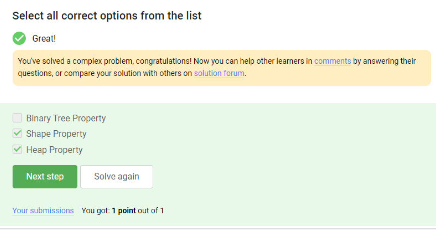
Amir Hossein yahyaei

811900087



A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A white and green background with black lines

Description automatically generated with medium confidence

A screenshot of a notebook

Description automatically generated

A screenshot of a computer

Description automatically generated

A white and green line

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

BST

#include <iostream>

using namespace std;

class Node {

private:

int key;

Node\* left;

Node\* right;

public:

Node(int key) {

this->key = key;

left = nullptr;

right = nullptr;

}

int getKey() const {

return key;

}

Node \*getLeft() const {

return left;

}

void setLeft(Node \*left) {

Node::left = left;

}

Node \*getRight() const {

return right;

}

void setRight(Node \*right) {

Node::right = right;

}

};

Node\* newNode(int key) {

return new Node(key);

}

Node\* insertRec(Node\* root, int key) {

if (root == nullptr) {

return newNode(key);

}

if (key < root->getKey()) {

root->setLeft(insertRec(root->getLeft(),key));

} else if (key > root->getKey()) {

root->setRight(insertRec(root->getRight(), key));

}

return root;

}

Node\* insert(Node\* root, int key) {

return insertRec(root, key);

}

bool search(Node\* root, int key) {

if (root == nullptr || root->getKey() == key) {

return root != nullptr;

}

if (key > root->getKey()) {

return search(root->getRight(), key);

}

return search(root->getLeft(), key);

}

void inorderTraversal(Node\* root) {

if (root != nullptr) {

inorderTraversal(root->getLeft());

std::cout << root->getKey() << " ";

inorderTraversal(root->getRight());

}

}

int main() {

Node \*root = nullptr;

while (true) {

cout << "Enter the number : \n"

<< "1. Add to the tree \n"

<< "2. Search in the tree \n"

<< "3. Exit \n";

char selected;

cin >> selected;

int key;

switch (selected) {

case '1':

cout << "Enter the Key: ";

cin >> key;

root = insert(root, key);

break;

case '2':

cout << "Enter the Key: ";

cin >> key;

if (search(root, key)) {

cout << key << " is found" << endl;

} else {

cout << key << " is not found" << endl;

}

break;

case '3':

exit(0);

default:

cout << "wrong.";

}

}

}

BFS&DFS

#include <iostream>

#include <list>

using namespace std;

class Graf {

private:

int numVertices;

vector<std::list<int>> adjLists;

vector<bool> visited;

void DFSUtil(int startVertex);

void BFSUtil(int startVertex);

public:

Graf(int vertices) : numVertices(vertices), adjLists(vertices), visited(vertices, false) {}

void addEdge(int src, int dest);

void DFS(int startVertex) {

visited.assign(numVertices, false);

DFSUtil(startVertex);

}

void BFS(int startVertex) {

visited.assign(numVertices, false);

BFSUtil(startVertex);

}

};

void Graf::addEdge(int src, int dest) {

adjLists[src].push\_back(dest);

adjLists[dest].push\_back(src);

}

void Graf::DFSUtil(int startVertex) {

visited[startVertex] = true;

cout << startVertex << " ";

for (int neighbor : adjLists[startVertex]) {

if (!visited[neighbor]) {

DFSUtil(neighbor);

}

}

}

void Graf::BFSUtil(int startVertex) {

queue<int> bfsQueue;

bfsQueue.push(startVertex);

visited[startVertex] = true;

while (!bfsQueue.empty()) {

int currentVertex = bfsQueue.front();

bfsQueue.pop();

cout << currentVertex << " ";

for (int neighbor : adjLists[currentVertex]) {

if (!visited[neighbor]) {

bfsQueue.push(neighbor);

visited[neighbor] = true;

}

}

}

}

int main() {

cout << "Enter number : ";

int gc;

cin >> gc;

Graf graf(gv);

while (true) {

cout << "Enter number : \n"

<< "1. Add edge \n"

<< "2. BFS \n"

<< "3. DFS \n"

<< "4. Exit \n";

char selected;

cin >> selected;

int src;

int dest;

int startVertex;

switch (selected) {

case '1':

cout << "Enter the source: ";

cin >> src;

cout << "Enter the destination: ";

cin >> dest;

graph.addEdge(src, dest);

break;

case '2':

cout << "Enter the startVertex: ";

cin >> startVertex;

graph.BFS(startVertex);

cout << "\n";

break;

case '3':

cout << "Enter the startVertex: ";

cin >> startVertex;

graph.DFS(startVertex);

cout << "\n";

break;

case '4':

exit(0);

default:

cout << "wrong.";

}

}

}

MAX HEAP

#include <iostream>

#include <vector>

using namespace std;

class MaxHeap {

private:

vector<int> heap;

void heapUp(int index) {

while (index > 0) {

int parentIndex = (index - 1) / 2;

if (heap[index] > heap[parentIndex]) {

swap(heap[index], heap[parentIndex]);

index = parentIndex;

} else {

break;

}

}

}

void heapDown(int index) {

int leftChild = 2 \* index + 1;

int rightChild = 2 \* index + 2;

int largest = index;

if (leftChild < heap.size() && heap[leftChild] > heap[largest]) {

largest = leftChild;

}

if (rightChild < heap.size() && heap[rightChild] > heap[largest]) {

largest = rightChild;

}

if (largest != index) {

swap(heap[index], heap[largest]);

heapDown(largest);

}

}

public:

void insert(int x) {

heap.push\_back(x);

heapUp(heap.size() - 1);

}

int extractMax() {

if (!heap.empty()) {

int max = heap[0];

heap[0] = heap.back();

heap.pop\_back();

heapDown(0);

return max;

} else {

cout << "heap is empty!";

return 0;

}

}

int getMax() const {

if (heap.empty()) {

throw

out\_of\_range("Heap is empty");

}

return heap[0];

}

bool isEmpty(){

return heap.empty();

}

};

int main() {

MaxHeap maxHeap;

while (true){

cout << "Enter number of what you want to do: \n"

<< "1. Add to heap \n"

<< "2. Extract Max \n"

<< "3. Get Max \n"

<< "4. Extract All \n"

<< "5. exit \n";

char selected;

cin >> selected;

int key;

switch (selected) {

case '1':

cout << "Enter the Key: ";

cin >> key;

maxHeap.insert(key);

break;

case '2':

cout << "Maximum of heap was " << maxHeap.extractMax() << "\n";

break;

case '3':

cout << "Maximum of heap is " << maxHeap.getMax() << "\n";

break;

case '4':

while (!maxHeap.isEmpty()){

cout << maxHeap.extractMax() << "\n";

}

break;

case '5':

exit(0);

}

}

return 0;

}