Laporan Tugas Kecil 1
Strategi Algoritma
Penyelesaian Permainan Queens dengan Algoritma Brute Force

Vincent Rionarlie
NIM: 13524031
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung

17 Februari 2026

Daftar Isi

1 Pendahuluan

1.1 Latar Belakang

Mata kuliah IF2211 Strategi Algoritma telah menurunkan tugas kecil 1 dengan output berupa penyelesaian permainan Queens menggunakan algoritma Brute-force. Permainan Queens (dari LinkedIn) sendiri adalah sebuah puzzle dimana pemain harus menempatkan n buah ratu dalam papan $n \times n$ dengan beberapa aturan.

1.2 Tujuan

Tugas kecil ini bertujuan untuk:

1. Mengimplementasikan algoritma brute force untuk menyelesaikan permainan Queens
2. Memahami kompleksitas dan efisiensi algoritma brute force

1.3 Spesifikasi Permainan Queens

Permainan Queens memiliki aturan sebagai berikut:

- Terdapat papan berukuran $n \times n$
- Terdapat n buah ratu untuk ditempatkan
- Pemain harus menempatkan seluruh queens pada papan sehingga:
  - Setiap baris dan kolom hanya berisi tepat satu queen
  - Tidak ada dua queens yang saling *adjacent*, yaitu berada pada baris, kolom, atau diagonal yang berjarak satu kotak dengan satu sama lain
  - Setiap daerah warna hanya berisi tepat satu queen

2 Algoritma Brute Force

2.1 Konsep Algoritma

Algoritma brute force adalah pendekatan penyelesaian masalah dengan mencoba semua kemungkinan solusi yang ada. Untuk permainan Queens, algoritma yang diterapkan adalah brute force murni:

1. Menempatkan queen satu per satu pada setiap baris yang belum terisi kolomnya
2. Memeriksa apakah hasil akhir memenuhi syarat
3. Jika belum, naik ke baris sebelumnya dan lanjutkan tahap 1

2.2 Optimasi Backtrack

Penggunaan backtracking bertujuan untuk mengurangi time complexity dari program:

- Menempatkan queen satu per satu pada setiap baris
- Memeriksa apakah setiap queen yang akan ditempatkan memenuhi syarat
- Jika pada satu baris sebuah queen tidak bisa lagi ditempatkan, langsung mundur ke baris sebelumnya dan lanjutkan tahap 1

2.3 Langkah-langkah Algoritma

2.3.1 Inisialisasi

1. Baca input papan permainan berukuran $n \times n$
2. Inisialisasi variabel `dataRow[n]` dengan nilai awal -1, yang merupakan array berisi kolom queen terpilih pada suatu baris i
3. Inisialisasi variabel `dataCol[n]` dengan nilai awal -1, yang merupakan array berisi baris queen terpilih pada suatu kolom j

2.3.2 Proses Penempatan Queens

1. **Mulai dari suatu baris**
   - Iterasi setiap kolom dari baris tersebut
   - Coba letakkan queen pada posisi `(i, j)`
2. **Validasi lokasi queen pada suatu baris**
   - Jika tidak ada kolom yang dapat diletakkan queen, kembali iterasi kolom untuk baris sebelumnya (tahap 2)
   - Periksa apakah kolom sudah berisi queen

- Jika valid, lanjutkan ke langkah berikutnya
- Jika tidak valid, ulangi tahap 2 untuk kolom berikutnya

3. **Simpan variabel**
    - Jika posisi valid, set `dataCol[j] = i`, juga set `dataRow[i] ] j`
    - Ulangi tahap 1 untuk baris berikutnya
4. **Validasi akhir**
    - Setelah semua queens ditempatkan, validasi syarat *adjacent* dan syarat *warna*
    - Jika tidak sesuai, kembali iterasi kolom pada baris terakhir (tahap 2, dipastikan tidak ada kolom kosong pada baris terakhir, maka baris diatasnya yang akan digerakkan)
    - Jika sesuai, lanjutkan ke tahap berikutnya
5. **Hasil akhir**
    - Menampilkan output CLI dari posisi queen di antara peta warna awal
    - Menampilkan informasi waktu dan jumlah kasus
    - Konfirmasi penyimpanan hasil kompilasi

2.3.3 Pseudocode

Algorithm 1 Brute Force untuk Queens Puzzle

```
1: function BruteForceQueens(board, n)
2:     dataRow[0..n − 1] ← −1                ▷ Posisi kolom queen untuk setiap baris
3:     dataCol[0..n − 1] ← −1                ▷ Baris mana yang menempati setiap kolom
4:     i ← 0                                 ▷ Baris saat ini
5:     mark ← 0                              ▷ Kolom awal untuk pencarian
6:     while true do
7:                                           ▷ Tahap 1: Mulai dari suatu baris
8:         while i ≥ 0 and i < n do
9:             found ← false
10:            j ← mark                      ▷ Iterasi setiap kolom dari baris tersebut
11:            while j < n do
12:                                          ▷ Tahap 2: Validasi lokasi queen pada suatu baris
13:                if dataCol[j] = −1 then   ▷ Periksa apakah kolom sudah berisi queen
14:                                          ▷ Tahap 3: Simpan variabel
15:                    dataCol[j] ← i        ▷ Set dataCol[j] = i
16:                    dataRow[i] ← j        ▷ Set dataRow[i] = j
17:                    mark ← 0
18:                    i ← i + 1             ▷ Ulangi tahap 1 untuk baris berikutnya
19:                    found ← true
20:                    break
21:                else
22:                    j ← j + 1             ▷ Jika tidak valid, ulangi tahap 2 untuk kolom berikutnya
23:                end if
24:            end while
25:            if not found then             ▷ Tidak ada kolom yang dapat diletakkan queen
26:                if i = 0 then
27:                    break
28:                end if
29:                dataCol[dataRow[i − 1]] ← −1   ▷ Kembali iterasi kolom untuk baris sebelumnya
30:                mark ← dataRow[i − 1] + 1
31:                dataRow[i − 1] ← −1
32:                i ← i − 1
33:            end if
34:        end while
35:                                          ▷ Tahap 4: Validasi akhir
36:        valid ← true
37:                                          ▷ Validasi syarat adjacent
38:        for k ← 1 to n − 1 do
39:            cur ← dataRow[k]
40:            prev ← dataRow[k − 1]
41:            if cur = prev − 1 or cur = prev + 1 then
42:                valid ← false
43:                break
44:            end if
45:        end for
46:                                          ▷ Validasi syarat warna
```

2.4 Kompleksitas Algoritma

- **Kompleksitas Waktu**: $O(n! \times n^2)$ worst case, $O(n!)$ untuk tahap penempatan queen, dan $O(n^2)$ untuk tahap validasi queen.
- **Kompleksitas Ruang**: $O(n)$ karena tidak ada array dua dimensi, juga tidak ada rekursi
- Optimasi backtracking memberikan kompleksitas waktu worst case $O(n!)$

3 Implementasi Program

3.1 Bahasa Pemrograman

Program diimplementasikan menggunakan bahasa pemrograman C++ 20 karena:

- Kecepatan lebih tinggi
- Penulis lebih mahir menggunakan bahasa ini

3.2 Source Code

```
// main.cpp

#include <iostream>
#include <fstream>
#include <string>
#include <map>
#include <utility>
#include <vector>
#include <chrono>
using namespace std;

void initData(int *data, int n) {
    for (int i=0; i<n; i++) {
        data[i] = -1;
    }
}

bool isEmpty(int n, int cur, int *dataCol, int prev, bool area) {
    if (cur == prev-1 || cur==prev+1 || cur==prev) return false;
    if (dataCol[cur] != -1) return false;
    return !area;
}

void printRes(int *dataRow, int *dataCol, vector<string> row, int n
    ) {
    for (int i=0; i<n; i++) {
        for (int j=0; j<dataRow[i]; j++) {
            cout << row[i][j];
        }
        if (dataRow[i] != -1) cout << "*";;
        for (int j=dataRow[i]+1; j<n; j++) {
            cout << row[i][j];
        }
        cout << endl;
    }
}

int main() {
    // input
    string fileName = "./testcase/", tempFilename;
    cout << "-- Queens Solver --\n";
    cout << "Input your test case filename in .txt (e.g. input.txt)
    :\n";
```

```
    cout << ">> ";
    cin >> tempFilename;
    fileName.append(tempFilename);
    ifstream file(fileName);

    if (!file.is_open()) {
        cout << "File not found!\n";
        return 1;
    }

    vector<string> row;
    int it=0, n=0;
    string temp;
    getline(file, temp);
    n = temp.length();
    row.push_back(temp);
    while (it < n-1 && getline(file, temp)) {
        row.push_back(temp);
        it++;
    }
    file.close();

    if (row.size() != n) {
        cout << "Invalid file: Row and Column mismatch\n";
        cout << "Row = " << row.size() << "\n";
        cout << "Col = " << n << "\n";
        return 1;
    }

    // vars
    map<char, bool> alfa;
    for (int i=0; i<n; i++) {
        for (int j=0; j<n; j++) {
            if (alfa.find(row[i][j]) == alfa.end()) {
                alfa[row[i][j]] = false;
            }
        }
    }
    if (alfa.size() != n) {
        cout << "Invalid file: Color mismatch" << endl;
        return 1;
    }

    int iter=1, i=0, mark = 0, dataRow[n], dataCol[n];
    initData(dataRow, n);
    initData(dataCol, n);

    // process
    int choice;
    cout << "Choose one of the two algorithms:\n";
    cout << "1. Pure Brute-force\n";
    cout << "2. Backtrack optimization\n";
    cout << ">> ";
    cin >> choice;

    cout << "\nSearching...\n";
```

```
98      cout << "----------\n";
99      auto start = chrono::high_resolution_clock::now();
100     if (choice == 1) { // exhaustive
101         while (true) {
102             while(i >= 0 && i < n) {
103                 bool found = false;
104                 int j=mark;
105                 while(j < n) {
106                     if (dataCol[j] == -1) {
107                         dataCol[j] = i;
108                         dataRow[i] = j;
109                         mark = 0;
110                         i++;
111                         found = true;
112                         break;
113                     }
114                     else j++;
115                 }
116                 if (!found) {
117                     if (i==0) break;
118                     dataCol[dataRow[i-1]] = -1;
119                     mark = dataRow[i-1]+1;
120                     dataRow[i-1] = -1;
121                     i--;
122                     iter++;
123                 }
124                 if (iter%100000000 == 0) {
125                     printRes(dataRow, dataCol, row, n);
126                     cout << "Tries = " << iter << endl;
127                     cout << "----------\n";
128                 }
129             }
130 
131             // iter++;
132             bool check = true;
133 
134             for (int k = 1; k < n; k++) {
135                 int cur = dataRow[k];
136                 int prev = dataRow[k-1];
137                 if (cur == prev - 1 || cur == prev + 1) {
138                     check = false;
139                     break;
140                 }
141             }
142 
143             if (check) {
144                 map<char, bool> secAlfa;
145                 for (int k = 0; k < n; k++) {
146                     char c = row[k][dataRow[k]];
147                     if (secAlfa.count(c)) {
148                         check = false;
149                         if (iter == 1) {
150                             cout << "alfa\n";
151                         }
152                         break;
153                     }
```

```
                    secAlfa[c] = true;
                }
            }

            if (check) {
                break;
            } else {
                if (i==0) break;
                i--;
                int prevCol = dataRow[i];
                dataCol[prevCol] = -1;
                dataRow[i] = -1;
                mark = prevCol + 1;
            }
        }
    } else if (choice == 2) { // backtrack
        while(i >= 0 && i < n) {
            int j=mark;
            while(j < n) {
                int prev = (i==0)?-2:dataRow[i-1];
                if (isEmpty(n, j, dataCol, prev, alfa[row[i][j]]))
{ // masukin kalo empty doang
                    dataCol[j] = i;
                    dataRow[i] = j;
                    alfa[row[i][j]] = true;
                    mark = 0;
                    i++;
                    break;
                } else {
                    j++;
                }
            }
            if (j >= n) {
                if (i==0) break;
                alfa[row[i-1][dataRow[i-1]]] = false;
                dataCol[dataRow[i-1]] = -1;
                mark = dataRow[i-1]+1;
                dataRow[i-1] = -1;
                i--;
                iter++;
            }
            // printRes(dataRow, dataCol, row, n);
            // cout << "----------\n";
            if (iter%100000000 == 0) {
                printRes(dataRow, dataCol, row, n);
                cout << "Tries = " << iter << endl;
                cout << "----------\n";
            }
        }
    } else {
        cout << "Invalid choice!\n";
        return 1;
    }

    // print
    printRes(dataRow, dataCol, row, n);
```

```
209     cout << "----------\n";
210 
211     auto end = chrono::high_resolution_clock::now();
212     auto dur = chrono::duration_cast<chrono::milliseconds>(end-
    start);
213     cout << "Program elapsed in " << dur.count() << "ms\n";
214     cout << "A total of " << iter << " cases were checked\n";
215 
216     if (dataRow[0] == -1 && dataCol[0] == -1) {
217         cout << "Test case configuration can't be solved!\n";
218     } else {
219         cout << "\nDo you want to save the result?\n";
220         cout << "1. Yes\n";
221         cout << "2. No\n";
222         cout << ">> ";
223         cin >> choice;
224 
225         if (choice == 1) {
226             fileName = "./result/", tempFilename = "";
227             cout << "\nInput your target filename in .txt (e.g.
    output.txt):\n";
228             cout << ">> ";
229             cin >> tempFilename;
230             fileName.append(tempFilename);
231             ofstream res(fileName);
232 
233             for (int i=0; i<n; i++) {
234                 for (int j=0; j<dataRow[i]; j++) {
235                     res << row[i][j];
236                 }
237                 if (dataRow[i] != -1) res << "*";;
238                 for (int j=dataRow[i]+1; j<n; j++) {
239                     res << row[i][j];
240                 }
241                 res << endl;
242             }
243             res.close();
244             cout << "File saved!\n";
245         }
246     }
247 
248     cout << "\nProgram terminated\n";
249     return 0;
250 }
```

Listing 1: Keseluruhan kode program

3.3 Struktur Program

Program terdiri dari:

1. **Method initData**: Inisialisasi array menjadi -1
2. **Method isEmpty**: Mengembalikan apakah sebuah sel dapat diletakkan queen atau tidak (digunakan untuk metode optimasi backtrack)

3. **Method printRes**: Mengeluarkan output peta warna beserta posisi queen

4 Hasil Pengujian

4.1 Format Input

File input memiliki format, seperti contoh:

```
AAAA
BBBC
CCCC
CDDC
```

4.2 Contoh Pengujian

4.2.1 Test Case 1: Papan 4x4

Input:

```
AAAA
BAAA
BBBD
BCCD
```

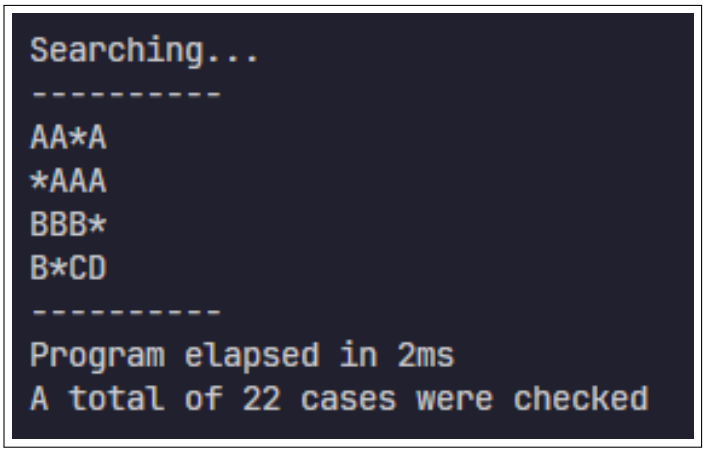
Output:

```
-- Queens Solver --
Input your test case filename in .txt (e.g. input.txt):
>> tc1.txt
Choose one of the two algorithms:
1. Pure Brute-force
2. Backtrack optimization
>> 1

Searching...
----------
AA*A
*AAA
BBB*
B*CD
----------
Program elapsed in 2ms
A total of 22 cases were checked

Do you want to save the result?
1. Yes
2. No
>> 2

Program terminated
```



Gambar 1: Screenshot Test Case 1 - Papan 4x4

4.2.2 Test Case 2: Papan 8x8

Input:

```
AAAAAAAA
BCCCDDDA
BCEEEEDA
BCCEEFFA
BBFEEFAA
BBFGGFAA
HHFFFFAA
HAAAAAAA
```

Output:

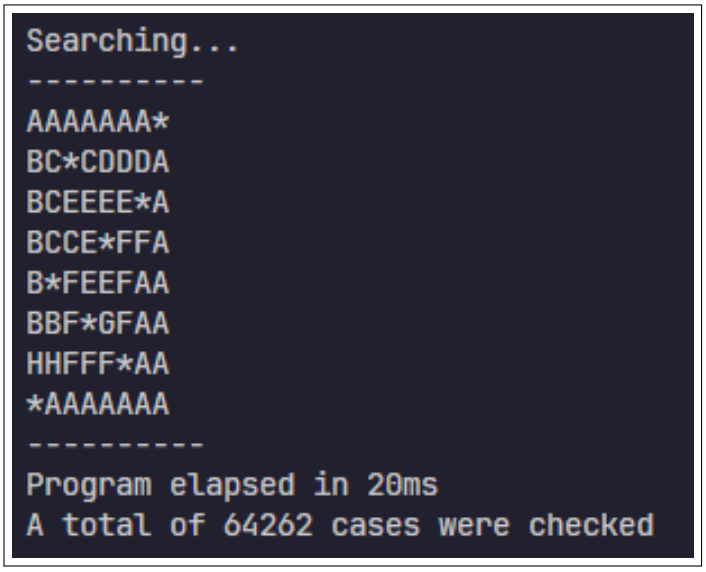
```
-- Queens Solver --
Input your test case filename in .txt (e.g. input.txt):
>> tc2.txt
Choose one of the two algorithms:
1. Pure Brute-force
2. Backtrack optimization
>> 1

Searching...
----------
AAAAAAA*
BC*CDDDA
BCEEEE*A
BCCE*FFA
B*FEEFAA
BBF*GFAA
HHFFF*AA
*AAAAAAA
----------
Program elapsed in 20ms
```

```
A total of 64262 cases were checked

Do you want to save the result?
1. Yes
2. No
>> 2

Program terminated
```



Gambar 2: Screenshot Test Case 2 - Papan 8x8

4.2.3 Test Case 3: Papan 9x9

Input:

```
AAABBCCCD
ABBBBCECD
ABBBDCECD
AAABDCCCD
BBBBDDDDD
FGGGDDHDD
FGIGDDHDD
FGIGDDHDD
FGGGDDHHH
```

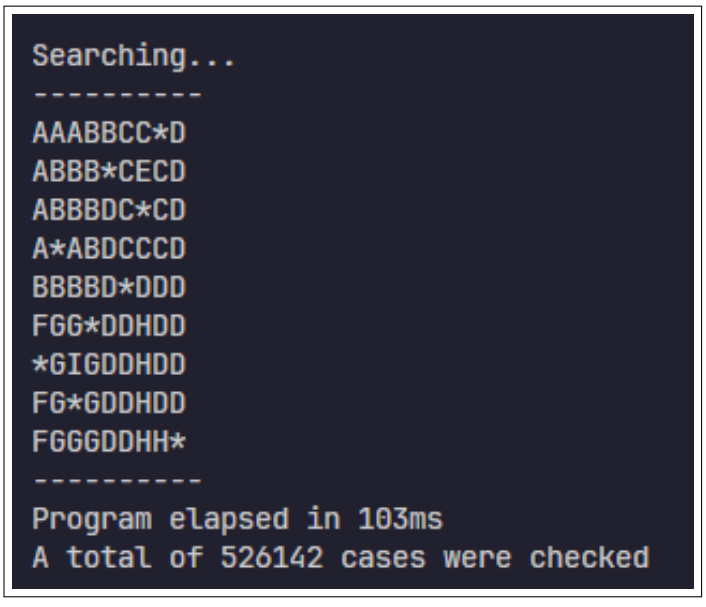
Output:

```
-- Queens Solver --
Input your test case filename in .txt (e.g. input.txt):
>> tc3.txt
Choose one of the two algorithms:
1. Pure Brute-force
2. Backtrack optimization
>> 1
```

```
Searching...
----------
AAABBCC*D
ABBB*CECD
ABBBDC*CD
A*ABDCCCD
BBBBD*DDD
FGG*DDHDD
*GIGDDHDD
FG*GDDHDD
FGGGDDHH*
----------
Program elapsed in 103ms
A total of 526142 cases were checked

Do you want to save the result?
1. Yes
2. No
>> 2

Program terminated
```



Gambar 3: Screenshot Test Case 3 - Papan 9x9

4.2.4 Test Case 4: Papan 14x14

Input:

```
AAAAAAAAAAAAAA
BBBBBBBBBBBBBB
```

```
CCCCCCCCCCCCCC
DDDDDDDDDDDDDD
EEEEEEEEEEEEEE
FFFFFFFFFFFFFF
GGGGGGGGGGGGG
HHHHHHHHHHHHH
IIIIIIIIIIIII
JJJJJJJJJJJJJ
KKKKKKKKKKKKK
LLLLLLLLLLLLL
MMMMMMMMMMMMM
NNNNNNNNNNNNN
```

Output:

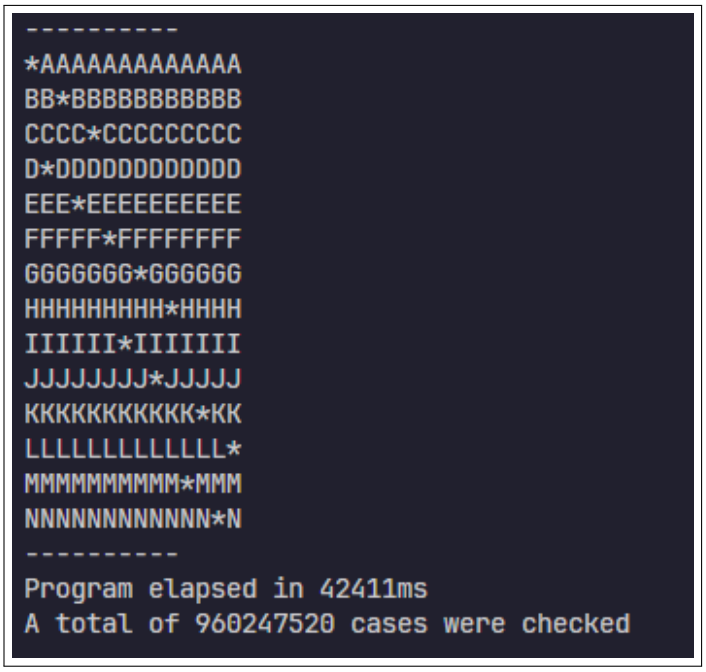
```
-- Queens Solver --
Input your test case filename in .txt (e.g. input.txt):
>> tc4.txt
Choose one of the two algorithms:
1. Pure Brute-force
2. Backtrack optimization
>> 1

Searching...
----------
*AAAAAAAAAAAAA
B*BBBBBBBBBBB
CCC*CCCCCCCCC
DDDDDDDD*DDDDD
EE*EEEEEEEEEE
FFFFFFF*FFFFFF
GGGGGGGGG*GGGG
HHHHH*HHHHHHHH
IIII*IIIIIIIII
JJJJJJJJJJ*JJ
KKKKKKKKKKKKK
LLLLLLLLLLLLL
MMMMMMMMMMMMM
NNNNNNNNNNNNN
Tries = 100000000
....
*AAAAAAAAAAAAA
BB*BBBBBBBBBB
CCC*CCCCCCCCC
DDDD*DDDDDDDDD
EEEEEEE*EEEEEE
FFFFFFFF*FFFFF
GGGGGGGGGG*GGG
HHHHHH*HHHHHHH
```

```
IIIIIIIIIIII*
JJJJJ*JJJJJJJJ
KKKKKKKKKK*KK
LLLLLLLLL*LLLL
M*MMMMMMMMMMM
NNNNNNNNNNN*N
Tries = 900000000
----------
*AAAAAAAAAAAA
BB*BBBBBBBBBB
CCCC*CCCCCCCCC
D*DDDDDDDDDDD
EEE*EEEEEEEEE
FFFFF*FFFFFFFF
GGGGGGG*GGGGGG
HHHHHHHHH*HHHH
IIIIII*IIIIIII
JJJJJJJJ*JJJJJ
KKKKKKKKKK*KK
LLLLLLLLLLLL*
MMMMMMMMM*MMM
NNNNNNNNNNN*N
----------
Program elapsed in 42411ms
A total of 960247520 cases were checked

Do you want to save the result?
1. Yes
2. No
>> 2

Program terminated
```



Gambar 4: Screenshot Test Case 4 - Papan 14x14

4.2.5 Test Case 5: Papan 5x5 tanpa solusi

Input:

```
AAAAA
BCCCD
BCCDD
BCCDE
BBBBB
```

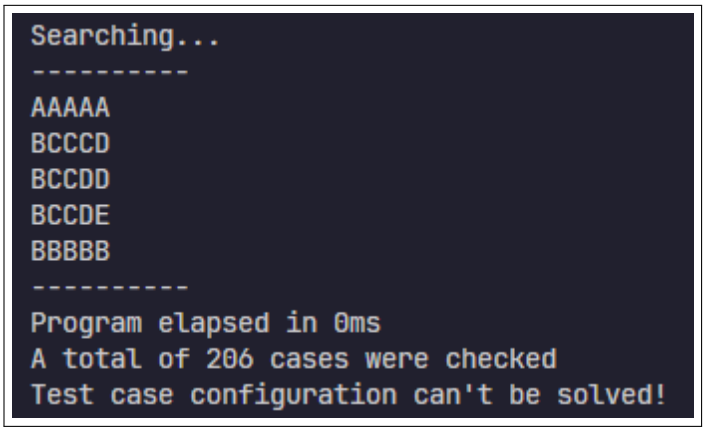
Output:

```
-- Queens Solver --
Input your test case filename in .txt (e.g. input.txt):
>> tc5.txt
Choose one of the two algorithms:
1. Pure Brute-force
2. Backtrack optimization
>> 1

Searching...
----------
AAAAA
BCCCD
BCCDD
BCCDE
BBBBB
----------
Program elapsed in 0ms
```

```
A total of 206 cases were checked
Test case configuration can’t be solved!

Program terminated
```



Gambar 5: Screenshot Test Case 5 - Papan 5x5 tanpa solusi

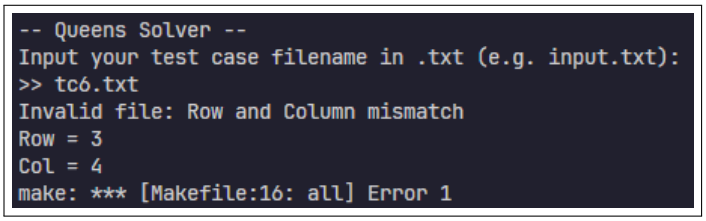
4.2.6 Test Case 6: Papan 3x4

Input:

```
AAAA
BBBB
CCCC
```

Output:

```
-- Queens Solver --
Input your test case filename in .txt (e.g. input.txt):
>> tc6.txt
Invalid file: Row and Column mismatch
Row = 3
Col = 4
make: *** [Makefile:16: all] Error 1
```



Gambar 6: Screenshot Test Case 6 - Papan 3x4

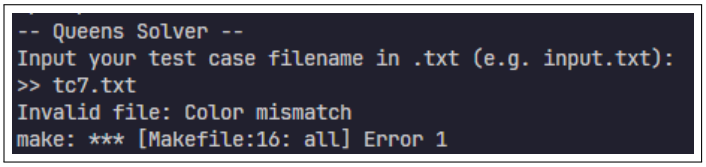
4.2.7 Test Case 7: Papan 5x5 dengan warna tidak cocok

Input:

```
AAAAA
BBBBB
CCCCC
DDDDC
DDDDD
```

Output:

```
-- Queens Solver --
Input your test case filename in .txt (e.g. input.txt):
>> tc7.txt
Invalid file: Color mismatch
make: *** [Makefile:16: all] Error 1
```



Gambar 7: Screenshot Test Case 7 - Papan 5x5 dengan warna tidak cocok

4.3 Analisis Hasil

- Program berhasil menyelesaikan semua test case yang memiliki solusi
- Program dapat mendeteksi konfigurasi peta yang tidak valid
- Waktu eksekusi meningkat secara eksponensial dengan meningkatnya ukuran papan
- Untuk papan 14x14, waktu eksekusi masih wajar ($\leq 1 menit$)

5 Optimisasi (Opsional)

5.1 Optimisasi yang Diterapkan

Optimisasi diterapkan dalam bentuk *backtracking*, dimana penempatan langsung diperiksa valid atau tidak. Jika tidak valid, maka kemungkinan tersebut tidak akan pernah dikunjungi lagi.

5.2 Perbandingan Performa

Ukuran Papan	**Waktu Brute Force**	**Waktu Optimisasi Backtrack**
4x4	2 ms	3 ms
8x8	20 ms	8 ms
9x9	103 ms	9 ms
14x14	42411 ms	13 ms

Tabel 1: Perbandingan waktu eksekusi

6 Kesimpulan

6.1 Kesimpulan

1. Algoritma brute force dapat menyelesaikan permainan Queens dengan mencoba semua kemungkinan penempatan queens dan memvalidasi setiap penempatan akhir

2. Penggunaan algoritma tambahan backtracking membantu dalam memotong waktu pencarian

3. Kompleksitas algoritma adalah $O(n! \times n^2)$ dalam worst case

4. Program berhasil menyelesaikan berbagai test case dengan ukuran papan 4x4 hingga 14x14 (konfigurasi 26x26 tertentu juga dapat diselesaikan dengan mudah dengan backtracking)

6.2 Kelebihan dan Kekurangan

Kelebihan:

- Implementasi sederhana sehingga mudah dipahami
- Solusi terjamin
- Mudah dimodifikasi

Kekurangan:

- Waktu eksekusi meningkat eksponensial seiring meningkatnya ukuran papan
- Kurang efisien untuk papan besar ($n \geq 10$)

7 Repository

Source code lengkap beserta test cases dapat diakses melalui repository GitHub:

https://github.com/Vixrlie/Tucil1_13524031

Repository berisi:

- Source code (`main.cpp`)
- Test cases (`tc1.txt` - `tc7.txt`)
- README dengan instruksi penggunaan
- Laporan dalam format PDF dan LaTeX

8 Referensi

1. Munir, Rinaldi. (2025). *Algoritma Brute Force, Bagian 1*. Program Studi Teknik Informatika ITB.
2. Munir, Rinaldi. (2025). *Algoritma Brute Force, Bagian 2*. Program Studi Teknik Informatika ITB.
3. LinkedIn Queens Game. https://www.linkedin.com/games/queens/

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.

Vincent Rionarlie