Pencarian Kata dengan Poin Terbesar di Permainan *Scrabble* dengan Memanfaatkan Algoritma *Greedy*

Rizky Andyno Ramadhan – 13516063
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
rizkydichi@gmail.com

***Abstrak*—Permainan kata sudah menjadi tradisi di dalam suatu bahasa. Sudah cukup banyak permainan kata yang pernah diciptakan dan menghibur generasi masa kini, dari permainan kata tradisional sampai permainan kata modern. Bahkan, beberapa permainan kata pun sudah memilliki turnamen *official*-nya sendiri. Salah satu diantaranya adalah permainan kata *Scrabble*. Tujuan dari permainan kata ini adalah dengan mendapatkan skor setinggi mungkin dengan membuat kata panjang di papan dengan menyusun tujuh huruf yang dimiliki pemain dan mendapatkan ubin bonus yang melipatgandakan poin tersebut. Dengan bantuan Algoritma *Greedy*, kita bisa melihat susunan huruf dengan poin terbesar sehingga kita dapat memprediksi poin terbesar yang dapat diperoleh oleh pemain pada tiap gilirannya.**

***Kata Kunci—Scrabble, greedy, dictionary,* poin terbesar, algoritma.**

I. PENDAHULUAN

Scrabble adalah sebuah permainan kata yang sangat populer di kalangan keluarga khususnya di daerah Amerika Serikat dan Eropa.[1]

Gambar 1.1 *Lexiko*, sebelum berganti nama menjadi *Scrabble*
(Sumber: https://img.wonderhowto.com/img/original/76/47/63448071788647/0/634480717886477647.jpg)

Permainan ini berawal dari ide seorang arsitek berkebangsaan Amerika Serikat ketika negaranya sedang dilanda depresi berat akibat jatuhnya pasar di sana. Pada tahun 1933, ia menggolongkan permainan yang populer pada masa itu menjadi tiga kategori: Permainan angka seperti dadu dan bingo, permainan bergerak seperti catur dan dam (*checker*), dan permainan kata seperti anagram. Satu hal yang membuat permainan kata tidak sepopuler permainan lainnya adalah permainan kata tidak memiliki sistem penilaian. Akhirnya Butts mencoba untuk menggabungkan ide anagram dengan teka teki silang (yang sedang populer pada masa itu) menjadi sebuah permainan berjudul *Lexiko*, permainan kata dengan sistem penilaian. Waktu terus berlalu dari tahun 1930 hingga 1950 dan permainan tersebut disempurnakan kembali menjadi permainan yang sudah memiliki merek dagang, *Scrabble*, dengan sistem penilaian per huruf yang berbobot berdasarkan distribusi huruf yang muncul di halaman depan koran *New York Times*.[2]

Scrabble terdiri dari sebuah papan dengan *grid* 15x15, seratus ubin huruf penyusun kata, dan empat buah rak yang digunakan oleh tiap pemain untuk menyimpan tujuh huruf yang dapat dimainkan di tiap gilirannya.

Di awal permainan, tiap pemain akan mengambil tujuh huruf pertama dan tiap pemain juga mengambil satu huruf dari kantong *Scrabble* untuk menentukan pemain mana yang akan memulai permainan.

Gambar 1.2 Sebuah papan *Scrabble* lengkap dengan empat buah rak masing-masing berisi tujuh huruf
(Sumber: https://images-na.ssl-images-amazon.com/images/I/81vG4h7krIL._SL1500_.jpg)

Para pemain saling bergantian menyusun huruf-huruf di

papan yang akan memberikan mereka skor. Bobot skor pada tiap hurufnya pun berbeda-beda, dari hanya satu poin untuk huruf yang paling sering digunakan dalam perbendaharaan kata, sampai dengan 10 poin (huruf Q dan huruf Z dalam permainan *Scrabble* versi bahasa Inggris, atau huruf J dan huruf Z dalam permainan *Scrabble* versi bahasa Indonesia).

Pemain menyusun strategi agar huruf-huruf yang disusunnya, selain mengumpulkan banyak poin, juga sambil memblok pemain lawan agar tidak dapat meraih ubin bonus. Ubin bonus dapat mengalikan skor dari huruf ataupun kata sebesar dua kali atau tiga kali dari huruf-huruf yang telah diletakkan di papan.

Bagian tersulit dari permainan *Scrabble* ini adalah bagaimana caranya untuk mendapatkan susunan huruf yang paling strategis yang dapat menghasilkan poin terbanyak di setiap gilirannya. Permasalahan utamanya adalah ketika pemain tidak memiliki perbendaharaan kata yang mumpuni, sehingga kata yang dihasilkan tidak maksimal dan dapat menguntungkan pihak lawan. Yang akan dibahas di makalah ini adalah pemanfaatan algoritma *greedy* yang dapat digunakan untuk mencari susunan huruf dengan nilai tertinggi yang dapat dicapai oleh pemain dalam satu gilirannya, dan detil-detil mengenai penerapannya.

II. LANDASAN TEORI

1. Algoritma *Greedy*

Algoritma *Greedy* merupakan metode yang paling populer untuk memecahkan permasalahan yang mementingkan optimasi (mencari nilai maksimum/minimum)[5].

Algoritma *Greedy*, seperti namanya, selalu mengutamakan pilihan yang terbaik pada saat itu juga. Maksudnya adalah, pada setiap langkahnya algoritma ini membuat pilihan optimum lokal dengan harapan pilihan ini akan mengarahkan pilihan berikutnya ke optimum global.[6]

Algoritma *greedy* adalah algoritma yang memecahkan masalah langkah per langkah. Pada setiap langkah, ambil pilihan yang terbaik yang dapat diperoleh pada saat itu tanpa memperhatikan konsekuensi ke depan, dengan memakai prinsip "*take what you can get now!*".[5]

Algoritma *greedy* hanya memiliki satu kesempatan untuk menemukan solusi yang optimal sehingga algoritma ini tidak akan mundur kembali (layaknya algoritma *backtracking*, yang tidak dibahas dalam makalah ini).[6]

Algoritma ini memiliki beberapa kelebihan dan kekurangan, seperti:

1. Cukup mudah untuk menerapkan algoritma *greedy* (atau bahkan banyak algoritma *greedy*) untuk memecahkan suatu permasalahan.
2. Menganalisis kinerja *run time* dari algoritma *greedy* jauh lebih mudah daripada algoritma lain (seperti *Divide and Conquer*). Pada algoritma *divide and conquer*, kinerjanya tidak dapat diukur apakah algoritma tersebut cepat atau lambat karena tiap tahap rekursif, ukuran permasalahan jadi lebih kecil dan jumlah upamasalah menjadi lebih banyak.
3. Kelemahannya dari algoritma *greedy* ini adalah sangatlah sulit untuk mencari pencarian solusi yang benar, karena memang algoritma *greedy* ini sulit untuk mencari yang benar-benar paling optimal.[5]

Dalam implementasinya, algoritma ini memiliki beberapa elemen, yaitu

1. Himpunan kandidat (C)
   Himpunan ini berisi seluruh kandidat solusi dari permasalahan.
2. Himpunan solusi (S)
   Himpunan ini berisi macam-macam solusi dari permasalahan tersebut.
3. Fungsi seleksi (*selection function*)
   Fungsi yang memilih elemen-elemen berdasarkan kebutuhan dari permasalahan yang ada.
4. Fungsi kelayakan (*feasible*)
   Fungsi yang menentukan apakah solusi sudah memenuhi kriteria dari fungsi pembatas.
5. Fungsi obyektif
   Fungsi yang mengoptimisasi pencarian solusi.

Dengan kata lain, algoritma *greedy* melibatkan pencarian sebuah himpunan bagian, S, dari himpunan kandidat, C; yang dalam hal ini, S harus memenuhi beberapa kriteria yang ditentukan, yaitu menyatakan suatu solusi dan S dioptimisasi oleh fungsi obyektif.[5]

Bagian terpenting dari algoritma ini adalah pemilihan elemen pada tiap tahapnya. Elemen yang dipilih berdasarkan fungsi seleksi yang sudah ditetapkan. Fungsi seleksi ditentukan berdasarkan heuristik, yaitu teknik berdasarkan data hasil percobaan dan intuisi, tidak berdasarkan analisis teoritis.[7]

Beberapa contoh kegunaan dari algoritma *greedy* yaitu:

1. *Travelling Salesman Problem*
2. *Job Scheduling Problem*
3. *Knapsack Problem* (dan *Fractional Knapsack Problem*)
   Permasalahan seputar pohon dan graf (pohon merentang minimum, *map coloring*)

2. Sedikit Mengenai String Matching

a. Definisi String matching

String matching adalah algoritma untuk menemukan pola *substring* di dalam sebuah badan teks. Permasalahan ini dapat disebut dengan "mencari jarium di dalam tumpukan jerami".

Terdapat beberapa algoritma yang dapat digunakan untuk menemukan pola di dalam teks, yaitu algoritma brute force, algoritma *Knuth-Morris-Pratt*, dan algoritma *Boyer-Moore*. Dari ketiga algoritma tersebut, algoritma *Boyer-Moore* merupakan algoritma yang paling mangkus karena *nature*-nya yang banyak menyingkat teks yang polanya tidak cocok.

b. Algoritma *Boyer-Moore*

Algoritma pencocokan pola *Boyer-Moore* didasarkan pada dua teknik, yaitu teknik *looking-glass* dan teknik *character-jump*.

Untuk teknik *looking-glass*, pencocokan karakter dilakukan dari **belakang** pola, bukan dari depan.

Untuk teknik *character-jump*, ketika posisi pattern yang dicocokan tidak sesuai dengan teks (misal T[i] != x, dengan T adalah teks dan P adalah pola), maka ada 3 kasus:

a. Apabila P mengandung karakter x dimanapun, geser P ke kanan untuk mencocokkan dengan kemunculan terakhir karakter x di pola dengan T[i]
b. Apabila P mengandung karakter x dimanapun, tetapi pergeseran ke kanan tidak dimungkinkan (karena kemunculan karakter x yang terakhir sudah pernah muncul sebelumya), geser P ke kanan sebanyak satu karakter (ke T[i+1]).
c. Apabila tidak sesuai dengan kasus a dan kasus b, maka geser P ke kanan sejauh (z – 1) dengan z adalah jumlah huruf yang ada di P.[8]

III. Implementasi Algoritma Greedy untuk Mencari Nilai Tertinggi dalam Scrabble

1. Implementasi Dasar

Tujuan dari algoritma ini adalah mencari kata dengan skor tertinggi, sehingga perlu diketahui bobot-bobot dari semua huruf yang ada di permainan *Scrabble*.

Berikut adalah tabel distribusi huruf untuk *Scrabble* berbahasa Inggris dan berbahasa Indonesia yang akan digunakan dalam penghitungan skor.

Nilai huruf	Huruf
0	* (huruf bebas)
1	A E I L N O R S T U
2	D G
3	B C M P
4	F H V W Y
5	K
8	J X
10	Q Z

Tabel T1 Distribusi nilai huruf untuk *Scrabble* versi bahasa Inggris[3]

Nilai huruf	Huruf
0	* (huruf bebas)
1	A E I N O R S T U
2	K M
3	D G
4	H L P
5	B F W Y
8	C V
10	J Z

Tabel T2 Distribusi nilai huruf untuk *Scrabble* versi bahasa Indonesia[4]

Untuk mencocokan apakah hasil susunan huruf sudah sahih atau belum, digunakanlah *dictionary* (kamus) yang berisi daftar kata-kata yang sahih dalam bahasa Inggris atau bahasa Indonesia (tergantung pada permainannya). Berikut adalah contoh *dictionary* yang akan digunakan (sumber: www.puzzlers.org/pub/wordlists/ospd.txt)

Score	Word	Occurence
...	...	...
10	Burbot	2b 1o 1r 1t 1u
9	Burden	1b 1d 1e 1n 1r 1u
16	Burdock	1b 1c 1d 1k 1o 1r 1u
8	Bureau	1a 1b 1e 1r 2u
16	Bureaux	1a 1b 1e 1r 2u 1x
10	Burettes	1b 2e 1r 1s 2t 1u
...	...	...

Tabel T3 Contoh *dictionary* bahasa Inggris yang akan digunakan dalam implementasi algoritma ini

Pada dasarnya, permainan *Scrabble* mirip dengan satu permainan kata yang lain, yaitu anagram. Pada permainan anagram, kata-kata lain dapat dibentuk dari huruf-huruf penyusun suatu kata (sebagai contoh, huruf-huruf penyusun dari kata "kasur" dapat disusun ulang menjadi kata "sukar" atau "rusak"). Huruf-huruf pembentuk kata-kata tersebut pun sama. Dengan memanfaatkan aturan tersebut, pencarian di dalam dictionary pun dapat di percepat dengan mencatat *occurence* (jumlah kemunculan) tiap huruf dari tiap kata yang ada di dalam *dictionary*.

Algoritma pencarian kata dengan poin terbesarnya sebagai berikut:

1. Hitung *occurence* (jumlah kemunculan) dari tiap huruf yang ada di rak pemain.
2. Pindai seluruh *occurence* tersebut di dalam *dictionary* satu per satu dimulai dari huruf dengan bobot tertinggi.
    a. Apabila ditemukan *occurence* di *dictionary* yang merupakan *subset* dari *occurence* yang ada di rak pemain, tandai indeks yang memiliki nilai maksimal.
    b. Apabila terdapat kata dengan poin maksimal lebih dari satu, maka tandai semua entri.
    c. Apabila terdapat kata dengan nilai yang lebih tinggi, hapus tanda yang sebelumnya, lalu tandai kata yang baru.

    Tandai entri yang cocok dan entri yang tidak cocok sama sekali (entri yang tidak memiliki subset dari *occurence* yang sedang dicari) agar tidak perlu diperiksa lagi occurence-nya pada iterasi selanjutnya.
3. Ulangi tahap dua hingga seluruh huruf telah digunakan.
4. Pilih kata yang telah ditandai sebelumnya. Apabila terdapat lebih dari satu kata, tampilkan semuanya.

Pencarian solusi di atas dijamin akan memberikan solusi yang optimum karena pasti ditemukan *subset* dari *occurence* yang terdapat di rak dan hasil yang dikeluarkan pasti memberikan poin maksimal.

Sebagai contoh, seorang pemain memiliki susunan huruf sebagai berikut

X L T O A L O

Dihitung dari nilai tiap huruf, urutan *occurence*-nya adalah sebagai berikut:

x1 a1 l2 o2 t1

Langkah pertama, algoritma akan memilih x1 untuk dicari di dalam *dictionary*. Karena di dalam *dictionary Scrabble* tidak ada kata yang terdiri dari satu huruf, maka langkah ini tidak akan menandai indeks manapun.

Langkah kedua, algoritma akan memilih a1, sehingga akan mencari kata yang memiliki subset dari x1 a1 dari *dictionary*. Setelah itu, akan terdapat kata yang ditandai, yaitu "AX" yang bernilai 9 poin.

Langkah ketiga, algoritma akan memilih l2, sehingga akan mencari kata yang memiliki subset dari x1 a1 l2 dari *dictionary*. Perhatikan bahwa l1 **juga merupakan subset dari l2.** Maka akan ditemukan kata-kata berikut dari *dictionary*:

Score	Word	Occurence
2	Al	a1 l1
3	All	a1 l2
9	Ax	a1 x1
2	La	a1 l1
10	Lax	a1 l1 x1

Tabel T4 Daftar subset dari x1 a1 l2

Tanda dari "AX" akan dihapus karena terdapat kata yang memiliki poin lebih besar, yaitu kata "LAX" yang bernilai 10 poin. Perhatikan juga untuk entri "AX" diberi tanda (warna merah) untuk memberitahu algoritma agar tidak perlu mengecek entri yang pernah dilalui sebelumnya untuk mempercepat pemrosesan.

Langkah keempat, algoritma akan memilih o2, sehingga akan mencari kata yang memiliki subset dari x1 a1 l2 o2 dari *dictionary*. Akan ditemukan kata-kata berikut:

Score	Word	Occurence
2	Al	a1 l1
3	All	a1 l2
9	Ax	a1 x1
2	La	a1 l1
10	Lax	a1 l1 x1
2	Lo	l1 o1
3	Loo	l1 o2
10	Lox	l1 o1 x1
4	Olla	a1 l2 o1
9	Ox	o1 x1
10	Oxo	o2 x1

Tabel T5 Daftar subset dari x1 a1 l2 o2

Oleh karena terdapat kata lain yang bernilai 10, maka entri "LOX" dan "OXO" pun akan ditandai juga.

Langkah terakhir, algoritma akan memilih t1, sehingga akan dicari kata yang memiliki subset dari x1 a1 l2 o2 t1.

Score	Word	Occurence
...	...	...
3	All	a1 l2
5	Allot	a1 l2 o1 t1
3	Alt	a1 l1
4	Alto	a1 l1 o1 t1
2	At	a1 t1
5	Atoll	a1 l2 o1 t1
9	Ax	a1 x1
14	Axolotl	x1 a1 l2 o2 t1
2	La	a1 l1
10	Lax	a1 l1 x1
...	...	...

Tabel T6 Daftar subset dari x1 a1 l2 o2 t1

Karena seluruh huruf sudah dicari, maka susunan huruf yang memberikan poin maksimum adalah kata "AXOLOTL" yang bernilai 14 poin.

2. Implementasi Tingkat Lanjut

a. Implementasi dengan huruf yang sudah ada di papan

Karena sifatnya *scrabble* yang mirip dengan teka-teki silang, bisa saja pemain ingin membentuk kata dari kata yang sudah ada di papan. Untuk itu, dibutuhkan algoritma *string matching* untuk mencari kombinasi huruf yang tepat sambil mencocokkan *substring* yang ada di papan dengan *dictionary* yang ada. Agar algoritma tetap mangkus, algoritma *string matching* yang digunakan adalah algoritma *Boyer-Moore* dan *string matching* hanya dilakukan apabila jumlah huruf dari kata dari entri yang *occurence*-nya sudah sesuai melebihi dari jumlah huruf dari kata yang sudah ada di papan.

Untuk algoritmanya sendiri, terdapat modifikasi untuk *occurence* yang ingin dicari. Sekarang, *occurence* yang akan dicari dari *dictionary* adalah *occurence* dari tiap huruf yang ada di rak pemain, ditambah dengan *occurence* tiap huruf dari kata yang ada di papan. Setelah itu, pencarian dimulai dari *occurence* tiap huruf dari kata di papan ditambah dengan *occurence* dari rak dengan nilai tertinggi.

Mengambil contoh dari sebelumnya, apabila pemain ingin mencari susunan huruf dengan kata "BACK" yang sudah berada di papan, maka *occurence* yang ingin dicari adalah

b1 a1 c1 k1 | x1 a1 l2 o2 t1

Pada langkah pertama, program akan mencari entri dengan *occurence* b1 a1 c1 k1 x1. Ketika algoritma dijalankan, entri yang akan tersaring adalah

Score	Word	Occurence
4	Ab	a1 b1
9	Ax	a1 x1
4	Ba	a1 b1
12	Back	a1 b1 c1 k1
7	Cab	a1 b1 c1
6	Ka	a1 k1
9	Kab	a1 b1 k1

Tabel T7 Daftar subset dari b1 a1 c1 k1 x1

Perhatikan kembali bahwa entri "BACK" terpilih bukan hanya karena memiliki poin tertinggi, tetapi juga valid ketika dilakukan *string matching* dengan pola "BACK".

Langkah berikutnya, karena pada rak terdapat huruf 'a' yang juga terdapat di papan, maka jumlah kemunculannya akan ditambahkan satu. Maka, algoritma akan mencoba untuk mencari entri yang ber-subset dengan b1 a2 c1 k1 x1.

Score	Word	Occurence
2	Aa	a2
4	Ab	a1 b1
5	Aba	a2 b1
13	Aback	a2 b1 c1 k1
9	Ax	a1 x1
4	Ba	a1 b1
5	Baa	a2 b1
12	Back	a1 b1 c1 k1
7	Cab	a1 b1 c1
6	Ka	a1 k1
9	Kab	a1 b1 k1

Tabel T8 Daftar subset dari b1 a2 c1 k1 x1

Oleh karena entri "ABACK" memiliki poin terbesar dan juga memuat *substring* "BACK", maka hapus tanda dari entri "BACK" dan tandai entri "ABACK" dengan nilai 13 poin.

Dengan melakukan langkah-langkah berikutnya mengikuti algoritma yang telah ditulis di atas, pada langkah terakhir, kata yang dipilih adalah kata "BACKLOT" dengan nilai 15 poin.

Score	Word	Occurence
13	Aback	a2 b1 c1 k1
12	Back	a1 b1 c1 k1
15	Backlot	a1 b1 c1 k1 l1 o1 t1

Tabel T9 Daftar subset dari b1 a2 c1 k1 x1 l2 o2 t1 yang mengandung pola "BACK" di entrinya

b. Implementasi dengan solusi terbaik untuk tiap jumlah huruf

Untuk implementasi ini, alternatif pertama adalah kita cukup memisahkan dictionary untuk tiap hurufnya sehingga kita memiliki *dictionary* untuk tiap jumlah huruf per katanya. Pencarian akan dilakukan berkali-kali untuk tiap jumlah hurufnya. Secara logika algoritma ini akan berjalan sebanyak jumlah huruf yang dimungkinkan di dalam kamus *Scrabble*, sehingga mungkin akan berjalan berkali-kali lipat lebih lambat daripada algoritma biasanya.

Untuk alternatif kedua, jalankan algoritma seperti biasa, tetapi simpan nilai tertinggi untuk masing-masing jumlah huruf. Dirujuk dari tabel T5, optimum lokal untuk masing-masing jumlah huruf adalah kata "OX" untuk dua huruf yang bernilai 9 poin, kata "LAX" "LOX" dan "OXO" untuk tiga huruf yang bernilai 10 poin, dan kata "OLLA" untuk empat huruf yang bernilai 4 poin. Alternatif ini akan jauh lebih cepat untuk dieksekusi apabila dibandingkan dengan alternatif pertama, akan tetapi implementasi dalam bentuk *coding* akan jauh lebih sulit karena banyak hal yang perlu diperhatikan agar algoritma ini dapat berjalan dengan baik.

IV. Kesimpulan

Algoritma *Greedy* dapat dimanfaatkan untuk banyak masalah yang membutuhkan solusi optimal dan optimasi yang terbaik. Salah satu manfaatnya adalah penerapannya dalam pencarian kata dengan poin terbesar di permainan kata *Scrabble*.

Dengan memanfaatkan algoritma *greedy* ini, pemain atau pelatih dari seorang pemain dapat menentukan kata yang akan memberikan poin maksimal pada setiap gilirannya. Tentunya ini dapat digunakan sebagai tolok ukur pemain dalam menentukan performanya dalam bermain permainan ini. Selain itu, ini juga dapat digunakan sebagai alat bantu dalam bermain *Scrabble* apabila pemain tidak dapat menentukan kata apa yang akan disusun dengan menggunakan huruf-huruf yang dimilikinya.

Tentunya algoritma yang ditulis di sini belum tentu merupakan algoritma termangkus dan algoritma ini dapat berubah sewaktu-waktu di masa depan. Masih banyak optimasi yang dapat dilakukan agar pemrosesan menjadi lebih efektif dan efisien. Penulis sangat berharap kepada para pembaca untuk ikut serta mengembangkan algoritma ini agar algoritma ini menjadi lebih sempurna dan lebih bermanfaat untuk digunakan oleh banyak orang.

V. Ucapan Terima Kasih

Pertama-tama, penulis ingin menyampaikan terima kasih yang sebesar-besarnya kepada Allah Swt. untuk semua berkah dan umur yang telah diberikan. Oleh karena berkat-Nya pula penulis dapat menyelesaikan masalah ini.

Penulis juga ingin berterima kasih kepada keluarga, terutama orang tua penulis yang sangat mendukung keberjalanan studi penulis dengan doa dan dukungan

moralnya.

Selain itu, penulis juga ingin berterima kasih kepada Bapak Rinaldi Minur karena berkatnya, penulis mendapatkan berbagai macam pengetahuan mengenai algoritma dan strategi penerapannya di kehidupan nyata yang tentunya akan bermanfaat bagi penulis di masa mendatang.

Terima kasih juga untuk para pembaca yang setia membaca makalah ini sampai akhir, karena berkat pembaca, penulis menjadi tahu bahwa penulis telah berkontribusi dalam menyumbangkan ilmu keinformatikaan ini kepada pembaca sekalian.

REFERENSI

[1] https://www.telegraph.co.uk/news/newstopics/howaboutthat/3776732/Scrabble-60-facts-for-its-60th-birthday.html
Diakses pada tanggal 11 Mei 2018 pukul 14.59

[2] https://scrabble.hasbro.com/en-us/history
Diakses pada tanggal 11 Mei 2018 pukul 15.07

[3] https://scrabblewizard.com/scrabble-tile-distribution/
Diakses pada tanggal 11 Mei 2018 pukul 16.24

[4] https://ipfs.io/ipfs/QmXoypizjW3WknFiJnKLwHCnL72vedxjQkDDP1mXWo6uco/wiki/Scrabble_letter_distributions.html
Diakses pada tanggal 11 Mei 2018 pukul 16.27

[5] http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2017-2018/Algoritma-Greedy-(2018).pdf
Diakses pada tanggal 12 Mei 2018 pukul 14.36

[6] https://www.hackerearth.com/practice/algorithms/greedy/basics-of-greedy-algorithms/tutorial/
Diakses pada tanggal 12 Mei 2018 pukul 14.44

[7] http://gabrielistrate.weebly.com/uploads/2/5/2/6/2526487/curs10.pdf
Diakses pada tanggal 12 Mei 2018 pukul 15.18

[8] http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2017-2018/Pencocokan-String-(2018).pdf
Diakses pada tanggal 12 Mei 2018 pukul 15.35

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, 11 Mei 2018

Rizky Andyno Ramadhan / 13516063