**TALLER ESTRUCTURAS DE DATOS AVANZADAS**

**DANIEL STEVEN CAÑIZARES ORTEGA 1152203**

**CAMILO BOTELLO 1152137**

**MÓNICA GIL GONZÁLEZ 1152183**

**MAURICIO DI DONATO SÁNCHEZ 1152186**

**ANDRÉS ALFONSO PARRA 1152185**

**MILTON JESÚS VERA CONTRERAS**

**UNIVERSIDAD FRANCISCO DE PAULA SANTANDER**

**INGENIERÍA DE SISTEMAS**

**ANÁLISIS DE ALGORITMOS**

**NORTE DE SANTANDER**

**CÚCUTA**

**MAYO 07**

**2023**

**1) Resolver los problemas en la plataforma.**

| **EJERCICIO** | **CÓDIGO ACCEPTED** |
| --- | --- |
| #1 Complete Binary Tree Inserter  <https://leetcode.com/problems/complete-binary-tree-inserter> | class CBTInserter(object):  def \_\_init\_\_(self, root):  self.deque = collections.deque()  self.root = root  q = collections.deque([root])  while q:  node = q.popleft()  if not node.left or not node.right:  self.deque.append(node)  if node.left:  q.append(node.left)  if node.right:  q.append(node.right)  def insert(self, v):  node = self.deque[0]  self.deque.append(TreeNode(v))  if not node.left:  node.left = self.deque[-1]  else:  node.right = self.deque[-1]  self.deque.popleft()  return node.val  def get\_root(self):  return self.root |
| #2 Binary Tree Inorder Traversal  <https://leetcode.com/problems/binary-tree-inorder-traversal/> | class Solution {  public:  vector<int> inorderTraversal(TreeNode\* root) {  vector<int> p;  if(root == nullptr) return p;  vector<int> left = inorderTraversal(root->left);  p.insert(p.end(), left.begin(), left.end());  p.push\_back(root->val);  vector<int> right = inorderTraversal(root->right);  p.insert(p.end(), right.begin(), right.end());  return p;  }  }; |
| #3 Validate Binary Search Tree  <https://leetcode.com/problems/validate-binary-search-tree> | class Solution {  public:  map<long long, long long> mp;  bool isValidBST(TreeNode\* node) {  if (node == NULL) return true;  if (mp.count(node->val) > 0) return false;  else mp[node->val] = 1;  if (node->left != NULL && maxValue(node->left) > node->val) return false;  if (node->right != NULL && minValue(node->right) < node->val) return false;  if (!isValidBST(node->left) || !isValidBST(node->right)) return false;  return true;  }  int maxValue(TreeNode\* node) {  if (node == NULL) {  return INT\_MIN;  }  int value = node->val;  int leftMax = maxValue(node->left);  int rightMax = maxValue(node->right);    return max(value, max(leftMax, rightMax));  }  int minValue(TreeNode\* node) {  if (node == NULL) {  return INT\_MAX;  }  int value = node->val;  int leftMax = minValue(node->left);  int rightMax = minValue(node->right);    return min(value, min(leftMax, rightMax));  }  }; |
| #4 Minimum Height Trees  <https://leetcode.com/problems/minimum-height-trees/> | class Solution {  public:  vector<int> findMinHeightTrees(int n, vector<vector<int>>& edges) {  vector<vector<int>> graph(n);  vector<int> indegree(n, 0), ans;    for(auto &e : edges){  graph[e[0]].push\_back(e[1]);  graph[e[1]].push\_back(e[0]);  indegree[e[0]]++;  indegree[e[1]]++;  }    queue<int> q;  for(int i=0; i<n;i++){  if(indegree[i]==1) q.push(i), indegree[i]--;  }    while(!q.empty()){  int s = q.size();  ans.clear();  for(int i=0; i<s;i++){  int curr = q.front(); q.pop();  ans.push\_back(curr);  for(auto child : graph[curr]){  indegree[child]--;  if(indegree[child]==1) q.push(child);  }  }  }  if(n==1) ans.push\_back(0);  return ans;    }  }; |
| #5 Find a Corresponding Node of a Binary Tree in a Clone of That Tree <https://leetcode.com/problems/find-a-corresponding-node-of-a-binary-tree-in-a-clone-of-that-tree/> | public class Solution {  public final TreeNode getTargetCopy(final TreeNode original, final TreeNode cloned, final TreeNode target) {  if (original == null) {  return null;  }  if (original == target) {  return cloned;  }  TreeNode left = getTargetCopy(original.left, cloned.left, target);  if (left != null) {  return left;  }  return getTargetCopy(original.right, cloned.right, target);  }  } |
| #6 Find if Path Exists in Graph  <https://leetcode.com/problems/find-if-path-exists-in-graph/> | from collections import defaultdict  class Solution:  def validPath(self, n, edges, start, end):  graph = defaultdict(list)  # Construir el grafo utilizando la lista de aristas  for u, v in edges:  graph[u].append(v)  graph[v].append(u)  visited = set()  def dfs(node):  if node == end:  return True  visited.add(node)  for neighbor in graph[node]:  if neighbor not in visited:  if dfs(neighbor):  return True  return False  return dfs(start) |
| #7 Relative Ranks  <https://leetcode.com/problems/relative-ranks/> | class Solution {  public:  vector<string> findRelativeRanks(vector<int>& score) {  vector<int> salida = score;  vector<string> s(score.size());  sort(salida.rbegin(), salida.rend());  for(int i = 0; i < salida.size(); i++){  for(int j = 0; j < score.size(); j++){  if(salida[i] == score[j]){  if(i == 0){  s[j] = "Gold Medal";  }else if(i == 1){  s[j] = "Silver Medal";  }else if(i == 2){  s[j] = "Bronze Medal";  }else{  s[j] = to\_string(i+1);  }  break;  }  }  }  return s;  }  }; |
| #8 C Heap Operations  <https://codeforces.com/contest/681/problem/C> | n = int(input())  operations = []  heap = set()  output = []  for \_ in range(n):  record = input().split()  operations.append(record)  for record in operations:  if record[0] == "insert":  x = int(record[1])  output.append(record)  heap.add(x)  elif record[0] == "getMin":  x = int(record[1])  output.append(record)  if len(heap) == 0 or min(heap) != x:  output.append(["removeMin"])  output.append(["insert", str(x)])  heap.add(x)  elif record[0] == "removeMin":  output.append(record)  if len(heap) == 0:  output.append(["insert", "1"])  heap.remove(min(heap))  print(len(output))  for record in output:  print(" ".join(record)) |
| #9 Count Distinct Integers  <https://atcoder.jp/contests/abc240/tasks/abc240_b?lang=en> | #include <bits/stdc++.h>  using namespace std;  int main(){  ios\_base::sync\_with\_stdio(false);  cin.tie(nullptr);  int n, x;  cin >> n;  vector<int> p;  while(n--){  cin >> x;  p.push\_back(x);  }  sort(p.begin(), p.end());  p.erase(unique(p.begin(), p.end()), p.end());  cout << p.size() << endl;  return 0;  } |
| #10 Exact Sum  <https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=1998> | #include <bits/stdc++.h>  using namespace std;  int main(){  ios\_base::sync\_with\_stdio(false);  cin.tie(nullptr);  string linea;  while(getline(cin >> ws, linea)){  stringstream ss(linea);  int n, m;  ss >> n;  int\* p = new int[n];  for(int i = 0; i < n; i++){  cin >> p[i];  }  cin >> m;  int min = 1000002, l, r;  sort(p, p+n);  for(int i = 0; i < n; i++){  for(int j = i+1; j < n; j++){  if(p[i]+p[j]==m){  if(p[j]-p[i] < min){  min = p[j]-p[i];  l = i;  r = j;  }  }  }  }  cout << "Peter should buy books whose prices are " << p[l] << " and " << p[r] << "." << endl;  cout << endl;  }  return 0;  } |

**2) Escribir un programa que genera dos ficheros TXT, un fichero con datos de entrada y otro con datos de salida. Estos datos deben servir para probar la misma solución anterior en otra plataforma.**

Se encuentra en el GitHub adjunto:

[https://github.com/byandrev/taller-estructura-de-datos-avanzada](https://github.com/byandrev/taller-estructura-de-datos-avanzadas)

Inputs y Outputs:

<https://github.com/byandrev/taller-estructura-de-datos-avanzadas/tree/main/outputs>

**3) Explicar la solución con un ejemplo gráfico, con colores, paso a paso. Por ejemplo, si se requiere armar un árbol y luego recorrerlo, debe mostrarse un ejemplo completo de manera gráfica. Si se deben considerar varios escenarios o casos bordes, deben incluirse en esta explicación.**

| **EJERCICIO** | **LINK A EXPLICACIÓN GRÁFICA** |
| --- | --- |
| #1 Complete Binary Tree Inserter <https://leetcode.com/problems/complete-binary-tree-inserter> | <https://youtu.be/7kr5_hCVRxs> |
| #2 Binary Tree Inorder Traversal  <https://leetcode.com/problems/binary-tree-inorder-traversal/> | <https://www.canva.com/design/DAFjLsT8rHo/cx5NRZ0acJCwnyRUnOXiVQ/edit?utm_content=DAFjLsT8rHo&utm_campaign=designshare&utm_medium=link2&utm_source=sharebutton> |
| #3 Validate Binary Search Tree  <https://leetcode.com/problems/validate-binary-search-tree> | <https://youtu.be/yC0Hg_kZn7E> |
| #4 Minimum Height Trees  <https://leetcode.com/problems/minimum-height-trees/> | <https://www.canva.com/design/DAFjkBHPWZo/Jv3XEbG84dp-O5RtxGrm_w/edit?utm_content=DAFjkBHPWZo&utm_campaign=designshare&utm_medium=link2&utm_source=sharebutton> |
| #5 Find a Corresponding Node of a Binary Tree in a Clone of That Tree <https://leetcode.com/problems/find-a-corresponding-node-of-a-binary-tree-in-a-clone-of-that-tree/> | <https://www.canva.com/design/DAFjLVz2pFM/2bE7uNo5OSdqlHVNIdyk0Q/view?utm_content=DAFjLVz2pFM&utm_campaign=designshare&utm_medium=link&utm_source=publishsharelink> |
| #6 Find if Path Exists in Graph  <https://leetcode.com/problems/find-if-path-exists-in-graph/> | <https://www.youtube.com/watch?v=sAhlXi5wTZY&t=12s&ab_channel=CamiloBotelloMaldonado> |
| #7 Relative Ranks  <https://leetcode.com/problems/relative-ranks/> | <https://www.canva.com/design/DAFjOfqiDuU/cOVcuSwcwPRMpu19kvcDDw/edit?utm_content=DAFjOfqiDuU&utm_campaign=designshare&utm_medium=link2&utm_source=sharebutton> |
| #8 C Heap Operations  <https://codeforces.com/contest/681/problem/C> | <https://www.youtube.com/watch?v=lj5qhAkL-Gk&t=4s&ab_channel=CamiloBotelloMaldonado> |
| #9 Count Distinct Integers  <https://atcoder.jp/contests/abc240/tasks/abc240_b?lang=en> | <https://www.canva.com/design/DAFjH2MV78E/d8RVzHUFqx9oL-1OwiXi2Q/view?utm_content=DAFjH2MV78E&utm_campaign=designshare&utm_medium=link&utm_source=publishsharelink> |
| #10 Exact Sum  <https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=1998> | <https://www.canva.com/design/DAFjNm9SqpA/8zIiRSlnKfvjLuviSNPm7w/view?utm_content=DAFjNm9SqpA&utm_campaign=designshare&utm_medium=link&utm_source=publishsharelink> |

**4) Publicar todo lo anterior en github en un repositorio, usando el README para la documentación. Subir a UVIRTUAL un ZIP completo del repositorio.**

Link al repositorio GitHub:

<https://github.com/byandrev/taller-estructura-de-datos-avanzadas>