МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА

Факультет електроніки та комп’ютерних технологій

ЗВІТ

про виконання лабораторної роботи №4

з дисципліни **“Алгоритми і структури даних”**

на тему **“ Структури даних. Піраміди”**

Виконав:

Студент 2 курсу

групи ФеП-23

Чепара Станіслав

Перевірив:

доцент кафедри РКТ, к.ф.-м.н.

Середницька Христина Ігорівна

Львів-2025

Мета роботи:

Навчитися працювати з пірамідами (кучами) та чергами з пріоритетами на основі Max-Heap і Min-Heap. Потрібно закріпити навички побудови піраміди, виконання пірамідального сортування, а також реалізації основних операцій із чергами з пріоритетами: пошуку максимального/мінімального елемента, видалення, зміни значення елемента та додавання нового елемента.

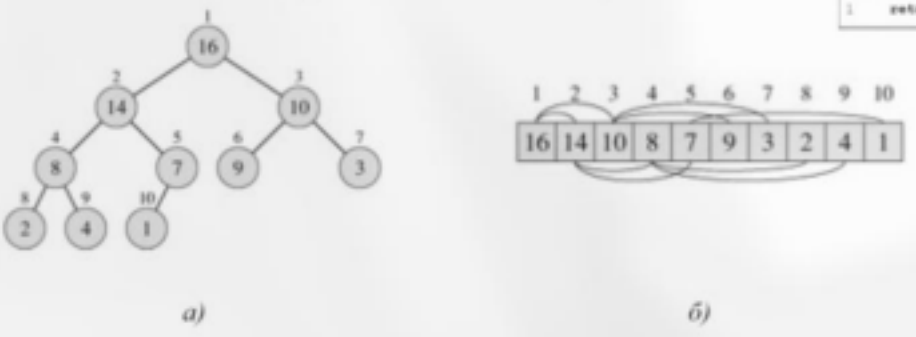
Теоретичні відомості:

Піраміда або *купа*, *стіс* (англ. *binary* ***heap***) – це структура даних, яка представляє собою масив, який можна розглядати як майже повне бінарне дерево. Кожний вузол цього дерева відповідає певному елементу масиву. На всіх рівнях, окрім, можливо, останнього, дерево повністю заповнено (заповнений рівень – це такий, який містить максимальну можливу кількість вузлів). Останній рівень заповнюється зліва направо до тих пір, доки в масиві не закінчаться елементи.

В корні дерева знаходиться елемент $A[1]$, а далі воно будується за наступним принципом: якщо якомусь вузлу відповідає індекс i, то індекс його батьківського вузла обраховується за допомогою наведеної нижче процедури `Parent(i)`, індекс лівого дочірнього вузла – за допомогою процедури `Left(i)`, а індекс правого дочірнього вузла – за процедурою `Right(i)`:

```
Parent(i)
1    return ⌊i/2⌋
Left(i)
1    return 2i
Right(i)
1    return 2i + 1
```

Батьківські елементи завжди розташовані зліва від дочірніх.



а) *б)*

Рис. 4.1. Піраміда у вигляді *а)* бінарного дерева та *б)* масиву.

Розрізняють два види бінарних пірамід: *незростаючі* (англ. *max-heaps*) та *неспадні* (англ. *min-heaps*). В пірамідах обох видів значення, які розташовані у вузлах, задовольняють властивості піраміди, яке вирізняє один тип пірамід від інших. Властивість незростаючих пірамід полягає в тому, що для кожного відмінного від кореня вузла з індексом i виконується наступна нерівність:

$$A[\text{Parent}(i)] \geq A[i].$$

Іншими словами, значення вузла не перевищує значення батьківського для нього вузла. Таким чином, у незростаючих пірамідах найбільший елемент знаходиться в корені дерева, а значення вузлів піддерева, яке бере початок в деякому елементі, не більше значення самого цього елементу.

Принцип організації неспадної піраміди прямо протилежний. Властивість таких пірамід полягає в тому, що для всіх відмінних від кореня вузлів з індексом i виконується нерівність:

$$A[\text{Parent}(i)] \leq A[i].$$

Таким чином, найменший елемент такої піраміди знаходиться в її корені.

Зміст роботи:

Хід роботи:

Частина 1. Побудова піраміди. Пірамідальне сортування.

1. Створити нову бібліотеку `Heap` (файли `Heap.h, Heap.cpp`).
2. У бібліотеці `Heap`, створити функції `Parent(…), Left(…)` та `Right(…)` для навігації по піраміді.
3. У бібліотеці `Heap`, згідно описаних в теоретичній частині алгоритмів, створити функції `BuildMaxHeap(…)` та `MaxHeapify(…)` для побудови та підтримки властивості незростаючої піраміди.
4. У цій же бібліотеці створити функції `BuildMinHeap(…)` та `MinHeapify(…)`. для побудови та підтримки властивості неспадної піраміди. Реалізація цих функцій є дзеркально симетричною до функцій `BuildMaxHeap(…)` та `MaxHeapify(…)`.
5. У бібліотеці `Heap`, згідно описаного в теоретичній частині алгоритму, створити функцію `HeapSort(…)` для реалізації пірамідального сортування одномірного масиву даних. У функції `HeapSort(…)` передбачити можливість сортування за зростанням (виклик функції `BuildMaxHeap(…)`) чи за спаданням (виклик функції `BuildMinHeap(…)`).
6. Створити новий проект `Lab_05_01` та підключити до нього бібліотеку `Heap`. У функції `main()` проекту реалізувати меню для введення одномірного масиву даних, побудову на основі цих даних незростаючої чи неспадної піраміди, сортування введених даних за зростанням чи за спаданням, відображення результатів.
7. Відкомпілювати проект та продемонструвати його роботу для одномірного масиву даних, отриманого від викладача.

Частина 2. Черга з пріоритетами.

1. У бібліотеці `Heap`, згідно описаних в теоретичній частині алгоритмів, створити функції `HeapMax(…)`, `HeapExtractMax(…)`, `HeapIncreaseKey(…)` та `MaxHeapInsert(…)` для роботи з чергою з пріоритетами, побудованою на основі незростаючої піраміди. Ці функції повинні, відповідно, повертати значення максимального елементу черги, видаляти максимальний елемент, змінювати (збільшувати) значення елементу, додавати елемент у чергу. У функції `HeapIncreaseKey(…)` передбачити обробку помилки зменшення ключа.
2. У цій же бібліотеці створити функції `HeapMin(…)`, `HeapExtractMin(…)`, `HeapDecreaseKey(…)` та `MinHeapInsert(…)` для роботи з чергою з пріоритетами, побудованою на основі неспадної піраміди. Реалізація цих функцій є дзеркально симетричною до функцій `HeapMax(…)`, `HeapExtractMax(…)`, `HeapIncreaseKey(…)` та `MaxHeapInsert(…)`. У функції `HeapDecreaseKey(…)` передбачити обробку помилки збільшення ключа.
3. Створити новий проект `Lab_05_02` та підключити до нього бібліотеку `Heap`. У функції `main()` проекту реалізувати меню для введення одномірного масиву даних, побудову на основі цих даних незростаючої чи неспадної піраміди та виконання описаних у п.1 та п.2 операцій для черги з пріоритетами, а також відображення результатів.
4. Відкомпілювати проект та продемонструвати його роботу для одномірного масиву даних, отриманого від викладача.

Heap.h

```cpp
#ifndef HEAP_LIB_H
#define HEAP_LIB_H

#include <vector>
#include <iostream>
#include <stdexcept>
#include <climits>
#include <cmath>
#include <functional>
#include <string>

// 1-based купа: A[1..heap_size], A[0] не використовується
inline size_t Parent(size_t i){ return i/2; }
inline size_t Left(size_t i){ return 2*i; }
inline size_t Right(size_t i){ return 2*i+1; }

void MaxHeapify(std::vector<int>& A, size_t heap_size, size_t i);
void MinHeapify(std::vector<int>& A, size_t heap_size, size_t i);
void BuildMaxHeap(std::vector<int>& A, size_t& heap_size);
void BuildMinHeap(std::vector<int>& A, size_t& heap_size);

// Сортування: зростання/спадання
void HeapSortAsc(std::vector<int>& arr);
void HeapSortDesc(std::vector<int>& arr);

// Принтер зверху вниз (тільки "/" і "\")
void PrintHeapTree(const std::vector<int>& A, size_t heap_size, std::ostream& os = std::cout);

// MAX-черга
int  HeapMax(const std::vector<int>& A, size_t heap_size);
int  HeapExtractMax(std::vector<int>& A, size_t& heap_size);
void HeapIncreaseKey(std::vector<int>& A, size_t i, int key);
void MaxHeapInsert(std::vector<int>& A, size_t& heap_size, int key);

// MIN-черга
int  HeapMin(const std::vector<int>& A, size_t heap_size);
int  HeapExtractMin(std::vector<int>& A, size_t& heap_size);
void HeapDecreaseKey(std::vector<int>& A, size_t i, int key);
void MinHeapInsert(std::vector<int>& A, size_t& heap_size, int key);

#endif
```

Heap.cpp

Heap.cpp > ...

```cpp
#include "Heap.h"
#include <algorithm>
#include <functional>

static inline void swapv(int& a, int& b){ int t=a; a=b; b=t; }

void MaxHeapify(std::vector<int>& A, size_t heap_size, size_t i){
    while(true){
        size_t l=Left(i), r=Right(i), m=i;
        if(l<=heap_size && A[l]>A[m]) m=l;
        if(r<=heap_size && A[r]>A[m]) m=r;
        if(m==i) break;
        swapv(A[i], A[m]); i=m;
    }
}
void MinHeapify(std::vector<int>& A, size_t heap_size, size_t i){
    while(true){
        size_t l=Left(i), r=Right(i), m=i;
        if(l<=heap_size && A[l]<A[m]) m=l;
        if(r<=heap_size && A[r]<A[m]) m=r;
        if(m==i) break;
        swapv(A[i], A[m]); i=m;
    }
}

void BuildMaxHeap(std::vector<int>& A, size_t& heap_size){
    if (A.empty() || A[0]!=0){ A.insert(A.begin(), 0); }
    heap_size = A.size()-1;
    for(size_t i=heap_size/2; i>=1; --i){ MaxHeapify(A, heap_size, i); if(i==1) break; }
}
void BuildMinHeap(std::vector<int>& A, size_t& heap_size){
    if (A.empty() || A[0]!=0){ A.insert(A.begin(), 0); }
    heap_size = A.size()-1;
    for(size_t i=heap_size/2; i>=1; --i){ MinHeapify(A, heap_size, i); if(i==1) break; }
}

void HeapSortAsc(std::vector<int>& arr){
    std::vector<int> A; A.reserve(arr.size()+1); A.push_back(0);
    for(int v: arr) A.push_back(v);
    size_t heap_size; BuildMaxHeap(A, heap_size);
    for(size_t i=heap_size; i>=2; --i){
        swapv(A[1], A[i]); --heap_size; MaxHeapify(A, heap_size, 1);
        if(i==2) break;
    }
    for(size_t i=0;i<arr.size();++i) arr[i]=A[i+1];
}
void HeapSortDesc(std::vector<int>& arr){
    std::vector<int> A; A.reserve(arr.size()+1); A.push_back(0);
    for(int v: arr) A.push_back(v);
    size_t heap_size; BuildMinHeap(A, heap_size);
    for(size_t i=heap_size; i>=2; --i){
        swapv(A[1], A[i]); --heap_size; MinHeapify(A, heap_size, 1);
        if(i==2) break;
    }
    for(size_t i=0;i<arr.size();++i) arr[i]=A[i+1];
}

// Топ-даун принтер без "----": лише "/" і "\"
void PrintHeapTree(const std::vector<int>& A, size_t heap_size, std::ostream& os){
    if (heap_size==0){ os<<"[порожньо]\n"; return; }

    auto digits = [](long long x){ return (int)std::to_string(x).size(); };
    int maxw=1; for(size_t i=1;i<=heap_size;i++) maxw=std::max(maxw, digits(A[i]));
    int cell = std::max(3, maxw+2);
    int h = (int)std::floor(std::log2((double)heap_size));
    int width = ((int)1<<(h+1))*cell;
    int rows = h*2+1;
    std::vector<std::string> canvas(rows, std::string(width,' '));

```

```
    auto place_center = [&](int r,int c,const std::string& t){
        int start=c-(int)t.size()/2; if(start<0) start=0;
        for(size_t k=0;k<t.size() && start+(int)k<width;k++) canvas[r][start+(int)k]=t[k];
    };

    std::function<void(size_t,int,int,int)> draw = [&](size_t i,int depth,int cx,int span){
        if(i>heap_size) return;
        place_center(depth*2, cx, std::to_string(A[i]));
        int child_span = std::max(cell, span/2);
        size_t li=Left(i), ri=Right(i);
        int y = depth*2+1;
        if(li<=heap_size){
            int lx = cx - child_span;
            int sx = (lx + cx)/2;
            if(y<rows && sx>=0 && sx<width) canvas[y][sx]='/';
            draw(li, depth+1, lx, child_span/2);
        }
        if(ri<=heap_size){
            int rx = cx + child_span;
            int sx = (cx + rx)/2;
            if(y<rows && sx>=0 && sx<width) canvas[y][sx]='\\';
            draw(ri, depth+1, rx, child_span/2);
        }
    };

    int root_x = width/2;
    draw(1,0,root_x, ((int)1<<h)*(cell/2));
    for(auto& line: canvas) os<<line<<"\n";
}

// MAX
int HeapMax(const std::vector<int>& A, size_t heap_size){ if(heap_size<1) throw std::runtime_error("Черга порожня"); return A[1]; }
int HeapExtractMax(std::vector<int>& A, size_t& heap_size){
    if(heap_size<1) throw std::runtime_error("Черга порожня");
    int v=A[1]; A[1]=A[heap_size]; --heap_size; if(heap_size>=1) MaxHeapify(A, heap_size, 1); return v;
}
void HeapIncreaseKey(std::vector<int>& A, size_t i, int key){
    if(key<A[i]) throw std::runtime_error("Новий ключ менший за поточний");
    A[i]=key; while(i>1 && A[Parent(i)]<A[i]){ std::swap(A[i],A[Parent(i)]); i=Parent(i); }
}
void MaxHeapInsert(std::vector<int>& A, size_t& heap_size, int key){
    if(A.empty() || A[0]!=0){ A.insert(A.begin(),0); heap_size=A.size()-1; }
    if(heap_size+1>=A.size()) A.push_back(INT_MIN); else A[heap_size+1]=INT_MIN;
    ++heap_size; HeapIncreaseKey(A, heap_size, key);
}

// MIN
int HeapMin(const std::vector<int>& A, size_t heap_size){ if(heap_size<1) throw std::runtime_error("Черга порожня"); return A[1]; }
int HeapExtractMin(std::vector<int>& A, size_t& heap_size){
    if(heap_size<1) throw std::runtime_error("Черга порожня");
    int v=A[1]; A[1]=A[heap_size]; --heap_size; if(heap_size>=1) MinHeapify(A, heap_size, 1); return v;
}
void HeapDecreaseKey(std::vector<int>& A, size_t i, int key){
    if(key>A[i]) throw std::runtime_error("Новий ключ більший за поточний");
    A[i]=key; while(i>1 && A[Parent(i)]>A[i]){ std::swap(A[i],A[Parent(i)]); i=Parent(i); }
}
void MinHeapInsert(std::vector<int>& A, size_t& heap_size, int key){
    if(A.empty() || A[0]!=0){ A.insert(A.begin(),0); heap_size=A.size()-1; }
    if(heap_size+1>=A.size()) A.push_back(INT_MAX); else A[heap_size+1]=INT_MAX;
    ++heap_size; HeapDecreaseKey(A, heap_size, key);
}
```

Part 1 main.cpp

Part1 > main.cpp > ...

```cpp
#include <bits/stdc++.h>
#include "Heap.h"
using namespace std;

static void print_vec(const vector<int>& a){
    cout << "[ ";
    for(size_t i=0;i<a.size();++i) cout<<a[i]<<(i+1==a.size()?"":" ");
    cout << " ]\n";
}

int main(){
    ios::sync_with_stdio(false);
    cin.tie(&cout);

    cout << "=== Частина 1: Купа та сортування ===\n";
    cout << "Введіть n: " << flush;
    int n; if(!(cin>>n) || n<0){ cerr<<"Некоректне n\n"; return 1; }
    vector<int> a(n);
    cout << "Введіть " << n << " цілих: " << flush;
    for(int i=0;i<n;i++) cin>>a[i];

    // Будуємо MAX-heap і показуємо дерево
    vector<int> A(1,0); A.insert(A.end(), a.begin(), a.end());
    size_t heap_size; BuildMaxHeap(A, heap_size);
    cout << "\nMAX-heap:\n";
    PrintHeapTree(A, heap_size);

    // HeapSort
    auto asc = a; HeapSortAsc(asc);
    cout << "\nПісля HeapSort (зростання): "; print_vec(asc);

    auto desc = a; HeapSortDesc(desc);
    cout << "Після HeapSort (спадання): "; print_vec(desc);
    return 0;
}
```

Part 2 main.cpp

Part2 > main.cpp > ...

```cpp
#include <bits/stdc++.h>
#include "Heap.h"
using namespace std;

int main(){
    ios::sync_with_stdio(false);
    cin.tie(&cout);

    cout << "=== Частина 2: Черги з пріоритетами ===\n";
    cout << "Введіть n: " << flush;
    int n; if(!(cin>>n) || n<0){ cerr<<"Некоректне n\n"; return 1; }
    vector<int> data(n);
    cout << "Введіть " << n << " цілих: " << flush;
    for(int i=0;i<n;i++) cin>>data[i];

    // MAX-черга
    vector<int> A(1,0); A.insert(A.end(), data.begin(), data.end());
    size_t heap_size; BuildMaxHeap(A, heap_size);
    cout << "\nMAX-heap:\n"; PrintHeapTree(A, heap_size);
    cout << "Максимум: " << HeapMax(A, heap_size) << "\n";
    cout << "ExtractMax: " << HeapExtractMax(A, heap_size) << "\n";
    PrintHeapTree(A, heap_size);
    MaxHeapInsert(A, heap_size, 100);
    cout << "Після Insert(100):\n"; PrintHeapTree(A, heap_size);

    // MIN-черга
    vector<int> B(1,0); B.insert(B.end(), data.begin(), data.end());
    size_t heap_size2; BuildMinHeap(B, heap_size2);
    cout << "\nMIN-heap:\n"; PrintHeapTree(B, heap_size2);
    cout << "Мінімум: " << HeapMin(B, heap_size2) << "\n";
    cout << "ExtractMin: " << HeapExtractMin(B, heap_size2) << "\n";
    PrintHeapTree(B, heap_size2);
    MinHeapInsert(B, heap_size2, -100);
    cout << "Після Insert(-100):\n"; PrintHeapTree(B, heap_size2);

    return 0;
}
```

Результат:

part1

```
stanislav@fedora:~/Desktop/Algoritms&Data_structure/Lab_05_Heaps/build$ "/h
=== Частина 1: Купа та сортування ===
Введіть n: 8
Введіть 8 цілих: 4
2
7
9
24
26
42
11

MAX-heap:
                          42
                        /    \
                      24       26
                     /  \      /  \
                   11    2    4    7
                  /
                 9

Після HeapSort (зростання): [ 2 4 7 9 11 24 26 42 ]
Після HeapSort (спадання): [ 42 26 24 11 9 7 4 2 ]
stanislav@fedora:~/Desktop/Algoritms&Data_structure/Lab_05_Heaps/build$
```

part2

```
=== Частина 2: Черги з пріоритетами ===
Введіть n: 5
Введіть 5 цілих: 20
55
30
20
90

MAX-heap:
          90
         /  \
       55    30
      /  \
    20    20
Максимум: 90
ExtractMax: 90
          55
         /  \
       20    30
      /
    20
Після Insert(100):
            100
           /   \
         55     30
        /  \
      20    20

MIN-heap:
          20
         /  \
       20    30
      /  \
    55    90
Мінімум: 20
ExtractMin: 20
          20
         /  \
       55    30
      /
    90
```

Висновок:

У ході виконання лабораторної роботи я навчився будувати незростаючу (Max Heap) та неспадну (Min-Heap) піраміди, реалізовувати пірамідальне сортування масиву, а також працювати з чергами з пріоритетами на їх основі. Я зрозумів, як працюють основні операції: знаходження найбільшого/найменшого елемента, видалення, збільшення чи зменшення ключа та вставка нового елемента. У процесі виконання завдання я також навчився правильно відображати структуру піраміди у вигляді дерева, що допомогло краще зрозуміти роботу алгоритмів.