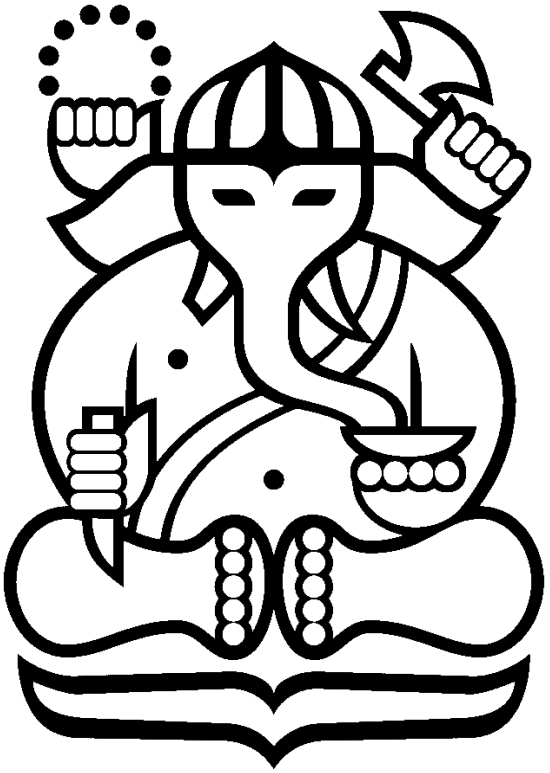
IF2211 Strategi Algoritma

IMPLEMENTASI ALGORITMA BRUTE FORCE DALAM PENCARIAN SOLUSI PERMAINAN KARTU 24

Laporan Tugas Kecil I

Disusun untuk memenuhi tugas mata kuliah IF2211 Strategi Algoritma pada Semester 2 (dua) Tahun Akademik 2022/2023



Disusun oleh:
Enrique Alifio Ditya
K02 / 13521142

PROGRAM STUDI TEKNIK INFORMATIKA SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA INSTITUT TEKNOLOGI BANDUNG

BANDUNG 2023

DAFTAR ISI

BAB I
PENDAHULUAN

Algoritma *brute force* adalah metode yang digunakan untuk mencari solusi dengan mencoba semua kemungkinan yang ada. Algoritma ini adalah metode yang paling sederhana dan paling tidak efisien, tetapi seringkali digunakan karena mudah diimplementasikan dan dapat digunakan untuk menyelesaikan berbagai jenis masalah. Secara umum, proses algoritma *brute force* dapat dibagi menjadi beberapa tahap:

1. Tentukan jenis masalah yang akan diselesaikan.
2. Buatlah daftar semua kemungkinan yang mungkin untuk masalah tersebut.
3. Lakukan pengecekan satu per satu pada setiap kemungkinan yang ada hingga ditemukan solusi yang sesuai.
4. Jika solusi tidak ditemukan setelah semua kemungkinan dicoba, maka algoritma akan mengakhiri prosesnya.

Kemampuan *brute force* dalam menyelesaikan masalah sangat bergantung pada jumlah kemungkinan yang ada. Semakin banyak kemungkinan yang ada maka semakin lama waktu yang diperlukan untuk menemukan solusi. Maka dari itu, algoritma *brute force* seringkali digunakan untuk memecahkan masalah yang relatif sederhana dan tidak mengandung penelusuran data banyak atau proses yang kompleks, sebab sangatlah mungkin terjadi *combinatorial explosion* dalam proses pencarian kemungkinan solusi.

Dalam tugas kecil ini, algoritma *brute force* digunakan dalam mencari solusi dalam permainan kartu 24. *Game* 24 merupakan permainan yang menguji kemampuan logika dan matematika seseorang. Dalam permainan ini, pemain diberi empat buah angka kartu dan ditugaskan untuk menemukan operasi matematika dasar seperti penjumlahan, pengurangan, perkalian, dan pembagian agar menghasilkan jumlah 24. Salah satu metode yang dapat digunakan dalam penemuan solusi permainan kartu 24 adalah dengan menggunakan algoritma *brute force*.

BAB II
ALGORITMA BRUTE FORCE

Algoritma *brute force* adalah metode yang digunakan untuk menemukan solusi dengan mencoba semua kemungkinan yang ada. Dalam hal ini, algoritma *brute forc*e akan mencoba seluruh operasi matematika yang mungkin dilakukan pada keempat buah kartu yang diberikan sampai menemukan solusi yang menghasilkan jumlah 24. Secara garis besar, penyelesaian dengan algoritma *brute force* dapat dilakukan sebagai berikut:

1. Inisialisasi empat buah kartu yang akan digunakan, baik dari masukan pengguna atau *generate* secara acak.
2. Lakukan permutasi di antara empat buah angka kartu tersebut dan simpan ke dalam suatu *set*.
3. Untuk setiap hasil permutasi, cari seluruh kemungkinan persamaan matematika dengan *operand* {+ , -, *, /, (,)} di antara empat angka kartu.
4. Evaluasi persamaan dan cek apabila menghasilkan 24. Jika iya, simpan persamaan dalam bentuk *string* ke dalam larik.
5. Apabila proses telah berakhir, yakni seluruh persamaan dari seluruh bentuk permutasi telah dievaluasi, cek isi larik. Apabila larik tidak kosong, maka solusi dengan empat angka kartu tersebut ada. Sebaliknya, jika larik tidak berisi maka solusi tidak ada.
6. Solusi yang disimpan dalam larik dapat di-*output*-kan jika solusi tersebut ada.

Pendekatan *brute force* dalam mencari seluruh kemungkinan solusi *game* 24 relatif sederhana, namun disertai dengan kelebihan dan kekurangannya. Pada bagian-bagian selanjutnya akan dilakukan deskripsi algoritma per bagian lebih dalam dan juga dilakukan analisis kompleksitas waktu untuk mengevaluasi performa dari algoritma ini.

BAB III
IMPLEMENTASI ALGORITMA

1. Deskripsi Source Program

Pada laporan ini, program ditulis dalam bahasa C++. *Source code* pada program ini dibagi menjadi dua bagian, yakni modul fungsi dan prosedur pendukung serta modul program utama. Modul fungsi dan prosedur pendukung mencakup sebagian besar dari proses pencarian solusi, mulai dari input kartu, pengacakan permutasi kartu, pencarian persamaan matematika, dan evaluasi tiap solusi. Program utama kemudian menggabungkan seluruh algoritma pada modul pendukung menjadi program utuh disertai penyajian tampilan CLI (*Command Line Interface*).

1.1 func.cpp

File *func.cpp* merupakan modul berisi seluruh algoritma pendukung, di antaranya mencakup fungsi dan prosedur sebagai berikut.

1.1.1 Include Library Pendukung

```
#include <iostream>
#include <sstream>
#include <fstream>
#include <ctime>
#include <set>
#include <cmath>
#include "func.h"

using std::vector;
using std::string;
using std::stringstream;
using std::getline;
using std::cin;
using std::cout;
using std::endl;
using std::set;
using std::fabs;
using std::to_string;
using std::ofstream;
```

Potongan kode di atas mengandung seluruh library yang digunakan sebagai *pre-requisite* dari pembuatan algoritma selanjutnya. Hal ini dilakukan untuk memudahkan dan mengoptimasi pembuatan kode program.

1.1.1 Inisiasi Kartu

```
vector<double> inputCards() {
    string input;
    cout << "Enter 4 card faces separated by spaces: ";
    getline(cin, input);
    vector<double> output = {};
    string token;
    stringstream ss(input);

    // Process input stream as tokenized strings separated by
spaces
    while (ss >> token) {
        if (token == "A") {
            output.push_back(1);
        } else if (token == "J") {
            output.push_back(11);
        } else if (token == "Q") {
            output.push_back(12);
        } else if (token == "K") {
            output.push_back(13);
        } else {
            int value = 0;
            for (char c : token) {
                if (c < '0' || c > '9') {
                    // Invalid character involved
                    cout << "Invalid input, only numbers 2-10
or J, Q, K, A are allowed.\n" << endl;
                    return vector<double>();
                } else {
                    value = value * 10 + (c - '0');
                }
            }

            if (value < 2 || value > 10) {
                // Invalid character/number involved
                cout << "Invalid input, only numbers 2-10 or
J, Q, K, A are allowed.\n" << endl;
                return vector<double>();
            } else {
                output.push_back(value);
            }
```

```
        }
    }

    if (output.size() != 4) {
        // Invalid card size or input format
        cout << "Invalid input, please enter 4 valid face
cards separated by spaces.\n" << endl;
        return vector<double>();
    }

    return output;
}

vector<double> generateRandomCards() {
    vector<double> output;

    // Generate random seed using current time
    srand(time(0));
    for (int i = 0; i < 4; i++) {
        int randCard = rand() % 13 + 1;
        output.push_back(randCard);
    }

    return output;
}
```

Pada bagian ini, dilakukan implementasi untuk menginisiasi kartu permainan dengan fungsi ‘inputCards()’ dan ‘generateRandomCards()’.

Fungsi ‘inputCards()’ meminta pengguna untuk memasukkan 4 kartu wajah yang dipisahkan oleh spasi, membaca input sebagai *string* menggunakan fungsi bawaan ‘getline’ dan ‘cin’, lalu memproses *string* input dengan menandainya dengan objek ‘stringstream’. Fungsi ini menggunakan *while loop* untuk menelusuri *string* token, dan memeriksa setiap token apabila merupakan kartu wajah yang valid yakni karakter A, J, Q, K atau angka di antara 2 dan 10. Jika token merupakan karakter valid, fungsi akan memetakan kartu ke nilai numerik yang sesuai (A=1, J=11, Q=12, K=13) dan memasukkannya ke dalam vektor. Jika token berupa angka di antara 2 dan 10, fungsi langsung memasukan nilai tersebut ke dalam vektor. Jika token tidak valid, fungsi akan mencetak pesan kesalahan dan mengembalikan vektor kosong. Setelah perulangan *while*, fungsi memeriksa apakah vektor keluaran berisi 4 elemen, jika tidak, pesan kesalahan akan tercetak dan mengembalikan vektor kosong.

Fungsi ‘generateRandomCards()’ menghasilkan empat kartu acak dengan *seed* dari fungsi ‘time(0)’, kemudian menggunakan *for loop* untuk menghasilkan empat bilangan bulat

acak antara 1 dan 13. Setiap bilangan bulat yang dihasilkan kemudian dimasukkan ke vektor dan dikembalikan.

1.1.2 Permutasi Kartu

```
void swapDoubles(double &a, double &b) {
    // Swap between doubles
    double temp = a;
    a = b;
    b = temp;
}

set<vector<double>> permuteCards(vector<double> &cards,
int l, int r) {
    // Permutations saved in a set of vectors
    set<vector<double>> permutations;

    // Base case: Last element of vector reached, save
permuted cards
    if (l == r) {
        permutations.insert(cards);
    } else {
    // Recursion: Permute sub sections of cards
        for (int i = l; i <= r; i++) {
            swapDoubles(cards[l], cards[i]);
            auto subPerm = permuteCards(cards, l + 1, r);
            permutations.insert(subPerm.begin(),
subPerm.end());
            swapDoubles(cards[l], cards[i]);
        }
    }

    return permutations;
}
```

Bagian ini terdiri dari dua fungsi, yakni ‘swapDoubles(double &a, double &b)’ dan‘permuteCards(vector<double> &cards, int l, int r)’.

‘swapDoubles(double &a, double &b)’ merupakan fungsi pembantu yang memiliki kegunaan untuk menukar dua variabel bertipe double yang di-*pass by reference*. Fungsi ini menggunakan variabel sementara bernama ‘temp’ untuk menyimpan nilai variabel ‘a’. Lalu, variabel ‘a’ diberi nilai dari variabel ‘b’, dan ‘b’ diberi nilai ‘temp’ yang mengandung nilai ‘a’ sebelumnya.

Fungsi ‘permuteCards(vector<double> &cards, int l, int r)’ menerima input berupa vektor berelemen double bernama ‘cards’ dan dua *integer* ‘l’ dan ‘r’. Fungsi kemudian akan mengembalikan suatu set berisi vektor yang masing-masing mengandung bentuk permutasi dari vektor ‘cards’. Fungsi ini diimplementasikan menggunakan rekursi dan kontainer set untuk menghasilkan dan menyimpan seluruh kemungkinan permutasi dari vektor ‘cards’.

Fungsi tersebut menggunakan kasus basis yakni nilai ‘l’ sama dengan ‘r’, artinya iterasi telah mencapai pada elemen terakhir vektor sehingga bentuk permutasi bisa disimpan dalam variabel set berelemen vektor bernama ‘permutations’ yang kemudian akan dikembalikan.

Apabila kasus basis belum tercapai, fungsi akan melakukan *loop* dari ‘l’ sampai ‘r’ dan untuk setiap iterasi akan dilakukan pemanggilan fungsi ‘swapDoubles’ untuk menukar nilai elemen vektor kartu pada indeks ‘l’ dengan indeks ‘i’. Fungsi tersebut kemudian akan dipanggil secara rekursif dengan masukan kartu yang sudah ditukar, kemudian akan ditukar kembali ke posisi semula setelah rekursi selesai.

1.1.3 Pencarian Persamaan Solusi

```
double eval(double num1, char op, double num2) {
    // Evaluate operations
    if (op == '+') {
        return num1 + num2;
    } else if (op == '-') {
        return num1 - num2;
    } else if (op == '*') {
        return num1 * num2;
    } else if (op == '/') {
        // Note: Zero division of doubles results in `inf`
        return num1 / num2;
    }
}

vector<string> getSolutions(vector<double> cards) {
    vector<string> solutions = {};
    string equation;

    vector<char> ops = {'+', '-', '*', '/'};

    // Iterate through all permutation of operations
    for (int i = 0; i < ops.size(); i++) {
        for (int j = 0; j < ops.size(); j++) {
            for (int k = 0; k < ops.size(); k++) {
```

```
                // Parentheses style 1: ((num1 op1 num2) op2
num3) op3 num 4
                if (fabs(eval(eval(eval(cards[0], ops[i],
cards[1]), ops[j], cards[2]), ops[k], cards[3]) - 24.0) <
1e-8) {
                    equation = "((" + to_string((int)
cards[0]) + " " + ops[i] + " " + to_string((int) cards[1]) +
") " + ops[j] + " " + to_string((int) cards[2]) + ") " +
ops[k] + " " + to_string((int) cards[3]);
                    solutions.push_back(equation);
                }

                // Parentheses style 2: (num1 op1 (num2 op2
num3)) op3 num4
                if (fabs(eval(eval(cards[0], ops[i],
eval(cards[1], ops[j], cards[2])), ops[k], cards[3]) - 24.0)
< 1e-8) {
                    equation = "(" + to_string((int)
cards[0]) + " " + ops[i] + " " + "(" + to_string((int)
cards[2]) + " " + ops[j] + " " + to_string((int) cards[3]) +
"))" + " " + ops[k] + " " + to_string((int) cards[3]);
                    solutions.push_back(equation);
                }

                // Parentheses style 3: (num1 op1 num2) op2
(num3 op3 num4)
                if (fabs(eval(eval(cards[0], ops[i],
cards[1]), ops[j], eval(cards[2], ops[k], cards[3])) - 24.0)
< 1e-8) {
                    equation = "(" + to_string((int)
cards[0]) + " " + ops[i] + " " + to_string((int) cards[1]) +
") " + ops[j] + " (" + to_string((int) cards[2]) + " " +
ops[k] + " " + to_string((int) cards[3]) + ")";
                    solutions.push_back(equation);
                }

                // Parentheses style 4: num1 op1 ((num2 op2
num3) op3 num4)
                if (fabs(eval(cards[0], ops[i],
eval(eval(cards[1], ops[j], cards[2]), ops[k], cards[3])) -
24.0) < 1e-8) {
                    equation = to_string((int) cards[0]) + "
" + ops[i] + " ((" + to_string((int) cards[1]) + " " + ops[j]
```

```
+ " " + to_string((int) cards[2]) + ") " + ops[k] + " " +
to_string((int) + cards[3]) + ")";
                    solutions.push_back(equation);
                }

                // Parentheses style 5: num1 op1 (num2 op2
(num3 op3 num4))
                if (fabs(eval(cards[0], ops[i],
eval(cards[1], ops[j], eval(cards[2], ops[k], cards[3]))) -
24.0) < 1e-8) {
                    equation = to_string((int) cards[0]) + "
" + ops[i] + " (" + to_string((int) cards[1]) + " " + ops[j]
+ " (" + to_string((int) cards[2]) + " " + ops[k] + " " +
to_string((int) cards[3]) + "))";
                    solutions.push_back(equation);
                }
            }
        }
    }

    return solutions;
}

vector<string> solve24(set<vector<double>> permCards) {
    vector<string> solutions;
    vector<string> permSols;

    // Sequentially iterate through a set of permutated cards
    for (auto it = permCards.begin(); it =/= permCards.end();
it++) {
        auto permVector = *it;
        permSols = getSolutions(permVector);
        solutions.insert(solutions.end(), permSols.begin(),
permSols.end());
    }

    return solutions;
}
```

Bagian kode ini berisi fungsi ‘eval’, ‘getSolutions’ dan ‘solve24’ yang bertujuan untuk mencari solusi pada *game* 24.

Fungsi ‘eval(double num1, char op, double num2)’ menerima tiga input berupa dua variabel bertipe double dan variabel operator bertipe char. Fungsi ini cukup *straightforward*, yakni mengembalikan hasil operasi aritmetika antara dua bilangan real yang dioperasikan dengan aturan tambah (+), kurang (-), kali (*), atau bagi (/).

Fungsi ‘getSolutions(vector<double> cards)’ menerima masukan berupa vektor berelemen double yang merepresentasikan empat buah kartu. Fungsi ini kemudian akan mengembalikan vektor berelemen *string* yakni mengandung persamaan matematika antara nomor kartu yang menghasilkan 24. Fungsi ini awalnya menginisialisasi sebuah vektor berelemen *string* kosong bernama ‘solutions’ dan variabel *string* bernama ‘equation’. Sebuah vektor berelemen karakter pun juga diinisialisasi bernama ‘ops’ dan berisi operator-operator aritmetika.

Fungsi ‘getSolutions’ menggunakan *nested for loop* untuk mengiterasi seluruh kombinasi dari operasi. Untuk setiap kombinasi operasi, fungsi ini menggunakan beberapa *if-statements* untuk mengecek apabila persamaan bisa menghasilkan 24. Apabila persamaan menghasilkan 24, fungsi kemudian akan memodelkan persamaan tersebut dalam bentuk *string* dalam variabel ‘equation’ yang kemudian akan disimpan pada vektor ‘solutions’. Pada tahap pengecekan evaluasi, digunakan fungsi bawaan ‘fabs’ (*floating absolute*) untuk mengecek apabila hasil sama dengan 24. Hasil pengurangan dari angka evaluasi dengan 24 kemudian akan dibandingkan dengan angka kecil (1e-8) untuk memperhitungkan kesalahan presisi dalam aritmatika *floating point*. Pada akhirnya, fungsi mengembalikan vektor yang berisi solusi persamaan yang menghasilkan 24 dari empat kartu yang diberikan berdasarkan aturan operasi matematika.

Akhir dari proses ini dikompilasi dalam fungsi ‘solve24’ yang menerima input berupa set dari kartu yang telah dipermutasi sebelumnya bernama ‘permCards’. Fungsi ini menggunakan *for loop* untuk menelusuri set kartu hasil permutasi dan akan memanggil fungsi ‘getSolutions’ untuk mendapatkan persamaan solusi untuk tiap-tiap bentuk permutasi. Fungsi kemudian akan meng-konkatenasi tiap solusi dan dikumpulkan dalam bentuk vektor berelemen *string* di variabel ‘solutions’. Setelah iterasi selesai, vektor yang berisi seluruh solusi akan dikembalikan.

1.1.4 Fungsi Output

```
void printCards(vector<double> cards) {
    // Iterate through vector to output all cards
    for (int i = 0; i < cards.size(); i++) {
        if (cards[i] == 1) {
            cout << "A ";
        } else if (cards[i] == 11) {
            cout << "J ";
        } else if (cards[i] == 12) {
```

```
            cout << "Q ";
        } else if (cards[i] == 13) {
            cout << "K ";
        } else {
            cout << (int) cards[i] << " ";
        }
    }
    cout << endl;
}

void printSolutions(vector<string> solutions) {
    // Outputs solution vector
    for (int i = 0; i < solutions.size(); i++) {
        cout << (i+1) << ". " << solutions[i] << endl;
    }
}

string stringifyCardVector(vector<double> vec) {
    string strVec = "";
    for (int i = 0; i < vec.size(); i++) {
        if ((int) vec[i] == 1) {
            strVec += "A";
        } else if ((int) vec[i] == 11) {
            strVec += "J";
        } else if ((int) vec[i] == 12) {
            strVec += "Q";
        } else if ((int) vec[i] == 13) {
            strVec += "K";
        } else {
            strVec += to_string((int) vec[i]);
        }
        strVec += " ";
    }

    return strVec;
}

void saveVectorToFile(const vector<string> &vec, const string
&fileName, const string cards) {
    ofstream outFile;
    outFile.open(fileName);
    outFile << "Cards: " << cards << endl << endl;
    for (const auto &line : vec) {
```

```
        outFile << line << endl;
    }
    outFile.close();
}
```

Pada bagian ini, didefinisikan fungsi dan prosedur yang menampilkan *output* pada layar atau menyimpan *output* pada *file*. Fungsi ‘printCards’ dan ‘printSolutions’ masing-masing bertujuan untuk menampilkan konten dari vektor kartu dan vektor solusi ke layar, sedangkan fungsi ‘stringifyCardVector’ dan ‘saveVectorToFile’ bertanggungjawab dalam menyimpan hasil keluaran program ke dalam file pilihan.

1.2 main.cpp

File *main.cpp* cukup *self descripting,* yakni berisi program utama dari program yang dapat dijalankan oleh pengguna, dan menggunakan proses-proses yang sudah diimplementasikan pada modul pendukung *func.cpp* sebelumnya. Berikut adalah kode dari program utama.

```
// Main program

#include <iostream>
#include <limits>
#include <chrono>
#include "func.h"

using std::cin;
using std::cout;
using std::endl;
using std::numeric_limits;
using std::streamsize;
using std::string;
using std::to_string;
using std::getline;

int main() {

    int option;
    vector<double> cards;

    cout << "<<< Welcome to the 24 Game Solver >>>\n" <<
endl;
```

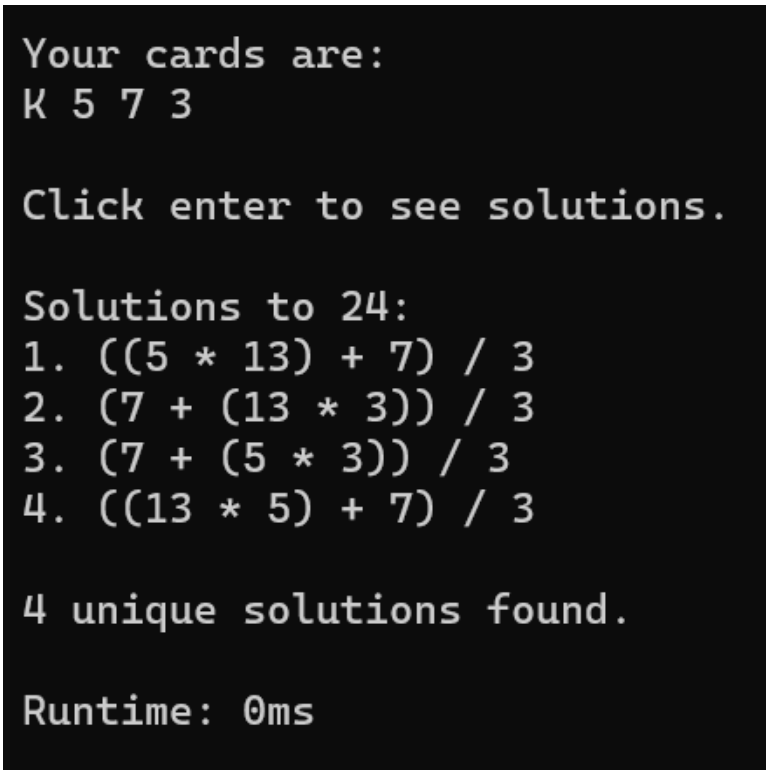
```
    // Loop until program is terminated
    bool running = true;

    while (running) {
        do {
            cout << "How would You like your cards
generated?" << endl;
            cout << "1. Input manually." << endl;
            cout << "2. Randomly generated.\n" << endl;
            cout << "Option: ";

            cin >> option;

            cout << endl;

            // Handle invalid option
            if (cin.fail() || (option ≠ 1 && option ≠ 2)) {
                cout << "Invalid option, please pick either 1
or 2.\n" << endl;
                cin.clear();
            }

            // Clear input stream
            cin.ignore(numeric_limits<streamsize>::max(),
'\n');

        } while (option ≠ 1 && option ≠ 2);

        if (option == 1) {
            // Manual input
            cards = inputCards();
            while (cards.size() ≠ 4) {
                cards = inputCards();
            }
        } else {
            // Random generated
            cards = generateRandomCards();
        }

        cout << "Your cards are:" << endl;
        printCards(cards);
        cout << endl;
```

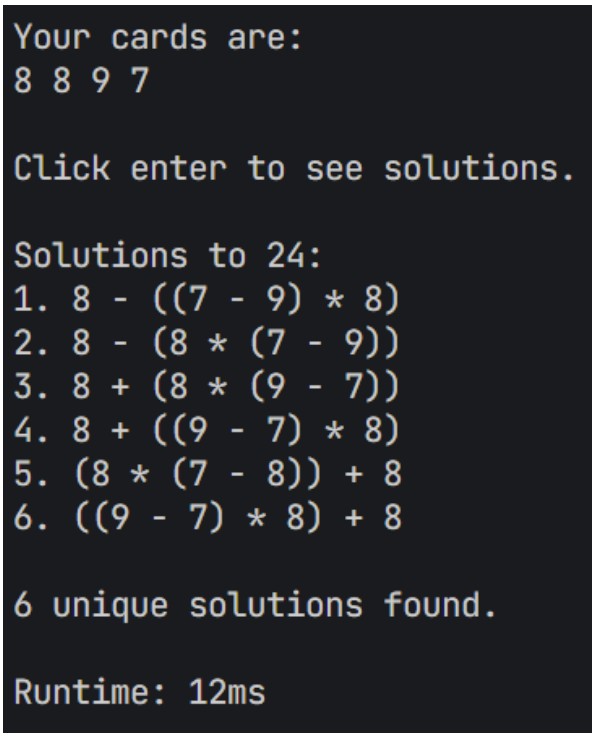
```
        string blank;
        cout << "Click enter to see solutions." << endl;
        getline(cin, blank);

        // Start clock
        auto start =
std::chrono::high_resolution_clock::now();

        // Get all possible card permutations
        auto perm = permuteCards(cards, 0, cards.size()-1);

        // Get solutions in vector of strings
        vector<string> solution = solve24(perm);

        if (solution.size() ≠ 0) {
            // Solutions found
            cout << "Solutions to 24:" << endl;
            printSolutions(solution);
            cout << endl << solution.size() << " unique
solutions found.\n" << endl;
        } else {
            // No solutions
            cout << "No solutions found for the given card
hand.\n" << endl;
        }

        // End clock
        auto end = std::chrono::high_resolution_clock::now();
        auto duration =
std::chrono::duration_cast<std::chrono::milliseconds>(end -
start);

        cout << "Runtime: " << duration.count() << "ms\n" <<
endl;

        // Option to save the solutions
        if (solution.size() ≠ 0) {
            char saveOpt;
            do {
                cout << "Would you like to save the solution?
[y/n]\n\nOption: ";
                cin >> saveOpt;
```

```
                cout << endl;

                if (saveOpt == 'y') {
                    // Note: ".txt" extension has to be
entered in order to save as a text file
                    string filename = "";
                    cout << "Enter save file name: ";
                    cin >> filename;
                    cout << endl;
                    saveVectorToFile(solution, filename,
stringifyCardVector(cards));
                    cout << "Solution saved in " << filename
<< endl << endl;
                }
            } while (saveOpt ≠ 'y' && saveOpt ≠ 'n');
        }

        // Option to re-run or terminate
        char rerun;
        do {
            cout << "Would you like to try the game again?
[y/n]\n\nOption: ";
            cin >> rerun;
            cout << endl;

            if (rerun == 'n') {
                running = false;
                cout << "Goodbye!" << endl;
            }
        } while (rerun ≠ 'y' && rerun ≠ 'n');
    }

    return 0;
}
```

Program utama ini memiliki alur yang cukup mudah untuk diikuti. Pada awal program, pengguna akan diminta opsi untuk menginput kartu baik secara manual maupun acak. Program kemudian melakukan permutasi dari kartu tersebut dan mengkalkulasi seluruh kemungkinan solusi. Keluaran pada program adalah seluruh solusi disertai dengan waktu *runtime* dan juga jumlah solusi yang didapatkan. Pada akhir program, pengguna diberikan opsi untuk menyimpan *output* dan memainkan kembali program solver 24.

2. *Screenshot Contoh Input dan Output*





```
Your cards are:
K A J 4

Click enter to see solutions.

No solutions found for the given card hand.

Runtime: 0ms
```

```
Your cards are:
J Q 4 7

Click enter to see solutions.

Solutions to 24:
1. 4 * ((7 + 11) - 12)
2. 4 * (7 + (11 - 12))
3. 4 * ((7 - 12) + 11)
4. 4 * (7 - (12 - 11))
5. 4 * ((11 + 7) - 12)
6. 4 * (11 + (7 - 12))
7. 4 * ((11 - 12) + 7)
8. 4 * (11 - (12 - 7))
9. ((7 + 11) - 12) * 4
10. (7 + (12 - 4)) * 4
11. ((7 - 12) + 11) * 4
12. (7 - (11 - 4)) * 4
13. ((11 + 7) - 12) * 4
14. (11 + (12 - 4)) * 4
15. ((11 - 12) + 7) * 4
16. (11 - (7 - 4)) * 4

16 unique solutions found.

Runtime: 15ms
```

```
Enter 4 card faces separated by spaces: 6 6 6 6
Your cards are:
6 6 6 6

Click enter to see solutions.

Solutions to 24:
1. ((6 + 6) + 6) + 6
2. (6 + (6 + 6)) + 6
3. (6 + 6) + (6 + 6)
4. 6 + ((6 + 6) + 6)
5. 6 + (6 + (6 + 6))
6. (6 * 6) - (6 + 6)
7. ((6 * 6) - 6) - 6

7 unique solutions found.

Runtime: 0ms
```

```
Enter 4 card faces separated by spaces: 9 2 8 7
Your cards are:
9 2 8 7

Click enter to see solutions.

Solutions to 24:
1. (2 * (9 + 8)) - 8
2. (2 * (7 + 8)) - 8
3. ((7 + 9) * 2) - 8
4. ((9 + 7) * 2) - 8

4 unique solutions found.

Runtime: 11ms
```

```
Enter 4 card faces separated by spaces: 3 10 8 Q
Your cards are:
3 10 8 Q

Click enter to see solutions.

Solutions to 24:
1. (10 / (3 - 12)) * 12
2. 10 / ((8 - 3) / 12)
3. (10 * 12) / (8 - 3)
4. 10 * (12 / (8 - 3))
5. (12 / (3 - 10)) * 10
6. 12 / ((8 - 3) / 10)
7. (12 * 10) / (8 - 3)
8. 12 * (10 / (8 - 3))

8 unique solutions found.

Runtime: 0ms
```

```
Enter 4 card faces separated by spaces: A A A A
Your cards are:
A A A A

Click enter to see solutions.

No solutions found for the given card hand.

Runtime: 0ms
```

BAB IV
ANALISIS KOMPLEKSITAS

Dalam pencarian solusi pada permainan kartu 24, program ini melewati tiga fase utama yakni permutasi kartu, pemodelan persamaan matematika, dan evaluasi persamaan untuk seluruh bentuk permutasi. Tiga fase ini masing-masing didukung oleh fungsi ‘permuteCards’, ‘getSolutions’, dan ‘solve24’.

Fungsi ‘permuteCards’ menggunakan algoritma terkenal yang disebut Algoritma Heap untuk melakukan permutasi pada elemen-elemen larik. Algoritma ini didasarkan pada pengamatan bahwa untuk menghasilkan seluruh permutasi dari sebuah *list* dengan panjang n, dapat dibuat seluruh permutasi dari sebuah *list* dengan panjang n-1, dan kemudian memasukkan elemen terakhir ke semua kemungkinan posisi dalam permutasi tersebut. Algoritma Heap adalah algoritme rekursif sederhana yang menghasilkan semua kemungkinan permutasi larik dengan menukar elemen, yang menjadikannya hemat ruang karena tidak memerlukan *storage* tambahan apa pun untuk permutasi. Kompleksitas waktu dari implementasi Algoritma Heap pada program ini adalah O(n!), sebab algoritma menghasilkan seluruh kemungkinan permutasi dari larik berisi n elemen.

Fungsi ‘getSolutions’ merupakan fungsi untuk mendapatkan seluruh kemungkinan solusi untuk satu buah himpunan empat kartu. Fungsi ini memiliki kompleksitas waktu O(n^3) dengan n berupa jumlah operator. Hal ini disebabkan fungsi menggunakan *nested for-loop* sebanyak tiga lapisan, satu untuk setiap operator. Loop terdalam di-iterasi sebanyak n kali, sedangkan loop tengah dan terluar masing-masing juga sebanyak n kali. Maka dari itu, total dari iterasi adalah sebanyak n*n*n = n^3. Selain itu, dapat diperhitungkan pula penggunaan *if-statement* untuk mengecek hasil dari tiap kombinasi persamaan. Tiap *if-statement* berwaktu konstan sehingga memiliki kompleksitas waktu O(1) untuk satu kali iterasi. Kompleksitas waktu dari fungsi ‘getSolutions’ juga dapat dipandang sebagai O(4^3), sebab jumlah operasi selalu tetap yakni berjumlah 4.

Fungsi ‘solve24’ menelusuri tiap elemen set hasil permutasi kartu dan kemudian akan mengevaluasi solusi untuk tiap bentuk permutasi, sehingga memiliki kompleksitas waktu O(n*m) dengan n berupa jumlah permutasi kartu dan m berupa jumlah operasi yang dilakukan dalam pemanggilan fungsi ‘getSolutions’.

Bagian *output* didukung oleh beberapa fungsi yakni ‘printCards’, ‘printSolutions’, ‘stringifyCardVector’, dan ‘saveVectorToFile’ yang masing-masing berjalan pada waktu linier sehingga memiliki kompleksitas waktu O(n) dengan n berupa jumlah elemen dalam struktur data yang ditelusuri.

LAMPIRAN

Link Repository Github

https://github.com/AlifioDitya/Tucil1_13521142

Tabel Keselesaian

Poin	Ya	Tidak
1. Program berhasil dikompilasi tanpa kesalahan	✓	
2. Program berhasil *running*	✓	
3. Program dapat membaca input / *generate* sendiri dan memberikan luaran	✓	
4. Solusi yang diberikan program memenuhi (berhasil mencapai 24)	✓	
5. Program dapat menyimpan solusi dalam file teks	✓	