Penyelesaian *Crossword Puzzle* dengan Algoritma Runut-balik dan *Greedy*

Intan Nurjanah / 13516131[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13516131@std.stei.itb.ac.id

***Abstrak*—Permainan *crossword puzzle* atau teka-teki silang adalah salah satu permainan asah otak yang sangat digemari. Aturan permainan ini adalah memasangkan semua kata yang tersedia ke dalam kotak-kotak yang bersesuaian baik mendatar maupun menurun. Algoritma runut-balik sudah dapat menyelesaikan permainan ini, akan tetapi waktu yang diperlukan untuk menyelesaikannya masih bisa dioptimalkan. Melalui makalah ini, penulis mencoba mengimplementasikan metode *greedy* bersama algoritma runut-balik untuk menyelesaikan *crossword puzzle* dengan tujuan memperoleh waktu penyelesaian yang lebih cepat.**

***Kata kunci—crossword puzzle, greedy,* runut-balik.**

I. PENDAHULUAN

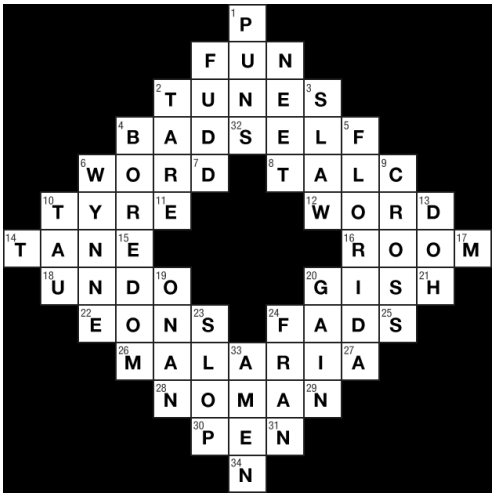
Sejarah ilmu pengetahuan dan teknologi menunjukkan bahwa perkembangannya sejak manusia ada, hingga saat ini, dan sampai masa mendatang dilandasi oleh kreaivitas manusia untuk memenuhi kebutuhan hidup yang lebih sejahtera. Dewasa ini, perkembangan teknologi semakin berkembang pesat. Kemampuan manusia yang tinggi dan terus berkembanglah yang telah menumbuhkan perkembangan tersebut.

Permainan *crossword puzzle* atau teka-teki silang adalah salah satu permainan asah otak yang sangat digemari. Perkembangan teknlogi telah memungkinkan penyelesaian *crossword puzzle* menggunakan mesin. Algoritma yang dapat menyelesaikan permainan ini, selain *bruteforce*, adalah algoritma runut-balik. Akan tetapi, waktu yang diperlukan untuk menyelesaikan permainan ini dengan runut-balik masih dapat dioptimalkan. Melalui makalah ini, penulis mencoba mengimplementasikan metode *greedy* bersama algoritma runut-balik untuk menyelesaikan *crossword puzzle* dengan tujuan memperoleh waktu penyelesaian yang lebih cepat.

II. CROSSWORD PUZZLE

Crossword puzzle atau permainan teka-teki silang pertama kali diterbitkan pada 21 Desember 1913 oleh Arthur Wynne pada majalah *New York World*. Permainan ini sebenarnya mengadopsi permainan kuno bernama *Pompeii*, yang jika diterjemahkan ke dalam bahasa Inggris menjadi *Magic Square*. Teka-teki silang ini disebut sebagai teka-teki silang pertama dengan Arthur Wynne sebagai penemunya.

Crossword puzzle berhasil menarik minat para pembaca hingga akhirnya permainan ini menjadi konten mingguan majalah *New York World*. Buku kumpulan *crossword puzzle* pertama terbit pada tahun 1924 oleh Simon dan Schuster. Buku tersebut pun menjadi salah satu benda terpopuler pada tahun 1924. Kesuksesan tersebut diikuti oleh terbitnya buku-buku serupa hingga akhirnya terbit buku *Asah Otak* di Indonesia pada era 70an. Hingga saat ini, teka teki silang masih dapat dengan mudah ditemukan di berbagai majalah atau *website* di internet.



Gambar 1. Crossword Puzzle pertama
Sumber: https://www.npr.org/sections/theprotojournalist/2013/12/19/250339795/100-years-of-solvitude-a-reported-crossword-puzzle dengan sedikit perubahan.

Crossword puzzle adalah pemainan yang memasangkan semua kata yang tersedia ke dalam kotak-kotak yang bersesuaian. Terdapat dua jenis ruang kosong pada permainan *Crossword puzzle*, yaitu jawaban mendatar (horizontal) dan jawaban menurun (vertikal). Pemain dinyatakan menang apabila dapat mengisi seluruh kotak kosong dengan jawaban yang benar.

III. DASAR TEORI

A. Algoritma Runut-balik

Runut-balik (*backtracking*) adalah algoritma yang berbasis pada *DFS* untuk mencari solusi persoalan secara lebih mangkus.

Runut-balik, yang merupakan perbaikan dari algoritma *bruteforce*, secara sistematis mencari solusi persoalan di antara semua kemungkinan solusi yang ada. Hanya pencarian yang mengarah ke solusi saja yang selalu dipertimbangkan. Akibatnya, waktu pencarian dapat dihemat. Runut-balik lebih alami dinyatakan dalam algoritma rekursif. Kadang-kadang disebutkan pula bahwa runut-balik merupakan bentuk tipikal dari algoritma rekursif.

Properti umum metode runut-balik:

1. Solusi persoalan
   Solusi dinyatakan sebagai vektor dengan *n-tuple*:
   $X = (x_1, x_2, \ldots, x_n)$ $x_i \in$ himpunan berhingga S_i
   Mungkin saja $S_1 = S_2 = \ldots = S_3$
2. Fungsi pembangkit nilai x_k
   Dinyatakan sebagai:
   $T(k)$
   $T(k)$ membangkitkan nilai untuk x_k, yang merupakan komponen vektor solusi.
3. Fungsi pembatas
   Dinyatakan sebagai:
   $B(x_1, x_2, \ldots, x_k)$
   Fungsi pembatas menentukan apakah $(x_1, x_2, \ldots, x_k)$ mengarah ke solusi. Jika ya, maka pembangkitan nilai untuk x_{k+1} dilanjutkan, tetapi jika tidak, maka $(x_1, x_2, \ldots, x_k)$ dibuang dan tidak dipertimbangkan lagi dalam pencarian solusi.
   Fungsi pembatas tidak selalu dinyatakan sebagai fungsi matematis. Ia dapat dinyatakan sebagai predikat yang bernilai *true* atau *false*, atau dalam bentuk lain yang ekivalen.

Adapun *pseudo-code* dari algoritma runut-balik adalah sebagai berikut.

```
procedure RunutBalik(input k: integer)
{ Mencari semua solusi persoalan dengan
      metode runut-balik; skema rekursif
  Masukan: k, yaitu indeks komponen vektor
      solusi, x[k]
  Keluaran: solusi x = (x[1], x[2], …, x[n])
}
Algoritma:
    for tiap x[k] yang belum dicoba sedemikian
          sehingga (x[k] ← T(k)) and
          B(x[1], x[2], …, x[k]) = true do
        if (x[1], x[2], …, x[k]) adalah lintasan
              dari akar ke daun then
            CetakSolusi(x)
        endif
        {tentukan nilai untuk x[k+1]}
        RunutBalik(k+1)
    endfor
```

Pemanggilan prosedur pertama kali: `RunutBalik(n)`

B. Algoritma Greedy

Algoritma *greedy* mungkin merupakan metode paling populer untuk memecahkan persoalan optimasi. Algoritma ini sederhana dan lempang (*straightforward*). Algoritma *greedy* membentuk solusi langkah per langkah (*step by step*). Terdapat banyak pilihan yang perlu dieksplorasi pada setiap langkah solusi. Oleh karena itu, pada setiap langkah harus dibuat keputusan yang terbaik dalam menentukan pilihan. Keputusan yang telah diambil pada suatu langkah tidak dapat diubah lagi pada langkah selanjutnya.

Pendekatan yang digunakan dalam algoritma *greedy* adalah membuat pilihan yang "tampaknya" memberikan perolehan terbaik, yaitu dengan membuat pilihan optimum lokal pada setiap langkah dengan harapan bahwa sisanya mengarah ke solusi optimum global.

Elemen-elemen algoritma *greedy*:

1. Himpunan kandidat, *C*
   Himpunan kandidat adalah himpunan yang berisi elemen-elemen pembentuk solusi.
2. Himpunan solusi, *S*
   Himpunan solusi adalah himpunan yang berisi kandidat-kandidat yang terpilih sebagai solusi persoalan.
3. Fungsi seleksi (*selection function*)
   Fungsi seleksi adalah fungsi yang memilih kandidat kandidat yang paling memungkinkan mencapai solusi optimal. Kandidat yang telah dipilih pada suatu langkah tidak pernah dipertimbangkan lagi pada langkah selanjutnya.
4. Fungsi kelayakan (*feasible*)
   Fungsi ini memeriksa apakah suatu kandidat yang telah dipilih dapat memberikan solusi yang layak, yakni kandidat tersebut bersama-sama dengan himpunan solusi yang sudah terbentuk tidak melanggar kendala yang ada. Kandidat yang layak dimasukkan ke dalam himpunan solusi, sedangkan kandidat yang tidak layak dibuang dan tidak pernah dipertimbangkan lagi.
5. Fungsi obyektif
   Fungsi ini memaksimumkan atau meminimumkan nilai solusi.

Adapun *pseudo-code* dari algoritma *greedy* adalah sebagai berikut.

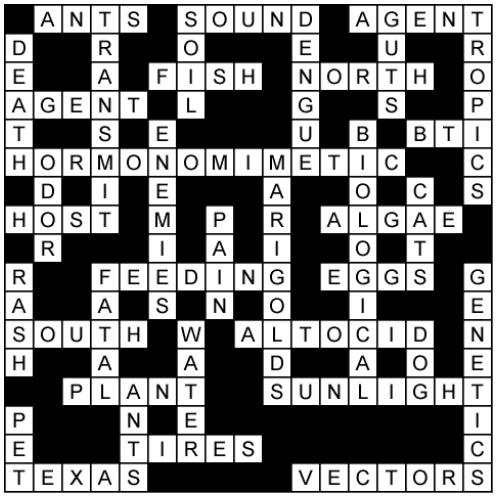
```
function Greedy(input C: himpunan_kandidat) →
      himpunan kandidat
{ Mengembalikan solusi dari persoalan optimasi
      dengan algoritma greedy
  Masukan: himpunan kandidat C
  Keluaran: himpunan solusi yang bertipe
      himpunan_kandidat

Deklarasi
    x: kandidat
    S: himpunan kandidat
Algoritma:
    S ← {} {inisialisasi S dengan kosong}
    while (not SOLUSI(S)) and (C ≠ {}) do
        {pilih sebuah kandidat dari C}
        x ← SELEKSI(C)
        {elemen himpunan kandidat berkurang satu}
        C ← C - {x}
        if LAYAK(S U {x}) then
            S ← S U {x}
     endif
    endwhile
    {SOLUSI(S) or C = {} }

    if SOLUSI(S) then
        return S
    else
        return {}
    endif
```

`SOLUSI(S)` akan mengembalikan *true* jika `S` adalah solusi dari persoalan. `SELEKSI(C)` akan mengembalikan kandidat yang dipilih dari `C` berdasarkan kriteria tertentu. Sedangkan, `LAYAK(S)` akan mengembalikan nilai *true* jika `S` merupakam solusi yang tidak melanggar kendala.

IV. Penerapan Algoritma Runut-balik dan Optimasi Pemilihan Kata Secara Greedy

A. Metode Pemilihan Kata

Hal yang membantu dalam menyusun *crossword puzzle* adalah keunikan dari kata yang ada. Kata yang paling uniklah yang akan memiliki prioritas tertinggi untuk dipasangkan ke dalam slot (ruang kosong) yang tersedia. Keunikan yang penulis maksud di sini dipandang dari segi jumlah karakter yang dimiliki kata tersebut. Sebagai contoh, penulis akan menggunakan Gambar 2. untuk menggambarkan metode ini.



Gambar 2. Contoh *Crossword Puzzle*
Sumber: http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2017-2018/Tugas-Kecil-1-(2018).pdf

Dari Gambar 2, kita akan memiliki daftar kata sebagai berikut.

No	Kata	Jumlah karakter
1.	ANTS	4
2.	SOUND	5
3.	AGENT	5
4.	FISH	4
5.	NORTH	5
6.	AGENT	5
7.	HORMONOMIMETIC	14
8.	BTI	3
9.	HOST	4
10.	ALGAE	5
11.	FEEDING	7
12.	EGGS	4
13.	SOUTH	5
14.	ALTOCID	7
15.	PLANT	5
16.	SUNLIGHT	8
17.	TIRES	5
18.	TEXAS	5
19.	VECTORS	7
20.	DEATH	5
21.	RASH	4
22.	PET	3
23.	TRANSMIT	8
24.	ODOR	4
25.	FATAL	5
26.	ANTS	4
27.	WATER	5
28.	PAIN	4
29.	ENEMIES	7
30.	SOIL	4
31.	DENGUE	6
32.	GUTS	4
33.	TROPICS	7
34.	CAST	4
35.	BIOLOGICAL	10
36.	DOG	3
37.	MARIGOLDS	9
38.	GENETICS	8
39.	CATS	4

Tabel 1. Daftar kata dan jumlah karakternya

Yang disebut dengan kata yang unik adalah kata yang memiliki jumlah karakter paling berbeda. Dari Tabel 1, dapat diperoleh informasi bahwa terdapat:

- 13 kata dengan 4 karakter
- 12 kata dengan 5 karakter
- 1 kata dengan 14 karakter.
- 3 kata dengan 3 karakter
- 5 kata dengan 7 karakter.
- 3 kata dengan 8 karakter.
- 1 kata dengan 6 karakter.
- 1 kata dengan 10 karakter.
- 1 kata dengan 9 karakter.

Sehingga kata yang memiliki prioritas tertinggi sampai terendah berdasarkan jumlah karakternya adalah sebagai berikut.

(1) 9, 10, 6, 14 (masing-masing 1 kata)
(2) 8, 3 (masing-masing 3 kata)
(3) 7 (5 kata)
(4) 5 (12 kata)
(5) 4 (13 kata)

Selanjutnya, perlu dilakukan pengurutan antara kata dengan prioritas sama. Urutkan kembali kata dengan prioritas sama terurut mengecil berdasarkan jumlah karakter.

Prioritas pemilihan	Jumlah Karakter	Jumlah Kata
(1)	14	1
(2)	10	1
(3)	9	1
(4)	6	1
(5)	8	3
(6)	3	3
(7)	7	5
(8)	5	12
(9)	4	13

Tabel 2. Daftar kata berdasarkan jumlah karakter dan prioritas pemilihannya

Dengan cara ini, maka akan didapatkan urutan sebagai berikut.

14, 10, 9, 6, 8, 3, 7, 5, 4

Cara ini dipilih dengan asumsi bahwa semakin banyak jumlah karakter, maka semakin banyak pula persilangan dengan kata lain pada slot yang ditempati oleh kata tersebut.

Berikut adalah skema dari pengurutan daftar kata.

```
procedure sortKata(input/output arrKata:
      himpunan_kata)
{mengurutkan kata agar terurut mengecil sesuai
panjang karakternya}

function getKataByLength(length: integer,
      arrKata: himpunan_kata, n: integer) →
      himpunan_kata
{mengembalikan himpunan kata sejumlah n dengan
panjang karakter = length}

function getDaftarKataByPrioritas(arrkata:
      himpunan_kata, size: integer) →
      himpunan_kata
{Mengurutkan himpunan kata berdasarkan prioritas
 Masukan: arrKata, yaitu himpunan kata yang akan
       Diurutkan; size adalah batas kata
       terpanjang yang mungkin ada, biasanya
       size dipilih dari panjang papan crossword
       puzzle
 Keluaran: himpunan kata berdasarkan prioritas
       (yang paling unik)
}
Deklarasi
    matKata: matriks kata berukuran size x 2
  {kolom pertama mendefinisikan jumlah karakter
   dan kolom kedua mendefinisikan jumlah kata
   dengan panjang karakter pada kolom pertama}
    prioKata: himpunan_kata
    temp: himpunan_kata
    i, j: integer
Algoritma:
    {urutkan himpunan kata}
    sortKata(arrkata)

    isikan jumlah kata yang ada pada arrkata ke
    dalam matKata sesuai panjang karakternya

    urutkan matKata terurut membesar sesuai
    dengan jumlah katanya (pada kolom kedua)

    {isikan setiap kata pada matKata ke dalam
    prioKata}
    i ← 0
    for array pada setiap baris matKata do
        if(array[1] ≠ 0) then
            temp ← getKataByLength(array[0],
                  arrKata, array[1])
            {salin temp ke prioKata}

            for j [0..jumlah elemen pada temp] do
                prioKata[i] ← temp[j]
                i ← i + 1
            endfor
        endif
    endfor

    → prioKata
```

B. Metode Penempatan Kata pada Slot

Setelah dilakukan pengurutan kata berdasarkan prioritasnya, masalah selanjutnya adalah bagaimana menempatkan kata-kata tersebut jika pada prioritas yang sama terdapat lebih dari satu kata? Untuk itu, dilakukan pencarian solusi menggunakan algoritma runut-balik.

Berikut adalah skema penempatan dengan runut-balik.

```
function applyKataToSlot(i: integer, k: integer)
      → boolean
{menempatkan kata arrKata[i] pada arrSlot[k];
 mengembalikan true apabila berhasil;
 mengembalikan false apabila gagal, tetapi hasil
 penempatan karakter yang sudah dilakukan tidak
 dapat dibatalkan}

procedure removeKataFromSlot(i: integer, k:
      integer)
{mengembalikan state saat kata belum dipasangkan
 pada slot}

function findCrosswordSolution(input i: integer)
      → boolean
{Mencari semua solusi yang mungkin dengan
     metode runut-balik dan mengisikan papan
     dengan kata yang mungkin
 Masukan: i, yaitu indeks elemen array of kata
     arrKata[i]
 Keluaran: true jika solusi ditemukan
}
Deklarasi
    matState: matrix of character
   {mendeklarasikan state papan permainan}
Algoritma:
    if i > jumlah elemen pada arrKata then
        → true
    endif

    matState ← matPapan
   {matPapan adalah matrix of character yang
    menggambarkan papan crossword puzzle yang
    akan ditampilkan}

    for tiap arrSlot[k] yang belum dicoba
          sedemikian do
        if (arrSlot[k] belum diisi penuh and
              panjang arrSlot[k] = panjang
              karakter arrKata[i] then
            {jika slot memenuhi kriteria,
             pasangkan kata}
            if applyKataToSlot(i, k) and
                  findCrosswordSolution(i+1) then
                → true
            endif
            {kembali ke state sebelumnya jika
             kata tidak cocok dengan slot}
            removeKataFromSlot(i, k)
            matPapan ← matState
        endif
    endfor

    → false
```

Pemanggilan fungsi pertama kali:

```
{deklarasi arrKata, matPapan, arrSlot, dan variable
 lain}
arrKata ← getDaftarKataByPrioritas(arrKata, size)
findCrosswordSolution(0)
```

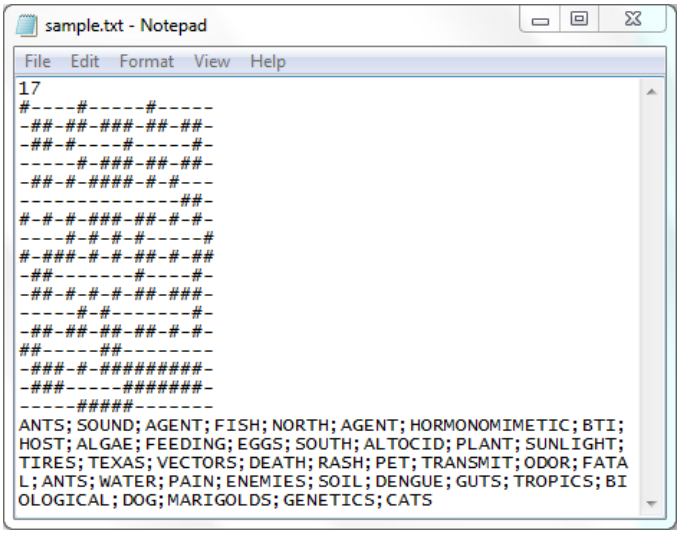
Metode *greedy* dipakai dalam memilih kata sesuai dengan prioritasnya, dan algoritma runut balik dipakai dalam menempatkan kata yang dipilih ke dalam slot yang cocok.

C. Pendekatan Lain

Pendekatan lain untuk menyelesaikan *crossword puzzle* adalah dengan membalik metode yang telah dijelaskan di atas. Yaitu, dengan menyeleksi slot sesuai dengan prioritasnya. Pada metode pemilihan kata di atas, kata yang didahulukan pada prioritas berdasarkan jumlah karakternya adalah kata terpanjang, sedangkan pada metode pemilihan slot, slot yang akan didahulukan adalah slot dengan persilangan terbanyak.

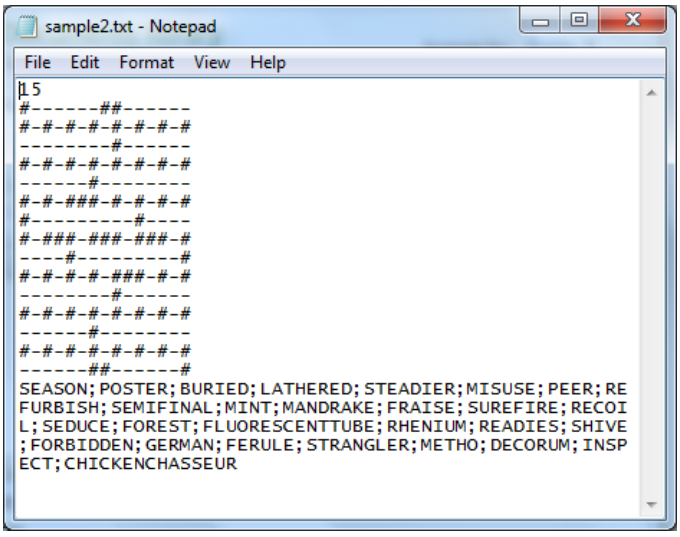
V. HASIL PENGUJIAN

Dari metode yang telah dijelaskan di atas, penulis membuatkan programnya dengan bahasa Java. Selain itu, penulis juga membuat program pembanding yang hanya menggunakan algoritma runut-balik saja (tanpa pengurutan kata terlebih dulu). Berikut adalah hasil pengujiannya.



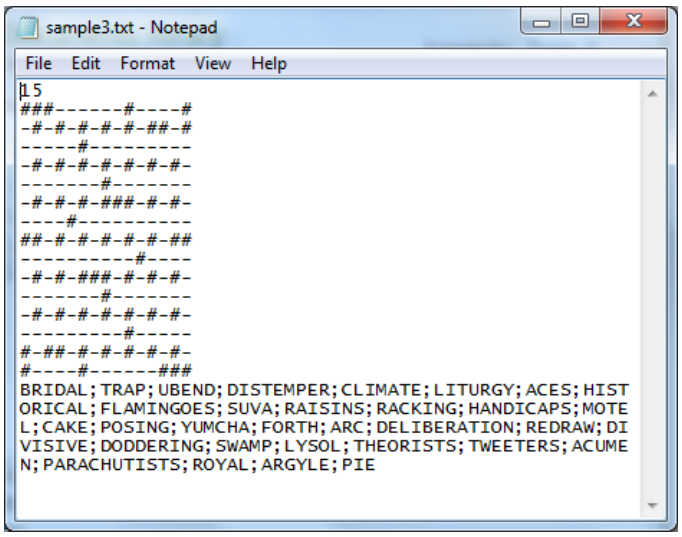
Gambar 3. Sampel 1 masukan program

Pada sampel 1, eksekusi program dengan algoritma runut-balik saja diselesaikan dengan waktu 730420 nanodetik. Sedangkan waktu penyelesaian oleh program dengan algoritma runut-balik dan *greedy* membutuhkan waktu 363400 nanodetik.



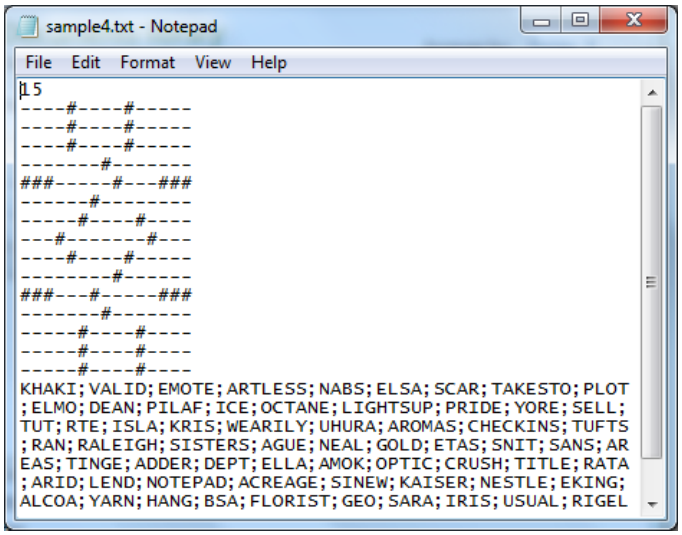
Gambar 4. Sampel 2 masukan program

Pada sampel 2, eksekusi program dengan algoritma runut-balik saja diselesaikan dengan waktu 930192 nanodetik. Sedangkan waktu penyelesaian oleh program dengan algoritma runut-balik dan *greedy* membutuhkan waktu 824765 nanodetik.



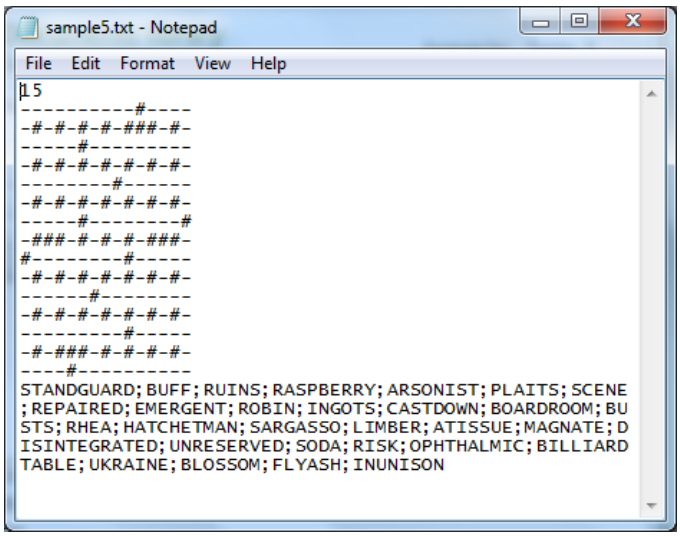
Gambar 5. Sampel 3 masukan program

Pada sampel 3, eksekusi program dengan algoritma runut-balik saja diselesaikan dengan waktu 4263728 nanodetik. Sedangkan waktu penyelesaian oleh program dengan algoritma runut-balik dan *greedy* membutuhkan waktu 278965 nanodetik.



Gambar 6. Sampel 4 masukan program

Pada sampel 4, eksekusi program dengan algoritma runut-balik saja diselesaikan dengan waktu 951941602 nanodetik. Sedangkan waktu penyelesaian oleh program dengan algoritma runut-balik dan *greedy* membutuhkan waktu 1843591 nanodetik.



Gambar 7. Sampel 5 masukan program

Terakhir, Pada sampel 5, eksekusi program dengan algoritma runut-balik saja diselesaikan dengan waktu 383926 nanodetik. Sedangkan waktu penyelesaian oleh program dengan algoritma runut-balik dan *greedy* membutuhkan waktu 227650 nanodetik.

Dari hasil pengujian kelima sampel di atas, dapat terlihat bahwa penggunaan algoritma runut-balik disertai dengan *greedy* membutuhkan waktu penyelesaian yang lebih pendek daripada hanya menggunakan algoritma runut-balik saja. Hal ini karena penggunaan *greedy* membantu dalam pemilihan kata yang paling unik menurut panjang karakternya sehingga akan mengurangi jumlah operasi perbandingan yang dilakukan dalam runut-balik.

VI. Simpulan

Permainan *crossword puzzle* adalah permainan yang diadaptasi dari permainan kuno *Pompeii* atau *Magic Square* dan salah satu permainan asah otak yang paling digemari. Karena perkembangan teknologi, *crossword puzzle* dapat diselesaikan oleh program. Algoritma runut-balik saja sudah dapat menyelesaikan permainan ini, akan tetapi waktu yang diperlukan untuk menyelesaikannya masih bisa dioptimalkan. Salah satu upaya untuk mengoptimalkan waktu penyelesaiannya adalah dengan menerapkan metode *greedy*. Yaitu, dengan cara memilih kata terunik yang tersedia berdasarkan kesamaan panjang karakternya. Pengurutan kata ini kemudian mengurangi operasi perbandingan yang dilakukan dalam runut-balik. Dari hasil pengujian yang dilakukan, algoritma runut-balik yang disertai metode *greedy* terbukti lebih cepat menyelesaikan masalah *crossword puzzle* dibandingkan hanya menggunakan algoritma runut-balik saja.

VII. Ucapan Terima Kasih

Penulis mengucapkan syukur kepada Allah S.W.T, karena atas rahmat-Nya, penulis dapat menyelesaikan makalah ini. Selain itu, penulis juga mengucapkan terima kasih kepada Ibu Dr. Nur Ulfa Maulidevi S.T, M.Sc. dan Bapak Dr. Ir. Rinaldi Munir, M.T selaku dosen pengampu kuliah IF2211 Strategi Algoritma yang telah memberikan ilmu yang bermanfaat kepada penulis.

Referensi

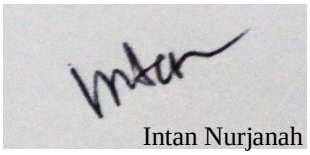
[1] Andi Nugraha FAP, "Sejarah Lahirnya Permainan Teka-teki Silang", http://tau-sejarah.blogspot.co.id/2013/05/sejarah-lahirnya-permainan-teka-teki.html?m=1 Diakses pada 12 Mei 2018 17.51.

[2] Rinaldi Munir, "Diktat Kuliah IF3051 Strategi Algoritma," Institut Teknologi Bandung, 2007, hlm 41–45, 148–152.

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, 12 Mei 2018



Intan Nurjanah
13516131