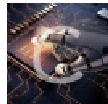
google的面试要求（自己的标杆）

转载

Garvin Li 于 2015-07-09 15:42:08 发布 3244 收藏
分类专栏： 学习笔记 文章标签： 面试


 学习笔记 专栏收录该内容

63 篇文章 0 订阅
订阅专栏

http://sites.google.com/site/steveyegge2/five-essential-phone-screen-questions

e Five Essential Phone-Screen Questions

Stevey's Drunken Blog Rants™

I've been on a lot of SDE interview loops lately where thecandidate failed miserably: not-inclined votes all around, even fromthe phone screeners who brought the person in initially.

It's usually pretty obvious when the candidate should have beeneliminated during the phone screens. Well, it's obvious inretrospect, anyway: during the interviews, we find some horrible flawin the candidate which, had anyone thought to ask about it during thephone screen, would surely have disqualified the person.

But we didn't ask. So the candidate came in for interviews and woundup wasting everyone's time.

Antipatterns

I've done informal postmortems on at least a hundred phone screens,many of them my own. Whenever a candidate bombs the interviews, Iwant to know what went wrong with the screen. And guess what? Apattern has emerged. Two patterns, actually.

The first pattern is that for most failed phone screens, *the candidate did most of the talking.*The screener only asked about stuff on the candidate's resume, and thecandidate was able to talk with passion and enthusiasm about thisincredibly cool thing they did, blah blah blah, and the screener wasduly impressed.

That's how many/most phone screens go wrong.

The right way to do a phone screen is to do most of the talking, or atleast the driving. You look for specific answers, and you guide theconversation along until you've got the answer or you've decided thecandidate doesn't know it. Whenever I forget this, and get lazy andlet the candidate drone on about their XML weasel-pin connectorproject, I wind up bringing in a dud.

The second pattern is that *one-trick ponies only knowone trick.* Candidates who have programmed mostlyin a single language (e.g. C/C++), platform (e.g. AIX) orframework (e.g. J2EE) usually have major, gaping holes intheir skills lineup. These candidates will fail their interviewshere because our interviews cover a broad range of skill areas.

These two phone screen (anti-)patterns are related: if you only askthe candidate about what they know, you've got a fairly narrow view oftheir abilities. And you're setting yourself up for a postmortem onyour phone screen.

Acid Tests

In an effort to make life simpler for phone screeners, I've puttogether this list of Five Essential Questions that you need to askduring an SDE screen. They won't guarantee that your candidate willbe great, but they will help eliminate a huge number of candidates whoare slipping through our process today.

These five areas are litmus tests -- very good ones. I've chosen thembased on the following criteria:

1) *They're universal* - every programmer needs to know them,regardless of experience, so you can use them in*all* SDEphone screens, from college hires through 30-year veterans.

2) *They're quick* - they're areas that you can probe veryquickly, without eating too much into your phone-screen time. Eacharea can be assessed with 1 to 5 minutes of "weeder questions", andeach area has almost unlimited weeder questions to choose from.

3) *They're predictors* - there are certain common "SDEprofiles" that are easy to spot because they tend to fail (and I mean*really* fail) in one or more of these five areas. So theareas are amazingly good at weeding out bad candidates.

You have to probe all five areas; you can't skip any of them. Eacharea is a proxy for a huge body of knowledge, and failing it verylikely means failing the interviews, even though the candidate didfine in the other areas.

Without further ado, here they are: The Five Essential Questions forthe first phone-screen with an SDE candidate:

1) **Coding.** The candidate has to write some simple code,with correct syntax, in C, C++, or Java.
2) **OO design.** The candidate has to define basic OOconcepts, and come up with classes to model a simple problem.
3) **Scripting and regexes.** The candidate has to describehow to find the phone numbers in 50,000 HTML pages.
4) **Data structures.** The candidate has to demonstratebasic knowledge of the most common data structures.
5) **Bits and bytes.** The candidate has to answersimple questions about bits, bytes, and binary numbers.

Please understand: what I'm looking forhere is a *total vacuum* in one of these areas.It's OK if they struggle a little and then figure it out.It's OK if they need some minor hints or prompting.I don't mind if they're rusty or slow. What you're lookingfor is candidates who are utterly clueless, or horriblyconfused, about the area in question.

For example, you may find a candidate who decides thata Vehicle class should be a subclass of ParkingGarage,since garages contain cars. This is just busted, and it'sun-fixable in any reasonable amount of training time.

Or a candidate might decide, when asked to search forphone numbers in a bunch of text files, to write a 2000-lineC++ program, at which point you discover they've neverheard of "grep", or at least never used it.

When a candidate is totally incompetent in one of theseBig Five areas, the chances are very high that they'llbomb horribly when presented with our typical interviewquestions. Last week I interviewed an SDE-2 candidatewho made both of the mistakes above (a vehicle inheritingfrom garage, and the 2000-line C++ grep implementation.)He was by no means unusual, even for the past month. We'vebeen bringing in *many* totally unqualified candidates.

The rest of this document describes each area in more detail,and gives example questions, and solutions.

Area Number One: Coding

The candidate has to write some code. Give them a codingproblem that requires writing a short, straightforwardfunction. They can write it in whatever language theylike, as long as they don't just call a library functionthat does it for them.

It should be a trivial problem, one that even a slow candidate cananswer in 5 minutes or less.

(If the candidate seems insulted by the thought of having to get theirhands dirty with a trivial coding question, after all their years ofexperience, patents, etc., tell them it's required procedure and askthem to humor you. If they refuse, tell them we only interview peoplewho can demonstrate coding skills over the phone, thank them for theirtime, and end the call.)

Give them a few minutes to write and hand-simulate the code. Tellthem they need to make it syntactically correct and complete. Makethem read the code to you over the phone. Copy down what they readback. Put it into your writeup. If they're sloppy, or don't want togive you exact details, give them one more chance to correct it, andthen go with Not Inclined.

(Note added 10/6/04) -- anothergood approach being used by many teams is to give the candidate"homework". E.g. you can give them an hour to solve some codingproblem (harder than the ones below) and email the solution to you.Works like a charm. Definitely preferable to reading code over thephone.

Anyway, here are some examples. I've given solutions in Java, mostly.I've gone back and forth on accepting solutions in other languages(e.g. Ruby, Perl, Python), and I've decided that candidates need to beable to code their answers in C, C++ or Java. It's wonderful if theyknow other languages, and in fact those who do tend to do *alot* better overall. But to be an Amazon SDE, you need toprove you can do C++ or Java first.

Example 1: *Write a function to reverse a string.*

Example Java code:

```
public static String reverse ( String s ) {
    int length = s.length(), last = length - 1;
    char[] chars = s.toCharArray();
    for ( int i = 0; i < length/2; i++ ) {
        char c = chars[i];
        chars[i] = chars[last - i];
        chars[last - i] = c;
    }
    return new String(chars);
}
```

Example output for "Madam, I'm Adam": madA m'I ,madaM

Example 2: *Write function to compute Nth fibonacci number:*

Java and C/C++:

```
static long fib(int n) {
    return n <= 1 ? n : fib(n-1) + fib(n-2);
}
```

(Java Test harness)

```
public static void main ( String[] args ) {
    for ( int i = 0; i < 10; i++ ) {
        System.out.print ( fib(i) + ", " );
    }
    System.out.println ( fib(10) );
}
```

(C/C++ Test Harness)

```
main () {
    for ( int i = 0; i < 10; i++ ) {
        printf ( "%d, ", fib(i) );
    }
    printf ( "%d\n", fib(10) );
}
```

Test harness output:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55

Example 3: *Print out the grade-school multiplication table up to 12x12*

Java: *(similar for C/C++)*

```
public static void multTables ( int max )
{
    for ( int i = 1; i <= max; i++ ) {
        for ( int j = 1; j <= max; j++ ) {
            System.out.print ( String.format ( "%4d", j * i ));
        }
        System.out.println();
    }
}
```

Example output:

```
   1   2   3   4   5   6   7   8   9  10  11  12
   2   4   6   8  10  12  14  16  18  20  22  24
   3   6   9  12  15  18  21  24  27  30  33  36
   4   8  12  16  20  24  28  32  36  40  44  48
   5  10  15  20  25  30  35  40  45  50  55  60
   6  12  18  24  30  36  42  48  54  60  66  72
   7  14  21  28  35  42  49  56  63  70  77  84
   8  16  24  32  40  48  56  64  72  80  88  96
   9  18  27  36  45  54  63  72  81  90  99 108
  10  20  30  40  50  60  70  80  90 100 110 120
  11  22  33  44  55  66  77  88  99 110 121 132
  12  24  36  48  60  72  84  96 108 120 132 144
```

Example 4: *Write a function that sums up integers froma text file, one int per line.*

Java:

```
public static void sumFile ( String name ) {
    try {
        int total = 0;
        BufferedReader in = new BufferedReader ( new FileReader ( name ));
        for ( String s = in.readLine(); s != null; s = in.readLine() ) {
            total += Integer.parseInt ( s );
        }
        System.out.println ( total );
        in.close();
    }
    catch ( Exception xc ) {
        xc.printStackTrace();
    }
}
```

Example 5: *Write function to print the odd numbers from 1 to 99.*

C/C++:

```
void printOdds() {
    for (int i = 1; i < 100; i += 2) {
        printf ("%d\n", i); // or cout << i << endl;
    }
}
```

Java:

```
public static void printOdds() {
    for (int i = 1; i < 100; i += 2) {
        System.out.println ( i );
    }
}
```

Example 6: *Find the largest int value in an int array.*

Java:

```
public static int largest ( int[] input ) {
int max = Integer.MIN_VALUE;
for ( int i = 0; i < input.length; i++ ) {
    if ( input[i] > max ) max = input[i];
    }
    return max;
}
```

Example 7: *Format an RGB value (three 1-bytenumbers) as a 6-digit hexadecimal string.*

Java:

```
public String formatRGB ( int r, int g, int b ) {
    return (toHex(r) + toHex(g) + toHex(b)).toUpperCase();
}

public String toHex ( int c ) {
    String s = Integer.toHexString ( c );
    return ( s.length() == 1 ) ? "0" + s : s;
}
```

Or in Java 1.5:

```
public String formatRGB ( int r, int g, int b ) {
    return String.format ( "%02X%02X%02X", r, g, b );
}
```

Example output for (255, 0, 128):

You can ask any question you like; doesn't have to be one ofthe ones above. They're just examples.

Some properties of a good weeder phone-screen coding question are:

1. *It's simple.* It has to be something that you should be able to solve, trivially, in about 2 minutes or less. Not too tricky. Basic stuff.
2. *You've solved it.* You shouldn't ask a question unless you've solved it yourself recently, so you know it's a reasonable question, and you can evaluate their answer to it. You should consider coding it yourself during the time you've given them to do it.
3. *It has loops or recursion.* Recursion is actually preferable. Being able to reason recursively or inductively is important for many areas of computing, including using heirarchical data representations (e.g. XML), distributed computing, searching, and sorting. Many candidates simply can't think recursively, and this often goes undetected until interview-time. Try to find out at compile-time! Er, phone-screen time, that is.
4. *It has formatted output.* This is a basic skill, useful for debugging, simple report generation, and lots of other things. "printf" is a universal standard; it exists in C, C++, Java, Perl, Ruby, Python, and virtually every other mainstream language, at *least* as a library call. Like file I/O, it's a good indicator as to whether the candidate has written "real" code before.
5. *It has text-file I/O.* Candidates who have worked in frameworks for too long often become unable to function as programmers outside that framework. Not being able to do simple file I/O is a common indicator that they've grown overly dependent on a particular framework.

It's hard to cover *all* these things and still bea short weeder question. If you think of a question thathas all these properties, let me know.

Area Number Two: Object-OrientedProgramming

We shouldn't hire SDEs (arguably excepting college hires)who aren't at least somewhat proficient with OOP. I'm notclaiming that OOP is good or bad; I'm just saying you haveto know it, just like you have to know the things you canand can't do at an airport security checkpoint.

Two reasons:

1) OO has been popular/mainstream for more than 20 years.Virtually every programming language supports OOP in some way.You can't work on a big code base without running into it.

2) OO concepts are an important building block for creatinggood service interfaces. They represent a shared understandingand a common vocabulary that are sometimes useful when talkingabout architecture.

So you have to ask candidates some OO stuff on the phone.

a) *Terminology*

The candidate should be able to give satisfactory definitionsfor a random selection of the following terms:

1. class, object (and the difference between the two)
2. instantiation
3. method (as opposed to, say, a C function)
4. virtual method, pure virtual method
5. class/static method
6. static/class initializer
7. constructor
8. destructor/finalizer
9. superclass or base class
10. subclass or derived class
11. inheritance
12. encapsulation

13. multiple inheritance (and give an example)
14. delegation/forwarding
15. composition/aggregation
16. abstract class
17. interface/protocol (and different from abstract class)
18. method overriding
19. method overloading (and difference from overriding)
20. polymorphism (without resorting to examples)
21. is-a versus has-a relationships (with examples)
22. method signatures (what's included in one)
23. method visibility (e.g. public/private/other)

These are just the bare basics of OO. Candidates shouldknow this stuff cold. It's not even a complete list; it'sjust off the top of my head.

Again, I'm not advocating OOP, or saying anything about it,other than that it's ubiquitious so you have to know it.You can learn this stuff by reading a single book andwriting a little code, so no SDE candidate (except maybea brand-new college hire) can be excused for not knowingthis stuff.

I draw a distinction between "*knows it*" and "*issmart enough to learn it.*" Normally I allow people throughfor interviews if they've got a gap in their knowledge,as long as I think they're smart enough to make it upon the job.

But for these five areas, I expect candidates to know them.It's not just a matter of being smart enough to learn them.There's a certain amount of common sense involved; I can'timagine coming to interview at Amazon and not having brushedup on OOP, for example. But these areas are also so fundamental that they serve as real indicators of how the person will do onthe job here.

b) *OO Design*

This is where most candidates fail with OO. They canrecite the textbook definitions, and then go on to producecertifiably insane class designs for simple problems. Forinstance:

- They may have Person multiple-inherit from Head, Body, Arm, and Leg.
- They may have Car and Motorcycle inherit from Garage.
- They may produce an elaborate class tree for Animals, and then declare an enum ("Lion = 1, Bear = 2", etc.) to represent the type of each animal.
- They may have exactly one static instance of every class in their system.

(All these examples are from real candidates I've interviewedin the past 3 weeks.)

Candidates who've only studied the terminology without everdoing any OOP often don't really get it. When they go toproduce classes or code, they don't understand thedifference between a static member and an instance member,and they'll use them interchangeably.

Or they won't understand when to use a subclass versus anattribute or property, and they'll assert firmly that acar with a bumper sticker is a subclass of car. (Yep,2 candidates have told me that in the last 2 weeks.)

Some don't understand that objects are supposed to knowhow to take care of themselves. They'll create a bunch ofclasses with nothing but data, getters, and setters (i.e.,basically C structs), and some Manager classes that containall the logic (i.e., basically C functions), and voila,they've implemented procedural programming perfectly usingclasses.

Or they won't understand the difference between a char*,an object, and an enum. Or they'll think polymorphism isthe same as inheritance. Or they'll have any number ofother fuzzy, weird conceptual errors, and their designswill be fuzzy and weird as well.

For the OO-design weeder question, have them describe:

1. What classes they would define.
2. What methods go in each class (*including signatures*).
3. What the class constructors are responsible for.
4. What data structures the class will have to maintain.
5. Whether any Design Patterns are applicable to this problem.

Here are some examples:

> Design a deck of cards that can be used for different card game applications.
>
> *Likely classes: a Deck, a Card, a Hand, a Board, and possibly Rank and Suit. Drill down on who's responsible for creating new Decks, where they get shuffled, how you deal cards, etc. Do you need a different instance for every card in a casino in Vegas?*
>
> Model the Animal kingdom as a class system, for use in a Virtual Zoo program.
>
> *Possible sub-issues: do they know the animal kingdom at all? (I.e. common sense.) What properties and methods do they immediately think are the most important? Do they use abstract classes and/or interfaces to represent shared stuff? How do they handle the multiple-inheritance problem posed by, say, a tomato (fruit or veggie?), a sponge (animal or plant?), or a mule (donkey or horse?)*
>
> Create a class design to represent a filesystem.
>
> *Do they even know what a filesystem is, and what services it provides? Likely classes: Filesystem, Directory, File, Permission. What's their relationship? How do you differentiate between text and binary files, or do you need to? What about executable files? How do they model a Directory containing many files? Do they use a data structure for it? Which one, and what performance tradeoffs does it have?*
>
> Design an OO representation to model HTML.
>
> *How do they represent tags and content? What about containment relationships? Bonus points if they know that this has already been done a bunch of times, e.g. with DOM. But they still have to describe it.*

The following commonly-asked OO design interview questions areprobably too involved to be good phone-screen weeders:

1. Design a parking garage.
2. Design a bank of elevators in a skyscraper.
3. Model the monorail system at Disney World.
4. Design a restaurant-reservation system.
5. Design a hotel room-reservation system.

A good OO design question can test coding, design, domain knowledge,OO principles, and so on. A good weeder question should probablyjust target whether they know when to use subtypes, attributes, andcontainment.

Area Number Three: Scripting andRegular Expressions

Many C/C++/Java candidates, even some with 10+ years ofexperience, would happily spend a week writing a 2,500-lineprogram to do something you could do in 30 seconds with asimple Unix command.

I now pose the following question to ALL candidates, whether onthe phone or in an interview, because it eliminates so many of them:

Last year my team had to remove all thephone numbers from 50,000 Amazon web page templates, since manyof the numbers were no longer in service, and we also wanted to routeall customer contacts through a single page.

Let's say you're on my team, and we have to identify the pageshaving probable U.S. phone numbers in them. To simplify the problemslightly, assume we have 50,000 HTML files in a Unix directory tree,under a directory called "/website". We have 2 days to get a list offile paths to the editorial staff. You need to give me a list of the.html files in this directory tree that appear to contain phonenumbers in the following two formats: (xxx) xxx-xxxx andxxx-xxx-xxxx.

How would you solve this problem? Keep in mind our team is on a short (2-day) timeline.

Here are some facts for you to ponder:

> Our Contact Reduction team really did have exactly this problem in 2003. This isn't a made-up example.
>
> Someone on our team produced the list within an hour, and the list supported more than just the 2 formats above.
>
> About 25% to 35% of all software development engineer candidates, independent of experience level,cannot solve this problem, even given the entire interview hour and lots of hints.

I take as much time as necessary to explain the problem to candidates, to ensure that they understand it and can paraphrase the problem requirements correctly.

For the record, I'm not being tricky here. Once candidates startdown the wrong path (i.e. writing a gigantic C++ program to openevery file and parse character by character, using a home-grownstate machine), I stop them, tell them this will take too long,and ask if there are any other possibilities. I ask if there areany tools or utilities that might be of use. I give them plentyof hints, and ultimately I tell them the answer.

Even after I tell them the answer, they often still don't get it.

Here's one of many possible solutions to the problem:

```
grep -l -R --perl-regexp "\b(\(\d{3}\)\s*|\d{3}-)\d{3}-\d{4}\b" * > output.txt
```

But I don't even expect candidates to get that far, really. Ifthey say, after hearing the question, "Um... grep?" then they'reprobably OK. I can ask them for the approximate syntax for theregular expression to use, and as long as they have a reasonable clue,I'm fine with it. Heck, if they can tell me where they'dlook to find the syntax, I'm fine with it.

They can also use find, or write aPerl script (or awk or bash or etc.). Anything that shows they haveeven the tiniest inkling of why Unix is Unix.

They can even write a Java or C++ program, provided they can actuallywrite an entire working program in, say, half an hour or less, on theboard, or at least convince me that they will get it working quickly.But I've only ever had that happen once; an insanely good C++programmer burned through a 175-line C++ program on the whiteboardthat more or less solved it. We made him an offer. But usually theythrow in the towel when they find out they have to remember how to dofile I/O, or traverse a directory tree.

For what it's worth, this failure mode is unique to Java andC/C++ programmers. Perl programmers laugh and solve it in30 seconds or less. I have some easy questions that makePerl programmers cry, but this isn't one of them.

In my experience, a programmer who only knows one language(where C and C++ count as one language for this exercise)is usually completely lost in one of these Five Essential Areas.

You don't necessarily have to ask the HTML phone-number question.Another one I used to ask, one that worked equally well, was:

Let's say you're on my team, andI've decided I'm a real stickler for code formatting.But I've got peculiar tastes, and one day I decide I wantto have all parentheses stand out very clearly in your code.

So let's say you've got a set of source files in C, C++, orJava. Your choice. And I want you to modify them so thatin each source file, every open- and close-paren has exactlyone space character before and after it. If there is anyother whitespace around the paren, it's collapsed into a single space character.

For instance, this code:

```
foo (bar ( new Point(x, graph.getY()) ));
```

Would be modified to look like this:

```
foo ( bar ( new Point ( x, graph.getY ( ) ) ) ) ;
```

I tell you (as your manager) that I don't care how yousolve this problem. You can take the code down to Kinko'sCopies and manually cut and paste the characters withscissors if you like.

How will you solve this problem?

Same thing, more or less. You'd do it with a Unix command like sed(using a regular expression), or do it in your editor using a regex,or write a quick Ruby script, whatever. I'd even accept having themuse a source-code formatter, provided they can tell me indetail how to use it, during the interview (to a level of detail thatconvinces me they've used it before.)

There are all sorts of variations on this problem. Generallyyou want to come up with a real-life scenario that involves searching text files for patterns, and see if the candidatewants to solve it by writing a giant chunk of C++ or Javacode.

Area Number Four: Data Structures

SDE candidates need to demonstrate a basic understanding of themost common data structures, and of the fundamentals of "big-O"algorithmic complexity analysis.

Here's what they need to know about big-O. They need to knowthat algorithms usually fall into the following performanceclasses: constant-time, logarithmic, linear, polynomial,exponential, and factorial.

For the standard data structures in java.util, STL, or those builtinto a higher-level language, they need to know the big-O complexityfor the operations on those data structures. Example: theyshould know that finding an element in a hashtable is usuallyconstant-time, that finding an element in a balanced binarytree is order log(n), that finding an element in a linked list isorder N, and that finding an element in a sorted array is order log(n).Similarly for insert/update/delete operations.

And they should be able to explain why each operation falls into aparticular complexity class. For instance: "Computing a hash valuedoesn't depend on the number of items in the hashtable." Or: "youhave to search the entire linked list, even if it's sorted, to find anarbitrary element in it." No math needed, no proofs, just explanations.

The (concrete) data structures they absolutely must understandare these:

1) arrays - I'm talking aboutC-language and Java-language arrays: fixed-sized, indexed, contiguousstructures whose elements are all of the same type, and whose elementscan be accessed in constant time given their indices.

2) vectors - also known as "growable arrays" or ArrayLists. Need to know that they're objects thatare backed by a fixed-size array, and that they resizethemselves as necessary.

3) linked lists - lists made of nodes that containa data item and a pointer/reference to the next (and possiblyprevious) node.

4) hashtables - amortized constant-time accessdata structures that map keys to values, and are backedby a real array in memory, with some form of collisionhandling for values that hash to the same location.

5) trees - data structures that consist of nodeswith optional data elements and one or more childpointers/references, and possibly parent pointers,representing a heirarchical or ordered set of data elements.

6) graphs - data structures that representarbitrary relationships between members of any data set,represented as networks of nodes and edges.

There are, to be sure, many other important data structures oneshould know about, but not knowing about the six listed above isinexcusable, and grounds for rejection in a phone screen.

Candidates should be able to describe, for any of the datastructures above:

- *what you use them for (real-life examples)*
- *why you prefer them for those examples*
- *the operations they typically provide (e.g. insert, delete, find)*
- *the big-O performance of those operations (e.g. logarithmic, exponential)*
- *how you traverse them to visit all their elements, and what order they're visited in*
- *at least one typical implementation for the data structure*

Candidates should know the difference between an abstract data typesuch as a Stack, Map, List or Set, and a concrete data structure such asa singly-linked list or a hash table. For a given abstract data type(e.g. a Queue), they should be able to suggest at least two possibleconcrete implementations, and explain the performance trade-offs betweenthe two implementations.

Example weeder questions:

1) What are some really common datastructures, e.g. in java.util?

2) When would you use a linked list vs. avector?

3) Can you implement a Map with a tree?What about with a list?

4) How do you print out the nodes of atree in level-order (i.e. first level, then 2nd level, then 3rd level,etc.)

5) What's the worst-case insertionperformance of a hashtable? Of a binary tree?

6) What are some options for implementinga priority queue?

And so on. Just a few quick questions should cover thisarea, provided you don't focus exclusively on linear orderedsequences (lists, arrays, vectors and the like).

Area Number Five: Bits and Bytes

This area is fairly contentious, at least inasmuch as peoplewho don't know this area claim you don't need to know it.

(Hint: that's true for everything. Nobody likes toadmit they don't know something you need to know. I'll start:I should know more about math; it's inexcusable. I'm doingall kinds of stuff the long, slow, dumb way because of my rustymath skills. But at least I admit it, and I've been studyingmy math books semi-regularly in an attempt to repair my skills.)

Candidates do need to know about bits and bytes, at least atthe level that I'm outlining here. Otherwise they're prone to havingan integer-overflow error in their code that brings the website downand costs us millions. Or spending a week trying to decode aserialized object they're debugging. Or whatever. Computers don'thave ten fingers; they have one. So people need to know this stuff.

Candidates should know what bits and bytes are. They shouldbe able to count in binary; e.g. they should be able to tellyou what 2^5 or 2^10 is, in decimal. They shouldn't stareblankly at you when you ask with 2^16 is. It's a special number.They should know it.

They should know at least the logical operations AND, OR,NOT, and XOR, and how to express them in their favorite/strongestprogramming language.

They should understand the difference between a bitwise-ANDand a logical-AND; similarly for the other operations.

Candidates should know the probable sizes of the primitivedata types for a standard 32-bit (e.g. Intel) architecture.

If they're a Java programmer, they should know exactly whatthe primitive types are (byte, short, int, long, float, double,char, boolean) and, except for boolean, exactly how much spaceis allocated for them per the Java Language specification.

Everyone should know the difference between signed and unsignedtypes, what it does to the range of representable valuesfor that type, and whether their language supports signedvs. unsigned types.

Candidates should know the bitwise and logical operatorsfor their language, and should be able to use them forsimple things like setting or testing a specific bit, orset of bits.

Candidates should know about the bit-shift operators in theirlanguage, and should know why you would want to use them.

A good weeder question for this area is:

Tell me how to test whether the high-orderbit is set in a byte.

Another, more involved one is:

Write a function to count all the bits in an int value;e.g. the function with the signature`int countBits(int x)`

Another good one is:

Describe a function that takes an int value,and returns true if the bit pattern of that int value is thesame if you reverse it (i.e. it's a palindrome); i.e.`boolean isPalindrome(int x)`

They don't have to code the last two, just convince you they'd take the right approach. Although if you have them code itcorrectly, it can count for your Coding weeder question too.

C/C++ programmers should know about the sizeof operatorand how (and why/when) to use it. Actually, come to think of it,everyone should know this.

All programmers should be able to count in hexadecimal, andshould be able to convert between the binary, octal, and hexrepresentations of a number.

Special Fast-Track Version

That's it for the Five Essential Phone Screen Questions. Hopeya liked it.

As a special reward for reading this far, here's a specialBonus Feature: a set of all-too-common answers that are almost always indicators of certain failure during ourinterviews. Even if I'm not on the loop!

Bad Sign #1:

```
Me:   So! What languages have you used, starting with your strongest?
Them: (briskly) C, C++.
Me:   (long, pregnant pause)
Them: (waiting patiently for me to continue)
Me:   Any others?
Them: Nope. C, C++.
```

Translation: (in thick Southern drawl) "We got both kinds of music here: country and western."

Probable failure modes for this candidate:

Will fail the HTML-phone-number question and the OO designquestion (but will get the OO terminology definitions mostly right.)

Bad Sign #1a:

```
Me:   So! What languages are you most familiar/proficient with?
Them: (worried) I've done mostly Java lately.
Me:   (long, pregnant pause)
Them: Yeah, um, Java.
Me:   Any others?
Them: Um, I did C in school a long time ago, but... pretty muchmostly Java now.
```

Translation: "Country and Western were both too hardcore for me. I got beat up in a bar."

Probable failure modes for this candidate: Will fail the bits and bytes questions, the HTML-phone-numberquestion, and most of the data structures questions.

Bad Sign #2:

```
Me:   So! What data structures do we have available to us,as programmers?
Them: Arrays, queues, vectors, stacks, lists, um, linked lists...
Me:   OK, any others?
Them: Um, doubly-linked lists, and, uh, array lists.
Me:   Have you ever used a tree?
Them: Oh! (laughs) Yeah, um, I forgot about those.
```

Translation: "My family tree doesn't branch."

Probable failure modes for this candidate: Very likely to fail data structures questions. Will fail anyrecursive problem, even a simple one like printing the elements of a linkedlist recursively. Will fail the HTML-phone-number question, sincethey obviously haven't ever used Perl if "hash" didn't leap to mind.

Bad Sign #3:

```
Me:   So! What the the primitive types in Java (or C++)?
Them: Ummmm, there's, um, int. And, uh, double.
Me:   Any others?
Them: Shoot, I'm drawing a blank right now. Um, String?
```

Translation: "C made my head hurt. Java is like sweet, sweet aspirin."

Probable failure modes for this candidate: Will fail bits and bytes questions, and probably just abouteverything else as well.

Bad Sign #4:

```
Me:   So! What text-editor do you use?
Them: Visual Studio.
Me:   OK. What about on Unix?
Them: On Unix I use vi.
Me:   Er, yeah, vi is cool... ever used VIM?
Them: No, just vi. Always worked just fine for me.
```

Translation: "Sometimes I type with my elbows when my hands are tired. It's just as fast."

Probable failure modes for this candidate:Will likely fail the HTML-phone-number question. Might pass theinterviews, but will need to be scheduled in geologic eras.

Bad Sign #5:

Me: So! What did you study in your Operating Systems class?

Them: Oh, that was a long time ago. I can hardly remember. Hehe.

Me: How long ago was it?

Them: 2 years.

Translation: "I want to use my MBA skills in a dynamic management role. When's lunch?"

Probable failure modes for this candidate: Will probably fail the coding question. Probably any OS questions, too.

All my Insta-Bad Signs above are cliches, in that I've heard theseanswers from at least 10 to 15 candidates (per question!), none ofwhom ever got an offer from us. I tend to ask questions like these asa matter of course now.

Summary

This stuff is the ABC's for programmers. Actually it'sonly going up through maybe J or K; it's not even halfwaythrough the alphabet. But most programmers out there in theBig Wide World will fail utterly in at least one of these areas.

Please cover all five areas if you're a phone screener. If you'rethe second screener, ask if you don't see evidence of themin the first screener's notes. (And then follow up andremind the first screener they should have asked these things.)

(Published Sep 28th 2004)

Comments

You can put a spin on your 'reverse a string' coding question - firsthave them write a func that prints out a C string without loopingconstructs or using local vars. Then if they get that, ask them toimplement a reverse string function in the same manner as the firstone. Don't say "use recursion" - let them figure out itsstraightforward applicability to the problem. That's, IMHO, how youcan gauge if they 'think recursively' when lightly nudged in thatdirection:

```
void print(char *s) {
  if (*s != 0) {
    putchar(*s);
    print(s+1);
  }
}

void printreverse(char *s) {
  if (*s != 0)
    printreverse(s+1);
    putchar(*s);
  }
}

int main() {
  char *s = "Hello world";
  print(s);
  putchar('\n');
  printreverse(s);
  putchar('\n');
}
```

Posted by: Martin N. at September 29, 2004 07:39 AM

Nice post Steve - thanks for the checklist.

Would you really accept this answer to the 'Write function to computeNth fibonacci number' question?

```
static long fib(int n) {
  return n <= 1 ? n : fib(n-1) + fib(n-2);
}
```

I'd hope that most candidates would know that (without memoization ofresults) the naive recursive solution is O(n!) in time. If they madethat error in production code it would be and utter disaster (probablylarge enough to be noticed early by QA, but still...).

If the candidate didn't at least mention this caveat about theirsolution, I'd prompt them to compare & contrast with alternatives. Ifthey didn't immediately give the iterative solution and explain thebig-O difference, that would be a red flag for me.

regards,

Chris

Posted by: Chris N. at September 29, 2004 08:36 PM

Er, a small error in my comment:

>>>I'd hope that most candidates would know that (withoutmemoization of results) the naive recursive solution is O(n!) intime.

I meant "is O(2^n) in time" of course.

Chris (blushing)

Posted by: Chris N. at September 29, 2004 08:39 PM

Yeah, that factorial fibo solution sucks. I'd be very happy if thecandidate told me that they could do it tail-recursively with anaccumulator parameter, even if I'm not sure you can do adoubly-recursive call tail-recursively. I'd still be happy.

I was thinking of splitting out recursion from basic coding, but thatwould be Six Essential Areas, and I only have five fingers.

Posted by: Steve Yegge at September 30, 2004 03:42 AM

Interesting post overall...a couple of comments/issues I've seen whendoing phone screens:

1. How do you have a candidate read code to you over the phone whenthe candidate isn't a native English speaker and the phone connectionis sub-optimal [I've had this more times than I can remember...]

2. I have some issues with the "scripting" category--it expresses apreference for hacky programmers who prefer speed overmaintainability. On our team (Customer Behavior) we're still cleaningup a fair amount of such code that broke the moment a database machinegot moved between data centers. Yes, the code was written quickly, butnow our managers are wondering why we can't get to a stable productionsystem so quickly. The section also prefers UNIX programmers over,say, people who worked with Windows for an entire career.

Posted by: Dan at October 7, 2004 01:44 AM

> 1. How do you have a candidate read code to you over the phone
> when the candidate isn't a native English speaker and the phone
> connection is sub-optimal [I've had this more times than I can
> remember...]

Me too.

The best approach I've seen is to give the candidate a "homework"question. Give them an hour to code up a solution to some problemand email it to you. Several teams are doing this regularly, withgood results.

> 2. I have some issues with the "scripting" category--it
> expresses a preference for hacky programmers who prefer speed
> over maintainability.

I'm sorry if I gave that impression. We don't want those kinds ofprogrammers here, obviously. The five question areas here are adelicate balance. This category is geared towards determining whethersomeone has the self-sufficiency to be able to respond quickly toemergencies affecting our customers. They still need to have goodjudgement and good design skills.

I never actually specify that they need to write it as a script. Ijust give them problems that are best handled that way: emergencyqueries and backfills, for example. I've had a few folks burn through200-line Java or C++ apps that solved the problem, right there on theboard, and they got offers.

But after 4 1/2 years over in Customer Service, I've found that knowinghow to use "grep" and its ilk is a pretty important survival skill.

And we -are- Unix shop, after all. Unix isn't exactly a nicheoperating system. There are people who've figured out the basics ontheir own, even if their professional experience has all been withMicrosoft technologies.

But feel free to ask whatever works for you!

Posted by: Steve Yegge at October 7, 2004 02:27 AM

The ultra-cool solution to Fib is the closed-form constant timesolution. You need floating point and exponentiation, but that'sconstant time on modern hardware.

There's also an off-by-one error of sorts in your printOdds()functions. i should start at 1, not 0, or the function should berenamed printEvens().

Posted by: Darren V. at October 7, 2004 02:42 AM

Thanks Darren. Obviously extra-cool solutions are bonus points for thecandidate.

Fixed the loop typo. It was caused, interestingly, when I corrected myoriginal working code, after someone emailed me and complained aboutthe performance. Originally it did this:

```
for (int i = 0; i < 100; i++) {
  if (i % 2 != 0) System.out.println(i);
}
```

The point was to see if the candidate could write something thatworked at all, and I was whipping up examples at 3:00am. Had no ideait would get so much attention.

In any case, someone objected to the fact that it wasn't simplyincrementing the loop counter by 2, so it performed poorly. So I wentand "optimized" it (an optimization that would go undetected by humansor profilers in this case, as it's totally I/O bound), and in myhaste, broke it.

I suppose I should claim that I rigged the whole thing as ademonstration of why optimizing stuff that doesn't matter isneedlessly risky. Or that I broke it so I could rant about why unittesting is critical, even for your personal blog content.

But really I just made a bad fix. :)

Thanks again.

Posted by: Steve Yegge at October 7, 2004 03:11 AM

Thanks for posting this. I have been trying to improve my interviewingskills, and this looks like a good set of basics for phone screens.

Posted by: Timothy K. at October 11, 2004 11:20 PM

Just an FYI...

I tried using the find-the-phone-numbers question on a candidateyesterday, and she went straight for Java. I was getting ready to slapher down when I noticed that Java 1.4 contains a regular expressionengine. With that enhancement, the write-a-complete-program solutionbecomes more reasonable.

Posted by: Christopher B. at November 17, 2004 09:37 AM

Yep. Most languages these days have (mostly) Perl5 compatible regexpengines. Java's file handling is a bit more cumbersome than it'd be ina scripting language, but not overly so. The problem is more aboutunderstanding fundamental pattern-matching tools than it is aboutscripting or any particular language.

Posted by: Steve Yegge at November 29, 2004 06:52 PM

I'm about to do my first phone screening and this was very helpful.

Maybe I just don't want to admit that I don't have vital information,but I don't know what 2^16 is. Jeff Bezos' phone number? I don'tunderstand the significance of knowing powers of two off the top ofone's head.

Posted by: Jason R. at March 22, 2005 10:58 PM

Jason: there are many domains for which knowing binary counting isuseful, if not essential.

One is when you're doing stuff that involves a lot of bit- andbyte-manipulation; examples include network protocols, writing binaryserialization/marshalling code, and reading or reverse-engineeringfile formats.

Another is when you're doing any sort of memory or pointer coding (ormore to the point, debugging) with C/C++ code. It's a lot easier tostare at hex-dumps if you're good at translating between decimal andhexadecimal (and binary).

Another broad class of problems involves data whose byterepresentation is especially significant. UTF-8 and Unicode are goodexamples. If you ever need to do internationalization, it can be a bighelp to have a crystal-clear understanding of the meaning of each bitin a byte, short, or int/word value. Actually, this may just beanother sub-example of the first domain I mentioned, but you get theidea.

Lastly (and most importantly; all the other examples I've listed paleto insignificance in comparison to this one), it's important foralgorithm time and space estimation. If you're trying to decide whatdata structure to use for a given data set, and you know off the topof your head that 2^16 is about 65,000, 2^20 is a million, 2^32 is 4billion, and 2^64 is "big enough", then you'll have an easier timewith the decision. Looking at it backwards -- if you use a balancedbinary tree (or a log-n search algorithm), you can quickly estimatethe base-2 logarithm of your data-set size, which gives you the roughnumber of steps involved in a lookup operation.

If you don't have a good feel for the growth rate of powers of2, then a little old man will save your daughter, and you'll grant himanything you want in your kingdom, and he'll say he just wants onegrain of rice on the first square of a chessboard, then 2 in thesecond, 4 on the third, and so on. And then: you'll be all out ofrice.

Fortunately, you can memorize this stuff in less time than it took youto post your comment. :)

Posted by: Steve Yegge at March 23, 2005 04:11 AM

---

*This is an *excellent* resource, Steve.*

I've noticed that a lot of the Java-steeped interviewees will useStrings in their string reverse solutions like so:

```
for (int i = s.length - 1; i >= 0; i--) {
    returnS += s.substring(i, 1);
}
```

Most will understand that objects are being allocated and discardedper loop, and will use a StringBuffer to make things "moreefficient". Interestingly, a couple of folks couldn't explain _why_ itwould be more efficient.

It's helpful to ask how they would design a limited StringBuffer classusing just primitive types to try to help dispel library addiction. Itcan be surprisingly intimidating for some folks, especially when itcomes to reallocating arrays. (Ironic, considering the allocation-festof their original solutions...)

Thus, I recommend it as another coding question -- though just to talkthrough, not to code over the phone. Alternatively, asking for thereverse() function using just primitives would do the same thing, butwithout the scariness of dreaded reallocation.

Posted by: Jeremy D. at March 29, 2005 12:54 AM