Practical File

Algorithms and Advanced Data Structures

Name:Upasna Kukreti

Course:B.Sc. Computer Science Hons.

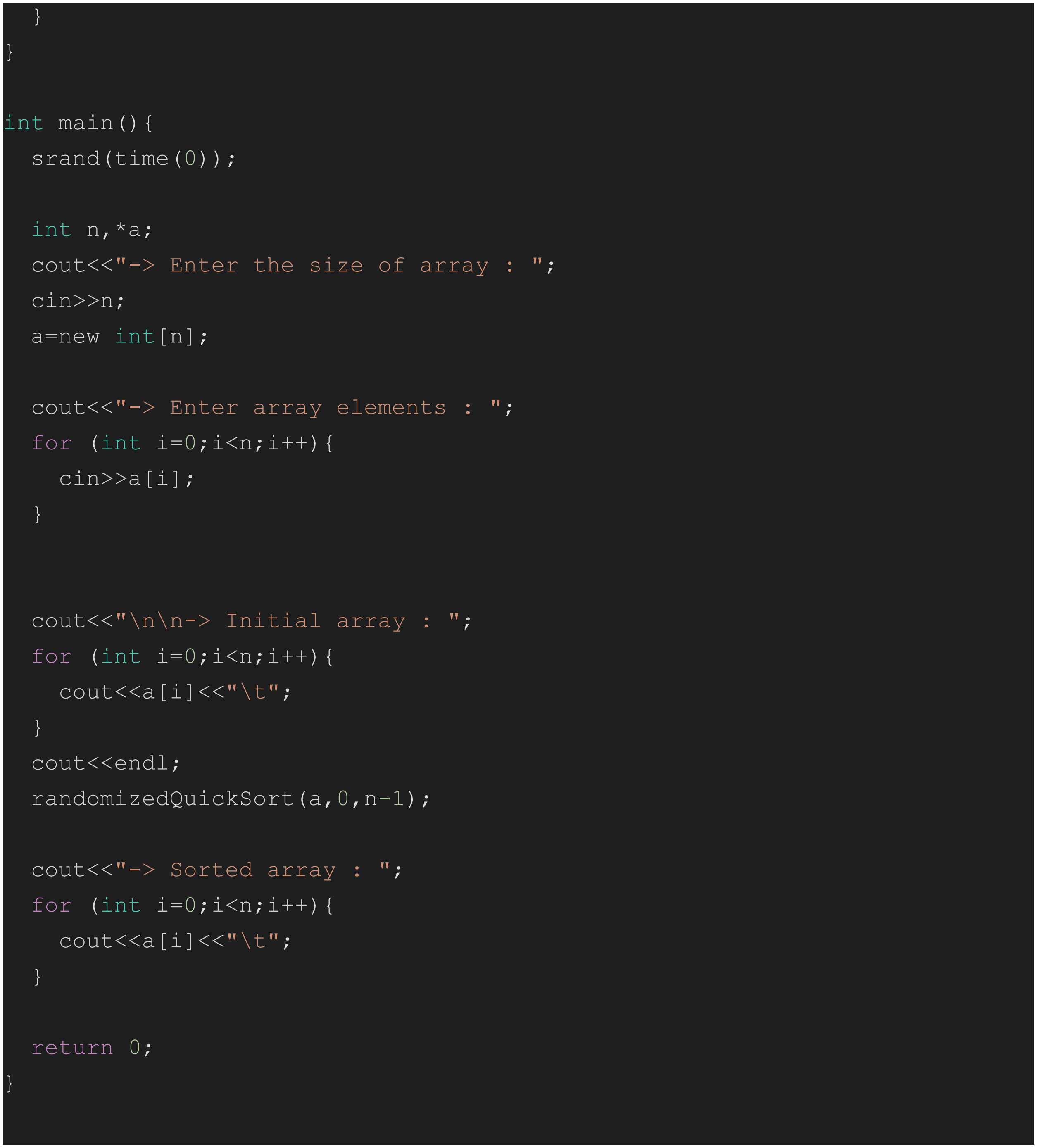
Roll No. : 22/Bs(H)Cs/16126

Semester : V-B

Question 1 : Write a program to sort the elements of an array using Randomized Quick Sort (the program should report the number of comparisons).

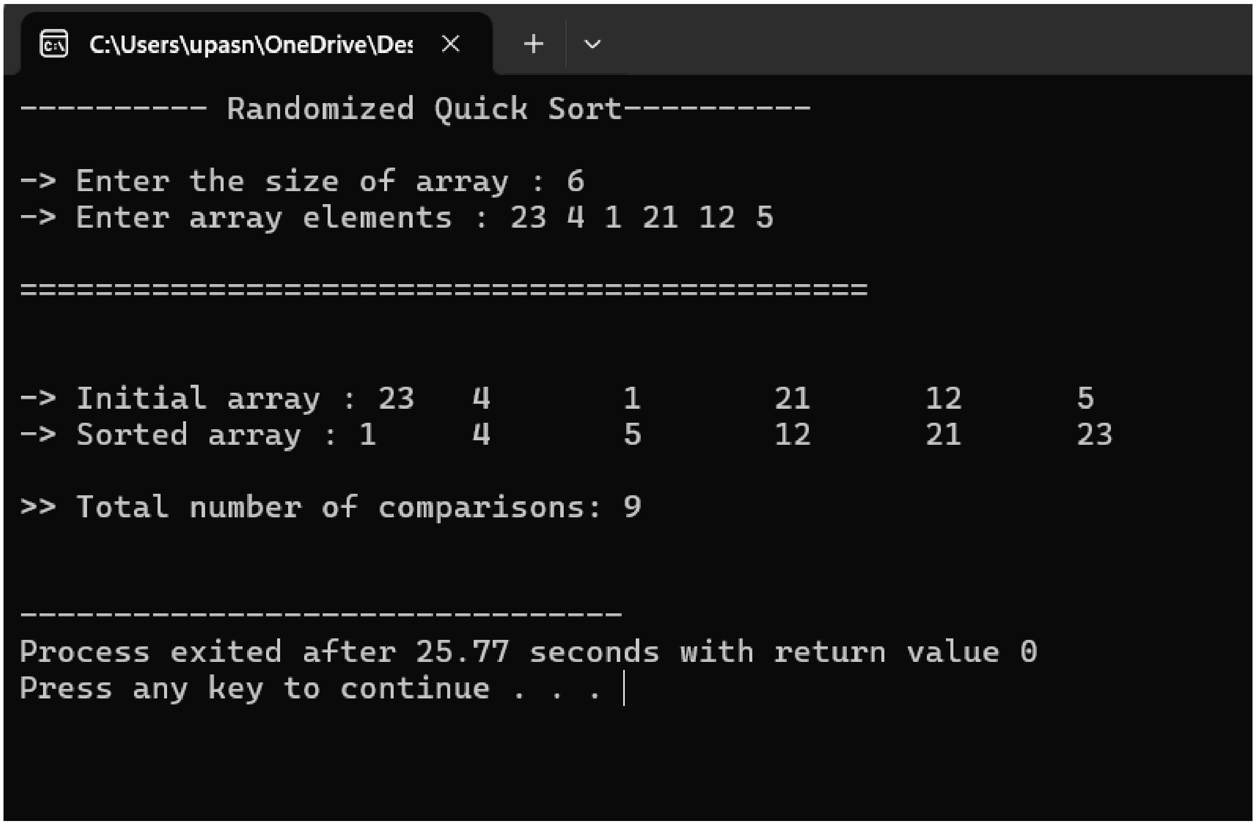
Code :

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

int comparisons;
void swap(int& a, int& b){
  int c = a;
  a = b;
  b = c;
}

int partition(int arr[], int low, int high){
  int pivotIndex= low+rand()%(high-low+1);
  swap(arr[pivotIndex],arr[high]);
  int pivot = arr[high];
  int i = low - 1;
  for(int j=low;j <= high -1;j++){
    comparisons++;
    if(arr[j] <= pivot){
      i++;
      swap(arr[i],arr[j]);
    }
  }
  swap(arr[i+1],arr[high]);
  return (i+1);
}

void randomizedQuickSort(int arr[], int low, int high){
  if(low<high){
    int pi=partition(arr,low, high);
    randomizedQuickSort(arr,low,pi-1);
    randomizedQuickSort(arr,pi+1,high);
```



```
  }
}

int main(){
  srand(time(0));

  int n,*a;
  cout<<"-> Enter the size of array : ";
  cin>>n;
  a=new int[n];

  cout<<"-> Enter array elements : ";
  for (int i=0;i<n;i++){
    cin>>a[i];
  }


  cout<<"\n\n-> Initial array : ";
  for (int i=0;i<n;i++){
    cout<<a[i]<<"\t";
  }
  cout<<endl;
  randomizedQuickSort(a,0,n-1);

  cout<<"-> Sorted array : ";
  for (int i=0;i<n;i++){
    cout<<a[i]<<"\t";
  }

  return 0;
}
```

<u>OUTPUT:-</u>



```
---------- Randomized Quick Sort----------

-> Enter the size of array : 6
-> Enter array elements : 23 4 1 21 12 5

=============================================


-> Initial array : 23   4       1       21      12      5
-> Sorted array : 1     4       5       12      21      23

>> Total number of comparisons: 9


--------------------------------
Process exited after 25.77 seconds with return value 0
Press any key to continue . . .
```

Question 2 : Write a program to find the ith smallest element of an array using Randomized select.

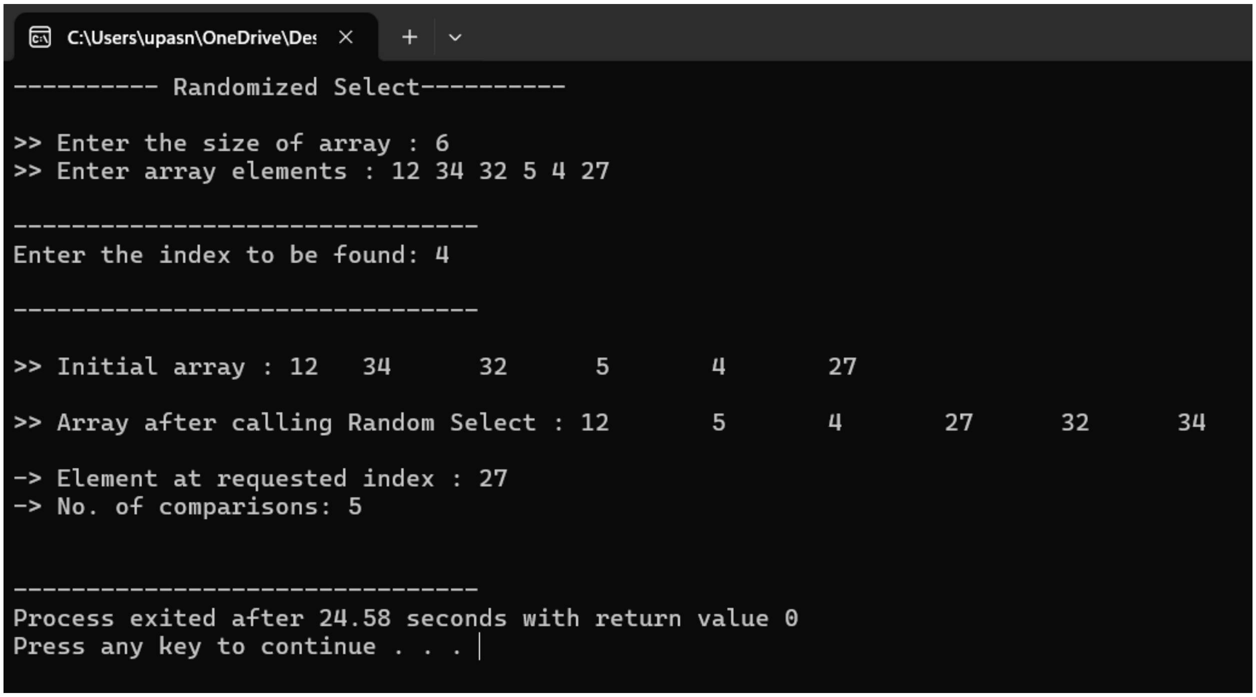
Code :

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

int comparisons;


int rand_partition(int arr[],int low, int high){
  int pivotIndex= low+rand()%(high-low+1);
  swap(arr[pivotIndex],arr[high]);
  int pivot = arr[high];
  int i = low - 1;
  for(int j=low;j <= high -1;j++){
    comparisons++;
    if(arr[j] <= pivot){
      i++;
      swap(arr[i],arr[j]);
    }
  }
  swap(arr[i+1],arr[high]);
  return (i+1);
}


int rand_select(int A[],int p, int r, int i){
  if(p==r){
    return A[p];
  }
  int q,k;
  q=rand_partition(A,p,r);
  k=q-p+1;
  if(i==k){
    return A[q];
  }
  else if(i<k){
    return rand_select(A,p,q-1,i);
  }
  else{
    return rand_select(A,q+1,r,i-k);
  }
}


int main(){
  srand(time(0));

  int n,*a;
  cout<<"-> Enter the size of array : ";
  cin>>n;
  a=new int[n];

  cout<<"-> Enter array elements : ";
  for (int i=0;i<n;i++){
    cin>>a[i];
```

```
}


int rank;
cout<<"\nEnter the index to be found: ";
cin>>rank;


if(rank>n){
  cout<<"\nInvalid Index!!!!";
  exit(0);
}


cout<<"\n\n-> Initial array : ";
for (int i=0;i<n;i++){
  cout<<a[i]<<"\t";
}
cout<<endl;
int ans;
ans=rand_select(a,0,n-1,rank);
```

```
  cout<<"\n\n-> Array after calling Random Select : ";

  for (int i=0;i<n;i++){

    cout<<a[i]<<"\t";

  }



  cout<<"\n\n-> Element at requested index : "<<ans;

  cout<<"\n\n No. of comparisons: "<<comparisons;

  return 0;

}
```

OUTPUT:-

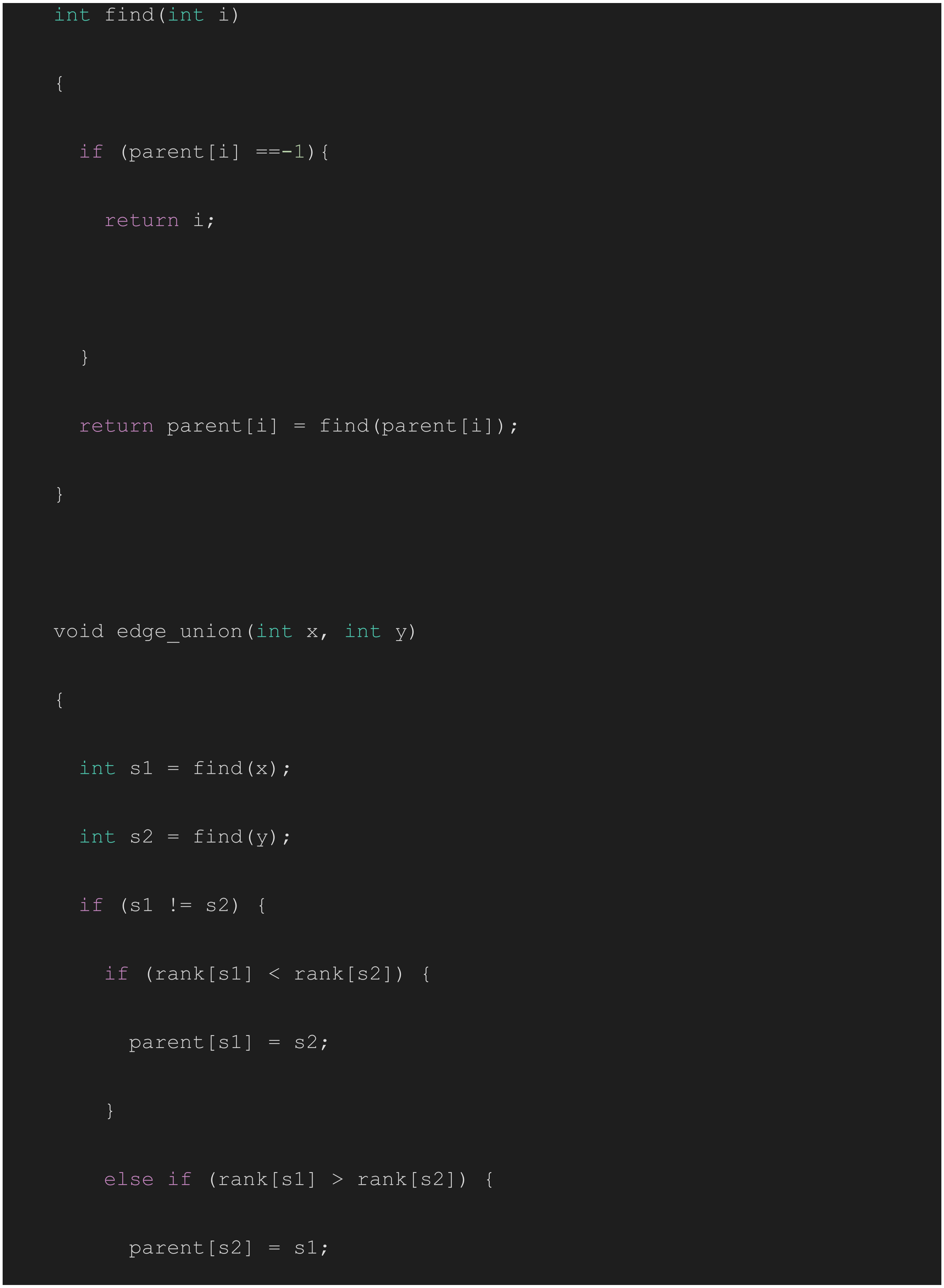


```
---------- Randomized Select----------

>> Enter the size of array : 6
>> Enter array elements : 12 34 32 5 4 27

--------------------------------
Enter the index to be found: 4

--------------------------------

>> Initial array : 12   34      32      5       4       27

>> Array after calling Random Select : 12       5       4       27      32      34

-> Element at requested index : 27
-> No. of comparisons: 5


--------------------------------
Process exited after 24.58 seconds with return value 0
Press any key to continue . . .
```

Question 3 : Write a program to determine the minimum spanning tree of a graph using Kruskal’s algorithm.

Code :

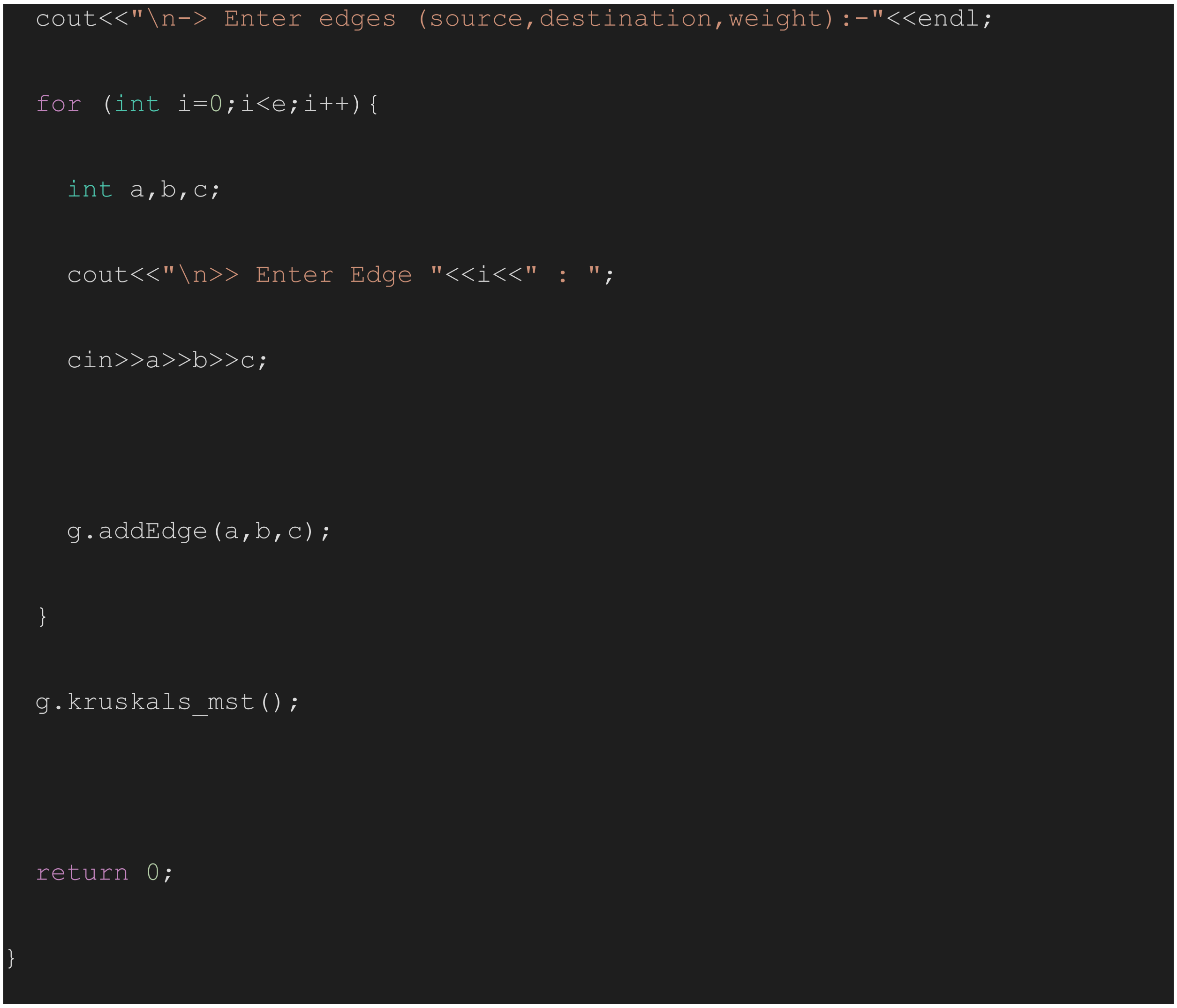
```
#include <bits/stdc++.h>

using namespace std;

class Union {

  int* parent;

  int* rank;

  public:

    Union (int n)

    {

      parent = new int[n];

      rank = new int[n];

      for (int i = 0; i < n; i++) {

        parent[i] =-1;

        rank[i] = 1;

      }

    }
```



```
int find(int i)
{
  if (parent[i] ==-1){
    return i;

  }
  return parent[i] = find(parent[i]);
}

void edge_union(int x, int y)
{
  int s1 = find(x);
  int s2 = find(y);
  if (s1 != s2) {
    if (rank[s1] < rank[s2]) {
      parent[s1] = s2;
    }
    else if (rank[s1] > rank[s2]) {
      parent[s2] = s1;
```

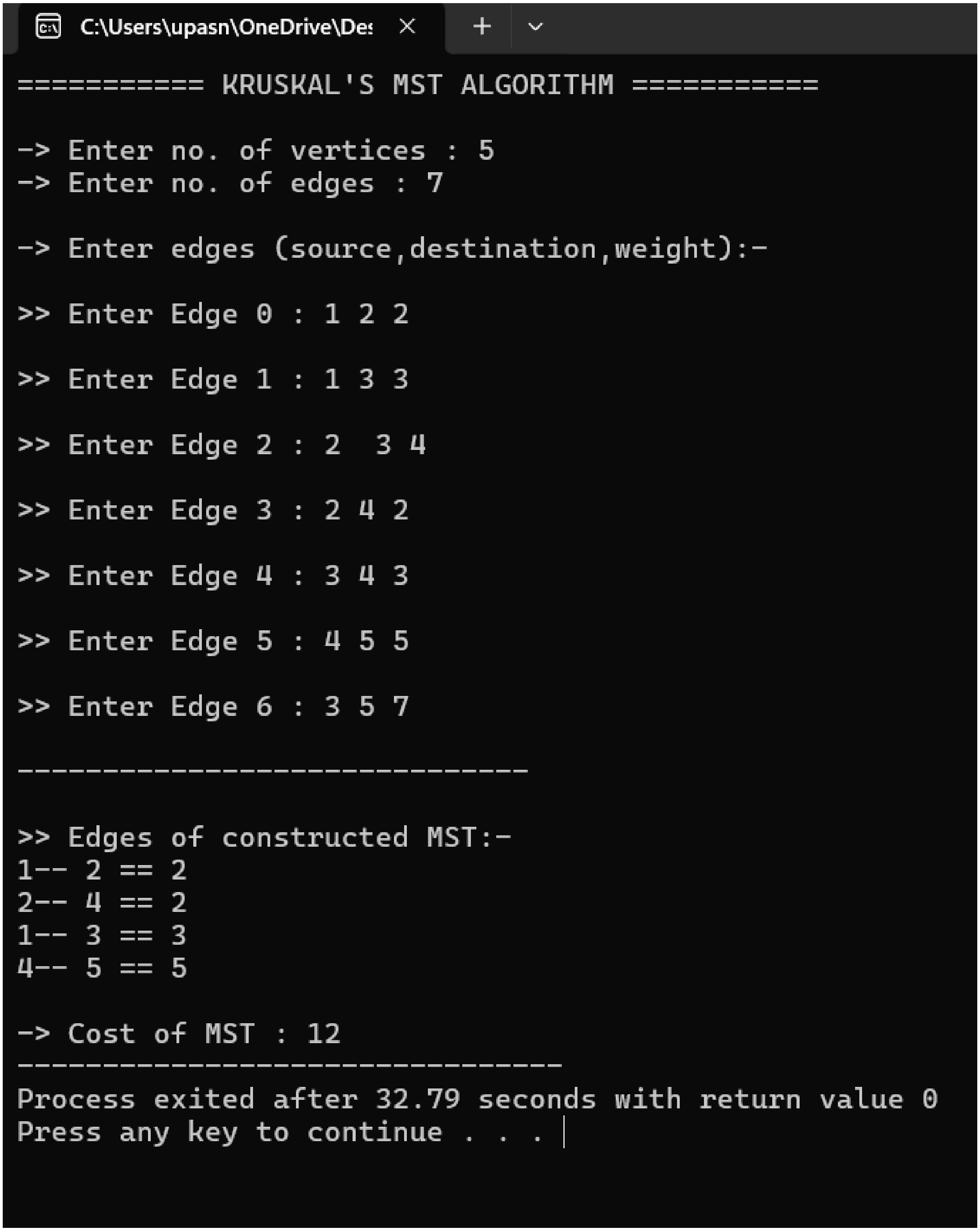
```
        }
        else {
          parent[s2] = s1;
          rank[s1] += 1;
        }
      }
    }
  };
class Graph {

  vector<vector<int> > edgelist;
  int V;

  public:
    Graph(int V) { this->V = V; }

    void addEdge(int x, int y, int w)
    {
      int arr[3] = {w, x , y};
      vector<int> vec;
      for(int i = 0; i<3; i++){
        vec.push_back(arr[i]);
      }
      edgelist.push_back(vec);
    }

    void kruskals_mst()
    {
      sort(edgelist.begin(), edgelist.end());

      Union s(V);
      int ans = 0;
      cout<<"\n-------------------------------"<<endl;
      cout<<"\n>> Edges of constructed MST:-"<< endl;
      for (int i=0; i<edgelist.size(); i++) {
        vector<int> edge = edgelist[i];
        int w = edge[0];
        int x = edge[1];
```

```
        int y = edge[2];

        if (s.find(x) != s.find(y)) {

          s.edge_union(x, y);

          ans += w;

          cout << x << "-- " << y << " == " << w<< endl;

        }

      }

      cout << "\n-> Cost of MST : " << ans;

    }

};

int main()

{

  cout<<"=========== KRUSKAL'S MST ALGORITHM =========== \n"<<endl;

  int vert,e;

  cout<<"-> Enter no. of vertices : ";

  cin>>vert;

  cout<<"-> Enter no. of edges : ";

  cin>>e;

  Graph g(vert);
```



```
  cout<<"\n-> Enter edges (source,destination,weight):-"<<endl;
  for (int i=0;i<e;i++){
    int a,b,c;
    cout<<"\n>> Enter Edge "<<i<<" : ";
    cin>>a>>b>>c;

    g.addEdge(a,b,c);
  }
  g.kruskals_mst();

  return 0;
}
```

OUTPUT:-

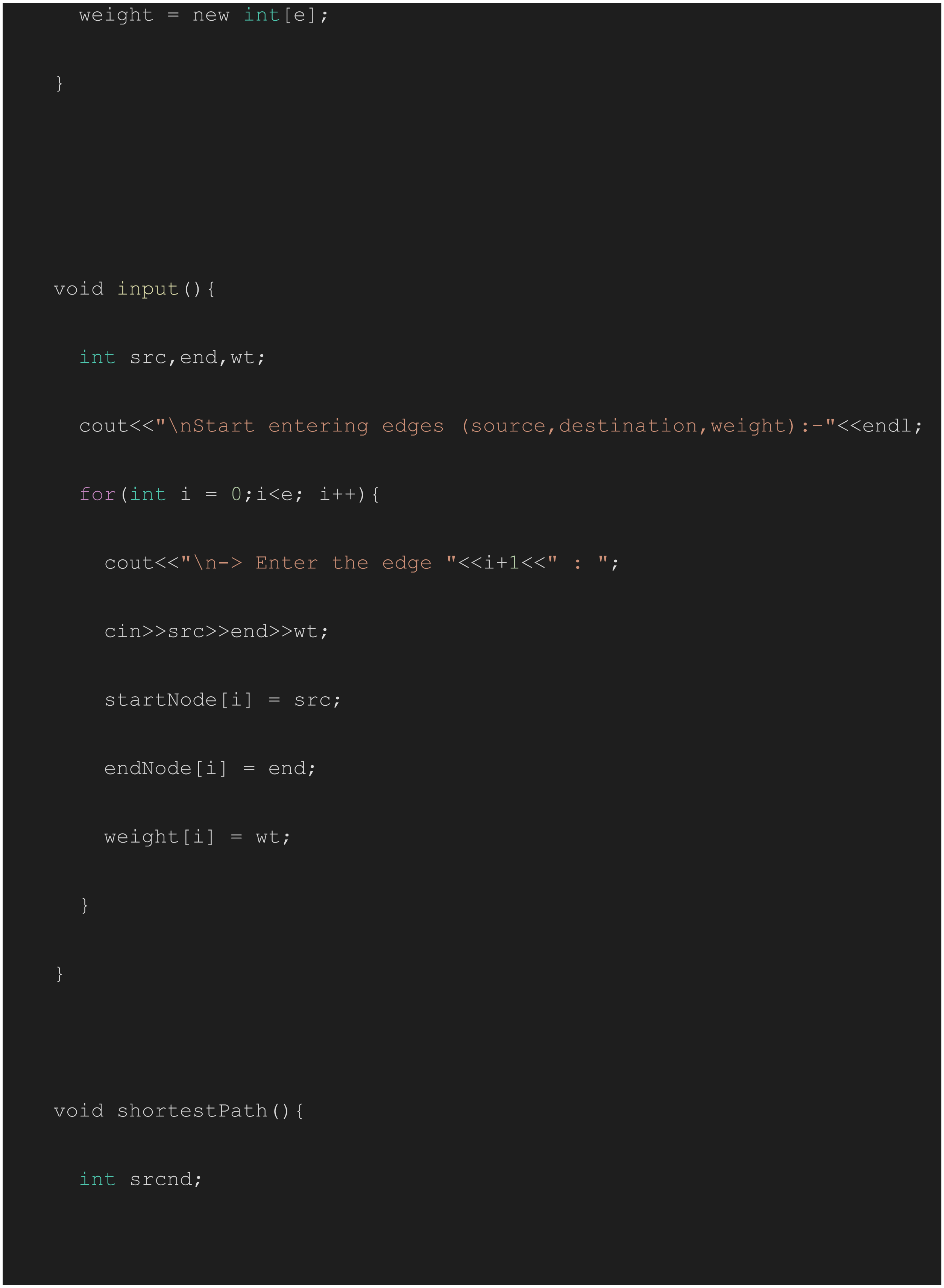


```
=========== KRUSKAL'S MST ALGORITHM ===========

-> Enter no. of vertices : 5
-> Enter no. of edges : 7

-> Enter edges (source,destination,weight):-

>> Enter Edge 0 : 1 2 2

>> Enter Edge 1 : 1 3 3

>> Enter Edge 2 : 2  3 4

>> Enter Edge 3 : 2 4 2

>> Enter Edge 4 : 3 4 3

>> Enter Edge 5 : 4 5 5

>> Enter Edge 6 : 3 5 7

------------------------------


>> Edges of constructed MST:-
1-- 2 == 2
2-- 4 == 2
1-- 3 == 3
4-- 5 == 5

-> Cost of MST : 12
--------------------------------
Process exited after 32.79 seconds with return value 0
Press any key to continue . . .
```

Question 4 : Write a program to implement the Bellman-Ford algorithm to find the shortest paths from a given source node to all other nodes in a graph.

Code :-

```
#include <iostream>
#include <climits>
using namespace std;

class Graph{
  int n;
  int e;

  int * startNode;
  int * endNode;
  int * weight;

  public:

    Graph(int node,int edges){
      n = node;
      e = edges;
      startNode = new int[e];
      endNode = new int[e];
```



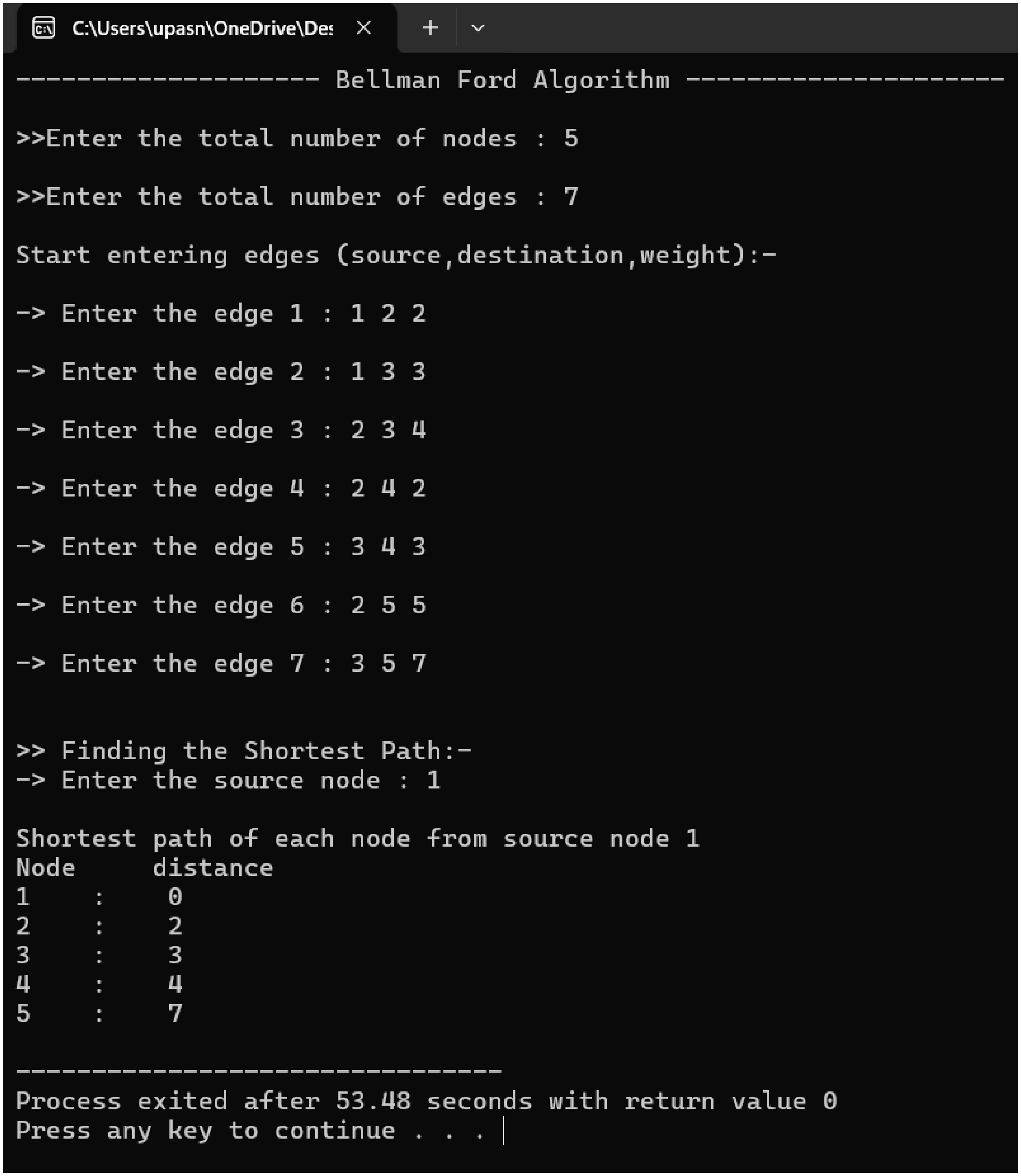
```
    weight = new int[e];

  }




  void input(){

    int src,end,wt;

    cout<<"\nStart entering edges (source,destination,weight):-"<<endl;

    for(int i = 0;i<e; i++){

      cout<<"\n-> Enter the edge "<<i+1<<" : ";

      cin>>src>>end>>wt;

      startNode[i] = src;

      endNode[i] = end;

      weight[i] = wt;

    }

  }



  void shortestPath(){

    int srcnd;
```

```
cout<<"\n\n>> Finding the Shortest Path:-";


cout<<"\n-> Enter the source node : ";

cin>>srcnd;


int dist[n+1];


for(int i = 0; i<=n ; i++){

  dist[i] = INT_MAX;

}


dist[srcnd] = 0;


for(int i = 1; i<n; i++){

  for(int j = 0; j<e; j++){

    int u = startNode[j];

    int v = endNode[j];

    int w = weight[j];
```

```
      if(dist[u]!=INT_MAX && dist[v] > dist[u] + w){
        dist[v] = dist[u] + w;
      }
    }
  }

  bool negCycle = 0;
  for(int j = 0; j<e; j++){
      int u = startNode[j];
      int v = endNode[j];
      int w = weight[j];

      if(dist[u]!=INT_MAX && dist[v] > dist[u] + w){
        negCycle = 1;
      }
  }

  if(negCycle){
    cout<<"\n--> Negative cycle exists in the graph !";
```

```
      }else{

        cout<<"\nShortest path of each node from source node
"<<srcnd<<endl;

        cout<<"Node \t distance\n";

        for(int i = 1; i<=n; i++){

          if(dist[i]==INT_MAX){

            cout<<i<<"    :    No available path"<<endl;

          }else{

            cout<<i<<"    :    "<<dist[i]<<endl;

          }

        }

      }

    }

};

int main(){

  cout<<"-------------------- Bellman Ford Algorithm
---------------------"<<endl;

  int n,e;

  cout<<"\n>>Enter the total number of nodes : ";

  cin>>n;
  cout<<"\n>>Enter the total number of edges : ";

  cin>>e;

  Graph g(n,e);

  g.input();

  g.shortestPath();

  return 0;

}
```

OUTPUT:

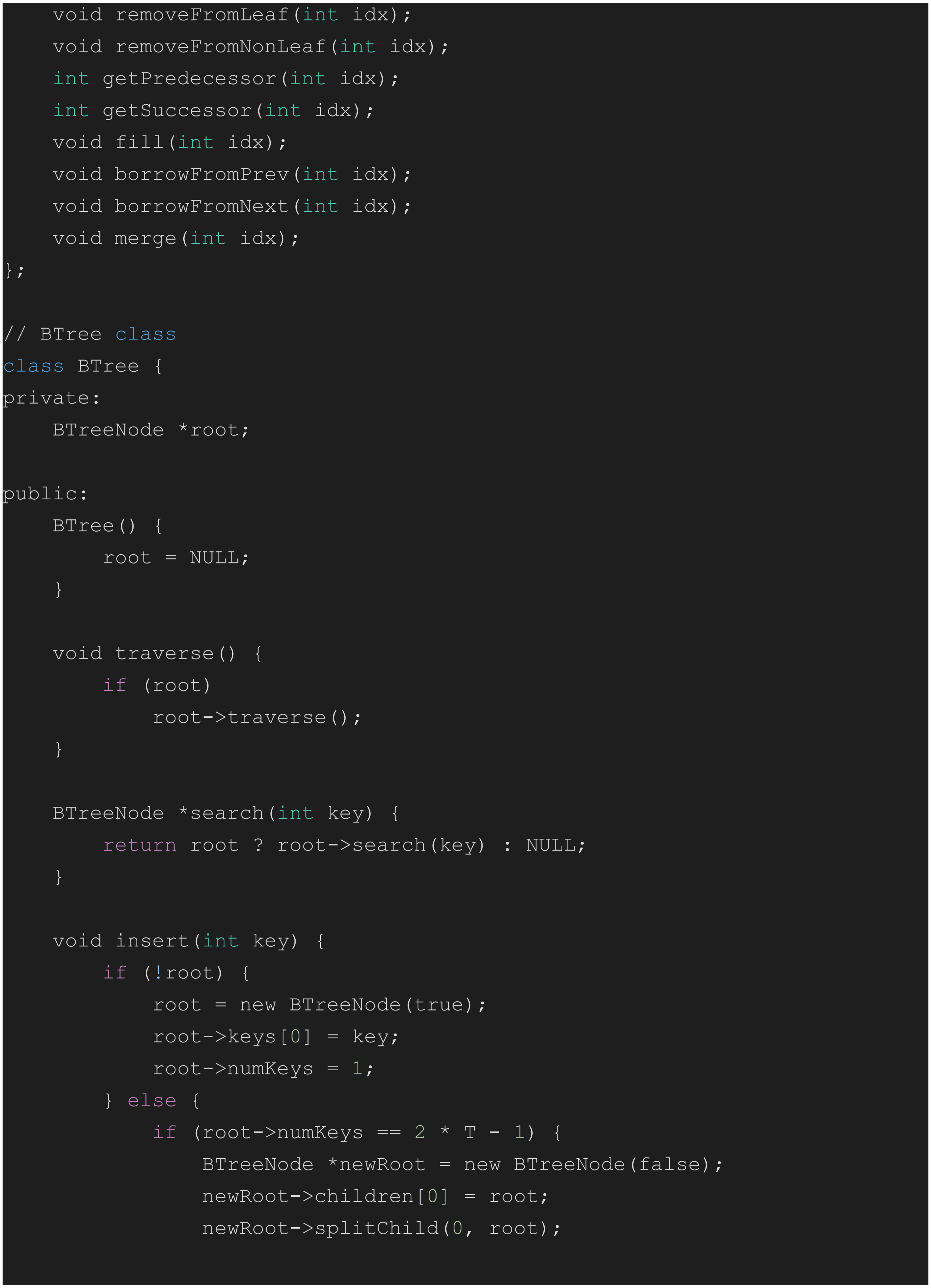


```
-------------------- Bellman Ford Algorithm ---------------------

>>Enter the total number of nodes : 5

>>Enter the total number of edges : 7

Start entering edges (source,destination,weight):-

-> Enter the edge 1 : 1 2 2

-> Enter the edge 2 : 1 3 3

-> Enter the edge 3 : 2 3 4

-> Enter the edge 4 : 2 4 2

-> Enter the edge 5 : 3 4 3

-> Enter the edge 6 : 2 5 5

-> Enter the edge 7 : 3 5 7


>> Finding the Shortest Path:-
-> Enter the source node : 1

Shortest path of each node from source node 1
Node     distance
1    :    0
2    :    2
3    :    3
4    :    4
5    :    7

--------------------------------
Process exited after 53.48 seconds with return value 0
Press any key to continue . . .
```

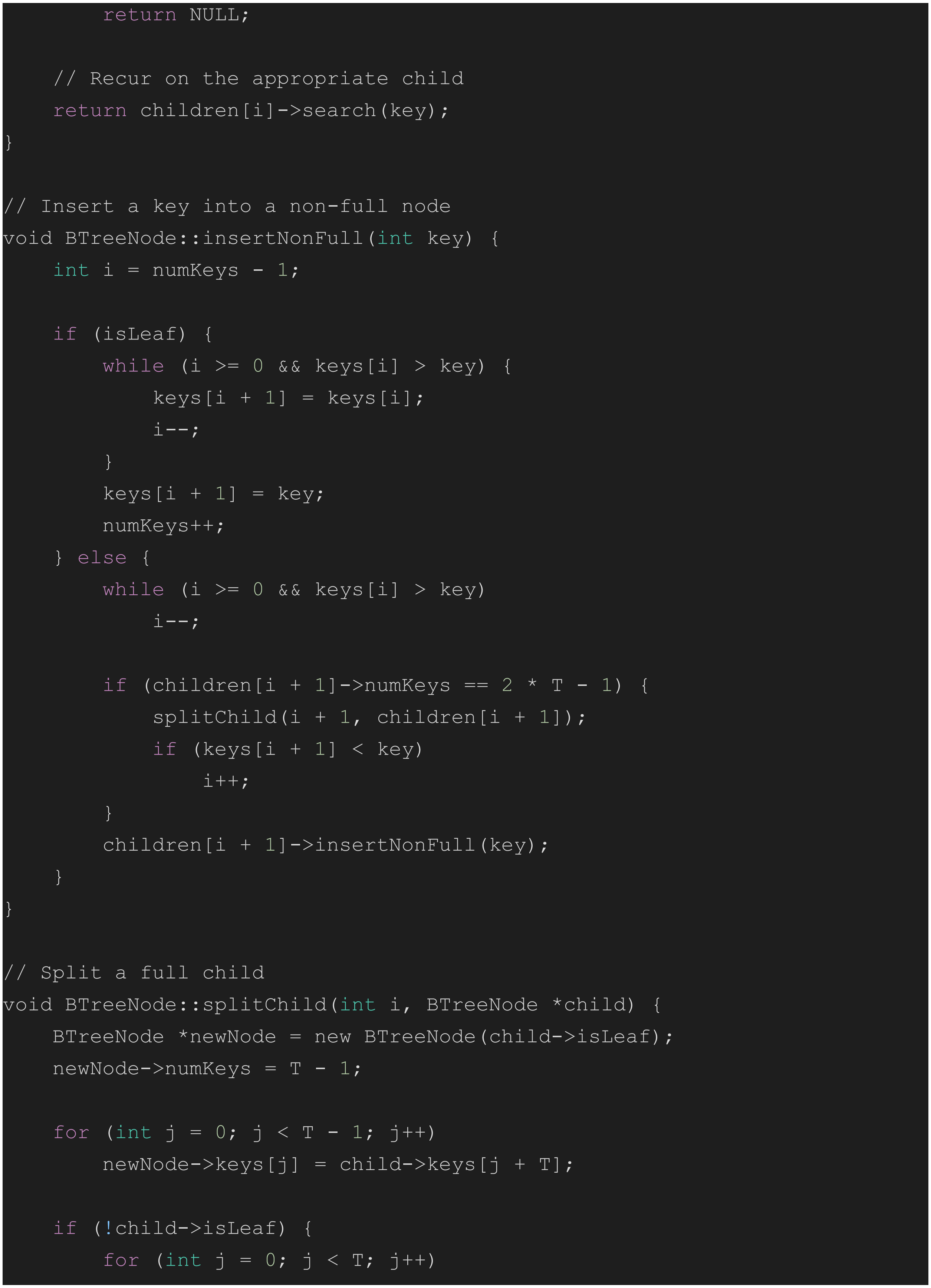
Question 5 : Write a program to implement a B-Tree.

Code:-

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

const int T = 3; // Minimum degree of B-Tree (adjustable)

// BTreeNode class
class BTreeNode {
public:
    int *keys;            // Array of keys
    BTreeNode **children; // Array of child pointers
    int numKeys;          // Current number of keys
    bool isLeaf;          // True if node is a leaf

    BTreeNode(bool leaf) {
        isLeaf = leaf;
        keys = new int[2 * T - 1];
        children = new BTreeNode *[2 * T];
        numKeys = 0;
    }

    void traverse() {
        for (int i = 0; i < numKeys; i++) {
            if (!isLeaf)
                children[i]->traverse();
            cout << keys[i] << " ";
        }
        if (!isLeaf)
            children[numKeys]->traverse();
    }

    BTreeNode *search(int key);

    void insertNonFull(int key);
    void splitChild(int i, BTreeNode *child);
    void remove(int key);
```

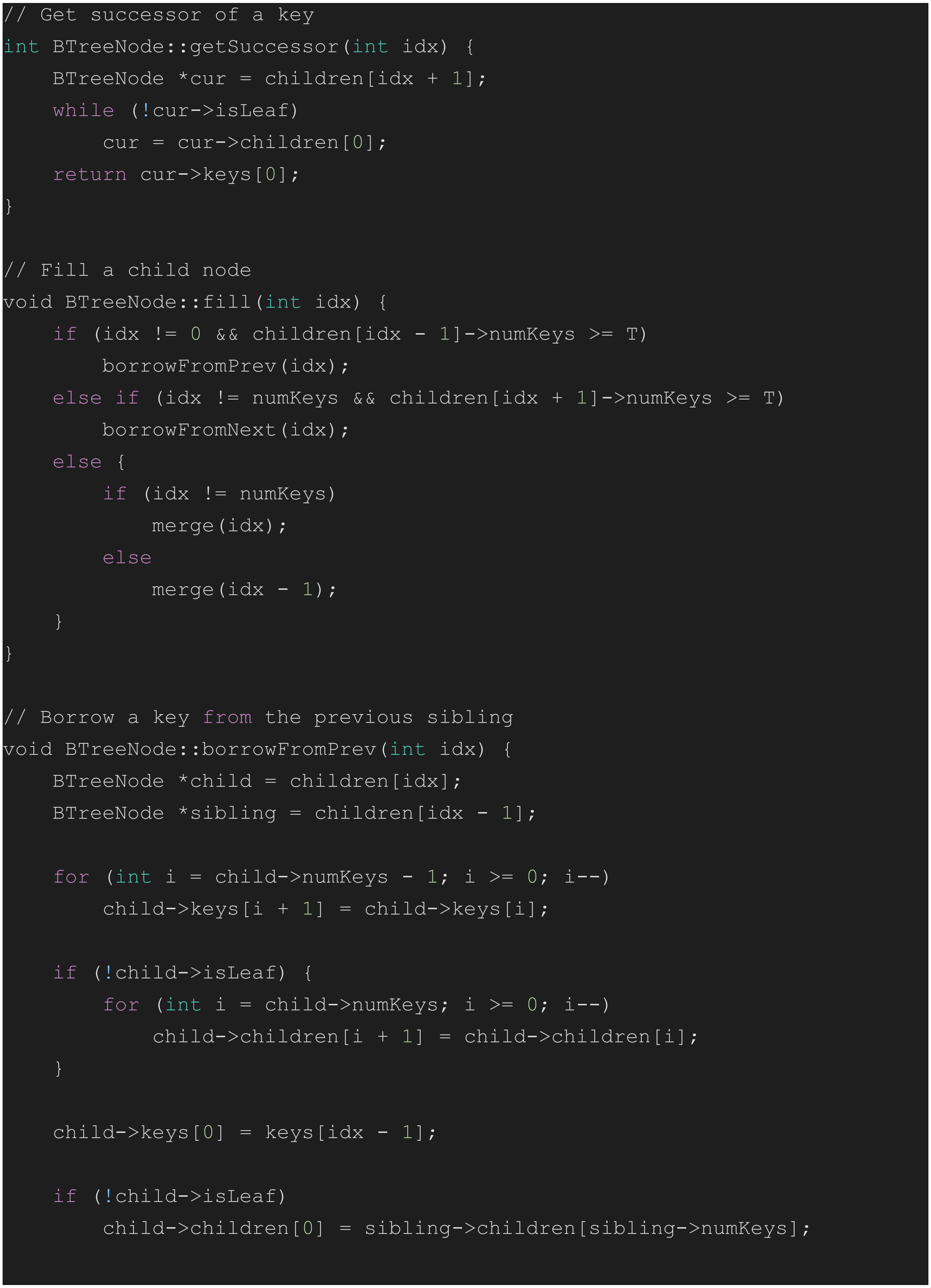


```
    void removeFromLeaf(int idx);
    void removeFromNonLeaf(int idx);
    int getPredecessor(int idx);
    int getSuccessor(int idx);
    void fill(int idx);
    void borrowFromPrev(int idx);
    void borrowFromNext(int idx);
    void merge(int idx);
};

// BTree class
class BTree {
private:
    BTreeNode *root;

public:
    BTree() {
        root = NULL;
    }

    void traverse() {
        if (root)
            root->traverse();
    }

    BTreeNode *search(int key) {
        return root ? root->search(key) : NULL;
    }

    void insert(int key) {
        if (!root) {
            root = new BTreeNode(true);
            root->keys[0] = key;
            root->numKeys = 1;
        } else {
            if (root->numKeys == 2 * T - 1) {
                BTreeNode *newRoot = new BTreeNode(false);
                newRoot->children[0] = root;
                newRoot->splitChild(0, root);
                int i = (newRoot->keys[0] < key) ? 1 : 0;
                newRoot->children[i]->insertNonFull(key);

                root = newRoot;
            } else {
                root->insertNonFull(key);
            }
        }
    }

    void remove(int key) {
        if (!root) {
            cout << "The tree is empty\n";
            return;
        }

        root->remove(key);

        if (root->numKeys == 0) {
            BTreeNode *temp = root;
            root = root->isLeaf ? NULL : root->children[0];
            delete temp;
        }
    }
};

// Search for a key in the B-Tree
BTreeNode *BTreeNode::search(int key) {
    int i = 0;

    // Find the first key greater than or equal to key
    while (i < numKeys && key > keys[i])
        i++;

    // If the key is present, return this node
    if (i < numKeys && keys[i] == key)
        return this;

    // If the node is a leaf, the key is not present
    if (isLeaf)
```

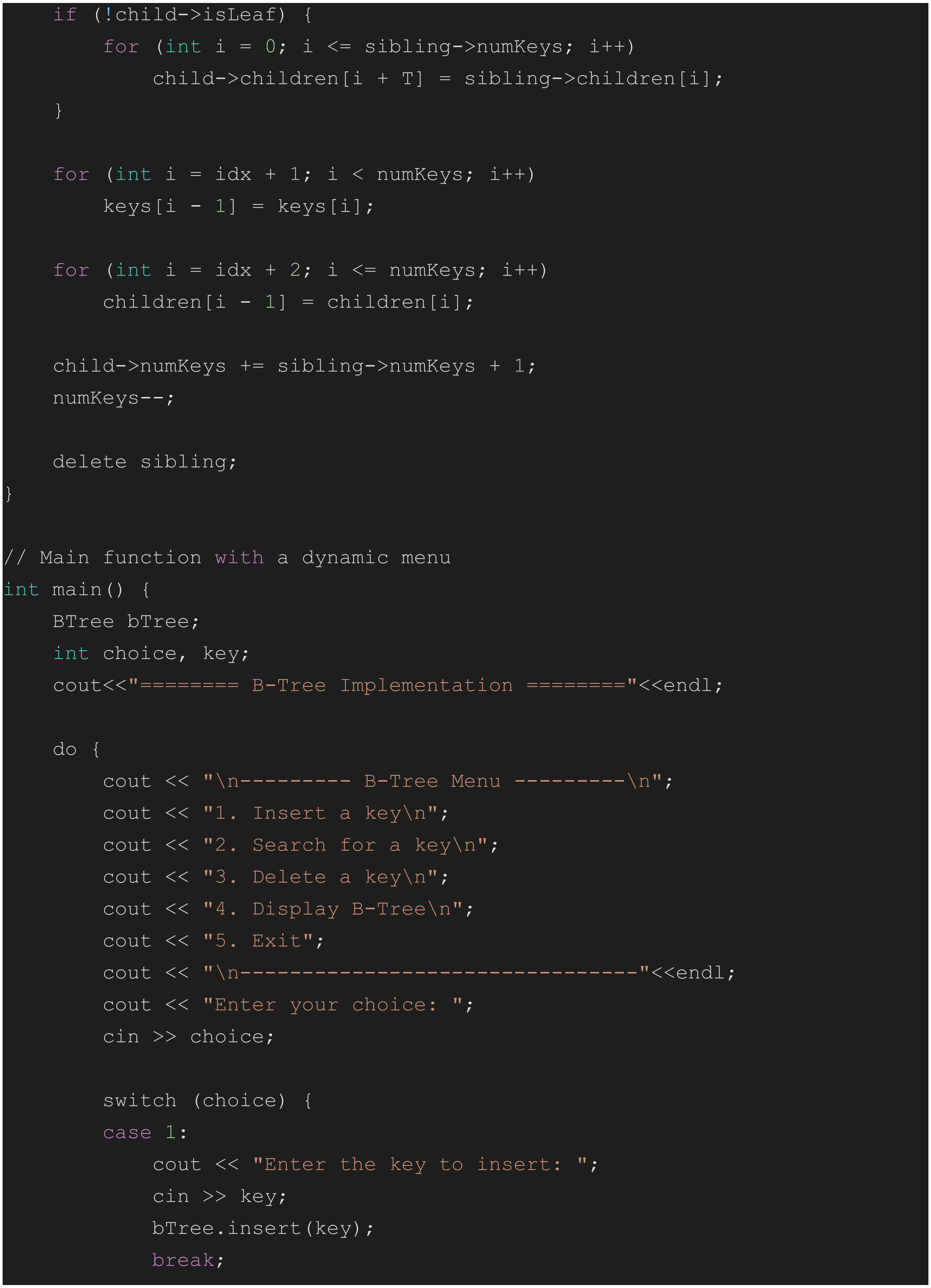


```
        return NULL;

    // Recur on the appropriate child
    return children[i]->search(key);
}

// Insert a key into a non-full node
void BTreeNode::insertNonFull(int key) {
    int i = numKeys - 1;

    if (isLeaf) {
        while (i >= 0 && keys[i] > key) {
            keys[i + 1] = keys[i];
            i--;
        }
        keys[i + 1] = key;
        numKeys++;
    } else {
        while (i >= 0 && keys[i] > key)
            i--;

        if (children[i + 1]->numKeys == 2 * T - 1) {
            splitChild(i + 1, children[i + 1]);
            if (keys[i + 1] < key)
                i++;
        }
        children[i + 1]->insertNonFull(key);
    }
}

// Split a full child
void BTreeNode::splitChild(int i, BTreeNode *child) {
    BTreeNode *newNode = new BTreeNode(child->isLeaf);
    newNode->numKeys = T - 1;

    for (int j = 0; j < T - 1; j++)
        newNode->keys[j] = child->keys[j + T];

    if (!child->isLeaf) {
        for (int j = 0; j < T; j++)
```

```
            newNode->children[j] = child->children[j + T];
    }

    child->numKeys = T - 1;

    for (int j = numKeys; j >= i + 1; j--)
        children[j + 1] = children[j];

    children[i + 1] = newNode;

    for (int j = numKeys - 1; j >= i; j--)
        keys[j + 1] = keys[j];

    keys[i] = child->keys[T - 1];
    numKeys++;
}

// Remove a key from the node
void BTreeNode::remove(int key) {
    int idx = 0;
    while (idx < numKeys && keys[idx] < key)
        idx++;

    if (idx < numKeys && keys[idx] == key) {
        if (isLeaf)
            removeFromLeaf(idx);
        else
            removeFromNonLeaf(idx);
        cout<<"\n>> Deleted sucessfully !!"<<endl;
    } else {
        if (isLeaf) {
            cout << "\n>> The key " << key << " is not in the tree\n";
            return;
        }

        bool flag = (idx == numKeys);

        if (children[idx]->numKeys < T)
            fill(idx);
        if (flag && idx > numKeys)
            children[idx - 1]->remove(key);
        else
            children[idx]->remove(key);
    }
}

// Remove a key from a leaf node
void BTreeNode::removeFromLeaf(int idx) {
    for (int i = idx + 1; i < numKeys; i++)
        keys[i - 1] = keys[i];
    numKeys--;
}

// Remove a key from a non-leaf node
void BTreeNode::removeFromNonLeaf(int idx) {
    int key = keys[idx];

    if (children[idx]->numKeys >= T) {
        int pred = getPredecessor(idx);
        keys[idx] = pred;
        children[idx]->remove(pred);
    } else if (children[idx + 1]->numKeys >= T) {
        int succ = getSuccessor(idx);
        keys[idx] = succ;
        children[idx + 1]->remove(succ);
    } else {
        merge(idx);
        children[idx]->remove(key);
    }
}

// Get predecessor of a key
int BTreeNode::getPredecessor(int idx) {
    BTreeNode *cur = children[idx];
    while (!cur->isLeaf)
        cur = cur->children[cur->numKeys];
    return cur->keys[cur->numKeys - 1];
}
```

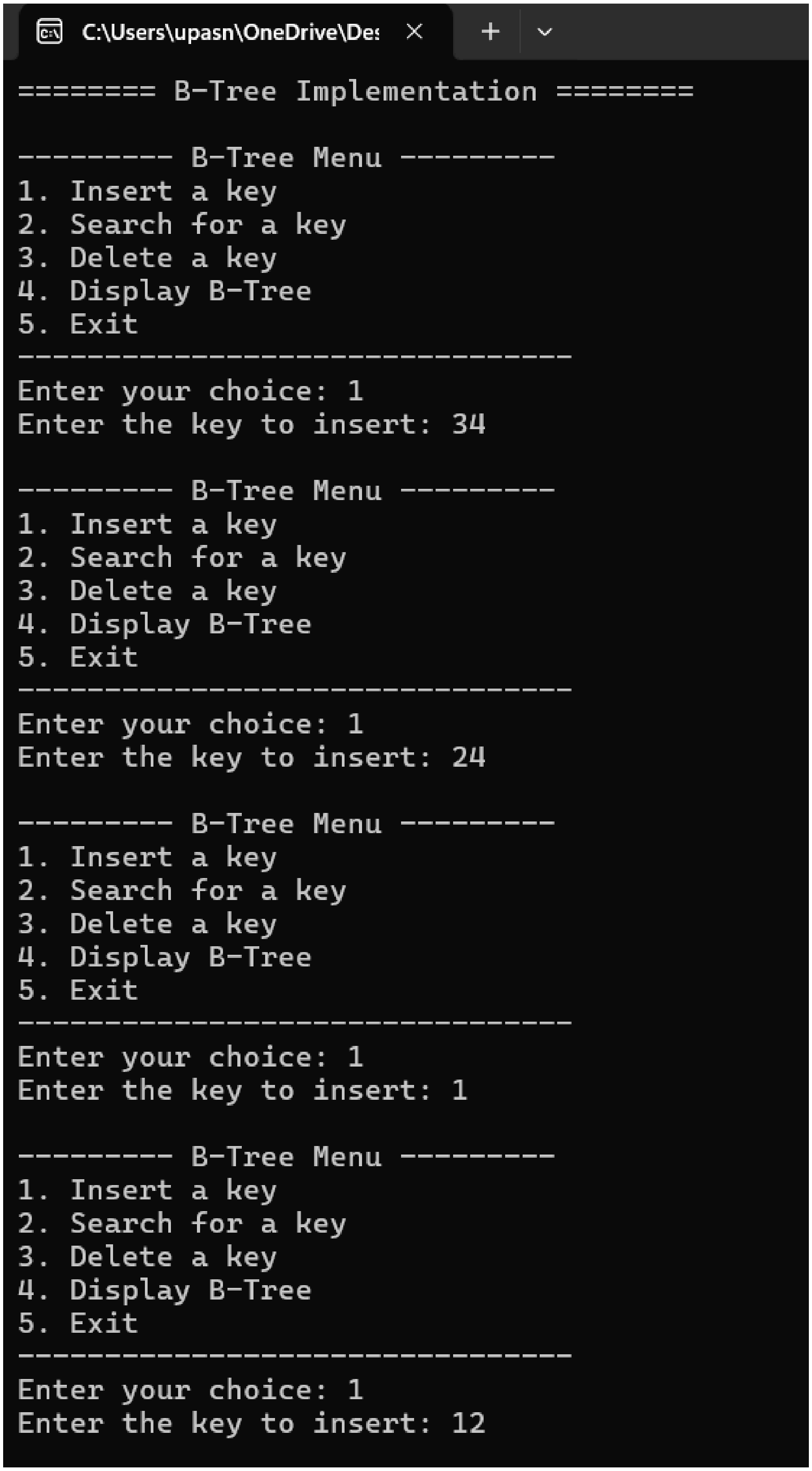


```cpp
// Get successor of a key
int BTreeNode::getSuccessor(int idx) {
    BTreeNode *cur = children[idx + 1];
    while (!cur->isLeaf)
        cur = cur->children[0];
    return cur->keys[0];
}

// Fill a child node
void BTreeNode::fill(int idx) {
    if (idx != 0 && children[idx - 1]->numKeys >= T)
        borrowFromPrev(idx);
    else if (idx != numKeys && children[idx + 1]->numKeys >= T)
        borrowFromNext(idx);
    else {
        if (idx != numKeys)
            merge(idx);
        else
            merge(idx - 1);
    }
}

// Borrow a key from the previous sibling
void BTreeNode::borrowFromPrev(int idx) {
    BTreeNode *child = children[idx];
    BTreeNode *sibling = children[idx - 1];

    for (int i = child->numKeys - 1; i >= 0; i--)
        child->keys[i + 1] = child->keys[i];

    if (!child->isLeaf) {
        for (int i = child->numKeys; i >= 0; i--)
            child->children[i + 1] = child->children[i];
    }

    child->keys[0] = keys[idx - 1];

    if (!child->isLeaf)
        child->children[0] = sibling->children[sibling->numKeys];
    keys[idx - 1] = sibling->keys[sibling->numKeys - 1];

    child->numKeys++;
    sibling->numKeys--;
}

// Borrow a key from the next sibling
void BTreeNode::borrowFromNext(int idx) {
    BTreeNode *child = children[idx];
    BTreeNode *sibling = children[idx + 1];

    child->keys[child->numKeys] = keys[idx];

    if (!child->isLeaf)
        child->children[child->numKeys + 1] = sibling->children[0];

    keys[idx] = sibling->keys[0];

    for (int i = 1; i < sibling->numKeys; i++)
        sibling->keys[i - 1] = sibling->keys[i];

    if (!sibling->isLeaf) {
        for (int i = 1; i <= sibling->numKeys; i++)
            sibling->children[i - 1] = sibling->children[i];
    }

    child->numKeys++;
    sibling->numKeys--;
}

// Merge two children
void BTreeNode::merge(int idx) {
    BTreeNode *child = children[idx];
    BTreeNode *sibling = children[idx + 1];

    child->keys[T - 1] = keys[idx];

    for (int i = 0; i < sibling->numKeys; i++)
        child->keys[i + T] = sibling->keys[i];
```

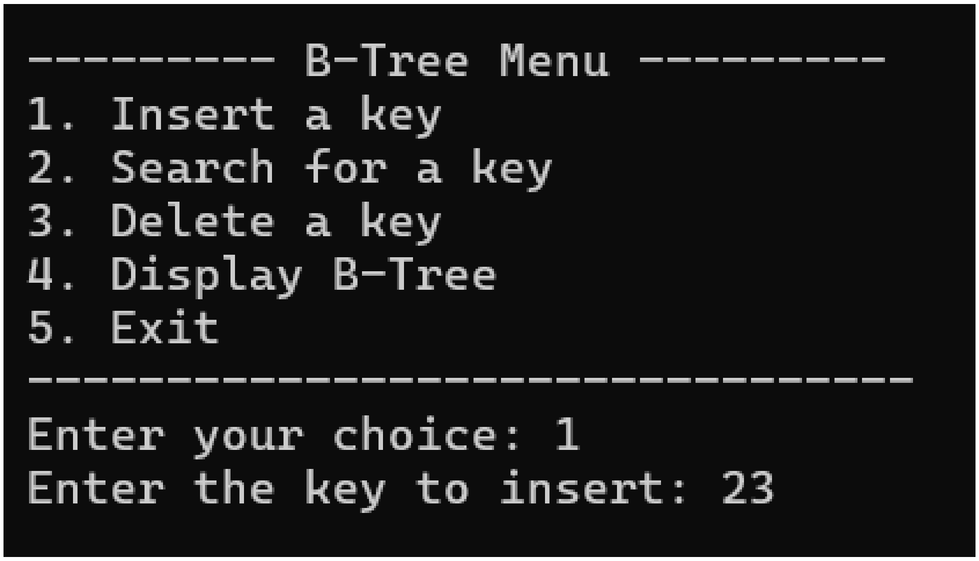


```
    if (!child->isLeaf) {
        for (int i = 0; i <= sibling->numKeys; i++)
            child->children[i + T] = sibling->children[i];
    }

    for (int i = idx + 1; i < numKeys; i++)
        keys[i - 1] = keys[i];

    for (int i = idx + 2; i <= numKeys; i++)
        children[i - 1] = children[i];

    child->numKeys += sibling->numKeys + 1;
    numKeys--;

    delete sibling;
}

// Main function with a dynamic menu
int main() {
    BTree bTree;
    int choice, key;
    cout<<"======== B-Tree Implementation ========"<<endl;

    do {
        cout << "\n--------- B-Tree Menu ---------\n";
        cout << "1. Insert a key\n";
        cout << "2. Search for a key\n";
        cout << "3. Delete a key\n";
        cout << "4. Display B-Tree\n";
        cout << "5. Exit";
        cout << "\n---------------------------------"<<endl;
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
        case 1:
            cout << "Enter the key to insert: ";
            cin >> key;
            bTree.insert(key);
            break;
        case 2:
            cout << "Enter the key to search: ";
            cin >> key;
            if (bTree.search(key))
                cout << "\n>> Key " << key << " is found in the 
B-Tree.\n";
            else
                cout << "\n>>Key " << key << " is not found in the 
B-Tree.\n";
            break;

        case 3:
            cout << "Enter the key to delete: ";
            cin >> key;
            bTree.remove(key);
            break;

        case 4:
            cout << "\n-> Keys in B-Tree: ";
            bTree.traverse();
            cout << endl;
            break;

        case 5:
            cout << "Exiting...\n";
            break;

        default:
            cout << ">> Invalid choice. Try again.\n";
        }
    } while (choice != 5);

    return 0;
}
```

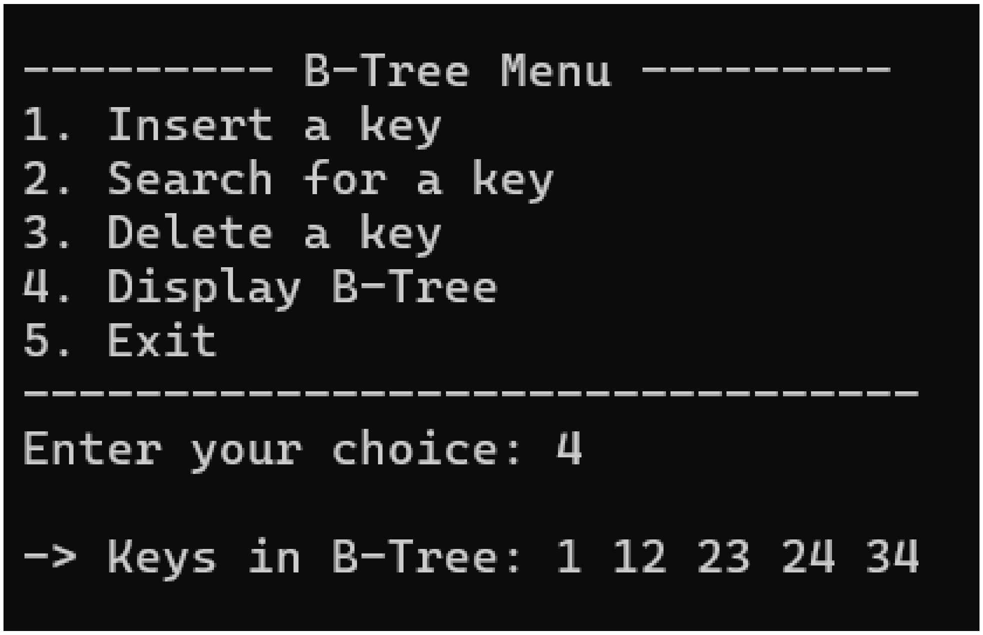
OUTPUT:



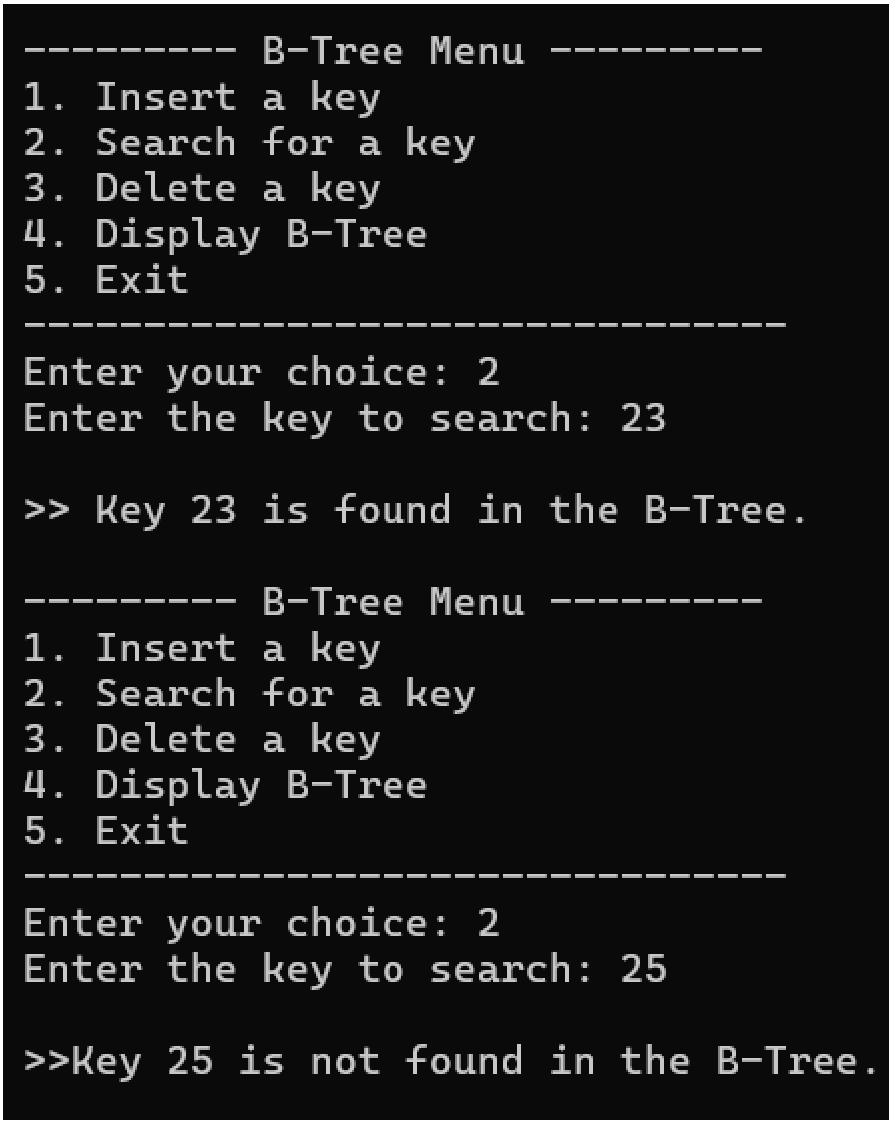
```
======== B-Tree Implementation ========

--------- B-Tree Menu ---------
1. Insert a key
2. Search for a key
3. Delete a key
4. Display B-Tree
5. Exit
--------------------------------
Enter your choice: 1
Enter the key to insert: 34

--------- B-Tree Menu ---------
1. Insert a key
2. Search for a key
3. Delete a key
4. Display B-Tree
5. Exit
--------------------------------
Enter your choice: 1
Enter the key to insert: 24

--------- B-Tree Menu ---------
1. Insert a key
2. Search for a key
3. Delete a key
4. Display B-Tree
5. Exit
--------------------------------
Enter your choice: 1
Enter the key to insert: 1

--------- B-Tree Menu ---------
1. Insert a key
2. Search for a key
3. Delete a key
4. Display B-Tree
5. Exit
--------------------------------
Enter your choice: 1
Enter the key to insert: 12
```



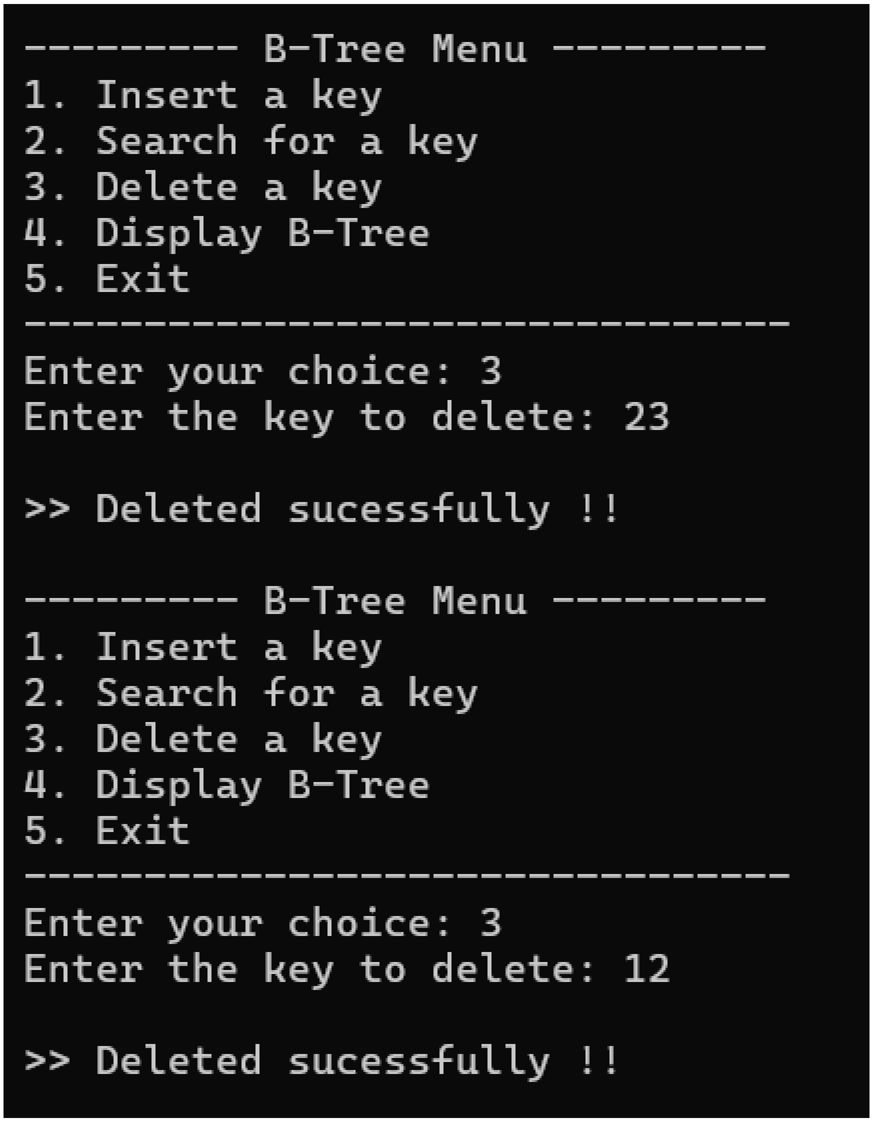
```
--------- B-Tree Menu ---------
1. Insert a key
2. Search for a key
3. Delete a key
4. Display B-Tree
5. Exit
--------------------------------
Enter your choice: 1
Enter the key to insert: 23
```



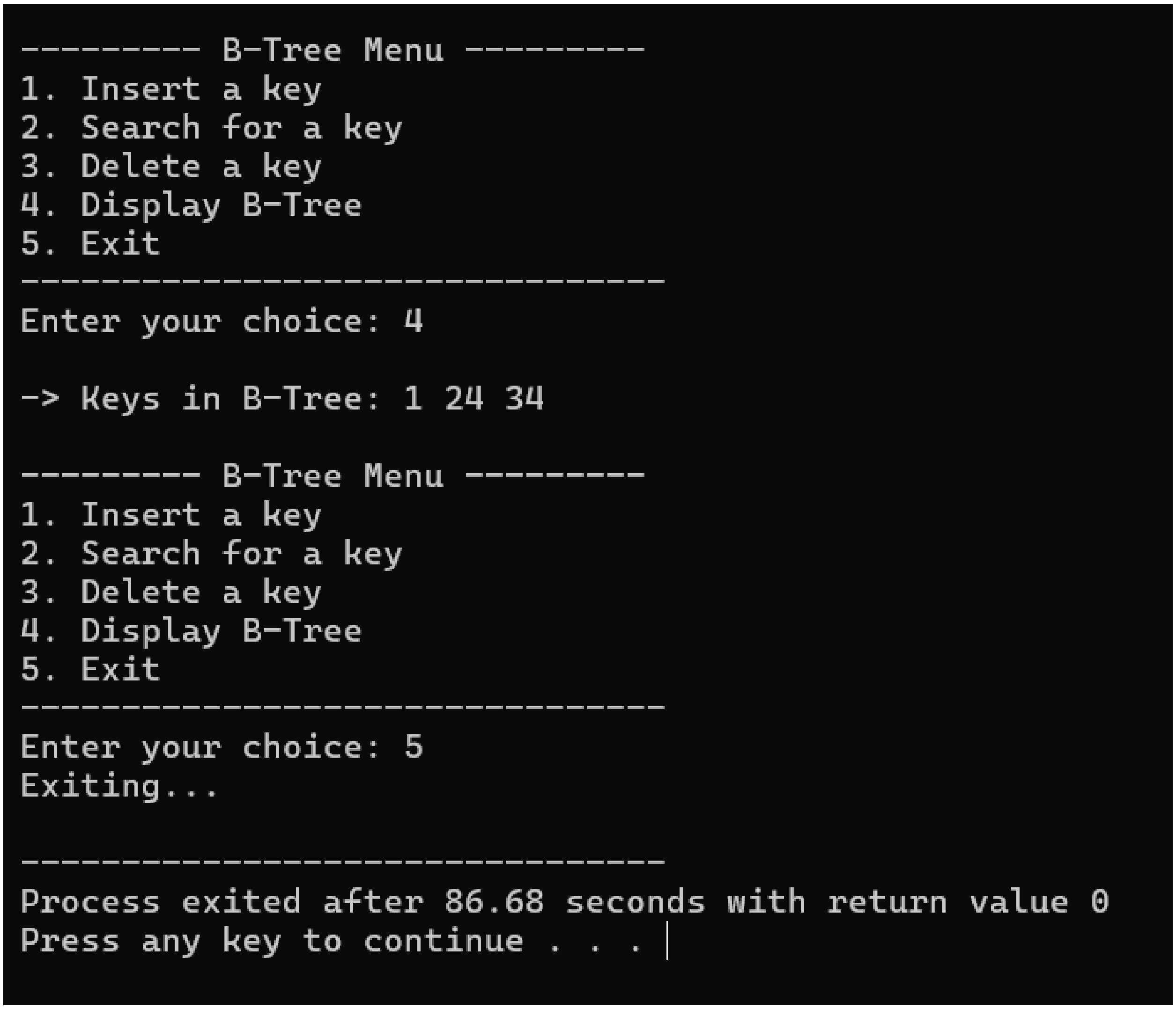
```
--------- B-Tree Menu ---------
1. Insert a key
2. Search for a key
3. Delete a key
4. Display B-Tree
5. Exit
--------------------------------
Enter your choice: 4

-> Keys in B-Tree: 1 12 23 24 34
```



```
--------- B-Tree Menu ---------
1. Insert a key
2. Search for a key
3. Delete a key
4. Display B-Tree
5. Exit
--------------------------------
Enter your choice: 2
Enter the key to search: 23

>> Key 23 is found in the B-Tree.

--------- B-Tree Menu ---------
1. Insert a key
2. Search for a key
3. Delete a key
4. Display B-Tree
5. Exit
--------------------------------
Enter your choice: 2
Enter the key to search: 25

>>Key 25 is not found in the B-Tree.
```



```
--------- B-Tree Menu ---------
1. Insert a key
2. Search for a key
3. Delete a key
4. Display B-Tree
5. Exit
--------------------------------
Enter your choice: 3
Enter the key to delete: 23

>> Deleted sucessfully !!

--------- B-Tree Menu ---------
1. Insert a key
2. Search for a key
3. Delete a key
4. Display B-Tree
5. Exit
--------------------------------
Enter your choice: 3
Enter the key to delete: 12

>> Deleted sucessfully !!
```

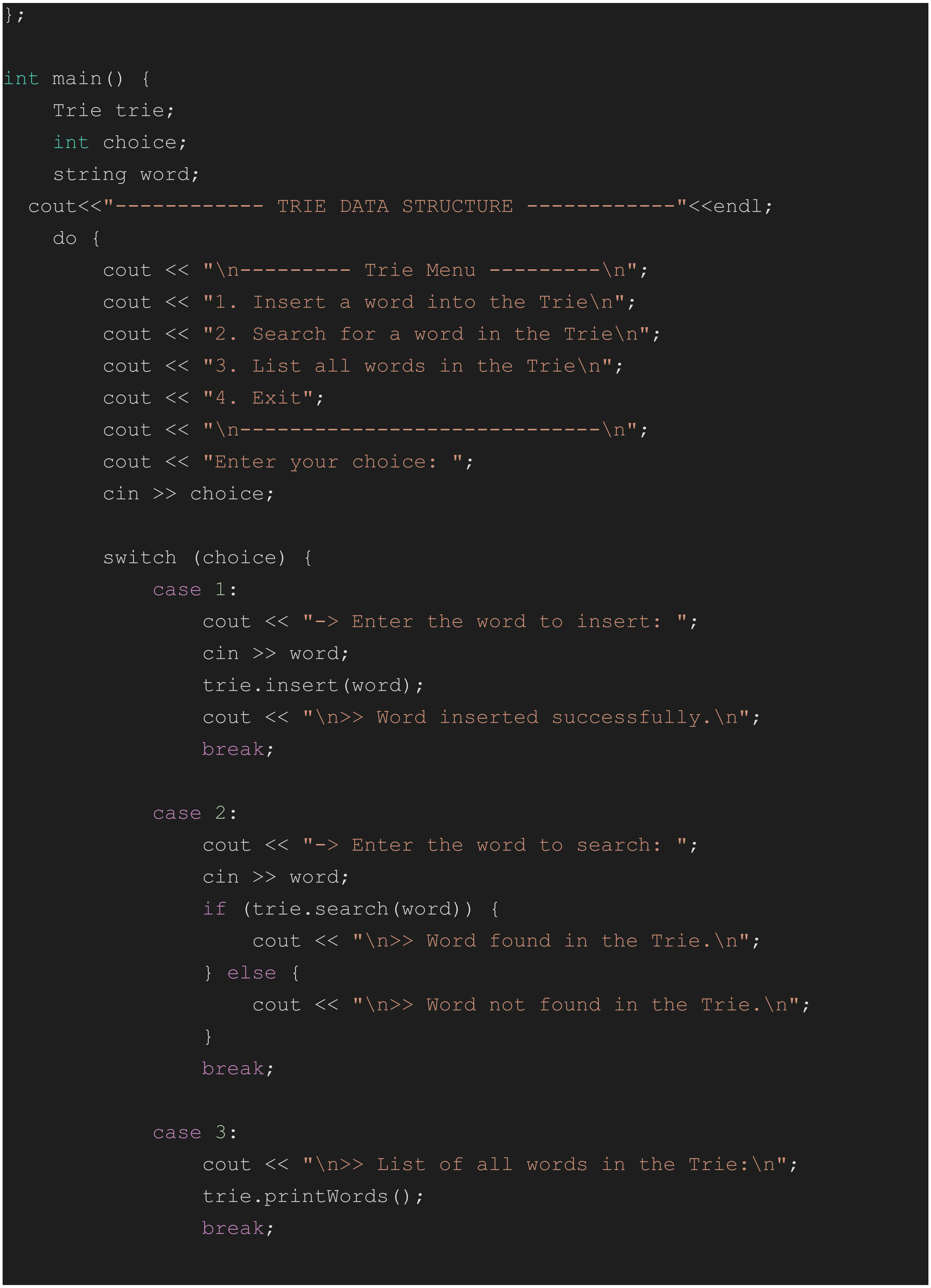


```
--------- B-Tree Menu ---------
1. Insert a key
2. Search for a key
3. Delete a key
4. Display B-Tree
5. Exit
--------------------------------
Enter your choice: 4

-> Keys in B-Tree: 1 24 34

--------- B-Tree Menu ---------
1. Insert a key
2. Search for a key
3. Delete a key
4. Display B-Tree
5. Exit
--------------------------------
Enter your choice: 5
Exiting...

--------------------------------
Process exited after 86.68 seconds with return value 0
Press any key to continue . . .
```

_Question 6 : Write a program to implement the Trie Data Structure, which supports the following operations :
a) Insert
b) Search

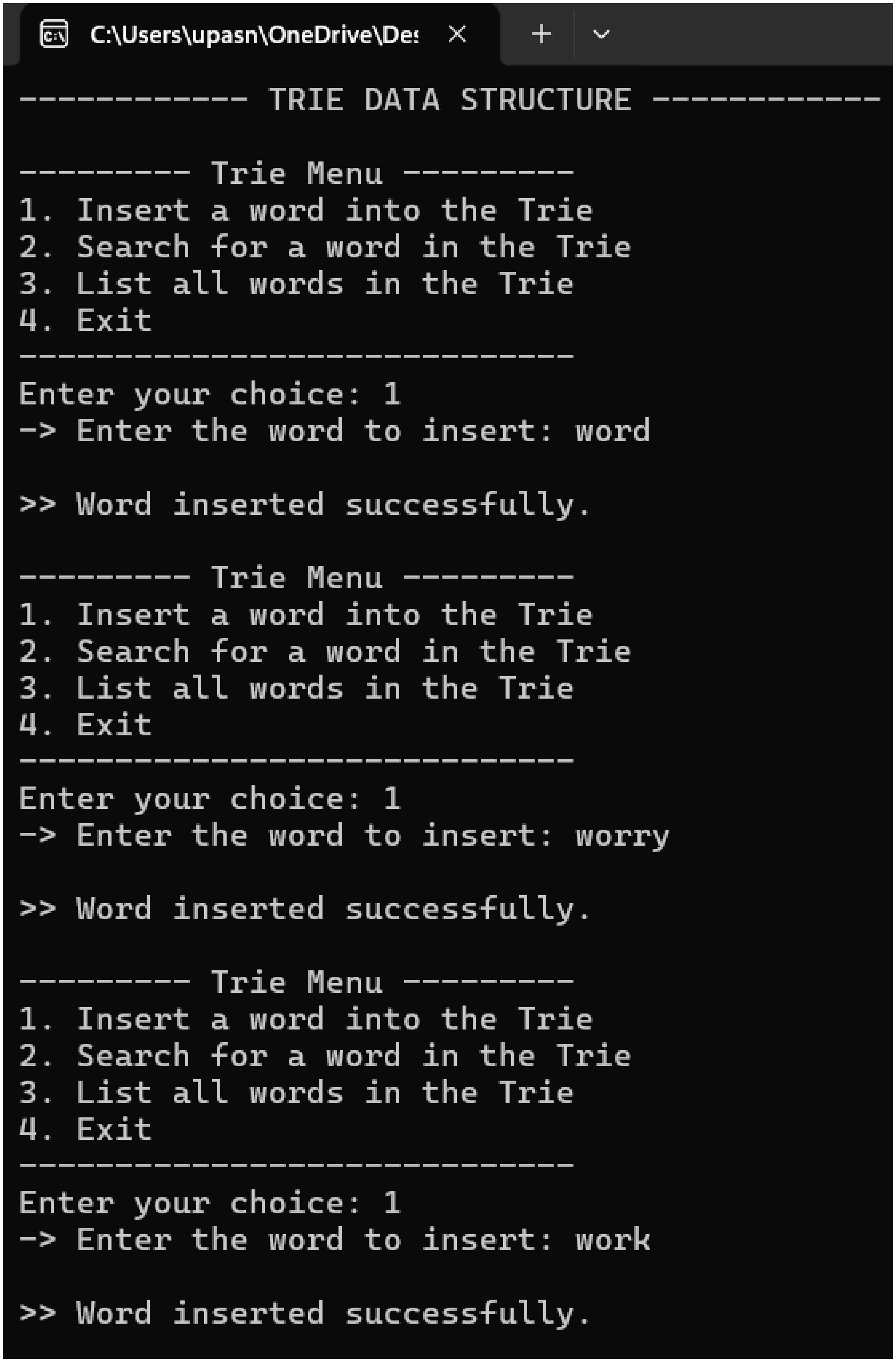
Code :-

```
#include <iostream>
#include <string>

using namespace std;

struct TrieNode {
    TrieNode* children[26];
    bool isEndOfWord;

    TrieNode() {
        isEndOfWord = false;
        for (int i = 0; i < 26; i++) {
            children[i] = NULL;
        }
    }
};

class Trie {
private:
    TrieNode* root;

    void printWordsHelper(TrieNode* node, string currentWord, bool&
isEmpty) {
        if (node->isEndOfWord) {
            cout << currentWord << endl;
            isEmpty = false;
        }
        for (int i = 0; i < 26; i++) {
            if (node->children[i] != NULL) {
                printWordsHelper(node->children[i], currentWord + char(i +
'a'), isEmpty);
            }
        }
    }

public:
    Trie() {
        root = new TrieNode();
    }

    void insert(const string& word) {
        TrieNode* current = root;
        for (size_t i = 0; i < word.length(); i++) {
            int index = word[i] - 'a';
            if (current->children[index] == NULL) {
                current->children[index] = new TrieNode();
            }
            current = current->children[index];
        }
        current->isEndOfWord = true;
    }

    bool search(const string& word) {
        TrieNode* current = root;
        for (size_t i = 0; i < word.length(); i++) {
            int index = word[i] - 'a';
            if (current->children[index] == NULL) {
                return false;
            }
            current = current->children[index];
        }
        return current->isEndOfWord;
    }

    void printWords() {
        bool isEmpty = true;
        printWordsHelper(root, "", isEmpty);

        if (isEmpty) {
            cout << "\n>> The Trie is empty." << endl;
        }
    }
```

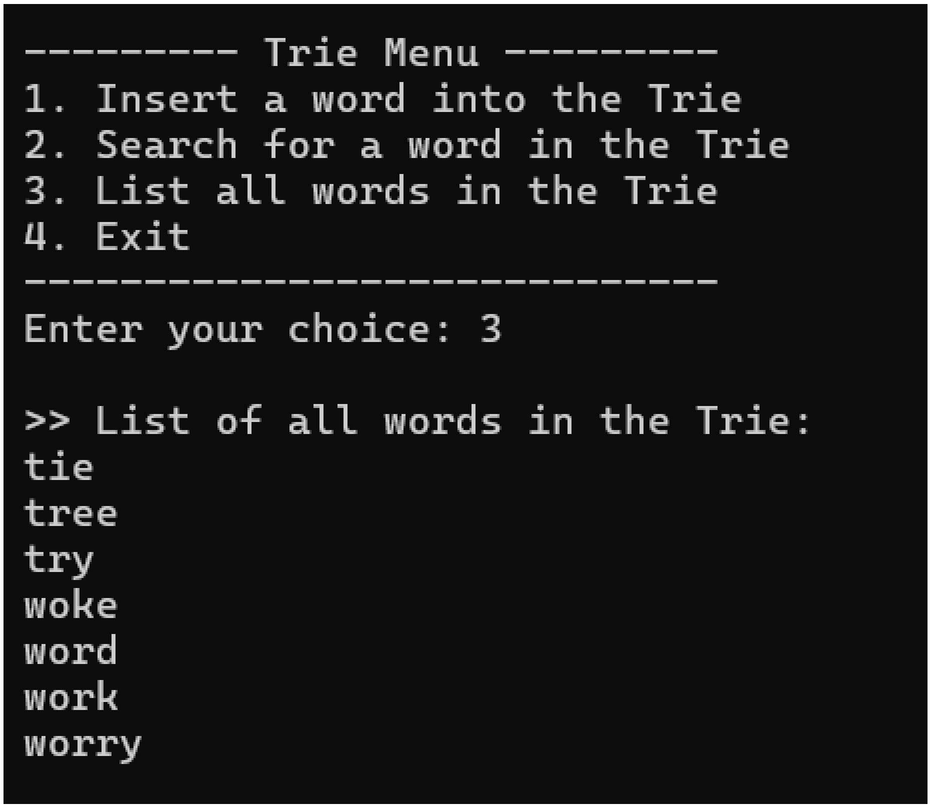


```
};

int main() {
    Trie trie;
    int choice;
    string word;
  cout<<"------------ TRIE DATA STRUCTURE ------------"<<endl;
    do {
        cout << "\n--------- Trie Menu ---------\n";
        cout << "1. Insert a word into the Trie\n";
        cout << "2. Search for a word in the Trie\n";
        cout << "3. List all words in the Trie\n";
        cout << "4. Exit";
        cout << "\n-----------------------------\n";
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                cout << "-> Enter the word to insert: ";
                cin >> word;
                trie.insert(word);
                cout << "\n>> Word inserted successfully.\n";
                break;

            case 2:
                cout << "-> Enter the word to search: ";
                cin >> word;
                if (trie.search(word)) {
                    cout << "\n>> Word found in the Trie.\n";
                } else {
                    cout << "\n>> Word not found in the Trie.\n";
                }
                break;

            case 3:
                cout << "\n>> List of all words in the Trie:\n";
                trie.printWords();
                break;
            case 4:
                cout << "Exiting the program.\n";
                break;

            default:
                cout << "\n>> Invalid choice. Please try again.\n";
                break;
        }
    } while (choice != 4);

    return 0;
}
```

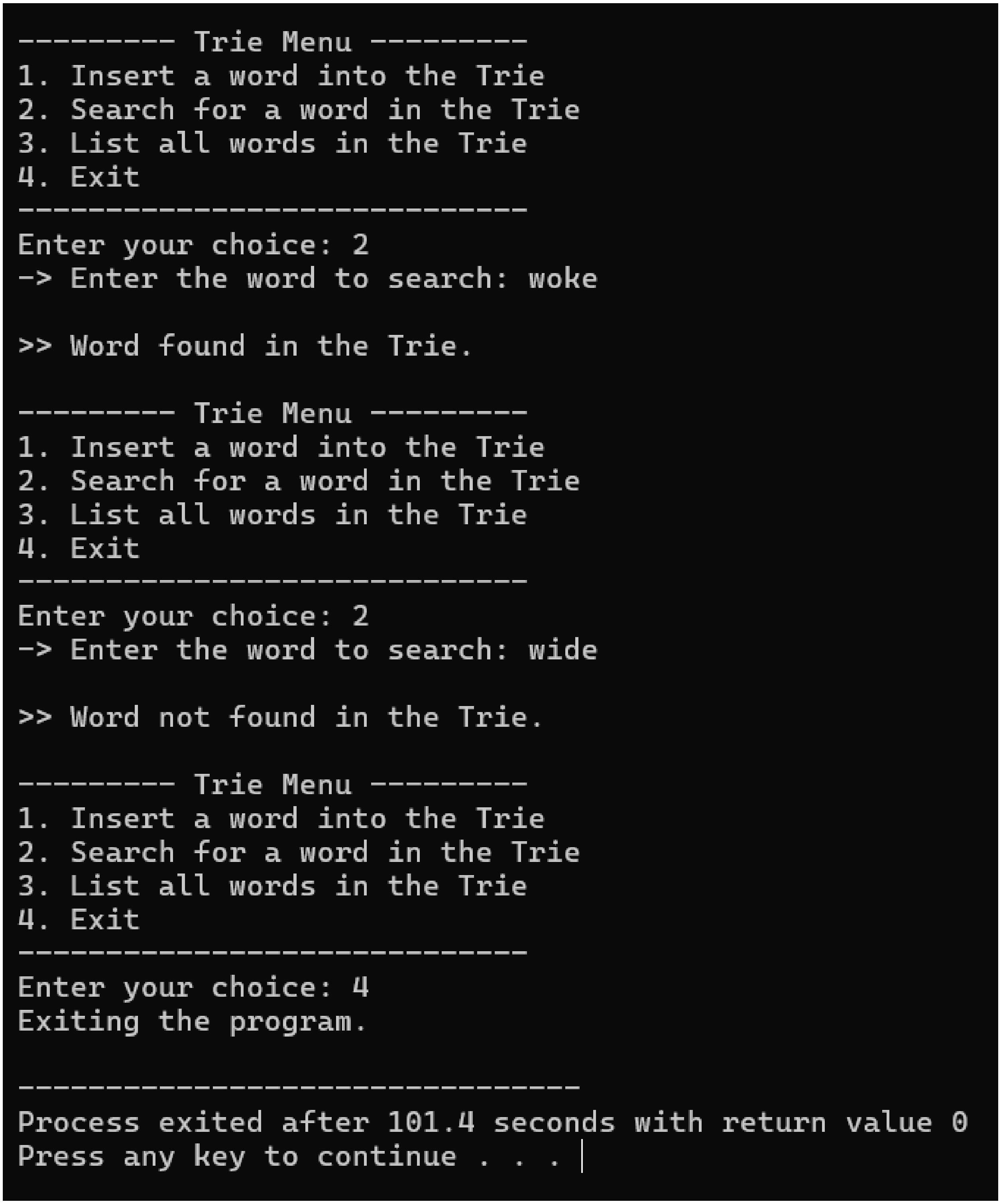
OUTPUT:



```
------------ TRIE DATA STRUCTURE ------------

--------- Trie Menu ---------
1. Insert a word into the Trie
2. Search for a word in the Trie
3. List all words in the Trie
4. Exit
-----------------------------
Enter your choice: 1
-> Enter the word to insert: word

>> Word inserted successfully.

--------- Trie Menu ---------
1. Insert a word into the Trie
2. Search for a word in the Trie
3. List all words in the Trie
4. Exit
-----------------------------
Enter your choice: 1
-> Enter the word to insert: worry

>> Word inserted successfully.

--------- Trie Menu ---------
1. Insert a word into the Trie
2. Search for a word in the Trie
3. List all words in the Trie
4. Exit
-----------------------------
Enter your choice: 1
-> Enter the word to insert: work

>> Word inserted successfully.
```



```
--------- Trie Menu ---------
1. Insert a word into the Trie
2. Search for a word in the Trie
3. List all words in the Trie
4. Exit
-----------------------------
Enter your choice: 3

>> List of all words in the Trie:
tie
tree
try
woke
word
work
worry
```

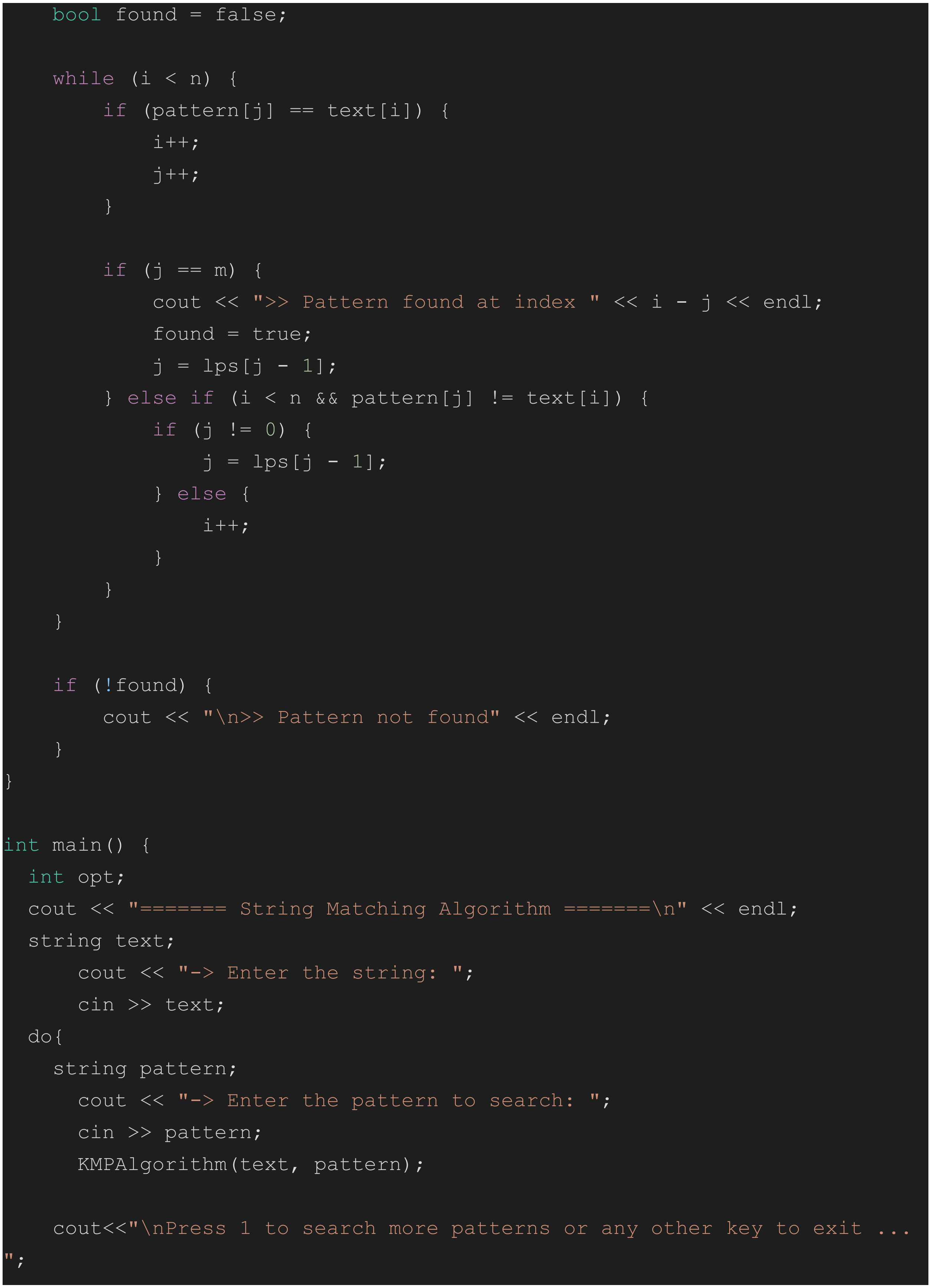


```
--------- Trie Menu ---------
1. Insert a word into the Trie
2. Search for a word in the Trie
3. List all words in the Trie
4. Exit
-----------------------------
Enter your choice: 2
-> Enter the word to search: woke

>> Word found in the Trie.

--------- Trie Menu ---------
1. Insert a word into the Trie
2. Search for a word in the Trie
3. List all words in the Trie
4. Exit
-----------------------------
Enter your choice: 2
-> Enter the word to search: wide

>> Word not found in the Trie.

--------- Trie Menu ---------
1. Insert a word into the Trie
2. Search for a word in the Trie
3. List all words in the Trie
4. Exit
-----------------------------
Enter your choice: 4
Exiting the program.

--------------------------------
Process exited after 101.4 seconds with return value 0
Press any key to continue . . .
```

Question 7 : Write a program to search a pattern in a given text using the KMP algorithm.

Code :-

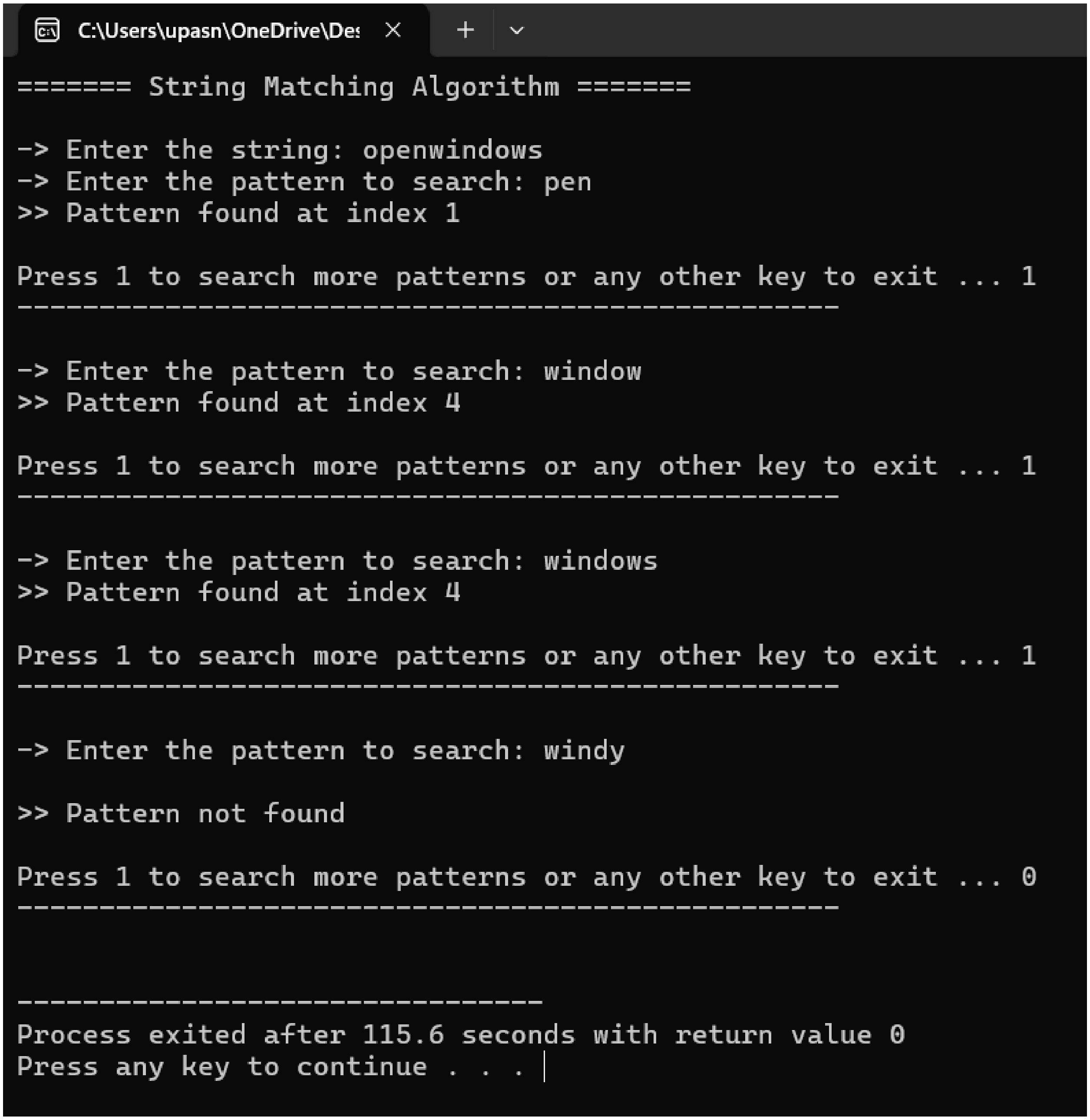
```
#include <iostream>
#include <vector>
using namespace std;

void computeLPS(const string &pattern, vector<int> &lps) {
    int length = 0;
    lps[0] = 0;
    int i = 1;

    while (i < pattern.length()) {
        if (pattern[i] == pattern[length]) {
            length++;
            lps[i] = length;
            i++;
        } else {
            if (length != 0) {
                length = lps[length - 1];
            } else {
                lps[i] = 0;
                i++;
            }
        }
    }
}

void KMPAlgorithm(const string &text, const string &pattern) {
    int n = text.length();
    int m = pattern.length();
    vector<int> lps(m);
    computeLPS(pattern, lps);

    int i = 0;
    int j = 0;
```



```
    bool found = false;

    while (i < n) {
        if (pattern[j] == text[i]) {
            i++;
            j++;
        }

        if (j == m) {
            cout << ">> Pattern found at index " << i - j << endl;
            found = true;
            j = lps[j - 1];
        } else if (i < n && pattern[j] != text[i]) {
            if (j != 0) {
                j = lps[j - 1];
            } else {
                i++;
            }
        }
    }

    if (!found) {
        cout << "\n>> Pattern not found" << endl;
    }
}

int main() {
  int opt;
  cout << "======= String Matching Algorithm =======\n" << endl;
  string text;
      cout << "-> Enter the string: ";
      cin >> text;
  do{
    string pattern;
      cout << "-> Enter the pattern to search: ";
      cin >> pattern;
      KMPAlgorithm(text, pattern);

    cout<<"\nPress 1 to search more patterns or any other key to exit ...
";
```

```
    cin>>opt;
    cout<<"--------------------------------------------------\n"<<endl;
  }while(opt == 1);

    return 0;
}
```

OUTPUT:

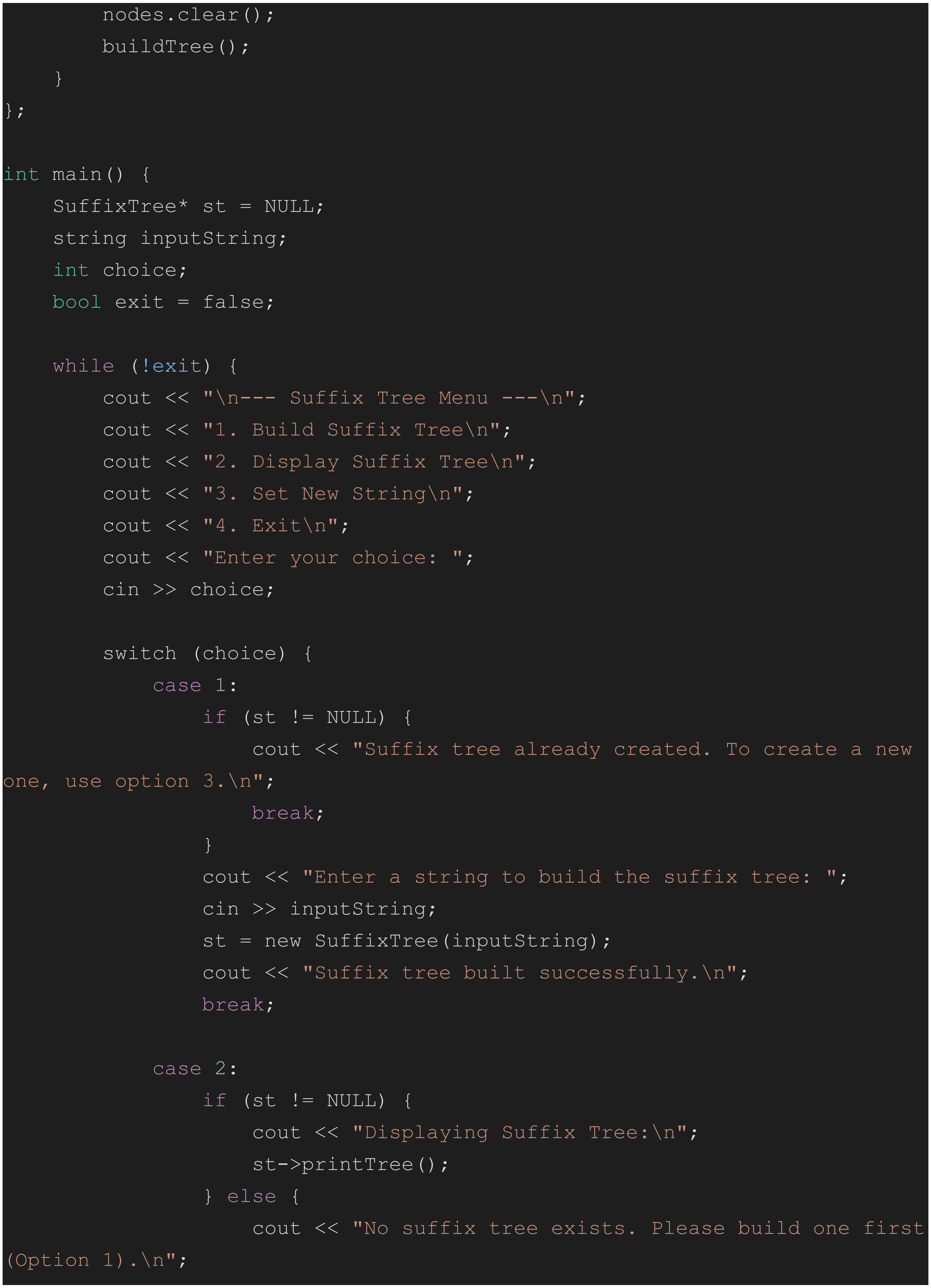


```
======= String Matching Algorithm =======

-> Enter the string: openwindows
-> Enter the pattern to search: pen
>> Pattern found at index 1

Press 1 to search more patterns or any other key to exit ... 1
--------------------------------------------------


-> Enter the pattern to search: window
>> Pattern found at index 4

Press 1 to search more patterns or any other key to exit ... 1
--------------------------------------------------


-> Enter the pattern to search: windows
>> Pattern found at index 4

Press 1 to search more patterns or any other key to exit ... 1
--------------------------------------------------


-> Enter the pattern to search: windy

>> Pattern not found

Press 1 to search more patterns or any other key to exit ... 0
--------------------------------------------------



--------------------------------
Process exited after 115.6 seconds with return value 0
Press any key to continue . . .
```

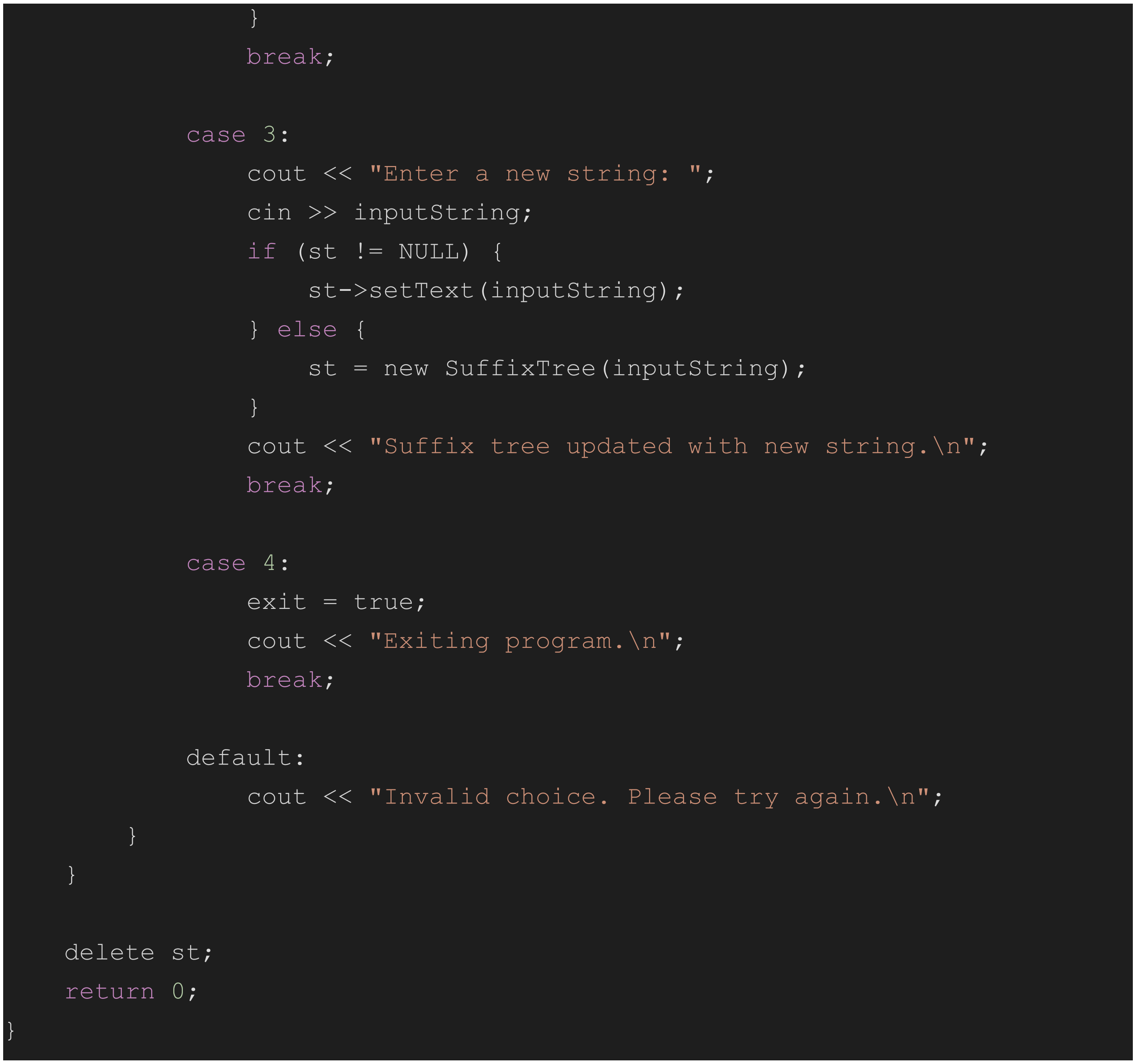
Question 8 : Write a program to implement a Suffix tree.

Code :-

```
#include <iostream>
#include <vector>
#include <string>
#include <map>
using namespace std;

class SuffixTree {
private:
    struct Node {
        map<char, Node*> children;
        int start, *end;
        int suffixLink;

        Node(int start, int* end) : start(start), end(end), suffixLink(-1)
{}
    };

    string text;
    vector<Node*> nodes;
    int size;

    void buildTree() {
        int n = text.size();
        int* end = new int(-1);  // For representing the end of a string
in the tree
        Node* root = new Node(-1, end);
        nodes.push_back(root);

        int activeNodeIndex = 0;  // Track the index of the active node
        int activeEdge = -1;
        int activeLength = 0;
        int remainder = 0;  // Number of suffixes to be added

        for (int i = 0; i < n; ++i) {
            remainder++;
            while (remainder > 0) {
                if (activeLength == 0) activeEdge = i;

                if
(nodes[activeNodeIndex]->children.find(text[activeEdge]) ==
nodes[activeNodeIndex]->children.end()) {
                    nodes[activeNodeIndex]->children[text[activeEdge]] =
new Node(i, end);

nodes.push_back(nodes[activeNodeIndex]->children[text[activeEdge]]);
                    remainder--;
                } else {
                    Node* nextNode =
nodes[activeNodeIndex]->children[text[activeEdge]];

                    int edgeLength = *nextNode->end - nextNode->start + 1;
                    if (activeLength >= edgeLength) {
                        activeNodeIndex = getNodeIndex(nextNode);  // Get
node index explicitly
                        activeEdge += edgeLength;
                        activeLength -= edgeLength;
                        continue;
                    }

                    if (text[nextNode->start + activeLength] == text[i]) {
                        activeLength++;
                        break;
                    }

                    // Split the edge
                    int* splitEnd = new int(nextNode->start + activeLength
- 1);

                    Node* splitNode = new Node(nextNode->start, splitEnd);
                    nodes.push_back(splitNode);

                    // Add the new split node and children
                    nodes[activeNodeIndex]->children[text[activeEdge]] =
splitNode;

                    splitNode->children[text[i]] = new Node(i, end);
                    nodes.push_back(splitNode->children[text[i]]);
                    nextNode->start += activeLength;
                    splitNode->children[text[nextNode->start]] = nextNode;
                    remainder--;
                }
            }
        }
    }

    // Helper function to get the index of a node in the vector
    int getNodeIndex(Node* node) {
        for (int i = 0; i < nodes.size(); i++) {
            if (nodes[i] == node) {
                return i;
            }
        }
        return -1;
    }

    void displayTree(Node* node, int level) {
        for (map<char, Node*>::iterator it = node->children.begin(); it !=
node->children.end(); ++it) {
            for (int i = 0; i < level; ++i) cout << "  ";
            cout << text.substr(it->second->start, *it->second->end -
it->second->start + 1) << endl;
          displayTree(it->second, level + 1);
        }
    }

public:
    SuffixTree(const string& s) : text(s), size(s.size()) {
        buildTree();
    }

    void printTree() {
        Node* root = nodes[0];
        displayTree(root, 0);
    }

    void setText(const string& s) {
        text = s;
```

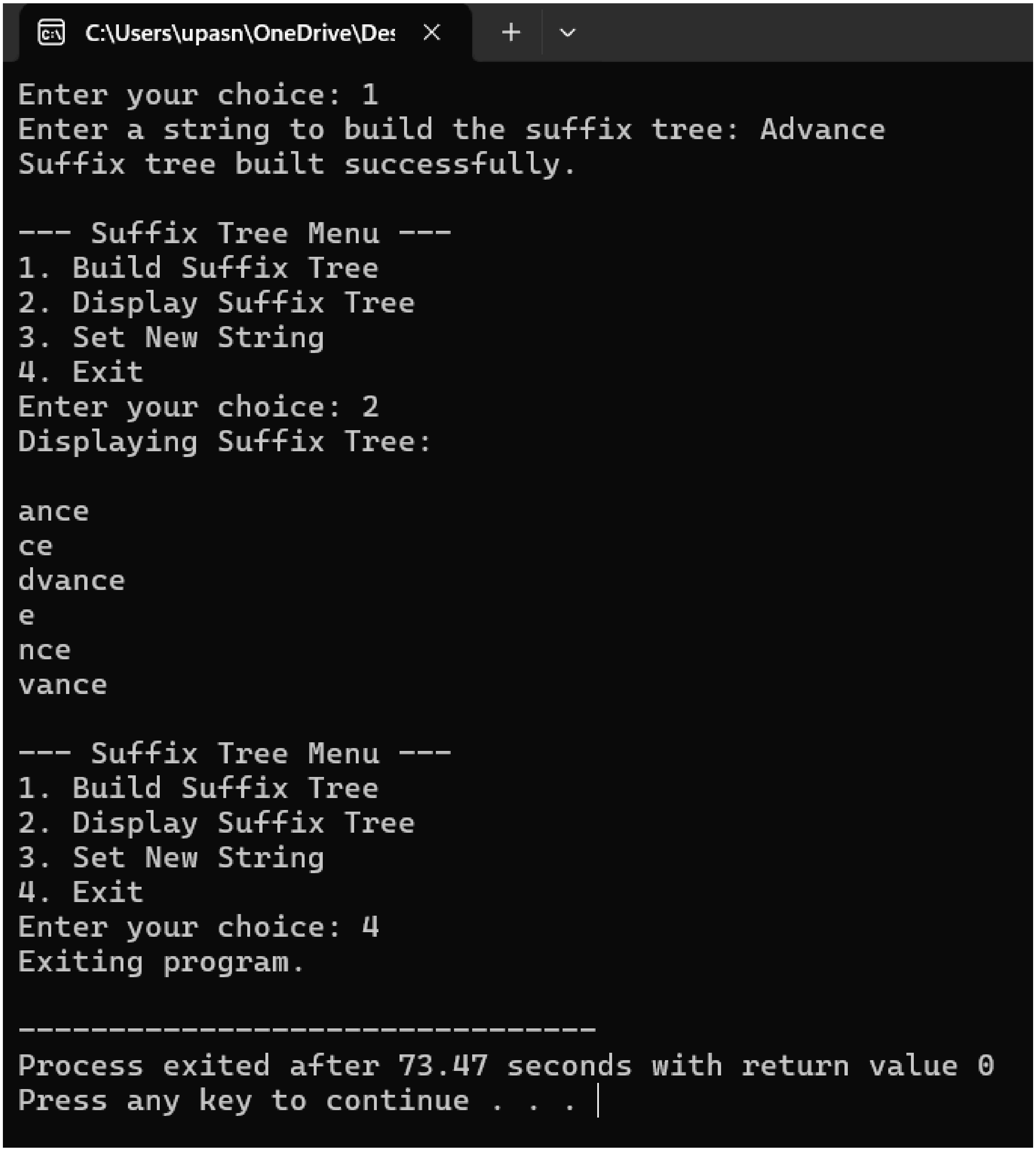


```
        nodes.clear();
        buildTree();
    }
};

int main() {
    SuffixTree* st = NULL;
    string inputString;
    int choice;
    bool exit = false;

    while (!exit) {
        cout << "\n--- Suffix Tree Menu ---\n";
        cout << "1. Build Suffix Tree\n";
        cout << "2. Display Suffix Tree\n";
        cout << "3. Set New String\n";
        cout << "4. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                if (st != NULL) {
                    cout << "Suffix tree already created. To create a new
one, use option 3.\n";
                    break;
                }
                cout << "Enter a string to build the suffix tree: ";
                cin >> inputString;
                st = new SuffixTree(inputString);
                cout << "Suffix tree built successfully.\n";
                break;

            case 2:
                if (st != NULL) {
                    cout << "Displaying Suffix Tree:\n";
                    st->printTree();
                } else {
                    cout << "No suffix tree exists. Please build one first
(Option 1).\n";
```



```
                }
                break;

            case 3:
                cout << "Enter a new string: ";
                cin >> inputString;
                if (st != NULL) {
                    st->setText(inputString);
                } else {
                    st = new SuffixTree(inputString);
                }
                cout << "Suffix tree updated with new string.\n";
                break;

            case 4:
                exit = true;
                cout << "Exiting program.\n";
                break;

            default:
                cout << "Invalid choice. Please try again.\n";
        }
    }

    delete st;
    return 0;
}
```

OUTPUT:



```
Enter your choice: 1
Enter a string to build the suffix tree: Advance
Suffix tree built successfully.

--- Suffix Tree Menu ---
1. Build Suffix Tree
2. Display Suffix Tree
3. Set New String
4. Exit
Enter your choice: 2
Displaying Suffix Tree:

ance
ce
dvance
e
nce
vance

--- Suffix Tree Menu ---
1. Build Suffix Tree
2. Display Suffix Tree
3. Set New String
4. Exit
Enter your choice: 4
Exiting program.

--------------------------------
Process exited after 73.47 seconds with return value 0
Press any key to continue . . .
```