Penggunaan *dynamic programming* dengan memoisasi untuk menyelesaikan beberapa jenis puzzle

Bintang Hijriawan Jachja - 13521003
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): 13521003@std.stei.itb.ac.id

***Abstrak*— Makalah ini membahas penggunaan dynamic programming dengan memoisasi sebagai pendekatan efektif untuk menyelesaikan beberapa jenis puzzle. Puzzle sering kali melibatkan pemecahan masalah dengan mempertimbangkan langkah-langkah yang mungkin diambil dan memilih langkah terbaik untuk mencapai solusi optimal. Dynamic programming adalah teknik pemrograman yang memecah masalah menjadi submasalah yang lebih kecil, memecahkan masing-masing submasalah secara terpisah, dan menyimpan solusinya untuk penggunaan berulang.**

Dalam konteks ini, memoisasi merupakan teknik yang digunakan untuk menyimpan hasil komputasi yang sudah dilakukan sebelumnya, sehingga menghindari pengulangan yang tidak perlu dan meningkatkan efisiensi algoritma. Dalam makalah ini, penulis menjelaskan bagaimana dynamic programming dengan memoisasi dapat diterapkan untuk menyelesaikan berbagai jenis puzzle, seperti teka-teki logika, teka-teki matematika, dan teka-teki kombinatorik.

Penulis membahas langkah-langkah umum dalam menggunakan dynamic programming dengan memoisasi untuk menyelesaikan puzzle, termasuk identifikasi submasalah, pembuatan fungsi rekursif, dan penyimpanan hasil dalam struktur data yang sesuai. Mereka juga memberikan contoh implementasi dan analisis kinerja untuk beberapa jenis puzzle yang sering ditemui.

Makalah ini memberikan wawasan yang berharga tentang penggunaan dynamic programming dengan memoisasi dalam konteks pemecahan puzzle. Dengan menggunakan pendekatan ini, penyelesaian puzzle menjadi lebih efisien dan memungkinkan eksplorasi solusi yang lebih kompleks. Penelitian lebih lanjut dalam pengembangan teknik ini dapat memberikan kontribusi yang signifikan terhadap pemahaman dan penyelesaian puzzle yang lebih canggih.

Keywords—*dynamic programming*, pemrograman dinamis, memoisasi, puzzle

I. PENDAHULUAN

Dynamic programming atau pemrograman dinamis adalah salah satu metode pemrogramman yang dapat digunaakan untuk memcah masalah menjadi masalah yang lebih kecil, kemudian memecahkan masalah-masalah yang lebih kecil tersenut dan menyimpan solusinya untuk digunakan kembali. Pendekatan ini bertujuan untuk mengurangi kompleksitas algoritma yang digunakan dengan menghindari perhitungan berulang. Pendekatan pemrograman dinamis dilakukan dengan membagi masalah menjadi masalah lain yang lebih kecil dan memecahkan masalah-masalah tersebut secara iteratif dengan memanfaatkan hasil dari pemecahan masalah lainnya. Solusi optimal untuk masalah awal didapatkan dengan menggabungkan solusi-solusi optimal dari masalah-masalah yang lebih kecil.

Pemrograman dinamis sering digunakan untuk memecahkan masalah-masalah terkait optimasi, seperti masalah penjadwalan, masalah rute terpendek, dan masalah optimasi lainnya. Pada makalah ini, penulis akan membahas penggunaan pendekatan Pemrograman dinamis untuk menyelesaikan beberapa masalah perhitungan yang bisa dioptimasikan dengan pendekatan pemrograman dinamis pada puzzle.

Teknik memoisasi pada pemrogramman yaitu menyimpan hasil komputasi yang akan dilakukan berulang-ulang kedalam memori sehingga kita hanya perlu melakukan satu kali komputasi dan menggunakan hasil komputasi tersebut berulang-ulang. Teknik ini dapat digunakan jika ada komputasi yang menghasilkan suatu hasil yang tetap ketika dimasukkan parameter yang sama.

Dalam penggunaan teknik ini, kita mengorbankan kompleksitas ruang untuk meningkatkan efisiensi waktu dari sebuah algoritma. Dengan menyimpan hasil dari suatu komputasi kedalam sebuah struktur data, misalkan array, dictionary, atau map, pemanggilan fungsi dengan parameter yang sama bisa dilakukan secara konstan dan tidak perlu melakukan komputasi lagi terhadap fungsi tersebut.

II. TEORI DASAR

2.1 Pemrogramman dinamis

Pemrograman dinamis adalah teknik algoritmik yang kuat untuk memecahkan masalah optimisasi dengan menguraikannya menjadi submasalah dan menyelesaikannya secara rekursif. Berdasarkan prinsip substruktur optimal, solusi optimal suatu masalah dapat dibangun dari solusi optimal submasalahnya.

Ide utama pemrograman dinamis adalah menyimpan hasil pemanggilan fungsi yang mahal dalam sebuah tabel atau struktur data sehingga dapat digunakan kembali saat dibutuhkan. Pemrograman dinamis dapat sangat meningkatkan efisiensi algoritma dengan menghindari perhitungan yang berlebihan.

Langkah-langkah berikut menggambarkan proses umum penyelesaian masalah menggunakan pemrograman dinamis.

1. Mencirikan struktur solusi optimal.

Memahami masalah dan mengidentifikasi komponen dan elemen yang membentuk solusi terbaik. Tentukan bagaimana elemen-elemen ini berhubungan satu sama lain dan bagaimana mereka dapat digabungkan untuk membentuk solusi keseluruhan yang terbaik.

2. Mendefinisikan sifat rekursif dari masalah.

Ekspresikan masalah dalam bentuk submasalah yang lebih kecil yang mirip dengan masalah aslinya tetapi memiliki ukuran masukan yang lebih kecil. Merumuskan hubungan iteratif yang menghubungkan solusi dari masalah awal dengan solusi dari submasalahnya.

3. Identifikasi submasalah yang tumpang tindih.

Tentukan apakah ada submasalah yang tumpang tindih. H. Apakah sub-masalah yang sama diselesaikan berkali-kali. Ini adalah langkah penting dalam pemrograman dinamis karena menyimpan hasil dalam tabel atau struktur memo menghindari perhitungan yang berlebihan. Buat tabel memo.

4. Buat struktur data (array, matriks, tabel hash, dll.) untuk menyimpan hasil subproblem. Inisialisasi tabel dengan nilai awal yang sesuai yang mewakili kasus dasar dari masalah.

5. Menerapkan pendekatan top-down atau bottom-up.

Ada dua pendekatan utama untuk memecahkan masalah pemrograman dinamis.

- Pendekatan top-down (memoisasi):

Mulailah dari masalah asli dan selesaikan secara rekursif, menggunakan memoisasi untuk menyimpan hasil submasalah. Setiap kali kami perlu menyelesaikan sub-masalah, kami memeriksa apakah hasilnya sudah ada di tabel memo. Jika demikian, dapatkan hasilnya. Jika tidak, hitung secara rekursif dan simpan ke spreadsheet untuk digunakan nanti.

- Pendekatan bottom-up (tabulasi):

Mulailah dengan menyelesaikan sub-masalah terkecil dan secara iteratif membangun solusi untuk sub-masalah yang lebih besar hingga akhirnya menyelesaikan masalah aslinya. Menggunakan tabel memo untuk menyimpan hasil submasalah dalam urutan yang sistematis memastikan bahwa setiap submasalah diselesaikan hanya sekali untuknya dan solusi tersebut tersedia saat dibutuhkan.

6. Ekstrak solusi dari tabel.

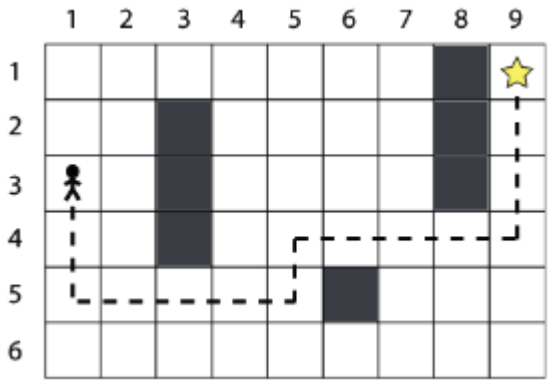
Setelah semua solusi sub-masalah dimasukkan ke dalam tabel, ambil solusi masalah asli dari tabel. Ini adalah solusi optimal. Pemrograman dinamis dapat diterapkan pada banyak jenis masalah, seperti: *shortest path problem*, *resource allocation problem*, pengurutan, penjadwalan, dll. Kuncinya adalah mengidentifikasi substruktur yang optimal dan submasalah yang tumpang tindih dari masalah yang dihadapi.

Pemrograman dinamis tidak selalu merupakan pendekatan terbaik untuk semua masalah. Dalam beberapa kasus, algoritma *greedy*, *divide and conquer*, atau teknik lain mungkin lebih tepat.

2.2 Puzzle

2.2.1 Grid Problem

Pada permasalahan ini, diberikan sebuah matriks / grid berukurna n x m. Berapa banyak cara yang bisa dilalui dari titik di pojok kiri atas menuju titik di pojok kanan bawah. Permasalahan ini juga bisa dikembangkan dengan titik awal diubah menjadi sembarang titik A dan titik akhir menjadi sembarang titik B, atau menambahkan koordinat yang tidak bisa dilalui.



Gambar 2.1. Grid Puzzle

2.2.2 Fibonacci Problem

Angka fibonacci adalah urutan angka dimana angka ke n, F(n) = F(n-1) + F(n-2), dimana n >=2 dan biasanya F(0) = 0 dan F(1) = 1. Angka fibonacci ini biasa digunakan sebagai contoh untuk menggambarkan beberapa teknik algoritma, seperti rekursif dan iteratif.

Pada permasalahan ini, diberikan rumus fibonacci yaitu F(n) = F(n-1) + F(n-2), dimana n >=2 dan F(0) = 0 dam F(1) = 1. Kita diminta untuk menentukan angka ke-n dari urutan fibonacci tersebut.

Permasalahan ini bisa dikembangkan dengan mengubah urutan menjadi urutan yang mirip dengan urutan fibonacci, yaitu

dengan mengubah rumus F(n) menjadi suatu perhitungan lain yang melibatkan 2 angka sebelumnya. Urutan serupa dengan rumus F(n) = F(n-1) + F(n-2) + (F(n-3) disebut dengan urutan tribonacci.

2.2.3 Coins problem

Diberikan sekumpulan koin dengan jumlah tak hingga dengan nominal tertentu. Berapa banyak koin minimum untuk mencapai target total N. Masalah ini mirip dengan beberapa masalah yang sudah ada, seperti berapa minimum langkah yang diperlukan untuk mencapai anak tangga ke-N, dsb. Tapi dalam makalah ini, penulis memilih menggunakan permasalahan koin sebagai contoh.

Implementasi Algoritma Pemrograman Dinamis dengan Memoisasi Untuk Menyelesaikan Beberapa Jenis Puzzle

3.1 Grid Problem

Pada permasalahan ini, diberikan sebuah matriks / grid berukuran 12 x 17. Berapa banyak cara yang bisa dilalui dari titik di pojok kiri atas menuju titik di pojok kanan bawah, jika kita hanya bisa bergerak kekanan / kebawah.

Dalam masalah ini, gerakkan perlu di batasi, seperti hanya bisa kebawah dan kekanan, agar komputasi yang dilakukan tidak menjadi rekursif tanpa batas. Untuk pengembangan masalah dengan obstacle, dimana suatu titik tidak bisa dilalui, atau modifikasi titik awal dan akhir menjadi sembarang titik pada matriks, diperlukan perubahan pada persoalan dan program agar bisa mendapatkan hasil yang diinginkan.

Untuk menyelesaikan masalah ini dengan menggunakan pendekatan pemrograman dinamis dengan memoisasi, kita perlu melakukan langkah-langkah berikut.

1. Buat tabel memoisasi, isi semua nilai dalam tabel dengan –1.
2. Untuk setiap sel pada baris 1 dan kolom 1, isi tabel dengan angka 1.
3. Iterasi untuk setiap baris dan kolom, untuk setiap sel MATRIKS[I][j] pada tabel, isi sel dengan nilai MATRIKS[i-1][j] + MATRIKS[i][j-1]
4. Banyak cara untuk mencapai sebuah titik (i,j) adalah MATRIKS[i][j]

indeks	0	1	2
0	1	1	1
1	1	2	3
2	1	3	6
3	1	4	10
4	1	5	15

Gambar 3.1. tabel memoisasi dalam grid problem ukuran 5 x 6

Pengimplementasian algoritma dalam program python dicontohkan sebagai berikut:

```python
import time

def countPathsDynamicProgramming(row, col, m
    for i in range(col):
        for j in range(row):
            if i == 0 or j == 0:
                memo[j][i] = 1
            else:
                memo[j][i] = memo[j - 1][i]
    return memo[row-1][col-1]

def uniquePaths(m, n):
    memo = [[-1] * n for _ in range(m)]
    return countPathsDynamicProgramming(m, n

def countPaths(n, m):
    if n == 1 or m == 1:
        return 1
    return countPaths(n-1, m) + countPaths(n
```

Gambar 3.2. kode program python untuk memecahkan masalah grid problem dengan pemrograman dinamis dan brute force

```python
m = 12
n = 17

start_time = time.time()
unique_paths = uniquePaths(m, n)
end_time = time.time()
runtime = end_time - start_time
print(unique_paths)
print(f"Runtime: {runtime}")

start_time = time.time()
unique_paths = countPaths(m, n)
end_time = time.time()
runtime = end_time - start_time
print(unique_paths)
print(f"Runtime: {runtime}")
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
PS C:\Data\Code\Python\Makalah Stima> & C:/Users/binta/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Data/Code
13037895
Runtime: 0.0
13037895
Runtime: 1.1953210830688477
PS C:\Data\Code\Python\Makalah Stima>
```

Gambar 3.3 pengujian waktu antara algoritma dengan pemrograman dinamis dan brute force

Dari hasil uji diatas, algoritma dengan pemrograman dinamis membutuhkan waktu yang sangat kecil untuk ukuran matriks 12 x 17, yaitu mendekati 0. sedangkan pendekatan brute force membutuhkan waktu sekitar 1.2 detik. Hal ini menunjukkan pendekatan pemrograman dinamis lebih efisien untuk kasus ini daripada algoritma brute force.

3.2 Fibonacci Problem

Pada permasalahan ini, diberikan rumus fibonacci yaitu F(n) = F(n-1) + F(n-2), dimana n >=2 dan F(0) = 0 dam F(1) = 1. Kita diminta untuk menentukan angka ke-35 dari urutan fibonacci tersebut.

Untuk menyelesaikan masalah ini dengan menggunakan pendekatan pemrograman dinamis dengan memoisasi, kita perlu melakukan langkah-langkah berikut.

1. Buat sebuah array memoisasi dengan panjang 15+1 dengan nilai awal –1.
2. Isi array[0] dengan angka 0, array[1] dengan angka 1.
3. Lakukan iterasi, untuk setiap array[n] pada array, isi dengan array[n-1] + array[n-2]

Pengimplementasian algoritma daalm program python dicontohkan sebagai berikut:

FibonacciProblem.py

```python
import time

def fibonacciDynamicProgramming(n
    for i in range(n+1):
        if i == 0:
            memo[i] = 0
        elif i == 1:
            memo[i] = 1
        else:
            memo[i] = memo[i-1] +
    return memo[n]

def nthFibonacci(n):
    memo = [-1] * (n + 1)
    return fibonacciDynamicProgra

def fibonacci(n):
    if n <= 1:
        return n
    return fibonacci(n - 1) + fib
```

Gambar 3.3. kode program untuk menyelesaikan masalah fibonacci dengan pemrograman dinamis dan brute force

```
FibonacciProblem.py > ...
24  n = 35
25
26  start_time = time.time()
27  fibonacci_number_dynamic_programming = nthFibonacci(n)
28  end_time = time.time()
29  runtime = end_time - start_time
30  print(fibonacci_number_dynamic_programming)
31  print(f"Runtime: {runtime}")
32
33  start_time = time.time()
34  fibonacci_number = fibonacci(n)
35  end_time = time.time()
36  runtime = end_time - start_time
37  print(fibonacci_number)
38  print(f"Runtime: {runtime}")

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

Runtime: 1.2293610572814941
PS C:\Data\Code\Python\Makalah Stima> & C:/Users/binta/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Data/Code/Pyt
"
9227465
Runtime: 0.0
9227465
Runtime: 1.176055908203125
PS C:\Data\Code\Python\Makalah Stima>
```

Gambar 3.5 pengujian waktu antara algoritma dengan pemrograman dinamis dan brute force

Dalam hal ini, nilai dari memo[i-1] dan memo[i-2] tidak perlu dihitung kembali, melainkan disimpan dalam suatu array, yang kemudian digunakan kembali untuk perhitungan angka selanjutnya. Pada algoritma dengan pemrograman dinamis, waktu yang diperlukan sangat kecil, yaitu mendekati 0, sedangkan waktu untuk brute force adalah sekitar 1.1 detik. Hal ini menunjukkan pendekatan pemrograman dinamis lebih efisien untuk kasus ini daripada algoritma brute force.

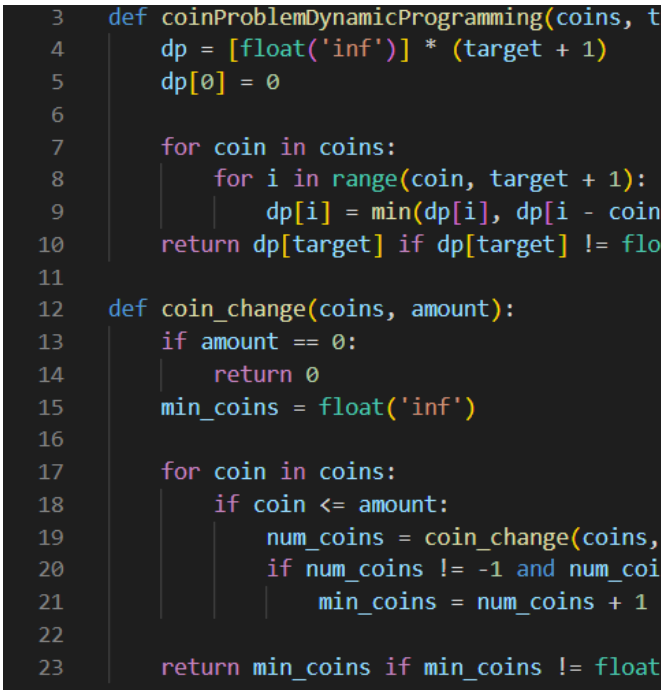
3.3.3 Coins Problem

Pada permasalahan ini, diberikan sekumpulan koin dengan jumlah tak hingga dengan nominal [1, 2, 5, 10, 20, 50]. Berapa banyak koin minimum untuk mencapai target total 1123.

Untuk menyelesaikan masalah ini dengan menggunakan pendekatan pemrograman dinamis dengan memoisasi, kita perlu melakukan langkah-langkah berikut.

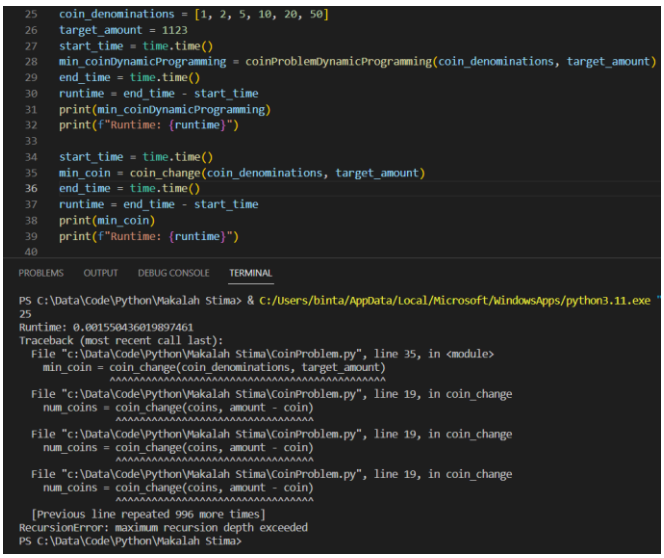
1. Buat array memoisasi dengan panjang 1123+1, isi nilai dengan angka tak hingga
2. Array[0] diisi dengan 0
3. Lakukan iterasi pada coin, lalu iterasi pada array dari coin hingga target+1
4. Untuk setiap array[I] diisi dengan nilai minimum(array[i], array[i-coin] + 1)
5. nilai array[target] menyatakan minimum koin yang diperlukan, nilai array[target] tak hingga, artinya target tidak bisa dipenuhi.

Pengimplementasian algoritma daalm program python dicontohkan sebagai berikut:



```python
def coinProblemDynamicProgramming(coins, t
    dp = [float('inf')] * (target + 1)
    dp[0] = 0

    for coin in coins:
        for i in range(coin, target + 1):
            dp[i] = min(dp[i], dp[i - coin
    return dp[target] if dp[target] != flo

def coin_change(coins, amount):
    if amount == 0:
        return 0
    min_coins = float('inf')

    for coin in coins:
        if coin <= amount:
            num_coins = coin_change(coins,
            if num_coins != -1 and num_coi
                min_coins = num_coins + 1

    return min_coins if min_coins != float
```

Gambar 3.4 kode program untuk menyelesaikan coin problem dengan pemrograman dinamis dan brute force



```python
coin_denominations = [1, 2, 5, 10, 20, 50]
target_amount = 1123
start_time = time.time()
min_coinDynamicProgramming = coinProblemDynamicProgramming(coin_denominations, target_amount)
end_time = time.time()
runtime = end_time - start_time
print(min_coinDynamicProgramming)
print(f"Runtime: {runtime}")

start_time = time.time()
min_coin = coin_change(coin_denominations, target_amount)
end_time = time.time()
runtime = end_time - start_time
print(min_coin)
print(f"Runtime: {runtime}")
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

PS C:\Data\Code\Python\Makalah Stima> & C:/Users/binta/AppData/Local/Microsoft/WindowsApps/python3.11.exe
25
Runtime: 0.001550436019897461
Traceback (most recent call last):
  File "c:\Data\Code\Python\Makalah Stima\CoinProblem.py", line 35, in <module>
    min_coin = coin_change(coin_denominations, target_amount)
               ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "c:\Data\Code\Python\Makalah Stima\CoinProblem.py", line 19, in coin_change
    num_coins = coin_change(coins, amount - coin)
                ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "c:\Data\Code\Python\Makalah Stima\CoinProblem.py", line 19, in coin_change
    num_coins = coin_change(coins, amount - coin)
                ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "c:\Data\Code\Python\Makalah Stima\CoinProblem.py", line 19, in coin_change
    num_coins = coin_change(coins, amount - coin)
                ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  [Previous line repeated 996 more times]
RecursionError: maximum recursion depth exceeded
PS C:\Data\Code\Python\Makalah Stima>
```

Gambar 3.5 pengujian waktu antara algoritma dengan pemrograman dinamis dan brute force

Dalam hal ini, nilai array[i-coin] disimpan sehingga tidak perlu dilakukan komputasi ulang untuk menentukan berapa minimum koin yang diperlukan untuk mencapat total i-koin. Pada algoritma brute force, program python mengeluarkan error *maximum recursion depth exceeded*, yang mana diakibatkan pemanggilan fungsi rekursif yang melebihi batas. Artinya, untuk nilai target yang besar, algoritma brute force tidak bisa digunakan, sedangkan algoritma dengan pemrograman dinamis membutuhkan waktu yang sangat kecil untuk menghitung minimum koin dengan nilai target 1123.

Kesimpulan

4.1. Kesimpulan

Dalam makalah ini, telah diuji penggunaan dynamic programming dengan memoisasi sebagai pendekatan untuk menyelesaikan beberapa jenis puzzle. Hasil pengujian menunjukkan bahwa teknik ini dapat secara signifikan meningkatkan efisiensi dan kecepatan pemecahan puzzle.

Dengan menerapkan dynamic programming, masalah yang kompleks dapat dipecahkan dengan memecahkannya menjadi submasalah yang lebih kecil dan memanfaatkan solusi yang sudah diketahui sebelumnya. Hal ini mengurangi waktu komputasi yang dibutuhkan dan menghindari pengulangan yang tidak perlu.

Penelitian ini juga menunjukkan bahwa memoisasi, yaitu menyimpan hasil komputasi sebelumnya, dapat memberikan manfaat yang signifikan dalam pemecahan puzzle. Dengan menyimpan solusi yang sudah diketahui, pengulangan perhitungan dapat dihindari, sehingga meningkatkan efisiensi dan mengurangi kompleksitas algoritma.

Selain itu, penggunaan dynamic programming dengan memoisasi juga memungkinkan eksplorasi solusi yang lebih kompleks dan memungkinkan penyelesaian puzzle yang lebih rumit. Dalam pengujian yang dilakukan, teknik ini telah berhasil diterapkan pada berbagai jenis puzzle, termasuk teka-teki logika, teka-teki matematika, dan teka-teki kombinatorik.

Secara keseluruhan, penelitian ini menunjukkan bahwa penggunaan dynamic programming dengan memoisasi adalah pendekatan yang efektif dan bermanfaat untuk menyelesaikan puzzle. Implementasi yang baik dari teknik ini dapat meningkatkan efisiensi pemecahan puzzle dan membuka peluang untuk eksplorasi solusi yang lebih kompleks di masa depan.

VIDEO LINK AT YOUTUBE *(Heading 5)*

UCAPAN TERIMA KASIH

Penulis ingin mengucapkan terima kasih yang tulus kepada berbagai pihak yang telah memberikan dukungan dan kontribusi yang berarti dalam penulisan makalah ini. Terima kasih yang sebesar-besarnya kepada dosen yang telah memberikan bimbingan, arahan, dan pengetahuan yang sangat berharga dalam proses penulisan makalah ini.

Penulis juga ingin mengucapkan terima kasih kepada semua sumber daya dan referensi yang telah digunakan sebagai acuan dalam penulisan makalah ini. Kontribusi dan pengetahuan yang terkandung dalam sumber daya tersebut telah memberikan wawasan yang berharga dan memperkaya isi makalah ini.
Terima kasih juga kepada keluarga dan teman yang memberikan dukungan moral dan semangat selama proses penulisan. Dukungan dan motivasi yang diberikan sangat berarti dan memberikan inspirasi dalam menyelesaikan makalah ini.
Penulis juga ingin menyampaikan terima kasih kepada institusi pendidikan yang telah memberikan fasilitas dan sarana yang

diperlukan dalam penulisan makalah ini. Infrastruktur yang disediakan telah memudahkan akses ke informasi dan sumber daya yang diperlukan.

REFERENSI

[1] Munir, Rinaldi. 2021. Bahan Kuliah IF2211 Strategi Algoritma: Algoritma Brute Force (Bagian 1). https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-Brute-Force-(2022)-Bag1.pdf Diakses pada 21 Mei 2023.

[2] Munir, Rinaldi. 2021. Bahan Kuliah IF2211 Strategi Algoritma: Algoritma Brute Force (Bagian 2). https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-Brute-Force-(2022)-Bag2.pdf Diakses pada 21 Mei 2023.

[3] bamsbung.id. Apa itu memoisasi dan Mengapa Penting? https://bamsbung.id/blog/apa-itu-memoisasi-dan-mengapa-penting/

Diakses pada 21 Mei 2023

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, 22 Mei 2023

Ttd

Bintang Hijriawan Jachja – 13521003