Tugas Kecil 1 IF2211 Strategi Algoritma

Semester II Tahun 2025/2026

Penyelesaian Permainan ***Queens*** LinkedIn

---


Disusun oleh :

Nathan Edward Christofer Marpaung

13524062

Program Studi Teknik Informatika

Sekolah Teknik dan Informatika

Institut Teknologi Bandung

2024

## I. Permainan *Queens* dan Deskripsi Permasalahan

Permainan *Queens* merupakan gim yang tersedia di situs LinkedIn. Gim ini berisi sebuah papan persegi yang berwarna, dimana kita harus meletakkan ratu di masing-masing daerah warna dengan syarat:

1. Tidak ada ratu yang berada di daerah warna yang sama
2. Tidak ada ratu yang berada di baris yang sama
3. Tidak ada ratu yang berada di kolom yang sama
4. Tidak ada ratu yang berdiri bersebelahan (termasuk diagonal

Dalam tugas kecil ini, program didesain untuk mencari seluruh solusi yang memungkinkan dari sebuah papan dan daerah warna yang sudah diinput. Beberapa batasan yang sudah ditetapkan di dalam program ini adalah:

1. Terdapat sebanyak N region warna pada sebuah papan persegi NxN
2. Warna di input dalam bentuk alfabet
3. Jumlah warna maksimal sebanyak banyak maksimum alfabet (26 warna)
4. Urutan dan daerah warna di papan akan diinput melalui file .txt, dan akan di output pada .txt jika diinginkan.

## II. Algoritma Brute Force

Algoritma Brute Force adalah strategi algoritma untuk mencari solusi dengan mencari dan mengetes seluruh kombinasi solusi yang memungkinkan. Pada konteks permainan ini, program akan mengecek seluruh kemungkinan kombinasi ratu yang ada pada masing-masing daerah warna dan akan menyimpan solusi yang ditemukan jika memenuhi kondisi dari permainan.

Algoritma Brute Force yang digunakan di dalam program ini adalah Brute Force dengan Backtracking tanpa pruning. Program akan mencari kombinasi awal terlebih dahulu, kemudian akan backtrack dan menguji kombinasi lainnya. Implementasi algoritma akan dilakukan dalam bahasa Python.

## III. Pembahasan kode

Program akan menerima inputan file berupa file .txt (contoh : test1.txt). Program akan membuat sebuah dictionary terlebih dahulu dengan membaca seluruh karakter alfabet yang muncul di dalam input file .txt, dan memetakan menjadi N index. Kemudian, program akan membaca kembali seluruh baris dan karakter dari file input .txt, dan memetakan koordinat masing-masing warna sesuai dengan letak karakter alfabet (warna) dan indeks dictionary yang sudah dibuat dalam sebuah list. Pemetaan ini dilakukan untuk membagi dan memetakan koordinat pada masing-masing warna, sehingga mendapatkan sebuah list dengan panjang N warna dimana masing-masing indeks list berisi list koordinat region yang ditempati oleh warna/alfabet tersebut.

```
#Dictionary koordinat untuk warna
ColorCoordinate = {}
def MakeColorDict(c):
    n_keys = len(ColorCoordinate)
    ColorCoordinate.update({c.lower(): n_keys})

#Baca posisi warna
def ReadWarna(line, KoorWarna, row):
    j = 0
    for c in line:
        if c.lower() not in ColorCoordinate:
            MakeColorDict(c)
        index = ColorCoordinate[c.lower()]
        KoorWarna[index].append((row, j))
        j += 1

#Read test.txt and paste masing-masing coord warna
def OpenCaseFile(filename):
    with open(filename, "r") as f:
        first_line = f.readline().strip()
        KoorWarna = [[] for _ in range(len(first_line))]
        ReadWarna(first_line, KoorWarna, 0)
        row = 1
```

```
        for line in f:
            line = line.strip()
            ReadWarna(line, KoorWarna, row)
            row += 1
    f.close()
    return KoorWarna
```

Hasil pemetaan koordinat warna akan digunakan untuk melakukan algoritma brute force secara backtracking tanpa pruning.

Setiap koordinat akan di test secara rekursif dimulai dari indeks 0 untuk warna pertama hingga akhir, dan dilanjutkan secara progresif dari warna terakhir hingga indeks terakhir untuk warna pertama hingga akhir.

Algoritma meninjau setiap indeks pertama dari warna pertama, dan menyimpan nilai baris dan kolom dari posisi ratu di warna tersebut. Kemudian, algoritma akan rekursif ke warna selanjutnya, dan menyimpan kembali nilai baris, kolom, dan koordinat dari ratu di warna tersebut. Hal ini dilakukan berulang kali, hingga sampai warna terakhir. Jika algoritma mendeteksi sudah berada pada region warna terakhir, maka program akan looping dan menguji pada seluruh kombinasi koordinat akhir yang memungkinkan. Untuk pengetesan baris dan kolom, algoritma akan melihat apakah ada nilai yang sama di dalam nilai baris dan kolom yang disimpan. Untuk pengetesan posisi ratu yang bersebelahan, algoritma cukup menguji apakah ada 2 ratu yang bersebelahan secara diagonal, dengan cara menyimpan koordinat diagonal dari seluruh posisi ratu di setiap kombinasi, dan melihat apakah ada posisi ratu di dalam kombinasi yang berada di dalam list koordinat diagonal dari seluruh ratu.

Jika hasil kombinasi akhir tidak memberikan sebuah solusi, maka program akan lanjut pada kombinasi berikutnya baik itu mengecek posisi ratu selanjutnya, ataupun backtracking pada warna sebelumnya. Jika hasil kombinasi akhir memberikan sebuah solusi yang memenuhi peraturan gim tersebut, maka solusi akan disimpan pada sebuah list Final_Result. Program ini

kemudian akan looping hingga seluruh kombinasi ratu berdasarkan warna yang berbeda, dan akan mendisplay seluruh solusi yang sudah didapat dan disimpan pada Final_Result.

Untuk setiap kombinasi akhir ratu yang didapatkan dan diuji, sebuah variabel global Iteration akan dihitung untuk melihat jumlah kombinasi yang sudah diuji. Setiap 500 iterasi yang diuji oleh algoritma akan diprint untuk membuat sebuah live update dari algoritma yang dilakukan.

Berikut adalah pseudocode dari algoritma bruteforce tersebut:

```
function check (RowWarna, ColWarna : array of integer, ListofKoordWarna,
Result : array of points) -> array of array of points
```

```
KAMUS LOKAL
Warna_terakhir : boolean
Global Iteration : integer
Check_result : array of array of point
```

```
Final_Result <- []
if (panjang(ListofKoordWarna) < 2) then {cek apakah sisa satu warna lagi}
    Warna_terakhir <- true
Else
    Warna_terakhir <- false

FOR koordinat di dalam KoordWarna DO
    x <- KoordWarna[0] {Ambil absis}
    y <- KoordWarna[1] {Ambil ordinat}

    tambahkan x ke RowWarna
    tambahkan y ke RowWarna

    If (last_color) then
        Iteration += 1 {tambahkan iterasi setelah dapat kombinasi final}
        tambahkan (x,y) pada Result
        If (iteration mod 500 = 0) then {print kombinasi setiap 500 iterasi}
            print(Result)
        If (check_kolom_dan_row(RowWarna,ColWarna)) then {cek apakah ada ratu
```

```
dengan baris atau kolom yang sama}
            If (check_diagonal(Result)) then {cek dari kombinasi final apakah
ada ratu yang bersebelahan secara diagonal}
                Tambahkan result ke Final_Result {tambahkan kombinasi sebagai
solusi akhir jika memenuhi kondisi row/baris yang berbeda dan tidak ada yang
diagonal}
    Else {bukan warna terakhir}
        Tambahkan (x,y) ke Result
        {Rekursif ke warna selanjutnya, dan cek apakah ada kombinasi yang
berhasil dibuat}
        Check_result <- check(RowWarna, ColWarna, KoordWarna[1…], Result)
        {Hapus nilai kolom dan baris (x dan y) dari RowWarna dan ColWarna
sebagai bentuk backtrack, dan lanjut ke koordinat yang baru di warna yang
sama)
        Hapus x dari RowWarna
        Hapus y dari RowWarna
        Hapus (x,y) dari Result

-> Final_Result {return hasil solusi yang ditemukan. Hasil solusi dapat berupa
[] jika tidak ada solusi}
```

Hasil seluruh solusi di dalam Final_Result kemudian akan ditinjau kembali, dan digunakan untuk menampilkan setiap solusi akhir dari permainan *Queens* yang diinput.

Untuk mengetes dan membantu optimisasi dari program bruteforce ini, kita dapat menambahkan early pruning di dalam backtracking, yaitu menghapus seluruh proses rekursif selanjutnya jika koordinat yang dipilih sudah melanggar aturan permainan yaitu memiliki baris atau kolom yang sama. Berikut pseudocode dari optimisasi bruteforce yang dibuat :

```
function check (RowWarna, ColWarna : array of integer, ListofKoordWarna,
Result : array of points) -> array of array of points
```

```
KAMUS LOKAL
```

```
Warna_terakhir : boolean
Global Iteration : integer
Check_result : array of array of point
```

```
Final_Result <- []
if (panjang(ListofKoordWarna) < 2) then {cek apakah sisa satu warna lagi}
    Warna_terakhir <- true
Else
    Warna_terakhir <- false

FOR koordinat di dalam KoordWarna DO
    x <- KoordWarna[0] {Ambil absis}
    y <- KoordWarna[1] {Ambil ordinat}

    tambahkan x ke RowWarna
    tambahkan y ke RowWarna

    If (last_color) then
        Iteration += 1 {tambahkan iterasi setelah dapat kombinasi final}
        tambahkan (x,y) pada Result
        If (iteration mod 500 = 0) then {print kombinasi setiap 500 iterasi}
            print(Result)
        If (check_kolom_dan_row(RowWarna,ColWarna)) then {cek apakah ada ratu
dengan baris atau kolom yang sama}
            If (check_diagonal(Result)) then {cek dari kombinasi final apakah
ada ratu yang bersebelahan secara diagonal}
                Tambahkan result ke Final_Result {tambahkan kombinasi sebagai
solusi akhir jika memenuhi kondisi row/baris yang berbeda dan tidak ada yang
diagonal}
    Else {bukan warna terakhir}
        If (x in RowWarna) or (y in ColWarna) then
            Continue {skip koordinat dan pengecekan lebih dalam, dan langsung
ke koordinat selanjutnya{
        Tambahkan (x,y) ke Result
        {Rekursif ke warna selanjutnya, dan cek apakah ada kombinasi yang
berhasil dibuat}
        Check_result <- check(RowWarna, ColWarna, KoordWarna[1…], Result)
```

```
        {Hapus nilai kolom dan baris (x dan y) dari RowWarna dan ColWarna
sebagai bentuk backtrack, dan lanjut ke koordinat yang baru di warna yang
sama)
        Hapus x dari RowWarna
        Hapus y dari RowWarna
        Hapus (x,y) dari Result

-> Final_Result {return hasil solusi yang ditemukan. Hasil solusi dapat berupa
[] jika tidak ada solusi}
```

Perhatikan bahwa pembuatan optimisasi ini membuat algoritma tidak lagi menjadi algoritma pure bruteforce. Namun, optimisasi ini dibuat untuk mengetes dan membandingkan tingkat performa dari pure bruteforce dengan optimisasi berupa early pruning menjadi backtracking algorithm.

**IV. Source Code**

Berikut adalah source code dari program secara keseluruhan :

```
import time

#Dictionary koordinat untuk warna
ColorCoordinate = {}
def MakeColorDict(c):
    n_keys = len(ColorCoordinate)
    ColorCoordinate.update({c.lower(): n_keys})

#Baca posisi warna
def ReadWarna(line, KoorWarna, row):
    j = 0
    for c in line:
        if c.lower() not in ColorCoordinate:
            MakeColorDict(c)
        index = ColorCoordinate[c.lower()]
```

```
        KoorWarna[index].append((row, j))
        j += 1

#Read test.txt and paste masing-masing coord warna
def OpenCaseFile(filename):
    with open(filename, "r") as f:
        first_line = f.readline().strip()
        KoorWarna = [[] for _ in range(len(first_line))]
        ReadWarna(first_line, KoorWarna, 0)
        row = 1
        for line in f:
            line = line.strip()
            ReadWarna(line, KoorWarna, row)
            row += 1
    f.close()
    return KoorWarna

def check_Row_and_Col(RowWarna, ColWarna):
    Rows = []
    Cols = []
    for Row in RowWarna:
        if Row in Rows:
            return False
        else:
            Rows.append(Row)
    for Col in ColWarna:
        if Col in Cols:
            return False
        else:
            Cols.append(Col)
    return True

def check_Diagonal(Result):
    DiagKoords = []
```

```
    for Koord in Result:
        x = Koord[0]
        y = Koord[1]
        DiagKoords.append((x+1, y+1))
        DiagKoords.append((x+1, y-1))
        DiagKoords.append((x-1, y+1))
        DiagKoords.append((x-1, y-1))
    for Koord in DiagKoords:
        if Koord in Result:
            return False
    return True

def check(RowWarna, ColWarna, KoordWarna, Result):
    global Iteration
    Final_Result = []
    #Check sisa 2 warna atau nggak
    if (len(KoordWarna) < 2):
        last_color = True
    else:
        last_color = False
    for Koord in KoordWarna[0]:
        x = Koord[0]
        y = Koord[1]
        RowWarna.append(x)
        ColWarna.append(y)
        if (last_color):
            Iteration += 1
            Result.append((x, y))
            if (Iteration % 500) == 0:
                print(Result)
            if (check_Row_and_Col(RowWarna, ColWarna)):
                if (check_Diagonal(Result)):
                    Final_Result.append(Result.copy())
        else:
```

```
            Result.append((x, y))
            check_result = check(RowWarna, ColWarna,
KoordWarna[1:], Result)
            if check_result != []:
                Final_Result.extend(check_result)
        RowWarna.pop()
        ColWarna.pop()
        Result.pop()

    return Final_Result

def check_Optimized(RowWarna, ColWarna, KoordWarna, Result):
    global Iteration
    Final_Result = []
    #Check sisa 2 warna atau nggak
    if (len(KoordWarna) < 2):
        last_color = True
    else:
        last_color = False
    for Koord in KoordWarna[0]:
        x = Koord[0]
        y = Koord[1]
        if (last_color):
            Iteration += 1
            Result.append((x, y))
            RowWarna.append(x)
            ColWarna.append(y)
            if (Iteration % 500) == 0:
                print(Result)
            if (check_Row_and_Col(RowWarna, ColWarna)):
                if (check_Diagonal(Result)):
                    Final_Result.append(Result.copy())
            Result.pop()
            RowWarna.pop()
```

```
            ColWarna.pop()
        else:
            if (x in RowWarna) or (y in ColWarna):
                continue
            else:
                RowWarna.append(x)
                ColWarna.append(y)
                Result.append((x, y))
                check_result = check_Optimized(RowWarna, ColWarna,
KoordWarna[1:], Result)
                if check_result != []:
                    Final_Result.extend(check_result)
                RowWarna.pop()
                ColWarna.pop()
                Result.pop()
    return Final_Result

def getSolvedLines(Col_Queen, filename):
    with open(filename, "r") as f:
        row = 0
        baris_solve = []
        for line in f:
            string_baris = ""
            line = line.strip()
            col = 0
            for c in line:
                if col == Col_Queen[row]:
                    string_baris += "#"
                else:
                    string_baris += c
                col += 1
            baris_solve.append(string_baris)
            row += 1
    f.close()
```

```
    return baris_solve


#Print tiap baris solusi. Koordinat Queen diganti ke array sebesar
n, dengan nilai x = indeks col_queen
def printResult(Final_Result, filename):
    num = 1
    all_solution = []
    for Result in Final_Result:
        Col_Queen = [0] * len(Result)
        for Koord in Result:
            x = Koord[0]
            y = Koord[1]
            Col_Queen[x] = y
        print(f"Solusi ke - {num} : \n")
        full_solve = getSolvedLines(Col_Queen, filename)
        for baris in full_solve:
            print(baris)
        all_solution.append(full_solve)
        num += 1
    save = input("Apakah ingin menyimpan solusi? (y/n) : \n")
    if save == "y":
        with open(f"./test/solusi_{nama_file}.txt", "w") as f:
            num = 1
            for solution in all_solution:
                f.write(f"Solusi ke - {num} : \n")
                for baris in solution:
                    f.write(baris + "\n")
                num += 1
        f.close()



def BruteForce_Warna(Koorwarna):
```

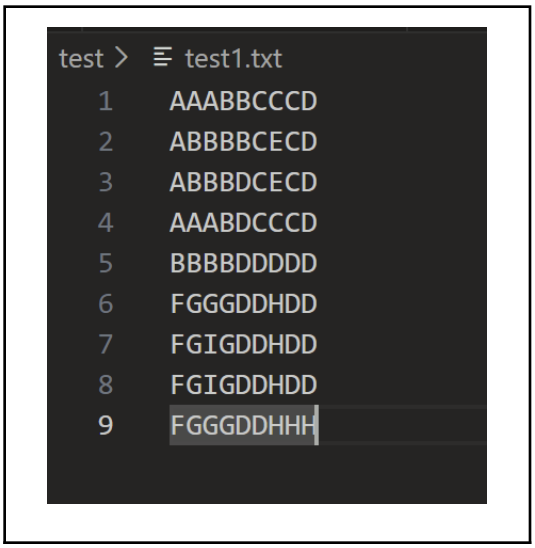
```
    optimized = input(f"Optimized? (y/n): \n")
    RowWarna = []
    ColWarna = []
    Result = []
    if optimized == "y":
        start = time.perf_counter()
        Result = check_Optimized(RowWarna, ColWarna, Koorwarna,
Result)
        end = time.perf_counter()
    else:
        start = time.perf_counter()
        Result = check(RowWarna, ColWarna, Koorwarna, Result)
        end = time.perf_counter()
    runtime = end - start
    return Result, runtime



Iteration = 0
nama_file = input(f"Masukkan nama file test case : \n")
filename = f"./test/{nama_file}.txt"
# print(OpenCaseFile(filename))
KoorWarna = OpenCaseFile(filename)
if KoorWarna == []:
    print(f"File tidak valid")
Result, Runtime = BruteForce_Warna(KoorWarna)
if Result == []:
    print(f"Tidak ada solusi yang ditemukan")
else:
    print(f"Solusi ditemukan!\n")
    print(f"Jumlah solusi yang ditemukan : {len(Result)}\n")
printResult(Result, filename)
print(f"Jumlah iterasi/kasus : {Iteration}")
print(f"Runtime : {Runtime:.4f} s")
```

**V. Testing**

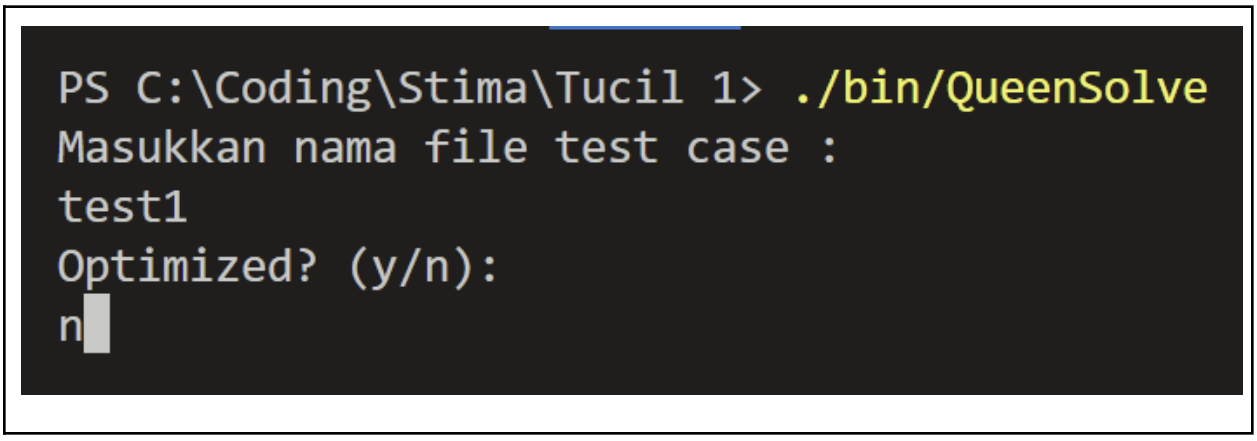
Inputan test case masing-masing terletak pada folder ./test yang sudah disediakan di dalam repository.

a. Test Case 1

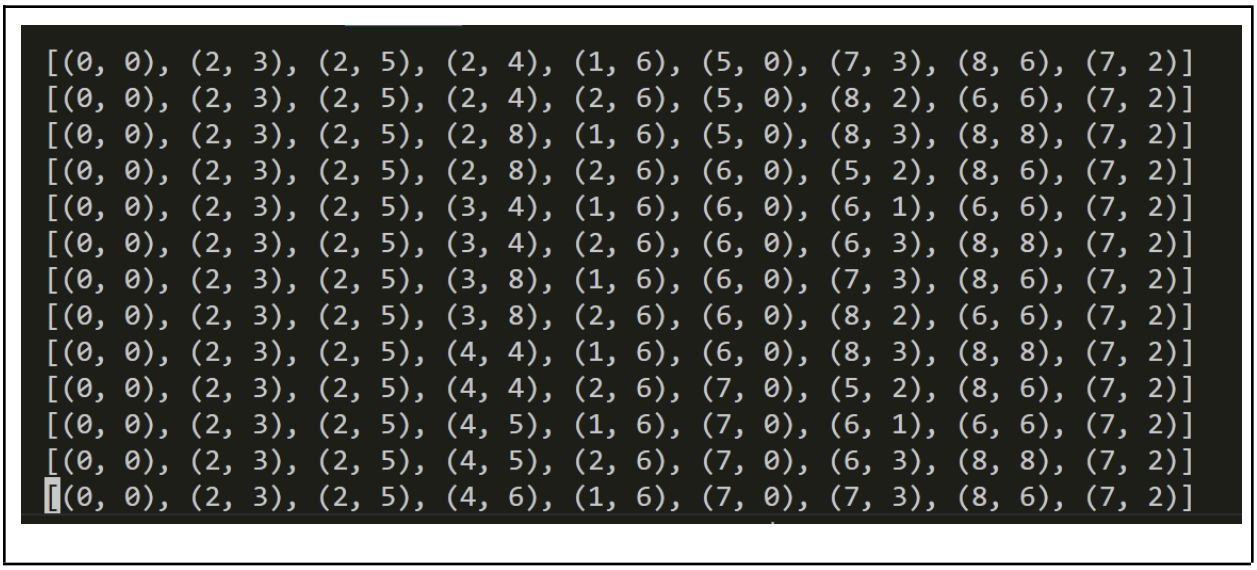
File test1.txt :



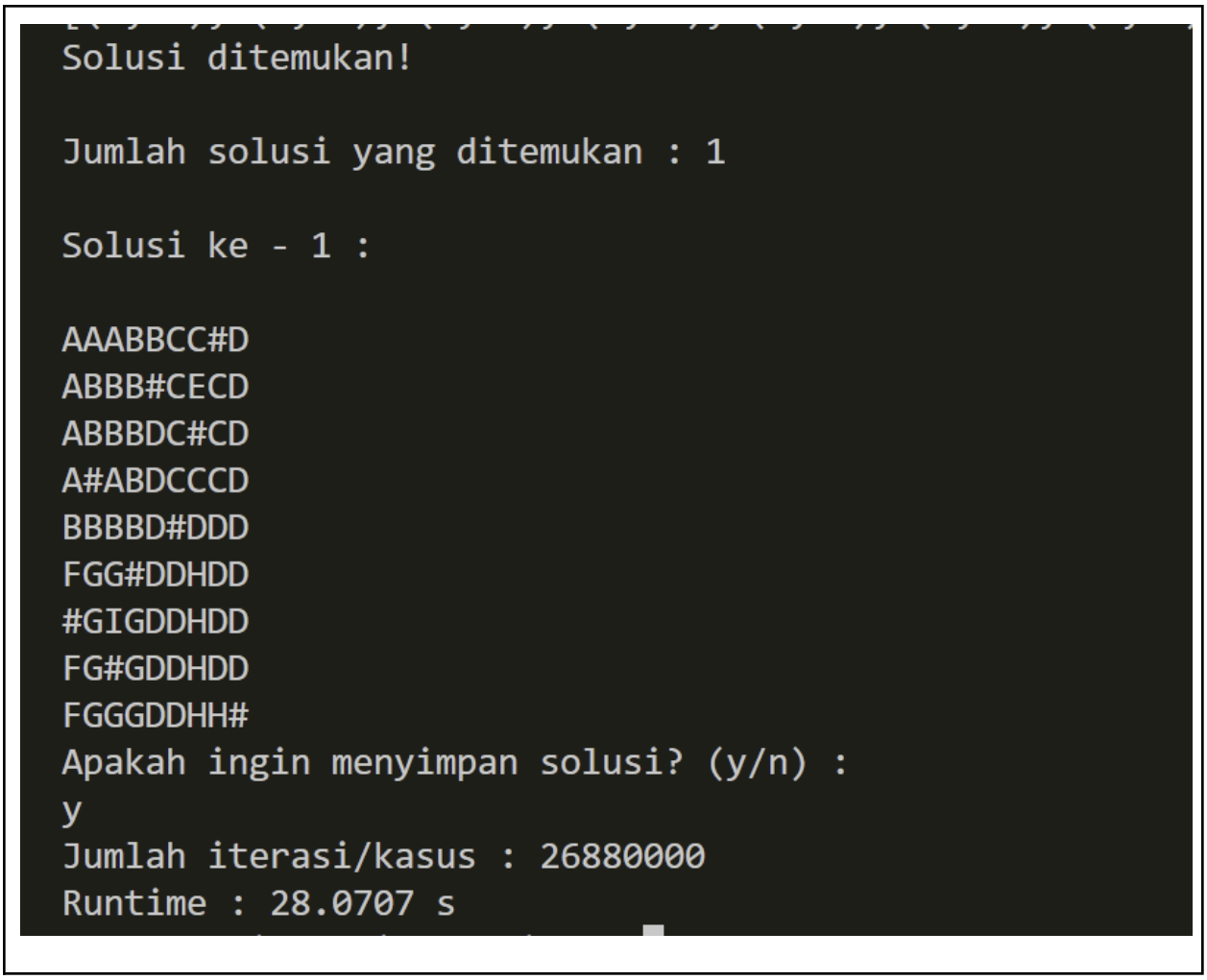

Menginput nama file dan pencarian solusi secara bruteforce tanpa optimisasi :



Live update :




Solusi, jumlah iterasi, waktu runtime, dan penyimpanan solusi :




Solusi, jumlah iterasi, dan waktu runtime dengan optimisasi :

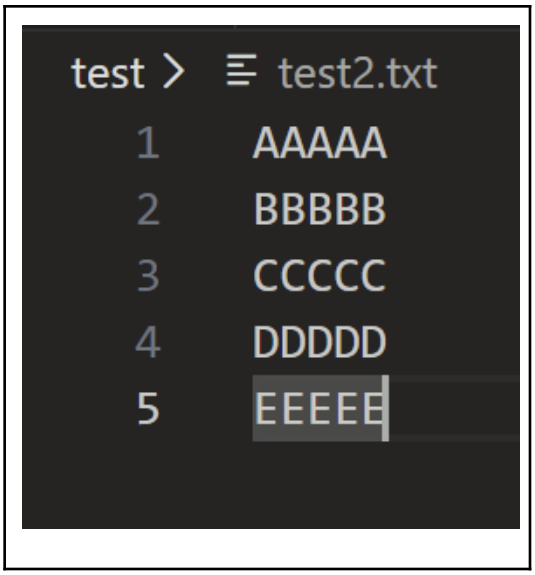
```
Solusi ditemukan!

Jumlah solusi yang ditemukan : 1

Solusi ke - 1 :

AAABBCC#D
ABBB#CECD
ABBBDC#CD
A#ABDCCCD
BBBBD#DDD
FGG#DDHDD
#GIGDDHDD
FG#GDDHDD
FGGGDDHH#
Apakah ingin menyimpan solusi? (y/n) :
n
Jumlah iterasi/kasus : 6368
Runtime : 0.0230 s
```

Hasil penyimpanan solusi di solusi_test1.txt :

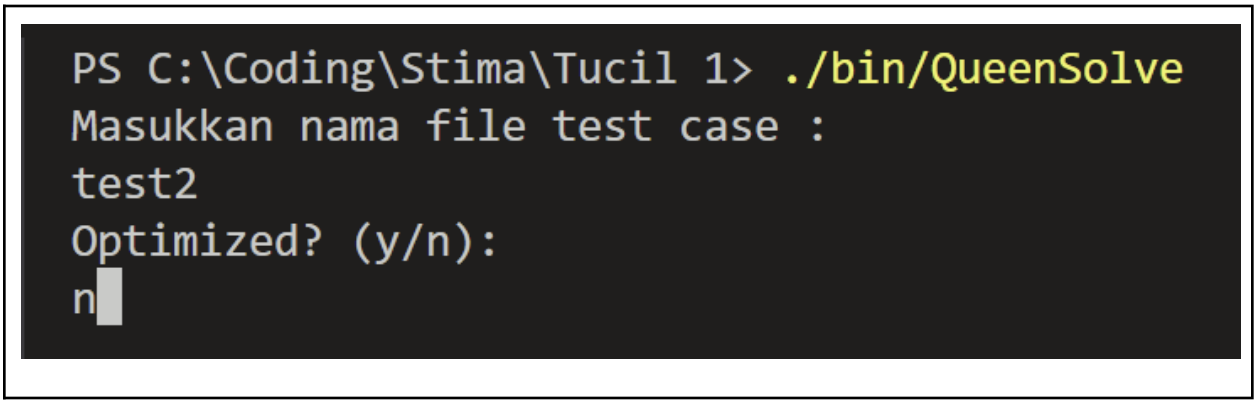
```
test > solusi_test1.txt
1   Solusi ke - 1 :
2   AAABBCC#D
3   ABBB#CECD
4   ABBBDC#CD
5   A#ABDCCCD
6   BBBBD#DDD
7   FGG#DDHDD
8   #GIGDDHDD
9   FG#GDDHDD
10  FGGGDDHH#
11
```

b. Test Case 2

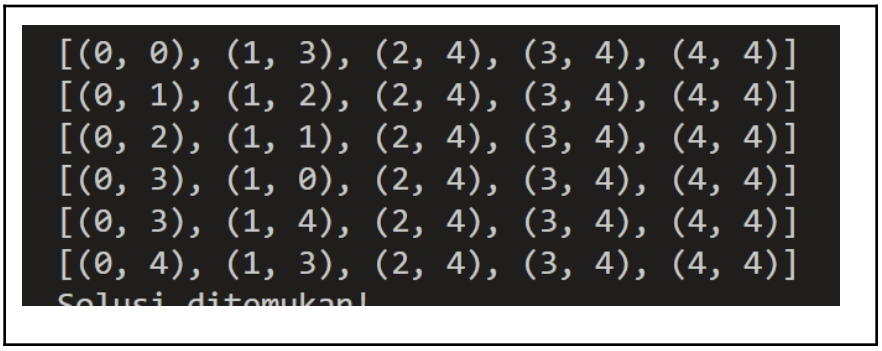
File test2.txt :



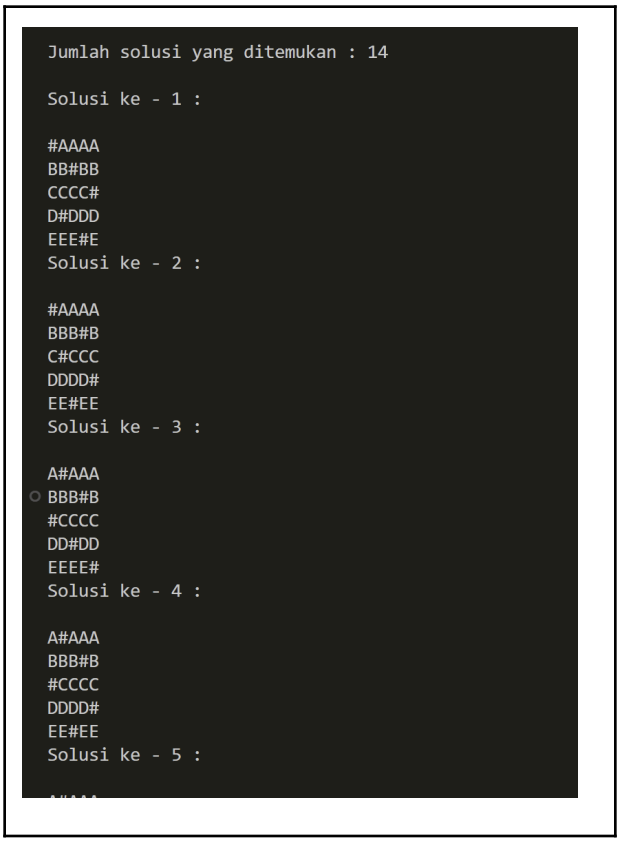

Menginput nama file dan pencarian solusi secara bruteforce tanpa optimisasi :



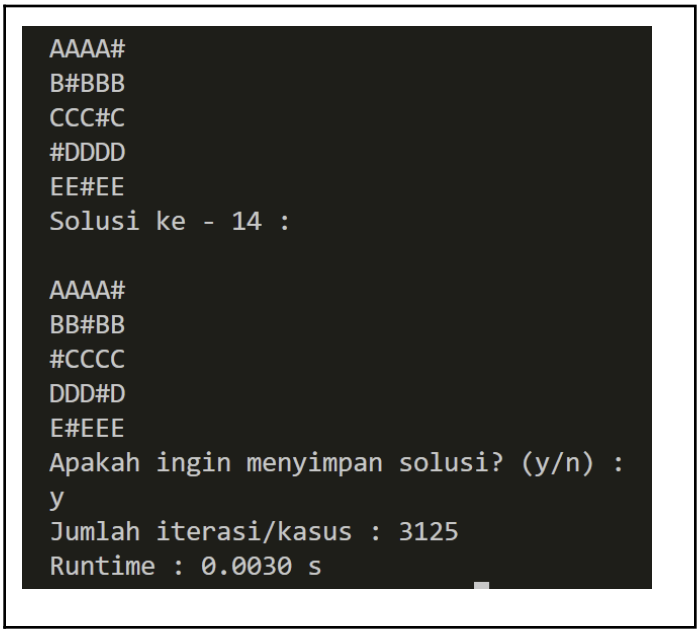

Live update :



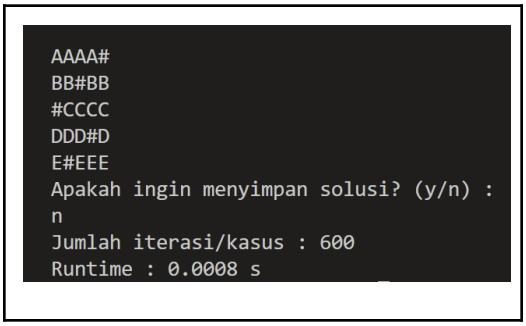

Beberapa solusi untuk test case:



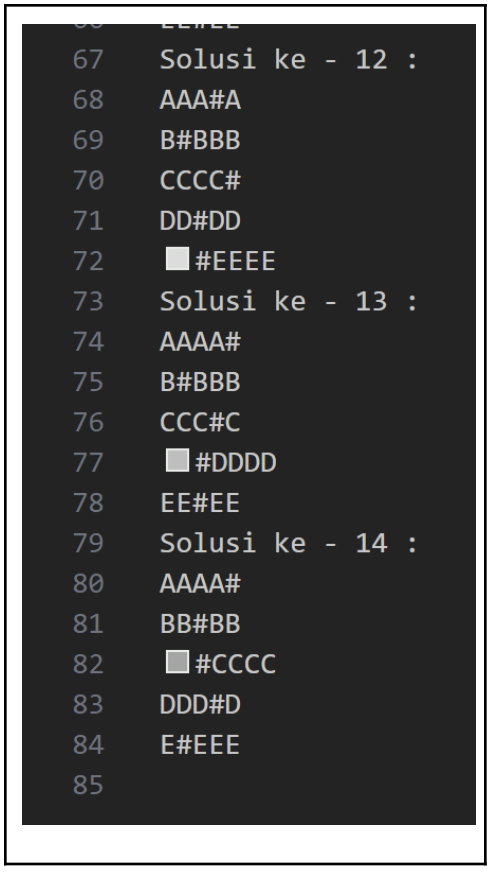

Penyimpanan, jumlah iterasi, dan runtime dari algoritma pure bruteforce :



Solusi, jumlah iterasi, dan waktu runtime dengan optimisasi :

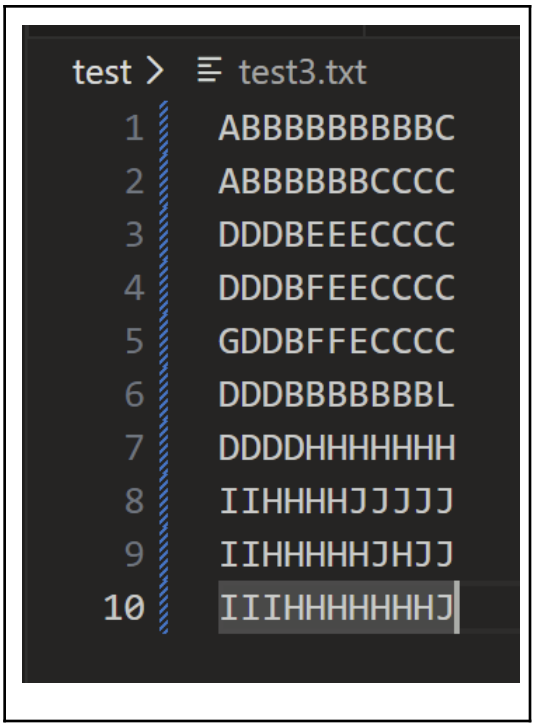



Hasil penyimpanan solusi di solusi_test2.txt :

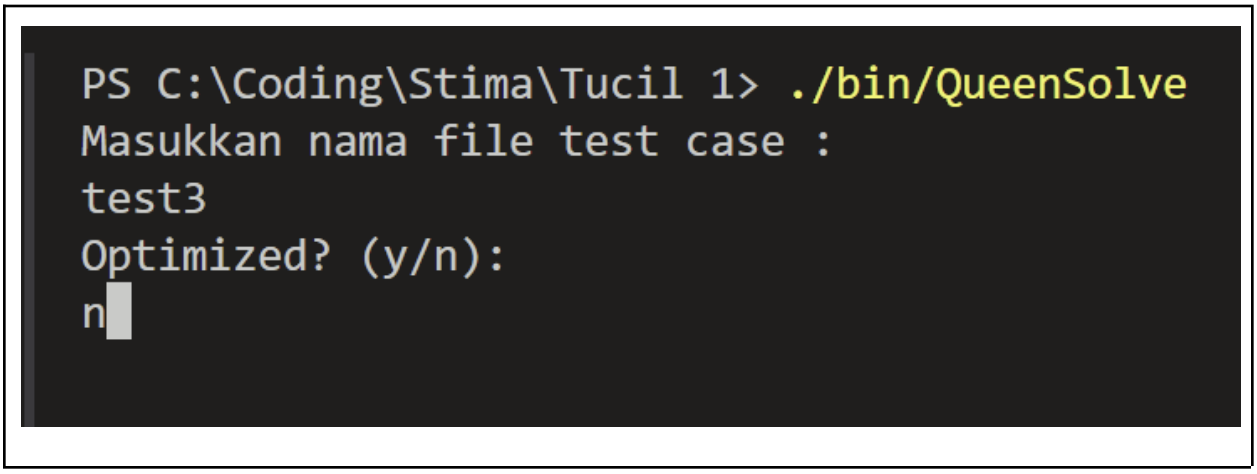



c. Test Case 3

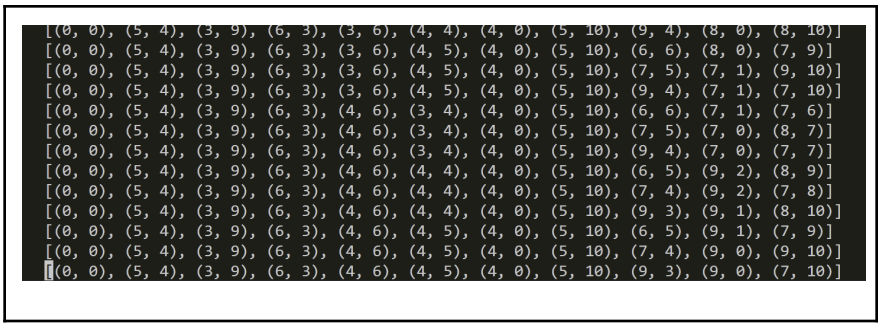
File test3.txt :



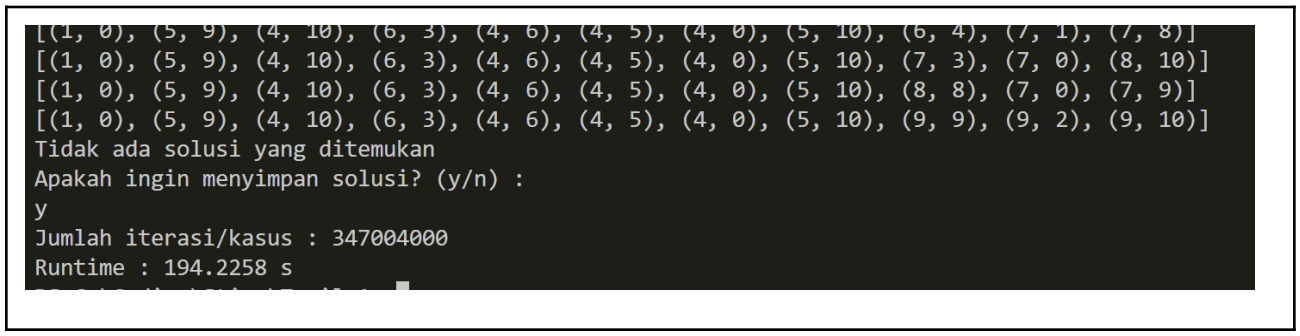

Menginput nama file dan pencarian solusi secara bruteforce tanpa optimisasi :



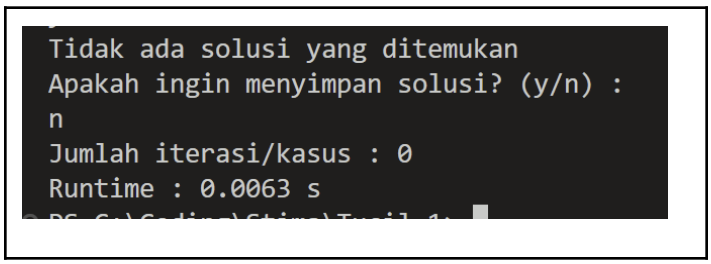

Live update :



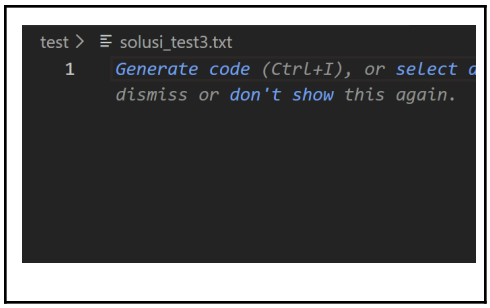

Solusi, penyimpanan, jumlah iterasi, dan runtime dari algoritma pure bruteforce :




Solusi, jumlah iterasi, dan waktu runtime dengan optimisasi :



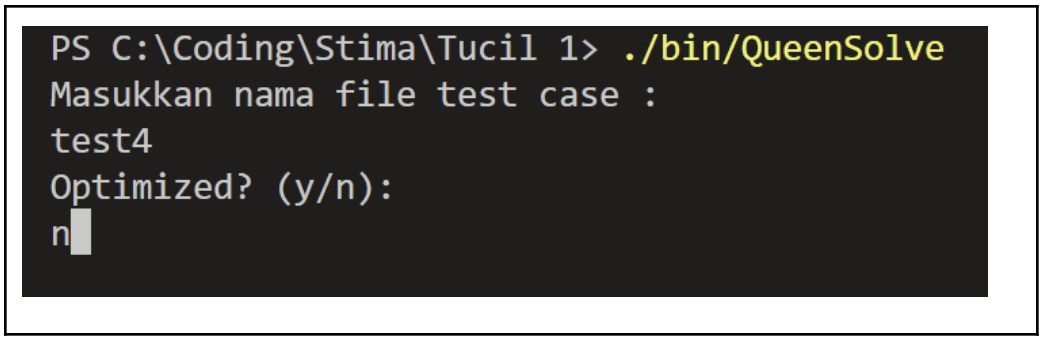

Hasil penyimpanan solusi di solusi_test3.txt (tidak ada) :



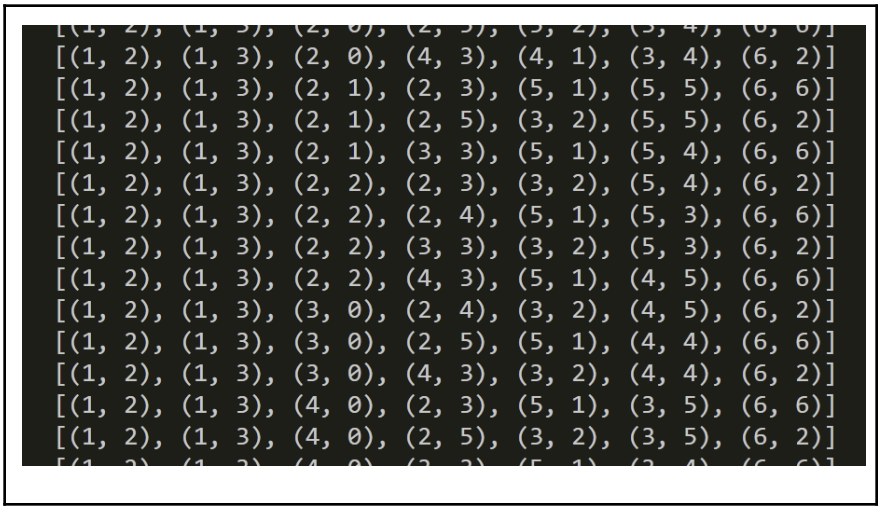

d. Test Case 4

File test4.txt :

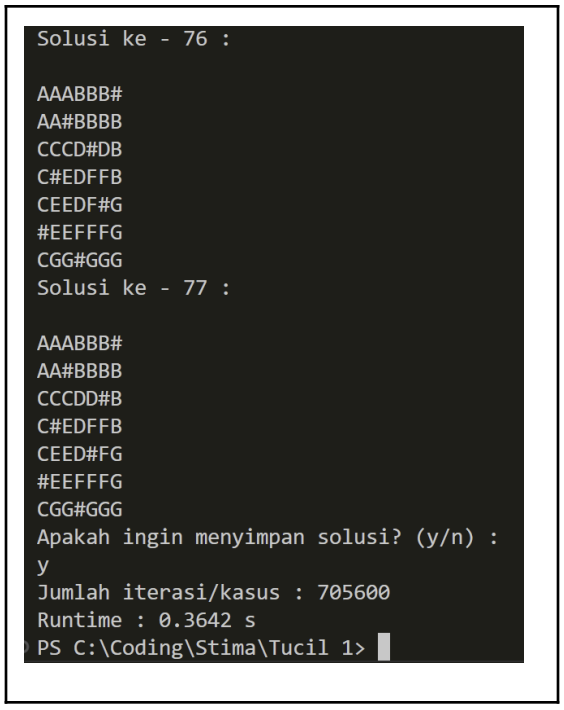

Menginput nama file dan pencarian solusi secara bruteforce tanpa optimisasi :



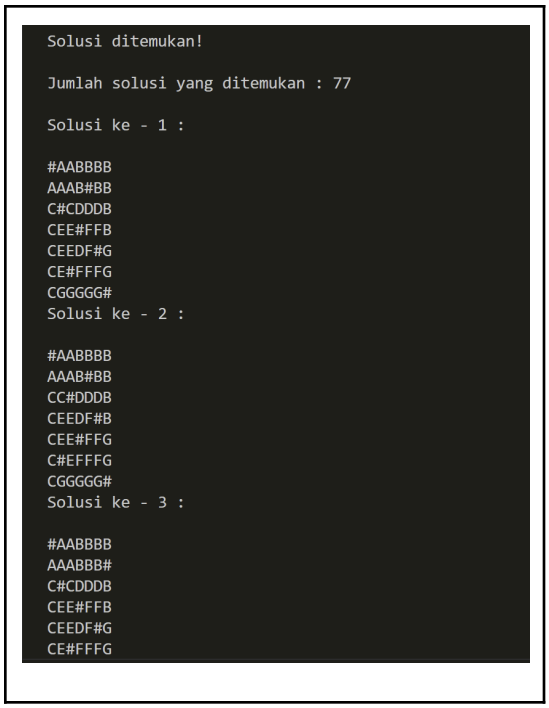
Live update :



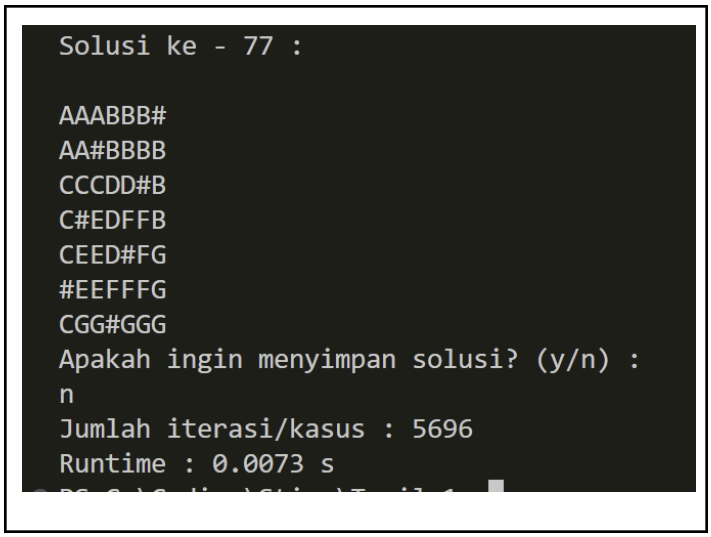

Penyimpanan, jumlah iterasi, dan runtime dari algoritma pure bruteforce :



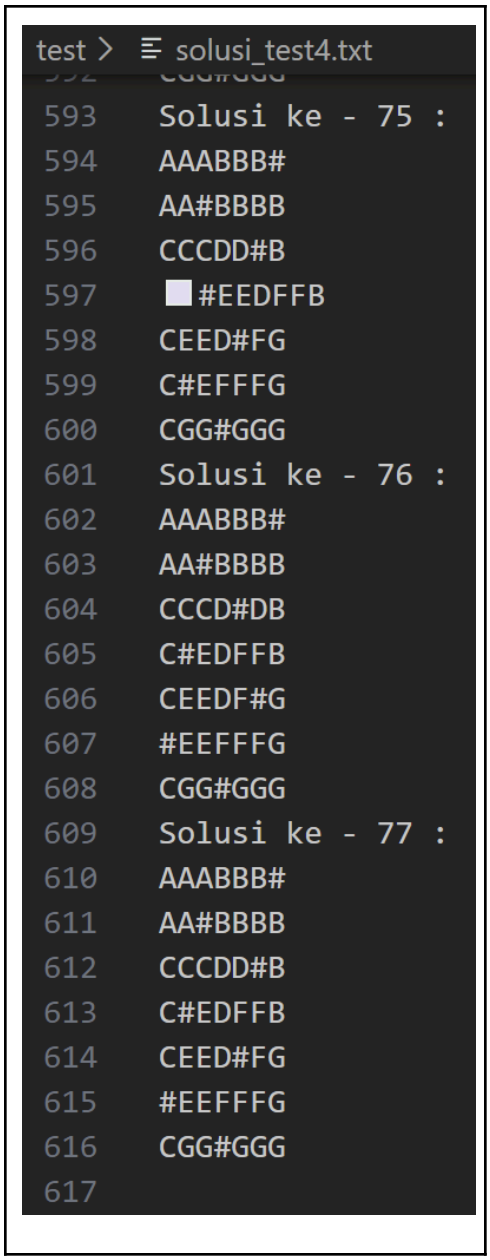

Beberapa solusi dari test case :




Jumlah iterasi, dan waktu runtime dengan optimisasi :



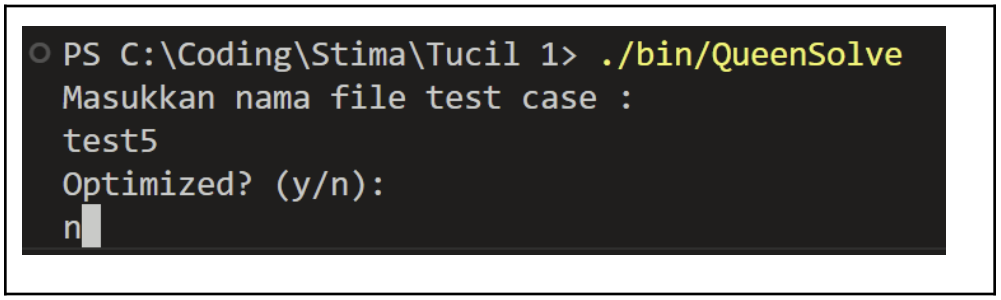

Hasil penyimpanan solusi di solusi_test4.txt :




e. Test Case 5

File test5.txt :


Menginput nama file dan pencarian solusi secara bruteforce tanpa optimisasi :



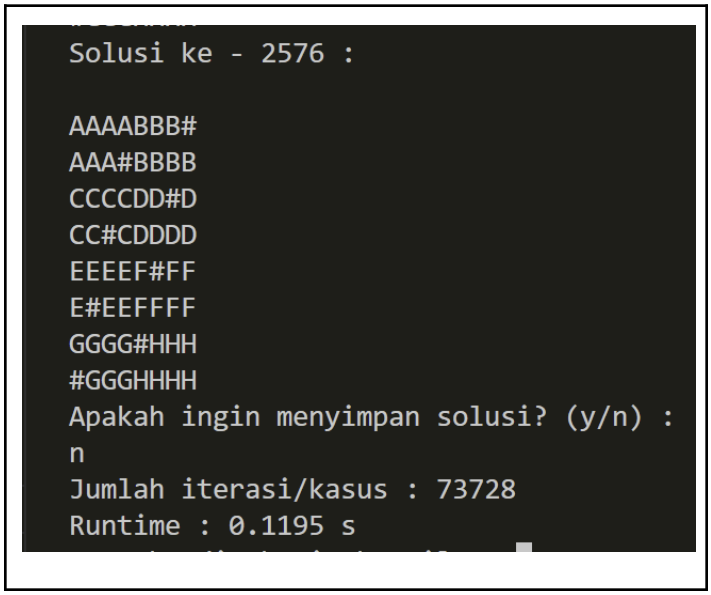

Live update :

```
[(0, 1), (1, 6), (3, 2), (2, 6), (4, 0), (4, 6), (7, 2), (7, 7)]
[(0, 1), (1, 6), (3, 2), (2, 6), (4, 1), (4, 6), (7, 1), (6, 7)]
[(0, 1), (1, 6), (3, 2), (2, 6), (4, 2), (4, 6), (6, 3), (7, 7)]
[(0, 1), (1, 6), (3, 2), (2, 6), (4, 3), (4, 6), (6, 2), (6, 7)]
[(0, 1), (1, 6), (3, 2), (2, 6), (5, 0), (4, 6), (6, 0), (7, 7)]
[(0, 1), (1, 6), (3, 2), (2, 6), (5, 1), (4, 5), (7, 3), (6, 7)]
[(0, 1), (1, 6), (3, 2), (2, 6), (5, 2), (4, 5), (7, 1), (7, 7)]
[(0, 1), (1, 6), (3, 2), (2, 6), (5, 3), (4, 5), (7, 0), (6, 7)]
[(0, 1), (1, 6), (3, 2), (2, 7), (4, 0), (4, 5), (6, 2), (7, 7)]
[(0, 1), (1, 6), (3, 2), (2, 7), (4, 1), (4, 5), (6, 1), (6, 7)]
[(0, 1), (1, 6), (3, 2), (2, 7), (4, 2), (4, 4), (7, 3), (7, 7)]
[(0, 1), (1, 6), (3, 2), (2, 7), (4, 3), (4, 4), (7, 2), (6, 7)]
[(0, 1), (1, 6), (3, 2), (2, 7), (5, 0), (4, 4), (7, 0), (7, 7)]
[(0, 1), (1, 6), (3, 2), (2, 7), (5, 1), (4, 4), (6, 3), (6, 7)]
```

Solusi, penyimpanan, jumlah iterasi, dan runtime dari algoritma pure bruteforce :

```
CC#CDDDD
EEEEF#FF
#EEEFFFF
GGGG#HHH
G#GGHHHH
Solusi ke - 2575 :

AAAABBB#
AAA#BBBB
CCCCDD#D
CC#CDDDD
EEEE#FFF
E#EEFFFF
GGGGH#HH
#GGGHHHH
Solusi ke - 2576 :

AAAABBB#
AAA#BBBB
CCCCDD#D
CC#CDDDD
EEEEF#FF
E#EEFFFF
GGGG#HHH
#GGGHHHH
Apakah ingin menyimpan solusi? (y/n) :
n
Jumlah iterasi/kasus : 16777216
Runtime : 11.6179 s
PS C:\Coding\Stima\Tucil 1>
```

Solusi, jumlah iterasi, dan waktu runtime dengan optimisasi :




Hasil penyimpanan solusi di solusi_test5.txt :

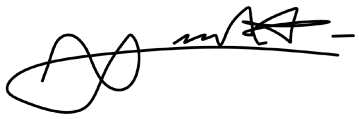

**VI. Lampiran**

Link Repository Program : link

| No | Poin | Ya | Tidak |
|---|---|---|---|
| 1 | Program berhasil di kompilasi tanpa kesalahan | ✔ | |
| 2 | Program berhasil di jalankan | ✔ | |
| 3 | Solusi yang diberikan program benar dan mematuhi aturan permainan | ✔ | |
| 4 | Program dapat membaca masukan berkas .txt serta menyimpan solusi dalam berkas .txt | ✔ | |
| 5 | Program memiliki Graphical User Interface | | ✔ |

| | | | |
|---|---|---|---|
| | (GUI) | | |
| 6 | Program dapat menyimpan solusi dalam bentuk file gambar | | ✔ |

Tugas ini disusun sepenuhnya tanpa bantuan kecerdasan buatan (Generative AI), melainkan hasil pemikiran dan analisis mandiri.



Nathan Edward Christofer Marpaung