Puzzle Solver

By Applying Problem Solving Techniques

Abdullah Al Raqibul Islam

University of North Carolina at Charlotte | Id# 801151189

Project Goal

The goal of this project is to build a generic interactive pluggable application for solving puzzles (i.e. 8-puzzle, 15-puzzle, n-queen, Sudoku, etc.) using different problem-solving techniques (i.e. informed search, uninformed search, hill-climbing search etc.). Here by word "pluggable" we mean in solving puzzles user can independently decide the search strategy along with the custom heuristic functions. If needed, user can extend this project to device their own solution with a very low effort. Besides, this application is designed in a way that, it will be an easy-going platform for benchmarking any puzzles w.r.t. different state space search strategy, optimization techniques and heuristic functions.

Project Structure

The project has several independent parts that we combine to work as a whole. Directory "core" contains two factory methods that produce the puzzle solver and heuristic instance based on the parameter passed. Puzzle solutions can be implemented in separate directory as we have done for "n-queen" here. Directory "utils" all the utility methods that help other functions to operate.

Please keep in mind, this is an active project and project architecture may change without any prior announcement. But all the future updates will be maintained in a way so that the project integrity will ke kept, meaning you will not face any difficulty in building or running the project.

How to Build

```
# go to project directory
$ cd puzzle-solver

# build the project
$ make clean && make
```

n-queen

Problem Formulation

Given an integer N > 3, place N queens in an NxN board in such a way that no queens can attack each other. According to the chess rule, a queen is able to attack another queen if and only if both queens occupy the same row, column or diagonal on the board.

In this part of the project, we will start from a random board state and apply different variation of hill-climbing search to find a board configuration where all the queens will be placed in a way so that they don’t able to attack each other.

Implementation Domain

To solve n-queen problem, currently we have implemented different variation of hill-climbing search, they are:

Hill-Climbing variations	Is Complete?	Parameter token for this mode
Default Hill-Climbing search	No	-
Random-best-choice Hill-Climbing search	No	random-best
Stochastic Hill-Climbing search	No	stochastic
First-choice Hill-Climbing search	No	first-choice
Hill-Climbing search with sideways move	No	sideways-move
Random-restart Hill-Climbing search without sideways move	Yes	random-restart
Random-restart Hill-Climbing search with sideways move	Yes	random-restart-sways-move

How to Run

General run command:

```
> ./puzzle -problem {PUZZLE_NAME} \
-algo {ALGORITHM_NAME} -mode {MODE} \
-board_dim {DIMENSION_OF_THE_BOARD} -mx_sideways_move
{MAX_SIDEWAYS_MOVE}\
-print_path {PRINT_INITIAL_TO_GOAL}
```

Here is the parameter definition,

1. -problem:** Specify the puzzle name to solve, for example, "n-queen".
2. -algo:** Specify the search strategy to solve the puzzle, for example, "A*", "bfs", "dfs", etc.
3. -mode:** Specify the inner methodology for the search strategy, for example, "bi-directional" bfs, "stack-based" dfs, etc.
4. -board_dim:** Board dimension, may use representing any square board
5. -mx_sideways_move:** Maximum allowed sideways move in hill-climbing search
6. -print_path:** Flag to indicate printing path if solution exist, accepts boolean string, i.e., "true" or "false".

Basic Algorithm Description & Run Commands

For n-queen, here is the list of run commands for different implemented modes of the hill-climbing search algorithm -

- Default Hill-Climbing search

Implemented standard Hill-Climbing search algorithm, picked the best possible successor from a board configuration. If found multiple best possible successor, picked the first one.

Run command:
> ./puzzle -problem n-queen -algo hill-climbing -mode na -board_dim 10 -mx_sideways_move 0 -print_path false

- Random-best-choice Hill-Climbing search

Stored all the possible best successors in memory and picked a random one from them. Not memory efficient like the previous one.
The motivation of this implementation is to know whether the selection of best successor has any impact on the performance.

Run command:
> ./puzzle -problem n-queen -algo hill-climbing -mode random-best -board_dim 10 -mx_sideways_move 0 -print_path false

- Stochastic Hill-Climbing search

Stochastic hill climbing chooses the successor at random from among the uphill moves. This usually converges more
slowly than steepest ascent, but in some state landscapes, it finds better solutions.

Run command:
> ./puzzle -problem n-queen -algo hill-climbing -mode stochastic -board_dim 10 -mx_sideways_move 0 -print_path false

- First-choice Hill-Climbing search

First-choice hill climbing generates successors until one is generated that is better than the current state.
This is a good strategy when a state has many (e.g., thousands) of successors.

Run command:
> ./puzzle -problem n-queen -algo hill-climbing -mode first-choice -board_dim 10 -mx_sideways_move 0 -print_path false

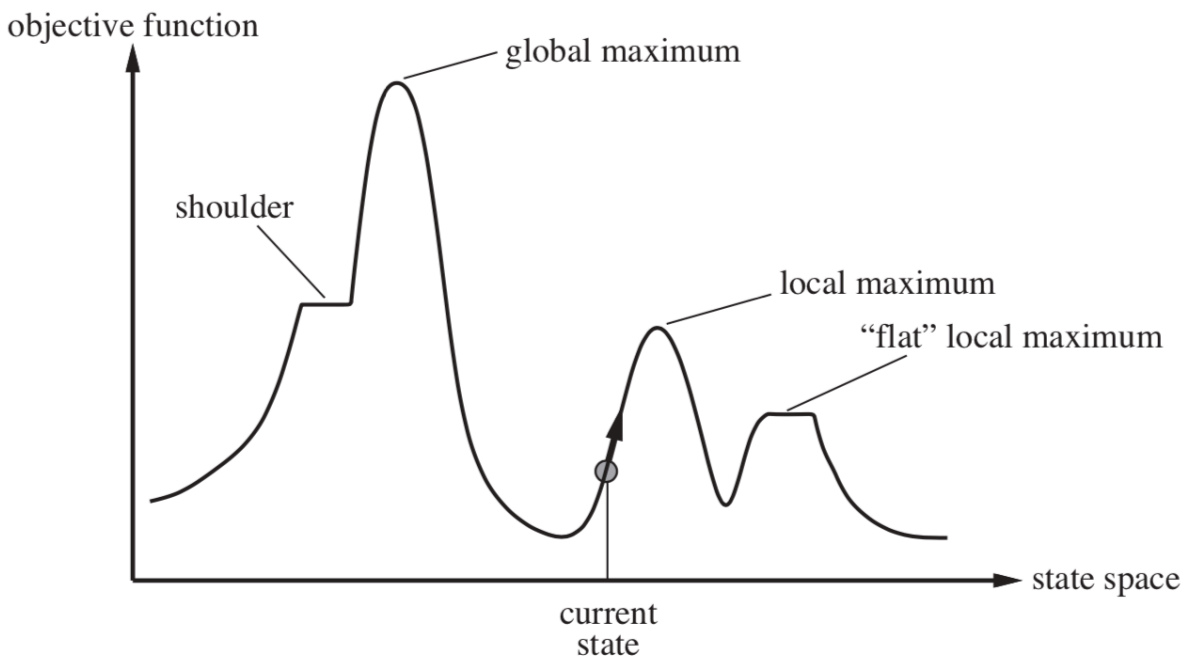


Figure 1: State-space landscape

- Hill-Climbing search with sideways move

The default Hill-climbing algorithm halts if it reaches a plateau where the best successor has the same value as the current state. This implementation allows a sideways move in the hope that the plateau is really a shoulder. An infinite loop will occur whenever the algorithm reaches a flat local maximum that is not a shoulder. So passes a parameter to the program to provide a limit on the number of consecutive sideways moves allowed.

Run command:

```
> ./puzzle -problem n-queen -algo hill-climbing -mode sideways-move -board_dim 10 -mx_sideways_move 10 -print_path false
```

- Random-restart Hill-Climbing search without sideways move

It conducts a series of hill-climbing searches from randomly generated initial states, until a goal is found. It is complete with probability 1 to reach the goal, because it will eventually generate a goal state as the initial state.

Run command:

```
> ./puzzle -problem n-queen -algo hill-climbing -mode random-restart -board_dim 10 -mx_sideways_move 0 -print_path false
```

- Random-restart Hill-Climbing search with sideways move

Allowed to walk in the plateau to see whether it is a shoulder or not.

Run command:

> ./puzzle -problem n-queen -algo hill-climbing -mode random-restart-sways-move -board_dim 10 -mx_sideways_move 10 -print_path false

Performance Characterization

All the tests performed on 10x10 chess board. Used 10 as the maximum limit of sideways move. While finding successor, moved the row cursor in the range of highest possible UP and DOWN limit.

How to reproduce

To reproduce the performance evaluation, please run scripts kept in "benchmark/n-queen" directory. You will get bash scripts to run different hill-climbing algorithms to solve custom n-queen problem. You can pass the board dimension and maximum number of consecutive steps in sideways move as the parameter of the program. You can save the output of these scripts in a file and use the parser python script to parse it. For incomplete searches you should use parser "parser_incomplete_search.py" and for complete searches you should use "parser_complete_search.py". Here I am giving the steps to reproduce the evaluation (one for incomplete search and another for complete search).

```
#incomplete search [Default Hill-Climbing search]
> cd benchmark/n-queen
> ./hill_climbing-benchmark.sh > hill_climbing-benchmark.out
> python3 parser_incomplete_search.py hill_climbing-benchmark.out

#complete search [Random-restart Hill-Climbing search without sideways move]
> cd benchmark/n-queen
> ./hill_climbing_rr-benchmark.sh > hill_climbing_rr-benchmark.out
> python3 parser_complete_search.py hill_climbing_rr-benchmark.out
```

Algorithm Performance

Incomplete Searches	Success Rate (in %)	Avg. steps in success	Avg. steps in failure
Default Hill-Climbing search	5.4	5.26	4.92
Random-best-choice Hill-Climbing search	6.8	5.21	5.04
Stochastic Hill-Climbing search	6.0	8.03	7.41
First-choice Hill-Climbing search	6.2	8.32	8.18
Hill-Climbing search with sideways move	19.6	6.59	15.33

Observations:

1. Picking a random from the best successors performs better than picking the first one.
2. Stochastic and First-choice Hill-Climbing search has less impact on performance for 10x10 board configuration.

3. Hill-Climbing search with sideways move has better success rate with a increased number of steps required in failure cases.
4. Without sideways move, it required less steps in failure cases.

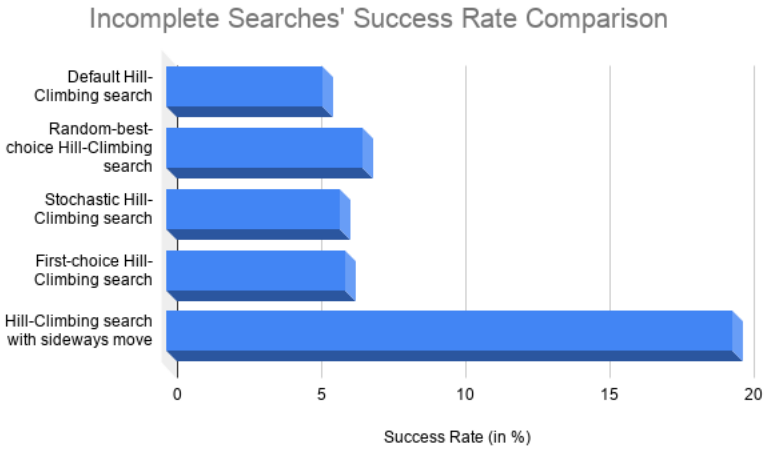


Figure 2 (a): Incomplete Searches' Success Rate Comparison

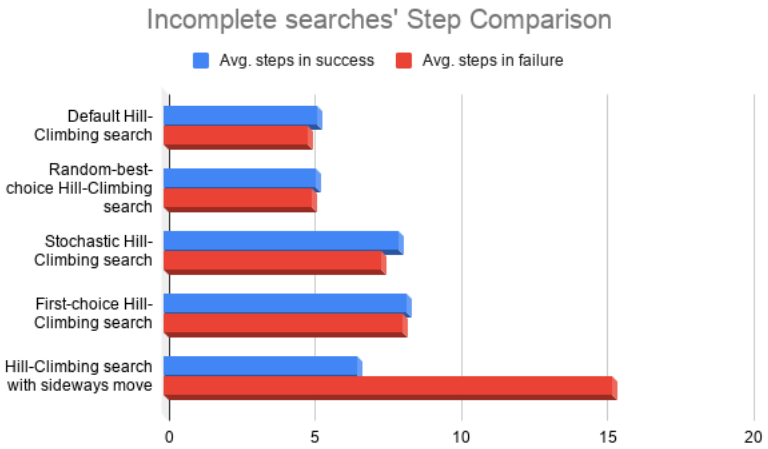


Figure 2 (b): Incomplete searches' Step Comparison

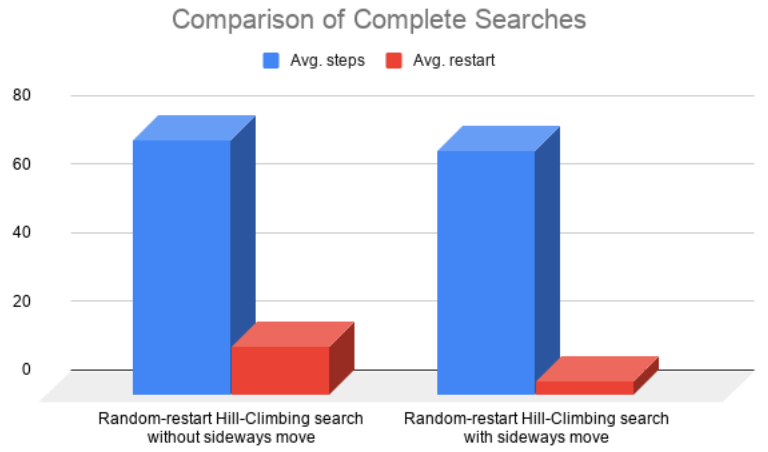


Figure 2 (c): Comparison of Complete Searches

Figure 2: Comparing performance of different algorithms for solving n-queen

Performance of Complete Search

Complete Searches	Success Rate (in %)	Avg. steps	Avg. restart
Random-restart Hill-Climbing search without sideways move	100	74.43	14.03
Random-restart Hill-Climbing search with sideways move	100	71.33	4.19

Observations:

1. Random-restart Hill-Climbing search perform better with sideways move and reduces the number of restarts 70%.

Future Works

Do analysis on complete searches' random restart required count distribution chart

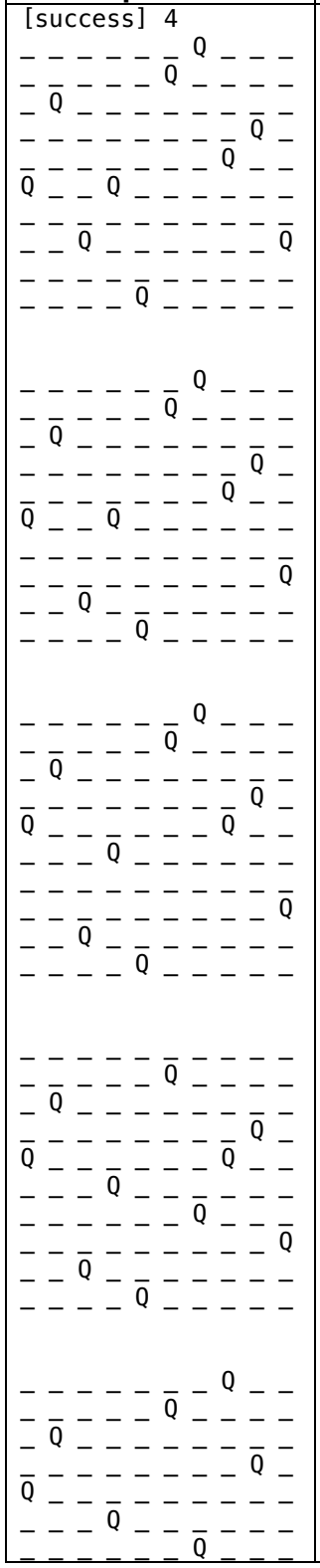
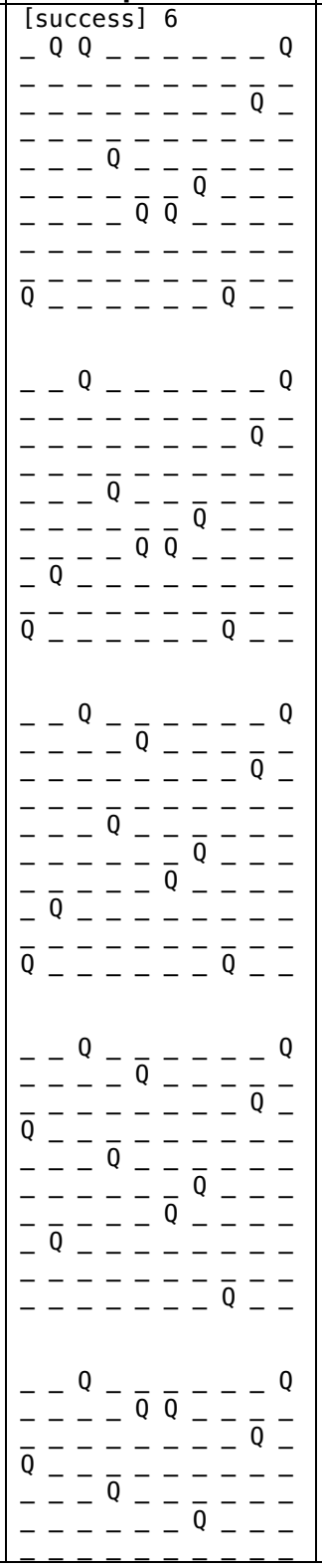
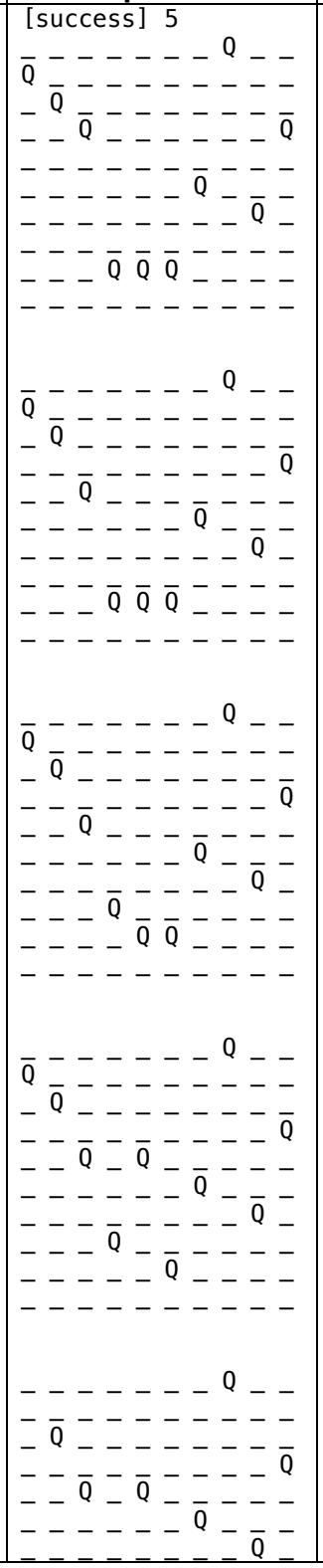
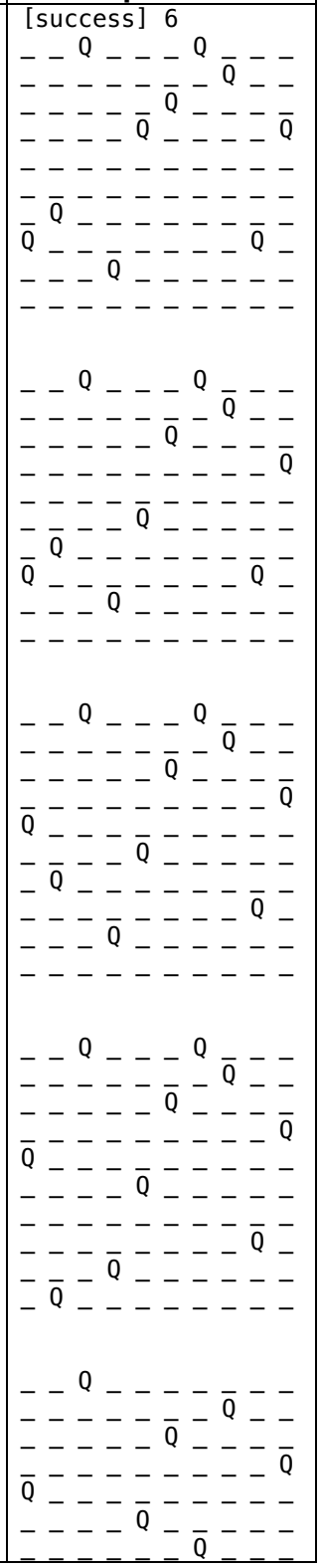
- Do similar performance test on higher dimension board.
- Do performance characterization of sideways move (impact on increasing limit of sideways move for fixed board dimension).
- Do compare performance (w.r.t. execution time) of backtracking Vs. Random-restart Hill-Climbing search w/wo sideways walk for smaller board.

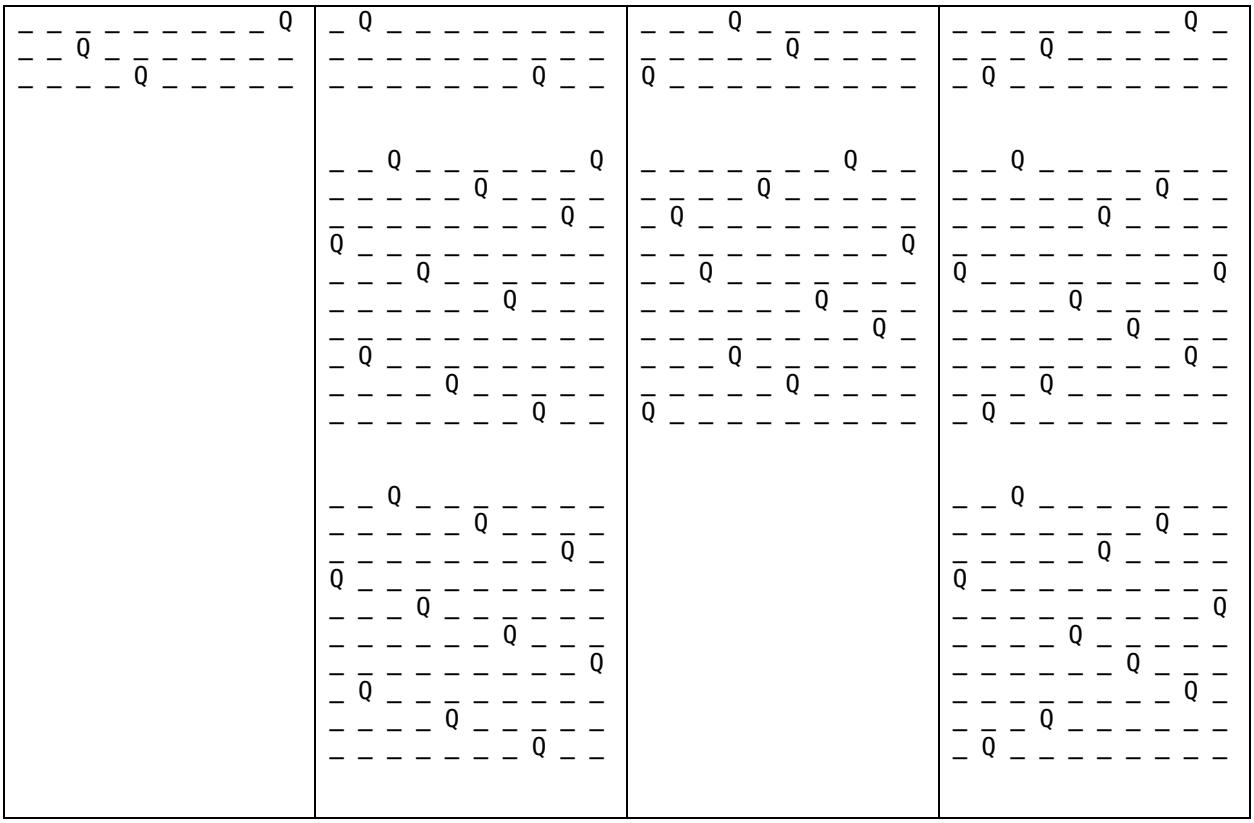
Resources

1. TBA

Search Sequence

- **Hill-Climbing search**

Sequence 1	Sequence 2	Sequence 3	Sequence 4
			



- **Hill-Climbing search with sideways move**

Sequence 1	Sequence 2	Sequence 3	Sequence 4
[success] 6	[success] 5	[success] 6	[success] 4
_ _ _ Q _ _ _ _ _ _ _ _ _ _ Q _ Q _ Q _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ Q _ Q _ _ _ _ _ _ _ _ _ _ _ _ Q _ _ _ _ _ _ _ Q Q _ _ _ Q _ _ _ _ _ _ _ _ _ _ _ _ _ _	Q _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ Q _ _ _ _ _ _ _ Q _ _ _ _ _ _ Q _ _ _ Q _ _ Q _ Q _ _ _ _ _ _ _ _ Q _ Q Q _ _ _ _ _ _ _ _ _ _ _ _ _	_ Q _ _ _ _ _ _ _ _ Q _ _ _ Q _ _ _ Q _ _ _ _ _ _ Q Q _ _ _ _ _ _ _ _ _ _ Q _ Q _ _ _ _ _ _ _ _ _ _ _ _ Q _ _ _ _ _ _ _ _ _ _ Q _ _ _ _ _ _	_ _ _ _ _ Q _ _ _ _ Q _ _ _ _ _ _ _ _ _ _ Q _ _ _ _ _ Q _ _ _ _ _ _ Q _ _ _ _ Q _ _ _ _ _ _ _ _ _ _ _ _ _ Q _ _ _ _ _ _ _ _ Q _ Q _ _ _ _ _ _ _ Q _ _ _
_ _ _ Q _ _ _ _ _ _ _ _ _ _ Q _ Q _ Q _ Q _ _ _ _ _ _ _ _ _ _ _ _ Q _ _ _ _ _ _ _ Q Q _ _ _ Q _ _ _ _ _ _ _ _ _ _ _ _ Q _	Q _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ Q _ _ _ _ _ _ _ Q _ _ _ _ _ _ Q _ _ _ Q _ _ Q _ Q _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ Q _ Q Q _ _ _ _ _ _ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _ _ _ Q _ Q _ _ _ _ _ _ _ _ _ _ _ _ Q _ _ _ Q _ _ _ _ _ _ Q Q _ _ _ _ _ _ _ _ _ _ Q _ Q _ _ _ _ _ _ _ _ _ _ _ _ Q _ _ _ _ _ _ _ _ _ _ Q _ _ _ _ _ _	_ _ _ _ _ Q _ _ _ _ Q _ _ _ _ _ _ _ _ _ _ Q _ _ _ _ _ _ _ _ _ _ _ _ Q _ _ _ _ Q _ _ _ _ _ _ _ Q _ _ _ _ _ Q _ _ _ _ _ _ _ _ Q _ Q _ _ _ _ _ _ _ Q _ _ _
_ _ _ Q _ _ _ _ _ _ _ _ _ _ Q _ Q _ Q _ Q Q _ _ _ _ _ _ _ _ _ _ _ Q _ _ _ _ _ _ _ _ Q _ _ _ Q _ _ _ _ _ _ _ _ _ _ _ _ Q _	Q _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ Q _ _ _ _ _ _ _ Q _ _ _ _ _ _ Q _ _ _ Q _ _ Q _ Q _ Q Q _ _ _ _ _ _ Q _ _ _ _ _ _	_ _ _ _ _ Q _ _ _ _ Q _ Q _ _ _ _ _ _ _ _ _ _ _ _ Q _ _ _ Q _ _ _ _ _ _ _ Q _ _ _ _ _ _ _ _ _ _ Q _ Q _ _ _ _ _ _ _ _ _ _ _ _ Q _ _ _ _ _ _ _ _ _ _ Q _ _ _ _ _ _	_ _ _ _ _ Q _ _ _ _ Q _ _ _ _ _ _ _ _ _ _ Q _ _ _ _ _ _ _ _ _ _ _ _ Q _ _ _ _ Q _ _ _ _ _ _ _ Q _ _ _ _ _ Q _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ Q _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ Q _ _ _ _ _ _ _ Q _ _ _
_ _ _ Q _ _ _ _ _ _ _ _ _ _ _ _ Q _ Q _ _ _ _ _ _ Q _ _ _ _ _ _ _ _ _ _ _ _ _ _ Q Q _ _ _ _ _ _ _ _ _ _ _ Q _ _ _ _ _ _ _ _ Q _ _ _ Q _ _ _ _ _ _ _ _ _ _ _ _ Q _	Q _ _ _ _ _ _ _ _ _ _ _ _ _ _ Q _ _ _ Q _ _ _ _ _ _ _ Q _ _ _ _ _ _ Q _ _ _ Q _ _ Q _ Q _ Q _ _ _ _ _ _ Q _ _ _ _ _ _	_ _ _ _ _ Q _ _ _ _ Q _ Q _ _ _ _ _ _ _ _ _ _ _ _ Q _ _ _ Q _ _ _ _ _ _ _ Q _ _ _ _ _ _ _ _ _ _ Q _ Q _ _ Q _ Q _ _ _ _ _ _	_ _ _ _ _ Q _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ Q _ _ _ _ _ _ _ _ _ _ _ _ Q _ _ _ _ Q _ _ _ _ _ _ _ Q _ _ _ _ _ Q _ _ _ _ _ _ Q _ _ _ _ _ _ _ _ _ _ _ Q _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ Q _ _ _ _ _ _ _ Q _ _ _
_ _ _ Q _ _ _ _ _ _ _ _ _ _ _ _ Q _ _ _ _ _ _ _ _ _ _ _ _ _ _ Q _ _ _ _ _ Q _ _ _ _ _ _ Q _ _ _ _ _ _ _ _ _ _ _ _ _ _ Q Q _ _ _ _ _ _ _ _ _ _ _ Q _ Q _ _ _ _	Q _ _ _ _ _ _ _ _ _ _ _ _ _ _ Q _ _ _ Q _ _ _ _ _ _ _ Q _ _ _ _ _ _ Q _ _ _ _ _ _ Q _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ Q _ _ _ Q _ Q _ _ _	_ _ _ _ _ Q _ _ _ _ Q _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ Q _ _ Q _ _ _ _ _ _ _ _ _ _ _ _ Q _ _ _ _ _ _ _ _ _ _ _ Q _ _ _ _ _ _ _ _ _ _ Q _ Q _ _ Q _	_ _ _ _ _ Q _ _ _ _ _ _ _ _ _ _ _ _ _ Q _ Q _ _ _ _ _ _ _ _ _ _ _ _ Q _ _ _ _ _ _ _ _ _ _ _ _ Q _ _ _ _ _ Q _ _ _ _ _ _ Q _ _ _ _ _ _ _ _ _ _ _ Q _ Q _