer) {
{Mencari semua solusi persoalan dengan metode dfs dengan menggunakan skema iterative.
I.S : pos, yaitu posisi atau indeks untuk vector yang sekarang ingin di check
F.S : mengembalikan solusi dalam bentuk yang diinginkan}

   stack<int> now;
   now.push(pos) //melakukan pengisian pada stack dengan posisi kita saat ini
   while (!now.empty())
   //mengecek hingga stack kosong
   {
      Int temp = new.top()
      Now.pop()
      For(tiap indeks dalam ruang solusi) {
         If (dapat menghasilkan solusi) {
            Now.push(indeks)
         }
      }
   }
}
```

Pemanggilan prosedur pertama kali dfs(0) atau dfs(1) bergantung dengan indeks awal.

Prosedur cetak solusi adalah sebagai berikut:

```
Procedure cetaksolusi (input x : TabelSolusi)
{Mencetak solusi persoalan
I.S. : solusi dari penyelesaian metode dfs
F.S. : - }

For indeks dalam x {
   Tulis x
}
```

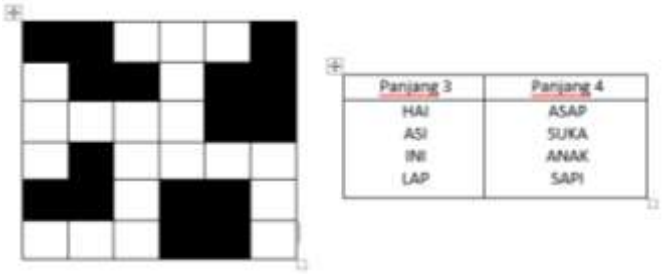
Setiap simpul dalam pohon ruang status berasosiasi dengan sebuah pemangilan rekusif. Kasus terburuk dari algoritma dfs adalah jika jumlah simpul dalam pohon ruang status adalah 2^n atau $n!$ yang membutuhkan waktu dalam $O(p(n)2^n)$ atau $O(q(n)n!)$ dengan $p(n)$ dan $q(n)$ adalah polinom berderajat n yang menyatakan waktu komputasi setiap simpul.

III. PENYELESAIAN TEKA-TEKI SILANG DENGAN MENGGUNAKAN ALGORITMA DFS

A. Langkah Penyelesaian

Pencarian solusi diawali dengan mencari sel – sel mana aja yang harus diisi dan juga kategori tiap sel dalam teka-teki dalam bentuk matriks. Sel-sel tersebut adalah awalan dari setiap sel kosong yang bakal diisi oleh daftar kata. Untuk mencari awalan sel tersebut digunakan algoritma brute force dengan exhaustive search.

Dalam pencarian solusi, algoritma dfs digunakan dalam proses pengisian sel dengan huruf – huruf yang sesuai. Pada pengisian selanjutnya akan dicari apakah ada kata dari daftar kata yang sesuai pada suatu slot tempat tertentu, kata yang dikatakan sesuai apabila panjang katanya sama dengan panjang slot yang bakal diisi dan jika ada dua slot yang bersinggungan (menggunakan sel yang sama), maka huruf pada indeks kata harus sama dengan sel yang sudah terisi. Jika ada kata yang sesuai, proses dilanjutkan pengisian slot tempat berikutnya. Jika tidak ada kata yang sesuai, proses yang telah dijalankan tidak mengarah pada solusi. Sehingga, dilakukan proses *runut balik* dengan mengosongkan sel – sel yang telah diisi sebelumnya. Lalu mencari kata yang sesuai lagi dalam indeks. Untuk mengetahui lebih jelas tentang penggunaan algoritma DFS ini, penulis menggunakan salah satu contoh :

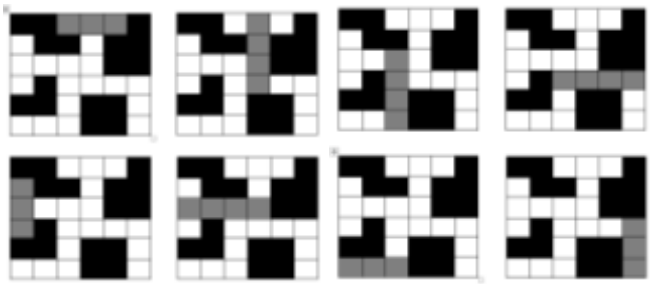


Panjang 3	Panjang 4
HAI	ASAP
ASI	SUKA
INI	ANAK
LAP	SAPI

Gambar 3.1 Contoh teka-teki silang

Langkah pertama yang dilakukan adalah membuat array of awalan sel mana aja yang bakal diisi sesuai dengan kategorinya yaitu mendatar atau menurut dengan cara melakukan iterasi setiap sel yang ada pada matriks dari (0,0) hingga (5, 5) dan melakukan pengecekan. Sebuah sel dikatakan awalan ketika sel kiri dari sel putih tersebut bukanlah sel putih, lalu sel kanan dari sel tersebut adalah putih. Sel tersebut adalah awalan dengan kategori mendatar. Sebuah sel dikatakan awalan juga ketika sel atas dari sel putih tersebut bukanlah sel putih, lalu sel bawah dari sel tersebut adalah putih. Sel tersebut adalah awalan dengan kategori menurun. Dihitung juga panjang dari setiap sel untuk memudahkan dalam penyelesaian menggunakan metode DFS. Dari soal sebelumnya didapatkan daftar slot awal sebagai berikut:

Slot	Sel	Arah	Panjang
1	(0,2)	mendatar	3
2	(0,3)	menurun	4
3	(1,0)	menurun	3
4	(2,0)	mendatar	4
5	(2,2)	menurun	4
6	(3,2)	mendatar	4
7	(3,5)	menurun	3
8	(5,0)	mendatar	3



Gambar 3.2 Slot pengisian kata pada teka-teki

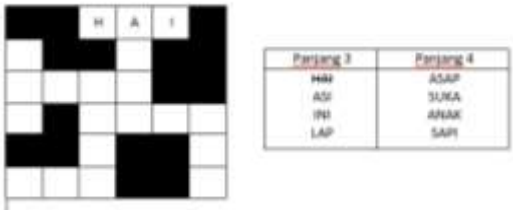
Pengisian kata ke dalam sel yang tersedia dilakukan secara rekursif dengan urutan sesuai pada tabel urutan slot. Pada setiap tahap pengisian dilakukan langkah sebagai berikut:

1. Mencari kata yang belum digunakan yang memiliki panjang yang sama dengan panjang slot.
2. Mengecek apakah kata tersebut sesuai dengan huruf yang sudah ada sebelumnya. (Sel yang berpotongan menggunakan huruf yang sama).
3. Jika ya, isi slot dengan kata tersebut, dan ubah array boolean kata tersebut menjadi true. Lanjutkan proses pengisian untuk slot berikutnya (kembali ke langkah 1).
4. Jika tidak memenuhi, kembali ke langkah 1 hingga semua kata sudah di cek.
5. Jika tidak ada kata yang memenuhi, lakukan runut balik, dengan mengganti slot sebelumnya yang telah terisi dengan kata lain. Kata pada slot yang dihapus kembali diubah status array boolean-nya menjadi false.

Jika dilihat dari pohon ruang sattus, simpul akar adalah teka-teki soal yaitu matriks kosong, dan setiap iterasinya melakukan pengisian dan membentuk simpul baru.

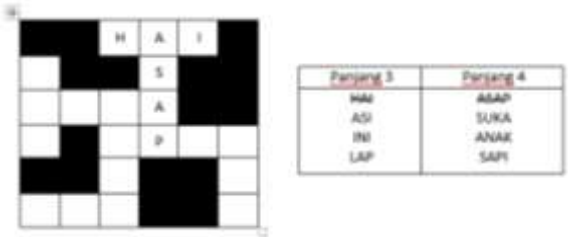
Berikutnya akan ditunjukkan implementasi langkah pengisian pada contoh teka-teki silang yang telah diberikan:

1. Slot pertama memiliki panjang 3 dan kata pertama yang memiliki panjan 3 adalah “HAI”. Isikan kata “HAI” pada slot 1.



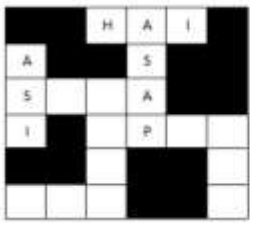
Gambar 3.3 Pembangkitan simpul 1

2. Slot kedua memiliki panjang 4. Huruf pertama dari kata yang mengisi slot 2 haruslah memiliki huruf pertama “A”. Kata pertama yang memiliki panjang 4 adalah “ASAP”. “ASAP” memenuhi kriteria slot kedua karena memiliki huruf pertama yaitu “A”, sama dengan slot yang telah terisi sebelumnya. Isikan kata “ASAP” pada slot 2.



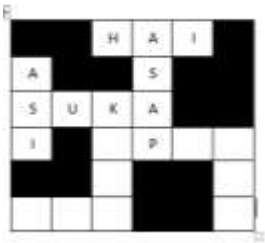
Gambar 3.4 Pembangkitan simpul 2

3. Slot ketiga memiliki panjang 3. Kata pertama yang memiliki panjang 3 dan masih memiliki status true adalah “ASI”. Isikan kata “ASI” pada slot 3.



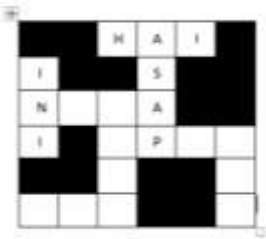
Gambar 3.5 Pembangkitan simpul 3

4. Slot keempat memiliki panjang 4. Kata yang mengisi slot 4 haruslah memiliki huruf pertama "S" dan huruf keempat "A". Kata pertama yang memiliki panjang 4, memiliki huruf pertama "S" dan huruf keempat "A" adalah "SUKA". Isikan kata "SUKA" pada slot 4.



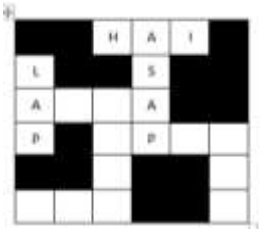
Gambar 3.6 Pembangkitan simpul 4

5. Pada pengisian slot 5(memiliki panjang 4), tidak ada kata yang panjangnya 4 dan memiliki huruf pertama "K", sehingga dilakukan runut balik. Runut balik dilakukan dengan mengganti kata yang digunakan sebelumnya (yaitu slot 4) dengan kata lain. Tetapi, sudah tidak ada lagi kata yang memenuhi kriteria untuk slot 4. Maka dilakukan runut balik lagi, dengan mengganti kata pada slot 3. Kata yang memenuhi slot 3 adalah "INI".



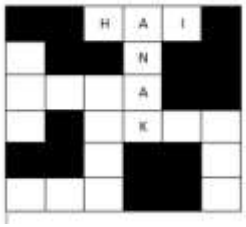
Gambar 3.7 Runut balik simpul 3

6. Pada pengisian slot 4(memiliki panjang 4), tidak ada kata yang panjangnya 4 dan memiliki huruf pertama "N" dan hurufkeempat "A", sehingga dilakukan runut balik. Bactraing dilakukan dengan mengganti kata pada slot ke 3. Kata yang memenushi slot ke 3 adalah "LAP"



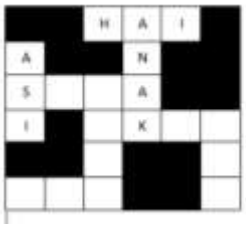
Gambar 3.8 Runut balik simpul 3

7. Pada pengisian slot 4(memiliki panjang 4) =, tidak ada kata yang panjangnya 4 dan memiliki huruf pertama "A" dan huruf terakhir "A", sehingga dilakukan runut balik dengan mengganti kata pada slot 3. Karena sudah tidak ada lagi kata yang dapat mengisi slot 3, maka dilakukan runut balik dengan mengganti kata pada slot 2. Kata yang memenuhi slot 2 adalah "ANAK"



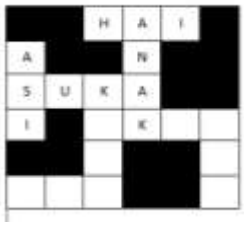
Gambar 3.9 Runut balik simpul 2

8. Slot 3 memiliki panjang 3. Kata yang memenuhi slot 3 adalah "ASI"



Gambar 3.10 Pembangkitan simpul 3

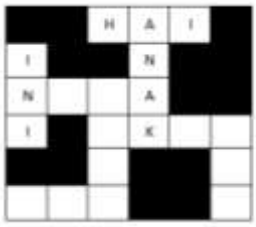
9. Slot 4 memiliki panjang 4. Kata yang mempunyai panjang 4 dan memiliki huruf pertama "S" dan huruf terakhir "A" adalah "SUKA".



Gambar 3.11 Pembangkitan simpul 4

10. Pada pengisian slot 5(memiliki panjang 4), tidak ada kata yang panjangnya 4 dan memilki huruf pertama "K", sehingga dilakukan runut balik. Tetapi, sudah tidak ada lagi kata yang memenuhi kriteria untuk slot

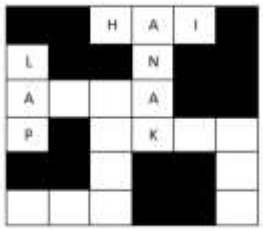
4. Maka dilakukan runut balik lagi, dengan mengganti kada pada slot 3. Kata yang memenuhi slot 3 adalah "INI"



Panjang 3	Panjang 4
~~HAI~~	ASAP
ASI	SUKA
~~INI~~	~~ANAK~~
LAP	SAPI

Gambar 3.12 Runut balik simpul 3

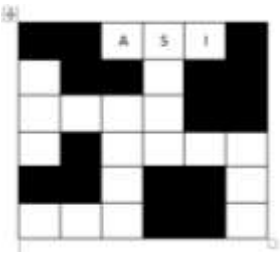
11. Pada pengisian slot 4(memiliki panjang 4), tidak ada kata yang panjangnya 4 dan memiliki huruf pertama "N" dan huruf terakhir "A". sehingga dilakukan runut balik, dan mengisi slot 3 dengan kata "LAP".



Panjang 3	Panjang 4
~~HAI~~	ASAP
ASI	SUKA
INI	~~ANAK~~
~~LAP~~	SAPI

Gambar 3.13 Runut balik simpul 3

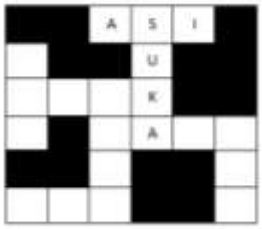
12. Pada pengisian slot 4(memiliki panjang 4), tidak ada kata yang panjangnya 4 dan memiliki huruf pertama "A" dan huruf terakhir "A". sehingga dilakukan runut balik. Tidak ada kata lagi yang dapat memenuhi slot 3. Maka dilakukan runut balik lagi. Untuk slot 2, tidak ada lagi kata yang memenuhi kriteria. Dilakukan runut balik lagi, lalu mengisi slot 1 dengan kata "ASI"



Panjang 3	Panjang 4
HAI	ASAP
~~ASI~~	SUKA
INI	ANAK
LAP	SAPI

Gambar 3.14 Runut balik simpul 1

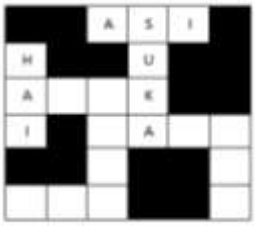
13. Slot 2 memiliki panjang 4. Kata yang memenuhi slot 2 adalah "SUKA"



Panjang 3	Panjang 4
HAI	ASAP
~~ASI~~	~~SUKA~~
INI	ANAK
LAP	SAPI

Gambar 3.15 Pembangkitan simpul 2

14. Slot 3 memiliki panjang 3. Kata yang memenuhi slot 3 adalah "HAI".



Panjang 3	Panjang 4
~~HAI~~	ASAP
~~ASI~~	~~SUKA~~
~~INI~~	~~ANAK~~
LAP	SAPI

Gambar 3.16 Pembangkitan simpul 3

15. Slot 4 memiliki panjang 4. Kata yang memenuhi slot 4 adalah "ANAK".

Panjang 3	Panjang 4
~~HAI~~	ASAP
~~ASI~~	~~SUKA~~
~~INI~~	~~ANAK~~
LAP	SAPI

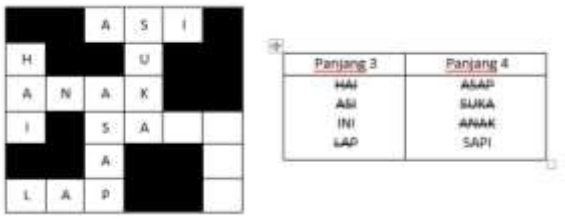
Gambar 3.17 Pembangkitan simpul 4

16. Slot 5 memiliki panjang 4. Kata yang memenuhi slot 5 adalah "ASAP".

Panjang 3	Panjang 4
~~HAI~~	~~ASAP~~
~~ASI~~	~~SUKA~~
~~INI~~	~~ANAK~~
LAP	SAPI

Gambar 3.18 Pembangkitan simpul 5

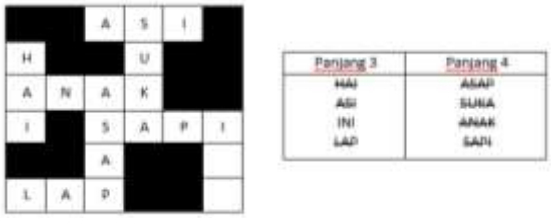
17. Slot 6 memiliki panjang 3. Kata yang memenuhi slot 6 adalah "LAP".



Panjang 3	Panjang 4
HAI	ASAP
ASI	SUKA
INI	ANAK
LAP	SAPI

Gambar 3.19 Pembangkitan simpul 6

18. Slot 7 memiliki panjang 4. Kata yang memenuhi slot 7 adalah “SAPI”.

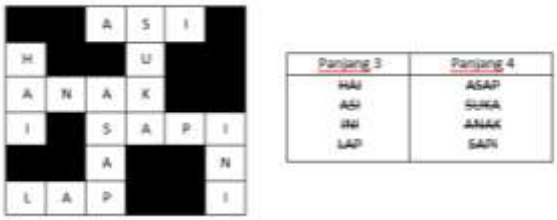


Panjang 3	Panjang 4
HAI	ASAP
ASI	SUKA
INI	ANAK
LAP	SAPI

Gambar 3.20 Pembangkitan simpul 7

19. Slot 8 memiliki panjang 3. Kata yang memenuhi slot 8 adalah “INI”.

Setelah proses tersebut dilakukan hingga seluruh slot/mastrks terisi, akan didapatkan solusi sebagai berikut:



Panjang 3	Panjang 4
HAI	ASAP
ASI	SUKA
INI	ANAK
LAP	SAPI

Gambar 3.21 Solusi Akhir

Jika dilihat dari pohon ruang solusi, akan dibangkitkan sebanyak total 20 simpul hingga ditemukan solusi. Pembangkitan pohon ruang solusi sama seperti solusi yang telah ditampilkan.

B. Penghitungan Kompleksitas

Penghitungan kompleksitas waktu didapakan dari jumlah pemeriksaan terhadap himpunan solusi yang telah ada. Jika pengerjaan dilakukan dengan algoritma brute force dengan exhaustive search, akan dicari semua kombinasi pengisian kata pada slot. Sehingga kompleksitasnya adalah $O(n!)$, dimana n adalah jumlah kata yang ada pada daftar kata. Sehingga kompleksitas waktu pada teka-teki contoh adalah 8! = 40320.

Penghitungan kompleksitas menggunakan algoritma DFS, dilihat dari pohon ruang status, hanya diperlukan pembangkitan 20 simpul. Hal ini disebabkan karena pada algoritma DFS, tidak semua kemungkinan dicoba. Maka, kompleksitas DFS pada teka-teki contoh adalah 20. Namun pada kasus terburuk dari algoritma DFS ini dapat terjadi jika solusi baru ditemukan setelah mencoba semua kemungkinan.

C. Uji coba pada program

Berdasarkan langkah-langkah yang telah dijelaskan sebelumnya, penulis mencoba untuk membuat program untuk menyelesaikan teka-teki silang ini. Program diimplementasikan dalam bahasa C++, dan input baca dari file eksternal dengan format .txt yang berisi matriks dari teka-teki silang dan juga daftar kata. Struktur data yang digunakan dalam penyusunan program ini antara lain:

1. Matriks of char, yang digunakan sebagai tempat mengisi kata-kata.
2. Tipe bentukan DaftarKata yang memuat array of string berisi daftar kata, array of integer berisi panjang dari setiap dari setiap kata, dan juga array of Boolean untuk menyimpan apakah kata tersebut digunakan atau belum.
3. Tipe bentukan slot yang menyimpan posisi awal slot, arah, dan juga panjang slot. Slot ini disatukan dalam sebuah array of slot.

Pada file masukan, karakter “#” melambangkan sel hitam dan karakter “-“ melambangkan sel putih.

```
17
#----#-----#-----
-##-##-###-##-##-
--##-#----#-----#-
------#-###-##-##-
--##-#-####-#-#---
---------------##-
#-#-#-###-##-#-#-
----#-#-#-#-----#
#-###-#-#-##-#-##
-##-------#-----#-
--##-#-#-#-##-###-
------#-#-------#-
-##-##-##-##-#-#-
##-----##---------
-###-#-##########
-###-----########
-----#####-------
ANTS;SOUND;AGENT;FISH;NORTH;AGENT;HORMONOMIMETIC;BTI;HOST;ALGAE;FEEDING;EGGS;SOUTH;
ALTOCID;PLANT;SUNLIGHT;TIRES;TEXAS;VECTORS;DEATH;WASH;PET;TRANSMIT;ODOR;FATAL;ANTS;
WATER;PAIN;ENEMIES;SOIL;DENGUE;GUTS;TROPICS;CATS;BIOLOGICAL;DOG;MARIGOLDS;GENETICS;
```

Gambar 3.22 Contoh teka teki silang

Program akan menamplkan matriks solusi dari persoalan tersebut dan juga wakt yang dibutuhkan untuk menyelesaikan permasalahan tersebut.

```
#ANTS#SOUND#AGENT
D##R##O###E##U##R
E##A#FISH#NORTH#O
AGENT#L###G##S##P
T##S#E####U#B#BTI
HORMONOMIMETIC##C
#D#I#E###A##O#C#S
HOST#M#P#R#ALGAE#
#R###I#A#I##O#T##
R##FEEDING#EGGS#G
A##A#S#N#O##I###E
SOUTH#W#ALTOCID#N
H##A##A##D##A#O#E
##PLANT##SUNLIGHT
P###N#E##########I
E###TIRES#######C
TEXAS#####VECTORS

Waktu Eksekusi : 0.016s
```

Gambar 3.23 Solusi Akhir

IV. KESIMPULAN

Penyelesaian suatu persoalan dapat diselesaikan dengan berbagai algoritma untuk menyelesaikannya. Tetapi untuk setiap kasus, dapat ditentukan algoritma mana yang dapat menyelesaikan permasalahan tersebut dengan paling mangkus dan sangkil. Salah satu algoritma yang terbaik adalah algoritma DFS. Permasalahan yang dapat diselesaikan oleh algoritma DFS dengan mangkus dan sangkil adalah penyelesaian maze, teka-teki silang. Untuk penyelesaian teka-teki silang, kompleksitas terbaik adalah menemukan solusi tersebut tanpa runut balik, dan kompleksitas terburuk adalah menemukan solusi dengan mencoba semua kemungkinan solusi yang ada.

V. PENUTUP

Puji Syukur pertama kali saya ucapkan kepada Tuhan Yang Maha Esa karena berkat-Nya yang melimpah sehingga penulis dapat menyelesaikan makalah ini dengan baik. Ucapan terima kasih juga penulis sampaikan kepada kedua orang tua, keluarga, serta teman – teman yang terus memberikan dukungan baik secara moral maupun doa. Penulis juga turut mengucapkan terima kasih kepada Bu Masayu, selaku dosen dari mata kuliah Strategi Algoritma. Akhir kata, penulis memohon maaf atas salah kata atau ketidaksempurnaannya makalah ini. Penulis berharap makalah ini dapat bermanfaat bagi orang lain dan terus dikembangkan sehingga menjadi semakin baik lagi.

REFERENCES

[1] http://qa.geeksforgeeks.org/6653/qa.geeksforgeeks.org/6653/what-is-the-history-of-depth-first-search-algorithm

[2] http://maukar.staff.gunadarma.ac.id/Downloads/files/45649/Penerapan+BFS+dan+DFS+pada+Pencarian+Solusi.ppt

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, 13 Mei 2018

Muh. Habibi Haidir / 13516085