SIG Algorithm Challenges

Week 9: Introduction to system design

Dane's Contact Email: dziema2@uic.edu
Github: Dane8373

Slides for this week available NOW at
https://github.com/dane8373/SIG_Algorithm_Challenges/tree/master/Week8

Join our Slack channel, #sig_algorithm_chal on the UIC ACM slack (uicacm.slack.com)

Mock Technical Interviews Available!

Sign up here: http://bit.ly/SIG_AC_TECH

Good Resources for System Design

Hiredintech.com
http://blog.gainlo.co/index.php/category/system-design-interview-questions/
“Success in tech” youtube channel

System Design

Most of the questions we have covered so far have been algorithm-based questions.

- in addition to these types of questions, ~25% of questions at job interviews are system design questions.
- Additionally, many times after solving an algorithms question, an interviewer will ask something like “How would you change this to work in X system or for Y purpose”
- These questions are generally quite long, so we will just introduce them today. I highly recommend studying them on your own time.

System design questions, in contrast to algorithm questions, are quite vague and have many different ways to approach

Examples

- Design Twitter/Uber from scratch
- Design a parking lot (or other real world concept)
- They define a specific set of features, and ask you to implement a system to perform it

First Step: Gather Requirements

The first step to solving a system design problem is to gather the specific requirements your interviewer is asking about

Example: You are asked to design Twitter from scratch

- The most important feature of twitter is reading and writing tweets, but there are other features as well
  - Retweeting and liking
  - Following
  - Recommended Followers
  - Search
  - Advertisements
  - Trending Stories
  - Timeline
  - Personal Pages
- It is very important to ask about which of these specific features the interviewer would like you to base you implementation around.
- Getting this right is essential, if the interviewer doesn't care about trending stories or search, then a good design would be very different if they had

Second Step: High level Overview

Once you have all the requirements spelled out, the next step is to propose a high level solution for this system

Keeping with the twitter example, to implement a simplistic version where you have a timeline, and your own personal page, some necessary implementations are

- Database used to store the tweets
  - Need to spell out how the tweets will be stored (Will you group them by the who posted them? Will you store each persons individual page? Or will you generate it)
- Interface used to interact with the database

Without going into too much detail, just give an overview of the pieces involved

Third Step: Discuss Trade-offs

No matter what you answer for a system design question, you will be wrong in some way! The time/space tradeoff is king in system design problems and must always be discussed.

Consider a design for twitter where we store everyone's individual pages vs one where we generate it

- The one that stores each page will have lower latency, but will take up more space
- Depending on the requirements, the tradeoff may or may not be worth it

It is important to talk about this tradeoff, and get feedback about which side is more important to the interviewer.

Additionally, there is also a redundancy/latency tradeoff that should be discussed.

- There are often strategies that provide more reliability via redundancy, but as a result require more overhead to keep all the redundant parts consistent

Fourth Step: Build a Small Scale Design

Once you have discussed tradeoffs, gotten a good idea for the requirements and made a high level design it is time to get into the nitty gritty

- Describe how users will interact with your system
- Describe how the various parts will of your system will work together
- Specify what specific types of implementations you will use (e.g. relational vs non-relational database)

This point generally requires a bit of area specific knowledge, but for an entry level role you can get away with being a bit vague

Final Step: Scale up

After you make your small scale design, the final step is to discuss how your design would scale

- Would your database design still work with 1,000,000 users?
- Would you use a distributed database? How would you handle redundancy/consistency?
- Is there opportunities for parallelism?

One major part of scaling up a design is to identify any “Single point of failure”, these are points in the system where if that part breaks, then the entire system would break

- Sometimes, these are unavoidable, but whenever possible you should seek to eliminate these.

Practice Problems From All Sections

Given two non-empty binary trees s and t, check whether tree t has exactly the same structure and node values with a subtree of s. A subtree of s is a tree consists of a node in s and all of this node's descendants. The tree s could also be considered as a subtree of itself.

Example 1:

Given tree s:

Given tree t:

Return true, because t has the same structure and node values with a subtree of s.

Problem Approach

Like most tree problems, this problem can best be solved using recursion

Call the tree we are looking for S and the tree we are searching in T

1: If S is null, return true

2: if the T is null, return false

3: if the root of S is equal to the root of T,, then return true if the left and right subtrees of S and T are equal.

4: if not, then search for S in the left and right subtree of T

Practice Problems From All Sections

Given two strings *s* and *t*, determine if they are isomorphic.

Two strings are isomorphic if the characters in *s* can be replaced to get *t*.

All occurrences of a character must be replaced with another character while preserving the order of characters. No two characters may map to the same character but a character may map to itself.

Problem Approach

Observation: if there is any hope of the strings being isomorphic, then the first letter in the first string must map to the first letter of the second tree, the second letter must morph to the second etc.

A hashMap is a good way to store this isomorphism

1: Starting at the first character, add the pair (String1.char(0),String2.char(0))

2: For every other index i, check to see if character i in string 1 already maps to something

3: If it does, and it is not equal to character i in string2, then return false. Otherwise move on to the next character

4: if it doesn't, check to see if charcter i in string 2 has been mapped to already, if it has, then return false. Otherwise add String1.char(i),String2.char(i)) to the map

5: return true if no violations are found

Practice Problems From All Sections

Implement the following operations of a stack using one or more queues as the backing data structure.

- push(x) -- Push element x onto stack.
- pop() -- Removes the element on top of the stack.
- top() -- Get the top element.
- empty() -- Return whether the stack is empty.

Problem Approach

Key idea: Use an “in queue” and an “out queue”

Push: Add the element to the inqueue, then empty the outqueue into the inqueue, and switch the inQueue to be the new outQueue, and the inQueue to be the now empty outQueue

Pop: remove from the outqueue;

Top: Return the top of the outqueue

Empty: Return whether or not the outqueue is empty

Next Week: Problem Examples