Lecture 32 Deque, Binary Trees

1. <https://leetcode.com/problems/sliding-window-maximum/>

class Solution {

public:

vector<int> maxSlidingWindow(vector<int>& nums, int k) {

deque<int> Q(k);

int n = nums.size();

int i=0;

vector<int> ans;

// process k elements

for(i=0; i<k; i++)

{

while(!Q.empty() && nums[i] > nums[Q.back()])

{

Q.pop\_back();

}

Q.push\_back(i);

}

// process remaining elements

for(; i<n; i++)

{

ans.push\_back(nums[Q.front()]);

// remove elements which are not part of window

while(!Q.empty() && (Q.front() <= i-k))

{

Q.pop\_front();

}

// remove elements which are not useful

while(!Q.empty() && nums[i] > nums[Q.back()])

{

Q.pop\_back();

}

// add new element

Q.push\_back(i);

}

ans.push\_back(nums[Q.front()]);

return ans;

}

};

1. Preorder, Inorder, PostOrder
2. #include <iostream>
3. using namespace std;
4. class node
5. {
6. public:
7. int data;
8. node\* left;
9. node\* right;
10. node(int d)
11. {
12. data = d;
13. left = NULL;
14. right = NULL;
15. }
16. };
17. node\* buildTree()
18. {
19. int d;
20. cin>>d;
21. if(d==-1)
22. {
23. return NULL;
24. }
25. node\* root = new node(d);
26. root->left = buildTree();
27. root->right = buildTree();
28. return root;
29. }
30. void print(node\* root)
31. {
32. if(root==NULL)
33. {
34. return;
35. }
36. cout<<root->data<<" ";
37. print(root->left);
38. print(root->right);
39. }
40. void printInorder(node\* root)
41. {
42. if(root==NULL)
43. {
44. return;
45. }
46. printInorder(root->left);
47. cout<<root->data<<" ";
48. printInorder(root->right);
49. }
50. void printPostorder(node\* root)
51. {
52. if(root==NULL)
53. {
54. return;
55. }
56. printPostorder(root->left);
57. printPostorder(root->right);
58. cout<<root->data<<" ";
59. }
60. int main() {
61. node\* root = buildTree();
62. print(root);
63. cout<<endl;
64. printInorder(root);
65. cout<<endl;
66. printPostorder(root);
67. return 0;
68. }

3. <https://leetcode.com/problems/maximum-depth-of-binary-tree/>

class Solution {

public:

int maxDepth(TreeNode\* root) {

if(root==NULL)

{

return 0;

}

int ls = maxDepth(root->left);

int rs = maxDepth(root->right);

return max(ls, rs)+1;

}

};