Analisis Komparatif Algoritma Greedy dan Dynamic Programming pada Strategi Buy/Sell Saham Perbankan Berbasis Data Historis Likuiditas Pasar

Nicholas Luis Chandra - 13524105
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: nicholasluischandra@gmail.com, 13524105@std.stei.itb.ac.id

***Abstract*—Pasar saham merupakan salah satu instrumen investasi yang memiliki karakter dinamis. Dalam aktivitas perdagangan saham, investor perlu menentukan keputusan beli, jual, tahan, atau berpindah saham berdasarkan informasi yang tersedia. Keputusan tersebut dapat dibantu dengan pendekatan algoritmik, salah satunya melalui strategi algorithmic trading. Makalah ini membahas perbandingan algoritma Greedy dan Program Dinamis dalam strategi buy/sell saham perbankan Indonesia berbasis data historis. Data yang digunakan adalah data historis saham BBCA, BBRI, BMRI, dan BBNI pada periode 2021–2025. Harga penutupan digunakan sebagai dasar transaksi, sedangkan volume transaksi digunakan sebagai indikator likuiditas.**

Keywords—Algoritma Greedy, Program Dinamis, Saham Perbankan, Algorithmic Trading

I. PENDAHULUAN

Pasar saham merupakan tempat terjadinya aktivitas jual beli kepemilikan perusahaan dalam bentuk saham. Melalui pasar saham, investor dapat memperoleh keuntungan dari kenaikan harga saham maupun dari pembagian dividen. Namun, pasar saham juga memiliki risiko karena harga saham dapat berubah akibat berbagai faktor, seperti kondisi ekonomi, kinerja perusahaan, sentimen pasar, tingkat likuiditas, dan perilaku investor. Dalam praktiknya, pengambilan keputusan di pasar saham tidak selalu mudah. Investor harus menentukan kapan waktu yang tepat untuk membeli, menjual, menahan, atau berpindah dari satu saham ke saham lain. Keputusan tersebut menjadi semakin kompleks karena data pasar terus berubah dan jumlah informasi yang perlu dianalisis cukup besar. Oleh karena itu, pendekatan berbasis algoritma dapat digunakan untuk membantu proses pengambilan keputusan.

Salah satu pendekatan yang dapat digunakan adalah algorithmic trading, yaitu penggunaan algoritma untuk membantu menentukan keputusan jual/beli berdasarkan aturan tertentu. Penggunaan algoritma dalam strategi perdagangan saham menarik karena setiap algoritma memiliki cara kerja dan karakteristik yang berbeda. Algoritma Greedy, misalnya, mengambil keputusan berdasarkan pilihan terbaik pada saat itu. Pendekatan ini sederhana dan cepat, tetapi tidak selalu menghasilkan solusi terbaik secara keseluruhan. Sebaliknya, Program Dinamis dapat digunakan untuk mengevaluasi banyak kemungkinan keputusan dan mencari hasil yang lebih optimal, meskipun membutuhkan proses komputasi yang lebih kompleks.

Berdasarkan hal tersebut, penelitian ini membahas perbandingan algoritma Greedy dan Program Dinamis dalam strategi buy/sell saham perbankan. Perbandingan ini dilakukan untuk melihat bagaimana perbedaan cara kerja algoritma memengaruhi hasil akhir portofolio, risiko, jumlah transaksi, dan waktu eksekusi. Dengan demikian, penelitian ini diharapkan dapat memberikan gambaran mengenai trade-off antara kesederhanaan algoritma dan kemampuan optimasi dalam konteks strategi perdagangan saham berbasis data historis.

II. LANDASAN TEORI

A. Algoritma Greedy

1) Definisi

Algoritma Greedy merupakan metode atau algoritma yang umumnya digunakan untuk menyelesaikan persoalan optimasi. Persoalan optimasi merupakan persoalan untuk memperoleh solusi terbaik berdasarkan suatu aturan tertentu. Solusi terbaik tersebut dapat berupa nilai maksimum ataupun nilai minimum, bergantung pada fungsi objektif yang ingin dicapai. Secara umum, persoalan optimasi terbagi atas dua jenis, yaitu:

- Maksimasi (maximization), yaitu persoalan untuk mencari nilai paling besar dari sekumpulan pilihan yang tersedia. Contohnya adalah memaksimumkan keuntungan, nilai, reward, atau jumlah aktivitas yang dapat dipilih.
- Minimisasi (minimization), yaitu persoalan untuk mencari nilai paling kecil dari sekumpulan pilihan yang tersedia. Contohnya adalah meminimumkan biaya, waktu, jarak, jumlah koin, atau total bobot.

2) Sifat-sifat Algoritma Greedy

a) Greedy Choice Property

Greedy choice property adalah sifat yang menyatakan bahwa solusi optimal dapat dibangun dengan terlebih dahulu mengambil pilihan lokal terbaik. Artinya, pada setiap langkah, algoritma memilih kandidat yang dianggap paling baik berdasarkan fungsi seleksi tertentu.

b) Optimal Substrcture

Optimal substructure adalah sifat yang menyatakan bahwa solusi optimal dari suatu persoalan dapat dibentuk dari solusi optimal sub-persoalannya. Setelah algoritma mengambil suatu pilihan greedy, persoalan utama akan berkurang menjadi sub-persoalan yang lebih kecil. Jika solusi optimal dari sub-persoalan tersebut dapat digabungkan dengan pilihan greedy sebelumnya untuk membentuk solusi optimal persoalan awal, maka persoalan tersebut memiliki optimal substructure.

c) Solusi Akhir Tidak Selalu Optimal

Walaupun Algoritma Greedy sederhana dan efisien, solusi akhirnya tidak selalu optimal. Jika hasilnya tidak optimal, maka solusi tersebut disebut solusi sub-optimum atau pseudo-optimum. Hal tersebut dapat terjadi karena dua alasan utama. Pertama, Algoritma Greedy tidak mengevaluasi seluruh kemungkinan solusi. Kedua, pemilihan fungsi seleksi yang kurang tepat dapat menyebabkan algoritma terjebak pada pilihan lokal yang terlihat baik, tetapi mencegah tercapainya solusi global yang optimal.

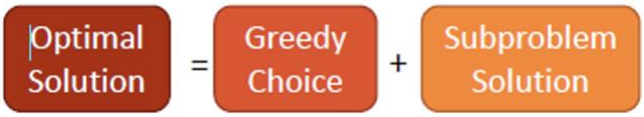

Fig 1. Visualisasi sifat Algoritma Greedy

Source : https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2025-2026/04-Algoritma-Greedy-(2026)-Bag1.pdf

3) Komponen Algoritma Greedy

a) Himpunan Kandidat (C)

Himpunan kandidat adalah himpunan yang berisi semua elemen yang masih mungkin dipilih untuk membentuk solusi. Kandidat dapat berupa koin, aktivitas, job, sisi graf, simpul graf, objek knapsack, karakter, atau elemen lain sesuai jenis persoalan.

b) Himpunan Solusi (S)

Himpunan solusi adalah himpunan yang berisi kandidat-kandidat yang telah dipilih oleh algoritma. Pada awalnya, himpunan solusi biasanya kosong. Seiring berjalannya algoritma, kandidat yang layak akan dimasukkan ke dalam himpunan solusi.

c) Fungsi Solusi

Fungsi solusi adalah fungsi yang memeriksa apakah himpunan solusi yang sudah terbentuk telah menjadi solusi lengkap.

d) Fungsi seleksi

Fungsi seleksi adalah fungsi yang memilih kandidat terbaik dari himpunan kandidat berdasarkan strategi greedy tertentu. Fungsi ini bersifat heuristik, artinya pemilihan kandidat dilakukan berdasarkan aturan praktis yang dianggap baik untuk mencapai tujuan optimasi.

e) Fungsi Kelayakan

Fungsi kelayakan adalah fungsi yang memeriksa apakah kandidat yang dipilih dapat dimasukkan ke dalam himpunan solusi tanpa melanggar batasan persoalan. Jika kandidat tersebut layak, maka kandidat dimasukkan ke dalam solusi. Jika tidak layak, kandidat ditolak.

f) Fungsi Objektif

Fungsi objektif adalah fungsi yang menentukan ukuran optimalitas solusi. Fungsi ini dapat berupa fungsi maksimasi atau minimisasi. Pada persoalan maksimasi, fungsi objektif digunakan untuk memperoleh nilai sebesar mungkin. Pada persoalan minimisasi, fungsi objektif digunakan untuk memperoleh nilai sekecil mungkin.

4) Pseudocode Greedy

```
function Greedy (C : HimpunanKandidat) → Himpunan_Solusi
{Menghasilkan solusi optimal dari permasalahan dengan
strategi greedy }

Deklarasi
    x : Kandidat
    S : Himpunan_Solusi

Algoritma:
    S ← {}
    while (not IsSolution(S)) and (C ≠ {}) do
        x ← SelectBest(C)
        C ← C - {x}
        if IsFeasible(S U {x}) then
            S ← S U {x}
        endif
    endwhile
    if IsSolution(S) then
        return S
    else
        write('Solusi tidak ditemukan')
    endif
```

B. Program Dinamis

1) Definisi

Program Dinamis atau *Dynamic Programming* merupakan metode pemecahan masalah optimasi dengan cara menguraikan persoalan menjadi beberapa tahapan atau stage. Pada setiap tahap, terdapat keputusan yang harus diambil, sehingga solusi akhir dari persoalan dapat dipandang sebagai rangkaian keputusan yang saling berkaitan.

Perbedaan utama antara Algoritma Greedy dan Program Dinamis terletak pada cara mempertimbangkan keputusan. Algoritma Greedy hanya membentuk satu rangkaian keputusan berdasarkan pilihan terbaik pada setiap langkah. Sebaliknya, Program Dinamis mempertimbangkan lebih dari

satu rangkaian keputusan dengan menyimpan hasil perhitungan sub-persoalan agar dapat digunakan kembali. Oleh karena itu, Program Dinamis lebih sistematis dalam mencari solusi optimal dibandingkan Greedy, terutama pada persoalan yang pilihan lokal terbaiknya belum tentu menghasilkan solusi global terbaik.

2) Sifat-sifat Program Dinamis

a) Prinsip Optimalitas

Prinsip utama dalam Program Dinamis adalah prinsip optimalitas. Prinsip ini menyatakan bahwa jika solusi total suatu persoalan bersifat optimal, maka bagian dari solusi tersebut sampai tahap tertentu juga harus optimal. Artinya, apabila suatu persoalan diselesaikan dari tahap k ke tahap k + 1, hasil optimal dari tahap sebelumnya dapat digunakan langsung untuk membentuk solusi pada tahap berikutnya. Dengan demikian, algoritma tidak perlu menghitung ulang seluruh proses dari tahap awal.

b) Optimal Substructure

Program Dinamis dapat digunakan jika persoalan memiliki struktur solusi optimal atau optimal substructure. Artinya, solusi optimal dari persoalan utama dapat dibentuk dari solusi optimal beberapa sub-persoalan yang lebih kecil.

c) Memoization

Memoisasi merupakan teknik menyimpan nilai hasil perhitungan suatu fungsi atau sub-persoalan ke dalam tabel. Tujuannya adalah agar nilai tersebut dapat digunakan kembali ketika dibutuhkan. Dengan memoisasi, Program Dinamis dapat menghindari perhitungan berulang. Hal ini membuat penyelesaian persoalan menjadi lebih efisien dibandingkan metode rekursif biasa yang menghitung sub-persoalan yang sama berkali-kali.

3) Komponen Program Dinamis

a) Tahap (Stage)

Tahap adalah bagian-bagian penyelesaian persoalan. Setiap tahap merepresentasikan satu langkah pengambilan keputusan

b) Status (State)

Status adalah keadaan yang mungkin terjadi pada suatu tahap. Status menjadi informasi penting untuk menentukan keputusan yang dapat diambil.

c) Peubah Keputusan

Peubah keputusan adalah pilihan yang diambil pada suatu tahap. Keputusan ini menentukan perubahan dari satu status ke status berikutnya.

d) Fungsi Objektif

Fungsi objektif adalah fungsi yang ingin dioptimalkan. Fungsi ini dapat berupa maksimasi atau minimisasi.

e) Relasi Rekurens

Relasi rekurens adalah rumus yang menghubungkan nilai optimal pada suatu tahap dengan nilai optimal tahap sebelumnya atau tahap berikutnya. Relasi ini menjadi inti dari Program Dinamis karena menentukan bagaimana tabel diisi.

f) Basis

Basis adalah kondisi awal dari perhitungan. Basis diperlukan agar relasi rekurens memiliki titik awal perhitungan.

g) Table Program Dinamis

Tabel digunakan untuk menyimpan hasil perhitungan nilai optimal setiap status pada setiap tahap. Tabel ini memungkinkan algoritma mengambil kembali nilai yang sudah dihitung tanpa melakukan perhitungan ulang.

h) Rekonstruksi Solusi

Rekonstruksi solusi adalah proses menelusuri kembali keputusan-keputusan yang menghasilkan nilai optimal. Tahap ini bersifat opsional, tetapi penting jika yang dibutuhkan bukan hanya nilai optimal, melainkan juga bentuk solusi optimalnya.

4) Pendekatan Program Dinamis

a) Program Dinamis Maju

Program Dinamis maju adalah pendekatan yang melakukan perhitungan dari tahap awal menuju tahap akhir. Perhitungan dimulai dari tahap 1, kemudian dilanjutkan ke tahap 2, tahap 3, dan seterusnya sampai tahap n. Pendekatan ini cocok digunakan ketika nilai optimal pada tahap berikutnya dapat dihitung dari nilai optimal tahap sebelumnya.

b) Program Dinamis Mundur

Program Dinamis mundur adalah pendekatan yang melakukan perhitungan dari tahap akhir menuju tahap awal. Perhitungan dimulai dari tahap n, kemudian mundur ke tahap n - 1, tahap n - 2, dan seterusnya sampai tahap 1. Pendekatan ini cocok digunakan ketika lebih mudah menghitung nilai optimal dari kondisi akhir terlebih dahulu, kemudian menelusuri kembali ke kondisi awal.

5) Pseudocode Program Dinamis

```
function DynamicProgramming(input) → SolusiOptimal
{Menghasilkan nilai optimal dan, jika diperlukan, bentuk solusi optimal}

Deklarasi:
DP : tabel nilai optimal
Choice : tabel keputusan optimal
k : integer
s : state
d : decision

Algoritma:
for setiap tahap k do
    for setiap status s pada tahap k do
        DP[k][s] ← nilai awal

        for setiap keputusan d yang layak pada status s do
```

```
            nilai ← nilai dari keputusan d + nilai optimal
sub-persoalan

            if nilai lebih baik daripada DP[k][s] then
                DP[k][s] ← nilai
                Choice[k][s] ← d
            endif
        endfor
    endfor
endfor

if rekonstruksi solusi diperlukan then
    telusuri tabel Choice untuk memperoleh keputusan
optimal
endif

return solusi optimal
```

C. Dasar-Dasar Investasi dan Pasar Modal

1) Investasi

Investasi merupakan kegiatan menempatkan dana pada suatu instrumen tertentu dengan harapan memperoleh keuntungan pada masa mendatang. Dalam pasar modal, salah satu instrumen investasi yang umum digunakan adalah saham. Keuntungan dari saham dapat diperoleh melalui capital gain, yaitu selisih positif antara harga jual dan harga beli, serta dividen, yaitu pembagian laba perusahaan kepada pemegang saham.

2) Pasar Modal

Pasar modal merupakan tempat terjadinya aktivitas perdagangan instrumen keuangan jangka panjang, seperti saham dan obligasi. Pasar modal mempertemukan pihak yang membutuhkan dana, seperti perusahaan, dengan pihak yang memiliki dana, yaitu investor. Dalam konteks saham, harga terbentuk melalui mekanisme permintaan dan penawaran. Jika permintaan terhadap suatu saham meningkat, harga saham cenderung naik. Sebaliknya, jika tekanan jual lebih besar, harga saham cenderung turun.

Fig 2. Bursa Efek Indonesia
Source : https://www.mandirisekuritas.co.id/id/siaran-pers-artikel/artikel/pengertian-pasar-modal

3) Saham dan Strategi Buy/Sell

Saham adalah bukti kepemilikan atas suatu perusahaan. Investor yang membeli saham memiliki peluang memperoleh keuntungan dari kenaikan harga saham maupun pembagian dividen.

Strategi buy/sell saham adalah aturan untuk menentukan kapan saham dibeli, dijual, atau ditahan. Keputusan buy dilakukan ketika terdapat peluang kenaikan harga, sedangkan keputusan sell dilakukan ketika terdapat sinyal penurunan harga atau ketika keuntungan ingin direalisasikan.

.

4) Likuiditas Pasar

Likuiditas pasar menunjukkan seberapa mudah suatu saham dapat dibeli atau dijual tanpa menyebabkan perubahan harga yang besar. Saham yang likuid umumnya memiliki volume transaksi tinggi, nilai transaksi besar, dan frekuensi transaksi yang aktif. Likuiditas penting dalam strategi buy/sell karena keputusan transaksi harus realistis untuk dieksekusi. Saham dengan likuiditas rendah dapat menyulitkan investor untuk membeli atau menjual pada harga yang diinginkan.

D. Saham Perbankan

1) Definisi

Saham perbankan adalah saham yang diterbitkan oleh perusahaan sektor perbankan dan diperdagangkan di pasar modal. Perusahaan perbankan memiliki kegiatan utama menghimpun dana dari masyarakat dan menyalurkannya kembali dalam bentuk kredit atau pembiayaan.

2) Karakteristik Saham Perbankan

Saham perbankan memiliki karakteristik yang dipengaruhi oleh kinerja bank dan kondisi ekonomi. Beberapa faktor yang dapat memengaruhi saham perbankan antara lain pertumbuhan laba, kualitas kredit, kecukupan modal, suku bunga, dan kondisi makroekonomi. Selain itu, saham perbankan, terutama bank-bank besar, umumnya memiliki aktivitas perdagangan yang cukup tinggi. Hal ini membuat saham perbankan relevan untuk dianalisis berdasarkan data historis likuiditas pasar.

Fig 3. Saham Perbankan PT Bank Central Asia Tbk.
Source : https://id.tradingview.com/chart/?symbol=BBCA

III. IMPLEMENTASI

A. Deskripsi Data

Penelitian ini menggunakan pendekatan eksperimen untuk membandingkan kinerja algoritma Greedy dan Program Dinamis dalam strategi buy/sell saham perbankan Indonesia.

Data yang digunakan adalah data historis harian saham BBCA, BBRI, BMRI, dan BBNI pada periode 2021–2025. Data diperoleh dari Yahoo Finance dengan bantuan library yfinance pada bahasa pemrograman Python.

Data historis yang digunakan meliputi tanggal perdagangan, harga pembukaan, harga tertinggi, harga terendah, harga penutupan, dan volume transaksi harian. Dalam penelitian ini, harga penutupan atau Close digunakan sebagai dasar harga transaksi, sedangkan Volume digunakan sebagai indikator likuiditas pasar. Volume dipilih karena menunjukkan tingkat aktivitas perdagangan suatu saham. Saham dengan volume lebih tinggi dianggap lebih likuid karena lebih banyak diperdagangkan oleh pelaku pasar.

Parameter simulasi dari implementasi adalah sebagai berikut:

Parameter	Nilai
Modal awal	Rp100.000.000
Saham	BBCA, BBRI, BMRI, BBNI
Periode	2021–2025
Sumber data	Yahoo Finance
Fee transaksi	Tidak digunakan

B. Implementasi Algoritma

a) Data Collecting dan Cleaning

Tahap pertama dalam penelitian ini adalah pengambilan dan pembersihan data. Data saham diambil menggunakan library yfinance berdasarkan daftar ticker saham perbankan yang telah ditentukan. Rentang data yang digunakan dimulai dari 1 Januari 2021 sampai 1 Januari 2026 agar seluruh data perdagangan tahun 2025 tercakup.

```
import yfinance as yf
import pandas as pd

tickers = ["BBCA.JK", "BBRI.JK", "BMRI.JK", "BBNI.JK"]

start_date = "2021-01-01"
end_date = "2026-01-01"

raw_data = yf.download(
    tickers=tickers,
    start=start_date,
    end=end_date,
    interval="1d",
    group_by="ticker",
    auto_adjust=False,
    progress=True
)
```

Setelah data berhasil diperoleh, data dari masing-masing saham digabungkan ke dalam satu tabel. Kolom yang digunakan dalam penelitian ini adalah Date, Open, High, Low, Close, Volume, dan Ticker.

```
all_data = []

for ticker in tickers:
    df_ticker = raw_data[ticker].copy()
    df_ticker = df_ticker.reset_index()
    df_ticker["Ticker"] = ticker
    all_data.append(df_ticker)

stock_data = pd.concat(all_data, ignore_index=True)
```

Tahap berikutnya adalah pembersihan data. Baris yang memiliki nilai kosong pada kolom penting dihapus. Data juga diurutkan berdasarkan ticker dan tanggal agar perhitungan return harian dapat dilakukan dengan benar.

```
stock_data = stock_data.dropna(
    subset=["Date", "Open", "High", "Low", "Close",
"Volume"]
)

stock_data = stock_data[stock_data["Volume"] > 0]

stock_data = stock_data.sort_values(
    ["Ticker", "Date"]
).reset_index(drop=True)
```

Setelah data bersih, dilakukan perhitungan return harian. Return harian digunakan oleh algoritma Greedy untuk menentukan apakah suatu saham layak dipilih pada hari berikutnya.

```
stock_data["Daily_Return"] =
stock_data.groupby("Ticker")["Close"].pct_change()
```

Secara umum, rumus return harian adalah:

Return Harian = (Close Hari Ini - Close Hari Sebelumnya) / Close Hari Sebelumnya

Setelah proses cleaning dan feature engineering selesai, data disimpan ke dalam file CSV agar dapat digunakan kembali pada proses analisis.

Agar data lebih mudah digunakan dalam implementasi algoritma, data kemudian diubah menjadi bentuk tabel pivot. Tabel price menyimpan harga penutupan masing-masing saham pada setiap tanggal, sedangkan tabel volume menyimpan volume transaksi masing-masing saham pada setiap tanggal.

```
price = stock_data.pivot(
     index="Date",
     columns="Ticker",
     values="Close"
 ).sort_index()

volume = stock_data.pivot(
     index="Date",
     columns="Ticker",
     values="Volume"
```

```
).sort_index()

daily_return = stock_data.pivot(
      index="Date",
      columns="Ticker",
      values="Daily_Return"
).sort_index()
```

b) Implementasi Algoritma Greedy

Algoritma Greedy merupakan algoritma yang mengambil keputusan terbaik secara lokal pada setiap langkah. Dalam konteks penelitian ini, Greedy menentukan keputusan buy/sell harian berdasarkan informasi hari sebelumnya. Algoritma tidak mengevaluasi seluruh kemungkinan keputusan sampai akhir periode, melainkan hanya memilih saham yang tampak paling baik pada saat keputusan dibuat.

Aturan Greedy yang digunakan adalah sebagai berikut.

a. Pada hari ke-t, algoritma melihat data return dan volume pada hari ke-(t-1).
b. Saham yang menjadi kandidat beli adalah saham yang memiliki return positif pada hari sebelumnya.
c. Dari kandidat tersebut, dipilih saham dengan volume transaksi terbesar.
d. Jika portofolio sedang dalam posisi cash, algoritma membeli saham terpilih.
e. Jika portofolio sedang memegang saham lain, algoritma menjual saham lama lalu membeli saham baru.
f. Jika tidak ada saham dengan return positif, algoritma menjual posisi yang sedang dimiliki dan kembali ke cash.

Implementasinya dalam Python adalah sebagai berikut :

```
INITIAL_CAPITAL = 100_000_000
LOT_SIZE = 100

def run_greedy_strategy(price, volume, daily_return,
tickers):
    dates = list(price.index)

    cash = INITIAL_CAPITAL
    holding_ticker = None
    lots = 0

    portfolio_rows = []
    trade_rows = []

    for i, current_date in enumerate(dates):
        price_today = price.loc[current_date]

        if i == 0:
            action = "INIT_CASH"

        else:
            prev_date = dates[i - 1]
            prev_returns = daily_return.loc[prev_date]
            prev_volumes = volume.loc[prev_date]

            positive_candidates = [
                t for t in tickers
                if prev_returns[t] > 0 and prev_volumes[t] >
0
            ]

            if len(positive_candidates) == 0:
                selected_ticker = None
            else:
                selected_ticker = max(
                    positive_candidates,
                    key=lambda t: prev_volumes[t]
                )

            if selected_ticker is None:
                if holding_ticker is not None:
                    sell_price =
int(price_today[holding_ticker])
                    cash += lots * LOT_SIZE * sell_price

                    trade_rows.append({
                        "Date": current_date,
                        "Action": "SELL",
                        "Ticker": holding_ticker,
                        "Price": sell_price,
                        "Lots": lots,
                        "Shares": lots * LOT_SIZE,
                        "Cash_After": cash,
                        "Reason": "Tidak ada kandidat return
positif H-1"
                    })

                    holding_ticker = None
                    lots = 0
                    action = "SELL_TO_CASH"
                else:
                    action = "HOLD_CASH"

            else:
                if holding_ticker == selected_ticker:
                    action = "HOLD_STOCK"

                else:
                    if holding_ticker is not None:
                        sell_price =
int(price_today[holding_ticker])
                        cash += lots * LOT_SIZE * sell_price

                        trade_rows.append({
                            "Date": current_date,
                            "Action": "SELL",
                            "Ticker": holding_ticker,
                            "Price": sell_price,
                            "Lots": lots,
                            "Shares": lots * LOT_SIZE,
                            "Cash_After": cash,
                            "Reason": "Switch ke saham
dengan volume H-1 terbesar"
                        })

                        holding_ticker = None
                        lots = 0

                    buy_price =
int(price_today[selected_ticker])
                    lot_cost = buy_price * LOT_SIZE
                    buy_lots = cash // lot_cost
                    remaining_cash = cash - buy_lots *
lot_cost

                    if buy_lots > 0:
                        cash = remaining_cash
                        holding_ticker = selected_ticker
                        lots = buy_lots
```

```
                        trade_rows.append({
                            "Date": current_date,
                            "Action": "BUY",
                            "Ticker": selected_ticker,
                            "Price": buy_price,
                            "Lots": buy_lots,
                            "Shares": buy_lots * LOT_SIZE,
                            "Cash_After": cash,
                            "Reason": "Return H-1 positif
dan volume H-1 terbesar"
                        })

                        action = "BUY_OR_SWITCH"
                    else:
                        action = "INSUFFICIENT_CASH"

        if holding_ticker is None:
            portfolio_value = cash
        else:
            portfolio_value = cash + lots * LOT_SIZE *
int(price_today[holding_ticker])

        portfolio_rows.append({
            "Date": current_date,
            "Strategy": "Greedy",
            "Portfolio_Value": portfolio_value,
            "Cash": cash,
            "Holding_Ticker": holding_ticker if
holding_ticker is not None else "CASH",
            "Lots": lots,
            "Shares": lots * LOT_SIZE,
            "Action": action
        })

    portfolio_df = pd.DataFrame(portfolio_rows)
    trade_df = pd.DataFrame(trade_rows)

    return portfolio_df, trade_df
```

c) Implementasi Program Dinamis

Program Dinamis atau Dynamic Programming digunakan untuk mencari lintasan keputusan yang menghasilkan nilai akhir portofolio maksimum selama periode historis. Berbeda dengan Greedy yang hanya memilih keputusan terbaik secara lokal, Program Dinamis mengevaluasi berbagai kemungkinan kondisi portofolio dari awal sampai akhir periode. Dalam penelitian ini, Program Dinamis digunakan untuk menentukan keputusan buy, sell, hold, atau switch antar saham berdasarkan nilai portofolio terbaik yang dapat dicapai pada setiap state. State menggambarkan kondisi portofolio pada suatu hari. State dapat berupa posisi cash atau posisi sedang memegang salah satu saham, yaitu BBCA, BBRI, BMRI, atau BBNI.

State dalam Program Dinamis terdiri dari tiga komponen, yaitu:

a. ticker : saham yang sedang dipegang;
b. lots : jumlah lot saham yang dimiliki;
c. cash : sisa uang tunai yang tersedia.

Jika ticker = None, maka portofolio berada dalam posisi cash. Namun, jika ticker berisi kode saham tertentu, maka portofolio sedang memegang saham tersebut.

Aturan Program Dinamis yang digunakan adalah sebagai berikut :

a. Pada awal simulasi, portofolio berada pada posisi cash sebesar Rp100.000.000.
b. Pada setiap hari perdagangan, algoritma membaca harga penutupan setiap saham.
c. Untuk setiap state yang mungkin, algoritma mengevaluasi semua aksi yang dapat dilakukan.
d. Jika portofolio berada dalam posisi cash, maka aksi yang mungkin adalah tetap cash atau membeli salah satu saham.
e. Jika portofolio sedang memegang saham, maka aksi yang mungkin adalah tetap memegang saham, menjual saham, atau berpindah ke saham lain.
f. Setiap aksi akan menghasilkan state baru dengan nilai cash, jumlah lot, dan saham yang berbeda.
g. Nilai setiap state dihitung berdasarkan cash ditambah nilai saham yang sedang dipegang.
h. Jika terdapat beberapa cara untuk mencapai state yang sama, algoritma menyimpan state dengan nilai portofolio terbaik.
i. Untuk mengurangi jumlah state, dilakukan pruning terhadap state yang terdominasi.
j. Setelah seluruh hari perdagangan dievaluasi, algoritma memilih state dengan nilai portofolio terbesar pada hari terakhir.
k. Dari state akhir terbaik tersebut, algoritma merekonstruksi lintasan keputusan untuk mengetahui urutan buy, sell, hold, dan switch yang menghasilkan nilai akhir maksimum.

Nilai portofolio pada setiap state dihitung menggunakan rumus berikut.

$$\textit{Nilai Portofolio} = \textit{Cash} + \textit{Lots} \times 100 \times \textit{Harga Close}$$

Program Dinamis dalam penelitian ini menggunakan pendekatan retrospective optimum. Artinya, algoritma mencari lintasan keputusan terbaik berdasarkan seluruh data historis periode 2021–2025. Oleh karena itu, hasil Program Dinamis digunakan sebagai pembanding kemampuan optimasi historis, bukan sebagai strategi trading real-time.

```
def run_dp_strategy(price, volume, tickers):
    dates = list(price.index)

    initial_state = State(
        ticker=None,
        lots=0,
        cash=INITIAL_CAPITAL
    )

    states = {
        initial_state: (INITIAL_CAPITAL, None, "INIT_CASH")
    }

    history = {}

    for current_date in dates:
        price_today = price.loc[current_date]
        volume_today = volume.loc[current_date]
```

```
        next_states = {}

        def add_state(new_state, prev_state, action):
            value = state_value(new_state, price_today)
            old = next_states.get(new_state)

            if old is None or value > old[0]:
                next_states[new_state] = (value, prev_state,
action)

        for state in states.keys():

            if state.ticker is None:
                add_state(state, state, "HOLD_CASH")

                for ticker in tickers:
                    if volume_today[ticker] <= 0:
                        continue

                    buy_price = int(price_today[ticker])
                    lot_cost = buy_price * LOT_SIZE

                    buy_lots = state.cash // lot_cost
                    remaining_cash = state.cash - buy_lots *
lot_cost

                    if buy_lots > 0:
                        new_state = State(
                            ticker=ticker,
                            lots=buy_lots,
                            cash=remaining_cash
                        )

                        add_state(
                            new_state,
                            state,
                            f"BUY {ticker}"
                        )

            else:
                add_state(state, state, "HOLD_STOCK")

                sell_price = int(price_today[state.ticker])
                sell_cash = state.cash + state.lots *
LOT_SIZE * sell_price

                cash_state = State(
                    ticker=None,
                    lots=0,
                    cash=int(sell_cash)
                )

                add_state(
                    cash_state,
                    state,
                    f"SELL {state.ticker}"
                )

                for ticker in tickers:
                    if ticker == state.ticker:
                        continue

                    if volume_today[ticker] <= 0:
                        continue

                    buy_price = int(price_today[ticker])
                    lot_cost = buy_price * LOT_SIZE

                    buy_lots = sell_cash // lot_cost
                    remaining_cash = sell_cash - buy_lots *
lot_cost

                    if buy_lots > 0:
                        new_state = State(
                            ticker=ticker,
                            lots=buy_lots,
                            cash=int(remaining_cash)
                        )

                        add_state(
                            new_state,
                            state,
                            f"SWITCH {state.ticker} TO
{ticker}"
                        )

        next_states = prune_states(next_states)
        history[current_date] = next_states
        states = next_states

    final_date = dates[-1]
    final_price = price.loc[final_date]

    best_final_state = max(
        states.keys(),
        key=lambda state: state_value(state, final_price)
    )

    path = []
    current_state = best_final_state

    for current_date in reversed(dates):
        value, prev_state, action =
history[current_date][current_state]

        path.append({
            "Date": current_date,
            "Strategy": "Dynamic Programming",
            "Portfolio_Value": state_value(current_state,
price.loc[current_date]),
            "Cash": current_state.cash,
            "Holding_Ticker": current_state.ticker if
current_state.ticker is not None else "CASH",
            "Lots": current_state.lots,
            "Shares": current_state.lots * LOT_SIZE,
            "Action": action
        })

        if prev_state is None or current_date == dates[0]:
            break

        current_state = prev_state

    portfolio_df =
pd.DataFrame(reversed(path)).reset_index(drop=True)

    trade_rows = []

    for _, row in portfolio_df.iterrows():
        action = row["Action"]
        date = row["Date"]

        if action.startswith("BUY"):
            ticker = action.split()[1]

            trade_rows.append({
                "Date": date,
                "Action": "BUY",
                "Ticker": ticker,
                "Price": int(price.loc[date, ticker]),
                "Lots": int(row["Lots"]),
```

```
                "Shares": int(row["Shares"]),
                "Cash_After": int(row["Cash"]),
                "Reason": "DP memilih lintasan nilai
portofolio maksimum"
            })

        elif action.startswith("SELL"):
            ticker = action.split()[1]

            trade_rows.append({
                "Date": date,
                "Action": "SELL",
                "Ticker": ticker,
                "Price": int(price.loc[date, ticker]),
                "Lots": np.nan,
                "Shares": np.nan,
                "Cash_After": int(row["Cash"]),
                "Reason": "DP memilih lintasan nilai
portofolio maksimum"
            })

        elif action.startswith("SWITCH"):
            parts = action.split()
            from_ticker = parts[1]
            to_ticker = parts[3]

            trade_rows.append({
                "Date": date,
                "Action": "SWITCH",
                "Ticker": f"{from_ticker}->{to_ticker}",
                "Price": (
                    f"sell {int(price.loc[date,
from_ticker])}; "
                    f"buy {int(price.loc[date, to_ticker])}"
                ),
                "Lots": int(row["Lots"]),
                "Shares": int(row["Shares"]),
                "Cash_After": int(row["Cash"]),
                "Reason": "DP memilih lintasan nilai
portofolio maksimum"
            })

    trade_df = pd.DataFrame(trade_rows)

    return portfolio_df, trade_df
```

C. Hasil

Berdasarkan hasil pengujian, Algoritma Greedy menghasilkan nilai akhir portofolio sebesar Rp42.751.700 dari modal awal Rp100.000.000. Dengan demikian, Greedy mengalami kerugian sebesar Rp57.248.300 atau setara dengan -57,2483%. Nilai max drawdown sebesar -65,6276% menunjukkan bahwa portofolio Greedy mengalami penurunan yang cukup besar dari nilai puncaknya selama periode pengujian.

Hasil tersebut menunjukkan bahwa strategi Greedy yang hanya menggunakan return positif dan volume terbesar pada hari sebelumnya belum mampu menghasilkan performa yang baik. Kenaikan harga pada hari sebelumnya tidak selalu berlanjut pada hari berikutnya, sehingga sinyal momentum sederhana dapat menghasilkan keputusan yang kurang optimal. Selain itu, Greedy menghasilkan 1319 catatan transaksi, yang menunjukkan bahwa strategi ini cukup sering berpindah posisi.

Program Dinamis menghasilkan nilai akhir portofolio sebesar Rp1.376.829.549.058.000 dari modal awal Rp100.000.000, dengan keuntungan sebesar Rp1.376.829.449.058.000 atau total return sebesar 1.376.829.449,058%. Nilai max drawdown sebesar 0,0% menunjukkan bahwa lintasan keputusan yang dipilih tidak mengalami penurunan dari puncak nilai portofolio dalam simulasi historis.

Namun, hasil Program Dinamis perlu ditafsirkan sebagai hasil optimasi historis, bukan sebagai strategi trading aktual. Program Dinamis pada penelitian ini bersifat *retrospective optimum*, yaitu mencari lintasan keputusan terbaik setelah seluruh data historis diketahui. Dengan demikian, algoritma program dinamis memiliki pengetahuan penuh terhadap pergerakan harga dalam periode pengujian. Akibatnya, Program Dinamis dapat memilih kombinasi buy, sell, hold, dan switch yang paling menguntungkan secara historis, tetapi tidak dapat digunakan secara langsung untuk memprediksi harga masa depan. Oleh karena itu, hasil Program Dinamis lebih tepat dipandang sebagai *benchmark* atau batas atas performa historis.

Dari sisi waktu eksekusi, Greedy lebih cepat dengan runtime 0,229019 detik, sedangkan Program Dinamis membutuhkan waktu 3,034614 detik. Perbedaan ini sesuai dengan karakteristik kedua algoritma. Greedy lebih sederhana karena hanya memilih kandidat terbaik pada setiap hari, sedangkan Program Dinamis mengevaluasi lebih banyak state dan transisi untuk memperoleh solusi optimal secara global.

IV. KESIMPULAN

Berdasarkan hasil eksperimen, algoritma Greedy dan Program Dinamis memiliki karakteristik yang berbeda dalam strategi buy/sell saham perbankan berbasis data historis. Greedy lebih sederhana dan cepat karena hanya mengambil keputusan berdasarkan kondisi lokal, yaitu return positif dan volume terbesar pada hari sebelumnya. Namun, pendekatan ini tidak menjamin hasil optimal karena tidak mempertimbangkan dampak keputusan terhadap periode berikutnya.

Program Dinamis menghasilkan performa historis yang jauh lebih tinggi karena mengevaluasi banyak kemungkinan state dan transisi untuk mencari lintasan keputusan terbaik. Walaupun demikian, implementasi Program Dinamis pada penelitian ini belum realistis untuk trading aktual karena bersifat *retrospective optimum*, yaitu menggunakan seluruh data historis untuk menentukan keputusan terbaik. Dengan kata lain, algoritma seolah-olah mengetahui pergerakan harga di masa depan.

Meskipun tidak realistis sebagai strategi trading langsung, Program Dinamis tetap berguna untuk menemukan pola strategi optimal dalam menghadapi pasar berdasarkan data historis. Hasilnya dapat digunakan sebagai pembanding maksimum, bahan evaluasi strategi, serta dasar untuk mengembangkan model yang lebih realistis. Secara

keseluruhan, Greedy unggul dari sisi kesederhanaan dan efisiensi waktu, sedangkan Program Dinamis unggul sebagai alat optimasi historis untuk memahami pola keputusan terbaik pada data masa lalu.

LAMPIRAN

https://github.com/staplesmaster/IF2211-Strategi-Algoritma---Tugas-Makalah

UCAPAN TERIMA KASIH

Penulis mengucap syukur kepada Tuhan Yang Maha Esa karena perlindungan dan berkat-Nya saya dapat menyelesaikan makalah ini. Penulis juga mengucapkan banyak terima kasih kepada Dr. Ir. Rinaldi, M.T., sebagai dosen pengampu mata kuliah IF2211 Strategi Algoritma yang telah memberikan bimbingan dan dukungan sepanjang semester mata kuliah.

REFERENSI

[1] R. Aroussi, "yfinance: Download market data from Yahoo! Finance's API," GitHub repository, 2026. [Online]. Available: https://github.com/ranaroussi/yfinance.

[2] R. Aroussi, "yfinance documentation," yfinance, 2026. [Online]. Available: https://ranaroussi.github.io/yfinance/

[3] R. Munir, "Algoritma Greedy Bagian 1," bahan kuliah IF2211 Strategi Algoritma, Institut Teknologi Bandung, Bandung, Indonesia, 2026

[4] R. Munir, "Algoritma Greedy Bagian 2," bahan kuliah IF2211 Strategi Algoritma, Institut Teknologi Bandung, Bandung, Indonesia, 2026

[5] R. Munir, "Algoritma Greedy Bagian 3," bahan kuliah IF2211 Strategi Algoritma, Institut Teknologi Bandung, Bandung, Indonesia, 2026

[6] R. Munir, "Program Dinamis Bagian 1," bahan kuliah IF2211 Strategi Algoritma, Institut Teknologi Bandung, Bandung, Indonesia, 2026

R. Munir, "Program Dinamis Bagian 2," bahan kuliah IF2211 Strategi Algoritma, Institut Teknologi Bandung, Bandung, Indonesia, 2026

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, 18 Juni 2026

Nicholas Luis Chandra
13524105