# TUGAS KECIL 1 IF2211 STRATEGI ALGORITMA 2022/2023

Disusun oleh :

Rayhan Hanif Maulana Pradana/13521112


TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

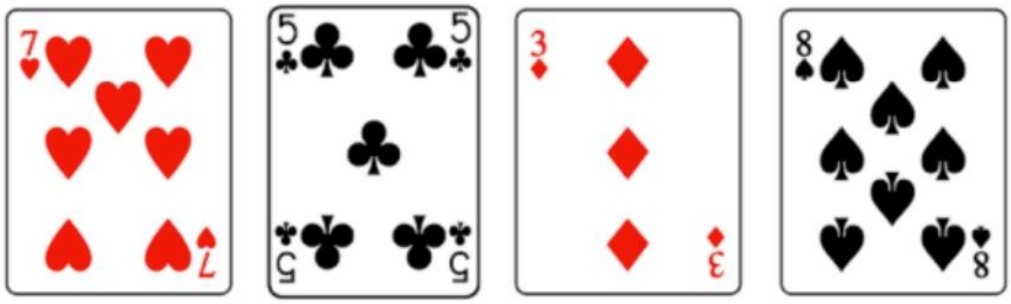
INSTITUT TEKNOLOGI BANDUNG

2022

# BAB I

# DESKRIPSI MASALAH

Permainan kartu 24 adalah permainan kartu aritmatika dengan tujuan mencari cara untuk mengubah 4 buah angka random sehingga mendapatkan hasil akhir sejumlah 24. Permainan ini menarik cukup banyak peminat dikarenakan dapat meningkatkan kemampuan berhitung serta mengasah otak agar dapat berpikir dengan cepat dan akurat. Permainan Kartu 24 biasa dimainkan dengan menggunakan kartu remi. Kartu remi terdiri dari 52 kartu yang terbagi menjadi empat suit (sekop, hati, keriting, dan wajik) yang masing-masing terdiri dari 13 kartu (As, 2, 3, 4, 5, 6, 7, 8, 9, 10, Jack, Queen, dan King). Yang perlu diperhatikan hanyalah nilai kartu yang didapat (As, 2, 3, 4, 5, 6, 7, 8, 9, 10, Jack, Queen, dan King). As bernilai 1, Jack bernilai 11, Queen bernilai 12, King bernilai 13, sedangkan kartu bilangan memiliki nilai dari bilangan itu sendiri. Pada awal permainan moderator atau salah satu pemain mengambil 4 kartu dari dek yang sudah dikocok secara random. Permainan berakhir ketika pemain berhasil menemukan solusi untuk membuat kumpulan nilainya menjadi 24. Pengubahan nilai tersebut dapat dilakukan menggunakan operasi dasar matematika penjumlahan (+), pengurangan (-), perkalian (×), divisi (/) dan tanda kurung ( () ). Tiap kartu harus digunakan tepat sekali dan urutan penggunaannya bebas.



MAKE IT 24

(Sumber : Spesifikasi Tugas Kecil 1 IF2211 Strategi Algoritma 2022)

# BAB II

# ALGORITMA BRUTE FORCE UNTUK PENCARIAN SOLUSI PERMAINAN KARTU 24

Algoritma *Brute Force* merupakan algoritma yang menggunakan pendekatan yang lempang (*straightforward*) untuk memecahkan suatu perosalan. Pada umumnya, algoritma *Brute Force* didasarkan secara langsung pada persoalan (*problem statement*) yang ditemui dan pada definisi atau konsep yang dilibatkan. Algoritma *Brute Force* dapat memecahkan persoalan-persoalan yang sederhana dengan cara langsung dan jelas caranya (*obvious way*). Bahkan, terdapat beberapa persoalan yang hanya dapat diselesaikan dengan *Brute Force*, seperti pencarian elemen pada larik yang tidak berurut.

Permainan kartu 24 memiliki *problem statement* dan definisi yang cukup sederhana, yaitu mencari solusi dari operasi antara empat buah kartu sehingga mendapatkan hasil 24. Permasalahan tersebut dapat diperluas dengan pencarian semua solusi yang mungkin. Hal tersebut dapat dicapai dengan menggunakan bantuan program dengan mengimplementasikan algoritma *Brute Force*. Pada dasarnya, implementasi algoritma *Brute Force* yang digunakan adalah melakukan pengecekan terhadap semua susunan kartu, operator aritmatika, maupun urutan operasi dengan tanda kurung.

1. Pencarian semua susunan dari empat kartu yang mungkin

   Misalkan A, B, C, dan D adalah empat kartu yang akan dicari solusinya. Keempat kartu tersebut dapat disusun dengan cara :

   A, B, C, D

   A, B, D, C

   A, C, B, D

   A, D, B, C

   …

   D, C, B, A

   Semua susunan atau permutasi yang mungkin dari keempat kartu tersebut akan disimpan untuk perhitungan atau pencarian solusi dalam sebuah larik dua dimensi (matriks), dengan baris pada matriks menyatakan sebuah susunan yang mungkin dan kolom menyatakan posisi kartu pada sebuah susunan.

2. Pencarian semua susunan dari operator-operator aritmatika

Misalkan A, B, C, dan D adalah keempat kartu yang akan dicari solusinya dan Op1, Op2, dan Op3 adalah operator-operator aritmatika. Pada satu solusi, operator aritmatika yang muncul akan selalu berjumlah tiga, yaitu secara umum berbentuk:

A Op1 B Op2 C Op3 D

Sehingga, operator-operator aritmatika +, -, *, dan / dapat disusun menjadi tiga, dan susunan-susunan yang mungkin adalah :

+, +, +

+, +, -

+, -. +

-, +, +

+, +, *

…

*, +, -

+, -, /

Susunan-susunan tersebut juga akan disimpan pada larik dua dimensi (matriks) untuk perhitungan atau pencarian solusi, dengan baris pada matriks menyatakan sebuah susunan yang mungkin dan kolom menyatakan posisi sebuah operator aritmatika dalam sebuah susunan.

3. Pengecekan semua solusi yang mungkin

   Misalkan A, B, C, dan D adalah keempat kartu yang akan dicari solusinya dan Op1, Op2, dan Op3 adalah operator-operator aritmatika. Susunan-susunan dari kurung yang mungkin adalah :

(A Op1 B) Op2 (C Op3 D)

((A Op1 B) Op2 C) Op3 D

(A Op1 (B Op2 C)) Op3  D)

A Op1 ((B Op2 C) Op3 D)

A Op1 (B Op2 (C Op3 D))

Oleh karena itu, algoritma akan mengecek satu per satu dari semua susunan kartu, serta semua susunan operator aritmatika untuk setiap **satu** susunan kartu. Pada setiap susunan kartu dan operator aritmatikanya (e.g. susunan kartu 3, 7, 8, 5 dan susunan operator aritmatika *, -, +), akan dicek apakah terdapat operasi dengan kurung yang telah ditampilkan di atas paragraf ini yang akan menghasilkan 24 (e.g.  ((3 * 7) - 5) + 8 menghasilkan 24, sehingga operasi ((3 * 7) - 5) + 8  adalah solusi).

Solusi-solusi yang didapat akan disimpan berupa string dalam sebuah larik (memungkinkan karena bahasa yang digunakan adalah C++, sehingga dapat menggunakan vector) untuk di output ke layer dan juga dapat disimpan ke sebuah file.

# BAB III

## Source Code

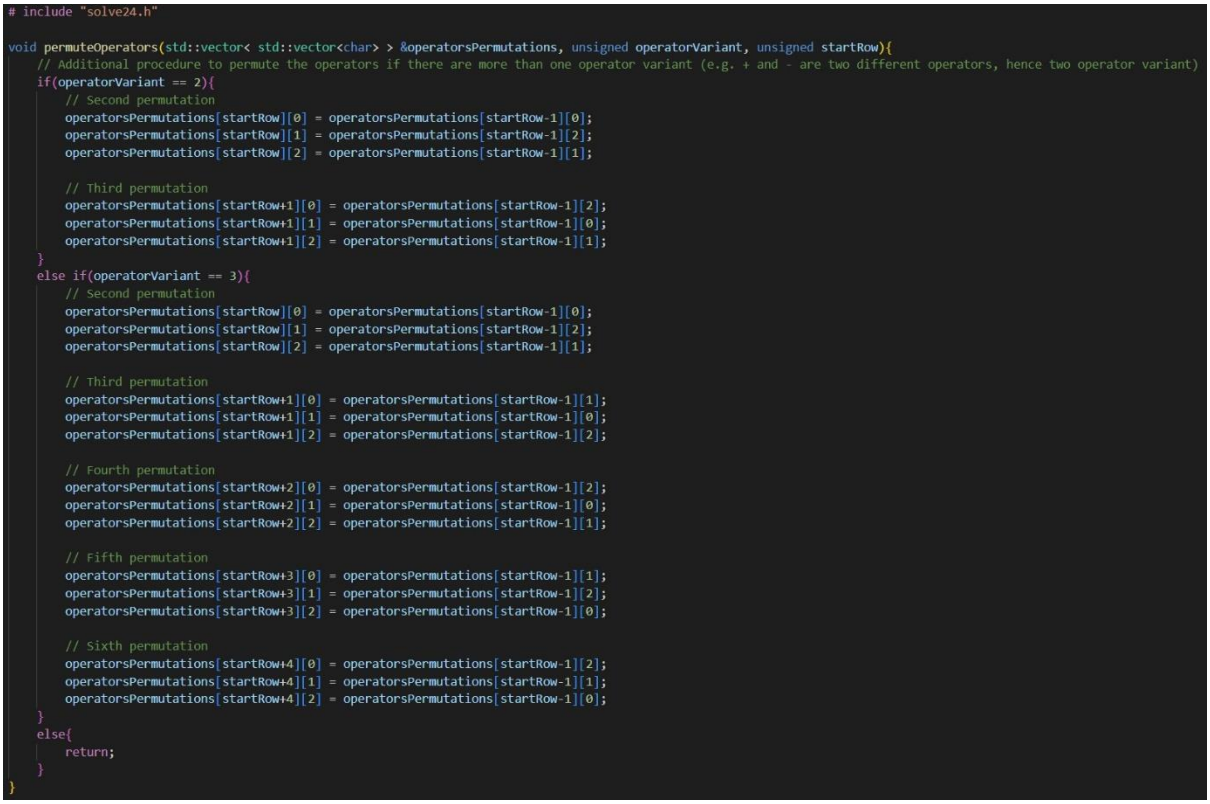
Bahasa yang digunakan untuk implementasi algoritma di atas adalah C++

1. File header untuk semua prosedur dan fungsi yang diimplementasikan

```
# include <iostream>
# include <vector>
# include <iomanip>
# include <sstream>
# include <fstream>

# ifndef _SOLVE24_H_
# define _SOLVE24_H_

void permuteOperators(std::vector< std::vector<char> > &operatorsPermutations, unsigned operatorVariant, unsigned startRow);
// Additional procedure to permute the operators if there are more than one operator variant (e.g. + and - are two different operators, hence two operator variant)

std::vector< std::vector<char> > operators();
// Store all permutations of operators +, -, *, and / in a 2D Vector

std::vector< std::vector<double> > cards(int a, int b, int c, int d);
// Store all permutations of the four cards in a 2D Vector

std::string doubleToString(double var);
// Convert double to string with precision set to 0

double convertInput(std::string inputVar);      You, 5 hours ago • Uncommitted changes
// Converts user input (string) to double

double operate(double operandA, double operandB, char op);
// Arithmetic operation

std::vector< std::string > solve24(double a, double b, double c, double d);
// Main algorithm of finding the solutions, returns the vector of the solution

# endif
```

2. Implementasi untuk permutasi kartu dan operator aritmatika

Catatan : Permutasi untuk kartu dilakukan secara manual dan untuk operator aritmatika secara semi-manual. Hal tersebut dilakukan karena susunan tidak terlalu banyak dan algoritma untuk permutasi yang dibuat penulis (tidak lagi digunakan dalam program) kurang efisien (Menggunakan beberapa *nested for-loop*).



```
# include "solve24.h"

void permuteOperators(std::vector< std::vector<char> > &operatorsPermutations, unsigned operatorVariant, unsigned startRow){
    // Additional procedure to permute the operators if there are more than one operator variant (e.g. + and - are two different operators, hence two operator variant)
    if(operatorVariant == 2){
        // Second permutation
        operatorsPermutations[startRow][0] = operatorsPermutations[startRow-1][0];
        operatorsPermutations[startRow][1] = operatorsPermutations[startRow-1][2];
        operatorsPermutations[startRow][2] = operatorsPermutations[startRow-1][1];

        // Third permutation
        operatorsPermutations[startRow+1][0] = operatorsPermutations[startRow-1][2];
        operatorsPermutations[startRow+1][1] = operatorsPermutations[startRow-1][0];
        operatorsPermutations[startRow+1][2] = operatorsPermutations[startRow-1][1];
    }
    else if(operatorVariant == 3){
        // Second permutation
        operatorsPermutations[startRow][0] = operatorsPermutations[startRow-1][0];
        operatorsPermutations[startRow][1] = operatorsPermutations[startRow-1][2];
        operatorsPermutations[startRow][2] = operatorsPermutations[startRow-1][1];

        // Third permutation
        operatorsPermutations[startRow+1][0] = operatorsPermutations[startRow-1][1];
        operatorsPermutations[startRow+1][1] = operatorsPermutations[startRow-1][0];
        operatorsPermutations[startRow+1][2] = operatorsPermutations[startRow-1][2];

        // Fourth permutation
        operatorsPermutations[startRow+2][0] = operatorsPermutations[startRow-1][2];
        operatorsPermutations[startRow+2][1] = operatorsPermutations[startRow-1][0];
        operatorsPermutations[startRow+2][2] = operatorsPermutations[startRow-1][1];

        // Fifth permutation
        operatorsPermutations[startRow+3][0] = operatorsPermutations[startRow-1][1];
        operatorsPermutations[startRow+3][1] = operatorsPermutations[startRow-1][2];
        operatorsPermutations[startRow+3][2] = operatorsPermutations[startRow-1][0];

        // Sixth permutation
        operatorsPermutations[startRow+4][0] = operatorsPermutations[startRow-1][2];
        operatorsPermutations[startRow+4][1] = operatorsPermutations[startRow-1][1];
        operatorsPermutations[startRow+4][2] = operatorsPermutations[startRow-1][0];
    }
    else{
        return;
    }
}
```

```
std::vector< std::vector<char> > operators(){
    // Store all permutations of operators +, -, *, and / in a 2D Vector
    std::vector< std::vector<char> > operatorsPermutations;
    operatorsPermutations.resize(64, std::vector<char>(3, 0));

    operatorsPermutations[0][0] = '+';
    operatorsPermutations[0][1] = '+';
    operatorsPermutations[0][2] = '+';

    operatorsPermutations[1][0] = '+';
    operatorsPermutations[1][1] = '+';
    operatorsPermutations[1][2] = '-';

    permuteOperators(operatorsPermutations, 2, 2);

    operatorsPermutations[4][0] = '+';
    operatorsPermutations[4][1] = '+';
    operatorsPermutations[4][2] = '*';

    permuteOperators(operatorsPermutations, 2, 5);

    operatorsPermutations[7][0] = '+';
    operatorsPermutations[7][1] = '+';
    operatorsPermutations[7][2] = '/';

    permuteOperators(operatorsPermutations, 2, 8);

    operatorsPermutations[10][0] = '-';
    operatorsPermutations[10][1] = '-';
    operatorsPermutations[10][2] = '-';

    operatorsPermutations[11][0] = '-';
    operatorsPermutations[11][1] = '-';
    operatorsPermutations[11][2] = '+';

    permuteOperators(operatorsPermutations, 2, 12);

    operatorsPermutations[14][0] = '-';
    operatorsPermutations[14][1] = '-';
    operatorsPermutations[14][2] = '*';

    permuteOperators(operatorsPermutations, 2, 15);

    operatorsPermutations[17][0] = '-';
    operatorsPermutations[17][1] = '-';
    operatorsPermutations[17][2] = '/';

    permuteOperators(operatorsPermutations, 2, 18);
```

```
operatorsPermutations[20][0] = '*';
operatorsPermutations[20][1] = '*';
operatorsPermutations[20][2] = '*';

operatorsPermutations[21][0] = '*';
operatorsPermutations[21][1] = '*';
operatorsPermutations[21][2] = '+';

permuteOperators(operatorsPermutations, 2, 22);

operatorsPermutations[24][0] = '*';
operatorsPermutations[24][1] = '*';
operatorsPermutations[24][2] = '-';

permuteOperators(operatorsPermutations, 2, 25);

operatorsPermutations[27][0] = '*';
operatorsPermutations[27][1] = '*';
operatorsPermutations[27][2] = '/';

permuteOperators(operatorsPermutations, 2, 28);

operatorsPermutations[30][0] = '/';
operatorsPermutations[30][1] = '/';
operatorsPermutations[30][2] = '/';

operatorsPermutations[31][0] = '/';
operatorsPermutations[31][1] = '/';
operatorsPermutations[31][2] = '+';

permuteOperators(operatorsPermutations, 2, 32);

operatorsPermutations[34][0] = '/';
operatorsPermutations[34][1] = '/';
operatorsPermutations[34][2] = '-';

permuteOperators(operatorsPermutations, 2, 35);

operatorsPermutations[37][0] = '/';
operatorsPermutations[37][1] = '/';
operatorsPermutations[37][2] = '*';

permuteOperators(operatorsPermutations, 2, 38);

operatorsPermutations[40][0] = '+';
operatorsPermutations[40][1] = '-';
operatorsPermutations[40][2] = '*';

permuteOperators(operatorsPermutations, 3, 41);
```

```
    operatorsPermutations[46][0] = '+';
    operatorsPermutations[46][1] = '-';
    operatorsPermutations[46][2] = '/';

    permuteOperators(operatorsPermutations, 3, 47);

    operatorsPermutations[52][0] = '-';
    operatorsPermutations[52][1] = '*';
    operatorsPermutations[52][2] = '/';

    permuteOperators(operatorsPermutations, 3, 53);

    operatorsPermutations[58][0] = '+';
    operatorsPermutations[58][1] = '*';
    operatorsPermutations[58][2] = '/';

    permuteOperators(operatorsPermutations, 3, 59);

    return operatorsPermutations;
}
```

```
std::vector< std::vector<double> > cards(int a, int b, int c, int d){
    // Store all permutations of the four cards in a 2D Vector
    std::vector< std::vector<double> > cardsPermutations;
    cardsPermutations.resize(24, std::vector<double>(4, 0));

    cardsPermutations[0][0] = a;
    cardsPermutations[0][1] = b;
    cardsPermutations[0][2] = c;
    cardsPermutations[0][3] = d;

    cardsPermutations[1][0] = a;
    cardsPermutations[1][1] = b;
    cardsPermutations[1][2] = d;
    cardsPermutations[1][3] = c;

    cardsPermutations[2][0] = a;
    cardsPermutations[2][1] = c;
    cardsPermutations[2][2] = b;
    cardsPermutations[2][3] = d;

    cardsPermutations[3][0] = a;
    cardsPermutations[3][1] = d;
    cardsPermutations[3][2] = b;
    cardsPermutations[3][3] = c;

    cardsPermutations[4][0] = a;
    cardsPermutations[4][1] = c;
    cardsPermutations[4][2] = d;
    cardsPermutations[4][3] = b;

    cardsPermutations[5][0] = a;
    cardsPermutations[5][1] = d;
    cardsPermutations[5][2] = c;
    cardsPermutations[5][3] = b;

    cardsPermutations[6][0] = b;
    cardsPermutations[6][1] = a;
    cardsPermutations[6][2] = c;
    cardsPermutations[6][3] = d;
```

```
cardsPermutations[7][0] = b;
cardsPermutations[7][1] = a;
cardsPermutations[7][2] = d;
cardsPermutations[7][3] = c;

cardsPermutations[8][0] = c;
cardsPermutations[8][1] = a;
cardsPermutations[8][2] = b;
cardsPermutations[8][3] = d;

cardsPermutations[9][0] = d;
cardsPermutations[9][1] = a;
cardsPermutations[9][2] = b;
cardsPermutations[9][3] = c;

cardsPermutations[10][0] = c;
cardsPermutations[10][1] = a;
cardsPermutations[10][2] = d;
cardsPermutations[10][3] = b;

cardsPermutations[11][0] = d;
cardsPermutations[11][1] = a;
cardsPermutations[11][2] = c;
cardsPermutations[11][3] = b;

cardsPermutations[12][0] = b;
cardsPermutations[12][1] = c;
cardsPermutations[12][2] = a;
cardsPermutations[12][3] = d;

cardsPermutations[13][0] = b;
cardsPermutations[13][1] = d;
cardsPermutations[13][2] = a;
cardsPermutations[13][3] = c;

cardsPermutations[14][0] = c;
cardsPermutations[14][1] = b;
cardsPermutations[14][2] = a;
cardsPermutations[14][3] = d;

cardsPermutations[15][0] = d;
cardsPermutations[15][1] = b;
cardsPermutations[15][2] = a;
cardsPermutations[15][3] = c;

cardsPermutations[16][0] = c;
cardsPermutations[16][1] = d;
cardsPermutations[16][2] = a;
cardsPermutations[16][3] = b;

cardsPermutations[17][0] = d;
cardsPermutations[17][1] = c;
cardsPermutations[17][2] = a;
cardsPermutations[17][3] = b;
```

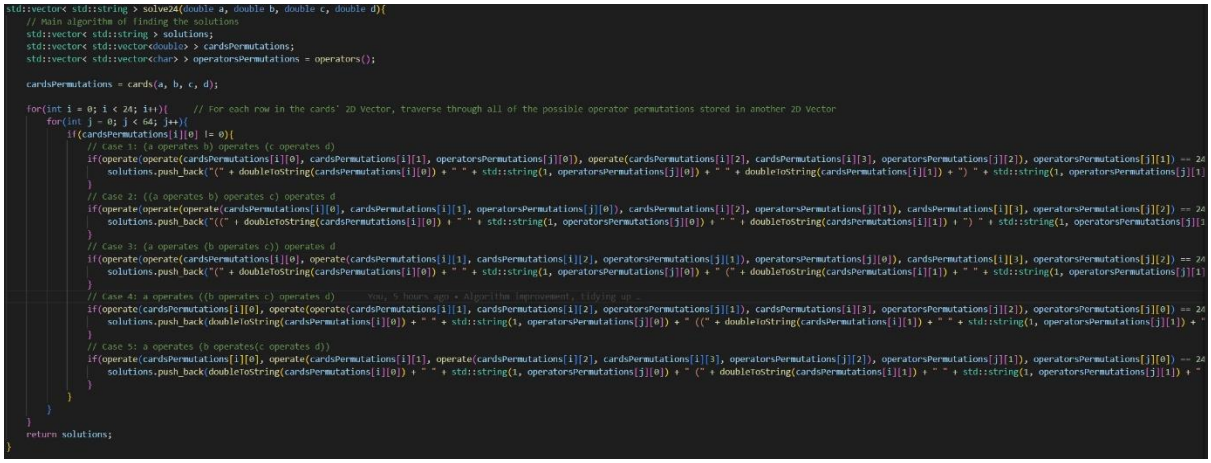
```
cardsPermutations[18][0] = b;
cardsPermutations[18][1] = c;
cardsPermutations[18][2] = d;
cardsPermutations[18][3] = a;

cardsPermutations[19][0] = b;
cardsPermutations[19][1] = d;
cardsPermutations[19][2] = c;
cardsPermutations[19][3] = a;

cardsPermutations[20][0] = c;
cardsPermutations[20][1] = b;
cardsPermutations[20][2] = d;
cardsPermutations[20][3] = a;

cardsPermutations[21][0] = d;
cardsPermutations[21][1] = b;
cardsPermutations[21][2] = c;
cardsPermutations[21][3] = a;

cardsPermutations[22][0] = c;
cardsPermutations[22][1] = d;
cardsPermutations[22][2] = b;
cardsPermutations[22][3] = a;

cardsPermutations[23][0] = d;
cardsPermutations[23][1] = c;
cardsPermutations[23][2] = b;
cardsPermutations[23][3] = a;
```

- Program di bawah ini digunakan untuk mengabaikan susunan yang duplikat (apabila dalam kasus terdapat dua atau lebih kartu yang sama)

```
    // Disable duplicate permutation
    unsigned isSameElmt; // Boolean value

    for(int i = 0; i < 24; i++){
        for(int j = 0; j < 24; j++){
            isSameElmt = 0;
            if(i != j){
                for(int k = 0; k < 4; k++){
                    if(cardsPermutations[i][k] == cardsPermutations[j][k]) isSameElmt = 1;
                    else{
                        isSameElmt = 0;
                        break;
                    }
                }
                if(isSameElmt == 1) cardsPermutations[j][0] = 0; // Make the element of the first column in a duplicate row 0 to disable the row
            }
        }
    }

    return cardsPermutations;
}
```

3. Implementasi algoritma utama dan fungsi atau prosedur tambahan

```
# include "solve24.h"

std::string doubleToString(double var){
    std::stringstream stream;
    stream << std::fixed << std::setprecision(0) << var;

    return stream.str();
}

double convertInput(std::string inputVar){      You, 5 hours ago • Uncommitted
    if(inputVar == "1" || inputVar == "A" || inputVar == "a") return 1;
    if(inputVar == "2") return 2;
    if(inputVar == "3") return 3;
    if(inputVar == "4") return 4;
    if(inputVar == "5") return 5;
    if(inputVar == "6") return 6;
    if(inputVar == "7") return 7;
    if(inputVar == "8") return 8;
    if(inputVar == "9") return 9;
    if(inputVar == "10") return 10;
    if(inputVar == "11" || inputVar == "J" || inputVar == "j") return 11;
    if(inputVar == "12" || inputVar == "Q" || inputVar == "q") return 12;
    if(inputVar == "13" || inputVar == "K" || inputVar == "k") return 13;
}

double operate(double operandA, double operandB, char op){
    switch (op){
        case '+':
            return operandA + operandB;
        case '-':
            return operandA - operandB;
        case '*':
            return operandA * operandB;
        case '/':
            return operandA / operandB;
        default:
            break;
    }
}
```

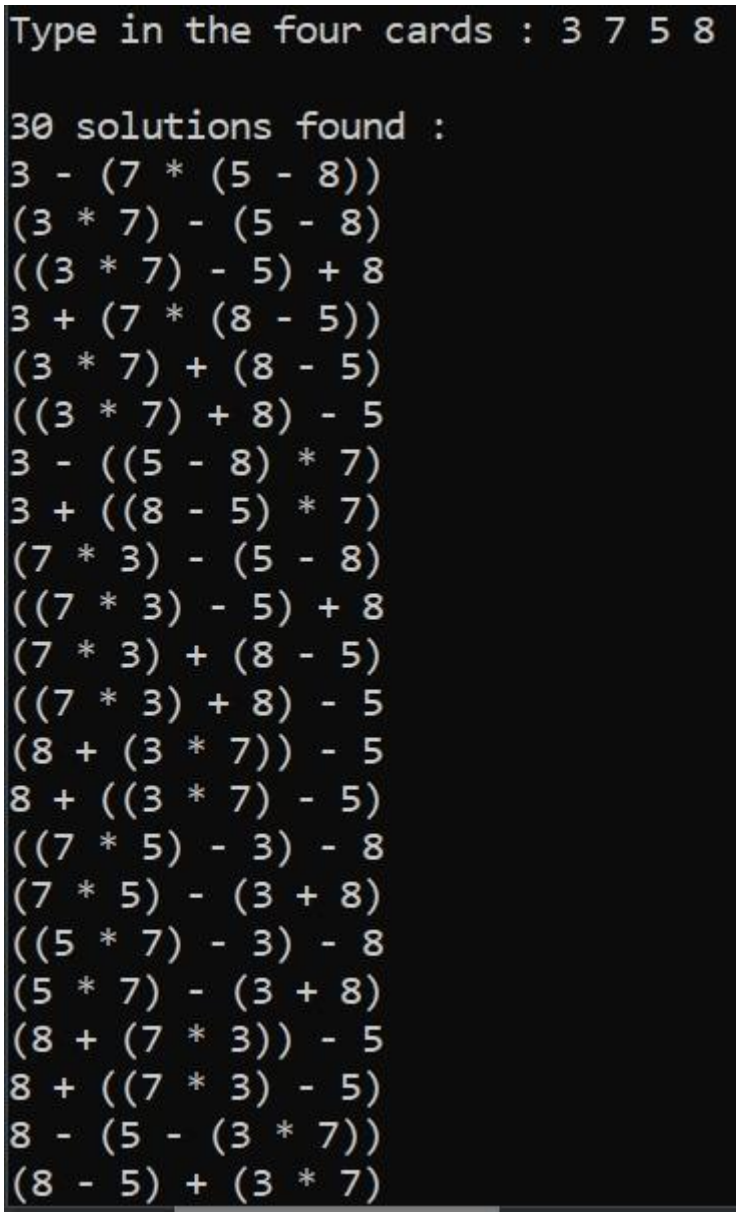


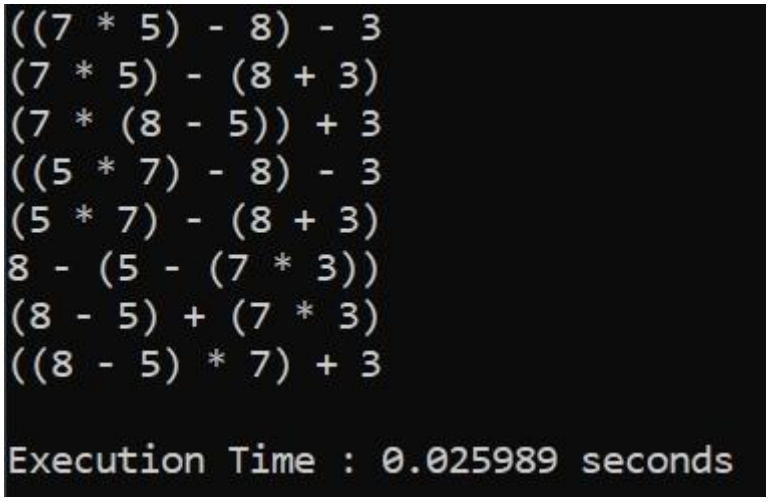
```
std::vector< std::string > solve24(double a, double b, double c, double d){
    // Main algorithm of finding the solutions
    std::vector< std::string > solutions;
    std::vector< std::vector<double> > cardsPermutations;
    std::vector< std::vector<char> > operatorsPermutations = operators();

    cardsPermutations = cards(a, b, c, d);

    for(int i = 0; i < 24; i++){      // For each row in the cards' 2D Vector, traverse through all of the possible operator permutations stored in another 2D Vector
        for(int j = 0; j < 64; j++){
            if(cardsPermutations[i][0] != 0){
                // Case 1: (a operates b) operates (c operates d)
                if(operate(operate(cardsPermutations[i][0], cardsPermutations[i][1], operatorsPermutations[j][0]), operate(cardsPermutations[i][2], cardsPermutations[i][3], operatorsPermutations[j][2]), operatorsPermutations[j][1]) == 24){
                    solutions.push_back("(" + doubleToString(cardsPermutations[i][0]) + " " + std::string(1, operatorsPermutations[j][0]) + " " + doubleToString(cardsPermutations[i][1]) + ") " + std::string(1, operatorsPermutations[j][1]) + " (" + doubleToString(cardsPermutations[i][2]) + " " + std::string(1, operatorsPermutations[j][2]) + " " + doubleToString(cardsPermutations[i][3]) + ")");
                }
                // Case 2: ((a operates b) operates c) operates d
                if(operate(operate(operate(cardsPermutations[i][0], cardsPermutations[i][1], operatorsPermutations[j][0]), cardsPermutations[i][2], operatorsPermutations[j][1]), cardsPermutations[i][3], operatorsPermutations[j][2]) == 24){
                    solutions.push_back("((" + doubleToString(cardsPermutations[i][0]) + " " + std::string(1, operatorsPermutations[j][0]) + " " + doubleToString(cardsPermutations[i][1]) + ") " + std::string(1, operatorsPermutations[j][1]) + " " + doubleToString(cardsPermutations[i][2]) + ") " + std::string(1, operatorsPermutations[j][2]) + " " + doubleToString(cardsPermutations[i][3]));
                }
                // Case 3: (a operates (b operates c)) operates d
                if(operate(operate(cardsPermutations[i][0], operate(cardsPermutations[i][1], cardsPermutations[i][2], operatorsPermutations[j][1]), operatorsPermutations[j][0]), cardsPermutations[i][3], operatorsPermutations[j][2]) == 24){
                    solutions.push_back("(" + doubleToString(cardsPermutations[i][0]) + " " + std::string(1, operatorsPermutations[j][0]) + " (" + doubleToString(cardsPermutations[i][1]) + " " + std::string(1, operatorsPermutations[j][1]) + " " + doubleToString(cardsPermutations[i][2]) + ") " + std::string(1, operatorsPermutations[j][2]) + " " + doubleToString(cardsPermutations[i][3]));
                }
                // Case 4: a operates ((b operates c) operates d)      Yay, 6 hours ago + Algorithm Improvement, tidying up
                if(operate(cardsPermutations[i][0], operate(operate(cardsPermutations[i][1], cardsPermutations[i][2], operatorsPermutations[j][1]), cardsPermutations[i][3], operatorsPermutations[j][2]), operatorsPermutations[j][0]) == 24){
                    solutions.push_back(doubleToString(cardsPermutations[i][0]) + " " + std::string(1, operatorsPermutations[j][0]) + " ((" + doubleToString(cardsPermutations[i][1]) + " " + std::string(1, operatorsPermutations[j][1]) + " " + doubleToString(cardsPermutations[i][2]) + ") " + std::string(1, operatorsPermutations[j][2]) + " " + doubleToString(cardsPermutations[i][3]) + ")");
                }
                // Case 5: a operates (b operates(c operates d))
                if(operate(cardsPermutations[i][0], operate(cardsPermutations[i][1], operate(cardsPermutations[i][2], cardsPermutations[i][3], operatorsPermutations[j][2]), operatorsPermutations[j][1]), operatorsPermutations[j][0]) == 24){
                    solutions.push_back(doubleToString(cardsPermutations[i][0]) + " " + std::string(1, operatorsPermutations[j][0]) + " (" + doubleToString(cardsPermutations[i][1]) + " " + std::string(1, operatorsPermutations[j][1]) + " (" + doubleToString(cardsPermutations[i][2]) + " " + std::string(1, operatorsPermutations[j][2]) + " " + doubleToString(cardsPermutations[i][3]) + "))");
                }
            }
        }
    }
    return solutions;
}
```

## BAB IV

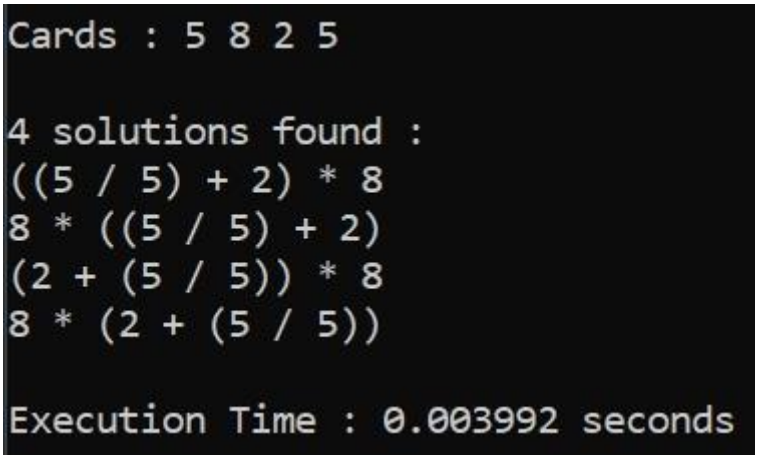
## Input dan Output Program

1. Test 1 (Input = 3 7 5 8)

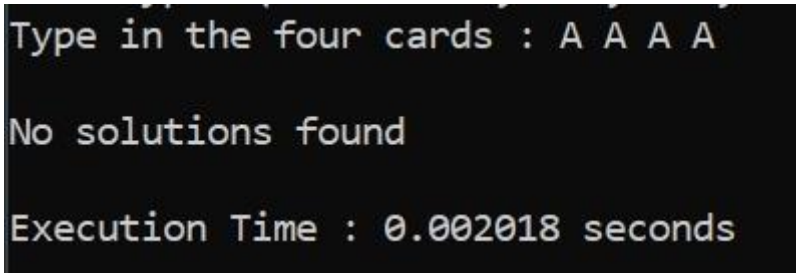




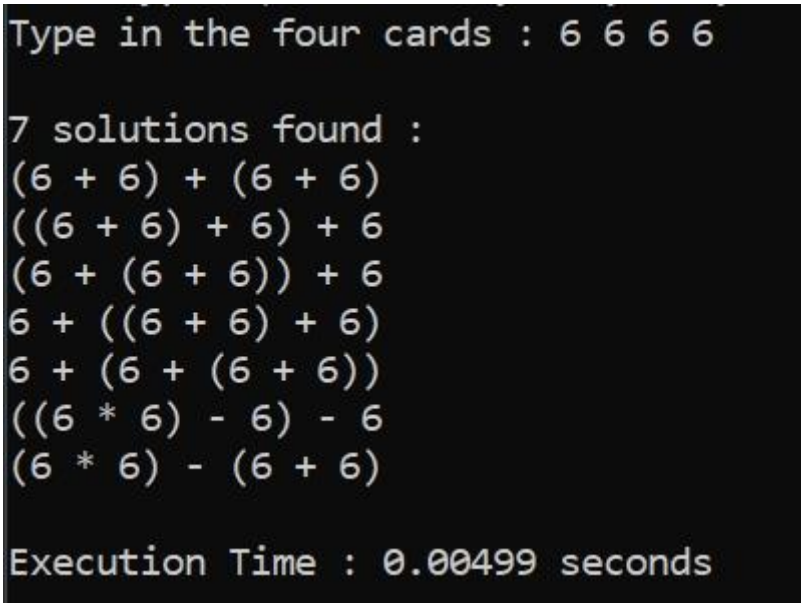

2. Test 2 (Input = Random, didapat 5 8 2 5)



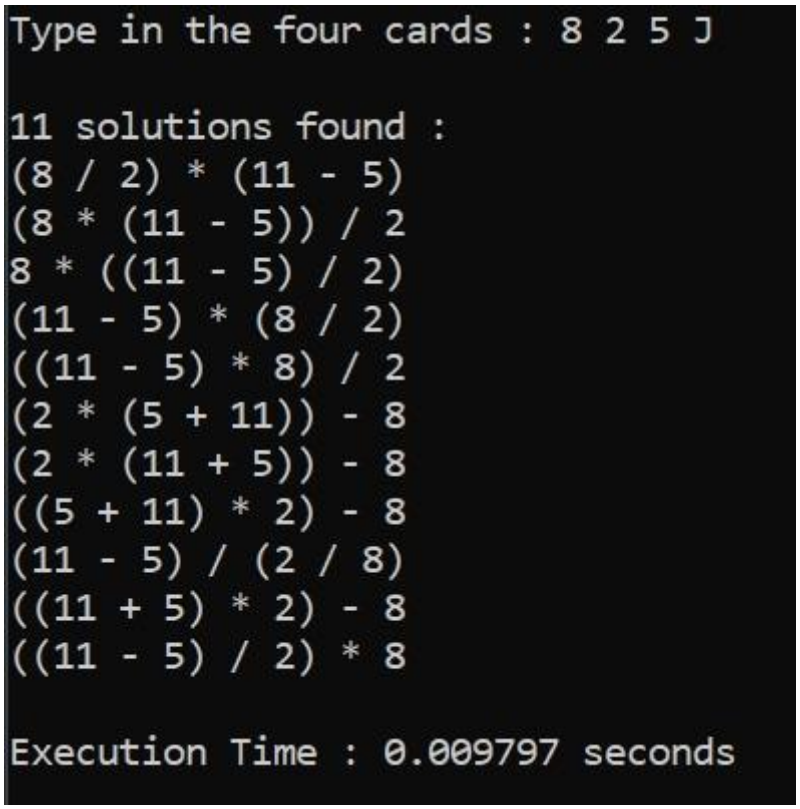

3. Test 3 (Input = A A A A atau 1 1 1 1)



4. Test 4 (Input = 6 6 6 6)




5. Test 5 (Input = 8 2 5 J atau 8 2 5 11)



6. Test 6 (Input = A 5 j Q atau 1 5 11 12)