Top 75 Leet Code Questions to save you time

New Year Gift to every fellow time-constrained engineer out there looking for a job, here's a list of the best LeetCode questions that teach you core concepts and techniques for each category/type of problems! Many other LeetCode questions are a mash of the techniques from these individual questions. I used this list in my last job hunt to only do the important questions.  
  
Good luck and Happy New Year!  
  
**Array**  
  
- Two Sum - <https://leetcode.com/problems/two-sum/>  
- Best Time to Buy and Sell Stock - [ttps://lehetcode.com/problems/best-time-to-buy-and-sell-stock/](https://leetcode.com/problems/best-time-to-buy-and-sell-stock/)

133-use two items, one lowest element, one is maxprofix  
- Contains Duplicate - <https://leetcode.com/problems/contains-duplicate/>

098 use unordered\_set + find(key) or insert(key)  
- Product of Array Except Self - <https://leetcode.com/problems/product-of-array-except-self/>  
l134: use res to find the pre multiplication in one loop, the use the reverse loop,temp is multiplied by nums[i] and res itself.

- Maximum Subarray - <https://leetcode.com/problems/maximum-subarray/>

l135 one for loop, set curr=max = nums[0], curr = std::max(curr+\*it,\*it); max=std::max(max,curr);

- Maximum Product Subarray - <https://leetcode.com/problems/maximum-product-subarray/>  
- Find Minimum in Rotated Sorted Array - <https://leetcode.com/problems/find-minimum-in-rotated-sorted-array/>  
- Search in Rotated Sorted Array - <https://leetcode.com/problems/search-in-rotated-sorted-array/>  
- 3Sum - <https://leetcode.com/problems/3sum/>  
- Container With Most Water - <https://leetcode.com/problems/container-with-most-water/>  
  
---  
  
**Binary**  
  
- Sum of Two Integers - <https://leetcode.com/problems/sum-of-two-integers/>

l136-two’s complement

while (b != 0)

{

int ans = a ^ b;

int carry = (a & b) << 1;

a = ans;

b = carry;

cout << a << " " << b << endl;

}

- Number of 1 Bits - <https://leetcode.com/problems/number-of-1-bits/>  
- Counting Bits - <https://leetcode.com/problems/counting-bits/>

l116

vector<int> countBits(int n)

{

vector<int> out(n + 1, 0);

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

{

int count = 0;

while (n> 0)

{

n &=(n-1); // removes the le (1<<i)?count++:0ast significant bit

++count;

}

out[i] = count;

}

return out;

}

- Missing Number - <https://leetcode.com/problems/missing-number/>

l137

// approach 3: using XOR time: O(N)-space: O(N)

// n XOR n = 0 - 0 XOR n = n - n XOR m = n XOR m.

class Solution

{

public:

int missingNumber(vector<int> nums)

{

int missing = 0, notMissing = nums.size();

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

{

missing ^= nums[i];

notMissing ^= i;

}

return missing ^ notMissing;

}

};

// approach 4:

class Solution

{

public:

int missingNumber(vector<int> nums)

{

int missing = 0, notMissing = nums.size();

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

{

missing += nums[i];

notMissing += i;

}

return notMissing - missing;

}

};

- Reverse Bits - <https://leetcode.com/problems/reverse-bits/>

l138

// approach 1

class Solution

{

public:

uint32\_t reverseBits(uint32\_t n)

{

int out = 0;

for (int i = 0; i<32; ++i)

{

int bit = n&1;

n>>=1;

out += bit<<(31-i);

// cout << bitset<32>(out) << " " << bitset<32>(n) << endl;

}

return out;

}

};

// approach 2

class Solution

{

public:

uint32\_t reverseBits(uint32\_t n)

{

n = (n>>16 | n<<16);

n = ((n&0xff00ff00)>>8 | (n&0x00ff00ff)<<8 );

n = ( (n&0xf0f0f0f0)>>4 | (n&0x0f0f0f0f)<<4 );

n = ( (n&0xcccccccc)>>2 | (n&0x33333333)<<2 );

n = ( (n&0xaaaaaaaa)>>1 | (n&0x55555555)<<1 );

return n;

}

};

---  
  
**Dynamic Programming**  
  
- Climbing Stairs - <https://leetcode.com/problems/climbing-stairs/>

int climbStairs(int n) {

if (n<=2) return n;

int out = 0;

int t1 = 1;

int t2 = 2;

for (int i = 3; i<=n; i++)

{

out = t1 + t2;

t1 = t2;

t2 = out;

}

return out;

}  
- Coin Change - <https://leetcode.com/problems/coin-change/>

class Solution {

public:

int coinChange(vector<int>& coins, int amount) {

vector<int> num(amount+1, amount+1);

num[0] = 0;

for (int i = 1; i<=amount; i++)

for (auto coin:coins)

if (i>= coin)

num[i] = min(num[i], num[i-coin]+1);

return num[amount] > amount?-1:num[amount];

}

};

- Longest Increasing Subsequence - <https://leetcode.com/problems/longest-increasing-subsequence/>  
- Longest Common Subsequence -  
- Word Break Problem - <https://leetcode.com/problems/word-break/>  
- Combination Sum - <https://leetcode.com/problems/combination-sum-iv/>  
- House Robber - <https://leetcode.com/problems/house-robber/>

class Solution {

public:

int rob(vector<int>& nums) {

std::vector<int> money(nums.size(),0);

money[0] = nums[0];

money[1] = max(nums[0], nums[1]);

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

money[i] = max(money[i-1], money[i-2]+nums[i]);

return money[nums.size()-1];

}

};

- House Robber II - <https://leetcode.com/problems/house-robber-ii/>  
- Decode Ways - <https://leetcode.com/problems/decode-ways/>  
- Unique Paths - <https://leetcode.com/problems/unique-paths/>  
- Jump Game - <https://leetcode.com/problems/jump-game/>  
  
---  
  
**Graph**  
  
- Clone Graph - <https://leetcode.com/problems/clone-graph/>

Node\* cloneGraph(Node\* node)

{

if (!node) return nullptr;

stack<Node\*> st;

map<Node\*, Node\*> myMap;

st.push(node);

Node \* newNode = new Node(node->val);

myMap.insert(pair<Node\*, Node\*>(node,newNode));

while(!st.empty())

{

Node\* orig = st.top();

st.pop();

for (auto nbr:orig->neighbors)

{

if (myMap[nbr] == nullptr)

{

st.push(nbr);

newNode = new Node(nbr->val);

myMap[nbr] = newNode;

}

myMap[orig]->neighbors.push\_back(myMap[nbr]);

}

}

return myMap[node];

}

- Course Schedule - <https://leetcode.com/problems/course-schedule/>

bool canFinish(int numCourses, vector<vector<int>>& prerequisites)

{

// adjacency list + indegree

vector<int> indegree(numCourses, 0);

vector<vector<int>> adj(numCourses);

for (auto pr : prerequisites)

{

indegree[pr[0]]++;

adj[pr[1]].push\_back(pr[0]);

}

// BFS

queue<int> mq;

// adding indegree zeros

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

if (indegree[i] == 0) mq.push(i);

int visitedCourses = 0;

while (!mq.empty())

{

int course = mq.front();

mq.pop();

visitedCourses++;

for (int j = 0; j < adj[course].size(); j++)

{

indegree[adj[course][j]]--;

if (indegree[adj[course][j]] == 0) mq.push(adj[course][j]);

}

}

return visitedCourses == numCourses;

}

- Pacific Atlantic Water Flow - <https://leetcode.com/problems/pacific-atlantic-water-flow/>  
- Number of Islands - https://leetcode.com/problems/number-of-islands/

int numIslands(vector<vector<char>>& grid) {

int row = grid.size();

if (row == 0) return 0;

int col = grid[0].size();

int out = 0;

for(int rr = 0; rr<row; rr++)

for(int cc = 0; cc<col; cc++)

if (grid[rr][cc]== '1')

{

out++;

queue<pair<int,int>> qq;

qq.push(make\_pair(rr, cc));

grid[rr][cc]='0';

while(!qq.empty())

{

pair<int,int> tmp = qq.front();

qq.pop();

if ( tmp.first+1 < row && grid[tmp.first+1][tmp.second] == '1')

{

qq.push(make\_pair(tmp.first+1,tmp.second));

grid[tmp.first+1][tmp.second]='0';

}

if ( tmp.first-1 >=0 && grid[tmp.first-1][tmp.second] == '1')

{

qq.push(make\_pair(tmp.first-1,tmp.second));

grid[tmp.first-1][tmp.second]='0';

}

if ( tmp.second+1 < col && grid[tmp.first][tmp.second+1] == '1')

{

qq.push(make\_pair(tmp.first,tmp.second+1));

grid[tmp.first][tmp.second+1]='0';

}

if ( tmp.second-1 >=0 && grid[tmp.first][tmp.second-1] == '1')

{

qq.push(make\_pair(tmp.first,tmp.second-1));

grid[tmp.first][tmp.second-1]='0';

}

}

}

return out;

}

- Longest Consecutive Sequence - <https://leetcode.com/problems/longest-consecutive-sequence/>  
- Alien Dictionary (Leetcode Premium) - <https://leetcode.com/problems/alien-dictionary/>

see graph in LC

- Graph Valid Tree (Leetcode Premium) - https://leetcode.com/problems/graph-valid-tree/

bool validTree(int n, vector<vector<int>>& edges) {

if (n -1 != edges.size()) return false;

// create adjacency list

vector<list<int>> adj(n);

for (auto edge: edges)

{

adj[edge[0]].push\_back(edge[1]);

adj[edge[1]].push\_back(edge[0]);

}

// run a dfs to see if all nodes are connected

stack<int> st;

vector<bool> visited(n, false);

st.push(0);

visited[0]=true;

while (!st.empty())

{

int tmp = st.top();

cout << tmp << endl;

st.pop();

for (auto ed:adj[tmp])

{

cout << "ed: " << ed << endl;

if (!visited[ed])

{

st.push(ed);

visited[ed]=true;

}

}

}

bool out = any\_of(begin(visited), end(visited), [](auto x){return !x;});

return !out;

}

- Number of Connected Components in an Undirected Graph (Leetcode Premium) - <https://leetcode.com/problems/number-of-connected-components-in-an-undirected-graph/>  
int countComponents(int n, vector<vector<int>>& edges) {

vector<bool> visited(n, false);

vector<vector<int>> adj(n);

// fill the adjacency list

for (auto edge:edges)

{

adj[edge[0]].push\_back(edge[1]);

adj[edge[1]].push\_back(edge[0]);

}

int islands = 0;

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

{

if (visited[i]==false)

{

islands++;

queue<int> gr;

gr.push(i);

visited[i] = true;

// cout << " ----i: " <<i << endl;

while(!gr.empty())

{

int node = gr.front();

gr.pop();

// cout << " node: " << node << endl;

for (int j = 0; j< adj[node].size(); j++)

{

int tmp = adj[node][j];

// cout << " j: " << j << " -- " << tmp<< endl;

if (visited[tmp] == false)

{

// cout << " adding to graph: " << tmp << endl;

visited[tmp] = true;

gr.push(tmp);

}

}

}

}

}

return islands;

}

---  
  
**Interval**  
  
- Insert Interval - <https://leetcode.com/problems/insert-interval/>

class Solution

{

public:

vector<vector<int>> insert(vector<vector<int>>& intervals, vector<int>& newInterval)

{

if (!intervals.size()) return {newInterval};

vector<vector<int>> out;

if (newInterval[0]>intervals[intervals.size()-1][1])

{

out = intervals;

out.push\_back(newInterval);

return out;

}

int inter = 0;

bool cond = true;

if (newInterval[0]<intervals[0][0] ) {out.push\_back(newInterval); cond = false;}

else {out.push\_back(intervals[inter++]);}

while ( inter<intervals.size() || cond)

{

vector<int> newin;

if ( inter == intervals.size() || (newInterval[0]<intervals[inter][0] && cond)) {newin = newInterval;cond = false;}

else {newin = intervals[inter++]; }

if (out[out.size()-1][1]<newin[0]) out.push\_back(newin);

else out[out.size()-1] = {min(newin[0], out[out.size()-1][0]), max(newin[1], out[out.size()-1][1]) };

cout<< newin[0] << ", " <<newin[1] << endl;

}

return out;

}

};

- Merge Intervals - <https://leetcode.com/problems/merge-intervals/>

class Solution

{

public:

vector<vector<int>> insert(vector<vector<int>>& intervals, vector<int>& newInterval)

{

if (!intervals.size()) return {newInterval};

vector<vector<int>> out;

if (newInterval[0]>intervals[intervals.size()-1][1])

{

out = intervals;

out.push\_back(newInterval);

return out;

}

int inter = 0;

bool cond = true;

if (newInterval[0]<intervals[0][0] ) {out.push\_back(newInterval); cond = false;}

else {out.push\_back(intervals[inter++]);}

while ( inter<intervals.size() || cond)

{

vector<int> newin;

if ( inter == intervals.size() || (newInterval[0]<intervals[inter][0] && cond)) {newin = newInterval;cond = false;}

else {newin = intervals[inter++]; }

if (out[out.size()-1][1]<newin[0]) out.push\_back(newin);

else out[out.size()-1] = {min(newin[0], out[out.size()-1][0]), max(newin[1], out[out.size()-1][1]) };

cout<< newin[0] << ", " <<newin[1] << endl;

}

return out;

}

};

- Non-overlapping Intervals - <https://leetcode.com/problems/non-overlapping-intervals/>

class Solution {

public:

int eraseOverlapIntervals(vector<vector<int>>& intervals)

{

vector<list<int>> ind(intervals.size());

sort(begin(intervals), end(intervals), [](auto x, auto y){return x[1]<y[1];});

int ret=0;

int k = INT\_MIN;

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

if (intervals[i][0] >= k) k = intervals[i][1];

else ret++;

return ret;

}

};

- Meeting Rooms (Leetcode Premium) - https://leetcode.com/problems/meeting-rooms/

bool canAttendMeetings(vector<vector<int>>& intervals)

{

if (intervals.size() == 0) return true;

sort(begin(intervals), end(intervals), [](auto x, auto y) { return x[0] < y[0]; });

for (int i = 0; i < intervals.size() - 1; i++)

if (intervals[i + 1][0] < intervals[i][1]) return false;

return true;

}

- Meeting Rooms II (Leetcode Premium) - <https://leetcode.com/problems/meeting-rooms-ii/>  
int minMeetingRooms(vector<vector<int>>& intervals) {

vector<int> starts(intervals.size()), ends(intervals.size());

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

{

starts[i] = intervals[i][0];

ends[i] = intervals[i][1];

}

sort(begin(starts), end(starts));

sort(begin(ends), end(ends));

int rooms = 0;

int idxEnd = 0;

for (int idxStart = 0; idxStart < intervals.size(); idxStart++)

{

if (starts[idxStart]<ends[idxEnd]) rooms++;

else idxEnd++;

}

return rooms;

}

---  
  
**Linked List**  
  
- Reverse a Linked List - <https://leetcode.com/problems/reverse-linked-list/>

ListNode \* reverseList(ListNode \*head)

{

// return if head is null

if (!head) return head;

// traverse the list

ListNode \*curr = head, \*currNext =curr->next;

while(currNext)

{

ListNode \*tmp = currNext->next;

currNext->next = curr;

curr = currNext;

currNext = tmp;

}

return curr;

}

- Detect Cycle in a Linked List - <https://leetcode.com/problems/linked-list-cycle/>

bool hasCycle(ListNode\* head)

{

if (!head)

return false;

ListNode\* slow = head, \*fast=head->next;

// cout <<" starts-slow: " << slow->val << " fast: "<< fast->val << endl;

while(fast && fast->next)

{

// cout <<"slow: " << slow->val << " fast: "<< fast->val << endl;

if (slow==fast)

return true;

slow=slow->next;

fast= fast->next->next;

}

return false;

}

- Merge Two Sorted Lists - <https://leetcode.com/problems/merge-two-sorted-lists/>

ListNode\* mergeTwoLists(ListNode \*list1, ListNode \*list2)

{

if (!list1)

return list2;

else if (!list2)

return list1;

ListNode \*head= nullptr;

if (list1->val < list2->val)

{

head = list1;

list1 = list1->next;

}

else

{

head = list2;

list2 = list2->next;

}

ListNode \*tmp=head;

while(list1 && list2)

{

if (list1->val < list2->val)

{

tmp->next = list1;

list1 = list1->next;

}

else

{

tmp->next = list2;

list2 = list2->next;

}

tmp = tmp->next;

}

if (!list1)

tmp->next= list2;

else if (!list2)

tmp->next = list1;

// tmp→next = !list1?list2:list1;

return head;

}

- Merge K Sorted Lists - https://leetcode.com/problems/merge-k-sorted-lists/

ListNode \*mergeKLists(vector<ListNode\*> &lists)

{

ListNode\* head=nullptr;

ListNode\* curr=nullptr;

int max = 100000;

int ind = -1;

bool allNull = true;

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

{

if (lists[i] && lists[i]->val<max)

{

max = lists[i]->val;

ind = i;

allNull=false;

}

}

if (ind>-1)

{

head = lists[ind];

// head->next = nullptr;

lists[ind]=lists[ind]->next;

}

curr = head;

while(!allNull)

{

allNull = true;

max = 10000;

ind = -1;

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

{

if (lists[i] && lists[i]->val<max)

{

max = lists[i]->val;

ind = i;

allNull=false;

}

}

if (ind>-1)

{

// cout << " adding: "<< ind << " " << lists[ind]->val << " " << max << endl;

curr->next = lists[ind];

lists[ind]=lists[ind]->next;

curr = curr->next;

// curr->next = nullptr;

}

// printList(head);

}

// cout << " end of sol\n";

// printList(head);

return head;

}

- Remove Nth Node From End Of List - https://leetcode.com/problems/remove-nth-node-from-end-of-list/

class Solution {

public:

ListNode \*removeNthFromEnd(ListNode \*head, int n) {

if (head == nullptr)

return head;

ListNode \*cur = head;

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

cur = cur->next;

if (cur == nullptr) {

cur = head;

head = head->next;

delete cur;

return head;

}

ListNode \*pre = head;

while (cur->next != nullptr) {

cur = cur->next;

pre = pre->next;

}

cur = pre->next;

pre->next = pre->next->next;

delete cur;

return head;

}

};

- Reorder List - <https://leetcode.com/problems/reorder-list/>  
void reorderList(ListNode\* head)

{

if (!head->next) return ;

// find middle node

ListNode \*slow = head, \*fast = head;

while(fast && fast->next)

{

slow = slow->next;

fast = fast->next->next;

}

// reverse the second half

ListNode \*pre = slow;

ListNode \*cur = slow->next;

pre->next = nullptr;

while (cur)

{

ListNode \* tmp = cur->next;

cur->next = pre;

pre = cur;

cur = tmp;

}

ListNode\* start = head;

ListNode \* end = pre;

while (start->next && end->next)

{

ListNode \*tmp1 = end->next;

end->next = start->next;

start->next = end;

start = end->next;

end = tmp1;

}

}

---  
  
**Matrix**  
  
- Set Matrix Zeroes - <https://leetcode.com/problems/set-matrix-zeroes/>

public void setZeroes(int[][] matrix) {

Boolean isCol = false;

int R = matrix.length;

int C = matrix[0].length;

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

// Since first cell for both first row and first column is the same i.e. matrix[0][0]

// We can use an additional variable for either the first row/column.

// For this solution we are using an additional variable for the first column

// and using matrix[0][0] for the first row.

if (matrix[i][0] == 0) {

isCol = true;

}

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

// If an element is zero, we set the first element of the corresponding row and column to 0

if (matrix[i][j] == 0) {

matrix[0][j] = 0;

matrix[i][0] = 0;

}

}

}

// Iterate over the array once again and using the first row and first column, update the elements.

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

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

if (matrix[i][0] == 0 || matrix[0][j] == 0) {

matrix[i][j] = 0;

}

}

}

// See if the first row needs to be set to zero as well

if (matrix[0][0] == 0) {

for (int j = 0; j < C; j++) {

matrix[0][j] = 0;

}

}

// See if the first column needs to be set to zero as well

if (isCol) {

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

matrix[i][0] = 0;

}

}

}  
- Spiral Matrix - <https://leetcode.com/problems/spiral-matrix/>

class Solution

{

public:

vector<int> spiralOrder(vector<vector<int>> &matrix)

{

int m = matrix.size();

int n = matrix[0].size();

int rowBoundaryTop = 0, rowBoundaryBot = m-1;

int colBoundaryLeft = 0, colBoundaryRight = n-1;

vector<int> out(m\*n,-1);

int counter = 0;

while (counter<m\*n)

{

for (int col = colBoundaryLeft; col<=colBoundaryRight; ++col)

out[counter++] = matrix[rowBoundaryTop][col];

for (int row = rowBoundaryTop+1; row<=rowBoundaryBot; ++row)

out[counter++] = matrix[row][colBoundaryRight];

if (rowBoundaryBot != rowBoundaryTop)

for (int col=colBoundaryRight-1; col>=colBoundaryLeft; --col)

out[counter++] = matrix[rowBoundaryBot][col];

if (colBoundaryLeft != colBoundaryRight)

for (int row = rowBoundaryBot-1; row>rowBoundaryTop; --row)

out[counter++] = matrix[row][colBoundaryLeft];

++rowBoundaryTop;

--rowBoundaryBot;

++colBoundaryLeft;

--colBoundaryRight;

}

return out;

}

};

- Rotate Image - <https://leetcode.com/problems/rotate-image/>

class Solution

{

public:

void rotate(vector<vector<int>> &matrix)

{

int n = matrix.size();

// transpose

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

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

{

int tmp = matrix[i][j];

matrix[i][j] = matrix[j][i];

matrix[j][i] = tmp;

}

// reverse

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

for(int j = 0; j<n/2; ++j)

{

int tmp = matrix[i][j];

matrix[i][j] = matrix[i][n-1-j];

matrix[i][n-1-j] = tmp;

}

}

};

- Word Search - <https://leetcode.com/problems/word-search/>  
  
---  
  
**String**  
  
- Longest Sub string Without Repeating Characters - <https://leetcode.com/problems/longest-substring-without-repeating-characters/>

- Longest Repeating Character Replacement - <https://leetcode.com/problems/longest-repeating-character-replacement/>  
- Minimum Window Substring - <https://leetcode.com/problems/minimum-window-substring/>  
- Valid Anagram - <https://leetcode.com/problems/valid-anagram/>  
- Group Anagrams - <https://leetcode.com/problems/group-anagrams/>  
- Valid Parentheses - <https://leetcode.com/problems/valid-parentheses/>  
- Valid Palindrome - <https://leetcode.com/problems/valid-palindrome/>  
- Longest Palindromic Substring - <https://leetcode.com/problems/longest-palindromic-substring/>  
- Palindromic Substrings - <https://leetcode.com/problems/palindromic-substrings/>  
- Encode and Decode Strings (Leetcode Premium) - <https://leetcode.com/problems/encode-and-decode-strings/>  
  
---  
  
**Tree**  
  
- Maximum Depth of Binary Tree - https://leetcode.com/problems/maximum-depth-of-binary-tree/

int maxDepth(TreeNode\* root) {

if (!root) return 0;

int depth = 1;

queue<mypair> qq;

qq.push( make\_pair(root,1));

while(!qq.empty())

{

mypair temp = qq.front();

qq.pop();

if (temp.first->left) qq.push( make\_pair(temp.first->left, temp.second +1));

if (temp.first->right) qq.push( make\_pair(temp.first->right, temp.second+1));

depth = max(depth,temp.second);

cout << temp.first->val << " - " << temp.second << endl;

}

return depth;

}

- Same Tree - <https://leetcode.com/problems/same-tree/>

bool isSameTree(TreeNode \*p, TreeNode \*q)

{

if (!p && !q ) return true;

if (!p || !q) return false;

queue<TreeNode\*> qOne, qTwo;

qOne.push(p);

qTwo.push(q);

while (!qOne.empty() && !qTwo.empty())

{

TreeNode \* tmp1 = qOne.front();

TreeNode \* tmp2 = qTwo.front();

qOne.pop();

qTwo.pop();

if (tmp1->val != tmp2->val) return false;

if (tmp1->left && tmp2->left)

{

qOne.push(tmp1->left);

qTwo.push(tmp2->left);

}

else if ((!tmp1->left && tmp2->left) || (tmp1->left && !tmp2->left) ) return false;

if (tmp1->right && tmp2->right)

{

qOne.push(tmp1->right);

qTwo.push(tmp2->right);

}

else if ((!tmp1->right && tmp2->right) || (tmp1->right && !tmp2->right) ) return false;

}

return (qOne.empty() && qTwo.empty()) ;

}

- Invert/Flip Binary Tree - <https://leetcode.com/problems/invert-binary-tree/>

TreeNode \* invertTree(TreeNode \*root)

{

if (!root) return root;

queue<TreeNode\*> mq;

mq.push(root);

while (!mq.empty())

{

TreeNode \*tmp = mq.front();

mq.pop();

TreeNode \* node = tmp->left;

tmp->left = tmp->right;

tmp->right = node;

if (tmp->left) mq.push(tmp->left);

if (tmp->right) mq.push(tmp->right);

}

return root;

}

- Binary Tree Maximum Path Sum - <https://leetcode.com/problems/binary-tree-maximum-path-sum/>

- Binary Tree Level Order Traversal - <https://leetcode.com/problems/binary-tree-level-order-traversal/>

using myp = pair<TreeNode \*, int>;

class Solution {

public:

vector<vector<int>> levelOrder(TreeNode\* root) {

vector<vector<int>> out;

queue<myp> myqq;

if (root) myqq.push(make\_pair(root, 0));

while (!myqq.empty())

{

myp temp = myqq.front();

myqq.pop();

if (temp.first->left) myqq.push(make\_pair(temp.first->left,temp.second+1));

if (temp.first->right) myqq.push(make\_pair(temp.first->right,temp.second+1));

if (out.empty() || out.size()<=temp.second) out.push\_back({});

out[temp.second].push\_back(temp.first->val);

}

return out;

}

};

- Serialize and Deserialize Binary Tree - https://leetcode.com/problems/serialize-and-deserialize-binary-tree/

class Codec

{

public:

// Encodes a tree to a single string.

string serialize(TreeNode \*root)

{

string out = "";

if (!root) return out;

queue<TreeNode \*> mq; // level order

mq.push(root);

while (!mq.empty())

{

TreeNode \*node = mq.front();

mq.pop();

if (node)

{

out += (to\_string(node->val) + " ");

mq.push(node->left);

mq.push(node->right);

}

else

out += "+ ";

}

return out;

}

TreeNode \*nextNode(int &counter, string &data)

{

TreeNode \*node = nullptr;

int start = ++counter;

while (data[counter] != ' ') counter++;

string nodeval = data.substr(start, counter - start);

// cout <<start<< " " << counter<< " nodeval: " << nodeval << endl;

if (nodeval != "+")

{

node = new TreeNode(stoi(nodeval));

}

return node;

}

// Decodes your encoded data to tree.

TreeNode \*deserialize(string data)

{

int len = data.size();

TreeNode \*root = nullptr;

if (len == 0) return root;

queue<TreeNode \*> mq;

int counter = -1;

root = nextNode(counter, data);

mq.push(root);

while (!mq.empty())

{

TreeNode \*tmp = mq.front();

mq.pop();

if (!tmp) continue;

tmp->left = nextNode(counter, data);

mq.push(tmp->left);

tmp->right = nextNode(counter, data);

mq.push(tmp->right);

}

return root;

}

};  
- Subtree of Another Tree - <https://leetcode.com/problems/subtree-of-another-tree/>

class Solution

{

public:

bool isSameTree(TreeNode \*p, TreeNode \*q)

{

if (!p && !q) return true;

if (!p || !q) return false;

queue<TreeNode \*> qOne, qTwo;

qOne.push(p);

qTwo.push(q);

while (!qOne.empty() && !qTwo.empty())

{

TreeNode \*tmp1 = qOne.front();

TreeNode \*tmp2 = qTwo.front();

qOne.pop();

qTwo.pop();

if (tmp1->val != tmp2->val) return false;

if (tmp1->left && tmp2->left)

{

qOne.push(tmp1->left);

qTwo.push(tmp2->left);

}

else if ((!tmp1->left && tmp2->left) || (tmp1->left && !tmp2->left))

return false;

if (tmp1->right && tmp2->right)

{

qOne.push(tmp1->right);

qTwo.push(tmp2->right);

}

else if ((!tmp1->right && tmp2->right) || (tmp1->right && !tmp2->right))

return false;

}

return (qOne.empty() && qTwo.empty());

}

bool isSubtree(TreeNode \*root, TreeNode \*subRoot)

{

// queue<TreeNode\*> mq;

stack<TreeNode \*> mq;

mq.push(root);

while (!mq.empty())

{

// TreeNode \* node = mq.front();

TreeNode \*node = mq.top();

mq.pop();

// cout << node->val << " " << subRoot->val << endl;

if (isSameTree(node, subRoot)) return true;

if (node->left) mq.push(node->left);

if (node->right) mq.push(node->right);

}

return false;

}

};

- Construct Binary Tree from Preorder and Inorder Traversal - <https://leetcode.com/problems/construct-binary-tree-from-preorder-and-inorder-traversal/>

- Validate Binary Search Tree - https://leetcode.com/problems/validate-binary-search-tree/

bool isValidBST(TreeNode \*root)

{

if (!root) return true;

// DFS:in-order

stack<TreeNode \*> mys;

vector<int> out;

TreeNode \*cur = root;

while (cur || !mys.empty())

{

while (cur)

{

mys.push(cur);

cur = cur->left;

}

cur = mys.top();

mys.pop();

out.push\_back(cur->val);

cur = cur->right;

}

for (auto it = begin(out); it != end(out) - 1; it++)

if (\*it >= \*(it + 1)) return false;

return true;

}

- Kth Smallest Element in a BST - https://leetcode.com/problems/kth-smallest-element-in-a-bst/

int kthSmallest(TreeNode\* root, int k) {

stack<TreeNode\*> ms;

int counter = 0;

TreeNode \*cur = root;

while(cur || !ms.empty())

{

while(cur)

{

ms.push(cur);

cur=cur->left;

}

cur = ms.top();

ms.pop();

counter++;

if (counter==k) return cur->val;

cur=cur->right;

}

return 0;

}

- Lowest Common Ancestor of BST - https://leetcode.com/problems/lowest-common-ancestor-of-a-binary-search-tree/

TreeNode\* lowestCommonAncestor(TreeNode\* root, TreeNode\* p, TreeNode\* q)

{

TreeNode\* node = root;

while ( node )

{

if ( p->val > root->val && q->val > root->val ) root = root->right;

else if (p->val < root->val && q->val < root->val ) root = root->left;

else return node;

}

return nullptr;

}

- Implement Trie (Prefix Tree) - <https://leetcode.com/problems/implement-trie-prefix-tree/>  
- Add and Search Word - <https://leetcode.com/problems/add-and-search-word-data-structure-design/>  
- Word Search II - <https://leetcode.com/problems/word-search-ii/>  
  
---  
  
**Heap**  
  
- Merge K Sorted Lists - <https://leetcode.com/problems/merge-k-sorted-lists/>  
- Top K Frequent Elements - https://leetcode.com/problems/top-k-frequent-elements/

vector<int> topKFrequent(vector<int>& nums, int k)

{

if (k == nums.size()) return nums;

vector<int> out(k,0);

unordered\_map<int, int> freq;

for (auto num : nums)

freq[num] +=1;

vector<list<int>> buckets(nums.size());

for (auto p:freq)

buckets[p.second-1].push\_back(p.first);

int i = 0;

for(auto it = rbegin(buckets); it!=rend(buckets) && i<k; it++)

for (auto elem : \*it)

out[i++] = elem;

return out;

}

- Find Median from Data Stream - <https://leetcode.com/problems/find-median-from-data-stream/>

---------

shortest path

vector<vector<int>> allPathsSourceTarget(vector<vector<int>>& graph) {

vector<vector<int>> paths;

vector<int> path;

queue<vector<int>> myq;

int nVertices = graph.size();

path = {0};

myq.push(path);

while(!myq.empty())

{

path = myq.front();

myq.pop();

int lastNode = path.back();

if ( lastNode == nVertices-1)

paths.push\_back(path);

for (auto edges: graph[lastNode])

{

vector<int> tmp = path;

tmp.push\_back(edges);

myq.push(tmp);

}

}

return paths;

}