Bitmask Technique

Hello, and welcome to Competitive Programming! Today we are going to talk about using bitmasks to speed up certain kinds of recursive searches.

Objectives

Your objective is to be able to use this technique to solve the n-queens problem.

The N Queens Problem

I'm sure you remember the n-queens problem. You are given a chess board of size (n \times n) and you need to place (n) queens on it.

You could do this by making an (n \times n) array and looping through it, but this algorithm would easily be (O(n^3)). We can do much better than that.

Using bitmasks

Here's the technique. We are going to use four bitmasks.

The first one is going to be called `OK`. It will have (n) ones set, and we are going to use it to calculate which positions are open. It's easy to do that by shifing a one (n) positions over and then subtracting one from the result.

We will need three other sets too: one to keep track of which rows have been used up, one to keep track of diagonals attacking upward, and one to keep track of diagonals attacking downward.

These three will be initialized to zero.

Backtracking

Here's the code. We will use `ans` and `OK` as global variables.

The first thing we do in the backtracking function is to check the `rw` variable. If we have assigned all (n) rows, it means that we have found a solution, so we increment `ans`. We could also do this by keeping track of which column we were on.

If we haven't yet built a solution, we take the three attack bit patterns and or them together. This gives us bits for the rows that are blocked. We negate this and and it with the `OK` pattern to get the bits that are available.

This trick by itself takes out an order-n factor from our algorithm. Rather than checking all the previous queens to see what effect they have on the current column, we can calculate it in order-1 time.

Now we do another trick to save time. You may remember the formula to get the least significant one by anding a number with it's negative. Toward the end of the problem there are only going to be a few squares that are available. Rather than looping through everything and checking mostly unavailable squares, we only enumerate the squares that are available.

The last trick is clever. We make our recursive call and or the current row to the three attack patterns. But the we left-shift one of the diagonal patterns and right shift the other. This updates the diagonals to the values they should have for the next column.

This algorithm now is very fast. By using the bitmasks, we are effectively getting parallelism for free.

Next
Transcript — Balanced Binary Trees and Heaps →