Assignment #4

Question #1:

#include <iostream>

#include <string>

#include<unordered\_map>

using namespace std;

class Deque

{

private:

char\* arr;

int sizee, front, rear;

public:

Deque(int size)

{

arr = new char[size];

sizee = size;

front = -1;

rear = 0;

}

void insertFront(char x)

{

if (front == -1)

{

arr[++front] = x;

}

else if (front == 0)

{

cout << "Deque is full from the front \n";

}

else

{

arr[--front] = x;

}

}

void insertRear(char x)

{

if (front == -1)

{

arr[++front] = x;

}

else if (rear == sizee - 1)

{

cout << "Deque is full from the rear\n";

}

else

{

arr[++rear] = x;

}

}

char deleteFront()

{

if (front == -1)

{

cout << "Deque is empty\n";

return '10';

}

else

{

char x = arr[front];

if (front == rear)

{

front = -1;

rear = 0;

}

else

{

front++;

}

return x;

}

}

char deleteRear()

{

if (front == -1)

{

cout << "Deque is empty\n";

return '0';

}

else

{

char x = arr[rear];

if (front == rear)

{

front = -1;

rear = 0;

}

else

{

rear -= 1;

}

return x;

}

}

bool isEmpty()

{

return (front == -1);

}

bool isFull()

{

return ((front == 0 && rear == sizee - 1) || front == rear + 1);

}

int size()

{

if (front == -1)

{

return 0;

}

else if (rear >= front)

{

return rear - front + 1;

}

else

{

return sizee - (front - rear) + 1;

}

}

char frontElement()

{

return arr[front];

}

char rearElement()

{

return arr[rear];

}

~Deque()

{

delete[] arr;

}

};

string findLongestSubstring(string S, int K)

{

int start = 0, end = 0, maxLen = 0;

unordered\_map<char, int>freq;

Deque dq(S.size());

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

{

dq.insertRear(S[i]);

freq[S[i]]++;

while (freq.size() > K)

{

char ch = dq.deleteFront();

freq[ch]--;

if (freq[ch] == 0)

{

freq.erase(ch);

}

start++;

}

int len = i - start + 1;

if (len > maxLen)

{

maxLen = len;

end = i;

}

}

return S.substr(start, maxLen);

}

int main()

{

string S;

int K;

cout << "Enter string S:";

cin >> S;

cout <<"Enter integer K:";

cin >> K;

string result = findLongestSubstring(S, K);

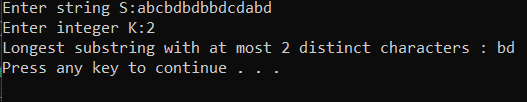
cout << "Longest substring with at most " << K << " distinct characters : "<<result << endl;

system("pause");

return 0;

}

Output:



Question #2:

#include<iostream>

using namespace std;

class BST

{

public:

int\* array;

int\* arr;

int size;

int tree[10];

BST(int s)

{

size = s;

array = new int[size];

arr = new int[size];

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

{

array[i] =arr[i]= - 1;

}

}

void input()

{

cout << "ENTER ELEMENTS OF AN ARRAY : " << endl;

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

{

cin >> array[i];

}

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

{

arr[i]=array[i];

}

int parent, parent1;

for (int i = 0; i < 10; i++)

{

tree[i] = -1;

}

tree[1] = array[0];

parent = parent1 = 1;

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

{

if (array[i] > tree[parent])

{

if (array[i] < tree[parent])

{

tree[parent \* 2] = array[i];

parent = parent \* 2;

}

else

{

tree[(parent \* 2) + 1] = array[i];

parent = (parent \* 2) + 1;

}

}

}

parent = 1;

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

{

if (array[i] < tree[parent1])

{

if (array[i] < tree[parent])

{

tree[parent \* 2] = array[i];

parent = parent \* 2;

}

else if (array[i] > tree[parent])

{

tree[(parent \* 2) + 1] = array[i];

parent = (parent \* 2) + 1;

}

}

}

cout << " THE TREE IS : " << endl;

for (int i = 0; i < 10; i++)

{

cout << i << " --> " << tree[i] << endl;

}

}

void permutation()

{

for (int i = 1; i < 5 - 1; i++)

{

swap(arr[1], arr[i + 1]);

int parent, parent1;

int tree1[10];

for (int i = 0; i < 10; i++)

{

tree1[i] = -1;

}

tree1[1] = arr[0];

parent = parent1 = 1;

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

{

if (arr[i] > tree1[parent])

{

if (arr[i] < tree1[parent])

{

tree1[parent \* 2] = arr[i];

parent = parent \* 2;

}

else

{

tree1[(parent \* 2) + 1] = arr[i];

parent = (parent \* 2) + 1;

}

}

}

parent = 1;

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

{

if (arr[i] < tree1[parent1])

{

if (arr[i] < tree1[parent])

{

tree1[parent \* 2] = arr[i];

parent = parent \* 2;

}

else if (arr[i] > tree[parent])

{

tree1[(parent \* 2) + 1] = arr[i];

parent = (parent \* 2) + 1;

}

}

}

for (int i = 0; i < 10; i++)

{

if (tree1[i] == tree[i])

{

if (i == 9)

{

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

{

cout << arr[i];

}

}

}

else

{

break;

}

}

}

}

};

int main()

{

int s;

cout << " ENTER NO OF ELEMENTS IN AN ARRAY : " << endl;

cin >> s;

BST obj(s);

cout << "=============================================================" << endl;

obj.input();

cout << "=============================================================" << endl;

cout << " THE PERMUTATION ARE GIVEN BVELOW " << endl;

for (int i = 0; i < 5; i++)

{

obj.permutation();

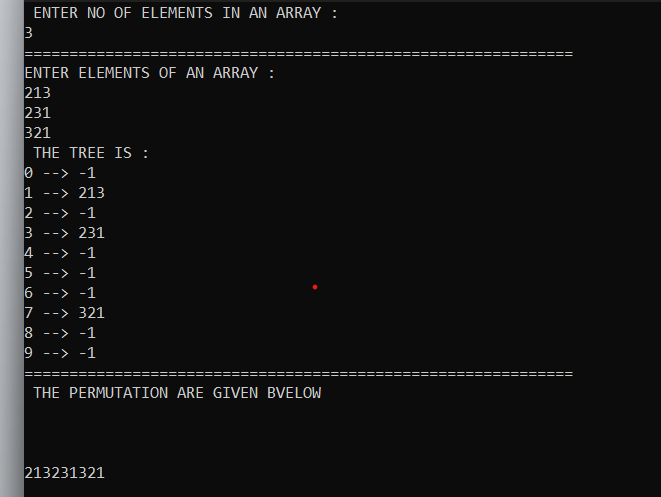
cout << endl;

}

system("pause");

}

**Output:**

****

Question #3:

#include<iostream>

using namespace std;

class Node

{

public:

int data;

Node\* left;

Node\* right;

Node(int d)

{

data = d;

left = NULL;

right = NULL;

}

};

void preorder(Node\* root)

{

if (root == NULL)

{

return;

}

cout << root->data << " ";

preorder(root->left);

preorder(root->right);

}

void inorder(Node\* root)

{

if (root==NULL)

{

return;

}

inorder(root->left);

cout << root->data << " ";

inorder(root->right);

}

void postorder(Node\* root)

{

if (root==NULL)

{

return;

}

postorder(root->left);

postorder(root->right);

cout << root->data << " ";

}

int main()

{

Node\* root = new Node(1);

root->left = new Node(2);

root->right = new Node(3);

root->left->left = new Node(4);

root->left->right = new Node(5);

root->right->left = new Node(6);

root->right->right = new Node(7);

root->right->left->left = new Node(8);

root->right->left->right = new Node(9);

int choice;

cout << "Enter the Choice " << endl;

cout << "1. To Check preorder Nodes " << endl;

cout << "2. To Check Inorder Nodes " << endl;

cout << "3. To Check Post Order Nodes " << endl;

cin >> choice;

if (choice==1)

{

cout << "The Nodes of preoder : ";

preorder(root);

cout << endl;

}

else if (choice == 2)

{

cout << "The Nodes of Inorder : ";

inorder(root);

cout << endl;

}

else if (choice == 3)

{

cout << "The Nodes of Postorder : ";

postorder(root);

cout << endl;

}

else

{

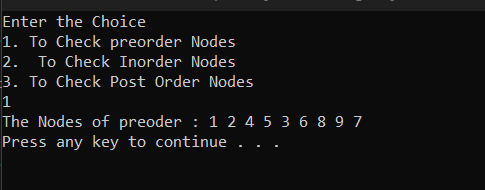
cout << "Sorry Wrong Input " << endl;

}

system("pause");

}

Output:



Question #4:

#include<iostream>

#include<algorithm>

using namespace std;

struct Node

{

public:

int data;

Node\*left, \*right;

Node(int va)

{

data = va;

left = NULL;

right = NULL;

}

};

Node\* insertBSt(Node\* root, int val)

{

if (root == NULL)

{

return new Node(val);

}

else if (val < root->data)

{

root->left = insertBSt(root->left, val);

}

else

{

root->right = insertBSt(root->right, val);

}

return root;

}

//void inorder(Node\* root)

//{

// if (root == NULL)

// {

// return;

// }

// inorder(root->left);

// cout << root->data << " ";

// inorder(root->right);

//}

Node\* searchinBST(Node\* root,int key)

{

if (root==NULL)

{

return NULL;

}

if (root->data==key)

{

return root;

}

//key>data

if (root->data>key)

{

return searchinBST(root->left, key);

}

//data<key

return searchinBST(root->right, key);

}

Node\* inordersucc(Node\* root)

{

Node\* curr = root;

while (curr&&curr->left!=NULL)

{

curr = curr->left;

}

return curr;

}

Node\* deleteinBST(Node\* root, int key)

{

if (key<root->data)

{

root->left = deleteinBST(root->left, key);

}

else if (key > root->data)

{

root->right = deleteinBST(root->right, key);

}

else

{

if (root->left==NULL)

{

Node\* temp = root->right;

free(root);

return temp;

}

else if (root->right == NULL)

{

Node\* temp = root->left;

free(root);

return temp;

}

//case 3

Node\* temp = inordersucc(root->right);

root->data = temp->data;

root->right = deleteinBST(root->right, temp->key);

}

return root;

}

int countNodes(Node\* root)

{

if (root==NULL)

{

return 0;

}

return countNodes(root->left) + countNodes(root->right) + 1;

}

void inorder(Node\* root)

{

if (root==NULL)

{

return;

}

inorder(root->left);

cout << root->data << " ";

inorder(root->right);

}

int sumNodes(Node\* root)

{

if (root==NULL)

{

return 0;

}

return sumNodes(root->left) + sumNodes(root->right) + root->data;

}

int caluheight(Node\* root)

{

if (root==NULL)

{

return 0;

}

int lHeight = caluheight(root->left);

int rHeight = caluheight(root->right);

return max(lHeight, rHeight) + 1;

}

int main()

{

//Node\* root = new Node(4);

/\*root->left = new Node(2);

root->right = new Node(5);

root->left->left = new Node(1);

root->left->right = new Node(3);

root->right->right = new Node(6);\*/

Node\* root = NULL;

root = insertBSt(root, 3);

insertBSt(root,2);

insertBSt(root, 5);

insertBSt(root,4);

insertBSt(root, 6);

cout << "The inorder traversal is " << endl;

inorder(root);

cout << endl;

cout << "The number of Nodes in the tree : "<<countNodes(root)<< " " << endl;

cout << "The height of the tree is : " << caluheight(root) << " " << endl;

cout << endl;

if (searchinBST(root,5)==NULL)

{

cout << "Key does not exsist " << endl;

}

else

{

cout << "key exsist " << endl;

}

root = deleteinBST(root, 5);

//inorder(root);

cout << "The sum of Nodes = " << sumNodes(root) << endl; //b part

system("pause");

}

Output:

