

**LAPORAN TUGAS KECIL 1**  
**IF2211 STRATEGI ALGORITMA**  
**PENYELESAIAN PERMAINAN QUEENS LINKEDIN**  
**DENGAN ALGORITMA BRUTE FORCE**



Disusun oleh:  
Reva Natania Sitohang 13524098

**PROGRAM STUDI TEKNIK INFORMATIKA**  
**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA**  
**INSTITUT TEKNOLOGI BANDUNG**  
**BANDUNG**  
**2026**

## DAFTAR ISI

BAB 1 ALGORITMA PENYELESAIAN.....	3
1.1. Algoritma Brute Force.....	3
1.2. Permainan Queens LinkedIn.....	3
1.3. Penyelesaian Permasalahan Queens LinkedIn Secara Brute Force.....	3
BAB 2 SOURCE CODE.....	6
BAB 3 TESTING.....	11
3.1. Test Case 1.....	11
3.2. Test Case 2.....	12
3.3. Test Case 3.....	13
3.4. Test Case 4.....	15
3.5. Test Case 5.....	16
3.6. Test Case 6.....	18
LAMPIRAN.....	20

# BAB 1

## ALGORITMA PENYELESAIAN

### 1.1. Algoritma Brute Force

Algoritma *Brute Force* adalah sebuah strategi pemecahan masalah yang dilakukan dengan cara *straightforward*, sederhana, langsung, dan dengan cara yang jelas. Langkah-langkah algoritma ini mudah dipahami dan biasa didasarkan langsung pada pernyataan persoalan atau definisi konsep yang terlibat. Algoritma ini bekerja dengan mencoba seluruh kemungkinan solusi yang ada secara sistematis hingga ditemukan solusi yang memenuhi seluruh kendala permasalahan, atau hingga seluruh kemungkinan telah dievaluasi tanpa menemukan solusi yang valid.

Salah satu bentuk *brute force* adalah *exhaustive search*. Metode ini bekerja dengan mengenumerasi setiap kemungkinan solusi secara sistematis dan mengevaluasinya satu per satu sampai ditemukan solusi yang tepat. Walaupun algoritma *brute force* sering kali membutuhkan biaya komputasi yang besar dan waktu yang lama, algoritma ini memiliki kelebihan yaitu dapat menjamin penemuan solusi apabila solusi tersebut memang ada. Oleh karena itu, algoritma *brute force* sering digunakan sebagai pendekatan awal dalam pemecahan masalah, membandingkan algoritma yang lebih efisien, dan sebagai metode yang sesuai untuk persoalan dengan ukuran input kecil hingga menengah.

### 1.2. Permainan Queens LinkedIn

Permainan Queens LinkedIn adalah sebuah gim logika yang dimainkan pada papan persegi berwarna. Tujuan utama dari permainan ini adalah menempatkan *queen* pada papan tersebut dengan memenuhi batasan-batasan berikut:

- Tiap baris hanya boleh memiliki tepat satu *queen*.
- Tiap kolom hanya boleh memiliki tepat satu *queen*.
- Tiap daerah warna hanya boleh memiliki tepat satu *queen*.
- Satu *queen* tidak boleh ditempatkan bersebelahan dengan *queen* lainnya, baik secara horizontal, vertikal, maupun diagonal.

Tujuan utama dari permasalahan ini adalah membuat program yang mampu menemukan solusi yang valid berdasarkan papan yang diberikan atau menyatakan jika tidak ada solusi yang memungkinkan.

### 1.3. Penyelesaian Permasalahan Queens LinkedIn Secara *Brute Force*

Permasalahan Queens LinkedIn merupakan sebuah pencarian konfigurasi atau penempatan yang valid dalam sebuah ruang kemungkinan yang sangat besar. Diberikan sebuah papan berukuran  $n \times n$  dengan tujuan utama untuk menentukan posisi peletakan  $n$  buah *queen* sehingga seluruh batasan permainan terpenuhi.

Algoritma *brute force* yang digunakan dalam program ini bekerja dengan prinsip yaitu menghasilkan seluruh kemungkinan penempatan yang mungkin, lalu mengevaluasi masing-masing kemungkinan secara menyeluruh sampai ditemukan solusi yang tepat atau seluruh ruang kemungkinan sudah habis diperiksa.

Agar proses enumerasi dapat dilakukan secara sistematis, solusi dapat direpresentasikan dalam bentuk array satu dimensi sepanjang  $n$ . Indeks merepresentasikan baris, sedangkan nilai pada indeks tersebut merepresentasikan kolom tempat queen diletakkan. Dengan representasi ini, aturan tepat satu queen per baris terpenuhi. Masalah selanjutnya adalah mencari kolom yang memenuhi semua batasan lain. Proses pencarian dilakukan dengan cara *brute force (exhaustive search)*, yaitu menghasilkan kemungkinan penempatan secara sistematis lalu mengevaluasi semua kemungkinan dengan pemeriksaan terhadap semua aturan permainan sampai ditemukan penempatan yang benar atau semua kemungkinan sudah habis.

Program yang dibuat menyediakan dua mode *brute force* yang berbeda yaitu dengan mode ruang pencarian  $n^n$  dan mode ruang pencarian  $n!$ . Keduanya sama-sama brute force karena keduanya mengenumerasi semua kandidat secara menyeluruh tanpa heuristik, tapi berbeda pada seberapa besar ruang kandidat dieksplorasi.

a. Algoritma Brute Force dengan Enumerasi Kombinasi

Pada mode ini, algoritma bekerja dengan brute force dimana papan yang memiliki  $n^2$  sel dianggap sebagai himpunan seluruh kemungkinan posisi queen. Algoritma ini membangkitkan semua kombinasi pemilihan  $n$  sel dari total  $n^2$  sel tersebut. Ruang kandidat yang dieksplorasi pada mode ini berukuran:

$$\binom{n^2}{n}$$

Algoritma mencoba setiap kemungkinan cara memilih  $n$  sel berbeda dari papan tanpa mempertimbangkan apakah penempatan tersebut melanggar aturan permainan.

Pertama, dibentuk sebuah kombinasi awal. Setiap indeks sel diubah menjadi koordinat dua dimensi  $(r,c)$  lalu membentuk penempatan queen sebanyak  $n$  buah. Di sini, tidak ada aturan yang diterapkan. Jika dibayangkan pada papan, ini setara dengan meletakkan queen berurutan di sel-sel paling awal. Untuk  $n = 4$ , konfigurasi awal berarti queen berada di indeks 0,1,2,3 yang semuanya berada pada baris pertama. Konfigurasi ini hampir pasti tidak valid, namun tetap dicoba karena brute force tidak menyaring kandidat di awal. Setelah semua queen ditempatkan dan terbentuk sebuah kombinasi, dilakukan proses validasi. Jika memenuhi aturan, solusi telah ditemukan dan proses pencarian dihentikan. Jika tidak, algoritma beralih ke penempatan berikutnya.

Perpindahan dari satu kandidat ke kandidat berikutnya dilakukan dengan mekanisme enumerasi kombinasi secara sistematis. Queen yang direpresentasikan oleh indeks paling kanan akan bergeser terlebih dahulu ke sel berikutnya selama masih memungkinkan. Jika sudah mencapai batas maksimum, maka queen di sebelah kirinya yang digeser satu langkah, dan queen-queen di sebelah kanan diatur ulang ke posisi

terkecil yang masih memungkinkan agar tetap membentuk kombinasi yang sah. Pola ini membuat seluruh kemungkinan pemilihan  $n$  sel dari  $n^2$  sel dihasilkan satu per satu tanpa pengulangan dan tanpa ada kandidat yang terlewat.

b. Mode Ruang Pencarian  $n!$

Dalam mode ini, algoritma menggunakan pengamatan sederhana dari aturan permainan, yaitu setiap kolom hanya boleh memiliki tepat satu queen. Karena representasi kolom sudah menjamin satu queen per baris, maka penempatan yang valid haruslah merupakan suatu permutasi dari himpunan kolom. Maka dari itu, ruang pencarian yang dieksplorasi merupakan seluruh permutasi kolom yang jumlahnya  $n!$ . Algoritma memulai dari satu permutasi awal kemudian menghasilkan permutasi berikutnya secara sistematis, dan setiap permutasi penuh dianggap sebagai satu kemungkinan penempatan lengkap yang bisa diuji. Pada mode ini, tidak dilakukan evaluasi parsial selama penempatan, pemeriksaan tetap dilakukan hanya ketika penempatan lengkap sudah tersedia. Setiap permutasi lengkap kemudian divalidasi terhadap semua aturan. Jika valid, penempatan tersebut menjadi solusi. Jika tidak, algoritma lanjut ke kemungkinan selanjutnya hingga seluruh  $n!$  kemungkinan habis.

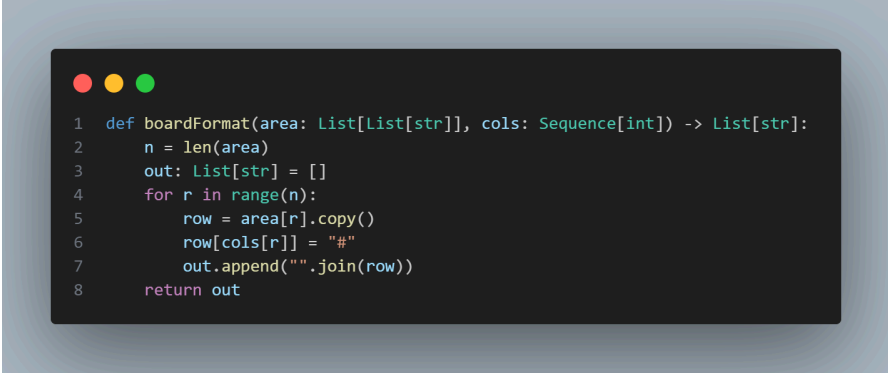
Mode ini jauh lebih optimal karena mengurangi redundansi struktural dibanding mode  $n^n$  yang menghasilkan banyak kemungkinan yang jelas melanggar aturan unik kolom. Walaupun ini lebih cepat dalam waktu komputasi, mode ini termasuk *brute force* karena masih memenuhi karakter utama *brute force* karena algoritma ini mengenumerasi seluruh kandidat dalam ruang yang didefinisikan secara sistematis tanpa adanya strategi lain seperti heuristik untuk menebak penempatan yang lebih menjanjikan, dan tanpa pruning berbasis pengecekan parsial yang memotong ruang pencarian sebelum kandidat lengkap divalidasi. Perbedaan mode ini hanya dalam definisi ruang kandidat yang sejak awal sudah disesuaikan dengan salah satu aturan permainan.

## BAB 2

### SOURCE CODE

Program ini diimplementasikan menggunakan Python sebagai bahasa pemrograman. Ketika dijalankan, program membaca papan menggunakan `core.boardParser(path)` sehingga didapatkan nilai  $n$  dan matriks area. Program memilih mode pencarian antara `pow` atau `fact` allu memvalidasi kandidat penempatan dengan fungsi pengecekan aturan pada `core.py`.

Fungsi ini membentuk representasi papan solusi dalam bentuk list string. `cols[r]` menyatakan kolom queen pada baris `r`. Fungsi menyalin setiap baris papan area, mengganti karakter pada posisi queen dengan '#', lalu menggabungkan baris menjadi string.



```
1 def boardFormat(area: List[List[str]], cols: Sequence[int]) -> List[str]:
2     n = len(area)
3     out: List[str] = []
4     for r in range(n):
5         row = area[r].copy()
6         row[cols[r]] = "#"
7         out.append("".join(row))
8     return out
```

Fungsi ini mengimplementasikan brute force dengan cara mengenumerasi semua kemungkinan penempatan queen berdasarkan pemilihan sel pada papan

```

1 def solvePower(path: str, *, update: int = 1000, on_progress: ProgressCb = None) -> Dict[str, Any]:
2     n, area = core.boardParser(path)
3     if update <= 0:
4         update = 1
5
6     N = n * n
7     iterations = 0
8     solution: Optional[Tuple[int, ...]] = None
9     start = perf_counter()
10
11     def maybe_progress(cols_preview: List[int]) -> None:
12         if on_progress is None:
13             return
14         if iterations % update == 0:
15             elapsed_ms = (perf_counter() - start) * 1000
16             on_progress(iterations, elapsed_ms, cols_preview, n, area)
17
18     def is_valid_placement(pos: List[Tuple[int, int]]) -> bool:
19         rows = [r for r, _ in pos]
20         if len(set(rows)) != n:
21             return False
22         cols = [c for _, c in pos]
23         if len(set(cols)) != n:
24             return False
25         used_colors = set()
26         for r, c in pos:
27             color = area[r][c]
28             if color in used_colors:
29                 return False
30             used_colors.add(color)
31         for i in range(n):
32             r1, c1 = pos[i]
33             for j in range(i + 1, n):
34                 r2, c2 = pos[j]
35                 if abs(r1 - r2) <= 1 and abs(c1 - c2) <= 1:
36                     return False
37         return True
38
39     def next_combination(comb: List[int], N: int, k: int) -> bool:
40         i = k - 1
41         while i >= 0 and comb[i] == N - k + i:
42             i -= 1
43         if i < 0:
44             return False # ud kombinasi terakhir
45
46         comb[i] += 1
47         for j in range(i + 1, k):
48             comb[j] = comb[j - 1] + 1
49         return True
50
51     if on_progress is not None:
52         on_progress(0, 0.0, [-1] * n, n, area)
53     comb: List[int] = list(range(n))
54
55     while True:
56         iterations += 1
57         pos = [(idx // n, idx % n) for idx in comb]
58
59         cols_preview = [-1] * n
60         for r, c in pos:
61             cols_preview[r] = c
62         maybe_progress(cols_preview.copy())
63
64         if is_valid_placement(pos):
65             cols_sol = [-1] * n
66             for r, c in pos:
67                 cols_sol[r] = c
68             solution = tuple(cols_sol)
69             break
70
71         if not next_combination(comb, N, n):
72             break # udh semua kombinasi
73
74     time_ms = (perf_counter() - start) * 1000
75     solved = boardFormat(area, solution) if solution is not None else None
76     return {
77         "mode": "pow",
78         "n": n,
79         "area": area,
80         "found": solution is not None,
81         "cols": list(solution) if solution is not None else None,
82         "iterations": iterations,
83         "solved": solved,
84         "time_ms": time_ms,
85     }

```

Fungsi ini mengimplementasikan brute force dengan cara mengenumerasi seluruh permutasi kolom cols sepanjang  $n$ . Setiap permutasi dianggap sebagai konfigurasi lengkap (satu queen per baris dan kolom), lalu divalidasi menggunakan `core.isValidMove(cols, n, area)` untuk

mengecek batasan daerah warna dan adjacency. Sama seperti mode pow, fungsi ini mencatat iterasi, waktu, dan mendukung pembaruan progres.

```
1 def solveFactorial(path: str, *, update: int = 1000, on_progress: Prog  
ressCb = None) -> Dict[str, Any]:  
2     n, area = core.boardParser(path)  
3  
4     if update <= 0:  
5         update = 1  
6  
7     iterations = 0  
8     solution: Optional[Tuple[int, ...]] = None  
9  
10    cols: List[int] = list(range(n))  
11    start = perf_counter()  
12  
13    def maybe_progress() -> None:  
14        if on_progress is None:  
15            return  
16        if iterations % update == 0:  
17            elapsed_ms = (perf_counter() - start) * 1000  
18            on_progress(iterations, elapsed_ms, cols.copy(), n, area)  
19  
20    def next_permutation(a: List[int]) -> bool:  
21        i = len(a) - 2  
22        while i >= 0 and a[i] >= a[i + 1]:  
23            i -= 1  
24        if i < 0:  
25            return False # udah all permutations  
26  
27        j = len(a) - 1  
28        while a[j] <= a[i]:  
29            j -= 1  
30  
31        a[i], a[j] = a[j], a[i]  
32  
33        l, r = i + 1, len(a) - 1  
34        while l < r:  
35            a[l], a[r] = a[r], a[l]  
36            l += 1  
37            r -= 1  
38  
39        return True  
40  
41    # preview empty board di awal  
42    if on_progress is not None:  
43        on_progress(0, 0.0, cols.copy(), n, area)  
44  
45    while True:  
46        iterations += 1  
47        maybe_progress()  
48  
49        if core.isValidMove(cols, n, area):  
50            solution = tuple(cols)  
51            break  
52  
53        if not next_permutation(cols):  
54            break # udah all n!  
55  
56    time_ms = (perf_counter() - start) * 1000  
57    solved = boardFormat(area, solution) if solution is not None else  
None  
58  
59    return {  
60        "mode": "fact",  
61        "n": n,  
62        "area": area,  
63        "found": solution is not None,  
64        "cols": list(solution) if solution is not None else None,  
65        "iterations": iterations,  
66        "solved": solved,  
67        "time_ms": time_ms,  
68    }
```

Fungsi ini bertindak sebagai dispatcher/pengatur mode. Jika mode poq maka memanggil solvePower, dan jika mode fact maka memanggil solveFactorial. Fungsi ini juga menormalisasi nilai update agar tidak nol/negatif.



```

1 def solve( path: str, *, mode: Mode = "pow", update: int = 1000, on_progress: ProgressCb = None) -> Dict[str, Any]:
2     if update <= 0:
3         update = 1
4
5     if mode == "pow":
6         return solvePower(path, update=update, on_progress=on_progress)
7     elif mode == "fact":
8         return solveFactorial(path, update=update, on_progress=on_progress)
9     else:
10        raise ValueError(f"Unknown mode: {mode}. Use 'pow' or 'fact'.")

```

Membaca file input, menghapus baris kosong, memastikan papan berbentuk persegi  $n \times n$ , mengubahnya menjadi matriks karakter, lalu memanggil validateBoard untuk memastikan format papan valid.

```

1 def boardParser(path: str | Path) -> Tuple[int, List[List[str]]]:
2     p = Path(path)
3     if not p.exists():
4         raise ValueError(f"File not found: {p}")
5
6     lines: List[str] = []
7     for ln in p.read_text(encoding="utf-8").splitlines():
8         ln = ln.strip()
9         if ln:
10            lines.append(ln)
11
12    if not lines:
13        raise ValueError(f"Empty file: {p}")
14
15    n = len(lines)
16    if any(len(row) != n for row in lines):
17        wrong = [(i, len(lines[i])) for i in range(n) if len(lines[i]) != n]
18        raise ValueError(f"Invalid board, grid must be square (NXN). Wrong rows: {wrong}")
19
20    area = [list(row) for row in lines]
21    validateBoard(n, area)
22    return n, area

```

Fungsi validasi aturan permainan untuk representasi cols (satu queen per baris).

```
1 def isValidMove(cols: Sequence[int], n: int, region: List[List[str]]) -> bool:
2     if len(cols) != n:
3         return False
4
5     # columns must be within range and unique
6     scols = set()
7     for c in cols:
8         if c < 0 or c >= n:
9             return False
10        if c in scols:
11            return False
12        scols.add(c)
13
14    # regions must be unique
15    sareas: Set[str] = set()
16    for r in range(n):
17        c = cols[r]
18        rid = region[r][c]
19        if rid in sareas:
20            return False
21        sareas.add(rid)
22
23    # adjacency restriction: no queens touching (8-neighborhood)
24    for r1 in range(n):
25        c1 = cols[r1]
26        for r2 in range(r1 + 1, n):
27            c2 = cols[r2]
28            if abs(r1 - r2) <= 1 and abs(c1 - c2) <= 1:
29                return False
30
31    return True
```

## BAB 3 TESTING

### 3.1. Test Case 1

Input : test4.txt

AAAAAAA

BBBBBBB

BCBBDDD

CCBBDEE

FCBBDDD

FCBBDGD

CCCBDDD

Output :

a. Mode Pow (Default)

LinkedIn Queens Solver

Save

Upload

			♔			
♔						
				♔		
						♔
♔						
					♔	
		♔				

Solve

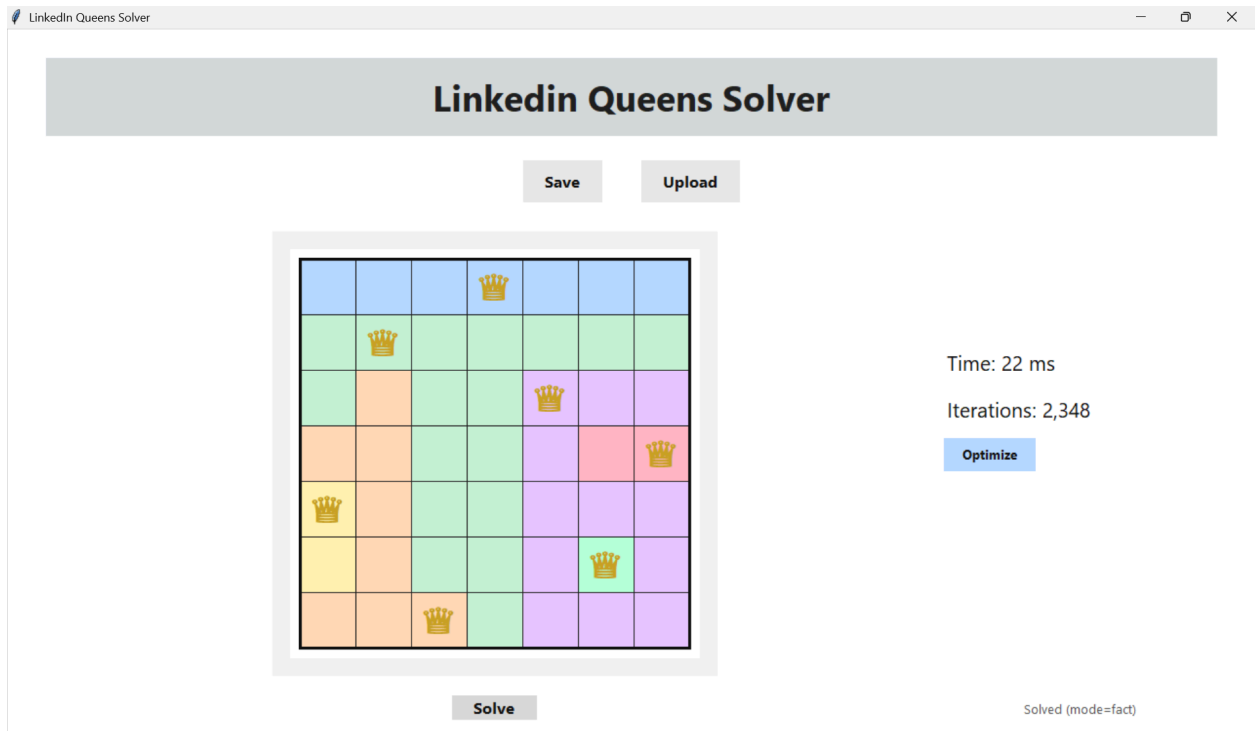
Time: 86821 ms

Iterations: 36,532,921

Optimize

Solved (mode=pow)

b. Mode Factorial (Optimized)



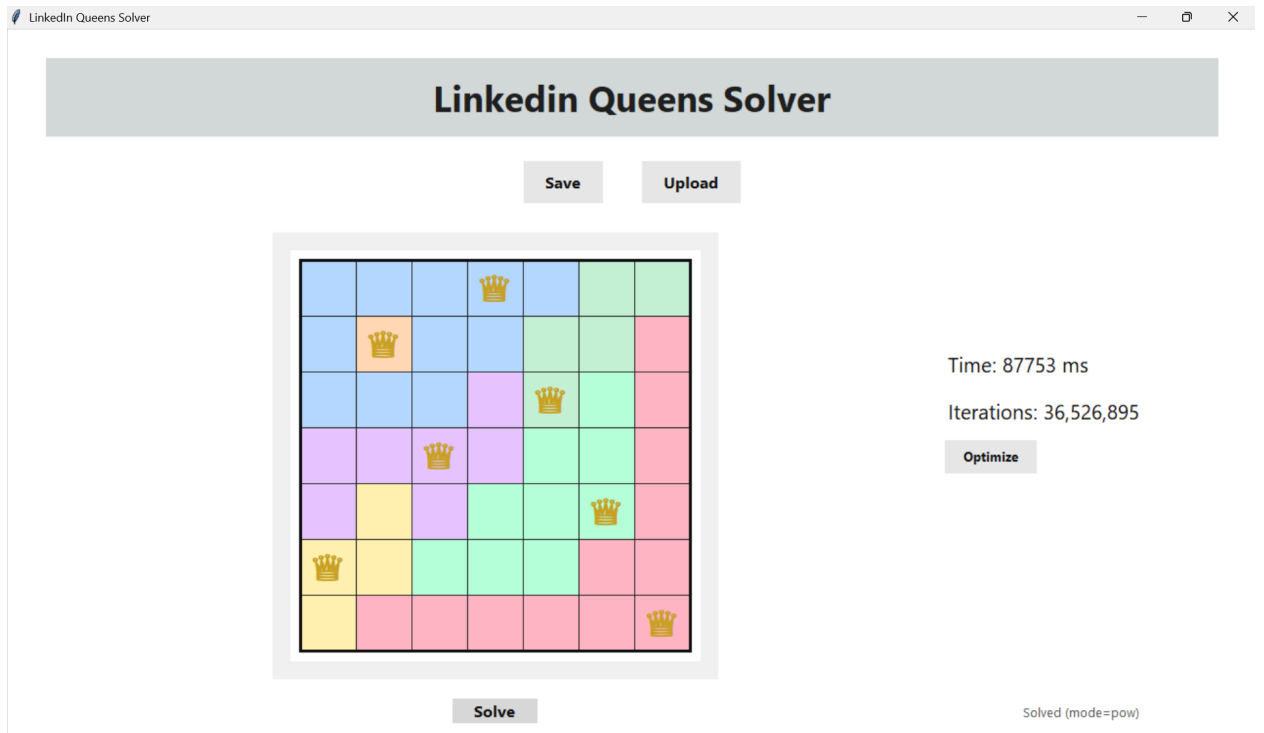
### 3.2. Test Case 2

Input : test5.txt

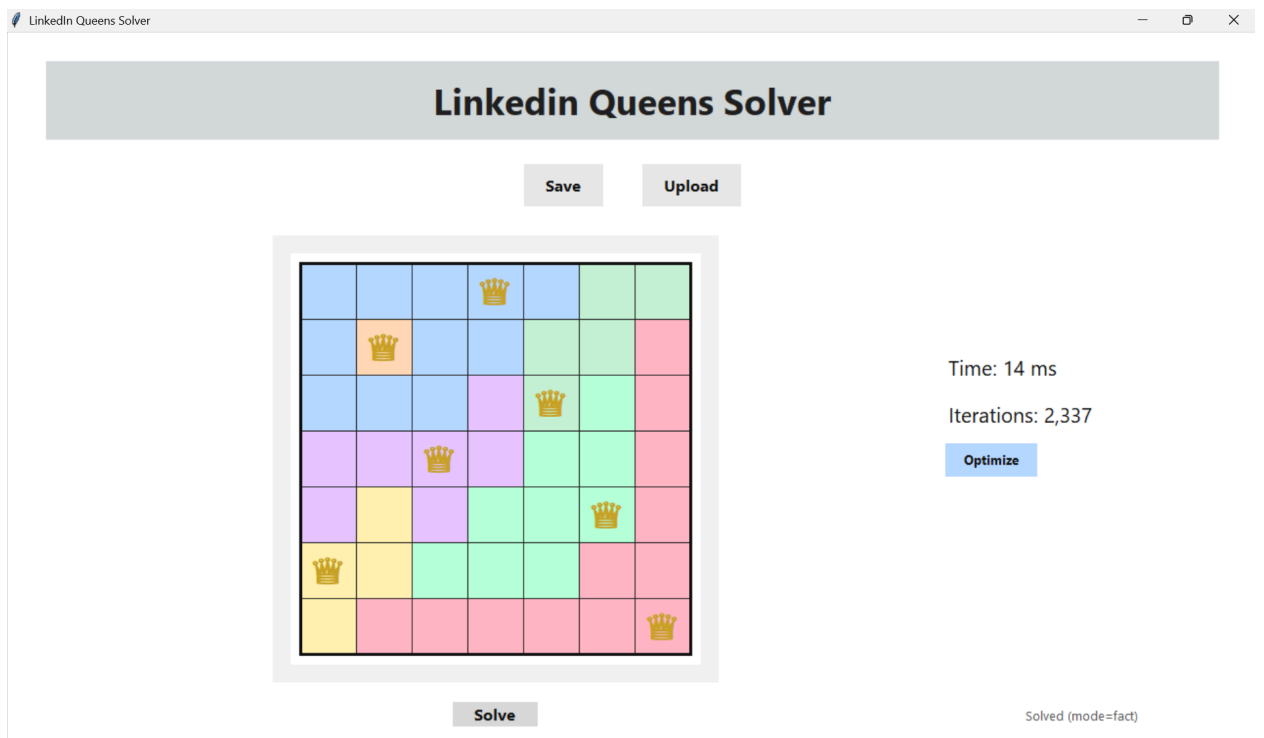
AAAAAAA  
AAAAABB  
ACAABBE  
AAADBGE  
DDDDGGE  
DFDGGGE  
FFGGGEE  
FEEEEEE

Output :

- a. Mode Pow (Default)



b. Mode Factorial (Optimized)



### 3.3. Test Case 3

Input : test6.txt

AAAAAAA  
AAABBBB  
ACCCBBB  
CCCCBBB  
CCCCCBD  
CCECCFD  
GGGCCFD  
GGGGGFD

Output :

a. Mode Pow (Default)

The screenshot displays the 'LinkedIn Queens Solver' application interface. At the top, the title 'LinkedIn Queens Solver' is centered. Below the title are 'Save' and 'Upload' buttons. The main area features an 8x8 chessboard with a solved solution. The queens are positioned at (1,4), (2,1), (3,5), (4,8), (5,3), (6,6), (7,2), and (8,7) using 0-indexing from the top-left. The board is color-coded: blue for columns 1-3, green for columns 4-6, orange for columns 7-8, and purple for columns 9-10. A 'Solve' button is located below the board. To the right of the board, the performance metrics are displayed: 'Time: 105785 ms' and 'Iterations: 42,906,883'. An 'Optimize' button is positioned below these metrics. At the bottom right, the status 'Solved (mode=pow)' is shown. The application is running on a Windows operating system, as indicated by the taskbar at the bottom.

LinkedIn Queens Solver

Save Upload

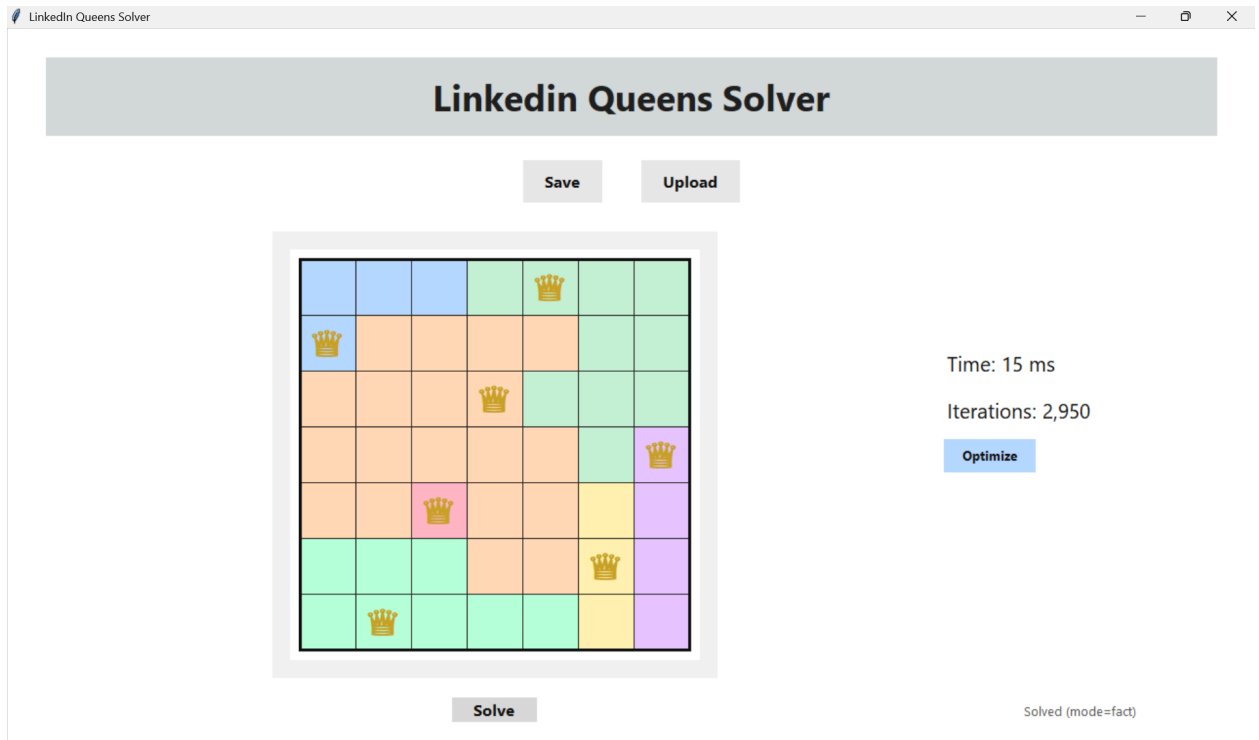
Time: 105785 ms  
Iterations: 42,906,883

Optimize

Solve

Solved (mode=pow)

b. Mode Factorial (Optimized)



### 3.4. Test Case 4

Input : test7.txt

AAAAAAAAA

ABCCHHHH

ABCCGGGH

ABCIGJJH

ABIIMJJH

ABMMMJJH

ABMMHHHH

ABBBBBBB

Output :

- a. Mode Pow (Default)





BBBBBBB  
CCCCCCC  
CDDCECC  
CCDCECF  
CCDCECC  
CGGHHCC  
GGGHHCC

Output :  
c. Mode Pow (Default)

LinkedIn Queens Solver

LinkedIn Queens Solver

Save

Upload

♔						
					♔	
		♔				
						♔
				♔		
	♔					
			♔			

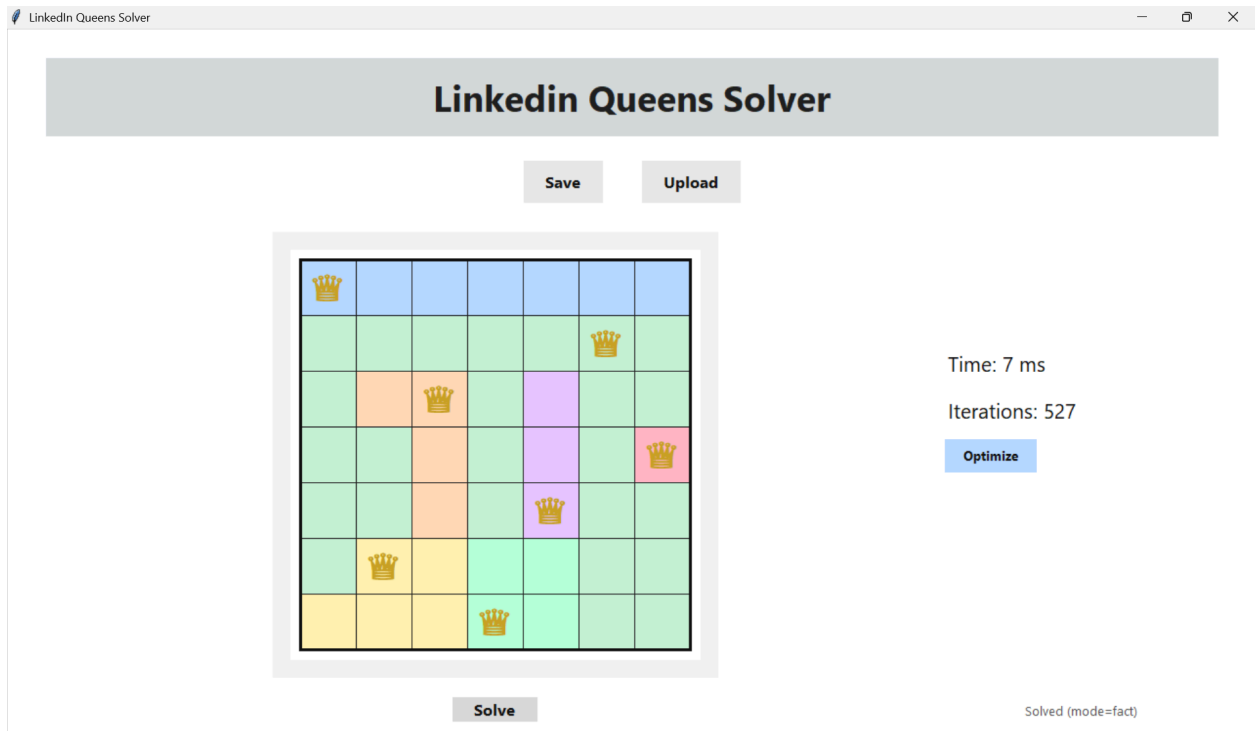
Solve

Time: 17545 ms  
Iterations: 10,115,730  

Optimize

Solved (mode=pow)

d. Mode Factorial (Optimized)



### 3.6. Test Case 6

Input : test9.txt

AAAAAA

BBBCAC

BDDCAC

BBBCCC

EEEEEE

EEEEEF

Output :

- a. Mode Pow (Default)

LinkedIn Queens Solver

# LinkedIn Queens Solver

Save

Upload

			♔		
♔					
		♔			
				♔	
	♔				
					♔

Time: 1354 ms

Iterations: 916,529

Optimize

Solve

Solved (mode=pow)

## b. Mode Factorial (Optimized)

The screenshot shows a web application titled "LinkedIn Queens Solver". The interface includes a header bar with the title and window controls. Below the header, there is a large grey banner with the title "LinkedIn Queens Solver" in bold black text. Underneath the banner, there are two buttons: "Save" and "Upload". The main area features a 6x6 grid representing a chessboard. The grid is divided into six colored regions: blue (top-left 2x2), green (top-middle 2x2), orange (top-right 2x2), pink (bottom-left 2x2), pink (bottom-middle 2x2), and yellow (bottom-right 1x1). Six golden queen pieces are placed on the board at the following coordinates (row, column): (1, 4), (2, 1), (3, 3), (4, 5), (5, 2), and (6, 6). To the right of the grid, the text "Time: 8 ms" and "Iterations: 369" is displayed. Below this text is a blue button labeled "Optimize". At the bottom of the grid, there is a "Solve" button. In the bottom right corner, the text "Solved (mode=fact)" is shown.

LinkedIn Queens Solver

# LinkedIn Queens Solver

Save Upload

			♔		
♔					
		♔			
				♔	
	♔				
					♔

Time: 8 ms  
Iterations: 369

Optimize

Solve

Solved (mode=fact)

## LAMPIRAN

Link Github: [https://github.com/revanatania/Tucil1\\_13524098](https://github.com/revanatania/Tucil1_13524098)

No	Poin	Ya	Tidak
1	Program berhasil dikompilasi tanpa kesalahan	✓	
2	Program berhasil dijalankan	✓	
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.



Reva Natania Sitohang

