Complete Search

Hello, welcome to competitive programming.

Today we are going to talk about a strategy called “complete search”, or more commonly known as brute force.

Objectives

Your objectives are to be able to describe the different kinds of brute force algorithms, know when a brute force algorithm is a good idea, and describe some ways to optimize brute force code.

What is it?

A complete search algorithm is one in which you traverse the entire search space to find the solution to the problem.

For example, suppose you are given the following array. If we ask you to find the minimum element, a brute force approach would be to simply use a for loop to select the element, like this.

Of course, a real problem is going to be more complicated.

When to use it

A complete search algorithm has a serious downside: it is slow. If you can prune the search space somehow that will speed things up.

But there is a major advantage as well: you are very unlikely to get a wrong answer verdict, since you will have considered all possible solutions.

There are three situations, then, when you will want to use complete search.

First, with some problem sets you simply don't have a choice. It will be important that you develop the skill to recognize when this is the case.

Second, if the problem set is small enough, you may be able to use a brute force solution and still be fast enough to beat the time limit. If you know this to be the case, it is good contest strategy to use brute force since it will be faster to program a correct solution, and you will get a better time score this way.

A good example of this might be a range max query for a static array of size 100. You could program a segment tree for this, but it's overkill for a problem space that small unless you are going to get a very large number of queries.

The third reason is a bit more subtle: you can use a brute force implementation to help write the implementation you intend to submit. One way this can help is as a way to verify your real solution on small test cases. A brute force algorithm can also give you some sample values to help you get an intuition for what the solutions look like.

Kinds of Search

There are two main kinds of brute force searches. The first is iterative search. Basically this means using `for` loops. Many problems will need nested `for` loops.

The second kind uses recursion. The n queens type problems are a famous example of these.

We'll give you one of each of these problems to look at for class.

There are two other ways to categorize these solutions as well. One is called filtering. In filtering, you generate all the possible solutions and keep the ones that work.

A faster method, if you can use it, is called generating. With this technique, you construct the solutions you know to be correct.

Speed

The fact that you have to do a complete search doesn't leave you without options. There are two things you can do to speed the search. The first is to be clever about how you prune the search space. Sometimes you can rule out a large amount of candidates before you need to do a full evaluation of them.

The second thing you can do is optimize the code where you check if the searched solution is valid or not.

Here are some tips to speed things up. I won't read them to you. The section in the textbook has a lot more details about these kinds of problems and many examples; do take a look at that if you have access to it.

Next
Transcript — Dijkstra's Algorithm →