Tree Questions Cheat Sheet for Competitive Programming

# 1. Tree Basics

struct TreeNode {  
 int val;  
 TreeNode \*left, \*right;  
 TreeNode(int x) : val(x), left(NULL), right(NULL) {}  
};

# 2. Tree Traversal

## Inorder Traversal

void inorder(TreeNode\* root) {  
 if (root) {  
 inorder(root->left);  
 cout << root->val << " ";  
 inorder(root->right);  
 }  
}

## Preorder Traversal

void preorder(TreeNode\* root) {  
 if (root) {  
 cout << root->val << " ";  
 preorder(root->left);  
 preorder(root->right);  
 }  
}

## Postorder Traversal

void postorder(TreeNode\* root) {  
 if (root) {  
 postorder(root->left);  
 postorder(root->right);  
 cout << root->val << " ";  
 }  
}

# 3. Binary Search Tree (BST)

## Insert

TreeNode\* insert(TreeNode\* root, int key) {  
 if (!root) return new TreeNode(key);  
 if (key < root->val) root->left = insert(root->left, key);  
 else root->right = insert(root->right, key);  
 return root;  
}

## Search

bool search(TreeNode\* root, int key) {  
 if (!root) return false;  
 if (root->val == key) return true;  
 if (key < root->val) return search(root->left, key);  
 else return search(root->right, key);  
}

# 4. Lowest Common Ancestor (LCA)

## Binary Tree LCA

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

# 5. Segment Trees

## Build Segment Tree

void build(int arr[], int segTree[], int left, int right, int pos) {  
 if (left == right) {  
 segTree[pos] = arr[left];  
 return;  
 }  
 int mid = (left + right) / 2;  
 build(arr, segTree, left, mid, 2\*pos+1);  
 build(arr, segTree, mid+1, right, 2\*pos+2);  
 segTree[pos] = segTree[2\*pos+1] + segTree[2\*pos+2];  
}

## Range Query

int rangeQuery(int segTree[], int qlow, int qhigh, int low, int high, int pos) {  
 if (qlow <= low && qhigh >= high) return segTree[pos];  
 if (qlow > high || qhigh < low) return 0;  
 int mid = (low + high) / 2;  
 return rangeQuery(segTree, qlow, qhigh, low, mid, 2\*pos+1) +  
 rangeQuery(segTree, qlow, qhigh, mid+1, high, 2\*pos+2);  
}

# 6. Fenwick Tree (Binary Indexed Tree)

## Update

void updateBIT(int BITree[], int n, int index, int val) {  
 index = index + 1;  
 while (index <= n) {  
 BITree[index] += val;  
 index += index & (-index);  
 }  
}

## Query

int getSum(int BITree[], int index) {  
 int sum = 0;  
 index = index + 1;  
 while (index > 0) {  
 sum += BITree[index];  
 index -= index & (-index);  
 }  
 return sum;  
}

# 7. Heavy-Light Decomposition

## HLD Basics

void hld(int curNode, int cost, int prev) {  
 if (chainHead[chainNo] == -1) chainHead[chainNo] = curNode;  
 chainInd[curNode] = chainNo;  
 posInBase[curNode] = ptr;  
 baseArray[ptr++] = cost;  
  
 int sc = -1, ncost;  
 for (int i = 0; i < adj[curNode].size(); i++) {  
 if (adj[curNode][i] != prev) {  
 if (sc == -1 || subsize[sc] < subsize[adj[curNode][i]]) {  
 sc = adj[curNode][i];  
 ncost = costs[curNode][i];  
 }  
 }  
 }  
 if (sc != -1) hld(sc, ncost, curNode);  
 for (int i = 0; i < adj[curNode].size(); i++) {  
 if (adj[curNode][i] != prev && adj[curNode][i] != sc) {  
 chainNo++;  
 hld(adj[curNode][i], costs[curNode][i], curNode);  
 }  
 }  
}

# 8. Tree Diameter

## Tree Diameter using DFS

pair<int, int> dfs(int node, int parent, int depth) {  
 pair<int, int> res = {depth, node};  
 for (auto &child : adj[node]) {  
 if (child != parent) {  
 res = max(res, dfs(child, node, depth + 1));  
 }  
 }  
 return res;  
}  
  
int treeDiameter(int root) {  
 pair<int, int> p = dfs(root, -1, 0);  
 pair<int, int> q = dfs(p.second, -1, 0);  
 return q.first;  
}

# 9. Centroid Decomposition

## Centroid Decomposition Basics

void dfs(int node, int parent) {  
 subsize[node] = 1;  
 for (int child : adj[node]) {  
 if (child != parent && !deleted[child]) {  
 dfs(child, node);  
 subsize[node] += subsize[child];  
 }  
 }  
}  
  
int getCentroid(int node, int parent, int n) {  
 for (int child : adj[node]) {  
 if (child != parent && !deleted[child] && subsize[child] > n / 2) {  
 return getCentroid(child, node, n);  
 }  
 }  
 return node;  
}  
  
void centroidDecomposition(int root) {  
 dfs(root, -1);  
 int centroid = getCentroid(root, -1, subsize[root]);  
 deleted[centroid] = true;  
 // Process the centroid  
  
 for (int child : adj[centroid]) {  
 if (!deleted[child]) {  
 centroidDecomposition(child);  
 }  
 }  
}

# 10. Miscellaneous Tips

- Use recursion wisely and handle base cases properly.  
- Always keep an eye on time complexity, especially for large inputs.  
- Make use of efficient data structures like Fenwick Tree, Segment Tree, etc.  
- Practice problems on tree traversals, subtree queries, path queries, etc.  
- Read editorial and other solutions after solving to learn new techniques.