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

1.1. Algorima 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 x 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:

   $$\left(\frac{n^2}{n}\right)$$

   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 $n!$

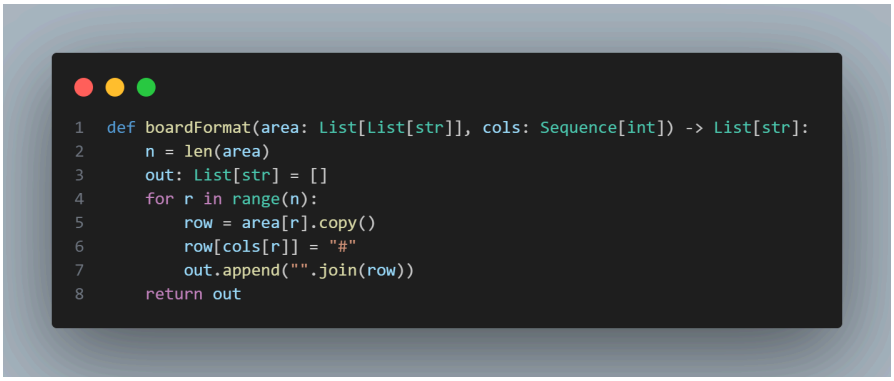
Dalam mode ini, algoritma menggunakan pengamatan sederhana dari aturan permainan, yaitu setiap kolom hanya boleh memiliki tepat satu queen. Karena representasi kolom sudah menjamin satu queen per baris, maka penempatan yang valid haruslah merupakan suatu permutasi dari himpunan kolom. Maka dari itu, 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 redudansi 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.



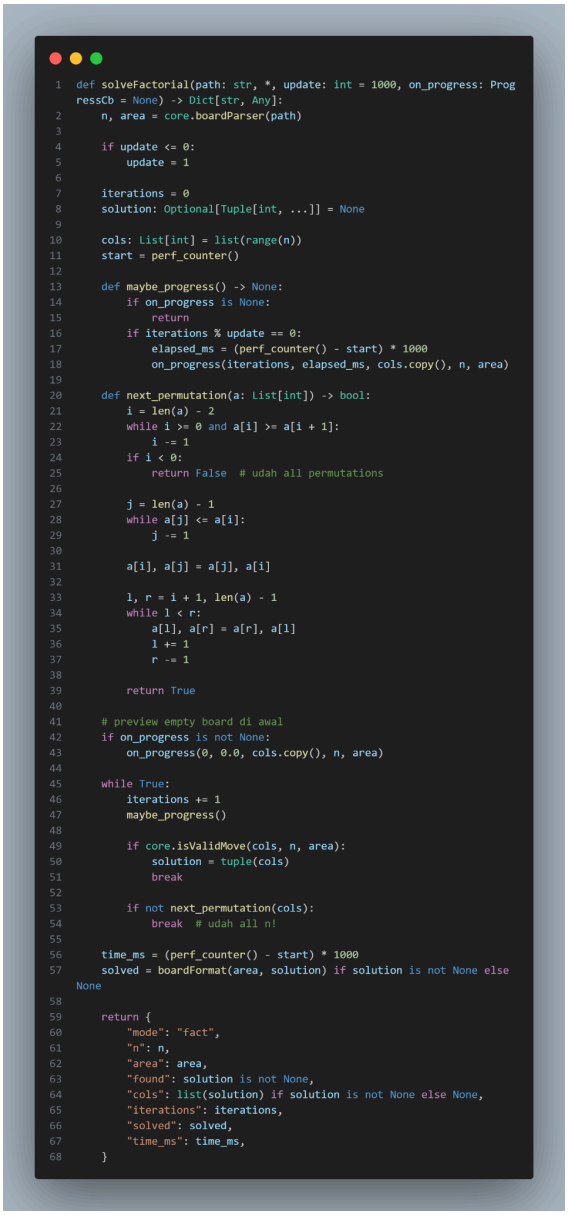
```python
def boardFormat(area: List[List[str]], cols: Sequence[int]) -> List[str]:
    n = len(area)
    out: List[str] = []
    for r in range(n):
        row = area[r].copy()
        row[cols[r]] = "#"
        out.append("".join(row))
    return out
```

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

```python
def solvePower(path: str, *, update: int = 1000, on_progress: ProgressCb = None) -> Dict[str, Any]:
    n, area = core.boardParser(path)
    if update <= 0:
        update = 1

    N = n * n
    iterations = 0
    solution: Optional[Tuple[int, ...]] = None
    start = perf_counter()

    def maybe_progress(cols_preview: List[int]) -> None:
        if on_progress is None:
            return
        if iterations % update == 0:
            elapsed_ms = (perf_counter() - start) * 1000
            on_progress(iterations, elapsed_ms, cols_preview, n, area)

    def is_valid_placement(pos: List[Tuple[int, int]]) -> bool:
        rows = [r for r, _ in pos]
        if len(set(rows)) != n:
            return False
        cols = [c for _, c in pos]
        if len(set(cols)) != n:
            return False
        used_colors = set()
        for r, c in pos:
            color = area[r][c]
            if color in used_colors:
                return False
            used_colors.add(color)
        for i in range(n):
            r1, c1 = pos[i]
            for j in range(i + 1, n):
                r2, c2 = pos[j]
                if abs(r1 - r2) <= 1 and abs(c1 - c2) <= 1:
                    return False
        return True

    def next_combination(comb: List[int], N: int, k: int) -> bool:
        i = k - 1
        while i >= 0 and comb[i] == N - k + i:
            i -= 1
        if i < 0:
            return False  # ud kombinasi terakhir

        comb[i] += 1
        for j in range(i + 1, k):
            comb[j] = comb[j - 1] + 1
        return True

    if on_progress is not None:
        on_progress(0, 0.0, [-1] * n, n, area)
    comb: List[int] = list(range(n))

    while True:
        iterations += 1
        pos = [(idx // n, idx % n) for idx in comb]

        cols_preview = [-1] * n
        for r, c in pos:
            cols_preview[r] = c
        maybe_progress(cols_preview.copy())

        if is_valid_placement(pos):
            cols_sol = [-1] * n
            for r, c in pos:
                cols_sol[r] = c
            solution = tuple(cols_sol)
            break

        if not next_combination(comb, N, n):
            break  # udh semua kombinasi

    time_ms = (perf_counter() - start) * 1000
    solved = boardFormat(area, solution) if solution is not None else None
    return {
        "mode": "pow",
        "n": n,
        "area": area,
        "found": solution is not None,
        "cols": list(solution) if solution is not None else None,
        "iterations": iterations,
        "solved": solved,
        "time_ms": time_ms,
    }
```

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.



```python
def solveFactorial(path: str, *, update: int = 1000, on_progress: ProgressCb = None) -> Dict[str, Any]:
    n, area = core.boardParser(path)

    if update <= 0:
        update = 1

    iterations = 0
    solution: Optional[Tuple[int, ...]] = None

    cols: List[int] = list(range(n))
    start = perf_counter()

    def maybe_progress() -> None:
        if on_progress is None:
            return
        if iterations % update == 0:
            elapsed_ms = (perf_counter() - start) * 1000
            on_progress(iterations, elapsed_ms, cols.copy(), n, area)

    def next_permutation(a: List[int]) -> bool:
        i = len(a) - 2
        while i >= 0 and a[i] >= a[i + 1]:
            i -= 1
        if i < 0:
            return False  # udah all permutations

        j = len(a) - 1
        while a[j] <= a[i]:
            j -= 1

        a[i], a[j] = a[j], a[i]

        l, r = i + 1, len(a) - 1
        while l < r:
            a[l], a[r] = a[r], a[l]
            l += 1
            r -= 1

        return True

    # preview empty board di awal
    if on_progress is not None:
        on_progress(0, 0.0, cols.copy(), n, area)

    while True:
        iterations += 1
        maybe_progress()

        if core.isValidMove(cols, n, area):
            solution = tuple(cols)
            break

        if not next_permutation(cols):
            break  # udah all n!

    time_ms = (perf_counter() - start) * 1000
    solved = boardFormat(area, solution) if solution is not None else None

    return {
        "mode": "fact",
        "n": n,
        "area": area,
        "found": solution is not None,
        "cols": list(solution) if solution is not None else None,
        "iterations": iterations,
        "solved": solved,
        "time_ms": time_ms,
    }
```

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.

```python
def solve( path: str, *, mode: Mode = "pow", update: int = 1000, on_progress: ProgressCb = None) -> Dict[str, Any]:
    if update <= 0:
        update = 1

    if mode == "pow":
        return solvePower(path, update=update, on_progress=on_progress)
    elif mode == "fact":
        return solveFactorial(path, update=update, on_progress=on_progress)
    else:
        raise ValueError(f"Unknown mode: {mode}. Use 'pow' or 'fact'.")
```

Membaca file input, menghapus baris kosong, memastikan papan berbentuk persegi n×n, mengubahnya menjadi matriks karakter, lalu memanggil

validateBoard untuk memastikan format papan valid.

```python
def boardParser(path: str | Path) -> Tuple[int, List[List[str]]]:
    p = Path(path)
    if not p.exists():
        raise ValueError(f"File not found: {p}")

    lines: List[str] = []
    for ln in p.read_text(encoding="utf-8").splitlines():
        ln = ln.strip()
        if ln:
            lines.append(ln)

    if not lines:
        raise ValueError(f"Empty file: {p}")

    n = len(lines)
    if any(len(row) != n for row in lines):
        wrong = [(i, len(lines[i])) for i in range(n) if len(lines[i]) != n]
        raise ValueError(f"Invalid board, grid must be square (NXN). Wrong rows: {wrong}")

    area = [list(row) for row in lines]
    validateBoard(n, area)
    return n, area
```

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

```python
def isValidMove(cols: Sequence[int], n: int, region: List[List[str]]) -> bool:
    if len(cols) != n:
            return False

    # columns must be within range and unique
    scols = set()
    for c in cols:
        if c < 0 or c >= n:
            return False
        if c in scols:
            return False
        scols.add(c)

    # regions must be unique
    sareas: Set[str] = set()
    for r in range(n):
        c = cols[r]
        rid = region[r][c]
        if rid in sareas:
            return False
        sareas.add(rid)

    # adjacency restriction: no queens touching (8-neighborhood)
    for r1 in range(n):
        c1 = cols[r1]
        for r2 in range(r1 + 1, n):
            c2 = cols[r2]
            if abs(r1 - r2) <= 1 and abs(c1 - c2) <= 1:
                return False

    return True
```

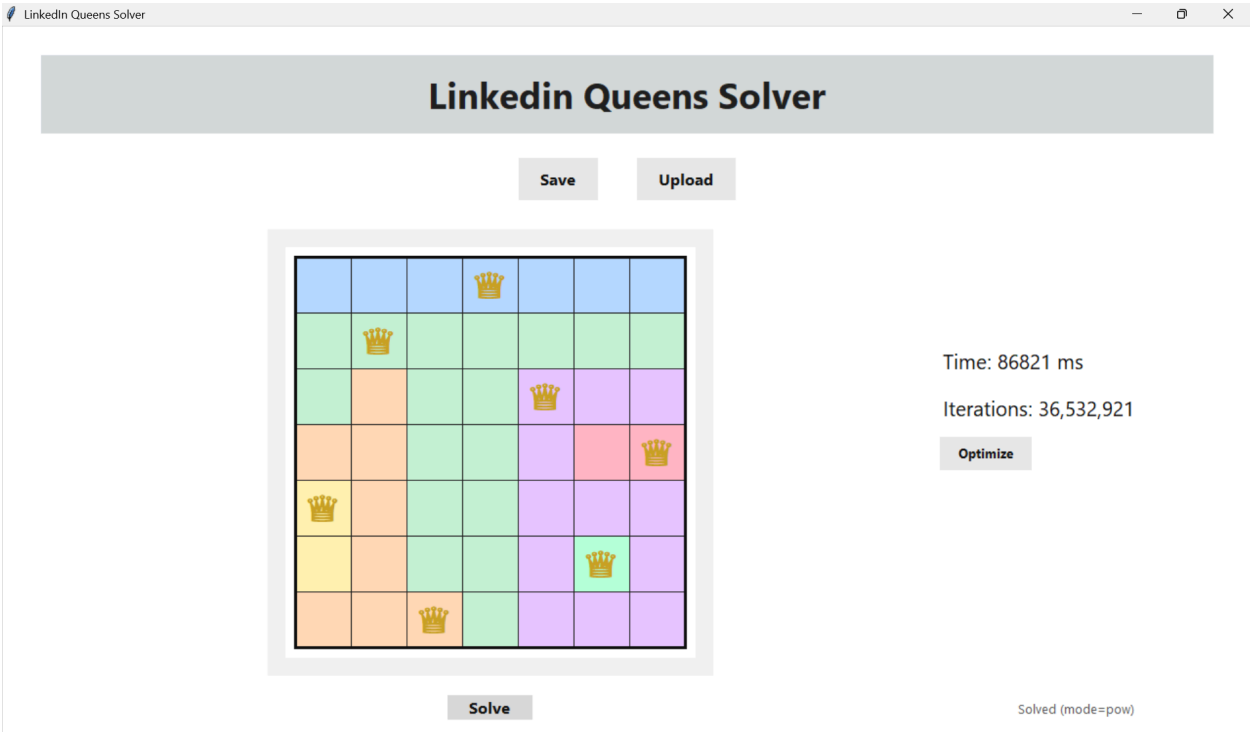
BAB 3
TESTING

3.1. Test Case 1

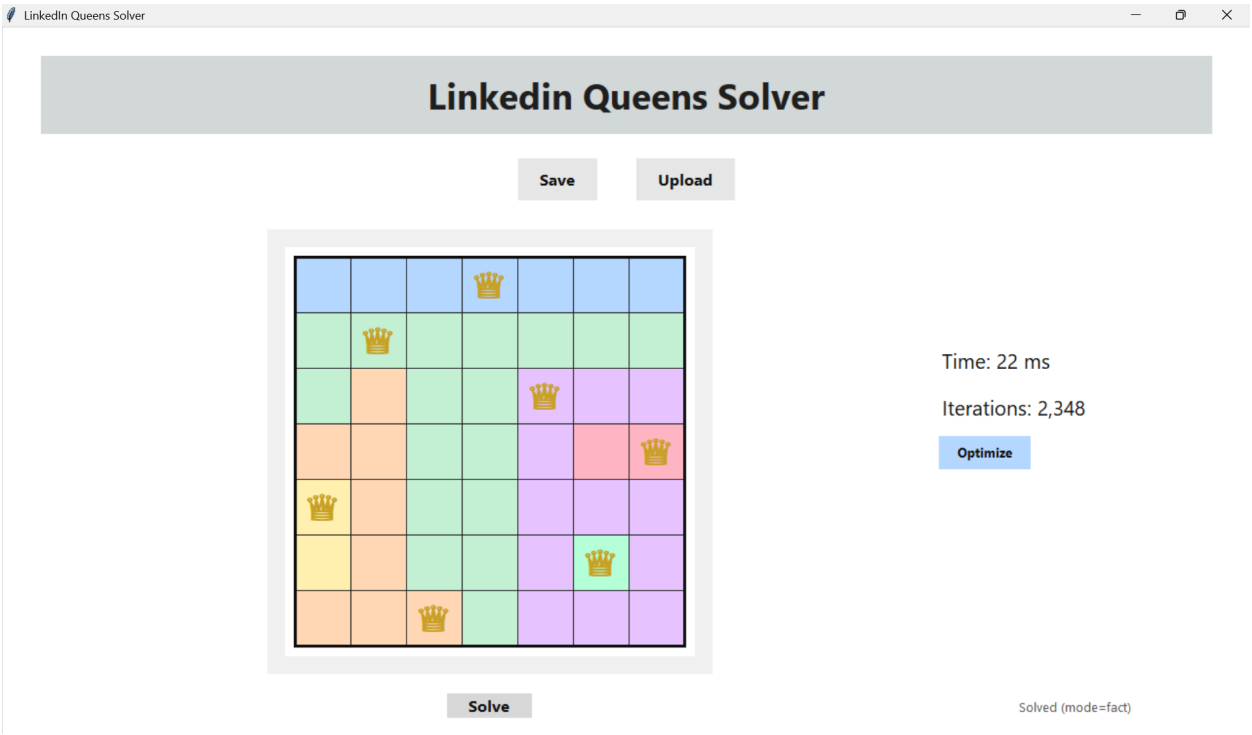
Input : test4.txt
AAAAAAA
BBBBBBB
BCBBDDD
CCBBDEE
FCBBDDD
FCBBDGD
CCCBDDD

Output :

a. Mode Pow (Default)



b. Mode Factorial (Optimized)

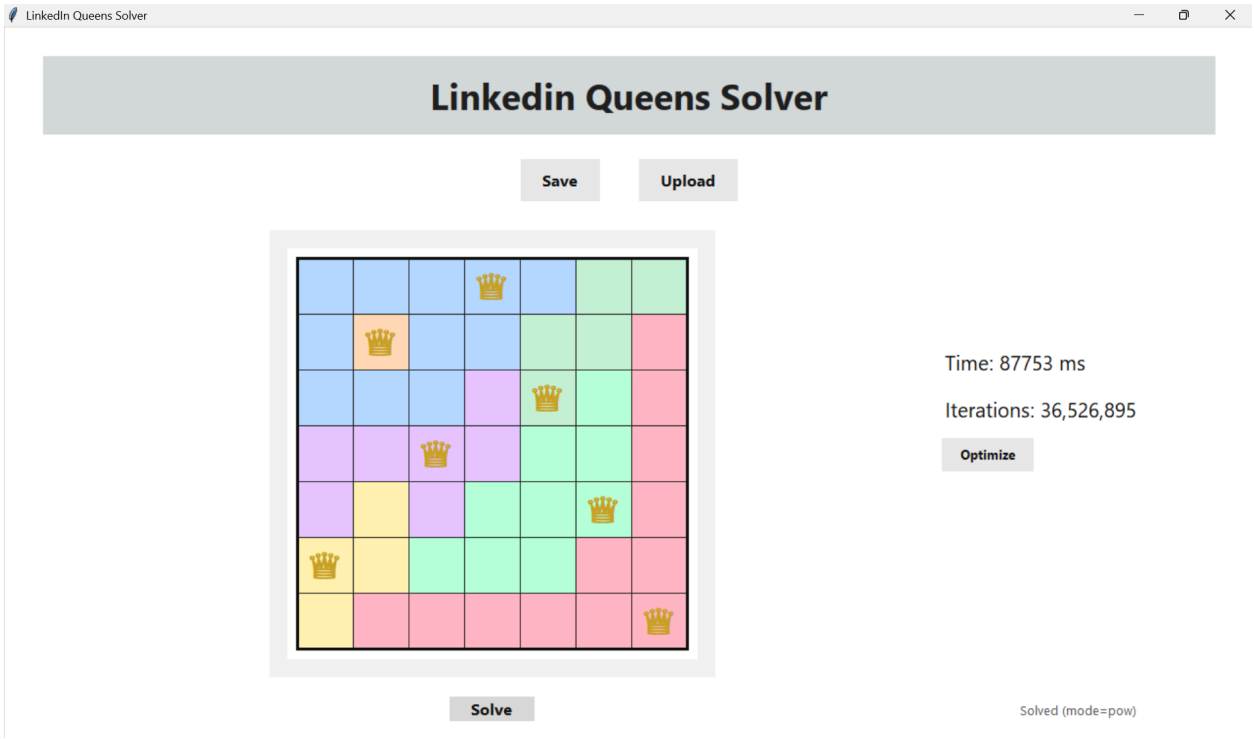


3.2. Test Case 2

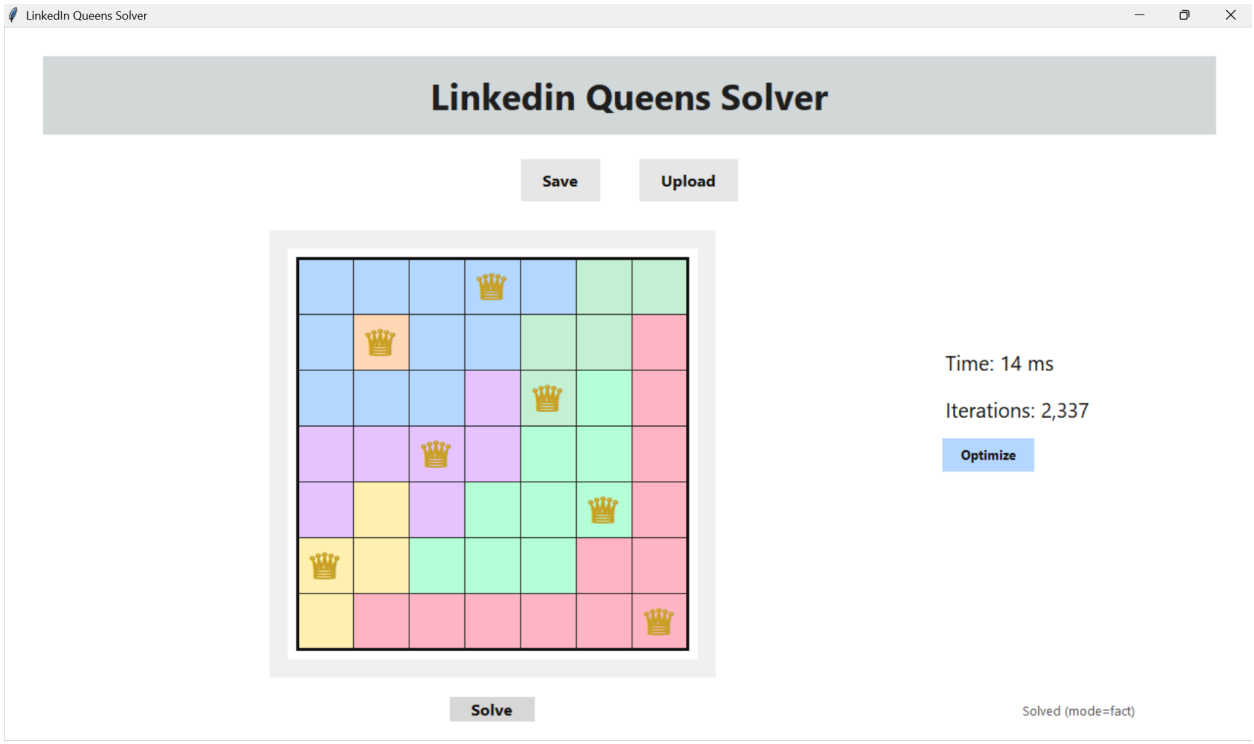
Input : test5.txt
AAAAAAA
AAAAABB
ACAABBE
AAADBGE
DDDDGGE
DFDGGGE
FFGGGEE
FEEEEEE

Output :

a. Mode Pow (Default)



b. Mode Factorial (Optimized)



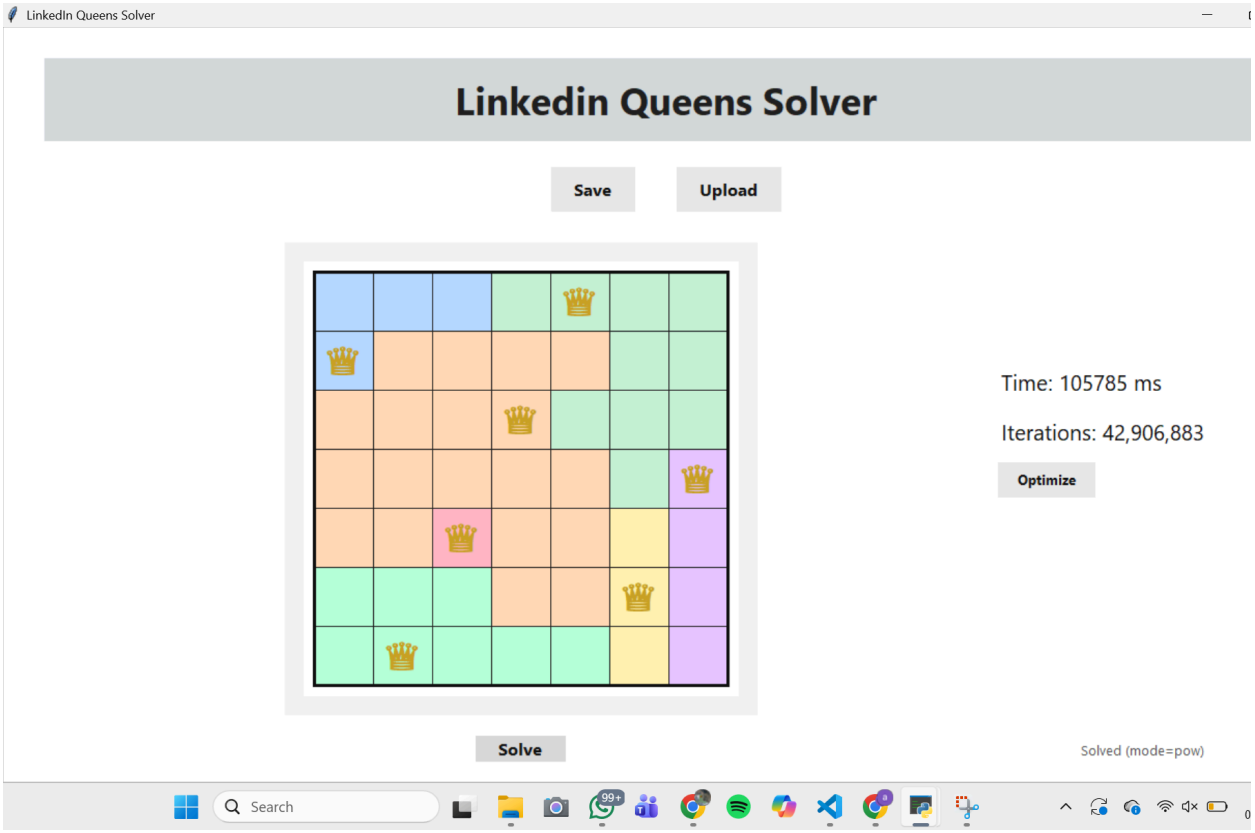
3.3. Test Case 3

Input : test6.txt

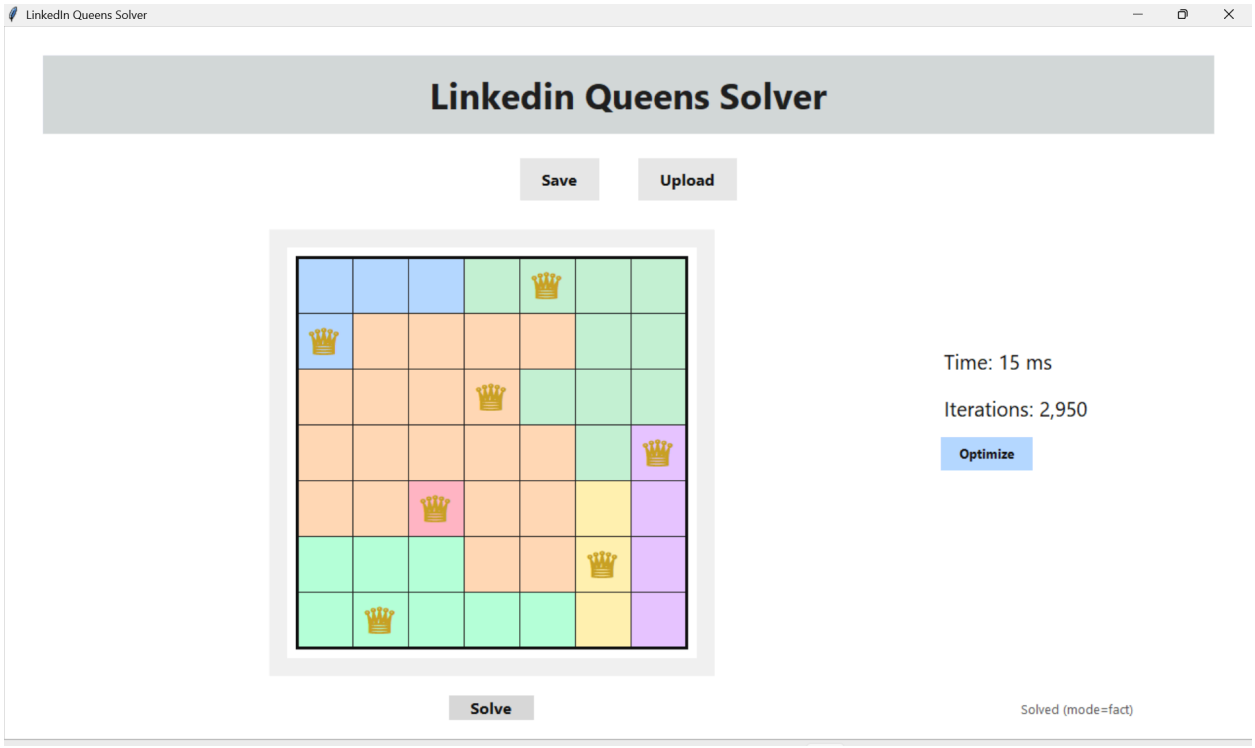
AAAAAAA
AAABBBB
ACCCCBB
CCCCBBB
CCCCCBD
CCECCFD
GGGCCFD
GGGGGFD

Output :

a. Mode Pow (Default)



b. Mode Factorial (Optimized)

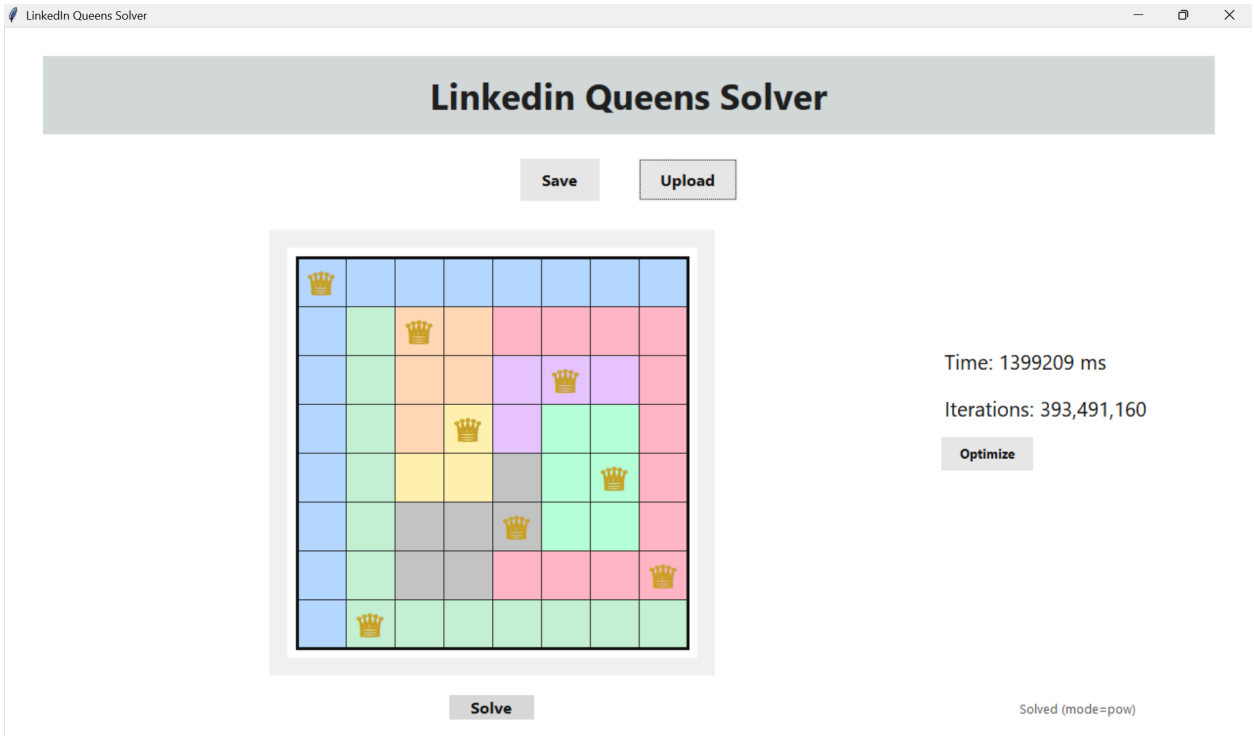


3.4. Test Case 4

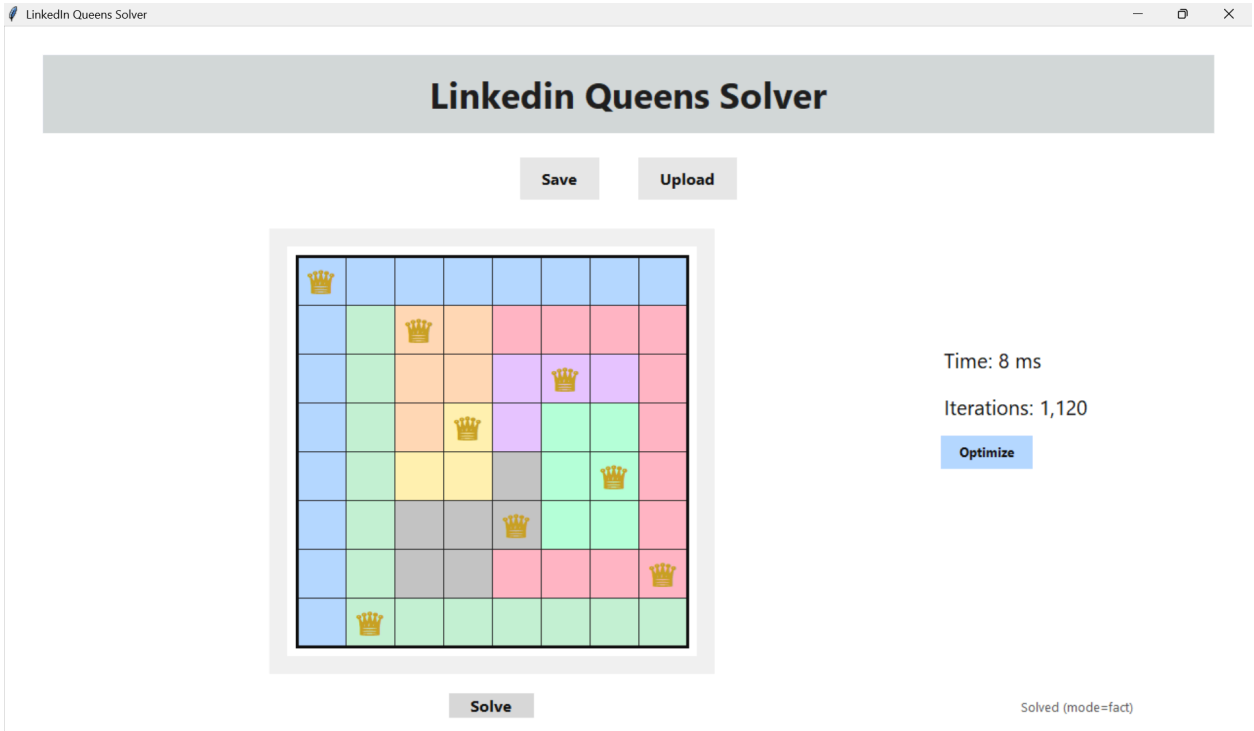
Input : test7.txt
AAAAAAAA
ABCCHHHH
ABCCGGGH
ABCIGJJH
ABIIMJJH
ABMMMJJH
ABMMHHHH
ABBBBBBB

Output :

a. Mode Pow (Default)



b. Mode Factorial (Optimized)



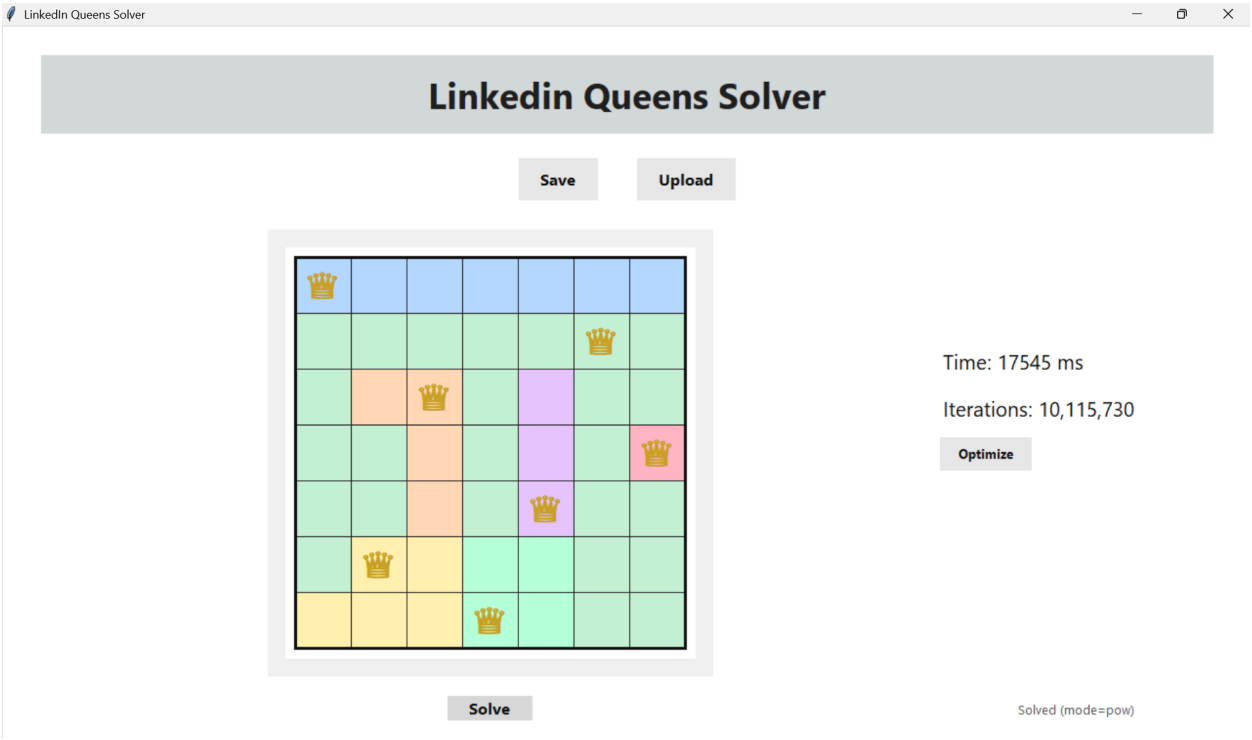
3.5. Test Case 5

Input : test8.txt

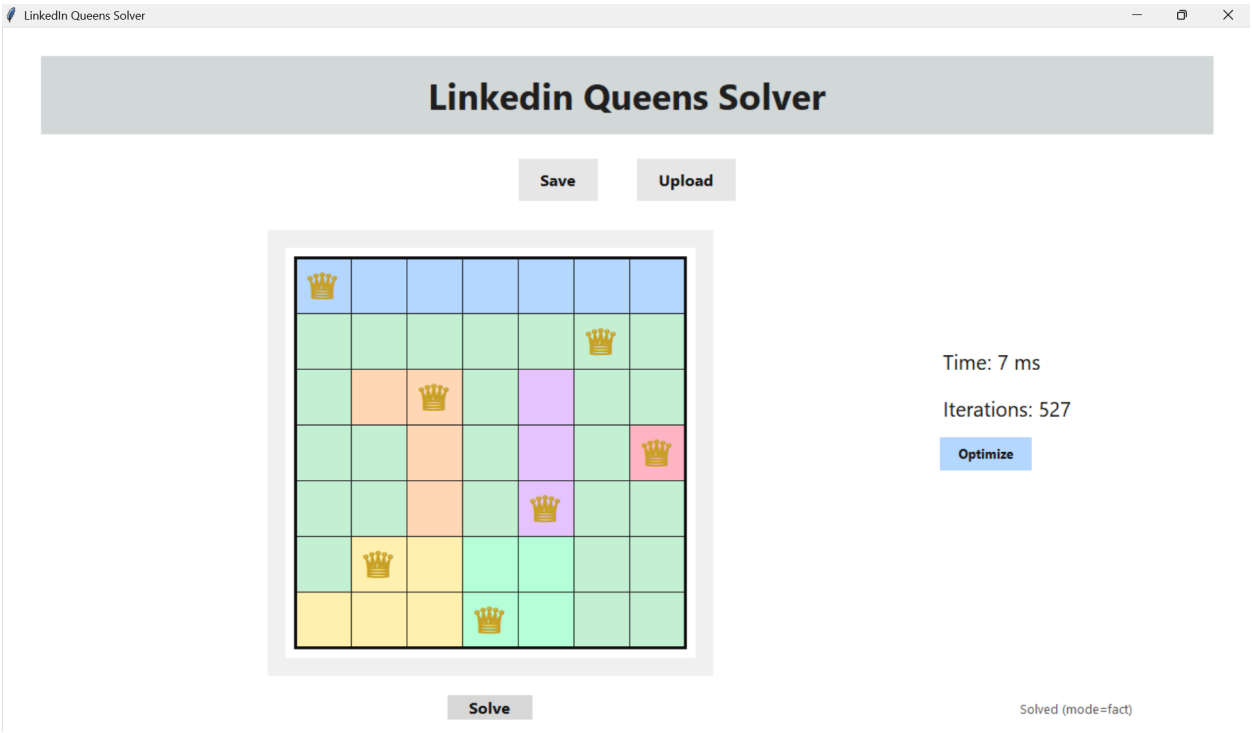
BBBBBBB
CCCCCCC
CDDCECC
CCDCECF
CCDCECC
CGGHHCC
GGGHHCC

Output :

c. Mode Pow (Default)



d. Mode Factorial (Optimized)

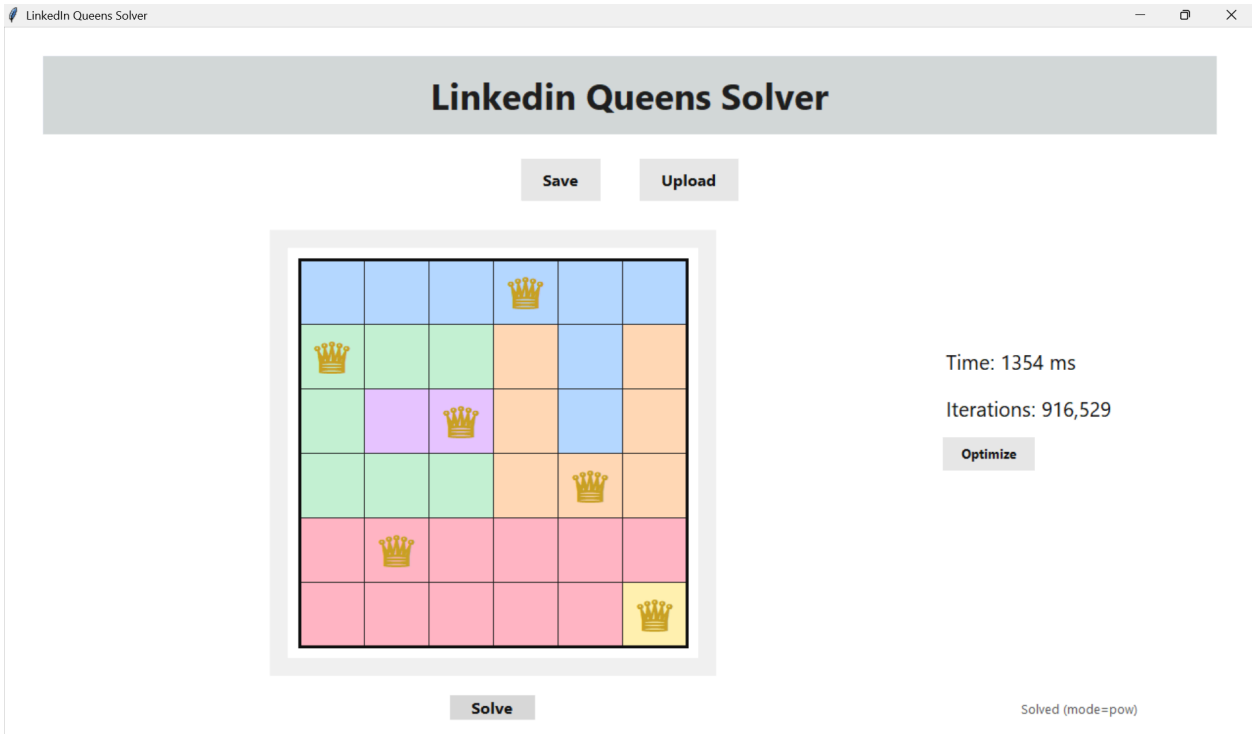


3.6. Test Case 6

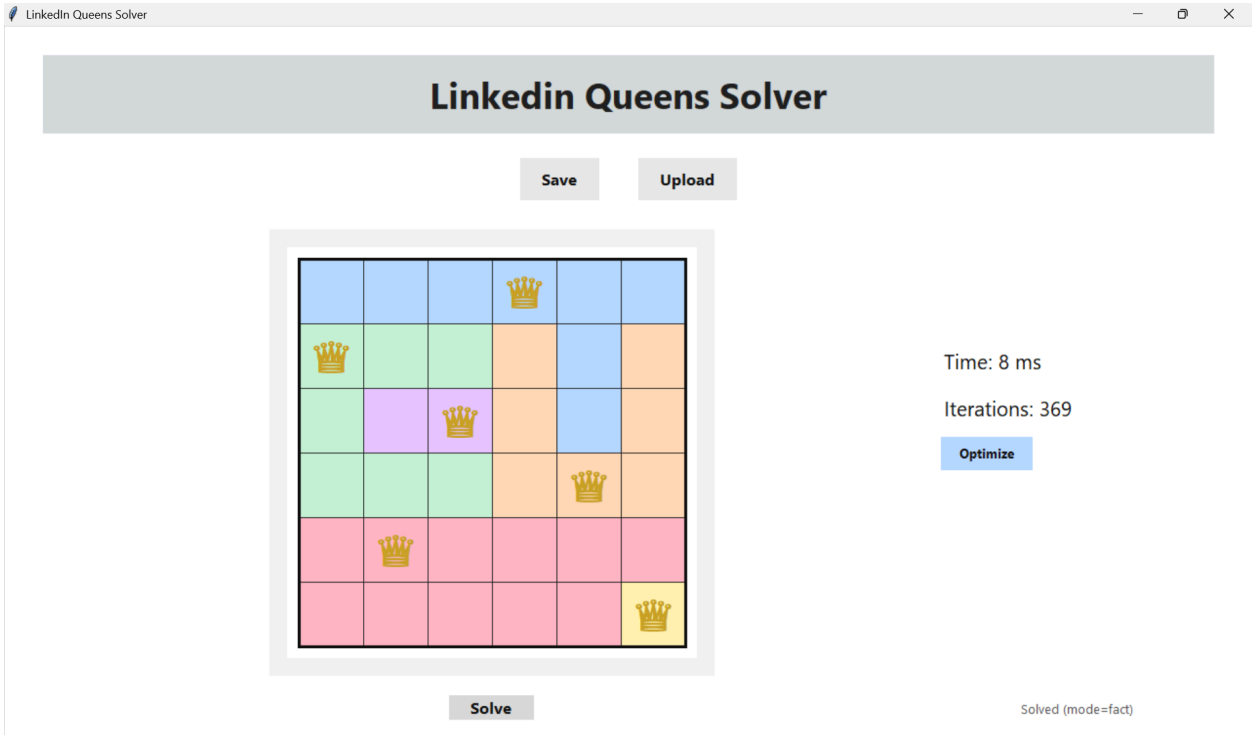
Input : test9.txt
AAAAAA
BBBCAC
BDDCAC
BBBCCC
EEEEEE
EEEEEF

Output :

a. Mode Pow (Default)



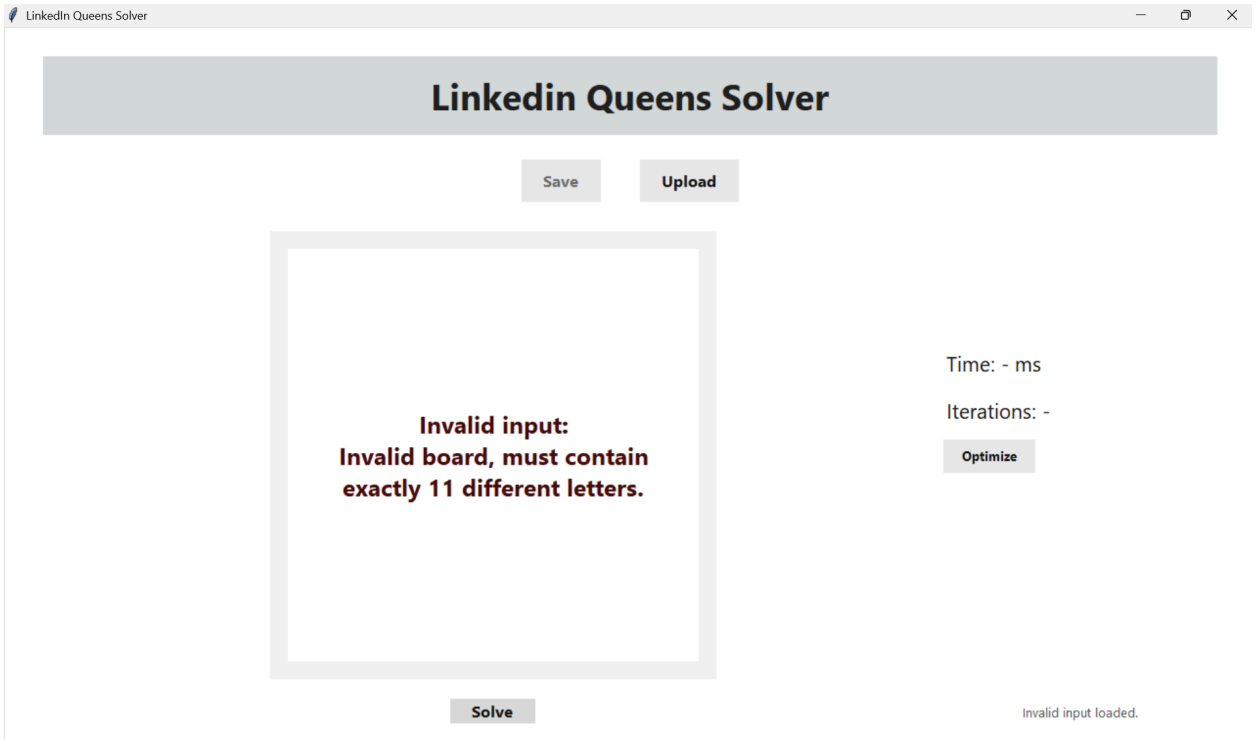
b. Mode Factorial (Optimized)



3.7. Test Case 7

Input : test3.txt
AABBBCCCCCC
ABBBBBBBBBC
ABBBBBCCCCC
DDDBEEECCGC
DDDBFEECCCC
DDDBFFECCGC
DDDBBBBBBBB
DDDDIIIIIII
HHIIIIJJJJJ
HHIIIIIJIJJ
HHHIIIIIIIJ

Output :

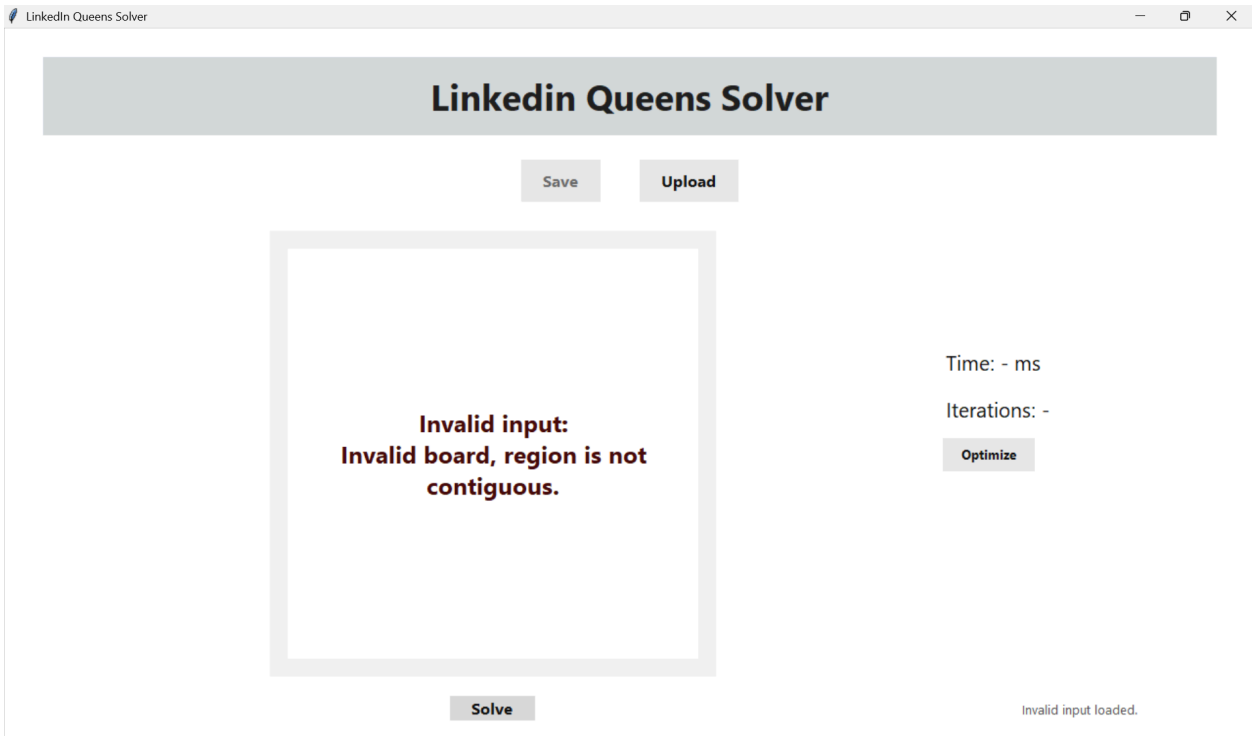


3.8. Test Case 8

Input : test2.txt
BBBAAACCC

BBAADAACC
BAADDDAAC
AAEEEDFAA
AEEEDDFFA
AAEEAEFAA
GAAEEEAAI
GGAAEAAII
HHHAAAIII

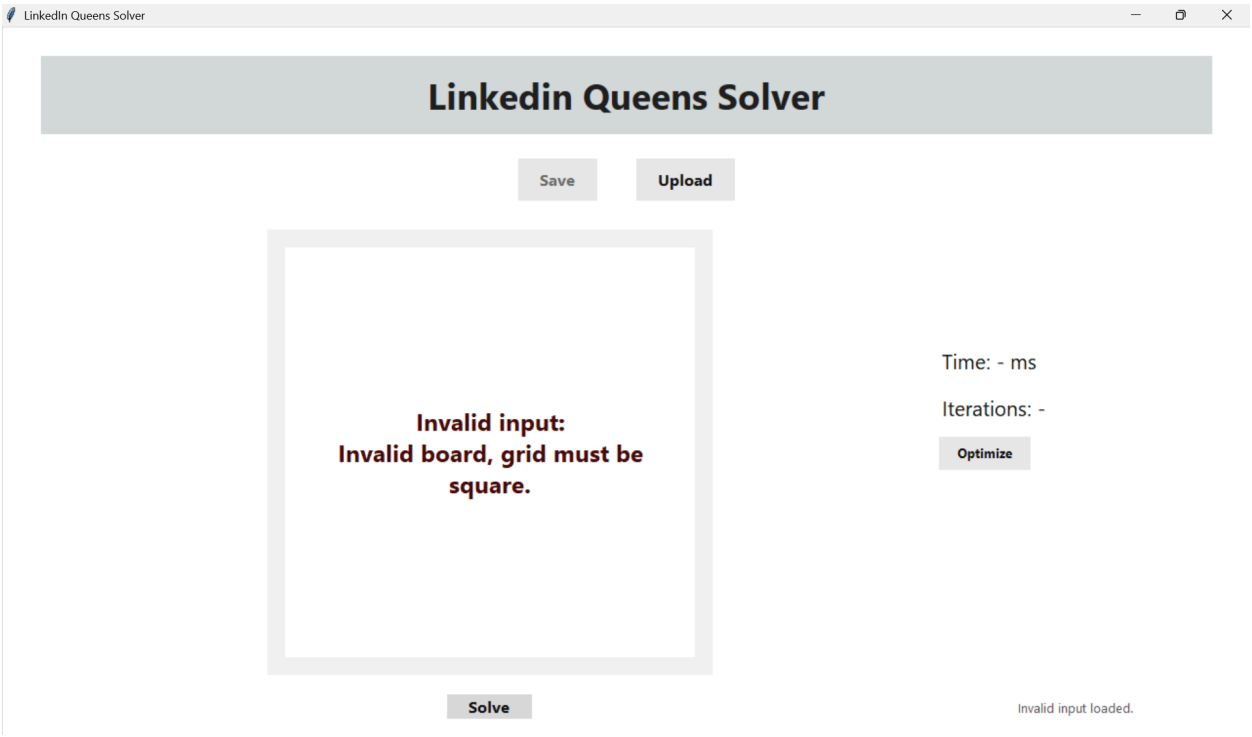
Output:



3.9. Test Case 9

Input : test1.txt
AAAAAA
BBBCAC
BDDCAC
BBBCCC
EEEEEE
EEEEEF

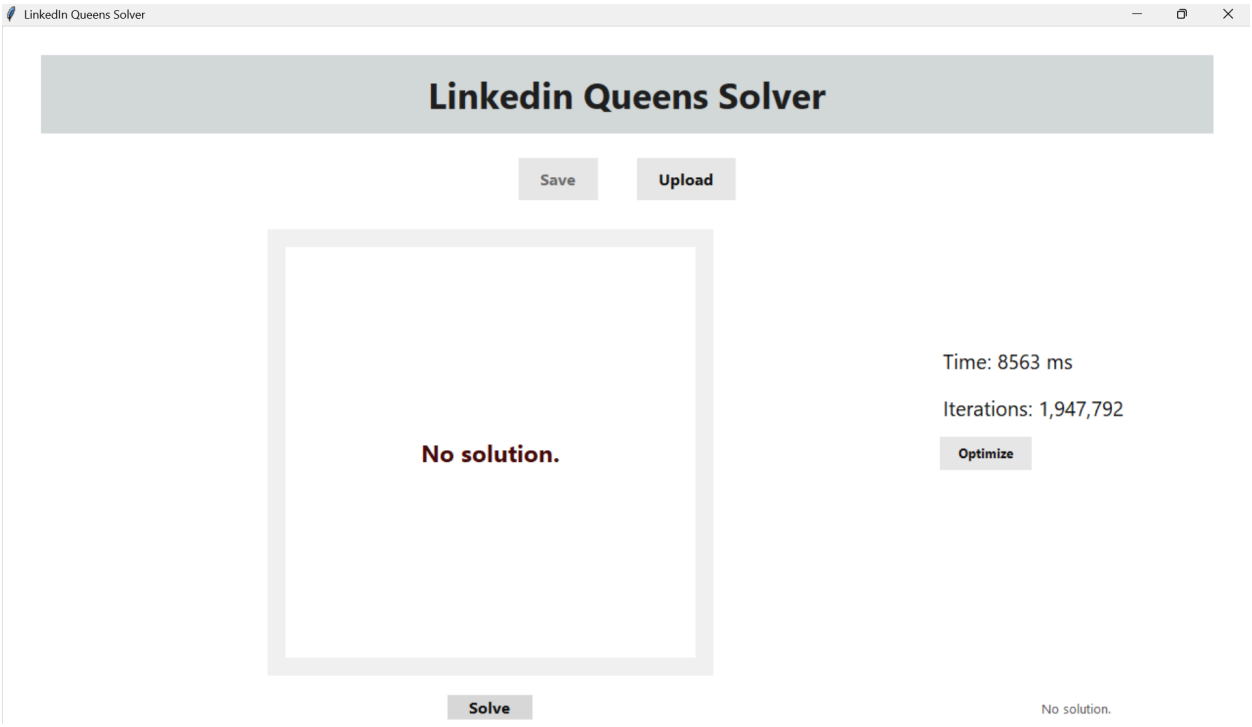
Output:



3.7. Test Case 10

Input : test11.txt
AAAAAA
ABBBBA
ACBBBD
ACCBBD
AFCCBD
AFFEEE

Output:


LinkedIn Queens Solver
Linkedin Queens Solver
Save
Upload
No solution.
Time: 8563 ms
Iterations: 1,947,792
Optimize
Solve
No solution.

LAMPIRAN

Link Github: 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