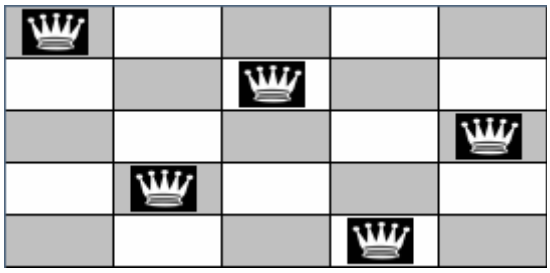
Backtracking

- Problem solving strategy that, when it reaches an impasse, **retraces its steps in reverse order** before trying a new sequence of steps

- Often used with **recursion**

Example: **N-Queens Problem**



Solution for N = 5

Strategy Using Backtracking:

1) Start with a queen in the 1st row and 1st column.

2) Place the 2nd queen in the 2nd column, at the first row where this queen will not be under attack.

3) Repeat step (2) with additional queens until the *problem is solved* or, it is *not possible to place a queen* in a column without being attacked.

4) If step (3) failed to place a queen in a column, then undo what you just tried for this queen. Then backup to the previous column and move that queen to the next, unattacked row.

5) Repeat steps (3) and (4) until the *problem is solved* or, there are no more possibilities for placing a queen in the 1st column. In this case, there is no solution.

Incorporating Recursion:

Solution = (positioning of 1^{st} queen) + (positioning of the other N-1 queens)

Example of Dynamic Allocation of Memory for 2D Array

```
#include <cstdlib>
#include <iostream>
#include <cassert>
using namespace std;

void memError( ) {
  cout << "Memory allocation error!\n";
  exit(1);
}

int main() {
int **A;  // pointer to a 2D array of int
int i, j, numRows, numColumns;

  cout << "Enter number of rows: ";
  cin >> numRows;
  cout << "Enter number of columns: ";
  cin >> numColumns;

  assert((numRows > 0) && (numColumns > 0));

  // Allocate storage for the array
  A = new int*[numRows];  // Allocate storage for rows
  if (! A) memError( );

  for (i = 0; i < numRows; i++) {
    A[ i ] = new int[numColumns]; // Allocate storage for columns in row i
     if ( !A[ i ] ) memError( );
  }

  // Put some values in the array
  // (just to show that you can now reference elements as A[ i ][ j ])
  for (i = 0; i < numRows; i++)
    for (j = 0; j < numColumns; j++)
       A[ i ][ j ] = i * j;

  // Deallocate storage for the array
  for (i = 0; i < numRows; i++)
    delete [ ] A[ i ];     // Deallocate storage for columns in row i

   delete [ ] A;          // Deallocate storage for rows

  return(0);
}
```