Aplikasi Decrease and Conquer Pada Analisis Fungsi Monotonik

Jun Ho Choi Hedyatmo 13518044
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): 13518044@std.stei.itb.ac.id

***Abstract*—Dalam matematika, fungsi memiliki banyak karakteristik unik yang membedakan satu fungsi dengan fungsi lainnya. Kemonotonan fungsi adalah salah satunya. Ada beberapa hal menarik yang dapat dianalisa dalam fungsi yang bersifat monoton. Dan hal-hal ini memungkinkan untuk menggunakan teknik-teknik tertentu untuk mencari nilai-nilai optimal untuk suatu permasalahan.**

Keywords—decrease and conquer, fungsi monotonik, binary search

I. Pendahuluan

Fungsi merupakan salah satu hal di matematika yang memiliki banyak kegunaan. Dalam dunia algoritma fungsi juga banyak digunakan untuk menyelesaikan masalah. Namun dalam pemrograman terkadang definisi matematis dari sesuatu tidak dapat digunakan secara semestinya. Terdapat computational *error* yang mungkin terjadi saat program menjalankan suatu formula matematika. Contohnya saja untuk mencari akar kuadrat yang di bulatkan pada program. Jika menggunakan fungsi *sqrt* maka ada kemungkinan terjadi *rounding error* dan membuat nilai yang dihasilkan salah. Namun masalah ini dapat diselesaikan dengan mengorbankan sedikit kompleksitas waktu untuk mencapai jawaban yang pasti benar.

Untuk mencari akar kuadrat dengan pembulatan diatas dengan formula yang ada mungkin akan menghasilkan kompleksitas konstan atau *O(1)*, namun dapat terjadi *rounding error* sehingga apakah hal yang tepat untuk tetap menggunakan cara diatas untuk masalah seperti ini?. Jika domain permasalahan dalam bilangan real ini tidak akan menjadi masalah, namun dalam domain bilangan bulat, ini menjadi masalah. Karena itu dibutuhkan cara lain, metode yang memungkinkan mencari jawaban dalam domain bilangan bulat dan menghasilkan nilai yang eksak dan tepat.

II. Dasar Teori

A. Fungsi

Fungsi adalah relasi yang memetakan sebuah nilai ke satu nilai lainnya yang unik dan selalu konsisten. dari sini didapat bahwa tidak mungkin fungsi memetakan satu nilai ke lebih dari satu nilai. himpunan bilangan yang mendefinisikan ruang nilai masukan fungsi disebut dengan *domain* dari fungsi tersebut. sementara nilai hasil dari fungsi disebut *range* atau *codomain.* pada permasalahan yang akan dibahas disini, fungsi yang digunakan akan memiliki *domain* bilangan bulat.

B. Fungsi Monotonik

Fungsi monotonik adalah sebuah fungsi yang menerima input sebuah *ordered set* (seperti bilangan bulat) dan menghasilkan nilai yang juga *ordered set* (entah itu *increasing* ataupun *decreasing)*. Secara matematis didefinisikan sebagai berikut

$$x \leq y \Rightarrow f(x) \leq f(y)$$

atau

$$x \leq y \Rightarrow f(x) \geq f(y)$$

untuk sebuah fungsi f dan semua nilai x dan y. Sifat monoton ini berguna untuk beberapa hal juga karena jika kita berada di suatu titik pada fungsi tersebut, kita tahu bahwa jika kita terus bergerak ke arah x positif maka nilai fungsi pasti akan terus naik atau terus turun. Demikian sebaliknya jika kita bergerak ke arah x negatif maka nilai fungsi juga pasti akan terus naik atau terus turun (tergantung jenis fungsi monotoniknya).

C. Decrease and Conquer

Decrease and Conquer adalah metode desain algoritma dengan memecah persoalan menjadi beberapa sub-persoalan yang lebih kecil, tetapi selanjutnya hanya memproses satu sub-persoalan saja. Berbeda dengan *divide and conquer* yang memproses semua sub-persoalan dan menggabung semua solusi setiap sub persoalan.

Decrease and Conquer terdiri dari dua tahapan: yang pertama adalah *Decrease,* yaitu mereduksi persoalan menjadi beberapa persoalan yang lebih kecil (biasanya dua sub-persoalan). lalu tahap selanjutnya adalah *Conquer* yaitu memproses satu sub-persoalan secara rekursif (atau iteratif) dengan memilih sub-persoalan tadi sesuai kebutuhan (karena dalam *Decrease and Conquer* kita tidak memproses semua sub-persoalan, hanya salah satu saja tiap kita membagi persoalannya). Karena itu tidak ada tahap *Combine* dalam

Decrease and Conquer. ada beberaa varian dari *Decrease and Conquer* ini. diantaranya

1. **Decrease by a constant**

   Pada varian ini ukuran pemecahan persoalan dilakukan sebesar konstanta yang sama setiap iterasi algoritmanya. Biasanya konstanta = 1.

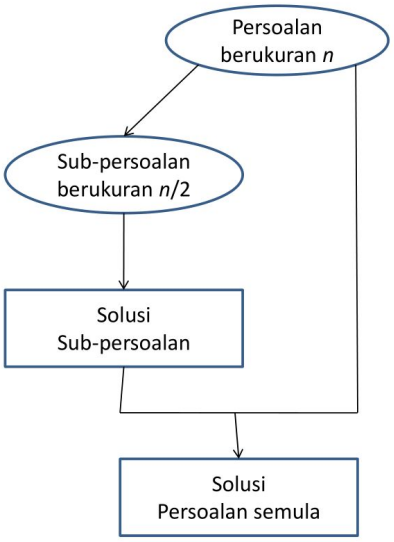
2. **Decrease by a constant factor**

   Ukuran persoalan dipecah sebesar faktor konstanta yang sama setiap iterasi algoritma. Biasanya faktor konstanta = 2 (binary).

3. **Decrease by a variable size**

   Untuk tiap pemecahan persoalan menjadi sub-persoalan, ukuran yang dipecah-pecah berbeda-beda tergantung iterasinya. [1]

Lalu untuk analisis kompleksitas, decrease by a constant akan memiliki kompleksitas yang sama dengan brute force. sementara untuk decrease by a variable size kompleksitasnya tergantung dari ukuran pemecahan pada tiap iterasinya. lalu untuk decrease by a constant factor kompleksitas waktunya adalah logaritmik sesuai dengan faktor konstantanya. Sehingga umumnya yang dipakai untuk menyelesaikan masalah adalah decrease by a constant factor, dimana faktor konstanta yang digunakan biasanya bernilai 2.



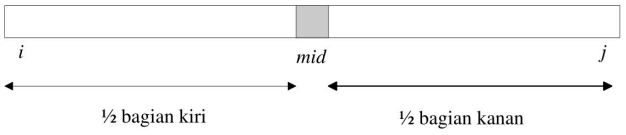
Gambar 2.1 Ilustrasi Decrease by a constant factor [2]

Gambar diatas menunjukkan cara kerja *Decrease and Conquer* pada constant factor = 2. misalkan awalnya kita punya persoalan yang berukuran *n*. lalu persoalan ini memenuhi karakteristik tertentu sehingga dapat di pecah menjadi dua persoalan. Maka setelah dipecah itu dapat diverifikasi bahwa solusi yang kita inginkan hanya mungkin berada pada salah satu sub-persoalan saja. maka kita dapat membuang sub-persoalan yang lain itu dan memproses sub-persoalan yang mungkin mengandung solusi yang kita cari. proses ini dilakukan terus menerus sampai kita mencapai solusi yang diinginkan. pada awalnya ukuran yang diproses *n*, lalu *n / 2*,

dan seterusnya sehingga kompleksitas waktunya adalah *O(log n)*.

III. Binary Search

Binary Search adalah salah satu algoritma penyelesaian masalah yang memanfaatkan prinsip *Decrease and Conquer*. contoh yang sering diberikan untuk binary search adalah mencari suatu elemen pada suatu *container* atau *array*. Konsepnya sama seperti mencari nama pada daftar nama telepon. Karena terurut, kita cukup melihat tengahnya saja, lalu bandingkan dengan yang kita cari, apakah berada sebelum tengah atau sesudah tengah. Lalu lanjut ke sub-persoalannya. contoh nyata mencari elemen pada *array* terurut:



Gambar 2.2 Ilustrasi Binary Search [3]

misalkan A = [1,4, 16, 19, 26] lalu kita ingin mencari apakah ada 1 didalam array tersebut. Maka pertama kali kita cek elemen tengah dari *array* tersebut yaitu 16. ternyata 1 < 16. maka kita buang sub-persoalan yang berada diatas 16 karena tidak mungkin 1 berada disana. Lalu sub-persoalan menjadi [1,4,16]. Cek elemen tengah lagi yaitu 4, ternyata 1 < 4. lalu sub-persoalan menjadi [1]. Maka 1 dapat ditemukan, iterasi selesai.

Apakah binary search hanya dapat dilakukan pada *array* atau larik saja? tentu tidak. Sekarang akan dijelaskan hubungan dari fungsi monotonik dengan binary search. Tanpa menghilangkan generalitas, asumsikan fungsi monotonik yang dimaksud adalah fungsi monotonik yang naik (*increasing)*. maka berlaku

$$x \leq y \Rightarrow f(x) \leq f(y)$$

misalkan juga kita punya sebuah kondisi yang harus dipenuhi nilai fungsi tersebut, dan kondisinya berupa pertidaksamaan. misalkan kondisinya adalah

$$f(a) \geq K$$

maka dengan memanfaatkan fakta bahwa fungsi monotonik, maka jelas kondisi diatas juga akan selalu benar mengikuti perkembangan fungsinya. jika pada saat ini nilai kondisi benar, maka jika kita bergerak ke arah sumbu *x* positif maka kondisi diatas juga akan tetap benar. Inilah yang membuat fungsi monotonik sangat berguna untuk masalah seperti ini. Mengapa demikian? jika nilai diatas hanya bisa dipenuhi untuk *range* nilai tertentu maka kita dapat mengaplikasikan binary search untuk mempercepat proses pencarian solusinya.

karena kita dapat membuang bagian yang tidak perlu diproses. Dan hal ini lebih berguna lagi jika kita mencari titik-titik ekstrim dalam suatu fungsi dengan kondisi-kondisi tadi.

sekarang kita coba menggunakan metode ini untuk mencari akar kuadrat dari suatu bilangan yang sudah di bulatkan. misalkan kita ingin mencari akar dari 10, dalam hal ini jawabannya adalah 3. namun bagaimana cara mencarinya secara eksak dengan binary search secara formal? berikut caranya.

definisikan fungsi $f(x) = x^2$ dengan domain bilangan cacah, maka jelas bahwa fungsi ini monotonik naik. sehingga kita bisa menggunakan binary search untuk mencari nilai ekstrim pada kondisi tertentu. Lalu apa kondisinya?

$f(x) \leq 10$

untuk range pencarian kita tetapkan antara 1 sampai 10. lalu kita coba telusuri solusinya, pertama cek tengahnya yaitu

$(1 + 10) / 2 = 5$

ternyata f(5) tidak memenuhi kondisi diatas, maka tidak mungkin nilai diatas 5 memenuhi kondisi diatas, sehingga kita harus menuju subpersoalan dibawah 5. diantara 1 dan 5, cari nilai tengahnya

$(1 + 5) / 2 = 3$

ternyata f(3) memenuhi kondisi, maka kita telah menemukan nilainya. dan benar bahwa akar dari 10 dibulatkan adalah 3. sekarang, mengapa kita menggunakan cara ini? bukankah bahasa pemrograman kebanyakan sudah memiliki implementasi fungsi akarnya masing-masing. Memang benar, tetapi dalam pembulatannya akan terjadi rounding error sehingga nilai yang didapat cukup sering tidak sesuai dengan apa yang diinginkan. jika kita bekerja pada bilangan bulat tentu saja metode binary search ini lebih terpercaya.

cara lain untuk memandang teknik ini adalah dengan memandang fungsi sebagai larik yang sangat panjang. dengan nilai-nilainya menunjukkan nilai array dan inputnya menunjukkan indeks dari larik.

A = [f(0), f(1), f(2), …]

dan karena fungsi monotonik (entah itu *increasing* atau *decreasing)*. kita bisa memakai binary search didalamnya. Mengubahnya tetap dalam bentuk fungsi memiliki beberapa manfaat tersendiri.

1. Menghemat penggunaan memori. andaikan ada permasalahan yang *domain* permasalahannya sampai lebih dari 1 juta. maka untuk membuat lariknya saja akan memakan banyak sekali memori.
2. Mendefinisikannya dalam fungsi membuat scope permasalahan lebih luas dan membuat persoalan yang tidak ada lariknya sama sekali menjadi jelas solusinya menggunakan binary search, istilahnya memperluas cakupan binary search itu sendiri.

IV. Aplikasi Binary Search pada Penentuan kapan virus tersebar pada suatu daerah

Beberapa bulan yang lalu, kelas Strategi Algoritma memberikan tugas untuk membuat program simulasi penyebaran virus dengan menggunakan BFS. Dan ada beberapa hal menarik yang dapat dianalisa dalam tugas ini, berikut beberapa hal kunci dalam pembuatan program tersebut.

1. Virus berasal dari suatu daerah dan dapat menyebarkannya ke daerah lain
2. populasi masyarakat di suatu daerah didefinisikan dengan konstan P(A)
3. waktu pertama kali virus menginfeksi suatu kota didefinisikan dengan T(A) dan total hari sejak infeksi pertama muncul di daerah tersebut didefinisikan sebagai t(A).
4. Populasi masyarakat yang terkena virus didefinisikan dengan fungsi I(A, t(A)). yang merupakan fungsi logistik

$$I(A, t(A)) = \frac{P(A)}{1 + (P(A) - 1)e^{-\gamma t(A)}}, \gamma = 0.25$$

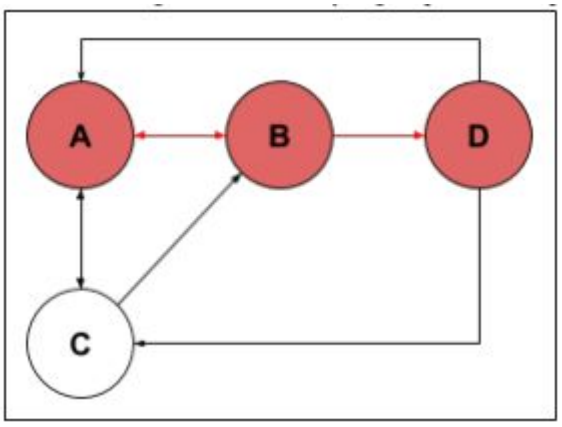
5. setiap jalan yang menghubungkan satu kota ke kota lainnya memiliki nilai peluang, dimana nilai peluang tersebut menunjukkan kemungkinan terjadinya perjalanan dari daerah satu ke daerah lainnya. misalkan daerah A dan daerah B terhubung. maka hubungan tersebut memiliki nilai konstanta Tr(A, B). dengan A sebagai kota asal dan B adalah kota tujuan. konstanta tersebut adalah peluang yang dimaksud.
6. Fungsi penyebaran virus dari suatu daerah A ke daerah B dirumuskan sebagai S(A, B). dengan syarat jika S(A, B) > 1 maka daerah A berhasil menularkan virus ke daerah B.

   $S(A, B) = I(P(A), t(A)) \times Tr(A, B)$

7. Jika virus dari daerah awal berhasil tersebar ke daerah tujuan, maka daerah tujuan akan di cek juga apakah bisa menyebarkannya ke daerah lain.

Pada awalnya virus hanya berada di satu kota saja, dan perkembangannya dilihat sebagai fungsi waktu. program akan menerima input text berupa informasi tentang graph keterhubungan kota-kota tersebut dan input T sebagai waktu yang di uji. sudah sampai kemana saja kah virus tersebut menyebar.

Pengerjaan tugas ini memanfaatkan algoritma BFS dengan beberapa modifikasi. jika pada BFS biasa tiap child dari node akan selalu di push ke queuenya, disini yang akan di push hanya child yang berpotensi menyebarkan virus tersebut ke daerah lain. setelah menjalankan algoritma BFS tersebut harus ditampilkan daerah mana saja yang sudah terinfeksi oleh virus dari daerah awal. daerah yang sudah terinfeksi digambarkan dengan warna merah. berikut contohnya.



Gambar 2.3 Ilustrasi Daerah yang terinfeksi [4]

Lalu bagian manakah yang dapat dioptimisasi menggunakan binary search?. beberapa hal penting yang bisa dicari saat mencari permasalahan yang mungkin menggunakan binary search.

1. terdapat hal yang konstan atau monoton
2. terdapat kata-kata optimal / ekstrim seperti nilai minimum atau nilai maksimum
3. ada range bilangan tertentu

hal-hal ini dapat ditemui pada saat proses menentukan apakah pada waktu tertentu, suatu daerah berhasil menyebarkan virus dari daerahnya ke daerah tetangganya. yakni dengan fungsi logistik tersebut, dan disana diperlukan nilai t paling minimum yang membuatnya berhasil menyebarkan virus tersebut.

Agar mudah, kita buat definisinya dulu. fungsi logistiknya adalah sebagai berikut

$$I(A, t(A)) = \frac{P(A)}{1 + (P(A) - 1)e^{-\gamma t(A)}}, \gamma = 0.25$$

dan yang perlu dicek adalah S(A, B) > 1. dengan fungsi S.

$$S(A, B) = I(A, t) \times tr(A, B)$$

namun fungsi masih dua dimensi, kita tidak bisa melakukan binary search pada kedua variabel. Apabila dilihat lagi maka nilai dari variabel A dan B sebenarnya tidak penting, karena saat perhitungan yang akan berubah hanyalah t yang diuji terhadap fungsi. sehingga A dan B bisa dianggap konstan. secara sederhana fungsi S menjadi hanya fungsi t, dengan bentuk sebagai berikut

$$S(t) = \frac{1}{1 + e^{-t}}$$

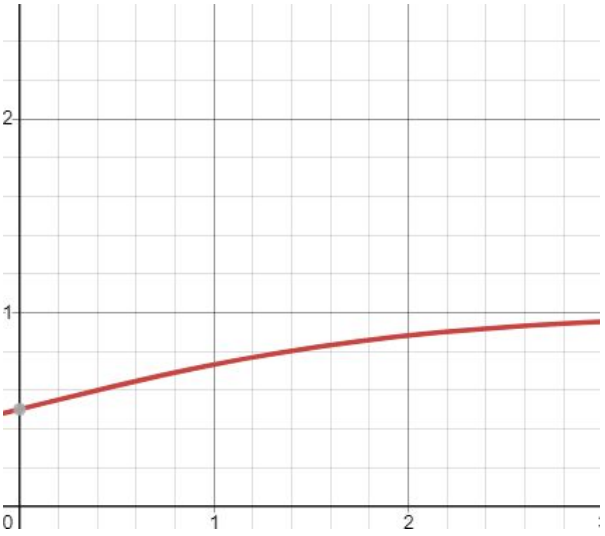
perhatikan bahwa ini bukanlah fungsi dari spesifikasi tugas diatas, namun bersifat sama yaitu fungsi logistik dengan beberapa konstanta yang diubah.

lalu apakah fungsi diatas memenuhi syarat bahwa fungsinya harus monoton?. Ya dan dapat dibuktikan dengan uji turunan.

dengan syarat bahwa turunan dari fungsi S selalu positif atau selalu negatif dapat diambil kesimpulan bahwa fungsi S monoton naik atau turun.

$$S'(t) = \frac{e^{-t}}{(1 + e^{-t})^2} > 0$$

maka terbukti bahwa fungsi S monoton naik. dapat juga dilihat pada grafik dibawah.



Gambar 2.4 Ilustrasi fungsi monoton [5]

Sekarang, bagaimana dengan rangenya? kita dapat mencoba dari t = 0 sampai dengan t = T. dimana T adalah masukan dari pengguna. sehingga permasalahan ini dapat juga diubah menjadi, diberikan fungsi S sebagai berikut

$$S(t) = \frac{1}{1 + e^{-t}}$$

dengan t = 0 sampai dengan t = T. carilah nilai t (bulat) terkecil sedemikian hingga S(t) > 0.7 (asumsikan kondisinya juga baru). Dengan bruteforce akan menghasilkan solusi yang berjalan pada kompleksitas waktu *O(T)*. Namun, ada cara matematis lain untuk menyelesaikan masalah ini. kita bisa mencoba untuk mencari fungsi invers atau kebalikan dari fungsi S. Dan ini dapat dicoba juga karena fungsi invers dari S adalah

$$S^{-1}(t) = ln(\frac{1}{t} - 1)$$

Namun kembali lagi ke argumen awal, untuk fungsi-fungsi semacam ini, dapat terjadi *rounding error* pada bilangan bulat dan karena memang domainnya bilangan bulat. lebih baik mengorbankan kompleksitas waktu demi kebenaran program. secara general, proses pencarian nilainya adalah seperti ini

sekarang range berada di [0, T]. cek titik tengahnya yaitu T / 2. lalu cek apakah S(t) > 0.7 jika ya, maka periksalah sub-persoalan yang berada dibawah T / 2. Mengapa? karena fungsi monoton, sehingga yang berada diatas T / 2 pasti akan memenuhi S(t) > 0.7 juga. Sebaliknya jika saat t = T / 2 dan S(t) tidak lebih dari 0.7. Maka tidak mungkin jawabannya berada di subpersoalan yang dibawah T / 2. Sehingga yang harus di cek adalah subpersoalan diatas T / 2. Perkecil rangenya dan lakukan proses yang sama berkali-kali sampai iterasinya berakhir. lalu akan didapatkan lah nilainya.

V. Implementasi Program Binary Search

Untuk mendemonstrasikan solusi dari permasalahan diatas, sudah disiapkan console program sederhana dengan implementasi binary search dengan fungsi yang dimaksud. implementasi fungsinya sebagai berikut

```
def S(t):
    e = 2.71828 #aproksimasi
    a = e ** (-1 * t)
    b = 1 / (1 + a)
    return b
```

lalu untuk program binary searchnya sebagai berikut.

```
T = int(input())

l = 0
r = T
answer = -1

while(l <= r):
    mid = (l + r) // 2

    if(S(mid) > 0.7):
        r = mid - 1
        answer = mid
    else:
        l = mid + 1

if(answer == -1):
    print("tidak ada")
else:
    print(answer)
```

Program diatas akan menghasilkan jawaban 1. tentu dapat juga dimodifikasi agar lebih sesuai dengan spesifikasi tugas BFS tersebut.

```
def S(t, A, tr):
    e = 2.71828 #aproksimasi
    a = e ** (-0.25 * t)
    a = a * (A - 1)
    b = A / (1 + a)
    return b
```

dan pada program utama diinputkan juga nilai A dan tr, lalu di kondisi pada loop binary searchnya diganti menjadi

```
if(S(mid, A, tr) > 1):
```

seperti pada kondisi awal.

VI. Kesimpulan

Banyak cara untuk menyelesaikan sebuah permasalahan. Terkadang solusi sederhana yang didapat dari fakta matematis tidak bisa dikomputasi secara sederhana dan tanpa *rounding error* apabila domain permasalahannya adalah bilangan bulat. Contoh yang lain adalah masalah perhitungan bilangan Fibonacci. Formula dari barisan Fibonacci adalah

$$F(n) = F(n-1) + F(n-2)$$

dan secara algoritmik dapat dikerjakan dalam waktu linear terhadap *n*. Lalu secara matematis bisa didapatkan formula untuk langsung mendapatkan nilai bilangan Fibonacci ke - *n* yaitu dengan formula

$$F(n) = \frac{\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n}{\sqrt{5}}$$

Dan lagi melibatkan bilangan akar dan dapat membuat *rounding error* sehingga dicarilah metode untuk mencari bilangan Fibonacci tanpa perlu bilangan irasional, salah satunya dengan cara Matrix Exponentiation.

Permasalahan binary search ini juga sama, secara matematis terlihat bahwa ada solusi mudah yang dapat dilakukan seperti mencari fungsi invers dan sebagainya. Namun, secara komputasi ada hal-hal yang perlu diperhatikan seperti penggunaan *double*, *rounding error,* dan sebagainya yang membuat penggunaan binary search dalam masalah ini sangat berguna.

Video Link at Youtube

Berikut video penjelasan tentang makalah ini di youtube.

https://www.youtube.com/watch?v=PEEsOd6P1lw

Ucapan Terima Kasih

Saya mengucapkan terima kasih sebesar-besarnya kepada Allah S.W.T yang telah memberikan rahmatnya agar saya bisa menyelesaikan makalah ini. Saya juga ingin berterima kasih kepada dosen Dr. Nur Ulfa, M.Sc. yang telah membimbing saya dengan materi strategi algoritma di kelas. terima kasih juga kepada teman-teman saya yang selalu mendukung saya dan memberikan motivasi untuk mengerjakan makalah ini.

Referensi

[1] Rinaldi Munir, Decrease and Conquer (2020), Bahan Kuliah IF2211, pp.2-4.

[2] Rinaldi Munir, Decrease and Conquer (2020), Bahan Kuliah IF2211, pp.18.

[3] Rinaldi Munir, Decrease and Conquer (2020), Bahan Kuliah IF2211, pp.19..

[4] Tugas Besar II IF2211 Strategi Algoritma, Simulasi Penyebaran Virus Penyakit dengan Memanfaatkan Algoritma BFS untuk Penelusuran pada Graf, pp.5.

[5] https://www.desmos.com/calculator, diakses pada 2 mei 2020.

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, 3 Mei 2020

Jun Ho Choi Hedyatmo 13518044