[**Fibonacci**](#_pe0g0yjjv4q4) **6**

[**Word pattern, LC 290**](#_73pxm3ovqcd6) **6**

[**Generate minesweeper board**](#_agyy842iq4cb) **7**

[Follow up: LC 529](#_vbyzi5rfeooe) 8

[**Read N given read4, call multiple times, LC 158**](#_c6l6j0j3y8eq) **8**

[**Find largest K numbers in an array, LC 215(similar)**](#_m0ixbsmj8xs7) **8**

[**BST iterator, LC 173**](#_qiun71iyy65b) **9**

[**Dungeon game, LC 174**](#_a2p5fv1luhtp) **11**

[**Determine if array is monotonic**](#_kwcncdm9aobh) **11**

[**Walls and gates, LC 286**](#_fgig1fldqk0x) **12**

[**Decode ways, LC 91**](#_asnjt9fvwwxq) **13**

[**Minimum size subarray sum, LC 209**](#_t8ldck98dm6p) **13**

[**Word ladder, LC 127**](#_quo1vp6mh3o9) **14**

[**Remove duplicate characters in a string and print it in alphabetical order**](#_h67itkov6w1b) **15**

[Follow up: LC 316, Remove duplicate letters](#_iw5wz0bhtbco) 15

[**Sliding window maximum LC 239**](#_20dpyg44260r) **15**

[**Add binary strings**](#_f8zvwqvvyzh) **16**

[Follow-up: decimal(LC 415) or hex strings](#_q6qzbg4o158b) 17

[**Tic-tac-toe, LC 348**](#_gkey10n40qw5) **17**

[**Letter combinations of a phone number, LC 17**](#_o4j1bv5r8578) **17**

[**Minimum depth of binary tree, LC 111**](#_8y9nkvqwj7zt) **18**

[**Find the celebrity, LC 277**](#_8y8vk0w423gm) **19**

[**Continuous subarray sum, LC 523**](#_a4g683ho63bu) **19**

[**Minimum window substring, LC 76**](#_43bjo7hacp41) **20**

[**Inorder BST successor, LC 285**](#_eaeswwp38348) **20**

[**Combination sum, LC 39**](#_ynhjje1l9foo) **21**

[**Move zeros, LC 283**](#_qib0yb28mp6b) **22**

[Follow up, minimal swap/assign.](#_b8hewka0god1) 23

[**Find path in matrix with obstacles.**](#_9h1c0p3rlbn5) **23**

[**Friend circles, LC 547**](#_an9j37ypoisp) **23**

[**Friend ID**](#_gk53jryx9j4v) **25**

[**Validate BST, LC 98**](#_d9gnyqlzjuy0) **25**

[**Remove duplicate letters, LC 316**](#_kb879ehyyamu) **27**

[**Longest increasing subsequence, LC 300**](#_pz1x0iaph031) **28**

[**Implement strStr(), LC 28**](#_qfe869kiezgv) **29**

[**Task scheduler, LC 621**](#_6czo311emx25) **29**

[**Follow up: need to main original order.**](#_z3o9fc33aren) **29**

[**Schedule meeting**](#_wixcgyt2v0n5) **30**

[**Remove duplicate strings in a string list**](#_ggvmv2bn7ki5) **30**

[**Rearrange strings k distance apart, LC 358**](#_kaha0q8llaxw) **30**

[**Binary tree level order traversal, LC 102**](#_pey8q2z6z6c) **32**

[**Previous permutation**](#_1mhb20updqy4) **32**

[**Next permutation, LC 31**](#_t3bzkxzhhstb) **33**

[**Intersection of two arrays, no duplicate, LC 349**](#_iqbpe0h2iu0y) **33**

[**Intersection of two arrays, include duplicates, LC 350**](#_umou7rfazrzn) **34**

[**Flatten binary tree into a linked list, LC 114**](#_hmvagzxkabu) **35**

[**Convert a binary tree to a doubly-linked list**](#_p18ut8yhx75w) **36**

[**Convert binary tree to circular doubly linked list**](#_gkztz4gka2h) **36**

[**Find Kth largest element in an array, LC 215**](#_8wafb1ab5499) **36**

[**Replace “%20” in a string to space**](#_2u4j4uahbhsf) **37**

[**Equation evaluation, LC 640**](#_y5n3jdn3m2bn) **37**

[**Longest palindrome of a string, LC 409**](#_1yyf63czcgou) **38**

[**Valid palindrome, LC 125**](#_ka3cvyw462b1) **38**

[**Longest arithmetic progression sequence**](#_t555dpbwouuj) **39**

[**Best time to buy and sell stock, LC 121**](#_7ekjqsukceh3) **40**

[**Pascal’s triangle, LC 118**](#_4fs0pgt6wmkf) **40**

[**Expression add operators (+ or -)**](#_gwno1lp4mwlu) **41**

[**Expression add operators (+ or - or \*), LC 282**](#_oka06s846dqq) **41**

[**Intersection of K sorted arrays**](#_1s40otao8lx9) **42**

[**Remove duplicates from sorted list, LC 83**](#_i1gtdzws4tz2) **42**

[**Remove duplicates from sorted list II, LC 82**](#_6rmyfzyp31v6) **43**

[**Linked list cycle II, LC 142**](#_nqnuv2o6yvs1) **44**

[**Word break, LC 139**](#_i7bc23djwumm) **45**

[**N-Queens, LC 51**](#_e76nx5qcahw2) **45**

[Follow up, LC 52, return # of results](#_xncghudxtq51) 46

[**Super ugly number, LC 313**](#_agyb7w5r6xif) **47**

[**Intersection of two linked lists, LC 160**](#_x9q0107ihjgq) **47**

[**Sparse matrix multiplication, LC 311**](#_p0on8c5i13zx) **49**

[**Lowest common ancestor, LC 236**](#_hvljr7bkgkw) **49**

[**Subset sum**](#_510251dzi5sq) **50**

[**Three sum, LC 15**](#_1gr2rdg3fnai) **50**

[**Remove invalid parenthesis, LC 301**](#_5lh2ody1oo2q) **51**

[**Longest valid parentheses, LC 32:**](#_oskiznb3kqkg) **53**

[**Add and search word, LC 211**](#_u9d0fc7wb40l) **55**

[**Unique path II, LC 63**](#_8kjdliue217m) **56**

[**Reverse linked list, LC 206**](#_gld9u4lxrh8p) **56**

[**Merge two sorted lists, LC 21**](#_ek0jfmeuz0tr) **57**

[**Merge K sorted lists, LC 23**](#_8nrrrqn2a3es) **57**

[**Regular expression, LC 10**](#_pt9vtgssw87l) **58**

[**Flatten nested list iterator, LC 341**](#_aorn8nkrhysh) **58**

[**Flatten multi-level linked list**](#_n25fwunshkyx) **59**

[**Word search, LC 79**](#_268kaixd3o0w) **59**

[**Valid IP, LC 468**](#_xaepyawvdsls) **60**

[**Serialize and deserialize a binary tree, LC 297**](#_yqek75osl9xu) **61**

[**Josephus problem**](#_jvoqebchqgi8) **63**

[**Binary search tree range sum**](#_a4m7g8wj08z9) **64**

[**Strobogrammatic number, LC 246**](#_4jnyjotlehkg) **64**

[**Strobogrammatic number II, LC 247**](#_7nqgmrrfv0n3) **65**

[**Number of islands, LC 200**](#_49mcii8faenf) **65**

[**Course schedule, LC 207**](#_ajqq0pehkzyk) **66**

[**Course schedule II, LC 210**](#_31o9n2ancxk5) **66**

[**Populate next right pointer, LC 116**](#_5aydrc7w8022) **67**

[**Sparse vector dot product**](#_klo62t5boyjw) **67**

[**Interweave list of list**](#_p73mkjpu7tir) **68**

[**Reachable**](#_3my0yw6bvife) **68**

[**Largest cross**](#_yaam0b2jvh3o) **69**

[**Reorder List, LC 143**](#_cjuf9b4lxtt0) **69**

[**Preorder to inorder traversal of BST**](#_d1rm6bu7i68v) **70**

[**2D sum**](#_zgirxk9bpzxi) **70**

[**Buy and sell stock, LC 121**](#_9rgopnbb2e6g) **71**

[**One edit distance, LC 161**](#_ucbt061jdz3b) **71**

[**Meeting rooms II, LC 253**](#_8bqfhptjnwcl) **71**

[**Divide two strings**](#_edude26twjl0) **73**

[**Divide two integers, LC 29**](#_rn4r3sv2irav) **73**

[**Subsets, LC 78**](#_chnf7me7gs49) **74**

[**Binary tree vertical traversal, LC 314**](#_nbx27alrjv19) **75**

[**Integer to English words, LC 273**](#_mjfv1dh26uj9) **75**

[**Range sum query 2D, LC 304**](#_2vxsp36q2h8f) **76**

[**Construct binary tree from string, LC 536**](#_g6bj8kh1c5fh) **77**

[**Alien dictionary, LC 269**](#_sxuw1or3vhx4) **77**

[**Coin change, LC 322**](#_v6k6l7v13hp4) **79**

[**Binary tree paths, LC 257**](#_rde1fy7xe46z) **80**

[**Search for a range, LC 34**](#_bjpm7f29nz53) **81**

[**Find K-closest points to the origin**](#_22yxytgq49s4) **82**

[**Trapping rain water, LC 42**](#_3d5rn6bpvk2z) **82**

[**First bad version, LC 278**](#_jm7ipyugncmp) **83**

[**Symmetric tree, LC 101**](#_bo0i2m99h29v) **83**

[**Diameter of binary tree, LC 543**](#_duuldptb97m8) **84**

[**Subtree of another tree, LC 572**](#_ehybojda3g7c) **84**

[**Shortest unsorted continuous array, LC 581**](#_v2p8m76h9xx2) **85**

[**Remove duplicates from sorted array, LC 26**](#_oyb3ax45mqwe) **86**

[**Excel sheet column title, LC 168**](#_apzcy8o0noii) **87**

[**Construct the rectangle, LC 492**](#_ew53qqbj0vlz) **87**

[**Permutations, LC 46**](#_djrirmtpry74) **87**

[**Permutation II, LC 47**](#_dkthe5h2z136) **88**

[**Product of array except self, LC 238**](#_a86vs6b1gpr6) **89**

[**Count and say, LC 38**](#_ky96hyr9636o) **90**

[**Summary ranges, LC 228**](#_t8ruwxky42ib) **90**

[**Add digits, LC 258**](#_gm7dmc1688pd) **90**

[**Number complement, LC 476**](#_uls91zeoj0rk) **91**

[**Longest consecutive sequence, LC 128**](#_ryvu1dxhb7ck) **91**

[**Insert, Delete and getRandom O(1), LC 380**](#_8drl5d5rubqm) **92**

[**Wildcard matching, LC 44**](#_llg5v5u5pg7s) **93**

[**Construct binary tree from pre/in-order traversal, LC 105**](#_of13m3g57cv9) **94**

[**Word search II, LC 212**](#_tjbf0gdmo235) **94**

[**Ransom node, LC 383**](#_7exb7gkffyrw) **95**

[**Two Sum IV - Input is a BST, LC 653**](#_c9wvkcral4h) **95**

[**Exclusive time of functions, LC 636**](#_8p4sx6hjr8ds) **97**

[**Multiply string, LC 43**](#_rld62rbzb9c6) **98**

[**Unique user given email**](#_vfysnh7t72ch) **99**

[**Find island with largest perimeter**](#_5a2vnulw8q5y) **99**

[**Onsite ML SWE interview Qs**](#_kkeze11u5m5i) **99**

[**Onsite SE**](#_r0nioobp7d9w) **100**

# 

# 

# Fibonacci

|  |
| --- |
| Int fib(int n,vector<int>& f) {  If (f[n] != -1) return f[n];  If ( n < 2 ) { f[n] = 1; return f[n]; }  F[n] = fib(n-1,f)+fib(n-2,f);  Return f[n];  } |

Time complexity: O(n)

Space complexity: O(n)

|  |
| --- |
| Int fib(int n) {  If ( n < 2) return 1;  Int pprev = 1, prev = 1;  for(int i = 2; i <=n; i++) {  Int cur = pprev + prev;  Pprev = prev;  Prev = cur;  }  Return prev;  } |

Time complexity: O(n)

Space complexity: O(1)

# Word pattern, LC 290

|  |
| --- |
| class Solution {  public:  bool wordPattern(string pattern, string str) {  unordered\_map<char,string> c2s;  unordered\_map<string,char> s2c;  stringstream ss(str);  vector<string> words;  for(string word; getline(ss,word,' '); words.push\_back(word));  if (words.size() != pattern.length()) return false;  for(int i = 0; i < words.size(); i++) {  if (!check(c2s,s2c,pattern[i],words[i])) return false;  }  return true;  }  private:  bool check(unordered\_map<char,string>& c2s,unordered\_map<string,char>& s2c, char c, string word) {  if ( s2c.find(word) != s2c.end() && s2c[word] == c && c2s[c] == word) return true;  else if ( s2c.find(word) == s2c.end() && c2s.find(c) == c2s.end() ) {  c2s[c] = word;  s2c[word] = c;  return true;  }else return false;  }  }; |

Time complexity: O(n)

Space complexity: O(n)

|  |
| --- |
| **bool** **wordPattern**(string pattern, string str) {  map<**char**, **int**> p2i;  map<string, **int**> w2i;  istringstream **in**(str);  **int** i = 0, n = pattern.size();  **for** (string word; in >> word; ++i) {  **if** (i == n || p2i[pattern[i]] != w2i[word])  **return** false;  p2i[pattern[i]] = w2i[word] = i + 1;  }  **return** i == n; } |

Time complexity: O(n)

Space complexity: O(1)

# Generate minesweeper board

Given height H and width W, and the number of bombs (label as -1) M. Also fill the blanks with the number of bombs in the neighborhood (8 directions).

|  |
| --- |
| vector<vector<int>> genBoard(int H, int W, int bombs) {  vector<vector<int>> board(H,vector<int>(W,0));  //1. Generate bombs positions  Int n = bombs;  while( n > 0 ) {  Int curh = rand()%H, curw = rand()%W;  while(board[curh][curw] != 0) { curh = rand()%H; curw = rand()%W; }  Board[curh][curw] = -1;  n--;  }  //2. Update board with number of bombs in the neighborhood  for(int i = 0; i < H; i++)  for(int j = 0; j < W; j++) {    for(int dh = -1; dh < 2; dh++)  for(int dw = -1; dw < 2; dw++) {  Int ni = i+dh, nj = j+dw;  If ( (ni == i && nj == j) || i < 0 || j < 0 || i == H || j == W) continue;  If (board[ni][nj] == -1) board[i][j]++;  }  }  Return board;  } |

## Follow up: LC 529

# Read N given read4, call multiple times, LC 158

|  |
| --- |
| // Forward declaration of the read4 API.  int read4(char \*buf);  class Solution {  public:  /\*\*  \* @param buf Destination buffer  \* @param n Maximum number of characters to read  \* @return The number of characters read  \*/  int read(char \*buf, int n) {  int total = 0;  if ( i == in ) { in = read4(temp); i = 0; }  while( i <= in && total < n ) {  if ( i == in ) { in = read4(temp); i = 0; if ( in == 0 ) return total;}  buf[total] = temp[i];  i++;  total++;  }  return total;  }  char temp[4];  int i = 0, in = 0;  }; |

Time complexity: O(total)

Space complexity: O(1)

# Find largest K numbers in an array, LC 215(similar)

Solution 1: Need the numbers be in order

|  |
| --- |
| vector<int> findK(vector<int> arr,int k) {  priority\_queue<int> mypq(int,vector<int>,greater<int>);  for(auto & n : arr) {  If (mypq.size() < k) mypq.insert(n);  Else {  Int top = mypq.top();  If ( top < n) { mypq.pop(); mypq.insert(n); }  }  }  vector<int> res(k,0);  for(int i = k; i > 0; i--) { res[i-1] = mypq.top(); mypq.pop(); }  Return res;  } |

Time complexity: O(n)

Space complexity: O(k)

Solution 2: No need to maintain order

|  |
| --- |
| vector<int> findK(vector<int> arr,int k) {  Int lo = 0, hi = arr.size()-1;  while(true) {  Int pivot = arr[hi];  Int i = lo-1;  for(int j = lo; j < hi; j++) {  If ( arr[j] >= pivot ) swap(arr[++i],arr[j]);  }  swap(arr[i+1],arr[hi]);  If (i+1 == k-1) break;  Else if (i+1 > k-1) { hi = i; }  Else lo = i+2;  }  arr.resize(k);  Return arr;  } |

Time complexity: O(nlogn)

Space complexity: O(1)

# BST iterator, LC 173

Solution 1:

Use stack

|  |
| --- |
| /\*\*  \* Definition for binary tree  \* struct TreeNode {  \* int val;  \* TreeNode \*left;  \* TreeNode \*right;  \* TreeNode(int x) : val(x), left(NULL), right(NULL) {}  \* };  \*/  class BSTIterator {  public:  BSTIterator(TreeNode \*root) {  TreeNode \*p = root;  while (p) {  mystack.push(p);  p = p->left;  }  }  /\*\* @return whether we have a next smallest number \*/  bool hasNext() {  return !mystack.empty();  }  /\*\* @return the next smallest number \*/  int next() {  if (mystack.empty()) { return 0; }  TreeNode\* temp = mystack.top();  int res = temp->val;  mystack.pop();  //Add path to next number to stack  // temp's right subtree has children. Push path to the smallest child in its right subtree.  temp = temp->right;  while (temp) {  mystack.push(temp);  temp = temp->left;  }  return res;  }  private:  stack<TreeNode\*> mystack;  };  /\*\*  \* Your BSTIterator will be called like this:  \* BSTIterator i = BSTIterator(root);  \* while (i.hasNext()) cout << i.next();  \*/ |

Time complexity: O(h)

Space complexity: O(h)

Solution 2:

|  |
| --- |
| /\*\*  \* Definition for binary tree  \* struct TreeNode {  \* int val;  \* TreeNode \*left;  \* TreeNode \*right;  \* TreeNode(int x) : val(x), left(NULL), right(NULL) {}  \* };  \*/  class BSTIterator {  public:  BSTIterator(TreeNode \*root) {  this->root = root;  cur = root;  while(cur && cur->left) cur = cur->left;  }  /\*\* @return whether we have a next smallest number \*/  bool hasNext() {  return cur;  }  /\*\* @return the next smallest number \*/  int next() {  if (!hasNext()) return -1;  int v = cur->val;  getNextPointer();  return v;  }  private:  void getNextPointer() {  if ( cur->right ) {  cur = cur->right;  while(cur->left) cur = cur->left;  }else {  TreeNode \*p = this->root, \*res = nullptr;  int curmin = INT\_MAX;  while(p != cur) {  if (p->val < cur->val) p = p->right;  else if (p->val > cur->val) {  if (p->val < curmin) { curmin = p->val; res = p; }  p = p->left;  }  }  cur = res;  }  }  TreeNode \*root, \*cur;  };  /\*\*  \* Your BSTIterator will be called like this:  \* BSTIterator i = BSTIterator(root);  \* while (i.hasNext()) cout << i.next();  \*/ |

Time complexity: O(h)

Space complexity: O(1)

# Dungeon game, LC 174

|  |
| --- |
| class Solution {  public:  int calculateMinimumHP(vector<vector<int>>& dungeon) {  int m = dungeon.size();  if (m == 0) return 1;  int n = dungeon[0].size();  dungeon[m-1][n-1] = dungeon[m-1][n-1] >= 1 ? 1 : 1-dungeon[m-1][n-1];    for(int i = m-1; i >=0; i--)  for(int j = n-1; j >=0; j--) {  if ( i == m-1 && j == n-1) continue;  int x = INT\_MAX;  x = i < m-1 ? min(x,dungeon[i+1][j]) : x;  x = j < n-1 ? min(x,dungeon[i][j+1]) : x;  dungeon[i][j] = dungeon[i][j] >= x ? 1 : x-dungeon[i][j];  }  return dungeon[0][0];  }  }; |

Time complexity: O(mn)

Space complexity: O(1)

# Determine if array is monotonic

|  |
| --- |
| Bool isMonotonic(vector<int> arr) {  If (arr.size() < 2) return true;  Int flag = 0; //undetermined. 1: ascend; -1: descend  If (arr[0] < arr[1]) flag = 1;  Else if (arr[0] > arr[1] ) flag = -1;  for(int i = 2; i < arr.size(); i++) {  If ((arr[i-1] < arr[1] && flag == -1)||(arr[i-1]>arr[i]&&flag == 1) ) return false;  }  Return true;  } |

Time complexity: O(n)

Space complexity: O(1)

# Walls and gates, LC 286

|  |
| --- |
| class Solution {  public:  void wallsAndGates(vector<vector<int>>& rooms) {  int m = rooms.size();  if ( m == 0 ) { return; }  int n = rooms[0].size();    queue<pair<int,int>> myq;  for(int i = 0; i < m; i++)  for(int j = 0; j < n; j++) {  if (rooms[i][j] == 0) myq.push(make\_pair(i,j));  }    vector<vector<int>> offset={{-1,0},{1,0},{0,-1},{0,1}};    while(!myq.empty()) {  pair<int,int> f = myq.front();  myq.pop();  for(int i = 0; i < 4; i++) {  int newi = f.first+offset[i][0], newj = f.second+offset[i][1];  if (newi < 0 || newj < 0 || newi == m || newj == n || rooms[newi][newj] == -1) continue;  if ( rooms[newi][newj] > rooms[f.first][f.second]+1) {  rooms[newi][newj] = rooms[f.first][f.second]+1;  myq.push(make\_pair(newi,newj));  }  }  }    }  }; |

Note: only add to queue when rooms[newi][newj] is being updated. Otherwise will have infinite loop!

Time complexity: O(n^2k)

Space complexity: O(n^2)

# Decode ways, LC 91

|  |
| --- |
| class Solution {  public:  int numDecodings(string s) {  if (s.length() == 0 || s[0] == '0') return 0;  int pprev = 1, prev = 1;  for(int i = 1; i < s.length(); i++) {  int cur = 0;  cur += s[i]=='0'?0:prev;  cur += stoi(s.substr(i-1,2)) >= 10 && stoi(s.substr(i-1,2)) <= 26 ? pprev : 0;  pprev = prev;  prev = cur;  }  return prev;  }  }; |

Time complexity: O(n)

Space complexity: O(1)

# Minimum size subarray sum, LC 209

Solution 1:

|  |
| --- |
| class Solution {  public:  int minSubArrayLen(int s, vector<int>& nums) {  if (nums.size() == 0 || (nums.size() == 1 && nums[0] < s)) return 0;  vector<int> sum(nums.size()+1,0);  for(int i = 0; i < nums.size(); i++)  sum[i+1] = nums[i]+sum[i];  if (sum[sum.size()-1] < s) return 0;  int l = 0, r = 1, res = nums.size();  while( r < sum.size()) {  if ( sum[r]-sum[l] >= s ) {res = min(res,r-l); l++; }  else if (sum[r]-sum[l] < s) r++;  }  return res;  }  }; |

Time complexity: O(n)

Space complexity: O(n)

Solution 2:

No extra memory! Store current subarray left/right indices & cursum. Update both.

|  |
| --- |
| class Solution {  public:  int minSubArrayLen(int s, vector<int>& nums) {  if (nums.size() == 0 || (nums.size() == 1 && nums[0] < s)) return 0;    int l = 0, r = 0, res = 0, cursum = nums[0];  while ( r < nums.size()) {  if (cursum >= s) { cursum -= nums[l]; res = res==0||res>r-l+1? r-l+1 : res; l++; }  else cursum += nums[++r];  }    return res;  }  }; |

Time complexity: O(n)

Space complexity: O(1)

Solution 3: binary search. O(nlogn).

# Word ladder, LC 127

|  |
| --- |
| class Solution {  public:  int ladderLength(string beginWord, string endWord, vector<string>& wordList) {  unordered\_set<string> dict;  for(int i = 0; i < wordList.size(); i++) dict.insert(wordList[i]);  if ( dict.find(endWord) == dict.end() ) return 0;  queue<pair<string,int>> q1,q2;  q1.push(make\_pair(beginWord,0)); q2.push(make\_pair(endWord,0));  unordered\_map<string,int> visited1, visited2;  visited1.insert(make\_pair(beginWord,0));  visited2.insert(make\_pair(endWord,0));    int res = 0;  while(!q1.empty() && !q2.empty()) {  bfs\_one(q1,dict,visited1);  bfs\_one(q2,dict,visited2);  for(auto it = visited1.begin(); it != visited1.end(); it++) {  if (visited2.find(it->first) != visited2.end() ) { //intersect!  res = min(res,it->second + visited2[it->first])+1;  }  }  if ( res != 0 ) return res;  }  return 0;  }  private:  void bfs\_one(queue<pair<string,int>>& myq, unordered\_set<string>& dict, unordered\_map<string,int>& visited) {  string s = myq.front().first;  int level = myq.front().second;  myq.pop();    for(int i = 0; i < s.length(); i++) {  for(int j = 0; j < 26; j++) {  string neighbor = s;  if ( s[i]-'a' == j ) continue;  neighbor[i] = char('a'+j);  if ( dict.find(neighbor) != dict.end() && visited.find(neighbor) == visited.end() ) {  myq.push(make\_pair(neighbor,level+1));  visited.insert(make\_pair(neighbor,level+1));  }  }  }    }  }; |

Time complexity: O(nl)

Space complexity: O(n)

Note: Must return in while loop! Important!

# Remove duplicate characters in a string and print it in alphabetical order

|  |
| --- |
| Void removeDup(string s) {  vector<int> occur(26,0);  for(auto &c : s ) occur[c-'a']++;  for(int i = 0; i < 26; i++) {  If (occur[i] > 0 ) cout << char(i+'a') << endl;  }  } |

## Follow up: LC 316, Remove duplicate letters

# Sliding window maximum LC 239

Solution 1: use priority\_queue

|  |
| --- |
| struct Ele {  Ele(int v, int i) { val = v; idx = i; }  int val;  int idx;  };  struct MyComp {  bool operator()(Ele a, Ele b) { return a.val < b.val; }  };  class Solution {  public:  vector<int> maxSlidingWindow(vector<int>& nums, int k) {  if ( k == 0 || nums.size() == 0) return vector<int>{};    vector<int> res;  priority\_queue<Ele,vector<Ele>,MyComp> mypq;  //1. Initialize pq  for(int i = 0; i < k; i++) { Ele newe(nums[i],i); mypq.push(newe); } //mypq.insert(Ele(nums[i],i));  res.push\_back(mypq.top().val);  //2. Move window  for(int r = k; r < nums.size(); r++) {  //current window range: r-k+1...r  //remove anything before r-k+1  mypq.push(Ele(nums[r],r));  while(mypq.top().idx < r-k+1) mypq.pop();  res.push\_back(mypq.top().val);  }  return res;  }  }; |

Time complexity: O(nlogn)

Solution 2: use deque (double-ended queue)

|  |
| --- |
| class Solution {  public:  vector<int> maxSlidingWindow(vector<int>& nums, int k) {  if ( k == 0 || nums.size() == 0) return vector<int>{};    vector<int> res;  deque<int> dq;  for(int i = 0; i < nums.size(); i++) {  if (!dq.empty() && dq.front() == i-k ) dq.pop\_front();  while (!dq.empty() && nums[dq.back()] < nums[i] ) dq.pop\_back();  dq.push\_back(i);  if (i >= k-1 ) res.push\_back(nums[dq.front()]);  }    return res;  }  }; |

Time complexity: O(n)

# Add binary strings

|  |
| --- |
| String addStrings(string s1, string s2) {  Int m = s1.length(), n = s2.length();  If ( m == 0 ) return s2;  If ( n == 0 ) return s1;  Int carry = 0;  String res(max(m,n),'0');  Int i = m-1, j = n-1;  while(i>=0 || j>= 0) {  If (i>=0) { carry += s1[i]-'0'; i--; }  If (j>=0) { carry += s2[j]-'0'; j--; }  res[max(i,j)+1] = char('0'+carry%2);  Carry /= 2;  }  If (carry == 1) res = '1' + res;  Return res;  } |

Note: bug-prone at res[max(i,j)+1], because i/j already --.

## Follow-up: decimal(LC 415) or hex strings

(map between char and number.)

# Tic-tac-toe, LC 348

|  |
| --- |
| class TicTacToe {  public:  /\*\* Initialize your data structure here. \*/  TicTacToe(int n) {  siz = n;  rowvec.resize(2,vector<int>(n,0));  colvec.resize(2,vector<int>(n,0));  diag.resize(2,vector<int>(2,0));  }    /\*\* Player {player} makes a move at ({row}, {col}).  @param row The row of the board.  @param col The column of the board.  @param player The player, can be either 1 or 2.  @return The current winning condition, can be either:  0: No one wins.  1: Player 1 wins.  2: Player 2 wins. \*/  int move(int row, int col, int player) {  rowvec[player-1][row]++;  colvec[player-1][col]++;  if (row == col) diag[player-1][0]++;  if (row+col == siz-1) diag[player-1][1]++;  if (rowvec[player-1][row] == siz || colvec[player-1][col] == siz || diag[player-1][0] == siz || diag[player-1][1] == siz) return player;  return 0;  }  private:  vector<vector<int>> rowvec,colvec,diag;  int siz;  };  /\*\*  \* Your TicTacToe object will be instantiated and called as such:  \* TicTacToe obj = new TicTacToe(n);  \* int param\_1 = obj.move(row,col,player);  \*/ |

Time complexity: O(n)

Space complexity: O(n)

# Letter combinations of a phone number, LC 17

Solution 1:

Recursion

|  |
| --- |
| class Solution {  public:  vector<string> letterCombinations(string digits) {  if (digits.length() == 0) return vector<string>{};  vector<string> res;  bool flag = helper(digits,0,"",res);  return flag ? res : vector<string>{};  }  private:  bool helper(string digits,int i, string cur, vector<string>& res) {  if ( i == digits.length()) { res.push\_back(cur); return true; }  if (cur[i] == '0' || cur[i] == '1') return false;  for(auto &c : dict[digits[i]]) {  bool temp = helper(digits,i+1,cur+c,res);  if (!temp) return false;  }  return true;  }    unordered\_map<char,vector<char>> dict = {{'2',{'a','b','c'}},{'3',{'d','e','f'}},{'4',{'g','h','i'}},{'5',{'j','k','l'}},{'6',{'m','n','o'}},{'7',{'p','q','r','s'}},{'8',{'t','u','v'}},{'9',{'w','x','y','z'}}};  }; |

Solution 2:

Iterative

|  |
| --- |
| class Solution {  public:  vector<string> letterCombinations(string digits) {  if (digits.length() == 0) return vector<string>{};  vector<string> res={""};    for(auto &ch : digits) {  if (ch <'2' || ch > '9') return vector<string>{};  vector<string> tmp;  for(auto &s : res) {  for(auto &newc : dict[ch])  tmp.push\_back(s+newc);  }  res.swap(tmp);  }    return res;  }    unordered\_map<char,vector<char>> dict = {{'2',{'a','b','c'}},{'3',{'d','e','f'}},{'4',{'g','h','i'}},{'5',{'j','k','l'}},{'6',{'m','n','o'}},{'7',{'p','q','r','s'}},{'8',{'t','u','v'}},{'9',{'w','x','y','z'}}};  }; |

Note: must initialize res to {“”}

# Minimum depth of binary tree, LC 111

|  |
| --- |
| /\*\*  \* Definition for a binary tree node.  \* struct TreeNode {  \* int val;  \* TreeNode \*left;  \* TreeNode \*right;  \* TreeNode(int x) : val(x), left(NULL), right(NULL) {}  \* };  \*/  class Solution {  public:  int minDepth(TreeNode\* root) {  if (!root) return 0;  if (!root->left &&!root->right) return 1;  if (root->left && !root->right) return minDepth(root->left)+1;  else if (!root->left && root->right) return minDepth(root->right)+1;  else return min(minDepth(root->left),minDepth(root->right))+1;  }  }; |

Note: bug prone! Must not consider empty child node!

Can also do BFS, and stop when found leaf.

# Find the celebrity, LC 277

|  |
| --- |
| // Forward declaration of the knows API.  bool knows(int a, int b);  class Solution {  public:  int findCelebrity(int n) {  if ( n < 2 ) return -1;  int candidate = 0;  for(int i = 1; i < n; i++) {  if (knows(candidate,i)) candidate = i;  }  for(int i = 0; i < n; i++) {  if (candidate == i ) continue;  if (!knows(i,candidate) || knows(candidate,i)) return -1;  }  return candidate;  }  }; |

# Continuous subarray sum, LC 523

|  |
| --- |
| class Solution {  public:  bool checkSubarraySum(vector<int>& nums, int k) {  if ( nums.size() < 2 ) return false;  if ( k == 0 ) {  for(int i = 1; i < nums.size(); i++) {  if (nums[i]+nums[i-1] == 0) return true;  }  return false;  }  unordered\_map<int,int> m; // m[i]: first index for sum(0...index)%k == i  m[0] = 0;  int cur = 0;  for(int i = 1; i <= nums.size(); i++) {  cur += nums[i-1];  if ( m.find(cur%k) == m.end()) m[cur%k] = i;  else {  if (i-m[cur%k] >= 2 ) return true;  }  }  return false;  }  }; |

Note: corner cases: k is 0/negative or very large! Must use hashmap to store rather than vector.

# Minimum window substring, LC 76

|  |
| --- |
| class Solution {  public:  string minWindow(string s, string t) {  if ( t.length() == 0 || t.length() > s.length() ) return "";  vector<int> count(128,0),window(128,0);  int need = 0;  for(auto &c : t) { count[c]++; need++; }  int l = 0, r = 0;  window[s[0]]++;  if ( count[s[0]] > 0 ) { need--; }  string res = "";  while ( r < s.length()) {  if ( need == 0 ) {  res = res == "" || res.length() > r-l+1 ? s.substr(l,r-l+1) : res;  window[s[l]]--;  if ( count[s[l]] > window[s[l]] ) {  need++;  }  l++;  }else {  r++;  if ( r == s.length()) break;  window[s[r]]++;  if ( window[s[r]] <= count[s[r]]) need--;  }  }  return res;  }  }; |

Time complexity: O(n)

Space complexity: O(1)

# Inorder BST successor, LC 285

|  |
| --- |
| /\*\*  \* Definition for a binary tree node.  \* struct TreeNode {  \* int val;  \* TreeNode \*left;  \* TreeNode \*right;  \* TreeNode(int x) : val(x), left(NULL), right(NULL) {}  \* };  \*/  class Solution {  public:  TreeNode\* inorderSuccessor(TreeNode\* root, TreeNode\* p) {  if ( !root || !p) { return nullptr; }  TreeNode\* temp = nullptr;  if ( p->right ) {  temp = p->right;  while ( temp->left) { temp = temp->left; }  }else {  while ( root != p) {  temp = root->val > p->val ? root : temp;  root = root->val < p->val ? root->right:root->left;  }  }  return temp;  }      }; |

# Combination sum, LC 39

|  |
| --- |
| class Solution {  public:  vector<vector<int>> combinationSum(vector<int>& candidates, int target) {  sort(candidates.begin(),candidates.end());  if ( candidates.size() == 0 || candidates[0] > target) return vector<vector<int>>{};  vector<vector<int>> res;  vector<int> cur;  helper(candidates,0,cur,0,target,res);  return res;  }  private:  void helper(vector<int>& candidates, int i, vector<int>& cur, int cursum, int target, vector<vector<int>>& res) {  if ( cursum == target ) { res.push\_back(cur); return; }  int x = target-cursum;  for(int j = i; j < candidates.size(); j++) {  if (candidates[j] > x) return;    cur.push\_back(candidates[j]);  cursum += candidates[j];  helper(candidates,j,cur,cursum,target,res);  cur.pop\_back();  cursum -= candidates[j];  }  }  }; |

Note: must rest both cur and cursum!

Combination sum II, LC 40

|  |
| --- |
| class Solution {  public:  vector<vector<int>> combinationSum2(vector<int>& candidates, int target) {  sort(candidates.begin(),candidates.end());  if ( candidates.size() == 0 || candidates[0] > target) return vector<vector<int>>{};  vector<vector<int>> res;  vector<int> cur;  helper(candidates,0,cur,0,target,res);  return res;  }  private:  void helper(vector<int>& candidates, int i, vector<int>& cur, int cursum, int target, vector<vector<int>>& res) {  if ( cursum == target ) { res.push\_back(cur); return; }  int x = target-cursum;  for(int j = i; j < candidates.size(); j++) {  if (candidates[j] > x) return;  if ( j > i && candidates[j] == candidates[j-1]) continue;  cur.push\_back(candidates[j]);  cursum += candidates[j];  helper(candidates,j+1,cur,cursum,target,res);  cur.pop\_back();  cursum -= candidates[j];  }  }  }; |

Note: must skip all duplicate js!

# Move zeros, LC 283

|  |
| --- |
| class Solution {  public:  void moveZeroes(vector<int>& nums) {  if ( nums.size() < 2 ) return;  int p = 0, q = 0;  while( q < nums.size()) {  while ( q < nums.size() && nums[q] == 0) q++;  if ( q == nums.size()) break;  nums[p] = nums[q];  if ( p!= q) nums[q] = 0;  p++; q++;  }    }  }; |

Note: must have p!=q

## Follow up, minimal swap/assign.

Does not need to maintain order or worry about non-zeros at the end.

|  |
| --- |
| Int moveZeros(vector<int> nums) {  Int i = 0, j = nums.size()-1;  while(i<j) {  If ( nums[i] == 0 && nums[j] != 0) { swap(nums[i],nums[j]); i++;j--; }  if ( nums[i] != 0) i++;  If ( nums[j] == 0) j--;  }  Return i;  } |

# Find path in matrix with obstacles.

Given 2-d matrix with 1s and 0s, where 1 represents obstacle and 0 represents road, find a path from upper left corner to the bottom right corner.

|  |
| --- |
| vector<pair<int,int>> findPath(vector<vector<int>> matrix) {  Int m = matrix.size();  If ( m == 0 ) return vector<pair<int,int>>{};  Int n = matrix[0].size();  vector<pair<int,int>> res;  Bool flag = helper(matrix,res,0,0);  If (flag) return res;  Else return vector<pair<int,int>>{};  }  Bool helper(vector<vector<int>>& matrix, vector<pair<int,int>>& curpath, int i, int j) {  If ( i == matrix.size()-1 && j == matrix[0].size()-1 && matrix[i][j] != 1) { curpath.push\_back(make\_pair(i,j)); return true; }  If ( i == matrix.size() || j == matrix[0].size() || matrix[i][j] == 1) return false;  curpath.push\_back(make\_pair(i,j));  Bool right = helper(matrix,curpath,i,j+1);  If (right) return true;  Bool down = helper(matrix,curpath,i+1,j);  If (down) return true;  curpath.pop\_back();  Return false;  } |

If allows four direction move: need to store visited positions.

# Friend circles, LC 547

Solution 1: Union-Find

|  |
| --- |
| class Solution {  public:  int findCircleNum(vector<vector<int>>& M) {  int n = M.size();  if ( n < 2 ) return n;  int res = n;  vector<int> map(n,0);  for(int i = 0; i < n; i++) map[i] = i;  for(int i = 0; i < n-1; i++)  for(int j = i+1; j < n; j++) {  if (M[i][j] == 1) myUnion(i,j,map,res);  }      return res;  }  private:  int myFind(int x, vector<int>& map) {  while ( x != map[x] ) x = map[x];  return x;  }    void myUnion(int x, int y, vector<int>& map,int& res) {  int fx = myFind(x,map), fy = myFind(y,map);  if ( fx != fy ) {  map[fx] = fy;  res--;  }  }  }; |

Note: in union, reset ROOT nodes, not x,y.

Solution 2:

DFS

|  |
| --- |
| class Solution {  public:  int findCircleNum(vector<vector<int>>& M) {  int n = M.size();  if ( n < 2 ) return n;  int res = 0;  vector<bool> visited(n,false);  for(int i = 0; i < n; i++) {  if (!visited[i]) {  res++;  dfs(M,visited,i);  }  }  return res;  }  private:  void dfs(vector<vector<int>>& M, vector<bool>& visited, int i) {  visited[i] = true;  for(int j = 0; j < M.size(); j++) {  if ( j!=i && !visited[j] && M[i][j] == 1) dfs(M,visited,j);  }  }  }; |

# Friend ID

1) Given two “ids” and a function getFriends(id) to get the list of friends of that person id, write a function that returns the list of mutual friends

|  |
| --- |
| vector<int> mutualFriend(int id1, int id2) {  vector<int> flist1 = getFriends(id1);  vector<int> flist2 = getFriends(id2);  unordered\_set<int> set2(flist2);  vector<int> res;  for(auto & id : flist1) {  If (set2.find(id) != set2.end()) res.push\_back(id);  }  Return res;  } |

2) Given an “id” and a function getFriends(id) to get the list of friends of that person id, write a function that returns the list of “friends of friends” in the order of decreasing number of mutual friends, as in friend recommendations

|  |
| --- |
| vector<int> friendsFriend(int id) {  unordered\_map<int,int> m;  vector<int> flist = getFriends(id);  unordered\_set<int> curFriends(flist);  for(auto & fid : flist) {  vector<int> ffids = getFriends(fid);  for(auto & x : ffids) {  If ( m.find(x) == m.end() && curFriends.find(x) == curFriends.end()) {  vector<int> mutual = mutualFriend(id,x);  M[x] = mutual.size();  }  }  }  vector<pair<int,int>> respair;  for(auto it = m.begin(); it!=m.end(); it++) respair.push\_back(make\_pair(it->first,it->second));  sort(respair.begin(),respair.end(),...)  } |

# Validate BST, LC 98

Solution 1:

|  |
| --- |
| /\*\*  \* Definition for a binary tree node.  \* struct TreeNode {  \* int val;  \* TreeNode \*left;  \* TreeNode \*right;  \* TreeNode(int x) : val(x), left(NULL), right(NULL) {}  \* };  \*/  class Solution {  public:  bool isValidBST(TreeNode\* root) {  if ( !root || (!root->left && !root->right)) return true;  long initmin = -2147483649, initmax = 2147483648;  return helper(root,initmin,initmax);  }  private:  bool helper(TreeNode\* cur, long curmin, long curmax) {  if (!cur) return true;  long x = cur->val;  if ( x <= curmin || x >= curmax) return false;  return helper(cur->left,curmin, x) && helper(cur->right,x,curmax);  }  }; |

Note: must hardcode initmin/initmax, otherwise won’t work! Must use ‘long’.

Solution 2:

Instead of values, use minNode, maxNode pointer!

|  |
| --- |
| **bool** isValidBST(TreeNode\* root) {  **return** isValidBST(root, **NULL**, **NULL**); }  **bool** isValidBST(TreeNode\* root, TreeNode\* minNode, TreeNode\* maxNode) {  **if**(!root) **return** **true**;  **if**(minNode && root->val <= minNode->val || maxNode && root->val >= maxNode->val)  **return** **false**;  **return** isValidBST(root->left, minNode, root) && isValidBST(root->right, root, maxNode); } |

Solution 3:

In-order traversal.

# Group anagram, LC 49

|  |
| --- |
| class Solution {  public:  vector<vector<string>> groupAnagrams(vector<string>& strs) {  unordered\_map<string,vector<string>> mymap;  for ( string & s : strs) {  string tag = s;  sort(tag.begin(),tag.end());  if ( mymap.find(tag) == mymap.end()) { mymap.insert(make\_pair(tag,vector<string>{})); }  mymap[tag].push\_back(s);  }  vector<vector<string>> result;  for(auto it = mymap.begin(); it != mymap.end(); it++) {  result.push\_back(it->second);  }  return result;  }  }; |

# Remove duplicate letters, LC 316

Solution 1:

Backtracking. Each call tries to find a smallest solution for s[i...end] and t. t is sorted in ascending order. Beginning from t[0], find occurrence of it at s[ii] and see if it’s possible for s[ii+1...end] to match t/t[0]... First solution is the smallest.

|  |
| --- |
| class Solution {  public:  string removeDuplicateLetters(string s) {  if (s.length() < 2 ) return s;  string t;  unordered\_map<char,int> lastIndex;  for(int i = 0; i < s.length(); i++) { lastIndex[s[i]] = i; }  for(auto it = lastIndex.begin(); it != lastIndex.end(); it++) t += it->first;  sort(t.begin(),t.end());  string res;  bool flag = helper(s,t,0,lastIndex,res);  return res;  }  private:  bool hasAll(string t, int i, unordered\_map<char,int> lastIndex) {  //returns true if s[i...end] has all letters in t  for(int k = 0; k < t.length(); k++) {  if (lastIndex[t[k]] < i ) return false;  }  return true;  }    bool helper(string s, string t, int i, unordered\_map<char,int> lastIndex, string& cur) {  // cout <<" cur = " << cur << " i = " << i << " t = " << t << endl;  if (cur.length() == lastIndex.size()) return true;  for(int j = 0; j < t.length(); j++) { //find smallest possible solution beginning at t[j]  for(int ii = i; ii <= s.length()-t.length();ii++) {  if (s[ii] == t[j] && hasAll(t,ii,lastIndex)) {  cur += s[ii];  string newt = j>0?t.substr(0,j):"";  newt += t.substr(j+1,t.length()-j-1);  bool flag = helper(s,newt,ii+1,lastIndex,cur);  if (flag) return true;  else {  cur.pop\_back();  }  }  }  }  return false;  }  }; |

Works but slow!

Solution 2:

Greedy, O(n) solution:

Given the string s, the greedy choice (i.e., the leftmost letter in the answer) is the smallest s[i], s.t.

the suffix s[i .. ] contains all the unique letters. (Note that, when there are more than one smallest s[i]'s, we choose the leftmost one. Why? Simply consider the example: "abcacb".)

After determining the greedy choice s[i], we get a new string s' from s by

1. removing all letters to the left of s[i],
2. removing all s[i]'s from s.

We then recursively solve the problem w.r.t. s'.

The runtime is O(26 \* n) = O(n).

|  |
| --- |
| **public** **class** **Solution** {  **public** String **removeDuplicateLetters**(String s) {  **int**[] cnt = **new** **int**[26];  **int** pos = 0; *// the position for the smallest s[i]*  **for** (**int** i = 0; i < s.length(); i++) cnt[s.charAt(i) - 'a']++;  **for** (**int** i = 0; i < s.length(); i++) {  **if** (s.charAt(i) < s.charAt(pos)) pos = i;  **if** (--cnt[s.charAt(i) - 'a'] == 0) **break**;  }  **return** s.length() == 0 ? "" : s.charAt(pos) + removeDuplicateLetters(s.substring(pos + 1).replaceAll("" + s.charAt(pos), ""));  } } |

# Longest increasing subsequence, LC 300

|  |
| --- |
| class Solution {  public:  int lengthOfLIS(vector<int>& nums) {  if ( nums.size() < 2 ) return nums.size();  vector<int> f(nums.size(),0);  f[0] = nums[0];  int tail = 0;  for(int i = 1; i < nums.size(); i++) {  if (nums[i] > f[tail]) { f[++tail] = nums[i]; }  else {  int l = 0, r = tail;  bool dup = false;  while(l<r) {  int mid = l+(r-l)/2;  if (f[mid] < nums[i]) l = mid+1;  else if (f[mid] > nums[i]) r = mid;  else { dup = true; break;}  }  if (!dup) {f[l] = nums[i]; }  }  }  return tail+1;  }  }; |

Use an array to store current longest increasing sequence. Use binary search to substitute nums[i] to a smaller value which maintains order.

Time complexity: O(nlogn)

# Implement strStr(), LC 28

|  |
| --- |
| class Solution {  public:  int strStr(string haystack, string needle) {  int m = haystack.length(), n = needle.length();  if ( n == 0 ) { return 0; }  if ( m == 0 || m < n ) { return -1; }  for(int i = 0; i <= m-n; i++ ) {  if ( haystack[i] != needle[0]) { continue; }  if ( haystack.substr(i,n) == needle ) { return i; }  }  return -1;  }  }; |

# Task scheduler, LC 621

|  |
| --- |
| class Solution {  public:  int leastInterval(vector<char>& tasks, int n) {  if ( tasks.size() < 2 || n == 0) { return tasks.size(); }  vector<int> count(26,0);  for(auto &x : tasks) count[x-'A']++;  sort(count.begin(),count.end());  int curmax = count[count.size()-1];  int totalidle = (curmax-1)\*n;  for(int i = count.size()-2; i>=0&&count[i]>0; i--) {  totalidle -= min(curmax-1,count[i]);  }  return totalidle>0? totalidle+tasks.size():tasks.size();  }  }; |

Note: must process in descending order. Must check if totalidle > 0!

# Follow up: need to main original order.

|  |
| --- |
| Int cooldowntime(vector<char> tasks, int n) {  If (tasks.size() < 2 || n == 0) return tasks.size();  vector<int> lastSeen(26,-1);  Int timestamp = 0;  for(auto &task : tasks) {  If (lastSeen[task-'A'] == -1) lastSeen[task-'A'] = timestamp;  Else {  If (lastSeen[task-'A'] > timestamp-n) timestamp += lastSeen[task-'A']+n;  Timestamp++;  lastSeen[task-'A'] = timestamp;  }  }  Return timestamp;  } |

# Schedule meeting

Given a desired time range for a meeting and list of busy times, return all possible time for a meeting.

|  |
| --- |
| vector<pair<int,int>> scheduleMeeting(pair<int,int> desired, vector<pair<int,int>> busy) {  sort(busy.begin(),busy.end());  Int i = 1, prevend = busy[0][1];  vector<pair<int,int>> slots;  while(i < busy.size()) {  If (busy[i][1] > desired[1]) break;  If (busy[i][1] < desired[0]) i++;  Else {  If (busy[i][0] <= desired[0]) i++;  Else {  slots.push\_back(make\_pair(max(prevend,desired[0]),busy[i][0]));  Prevend = busy[i][1];  i++;  }  }  }  Return slots;  } |

# Remove duplicate strings in a string list

Keep the original order.

Create new list or use set to store strings.

# Rearrange strings k distance apart, LC 358

Solution 1: similar to task scheduler, insert elements in descending order into bins.

|  |
| --- |
| struct MyComp{  bool operator()(pair<char,int> a, pair<char,int> b) {  return a.second > b.second;  }  };  class Solution {  public:  string rearrangeString(string s, int k) {  if ( s.length() < 2 || k < 2) return s;  vector<pair<char,int>> counter(26,make\_pair('.',0));  for(auto &c : s) {  if (counter[c-'a'].first == '.') { counter[c-'a'].first = c; }  counter[c-'a'].second++;  }  sort(counter.begin(),counter.end(),MyComp());  if (counter[0].second < 2) return s;  int maxv = counter[0].second, totalidle = (maxv-1)\*(k-1);  vector<string> vec(maxv,"");  for(int j = 0; j < maxv; j++) vec[j] += counter[0].first;  for(int i = 1; i < 26 && counter[i].second > 0; i++) {  totalidle -= min(maxv-1,counter[i].second);  for(int j = 0; j < maxv && counter[i].second > 0; j++) {  if ( j < maxv && vec[j].length() == k ) continue;  vec[j] += counter[i].first;  counter[i].second--;  }  if (counter[i].second > 0) {  for(int j = 0; j < counter[i].second;j++) vec[j] += counter[i].first;  }  }  string res;  for(int j = 0; j < maxv; j++) res += vec[j];  return totalidle > 0 ? "":res;  }  }; |

Note: when appending chars, first try to insert to strings with length < k. Eg., for “aaabc” -> “abaca”; if cannot find such strings, simply append.

Solution 2:

Sequentially handle each ‘bin’ (k slots). Use ‘cache’ to store stuff poped from priority\_queue and re-insert.

|  |
| --- |
| **class** **Solution** { **public**:  string **rearrangeString**(string str, **int** k) {  **if**(k == 0) **return** str;  **int** length = (**int**)str.size();     string res;  unordered\_map<**char**, **int**> dict;  priority\_queue<pair<**int**, **char**>> pq;    **for**(**char** ch : str) dict[ch]++;  **for**(**auto** it = dict.begin(); it != dict.end(); it++){  pq.push(make\_pair(it->second, it->first));  }    **while**(!pq.empty()){  vector<pair<**int**, **char**>> cache; //store used char during one while loop  **int** count = min(k, length); //count: how many steps in a while loop  **for**(**int** i = 0; i < count; i++){  **if**(pq.empty()) **return** "";  **auto** tmp = pq.top();  pq.pop();  res.push\_back(tmp.second);  **if**(--tmp.first > 0) cache.push\_back(tmp);  length--;  }  **for**(**auto** p : cache) pq.push(p);  }  **return** res;  } }; |

# Binary tree level order traversal, LC 102

|  |
| --- |
| /\*\*  \* Definition for a binary tree node.  \* struct TreeNode {  \* int val;  \* TreeNode \*left;  \* TreeNode \*right;  \* TreeNode(int x) : val(x), left(NULL), right(NULL) {}  \* };  \*/  class Solution {  public:  vector<vector<int>> levelOrder(TreeNode\* root) {  if (!root) return vector<vector<int>>{};  queue<pair<TreeNode\*,int>> myq;  myq.push(make\_pair(root,0));  vector<vector<int>> res(1,vector<int>{});  while(!myq.empty()) {  pair<TreeNode\*,int> f = myq.front();  myq.pop();    if (res.size()-1 < f.second) res.push\_back(vector<int>{f.first->val});  else res[f.second].push\_back(f.first->val);    if(f.first->left) myq.push(make\_pair(f.first->left,f.second+1));  if(f.first->right) myq.push(make\_pair(f.first->right,f.second+1));  }  return res;  }  }; |

Note: must initialize res to (1,vector<int>{}), otherwise the call ‘res.size()’ returns weird value!

# Previous permutation

Given a string s, return the previous permutation in lexicographical order.

|  |
| --- |
| String prevPermutation(string s) {  If (s.length() < 2 ) return "";  Int i = s.length()-2;  Char prev = s[s.length()-1];  while(i>=0) {  if (prev < s[i]) { break; }  Else { prev = s[i]; i--; }  }  If ( i == -1) return s;  Int j = s.length()-1;  While ( j > i) { if (s[j] < s[i]) break; else j--; }  swap(s[i],s[j]);  sort(s.begin()+i+1,s.end(),greater<char>);  Return s;  } |

# Next permutation, LC 31

|  |
| --- |
| class Solution {  public:  void nextPermutation(vector<int>& nums) {  if (nums.size() < 2 ) return;  int i = nums.size()-2, prev = nums[nums.size()-1];  while(i>=0) {  if ( nums[i] >= prev ) { prev = nums[i]; i--;}  else break;  }  if ( i == -1) { sort(nums.begin(),nums.end()); return; }    int j = nums.size()-1;  while(j>i) {  if (nums[j] <= nums[i]) j--;  else break;  }  swap(nums[i],nums[j]);  sort(nums.begin()+i+1,nums.end());  }  }; |

Note: nums.begin()+i+1

# Intersection of two arrays, no duplicate, LC 349

|  |
| --- |
| class Solution {  public:  vector<int> intersection(vector<int>& nums1, vector<int>& nums2) {  int m = nums1.size(), n = nums2.size();  if ( m == 0 || n == 0 ) return vector<int>{};    sort(nums1.begin(),nums1.end());  sort(nums2.begin(),nums2.end());    unordered\_set<int> res;  int i = 0, j = 0;  while ( i < m && j < n) {  if (nums1[i] == nums2[j]) { res.insert(nums1[i]); i++; j++; }  else if (nums1[i] > nums2[j]) {  if ( j == n-1 ) break;  int newj = bsearch(nums2,nums1[i],j+1);  if (nums2[newj] == nums1[i]) { res.insert(nums1[i]); j = newj+1; }  else j = newj;  i++;    }else {  if ( i == m-1 ) break;  int newi = bsearch(nums1,nums2[j],i+1);  if (nums2[j] == nums1[newi]) { res.insert(nums2[j]); i = newi+1; }  else i = newi;  j++;  }  }  vector<int> rres(res.begin(),res.end());  return rres;  }  private:  int bsearch(vector<int>& nums, int target, int lo) {  //search for target starting at lo in nums. If not found, return index of smallest ele in nums that > target  int l = lo, r = nums.size()-1;  while(l < r) {  int mid = l+(r-l)/2;  if (nums[mid] == target) return mid;  else if (nums[mid] > target ) r = mid;  else l = mid+1;  }  return l;  }  }; |

Note: no duplicates allowed, so used set to store result.

# Intersection of two arrays, include duplicates, LC 350

|  |
| --- |
| class Solution {  public:  vector<int> intersect(vector<int>& nums1, vector<int>& nums2) {  int m = nums1.size(), n = nums2.size();  if ( m == 0 || n == 0 ) return vector<int>{};    sort(nums1.begin(),nums1.end());  sort(nums2.begin(),nums2.end());    vector<int> res;  int i = 0, j = 0;  while ( i < m && j < n) {  //cout << " i = " << i << " nums1[i]=" <<nums1[i] << " j = " << j << " nums2[j]=" << nums2[j] << endl;  if (nums1[i] == nums2[j]) { res.push\_back(nums1[i]); i++; j++; }  else if (nums1[i] > nums2[j]) {  if ( j == n-1 ) break;  int newj = bsearch(nums2,nums1[i],j+1);  if (nums2[newj] == nums1[i]) { res.push\_back(nums1[i]); j = newj+1; }  else j = newj;  i++;    }else {  if ( i == m-1 ) break;  int newi = bsearch(nums1,nums2[j],i+1);  if (nums2[j] == nums1[newi]) { res.push\_back(nums2[j]); i = newi+1; }  else i = newi;  j++;  }  }    return res;  }  private:  int bsearch(vector<int>& nums, int target, int lo) {  //search for target starting at lo in nums. If not found, return index of smallest ele in nums that > target  int l = lo, r = nums.size()-1;  while(l < r) {  int mid = l+(r-l)/2;  if (nums[mid] >= target) r = mid;  else l = mid+1;  }  return l;  }  }; |

**Note:** to deal with duplicate elements in result, **binary search returns the leftmost index** that is equal to target!

Other solutions include: hash map, move two pointers without binary search

Follow up: What if elements of *nums2* are stored on disk, and the memory is limited such that you cannot load all elements into the memory at once?

* If only nums2 cannot fit in memory, put all elements of nums1 into a HashMap, read chunks of array that fit into the memory, and record the intersections.
* If both nums1 and nums2 are so huge that neither fit into the memory, sort them individually (external sort), then read 2 elements from each array at a time in memory, record intersections.

# Flatten binary tree into a linked list, LC 114

|  |
| --- |
| /\*\*  \* Definition for a binary tree node.  \* struct TreeNode {  \* int val;  \* TreeNode \*left;  \* TreeNode \*right;  \* TreeNode(int x) : val(x), left(NULL), right(NULL) {}  \* };  \*/  class Solution {  public:  void flatten(TreeNode\* root) {  if (!root) return;  helper(root);  }  private:  TreeNode\* helper(TreeNode\* root) {  if (!root->left && !root->right) return root;  TreeNode \*p = root,\*q = root->right, \*res = nullptr;  if (root->left) {p = helper(root->left); res = p; root->right = root->left; root->left = nullptr; }  if (q) {  res = helper(q);  p->right = q;  }  return res;  }  }; |

# Convert a binary tree to a doubly-linked list

Solution 1: recursive call

|  |
| --- |
| void bt2dl(TreeNode\* root) {  helper(root);  }  pair<TreeNode\*,TreeNode\*> helper(TreeNode\* root) {  if (!root || (!root->left &&!root->right)) return make\_pair(root,root);  pair<TreeNode\*,TreeNode\*> l = helper(root->left);  pair<TreeNode\*,TreeNode\*> r = helper(root->right);  root->left = l.second; if (l.second) l.second.right = root;  root->right = r.first; if (r.first) r.first.left = root;  return make\_pair(l.first,r.second);  } |

Solution 2: in-order traversal

|  |
| --- |
| void bt2dl(TreeNode\* root) {  If (!root || root->left&&root->right) return;  stack<TreeNode\*> mys;  while(root) {mys.push(root); root = root->left; }  TreeNode\* prev = nullptr;  while(!mys.empty()){  TreeNode\* t = mys.top();  if(prev) { prev->right = t; t->left = prev; }  mys.pop();  prev = t;  if(t->right) { t = t->right; while(t) {mys.push(t); t = t->left; } }  }  } |

# Convert binary tree to circular doubly linked list

* Convert left subtree to a CDLL
* Merge this with root
* Convert right subtree to a CDLL
* Merge with first half

# Find Kth largest element in an array, LC 215

|  |
| --- |
| class Solution {  public:  int findKthLargest(vector<int>& nums, int k) {  if ( nums.size() == 1 ) return nums[0];    for(int i = nums.size()-1; i > 0; i--) {  int j = rand()%(i+1);  swap(nums[i],nums[j]);  }    int index = -1;  int left = 0, right = nums.size()-1;  return helper(nums,left,right,k);  }  private:  int helper(vector<int>& nums, int left, int right, int k) {  int pivot = nums[right];  int i = left-1;    for(int j = left; j < right; j++ ) {  if (nums[j] >= pivot ) {  swap(nums[++i],nums[j]);  }  }  swap(nums[++i],nums[right]);    if ( i == k-1 ) return pivot;  else if ( i > k-1 ) return helper(nums,left,i-1,k);  else return helper(nums,i+1,right,k);  }  }; |

**Note: ordering using pivot. Pivot is the last element, swap from left.**

# Replace “%20” in a string to space

|  |
| --- |
| Void replace(string& s) {  Int i = 0, j = 0;  while( j < s.length()) {  If (s[j] != '%') { s[i] = s[j]; i++; j++; }  Else {  If ( j < s.length()-2 && s[j+1] == '2' && s[j+2] == '0' ) { s[i] = ' '; i++; j+=3; }  Else { s[i] = s[j]; i++; j++; }  }  }  } |

# Equation evaluation, LC 640

|  |
| --- |
| class Solution {  public:  string solveEquation(string equation) {  if (equation.length() == 0 ) return equation;  int xcoef = 0, value = 0, i = 0, prevop = 1, sign = 1;  if (equation[0] == '-' || equation[0] == '+') equation = '0'+equation;  while ( i < equation.length()) {  string cur;  for(; i < equation.length(); i++) {  if (equation[i] == '+' || equation[i] == '-' || equation[i] == '=') break;  cur += equation[i];  }//i points to next operator +/-/=  if (cur[cur.length()-1] == 'x') xcoef += cur.length() == 1?prevop:prevop\*(stoi(cur.substr(0,cur.length()-1)));  else value += prevop\*(stoi(cur));  if (i == equation.length() ) break;    if ( equation[i] == '-') prevop = -1\*sign;  else if (equation[i] == '+') prevop = sign;  else {  sign = -1;  prevop = -1;  if (equation[i+1] == '+' || equation[i+1] == '-') { prevop = equation[i+1] == '-'? 1: -1; i++; }  }  i++;  }  // xcoef\*x+value = 0  if (xcoef == 0) {  return value == 0 ? "Infinite solutions" : "No solution";  }else {  int res = -value/xcoef;  return "x="+to\_string(res);  }  }    }; |

**Note:** corner case: left/right hand side begins with operators, ie., ‘-x=-1’

# Longest palindrome of a string, LC 409

|  |
| --- |
| class Solution {  public:  int longestPalindrome(string s) {  vector<int> count(52,0);  for(auto &c : s) {  if (isupper(c)) count[26+c-'A']++;  else count[c-'a']++;  }  int res = 0, odd = 0;  for(auto & n : count) {  res += n;  if (n%2) odd++;  }  if (odd) return res-odd+1;  else return res;  }  }; |

**Note:** be careful if odd = 0.

# Valid palindrome, LC 125

|  |
| --- |
| class Solution {  public:  bool isPalindrome(string s) {  if ( s.length() < 2 ) { return true; }  int i = 0, j = s.length()-1;  while (i<j) {  if ( isValidChar(s[i]) && isValidChar(s[j])) {  char c1 = s[i], c2 = s[j];  if ( c1 >= 'A' && c1 <= 'Z') { c1 += 'a'-'A'; }  if ( c2 >= 'A' && c2 <= 'Z') { c2 += 'a'-'A'; }  if ( c1 != c2 ) { return false; }  else {  i++; j--;  }  }  if (!isValidChar(s[i])) { i++; }  if (!isValidChar(s[j])) { j--; }  }  return true;  }  private:  bool isValidChar(char c) {  if ( (c >= '0' && c <= '9') || (c>= 'A' && c <= 'Z') || (c>='a' && c <='z') ) {  return true;  }else { return false; }  }  }; |

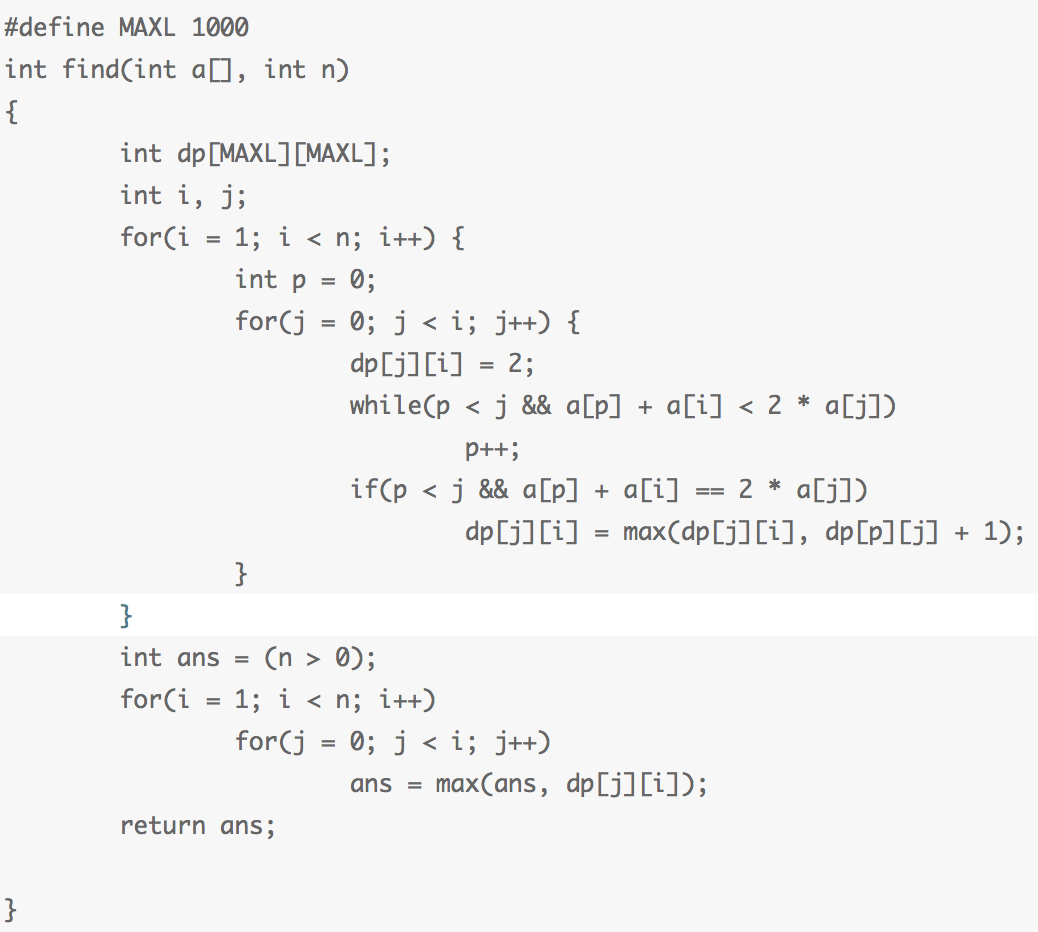
# Longest arithmetic progression sequence

Given an array, please get the length of the longest arithmetic sequence. The element order in the arithmetic sequence should be same as the element order in the array. For example, in the array {1, 6, 3, 5, 9, 7}, the longest arithmetic sequence is 1, 3, 5, and 7, whose elements have same order as they are in the array, and the length is 4.

|  |
| --- |
| Int longestArithmeticProgression(vector<int> nums) {  If (nums.size() < 2) return nums.size();  Int n = nums.size();  unordered\_map<int,vector<int>> m; //m[diff][i]: LAP ending at [i] with different = diff  Int res = 0;  for(int j = 1; j < n; j++)  for(int i = 0; i < j; i++) {  Int dif = nums[j]-nums[i];  If (m.find(dif) == m.end() ) m.insert(make\_pair(dif,vector<int>(n,0));  m[dif][j] = max(m[dif][j], m[dif][i]+1);  Res = max(res,m[dif][j]);  }  Return res;  } |

Related post: <http://codercareer.blogspot.com/2014/03/no-53-longest-arithmetic-sequence.html>

Other solution (?)



# Best time to buy and sell stock, LC 121

|  |
| --- |
| class Solution {  public:  int maxProfit(vector<int>& prices) {  if ( prices.size() < 2 ) { return 0; }  int res = 0, curmin = prices[0];  for(int i = 1; i < prices.size(); i++) {  res = max(res,prices[i]-curmin);  curmin = min(curmin,prices[i]);  }  return res;  }  }; |

# Pascal’s triangle, LC 118

|  |
| --- |
| class Solution {  public:  vector<vector<int>> generate(int numRows) {  if (numRows == 0 ) return vector<vector<int>>{};  else if (numRows == 1 ) return vector<vector<int>>{vector<int>{1}};  else if (numRows == 2) return vector<vector<int>>{vector<int>{1}, vector<int>{1,1}};  vector<vector<int>> prev = generate(numRows-1);  vector<int> lastRow = {1};  for(int i = 0; i < numRows-2; i++) {  lastRow.push\_back(prev[numRows-2][i]+prev[numRows-2][i+1]);  }  lastRow.push\_back(1);  prev.push\_back(lastRow);  return prev;  }  }; |

# Expression add operators (+ or -)

|  |
| --- |
| vector<string> addOperators(string s, int target) {  vector<string> res;  for(int i = 0; i < s.length(); i++) {  If (i > 0 && s[i] == '0') break;  Long long t = stoll(s.substr(0,i+1));  helper(s,i+1,s.substr(0,i+1),t,target,res);  }  Return res;  }  void helper(string s, int i, string cur, long long curval, int target, vector<string>& res) {  If ( i == s.length()) {  If ( curval == target ) res.push\_back(cur);  return;  }  for(int j = i; j < s.length(); j++) {  If ( j > i && s[i] == '0') return;  long long tmp = stoll(s.substr(i,j-i+1));  helper(s,j+1,cur+'+'+s.substr(i,j-i+1),curval+tmp,target,res);  helper(s,j+1,cur+'-'+s.substr(i,j-i+1),curval-tmp,target,res);  }  } |

# Expression add operators (+ or - or \*), LC 282

|  |
| --- |
| class Solution {  public:  vector<string> addOperators(string num, int target) {  vector<string> result;  if ( num.length() == 0 ) { return result; }    for(int i = 0; i < num.length(); i++ ) {  if ( num[0] == '0' && i > 0 ) break;  helper(num,i+1,target,num.substr(0,i+1),result,'+',stoll(num.substr(0,i+1)),stoll(num.substr(0,i+1)));  }    return result;  }  private:  void helper(string num, int i, int target, string curS,vector<string>& res,char op,long long prev,long long lastFactor) {  if ( i == num.length() ) { if ( prev == target) res.push\_back(curS); return; }  for(int j = i; j < num.length(); j++) {  if (num[i] == '0' && j > i ) break;  long long cur = stoll(num.substr(i,j-i+1));    helper(num,j+1,target,curS+"+"+num.substr(i,j-i+1),res,'+',prev+cur,cur);  helper(num,j+1,target,curS+"-"+num.substr(i,j-i+1),res,'-',prev-cur,cur);    if ( op == '+' ) {  helper(num,j+1,target,curS+"\*"+num.substr(i,j-i+1),res,op,prev-lastFactor+lastFactor\*cur,lastFactor\*cur);  }  else if ( op == '-'){  helper(num,j+1,target,curS+"\*"+num.substr(i,j-i+1),res,op,prev+lastFactor-lastFactor\*cur,lastFactor\*cur);  }  }  }    }; |

**Note:** Must check first char be ‘0’; Must update lastFactor

# Intersection of K sorted arrays

Assume no duplicates

|  |
| --- |
| vector<int> intersectionK(vector<vector<int>> arr) {  If (arr.size() < 2 ) return vector<int>{};  Int K = arr.size(), minID = 0;  for(int i = 1; i < K; i++) minID = arr[minID].size() > arr[i].size() ? i : minID;  unordered\_map<int,int> m;  for(int i = 0; i < arr[minID].size(); i++) m[arr[minID][i]] = 1;  vector<int> res;  for(int i = 0; i < K; i++) {  If ( i == minID) continue;  for(int j = 0; j < arr[i].size(); j++) {  if(m.find(arr[i][j]) != m.end()) m[arr[i][j]]++;  If (m[arr[i][j]] == K) res.push\_back(arr[i][j]);  }  }  Return res;  } |

Alternative solution: use 2 list intersection(binary search)

# Remove duplicates from sorted list, LC 83

S.t each element appear only once

|  |
| --- |
| /\*\*  \* Definition for singly-linked list.  \* struct ListNode {  \* int val;  \* ListNode \*next;  \* ListNode(int x) : val(x), next(NULL) {}  \* };  \*/  class Solution {  public:  ListNode\* deleteDuplicates(ListNode\* head) {  if (!head || !head->next) return head;  ListNode\* p = head, \*q = head->next;  while(q) {  while(q && q->val == p->val) q = q->next;    p->next = q;  if (q) {  p = q;  q = p->next;  }  }  return head;  }  }; |

# Remove duplicates from sorted list II, LC 82

Leave only distinct numbers

|  |
| --- |
| /\*\*  \* Definition for singly-linked list.  \* struct ListNode {  \* int val;  \* ListNode \*next;  \* ListNode(int x) : val(x), next(NULL) {}  \* };  \*/  class Solution {  public:  ListNode\* deleteDuplicates(ListNode\* head) {  if (!head || !head->next) return head;  ListNode\* dummy = new ListNode(0), \*p = head, \*q = head->next, \*l = dummy;  dummy->next = head;  while(p) {  bool isDup = false;  while(q && q->val == p->val) { q = q->next; isDup = true; }  if (!isDup) { l->next = p; l = p; }  p = q;  if (p) q = p->next;  }  l->next = nullptr;  return dummy->next;  }  }; |

**Note: must set l->next to nullptr at the end!**

Alternative solution: recursive, skipping duplicates

|  |
| --- |
| ListNode\* deleteDuplicates(ListNode\* head) {  **if** (!head) **return** 0;  **if** (!head->next) **return** head;    int **val** = head->**val**;  ListNode\* p = head->next;    **if** (p->**val** != **val**) {  head->next = deleteDuplicates(p);  **return** head;  } **else** {  **while** (p && p->**val** == **val**) p = p->next;  **return** deleteDuplicates(p);  }  } |

# Linked list cycle II, LC 142

Return the point where cycle begins in a linked list, if no cycle return null.

Solution 1: Use hashset to store linked nodes

|  |
| --- |
| class Solution {  public:  ListNode \*detectCycle(ListNode \*head) {  if (!head || !head->next) return nullptr;  unordered\_set<ListNode\*> mys;  while(head) {  if (mys.find(head) != mys.end()) return head;  else { mys.insert(head); head = head->next; }  }  return nullptr;  }  }; |

Solution 2:

Two pointers.

|  |
| --- |
| class Solution {  public:  ListNode \*detectCycle(ListNode \*head) {  if (!head || !head->next || !head->next->next) return nullptr;  ListNode \*slow = head, \*fast = head;  while(fast && fast->next) {  slow = slow->next;  fast = fast->next->next;  if (slow == fast) {  fast = head;  while(fast != slow) { slow = slow->next; fast = fast->next;}  return fast;  }  }  return nullptr;  }  }; |

# Word break, LC 139

|  |
| --- |
| class Solution {  public:  bool wordBreak(string s, vector<string>& wordDict) {  if ( s.length() == 0 ) { return false; }  unordered\_set<string> myset(wordDict.begin(), wordDict.end());  vector<int> f(s.length()+1,-1);  f[s.length()] = 1;  return helper(s,0,myset,f);  }  private:  bool helper(string s, int i, unordered\_set<string>& myset, vector<int>& f) {  if ( f[i] != -1 ) { return f[i]==1; }  for (int j = i; j < s.length(); j++ ) {  if ( myset.find(s.substr(i,j-i+1)) != myset.end() && helper(s,j+1,myset,f)) { f[i] = 1; return true;}  }  f[i] = 0;  return false;  }  }; |

# N-Queens, LC 51

|  |
| --- |
| class Solution {  public:  vector<vector<string>> solveNQueens(int n) {  vector<vector<string>> res;  if ( n == 0) return res;  else if ( n == 1 ) return vector<vector<string>>{vector<string>{"Q"}};  vector<bool> col(n,false);  vector<string> board;  helper(n,0,col,board,res);  return res;  }  private:  void helper(int n, int curRow, vector<bool>& col, vector<string>& board, vector<vector<string>>& res) {  if (curRow == n) { res.push\_back(board); return; }  for(int j = 0; j < n; j++) {  if (col[j]) continue;  //check if [curRow][j] is valid in diagonal  bool valid = true;  for(int ii = 0; ii < curRow && valid; ii++) {  for(int jj = 0; jj < n && valid; jj++) {  if (board[ii][jj] == 'Q' && (ii+jj == curRow+j || ii-curRow == jj-j)) { valid = false; }  }  }  if (valid) {  string newl(n,'.');  newl[j] = 'Q';  board.push\_back(newl);  col[j] = true;  helper(n,curRow+1,col,board,res);  board.pop\_back();  col[j] = false;  }    }  }  }; |

## Follow up, LC 52, return # of results

|  |
| --- |
| class Solution {  public:  int totalNQueens(int n) {  int res = 0;  if ( n < 2) return n;  vector<bool> col(n,false);  vector<string> board;  helper(n,0,col,board,res);  return res;  }  private:  void helper(int n, int curRow, vector<bool>& col, vector<string>& board, int &count) {  if (curRow == n) { count++; return; }  for(int j = 0; j < n; j++) {  if (col[j]) continue;  //check if [curRow][j] is valid in diagonal  bool valid = true;  for(int ii = 0; ii < curRow && valid; ii++) {  for(int jj = 0; jj < n && valid; jj++) {  if (board[ii][jj] == 'Q' && (ii+jj == curRow+j || ii-curRow == jj-j)) { valid = false; }  }  }  if (valid) {  string newl(n,'.');  newl[j] = 'Q';  board.push\_back(newl);  col[j] = true;  helper(n,curRow+1,col,board,count);  board.pop\_back();  col[j] = false;  }    }  }  }; |

Alternative solution:

Use 3 boolean arrays, column, diag1, diag2

|  |
| --- |
| **public** **class** **Solution** {  **int** **count** = 0;  **public** **int** totalNQueens(**int** n) {  **boolean**[] cols = **new** **boolean**[n]; // columns |  **boolean**[] d1 = **new** **boolean**[2 \* n]; // diagonals \  **boolean**[] d2 = **new** **boolean**[2 \* n]; // diagonals /  backtracking(0, cols, d1, d2, n);  **return** **count**;  }    **public** **void** backtracking(**int** row, **boolean**[] cols, **boolean**[] d1, **boolean** []d2, **int** n) {  **if**(row == n) **count**++;   **for**(**int** col = 0; col < n; col++) {  **int** id1 = col - row + n;  **int** id2 = col + row;  **if**(cols[col] || d1[id1] || d2[id2]) **continue**;    cols[col] = **true**; d1[id1] = **true**; d2[id2] = **true**;  backtracking(row + 1, cols, d1, d2, n);  cols[col] = **false**; d1[id1] = **false**; d2[id2] = **false**;  }  } } |

# Super ugly number, LC 313

|  |
| --- |
| class Solution {  public:  int nthSuperUglyNumber(int n, vector<int>& primes) {  if ( n < 2 ) return n;  if (primes.size() == 0) return 0;  vector<int> res(n,0), idx(primes.size(),0);  res[0] = 1;  for(int i = 1; i < n; i++) {  res[i] = INT\_MAX;  for(int j = 0; j < primes.size(); j++)  res[i] = min(res[idx[j]]\*primes[j],res[i]);  for(int j = 0; j < primes.size(); j++)  if (res[i] == res[idx[j]]\*primes[j]) idx[j]++;  }  return res[n-1];  }  }; |

**Note:** key is to use an array to store the multiplication point for each prime number!

Optimize with heap [link](https://discuss.leetcode.com/topic/34841/java-three-methods-23ms-36-ms-58ms-with-heap-performance-explained)

# Intersection of two linked lists, LC 160

|  |
| --- |
| /\*\*  \* Definition for singly-linked list.  \* struct ListNode {  \* int val;  \* ListNode \*next;  \* ListNode(int x) : val(x), next(NULL) {}  \* };  \*/  class Solution {  public:  ListNode \*getIntersectionNode(ListNode \*headA, ListNode \*headB) {  if (!headA || !headB) return nullptr;  int n1 = 0, n2 = 0;  ListNode \*p = headA, \*q = headB;  while(p) { n1++; p = p->next; }  while(q) { n2++; q = q->next; }  if ( n1 > n2 ) return helper(headA,headB,n1-n2);  else return helper(headB, headA, n2-n1);  }  private:  ListNode\* helper(ListNode \*p, ListNode \*q, int diff) {  //p is longer  while(diff > 0) { p = p->next; diff--; }  while( p != q ) { p = p->next; q = q->next; }  return p;  }  }; |

Alternative solution:

|  |
| --- |
| ListNode \*getIntersectionNode(ListNode \*headA, ListNode \*headB)  {  ListNode \*p1 = headA;  ListNode \*p2 = headB;    if (p1 == NULL **||** p2 == NULL) return NULL;   while (p1 != NULL && p2 != NULL && p1 != p2) {  p1 = p1->next;  p2 = p2->next;   //  // Any time they collide or reach end together without colliding   // then return any one of the pointers.  //  if (p1 == p2) return p1;   //  // If one of them reaches the end earlier then reuse **it**   // **by moving it** to the **beginning** of other list.  // Once **both** of them go through reassigning,   // they will **be** equidistant from the collision point.  //  if (p1 == NULL) p1 = headB;  if (p2 == NULL) p2 = headA;  }    return p1; } |

# Sparse matrix multiplication, LC 311

|  |
| --- |
| class Solution {  public:  vector<vector<int>> multiply(vector<vector<int>>& A, vector<vector<int>>& B) {  int I = A.size(), J = B.size(); if ( I == 0 || J == 0 ) return vector<vector<int>>{};  int K = B[0].size();  vector<vector<int>> res(I,vector<int>(K,0));  for(int j = 0; j < J; j++ ){  for(int i = 0; i < I; i++ ) {  if (A[i][j] == 0 ) continue;  for(int k = 0; k < K; k++ ) {  if (B[j][k] == 0 ) continue;  res[i][k] += A[i][j]\*B[j][k];  }  }  }  return res;  }  }; |

# Lowest common ancestor, LC 236

|  |
| --- |
| /\*\*  \* Definition for a binary tree node.  \* struct TreeNode {  \* int val;  \* TreeNode \*left;  \* TreeNode \*right;  \* TreeNode(int x) : val(x), left(NULL), right(NULL) {}  \* };  \*/  class Solution {  public:  TreeNode\* lowestCommonAncestor(TreeNode\* root, TreeNode\* p, TreeNode\* q) {  pair<bool,bool> res = {false,false};  return find(root,p,q,res);  }  private:  TreeNode\* find(TreeNode\* root, TreeNode\* p, TreeNode\* q, pair<bool,bool>& res) {  if (!root) return nullptr;    pair<bool,bool> lState = {false, false};  TreeNode\* rLeft = find(root->left,p,q,lState);  // cout << "Visiting " << root->val << "'s left state is " << lState.first << "," << lState.second << endl;  if (rLeft) return rLeft;    if (root == p ) res.first = true;  else if (root == q ) res.second = true;    if ((res.first && lState.second) || (res.second && lState.first)) return root;    pair<bool,bool> rState = {false,false};  TreeNode\* rRight = find(root->right,p,q,rState);  //cout << "Visiting " << root->val << "'s right state is " << rState.first << "," << rState.second << endl;  if (rRight) return rRight;    if ((res.first && rState.second) || (res.second && rState.first) || (lState.first && rState.second) || (lState.second && rState.first)) return root;  res.first |= lState.first | rState.first;  res.second |= lState.second | rState.second;    return nullptr;  }  }; |

Note: must pass-up state in res!

Alternative solution:

Disadvantage: visit every node

|  |
| --- |
| TreeNode\* lowestCommonAncestor(TreeNode\* root, TreeNode\* p, TreeNode\* q) {  **if** (!root || root == p || root == q) **return** root;  TreeNode\* left = lowestCommonAncestor(root->left, p, q);  TreeNode\* right = lowestCommonAncestor(root->right, p, q);  **return** !left ? right : !right ? left : root; } |

# Subset sum

Given a sorted list, count the number of subsets so that the sum of the largest and smallest element is smaller than a target.

|  |
| --- |
| Int subsetNumber(vector<int> nums, int target) {  If (nums.size() == 0 || nums[0] > target) return 0;  Int res = 0;  for(int i = 0; i < nums.size() && nums[i] < target; i++){  Int l = i+1, r = nums.size()-1;  while(l<r) {  int mid = l+(r-l)/2;  if(nums[mid] > target-nums[i]) r = mid-1;  Else l = mid;  }  If (nums[l]+nums[i] > target) res += 1;  Else res +=pow(2,l-i+1)-1;  }  Return res;  } |

# Three sum, LC 15

|  |
| --- |
| class Solution {  public:  vector<vector<int>> threeSum(vector<int>& nums) {  vector<vector<int>> res;  sort(nums.begin(), nums.end());  if ( nums.size() < 3|| (nums[0]+nums[1]+nums[2] > 0) || (nums[nums.size()-3]+nums[nums.size()-2]+nums[nums.size()-1] < 0) ) { return res; }    int i = 0;  while ( i < nums.size()-2 ) {  int x = 0 - nums[i];  int j = i+1, k = nums.size()-1;    while ( k > j ) {  if ( nums[j] + nums[k] > x ) { k--; }  else if ( nums[j] + nums[k] < x ) { j++; }  else {  res.push\_back(vector<int>{nums[i],nums[j],nums[k]});  while( j < k && nums[j] == nums[j+1] ) {j++; }  while( j < k && nums[k] == nums[k-1] ) { k--; }  j++; k--;  }    }    i++;  while( i < nums.size()-2 && nums[i] == nums[i-1] ) { i++; }  }  return res;  }  }; |

# Remove invalid parenthesis, LC 301

Solution 1:

Compute # of ‘(‘ and # of ‘)’ to remove. Then backtracking possible solutions.

|  |
| --- |
| class Solution {  public:  vector<string> removeInvalidParentheses(string s) {  vector<string> res;  if ( isValid(s) ) { res.push\_back(s); return res; }    int nl = 0, nr = 0, rmr = 0, rml = 0;  for(char &c : s ) {  if ( c == '(' ) nl++;  else if ( c== ')') nr++;  else continue;  if ( nr-rmr > nl ) rmr++;  }  rml = nl - (nr - rmr);    //cout << "# of right to remove is: " << rmr << " # of left to remove is: " << rml << endl;    //remove rmr right and rml left  unordered\_set<string> resSet;  helper(s,"",0,0,0,rmr,rml,resSet);    for(auto it = resSet.begin(); it != resSet.end(); it++ ) {  res.push\_back(\*it);  }  return res.empty()? vector<string>{""} : res;  }  private:  void helper(string s, string cur, int i, int nl, int nr, int rmr, int rml, unordered\_set<string>& resSet) {  if ( i == s.length() ) {  //cout << "Checking " << cur << endl;  if (isValid(cur)) resSet.insert(cur);  return;  }  if ( s[i] == '(' ) {    if ( rml > 0 && nl >= nr) {  helper(s,cur,i+1,nl,nr,rmr,rml-1,resSet);  }  helper(s,cur+'(',i+1,++nl,nr,rmr,rml,resSet);    }else if ( s[i] == ')' ) {  if ( rmr > 0 ) {  // cout << "rmr = " << rmr << ", do not add this ), cur = " << cur << endl;  helper(s,cur,i+1,nl,nr,rmr-1,rml,resSet);  }  helper(s,cur+')',i+1,nl,++nr,rmr,rml,resSet);  // cout << "rmr = " << rmr << ", add this ), cur = " << cur << endl;  }else helper(s,cur+s[i],i+1,nl,nr,rmr,rml,resSet);  }    bool isValid(string s) {  stack<char> mystack;  for ( auto c : s) {  if ( c==')' ) {  if (!mystack.empty() && mystack.top() == '(') { mystack.pop();}  else if ( mystack.empty() ) { return false; }  else {  mystack.push(c);  }  }else if ( c == '(') { mystack.push(c); }  }  return mystack.empty();  }  }; |

Solution 2:

Key Points:

1. Generate unique answer once and only once, do not rely on Set.
2. Do not need preprocess.
3. Runtime 3 ms.

Explanation:

We all know how to check a string of parentheses is valid using a stack. Or even simpler use a counter.

The counter will increase when it is ‘(‘ and decrease when it is ‘)’. Whenever the counter is negative, we have more ‘)’ than ‘(‘ in the prefix.

To make the prefix valid, we need to remove a ‘)’. The problem is: which one? The answer is any one in the prefix. However, if we remove any one, we will generate duplicate results, for example: s = ()), we can remove s[1] or s[2] but the result is the same (). Thus, we restrict ourself to remove the first ) in a series of concecutive )s.

After the removal, the prefix is then valid. We then call the function recursively to solve the rest of the string. However, we need to keep another information: the last removal position. If we do not have this position, we will generate duplicate by removing two ‘)’ in two steps only with a different order.

For this, we keep tracking the last removal position and only remove ‘)’ after that.

Now one may ask. What about ‘(‘? What if s = ‘(()(()’ in which we need remove ‘(‘?

The answer is: do the same from right to left.

However a cleverer idea is: reverse the string and reuse the code!

|  |
| --- |
| **public** List<**String**> removeInvalidParentheses(**String** s) {  List<**String**> ans = **new** ArrayList<>();  remove(s, ans, 0, 0, **new** **char**[]{'(', ')'});  return ans; }  **public** **void** remove(**String** s, List<**String**> ans, **int** last\_i, **int** last\_j, **char**[] par) {  for (**int** stack = 0, i = last\_i; i < s.length(); ++i) {  if (s.charAt(i) == par[0]) stack++;  if (s.charAt(i) == par[1]) stack--;  if (stack >= 0) continue;  for (**int** j = last\_j; j <= i; ++j)  if (s.charAt(j) == par[1] && (j == last\_j || s.charAt(j - 1) != par[1]))  remove(s.substring(0, j) + s.substring(j + 1, s.length()), ans, i, j, par);  return;  }  **String** reversed = **new** StringBuilder(s).reverse().toString();  if (par[0] == '(') // finished left to right  remove(reversed, ans, 0, 0, **new** **char**[]{')', '('});  else // finished right to left  ans.add(reversed); } |

# Longest valid parentheses, LC 32:

|  |
| --- |
| class Solution {  public:  int longestValidParentheses(string s) {  int n = s.length();  if ( n < 2 ) return 0;  vector<int> f(n+1,0); // f[i]: longest valid parentheses ending at s[i-1]  int res = 0;  for(int i = 2; i < n+1; i++ ) {  if (s[i-1] == '(') continue;  if (s[i-2] == '(') f[i] = f[i-2]+2;  else {  if (s[i-2-f[i-1]] == '(') f[i] = f[i-1]+2+f[i-2-f[i-1]];  }  res = max(res,f[i]);  }  return res;  }  }; |

Note: must add f[i-2-f[i-1]], eg., for “()(())”.

This problem can be solved by using Dynamic Programming. We make use of a dp array where i*i*th element of dp represents the length of the longest valid substring ending at i*i*th index. We initialize the complete dp array with 0's. Now, it's obvious that the valid substrings must end with ‘)’. This further leads to the conclusion that the substrings ending with ‘(’ will always contain '0' at their corresponding dp indices. Thus, we update the dp array only when ‘)’ is encountered.

To fill dp array we will check every two consecutive characters of the string and if

1. [i]=s[*i*]=‘)’ and [i−1]=s[*i*−1]=‘(’, i.e. string looks like ‘‘.......()"⇒‘‘.......()"⇒
2. [i]=[i−2]+2dp[*i*]=dp[*i*−2]+2
3. We do so because the ending "()" portion is a valid substring anyhow and leads to an increment of 2 in the length of the just previous valid substring's length.
4. [i]=s[*i*]=‘)’ and [i−1]=s[*i*−1]=‘)’, i.e. string looks like ‘‘.......))"⇒‘‘.......))"⇒
5. if [i−[i−1]−1]=s[*i*−dp[*i*−1]−1]=‘(’ then
6. [i]=[i−1]+[i−[i−1]−2]+2dp[*i*]=dp[*i*−1]+dp[*i*−dp[*i*−1]−2]+2

The reason behind this is that if the 2nd last ‘)’ was a part of a valid substring (say subs*sub*​*s*​​), for the last ‘)’ to be a part of a larger substring, there must be a corresponding starting ‘(’ which lies before the valid substring of which the 2nd last ‘)’ is a part (i.e. before subs*sub*​*s*​​). Thus, if the character before subs*sub*​*s*​​ happens to be ‘(’, we update the [i]dp[*i*] as an addition of 22 in the length of subs*sub*​*s*​​ which is [i−1]dp[*i*−1]. To this, we also add the length of the valid substring just before the term "(,sub\_s,)" , i.e. [i−[i−1]−2]dp[*i*−dp[*i*−1]−2].

|  |
| --- |
| **public** **class** **Solution** **{**  **public** **int** **longestValidParentheses(**String s**)** **{**  **int** maxans **=** 0**;**  **int** dp**[]** **=** **new** **int[**s**.**length**()];**  **for** **(int** i **=** 1**;** i **<** s**.**length**();** i**++)** **{**  **if** **(**s**.**charAt**(**i**)** **==** ')'**)** **{**  **if** **(**s**.**charAt**(**i **-** 1**)** **==** '('**)** **{**  dp**[**i**]** **=** **(**i **>=** 2 **?** dp**[**i **-** 2**]** **:** 0**)** **+** 2**;**  **}** **else** **if** **(**i **-** dp**[**i **-** 1**]** **>** 0 **&&** s**.**charAt**(**i **-** dp**[**i **-** 1**]** **-** 1**)** **==** '('**)** **{**  dp**[**i**]** **=** dp**[**i **-** 1**]** **+** **((**i **-** dp**[**i **-** 1**])** **>=** 2 **?** dp**[**i **-** dp**[**i **-** 1**]** **-** 2**]** **:** 0**)** **+** 2**;**  **}**  maxans **=** Math**.**max**(**maxans**,** dp**[**i**]);**  **}**  **}**  **return** maxans**;**  **}** **}** |

# Add and search word, LC 211

|  |
| --- |
| struct TrieNode{  TrieNode() { next.resize(26,nullptr); isWord = false; }  vector<TrieNode\*> next;  bool isWord;  };  class WordDictionary {  public:  /\*\* Initialize your data structure here. \*/  WordDictionary() {  root = new TrieNode();  }    /\*\* Adds a word into the data structure. \*/  void addWord(string word) {  TrieNode \*p = root;  int i = 0;  while(i<word.length()) {  if (p->next[word[i]-'a']) p = p->next[word[i]-'a'];  else {  TrieNode \*temp = new TrieNode();  p->next[word[i]-'a'] = temp;  p = temp;  }  i++;  }  p->isWord = true;  }    /\*\* Returns if the word is in the data structure. A word could contain the dot character '.' to represent any one letter. \*/  bool search(string word) {  return helper(word,0,root);  }  private:  TrieNode\* root;  bool helper(string word, int i, TrieNode \*p) {  if ( i == word.length() ) {  if (p->isWord) return true;  else return false;  }  if ( word[i] != '.' ) return p->next[word[i]-'a'] && helper(word,i+1,p->next[word[i]-'a']);  else {  for(int j = 0; j < 26; j++ ) {  if (p->next[j]) {  if (helper(word,i+1,p->next[j])) return true;  }  }  return false;  }  }  };  /\*\*  \* Your WordDictionary object will be instantiated and called as such:  \* WordDictionary obj = new WordDictionary();  \* obj.addWord(word);  \* bool param\_2 = obj.search(word);  \*/ |

# Unique path II, LC 63

|  |
| --- |
| class Solution {  public:  int uniquePathsWithObstacles(vector<vector<int>>& obstacleGrid) {  int m = obstacleGrid.size();  if ( m == 0 ) return 0;  int n = obstacleGrid[0].size();  if (obstacleGrid[m-1][n-1] == 1 || obstacleGrid[0][0] == 1 ) return 0;    vector<vector<int>> dp(2,vector<int>(n,0));  dp[(m-1)%2][n-1] = 1;  for(int i = m-1; i >= 0; i--)  for(int j = n-1; j >=0; j--) {  if ( i == m-1 && j == n-1 ) continue;  if (obstacleGrid[i][j] == 1 ) dp[i%2][j] = 0;  else {  int cur = 0;  if ( i < m-1) cur += dp[(i+1)%2][j];  if ( j < n-1) cur += dp[i%2][j+1];  dp[i%2][j] = cur;  }  }  return dp[0][0];  }  }; |

**Note:** be careful with dp[(m-1)%2][n-1] =1

# Reverse linked list, LC 206

|  |
| --- |
| /\*\*  \* Definition for singly-linked list.  \* struct ListNode {  \* int val;  \* ListNode \*next;  \* ListNode(int x) : val(x), next(NULL) {}  \* };  \*/  class Solution {  public:  ListNode\* reverseList(ListNode\* head) {  if ( !head || !head->next ) { return head; }  ListNode\* dummy = new ListNode(0);  dummy->next = head;  ListNode\* p = dummy->next;  while ( p->next ) {  ListNode\* temp = dummy->next;  dummy->next = p->next;  p->next = p->next->next;  dummy->next->next = temp;  }  return dummy->next;  }  }; |

# Merge two sorted lists, LC 21

|  |
| --- |
| class Solution {  public:  ListNode\* mergeTwoLists(ListNode\* l1, ListNode\* l2) {  ListNode\* dummy = new ListNode(0), \*p = dummy;  while (l1 || l2) {  if ( (l1 && l2 && l1->val < l2->val) || !l2) {  p->next = l1; p = l1; l1 = l1->next;  }else {  p->next = l2; p = l2; l2 = l2->next;  }  }  return dummy->next;  }  }; |

# Merge K sorted lists, LC 23

|  |
| --- |
| /\*\*  \* Definition for singly-linked list.  \* struct ListNode {  \* int val;  \* ListNode \*next;  \* ListNode(int x) : val(x), next(NULL) {}  \* };  \*/  class myComparitor {  public:  bool operator()(const ListNode\* h1, const ListNode\* h2) {  return h1->val > h2->val;  }  };    class Solution {  public:    ListNode\* mergeKLists(vector<ListNode\*>& lists) {  ListNode\* dummy = new ListNode(0);  ListNode\* p = dummy;  priority\_queue<ListNode\*, vector<ListNode\*>, myComparitor> myq;  for ( int i = 0; i < lists.size(); i++ ) {  if ( lists[i] ) { myq.push(lists[i]); }  }  while (!myq.empty()) {  ListNode\* temp = myq.top();  p->next = temp;  p = p->next;  temp = temp->next;  p->next = nullptr;  myq.pop();  if ( temp ) { myq.push(temp); }  }  return dummy->next;  }  }; |

Alternative: Divide & conquer

# Regular expression, LC 10

|  |
| --- |
| class Solution {  public:  bool isMatch(string s, string p) {  int m = s.length(), n = p.length();  vector<vector<bool>> f(m+1,vector<bool>(n+1,false));  f[0][0] = true;  for(int j = 2; j <= n+1; j++ ) {  f[0][j] = p[j-1]=='\*'&&f[0][j-2];  }  for(int i = 1; i < m+1; i++ ) {  for(int j = 1; j < n+1; j++ ) {  if ( p[j-1] == '.' || p[j-1] == s[i-1] ) f[i][j] = f[i-1][j-1];  else if ( p[j-1] == '\*') {  if ( j > 1 ) f[i][j] = f[i][j-2] || (f[i-1][j] && ( s[i-1] == p[j-2] || p[j-2] == '.') );  }  }  }  return f[m][n];  }  }; |

**Note:** when p[j-1] == ‘\*’

# Flatten nested list iterator, LC 341

|  |
| --- |
| /\*\*  \* // This is the interface that allows for creating nested lists.  \* // You should not implement it, or speculate about its implementation  \* class NestedInteger {  \* public:  \* // Return true if this NestedInteger holds a single integer, rather than a nested list.  \* bool isInteger() const;  \*  \* // Return the single integer that this NestedInteger holds, if it holds a single integer  \* // The result is undefined if this NestedInteger holds a nested list  \* int getInteger() const;  \*  \* // Return the nested list that this NestedInteger holds, if it holds a nested list  \* // The result is undefined if this NestedInteger holds a single integer  \* const vector<NestedInteger> &getList() const;  \* };  \*/  class NestedIterator {  public:  NestedIterator(vector<NestedInteger> &nestedList) {  for ( int i = nestedList.size()-1; i >= 0; i--) {  mystack.push(nestedList[i]);  }  }  int next() {  int res = mystack.top().getInteger();  mystack.pop();  return res;  }  bool hasNext() {  sanityCheck();  return !mystack.empty();  }  private:  stack<NestedInteger> mystack;  void **sanityCheck() {** // deal with empty lists on top  while (!mystack.empty() && !mystack.top().isInteger()) {  NestedInteger temp = mystack.top();  mystack.pop();  if ( temp.getList().size() == 0 ) { continue; }  for ( int i = temp.getList().size()-1; i >=0; i-- ) { mystack.push(temp.getList()[i]);}  }  }  };  /\*\*  \* Your NestedIterator object will be instantiated and called as such:  \* NestedIterator i(nestedList);  \* while (i.hasNext()) cout << i.next();  \*/ |

**Note:** must deal with empty nested list!

# Flatten multi-level linked list

List node has ‘next’ and ‘child’.

|  |
| --- |
| vector<int> flatten(MListNode \*head) {  vector<int> res;  While (head) {  res.push\_back(head->val);  If (head->child) {  vector<int> clist = flatten(head->child);  res.push\_back(clist.begin(),clist.end());  }  Head = head->next;  }  Return res;  } |

# Word search, LC 79

|  |
| --- |
| class Solution {  public:  bool exist(vector<vector<char>>& board, string word) {  int m = board.size();  if ( m == 0 || word.length() == 0) { return false; }  int n = board[0].size();  unordered\_set<int> visited;  for ( int i = 0; i < m; i++ ) {  for ( int j = 0; j < n; j++) {  if ( board[i][j] == word[0] ) {  visited.insert(i\*n+j);  if ( helper(board,word,i,j,1,visited)) { return true; }  visited.erase(i\*n+j);  }  }  }  return false;  }  private:  bool helper(vector<vector<char>>& board, string word, int i, int j, int p, unordered\_set<int>& visited) {  if ( p == word.length() ) { return true; }  //cout << "Finding " << word[p] << " around " << i << "-"<<j<<endl;  //find word[p] at board[i][j]'s neighbors  int m = board.size(), n = board[0].size();  vector<vector<int>> neighbors={{i-1,j},{i+1,j},{i,j-1},{i,j+1}};    for ( int ni = 0; ni < 4; ni++ ) {  int x = neighbors[ni][0], y = neighbors[ni][1];  if ( x < 0 || y < 0 || x >= m || y >= n ) { continue; }  if ( board[x][y] == word[p] && visited.find(x\*n+y) == visited.end() ) {  visited.insert(x\*n+y);  if ( helper(board,word,x,y,p+1,visited)) { return true; }  visited.erase(x\*n+y);  }  }  return false;  }  }; |

# Valid IP, LC 468

|  |
| --- |
| class Solution {  public:  string validIPAddress(string IP) {  if (IP.length() == 0) return "Neither";  bool v4 = false, v6 = false;  for(auto &c : IP) {  if ( c == '.') { v4 = checkV4(IP); break; }  else if ( c == ':') { v6 = checkV6(IP); break; }  }  return v4 ? "IPv4" : v6 ? "IPv6" : "Neither";  }  private:  bool checkV4(string IP) {  if (IP[0] == '.' || IP[IP.length()-1] == '.') return false;  stringstream ss(IP);  vector<string> vec;  for(string cur; getline(ss,cur,'.'); vec.push\_back(cur));  if (vec.size() != 4) return false;  for(int i = 0; i < 4; i++) {  string s = vec[i];  if (s.length() == 0 || s.length() > 3 || (s.length() > 1 && s[0] =='0' && s[1]=='0')) return false;  for(auto &c :s) if (!(c>='0' && c<='9')) return false;  int curval = stoi(s);  if ( curval > 255 || curval < 0 || (curval == 0 && s.length() > 1) || (curval != 0 && s[0]=='0')) return false;  }  return true;  }    bool checkV6(string IP) {  if (IP[0] == ':' || IP[IP.length()-1] == ':') return false;  stringstream ss(IP);  vector<string> vec;  for(string cur; getline(ss,cur,':'); vec.push\_back(cur));  if (vec.size() != 8) return false;  for (int i = 0; i < 8; i++) {  //Invalid: length > 4, digits not in 0-9, A-F, a-f  string s = vec[i];  if (s.length() > 4 || s.length() == 0) return false;  for(auto &c : s) {  if ( (c>='0' && c <= '9') || (c >= 'A' && c <= 'F') || (c>='a' && c <= 'f')) continue;  else return false;  }  }  return true;  }  }; |

# Serialize and deserialize a binary tree, LC 297

|  |
| --- |
| /\*\*  \* Definition for a binary tree node.  \* struct TreeNode {  \* int val;  \* TreeNode \*left;  \* TreeNode \*right;  \* TreeNode(int x) : val(x), left(NULL), right(NULL) {}  \* };  \*/  class Codec {  public:  // Encodes a tree to a single string.  string serialize(TreeNode\* root) {  string res;  if (!root) return res;  res += to\_string(root->val);  string l = serialize(root->left);  string r = serialize(root->right);  res += ',';  res += l == "" ? "x" : l;  res += ',';  res += r == "" ? "x" : r;  return res;    }  // Decodes your encoded data to tree.  TreeNode\* deserialize(string data) {  if ( data == "" ) return nullptr;  stringstream ss(data);  vector<string> vec;    for(string cur; getline(ss,cur,','); vec.push\_back(cur));  TreeNode \*root = new TreeNode(stoi(vec[0]));  TreeNode \*temp = new TreeNode(0); //placeholder  root->left = temp; root->right = temp;    stack<TreeNode\*> mys;  mys.push(root);    for(int i = 1; i < vec.size(); i++) {  if ( vec[i] != "x" ) {  TreeNode \*newnode = new TreeNode(stoi(vec[i]));  newnode->left = temp; newnode->right = temp;    if ( mys.top()->left == temp ) mys.top()->left = newnode;  else mys.top()->right = newnode;    mys.push(newnode);  }else {  if ( mys.top()->left == temp ) mys.top()->left = nullptr;  else { mys.top()->right = nullptr;  while(!mys.empty() && mys.top()->right != temp ) { mys.pop(); }  }  }  }    return root;    }    private:    };  // Your Codec object will be instantiated and called as such:  // Codec codec;  // codec.deserialize(codec.serialize(root)); |

Alternative solution:

Update s

|  |
| --- |
| **class** **Codec** { **public**:  // Encodes a tree to a single string.  string serialize(TreeNode\* root) {  **if** (root == nullptr) **return** "#";  **return** to\_string(root->**val**)+","+serialize(root->left)+","+serialize(root->right);  }   // Decodes your encoded data to tree.  TreeNode\* deserialize(string **data**) {  **return** mydeserialize(**data**);  }  TreeNode\* mydeserialize(string& **data**) {  **if** (**data**[0]=='#') {  **if**(**data**.size() > 1) **data** = **data**.substr(2);  **return** nullptr;  } **else** {  TreeNode\* node = new TreeNode(helper(**data**));  node->left = mydeserialize(**data**);  node->right = mydeserialize(**data**);  **return** node;  }  } **private**:  int helper(string& **data**) {  int pos = **data**.find(',');  int **val** = stoi(**data**.substr(0,pos));  **data** = **data**.substr(pos+1);  **return** **val**;  } }; |

# Josephus problem

There are n people standing in a circle waiting to be executed. The counting out begins at some point in the circle and proceeds around the circle in a fixed direction. In each step, a certain number of people are skipped and the next person is executed. The elimination proceeds around the circle (which is becoming smaller and smaller as the executed people are removed), until only the last person remains, who is given freedom. Given the total number of persons n and a number k which indicates that k-1 persons are skipped and kth person is killed in circle. The task is to choose the place in the initial circle so that you are the last one remaining and so survive.

|  |
| --- |
| Int josephus(int n, int k) {  //Assume index starts with 0  Int i = 0, round = 1;  vector<int> alive(n,true);  While ( round < n-1) {  Int cur = 1;  while( cur < k-1 ) {  while(!alive[i%n]) i++;  cur++;  i++;  }  alive[i%n] = false;  i++;  round++;  }  for(int i = 0; i < n; i++) if (alive[i]) return i;  } |

For 1-base position (starting from 1)

josephus(n, k) = (josephus(n-1,k)+k-1)%n + 1

josephus(1,k) = 1

for 0-base position

josephus(n,k) = (josephus(n-1,k)+k)%n

josephus(1,k)=0

# Binary search tree range sum

Given a range and a BST, find the sum of nodes whose value is in the range

|  |
| --- |
| Int BSTRange(TreeNode\* root, int lo, int hi) {  Int res = 0;  If (!root) return 0;  If (root->val > hi ) return BSTRange(root->left,lo,hi);  Else if (root->val < lo ) return BSTRange(root->right,lo,hi);  Else res = root->val + BSTRange(root->left,lo,hi) + BSTRange(root->right,lo,hi);  Return res;  } |

Solution 2: inorder traversal

|  |
| --- |
| Int BSTRange(TreeNode\* root, int lo, int hi) {  If (!root) return 0;  stack<TreeNode\*> mys;  while(root) { mys.push(root); root = root->left; }  Int res = 0;  While (!mys.empty()) {  If (mys.top()->val > hi ) break;  TreeNode \*cur = mys.top();  mys.pop();  If (cur->val >= lo ) res += cur;  If (cur->right) { cur = cur->right; while(cur) { mys.push(cur); cur = cur->left; } }  }  Return res;  } |

Alternative solution:

O(logN): add a field of (cumulative) sum to the tree node, and find the nodes at the boundary.

# Strobogrammatic number, LC 246

|  |
| --- |
| class Solution {  public:  bool isStrobogrammatic(string num) {  vector<char> m = {'0','1',' ',' ',' ',' ','9',' ','8','6'};  int i = 0, j = num.length()-1;  while(i<=j) {  if ( m[num[i]-'0'] != num[j] ) return false;  i++; j--;  }  return true;  }  }; |

# Strobogrammatic number II, LC 247

|  |
| --- |
| class Solution {  public:  vector<string> findStrobogrammatic(int n) {  vector<char> m = {'0','1',' ',' ',' ',' ','9',' ','8','6'};  vector<string> res;    helper(n,"",m,res);    return res;  }  private:  void helper(int n, string curS, vector<char>& m, vector<string>& res) {  if ( curS.length() == (n+1)/2 ) {  int last = n%2 == 1 ? curS.length()-2 : curS.length()-1;  while(last >= 0 ) { curS += m[curS[last]-'0']; last--; }  res.push\_back(curS);  return;  }    for(int i = 0; i < 10; i++) {  if ( m[i] == ' ') continue;  if ( i == 0 && curS.length() == 0 && n > 1) continue;  if ( n%2 == 1 && curS.length() == n/2 && m[i] != char(i+'0')) continue;  helper(n,curS+m[i],m,res);  }    }  }; |

**Note:** corner cases: n == 1, can have ‘0’ as first digit; n is odd, middle element cannot be 6/9.

# Number of islands, LC 200

|  |
| --- |
| class Solution {  public:  int numIslands(vector<vector<char>>& grid) {  if (grid.size() == 0) return 0;  int res = 0;  for(int i = 0; i < grid.size(); i++)  for(int j = 0; j < grid[0].size(); j++) {  if ( grid[i][j] == '1') {  res++;  dfs(i,j,grid);  }  }  return res;  }  private:  void dfs(int i, int j, vector<vector<char>>& grid) {  if ( i < 0 || j < 0 || i == grid.size() || j == grid[0].size() || grid[i][j] == '0') return;  grid[i][j] = '0';  dfs(i+1,j,grid);  dfs(i,j+1,grid);  dfs(i-1,j,grid);  dfs(i,j-1,grid);  }  }; |

# Course schedule, LC 207

|  |
| --- |
| class Solution {  public:  bool canFinish(int numCourses, vector<pair<int, int>>& prerequisites) {  if ( prerequisites.size() == 0 ) return true;  vector<vector<int>> in(numCourses,vector<int>{}), out(numCourses,vector<int>{});  for(auto &p : prerequisites) {  in[p.second].push\_back(p.first);  out[p.first].push\_back(p.second);  }    queue<int> myq;  for(int i = 0; i < numCourses; i++) {  if (in[i].empty() && !out[i].empty()) myq.push(i);  }    while(!myq.empty()) {  int f = myq.front();  myq.pop();  //remove f from in[out[f]]  for(auto &x : out[f]) {  for(int j = 0; j < in[x].size(); j++) {  if ( in[x][j] == f ) { in[x].erase(in[x].begin()+j); break; }  }  if ( in[x].empty()) myq.push(x);  }  }    for(int i = 0; i < numCourses; i++)  if (!in[i].empty() ) return false;  return true;  }  }; |

# Course schedule II, LC 210

|  |
| --- |
| class Solution {  public:  vector<int> findOrder(int numCourses, vector<pair<int, int>>& prerequisites) {  vector<int> res;  if ( numCourses == 0 ) { return res;}  if ( numCourses == 1 ) { res.push\_back(0); return res; }  unordered\_map<int,unordered\_set<int>> m;  vector<int> indegree(numCourses,0);  for( auto &x : prerequisites) {  m[x.second].insert(x.first);  indegree[x.first]++;  }    queue<int> myq;  for(int i = 0; i < numCourses; i++) { if (indegree[i] ==0) { myq.push(i);}}    while ( !myq.empty()) {  int cur = myq.front(); myq.pop();  res.push\_back(cur);  for(auto it = m[cur].begin(); it!= m[cur].end(); it++) {  indegree[\*it]--;  if ( indegree[\*it] == 0 ) { myq.push(\*it); }  }    }    return res.size() == numCourses ? res: vector<int>{};  }  }; |

# Populate next right pointer, LC 116

|  |
| --- |
| /\*\*  \* Definition for binary tree with next pointer.  \* struct TreeLinkNode {  \* int val;  \* TreeLinkNode \*left, \*right, \*next;  \* TreeLinkNode(int x) : val(x), left(NULL), right(NULL), next(NULL) {}  \* };  \*/  class Solution {  public:  void connect(TreeLinkNode \*root) {  if (!root) return;  queue<TreeLinkNode\*> myq;  myq.push(root);  while(!myq.empty()) {  TreeLinkNode \*f = myq.front();  myq.pop();  if (f->left ) {  f->left->next = f->right;  if (f->next) { f->right->next = f->next->left; }  myq.push(f->left);  myq.push(f->right);  }  }  }  }; |

# Sparse vector dot product

Use set, only common indices contribute to product.

Or, binary search.

# Interweave list of list

[[1,2,3], [4,5], [6,7,8,9]] => [1,4,6,2,5,7,3,8,9]

In-place.

|  |
| --- |
| ListNode\* interweave(vector<ListNode\*> lists){  If (lists.size() == 0 ) return nullptr;  Else if (lists.size() == 1) return lists[0];  ListNode\* head = new ListNode(0), \*p = head;  while(!done) {  bool done = true;  for(int i = 0; i < lists.size(); i++) {  If (lists[i]) {  done = false;  p->next = lists[i];  lists[i] = lists[i]->next;  p = p->next;  }  }  }  return head->next;  } |

# Reachable

Given an infinite matrix, a destination and source coordinates, and some blocks that cannot be reached. Is it reachable from source to destination? From (x, y) can only move (x+1, y+2), (x+2, y+1), (x+1, y-2), (x+2, y-1), (x-1,y+2), (x-2, y+1), (x-1, y-2), (x-2, y-1)

|  |
| --- |
| bool reachable(pair<int,int> source, pair<int,int> destination, vector<pair<int,int>> blocks) {  unordered\_set<pair<int,int>> visited1, visited2, b(blocks.begin(),block.end());  queue<pair<int,int>> q1, q2;  q1.push(source); q2.push(destination);  while( !q1.empty() || !q2.empty()){  If (!q1.empty()) bfsOne(visited1,b,q1);  If (!q2.empty()) bfsOne(visited2,b,q2);  If (intersect(visited1,visited2)) return true;  }  Return false;  }  void bfsOne(unordered\_set<pair<int,int>>& visited, unordered\_set<pair<int,int>> blocks, queue<pair<int,int>> myq) {  vector<vector<int>> offset={{1,2},{2,1},{1,-2},{2,-1},{-1,2},{-2,1},{-1,-2},{-2,-1}};  pair<int,int> pos = myq.front(); myq.pop();  for(int i = 0; i < offset.size(); i++) {  pair<int,int> neighbor = {pos.first+offset[i][0], pos.second+offset[i][1]}  If (visited.find(neighbor) == visited.end() && blocks.find(neighbor) == blocks.end())  { myq.push(neighbor); visited.insert(neighbor); }  }  } |

# Largest cross

Given a 2d matrix with 0 and 1, find the size of the largest plus (+) of 1’s, i.e., four directions with all 1’s and the same length.

Generate 4 matrices, each recording the number of consecutive 1’s from [i][j] to the four directions. Then loop over the board to find the largest of min(4 directions)

# Reorder List, LC 143

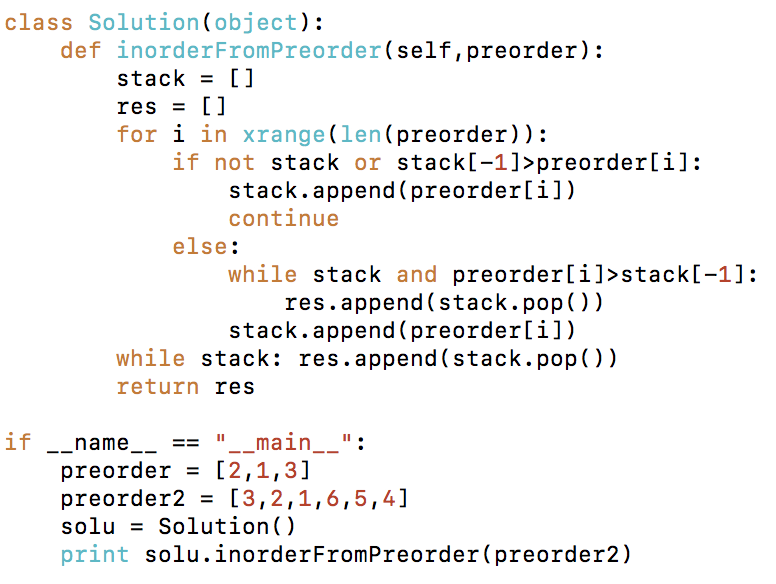
|  |
| --- |
| /\*\*  \* Definition for singly-linked list.  \* struct ListNode {  \* int val;  \* ListNode \*next;  \* ListNode(int x) : val(x), next(NULL) {}  \* };  \*/  class Solution {  public:  void reorderList(ListNode\* head) {  if (!head || !head->next ) return;    ListNode \*slow = head, \*fast = head->next->next;  while(fast && fast->next) {  slow = slow->next;  fast = fast->next->next;  }  fast = reverse(slow->next);  slow->next = nullptr;    slow = head;  ListNode \*p = head;  while(slow) {  slow = slow->next;  p->next = fast;  p = p->next;  fast = fast->next;  if (slow) {  p->next = slow;  p = p->next;  }  }  }  private:  ListNode\* reverse(ListNode\* head) {  ListNode\* dummy = new ListNode(0);  dummy->next = head;  ListNode \*p = head;    while(p->next) {  ListNode \*q = p->next;  p->next = q->next;  q->next = dummy->next;  dummy->next = q;  }  return dummy->next;  }  }; |

# Preorder to inorder traversal of BST

Given the preorder traversal, return the inorder traversal of BST

|  |
| --- |
| vector<int> pre2in(vector<int> preorder){  Return helper(preorder,0,preorder.length()-1);  }  vector<int> helper(vector<int> preorder, int i, int j) {  If ( i > j ) return vector<int>{};  If ( i == j) return vector<int>{preorder[i]};  Int root = preorder[i];  Int k = 0;  for(; k < j && preorder[k] < root; k++);  vector<int> left = helper(preorder,i+1,k-1);  vector<int> right = helper(preorder,k,j);  vector<int> res;  res.push\_back(left.begin(),left.end());  res.push\_back(root);  res.push\_back(right.begin(),right.end());  return res;  } |

Alternative: use stack



# 2D sum

Given a list of 2d vectors, determine if there exists xi+xj = yi+yj = k.

|  |
| --- |
| bool exist(vector<vector<int>> vec, int k) {  unordered\_map<int,unordered\_set<int>> m;  for(int i = 0; i < vec.size(); i++) {  If (m.find(k-vec[i][0]) != m.end() && m[k-vec[i][0]].find( k- vec[i][1]) !=m[k-vec[i][0]].end() ) return true;  If (m.find(vec[i][0]) == m.end()) m.insert(make\_pair(vec[i][0],unordered\_set<int>{}));  m[vec[i][0]].insert(vec[i][1]);  }  Return false;  } |

# Buy and sell stock, LC 121

|  |
| --- |
| class Solution {  public:  int maxProfit(vector<int>& prices) {  if ( prices.size() < 2 ) { return 0; }  int res = 0, curmin = prices[0];  for(int i = 1; i < prices.size(); i++) {  res = max(res,prices[i]-curmin);  curmin = min(curmin,prices[i]);  }  return res;  }  }; |

# One edit distance, LC 161

|  |
| --- |
| class Solution {  public:  bool isOneEditDistance(string s, string t) {  if ( s.length() > t.length() ) return isOneEditDistance(t,s);  if ( t.length() - s.length() > 1) { return false; }    bool edit = false;  for(int i = 0; i < s.length(); i++) {  if (s[i] != t[i]) {  if (edit) { return false; }  if (s.length() < t.length()) { s.insert(s.begin()+i,t[i]); }  else s[i] = t[i];  edit = true;  }  }  return (s==t && edit) || s.length() == t.length()-1;  }  }; |

**Note:** corner case: when s == t[0...n-2]. Must check s.length() == t.length()-1

# Meeting rooms II, LC 253

Solution 1: simulate meeting schedule. When new meeting starts but previous meeting not ended, need one more room; else release room. Keep ‘avail’ and ‘room’!

|  |
| --- |
| /\*\*  \* Definition for an interval.  \* struct Interval {  \* int start;  \* int end;  \* Interval() : start(0), end(0) {}  \* Interval(int s, int e) : start(s), end(e) {}  \* };  \*/  struct MyComp {  bool operator()(const Interval a, const Interval b) {  return a.start < b.start;  }  };  class Solution {  public:  int minMeetingRooms(vector<Interval>& intervals) {  if (intervals.size() < 2) { return intervals.size();}  vector<int> starts, ends;  for(int i = 0; i < intervals.size(); i++ ) {  starts.push\_back(intervals[i].start);  ends.push\_back(intervals[i].end);  }  sort(starts.begin(),starts.end());  sort(ends.begin(),ends.end());    int si = 0, ei = 0, avail = 0, room = 0;  while( si < intervals.size() ) {  if (starts[si] < ends[ei] ) {  if ( avail == 0 ) { room++; }  else avail--;  si++;  }else {  ei++;  avail++;  }  }  return room;  }  }; |

Alternative solution:

Simulate schedule by merging new meeting with the most recently ended meeting. Use min heap to keep track of end times.

|  |
| --- |
| **public** **int** minMeetingRooms(Interval[] intervals) {  if (intervals == null || intervals.length == 0)  return 0;    // Sort the intervals by start time  Arrays.sort(intervals, **new** Comparator<Interval>() {  **public** **int** compare(Interval a, Interval b) { return a.start - b.start; }  });    // Use a min heap to track the minimum end time of merged intervals  PriorityQueue<Interval> heap = **new** PriorityQueue<Interval>(intervals.length, **new** Comparator<Interval>() {  **public** **int** compare(Interval a, Interval b) { return a.end - b.end; }  });    // start with the first meeting, put it to a meeting room  heap.offer(intervals[0]);    for (**int** i = 1; i < intervals.length; i++) {  // get the meeting room that finishes earliest  Interval interval = heap.poll();    if (intervals[i].start >= interval.end) {  // if the current meeting starts right after   // there's no need for a new room, merge the interval  interval.end = intervals[i].end;  } else {  // otherwise, this meeting needs a new room  heap.offer(intervals[i]);  }    // don't forget to put the meeting room back  heap.offer(interval);  }    return heap.size(); } |

# Divide two strings

**Solution: simulate division. First locate prefix of dividend >= divisor, then keep adding the next character & compute.**

|  |
| --- |
| String stringDivide(string s1, string s2) {  If ( s1.length() < s2.length()) return "0";  Int idx = 0;  Long long dividend = 0, divisor = stoll(s2);  String res="";;  while(idx < dividend.size() && dividend < divisor) {  Dividend \*= 10;  Dividend += s1[idx]-'0';  idx++;  }  while( idx < dividend.size()) {  res += dividend/divisor+'0';  Dividend = (dividend%divisor)\*10 + s1[++idx]-'0';  }  Return res == "" ? "0" : res;  } |

# Divide two integers, LC 29

|  |
| --- |
| class Solution {  public:  int divide(int dividend, int divisor) {  if (divisor == 0 ) return dividend == 0 ? 1 : dividend < 0? INT\_MIN : INT\_MAX;  if (dividend == INT\_MIN && divisor == -1) return INT\_MAX;  if ( divisor == 1 ) return dividend;  else if ( divisor == -1) return dividend == INT\_MIN ? INT\_MAX: 0-dividend;    int sign = (dividend < 0 && divisor > 0 ) || (dividend > 0 && divisor < 0)? -1 : 1;  long long ltop = dividend, lbot = divisor;  ltop = abs(ltop); lbot = abs(lbot);    if ( ltop == lbot) return sign;    long long res = 0;  long long temp = lbot, k = 1;      while(temp <= ltop) {  k = k << 1;  temp = temp << 1;  if ( temp > ltop) { res += k/2; k = 1; ltop -= temp/2; temp = lbot; }  }  res \*= sign;    if ( res < INT\_MIN) return INT\_MIN;  else if ( res > INT\_MAX) return INT\_MAX;  else return res;  }  }; |

**Note:** corner cases: divisor == 0; overflow; **Must use long long for res, temp and k.**

Alternative solution:

|  |
| --- |
| **class** **Solution** { **public**:  **int** **divide**(**int** dividend, **int** divisor) {  **if** (!divisor || (dividend == INT\_MIN && divisor == -1))  **return** INT\_MAX;  **int** sign = ((dividend < 0) ^ (divisor < 0)) ? -1 : 1;  **long** **long** dvd = labs(dividend);  **long** **long** dvs = labs(divisor);  **int** res = 0;  **while** (dvd >= dvs) {   **long** **long** temp = dvs, multiple = 1;  **while** (dvd >= (temp << 1)) {  temp <<= 1;  multiple <<= 1;  }  dvd -= temp;  res += multiple;  }  **return** sign == 1 ? res : -res;   } }; |

# Subsets, LC 78

|  |
| --- |
| class Solution {  public:  vector<vector<int>> subsets(vector<int>& nums) {  int n = nums.size();  int total = pow(2,n);  vector<vector<int>> result;  for ( int k = 0; k < total; k++ ) {  vector<int> cur;  for ( int i = 0; i < n; i++) {  if ( k & (1 << i)) { cur.push\_back(nums[i]); }  }  result.push\_back(cur);  }  return result;  }  }; |

Note: use numbers 0...2^n-1 to index each subset. For each of the n bits, if it is 1, add i-th element into current subset.

# Binary tree vertical traversal, LC 314

|  |
| --- |
| /\*\*  \* Definition for a binary tree node.  \* struct TreeNode {  \* int val;  \* TreeNode \*left;  \* TreeNode \*right;  \* TreeNode(int x) : val(x), left(NULL), right(NULL) {}  \* };  \*/  class Solution {  public:  vector<vector<int>> verticalOrder(TreeNode\* root) {  if (!root) return vector<vector<int>>{};    unordered\_map<int,vector<int>> mymap;  int minCol = INT\_MAX, maxCol = INT\_MIN;    queue<pair<TreeNode\*,int>> myq;  myq.push(make\_pair(root,0));    while(!myq.empty()) {  pair<TreeNode\*,int> f = myq.front();  myq.pop();  minCol = min(minCol,f.second);  maxCol = max(maxCol,f.second);  if (mymap.find(f.second) == mymap.end()) mymap[f.second] = vector<int>{};  mymap[f.second].push\_back(f.first->val);  if (f.first->left) myq.push(make\_pair(f.first->left,f.second-1));  if (f.first->right) myq.push(make\_pair(f.first->right,f.second+1));  }  vector<vector<int>> result;  for(int level = minCol; level <= maxCol; level++) result.push\_back(mymap[level]);  return result;  }    }; |

# Integer to English words, LC 273

|  |
| --- |
| class Solution {  public:  string helper(int num) {  //num > 0 && < 1000  string res;  if ( num >= 100) { res += digits[num/100]+" Hundred "; num = num%100; }  if ( num >= 20 ) { res += tens[num/10\*10]+ " "; num -= num/10\*10; }  if ( num >= 10 ) { res += teens[num]; }  else if (num > 0 ) res += digits[num];  if (res[res.length()-1] == ' ') res = res.substr(0,res.length()-1);  return res;  }    string numberToWords(int num) {  if ( num == 0 ) { return "Zero";}  string res;  vector<string> vec = {"","Thousand", "Million", "Billion"};  int i = 0;  while( num > 0) {  string lastThree = helper(num%1000);  if ( i > 0 && lastThree != "") lastThree += ' '+vec[i]+' ';  res = lastThree + res;  num /= 1000;  i++;  }  if (res[res.length()-1] == ' ') res = res.substr(0,res.length()-1);  return res;    }  private:  unordered\_map<int,string> digits = {{1,"One"},{2,"Two"},{3,"Three"},{4,"Four"},{5,"Five"},{6,"Six"},{7,"Seven"},{8,"Eight"},{9,"Nine"}};  unordered\_map<int,string> teens = {{10,"Ten"},{11,"Eleven"},{12,"Twelve"},{13,"Thirteen"},{14,"Fourteen"},{15,"Fifteen"},{16,"Sixteen"},{17,"Seventeen"},{18,"Eighteen"},{19,"Nineteen"}};  unordered\_map<int,string> tens = {{20,"Twenty"},{30,"Thirty"},{40,"Forty"},{50,"Fifty"},{60,"Sixty"},{70,"Seventy"},{80,"Eighty"},{90,"Ninety"}};  }; |

Note: need to check if ‘lastThree’ is empty, ie., 1,000,000. Also, a function to handle three digits is helpful.

# Range sum query 2D, LC 304

|  |
| --- |
| class NumMatrix {  public:  NumMatrix(vector<vector<int>> matrix) {  if (matrix.size() == 0) return;  m = matrix.size();  n = matrix[0].size();  csum.resize(m+1,vector<int>(n+1,0));  for(int i = 1; i <=m; i++)  for(int j = 1; j <=n; j++) {  csum[i][j] = csum[i-1][j]+csum[i][j-1]-csum[i-1][j-1]+matrix[i-1][j-1];  }  }    int sumRegion(int row1, int col1, int row2, int col2) {  int a = csum[row2+1][col2+1], b = csum[row1][col1], c = csum[row1][col2+1], d = csum[row2+1][col1];  return a+b-c-d;  }  private:  int m, n;  vector<vector<int>> csum;  };  /\*\*  \* Your NumMatrix object will be instantiated and called as such:  \* NumMatrix obj = new NumMatrix(matrix);  \* int param\_1 = obj.sumRegion(row1,col1,row2,col2);  \*/ |

# Construct binary tree from string, LC 536

|  |
| --- |
| /\*\*  \* Definition for a binary tree node.  \* struct TreeNode {  \* int val;  \* TreeNode \*left;  \* TreeNode \*right;  \* TreeNode(int x) : val(x), left(NULL), right(NULL) {}  \* };  \*/  class Solution {  public:  TreeNode\* str2tree(string s) {  if (s.length() == 0 ) return nullptr;  s = '('+s+')';  stack<TreeNode\*> mys;  int i = 0;  while( i < s.length()) {  if (s[i] == '(') {  int j = i+1;  while(s[j] != '(' && s[j] != ')') j++;  TreeNode \*newnode = new TreeNode(stoi(s.substr(i+1,j-i-1)));  if (!mys.empty()) {  if (!mys.top()->left) mys.top()->left = newnode;  else mys.top()->right = newnode;  }  mys.push(newnode);  i = j;  }else {  if ( i == s.length()-1) break;  mys.pop();  i++;  }  }  return mys.top();  }  }; |

# Alien dictionary, LC 269

|  |
| --- |
| class Solution {  public:  string alienOrder(vector<string>& words) {  if (words.size() == 0) return "";  else if (words.size() == 1) {  vector<bool> exist(26,false);  string res;  for(auto &c : words[0]) {  if (!exist[c-'a']) { res += c; exist[c-'a'] = true; }  }  return res;  }  vector<bool> exist(26,false);  vector<int> indegrees(26,0);  vector<vector<bool>> out(26,vector<bool>(26,false));    for(auto &c : words[0]) { exist[c-'a'] = true; }    for(int i = 1; i < words.size(); i++) {  string prev = words[i-1], cur = words[i];  int j = 0, k = 0;  while(j<prev.length() && k < cur.length()) {  if (prev[j] == cur[k]) { j++; k++; }  else {  if (!out[prev[j]-'a'][cur[k]-'a']) {  indegrees[cur[k]-'a']++;  out[prev[j]-'a'][cur[k]-'a']=true;  }  break;  }  }  while(k<cur.length()) { exist[cur[k]-'a'] = true; k++; }  }    bool done = false;  string res;  while(!done) {  done = true;  for(int i = 0; i < 26; i++) {  if (exist[i] && indegrees[i] == 0) {  exist[i] = false;  res += char(i+'a');  for(int j = 0; j < 26; j++) {  if (out[i][j] && exist[j]) {  indegrees[j]--;  }  }  done = false;  break;  }  }    }    for(int i = 0; i < 26; i++ ) if (exist[i]) return "";  return res;    }    }; |

Alternative solution:

|  |
| --- |
| string **alienOrder**(vector<string>& words) {  map<**char**, set<**char**>> suc, pre;  set<**char**> chars;  string s;  **for** (string t : words) {  chars.insert(t.begin(), t.end());  **for** (**int** i=0; i<min(s.size(), t.size()); ++i) {  **char** a = s[i], b = t[i];  **if** (a != b) {  suc[a].insert(b);  pre[b].insert(a);  **break**;  }  }  s = t;  }  set<**char**> free = chars;  **for** (**auto** p : pre)  free.erase(p.first);  string order;  **while** (free.size()) {  **char** a = \*begin(free);  free.erase(a);  order += a;  **for** (**char** b : suc[a]) {  pre[b].erase(a);  **if** (pre[b].empty())  free.insert(b);  }  }  **return** order.size() == chars.size() ? order : ""; } |

# Coin change, LC 322

|  |
| --- |
| class Solution {  public:  int coinChange(vector<int>& coins, int amount) {  sort(coins.begin(),coins.end());  if (amount == 0 ) return 0;  else if (amount < coins[0]) return -1;    vector<vector<int>> f(amount+1,vector<int>(coins.size()+1,-1));  int res = helper(coins,coins.size()-1,amount,f);  return res == INT\_MAX? -1 : res;  }  private:  int helper(vector<int>& coins, int i, int amount, vector<vector<int>>& f) {  //min # of coins when largest coin is coins[i].  if ( i < 0 ) return INT\_MAX;  if (f[amount][i] != -1) return f[amount][i];    if (amount < coins[0]) { f[amount][i] = INT\_MAX; return INT\_MAX; }  int cur = INT\_MAX;  if (amount%coins[i] == 0) cur = amount/coins[i];  else {  for(int a = amount/coins[i]; a >= 0; a--) {  int tmp = helper(coins,i-1,amount-a\*coins[i],f);  if ( tmp != INT\_MAX) {  cur = min(cur,a+tmp);  }  }  }    f[amount][i] = cur;  //cout << "f[" << amount << "][" << i << "] = " << cur << endl;  return cur;  }  }; |

Better:

|  |
| --- |
| **class** **Solution** { **public**:  **int** **coinChange**(vector<**int**>& coins, **int** amount) {  **int** Max = amount + 1;  vector<**int**> dp(amount + 1, Max);  dp[0] = 0;  **for** (**int** i = 1; i <= amount; i++) {  **for** (**int** j = 0; j < coins.size(); j++) {  **if** (coins[j] <= i) {  dp[i] = min(dp[i], dp[i - coins[j]] + 1);  }  }  }  **return** dp[amount] > amount ? -1 : dp[amount];  } }; |

# Binary tree paths, LC 257

|  |
| --- |
| /\*\*  \* Definition for a binary tree node.  \* struct TreeNode {  \* int val;  \* TreeNode \*left;  \* TreeNode \*right;  \* TreeNode(int x) : val(x), left(NULL), right(NULL) {}  \* };  \*/  class Solution {  public:  vector<string> binaryTreePaths(TreeNode\* root) {  vector<string> result;  if (!root) { return result; }  if ( !root->left && !root->right) { result.push\_back(to\_string(root->val)); return result; }  vector<string> result\_left = binaryTreePaths(root->left);  vector<string> result\_right = binaryTreePaths(root->right);  for( auto & s : result\_left) {  result.push\_back(to\_string(root->val)+"->"+s);  }  for( auto & s : result\_right) {  result.push\_back(to\_string(root->val)+"->"+s);  }  return result;  }  }; |

# Search for a range, LC 34

|  |
| --- |
| class Solution {  public:  vector<int> searchRange(vector<int>& nums, int target) {  if ( nums.size() == 0 || nums[0] > target || nums[nums.size()-1] < target) return vector<int>{-1,-1};  if ( nums.size() == 1 ) {  if (nums[0] == target) return vector<int>{0,0};  else return vector<int>{-1,-1};  }    int lo = 0, hi = nums.size()-1;  vector<int> res = {-1,-1};    //1. Find left  while(lo<hi) {  int mid = lo+(hi-lo)/2;  if ( nums[mid] == target && (mid == 0 || nums[mid-1] < target)) { lo = mid; cout<<"found lo"; break;}  if (nums[mid] >= target) hi = mid;  else lo = mid+1;  }    if (nums[lo] != target) return res;  res[0] = lo;    //2. Find right  lo = res[0];  hi = nums.size()-1;  while(lo<=hi) {  int mid = lo+(hi-lo)/2;  if ( nums[mid] == target && (mid == nums.size()-1 || nums[mid+1] > target)) {res[1] = mid; cout<<"found hi"; break;}  if (nums[mid] > target) hi = mid-1;  else lo = mid+1;  }  return res;  }  }; |

Better solution:

|  |
| --- |
| vector<**int**> searchRange(**int** A[], **int** n, **int** target) {  **int** i = 0, j = n - 1;  vector<**int**> ret(2, -1);  // Search for the left one  **while** (i < j)  {  **int** mid = (i + j) /2;  **if** (A[mid] < target) i = mid + 1;  **else** j = mid;  }  **if** (A[i]!=target) **return** ret;  **else** ret[0] = i;    // Search for the right one  j = n-1; // We don't have to set i to 0 the second time.  **while** (i < j)  {  **int** mid = (i + j) /2 + 1; // Make mid biased to the right  **if** (A[mid] > target) j = mid - 1;   **else** i = mid; // So that this won't make the search range stuck.  }  ret[1] = j;  **return** ret;  } |

# Find K-closest points to the origin

Given n points in 2D, find the k-closest points to (0,0)

|  |
| --- |
| vector<pair<int,int>> findKCloses(vector<pair<int,int>> points, int k) {  priority\_queue<pair<int,int>> pq;  for(int i = 0;i < points.size(); i++) {  Int curdist = points[i].first\*points[i].first+points[i].second\*poinst[i].second;  pq.push(make\_pair(,i));  If (pq.size() > k) pq.pop();  }  vector<pair<int,int>> res;  while(!pq.empty()) { res.push\_back(points[pq.top().second]); pq.pop();}  return res;  } |

# Trapping rain water, LC 42

|  |
| --- |
| class Solution {  public:  int trap(vector<int>& height) {  if (height.size() < 3 ) return 0;  int left = 0, right = height.size()-1, res = 0, leftMax = height[0], rightMax = height[right];    while(left < right) {  if (height[left] < height[right]) {  if (height[left] > leftMax) leftMax = height[left];  else res += leftMax-height[left];  left++;  }else {  if (height[right] > rightMax) rightMax = height[right];  else res += rightMax-height[right];  right--;  }  }  return res;  }  }; |

# First bad version, LC 278

|  |
| --- |
| // Forward declaration of isBadVersion API.  bool isBadVersion(int version);  class Solution {  public:  int firstBadVersion(int n) {  int left = 1, right = n;  while (left<right){  int mid = left + (right-left)/2;  if (isBadVersion(mid)) {  right = mid;  }else {  left = mid+1;  }  }  return left;  }  }; |

# Symmetric tree, LC 101

|  |
| --- |
| /\*\*  \* Definition for a binary tree node.  \* struct TreeNode {  \* int val;  \* TreeNode \*left;  \* TreeNode \*right;  \* TreeNode(int x) : val(x), left(NULL), right(NULL) {}  \* };  \*/  class Solution {  public:  bool isSymmetric(TreeNode\* root) {  if (!root) return true;  return isMirror(root->left,root->right);  }  private:  bool isMirror(TreeNode\* r1, TreeNode\* r2) {  if (!r1 && !r2) return true;  else if (!r1 || !r2) return false;  return r1->val == r2->val && isMirror(r1->left,r2->right) && isMirror(r1->right,r2->left);  }  }; |

Alternative: level order traversal, check if each level is ‘palindrome’

# Diameter of binary tree, LC 543

|  |
| --- |
| /\*\*  \* Definition for a binary tree node.  \* struct TreeNode {  \* int val;  \* TreeNode \*left;  \* TreeNode \*right;  \* TreeNode(int x) : val(x), left(NULL), right(NULL) {}  \* };  \*/  class Solution {  public:  int diameterOfBinaryTree(TreeNode\* root) {  if (!root || (!root->left && !root->right)) { return 0; }  pair<int,int> res = helper(root);  return res.second;  }  private:  pair<int,int> helper(TreeNode\* root) {    if (!root) return pair<int,int>{0,0};  pair<int,int> l = helper(root->left);  pair<int,int> r = helper(root->right);  pair<int,int> res;  //first: depth, second: diameter  int depth = max(l.first,r.first)+1;  int diameter = max(l.first+r.first,max(l.second,r.second));  res.first = depth; res.second = diameter;    return res;  }  }; |

# Subtree of another tree, LC 572

|  |
| --- |
| /\*\*  \* Definition for a binary tree node.  \* struct TreeNode {  \* int val;  \* TreeNode \*left;  \* TreeNode \*right;  \* TreeNode(int x) : val(x), left(NULL), right(NULL) {}  \* };  \*/  class Solution {  public:  bool isSubtree(TreeNode\* s, TreeNode\* t) {  if (!t) { return true; }  if (!s) { return false; }  if ( s->val == t->val && isSameTree(s,t)) { return true; }  return isSubtree(s->left,t) || isSubtree(s->right,t);  }  private:  bool isSameTree(TreeNode\* s, TreeNode\* t) {  if (!s && !t) { return true; }  if (!s || !t) { return false; }  return s->val == t->val && isSameTree(s->left,t->left) && isSameTree(s->right,t->right);  }    }; |

Alternative solution: serialize both trees and check if t is included in s.

# Shortest unsorted continuous array, LC 581

|  |
| --- |
| class Solution {  public:  int findUnsortedSubarray(vector<int>& nums) {  if (nums.size() < 2 ) return 0;    stack<int> mys;  int left = nums.size(), right = -1;    //1. Find left  for(int i = 0; i < nums.size(); i++) {  while(!mys.empty() && nums[i] < nums[mys.top()]) {  left = min(left,mys.top());  mys.pop();  }  mys.push(i);  }  if ( left == nums.size() ) return 0;    while(!mys.empty()) mys.pop();  //2. Find right    for(int i = nums.size()-1; i >=0; i--) {  while(!mys.empty() && nums[i] > nums[mys.top()]) {  right = max(right,mys.top());  mys.pop();  }  mys.push(i);  }    return right-left+1;  }  }; |

Without stack: remember the value for the displaced ‘min’ and ‘max’, and get their correct positions from another loop.

|  |
| --- |
| **public** **class** **Solution** **{**  **public** **int** **findUnsortedSubarray(int[]** nums**)** **{**  **int** min **=** Integer**.**MAX\_VALUE**,** max **=** Integer**.**MIN\_VALUE**;**  **boolean** flag **=** **false;**  **for** **(int** i **=** 1**;** i **<** nums**.**length**;** i**++)** **{**  **if** **(**nums**[**i**]** **<** nums**[**i **-** 1**])**  flag **=** **true;**  **if** **(**flag**)**  min **=** Math**.**min**(**min**,** nums**[**i**]);**  **}**  flag **=** **false;**  **for** **(int** i **=** nums**.**length **-** 2**;** i **>=** 0**;** i**--)** **{**  **if** **(**nums**[**i**]** **>** nums**[**i **+** 1**])**  flag **=** **true;**  **if** **(**flag**)**  max **=** Math**.**max**(**max**,** nums**[**i**]);**  **}**  **int** l**,** r**;**  **for** **(**l **=** 0**;** l **<** nums**.**length**;** l**++)** **{**  **if** **(**min **<** nums**[**l**])**  **break;**  **}**  **for** **(**r **=** nums**.**length **-** 1**;** r **>=** 0**;** r**--)** **{**  **if** **(**max **>** nums**[**r**])**  **break;**  **}**  **return** r **-** l **<** 0 **?** 0 **:** r **-** l **+** 1**;**  **}** **}** |

# Remove duplicates from sorted array, LC 26

|  |
| --- |
| class Solution {  public:  int removeDuplicates(vector<int>& nums) {  if ( nums.size() < 2 ) return nums.size();  int i = 0, j = 1;    while( j < nums.size()) {  while ( j < nums.size() && nums[j] == nums[i] ) j++;  if ( j < nums.size() ) { nums[++i] = nums[j]; j++; }  }  return i+1;  }  }; |

# Excel sheet column title, LC 168

|  |
| --- |
| class Solution {  public:  string convertToTitle(int n) {  string res;  while ( n > 0 ) {  char cur = (n-1)%26+'A';  n -= (n-1)%26;  n /= 26;  res = cur + res;  }  return res;  }  }; |

# Construct the rectangle, LC 492

|  |
| --- |
| class Solution {  public:  vector<int> constructRectangle(int area) {  int mid = sqrt(area);  for(int w = mid; w >= 1; w--) {  if ( area%w == 0 ) return vector<int>{area/w,w};  }  return vector<int>{};  }  }; |

# Permutations, LC 46

|  |
| --- |
| class Solution {  public:  vector<vector<int>> permute(vector<int>& nums) {  if (nums.size() == 0 ) return vector<vector<int>>{};  vector<vector<int>> res = {vector<int>{nums[0]}};  for(int i = 1; i < nums.size(); i++) {  vector<vector<int>> cur;  for(int j = 0; j < res.size(); j++) {  for(int k = 0; k <= res[j].size(); k++) {  vector<int> tmp = res[j];  tmp.emplace(tmp.begin()+k,nums[i]);  cur.push\_back(tmp);  }  }  res = cur;  }  return res;  }  }; |

Alternative:

|  |
| --- |
| **class** **Solution** { **public**:  vector<vector<**int**> > permute(vector<**int**> &num) {  vector<vector<**int**> > result;    permuteRecursive(num, 0, result);  **return** result;  }    // permute num[begin..end]  // invariant: num[0..begin-1] have been fixed/permuted  **void** **permuteRecursive**(vector<**int**> &num, **int** begin, vector<vector<**int**> > &result) {  **if** (begin >= num.size()) {  // one permutation instance  result.push\_back(num);  **return**;  }    **for** (**int** i = begin; i < num.size(); i++) {  swap(num[begin], num[i]);  permuteRecursive(num, begin + 1, result);  // reset  swap(num[begin], num[i]);  }  } }; |

# Permutation II, LC 47

|  |
| --- |
| class Solution {  public:  vector<vector<int>> permuteUnique(vector<int>& nums) {  if (nums.size() == 0 ) return vector<vector<int>>{};  vector<vector<int>> res = {vector<int>{nums[0]}};  for(int i = 1; i < nums.size(); i++) {  vector<vector<int>> cur;  for(int j = 0; j < res.size(); j++) {  vector<int> tmp = res[j];  tmp.emplace(tmp.begin(),nums[i]);  cur.push\_back(tmp);  for(int k = 1; k <= res[j].size(); k++) {  if (res[j][k-1] == nums[i]) break;  vector<int> tmp = res[j];  tmp.emplace(tmp.begin()+k,nums[i]);  cur.push\_back(tmp);  }  }  res = cur;  }  return res;  }  }; |

Note: make sure not to insert same element after occurrence of it.

Alternative solution:

|  |
| --- |
| class Solution {  public:  void recursion(vector<int> num, int i, vector<vector<int> > &res) {  if (i == num.size()-1) {  res.push\_back(num);  return;  }  for (int k = i; k < num.size(); k++) {  if (i != k && num[i] == num[k]) continue;  swap(num[i], num[k]);  recursion(num, i+1, res);  }  }  vector<vector<int> > permuteUnique(vector<int> &num) {  sort(num.begin(), num.end());  vector<vector<int> >res;  recursion(num, 0, res);  return res;  }  }; |

Note: must pass by value!

# Product of array except self, LC 238

|  |
| --- |
| class Solution {  public:  vector<int> productExceptSelf(vector<int>& nums) {  vector<int> res(nums.size(),1);  for ( int i = 1; i < nums.size(); i++ ) {  res[i] \*= res[i-1]\*nums[i-1];  }  int accu = nums[nums.size()-1];  for ( int i = nums.size()-2; i >= 0; i-- ) {  res[i] \*= accu;  accu \*= nums[i];  }  return res;  }  }; |

# Count and say, LC 38

|  |
| --- |
| class Solution {  public:  string countAndSay(int n) {  if ( n == 1 ) { return "1"; }  string prev = countAndSay(n-1);  string res;  int i = 0;  while ( i < prev.length() ) {  int p = i++;  while (i < prev.length() && prev[i] == prev[p]) { i++; }  res += to\_string(i-p)+prev[p];  }  return res;  }  }; |

# Summary ranges, LC 228

|  |
| --- |
| class Solution {  public:  vector<string> summaryRanges(vector<int>& nums) {  if (nums.size() == 0 ) return vector<string>{};    //Find next continuous range  int start = nums[0], i = 1;  vector<string> res;  while( i <= nums.size()) {  if ( i == nums.size() || nums[i] != nums[i-1]+1) {  //end at nums[i-1]  if (nums[i-1] == start) res.push\_back(to\_string(start));  else res.push\_back(to\_string(start)+"->"+to\_string(nums[i-1]));  if ( i < nums.size()) start = nums[i];  }  i++;  }  return res;  }  }; |

# Add digits, LC 258

|  |
| --- |
| class Solution {  public:  int addDigits(int num) {  return (num-1)%9+1;  }  }; |

# Number complement, LC 476

|  |
| --- |
| class Solution {  public:  int findComplement(int num) {  int res = 0, i = 0;  while( num > 0) {  int lastdigit = (num&1)^1;  res += lastdigit<<i;  num = num >> 1;  i++;  }  return res;  }  }; |

Alternative:

|  |
| --- |
| **class** **Solution** { **public**:  **int** **findComplement**(**int** num) {  **unsigned** mask = ~0;  **while** (num & mask) mask <<= 1;  **return** ~mask & ~num;  } }; |

For example,

num = 00000101  
mask = 11111000  
~mask & ~num = 00000010

# Longest consecutive sequence, LC 128

|  |
| --- |
| class Solution {  public:  int longestConsecutive(vector<int>& nums) {  if ( nums.size() < 2 ) { return nums.size(); }  unordered\_map<int,int> mymap;  int res = 1;  for(auto &n : nums) {  if ( mymap.find(n) != mymap.end() ) { continue; }  int left = (mymap.find(n-1) != mymap.end()) ? mymap[n-1] : 0;  int right =(mymap.find(n+1) != mymap.end()) ? mymap[n+1] : 0;  int cursum = left+right+1;  mymap[n] = cursum;  mymap[n-left] = cursum;  mymap[n+right] = cursum;  res = max(res,cursum);  }  return res;  }  }; |

**Note:** must add left,right,1, and update all three.

# Insert, Delete and getRandom O(1), LC 380

|  |
| --- |
| class RandomizedSet {  public:  /\*\* Initialize your data structure here. \*/  RandomizedSet() {  last = -1;  }    /\*\* Inserts a value to the set. Returns true if the set did not already contain the specified element. \*/  bool insert(int val) {  if ( data.find(val) != data.end() ) return false;  data[val] = last+1;  vec.push\_back(val);  last++;  return true;  }    /\*\* Removes a value from the set. Returns true if the set contained the specified element. \*/  bool remove(int val) {  if ( data.find(val) == data.end() ) return false;  int i = data[val];  data.erase(val);  if ( i != last ) {  vec[i] = vec[last];  data[vec[last]] = i;  }  vec.pop\_back();  last--;  return true;  }    /\*\* Get a random element from the set. \*/  int getRandom() {  if ( last == -1 ) return 0;  if ( last == 0 ) return vec[0];  return vec[rand()%(last+1)];  }  private:  unordered\_map<int,int> data;  vector<int> vec;  int last;  };  /\*\*  \* Your RandomizedSet object will be instantiated and called as such:  \* RandomizedSet obj = new RandomizedSet();  \* bool param\_1 = obj.insert(val);  \* bool param\_2 = obj.remove(val);  \* int param\_3 = obj.getRandom();  \*/ |

# Wildcard matching, LC 44

|  |
| --- |
| class Solution {  public:    bool isMatch(string s, string p) {  if ( s == p ) { return true; }  int m = s.length(), n = p.length();  vector<vector<bool>> f(m+1,vector<bool>(n+1,false));  f[0][0] = true;  for(int j = 1; j < n+1;j++ ) f[0][j] = f[0][j-1]&&p[j-1]=='\*';    vector<bool> prev(m+1,true);    for(int j = 1; j < n+1; j++) {  vector<bool> cur(m+1,false);  for(int i = 1; i < m+1; i++) {  if (p[j-1] == s[i-1] || p[j-1] == '?' ) f[i][j] = f[i-1][j-1];  else if (p[j-1] == '\*' ) f[i][j] = prev[i];  cur[i] = cur[i-1] || f[i][j];  }  prev = cur;  }  return f[m][n];  }  }; |

Alternative solution:

|  |
| --- |
| bool isMatch(**const** char \*s, **const** char \*p) {  **const** char\* star=NULL;  **const** char\* ss=s;  **while** (\*s){  //advancing both pointers **when** (both characters match) **or** ('?' found **in** pattern)  //note that \*p will **not** advance beyond its length   **if** ((\*p=='?')||(\*p==\*s)){s++;p++;**continue**;}    // \* found **in** pattern, track index **of** \*, only advancing pattern pointer   **if** (\*p=='\*'){star=p++; ss=s;**continue**;}    //current characters didn't match, last pattern pointer was \*, current pattern pointer **is** **not** \*  //only advancing pattern pointer  **if** (star){ p = star+1; s=++ss;**continue**;}    //current pattern pointer **is** **not** star, last patter pointer was **not** \*  //characters **do** **not** match  **return** false;  }   //check **for** remaining characters **in** pattern  **while** (\*p=='\*'){p++;}   **return** !\*p;   } |

# Construct binary tree from pre/in-order traversal, LC 105

|  |
| --- |
| /\*\*  \* Definition for a binary tree node.  \* struct TreeNode {  \* int val;  \* TreeNode \*left;  \* TreeNode \*right;  \* TreeNode(int x) : val(x), left(NULL), right(NULL) {}  \* };  \*/  class Solution {  public:  TreeNode\* buildTree(vector<int>& preorder, vector<int>& inorder) {  if ( preorder.size() == 0 ) return nullptr;  return helper(preorder,0,preorder.size()-1,inorder,0,inorder.size()-1);  }  private:  TreeNode\* helper(vector<int>& preorder, int pl, int pr, vector<int>& inorder, int il, int ir) {  if ( pl > pr ) return nullptr;  TreeNode\* root = new TreeNode(preorder[pl]);  int leftSiz = 0;  for(int i = il; i <= ir && inorder[i] != root->val; i++ ) leftSiz++;  TreeNode\* left = helper(preorder,pl+1,pl+leftSiz,inorder,il,il+leftSiz-1);  TreeNode\* right = helper(preorder,pl+leftSiz+1,pr,inorder,il+leftSiz+1,ir);  root->left = left; root->right = right;  return root;  }  }; |

Iterative solution: use stack to record left nodes, flag to indicate insert to right node…

[Link](https://discuss.leetcode.com/topic/795/the-iterative-solution-is-easier-than-you-think)

# Word search II, LC 212

Note: hashmap solution TLE. Must use Trie.

Optimize Trie: store word in node if node is a word. When found an answer, set it null.

|  |
| --- |
| **public** List<**String**> findWords(**char**[][] board, **String**[] words) {  List<**String**> res = **new** ArrayList<>();  TrieNode root = buildTrie(words);  for (**int** i = 0; i < board.length; i++) {  for (**int** j = 0; j < board[0].length; j++) {  dfs (board, i, j, root, res);  }  }  return res; }  **public** **void** dfs(**char**[][] board, **int** i, **int** j, TrieNode p, List<**String**> res) {  **char** c = board[i][j];  if (c == '#' || p.next[c - 'a'] == null) return;  p = p.next[c - 'a'];  if (p.**word** != null) { // found one  res.add(p.**word**);  p.**word** = null; // de-duplicate  }   board[i][j] = '#';  if (i > 0) dfs(board, i - 1, j ,p, res);   if (j > 0) dfs(board, i, j - 1, p, res);  if (i < board.length - 1) dfs(board, i + 1, j, p, res);   if (j < board[0].length - 1) dfs(board, i, j + 1, p, res);   board[i][j] = c; }  **public** TrieNode buildTrie(**String**[] words) {  TrieNode root = **new** TrieNode();  for (**String** w : words) {  TrieNode p = root;  for (**char** c : w.toCharArray()) {  **int** i = c - 'a';  if (p.next[i] == null) p.next[i] = **new** TrieNode();  p = p.next[i];  }  p.**word** = w;  }  return root; }  class TrieNode {  TrieNode[] next = **new** TrieNode[26];  **String** **word**; } |

# Ransom node, LC 383

|  |
| --- |
| class Solution {  public:  bool canConstruct(string ransomNote, string magazine) {  vector<int> count1(26,0), count2(26,0);  for(char &c : ransomNote ) count1[c-'a']++;  for(char &c : magazine ) count2[c-'a']++;  for(int i = 0; i < 26; i++ ) {  if ( count2[i] < count1[i] ) return false;  }  return true;  }  }; |

# Two Sum IV - Input is a BST, LC 653

|  |
| --- |
| class Solution {  public:  bool findTarget(TreeNode\* root, int k) {  if (!root) { return false; }  vector<int> v;  inorder(root,v);  int i = 0, j = v.size()-1;  while(i < j ) {  if (v[i] + v[j] > k ) { j--; }  else if ( v[i] + v[j] < k ) { i++; }  else {return true; }  }  return false;  }    private:  void inorder(TreeNode\* root, vector<int>& v) {  if (root) {  inorder(root->left,v);  v.push\_back(root->val);  inorder(root->right,v);  }  }  }; |

A much faster solution using inorder:

|  |
| --- |
| /\*\*  \* Definition for a binary tree node.  \* struct TreeNode {  \* int val;  \* TreeNode \*left;  \* TreeNode \*right;  \* TreeNode(int x) : val(x), left(NULL), right(NULL) {}  \* };  \*/  class Solution {  public:  bool findTarget(TreeNode\* root, int k) {  if (!root) { return false; }    stack<TreeNode\*> mys;  TreeNode\* cur = root;  while(cur || !mys.empty()) {  if (cur) {  mys.push(cur);  cur = cur->left;  }else {  TreeNode\* t= mys.top();  mys.pop();  TreeNode\* tmp = findNode(root,k-t->val);  if (tmp && tmp != t) return true;  cur = t->right;  }  }      return false;  }    private:  TreeNode\* findNode(TreeNode\* root, int v){  if (!root) return nullptr;  if (root->val == v ) return root;  else if (root->val < v ) return findNode(root->right,v);  else return findNode(root->left,v);  }  }; |

Alternative: BFS and hashset.

# Exclusive time of functions, LC 636

|  |
| --- |
| class Solution {  public:    vector<int> exclusiveTime(int n, vector<string>& logs) {  if ( n == 0 || logs.size() == 0) { return vector<int>(n,0); }  stack<pair<int,int>> mys;  int prev = -1;  vector<int> res(n,0);    for(auto & s : logs) {  stringstream ss(s);  string tmp;  getline(ss,tmp,':');  int curId = stoi(tmp);  getline(ss,tmp,':');  bool start = tmp == "start";  getline(ss,tmp,':');  int timestamp = stoi(tmp);    if (!mys.empty()) {  res[mys.top().first] += timestamp-prev;  if (start && mys.top().second != prev) res[mys.top().first]--;  else if (!start) {  if (mys.top().second == prev) res[mys.top().first]++;  mys.pop();  }  }  if (start) mys.push(make\_pair(curId,timestamp));  prev = timestamp;      }  return res;    }  }; |

Cleaner code:

|  |
| --- |
| vector<**int**> exclusiveTime(**int** n, vector<string>& logs) {  vector<**int**> res(n, 0);  stack<**int**> st;  **int** preTime = 0;  **for** (string log : logs) {  **int** found1 = log.find(":");  **int** found2 = log.find\_last\_of(":");  **int** idx = stoi(log.substr(0, found1));  string type = log.substr(found1 + 1, found2 - found1 - 1);  **int** time = stoi(log.substr(found2 + 1));  **if** (!st.empty()) {  res[st.top()] += time - preTime;  }  preTime = time;  **if** (type == "start") st.push(idx);  **else** {  **auto** t = st.top(); st.pop();  ++res[t];  ++preTime;  }  }  **return** res;  } |

Note: can use string.find/ string.find\_last\_of()!

# Multiply string, LC 43

|  |
| --- |
| class Solution {  public:  string multiply(string num1, string num2) {  if (num1.length() == 0 || num2.length() == 0 ) { return ""; }  if ( (num1.length()==1 && num1[0] == '0') || (num2.length()==1 && num2[0] == '0')) { return "0"; }  int m = num1.length(), n = num2.length(), l = m+n+1;  string res(m+n+1,'0');  for(int i = 0; i < m; i++)  for(int j = 0; j < n; j++) {  int tmp = (num1[m-i-1]-'0')\*(num2[n-j-1]-'0'), cur = l-1-(i+j);  tmp += res[cur]-'0';  res[cur] = char('0'+tmp%10);  int carryover = tmp/10;  while(carryover > 0) {  tmp = carryover+res[--cur]-'0';  res[cur] = char('0'+tmp%10);  carryover = tmp/10;  }  }  int i = 0;  while(res[i] == '0') i++;  return res.substr(i,res.length()-i);  }  }; |

Cleaner code:

|  |
| --- |
| **string** multiply(**string** num1, **string** num2) {  **string** sum(num1.size() + num2.size(), '0');    for (**int** i = num1.size() - 1; 0 <= i; --i) {  **int** carry = 0;  for (**int** j = num2.size() - 1; 0 <= j; --j) {  **int** tmp = (sum[i + j + 1] - '0') + (num1[i] - '0') \* (num2[j] - '0') + carry;  sum[i + j + 1] = tmp % 10 + '0';  carry = tmp / 10;  }  sum[i] += carry;  }    size\_t startpos = sum.find\_first\_not\_of("0");  if (**string**::npos != startpos) {  return sum.substr(startpos);  }  return "0"; } |

# Unique user given email

输入是Map<UserID, List<EmailString>>，任意两个UserID，只要有overlap的email就认为是同一个User。

比如UserA -> {a@xx.com, b@xy.com}，UserB -> {c@yz.com, b@xy.com}，那么UserA和UserB是同一个User。

输出Map<List<UserID>, List<EmailString>>，每个Key是每个User对应的所有UserID，每个Value是这个User所使用的所有Email。

# Find island with largest perimeter

# Onsite ML SWE interview Qs

onsite第一轮coding之三哥哥：三哥哥都没让我自我介绍直接开始做题。一道heap秒掉的常规题，但是follow up是让楼主实现heap。。。就是这样迷幻。。。楼主就磕磕巴巴的写了heapify，还没来的及写push和pop，就没时间了。当时以为自己gg了，现在看来三哥抬了我一手，只能尴尬而不是优雅的感恩了=。=

onsite第二轮ML design之国人哥哥：题目很常规，给用户推荐pages，问了协同过滤，怎么train，evaluate以及validate模型。总觉得国人哥哥一脸看着智杖的表情看着我，最后在qna等人领我吃午饭的时候，我问他在fb一天被内部messenger pin很多回吗，他才乐了下，感谢他最后也抬了我一手。

Lunchbreak： 吸取了indeed教训，这把没矜持，吃的挺开心。

onsite第三轮 phd research + bq + short coding之国人姐姐：小姐姐很直接，进来就说我觉得你research跟ML没半毛钱关系，直接bq，于是被问了40min如何handle conflict，work with difficultppl啥的。最后一道short coding是高频面筋题。qna的时候跟小姐姐一起感慨虾图房价长得太快了。

onsite第四轮 coding之美国小妹妹：小妹妹进屋就说咱开始做题。两道lc非fb tag题+一题fbtag题目。非fb tag题是gg tag下的，一个数字旋转180度还是这个数字的I和II。

第一轮：法国大叔，聊我的thesis research，还蛮顺的，我试图宏观解释我做的东西，并在其中点出几篇我比较得意的文章，但他似乎对细节比较在意，还谈了实习经历，但总体这类讨论还是比较轻松的。后知后觉他开始问一些bq，我之前并不知道要问bq，没怎么准备过，就临场发挥，谈了一下自己做科研的心路历程的，希望引起共鸣（面试官也是从学术界转工业界的）。最后做一道简单的coding题目（应该是lc原题，有印象），都是线性时间复杂度，我一开始给出的方案要O(n) memory，在他的提示下写了一个只一次循环并且O(1)memory的solution。

第二轮：印度大叔，上来就让我写一个trietree+wildcard，花了挺长时间才写完，应该bugfree，分析一下复杂度。最后时间不多，问了一个lc偶五三的变体，没时间写了我解释了用pq的算法，没解释清楚，他最后要我解释一下bruteforce怎么做，我也没答上来，然后时间就到了。

第三轮：欧洲小哥，问了两题（都是lc原题），第一题有corner case没考虑，在提示下修正，第二题又问了lc偶五三（有可能补了第二轮的短板），这回有时间写，写了用pq的最优复杂度算法，并解释了一下为什么是这个复杂度。最后有个bonus problem，是关于第二题怎么并行的做，也基本答上来了。

第四轮：中国小哥，算是同龄人，ML design。基本就是针对一个具体问题，扯扯可能会用到的方法和模型，谈谈针对其业务需求可能会碰到哪些难点，该怎么解决。中国小哥非常友好，我一边说他一遍帮我在墙上逐条记提纲，整个过程没什么压力，自由发挥，多和面试官互动就行。面完后和他用中文聊了会天。

# Onsite SE

第一轮：1，一个array，有m段递增序列，输出排序好的。2， 打印root到leaf的所有path

第二轮：BQ 跟上级(manager,professor）的矛盾，跟队友的conflict，做过什么错误的决，本来要问类似301的面经题，后来看了下说时间可能不太够，换了个压缩数字 :(

第三轮：电话号码那题，还有282简化版，只用＋－