**LeetCode goodies**

My LeetCode Journey and Personal Training Program

Hello everyone.  
I am writing this post in order to share my personal LeetCode training program and my journey in coding with LeetCode. First of all, I want to thank people who created this platform with an amazing design and interesting problems, great community, for creating an environment where there are so many amazing people who are open to help with code, problem understanding, etc.

My journey began with my friend's invite. I love solving coding problems and I always practiced using different websites. Once my friend asked me to practice and prepare for interviews together and showed me LeetCode. At the beginning, I did not clearly understand how to run and send problems because I always wrote the code from beginning till the end, while on LeetCode the main program was already built and we needed to write several functions. Then I loved this platform and began practicing everyday. My initial level was a beginner and I tried to reach the intermediate level (to be able to solve medium and hard problems). You may ask how I understood my level and the answer is that I was able to solve only easy problems and was not able to solve medium problems and that is why I thought that my level is a beginner. Then I created the rules and schedule for my training.

YES, TRAINING BECAUSE WE TRAIN OUR PROBLEM-SOLVING SKILLS, OUR BRAIN AND LOGIC !!!

RULE NUMBER 1: SET CLEAR GOALS AND CREATE YOUR TRAINING PROGRAM AND KEEP TRAINING.  
Setting clear goals and creating the right training program will really help you to see the objectives of what you are going to do. I grouped many data structures and algorithms topics and created this program:

1. Recursion and backtracking. (1-2 weeks)
2. Two pointers. (1 week)
3. Linked List (1 week)
4. Stack (1-2 weeks)
5. Queue (1 week)
6. Tree (2-3 weeks)
7. Hash Table and Map (2-3 weeks)
8. Sorting and Heap. (2-3 weeks)
9. Binary Search. (1-2 weeks)
10. Greedy (2-3 weeks)
11. Dynamic Programming. (3-4 weeks)
12. Graphs, BFS, DFS. (4-5 weeks)
13. Strings and trie. (3-4 weeks)

First of all, I began my journey in the mid of September and tried to practice LeetCode and combine with studies. At the beginning everything was okay but then my studies become harder and there were a lot of exams and assignments at university so that I made a pause in coding. DO NOT DO LIKE ME. TRY TO SOLVE AT LEAST ONE PROBLEM EVERYDAY AND REMEMBER THAT SMALL THINGS MAKE BIG THINGS. Later I returned to my journey and said myself that NO MATTER WHAT WILL HAPPEN I WILL SOLVE AT LEAST ONE PROBLEM EACH DAY and from 23rd of December I am solving at least one problem everyday.

RULE NUMBER 2: THEORY IS REALLY IMPORTANT AND YOU NEED TO UNDERSTAND HOW THE DATA STRUCTURES AND ALGORITHMS WORK AND HOW YOU CAN APPLY THEM.  
Now let's discuss how I studied theory. I always watch lectures on YouTube and try to watch for a particular topic as many lectures as I can. Understanding what particular data structure or algorithm does is very important in applying it in solving problems. Here I think all YouTube amazing channels which I watched and continue to watch:

1. Jenny's Lectures.
2. MIT Algorithms.
3. mycodeschool.
4. Abdul Bari Algorithms.

There's also other amazing websites that can help you on your journey.

These sources helped and continue to help me to understand all the concepts of data structures and algorithms which I listed above and I want to also thank these people for sharing their knowledge with us.

RULE NUMBER 3: AFTER LEARNING THE THEORY OF PARTICULAR TOPIC GO TO LEETCODE, OPEN THE PROBLEMS OF THAT TOPIC AND TRY TO SOLVE AS MANY PROBLEMS AS YOU CAN.  
I understood on myself how practicing makes a real difference. I sorted all problems by difficulty level and clicked to "show problem tags". When I solved problems I began with easy problems and skip problems with the topics which I did not know. For example, when I solved Linked List problems I saw that one problem has also Sorting as a relevant topic and I skipped this problem and tried to solve it later when I will learn Sorting. This approach really helped me to solve problems faster and in an efficient way. I began to solve almost all easy problems, many medium problems, and some hard problems.

In conclusion, currently, I can easily solve many easy and medium problems and can solve some hard problems. I am now on Greedy topic and I am continuing with my program. I think that I reached the intermediate level and set a new goal of becoming advanced. My initial goal was to solve 100 problems, then 200, and then 300 and know I solved 313 problems in a small period I think. I want to reach 1000 solved problems and hope I will make another post when I will reach 1000 problems. I hope my personal experience and sources with which I shared will help many people in their LeetCode journey. I wish everyone to get an offer from the companies which you want and REMEMBER THAT EVERYONE CAN DO EVERYTHING, ACHIEVE EVERYTHING BY HARD WORK. KEEP PRACTICING AND BE LOYAL TO YOUR DREAMS. THANKS !

##### A noob's guide to Dijkstra's Algorithm

If you have ever wondered about how Dijkstra's algorithm works or what the intuition behind it is then you might have something to learn here, hopefully. I shall try to explain my understanding of the Dijkstra's algorithm. If some errors have crept in, let me know.

Dijkstra's algorithm is a **Single-Source-Shortest-Path algorithm**, which means that it calculates shortest distance from one vertex to all the other vertices.

The node from where we want to find the shortest distance is known as the **source node**. For the rest of the tutorial, I'll always label the source node as **S**. And the distance to source node is naturally going to be **zero**.

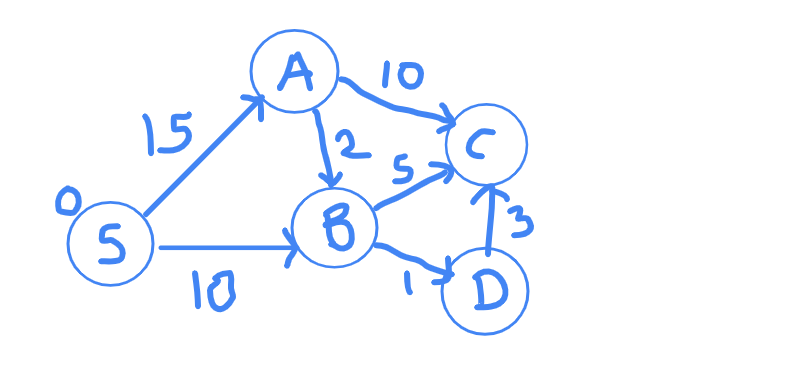
So, if we had an array of distances called dist[] and source node is 2 then dist[] would be

[INF, INF, 0, INF, INF]

0 1 2 3 4

// Consider INF as infinity since we don't know what the actual distance is. But we are very much sure that the distance to the source node is 0.

Let's start.

Have a look at the image below. We shall work with it in this tutorial.  


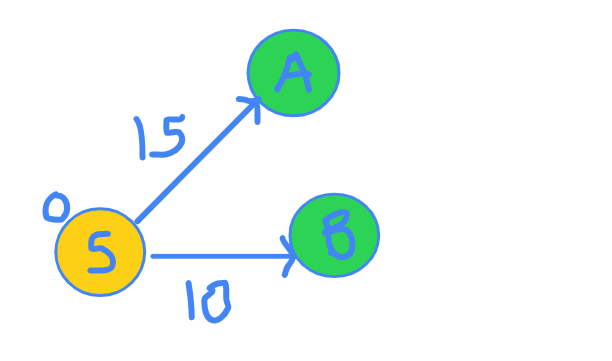
Remember what I said about the source node? I said that we were **SURE** that the shortest distance to source node is going to be zero. Do you agree with me on that point? Of course, yes, because there was no path to travel.

What does the distance array for our graph look like?

Node | S A B C D

Distance | 0 INF INF INF INF

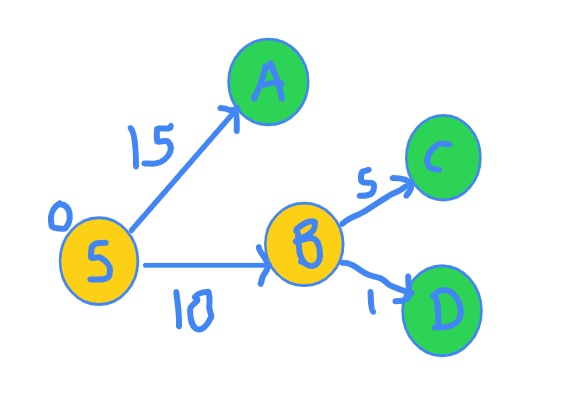
Let's break the problem. Let's rip apart this graph.

We are at S currently. Let's see where we can go from S.  


You are at node S. There are 2 edges that come out of S. One edge goes to A and other goes to B. Edge SA distance is 15 and edge SB distance if 10. You can clearly verify this in the figure.

I now claim that I am SURE that the shortest distance to B is 10. Why though? How many other ways are possible, through which you can reach to node B? Maybe there will be some edge from A to B. But since distance to A is already 15, any edge from A to B will only increase that distance and will never be able to become less than 10. This is the most important step. Read it again if you did not get it.

Now, that I am SURE of the shortest distance to reach B, I discover more vertices from B.

Now, our exploration graph looks something like this.  
  
and the distance array has been updated.

Node | S A B C D

Distance | 0 INF 10 INF INF

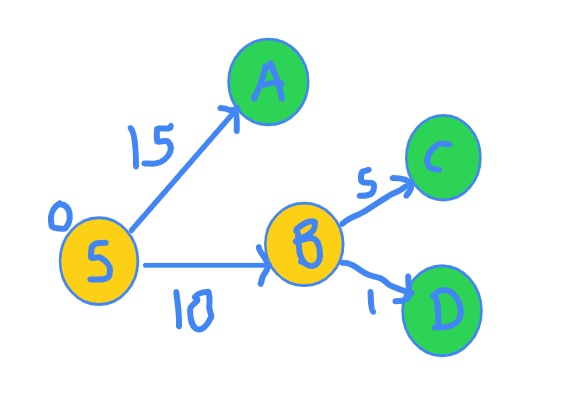
Let's see what the current scenario looks like.

Nodes for which we know the shortest distance: S and B.

What are we looking at now?

At all the nodes we can reach from the S and B.

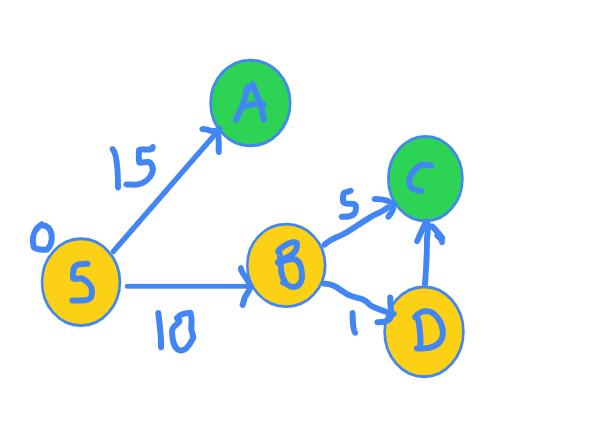
Again, the same question. Which node's shortest distance can we be SURE of ?

I claim it's node D.  
Look at the figure again.  
  
The distance to D is 10 + 1 = 11, and no other path can have length smaller than this since their own distances are greater than 11.

Hence, we now include D in our set of vertices for which shortest distances is known. Also, we explore more edges going from node D.

Note that we are always picking the green vertex with the minimum distance.

So, now our exploration graph looks something like this.

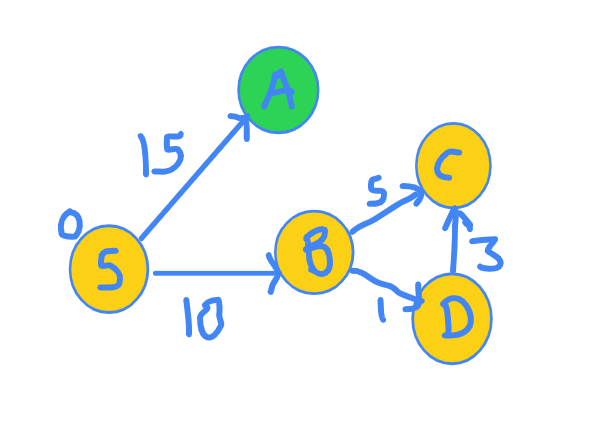
  
Distance has been updated.

Node | S A B C D

Distance | 0 INF 10 INF 11

Now, same question. Which node's shortest distance can we be SURE of? More specifically. Which green node's shortest distance can we be SURE of? Yes, right. It is C because the path S-B-D-C gives distance of 10 + 1 + 3 = 14 which is the lowest.

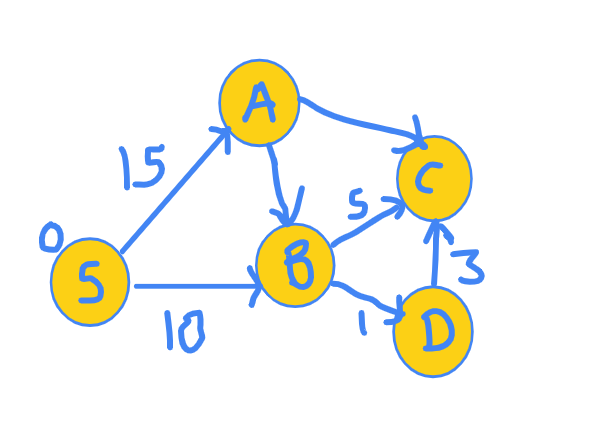
Hence, we include C into the set of nodes for which final distance have been finalized and we also explore it's neighbours.

The exploration graph now looks like this.  
  
Distance has been updated again.

Node | S A B C D

Distance | 0 INF 10 14 11

Again, for which node are we SURE of? There is just one node remaining and so it is the shortest. So, we include it in the set of nodes whose final distances are known and explore it's neighbours.

Now, the exploration graph is as follows:  


Final distance array now looks like this:

Node | S A B C D

Distance | 0 15 10 14 11

But now, since we have no green nodes, we stop. We have visited all the nodes.

Couple of key points here. How were we SURE of the shortest distance of a green node? That node had the shortest distance among all the other green nodes.

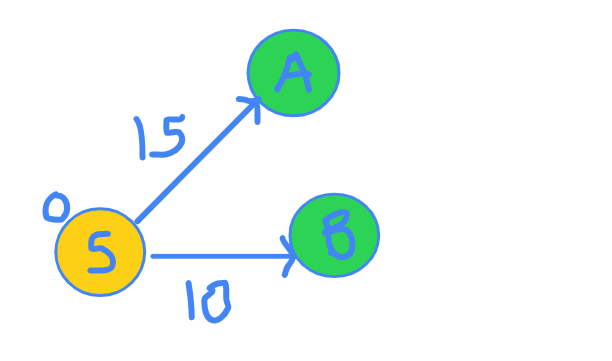
**Now, summarizing Djikstra's algorithm:**

1. We keep a set of vertices for which final shortest distance is already known to us. Initially only the source vertex S belongs to this set.
2. We do several iterations, during which we pick the green node with the minimum distance and add it to the set of vertices whose distances are finalized and also all nodes reaching from this node and not VISITED (i.e. NOT MARKED YELLOW), are made green.

And that's it. That's the working of Djikstra's algorithm.

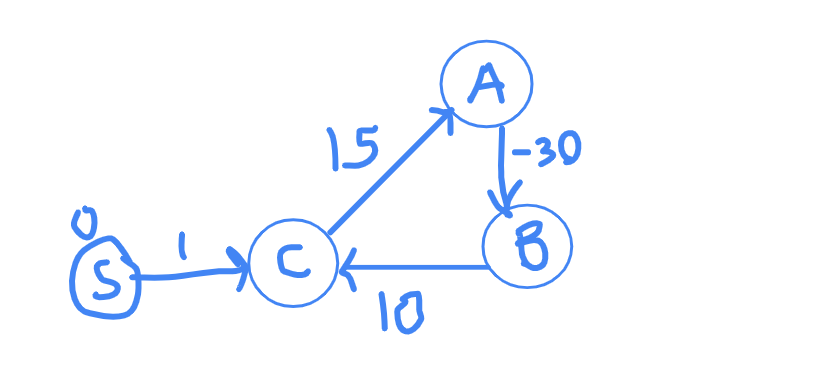
Regarding the implementation, since we need the node with minimum distance, a min-heap is prefered. Implementations with sets( the ones based on BSTs) are also popular. You can find the implementations online with both the data structures.

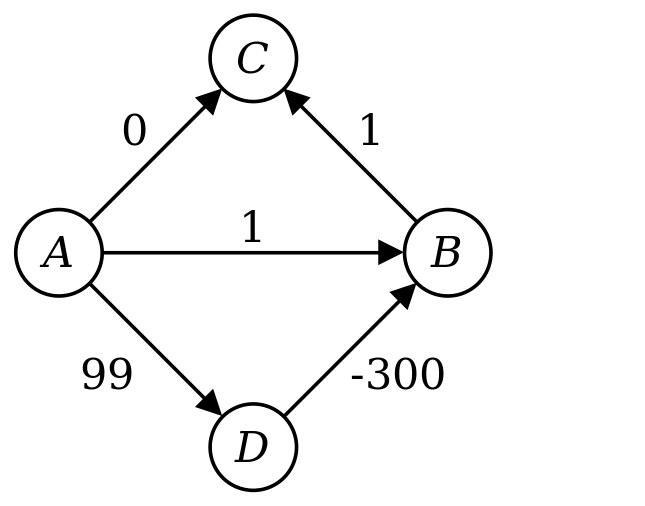
Now, for a moment, let's go back to where we started.

  
At this point, we chose B. So, what we have done is finalized the distance to B and it can never be changed in coming iterations. Now, let's say that there was an edge from A to B with cost as -10 (Negative ten). The distance then is 15 + (-10) = 5, which is lower than 10.

But, we had said that it can never be less than 10. Then, how come this?

Here is the truth. Djikstra's algorithm doesn't work for negative edge weights.

Let's think about the graphs involving negative weights cycles. If there is a negative weight cycle like the C-A-B-C below, we can keep moving in cycles and each iteration of the cycle will decrease the distance, so negative weight cycle is a big NO for Djikstra.  
  
It's intuitive to understand that Djikstra doesn't work for negative cycles but not very intuitive to understand why it doesn't work for negative edges with no negative cycles. I have already given an example above for which it fails. Here is another very good example for which Djikstra's algorithm gives wrong answer. Take A as the source node.

  
Remember that the distance once finalized for a node, it must never be changed again. NEVER.

Time to part :)

I hope you enjoyed it.

Edit:  
**An excerpt from an interview of Edsger Dijkstra, in 2001:**  
" What is the shortest way to travel from Rotterdam to Groningen, in general: from given city to given city. It is the algorithm for the shortest path, which I designed in about twenty minutes. One morning I was shopping in Amsterdam with my young fiancée, and tired, we sat down on the café terrace to drink a cup of coffee and I was just thinking about whether I could do this, and I then designed the algorithm for the shortest path. As I said, it was a twenty-minute invention. In fact, it was published in '59, three years later. The publication is still readable, it is, in fact, quite nice. One of the reasons that it is so nice was that I designed it without pencil and paper. I learned later that one of the advantages of designing without pencil and paper is that you are almost forced to avoid all avoidable complexities. Eventually, that algorithm became to my great amazement, one of the cornerstones of my fame."

##### Must do Dynamic programming Problems Category wise

Hi all,  
I have been following leetcode discussion for a long time and maintaining resources for personal training. People here are really awesome. I have created list of problems based on the categorization of problems which I found during contest, practise sessions or other resources/ dicuss posts.  
This journey has been marathon for me instead of a sprint. To keep a record of progress these list will help us as we can train breadth wise and depth wise on various topics. You can clone and watch these lists. I will keep updating them.

1. **Linear Dp** [link](https://leetcode.com/list/50vlu3z5)  
   This type of questions are easy to figure out. You just need to find the repetitive part of soultion and improve it by saving its result somewhere. A classical example is fibonacci series.
2. **String and Dp** [link](https://leetcode.com/list/50v8wybv)  
   In interviews or contests, problems on string are really common and one has to be strong on this. To make problem more complex soultion requires backtracking or recursion, after that dp can be applied to optimize further.
3. **Dp with Tree and Graph** [link](https://leetcode.com/list/50v8rtm7)  
   Tree and Graph are maily based of DFS and BFS. Its very rare to see some direct relation with dp. But for best understanding one should practise hard with BFS, DFS and other direct graph based algo. Then you can provide some advance solution with dp. I would recommend to start with problem Cheapest Flights Within K Stops ( problem 787).
4. **Knapsack based Dp** [link](https://leetcode.com/list/50vif4uc)  
   Solution is built upon subset, but with few more restrictions. For example you want to complete some courses, they have some reward points associated. But you can attend only k number of courses. Now try to maximize your points. This type of problems are just extension to simple dp, where you add one more dimension to consider provided restriction.
5. **Dp with bits manipulation** [link](https://leetcode.com/list/50vt8ied)  
   Usually we create array or map to store computed state. But Some problems may need you to save space further more by just encoding state and its result into bits. Solve some problems in this list you will realize bits are so powerful.
6. **Dp on math problems** [link](https://leetcode.com/list/50w545lj)  
   These are rare in interview, so one can start this list accordingly. But if you are a math lover, its a great category.
7. **Classical dp problems** [link](https://leetcode.com/list/50wroh7h)  
   Its a must do list. It consits many direct interview question. You will find them as it is. Also these question are base to your understanding. It will help in logic building in contests too. I suggest solve these problems several times. Its like Inception movie.
8. **Grid based dp** [link](https://leetcode.com/list/50izszui)  
   Grid based questions are easy to solve. It just require practise.
9. **Multidimensional Dp** [link](https://leetcode.com/list/50wob6ze)
10. **Digit problems with dp** [link](https://leetcode.com/list/50vtbd3v)
11. **Interval problems with dp** [link](https://leetcode.com/list/50vtr1g3)

Later I'll add more lists based on selected problems in other categories. *Suggestions are most welcome*.

**Approach** - Drawback of solving problems topic wise is, we think in narrow space and do not consider other approaches. So always question yourself why are you using this approach. Also always use **timer** while practising.

**Mentions/Credits**  
Thanks to these great posts

1. @[chuka231](https://leetcode.com/chuka231/)
2. <https://leetcode.com/discuss/general-discussion/662866/DP-for-Beginners-Problems-or-Patterns-or-Sample-Solutions>
3. <https://leetcode.com/discuss/general-discussion/458695/Dynamic-Programming-Patterns>

##### LeetCode Advised Problems Sorted by Topics and Directions

Hello everyone. I received a lot of questions regarding how to solve problems, your approach, which we need to know, important problems and etc. and in this post I wanted to show and share with all of you with the list of my favorite and in my opinion the problems which will be better to solve to better understand the topic and solve problems efficiently. I will try to add every topic from LeetCode, but it is somehow hard do everything at one time, so time by time I will update this post. Let's begin.

**LINKED LIST PROBLEMS:**

LINK TO LIST: <https://leetcode.com/list/50sfo32d>

This list contains the problems to all levels. I included mostly easy and medium problems and one hard problem which is my favorite. How to approach linked list problems:

1. Learn main concepts of Linked List  
   1.1. What is Linked List ?  
   1.2. How to add node to head, middle, end of Linked List ?  
   1.3. How to delete node from head, middle, end of Linked List ?
2. Linked List traversal  
   2.1. Iterative method.  
   2.2. Recursive method.
3. Try to solve problems by using paper and pen or pencil. Why it is important ? When you will visually see how you are going to traverse or make changes to Linked List you will understand what you are actually doing right or wrong. I solved hard problem by drawing it on paper and it was really easy.

**STACK PROBLEMS:**

LINK TO LIST: <https://leetcode.com/list/504xdrcr>

This list contains the problems to all levels. I included many easy and medium problems. How to approach stack problems:

1. Learn main concepts of Stack  
   1.1. What is Stack ?  
   1.2. Basick stack functions such as push(), pop(), top(), and etc.  
   1.3. Prefix, Infix, Postfix notations.
2. Standard problems:  
   2.1. Reverse string with stack.  
   2.2. Reverse linked list using stack.  
   2.3. Valid Parentheses.

**After learning Stack learn the Queue. It is important to know both of them. For both Stack and Queue learn how to implement them using Array and Linked List. It is just to understand their work principle better.**

**TREE PROBLEMS:**

LINK TO LIST: <https://leetcode.com/list/504mfxd2>

This list contains the problems to all levels. I included many easy and medium problems. This topic is one of the favorite interview topics. How to approach tree problems:

1. Learn main concepts of Trees  
   1.1. What are the different types of trees ?  
   1.2. How to search value in BST ?  
   1.3. How to add or delete a value in BST ?  
   1.4. How to find a height of a tree ?  
   1.5. How to construct a BST ?
2. Tree traversals  
   2.1. Level-order traversal both iterative and recursive methods.  
   2.2. Preorder traversal both iterative and recursive methods.  
   2.3. Inordered traversal both iterative and recursive methods.  
   2.4. Postorder traversal both iterative and recursive methods.
3. Also learn how to represent root-leaf node paths, I promise you there are many problems where you will use this.

You ask you I included so many problems to Trees and the answer is that many of them are similar and can be solved by applying the similar logic. Revision can help to build an strong base.

**HASH TABLE AND MAP PROBLEMS:**

LINK TO LIST: <https://leetcode.com/list/504wrexe>

This list contains the problems to all levels. I included many easy and medium problems. How to approach hash table and map problems:

1. Learn main concepts  
   1.1. What is Hash Table ?  
   1.2. What is an Map ?  
   1.3. Their implementation: unordered\_map<>, map<>.
2. To be honest, I watched many lectures and videos and all of them repeated the same things and were not so useful in problem solving. What I advise is it look for their implementation from library, which build-in functions they have and what particular function does.
3. Begin from easy problems, and think which functions will be useful in solving this particular problem, work with library.

**SORTING PROBLEMS:**

LINK TO LIST: <https://leetcode.com/list/5047kw65>

This list contains the problems to all levels. I included many easy and medium problems and also one hard problem for which we actually needed to use linear time sorting algorithm. How to approach sorting problems:

1. Learn main concepts sorting algorithms  
   1.1. Quadratic sorting algorithms: bubble, insertion, selection sorts  
   1.2. Divide and Conquer sorting algorithms: merge and quick sorts  
   1.4. Heap Sort  
   1.3. Linear sorting algorithms: count, radix. Here we should also know some cases when they sort not at linear time.
2. Practice

**SEARCHING PROBLEMS:**

LINK TO LIST: <https://leetcode.com/list/504ixc37>

This list contains the problems to all levels. I included many easy and medium problems. How to approach searching problems:

1. Learn main concepts searching algorithms  
   1.1. Linear search.  
   1.2. Binary search.  
   1.3. Ternary search.
2. Practice

**GREEDY PROBLEMS:**

LINK TO LIST: <https://leetcode.com/list/504i9gu6>

This list contains the problems to all levels. I included many easy and medium problems. How to approach greedy problems:

1. Think how sorting can help you.
2. Practice
3. There are standard problems related to greedy, but actually it will be better if you will try to solve first by yourself. I will update the Greedy and add other problems.

I did not include DP, Graphs, and Strings. I will do it later and will update this post.

I hope this post will help you. Begin from easy problems and try to solve some similar problems to better understand. Do not try to cheat. If you are cheating the you are making no progress. If you cannot solve problem more than 2 hours then skip but do not forget to return and solve it. I think that even if you will spend large amount of time but solve by yourself then it is an progress. If you have any suggestions like what to include to this post, which additional problems to add, or if you have questions please write to comments section. I will read every comment and try to answer when I have free time. Thanks !

##### ABCs of Greedy

I've seen there are many amazing tutorials and problem lists out there for various Concepts like [Dynamic Programming](https://leetcode.com/discuss/general-discussion/458695/Dynamic-Programming-Patterns) by @aatalyk. I plan on compiling one for Greedy in C++ so that it's helpful for me as well as people starting their interview preps. Java and Python solution can be found in the solution column of the questions. It's incomplete but feel free to suggest some resources which can benefit people while revising Greedy.

#### **Some Background**

Greedy algorithm is nothing but a paradigm which builds problems piece by piece. In recursion, we keep on dividing a big problem into multiple smaller chunks and solving those sub problems which is finally used to solve our actual problem. But this isn't the case for Greedy. In this, at any instant, we choose a piece of solution which will offer the most obvious and immediate benefit. For example the famous problem Fractional Knapsack. We keep on choosing the items which are local maximathat is, the items which have maximum value of "profit per unit weight" and in the end get the global maximum.

#### **Some popular Greedy Algorithms**

* Dijkstra’s Shortest Path
* Kruskal’s Minimum Spanning Tree (MST)
* Prim's Minimum Spanning tree
* Huffman Coding

#### **Problem List for Greedy**

These problems are sufficient for developing intuition for Greedy.

* <https://leetcode.com/list/50f6p33i>

#### **When to use it ?**

Whenever we see optimum or maximum or minimum or larget or smallest, the first approach which should strike our mind should be Greedy or Dynamic Programming. If the problem is solvable via recursion, one should do for memoized recursion or DP else start with the brute force and reduce it to Greedy.

#### **Some problems based on Greedy for beginners with the intuition behind solving them:**

#### [**Max-Consecutive-Ones**](https://leetcode.com/problems/max-consecutive-ones/)

**Problem Statement** In an array of 0s and 1s, we are to fing length of the longest chain of 1s.

**Intuition** Traverse the whole array once and find lengths of various chains of 1. Finally return the length of the longest chain.

**Code**

int count = 0 , count\_max = 0;

for(auto& num : nums)

{

if(num == 1)

++count;

else

{

count\_max = (count\_max < count) ? count : count\_max;

count = 0;

}

}

return (count > count\_max) ? count : count\_max ;

#### [**Best time to buy and sell stocks**](https://leetcode.com/problems/best-time-to-buy-and-sell-stock/)

**Problem Statement** Given array of prices, you can buy stock and sell it on a later date only once. Goal is to maximize profit.

**Intuition** The brute for solution would be finding all such possibilities of profit but the time complexity shoots to O(n^2) so this is rejected because it's possible to do it in O(n) time. Simply keep track of the local minima at each step and which traversing the array, find the difference between array element and local minima and return the max difference.

**Code**

int max\_profit = 0, min\_price = INT\_MAX;

for(int i = 0 ; i < prices.size() ; ++i )

{

min\_price = (min\_price > prices[i]) ? prices[i] : min\_price ;

max\_profit = (max\_profit < prices[i] - min\_price) ? prices[i] - min\_price : max\_profit;

}

return max\_profit ;

#### [**Jump Game**](https://leetcode.com/problems/jump-game/)

**Problem Statement** Given an array of non-negative integers nums, you are initially positioned at the first index of the array. Each element in the array represents your maximum jump length at that position. Determine if you are able to reach the last index.

**Intuition** We need to determine the farthest\_index that can be reached while traversing the array. If the farthest\_index is greater than or equal to the last index, we should return true else false. The time complexity will be O(N) and space complexity O(1).

**Code**

bool canJump(vector<int>& nums) {

int n = nums.size(), farthest\_index = 0 ;

for(int i=0; i<n; i++) {

if(farthest\_index < i) return false;

farthest\_index = max (i + nums[i] , farthest\_index) ;

}

return true;

}

#### [**Jump Game II**](https://leetcode.com/problems/jump-game-ii/)

**Problem Statement** Given an array of non-negative integers nums, you are initially positioned at the first index of the array. Each element in the array represents your maximum jump length at that position. Your goal is to reach the last index in the minimum number of jumps. You can assume that you can always reach the last index.

**Intuition** As soon as we see the word minimum number of steps, our aim should be to maximize our reach at every step thus giving the greedy solution of O(N) time. As soon as the right index where we reach is greater than or equal to our last index, we can safely return the step number.

**Code**

int jump(vector<int>& nums) {

int step = 0;

for(int l = 0, r = 0; r < nums.size() - 1; step++){

int r\_new = 0;

for(int i = l; i <= r; i++) r\_new = max(r\_new, i + nums[i]);

l = r + 1;

r = r\_new;

}

return step;

}

#### [**Gas Station**](https://leetcode.com/problems/gas-station/submissions/)

**Problem Statement** Given two integer arrays gas and cost, return the starting gas station's index if you can travel around the circuit once in the clockwise direction, otherwise return -1. If there exists a solution, it is guaranteed to be unique.

**Intuition** The starting index would be such that gas[index] < cost[index] at that particular index. Idea is to go to stations from which we can travel to the farthest so that you can travel the farthest.

**Code**

int n = cost.size();

int sum;

for(int i=0; i<n; ++i) {

if(gas[i] < cost[i]) continue; //idea starting index should've gas[i] >= cost[i]

int j = 0, k, sum = 0;

for(j=0 ; j < n ; ++j){

k = (i+ j) % n;

sum += gas[k] - cost[k];

if (sum < 0) {

if(k<i)

return -1;

else

i = k;

break;

}

}

if(j==n) return i;

}

return -1;

#### [**Lemonade Change**](https://leetcode.com/problems/lemonade-change/submissions/)

**Problem Statement** Return true if and only if you can provide every customer with correct change.

**Intuition** You need to keep the track of the amount of change you have. Customer gives 5 then no problem. If customer gives 10 then you should have $5 bill and decrement the amount of $5 bills you have and increment $10 bill. If customer gives $20, first you try to give $10 + $5. If you don't have $10, then you try to give 3 \* $5 but if you don't have it, return false.

**Code**

int n5 = 0 , n10 = 0 ;

int n= bills.size();

for(int i=0; i<n; i++)

{

if (bills[i] == 5) ++n5;

else if(bills[i] == 10 )

{

if(n5) {

--n5;

++n10;

}

else return false;

}

else

{

if(n10 && n5)

{

--n10;

--n5;

}

else

{

if(n5>=3)

{

n5-=3;

}

else return false;

}

}

}

return true;

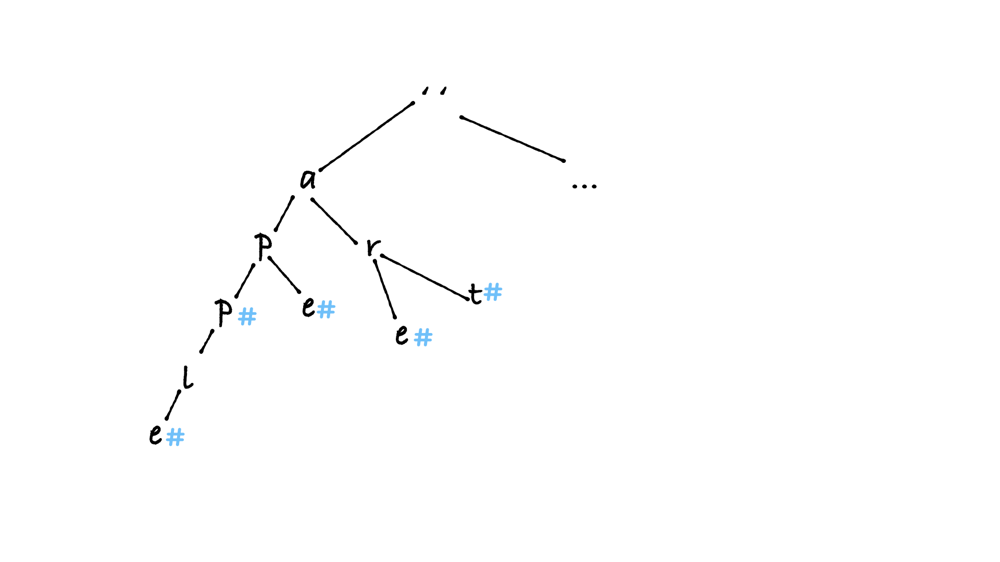
[4. Candy](https://leetcode.com/problems/candy)

**Some amazing General Discussions**

[Greedy Algorithms in Graphs](https://leetcode.com/discuss/general-discussion/969327/graph-algorithms-one-place-dijkstra-bellman-ford-floyd-warshall-prims-kruskals-dsu)

##### Introduction to Trie

## **Overview**



**Figure 1. An example of a trie.**

A trie, or a prefix tree, is a type of search tree that is usually used to store strings. Figure 1 is an example when a trie stores [ape,apple,are,art, ...]. Now let’s take a closer look, and it’s not hard to find that:

* Each path from the root to leaves forms a word.
* Each node except for the root node contains a value.
* All the descendants of a node share a common prefix associated to that node. For example, are and art share ar as the prefix.

There are two operations provided by a trie: **inserting** a new string, and **searching** for a given string.

The advantage of using a trie is that, regardless of the number of strings stored in it, the time complexity for both inserting and searching is always O(L) when L is the length of the input string.

## **Implementations**

There are different approaches to implement a trie in Java and Python. Normally we would define a class for the trie node in Java, however, we can alternatively use dictionaries in Python to improve the efficiencies.

Let's take [LeetCode 208. Implement Trie (Prefix Tree)](https://leetcode.com/problems/implement-trie-prefix-tree/) as an example.

Implement a trie with insert, search, and startsWith methods.

Example:

Trie trie = new Trie();

trie.insert("apple");

trie.search("apple"); // returns true

trie.search("app"); // returns false

trie.startsWith("app"); // returns true

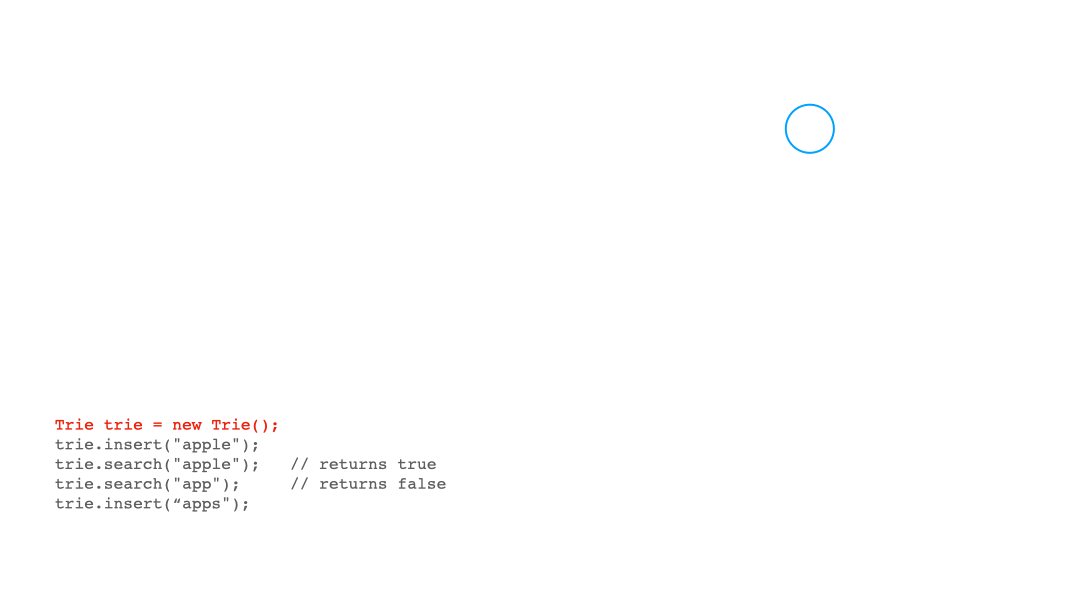
trie.insert("app");

trie.search("app"); // returns true

Note:

You may assume that all inputs are consist of lowercase letters a-z.

All inputs are guaranteed to be non-empty strings.

**Demo:**  


### **Java**

**Define TrieNode**

Firstly we need to define a class TrieNode with:

* A boolean variable isWord to indicate whether we can form a word or it's only a prefix.
* An array of TrieNode named children to store its children node.
* A constructor which initializes isWord to false, and, as only lowercase letters will be used, initializes children to an array of size 26.

class TrieNode {

public boolean isWord;

public TrieNode[] children;

public TrieNode(){

children = new TrieNode[26];

isWord = false;

}

}

Starting from there, we can initialize a trie structure by simply creating a dummy node by private TrieNode root = new TrieNode();.

**insert**

Given a new string word, we would iterate through it. Starting from the dummy node root and the first character c, we would check whether c is in root.children:

* if it is, we can move to that node and increment to next character as well;
* if not, we need to initiate a new node so that we can attach c to the trie.

public void insert(String word) {

// start from the dummy node

TrieNode current\_node = root;

for (int i = 0; i < word.length(); i++) {

int index = word.charAt(i) - 'a';

if (current\_node.children[index] == null) {

// if the current character does not exist,

// initialize a new node

current\_node.children[index] = new TrieNode();

}

current\_node = current\_node.children[index];

}

// remember to set isWord to true after

// reaching the end of word

current\_node.isWord = true;

}

**search**

Similary to insert, we also start the iteration with the first character and the dummy node. If we do not find the character in its children, we can return false. Remember to check isWord after reaching the end of word.

public boolean search(String word) {

TrieNode current\_node = root;

for (int i = 0; i < word.length(); i++) {

int index = word.charAt(i) - 'a';

if (current\_node.children[index] == null) return false;

current\_node = current\_node.children[index];

}

return current\_node.isWord;

}

**startsWith**

The only different to search is that, we do not need to check isWord at the end.

public boolean startsWith(String prefix) {

TrieNode current\_node = root;

for (int i = 0; i < prefix.length(); i++) {

int index = prefix.charAt(i) - 'a';

if (current\_node.children[index] == null) return false;

current\_node = current\_node.children[index];

}

return true;

}

**Full version**

class TrieNode {

public boolean isWord;

public TrieNode[] children;

public TrieNode(){

children = new TrieNode[26];

isWord = false;

}

}

class Trie {

private TrieNode root;

public Trie() {

root = new TrieNode();

}

/\*\* Inserts a word into the trie. \*/

public void insert(String word) {

TrieNode current\_node = root;

for (int i = 0; i < word.length(); i++) {

int index = word.charAt(i) - 'a';

if (current\_node.children[index] == null) {

current\_node.children[index] = new TrieNode();

}

current\_node = current\_node.children[index];

}

current\_node.isWord = true;

}

public boolean search(String word) {

TrieNode current\_node = root;

for (int i = 0; i < word.length(); i++) {

int index = word.charAt(i) - 'a';

if (current\_node.children[index] == null) return false;

current\_node = current\_node.children[index];

}

return current\_node.isWord;

}

public boolean startsWith(String prefix) {

TrieNode current\_node = root;

for (int i = 0; i < prefix.length(); i++) {

int index = prefix.charAt(i) - 'a';

if (current\_node.children[index] == null) return false;

current\_node = current\_node.children[index];

}

return true;

}

}

### **Python with Dictionaries**

For dynamic programming languages like Python, instead of defining a trie node class, we can use a dictionary as a trie node. For each node, we can define a special character such as # to indicate whether it's a word or only a prefix. Otherwise, ideas between Java and Python are quite similar.

class Trie:

def \_\_init\_\_(self):

self.root = {}

def insert(self, word: str) -> None:

current\_node = self.root

for char in word:

if not current\_node.get(char):

current\_node[char] = {}

current\_node = current\_node[char]

current\_node['#'] = True

def search(self, word: str) -> bool:

current\_node = self.root

for char in word:

if current\_node.get(char):

current\_node = current\_node[char]

else:

return False

return current\_node.get('#')

def startsWith(self, prefix: str) -> bool:

current\_node = self.root

for char in prefix:

if current\_node.get(char):

current\_node = current\_node[char]

else:

return False

return True

##### Understanding how numbers are stored in the computer using only 0 and 1.

Hi, up for a light read on CS fundamentals? I am very excited for this post. Integers form the basis of everything we do on the computer. But, do you really know how computers store numbers?  
Will someone ever ask you this in an interview? Probably not.  
Will it help you as a programmer? Yes :)  
This will be a good tutorial for the beginners and a quick revision for the experienced.  
Let's begin. I hope there are no errors but if they have creeped in, let me know in the comments.

The day-to-day system of writing numbers is called decimal. It uses 10 digits (0 to 9) to represent any number. When we write a number in decimal, each digit is multiplied by a power of 10, which is it's distance from the rightmost bit.  
For example, 89 = (8 \* 10^1) + (9 \* 10^0).

In a similar way, we can write all numbers using only 0 and 1, and multiplying each digit by  
a power of 2, which is it's distance from the rightmost bit.  
A computer only knows 0's and 1's. That's why we need to represent numbers in binary.

Let's do some simple calculations:-

i) What does (101) represent in binary?  
A. (101) = (1 \* 2^2) + (0 \* 2^1) + (1 \* 2^0) = (4) + (0) + (1) = 5.

ii) What does (111) represent in binary?  
A. (111) = (1 \* 2^2) + (1 \* 2^1) + (1 \* 2^0) = (4) + (2) + (1) = 7.

Hope, this was clear. Also, when we say that a bit is set, it means that the bit is 1. To set a bit means to make it 1.

Now, let's put our newly gained knowledge to use.

Notice that, since 2 is an even number, all powers of 2 will contribute an even number, except the rightmost bit, which contributes 1. Hence, in every odd number, the last bit will be set.  
Let's take some examples to make that sink.

Let's represent some odd decimal numbers in Binary:-

Decimal | Binary

3 | 11

5 | 101

9 | 1001

15 | 1111

Do you see that the rightmost bit is 1 for all the odd numbers? Similarly, the rightmost bit is 0 for all even numbers. It makes sense right? Since the last bit contributes 1 which is an odd number.

Hopefully everything was clear upto this point.

Now, if we have 3 binary digits, what is the maximum and minimum integer that can be represented?  
The minimum is (000) which is 0. Maximum is (111) which is (7).

IMP: An n-bit binary number can represent numbers in the range 0 to (2^n)-1.

Now, you might ask, how do we represent negative numbers then?  
Well, let's see how we can represent -5.  
Since, 5 is represented as (101), we might represent -5 as -(101). But, we don't really want to include special symbols while storing numbers inside the computer. So, we discard this idea.

We can use the sign-magnitude representation for our problem.  
We will use the left-most bit as the sign bit. If the leftmost bit is 0, it means that it is positive and if it is 1, the number is negative.

Hence,  
5 is represented as (0101) whereas (-5) is represented as (1101). (Left most bit is the sign bit)  
7 is represented as (0111) whereas (-7) is represented as (1111).  
20 is represented as (0001 0100) whereas (-20) is represented as (1001 0100).

Notice that I used 8 bits to represent 20. Most operating systems will use 32 bits to represent integers. So, most bits in between will be filled with 0.

So, in summary, in sign-magnitude representation,  
i) The left most bit is the sign bit.  
ii) Rest all bits represent the magnitude of the number.  
Example. (1000 0010) represents (-2) because left most bit is 1 so it's negative.

Now, let's see if you have understood till now:-

Can you guess what the following binary numbers in sign-magnitude represent?  
(0010)  
(1010)  
(1111)  
(0000 1001)  
(1000 0110)

I hope you did them, you can find correct answers at the end of the post. (No, kidding. Just do it yourselves :) )

Now, we have solved the problem of representing negative numbers without using any special symbol.  
I have a task for you. Let's stick to 4 bits to represent numbers.  
Can you write sign-magnitude for 0?

We can write 0 as (0000) or (1000). One is +0 and other is -0.  
If you have been programming for a while, you might realize that the operation of comparing something to 0 is done quite frequently and it's unnecessary to check it's equality with both these numbers. It would be easier if we had just one representation for zero.

In short, the first issue is, 0 has two representations .i.e.(0000) and (1000).

Great! Let's do some operations. Do you remember the rules for binary addition?

Here they are:

0 + 0 = 0 with carry as 0  
1 + 0 = 1 with carry as 0  
0 + 1 = 1 with carry as 0  
1 + 1 = 0 with carry as 1

Addition of two numbers will take place bitwise .i.e. we add two numbers bit by bit.

Let's add 2 and 5.

(2) = (0010)  
(5) = (0101)

2 + 5 = 0010 + 0101 = 0010

+ 0101

= 0111

What does (0111) represent? 7 right? +7 to be precise.

Try adding -3 and 4.

(-3) = (1011)  
(+4) = (0100)

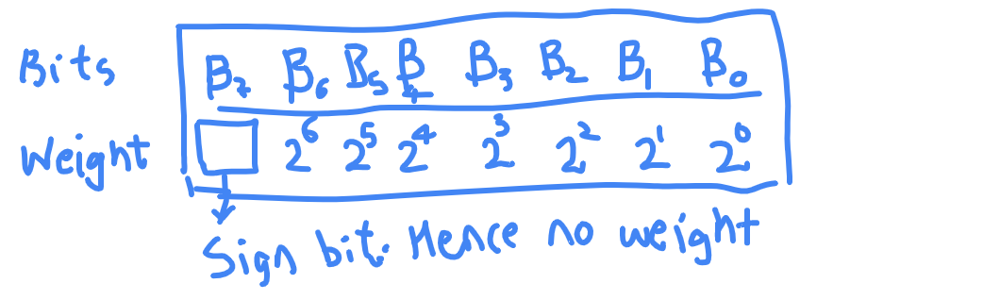
-3 + (+4) = 1011 + 0100 = 1111

Wew! What does 1111 represent? -7? Yep. But, is that the answer? Nope. This is another issue with sign-magnitude representation. We need to consider the sign of both numbers before doing any operation.

We have discussed the two major drawbacks of sign-magnitude representation. We will see how we can improve our current representation using the two's complement representation. The drawbacks are overcome by two's complement representation. This is the reason sign-magnitude represenation is rarely used. But, we don't deny it's existence, it is still used in some areas and it's better to just know about it.

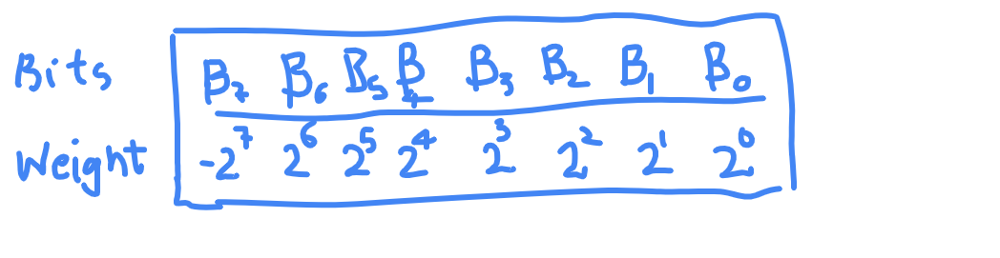
Let's see what two's complement representation of a number is.

Firstly let's define the weight of a bit. The weight of a bit is the amount it contributes to the actual number if the corresponding bit in the number is set. Look at the following image to understand better.



This is what we have seen already. This is what we did to get the value of (0101) as 5. By multiplying the weight by corresponding bit.

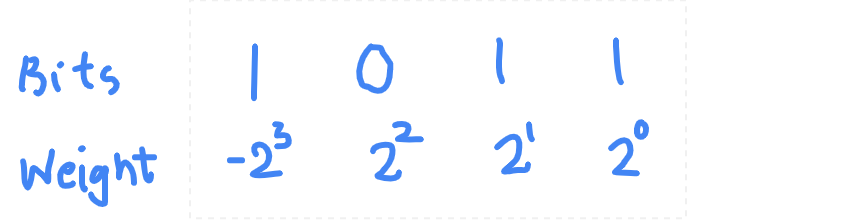
The left most bit ofcourse doesn't have any weight as it just aids us in knowing if number is positive or negative.

This is where it becomes interesting. The left-most bit is sign-bit in Twos complement too but now, it also has a weight just like every other bit albeit with negative sign.  


Notice the weight of Bit7(B7).

This tremendously changes how each integer is stored in computer.

Let's take 1011 for example. Let's see what it represents.



We use the same method of getting the value.  
Multiply the bit and weight.  
So (1011) in Twos complement is:  
1 \* (-(2^3)) + 0 \* (2^2) + 1 \* (2^1) + 1 \* (2^0)  
= -8 + 0 + 2 + 1  
= -8 + 3  
= -5

Hence, 1011 represents -5.

What is representation for -3? 1101. (Please verify it by adding weights)  
What is representation for +4? 0100. (Please verify this too)

The representation for 0 in Twos complement is 0000 .i.e +0. So, a simple comparison to know if a value is 0 or not.

Alright, a small question before we part.

If I give you a number in Twos complement, can you tell me if it is positive or negative?

In sign-magnitude, it is simple, just look at the sign bit and you know if it is positive or negative. But, what about 2's complement?  
Think before you read further.

The answer is yes, you can know if a number if +ve or -ve just by looking at leftmost bit.

If leftmost bit is 0, the number is always greater than or equal to zero because each weight except the leftmost is positive and leftmost bit is 0, it means that there is no negative contribution from any bit and so value can never be negative.

If leftmost bit is 1, the number is always negative. Even if all other bits are 1 then too the number is negative. Let's check it.

(1111) = (-8 + 4 + 2 + 1) = -1

(1101) = (-8 + 4 + 0 + 1) = -3

You can prove it with simple mathematics involving powers of 2 but that is left as an exercise for the reader.

From now, we will write all the representations in twos complement.  
Also, if you want to know, how an integer is represented in your programming language, you can use following code:

For example, C++ integer is of 32 bits. So, if I want to know all the bits of a number 'n', I can run the following code. I believe that the code will be similar in other programming languages with minor changes.

void printTwosComplementRepresentation(int n)

{

// We start from 31 and go to 0.

for (int i = 31; i >= 0; --i)

{

if ((1 << i) & n) // Check if the ith bit is set or not

std::cout << 1 ;

else std::cout << 0 ;

}

}

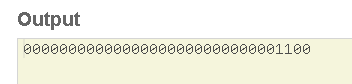
int main()

{

printTwosComplementRepresentation(12);

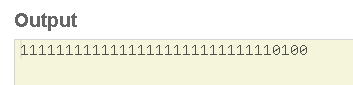
return 0;

}

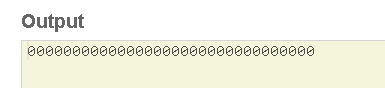


Above image is for +12.

For -12, it looks something like below.



Now, I am interested in knowing how 0 is represented. I ran the same program with n as 0 and it gave following output.



So, leftmost bit is 0 in 0.

Do you wonder what if leftmost bit is 1 and all other bits are 0?  
Since, we know that weight of leftmost bit is negative, the final number will be very big negative number and it's absolute value will be a power of 2.

So, that's it. This is how numbers are represented in memory. If you want to learn how various operations are done with Twos complement, you are free to check out other amazing sources on the web.  
I'll be more than happy to take any questions :)  
Have a good day!

##### Explain Time Complexity to a 10 year old!

Explain time complexity to me as If I were a 10 year old.

Say you have a test tomorrow in which you will be given a number N and you will need to find out the sum 1 + 2 + .. + N. So if N = 3 you will need to tell the answer as 1 + 2 + 3 = 6. The Teacher has already announced that N will not be larger than 10.

You are good with addition but you were not fast enough so could not complete all the problems. **Your friend aced the test.** You asked him how, he told you honestly that he had **memorized all the answers** and came for the test. He memorized 10 integers while you had to memorize nothing.

The next test was harder, the same N can be up to 100. Both you and your friend did poorly this time. He couldn't memorize 100 integers and you were slow. **As this N got larger**, the time you need to answer increased whereas for your friend the time stayed the same but the integers he had to memorize increased.

**The topper of the class** noticed that actually a simple formula can be used to answer any problem. If the teacher asked 1+2+..+N he would answer N\*(N+1)/2.

**Here is the grading:**  
You: O(1) space but O(N) time  
Your friend: O(1) time but O(N) space  
Topper: O(1) time and O(1) space

Guess which one grew up to be a programmer, all of them :p

##### Dynamic Programming Patterns

Before starting the topic let me introduce myself. I am a Mobile Developer currently working in Warsaw and spending my free time for interview preparations. I started to prepare for interviews two years ago. At that time I should say I could not solve the two sum problem. Easy problems seemed to me like hard ones so most of the time I had to look at editorials and discuss section. Currently, I have solved ~800 problems and time to time participate in contests. I usually solve 3 problems in a contest and sometimes 4 problems. Ok, lets come back to the topic.

Recently I have concentrated my attention on Dynamic Programming cause its one of the hardest topics in an interview prep. After solving ~140 problems in DP I have noticed that there are few patterns that can be found in different problems. So I did a research on that and find the following topics. I will not give complete ways how to solve problems but these patterns may be helpful in solving DP.

# **Patterns**

[Minimum (Maximum) Path to Reach a Target](https://leetcode.com/discuss/general-discussion/458695/Dynamic-Programming-Patterns#Minimum-(Maximum)-Path-to-Reach-a-Target)  
[Distinct Ways](https://leetcode.com/discuss/general-discussion/458695/Dynamic-Programming-Patterns#distinct-ways)  
[Merging Intervals](https://leetcode.com/discuss/general-discussion/458695/Dynamic-Programming-Patterns#Merging-Intervals)  
[DP on Strings](https://leetcode.com/discuss/general-discussion/458695/Dynamic-Programming-Patterns#DP-on-Strings)  
[Decision Making](https://leetcode.com/discuss/general-discussion/458695/Dynamic-Programming-Patterns#Decision-Making)

# **Minimum (Maximum) Path to Reach a Target**

Problem list: <https://leetcode.com/list/55ac4kuc>

Generate problem statement for this pattern

### **Statement**

Given a target find minimum (maximum) cost / path / sum to reach the target.

### **Approach**

Choose minimum (maximum) path among all possible paths before the current state, then add value for the current state.

routes[i] = min(routes[i-1], routes[i-2], ... , routes[i-k]) + cost[i]

Generate optimal solutions for all values in the target and return the value for the target.

for (int i = 1; i <= target; ++i) {

for (int j = 0; j < ways.size(); ++j) {

if (ways[j] <= i) {

dp[i] = min(dp[i], dp[i - ways[j]] + cost / path / sum) ;

}

}

}

return dp[target]

### **Similar Problems**

[746. Min Cost Climbing Stairs](https://leetcode.com/problems/min-cost-climbing-stairs/) Easy

for (int i = 2; i <= n; ++i) {

dp[i] = min(dp[i-1], dp[i-2]) + (i == n ? 0 : cost[i]);

}

return dp[n]

[64. Minimum Path Sum](https://leetcode.com/problems/minimum-path-sum/) Medium

for (int i = 1; i < n; ++i) {

for (int j = 1; j < m; ++j) {

grid[i][j] = min(grid[i-1][j], grid[i][j-1]) + grid[i][j];

}

}

return grid[n-1][m-1]

[322. Coin Change](https://leetcode.com/problems/coin-change/) Medium

for (int j = 1; j <= amount; ++j) {

for (int i = 0; i < coins.size(); ++i) {

if (coins[i] <= j) {

dp[j] = min(dp[j], dp[j - coins[i]] + 1);

}

}

}

[931. Minimum Falling Path Sum](https://leetcode.com/problems/minimum-falling-path-sum/) Medium

[983. Minimum Cost For Tickets](https://leetcode.com/problems/minimum-cost-for-tickets/) Medium

[650. 2 Keys Keyboard](https://leetcode.com/problems/2-keys-keyboard/) Medium

[279. Perfect Squares](https://leetcode.com/problems/perfect-squares/) Medium

[1049. Last Stone Weight II](https://leetcode.com/problems/last-stone-weight-ii/) Medium

[120. Triangle](https://leetcode.com/problems/triangle/) Medium

[474. Ones and Zeroes](https://leetcode.com/problems/ones-and-zeroes/) Medium

[221. Maximal Square](https://leetcode.com/problems/maximal-square/) Medium

[322. Coin Change](https://leetcode.com/problems/coin-change/) Medium

[1240. Tiling a Rectangle with the Fewest Squares](https://leetcode.com/problems/tiling-a-rectangle-with-the-fewest-squares/) Hard

[174. Dungeon Game](https://leetcode.com/problems/dungeon-game/) Hard

[871. Minimum Number of Refueling Stops](https://leetcode.com/problems/minimum-number-of-refueling-stops/) Hard

# **Distinct Ways**

Problem List: <https://leetcode.com/list/55ajm50i>

Generate problem statement for this pattern

### **Statement**

Given a target find a number of distinct ways to reach the target.

### **Approach**

Sum all possible ways to reach the current state.

routes[i] = routes[i-1] + routes[i-2], ... , + routes[i-k]

Generate sum for all values in the target and return the value for the target.

for (int i = 1; i <= target; ++i) {

for (int j = 0; j < ways.size(); ++j) {

if (ways[j] <= i) {

dp[i] += dp[i - ways[j]];

}

}

}

return dp[target]

### **Similar Problems**

[70. Climbing Stairs](https://leetcode.com/problems/climbing-stairs/) easy

for (int stair = 2; stair <= n; ++stair) {

for (int step = 1; step <= 2; ++step) {

dp[stair] += dp[stair-step];

}

}

[62. Unique Paths](https://leetcode.com/problems/unique-paths/) Medium

for (int i = 1; i < m; ++i) {

for (int j = 1; j < n; ++j) {

dp[i][j] = dp[i][j-1] + dp[i-1][j];

}

}

[1155. Number of Dice Rolls With Target Sum](https://leetcode.com/problems/number-of-dice-rolls-with-target-sum/) Medium

for (int rep = 1; rep <= d; ++rep) {

vector<int> new\_ways(target+1);

for (int already = 0; already <= target; ++already) {

for (int pipe = 1; pipe <= f; ++pipe) {

if (already - pipe >= 0) {

new\_ways[already] += ways[already - pipe];

new\_ways[already] %= mod;

}

}

}

ways = new\_ways;

}

**Note**

Some questions point out the number of repetitions, in that case, add one more loop to simulate every repetition.

[688. Knight Probability in Chessboard](https://leetcode.com/problems/knight-probability-in-chessboard/) Medium

[494. Target Sum](https://leetcode.com/problems/target-sum/) Medium

[377. Combination Sum IV](https://leetcode.com/problems/combination-sum-iv/) Medium

[935. Knight Dialer](https://leetcode.com/problems/knight-dialer/) Medium

[1223. Dice Roll Simulation](https://leetcode.com/problems/dice-roll-simulation/) Medium

[416. Partition Equal Subset Sum](https://leetcode.com/problems/partition-equal-subset-sum/) Medium

[808. Soup Servings](https://leetcode.com/problems/soup-servings/) Medium

[790. Domino and Tromino Tiling](https://leetcode.com/problems/domino-and-tromino-tiling/) Medium

[801. Minimum Swaps To Make Sequences Increasing](https://leetcode.com/problems/minimum-swaps-to-make-sequences-increasing/)

[673. Number of Longest Increasing Subsequence](https://leetcode.com/problems/number-of-longest-increasing-subsequence/) Medium

[63. Unique Paths II](https://leetcode.com/problems/unique-paths-ii/) Medium

[576. Out of Boundary Paths](https://leetcode.com/problems/out-of-boundary-paths/) Medium

[1269. Number of Ways to Stay in the Same Place After Some Steps](https://leetcode.com/problems/number-of-ways-to-stay-in-the-same-place-after-some-steps/) Hard

[1220. Count Vowels Permutation](https://leetcode.com/problems/count-vowels-permutation/) Hard

# **Merging Intervals**

Problem List: <https://leetcode.com/list/55aj8s16>

Generate problem statement for this pattern

### **Statement**

Given a set of numbers find an optimal solution for a problem considering the current number and the best you can get from the left and right sides.

### **Approach**

Find all optimal solutions for every interval and return the best possible answer.

// from i to j

dp[i][j] = dp[i][k] + result[k] + dp[k+1][j]

Get the best from the left and right sides and add a solution for the current position.

for(int l = 1; l<n; l++) {

for(int i = 0; i<n-l; i++) {

int j = i+l;

for(int k = i; k<j; k++) {

dp[i][j] = max(dp[i][j], dp[i][k] + result[k] + dp[k+1][j]);

}

}

}

return dp[0][n-1]

### **Similar Problems**

[1130. Minimum Cost Tree From Leaf Values](https://leetcode.com/problems/minimum-cost-tree-from-leaf-values/) Medium

for (int l = 1; l < n; ++l) {

for (int i = 0; i < n - l; ++i) {

int j = i + l;

dp[i][j] = INT\_MAX;

for (int k = i; k < j; ++k) {

dp[i][j] = min(dp[i][j], dp[i][k] + dp[k+1][j] + maxs[i][k] \* maxs[k+1][j]);

}

}

}

[96. Unique Binary Search Trees](https://leetcode.com/problems/unique-binary-search-trees/) Medium

[1039. Minimum Score Triangulation of Polygon](https://leetcode.com/problems/minimum-score-triangulation-of-polygon/) Medium

[546. Remove Boxes](https://leetcode.com/problems/remove-boxes/) Medium

[1000. Minimum Cost to Merge Stones](https://leetcode.com/problems/minimum-cost-to-merge-stones/) Medium

[312. Burst Balloons](https://leetcode.com/problems/burst-balloons/) Hard

[375. Guess Number Higher or Lower II](https://leetcode.com/problems/guess-number-higher-or-lower-ii/) Medium

# **DP on Strings**

Problem List: <https://leetcode.com/list/55afh7m7>

General problem statement for this pattern can vary but most of the time you are given two strings where lengths of those strings are not big

### **Statement**

Given two strings s1 and s2, return some result.

### **Approach**

Most of the problems on this pattern requires a solution that can be accepted in O(n^2) complexity.

// i - indexing string s1

// j - indexing string s2

for (int i = 1; i <= n; ++i) {

for (int j = 1; j <= m; ++j) {

if (s1[i-1] == s2[j-1]) {

dp[i][j] = /\*code\*/;

} else {

dp[i][j] = /\*code\*/;

}

}

}

If you are given one string s the approach may little vary

for (int l = 1; l < n; ++l) {

for (int i = 0; i < n-l; ++i) {

int j = i + l;

if (s[i] == s[j]) {

dp[i][j] = /\*code\*/;

} else {

dp[i][j] = /\*code\*/;

}

}

}

[1143. Longest Common Subsequence](https://leetcode.com/problems/longest-common-subsequence/) Medium

for (int i = 1; i <= n; ++i) {

for (int j = 1; j <= m; ++j) {

if (text1[i-1] == text2[j-1]) {

dp[i][j] = dp[i-1][j-1] + 1;

} else {

dp[i][j] = max(dp[i-1][j], dp[i][j-1]);

}

}

}

[647. Palindromic Substrings](https://leetcode.com/problems/palindromic-substrings/) Medium

for (int l = 1; l < n; ++l) {

for (int i = 0; i < n-l; ++i) {

int j = i + l;

if (s[i] == s[j] && dp[i+1][j-1] == j-i-1) {

dp[i][j] = dp[i+1][j-1] + 2;

} else {

dp[i][j] = 0;

}

}

}

[516. Longest Palindromic Subsequence](https://leetcode.com/problems/longest-palindromic-subsequence/) Medium

[1092. Shortest Common Supersequence](https://leetcode.com/problems/shortest-common-supersequence/) Medium

[72. Edit Distance](https://leetcode.com/problems/edit-distance/) Hard

[115. Distinct Subsequences](https://leetcode.com/problems/distinct-subsequences/) Hard

[712. Minimum ASCII Delete Sum for Two Strings](https://leetcode.com/problems/minimum-ascii-delete-sum-for-two-strings/) Medium

[5. Longest Palindromic Substring](https://leetcode.com/problems/longest-palindromic-substring/) Medium

# **Decision Making**

Problem List: <https://leetcode.com/list/55af7bu7>

The general problem statement for this pattern is forgiven situation decide whether to use or not to use the current state. So, the problem requires you to make a decision at a current state.

### **Statement**

Given a set of values find an answer with an option to choose or ignore the current value.

### **Approach**

If you decide to choose the current value use the previous result where the value was ignored; vice-versa, if you decide to ignore the current value use previous result where value was used.

// i - indexing a set of values

// j - options to ignore j values

for (int i = 1; i < n; ++i) {

for (int j = 1; j <= k; ++j) {

dp[i][j] = max({dp[i][j], dp[i-1][j] + arr[i], dp[i-1][j-1]});

dp[i][j-1] = max({dp[i][j-1], dp[i-1][j-1] + arr[i], arr[i]});

}

}

[198. House Robber](https://leetcode.com/problems/house-robber/) Easy

for (int i = 1; i < n; ++i) {

dp[i][1] = max(dp[i-1][0] + nums[i], dp[i-1][1]);

dp[i][0] = dp[i-1][1];

}

[121. Best Time to Buy and Sell Stock](https://leetcode.com/problems/best-time-to-buy-and-sell-stock/) Easy

[714. Best Time to Buy and Sell Stock with Transaction Fee](https://leetcode.com/problems/best-time-to-buy-and-sell-stock-with-transaction-fee/) Medium

[309. Best Time to Buy and Sell Stock with Cooldown](https://leetcode.com/problems/best-time-to-buy-and-sell-stock-with-cooldown/) Medium

[123. Best Time to Buy and Sell Stock III](https://leetcode.com/problems/best-time-to-buy-and-sell-stock-iii/) Hard

[188. Best Time to Buy and Sell Stock IV](https://leetcode.com/problems/best-time-to-buy-and-sell-stock-iv/) Hard

I hope these tips will be helpful 😊

**Bonus reward for these topics:**

* **Summarize or organize certain types of topics/patterns.**
  + i.e. [Dynamic Programming Patterns](https://leetcode.com/discuss/general-discussion/458695/Dynamic-Programming-Patterns), [[Python] Powerful Ultimate Binary Search Template](https://leetcode.com/discuss/general-discussion/786126/Python-Powerful-Ultimate-Binary-Search-Template.-Solved-many-problems)
* **Tutorial for Beginners (Algorithm & data structure / OOD / System Design)**
  + i.e. Introduction to Dynamic Programming
  + Content must be your original content and is exclusive to LeetCode Discuss.
* **System Design related topics**
  + Post should include detailed problem description, problem solving, and solution walkthrough.
  + i.e. [Design Facebook](https://leetcode.com/discuss/interview-question/system-design/719253/Design-Facebook-%3A-System-Design-Interview), [Design URL Shortening service](https://leetcode.com/discuss/interview-question/system-design/124658/Design-URL-Shortening-service-like-TinyURL), [Design video sharing platform](https://leetcode.com/discuss/interview-question/system-design/496042/Design-video-sharing-platform-like-Youtube)

##### Design Facebook : System Design Interview

**Design a simple model of Facebook where people can add other people as friends. In addition, where people can post messages and that messages are visible on their friend's page. The design should be such that it can handle 10M of people. There may be, on an average 100 friends each person has. Every day each person posts around 10 messages on an average.**

## **Use Case**

1. A user can create their own profile.
2. A user can add other users to his friend list.
3. Users can post messages to their timeline.
4. The system should display posts of friends to the display board/timeline.
5. People can like a post.
6. People can share their friends post to their own display board/timeline.

## **Constraints**

1. Consider a whole network of people as represented by a graph. Each person is a node and each friend  
   relationship is an edge of the graph.
2. Total number of distinct users / nodes: 10 million
3. Total number of distinct friend’s relationship / edges in the graph: 100 \* 10 million
4. Number of messages posted by a single user per day: 10
5. Total number of messages posted by the whole network per day: 10 \* 10 million

## **Basic Design**

Our system architecture is divided into two parts:

1. First, the web server that will handle all the incoming requests.
2. The second database, which will store the entire person's profile, their friend relations and posts.

First, three requirements creating a profile, adding friends, posting messages are written some information  
to the database. While the last operation is reading data from the database.

The system will look like this:

1. Each user will have a profile.
2. There will be a list of friends in each user profile.
3. Each user will have their own homepage where his posts will be visible.  
   A user can like any post of their friend and that likes will reflect on the actual message shared by his  
   friend.

If a user shares some post, then this post will be added to the user home page and all the other friends of  
the user will see this post as a new post.

## **Bottleneck**

A number of requests posted per day is 100 million. Approximate some 1000 request are posted per  
second. There will be an uneven distribution of load so the system that we will design should be able to  
handle a few thousand requests per seconds.

## **Scalability**

Since there is, a heavy load we need horizontal scaling many web servers will be handling the requests.  
In doing this we need to have a load balancer, which will distribute the request among the servers.  
This approach gives us a flexibility that when the load increases, we can add more web servers to handle  
the increased load.  
These web servers are responsible for handling new post added by the user. They are responsible for  
generating various user homepage and timeline pages. In our case, the client is the web browser,  
which is rendering the page for the user.

We need to store data about user profile, Users friend list, User-generated posts, User like statues to the  
posts.  
Let us find out how much storage we need to store all this data. The total number of users 10 million. Let  
us suppose each user is using Facebook for 5 to 6 years, so the total number of posts that a user had  
produced in this whole time is approximately 20,000 million or 20 billion. Let us suppose each message  
consists of 100 words or 500 characters. Let us assume each character take 2 bytes.

Total memory required = 20 \* 500 \* 2 billion bytes.

= 20,000 billion bytes

= 20, 000 GB

= 20 TB

1 gigabyte (GB) = 1 billion bytes

1000 gigabytes (GB) = 1 Terabytes

Most of the memory is taken from the posts and the user profile and friend list will take nominal as  
compared with the posts. We can use a relational database like SQL to store this data. Facebook and  
twitter are using a relational database to store their data.

Responsiveness is key for social networking site. Databases have their own cache to increase their  
performance. Still database access is slow as databases are stored on hard drives and they are slower  
than RAM. Database performance can be increased by replication of the database. Requests can be  
distributed between the various copies of the databases.

Also, there will be more reads then writes in the database so there can be multiple slave DB which are  
used for reading and there can be few master DB for writing. Still database access is slow to we will use  
some caching mechanism like Memcached in between application server and database. Highly popular  
users and their home page will always remain in the cache.

There may be the case when the replication no longer solves the performance problem. In addition, we  
need to do some Geo-location based optimization in our solution.  
Again, look for a complete diagram in the scalability theory section.  
If it were asked in the interview how you would store the data in the database. The schema of the  
database can look like:

### **Table Users:**

1. User Id

2. First Name

3. Last Name

4. Email

5. Password

6. Gender

7. DOB

8. Relationship

### **Table Posts:**

1. Post Id

2. Author Id

3. Date of Creation

4. Content

### **Table Friends:**

1. Relation Id

2. First Friend Id

3. Second Friend Id

### **Table Likes:**

1. Id

2. Post Id

3. User Id

That's it ;)  
Thanks for reading out !

##### Design URL Shortening service like TinyURL

**Problem:** Design a service like TinyURL, a URL shortening service, a web service that provides short aliases for redirection of long URLs.

**Solution**: If you don't know about [TinyURL](https://tinyurl.com/), just check it. Basically we need a one to one mapping to get shorten URL which can retrieve original URL later. This will involve saving such data into database.  
We should check the following things:

* What's the traffic volume / length of the shortened URL?
* What's the mapping function?
* Single machine or multiple machines?

**Traffic**: Let's assume we want to serve more than 1000 billion URLs. If we can use 62 characters [A-Z, a-z, 0-9] for the short URLs having length n, then we can have total 62^n URLs. So, we should keep our URLs as short as possible given that it should fulfill the requirement. For our requirement, we should use n=7 i.e the length of short URLs will be 7 and we can serve 62^7 ~= 3500 billion URLs.

**Basic solution**:  
To make things easier, we can assume the alias is something like <http://tinyurl.com/><alias\_hash> and alias\_hash is a fixed length string.  
To begin with, let’s store all the mappings in a single database. A straightforward approach is using alias\_hash as the ID of each mapping, which can be generated as a random string of length 7.

Therefore, we can first just store <ID, URL>. When a user inputs a long URL “[http://www.google.com](http://www.google.com/)”, the system creates a random 7-character string like “abcd123” as ID and inserts entry <“abcd123”, “[http://www.google.com](http://www.google.com/)”> into the database.

In the run time, when someone visits <http://tinyurl.com/abcd123>, we look up by ID “abcd123” and redirect to the corresponding URL “[http://www.google.com](http://www.google.com/)”.

**Problem with this solution**:  
We can't generate unique hash values for the given long URL. In hashing, there may be collisions (2 long urls map to same short url) and we need a unique short url for every long url so that we can access long url back but hash is one way function.

**Better Solution**:

One of the most simple but also effective one, is to have a database table set up this way:

Table Tiny\_Url(  
ID : int PRIMARY\_KEY AUTO\_INC,  
Original\_url : varchar,  
Short\_url : varchar  
)  
Then the auto-incremental primary key ID is used to do the conversion: (ID, 10) <==> (short\_url, BASE). Whenever you insert a new original\_url, the query can return the new inserted ID, and use it to derive the short\_url, save this short\_url and send it to cilent.

**Code for methods** (that are used to convert ID to short\_url and short\_url to ID):

string idToShortURL(long int n)

{

// Map to store 62 possible characters

char map[] = "abcdefghijklmnopqrstuvwxyzABCDEF"

"GHIJKLMNOPQRSTUVWXYZ0123456789";

string shorturl;

// Convert given integer id to a base 62 number

while (n)

{

shorturl.push\_back(map[n%62]);

n = n/62;

}

// Reverse shortURL to complete base conversion

reverse(shorturl.begin(), shorturl.end());

return shorturl;

}

// Function to get integer ID back from a short url

long int shortURLtoID(string shortURL)

{

long int id = 0; // initialize result

// A simple base conversion logic

for (int i=0; i < shortURL.length(); i++)

{

if ('a' <= shortURL[i] && shortURL[i] <= 'z')

id = id\*62 + shortURL[i] - 'a';

if ('A' <= shortURL[i] && shortURL[i] <= 'Z')

id = id\*62 + shortURL[i] - 'A' + 26;

if ('0' <= shortURL[i] && shortURL[i] <= '9')

id = id\*62 + shortURL[i] - '0' + 52;

}

return id;

}

**Multiple machines:**

If we are dealing with massive data of our service, distributed storage can increase our capacity. The idea is simple, get a hash code from original URL and go to corresponding machine then use the same process as a single machine. For routing to the correct node in cluster, Consistent Hashing is commonly used.

Following is the pseudo code for example,

**Get shortened URL**

* hash original URL string to 2 digits as hashed value hash\_val
* use hash\_val to locate machine on the ring
* insert original URL into the database and use getShortURL function to get shortened URL short\_url
* Combine hash\_val and short\_url as our final\_short\_url (length=8) and return to the user

**Retrieve original from short URL**

* get first two chars in final\_short\_url as hash\_val
* use hash\_val to locate the machine
* find the row in the table by rest of 6 chars in final\_short\_url as short\_url
* return original\_url to the user

**Other factors**:

One thing I’d like to further discuss here is that by using GUID (Globally Unique Identifier) as the entry ID, what would be pros/cons versus incremental ID in this problem?

If you dig into the insert/query process, you will notice that using random string as IDs may sacrifice performance a little bit. More specifically, when you already have millions of records, insertion can be costly. Since IDs are not sequential, so every time a new record is inserted, the database needs to go look at the correct page for this ID. However, when using incremental IDs, insertion can be much easier – just go to the last page.

You can connect with me here: https://www.linkedin.com/in/shashi-bhushan-kumar-709a05b5/

References: <http://blog.gainlo.co/index.php/2016/03/08/system-design-interview-question-create-tinyurl-system/>  
<https://www.geeksforgeeks.org/how-to-design-a-tiny-url-or-url-shortener/>

##### Design video sharing platform like Youtube

I'm writing for the sole purpose of my system design interview preparation. Read it at your own discretion. Well, comments and discussion regarding this topic are always appreciated.

## **Gathering requirements**

The problem description is intended to be ambigious as the interviewer wants to check if we're able to ask reasonable questions, collect requirements and define a core set of features for a minimum viable product (MVP) that we can start to work with.

In this case, I'd like to ask a few questions to make things clear, including

* Q: What are main features of the video sharing platform?
  + A: The system will be a public platform that users can watch and upload videos.
* Q: Who will be the users of the system?
  + A: Users will be around the world.
* Q: How are they going to use it?
  + A: Users will use various kinds of devices to visit our service.
* Q: Is there any external dependencies? What are the upstream and downstream system?
  + A: The system will leverage existing authentication system.

Once I get answers for above questions, I'd confirm with the interviewer that I'll design a video platform that enables users to watch and upload videos. Authentication is out of the scope of this problem.

Based on information that I gathered so far, I come up with a bunch of functional and non-functional requirements below and confirm with interviewer.

**Functional requirements**

* Users can watch videos on the platform
* Users can upload their own videos

**Non-functional requirements**

* The platform should be highly available
* The response time for users in different regions should be at the same level
* The platform should scale while userbase is increasing

## **Estimation**

The goal of this step is to come up with estimated numbers of how scalable our system should be.

We have two types of requests, read and write. Write request(upload videos) will take much longer time and is ususally much less than read requests (recall how often you watch videos on Youtue and the times you upload videos).

So, let's focus on read request firstly. As an easy start, assume our service receives and completes 1000 request per second (RPS). I use request to reflect common server requirements rather than functionality specific requirements eg. Tweets, Views ... because it's business independent. Functionality specific requirements will be eventually mapped to server requirements anyway.

For one day, we'll have 1000 \* 24 \* 60 \* 60 ~= 1000 \* 30 \* 3000 = 90 million = 90 M requests  
For one year, we'll hvae 90M \* 12 \* 30 = 90 M \* 360 ~= 100 M \* 300 = 30 billion = 30 B requests  
For five years, we'll have 30B \* 5 = 150 billion = 150 B requests in total.

The next assumption I'm going to make is the response time. Assume our response time SLA is 200ms.

Then I need to have 1000 / (1s / 200ms) = 200 threads in total to handle 1000 RPS. So, the next question will be how many servers do I need to have 200 threads. One simple formula for estimating ideal Java thread pool size is

Number of threads = Number of Available Cores \* Target CPU utilization \* (1 + Wait time / Service time)

* Waiting time - is the time spent waiting for IO bound tasks to complete, say waiting for HTTP response from remote service.
* Service time - is the time spent being busy, say processing the HTTP response, marshaling/unmarshaling, etc.

Because our video sharing web service is not computing-intensive application(though it can be, considering video encoding, compression), I assume the Wait time / Service time ratio will be relatively large, say 50. And also assume our server has 2 CPU cores running at 50% utilization.

Then, each of this kind of server can support 2 \* 0.5 \* (1 + 50) ~= 50 threads and since we need to have 200 threads to handle 1000 RPS, so we need 200 / 50 = 4 servers to handle 1000 RPS.

Let's switch to write requests (upload video). Assume the read / write or say watch / upload ratio is 100. So, we're expecting 1000 / 100 = 10 write requests per second. Please note that the uploading time varies because the video size and network speed play important roles here. So, the estimate is even more subjective.

Assume average video size is 500 MB and network bandwidth on customer side is 100 mbps. Then,

* each video would take 500 MB / (100 mbps / 8) = 40 seconds to upload
* since we have 10 upload request per second, we have 40 \* 10 = 400 concurrent uploading, the bandwidth requirement on our side will be 400 \* 100 mbps = 40000 mbps
* total memory usage will be 40000 mbps / 8 = 5000 MB = 5GB, if our server has 2GB RAM, then we need to have 3 servers for uploading. However, because we have 400 concurrent uploading, it requires 400 threads to serve. This time the Wait time / Service time ratio is close to 1, and if we still use 2 CPU cores server running at 50% utilization, each server could support 2 \* 0.5 \* (1+1) = 2 threads. So, we need 400 / 2 = 200 servers for video uploading.

## **Design Goals**

**Latency** - Is our service latency sensitive (Or in other words, Are requests with high latency and a failing request, equally bad?)  
Yes, to provide great customer experience, latency is very important for video service

**Consistency** - Does our service require tight consistency?  
Not really, it's okay if things are eventually consistent.

**Availability** - Does this problem require 100% availability?  
Yes

## **System interface definition**

At this step, I'm designing the APIs that our service exposes to clients. Based on features and requirements I gathered at the first step, the video platform I'm going to design is apparently a web application. The best practice is to decouple the frontend and backend so that the frontend and backend can evolve independently as long as they obey the contract(backend service provides API and clients consume the API).

There are different API design style, SOAP, REST, and graphQL. I'll create a set of RESTful API as it's lightweight compared with SOAP and it's the broadly supported and the most popular one among developers.

Designing RESTful API requires us to first identity resources and then map HTTP methods to operations.

Apparently, we have at least two resources, video and user.

Our service supports operations including upload video, play video, create user, get user info. So, we can have following APIs.

Upload video API

POST /v1/videos

Request header

Content-Type: multipart/form-data

Request body

userID

videoTitle

videoDescription

language

videoFile binary data

Response body

videoProcessingJobID

The upload API uses HTTP POST method and v1/videos endpoint to upload a video file with videoTitle, videoDescription...... metadata passed in the request body. Wheter adding API versioning like "v1" in the API is still controversal but I think it'll help us evolve APIs freely. So, I use it in my design.

Using HTTP POST to upload a bianry file is not as simple as we expect. The Content-Type has to be multipart/form-data and the video file binary data will be included between boundary parameters. I'm not going to cover technical details at here. And also, we can even split the big video file and upload chunks in parallel. Again, that's implementation details.

Please note that the response of video uploading request will return a videoProcessingJobID that clients can use to check processing status. The reason is that once video is uploaded to our service, we'll do a series of time-consuming operations like deduplication, compression and create copies with different resolutions, etc.. I'd like to do them in an asynchronous way to provide a good customer experience. Client can polling with the videoProcessingJobID to check the progress later on.

The watch API is also straight forward and it's like

GET /v1/videos/<videoID>

Request header

access\_token

Response body

videoURL

As we learned earlier, authentication and authorization is out of the scope of this problem. I simply add the access\_token to the request header. Idealy, once the user is done authentication during login, an id\_token or access\_token (depends on the OIDC flow that the authentication flow is using) would be issued to the user and it'll be passed to backend, decoded and validated there.

At here, I just ignore the access control part for videos. In reality, the returned videoURL has to be either short lived or has some authorzation mechanism to prevent undesired access.

Next is the user creation and fetching user info API

POST /v1/users

Request body

userName

region

age

Response body

userID

GET /v1/users/<userID>

Request header

access\_token

Response body

userID

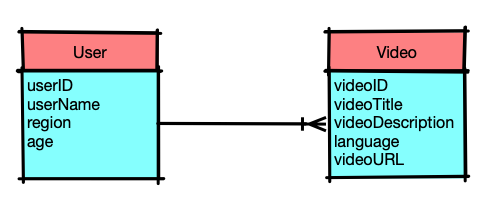
userName

region

age

## **Data Model**

After APIs are ready, let's talk about data model. We have two entities, video and user. One user can have multiple videos and one video can only belong to one user.



## **High Level Design**

Before we jump into the high level design, we need to have workflows in mind.

1. upload video
   * user upload video, once the upload is done, create a job and put it into the video post processing queue and return the job id to client for future progress checking
   * our service picks up job from the queue and conducts a serise processing, e.g. check duplication, compression
   * once post-processign is done, persist video in object storage (e.g. AWS S3 bucket)
   * persist video metadata (including video URL in S3 bucket)
2. watch video
   * user sends request to access a video by videoID
   * server return the URL of the requested video

Now, let's consider the database for user info and video metadata. Considering the scalability requirement of our service and relatively simple data model, I'm going to use NoSQL database, more specifically, AWS DynamoDB for our user info and video metadata. Using AWS S3 as the video storage.

Designing schema for NoSQL database is quite different from that of SQL database. In SQL schema, we first identify entites, denormalize them and put into tables. The relation between entities is expressed through foreign keys and the query flexibility is provided by SQL query language.

For NoSQL database, especially DynamoDB, we first identify access patterns. And then, design database schema and denormalize data if it's necessary. DynamoDB provides excellent scalability at the the cost of less access flexibility. DynamoDB recommends putting all entites into one table with carefully designed partition key and optional sort key. In case we want to support new access pattern in future, we can add GSI (Global Secondary Index) to DynamoDB, we do have flexibility to some extent.

The access patterns we'll support are

* given a videoID, access the video
* given a userID, access the user info
* given a userID, access all videos that are uploaded by the user

So, I design the DynamoDB schema as below

\*Entity\* \*Partition Key\* \*Sort Key\*

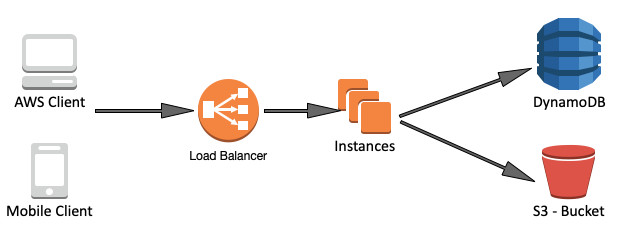
Video #VIDEO#<VideoID> #METADATA#<VideoID>

User #USER#<UserID> #METADATA#<UserID>

UserVideoMapping #USER#<VideoID> #VIDEO#<VideoID>

The #USER and #VIDEO are like name space that avoid to have collisons between entities.

Recall that we have to use at least 4 servers to support 1000 RPS, the high level design is like below



## **Detailed Design**

How to scale the architecture? Ideally, before making any decisions about scaling, we should first do performance tests against our system, monitor CPU and memory usage and latency and find the bottleneck. At here, I just assume our services will be running into common problems as other data intensive application scales.

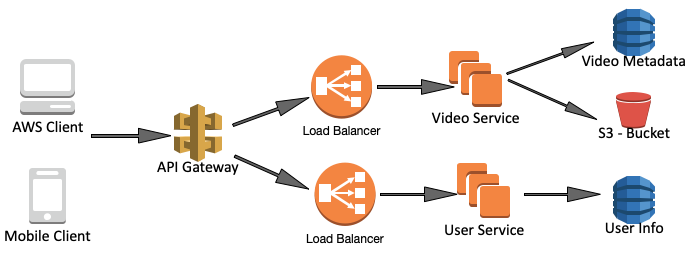
Servers and database are usually bottlenecks while visitors are increasing. Cahce can significantly reduce the presessure to servers and databases; hence, improve the scalability

* use CDN (AWS Cloudfront) to serve static resources, including videos, video thumbnails ...
* use cache (AWS DAX) for user info and video metadata access

Besides that, since I'm using DynamoDB, a managed service, as the database storage, it does all the heavy lifting for us, e.g. master-slave replication, multi-master writes, etc.. All we need to do is to setup proper auto scale settings.

The next part we want to change is the "Instances" in the above diagram. Our service is now a monolithic service, all functions are coupled together. Even though, we have multiple instances running, each functions can't scale individually. For example, the video deduplication and video compression are apparently computing intensive and require hardware with powerful CPU while the web serving part could be potentially memory intensive as it needs more RAM to serve incoming requests.

The solution is to break monolithic service into micro-services. Micro-services expose APIs to outside and they use APIs to communicate with others. As the number of services and hence, APIs increased, it's impossbile to ask clients to call each fine-grained APIs directly and we actually only want to expose stable coarse-grained APIs to clients so that client applications don't need to update their code frequently. To solve this problem, we introduce API gateway into our system. API gateway will encapsulate fine-grained APIs provided by each micro-services and provide APIs to clients.



Please note that I only drew two micro-services above; it's over-simplified. In reality, it'll definitely have more services and orchestrators that aggregate responses from subsets of micro-services. Anyway, it at least shows the basic idea that each micro-service can scale independently.

## **Summary**

I just write up a skeleton of the design. There can be numerous details to dive deep, like

* Add more features
  + how to add view count to a video
  + how to add user interaction, "Like", "+1", to a video,
  + how to follow a user
* System implementation details, with and without leveraging AWS
  + load balancing strategy, health check, service high availability
  + database sharding, fault tolerance
  + large volume of uploading requests, how to queueing them

Once again, thanks for reading, comments are appreciated.

Comments:

Trust me, you can't even cover 30% of what is described in a real interview. Interviewers usually don't care too much about trivial stuff like database model, API design and trivial services.  
System design interviews are generally focussed on particular area of a large system. I was asked a similar question in my Amazon interview.

In my case the focus was only on designing video upload and stream service and scaling (not just horizontally) it. Although interviewer didn't say that himself. I was not allowed to say “we can use CDN” :). I had to design it myself. Designing and explaining this alone takes 40 mins at least. Also, just don't blindly start capacity estimation, API design or data model design, ask your interviewer if he is interested in that.

Remember - Key is to communicate well with your interviewer and ensure he is satisfied with your knowledge. Focus on what they say and the direction they want the design to go.