

**LAPORAN IMPLEMENTASI PROGRAM 20 SOLVER MENGGUNAKAN
ALGORITMA BRUTE FORCE**



DISUSUN OLEH

MOHAMMAD ADZKA CROSAMER

L0123083

NAUFAL NARENDRO KUSUMO

L0123107

**PROGRAM STUDI INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI DAN SAINS DATA
UNIVERSITAS SEBELAS MARET**

2022

PENDAHULUAN

Dalam pemrograman, salah satu tantangan yang sering dihadapi adalah menyelesaikan masalah aritmatika dengan cara mencoba semua kemungkinan operasi yang tersedia. Salah satu contoh penerapannya adalah dalam masalah "20 Solver", di mana kita diberikan empat angka, dan tugasnya adalah mencari kombinasi operasi (+, -, *, /) yang menghasilkan nilai akhir 20.

Pada tugas ini, kami menggunakan pendekatan algoritma *brute force* untuk menyelesaikan masalah tersebut. Algoritma *brute force* adalah metode yang mengeksplor semua kemungkinan kombinasi tanpa memperhatikan efisiensi, sehingga semua solusi yang memungkinkan dapat ditemukan. Meskipun tidak optimal dalam hal waktu eksekusi untuk input yang besar, metode ini memastikan bahwa setiap kombinasi dieksplorasi dan solusi yang benar dapat ditemukan.

Tujuan dari laporan ini adalah untuk menjelaskan bagaimana algoritma *brute force* diterapkan untuk menyelesaikan masalah 20 Solver. Diharapkan setelah menyelesaikan masalah ini, kami dapat lebih memahami cara kerja algoritma *brute force* serta tantangan yang dihadapi dalam menangani masalah kombinatorik. Selain itu, kami juga diharapkan mampu menerapkan pendekatan serupa dalam masalah-masalah lain yang membutuhkan eksplorasi semua kemungkinan solusi, sekaligus meningkatkan pemahaman tentang operasi aritmatika dan logika pemrograman secara umum.

DESAIN ALGORITMA

Algoritma brute force yang digunakan untuk memecahkan 20 solver bertujuan untuk mencari semua kemungkinan cara menggunakan empat angka dan berbagai operasi matematika dasar, yaitu penjumlahan (+), pengurangan (-), perkalian (*), dan pembagian (/), untuk menghasilkan angka 20. Dalam pendekatan brute force, kita mencoba semua kombinasi urutan angka dan operasi tanpa mengabaikan kemungkinan apa pun. Pendekatan ini memastikan kita dapat menemukan solusi dengan memeriksa setiap kemungkinan, meskipun prosesnya tidak efisien.

Pertama-tama, kita perlu memahami bahwa urutan angka berpengaruh terhadap hasil akhirnya. Oleh karena itu, langkah pertama dalam algoritma brute force ini adalah menghasilkan semua **permutasi** dari empat angka yang diberikan. Misalnya, jika angka yang dimasukkan adalah [2, 3, 4, 5], kita perlu mencoba setiap urutan yang mungkin dari angka-angka tersebut, seperti [2, 3, 4, 5], [2, 4, 5, 3], [3, 5, 2, 4], dan seterusnya. Terdapat 24 permutasi ($4 \text{ factorial} = 4!$) untuk empat angka, sehingga kita perlu memeriksa setiap permutasi ini.

Setelah menghasilkan permutasi angka, langkah selanjutnya adalah menghasilkan kombinasi operasi yang mungkin di antara angka-angka tersebut. Ada empat operasi yang tersedia (+, -, *, /), dan kita harus menempatkan tiga operasi di antara empat angka. Dengan empat operasi yang bisa dipilih untuk tiga tempat, kita memiliki total $4^3 = 64$ kombinasi operasi yang harus diperiksa. Setiap kombinasi operasi tersebut akan dicoba untuk setiap permutasi angka.

Selanjutnya, kita perlu membentuk ekspresi matematika dari angka dan operasi tersebut, serta mempertimbangkan penggunaan tanda kurung untuk mengatur prioritas operasi. Tanda kurung ini penting karena dapat mengubah hasil perhitungan. Sebagai contoh, ekspresi $2 + 3 * 4 - 5$ akan memberikan hasil yang berbeda dari $(2 + 3) * (4 - 5)$ karena prioritas operasi berubah. Algoritma ini mencoba beberapa kemungkinan cara untuk meletakkan tanda kurung guna memastikan semua urutan operasi diperiksa.

Setelah ekspresi terbentuk, kita perlu **mengevaluasi** hasil dari ekspresi tersebut. Hasil evaluasi ini kemudian dibandingkan dengan target, yaitu angka 20. Jika hasil evaluasi mendekati 20 (dengan toleransi kecil untuk mengatasi kesalahan pembulatan

dalam pembagian), ekspresi tersebut dianggap sebagai solusi yang valid. Semua solusi yang valid akan disimpan dan ditampilkan kepada pengguna.

Satu masalah yang mungkin muncul dalam evaluasi adalah **pembagian dengan nol**, yang menghasilkan error. Untuk mengatasi hal ini, algoritma dilengkapi dengan mekanisme penanganan error (try-except dalam Python), yang akan mengabaikan ekspresi yang menyebabkan pembagian dengan nol dan melanjutkan evaluasi ekspresi lainnya.

Pada akhirnya, algoritma ini memastikan bahwa setiap kemungkinan kombinasi angka, operasi, dan tanda kurung diperiksa, meskipun jumlahnya cukup besar. Sebagai contoh, dengan 24 permutasi angka, 64 kombinasi operasi, dan 5 kemungkinan cara untuk menempatkan tanda kurung, total kemungkinan yang harus diperiksa adalah $24 \times 64 \times 5 = 7680$ kemungkinan. Meskipun tidak efisien, pendekatan brute force ini memastikan bahwa semua solusi yang mungkin akan ditemukan.

ANALISIS KODE

```
import streamlit as st
from itertools import permutations, product
```

- **Streamlit** adalah sebuah library Python yang digunakan untuk membuat aplikasi web sederhana. Dalam hal ini, program menggunakan `streamlit` agar pengguna bisa memasukkan angka melalui antarmuka web dan melihat hasilnya.

- **`itertools.permutations`** dan **`itertools.product`** adalah fungsi yang digunakan untuk menghasilkan berbagai kombinasi. Kombinasi ini digunakan nanti untuk mencoba berbagai kemungkinan urutan angka dan operasi.

```
def escape_asterisk(expr):
    return expr.replace("*", "\\*")
```

- Fungsi ini digunakan untuk mengganti simbol `*` (perkalian) dengan versi yang aman untuk ditampilkan di markdown. Markdown adalah bahasa yang digunakan Streamlit untuk menampilkan teks, dan simbol `*` biasanya menandakan **italic**. Jadi, kita perlu mengatur simbol ini agar ditampilkan sebagai `*` dan bukan dijadikan **italic**.

- **Contoh:** Jika `expr` berisi string `"2*3"`, maka fungsi ini akan mengubahnya menjadi `"2*3"` sehingga aman untuk ditampilkan.

```
def apply_ops(ops, nums):
    expr = str(nums[0])
    for i in range(3):
        expr += ops[i] + str(nums[i + 1])
    return expr
```

Fungsi ini bertugas **membentuk ekspresi matematika** dalam bentuk string, dengan menggabungkan angka dan operasi secara berurutan. Misalnya, jika kita memiliki daftar angka `[1, 2, 3, 4]` dan operasi `["+", "-", "*"]`, fungsi ini akan menyusun string yang mewakili ekspresi matematika, yaitu `"1+2-3*4"`. String ini nantinya akan dievaluasi untuk melihat apakah hasilnya mendekati 20.

Langkah langkah:

- **`expr = str(nums[0])`:** Langkah pertama adalah mengubah angka pertama dalam daftar `nums` menjadi string. Misalnya, jika angka pertama adalah 1, maka akan dikonversi menjadi string `"1"`. Hal ini diperlukan karena kita akan membentuk ekspresi dalam bentuk string untuk dievaluasi nanti.

- **Looping (pengulangan):** Setelah angka pertama diubah menjadi string, fungsi ini masuk ke dalam loop yang menambahkan **operasi** dan **angka berikutnya** satu per satu ke dalam string.
 - Pada setiap iterasi, fungsi akan menambahkan operasi (misalnya $+$, $-$, $*$, $/$) dan angka dari daftar `nums`. Sebagai contoh, jika setelah angka 1, kita memiliki operasi $+$, dan angka berikutnya adalah 2, maka string akan diperbarui menjadi `"1+2"`.
 - Proses ini terus berlangsung hingga semua angka dan operasi dalam daftar selesai digabungkan. Jika urutannya adalah 1, 2, 3, 4 dan operasinya adalah $+$, $-$, $*$, hasil akhirnya akan menjadi string `"1+2-3*4"`.

```
def valid_combinations(nums, ops):
    templates = [
        "({{}}){{}}{}{}", # ((a op b) op c) op d
        "({}({{}}{})){}{}", # (a op (b op c)) op d
        "{}({}({{}}({{}}{})))", # a op (b op (c op d))
        "{}({}({{}}{})){}{}", # a op ((b op c) op d)
        "({}{}{}){}({{}}{})", # (a op b) op (c op d)
    ]
```

Fungsi `valid_combinations` menguji setiap kombinasi angka dan operasi dengan beberapa template berbeda, untuk memastikan kita menemukan semua kemungkinan ekspresi yang menghasilkan angka 20. Dalam fungsi `valid_combinations`, kita menggunakan beberapa template yang mewakili berbagai cara untuk menempatkan tanda kurung pada ekspresi matematika. Contoh template termasuk cara yang berbeda untuk menyusun angka dan operasi, seperti:

- **$((a + b) * c) / d$:** Dalam ekspresi ini, dua angka pertama (a dan b) dioperasikan terlebih dahulu, kemudian hasilnya dikalikan dengan c, dan akhirnya dibagi dengan d.
- **$(a + (b * c)) / d$:** Di sini, b dan c dioperasikan terlebih dahulu, hasilnya kemudian ditambahkan dengan a, dan hasil keseluruhan dibagi oleh d.

Template ini bertindak seperti pola dasar, di mana angka dan operasi dimasukkan ke dalam urutan yang telah ditentukan.

```
for template in templates:
    try:
        expr = template.format(
            nums[0], ops[0], nums[1], ops[1], nums[2], ops[2],
            nums[3]
        )
        if abs(eval(expr) - 20) < 1e-6:
            results.append(escape_asterisk(expr))
    except ZeroDivisionError:
        continue
return results
```

Bagian ini berfungsi untuk mengevaluasi atau menghitung ekspresi matematika yang dibentuk dari kombinasi angka dan operasi. Dalam algoritma 20 solver, setelah semua angka dan operasi disusun menjadi sebuah ekspresi (seperti $2 + 3 * 4 - 5$), ekspresi tersebut kemudian dihitung untuk melihat apakah hasilnya mendekati angka 20.

- **eval(expr):** Fungsi `eval()` mampu mengeksekusi ekspresi yang ditulis dalam bentuk string. Misalnya, jika kita punya string `"1+2*3/4"`, maka `eval()` akan menghitung hasil dari ekspresi tersebut sesuai dengan aturan matematika. Fungsi ini memungkinkan kita mengubah ekspresi yang dibangun dari angka dan operator menjadi nilai yang sebenarnya. Namun, karena `eval()` mengeksekusi kode langsung, kita perlu hati-hati dalam penggunaannya, terutama dalam konteks keamanan.

- **abs(eval(expr) - 20) < 1e-6:** Setelah ekspresi dihitung menggunakan `eval()`, bagian ini memastikan apakah hasil ekspresi tersebut sangat dekat dengan angka 20. Fungsi `abs()` digunakan untuk menghitung selisih absolut antara hasil ekspresi dan 20. Karena komputer bekerja dengan angka floating-point (pecahan desimal), terkadang hasil perhitungan tidak tepat 20, tetapi bisa sangat dekat, misalnya 19.999999 atau 20.000001, karena kesalahan pembulatan. Maka, kita menggunakan batas toleransi kecil sebesar $1e-6$ (0.000001) untuk memastikan hasilnya dianggap valid jika selisihnya kurang dari batas toleransi ini.

- **try-except ZeroDivisionError:** Bagian ini menangani kemungkinan error yang terjadi jika ekspresi melibatkan pembagian dengan nol, seperti $5 / 0$. Pembagian dengan nol tidak didefinisikan dalam matematika dan akan menimbulkan error di Python. Oleh karena itu, kita membungkus proses evaluasi ekspresi dalam blok `try-except`. Jika terjadi pembagian dengan nol (`ZeroDivisionError`), program akan mengabaikan ekspresi tersebut dan melanjutkan evaluasi ekspresi lainnya, tanpa menghentikan keseluruhan program. Ini memberikan perlindungan agar program tetap berjalan meskipun ada kasus pembagian dengan nol.

```
def solve_20(nums):
    operations = ["+", "-", "*", "/"]
    found_solutions = set()

    # Generate all permutations of numbers and operations
    for num_perm in permutations(nums):
        for ops in product(operations, repeat=3):
            solutions = valid_combinations(num_perm, ops)
            found_solutions.update(solutions)

    return found_solutions
```

Bagian ini adalah inti dari algoritma brute force, yang mencoba semua kemungkinan urutan angka dan kombinasi operasi untuk melihat apakah ada yang bisa menghasilkan angka 20.

- **Menggunakan `permutations(nums)`:** Fungsi ini berasal dari modul `itertools` dan digunakan untuk menghasilkan semua urutan angka yang mungkin. Karena urutan angka mempengaruhi hasil operasi (misalnya, $2 + 3 * 4$ berbeda dengan $4 * 3 + 2$), kita harus mempertimbangkan semua kemungkinan susunan angka. Jika ada 4 angka yang diberikan, seperti `[1, 2, 3, 4]`, maka fungsi `permutations(nums)` akan menghasilkan setiap urutan angka yang mungkin, seperti `[1, 2, 3, 4]`, `[4, 3, 2, 1]`, `[2, 1, 4, 3]`, dan seterusnya. Ini memastikan bahwa tidak ada susunan angka yang terlewatkan dalam proses pencarian solusi.

- **Menggunakan `product(operations, repeat=3)`:** Fungsi ini juga berasal dari modul `itertools`, dan berfungsi untuk menghasilkan semua kombinasi operasi (+, -, *, /) yang dapat diterapkan di antara angka-angka. Karena kita memiliki 4 angka, maka ada 3 tempat untuk operasi di antara mereka (contohnya, untuk ekspresi seperti `a op b op c op d`, kita membutuhkan 3 operasi). **`product(operations, repeat=3)`** akan menghasilkan setiap kombinasi dari tiga operasi yang mungkin, misalnya (+, -, -), (+, /, +), dan seterusnya. Dengan cara ini, kita mengeksplorasi semua kemungkinan kombinasi operasi yang dapat diterapkan di antara angka-angka tersebut.

- Setelah kita memiliki satu urutan angka dan satu kombinasi operasi, langkah selanjutnya adalah **memeriksa apakah kombinasi tersebut dapat menghasilkan angka 20**. Fungsi **`valid_combinations`** akan membentuk berbagai ekspresi matematika dari kombinasi angka dan operasi tersebut, sesuai dengan berbagai kemungkinan urutan pengelompokan operasi (dengan menggunakan tanda kurung). Setelah ekspresi-ekspresi tersebut dibentuk, program akan mengevaluasi setiap ekspresi untuk melihat apakah hasil akhirnya mendekati angka 20. Jika ada ekspresi yang memenuhi syarat, hasilnya akan disimpan sebagai solusi yang valid.

```
# Konfigurasi aplikasi Streamlit
st.title("20 Solver")
st.write(
    "Masukkan 4 angka pada kolom di bawah ini, dan lihat apakah mereka  
bisa menghasilkan angka 20."
)

num1 = st.number_input("Angka 1", value=0, step=1)
num2 = st.number_input("Angka 2", value=0, step=1)
num3 = st.number_input("Angka 3", value=0, step=1)
num4 = st.number_input("Angka 4", value=0, step=1)
```

Dalam kode ini, fungsi `st.number_input` digunakan untuk menampilkan kotak input angka pada antarmuka pengguna, di mana pengguna dapat memasukkan nilai numerik.

Fungsi `st.number_input` memiliki beberapa parameter yang penting untuk dipahami. Salah satu parameter utama adalah `value`, yang mengatur nilai default atau nilai awal yang ditampilkan di kotak input. Dalam kasus ini, nilai default untuk setiap kotak input adalah 0 (`value=0`), yang berarti jika pengguna tidak memasukkan angka, nilai awal yang akan diproses adalah 0. Di sini, `step=1`, yang berarti bahwa ketika pengguna menambah atau mengurangi nilai, setiap klik atau perubahan akan menambah atau mengurangi angka sebesar 1.


```
numbers = [num1, num2, num3, num4]
if all(numbers): # Pastikan semua input diberikan
    solutions = solve_20(numbers)
    if solutions:
        st.success(f"{len(solutions)} solusi ditemukan:")
        for solution in solutions:
            st.write(solution)
    else:
        st.warning("Tidak ada solusi yang ditemukan untuk menghasilkan 20.")
```

Bagian ini memeriksa apakah semua angka sudah diisi menggunakan `if all(numbers):`, di mana `numbers` adalah daftar berisi empat angka input dari pengguna. Fungsi `all()` memastikan semua angka tidak bernilai 0. Jika semua angka sudah diisi, program memanggil `solve_20(numbers)` untuk mencari solusi yang menghasilkan angka 20.

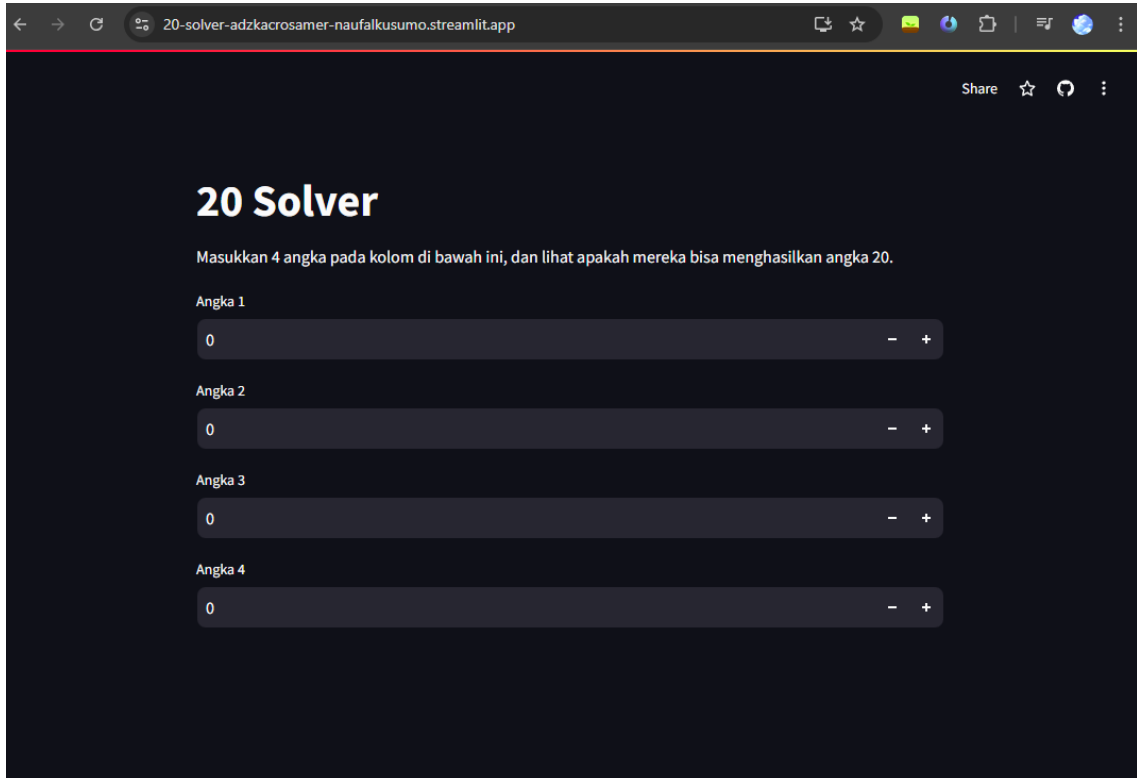
Jika solusi ditemukan, jumlah solusi ditampilkan dengan `st.success`, dan setiap solusi ditampilkan satu per satu menggunakan `st.write`. Jika tidak ada solusi, program menampilkan pesan peringatan dengan `st.warning`, memberi tahu pengguna bahwa tidak ada kombinasi angka yang bisa menghasilkan 20.

PENGUJIAN

Hasil pengerjaan dapat di akses melalui :

<https://20-solver-adzkacrosamer-naufalkusumo.streamlit.app/>

Pertama kita buka website tersebut, ketika kita sudah masuk kedalam website, maka akan muncul tampilan seperti berikut



The screenshot shows a web browser window with the address bar displaying "20-solver-adzkacrosamer-naufalkusumo.streamlit.app". The page has a dark background and features the title "20 Solver" in large white text. Below the title, a subtitle reads: "Masukkan 4 angka pada kolom di bawah ini, dan lihat apakah mereka bisa menghasilkan angka 20." There are four input fields, each labeled "Angka 1", "Angka 2", "Angka 3", and "Angka 4" respectively. Each field contains the number "0" and has minus and plus buttons on the right side for adjusting the value. In the top right corner of the application area, there are icons for "Share", a star, a refresh symbol, and a menu icon.

Lalu kita masukkan 4 angka yang kita inginkan, sebagai contoh kita inputkan angka 4, 5, 6, 7

20-solver-adzkacrosamer-naufalkusumo.streamlit.app

Share ☆ ↺ ⋮

20 Solver

Masukkan 4 angka pada kolom di bawah ini, dan lihat apakah mereka bisa menghasilkan angka 20.

Angka 1

4 - +

Angka 2

5 - +

Angka 3

6 - +

Angka 4

7 - +

24 solusi ditemukan:

- $(5/(7-6))*4$
- $(7-5)*(6+4)$
- $5*(4*(7-6))$
- $5/((7-6)/4)$
- $(4/(7-6))*5$
- $(4+6)*(7-5)$

Maka akan muncul jumlah solusi dan operasi operasinya

Dan jika kita menginputkan 4 nilai yang tidak bisa menghasilkan 20 maka akan muncul seperti ini

20-solver-adzkacrosamer-naufalkusumo.streamlit.app

Share ☆ ↺ ⋮

20 Solver

Masukkan 4 angka pada kolom di bawah ini, dan lihat apakah mereka bisa menghasilkan angka 20.

Angka 1

1 - +

Angka 2

1 - +

Angka 3

1 - +

Angka 4

1 - +

Tidak ada solusi yang ditemukan untuk menghasilkan 20.

PEMBAGIAN TUGAS

Nama	Peran
Mohammad Adzka Crosamer	Membuat Laporan, Mendesain sekaligus mendeploy kode menggunakan streamlit, Membantu pengerjaan Python
Naufal Narendro Kusmo	Mengerjakan kode menggunakan Python, Melakukan troubleshooting, Membantu pembuatan laporan, membuat video, Mengatur repository Github.