LAPORAN TUGAS BESAR STRATEGI ALGORITMA

Studi Kasus Mencari Jumlah Subarray Maksimum

Menggunakan Pendekatan *Brute Force* dan *Dynamic Programming*

Disusun untuk memenuhi tugas besar mata kuliah Strategi Algoritma

Disusun Oleh :
Hidayat Taufiqur Rahmah Achmad / 1301204300
Addin Amanatullah Suparjo / 1301204286

PROGRAM STUDI INFORMATIKA
FAKULTAS INFORMATIKA
UNIVERSITAS TELKOM
BANDUNG

2022

Abstrak

Tugas besar ini dilatarbelakangi karena adanya studi kasus tentang *"Mencari Jumlah Subarray Maksimum"*. Kami memilih menganalisis studi kasus tersebut dikarenakan permasalahan ini cukup terkenal. Selain itu, terdapat sebuah algoritma yang dapat menyelesaikan permasalahan ini dengan sangat efisien, dimana mungkin saat pertama kali melihat hal tersebut cukup susah untuk dibayangkan. Untuk mengatasi studi kasus tersebut, kami menggunakan teknik *Brute Force* dan *Dynamic Programming.* Hasil analisis kami bahwa pendekatan *Dynamic Programming* secara signifikan lebih efisien dibandingkan pendekatan *Brute Force*.

Kata kunci : analisis, *Brute Force*, *Dynamic Programming*, efisien.

BAB I
PENDAHULUAN

A. Latar Belakang

Seringkali ditemui masalah tentang memaksimumkan atau meminimumkan suatu fungsi objektif. Atau dengan kata lain kita sering menemui masalah optimasi. Beberapa contoh optimasi seperti optimasi memasak telur, memasak air hingga mendidih, mengatur penjadwalan dan lain-lain. Menyelesaikan masalah optimasi dapat dilakukan dengan berbagai macam strategi, diantaranya adalah algoritma *Brute Force, Dynamic Programming, Divide and Conquer*, dan lain-lain.

Pada tugas besar ini, kami mempunyai studi kasus yaitu "Mencari Jumlah Subarray Maksimum" dan kami menggunakan strategi algoritma *Brute Force* dan *Dynamic Programming* untuk mengatasi studi kasus tersebut. Strategi algoritma *Brute Force* sering digunakan karena sederhana dan mudah diimplementasikan serta hampir semua masalah dapat diselesaikan menggunakan algoritma *Brute Force*. Dan strategi algoritma *Dynamic Programming* memiliki sub masalah yang mampu diselesaikan secara dependent atau solusi dari satu sub masalah digunakan untuk mencari solusi dari sub masalah lainnya (*overlapping*).

Kami memilih studi kasus tersebut dikarenakan permasalahan ini cukup terkenal. disamping itu terdapat sebuah algoritma yang dapat menyelesaikan permasalahan ini dengan sangat efisien, dimana mungkin at first glance hal tersebut cukup susah untuk dibayangkan.

B. Rumusan Masalah

1. Bagaimana cara menyelesaikan studi kasus Maximum Subarray Sum tersebut?
2. Apa yang dimaksud dengan teknik Brute Force?
3. Bagaimana karakteristik teknik Brute Force?
4. Apa yang dimaksud dengan teknik Dynamic Programming?
5. Bagaimana karakteristik teknik Dynamic Programming?
6. Bagaimana cara kerja Algoritma Kadane?

C. Tujuan

1. Mengetahui cara menyelesaikan studi kasus Maximum Subarray Sum.
2. Mengetahui pengertian teknik *Brute Force.*
3. Mengetahui karakteristik teknik *Brute Force.*
4. Mengetahui teknik *Dynamic Programming.*
5. Mengetahui karakteristik teknik *Dynamic Programming.*
6. Mengetahui cara kerja *Algoritma Kadane.*

BAB II
DASAR TEORI

A. Penjelasan Kasus

Maximum subarray sum adalah permasalahan untuk mencari jumlah sub array yang berurutan maksimum dari sebuah array dengan panjang A[0…*n*]. Misalnya jika kita memiliki array [-10, 2, 3, 4, 5], maka jumlah maksimum subarray yang berurutannya adalah [-10, **2, 3, 4, 5**] dengan total 2+3+4+5 = 14. Permasalahan ini dapat diselesaikan dengan *approach* atau strategi yang berbeda, dan pada tugas ini kami akan menganalisis permasalahan *maximum subarray sum* dengan menggunakan strategi *Brute Force* dan *Dynamic Programming* (Kadane's algorithm).

B. *Brute Force*

Brute Force adalah sebuah algoritma pendekatan yang *straightforward* atau mudah untuk memecahkan suatu masalah dan biasanya didasarkan pada pernyataan masalah (*problem statement*) serta definisi konsep yang dilibatkan. Algoritma ini memecahkan masalah dengan sangat sederhana, langsung, dan jelas.

Brute Force dapat menghasilkan algoritma yang baku untuk berbagai tugas komputasi seperti penjumlahan/perkalian serta menentukan elemen minimum/maksimum dari suatu list. Namun, *Brute Force* jarang menghasilkan algoritma yang mangkus. Beberapa algoritma Brute Force sangat lambat sehingga tidak dapat diterima dan tidak sekreatif teknik pemecahan masalah lainnya.

C. *Dynamic Programming*

Kata "*Dynamic*" menggambarkan suatu metode pencarian solusi yang menggunakan tabel dinamis. Sedangkan kata "*Programming*" berarti perencanaan. *Dynamic Programming* atau biasa disingkat DP adalah pendekatan atau strategi yang memanfaatkan optimasi dari sebuah permasalahan.

Dynamic Programming akan memecah masalah menjadi beberapa sub masalah kemudian menyelesaikan sub masalah tersebut cukup sekali, sehingga ketika sub masalah yang sama muncul kita dapat memanggil hasil perhitungan sub masalah tersebut tanpa perlu melakukan perhitungan ulang.

D. Algoritma Kadane

Joseph "Jay" Born Kadane merupakan nama penemu algoritma Kadane. Algoritma Kadane sendiri adalah algoritma yang dapat menyelesaikan masalah *maximum subarray sum* hanya dengan menggunakan 1 buah loop. Algoritma Kadane bekerja dengan menyimpan hasil penjumlahan sub array terbaik pada iterasi tersebut ke dalam variabel lokal terbaik, kemudian membandingkannya dengan variabel global terbaik. Jika variabel lokal terbaik lebih besar, maka nilai variabel global terbaik akan di-*update*, sehingga setelah proses loop selesai, nilai yang disimpan oleh variabel global terbaik adalah jawaban akhirnya dan fungsi akan me-*return* nilai tersebut.

BAB III
IMPLEMENTASI

A. Brute Force

Dengan menggunakan metode *Brute Force,* program akan melakukan 3 *nested loop*. Idenya adalah *outermost loop* akan melakukan looping dari 0 sampai n - 1, kemudian di tiap iterasi, dilakukan *nested loop* dari *i* sampai *n* - 1, dan dari tiap iterasi loop kedua akan dilakukan perulangan (*innermost loop*) dari *i* sampai *j*. Tiap kali *innermost loop* dilakukan, operasi yang dilakukan adalah menjumlahkan total dari sub array indeks ke *i* sampai ke *j* pada iterasi tersebut sebagai lokal max, sehingga nilai lokal max tersebut dapat dibandingkan dengan global max di akhir perulangan loop kedua.

Metode *Brute Force:*

1. Menginisialisasi variabel maks_subarray_sum untuk menyimpan nilai global max penjumlahan sub array berurut.
2. Melakukan perulangan sebanyak *n* kali (*outermost loop*).
3. Melakukan perulangan sebanyak *n* - *i* - 1 kali.
4. Menginisialisasi variabel subarray_sum untuk menyimpan nilai lokal max penjumlahan sub array berurut.
5. Melakukan perulangan sebanyak *j* - *i* kali (*innermost loop)*.
6. Menjumlahkan semua nilai dalam array dari indeks ke *i* sampai indeks ke *j* dan menyimpannya ke dalam lokal max.
7. Membandingkan apakah nilai lokal max lebih besar daripada global max. Jika iya, maka nilai global max akan di-*update.*
8. Nilai global max di-*return.*

B. Dynamic Programming

Pada algoritma ini menggunakan metode *iterative Dynamic Programming,* yang artinya tidak ada proses rekursif yang terjadi. Algoritma ini sangat optimal untuk menyelesaikan permasalahan *maximum subarray sum* dikarenakan hanya membutuhkan single loop, sehingga *time complexity* nya sangat cepat. Idenya adalah dengan terus melakukan *tracking* terhadap penjumlahan sub array sampai indeks ke *i* - 1 + arr[i],, kemudian membandingkannya dengan arr[i]. Nilai yang maximum akan dimasukkan kedalam variabel maximum lokal.

Metode *Dynamic Programming:*

1. Menginisialisasi variabel maks_subarray_sum (global max) dan maks_subarray_sum_local (lokal max) dengan arr[0].
2. Melakukan perulangan sebanyak n - 1 kalu (1 sampai n-1).

3. Meng-*update* nilai lokal max dengan nilai max(lokal max + arr[i], arr[i]).
4. Membandingkan nilai global max dengan lokal max, jika nilai lokal max lebih besar maka nilai global max di-*update.*
5. Me-*return* nilai global max.

BAB IV
ANALISIS

A. Brute Force

a. T(n) dan kelas efisiensi

$$T(n) = \sum_{i=0}^{n-1} \sum_{j=i}^{n-1} \sum_{k=i}^{j} 1 = \sum_{i=0}^{n-1} \sum_{j=i}^{n-1} (j - i - 1)$$

$$T(n) = \sum_{i=0}^{n-1} \frac{1}{2}(i - n)(3 + i - n)$$

$$T(n) = \frac{1}{6} n(n^2 - 3n - 4)$$

$$T(n) = \frac{n^3}{6} - \frac{n^2}{2} - \frac{2n}{3} \in O(n^3)$$

B. Dynamic Programming

a. T(n) dan kelas efisiensi

$$T(n) = \sum_{i=1}^{n} 1$$

$$T(n) = \sum_{i=1}^{n} n \in O(n)$$

C. Graf dan tabel perbandingan

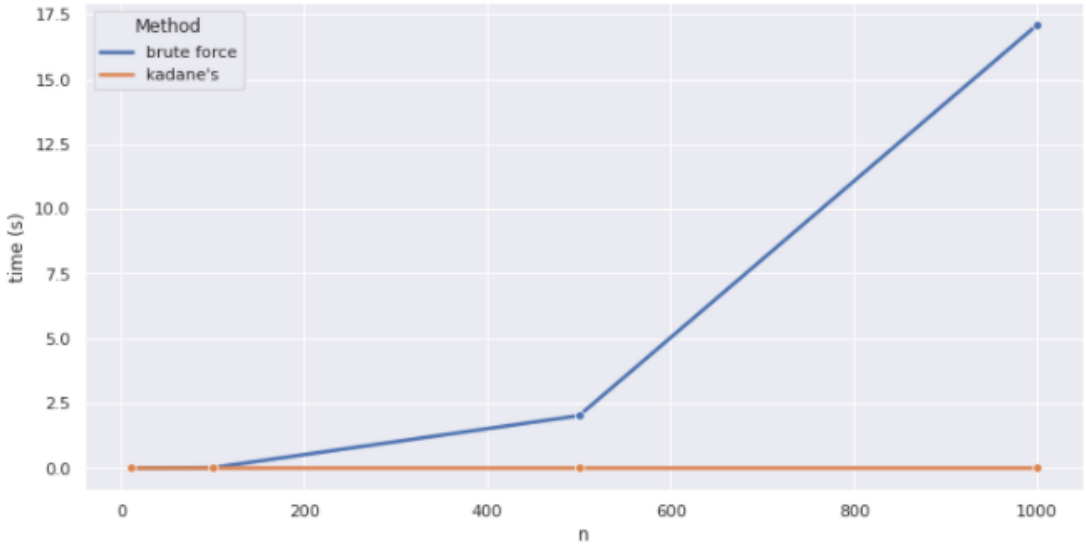
a. Tabel

	Method	n	Answer	Time
0	brute force	10	2480	0.000081
1	kadane's	10	2480	0.000006
2	brute force	100	8075	0.017926
3	kadane's	100	8075	0.000038
4	brute force	500	12880	2.023917
5	kadane's	500	12880	0.000270
6	brute force	1000	11874	17.108577
7	kadane's	1000	11874	0.000292

	Method	Time
0	brute force	19.150501
1	kadane's	0.000606

Dapat kita lihat bahwa pada *n* = 10 dan *n* = 100, perbedaan algoritma *Brute Force* dan *Dynamic Programming* tidak terlalu jauh. Tetapi, ketika *n* >= 500, perbedaannya cukup signifikan, bahkan ketika *n* = 1000, algoritma *Brute Force* kewalahan dan membutuhkan waktu 17 detik, dimana *Dynamic Programming* hanya membutuhkan 0,000292 detik.

b. Grafik



Perbandingannya juga tergambar pada grafik di atas, dimana algoritma *Brute Force* memiliki *order of growth* yang tinggi.

BAB V
KESIMPULAN

A. Kesimpulan

Dari hasil analisis yang kita lakukan di atas, dapat kita simpulkan bahwa pendekatan *Dynamic Programming* secara signifikan lebih efisien dibandingkan pendekatan *Brute Force*. Khususnya implementasi *Dynamic Programming* dengan menggunakan algoritma Kadane yang hanya menggunakan 1 loop untuk menyelesaikan permasalahan di atas. Sehingga, jelas bahwa strategi terbaik yang dapat digunakan untuk menyelesaikan permasalahan ini adalah *Dynamic Programming,* khususnya implementasi *iterative programming* pada algoritma Kadane.

Daftar Pustaka

"*Competitive Programmer's Handbook*". cses.fi. 3 Juli 2018.
https://cses.fi/book/book.pdf

"Maximum Subarray Sum". enjoyalgorithms.com. 2020.
https://www.enjoyalgorithms.com/blog/maximum-subarray-sum

"Analysis of loop in Programming". enjoyalgorithms.com. 2020.
https://www.enjoyalgorithms.com/blog/time-complexity-analysis-of-loop-in-programming

"Kadane's Algorithm". medium.com. 31 Desember 2018.
https://medium.com/@rsinghal757/kadanes-algorithm-dynamic-programming-how-and-why-does-it-work-3fd8849ed73d

"Tubes Strategi Algoritma Coin Change Problem". github.com. 21 Juni 2021.
https://github.com/ShinyQ/Tubes-Strategi-Algoritma_Coin-Change-Problem/blob/main/Tugas_Besar_Strategi_Algoritma_Coin_Change_Problem.ipynb

"Joseph Born Kadane". en.wikipedia.org. 17 Januari 2021.
https://en.wikipedia.org/wiki/Joseph_Born_Kadane

"Maximum Subarray Problem". en.wikipedia.org. 19 April 2022.
https://en.wikipedia.org/wiki/Maximum_subarray_problem#Kadane's_algorithm

"Kadane's Algorithm Explained". hackernoon.com. 20 Februari 2019.
https://hackernoon.com/kadanes-algorithm-explained-50316f4fd8a6

Lampiran

1. **Source code program:**
   https://github.com/hidayattaufiqur/Tubes-STRAGO

2. ***Brute Force* Code**

```python
# brute force menggunakan 3 nested loop
def max_subarray_sum_bf(arr, n):
    maks_subarray_sum = 0
    for i in range(n):
        for j in range(i, n):
            subarray_sum = 0
            for k in range(i, j+1):
                subarray_sum += arr[k]

            if subarray_sum > maks_subarray_sum:
                maks_subarray_sum = subarray_sum

    return maks_subarray_sum
```

3. ***Dynamic Programming* Code**

```python
# kadane's algorithm menggunakan 1 loop
def max_subarray_sum_kadane(arr, n):
    maks_subarray_sum = arr[0]
    maks_subarray_sum_local = arr[0]
    for i in range(1, n):
        maks_subarray_sum_local = max(maks_subarray_sum_local + arr[i], arr[i])

        if maks_subarray_sum < maks_subarray_sum_local:
            maks_subarray_sum = maks_subarray_sum_local

    return maks_subarray_sum
```