LAPORAN TUGAS BESAR

PERBANDINGAN ALGORITMA GREEDY DAN DYNAMIC PROGRAMMING PADA MESIN ATM

DISUSUN OLEH:

KELOMPOK 1

MUHAMAD MEIDY MAHARDIKA - 1304202024

FARHAN RANGKUTI - 1304202025

MUH. NUR - 1304201017

PROGRAM STUDI S1 INFORMATIKA PJJ

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY

2023

DAFTAR ISI

ABSTRAK

Mesin ATM, atau Automated Teller Machine, merupakan mesin otomatis yang digunakan untuk transaksi keuangan, terutama penarikan uang. Penelitian ini bertujuan menerapkan algoritma Greedy dan Dynamic Programming pada mesin ATM agar dapat menarik uang dalam bentuk uang kertas pada transaksi penarikan. Algoritma Greedy akan digunakan untuk menentukan kombinasi uang kertas secara efisien secara individu, sedangkan algoritma Dynamic Programming akan mencari kombinasi uang kertas yang menghasilkan jumlah kembalian minimal secara keseluruhan. Dengan menerapkan kedua algoritma ini, diharapkan mesin ATM dapat memberikan uang kembalian dalam bentuk uang kertas dengan jumlah minimal, mengoptimalkan penggunaan uang dan efisiensi penyimpanan dalam mesin ATM pada transaksi penarikan uang.

PENDAHULUAN

Mesin ATM atau Automated Teller Machine telah menjadi salah satu inovasi penting dalam industri perbankan modern. Mesin ini memungkinkan pengguna untuk melakukan berbagai transaksi keuangan, termasuk penarikan uang tunai, transfer dana, dan pemeriksaan saldo, dengan cepat dan mudah. Dalam konteks penarikan uang tunai, kemampuan mesin ATM untuk memberikan kembalian uang secara efisien merupakan aspek penting yang perlu dipertimbangkan.

Dalam laporan ini, kami akan membahas perbandingan antara dua algoritma yang digunakan dalam mesin ATM untuk menghasilkan kembalian uang secara optimal, yaitu algoritma Greedy dan Dynamic Programming. Kedua algoritma ini bertujuan untuk mengoptimalkan jumlah uang kertas yang diberikan sebagai kembalian kepada pengguna dengan meminimalkan jumlah uang kertas yang dikeluarkan.

Algoritma Greedy adalah pendekatan yang sederhana namun kuat dalam menyelesaikan masalah optimasi. Pada mesin ATM, algoritma Greedy digunakan untuk menentukan kombinasi uang kertas yang paling efisien secara individu pada setiap tahap penarikan. Algoritma ini memilih pecahan uang kertas terbesar yang masih memenuhi batas kembalian yang diminta oleh pengguna, sehingga menghasilkan jumlah kertas yang lebih sedikit namun mencapai jumlah yang diinginkan.

Namun, algoritma Greedy memiliki kelemahan dalam menghasilkan solusi optimal secara keseluruhan. Hal ini disebabkan karena pendekatan ini hanya mempertimbangkan kondisi lokal pada setiap tahap penarikan tanpa memperhatikan dampaknya pada kombinasi kembalian keseluruhan. Inilah mengapa diperlukan penggunaan algoritma Dynamic Programming.

Algoritma Dynamic Programming, di sisi lain, mencari solusi optimal dengan memecahkan masalah yang lebih besar menjadi submasalah yang lebih kecil. Pada mesin ATM, algoritma Dynamic Programming digunakan untuk mencari kombinasi uang kertas yang menghasilkan

jumlah kembalian yang minimal secara keseluruhan. Dengan menyimpan dan memanfaatkan informasi tentang submasalah yang sudah dipecahkan, algoritma ini dapat menghindari pengulangan dan menghasilkan solusi yang optimal secara keseluruhan.

Melalui perbandingan antara algoritma Greedy dan Dynamic Programming pada mesin ATM, kami bertujuan untuk mengevaluasi kelebihan dan kekurangan masing-masing algoritma dalam menghasilkan kembalian uang secara efisien. Kami juga akan mempertimbangkan aspek lain seperti kecepatan eksekusi, kompleksitas, dan penggunaan sumber daya dalam menerapkan kedua algoritma ini pada mesin ATM.

Dengan pemahaman yang mendalam tentang perbandingan antara algoritma Greedy dan Dynamic Programming pada mesin ATM, diharapkan laporan ini dapat memberikan wawasan yang berharga dalam pengembangan dan perbaikan sistem mesin ATM di masa depan. Selain itu, laporan ini juga dapat memberikan kontribusi pada pengembangan algoritma optimasi yang lebih baik untuk aplikasi ke

DASAR TEORI

Algoritma Greedy dan Dynamic Programming adalah dua pendekatan yang berbeda dalam mencari solusi optimal untuk masalah pada mesin ATM. Berikut ini adalah perbandingan antara keduanya:

Keputusan pada setiap tahap:

- Algoritma Greedy: Algoritma Greedy membuat keputusan berdasarkan informasi yang tersedia pada tahap tersebut tanpa mempertimbangkan konsekuensi masa depan. Keputusan yang diambil diharapkan akan memberikan solusi terbaik pada saat itu.
- Algoritma Dynamic Programming: Algoritma Dynamic Programming memecah masalah menjadi submasalah yang lebih kecil dan menyelesaikan sub masalah tersebut secara terpisah. Solusi optimal dari sub masalah yang lebih kecil digunakan untuk membangun solusi optimal untuk masalah utama. Keputusan pada setiap tahap didasarkan pada solusi submasalah yang sudah diketahui.

Pendekatan optimal:

- Algoritma Greedy: Algoritma Greedy tidak menjamin solusi yang ditemukan adalah solusi optimal secara keseluruhan. Keputusan yang diambil pada setiap tahap didasarkan pada harapan akan mencapai solusi terbaik pada saat itu, tanpa mempertimbangkan konsekuensi jangka panjang.
- Algoritma Dynamic Programming: Algoritma Dynamic Programming memiliki sifat optimal. Dengan memecah masalah menjadi submasalah yang lebih kecil dan menggunakan solusi optimal dari sub masalah tersebut, algoritma ini dapat mencapai solusi optimal untuk masalah utama.

Penggunaan memorisasi:

- Algoritma Greedy: Algoritma Greedy tidak menggunakan memorisasi, karena keputusan pada setiap tahap tidak dipengaruhi oleh keputusan sebelumnya.
- Algoritma Dynamic Programming: Algoritma Dynamic Programming menggunakan memorisasi untuk menyimpan solusi submasalah yang sudah diketahui. Dengan memanfaatkan informasi yang sudah ada, algoritma ini menghindari perhitungan ulang yang tidak perlu dan meningkatkan efisiensi.

Efisiensi dan kompleksitas:

- Algoritma Greedy: Algoritma Greedy umumnya memiliki kompleksitas waktu yang lebih rendah daripada algoritma Dynamic Programming. Namun, solusi yang dihasilkan mungkin tidak selalu optimal.
- Algoritma Dynamic Programming: Algoritma Dynamic Programming memiliki kompleksitas waktu yang lebih tinggi karena melibatkan pemecahan ulang masalah yang lebih kecil. Namun, algoritma ini menjamin solusi optimal.

Penerapan pada masalah mesin ATM:

- Algoritma Greedy: Algoritma Greedy dapat digunakan dalam beberapa aspek mesin ATM, seperti pengaturan jumlah uang di mesin ATM atau pengoptimalan pemecahan uang. Namun, perlu diperhatikan bahwa solusi yang dihasilkan mungkin tidak selalu optimal.
- Algoritma Dynamic Programming: Algoritma Dynamic Programming dapat digunakan untuk masalah yang lebih kompleks pada mesin ATM, seperti mengoptimalkan antrian transaksi atau penentuan jumlah minimum koin yang diperlukan untuk memberikan kembalian. Algoritma ini memberikan solusi optimal dalam konteks ini.

Dalam konteks mesin ATM, penerapan algoritma Greedy dan Dynamic Programming dapat memiliki perbedaan dalam hal efisiensi, ketepatan solusi, dan kompleksitas waktu. Berikut adalah poin-poin tambahan dalam perbandingan keduanya:

1. **Efisiensi:**
    - Algoritma Greedy cenderung lebih efisien dalam hal kompleksitas waktu, karena keputusan pada setiap tahap hanya didasarkan pada informasi saat itu. Namun, efisiensi ini mungkin tidak menghasilkan solusi yang optimal secara keseluruhan.
    - Algoritma Dynamic Programming cenderung memiliki kompleksitas waktu yang lebih tinggi karena melibatkan pemecahan ulang masalah yang lebih kecil. Namun, pendekatan ini memungkinkan penyelesaian masalah dengan solusi optimal.

2. **Ketepatan solusi:**
    - Algoritma Greedy tidak menjamin solusi yang ditemukan adalah solusi optimal secara keseluruhan. Keputusan yang diambil pada setiap tahap didasarkan pada pemikiran jangka pendek, mengoptimalkan solusi pada saat itu tanpa mempertimbangkan konsekuensi masa depan.
    - Algoritma Dynamic Programming memastikan solusi optimal karena memecah masalah menjadi submasalah yang lebih kecil dan menggunakan solusi optimal dari sub masalah tersebut. Pendekatan ini mempertimbangkan konsekuensi jangka panjang untuk mencapai solusi terbaik secara keseluruhan.

3. **Kebutuhan memorisasi**
    - Algoritma Greedy tidak menggunakan memorisasi, karena keputusan pada setiap tahap tidak dipengaruhi oleh keputusan sebelumnya. Setiap tahap dianggap independen.
    - Algoritma Dynamic Programming menggunakan memorisasi untuk menyimpan solusi submasalah yang sudah diketahui. Dengan cara ini, perhitungan ulang yang tidak perlu dapat dihindari, dan efisiensi algoritma dapat ditingkatkan.

4. **Kompleksitas masalah:**
    - Algoritma Greedy cenderung lebih cocok untuk masalah sederhana yang tidak melibatkan dependensi antara tahapan. Misalnya, dalam pengaturan jumlah uang di mesin ATM, algoritma Greedy dapat digunakan untuk memutuskan kombinasi mata uang yang paling efisien untuk disimpan.
    - Algoritma Dynamic Programming lebih cocok untuk masalah yang kompleks dan melibatkan dependensi antara tahapan. Misalnya, dalam mengoptimalkan antrian transaksi, algoritma Dynamic Programming dapat digunakan untuk menentukan urutan transaksi yang memberikan waktu penyelesaian tercepat.

A. IMPLEMENTASI

1. Greedy Algorithm

Pada mesin ATM, tujuannya adalah untuk memberikan penarikan dengan jumlah uang kertas seminimal mungkin. Algoritma Greedy akan terus memberikan uang kertas dengan denominasi terbesar sampai nilai yang tersisa untuk diberikan kurang dari nilai denominasi itu. Kemudian lanjut ke denominasi yang lebih rendah dan ulangi sampai jumlah nilai uang kertas yang dikeluarkan sama dengan nilai yang ingin ditarik.

```
1 usage
def withdrawal(amount, denomination):
    sorted_papers = sorted(denomination, reverse=True)
    change = []

    for papers in sorted_papers:
        while amount >= papers:
            change.append(papers)
            amount -= papers

    return change

1 usage
def hitung_total(Paper):
    total = 0
    for i in Paper:
        total = total + i
    return total


# Example usage:
denominations = [100, 50, 20, 10, 5]
withdraw_amount = int(input("Please input the amount you want to withdraw : "))


Paper = withdrawal(withdraw_amount, denominations)
total = hitung_total(Paper)

print("Uang:", *Paper)
print("Banyak lembaran : ", len(Paper))
print("Total uang : ", total, "Ribu Rupiah")
```

Program kemudian akan meminta berapa banyak uang yang ingin ditarik, kemudian program akan memanggil fungsi withdrawal yang berisi parameter nilai yang ingin ditarik dan denominasi uang kertas yang ada pada ATM, dalam kasus ini denominasinya adalah uang 100, 50, 20, 10, 5 Ribu Rupiah. Kemudian fungsi akan melakukan looping pada denominations mulai dari pecahan terbesar selama nilai tersebut tidak melebihi nilai input. Proses ini kemudian dilakukan sampai nilai yang diinginkan tercapai dan disimpan pada variabel array Paper.

```
C:\Users\User\PycharmProjects\pythonProject1\venv\Scripts
Please input the amount you want to withdraw : 1025
Uang: 100 100 100 100 100 100 100 100 100 100 20 5
Banyak lembaran :  12
Total uang :  1025 Ribu Rupiah

Process finished with exit code 0
```

2. Dynamic Programming

Fungsi withdraw() bertanggung jawab untuk menghitung jumlah minimum uang kertas yang diperlukan untuk menarik sejumlah uang tertentu dari mesin ATM menggunakan pendekatan dynamic programming. Fungsi tersebut menginisialisasi tabel dp untuk menyimpan jumlah minimum denominasi (uang kertas) yang diperlukan untuk setiap jumlah penarikan yang mungkin. Program tersebut melakukan iterasi melalui denominasi dan menghitung jumlah minimum denominasi yang diperlukan untuk setiap jumlah penarikan.

```python
# Dynamic Programming
1 usage
def withdraw(denominations, withdraw_amount):

    dp = [float('inf')] * (withdraw_amount + 1)
    dp[0] = 0
    Paper_picks = [[] for _ in range(withdraw_amount + 1)]


    for coin in denominations:

        for i in range(coin, withdraw_amount + 1):
            if dp[i - coin] + 1 < dp[i]:
                dp[i] = dp[i - coin] + 1
                Paper_picks[i] = Paper_picks[i - coin] + [coin]


    return dp[withdraw_amount] if dp[withdraw_amount] != float('inf') else print("You can't withdraw that amount"), Paper_picks[withdraw_amount]

denominations = [5,10,20,50,100]

withdraw_amount = int(input("input the withdraw_amount : "))
min_denominations, Paper_picks = withdraw(denominations, withdraw_amount)
print("Minimum number of papers needed:", min_denominations)
print("denominations picked:", Paper_picks)
```

Fungsi ini memiliki dua parameter: denominations (list yang berisi nilai-nilai uang kertas yang tersedia) dan withdraw_amount (jumlah uang yang akan ditarik). List dp diinisialisasi dengan nilai float('inf') (tak terhingga), dimana setiap indeks merepresentasikan jumlah uang yang akan ditarik dari 0 hingga

withdraw_amount. List ini akan menyimpan jumlah minimum uang kertas yang diperlukan untuk setiap jumlah penarikan. Kasus dasar ditetapkan dengan dp[0] = 0, menunjukkan bahwa tidak diperlukan uang kertas untuk jumlah penarikan sebesar 0. List Paper_picks diinisialisasi untuk menyimpan uang kertas yang dipilih untuk setiap jumlah penarikan. Program ini kemudian melakukan iterasi pada setiap nilai uang kertas dalam list denominations.

Jika jumlah minimum uang kertas untuk withdraw_amount adalah float('inf'), yang menunjukkan bahwa tidak mungkin menarik jumlah tersebut dengan menggunakan nilai uang kertas yang tersedia, fungsi ini mencetak pesan yang menyatakan bahwa jumlah tersebut tidak dapat ditarik. Terakhir, fungsi ini juga mengembalikan Paper_picks[withdraw_amount], yaitu list uang kertas yang dipilih untuk mencapai jumlah minimum uang kertas untuk withdraw_amount.

```
input the withdraw_amount : 1025
Minimum number of papers needed: 12
denominations picked: [5, 20, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100]
```

```
input the withdraw_amount : 144
You can't withdraw that amount
Minimum number of papers needed: None
denominations picked: []
```

B. ANALISIS

1. Greedy

Kompleksitas waktu dari Program tersebut adalah O(n), dimana n adalah jumlah denominasi uang yang tersedia.

Pada Program tersebut, terdapat satu perulangan for yang iterasi sebanyak jumlah denominasi uang yang tersedia. Dalam setiap iterasi, perulangan while digunakan untuk mencari kombinasi denominasi uang yang sesuai dengan jumlah penarikan yang diinginkan.

Karena denominasi uang telah diurutkan secara terbalik (reverse=True), maka iterasi while hanya akan berjalan saat nilai amount masih lebih besar atau sama dengan denominasi uang yang sedang diproses. Ketika iterasi while berjalan, jumlah uang (amount) akan dikurangi dengan denominasi uang yang diproses, dan denominasi uang tersebut akan ditambahkan ke daftar change.

Karena setiap denominasi uang hanya diambil satu kali dalam iterasi for, kompleksitas waktu Program ini bergantung pada jumlah denominasi uang yang tersedia, yaitu O(n).

Algoritma Greedy di sini sudah bisa dikatakan optimal dikarenakan Denominasi sudah dari yang terbesar ke yang terkecil dan setiap denominasi merupakan dari denominasi sebelumnya, yaitu 100, 50, 20, 10, 5. Sehingga hasilnya akan selalu optimal.

Ukuran input:

```
Please input the amount you want to withdraw : 15
Uang: 10 5
Banyak lembaran :  2
Total uang :  15 Ribu Rupiah
```

```
Please input the amount you want to withdraw : 55
Uang: 50 5
Banyak lembaran :  2
Total uang :  55 Ribu Rupiah
```

```
Please input the amount you want to withdraw : 135
Uang: 100 20 10 5
Banyak lembaran :  4
Total uang :  135 Ribu Rupiah
```

```
Please input the amount you want to withdraw : 285
Uang: 100 100 50 20 10 5
Banyak lembaran :  6
Total uang :  285 Ribu Rupiah
```

```
Please input the amount you want to withdraw : 1085
Uang: 100 100 100 100 100 100 100 100 100 100 50 20 10 5
Banyak lembaran :  14
Total uang :  1085 Ribu Rupiah
```

2. Dynamic Programming

Kompleksitas waktu dari program yang mengimplementasikan pendekatan dynamic programming menggunakan pendekatan bottom-up, adalah O(n * m), di mana n adalah jumlah denominasi uang yang tersedia dan m adalah jumlah penarikan yang diinginkan.

Hal ini disebabkan oleh dua tingkat perulangan dalam Program. Perulangan luar iterasi melalui setiap denominasi uang, yang berjalan sebanyak n kali. Sedangkan perulangan dalam iterasi melalui rentang dari coin hingga withdraw_amount + 1, yang berjalan sebanyak m kali.

Dalam setiap iterasi perulangan dalam, terdapat operasi pengecekan kondisi dan pembaruan nilai di dalamnya. Oleh karena itu, kompleksitas waktu keseluruhan Program adalah O(n * m).

Algoritma di sini juga sudah optimal dengan menginisialisasi array dp dengan nilai tak terhingga dan memperbarui nilainya berdasarkan nilai kombinasi uang kertas yang lebih efisien, algoritma dapat menemukan solusi optimal.

Ukuran input:

```
input the withdraw_amount : 15
Minimum number of papers needed: 2
denominations picked: 5 10
```

```
input the withdraw_amount : 55
Minimum number of papers needed: 2
denominations picked: 5 50
```

```
input the withdraw_amount : 135
Minimum number of papers needed: 4
denominations picked: 5 10 20 100
```

```
input the withdraw_amount : 285
Minimum number of papers needed: 6
denominations picked: 5 10 20 50 100 100
```

```
input the withdraw_amount : 1085
Minimum number of papers needed: 14
denominations picked: 5 10 20 50 100 100 100 100 100 100 100 100 100 100
```

KESIMPULAN

Algoritma Greedy dan Dynamic Programming dapat diterapkan sebagai solusi untuk menyelesaikan permasalahan uang kembalian pada mesin ATM. Dalam algoritma Greedy, mesin ATM akan memilih denominasi uang kertas terbesar secara berurutan untuk memberikan kembalian dengan jumlah yang minimal. Sementara itu, algoritma Dynamic Programming akan digunakan untuk mencari kombinasi uang kertas yang menghasilkan jumlah kembalian minimal secara keseluruhan dengan memanfaatkan sifat overlapping subproblem dan optimal substructure. Dengan menerapkan kedua algoritma ini, diharapkan mesin ATM dapat memberikan uang kembalian dalam bentuk uang kertas dengan jumlah yang minimal, meningkatkan penggunaan uang secara efisien dan efektif dalam mesin ATM pada transaksi penarikan uang.

DAFTAR PUSTAKA

[1] Gull, Judith. "Greedy, Teile und Herrsche." Zürich.
[2] Bellman, Richard. "Dynamic Programming." Princeton University Press, 1957.

LAMPIRAN

Greedy:
https://colab.research.google.com/drive/1rwtPE9cLKM8vww3WZmuFKfYy1UsqK08A?usp=sharing

Dynamic Programming:
https://colab.research.google.com/drive/1O1arBz4g2qZRJMtN-bfV6Qmeu-Lz4Lkk?usp=sharing