Course Introduction
Introduction to Competitive Programming

Dr. Mattox Beckman

University of Illinois at Urbana-Champaign
Department of Computer Science

Spring 2024

Objectives

▶ Describe four patterns of brute force;

▶ Describe the times when a brute force solution is necessary.

▶ Describe some techniques to optimize brute force algorithms.

What is it?

▶ You must traverse the entire problem space to get the answer.

▶ Sometimes you can prune the problem space.

| 8 | 6 | 7 | 5 | 3 | 0 | 9 |

```
1  max=a[0]; // why not just put 0 here?
2  for(int i=1; i<7; i++)
3      if (a[i]>max) max=a[i];
```

When to Use It

▶ Tradeoffs
  ▶ Bad: It's slow!
  ▶ Good: It's simple! More likely to give correct solution.
▶ Three situations:
  ▶ When you have no choice.
  ▶ When the problem set is small.
  ▶ To verify your real solution!Introduction

Categories

▶ Code Pattern
  ▶ Iterative
  ▶ Recursive
▶ Traversal Pattern
  ▶ Filtering
  ▶ GeneratingIntroduction

Speed

- ▶ Use bits instead of boolean arrays
- ▶ Use primitive types when appropriate:
    - ▶ int32 instead of int64
    - ▶ arrays instead of vector
    - ▶ character arrays instead of string
- ▶ Prefer iteration to recursion
- ▶ The STL algorithm include has `next_permutation`, which is very fast
- ▶ Declare large data structures in the global scope

The *n*-queens problem

	x		x	
x	x	x		
x	Q	x	x	x
x	x	x		
	x		x	

▶ If you don't know chess, you might want to learn the basic rules.

▶ Classic problem: place *n* queens on a $n \times n$ chessboard.

    ▶ How many ways can you do it?

Two examples

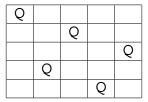## How to write this?

▶ Attempt 1 : Massive nested `for` loops

```
vvi board(8,vi(8)); // get
count = 0;
for(i=0; i<7; ++i) {
  board[0][i] = 1; // place queen
  for(j=0; j<7; ++j)
    if (! collides(board,1,j)) {
      board[1][j] = 1;
      for(k=0; ...) ; // rest of program
      board[1][j] = 0;
    }
  board[0][i] = 0; // remove queen
```

▶ Final position; if no collisions increment `count`.

▶ What do you think of this code? 8^8 attempts….

Improvements

- ▶ We don't need 8^8 checks.
- ▶ Instead of modeling the chess board, model where the queens are placed.
    - ▶ This example is {0,3,1,4,2}.

Backtracking

▶ Example code from *Competive Programming 4*

```
1  void backtrack(int c) {
2    if ((c == 8) && (row[b] == a)) { // a candidate solution
3      printf("%2d %d", ++lineCounter, row[0]+1);
4      for (int j = 1; j < 8; ++j)
5        printf(" %d", row[j]+1);
6      printf("\n");
7      return; // optional statement
8    }
9    for (int r = 0; r < 8; ++r) { // try all possible row
10     if ((c == b) && (r != a)) continue; // early pruning
11     if (canPlace(r, c)) // can place a Queen here?
12       row[c] = r, backtrack(c+1); // put here and recurse
13   }
14 }
```

Checking Placement

► Don't walk the diagonals; use math!

```
1 bool canPlace(int r, int c) {
2   for (int prev = 0; prev < c; ++prev) // check previou
3     if ((row[prev] == r) ||
4         (abs(row[prev]-r) == abs(prev-c)))
5       return false; // infeasible
6   return true;
7 }
```

Using bitmasks

► Keep three bit-vectors.

► Shifting the bits handles the diagonals.

```
// Place a queen in row $r$
rows |= (1 << r);
up |= (1 << r);
down |= (1 << r);

// Moving to the next column...

up <<= 1;
down >>= 1;
```