I was asked to stay away from HashMap or any sort of Hashing.

The question went something like this -

Lets say you have PRODUCT IDs of up to 20 decimals, along with Product Descriptions. Without using Maps or any sort of hashing function, what's the best/most efficient way to store/retrieve these product IDs along with their descriptions?

Why is using Maps a bad idea for such a scenario?

What changes would you make to sell your solution to Amazon?

A map is good to use when insert/remove/lookup operations are interleaved. Every operations are**amortized** in O(log n).

In your exemple you are only doing search operation. You may consider that any database update (inserting/removing a product) won't happen so much time. Therefore probably the interviewer want you to get the best data structure for lookup operations.

In this case I can see only some as already proposed in other answers:

* Sorted array (doing a binary search)
* Hasmap
* [trie](http://en.wikipedia.org/wiki/Trie)

With a trie , if product ids do not share a common prefix, there is good chance to find the product description only looking at the first character of the prefix (or only the very first characters). For instance, let's take that product id list , with 125 products:

* "1"
* "2"
* "3"  
  ...
* "123"
* "124"
* "1234567"

Let's assume you are looking for the product id titled "1234567" in your trie, only looking to the first letters: "1" then "2" then "3" then "4" will lead to the good product description. No need to read the remaining of the product id as there is no other possibilities. Considering the product id length as n , your lookup will be in O(n). But as in the exemple explained it above it could be even faster to retreive the product description. As the procduct ID is limited in size (20 characters) the trie height will be limited to 20 levels. That actually means you can consider the look up operations will never goes beyond a constant time, as your search will never goes beyong the trie height => O(1). While any BST lookups are at best **amortized O(log N), N being the number of items in your tree** .

While an hashmap could lead you to slower lookup as you'll need to compute an index with an hash function that is probably implemented reading the **whole** product id length. Plus browsing a list in case of collision with other product ids.

Doing a binary search on a sorted array, and performance in lookup operations will depends on the number of items in your database.

Suggest a data structure which can optimize all 3 below operations :

1. Insertion
2. Deletion
3. Return a random value from set of existing values

Assume that you are putting integers/numbers.

Dynamic Array + Hashmap = O(1) for all three operations

* Insertion

Append to the tail of Array O(1), and add a value-index pair to Hashmap O(1).

* Deletion

Find the index by the value in Hashmap O(1), delete by the index in array and swap the last element into the slot O(1), and update the value-index pair for the (used-to-be) last element O(1).

* Return random

Generate a random index for the array O(1).

I am stuck in one interview question.. The question is,

\*given two arrays A and B. A has integers unsorted. B has the same length as A and its values are in the set {-1,0,1}

you have to return an array C with the following processing on A.

if B[i] has 0 then C[i] must have A[i]  
if B[i] has -1 then A[i] must be in C within the sub array C[0] - C[i-1] ie. left subarray  
if B[i] has 1 then A[i] must be in C within the sub array C[i+1] - C[length(A)] ie right subarray.

if no such solution exists then printf("no solution");\*

I suggest the following algorithm:   
1. Initially consider all C[ i ] as empty nests.  
2. For each i where B[ i ] = 0 we put C[ i ] = A[ i ]  
3. Go through array **from left to right**, and for each i where B[ i ] = -1 put   
C[ j ] = A[ i ], where 0 <= j < i is the **smallest** index for which C[ j ] is still empty. If no such index exists, the answer is impossible.  
4. Go through array **from right to left**, and for each i where B[ i ] = 1 put   
C[ j ] = A[ i ], where i < j < n is the **greatest** index for which C[ j ] is still empty. If no such index exists, the answer is impossible.

Why do we put A[ i ] to the leftmost position in step 2 ? Well, we know that we **must** put it to some position j < i. On the other hand, putting it leftmost will increase our changes to not get stucked in step 3. See this example for illustration:

A: [ 1, 2, 3 ]

B: [ 1, 1,-1 ]

Initially C is empty: C:[ \_, \_, \_ ]  
We have no 0-s, so let's pass to step 2.  
We have to choose whether to place element A[ 2 ] to C[ 0 ] or to C[ 1 ].  
If we place it **not** leftmost, we will get the following situation:  
C: [ \_, 3, \_ ]   
And... Oops, we are unable to arrange elements A[ 0 ] and A[ 1 ] due to insufficient place :(   
**But**, if we put A[ 2 ] leftmost, we will get   
C: [ 3, \_, \_ ], And it is pretty possible to finish the algorithm with  
C: [ 3, 1, 2 ] :)  
  
**Complexity**:   
What we do is pass three times along the array, so the complexity is O(3n) = O(n) - linear.   
  
**Further example:**

A: [ 1, 2, 3 ]

B: [ 1, -1, -1 ]

Let's go through the algorithm step by step:  
1. C: [ \_, \_, \_ ]  
2. Empty, because no 0-s in B  
3. Putting A[ 1 ] and A[ 2 ] to leftmost empty positions:

C: [ 2, 3, \_ ]

4. Putting A[ 0 ] to the rightmost free (in this example the only one) free position:

C: [ 2, 3, 1 ]

Which is the answer.

I found this question on the web.

Given a stack S, write a C program to sort the stack (in the ascending order). We are not allowed to make any assumptions about how the stack is implemented. The only functions to be used are:

Push

Pop

Top

IsEmpty

IsFull

I think we can build heap and sort it. What is optimal solution to this?

Assuming that the only data structure allowed here is the Stack, then you could use 2 Stacks.

Iterate until the original stack is empty and in each iteration, pop an element from the original stack, while the top element in the second stack is bigger than the removed element, pop the second stack and push it to the original stack. Now you can push the element you originally popped off the original stack to the second stack.

The time complexity of this approach is O(N^2).

I am trying to implement LRU Cache using C++ . I would like to know what is the best design for implementing them. I know LRU should provide find(), add an element and remove an element. The remove should remove the LRU element. what is the best ADTs to implement this For ex: If I use a map with element as value and time counter as key I can search in O(logn) time, Inserting is O(n), deleting is O(logn).

One major issue with LRU caches is that there is little "const" operations, most will change the underlying representation (if only because they bump the element accessed).

This is of course very inconvenient, because it means it's not a traditional STL container, and therefore any idea of exhibiting iterators is quite complicated: when the iterator is dereferenced this is an access, which should modify the list we are iterating on... oh my.

And there are the performances consideration, both in term of speed and memory consumption.

It is unfortunate, but you'll need some way to organize your data in a queue (LRU) (with the possibility to remove elements from the middle) and this means your elements will have to be independant from one another. A std::list fits, of course, but it's more than you need. A singly-linked list is sufficient here, since you don't need to iterate the list backward (you just want a queue, after all).

However one major drawback of those is their poor locality of reference, if you need more speed you'll need to provide your own custom (pool ?) allocator for the nodes, so that they are kept as close together as possible. This will also alleviate heap fragmentation somewhat.

Next, you obviously need an index structure (for the cache bit). The most natural is to turn toward a hash map. std::tr1::unordered\_map, std::unordered\_map or boost::unordered\_map are normally good quality implementation, some should be available to you. They also allocate extra nodes for hash collision handling, you might prefer other kinds of hash maps, check out [Wikipedia's article](http://en.wikipedia.org/wiki/Hash_table) on the subject and read about the characteristics of the various implementation technics.

Continuing, there is the (obvious) threading support. If you don't need thread support, then it's fine, if you do however, it's a bit more complicated:

* As I said, there is little const operation on such a structure, thus you don't really need to differentiate Read/Write accesses
* Internal locking is fine, but you might find that it doesn't play nice with your uses. The issue with internal locking is that it doesn't support the concept of "transaction" since it relinquish the lock between each call. If this is your case, transform your object into a mutex and provide astd::unique\_ptr<Lock> lock() method (in debug, you can assert than the lock is taken at the entry point of each method)
* There is (in locking strategies) the issue of reentrance, ie the ability to "relock" the mutex from within the same thread, check Boost.Thread for more information about the various locks and mutexes available

Finally, there is the issue of error reporting. Since it is expected that a cache may not be able to retrieve the data you put in, I would consider using an exception "poor taste". Consider either pointers (Value\*) or [Boost.Optional](http://www.boost.org/doc/libs/1_44_0/libs/optional/doc/html/index.html) (boost::optional<Value&>). I would prefer Boost.Optional because its semantic is clear.

LRU cache implementation

http://timday.bitbucket.org/lru.html

I was preparing for technical interviews and would like to know how could I go about briefly explaining an interviewer about the approach to designing the following programs without going into unnecessary details

1. Program that lets people play tic tac toe with each other over the internet?

2. A suitable data structure for a photo editor or text editor? Why?

3. Implementing code to operate the elevators?

4. Implementing the rendering engine of a web browser?

I could come up with the following:

1. There are primarily 2 parts to the program. One deals with the rendering of the board display. The other is the actual engine containing methods to find the winning move, AI, game completion, etc. Then, there will be the code related to the network.
2. A linked list of strings seems appropriate for the text editor, since the primary element is text and related manipulations. Does a linked list of objects seem right for the photo editor?
3. This again involves a display rendering of the elevators along with resource synchronization.
4. Don't have much clue about how to approach this one.

Are the sufficient enough? Am I missing some important details in approaching these problems.

The photo editor is an interesting one.

You need to consider the requirements of a photo editor:

* rapid selection, decode and display of part of a multi-megabyte data structure (16M?+)
* responsive display of user operation of paint tools - the brush should not lag, even if brushing over the whole image
* can you efficiently operate paint tools when zoomed out?
* save to common formats efficiently
* consider impact of all this data on CPU cache and bandwidth - note that accessing main memory can take 100s of times longer than accessing cache. How do you make most operations happen in the cache?
* what common features do existing editors have - undo, layers, effects, real-time blend modes, gamma conversion at the final display pass

I suggest holding multiple zoom levels with lower resolution and bit-depth (like mipmaps) in memory at once to allow efficient operation at different zoom levels, encoding changes as patches to the base image, not directly editing the base (i.e. layer every change separately). The patches could be encoded as a grid with only painted squares having memory allocated to them. Edits can be encoded as actions (e.g. tool, mouse down, movement, mouse up) before they are rendered into pixels - this allows a cheap paint to be done as patches at the current mipmap level and quickly displayed, while a background thread creates patches at the other mipmap levels.

Elevator scheduling is achieved by [SCAN](http://en.wikipedia.org/wiki/Elevator_algorithm) algorithm. Its also used for disk scheduling.

Say, there are two lists of different lengths, **merging at a point**; how do we know where the merging point is?

Conditions:

1. We don't know the length
2. we should parse each list only once.

* by "modification is not allowed" it was meant "you may change but in the end they should be restored", and
* we could iterate the lists exactly *twice*

the following algorithm would be the solution.

First, the numbers. Assume the first list is of length a+c and the second one is of length b+c, wherec is the length of their common "tail" (after the mergepoint). Let's denote them as follows:

x = a+c

y = b+c

Since we don't know the length, we will calculate x and y without additional iterations; you'll see how.

Then, we iterate each list and reverse them while iterating! If both iterators reach the merge point at the same time, then we find it out by mere comparing. Otherwise, one pointer will reach the merge point before the other one.

After that, when the other iterator reaches the merge point, it won't proceed to the common tail. Instead will go back to the former beginning of the list that had reached merge-point before! So, before it reaches the end of the changed list (i.e. the former beginning of the other list), he will make a+b+1iterations total. Let's call it z+1.

The pointer that reached the merge-point first, will keep iterating, until reaches the end of the list. The number of iterations it made should be calculated and is equal to x.

Then, this pointer iterates back and reverses the lists again. But now it won't go back to the beginning of the list it originally started from! Instead, it will go to the beginning of the other list! The number of iterations it made should be calculated and equal to y.

So we know the following numbers:

x = a+c

y = b+c

z = a+b

From which we determine that

a = (+x-y+z)/2

b = (-x+y+z)/2

c = (+x+y-z)/2

Which solves the problem.

I need something like a bounded queue where I can only insert say 10 elements. A new element should override the last inserted element. **There should be only n distinct elements**

ITYM an implementation of a LRU algorithm.

* [How to set up a simple LRU cache using LinkedHashMap](http://java-planet.blogspot.com/2005/08/how-to-set-up-simple-lru-cache-using.html)
* [Simple LRU Caching with Expiration](http://www.codeproject.com/KB/java/lru.aspx)

[**Google Interview : Find the maximum sum of a polygon [closed]**](http://stackoverflow.com/questions/14412112/google-interview-find-the-maximum-sum-of-a-polygon)

<http://stackoverflow.com/questions/14412112/google-interview-find-the-maximum-sum-of-a-polygon?rq=1>

Another interesting interview question that I stumbled upon -

Design the data structures for a very large social network (Facebook, LinkedIn, etc)?

Also, design an algorithm to show the connection, or path, between two people (e.g. me->foo->bar->rob->ron)

I would probably consider an undirected graph of some variety, probably stored as a sparse adjacency matrix. As far as finding the shortest path between two people, since the cost of edges is uniform, I would consider going with a bidirectional search.

Basically, go out in concentric circles centered around each person, where each circle is the person himself, then his friends, then his friends-of-friends, etc., at each step testing if there is any person in both circles. Follow the path from the first person you find inward toward the center of each person, and you've found the shortest path.

You could try other shortest-path algorithms, but in general, most shortest-path algorithms only give you the distance and not the actual path.

<http://stackoverflow.com/questions/1953080/good-algorithm-and-data-structure-for-looking-up-words-with-missing-letters?rq=1>

I enjoy the classic "what's the difference between a LinkedList and an ArrayList (or between a linked list and an array/vector) and why would you choose one or the other?"

The kind of answer I hope for is one that includes discussion of:

* insertion performance
* iteration performance
* memory allocation/reallocation impact
* impact of removing elements from the beginning/middle/end
* how knowing (or not knowing) the maximum size of the list can affect the decision

Here are detailed instructions: [chaoticjava.com/posts/linkedlist-vs-arraylist](http://chaoticjava.com/posts/linkedlist-vs-arraylist/) – [Akshar Prabhu Desa](http://stackoverflow.com/users/54082/akshar-prabhu-desai" \o "550 reputation)

1. Please write an abstract factory method, that doesn't contain a switch and returns types with the base type of "X". (Looking for patterns, looking for reflection, looking for them to not side step and use an if else if)
2. Please split the string "every;thing|;|else|;|in|;|he;re" by the token "|;|".(multi character tokens are not allowed at least in .net, so looking for creativity, the best solution is a total hack)

I like the Project Euler problem that asks to find the most expensive path down a tree (16/67); common ancestor is a good warm up, but a lot of people have seen it. Asking somebody to design a tree class, perform traversals, and then figure out from which traversals they could rebuild a tree also gives some insight into data structure and algorithm implementation. The Stern-Brocot programming challenge is also interesting and quick to develop on a board

http://stackoverflow.com/questions/7425400/design-a-system-supporting-massive-data-storage-and-query

<http://stackoverflow.com/questions/6094293/amazon-interview-question>

<http://stackoverflow.com/questions/1538589/how-to-sort-all-possible-words-out-of-a-string/1539336#1539336>

<http://stackoverflow.com/questions/10017808/best-data-structure-for-dictionary-implementation/10017860#10017860>

<http://stackoverflow.com/questions/13014867/finding-a-sub-array-where-every-pair-has-sum-greater-than-a-given-k>

<http://stackoverflow.com/questions/5000836/selection-algorithms-on-sorted-matrix>