DSA Lab11

23K2001

M.Muzammil Siddiqui

BCS-3J

Q1:

//23K2001 - Muzammil

#include<iostream>

using namespace std;

class minHeap {

    private:

        int\* heap;

        int capacity;

        int currentSize;

        void swap(int& x, int& y){

            int temp = x;

            x = y;

            y = temp;

        }

        void heapify(int i) {

            int smallest = i;

            int left = 2 \* i + 1;

            int right = 2 \* i + 2;

            if (left < currentSize && heap[left] < heap[smallest])

                smallest = left;

            if (right < currentSize && heap[right] < heap[smallest])

                smallest = right;

            if (smallest != i) {

                swap(heap[i], heap[smallest]);

                heapify(smallest);

            }

        }

    public:

        minHeap():heap(nullptr),capacity(0),currentSize(0){}

        minHeap(int cap) {

            capacity = cap;

            heap = new int[capacity];

            currentSize = 0;

        }

        void insert(int value) {

            if (currentSize == capacity) {

                cout << "Heap is full!" << endl;

                return;

            }

            heap[currentSize] = value;

            int i = currentSize;

            currentSize++;

            while (i > 0 && heap[(i - 1) / 2] > heap[i]) {

                swap(heap[i], heap[(i - 1) / 2]);

                i = (i - 1) / 2;

            }

        }

        int deleteMin() {

            if (currentSize <= 0) {

                cout << "Heap is empty!" << endl;

                return -1;

            }

            int root = heap[0];

            heap[0] = heap[currentSize - 1];

            currentSize--;

            heapify(0);

            return root;

        }

        int peek() const {

            if (currentSize <= 0) {

                cout << "Heap is empty!" << endl;

                return -1;

            }

            return heap[0];

        }

        void buildHeap(int arr[], int n) {

            if (n > capacity) {

                cout << "Array size exceeds heap capacity!" << endl;

                return;

            }

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

                heap[i] = arr[i];

            currentSize = n;

            for (int i = currentSize / 2 - 1; i >= 0; i--)

                heapify(i);

        }

        void printHeap() const {

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

                cout << heap[i] << "  ";

            cout << endl;

        }

        int extractMin() {  return deleteMin(); }

        int size(){  return currentSize; }

        bool isEmpty(){  return currentSize == 0;    }

};

int main(){

    minHeap flex(10);

    flex.insert(3);

    flex.insert(1);

    flex.insert(5);

    flex.insert(4);

    flex.insert(2);

    flex.insert(5);

    cout << "Heap: ";

    flex.printHeap();

    cout << "Highest priority package: " << flex.extractMin() << endl;

    cout << "Heap after extraction: ";

    flex.printHeap();

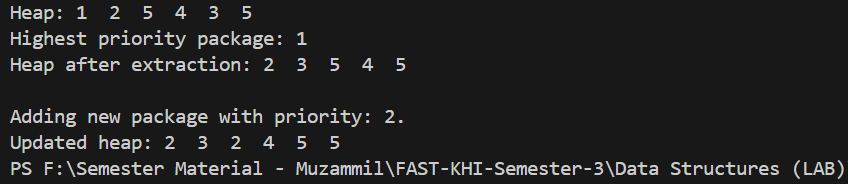
    cout<<endl<<"Adding new package with priority: 2."<<endl;

    flex.insert(2);

    cout<<"Updated heap: ";

    flex.printHeap();

    return 0;

}

Q2:

//23K2001 - Muzammil

#include<iostream>

using namespace std;

class MaxHeap{

private:

    int\* heap;

    int capacity;

    int currentSize;

    void swap(int& x, int& y) {

        int temp = x;

        x = y;

        y = temp;

    }

    void heapify(int i) {

        int largest = i;

        int left = 2 \* i + 1;

        int right = 2 \* i + 2;

        if (left < currentSize && heap[left] > heap[largest])

            largest = left;

        if (right < currentSize && heap[right] > heap[largest])

            largest = right;

        if (largest != i) {

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

            heapify(largest);

        }

    }

public:

    MaxHeap():heap(nullptr),capacity(0),currentSize(0){}

    MaxHeap(int cap) {

        capacity = cap;

        heap = new int[capacity];

        currentSize = 0;

    }

    void insert(int value) {

        if (currentSize == capacity) {

            cout << "Heap is full!" << endl;

            return;

        }

        heap[currentSize] = value;

        int i = currentSize;

        currentSize++;

        while (i > 0 && heap[(i - 1) / 2] < heap[i]) {

            swap(heap[i], heap[(i - 1) / 2]);

            i = (i - 1) / 2;

        }

    }

    int deleteMax() {

        if (currentSize <= 0) {

            cout << "Heap is empty!" << endl;

            return -1;

        }

        int root = heap[0];

        heap[0] = heap[currentSize - 1];

        currentSize--;

        heapify(0);

        return root;

    }

    int peek(){

        if (currentSize <= 0) {

            cout << "Heap is empty!" << endl;

            return -1;

        }

        return heap[0];

    }

    void buildHeap(int arr[], int n){

        if (n > capacity) {

            cout << "Array size exceeds heap capacity!" << endl;

            return;

        }

        for (int i = 0; i < n; i++)

            heap[i] = arr[i];

        currentSize = n;

        for (int i = currentSize / 2 - 1; i >= 0; i--)

            heapify(i);

    }

    void printHeap(){

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

            cout << heap[i] << " ";

        cout << endl;

    }

    int extractMax(){  return deleteMax(); }

    int size(){  return currentSize; }

    bool isEmpty(){  return currentSize == 0;    }

};

int main(){

    MaxHeap flex(10);

    flex.insert(3);

    flex.insert(1);

    flex.insert(5);

    flex.insert(4);

    flex.insert(2);

    cout << "Heap: ";

    flex.printHeap();

    cout << "Highest priority data: " << flex.extractMax() << endl;

    cout << "Heap after extraction: ";

    flex.printHeap();

    cout<<endl<<"Adding new data with priority: 2."<<endl;

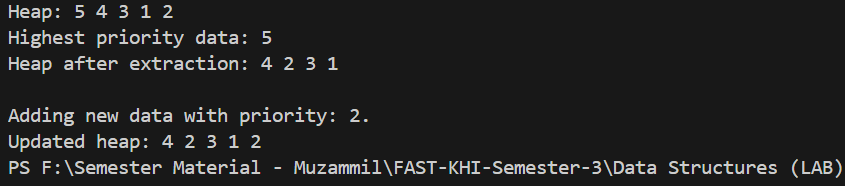
    flex.insert(2);

    cout<<"Updated heap: ";

    flex.printHeap();

    return 0;

}



Q3:

//23K2001 - Muzammil

#include<iostream>

using namespace std;

class minHeap {

    private:

        int\* heap;

        int capacity;

        int currentSize;

        void swap(int& x, int& y){

            int temp = x;

            x = y;

            y = temp;

        }

        void heapify(int i) {

            int smallest = i;

            int left = 2 \* i + 1;

            int right = 2 \* i + 2;

            if (left < currentSize && heap[left] < heap[smallest])

                smallest = left;

            if (right < currentSize && heap[right] < heap[smallest])

                smallest = right;

            if (smallest != i) {

                swap(heap[i], heap[smallest]);

                heapify(smallest);

            }

        }

    public:

        minHeap():heap(nullptr),capacity(0),currentSize(0){}

        minHeap(int cap) {

            capacity = cap;

            heap = new int[capacity];

            currentSize = 0;

        }

        void insert(int val) {

            if (currentSize == capacity) {

                cout << "Heap is full!" << endl;

                return;

            }

            heap[currentSize] = val;

            int i = currentSize;

            currentSize++;

            while (i > 0 && heap[(i - 1) / 2] > heap[i]) {

                swap(heap[i], heap[(i - 1) / 2]);

                i = (i - 1) / 2;

            }

        }

        int deleteMin() {

            if (currentSize <= 0) {

                cout << "Heap is empty!" << endl;

                return -1;

            }

            int root = heap[0];

            heap[0] = heap[currentSize - 1];

            currentSize--;

            heapify(0);

            return root;

        }

        int peek() const {

            if (currentSize <= 0) {

                cout << "Heap is empty!" << endl;

                return -1;

            }

            return heap[0];

        }

        void buildHeap(int arr[], int n) {

            if (n > capacity) {

                cout << "Array size exceeds heap capacity!" << endl;

                return;

            }

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

                heap[i] = arr[i];

            currentSize = n;

            for (int i = currentSize / 2 - 1; i >= 0; i--)

                heapify(i);

        }

        void printHeap() const {

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

                cout << heap[i] << "  ";

            cout << endl;

        }

        void deleteNode(int val){

            int index = -1;

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

                if (heap[i] == val) {

                    index = i;

                    break;

                }

            }

            if (index == -1) {

                cout << "Invalid value!" << endl;

                return;

            }

            heap[index] = heap[currentSize - 1];

            currentSize--;

            heapify(index);

            int parent = (index - 1) / 2;

            if (index > 0 && heap[parent] > heap[index]) {

                while (index > 0 && heap[parent] > heap[index]) {

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

                    index = parent;

                    parent = (index - 1) / 2;

                }

            }

            cout<<val;

        }

        int extractMin() {  return deleteMin(); }

        int size(){  return currentSize; }

        bool isEmpty(){  return currentSize == 0;    }

};

class MaxHeap{

    private:

    int\* heap;

    int capacity;

    int currentSize;

    void swap(int& x, int& y) {

        int temp = x;

        x = y;

        y = temp;

    }

    void heapify(int i) {

        int largest = i;

        int left = 2 \* i + 1;

        int right = 2 \* i + 2;

        if (left < currentSize && heap[left] > heap[largest])

            largest = left;

        if (right < currentSize && heap[right] > heap[largest])

            largest = right;

        if (largest != i) {

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

            heapify(largest);

        }

    }

    public:

    MaxHeap():heap(nullptr),capacity(0),currentSize(0){}

    MaxHeap(int cap) {

        capacity = cap;

        heap = new int[capacity];

        currentSize = 0;

    }

    void insert(int val) {

        if (currentSize == capacity) {

            cout << "Heap is full!" << endl;

            return;

        }

        heap[currentSize] = val;

        int i = currentSize;

        currentSize++;

        while (i > 0 && heap[(i - 1) / 2] < heap[i]) {

            swap(heap[i], heap[(i - 1) / 2]);

            i = (i - 1) / 2;

        }

    }

    int deleteMax() {

        if (currentSize <= 0) {

            cout << "Heap is empty!" << endl;

            return -1;

        }

        int root = heap[0];

        heap[0] = heap[currentSize - 1];

        currentSize--;

        heapify(0);

        return root;

    }

    int peek(){

        if (currentSize <= 0) {

            cout << "Heap is empty!" << endl;

            return -1;

        }

        return heap[0];

    }

    void buildHeap(int arr[], int n){

        if (n > capacity) {

            cout << "Array size exceeds heap capacity!" << endl;

            return;

        }

        for (int i = 0; i < n; i++)

            heap[i] = arr[i];

        currentSize = n;

        for (int i = currentSize / 2 - 1; i >= 0; i--)

            heapify(i);

    }

    void printHeap(){

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

            cout << heap[i] << "  ";

        cout << endl;

    }

    void deleteNode(int val){

        int index = -1;

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

            if (heap[i] == val) {

                index = i;

                break;

            }

        }

        if (index == -1) {

            cout << "Invalid value!" << endl;

            return;

        }

        heap[index] = heap[currentSize - 1];

        currentSize--;

        heapify(index);

        int parent = (index - 1) / 2;

        if (index > 0 && heap[parent] < heap[index]) {

            while (index > 0 && heap[parent] < heap[index]) {

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

                index = parent;

                parent = (index - 1) / 2;

            }

        }

        cout<<val;

    }

    void heapSort() {

        int originalSize = currentSize;

        for (int i = currentSize - 1; i > 0; i--) {

            swap(heap[0], heap[i]);

            currentSize--;

            heapify(0);

        }

        currentSize = originalSize;

    }

    int extractMax(){  return deleteMax(); }

    int size(){  return currentSize; }

    bool isEmpty(){  return currentSize == 0;    }

};

int main(){

    int f[10] = {25,30,35,11,15,19,18,55,78,36};

    MaxHeap flex1(10);

    flex1.buildHeap(f,10);

    minHeap flex2(10);

    flex2.buildHeap(f,10);

    cout << "Max Heap: ";

    flex1.printHeap();

    cout << "Min Heap: ";

    flex2.printHeap();

    cout<<endl<<"Deleting 55 from max heap."<<endl;

    cout<<"Deleted: ";

    flex1.deleteNode(55);

    cout<<endl<<"Updated max heap: ";

    flex1.printHeap();

    cout<<endl<<"Sorted max heap: ";

    flex1.heapSort();

    flex1.printHeap();

    cout<<endl<<"Deleting 18 from min heap."<<endl;

    cout<<"Deleted: ";

    flex2.deleteNode(18);

    cout<<endl<<"Updated min heap: ";

    flex2.printHeap();

    return 0;

}

