8-queens - backtracking solution 1

```
#include <cstdlib>                       // we use the int version of 'abs'
#include <cstdio>
#include <cstring>
using namespace std;

int row[8], TC, a, b, lineCounter;            // ok to use global variables

bool place(int r, int c) {
  for (int prev = 0; prev < c; prev++)    // check previously placed queens
    if (row[prev] == r || (abs(row[prev] - r) == abs(prev - c)))
      return false;        // share same row or same diagonal -> infeasible
  return true;
}

void backtrack(int c) {
  if (c == 8 && row[b] == a) {         // candidate sol, (a, b) has 1 queen
    printf("%2d      %d", ++lineCounter, row[0] + 1);
    for (int j = 1; j < 8; j++)
      printf(" %d", row[j] + 1);
    printf("\n"); }
  for (int r = 0; r < 8; r++)                       // try all possible row
    if (place(r, c)) {          // if can place a queen at this col and row
      row[c] = r; backtrack(c + 1);        // put this queen here and recurse
    }
}

int main() {
  scanf("%d", &TC);
  while (TC--) {
    scanf("%d %d", &a, &b); a--; b--;         // switch to 0-based indexing
    memset(row, 0, sizeof row); lineCounter = 0;
    printf("SOLN       COLUMN\n");
    printf(" #      1 2 3 4 5 6 7 8\n\n");
    backtrack(0);           // generate all possible 8! candidate solutions
    if (TC)
      printf("\n");
  }
} // return 0;
```

8-queens - backtracking solution 1

```
import java.util.*;

class Main {
  private static int[] row = new int[9];
  private static int TC, a, b, lineCounter;

  private static boolean place(int col, int tryrow) {
    for (int prev = 1; prev < col; prev++) // check previously placed queens
      if (row[prev] == tryrow || (Math.abs(row[prev] - tryrow) == Math.abs(prev
- col)))
        return false; // an infeasible solution if share same row or same
diagonal
    return true;
  }

  private static void backtrack(int col) {
    for (int tryrow = 1; tryrow <= 8; tryrow++) // try all possible row
      if (place(col, tryrow)) { // if can place a queen at this col and row...
        row[col] = tryrow; // put this queen in this col and row
        if (col == 8 && row[b] == a) { // a candidate solution & (a, b) has 1
queen
          System.out.printf("%2d      %d", ++lineCounter, row[1]);
          for (int j = 2; j <= 8; j++) System.out.printf(" %d", row[j]);
          System.out.printf("\n");
        }
        else
          backtrack(col + 1); // recursively try next column
      }
  }

  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    TC = sc.nextInt();
    while (TC-- > 0) {
      a = sc.nextInt();
      b = sc.nextInt();
      for (int i = 0; i < 9; i++) row[i] = 0;
      lineCounter = 0;
      System.out.printf("SOLN       COLUMN\n");
      System.out.printf(" #      1 2 3 4 5 6 7 8\n\n");
      backtrack(1); // generate all possible 8! candidate solutions
      if (TC > 0) System.out.printf("\n");
    }
  }
}
```

alternative to backtrack function for $n$ -queens

```
void backtrack(int c) {
  if (c == n) {         //we have the solution
    ans++;
    return;
  }
  for (int r = 0; r < n; r++)                      // try all possible rows
    if (board[r][c] != '*'
        && !rw[r]
        && !ld[r-c+n-1]
        && !rd[r+c] )
    {
      rw[r] = ld[r-c+n-1] = rd[r+c] = true;
      backtrack(c + 1);
      rw[r] = ld[r-c+n-1] = rd[r+c] = false;

    }
}
```