

IF2211 Strategi Algoritma

**PENYELESAIAN PERMAINAN KARTU 24 DENGAN
ALGORITMA *BRUTE FORCE***

Laporan Tugas Kecil 1

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



Oleh

Chiquita Ahsanunnisa

13521129

**PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
BANDUNG
2023**

DAFTAR ISI

DAFTAR ISI.....	i
BAB I DESKRIPSI MASALAH.....	1
1.1 Permainan Kartu 24.....	1
1.2 Spesifikasi Program	1
1.2.1 <i>Input</i>	1
1.2.2 <i>Output</i>	1
BAB II LANDASAN TEORI	2
2.1 Algoritma	2
2.2 Algoritma <i>Brute Force</i>	2
BAB III IMPLEMENTASI.....	3
3.1 Ide Algoritma	3
3.2 Implementasi Algoritma.....	4
3.3 <i>Source Code</i> dalam Bahasa Java.....	5
3.3.1 Kelas InputHandler (InputHandler.java).....	6
3.3.2 Kelas Deck (Deck.java)	7
3.3.3 Kelas OutputHandler (OutputHandler.java)	9
3.3.4 Kelas App (App.java).....	11
BAB IV UJI COBA	13
4.1 <i>Custom Deck</i>	13
4.2 <i>Random Deck</i>	18
LAMPIRAN.....	24
Lampiran 1 Link Repository GitHub	24
Lampiran 2 Tampilan Program	24
Lampiran 3 Tabel <i>Check List</i> Poin.....	26
DAFTAR REFERENSI	27

BAB I

DESKRIPSI MASALAH

1.1 Permainan Kartu 24

Permainan Kartu 24 adalah permainan menyusun kombinasi empat kartu remi secara aritmatik sehingga didapatkan hasil akhir berupa 24. Kartu remi sendiri terdiri dari empat simbol kartu, yaitu sekop, hati, wajik, dan keriting. Setiap simbol kartu terdiri dari 13 kartu yaitu As (A), 2, 3, 4, 5, 6, 7, 8, 9, 10, Jack (J), Queen (Q), dan King (K), sehingga total terdapat 52 kartu. As bernilai 1, Jack bernilai 11, Queen bernilai 12, King bernilai 13, sedangkan kartu bilangan memiliki nilai dari bilangan itu sendiri.

Penyusunan kombinasi empat kartu secara aritmatik untuk mencapai hasil akhir bernilai 24 dapat dilakukan dengan menggunakan operasi penjumlahan (+), pengurangan (-), perkalian (\times), pembagian (/) dan tanda kurung (). Setiap kartu harus digunakan tepat sekali dan urutan penggunaannya bebas.

1.2 Spesifikasi Program

Program penyelesaian permainan Kartu 24 yang ditulis memanfaatkan algoritma *brute force* untuk mencari seluruh kemungkinan solusi. Program ditulis dalam bahasa C/C++/Java. Berikut adalah detail dari spesifikasi program yang harus dibuat.

1.2.1 Input

Pengguna dapat memilih untuk memasukkan empat kartu sesuai keinginannya (*custom deck*) atau meng-*generate* empat kartu secara random (*random deck*). Kartu yang masuk ke *deck* harus *valid*, yaitu merupakan angka atau huruf di antara A, 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, dan/atau K. Masukan harus divalidasi hingga didapat masukan yang sesuai.

1.2.2 Output

Program menampilkan:

1. Banyaknya solusi yang ditemukan.
2. Solusi permainan Kartu 24, yang dapat dituliskan ke dalam *file* .txt juga (ada konfirmasi penulisan jawaban ke *file*).
3. Waktu eksekusi algoritma *brute force*.

BAB II LANDASAN TEORI

2.1 Algoritma

Algoritma adalah urutan langkah-langkah yang jelas dan terstruktur dalam menyelesaikan suatu permasalahan.

2.2 Algoritma *Brute Force*

Algoritma *brute force* adalah pendekatan yang lempang (*straightforward*) dalam menyelesaikan suatu masalah. Pada umumnya, algoritma ini secara langsung didasarkan pada pernyataan yang ada pada persoalan dan definisi atau konsep yang dilibatkan. Algoritma *brute force* digunakan untuk memecahkan persoalan yang sangat sederhana, secara langsung, dan sudah jelas caranya.

Algoritma *brute force* memiliki karakteristik sebagai berikut.

1. Umumnya tidak “cerdas” dan tidak mangkus karena membutuhkan volume komputasi yang relatif besar dan waktu eksekusi yang relatif lama.
2. Lebih cocok untuk persoalan yang masukannya sedikit.
3. Dapat menyelesaikan hampir semua persoalan.

Algoritma *brute force* memiliki kelebihan sebagai berikut.

1. Dapat diterapkan untuk memecahkan hampir sebagian besar permasalahan.
2. Sederhana dan mudah dimengerti.
3. Cukup layak untuk memecahkan masalah penting seperti perkalian matriks, pencarian, dan lain-lain.
4. Menghasilkan algoritma yang baku atau standar untuk tugas-tugas komputasi.

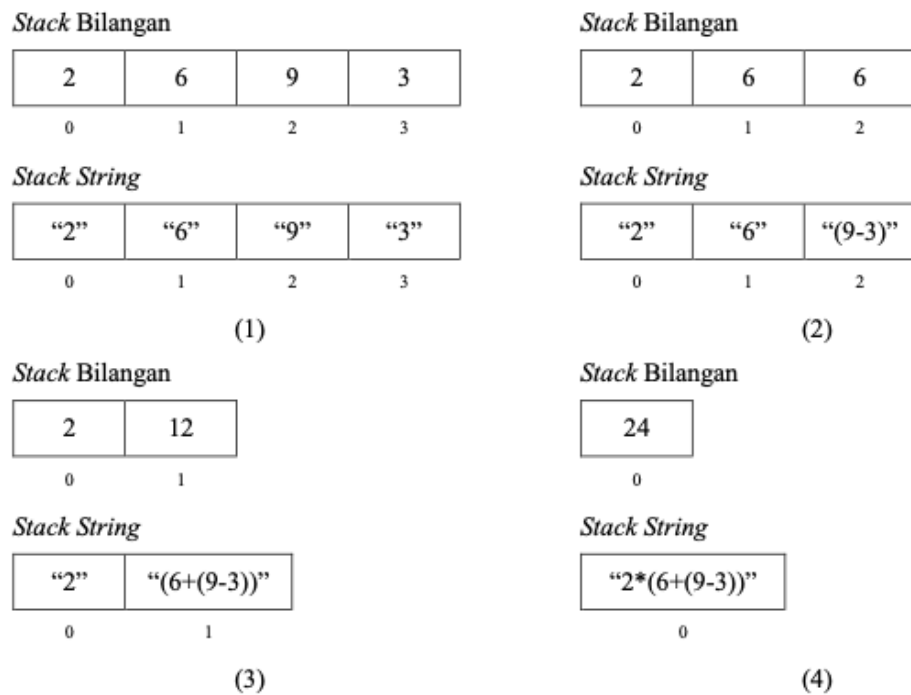
Di sisi lain, algoritma *brute force* juga memiliki kekurangan:

1. Jarang menghasilkan algoritma yang mangkus.
2. Umumnya lambat jika masukannya besar.
3. Tidak kreatif dan tidak konstruktif.

BAB III IMPLEMENTASI

3.1 Ide Algoritma

Algoritma *brute force* yang dibuat penulis terinspirasi dari sistem kerja struktur data *stack*. *Stack* adalah senarai dengan aturan penghapusan elemen (*pop*) hanya bisa dilakukan dari elemen paling atas dan pemasukan elemen (*push*) hanya bisa dilakukan di atas elemen paling atas.



Gambar 3.1 *Stack Bilangan* dan *Stack String*

Pada Gambar 3.1, ada dua jenis *stack*, yaitu *stack* bilangan yang berfungsi sebagai tempat proses aritmatika bilangan yang ada dan *stack string* yang berfungsi membentuk *string* yang mewakili proses aritmatika yang terjadi. Untuk melakukan operasi aritmatika, ambil elemen (*pop*) dari *stack* dua kali, lalu pilih operasi aritmatika yang akan dilakukan, lalu masukkan elemen (*push*) nilai baru yang merupakan hasil dari operasi aritmatika tersebut.

Gambar 3.1 (1) mewakili susunan awal keempat kartu (*deck*) yang belum dioperasikan sama sekali. Elemen bernilai 3 dan 9 di-*pop* dari *stack* bilangan. Begitu pula dengan *stack string*-nya. Dalam kasus ini, dipilih operasi pengurangan, yaitu $9-3$ yang bernilai 6. Elemen bernilai 6 pun di-*push* ke *stack* bilangan dan ekspresi *string* " $(9-3)$ " pun di-*push* ke *stack string* sehingga menghasilkan konfigurasi seperti pada Gambar 3.1 (2). Hal yang sama terus dilakukan sehingga elemen kedua *stack* tunggal, seperti Gambar 3.1 (4). Jika elemennya sudah tunggal,

cukup cek apakah elemen *stack* bilangan bernilai 24 atau tidak. Jika elemennya bernilai 24, *string* terakhir yang merupakan elemen *stack string* adalah rumusan yang benar (salah satu solusi).

Pada kasus di atas, elemen yang dapat dioperasikan (di-*pop*) hanya elemen yang terakhir, sesuai aturan *stack*. Namun, untuk menghasilkan segala solusi yang mungkin, operasi *pop* dapat dimodifikasi sehingga dapat dilakukan untuk menghapus elemen *stack* pada indeks tertentu (tidak harus di indeks terakhir).

Secara singkat, ide dari algoritma yang dibuat adalah memilih dua elemen dari tiap tahap (permutasi 2 dari panjang *stack*) dan menghapus kedua elemen tersebut dari *stack*, memilih salah satu operator (dari empat operator yang ada), menghitung hasil operasi dan membuat ekspresinya dalam *string* serta mem-*push* elemen hasil operasi dan ekspresinya ke *stack*. Kemudian, jika elemen kedua *stack* sudah tunggal, cukup cek apakah hasilnya 24 atau bukan. Jika iya, ekspresi yang terdapat pada *stack string* adalah salah satu jawaban. Dalam proses yang mengandung “pemilihan”, semua kemungkinan pilihan akan dicoba dan dievaluasi.

3.2 Implementasi Algoritma

Pada implementasinya, struktur data *stack* tidak benar-benar dipakai di algoritmanya, cukup konsepnya saja. Struktur data yang dipakai adalah *array* statik yang memiliki panjang bervariasi. Ada *array* yang panjangnya 4 (selanjutnya disebut `arrNum4` dan `arrStr4`), 3 (selanjutnya disebut `arrNum3` dan `arrStr3`), 2 (selanjutnya disebut `arrNum2` dan `arrStr2`), dan 1 (selanjutnya disebut `arrNum1` dan `arrStr1`). Setiap jenis *array* mewakili kondisi *stack* di setiap tahap. Hal ini dipertimbangkan karena penulis berpendapat bahwa proses penghapusan elemen dari *array* (*stack*) akan mempersulit algoritma yang dibangun. Oleh karena itu, dibanding menghapus, algoritma yang dipakai akan “menimpa” nilai *array* yang sebelumnya. Selain *array*, penulis juga menggunakan struktur data *set* untuk menyimpan seluruh solusi yang dihasilkan dari algoritma *brute force*. Hal ini bertujuan agar solusi yang disimpan bersifat unik.

Berikut adalah langkah-langkah dari algoritma *brute force* untuk penyelesaian permainan Kartu 24.

1. Pilih dua bilangan dari `arrNum4`.
2. Pilih satu operator dari empat operator yang ada.
3. Operasikan dua bilangan yang dipilih sebelumnya dengan operator terpilih. Hitung hasilnya dan buat ekspresi *string*-nya. Salin `arrNum4` ke `arrNum3` dan salin `arrStr4` ke `arrStr3`, kecuali elemen yang telah dioperasikan. Isi nilai elemen terakhir dari `arrNum3` dengan hasil operasi dan elemen terakhir dari `arrStr3` dengan ekspresi *string* yang dihasilkan.
4. Pilih dua bilangan dari `arrNum3`.

5. Pilih satu operator dari empat operator yang ada.
6. Operasikan dua bilangan yang dipilih sebelumnya dengan operator terpilih. Hitung hasilnya dan buat ekspresi *string*-nya. Salin `arrNum3` ke `arrNum2` dan salin `arrStr3` ke `arrStr2`, kecuali elemen yang telah dioperasikan. Isi nilai elemen terakhir dari `arrNum2` dengan hasil operasi dan elemen terakhir dari `arrStr2` dengan ekspresi *string* yang dihasilkan.
7. Pilih dua bilangan dari `arrNum2`.
8. Pilih satu operator dari empat operator yang ada.
9. Operasikan dua bilangan yang dipilih sebelumnya dengan operator terpilih. Hitung hasilnya dan buat ekspresi *string*-nya. Salin `arrNum2` ke `arrNum1` dan salin `arrStr2` ke `arrStr1`, kecuali elemen yang telah dioperasikan. Isi nilai elemen terakhir dari `arrNum1` dengan hasil operasi dan elemen terakhir dari `arrStr1` dengan ekspresi *string* yang dihasilkan.
10. Evaluasi apakah elemen `arrNum1` bernilai 24. Jika iya, masukkan elemen pada `arrStr1` ke *set* solusi.
11. Ulangi dari langkah 8 hingga semua operator terpakai.
12. Ulangi dari langkah 7 hingga semua permutasi dua elemen dari `arrNum2` terpakai.
13. Ulangi dari langkah 5 hingga semua operator terpakai.
14. Ulangi dari langkah 4 hingga semua permutasi dua elemen dari `arrNum3` terpakai.
15. Ulangi dari langkah 2 hingga semua operator terpakai.
16. Ulangi dari langkah 1 hingga semua permutasi dua elemen dari `arrNum4` terpakai.

Banyaknya perintah “ulangi” sejatinya adalah simbol dari *nested for loop* yang digunakan untuk mencari segala kemungkinan kombinasi angka, operasi, dan kurung yang ada. Jika ditelaah lebih dalam, algoritma di atas dapat dipersingkat dan diperumum dengan memanfaatkan prosedur rekursif seperti yang tertera pada *source code*, kelas `Deck`, prosedur `solve`.

Perhatikan bahwa dengan algoritma di atas dan skema pembentukan ekspresi *string* seperti pada Gambar 3.1, dimungkinkan adanya jawaban yang bermakna sama secara matematis, namun ekspresi *string*-nya berbeda. Sebagai contoh, jika kartu yang digunakan adalah 6, 6, 6, dan 6, program akan menghasilkan “(6 + 6) + (6 + 6)” dan “((6 + 6) + 6) + 6” sebagai solusi dan menganggapnya sebagai dua solusi yang berbeda, meskipun secara matematis dianggap sama.

3.3 Source Code dalam Bahasa Java

Seluruh *source code* terletak pada *folder* `src`. Program utama terletak pada kelas `App`.

3.3.1 Kelas InputHandler (InputHandler.java)

```
import java.util.Scanner;
import java.util.Random;
import java.util.Arrays;

public class InputHandler {
    /* Mengurus masalah input */
    private static final String[] validCards = {"A", "2", "3", "4", "5", "6", "7",
"8", "9", "10", "J", "Q", "K"};
    private static Scanner in = new Scanner(System.in);

    public static int rangedInput(int a, int b, String question) {
        /* Memvalidasi masukan pilihan antara a dan b (inklusif) */
        String[] prevAns;
        int ans;

        do {
            System.out.println(question);
            System.out.print("Your answer: ");
            prevAns = in.nextLine().split(" ");

            if (prevAns.length == 1) {
                try {
                    ans = Integer.parseInt(prevAns[0]);
                } catch (NumberFormatException e) {
                    ans = a - 1;
                }
            } else {
                ans = a - 1;
            }

            if (!(ans >= a && ans <= b)) {
                System.out.println("Invalid input, please try again.\n");
            }
        } while (!(ans >= a && ans <= b));

        return ans;
    }

    public static int inputMethod() {
        /* Memilih cara memasukkan deck, random atau custom */
        return rangedInput(1, 2, "\nChoose input method:\n1. Custom Deck\n2.
Random Deck");
    }

    public static int contConfirm() {
        /* Memilih melanjutkan program atau tidak */
        return rangedInput(1, 2, "\nContinue?\n1. Yes\n2. No");
    }

    private static boolean validArr(String[] arrStr) {
        /* Memvalidasi apakah array merupakan sebuah deck kartu yang valid */
        boolean valid = true;

        if (arrStr.length == 4) {
            for (int i = 0; i < 4; i++) {
                if (!(Arrays.asList(validCards).contains(arrStr[i]))) {
                    valid = false;
                    break;
                }
            }
        }
    }
}
```



```

    }
    }
    } else {
        valid = false;
    }

    return valid;
}

public static void readInput(String[] deck) {
    /* Membaca masukan deck custom */
    System.out.print("Insert deck: ");
    String[] arrOfStr = in.nextLine().split(" ");

    while (!validArr(arrOfStr)) {
        System.out.println("Invalid input, please try again.");
        System.out.print("Insert deck: ");
        arrOfStr = in.nextLine().split(" ");
    }

    for (int i = 0; i < 4; i++) {
        deck[i] = arrOfStr[i];
    }
}

public static void randInput(String[] deck) {
    /* Membuat deck random */
    Random rand = new Random();
    for (int i = 0; i < 4; i++) {
        deck[i] = validCards[rand.nextInt(13)];
    }
}

private static double convertStr(String card) {
    /* Mengubah kartu menjadi nilai angka */
    if (card.equals("A")) {
        return 1;
    } else if (card.equals("J")) {
        return 11;
    } else if (card.equals("Q")) {
        return 12;
    } else if (card.equals("K")) {
        return 13;
    } else {
        return Double.parseDouble(card);
    }
}

public static void convertDeck(String[] deck, double[] cardNum) {
    /* Mengubah setiap kartu dalam deck menjadi nilai angkanya */
    for (int i = 0; i < 4; i++) {
        cardNum[i] = convertStr(deck[i]);
    }
}
}

```

3.3.2 Kelas Deck (Deck.java)

```
import java.util.HashSet;
```

```

public class Deck {
    /* Mengurus proses pada deck */
    private static final char[] opList = {'+', '-', '*', '/'};
    private static double[] arrNum3 = new double[3];
    private static double[] arrNum2 = new double[2];
    private static double[] arrNum1 = new double[1];
    private static String[] arrStr3 = new String[3];
    private static String[] arrStr2 = new String[2];
    private static String[] arrStr1 = new String[1];

    private static double opSwitch(double a, double b, int op) {
        /* Mengembalikan nilai 'a op b', op menunjukkan indeks pada array opList
        * Prekondisi: tidak ada pembagian dengan nol */
        switch (op) {
            case 0:
                return a + b;
            case 1:
                return a - b;
            case 2:
                return a * b;
            default: // case 3
                return a / b;
        }
    }

    private static String createExp(String exp1, String exp2, int op, boolean
last) {
        /* Mengembalikan string "a op b" atau "(a op b)", op menunjukkan indeks
pada array opList
        * Prekondisi: tidak ada pembagian dengan nol */
        String exp = exp1 + " " + opList[op] + " " + exp2;
        if (!last) {
            exp = "(" + exp + ")";
        }
        return exp;
    }

    public static String stringDeck(String[] deck) {
        /* Mengubah array deck menjadi satu string, tiap kartu dipisahkan oleh
spasi */
        String str = "";
        for (int i = 0; i < 3; i++) {
            str += (deck[i] + " ");
        }
        str += deck[3];
        return str;
    }

    private static void copyExc(int level, double[] precNum, String[] precStr,
double[] newNum, String[] newStr, int i, int j, int op, boolean last) {
        /* Meng-copy isi array prec ke new, dengan menghapus nilai yang
dioperasikan dan menambahkan nilai hasil operasi */
        int ctr = 0;
        if (level > 2) {
            for (int k = 0; k < level; k++) {
                if (k != i && k != j) {
                    newNum[ctr] = precNum[k];
                    newStr[ctr] = precStr[k];
                    ctr++;
                }
            }
        }
    }
}

```



```

public class OutputHandler {
    /* Mengurus masalah output */
    private static Scanner in = new Scanner(System.in);

    public static void outputTerm(HashSet<String> setStr) {
        /* Menampilkan jawaban di terminal */
        if (setStr.size() == 0) {
            System.out.println("\nNo solution found");
        } else {
            System.out.println("\n" + setStr.size() + " solution(s) found");

            for (String ans : setStr) {
                System.out.println(ans);
            }
        }
    }

    public static int outputMethod() {
        /* Memilih menulis jawaban ke file atau tidak */
        return InputHandler.rangedInput(1, 2, "\nWrite the answer to file?\n1.
Yes\n2. No");
    }

    public static void writeAnsFile(String[] deck, HashSet<String> setStr, long
exec) {
        /* Menulis jawaban ke file .txt */
        FileWriter fw = null;
        BufferedWriter buff = null;

        System.out.print("\nFile name (without .txt): ");
        String filename = "./test/" + in.nextLine() + ".txt";

        try {
            fw = new FileWriter(filename);
            buff = new BufferedWriter(fw);
            buff.write("Deck: " + Deck.stringDeck(deck));
            buff.newLine();
            buff.newLine();
            if (setStr.size() == 0) {
                buff.write("No solution found");
                buff.newLine();
            } else {
                buff.write(setStr.size() + " solution(s) found");
                buff.newLine();

                for (String ans : setStr) {
                    buff.write(ans);
                    buff.newLine();
                }
            }

            buff.newLine();
            buff.write("Execution time: " + exec + " ms");
            buff.flush();
            buff.close();
            fw.close();
        } catch (IOException e) {
            System.out.println(e.getMessage());
        }
    }
}

```



```
System.out.println("\nExecution time: " + exec + " ms");

cmd = OutputHandler.outputMethod();

if (cmd == 1) {
    OutputHandler.writeAnsFile(deck, setStr, exec);
}
}
```

BAB IV

UJI COBA

Pada bagian ini, uji coba yang ditampilkan hanya uji coba yang berkaitan langsung dengan algoritma utama (algoritma penyelesaian). Untuk melihat tampilan uji coba validasi masukan, tampilan menu, konfirmasi, dan lain-lain silakan lihat pada Lampiran.

4.1 Custom Deck

1. *Deck 10 5 10 5*

```
Choose input method:
1. Custom Deck
2. Random Deck
Your answer: 1
Insert deck: 10 5 10 5

Deck: 10 5 10 5

1 solution(s) found
(5 * 5) - (10 / 10)

Execution time: 1 ms
```

Jawaban disimpan pada *folder test file cd_1.txt*.

2. *Deck A A A A*

```
Choose input method:
1. Custom Deck
2. Random Deck
Your answer: 1
Insert deck: A A A A

Deck: A A A A

No solution found

Execution time: 2 ms
```

Jawaban disimpan pada *folder test file cd_2.txt*.

3. *Deck 2 7 J K*

```
Choose input method:
1. Custom Deck
2. Random Deck
Your answer: 1
Insert deck: 2 7 J K

Deck: 2 7 J K

No solution found

Execution time: 1 ms
```

Jawaban disimpan pada *folder test file cd_3.txt*.

4. *Deck Q 5 7 3*

```

Choose input method:
1. Custom Deck
2. Random Deck
Your answer: 1
Insert deck: Q 5 7 3

Deck: Q 5 7 3

28 solution(s) found
(Q * (7 + 3)) / 5
((3 + 7) / 5) * Q
(Q / 5) * (3 + 7)
(Q * 3) - (7 + 5)
(3 * (7 + 5)) - Q
(3 * Q) - (7 + 5)
(3 + 7) * (Q / 5)
(3 * (5 + 7)) - Q
((5 + 7) * 3) - Q
(7 + 3) * (Q / 5)
((3 + 7) * Q) / 5
((3 * Q) - 5) - 7
(3 * Q) - (5 + 7)
(Q / 5) * (7 + 3)
Q / (5 / (7 + 3))
Q * ((3 + 7) / 5)
((Q * 3) - 7) - 5
(7 + 3) / (5 / Q)
(3 + 7) / (5 / Q)
((7 + 5) * 3) - Q
Q * ((7 + 3) / 5)
((3 * Q) - 7) - 5
Q / (5 / (3 + 7))
((7 + 3) / 5) * Q
(Q * 3) - (5 + 7)
((7 + 3) * Q) / 5
(Q * (3 + 7)) / 5
((Q * 3) - 5) - 7

Execution time: 1 ms

```

Jawaban disimpan pada *folder test file* `cd_4.txt`.

5. *Deck* 4 5 6 7


```

Choose input method:
1. Custom Deck
2. Random Deck
Your answer: 1
Insert deck: 4 5 6 7

Deck: 4 5 6 7

20 solution(s) found
(6 - 4) * (5 + 7)
((7 - 6) + 5) * 4
(5 + 7) * (6 - 4)
(7 + 5) * (6 - 4)
4 * (5 + (7 - 6))
4 * (7 + (5 - 6))
(7 + (5 - 6)) * 4
((5 + 7) - 6) * 4
(7 - (6 - 5)) * 4
4 * ((7 + 5) - 6)
4 * (5 - (6 - 7))
4 * ((5 + 7) - 6)
((7 + 5) - 6) * 4
4 * ((7 - 6) + 5)
4 * ((5 - 6) + 7)
(6 - 4) * (7 + 5)
(5 - (6 - 7)) * 4
((5 - 6) + 7) * 4
4 * (7 - (6 - 5))
(5 + (7 - 6)) * 4

Execution time: 2 ms

```

Jawaban disimpan pada *folder test file* `cd_5.txt`.

6. Deck A J Q K

```

Choose input method:
1. Custom Deck
2. Random Deck
Your answer: 1
Insert deck: A J Q K

Deck: A J Q K

32 solution(s) found
(K - J) / (A / Q)
Q / (A / (K - J))
((A * K) - J) * Q
Q * ((K - J) / A)
(K - J) * (Q / A)
A * (Q * (K - J))
((K / A) - J) * Q
(K - (J / A)) * Q
(K - J) * (A * Q)
((K * A) - J) * Q
(K - (A * J)) * Q
Q * (K - (J / A))
(K - (J * A)) * Q
(Q * (K - J)) / A
Q * (K - (J * A))
(A * (K - J)) * Q
((K - J) * A) * Q
(Q / A) * (K - J)
(Q * (K - J)) * A
Q * ((K / A) - J)
Q * (A * (K - J))
(K - J) * (Q * A)
((K - J) * Q) * A
A * ((K - J) * Q)
Q * ((A * K) - J)
((K - J) * Q) / A
Q * ((K - J) * A)
(Q * A) * (K - J)
((K - J) / A) * Q
Q * ((K * A) - J)
Q * (K - (A * J))
(A * Q) * (K - J)

Execution time: 1 ms

```

Jawaban disimpan pada *folder test file* cd_6.txt.

7. Deck A 2 9 6

```
Choose input method:
1. Custom Deck
2. Random Deck
Your answer: 1
Insert deck: A 2 9 6
```

```
Deck: A 2 9 6
```

```
72 solution(s) found
```

```
(9 - A) / (2 / 6)
(9 * 2) + (A * 6)
6 + ((2 * A) * 9)
6 + (A * (2 * 9))
((2 * 9) * A) + 6
A * ((2 * 9) + 6)
(6 + (9 * 2)) * A
6 + ((9 * 2) / A)
6 + (9 / (A / 2))
6 + ((2 * 9) / A)
((2 * 9) + 6) / A
(6 * (9 - A)) / 2
A * ((9 * 2) + 6)
((2 * 9) / A) + 6
((9 * A) * 2) + 6
((2 * A) * 9) + 6
6 + ((A * 2) * 9)
((A * 2) * 9) + 6
(9 - A) * (6 / 2)
6 / (2 / (9 - A))
(9 * 2) + (6 / A)
6 + (2 * (9 * A))
((9 / A) * 2) + 6
(6 * A) + (2 * 9)
((9 - A) / 2) * 6
A * (6 + (9 * 2))
(6 * A) + (9 * 2)
((9 * 2) + 6) * A
(6 / A) + (2 * 9)
```

```
(2 * 9) + (6 * A)
(2 / (A / 9)) + 6
(9 / (A / 2)) + 6
((2 / A) * 9) + 6
(6 + (2 * 9)) * A
6 + (2 / (A / 9))
6 + ((9 / A) * 2)
(6 + (2 * 9)) / A
(A * (2 * 9)) + 6
(2 * 9) + (6 / A)
(A * 6) + (2 * 9)
(6 / 2) * (9 - A)
(6 / A) + (9 * 2)
6 + ((A * 9) * 2)
(9 * 2) + (6 * A)
((9 * 2) * A) + 6
(2 * (9 / A)) + 6
(A * (9 * 2)) + 6
(9 * (A * 2)) + 6
((9 * 2) + 6) / A
6 + ((2 / A) * 9)
(A * 6) + (9 * 2)
6 + ((9 * A) * 2)
6 + ((9 * 2) * A)
((9 * 2) / A) + 6
6 + (2 * (A * 9))
((A * 9) * 2) + 6
6 + (A * (9 * 2))
6 + (9 * (A * 2))
(2 * 9) + (A * 6)
6 + (9 * (2 / A))
(9 * (2 / A)) + 6
6 + ((2 * 9) * A)
6 + (9 * (2 * A))
6 * ((9 - A) / 2)
(2 * (A * 9)) + 6
((2 * 9) + 6) * A
(9 * (2 * A)) + 6
(2 * (9 * A)) + 6
A * (6 + (2 * 9))
6 + (2 * (9 / A))
(6 + (9 * 2)) / A
((9 - A) * 6) / 2
```

```
Execution time: 1 ms
```

Jawaban disimpan pada *folder* test file cd_7.txt.

8. Deck 6 6 6 6

```

Choose input method:
1. Custom Deck
2. Random Deck
Your answer: 1
Insert deck: 6 6 6 6

Deck: 6 6 6 6

7 solution(s) found
(6 * 6) - (6 + 6)
6 + ((6 + 6) + 6)
((6 + 6) + 6) + 6
((6 * 6) - 6) - 6
(6 + 6) + (6 + 6)
6 + (6 + (6 + 6))
(6 + (6 + 6)) + 6

Execution time: 2 ms

```

Jawaban disimpan pada *folder test file* cd_8.txt.

4.2 Random Deck

1. Uji Coba 1

```

Choose input method:
1. Custom Deck
2. Random Deck
Your answer: 2

Deck: 9 9 9 5

No solution found

Execution time: 1 ms

```

Jawaban disimpan pada *folder test file* rd_1.txt.

2. Uji Coba 2

```

Choose input method:
1. Custom Deck
2. Random Deck
Your answer: 2

Deck: 7 K 4 10

No solution found

Execution time: 1 ms

```

Jawaban disimpan pada *folder test file* rd_2.txt.

3. Uji Coba 3

```
Choose input method:
1. Custom Deck
2. Random Deck
Your answer: 2

Deck: 10 3 3 9

16 solution(s) found
(10 * 3) + (3 - 9)
(3 - 9) + (10 * 3)
(3 + (10 * 3)) - 9
((10 * 3) + 3) - 9
((3 * 10) - 9) + 3
(10 * 3) - (9 - 3)
(3 * 10) + (3 - 9)
3 - (9 - (3 * 10))
(3 + (3 * 10)) - 9
3 + ((10 * 3) - 9)
(3 * 10) - (9 - 3)
((10 * 3) - 9) + 3
((3 * 10) + 3) - 9
3 - (9 - (10 * 3))
(3 - 9) + (3 * 10)
3 + ((3 * 10) - 9)

Execution time: 0 ms
```

Jawaban disimpan pada *folder test file* rd_3.txt.

4. Uji Coba 4

```

Choose input method:
1. Custom Deck
2. Random Deck
Your answer: 2

Deck: 2 6 6 A

76 solution(s) found
2 * ((6 + 6) * A)
(6 * (2 + A)) + 6
(A * 2) * (6 + 6)
6 + (6 * (2 + A))
((A * 6) + 6) * 2
6 + (6 * (A + 2))
2 / (A / (6 + 6))
((6 * A) - 2) * 6
6 * ((6 / A) - 2)
(6 * A) * (6 - 2)
(6 * (6 - 2)) * A
(6 * (A + 2)) + 6
2 * (6 + (6 * A))
A * (6 * (6 - 2))
((6 / A) - 2) * 6
(6 + (6 / A)) * 2
(6 - (A * 2)) * 6
(6 - 2) * (6 * A)
2 * (A * (6 + 6))
(2 * (6 + 6)) / A
6 * ((6 - 2) * A)
((6 / A) + 6) * 2
A * ((6 - 2) * 6)
(6 + 6) * (2 * A)
6 * (6 - (2 / A))
((6 + 6) * 2) / A
(A * 6) * (6 - 2)
(6 - (2 / A)) * 6
(6 - 2) / (A / 6)
A * ((6 + 6) * 2)
((6 - 2) * 6) * A
A * (2 * (6 + 6))
6 / (A / (6 - 2))
6 * ((6 / 2) + A)
((6 / 2) + A) * 6
6 * ((6 - 2) / A)
(6 - 2) * (6 / A)
2 * ((A * 6) + 6)
(6 - (2 * A)) * 6
(6 + (A * 6)) * 2
((2 + A) * 6) + 6
((6 - 2) * A) * 6
((6 - 2) / A) * 6
((6 + 6) / A) * 2
(A * (6 - 2)) * 6
(6 + 6) * (A * 2)
((6 + 6) * A) * 2
((A + 2) * 6) + 6
(6 * (6 - 2)) / A
6 + ((2 + A) * 6)
(2 * A) * (6 + 6)
2 * ((6 / A) + 6)
6 * ((A * 6) - 2)
(6 + 6) / (A / 2)
((6 * A) + 6) * 2
(6 - 2) * (A * 6)
((A * 6) - 2) * 6
(2 * (6 + 6)) * A
(6 / A) * (6 - 2)
(A * (6 + 6)) * 2
(A + (6 / 2)) * 6
2 * ((6 + 6) / A)
2 * (6 + (6 / A))
2 * (6 + (A * 6))
6 * ((6 * A) - 2)
6 * (6 - (2 * A))
(6 + (6 * A)) * 2
2 * ((6 * A) + 6)
6 + ((A + 2) * 6)
6 * (A * (6 - 2))
((6 - 2) * 6) / A
6 * (A + (6 / 2))
6 * (6 - (A * 2))
(2 / A) * (6 + 6)
((6 + 6) * 2) * A

Execution time: 2 ms

```

Jawaban disimpan pada *folder test file rd_4.txt*.

5. Uji Coba 5

```
Choose input method:
1. Custom Deck
2. Random Deck
Your answer: 2

Deck: J 10 Q 3

16 solution(s) found
Q * (10 + (3 - J))
Q * ((10 - J) + 3)
Q * (3 - (J - 10))
((10 + 3) - J) * Q
((3 - J) + 10) * Q
((10 - J) + 3) * Q
Q * ((3 - J) + 10)
Q * ((10 + 3) - J)
Q * (10 - (J - 3))
Q * (3 + (10 - J))
(3 + (10 - J)) * Q
(10 + (3 - J)) * Q
(3 - (J - 10)) * Q
(10 - (J - 3)) * Q
Q * ((3 + 10) - J)
((3 + 10) - J) * Q

Execution time: 1 ms
```

Jawaban disimpan pada *folder test file rd_5.txt*.

6. Uji Coba 6

```

Choose input method:
1. Custom Deck
2. Random Deck
Your answer: 2

Deck: 6 Q 6 9

30 solution(s) found
(9 / (6 / Q)) + 6
6 + (9 / (6 / Q))
(6 / (9 - 6)) * Q
(Q / (9 - 6)) * 6
(Q * (9 / 6)) + 6
6 + ((Q / 6) * 9)
6 + ((9 / 6) * Q)
6 - (6 * (9 - Q))
6 + (9 * (Q / 6))
(Q * 6) / (9 - 6)
6 / ((9 - 6) / Q)
6 - ((9 - Q) * 6)
(6 * (Q - 9)) + 6
(9 * (Q / 6)) + 6
Q / ((9 - 6) / 6)
6 + (6 * (Q - 9))
6 + (Q / (6 / 9))
6 + ((Q * 9) / 6)
((Q / 6) * 9) + 6
Q * (6 / (9 - 6))
((9 * Q) / 6) + 6
((Q * 9) / 6) + 6
6 + ((9 * Q) / 6)
6 * (Q / (9 - 6))
6 + (Q * (9 / 6))
(6 * Q) / (9 - 6)
(Q / (6 / 9)) + 6
((9 / 6) * Q) + 6
((Q - 9) * 6) + 6
6 + ((Q - 9) * 6)

Execution time: 2 ms

```

Jawaban disimpan pada *folder test file rd_6.txt*.

7. Uji Coba 7


```

Choose input method:
1. Custom Deck
2. Random Deck
Your answer: 2

Deck: Q 10 8 Q

20 solution(s) found
Q * ((Q + 8) / 10)
(10 * Q) - (Q * 8)
Q / (10 / (Q + 8))
Q * ((8 + Q) / 10)
((Q + 8) * Q) / 10
(Q / 10) * (Q + 8)
(Q + 8) * (Q / 10)
(Q * (8 + Q)) / 10
(Q * 10) - (Q * 8)
Q / (10 / (8 + Q))
((8 + Q) * Q) / 10
(Q + 8) / (10 / Q)
(Q * (Q + 8)) / 10
(Q / 10) * (8 + Q)
((Q + 8) / 10) * Q
(Q * 10) - (8 * Q)
(10 * Q) - (8 * Q)
(8 + Q) / (10 / Q)
((8 + Q) / 10) * Q
(8 + Q) * (Q / 10)

Execution time: 1 ms

```

Jawaban disimpan pada *folder test file rd_7.txt*.

8. Uji Coba 8

```

Choose input method:
1. Custom Deck
2. Random Deck
Your answer: 2

Deck: 5 9 5 J

4 solution(s) found
(5 - 9) * (5 - J)
(5 - J) * (5 - 9)
(9 - 5) * (J - 5)
(J - 5) * (9 - 5)

Execution time: 0 ms

```

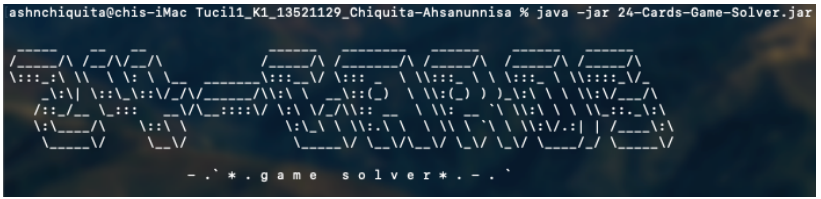
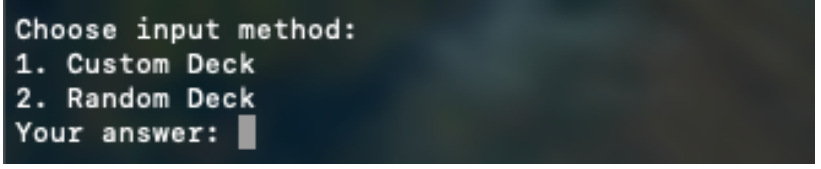
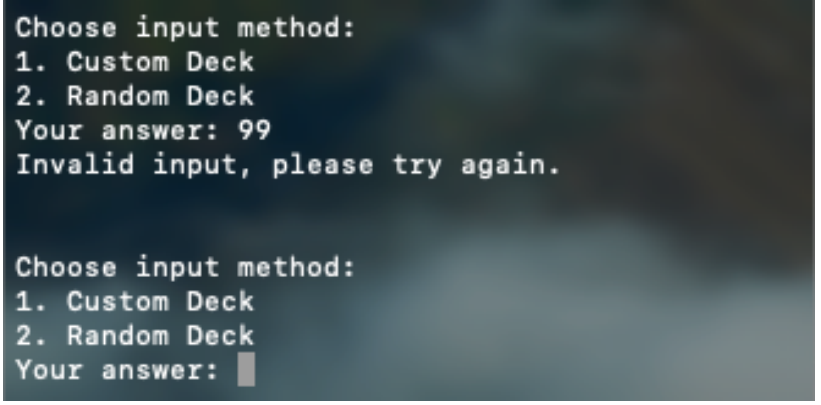
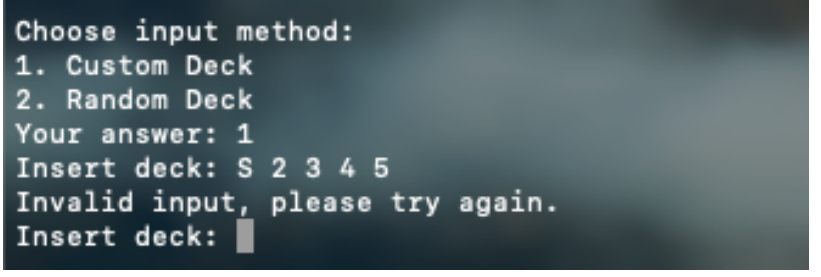
Jawaban disimpan pada *folder test file rd_8.txt*.

LAMPIRAN

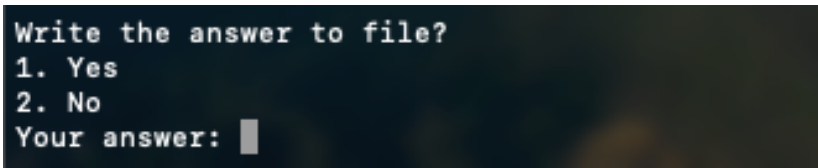
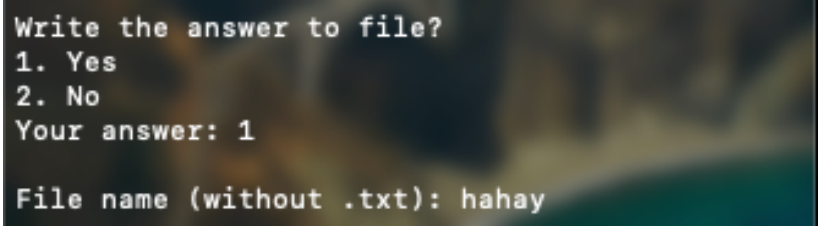
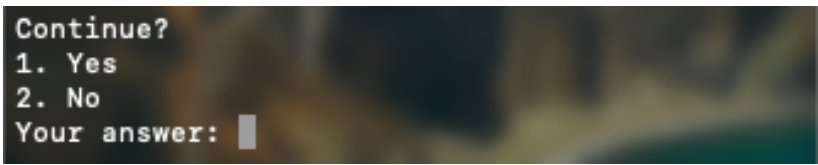
Lampiran 1 Link Repository GitHub

https://github.com/ashnchiquita/Tucil1_13521129

Lampiran 2 Tampilan Program

Nama	Tampilan
Memulai program	 <pre>ashnchiquita@chis-iMac Tucil1_K1_13521129_Chiquita-Ahsanunnisa % java -jar 24-Cards-Game-Solver.jar</pre>
Menu pemilihan metode masukan	 <pre>Choose input method: 1. Custom Deck 2. Random Deck Your answer: </pre>
Masukan pemilihan tidak valid	 <pre>Choose input method: 1. Custom Deck 2. Random Deck Your answer: 99 Invalid input, please try again. Choose input method: 1. Custom Deck 2. Random Deck Your answer: </pre>
Memasukkan <i>deck custom</i> tidak valid	 <pre>Choose input method: 1. Custom Deck 2. Random Deck Your answer: 1 Insert deck: S 2 3 4 5 Invalid input, please try again. Insert deck: </pre>

<p>Memasukkan <i>deck custom</i> valid</p>	<pre> Choose input method: 1. Custom Deck 2. Random Deck Your answer: 1 Insert deck: 6 6 6 6 Deck: 6 6 6 6 7 solution(s) found (6 * 6) - (6 + 6) 6 + ((6 + 6) + 6) ((6 + 6) + 6) + 6 ((6 * 6) - 6) - 6 (6 + 6) + (6 + 6) 6 + (6 + (6 + 6)) (6 + (6 + 6)) + 6 Execution time: 3 ms </pre>
<p>Memilih <i>random deck</i></p>	<pre> Choose input method: 1. Custom Deck 2. Random Deck Your answer: 2 Deck: 9 Q 4 9 10 solution(s) found 9 * (4 - (Q / 9)) ((9 - Q) + 9) * 4 (4 - (Q / 9)) * 9 4 * (9 - (Q - 9)) (9 - (Q - 9)) * 4 4 * (9 + (9 - Q)) (9 + (9 - Q)) * 4 4 * ((9 - Q) + 9) 4 * ((9 + 9) - Q) ((9 + 9) - Q) * 4 Execution time: 1 ms </pre>

Konfirmasi menulis jawaban ke <i>file</i>	 
Konfirmasi melanjutkan program	

Lampiran 3 Tabel *Check List* Poin

No.	Poin	Ya	Tidak
1.	Program dapat dikompilasi tanpa kesalahan	√	
2.	Program berhasil <i>running</i>	√	
3.	Program dapat membaca <i>input/generate</i> sendiri dan memberikan luaran	√	
4.	Solusi yang diberikan program memenuhi (berhasil mencapai 24)	√	
5.	Program dapat menyimpan solusi dalam teks*	√	

**Run* dari *file* .jar untuk mencegah *error* saat menuliskan solusi ke *file* .txt

DAFTAR REFERENSI

Levitin, Anany. (2012). *Introduction to the design and analysis of algorithms* (Third Edition). New Jersey: Pearson.

Munir, R. (2022). *Strategi Algoritma: Algoritma Brute Force (Bagian 1)*. Dilansir dari Homepage Rinaldi Munir: [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-Brute-Force-\(2022\)-Bag1.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-Brute-Force-(2022)-Bag1.pdf). Diakses pada 20 Januari 2023.