# 1\ Tips

## 1.1 pre-judgement for simple optimization.

e.g. determine a string has unique characters.

decide two string are permutation.

## 1.2 if the distinct cases are limited, use array instead of map.

## 1.3 check character encoding.

1.4 Examplify. For huge and abstract problem, study its base case and use base case to build a larger case until problems can be solved.

# 2\ Data Structure

## 2.1 Stack and Queue

e.g. use two stacks to implement queue: lazy solution. Simulate the usage situaion and optimize.

## 2.2 Hash Table

哈希表（Hash Table）是一种根据关键字直接访问内存存储位置的数据结构。HS is not for permanent store. It’s just about when you load data into memory, how could you find it as fast as possible.

# 3\ Algorithms

## 3.1 Calm down and really analyze it.

If you meet problems, solve it instead of stop thinking or move back.

e.g. find the next node in binary search tree.

## 3.2 Bit operation

try not to use while. Use more simple expression n <<= (zeros+ones); n |= (1 << ones )- 1;

bit operation: xor(how many different bits in a and b) ~ -1 << >> (\*2 /2)

# 4\ Others

## 4.1 Test

Make the problem/object very specific by asking questions: who are going to use it for what purpose? What functionality does it have? children? What age? Drawing? On what? How will you solve it in real world(don’t think about it’s an interview)

Design plans: Think about use cases. Breaking down big object/problem into small pieces. If object/software has different functionality, test each step(ATM: log in, withdraw, deposit, transfer, check balance). For object, fact check / intended usage / unintended usage / other aspects: safety, comfort, etc. When testing, clarify what aspect do you care about. For functions, normal input, extreme input, wrong input, etc.

1. Test a function:
   1. syntax test: size and type
   2. functional test:
      * 1. normal case: more detail according to your implementation , odd, even, etc
        2. extreme case: empty, large, small?
        3. Null and illegal: negative, overflow, null?
2. Troubleshooting Questions:
   1. Understand the scenario, get a deeper understand of what is happening
      * 1. Who is using it? What are they doing about it? How long has user been experiencing the issue?
        2. How often does it happen?
        3. What’s the version of it? What’s the os platform?
        4. Is there any error report? Any other information?
   2. Break down the problem. Find out the reasons step by step.

e.g How would you load test a webpage without using any test tools?(do not think about the detailed knowledge, use logical and structured thinking to solve it.)

1\ to load test a webpage, basically we are supposed to give the webpage from a minimum value to the maximum value we expected or until it crashes. The load should increase unit by unit. We are supposed to record its behavior for each load.

2\ the behavior should include: response time, throughput, resource utilization, etc.

3\ Because we have no test tools, we can develop our own tools. We use a multithread process to simulate lots of people load the page at the same time.

## 4.2 Brain Teaser

(1) Examplify: if you are given an abstract situation, or a situation with many states, or statement with n, just exemplify it by make detailed example, study one of its cases. Make some assumption to simplify the situation. Suppose there are XXX, Suppose at the beginning we have XXX.

(2) Generalize: if you are given detailed number, use x and y to represent it and study its general rule.

(3) Try to abstract some rules from your detailed example. Like dropping eggs. If you meet problems, declare your situation again to make sure you make full use of every condition and understand them right.

## 4.3 Scalability and Memory Limits

(1) Data is too large and has to be stored among interconnected machines. The key problem is how to divide data into different machine and how to find data afterwards

a. by order of appearance b. by hash table c. by relation d. random

(2) usually we divide big size problem into small similar problems and merge the result.

(3) Memory is 4G. Each int is 1byte. Maximum 1G(1 billion) ints can be stored in memory.

(4) First, simplify the problem without memory limits. Second, get real. Often you need to split the data into different machines or only load part of data once. Decide how you will split the data and how to merge results from multiple machines. E.g find documents containing list of words.

# 5\ Good Examples to Think Twice

## 5.1 Find all documents that contain a list of word

There is thinking flow of this problem. You propose a solution, analyze its performance and try to improve it using another method.

a. A naïve solution is to deal with documents one by one. Iterate through one document blablabla… If the average size of document is N, it takes O(N) time complexity and O(x) space.

b. However, the problem about this solution is that if we change the list, XXXXX

c. So we maintain a hash table and iterate through all documents. The hash value is word, the key is which document it appears. Now we get a list of words, we just need to find the numbers which appear in all the values.

d. How could this problem be solved? Bit operation &.

e. finally, the time complexity is , the space is. The advantage is that if list changed, we are able to ……

f. now the problem is millions of documents: the issue comes up with the situation is that we may not able to hold our documents in one machine. Or even hash table is to large to fit in one machine. We can solve it by either divide hash tables or divide documents.

h. Then each machine handles a small part of the data. Then we merge the results from each machine.

## 5.2 Find missing integers

suppose there are 2^31-1 distinct integers in range[0, 2^31-1]. Find the missing integers. what if the memory is 1GB? What if the memory if 10MB?

1GB is 8billion bits. So each bit can stand for a number. Use bit array(define by yourself) to solve problems. There are 2^31 numbers so we need 4billion bits. But now we only hae 80\*2^10bits. So divide into buckets and iterate twice.