1. You are given a list of integers nums with n elements (3 ≤ n ≤ 100000), the value of each element is between -1000 to 1000.

**Request:** Implement function:

int maximumProduct(vector<int>& nums);

Where nums is the given list of integers. This function returns the maximum product of 3 elements which can be found from the given list nums.

**Example:**

The list of integers is {5, 4, 1, 3, -2, -2}. Therefore, the maximum product is 60 by choosing 3 elements of the given list are 5, 4, 3.

**Note:**

In this exercise, the libraries iostream, string, cstring, climits, utility, vector, list, stack, queue, map, unordered\_map, set, unordered\_set, functional, algorithm has been included and namespace std are used. You can write helper functions and classes. Importing other libraries is allowed, but not encouraged, and may result in unexpected errors.

**For example:**

| **Test** | **Result** |
| --- | --- |
| vector<int>nums{5, 4, 1, 3, -2, -2};  cout << maximumProduct(nums); | 60 |
| vector<int>nums{4,4,1,2,-5,3,-2,-5,0,-2};  cout << maximumProduct(nums); | 100 |

|  |
| --- |
| int maximumProduct (vector<int>& nums) {  sort (nums.begin(), nums.end());  int len = nums.size();  return max (nums[0]\*nums[1]\*nums[len-1], max (nums[0]\*nums[len-1]\*nums[len-2], nums[len-1]\*nums[len-2]\*nums[len-3]));  } |

2. Trong ngày hội CSE Job Fair 2020, Ban tổ chức đã treo cờ dọc theo đường dẫn vào tòa H6, có N lá cờ được đánh số từ 1 đến N, lá cờ thứ i có màu Ai. Tuy nhiên, sau khi treo cờ lên, người tổ chức sự kiện - bạn Quang Anh thấy dãy cờ có nhiều màu sắc khác nhau là không hợp lý. Phía Đoàn hội tiến hành rà soát và cho biết còn dư M lá cờ, đánh số từ 1 đến M, lá cờ thứ j có màu Bj. Ban tổ chức quyết định sẽ thay thế một số lá cờ trong số N lá cờ ban đầu để được dãy cờ mới có ít màu nhất có thể. Lá cờ bị thay xuống sẽ bị rách nên không thể dùng lại cho các lần thay tiếp theo, cũng như do thời gian có hạn, bất kì lá cờ nào thay lên cũng không được phép gỡ xuống.

**Yêu cầu**: Hãy tìm cách thay một số (hoặc giữ nguyên) lá cờ đã treo bằng một số lá cờ trong số cờ còn dư sao cho tổng số màu xuất hiện trên dãy cờ chính thức (N lá) là ít nhất.

**Dữ liệu**:

* Dòng đầu ghi hai số N và M lần lượt là số lá cờ chính thức và số lá cờ còn dư
* Dòng thứ hai ghi N số nguyên Ai cho biết màu của các lá cờ đã treo (0 <= Ai <= 255, 1<= i <= N)
* Dòng thứ ba ghi M số nguyên Bj cho biết màu của các lá cờ còn dư (0 <= Bj <= 255, 1<= j <= M)

**Kết quả**: in ra một số duy nhất, không có bất kì khoảng trắng nào, là số màu mới của dãy cờ chính thức sau khi được thay thế.

**Giải thích testcase 1**:

Dãy cờ mới sẽ là 1 2 5 **5 2 5 5** 5 5 -> Có 3 màu trong dãy cờ mới.

**Các thư viện ĐÃ có trong bài, các bạn KHÔNG được phép thêm bất kì thư viện nào khác:**

#include <iostream>

#include <fstream>

#include <string>

#include <cmath>

#include <vector>

#include <algorithm>

#include <stack>

#include <queue>

#include <map>

**For example:**

| **Test** | **Result** |
| --- | --- |
| int n=9;  int m=4;  vector<int> A({1,2,5,4,8,9,3,5,5});  vector<int> B({2,5,5,5});  cout << flag(n, m,A,B); | 3 |
| int n=45;  int m=1;  vector<int> A({32,134,131,44,194,254,63,209,140,181,29,108,94,153,165,117,159,2,33,31,133,229,255,47,144,74,120,15,88,0,111,30,137,143,156,75,44,110,233,226,40,118,28,165,8});  vector<int> B({151});  cout << flag(n, m,A,B); | 43 |

|  |
| --- |
| //Helping functions goes here  int flag(int n, int m,vector<int> A, vector<int> B){  //TODO  } |

3. Given a Binary tree, the task is to traverse all the nodes of the tree using Breadth First Search algorithm and print the order of visited nodes (has no blank space at the end)

#include<iostream>

#include<string>

#include<queue>

using namespace std;

template<class K, class V>

class BinaryTree

{

public:

class Node;

private:

Node \*root;

public:

BinaryTree() : root(nullptr) {}

~BinaryTree()

{

// You have to delete all Nodes in BinaryTree. However in this task, you can ignore it.

}

class Node

{

private:

K key;

V value;

Node \*pLeft, \*pRight;

friend class BinaryTree<K, V>;

public:

Node(K key, V value) : key(key), value(value), pLeft(NULL), pRight(NULL) {}

~Node() {}

};

void addNode(string posFromRoot, K key, V value)

{

if(posFromRoot == "")

{

this->root = new Node(key, value);

return;

}

Node\* walker = this->root;

int l = posFromRoot.length();

for (int i = 0; i < l-1; i++)

{

if (!walker)

return;

if (posFromRoot[i] == 'L')

walker = walker->pLeft;

if (posFromRoot[i] == 'R')

walker = walker->pRight;

}

if(posFromRoot[l-1] == 'L')

walker->pLeft = new Node(key, value);

if(posFromRoot[l-1] == 'R')

walker->pRight = new Node(key, value);

}

// STUDENT ANSWER BEGIN

// STUDENT ANSWER END

};

You can define other functions to help you.

**For example:**

| **Test** | **Result** |
| --- | --- |
| BinaryTree<int, int> binaryTree;  binaryTree.addNode("",2, 4); // Add to root  binaryTree.addNode("L",3, 6); // Add to root's left node  binaryTree.addNode("R",5, 9); // Add to root's right node  binaryTree.BFS(); | 4 6 9 |

|  |
| --- |
| // STUDENT ANSWER BEGIN  // You can define other functions here to help you.  void breadthFirstSearch (Node \*root) {  if (root == NULL) return;  queue <Node\*> bf\_queue;  bf\_queue.push (root);  while (!bf\_queue.empty()) {  Node\* current = bf\_queue.front();  bf\_queue.pop();  cout << current -> value << " ";  if (current -> pLeft != NULL) {  bf\_queue.push (current -> pLeft);  }  if (current -> pRight != NULL) {  bf\_queue.push (current -> pRight);  }  }  }  void BFS() {  breadthFirstSearch (root);  }  // STUDENT ANSWER END |

4. Given a Binary tree, the task is to calculate the sum of leaf nodes. (Leaf nodes are nodes which have no children)

#include<iostream>

#include<string>

using namespace std;

template<class K, class V>

class BinaryTree

{

public:

class Node;

private:

Node \*root;

public:

BinaryTree() : root(nullptr) {}

~BinaryTree()

{

// You have to delete all Nodes in BinaryTree. However in this task, you can ignore it.

}

class Node

{

private:

K key;

V value;

Node \*pLeft, \*pRight;

friend class BinaryTree<K, V>;

public:

Node(K key, V value) : key(key), value(value), pLeft(NULL), pRight(NULL) {}

~Node() {}

};

void addNode(string posFromRoot, K key, V value)

{

if(posFromRoot == "")

{

this->root = new Node(key, value);

return;

}

Node\* walker = this->root;

int l = posFromRoot.length();

for (int i = 0; i < l-1; i++)

{

if (!walker)

return;

if (posFromRoot[i] == 'L')

walker = walker->pLeft;

if (posFromRoot[i] == 'R')

walker = walker->pRight;

}

if(posFromRoot[l-1] == 'L')

walker->pLeft = new Node(key, value);

if(posFromRoot[l-1] == 'R')

walker->pRight = new Node(key, value);

}

//Helping functions

int sumOfLeafs(){

//TODO

}

};

You can write other functions to achieve this task.

**For example:**

| **Test** | **Result** |
| --- | --- |
| BinaryTree<int, int> binaryTree;  binaryTree.addNode("", 2, 4);  cout << binaryTree.sumOfLeafs(); | 4 |
| BinaryTree<int, int> binaryTree;  binaryTree.addNode("", 2, 4);  binaryTree.addNode("L", 3, 6);  binaryTree.addNode("R", 5, 9);  cout << binaryTree.sumOfLeafs(); | 15 |

|  |
| --- |
| //Helping functions  int count (Node\* node) {  if (node == NULL) return 0;  else if (node -> pLeft == NULL && node -> pRight == NULL) return node -> value + count (node -> pLeft) + count (node -> pRight);  else return count (node -> pLeft) + count (node -> pRight);  }  int sumOfLeafs () {  return count (root);  } |

5. Given class **BinarySearchTree**, you need to finish method**find(i)** to check whether value i is in the tree or not; method **sum(l,r)** to calculate sum of all all elements v in the tree that has value greater than or equal to l and less than or equal to r.

#include <iostream>

#include <string>

#include <sstream>

using namespace std;

template<class T>

class BinarySearchTree

{

public:

class Node;

private:

Node\* root;

public:

BinarySearchTree() : root(nullptr) {}

~BinarySearchTree()

{

// You have to delete all Nodes in BinaryTree. However in this task, you can ignore it.

}

class Node

{

private:

T value;

Node\* pLeft, \* pRight;

friend class BinarySearchTree<T>;

public:

Node(T value) : value(value), pLeft(NULL), pRight(NULL) {}

~Node() {}

};

Node\* addRec(Node\* root, T value);

void add(T value) ;  
    // STUDENT ANSWER BEGIN

    // STUDENT ANSWER END  
};

**For example:**

| **Test** | **Result** |
| --- | --- |
| BinarySearchTree<int> bst;  for (int i = 0; i < 10; ++i) {  bst.add(i);  }  cout << bst.find(7) << endl;  cout << bst.sum(0, 4) << endl | 1  10 |

|  |
| --- |
| // STUDENT ANSWER BEGIN  // You can define other functions here to help you.  bool findRec (Node\* node, T i) {  if (node == NULL) return false;  else if (node -> value == i) return true;  else if (node -> value > i) return findRec (node -> pLeft, i);  else return findRec (node -> pRight, i);  }  T sumRec (Node\* node, T l, T r) {  if (node == NULL) return 0;  else if (node -> value < l) return sumRec (node -> pRight, l, r);  else if (node -> value > r) return sumRec (node -> pLeft, l, r);  else return node -> value + sumRec (node -> pLeft, l, r) + sumRec (node -> pRight, l, r);  }  bool find (T i) {  // TODO: return true if value i is in the tree; otherwise, return false.  return findRec (this -> root, i);  }  T sum (T l, T r) {  // TODO: return the sum of all element in the tree has value in range [l,r].  return this -> sumRec (this -> root, l, r);  }  // STUDENT ANSWER END |

6. Given class **BinarySearchTree**, you need to finish method getMin() and getMax() in this question.

#include <iostream>

#include <string>

#include <sstream>

using namespace std;

template<class T>

class BinarySearchTree

{

public:

class Node;

private:

Node\* root;

public:

BinarySearchTree() : root(nullptr) {}

~BinarySearchTree()

{

// You have to delete all Nodes in BinaryTree. However in this task, you can ignore it.

}

class Node

{

private:

T value;

Node\* pLeft, \* pRight;

friend class BinarySearchTree<T>;

public:

Node(T value) : value(value), pLeft(NULL), pRight(NULL) {}

~Node() {}

};

Node\* addRec(Node\* root, T value);

void add(T value) ;  
    // STUDENT ANSWER BEGIN

    // STUDENT ANSWER END  
};

**For example:**

| **Test** | **Result** |
| --- | --- |
| BinarySearchTree<int> bst;  for (int i = 0; i < 10; ++i) {  bst.add(i);  }  cout << bst.getMin() << endl;  cout << bst.getMax() << endl; | 0  9 |

|  |
| --- |
| // STUDENT ANSWER BEGIN  // You can define other functions here to help you.  T minRec (Node\* node) {  if (node -> pLeft == NULL) return node -> value;  else return minRec (node -> pLeft);  }  T maxRec (Node\* node) {  if (node -> pRight == NULL) return node -> value;  else return maxRec (node -> pRight);  }  T getMin() {  //TODO: return the minimum values of nodes in the tree.  return minRec (this -> root);  }  T getMax() {  //TODO: return the maximum values of nodes in the tree.  return maxRec (this -> root);  }  // STUDENT ANSWER END |