Ad Hoc Problems

Title

Hello, and welcome to Competitive Programming.

Today we are going to talk about a simple problem type, the “ad hoc” problems.

Objectives

Your goal is to be familiar with some of the problem types that show up, and some of the things you will want to keep in mind when solving them. There are a couple of key take-away points you will want to learn from solving these problems, too.

What is an Ad-Hoc Problem, Anyway?

When we say “ad hoc”, we usually mean that you do not need to apply a sophisticated algorithm to solve the problem; instead you will write something specific to the problem at hand. Often these problems involve simulations of games or modeling some kind of process. Many of these problems will have a “just follow the instructions” feel to them.

What can make these problems challenging is that the edge cases can be very tricky.

There are many kinds of ad-hoc problems, some of which do involve more specialized algorithmic knowledge, but today we will talk about four basic categories. They are games, simulations, strings, and the so-called time wasters.

Games

Card Games

One kind of problem set will involve the use of playing cards, especially poker and other variants. I once had a student who became extremely uncomfortable when he was given a problem involving poker cards. He protested that he didn't know about them, saying "I don't gamble". Don't worry, nobody is asking you to gamble money here … you're just gambling your time and a correct solution to a programming problem!

If you are not familiar with the cards, they have two features of interest. The first is their suit. They function a little like colors, except they have symbols to represent them. The four suits are hearts, diamonds, clubs (which use a clover like symbol), and spades (kind of like a pointy shovel with the pointy end pointing up).

The second feature is their rank, or value. There are numbers from 2 to 10, an "Ace" card that could have been a "1" except is uses the letter A instead, and three "royalty" cards of Jack, Queen, and King. Some systems also have a card called the Joker, and it usually does not have a suit. Depending on the game, the Ace is either the highest valued card or the lowest.

These problems are interesting because the combination of suit and rank make an ordered pair, and there are many ways of ranking individual cards as well as combinations of cards. This can lead to simulation problems, as well as combinatorial problems.

Almost always, a problem setter will not make huge assumptions about what card games you know, but you may find it helpful to read over the rules for games like 5 Card Draw Poker, since the kinds of card combinations that occur are *almost* considered common knowledge. Here's a list to get you started.

Chess and Checkers

The rules of chess are almost common knowledge as well, and occasionally there is a problem that assumes you know the rules and how the pieces move. The knight, in particular, shows up in a lot of problems. I'm not going to go over the rules here, since the odds are you already know them. But if you have not been exposed to Chess or Checkers, it's worth spending a few minutes learning the rules so they seem familiar when a problem comes up.

As an example of a more difficult problem: in the 2019 World Finals, one of the problems gave a list of checkers moves, and asked the programmer to check if a checker board existed in which those moves would be legal. It was tricky because you were not told the initial placement of any of the game pieces.

Again, usually the problem setter will explain the rules that are relevant to the problem, but it will be faster for you if you are familiar with the games.

Strings and Such

Strings

There are a lot of string manipulation problems and interesting algorithms, but two simple string problem types involve anagrams and palindromes. An anagram is a rearranging of a word or sentence to form another one. For example the latest tapes for the "Doctor Who" show were labeled with the anagram "Torchwood" to keep people from stealing them. A palindrome is simply a word that is spelled the same forwards and backwards, like the sample sentence "Madam, I'm Adam". One tricky thing with palindromes is that we often don't consider punctuation or spacing as part of the palindrome, so simply reversing a string might not be enough.

Dates

If you are working with dates, you will either have to memorize a bunch of calendrical algorithms, or else switch to either Python or Java. It is worth knowing

these languages, or at least ensuring someone on your team does, for this reason, and also because both support big integers natively.

With dates, it is easy to add in edge cases to mess you up. Periods that span a day, month, or year boundary can invalidate an assumption you made about your problem: for example, if you are simulating a commute you might not thing that the commuter would return at 1am.

The fact that months don't all have the same number of days and that there sometimes are leap-years can also cause trouble.

One utility that C and C++ do have though is `strftime`, in case all you need it to read or write strings.

Final Considerations

Time Wasters

Some problems seem designed to eat up programmer time. If you get one of these, your solution will test more your attention to detail and the efficiency of your coder — who can implement a tricky solution accurately within the time constraints?

Your team should have a strategy for dealing with this kind of problem. If you simply want to get a good score in the contest, consider saving that problem for last. The time you spend on this problem will increase the time on all the other unsolved problems as well. Of course, if your goal is to win, then you will need to solve it.

Why These are Valuable

We will start talking about more algorithmic stuff in the following lectures, but these problems do have some benefit.

The main one is that they teach you to pay careful attention to detail. These problems are not tricky because of the algorithmic knowledge you need, but

because if you are not careful with them you can get a wrong answer simply because your presentation was wrong or else you had a wrong assumption about something.

As your coding ability increases, you will notice you solve these kinds of problems more quickly and more accurately.

One recommendation I have: make it your goal that you never get a compile error when you are competing. It sounds crazy, but having that attitude will save you time later since you don't have to reopen the editor to fix a semicolon somewhere. The skill you need is the same: teach yourself to pay attention to details, and it will save you time in the end.

Next
Transcript — Advanced DP →