Emilie Gunti
CSE 310
Janaka Balasooriya – Summer 2023

Homework 4: Submission Template

1. Copy and paste your complete program for question 1 here.

```
/*
Author: Emilie Gunti
Class: CSE310
Assignment: Homework 4, Question 1
Description: This assignment is a binary search tree that includes the functions: insert, in-order, pre-order, and post-order traversal, finding and removing min, the successor method, and lastly the delete method. Given any integer, it will insert into the tree and upon request it will do the remaining actions of that are given.
it was developed based on the given BST cpp file.
*/

#include <iostream>
#include <cstdlib>
using namespace std;

class BST
{
public:
    class node
    {
    public:
        node* left;
        node* right;
```

```
23.         node* parent;
24.         int key;
25.         string data;
26.     };
27.
28.     node* root;
29.     BST()
30.     {
31.         root = NULL;
32.     }
33.     bool isEmpty() const { return root == NULL; }
34.     void TREE_INSERT(int);
35.     void INORDER_TREE_WALK(node*);
36.     void POSTORDER_TREE_WALK(node*);
37.     void PREORDER_TREE_WALK(node*);
38.     node* FIND_MIN(node*);
39.     void REMOVE_MIN();
40.     node* SUCCESSOR(node*);
41.     void CHANGE(node*, node*);
42.     void DELETE(int);
43.
44.
45.
46. };
47.
48. void BST::TREE_INSERT(int d){
49.
```

```
50.  // modified insert method, after noticing the parent pointer in the code wasn't
     correctly set
51.  node* z = new node();
52.  z->key = d;
53.  z->left = NULL;
54.  z->right = NULL;
55.
56.  node* y = NULL;
57.  node* x = root;
58.  node* parent = NULL;
59.
60.  while (x != NULL)
61.  {
62.      y = x;
63.      if (z->key < x->key)
64.          x = x->left;
65.      else
66.          x = x->right;
67.  }
68.
69.  parent = y;
70.  if (y == NULL)
71.      root = z;
72.  else if (z->key < parent->key)
73.      parent->left = z;
74.  else
75.      parent->right = z;
```

```
76.    z->parent = parent;
77.
78.
79.}
80.
81.void BST::INORDER_TREE_WALK(node* x){
82.    if (x == NULL) {
83.        cout << "Tree is currently empty.";
84.    }
85.    if (x != NULL)
86.    {
87.        if (x->left) INORDER_TREE_WALK(x->left);
88.        cout << " " << x->key << " ";
89.        if (x->right) INORDER_TREE_WALK(x->right);
90.    }
91.
92.
93.}
94.
95.void BST:: POSTORDER_TREE_WALK(node* x){
96.    if (x == NULL) {
97.        cout << "Tree is currently empty.";
98.    }
99.    if (x!= NULL) {
100.            if ( x-> left) POSTORDER_TREE_WALK(x-> left);
101.            if (x -> right) POSTORDER_TREE_WALK(x-> right);
102.            cout << " " << x->key << " ";
```

```
103.            }
104.
105.        }
106.
107.        void BST:: PREORDER_TREE_WALK(node* x) {
108.            if (x == NULL) {
109.                cout << "Tree is currently empty.";
110.            }
111.            if (x!= NULL) {
112.                cout << " " << x->key << " ";
113.                if ( x-> left) PREORDER_TREE_WALK(x-> left);
114.                if (x -> right) PREORDER_TREE_WALK(x-> right);
115.
116.        }
117.
118.        }
119.
120.        BST::node* BST:: FIND_MIN(node* x){
121.            if (x == NULL)
122.                return NULL;
123.                while (x->left != NULL)
124.                    x = x->left;
125.            return x;
126.
127.        }
128.
129.        void BST:: REMOVE_MIN() {
```

```
130.        if (root ==NULL){
131.            return;
132.        }
133.
134.        if( root ->left == NULL && root->right ==NULL){
135.            delete root;
136.            root = NULL;
137.            return;
138.        }
139.
140.        node* x = root;
141.        node* parent = NULL;
142.        while (x->left != NULL) {
143.            parent = x;
144.            x = x->left;
145.        }
146.        if (x->right != NULL){
147.            if (parent !=NULL) {
148.                parent->left = x->right;
149.            }
150.            else{
151.                root = x->right;
152.            }
153.
154.            delete x;
155.        }
156.        else {
```

```
157.             if (parent != NULL) {
158.                 parent->left = NULL;
159.             }
160.             else {
161.                 root = NULL;
162.             }
163.
164.             delete x;
165.         }
166.     }
167.
168.     BST::node* BST:: SUCCESSOR(node* x){
169.         if (x == NULL) {
170.             cout << "Element doesn't exist. No successor. ";
171.             return NULL;
172.         }
173.         if (x->right !=NULL) {
174.             x = x->right;
175.             while (x->left !=NULL)
176.                 x = x->left;
177.             return x;
178.
179.         }
180.         node* y = x->parent;
181.         while (y != NULL && x == y->right) {
182.             x=y;
183.             y = y->parent;
```

```
184.            }
185.            return y;
186.
187.        }
188.
189.
190.        //helpder method to replace a subtree with another subtree as the child,
    called transplant in textbook
191.        void BST::CHANGE(node* p, node* q){
192.            if (p ->parent == NULL)
193.                root = q;
194.            else if ( p == p->parent->left)
195.                p->parent->left = q;
196.            else
197.                p->parent->right = q;
198.            if (q != NULL){
199.                q->parent = p->parent;
200.            }
201.
202.        }
203.
204.        void BST:: DELETE( int n){
205.            node* z = root;
206.            while (z != NULL && z->key != n){
207.                if ( n < z->key)
208.                    z = z->left;
209.                else
```

```
210.                z = z->right;
211.            }
212.            if (z == NULL) {
213.                cout << "Element doesn't exist.";
214.                return;
215.            }
216.            if (z-> left==  NULL) {
217.                CHANGE(z, z->right);
218.                delete z;
219.                cout << "Node with element "<< n << " is deleted." << endl;
220.            }
221.            else if ( z->right == NULL){
222.                CHANGE(z, z->left);
223.                delete z;
224.                cout << "Node with element "<< n << " is deleted." << endl;
225.            }
226.            else {
227.                node* y = FIND_MIN( z->right);
228.                if (y->parent != z){
229.                    CHANGE(y, y->right);
230.                    y->right = z->right;
231.                    y->right->parent = y;
232.                }
233.                CHANGE(z, y);
234.                y->left = z->left;
235.                y->left->parent = y;
236.                delete z;
```

```
237.             cout << "Node with element "<< n << " is deleted." << endl;
238.         }
239.     }
240.
241.     int main()
242.     {
243.         BST bst;
244.         int choice, key;
245.         while (true)
246.         {
247.             cout << endl << endl;
248.             cout << "Binary Search Tree" << endl;
249.             cout << " ---------------------------- " << endl;
250.             cout << " 1. Insert a node " << endl;
251.             cout << " 2. Pre-Order Traversal " << endl;
252.             cout << " 3. Post-Order Traversal " << endl;
253.             cout << " 4. In-Order Traversal " << endl;
254.             cout << " 5. Find Min " << endl;
255.             cout << " 6. Remove Min " << endl;
256.             cout << " 7. Successor " << endl;
257.             cout << " 8. Delete a Node " << endl;
258.             cout << " 9. Exit " << endl;
259.             cout << " Enter your choice : ";
260.             cin >> choice;
261.
262.             if (choice == 1){
263.                 cout << " Enter key (int value) to be inserted : ";
```

```
264.                cin >> key;
265.                bst.TREE_INSERT(key);
266.                continue;
267.            }
268.            else if (choice == 2){
269.                cout << endl;
270.                cout << " Pre-Order Traversal " << endl;
271.                cout << " --------------------" << endl;
272.                bst.PREORDER_TREE_WALK(bst.root);
273.                continue;
274.            }
275.            else if (choice == 3){
276.                cout << " Post-Order Traversal " << endl;
277.                cout << " --------------------" << endl;
278.                bst.POSTORDER_TREE_WALK(bst.root);
279.                continue;
280.            }
281.            else if (choice == 4){
282.                cout << endl;
283.                cout << " In-Order Traversal " << endl;
284.                cout << " --------------------" << endl;
285.                bst.INORDER_TREE_WALK(bst.root);
286.            continue;
287.            }
288.
289.            else if (choice == 5){
290.                BST::node* min = bst.FIND_MIN(bst.root);
```

```
291.                if (min != NULL){
292.                    cout << "Minimum value in tree: " << min->key << endl;
293.                }
294.                else{
295.                    cout << "Tree is currently empty" << endl;
296.                }
297.            continue;
298.            }
299.
300.            else if (choice == 6){
301.                if (bst.isEmpty()){
302.                    cout << "Tree is currently empty";
303.                }
304.                else {
305.                    bst.REMOVE_MIN();
306.                    cout << "Minimum Node Removed";
307.
308.                }
309.            continue;
310.            }
311.
312.            else if (choice == 7){
313.                if (bst.isEmpty()) {
314.                    cout << "Tree is currently empty.";
315.                }
316.                else {
317.                cout << "Enter key (an int value) to find its successor: ";
```

```
318.                 cin >> key;
319.                     {
320.                         BST:: node* node = bst.root;
321.                         while (node !=NULL && node->key != key ){
322.                             if (key < node->key)
323.                                 node = node->left;
324.                             else
325.                                 node = node->right;
326.                         }
327.                         BST:: node* successor = bst.SUCCESSOR(node);
328.                         if (successor !=NULL){
329.                             cout << "Successor of " << key << " is " << successor->key
    << endl;
330.                         }
331.                         else {
332.                             cout << "Currently no successor found for element " << key
    << endl;
333.                         }
334.                     }
335.                 }
336.             continue;
337.             }
338.
339.             else if (choice == 8){
340.                 if (bst.isEmpty()) {
341.                     cout << "Tree is currently empty.";
342.                 }
```

```
343.                else {
344.                cout << "Enter value (int) to delete: ";
345.                cin >> key;
346.                bst.DELETE(key);
347.                }
348.            continue;
349.            }
350.
351.            else if (choice == 9){
352.                cout << "Exiting the program...";
353.                break;
354.            }
355.            else {
356.                cout << "Invalid choice. Please select a valid option.";
357.                cin.clear();
358.                while(cin.get() != '\n'){
359.                    continue;
360.                }
361.            }
362.        }
363.        return 0;
364.    }
365.
```

366. Copy and paste your complete program for question 2 here.

```
/*
Name: Emilie Gunti
Class: CSE310
Assignment: Homework 4, Question 2
Description: this code is an application of BST, where it stores the landing time and the plane flight number, withdraws landing requests, and lastly lists all landing times and the flight number.
For the request landing, it satisfies the K constraint that the user inputs, grants and inserts landing and name to the BST. Additionally, it removes operation based on whichever landing time they
choose. Lastly, it displays all the landing times with its plane information stored in the landingTimes BST.
*/

#include <iostream>
using namespace std;

struct Node{
    int landingTime;
    string planeName;
    Node* left;
    Node* right;

};

Node* root;
int k;
```

```
389.
390.    class landingTimes{
391.        public:
392.        landingTimes( int value){
393.            root = nullptr;
394.            k = value;
395.        }
396.
397.        void insert(int, string);
398.        Node* insertNode(Node*, int, string);
399.        Node* removeNode( Node*, int);
400.        Node* getMinNode(Node*);
401.        void listLanding(Node*);
402.        bool KTimeGapSatisfied(Node*,int);
403.
404.    };
405.
406.    Node* landingTimes :: insertNode( Node* x, int time, string plane){
407.        if (x == NULL){
408.            Node* newNode = new Node;
409.            newNode->landingTime = time;
410.            newNode->planeName = plane;
411.            newNode->left = newNode->right = NULL;
412.            return newNode;
413.        }
414.
415.        if (time < x->landingTime) {
```

```
416.             x->left = insertNode (x->left, time, plane);
417.         }
418.         else if (time > x->landingTime){
419.             x->right = insertNode(x->right, time, plane);
420.         }
421.         return x;
422.     }
423.
424.     void landingTimes::insert(int time, string plane) {
425.         if (KTimeGapSatisfied(root, time)){
426.             root = insertNode(root, time, plane);
427.             cout << "Landing request is granted." <<endl;
428.         }
429.         else {
430.             cout << "Landing request is NOT granted, not enough time gap." <<
    endl;
431.         }
432.     }
433.
434.     Node* landingTimes:: getMinNode(Node* x){
435.         if (x == NULL)
436.             return NULL;
437.             while (x->left != NULL){
438.                 x = x->left;
439.             }
440.             return x;
441.     }
```

```
442.
443.    Node* landingTimes::removeNode(Node* x, int time){
444.        if (x == nullptr){
445.            cout << "This landing time does not exist." << endl;
446.            return x;
447.        }
448.
449.        if (time < x->landingTime){
450.            x->left = removeNode(x->left, time);
451.
452.        }
453.        else if (time > x->landingTime){
454.            x->right = removeNode(x->right, time);
455.
456.        }
457.        else {
458.            if (x->left == NULL) {
459.                Node* t = x->right;
460.                delete x;
461.                return t;
462.            }
463.            else if (x->right == NULL){
464.                Node* t = x->left;
465.                delete x;
466.                return t;
467.            }
468.            Node* next = getMinNode(x->right);
```

```
469.                x->landingTime = next->landingTime;
470.                x->planeName = next->planeName;
471.                x->right = removeNode(x->right, next->landingTime);
472.
473.            }
474.
475.            return x;
476.        }
477.
478.
479.
480.        void landingTimes:: listLanding(Node* x){
481.            if (x != NULL){
482.                listLanding(x->left);
483.                cout << "Plane: " << x->planeName << "  Time: " << x->landingTime
    << endl;
484.                listLanding(x->right);
485.
486.            }
487.        }
488.
489.        bool landingTimes:: KTimeGapSatisfied(Node* x, int time){
490.            if (x == NULL){
491.                return true;
492.            }
493.            int diff = abs(x->landingTime - time);
494.            if (diff < k){
```

```
495.             return false;
496.         }
497.         if (time < x->landingTime){
498.             return  KTimeGapSatisfied(x->left, time);
499.         }
500.         else {
501.             return KTimeGapSatisfied(x->right, time);
502.         }
503.     }
504.
505.     int main() {
506.         int KVal, option;
507.         cout << "Enter your K value: ";
508.         cin >> KVal;
509.
510.         landingTimes landingSystem(KVal);
511.
512.         while (true) {
513.             cout << "------------------------" << endl;
514.             cout << "Plane Landing System Menu" << endl;
515.             cout << "------------------------" << endl;
516.             cout << "1. Request Landing." << endl;
517.             cout << "2. Withdraw Landing Request." << endl;
518.             cout << "3. List Landing Times and flight number." << endl;
519.             cout << "4. Exit" << endl;
520.             cout << "Select your option: ";
521.             cin >> option;
```

```
522.
523.            if (option == 1){
524.                int time;
525.                string planeName;
526.                cout << "Enter the landing time: ";
527.                cin >> time;
528.                cout << "Enter the Plane name: ";
529.                cin >> planeName;
530.                landingSystem.insert(time, planeName);
531.
532.
533.            }
534.            else if (option == 2){
535.                int time;
536.                cout << "Enter the landing time of plane to withdraw request: ";
537.                cin >> time;
538.                landingSystem.removeNode(root, time);
539.
540.
541.            }
542.            else if (option == 3){
543.                if (root == NULL) {
544.                    cout << "No landing request currently." << endl;
545.                }
546.                else {
547.                    cout << "Plane numbers and landing times : " << endl;
548.                    landingSystem.listLanding(root);
```

```
549.                }
550.
551.
552.            }
553.            else if (option == 4){
554.                cout << "Exiting the program..." << endl;
555.                break;
556.
557.            }
558.            else{
559.                cout << "Invalid option. Please try again." << endl;
560.                cin.clear();
561.                while (cin.get() != '\n'){
562.
563.                }
564.
565.            }
566.
567.        }
568.        return 0;
569.    }
570.
```

571. Upload your programs for question 1 and question 2 to google drive and provide the links for your programs

Link for the program for question1

Emilie Gunti
CSE 310
Janaka Balasooriya – Summer 2023

https://drive.google.com/file/d/1FHKW5saWgPobcGVoRF0mbzrUemKvWSGq/view?usp=sharing

Link for the program for question2

https://drive.google.com/file/d/1fWfpwOV9W0hIZfk7GisJARQv8vMbcL-h/view?usp=sharing