Perbandingan Performansi Pendekatan *Top-Down* dan *Bottom-Up* pada Pemrograman Dinamis

Hani'ah Wafa - 13516053
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13516053@std.stei.itb.ac.id

Abstract*—Makalah ini menjelaskan tentang strategi algoritma secara umum dan *dynamic programming* (pemrograman dinamis) sebagai salah satu strategi algoritma yang terkenal dan banyak digunakan di dalam dunia ilmu komputer. Pengimplementasian pemrograman dinamis ini tidak terlepas dari metode penyimpanan hasil suatu sub masalah yang sudah pernah dipanggil atau diselesaikan dengan tujuan untuk mematiskan bahwa setiap sub masalah hanya akan dipanggil satu kali. Sehingga akan menghemat waktu komputasi. Makalah ini akan lebih fokus dalam membahas perbandingan performansi dari dua pendekatan yang digunakan pada pengimplementasian algoritma pemrograman dinamis yaitu, pendekatan secara *top-down* atau juga disebut sebagai *memoization* dan pendekatan secara *bottom-up* atau yang juga disebut sebagai *tabulation.

Keywords—strategi algoritma; pemrograman dinamis; memoization; tabulation.

I. PENDAHULUAN

Algoritma adalah urutan langkah yang didefinisikan dngan baik untuk menyelesaikan masalah yang didefiniskan dengan baik pula dalam waktu tertentu. Sebuah Algoritma harus dapat menyelesaikan semua kemungkinan kejadian pada permasalahan tersebut dengan benar. Sejak algortima mulai lebih sering digunakan dalam komputasi modern, ada sebuah bidang baru yang berkembang berkaitan dengan desain, analisa dan perbaikan dari suatu algoritma. Bidang strategi algoritma atau juga disebut desain algoritma membutuhkan latar belakang kemampuan matematis yang kuat juga gelar ilmu komputer yang kemudian menjadi kualifikasi yang banyak dicari. Bidang ini menawarkan banyak pilihan karir mengingat kebutuhan terhadap algoritma yang secara berkelanjutan terus meningkat.

Desain algoritma yang merupakan pendekatan secara umum untuk mendapatkan suatu bentuk dari penyelesaian efisien dari sebuah masalah adalah metode yang menarik, alasannya adalah sebagai berikut.

- Memberikan template yang cocok untuk menyelesaikan bermacam-macam masalah.
- Dapat ditranslasikan ke struktur data yang disediakan oleh kebanyakan high-level languages.
- Kebutuhan temporal dan spasial dari algoritma dapat dianalisa secara tepat.

Ada banyak strategi algoritma yang terkenal di dunia ilmu komputer, seperti bruteforce, greedy, divide and conquer, decrease and conquer, backtracking, branch and bound, dynamic programming, dan lain-lain. Setiap jenis strategi algoritma ini tentu meiliki kelebihan dan kekurangannya masing-masing. Dalam makalah ini akan dibahas tentang pemrograman dinamis dan pendekatan-pendekatan yang dipakai dalam pemrograman dinamis.

II. PEMROGRAMAN DINAMIS

Pemrograman Dinamis (*Dynamic Programming*) atau terkadang disebut dengan *dynamic optimization* dicipakan pada tahun 1940 oleh Richard Bellman, seorang ilmuwan komputer yang pada saat itu bekerja sebagai peneliti matematis di sebuah perusahaan yang bernama RAND. Pemrograman Dinamis adalah sebuah teknik atau starategi algoritma yang biasanya berdasar pada perumusan yang berulang dan sebuah (atau lebih) starting state. Sebuah solusi pada suatu tahap dibangun dari solusi pada taha sebelumnya. Dengan kata lain, pemrograman dinamis adalah suatu pendekatan untuk menyelesaikan masalah yang kompleks dengan membagi-bagi masalah tersebut menjadi bagian yang lebih kecil. Solusi pemrograman dinamis memiliki kompleksitas polinomial yang memastikan bahwa solusi tersebut memiliki waktu eksekusi yang lebih cepat dibanding teknik algoritma lain seperti algoritma runut-balik, brute force, dan lain-lain.

Keunggulan pemrograman dinamis adalah algoritma ini membuat kita dapat menghindari pengulangan suatu perhitungan dengan cara menyimpan hasil dari sub permasalahan yang sudah pernah kita selesaikan tersebut. Sehingga setiap sub permasalahan hanya akan dieksekusi

satu kali. Atau dapat dikatakan bahwa pemrograman dinamis ini mengorbankan ruang untuk menghemat waktu eksekusi. Metode yang digunakan oleh pemrograman dinamis dalam menyimpan hasil dari sub masalah yang telah diselesaikan tersebut ada dua bentuk yaitu *memoization* dan *tabulation*.

A. TABULATION

Tabulation adalah pendekatan secara *bottom-up*. Metode ini dimulai dengan menyelesaikan sub masalah pada level yang paling bawah. Solusi yang dihasilkan kemudian akan membuat kita dapat menyelesaikan sub masalah yang selanjutnya dan begitu seterusnya. Dalam metode ini kita akan menyelesaikan setiap sub masalah yang ada secara iterative hingga semua sub masalah telah kita selesaikan dan akhirnya kita mendapatkan penyelesaian dari masalah utamanya. Dengan penggunaan tabulasi ini kita dapat menghemat waktu pada saat membutuhkan jawaban suatu sub masalah yang sebelumnya sudah pernah diselesaikan karena kita menyimpan jawaban tersebut dalam bentuk tabulasi.

B. MEMOIZATION

Berbeda dengan tabulation, memorization adalah pendekatan secara top-down. Metode ini memulai dengan sub masalah pada level tertinggi (sub masalah yang paling dekat dengan masalah utamanya) dan kemudian secara rekursif akan memanggil sub masalah selanjutnya di level yang lebih bawah dan begitu seterusnya. Dengan penggunaan memorization ini kita dapat menghemat waktu ketika sub masalah A secara rekursif memanggil sub masalah B yang sebelumnya sudah pernah diselesaikan. Karena sub masalah B dan semua sub masalah di bawahnya telah di memoisasi maka kita tidak perlu mengulangi semua pohon rekursi yang dibangkitkan oleh sub masalah B dan dapat menghemat banyak komputasi.

III. PERFORMANSI PENDEKATAN *TOP-DOWN* DAN *BOTTOM-UP*

Setelah dibahas mengenai pemrograman dinamis dan pendekatan-pendekatan yang digunakan pada pemrograman dinamis untuk meningkatkan performansinya, lalu bagaimanakah perbedaan performansi dari pendekatan-pendekatan tersebut?

Untuk dapat membandingkannya tidak hanya secara teoritis, tapi dengan lebih nyata, dalam makalah ini akan diperlihatkan perbandingan performansi pendekatan *top-down (memoization)* dan *bottom-up (tabulation)* dalam menyelesaikan dua buah permasalahan.

A. Boredom

Permasalahan pertama ini merupakan salah satu soal kontes codeforces round 260. Berikut adalah spesifikasi dari soal tersebut.

time limit per test : 1 second
memory limit per test : 256 megabytes
input : standard input
output : standard output

Alex doesn't like boredom. That's why whenever he gets bored, he comes up with games. One long winter evening he came up with a game and decided to play it.

Given a sequence *a* consisting of *n* integers. The player can make several steps. In a single step he can choose an element of the sequence (let's denote it a_k) and delete it, at that all elements equal to $a_k + 1$ and $a_k - 1$ also must be deleted from the sequence. That step brings a_k points to the player.

Alex is a perfectionist, so he decided to get as many points as possible. Help him.

Input

The first line contains integer n ($1 \le n \le 10^5$) that shows how many numbers are in Alex's sequence.

The second line contains n integers $a_1, a_2, ..., a_n$ ($1 \le a_i \le 10^5$).

Output

Print a single integer — the maximum number of points that Alex can earn.

Secara singkat, ide penyelesaian permasalahan ini adalah sebagai berikut.

- Simpan banyaknya nilai kemunculan suatu angka pada sebuah larik a dimana nilai a[k] akan menunjukkan berapa kali k muncul (atau langsung menjumlahkannya dimana nilai a[k] menunjukkan hasil perkalian antara k dengan jumlah kemunculan k).
- Dengan catatan bahwa kita menggunakan cara kedua untuk melakukan penyimpanan banyaknya kemunculan suatu angka. Kemudian secara rekursif (*top-down*) maupun iteratif (*bottom-up*) kita perlu melakukan perhitungan sebuah fungsi f, dimana nilai *f*(n)=max(*f*(n - 1), *f*(n - 2) + a[k]) dan solusi global dari permasalahan kita adalah nilai f(a_n) dimana a_n adalah nilai terbesar dari angka yang pernah muncul.

1) Pendekatan Rekursif Tanpa Memoization

Jika kita ingin menyelesaikan permasalahan diatas menggunakan pendekatan secara rekursif, tapi tanpa melakukan pencatatan terhadap hasil dari suatu sub masalah yang sudah pernah dipanggil sebelumnya,

maka salah satu caramengimplementasikannya adalah sebagai berikut.

Dengan asumsi bahwa kita telah memiliki larik a untuk menyimpan jumlah kemunculan suatu angka. Maka setelah itu kita perlu membuat fungsi rekursif yang akan menghitung nilai *f*(n), dengan spesifikasi sebagai berikut.

- Basis
  - *f*(0) = a[0]
  - *f*(1) = a[1]
- Rekurens

  f(n) = max(*f*(n - 1), *f*(n - 2) + a[k])

HASIL

Berikut adalah hasil secara umum dan juga beberapa kasus tes yang akan didapatkan jika kita melakukan submisi menggunakan pendekatan rekursif tanpa memoisasi yang diimplementasikan dalam bahasa c++.

Problem	Lang	Verdict	Time	Memory
455A - 35	GNU C++14	Time limit exceeded on test 11	1000 ms	4280 KB

1

Time: 15 ms, **memory:** 4052 KB

Verdict: OK

Input

```
2
1 2
```

11

Time: 1000 ms, **memory:** 4280 KB

Verdict: TIME_LIMIT_EXCEEDED

Input

```
100000
5489 5929 7152 8443 6028 8580 5449 8473 423
```

Gambar 3.1 Performansi pendekatan rekursif biasa pada persoalan *boredom.*

2) Pendekatan *Top-Down*

Dengan penggunaan pendekatan *top-down*, kita akan membutuhkan dua buah larik. Larik pertama akan digunakan untuk menyimpan jumlah kemunculan tiap bilangan seperti yang telah dijelaskan sebelumnya. Larik kedua akan digunakan untuk proses memoisasi yaitu menyimpan hasil sub masalah yang sudah pernah dipanggil sebelumnya. Ada beberapa cara untuk mengimplementasikan kedua larik ini. Namun dalam makalah ini, kedua larik tersebut diimplementasikan sebagai suatu variabel global.

Dengan asumsi bahwa kita telah memiliki larik a untuk menyimpan jumlah kemunculan suatu angka dan larik solved untuk menyimpan hasil dari suatu sub masalah, dimana solved[0] telah diinisiasi dengan a[0] dan solved[1] diinisiasi dengan a[1]. Maka setelah itu kita perlu membuat fungsi rekursif yang akan menghitung nilai *f*(n), dengan spesifikasi sebagai berikut.

- Basis
  - *f*(0) = solved[0]
  - *f*(1) = solved[1]
- Rekurens

  Jika sub masalah belum pernah dikomputasi :

  solved[n] = max(*f*(n - 1), *f*(n - 2) + a[k])

  //sub masalah sudah pernah dikomputasi

  f(n)=solved[n]

HASIL

Berikut adalah hasil secara umum dan juga beberapa kasus tes yang akan didapatkan jika kita melakukan submisi menggunakan pendekatan *top-down* yang diimplementasikan dalam bahasa c++.

Problem	Lang	Verdict	Time	Memory
455A - 35	GNU C++14	Accepted	156 ms	7964 KB

1

Time: 15 ms, **memory:** 4832 KB

Verdict: OK

Input

```
2
1 2
```

47

Time: 155 ms, **memory:** 7960 KB

Verdict: OK

Input

```
99999
99999 99999 99999 99999 99999 99999 99999 9
```

Gambar 3.2 Performansi pendekatan *top-down* pada persoalan *boredom.*

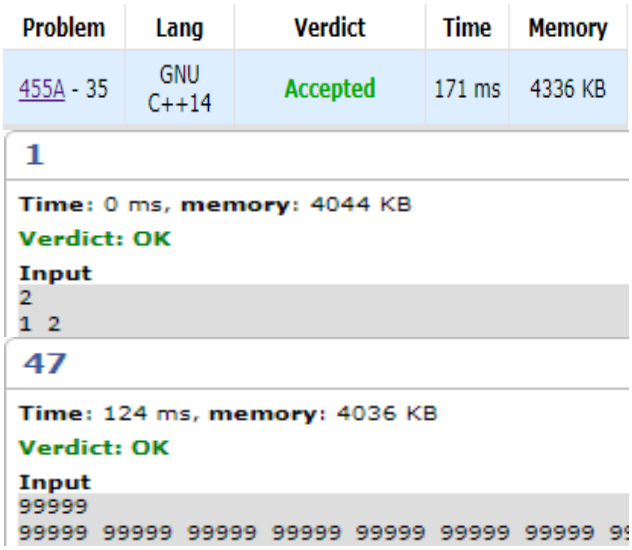
3) Pendekatan *Bottom-Up*

Dengan menggunakan pendekatan *bottom-up*, kita tidak memerlukan dua buah larik seperti pada pendekatan *top-down*. Pendekatan *bottom-up* dapat kita implementasikan hanya dengan satu buah larik, misalnya larik a yang pada awalnya digunakan untuk menyimpan nilai kemunculan dari suatu angka.

Setelah itu kita dapat melakukan iterasi untuk tiap nilai *i*, dimana $2 \leq i \leq a_n$, dengan a_n adalah nilai terbesar dari angka yang pernah muncul. Kemudian pada saat iterasi tersebut kita dapat memperbarui nilai a[k] dengan nilai max(a[k - 1], a[k - 2] + a[k]). Dengan pendekatan ini maka hasil global dari permasalahan akan kita dapatkan sebagai nilai dari a[a_n].

HASIL

Berikut adalah hasil secara umum dan juga beberapa kasus tes yang akan didapatkan jika kita melakukan submisi menggunakan pendekatan *bottom-up* yang diimplementasikan dalam bahasa c++.



Gambar 3.3 Performansi pendekatan *bottom-up* pada persoalan *boredom*.

B. Ilya and Queriese

Permasalahan yang kedua ini juga merupakan salah satu soal kontes codeforces pada round 186. Permasalahan yang kedua ini memiliki spesifikasi sebagai berikut.

time limit per test : 2 seconds
memory limit per test : 256 megabytes
input : standard input
output : standard output

Ilya the Lion wants to help all his friends with passing exams. They need to solve the following problem to pass the IT exam.

You've got string $s = s_1 s_2 ... s_n$ (n is the length of the string), consisting only of characters "." and "#" and m queries. Each query is described by a pair of integers l_i, r_i ($1 \leq l_i < r_i \leq n$). The answer to the query l_i, r_i is the number of such integers i ($l_i \leq i < r_i$), that $s_i = s_{i+1}$.

Ilya the Lion wants to help his friends but is there anyone to help him? Help Ilya, solve the problem.

Input

The first line contains string s of length n ($2 \leq n \leq 10^5$). It is guaranteed that the given string only consists of characters "." and "#".

The next line contains integer m ($1 \leq m \leq 10^5$) — the number of queries. Each of the next m lines contains the description of the corresponding query. The i-th line contains integers l_i, r_i ($1 \leq l_i < r_i \leq n$).

Output

Print m integers — the answers to the queries in the order in which they are given in the input.

Untuk menyelesaikan permasalahan yang kedua ini, dapat menggunakan strategi berikut.

- Untuk nilai k dimana $1 \leq k \leq n$, kita akan menghitung nilai f(k) yang menggambarkan jumlah bilangan i yang memenuhi kondisi $1 \leq i \leq k$ dan $s_i = s_{i+1}$, dimana f(k) = f(k - 1) + 1 jika $s_k = s_{k+1}$ atau f(k) = f(k - 1) jika $s_k \neq s_{k+1}$.
- Untuk tiap nilai l dan r yang akan diinputkan selanjutnya, solusinya akan kita dapatkan sebagai nilai dari $f(r - 1) - f(l - 1)$.

1) Pendekatan *Top-Down*

Dengan penggunaan pendekatan *top-down*, salah satunya dapat diimplementasikan dengan mendeklarasikan larik a yang berperan untuk menyimpan hasil komputasi dari suatu sub masalah sebagai variabel global.

Kemudian kita perlu membuat sebuah fungsi rekursif f(k) yang akan menggambarkan banyaknya bilangan i yang memenuhi dimana $1 \leq i \leq k$, dengan spesifikasi sebagai berikut.

- Basis

  //a[0] telah diinisiasi dengan 0

  $f(0) = a[0]$

- Rekurens

  Jika sub masalah belum pernah dikomputasi :

  Jika $s_k = s_{k+1}$:

  a[k] = 1 + f(k - 1)

  Jika tidak :

  a[k] = f(k - 1)

  //sub masalah sudah pernah dikomputasi

  f(k) = a[k]

Selanjutnya, untuk setiap nilai l dan r yang akan diinputkan maka kita akan dengan mudah mendapatkan solusi permasalahannya dalam bentuk nilai $f(r - 1) - f(l - 1)$.

HASIL

Berikut adalah hasil secara umum dan juga beberapa kasus tes yang akan didapatkan jika kita melakukan submisi menggunakan pendekatan *top-down* yang diimplementasikan dalam bahasa c++.

Problem	Lang	Verdict	Time	Memory
313B - 24	GNU C++14	Memory limit exceeded on test 10	498 ms	262144 KB

1

Time: 30 ms, **memory:** 3656 KB

Verdict: OK

Input

```
......
4
```

10

Time: 498 ms, **memory:** 262144 KB

Verdict: MEMORY_LIMIT_EXCEEDED

Input

```
.#.#.#.#.#.#.#.#.#.#.#.#.#.#.#.#.#.#.#.#.#.#.#.#.#.#.#.
```

Gambar 3.4 Performansi pendekatan *top-down* pada persoalan *Ilya and Queries.*

2) Pendekatan *Bottom-Up*

Dalam penggunaan pendekatan *bottom-up*, kita juga akan membutuhkan sebuah larik a. Pada larik a ini, nilai a[0] akan diinisialisasi dengan 0 dan a[1] akan diinisialisasi dengan nilai 1 jika $s_1 = s_2$ dan sebaliknya akan diinisialisasi dengan nilai 0 jika $s_1 \neq s_2$.

Kemudian secara iteratif kita akan melakukan komputasi untuk setiap nilai k dimana $2 \leq k < n$-1. Dalam iterasi tersebut kita akan memperbarui nilai larik a, sehingga nilai a[k] akan menggambarkan banyaknya nilai i yang memenuhi kebutuhan di soal dengan $1 \leq i \leq k$. Nilai a[k] akan diperbarui dengan nilai a[k - 1] + 1 jika $s_k = s_{k+1}$ dan sebaliknya akan diperbarui dengan nilai a[k - 1] jika $s_k \neq s_{k+1}$.

Selanjutnya, untuk setiap nilai l dan r yang akan diinputkan maka kita akan dengan mudah mendapatkan solusi permasalahannya dalam bentuk nilai $a(r - 1) - a(l - 1)$.

HASIL

Berikut adalah hasil secara umum dan juga beberapa kasus tes yang akan didapatkan jika kita melakukan submisi menggunakan pendekatan *top-down* yang diimplementasikan dalam bahasa c++.

Problem	Lang	Verdict	Time	Memory
313B - 24	GNU C++14	Accepted	1028 ms	3920 KB

1

Time: 30 ms, **memory:** 3660 KB

Verdict: OK

Input

```
......
4
```

10

Time: 934 ms, **memory:** 3916 KB

Verdict: OK

Input

```
.#.#.#.#.#.#.#.#.#.#.#.#.#.#.#.#.#.#.#.#.#.#.#.#.#.#.#.
```

Gambar 3.5 Performansi pendekatan *bottom-up* pada persoalan *Ilya and Queries.*

IV. PERBANDINGAN PERFORMANSI

A. *Boredom*

Pada penyelesaian permasalahan yang pertama ini, dapat kita lihat bahwa penggunaan pendekatan rekursif tanpa memoisasi ternyata membutuhkan waktu eksekusi yang lama, dibuktika dengan status *time limit exceeded* yang didapat oleh submisi ini. Hal ini disebabkan karena untuk tiap nilai n dimana $n > 1$, untuk mendapatkan nilai $f(n)$ maka kita perlu memanggil pengeksekusian $f(n - 1)$ dan $f(n - 2)$. Sementara itu, saat melakukan pemanggilan pengeksekusian $f(n - 1)$ kita akan memanggil pengeksekusian $f(n - 2)$ dan $f(n - 3)$. Dari contoh tersebut, kita mengetahui bahwa $f(n - 2)$ akan dieksekusi lebih dari satu kali. Untuk nilai n yang kecil hal ini tidak akan memberikan banyak dampak. Namun untuk nilai n yang besar, maka suatu sub masalah bisa jadi akan dieksekusi berulang kali dalam jumlah yang cukup besar. Sehingga tentu saja akan membuat waktu yang dibutuhkan untuk melakukan komputasi menjadi sangat lama.

Dengan penggunaan memoisasi atau tabulasi kita akan menyimpan nilai dari sub masalah yang sudah pernah kita panggil atau selesaikan. Sehingga kita dapat menghindari pemanggilan berulang *f*(*n* - 2) pada contoh di atas. Penggunaan memoisasi atau tabulasi ini akan membuat kita memerlukan lebih banyak memori sebagai ganti untuk penghematan waktu pada saat komputasi. Pada kasus-kasus dengan nilai input yang besar akan terlihat perbedaan dimana pendekatan menggunakan memoisasi atau tabulasi ini akan lebih banyak memakan memori, tapi lebih sedikit membutuhkan waktu eksekusi.

Karena pendekatan *top-down* dan *bottom-up* keduanya melakukan perulangan dengan jumlah yang sama, yaitu dari 2 hingga *n*, maka perbandingan performasi antara pendekatan *top-down* dan *bottom-up* pada persoalan *boredom* ini dapat kita lihat pada penggunaan memorinya. Pada pendekatan *bottom-up* kita hanya membutuhkan satu buah larik yang digunakan untuk menyimpan nilai awal sekaligus nilai pada saat proses iterasi dilakukan. Namun pada pendekatan *top-down* kita membutuhkan satu buah larik yang lain untuk menyimpan nilai dari sub masalah yang telah kita selesaikan. Perbedaan penggunaan memori ini salah satunya dapat dilihat pada hasil test case terakhir di persoalan ini, yaitu test case ke-47.

B. Ilya and Queries

Seperti pada persoalan *boredom* sebelumnya, perbandingan performansi antara pendekatan *top-down* dan *bottom-up* pada persoalan *Ilya and Queries* ini juga terlihat pada perbedaan penggunaan memori yang digunakan oleh kedua pendekatan tersebut. Hal ini terlihat jelas dari status submisi yang menggunakan pendekatan *top-down.*

Jika dilihat dari jumlah perulangan yang dilakukan oleh masing-masing pendekatan untuk menyelesaikan persoalan ini hampir sama. Keduanya akan melakukan perulangan dari sub masalah terkecil hingga sub masalah yang terbesar.

V. KESIMPULAN

Sebuah laman menyebutkan bahwa kebanyakan *computer scientist* menggunakan pendekatan *bottom-up* untuk menyelesaikan sebuah permasalahan yang berkaitan dengan pemrograman dinamis. Alasannya adalah karena pendekatan ini lebih menghemat banyak memori dengan kompleksitas yang bisadibilang sama pada beberapa kasus dengan pendekatan *top-down.*

Di sisi lain, keunggulan pendekatan *bottom-up* ini akan terlihat pada kasus-kasus dimana kita tidak butuh mengetahui nilai penyelesaian dari semua sub masalah yang ada. Berbeda dengan pendekatan *bottom-up* yang akan mengiterasi semua sub masalah, dengan pendekatan *top-down* kita dapat hanya memanggil sub masalah yang memang kita butuhkan. Sehingga tentu saja kita akan menghemat banyak komputasi dan membuat waktu eksekusi yang dibutuhkan menjadi lebih cepat.

Hal lain yang perlu kita sadari adalah bahwa ada banyak faktor yang akan mempengaruhi performansi suatu algoritma. Mulai dari cara kita melakukan pengimplementasian, seperti struktur data yang kita gunakan, hingga faktor-faktor lain yang sudah tidak berhubungan lagi dengan strategi algoritma itu sendiri.

Ucapan Terimakasih

Penulisan makalah ini tentu saja tidak terlepas dari dukungan serta bimbingan berbagai pihak. Untuk itu, saya ingin menyampaikan ucapan terimakasih khususnya kepada dosen pengampu mata kuliah IF2211 Strategi Algoritma, Ibu Dr. Nur Ulfa Maulidevi ST,M.Sc., Bapak Dr.Ir. Rinaldi MT dan Ibu Dr. Masayu Leylia Khodra ST,MT.

Daftar Pustaka

[1] https://www.topcoder.com/community/data-science/data-science-tutorials/dynamic-programming-from-novice-to-advanced/
Diakses pada 6 Mei 2018

[2] https://medium.com/basecs/less-repetition-more-dynamic-programming-43d29830a630
Diakses pada 6 Mei 2018

[3] https://ab.inf.uni-tuebingen.de/teaching/ss08/gbi/script/chapter03-algointro.pdf
Diakses pada 11 Mei 2018

[4] https://www.computersciencedegreehub.com/faq/what-is-algorithm-design/
Diakses pada 11 Mei 2018

[5] http://cgi.csc.liv.ac.uk/~ped/teachadmin/algor/intro.html
Diakses pada 11 Mei 2018

[6] https://awjin.me/algos-js/dp/tab-memo.html
Diakses pada 11 Mei 2018

[7] http://codeforces.com/problemset/problem/313/B
Diakses pada 12 Mei 2018

[8] http://codeforces.com/problemset/problem/455/A
Diakses pada 12 Mei 2018

[9] http://codeforces.com/blog/entry/13336
Diakses pada 12 Mei 2018

[10] http://codeforces.com/blog/entry/7826
Diakses pada 12 Mei 2018

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

Hani'ah Wafa – 13516053