Белорусский государственный технологический университет

Факультет информационных технологий

Кафедра программной инженерии

 Реферат

По дисциплине «Основы алгоритмизации и программирования»

На тему «Реализация кучи несколькими способами»

Выполнила:

Студент(ка) 1 курса 7 группы

Подшиваленко Диана Игоревна

Проверил:

Белодед Николай Иванович

2024, Минск

**Содержание**

1. [Предисловие … 3](#_Предисловие)
2. [Структура данных куча … 4](#_1._Структура_данных)
3. [Первый способ реализации… 5](#_2._Первый_способ)
4. [Второй способ реализации… 12](#_3._Второй_способ)

# **Предисловие**

В данном реферате будет рассмотрена такая структура данных, как куча, а также 2 способа ее реализации.

# **1. Структура данных куча**

Двоичная куча представляет собой полное бинарное дерево, для которого выполняется основное свойство кучи: приоритет каждой вершины больше(меньше) приоритетов её потомков.

В простейшем случае приоритет каждой вершины можно считать равным её значению. В таком случае структура называется max-heap, если приоритет вершины больше приоритетов потомков, или min-heap, если приоритет вершины меньше приоритета потомков.

Над кучей можно производить несколько операций:

1) создание кучи

2) добавление элемента

3) удаление элемента

4) поиск минимального и максимального

Примеры использования кучи:

1. Приоритетные очереди: куча используется для реализации приоритетных очередей, где элементы хранятся в порядке убывания (для кучи максимума) или возрастания (для кучи минимума) и доступны для извлечения в порядке приоритета. Это позволяет эффективно находить и извлекать элементы с наивысшим приоритетом.
2. Сортировка кучей (Heap Sort): куча используется в алгоритме сортировки кучей, который основан на принципе построения кучи из данных и последующей поэтапной сортировке элементов. Сортировка кучей является одной из наиболее эффективных сортировок со временем выполнения O(n log n).
3. Алгоритмы поиска с наилучшим приближением: куча используется в алгоритмах поиска с наилучшим приближением (best-first search), таких как алгоритм Дейкстры для поиска кратчайшего пути в графе. Куча позволяет эффективно хранить и обновлять приоритеты узлов во время поиска.
4. Динамическое выделение памяти: куча используется для динамического выделения памяти во время выполнения программы. Это позволяет создавать и управлять объектами переменного размера и продлевать их существование по мере необходимости.
5. Структуры данных: куча может быть использована для реализации других структур данных, таких как бинарные кучи (binary heap), биномиальные кучи (binomial heap), Фибоначчиевы кучи (Fibonacci heap) и другие.

# **2. Первый способ реализации**

В первом примере реализована куча с большим количеством функций.

#include <iostream>

#include <Windows.h>

#include <iomanip>

#include <vector>

#include <string>

#include <regex>

using namespace std;

struct AAA

{

int x;

void print();

};

namespace heap

{

enum CMP

{

LESS = -1, EQUAL = 0, GREAT = 1

};

struct Heap

{

int size;

int maxSize;

void\*\* storage;

CMP(\*compare)(void\*, void\*);

Heap(int maxsize, CMP(\*f)(void\*, void\*))

{

size = 0;

storage = new void\* [maxSize = maxsize];

compare = f;

};

int left(int ix);

int right(int ix);

int parent(int ix);

bool isFull() const

{

return (size >= maxSize);

};

bool isEmpty() const

{

return (size <= 0);

};

bool isLess(void\* x1, void\* x2) const

{

return compare(x1, x2) == LESS;

};

bool isGreat(void\* x1, void\* x2) const

{

return compare(x1, x2) == GREAT;

};

bool isEqual(void\* x1, void\* x2) const

{

return compare(x1, x2) == EQUAL;

};

void swap(int i, int j);

void heapify(int ix);

void insert(void\* x, int);

void\* extractMax();

void extractMin(heap::Heap&, heap::Heap&);

void\* extractI(int);

void scan(int i) const;

void unionHeap(heap::Heap, heap::Heap&);

};

Heap create(int maxsize, CMP(\*f)(void\*, void\*));

};

void AAA::print()

{

std::cout << x;

}

namespace heap

{

int pozmin = -100000, minnum = 10000000;

Heap create(int maxsize, CMP(\*f)(void\*, void\*))

{

return \*(new Heap(maxsize, f));

}

int Heap::left(int ix)

{

return (2 \* ix + 1 >= size) ? -1 : (2 \* ix + 1);

}

int Heap::right(int ix)

{

return (2 \* ix + 2 >= size) ? -1 : (2 \* ix + 2);

}

int Heap::parent(int ix)

{

return (ix + 1) / 2 - 1;

}

void Heap::swap(int i, int j)

{

void\* buf = storage[i];

storage[i] = storage[j];

storage[j] = buf;

}

void Heap::heapify(int ix)

{

int l = left(ix), r = right(ix), irl = ix;

if (l > 0)

{

if (isGreat(storage[l], storage[ix])) irl = l;

if (r > 0 && isGreat(storage[r], storage[irl])) irl = r;

if (irl != ix)

{

swap(ix, irl);

heapify(irl);

}

}

}

void Heap::insert(void\* x, int k)

{

int i;

if (!isFull())

{

storage[i = ++size - 1] = x;

while (i > 0 && isLess(storage[parent(i)], storage[i]))

{

swap(parent(i), i);

i = parent(i);

}

}

}

void\* Heap::extractMax()

{

void\* rc = nullptr;

if (!isEmpty())

{

rc = storage[0];

storage[0] = storage[size - 1];

size--;

heapify(0);

} return rc;

}

void Heap::extractMin(heap::Heap& h3, heap::Heap& h1)

{

if (h1.size == 0) {

cout << "Куча пуста\n";

return;

}

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

#define A1 ((AAA\*)h1.storage[i])

if (A1->x < minnum) {

minnum = A1->x;

pozmin = i;

}

}

storage[pozmin] = storage[size - 1];

h1.size--;

if (!isEmpty())

{

while (!h1.isEmpty()) {

#define A1 ((AAA\*)h1.storage[0])

h3.insert(A1, A1->x);

h1.extractMax();

}

}

pozmin = -100000, minnum = 10000000;

}

void\* Heap::extractI(int h)

{

void\* rc = nullptr;

if (h >= size || h < 0) {

cout << "Некорректный номер\n";

return nullptr;

}

if (!isEmpty())

{

rc = storage[h];

if (h != size - 1) {

swap(h, size - 1);

}

size--;

heapify(0);

heapify(h);

} return rc;

}

void Heap::scan(int i) const //Вывод значений элементов на экран

{

vector<int> a = { 52 };

vector<int> b = { 26, 24, 28 };

vector<int> c = { 16, 10, 16, 8, 16, 12, 16 };

vector<int> d = { 8, 8, 6, 6, 6, 8, 6, 4, 6, 8, 8, 4, 8, 8, 8 };

vector<int> e = { 4, 4, 4, 4, 4, 4, 3, 4, 3, 3, 3, 4, 3, 3, 3, 3, 3, 3, 3, 3, 4, 3, 3, 4, 3, 3, 4, 4, 4, 4, 4 };

vector<int> vnesh = { 66, 40, 28, 20, 16, 12 };

int y = 0;

int level = 0;

bool t = false;

int current = 0;

int j;

std::cout << '\n';

if (size == 0) {

std::cout << "Куча пуста\n";

return;

}

std::cout << std::setw(vnesh[level]) << std::setfill(' ');

level++;

((AAA\*)storage[current])->print();

current++;

cout << '\n';

y = 2;

for (int i = 0; current < size; i++) {

if (!t) {

std::cout << std::setw(vnesh[level]) << std::setfill(' ');

t = true;

j = 0;

}

((AAA\*)storage[current])->print();

if (level == 1 && j < a.size()) {

std::cout << std::setw(a[j]) << std::setfill(' ');

j++;

}

else if (level == 2 && j < b.size()) {

std::cout << std::setw(b[j]) << std::setfill(' ');

j++;

}

else if (level == 3 && j < c.size()) {

std::cout << std::setw(c[j]) << std::setfill(' ');

j++;

}

else if (level == 4 && j < d.size()) {

std::cout << std::setw(d[j]) << std::setfill(' ');

j++;

}

else if (level == 5 && j < e.size()) {

std::cout << std::setw(e[j]) << std::setfill(' ');

j++;

}

current++;

if (current > y) {

cout << '\n';

level++;

y += pow(2, level);

t = false;

}

}

cout << '\n';

}

void Heap::unionHeap(heap::Heap buf, heap::Heap& h1) {

while (!buf.isEmpty()) {

#define A1 ((AAA\*)buf.storage[0])

h1.insert(A1, A1->x);

buf.extractMax();

}

}

}

regex checknum("[\\-]?[0-9]+");

heap::CMP cmpAAA(void\* a1, void\* a2) //Функция сравнения

{

#define A1 ((AAA\*)a1)

#define A2 ((AAA\*)a2)

heap::CMP

rc = heap::EQUAL;

if (A1->x > A2->x)

rc = heap::GREAT;

else

if (A2->x > A1->x)

rc = heap::LESS;

return rc;

#undef A2

#undef A1

}

//-------------------------------

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int k, choice;

string b;

heap::Heap h1 = heap::create(100, cmpAAA);

for (;;)

{

cout << "1 - вывод кучи на экран\n";

cout << "2 - добавить элемент\n";

cout << "3 - удалить максимальный элемент\n";

cout << "4 - удалить минимальный элемент\n";

cout << "5 - удалить элемент по номеру\n";

cout << "6 - создать дополнительную кучу и объединить с основной\n";

cout << "0 - выход\n";

cout << "сделайте выбор\n";

getline(cin, b);

while (b.length() != 1) {

cout << "Некорректный ввод\n";

getline(cin, b);

}

choice = b[0];

switch (choice)

{

case '0': exit(0);

case '1': {

h1.scan(0);

break;

}

case '2': { AAA\* a = new AAA;

cout << "введите ключ\n";

getline(cin, b);

while (!regex\_match(b, checknum)) {

cout << "Некорректный ввод\n";

getline(cin, b);

}

k = stoi(b);

a->x = k;

h1.insert(a, k);

break;

}

case '3': {

h1.extractMax();

break;

}

case '4': {

heap::Heap h3 = heap::create(100, cmpAAA);

h1.extractMin(h3, h1);

h1 = h3;

break;

}

case '5': {

int n;

cout << "Введите номер элемента, который хотите удалить\n";

getline(cin, b);

while (!regex\_match(b, checknum)) {

cout << "Некорректный ввод\n";

getline(cin, b);

}

n = stoi(b);

h1.extractI(n);

break;

}

case '6': {

cout << "Вы выбрали опцию объединения двух куч. Для начала нужно создать дополнительную кучу\n";

int choice2;

heap::Heap h2 = heap::create(30, cmpAAA);

for (;;)

{

cout << "1 - вывод кучи на экран\n";

cout << "2 - добавить элемент\n";

cout << "3 - удалить максимальный элемент\n";

cout << "4 - удалить минимальный элемент\n";

cout << "5 - удалить элемент по номеру\n";

cout << "6 - объединить кучи и вернуться в главное меню\n";

cout << "сделайте выбор\n";

getline(cin, b);

while (b.length() != 1) {

cout << "Некорректный ввод\n";

getline(cin, b);

}

choice2 = b[0];

switch (choice2)

{

case '1': {

h2.scan(0);

break;

}

case '2': { AAA\* a = new AAA;

cout << "введите ключ\n";

getline(cin, b);

while (!regex\_match(b, checknum)) {

cout << "Некорректный ввод\n";

getline(cin, b);

}

k = stoi(b);

a->x = k;

h2.insert(a, k);

break;

}

case '3': {

h2.extractMax();

break;

}

case '4': {

heap::Heap h3 = heap::create(100, cmpAAA);

h2.extractMin(h3, h2);

h2 = h3;

break;

}

case '5': {

int n;

cout << "Введите номер элемента, который хотите удалить\n";

getline(cin, b);

while (!regex\_match(b, checknum)) {

cout << "Некорректный ввод\n";

getline(cin, b);

}

n = stoi(b);

h2.extractI(n);

break;

}

case '6': {

h1.unionHeap(h2, h1);

break;

}

default: {

cout << "Некорректный ввод\n";

}

}

if (choice2 == '6') {

break;

}

}

break;

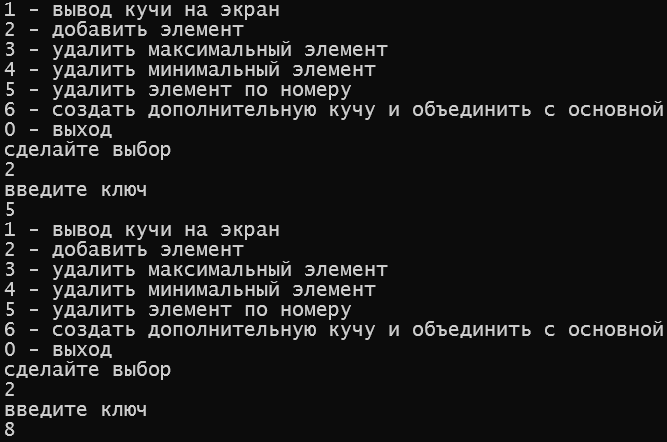
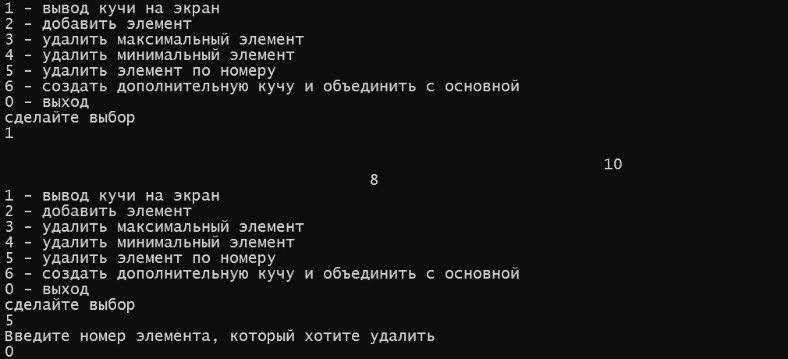
}

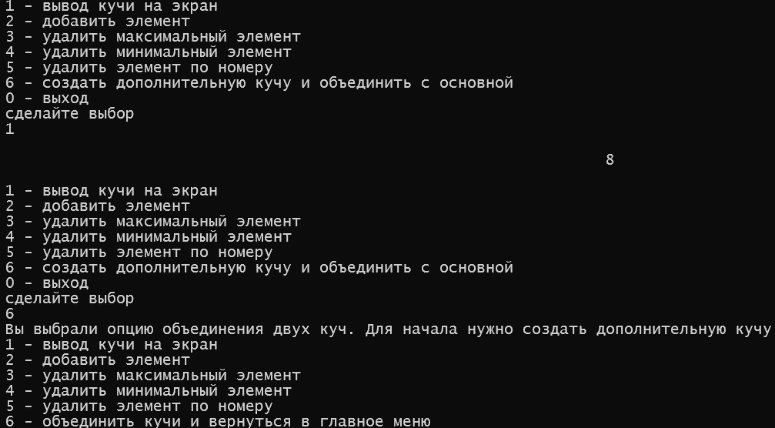
default: cout << "\nВведена неверная команда!\n";

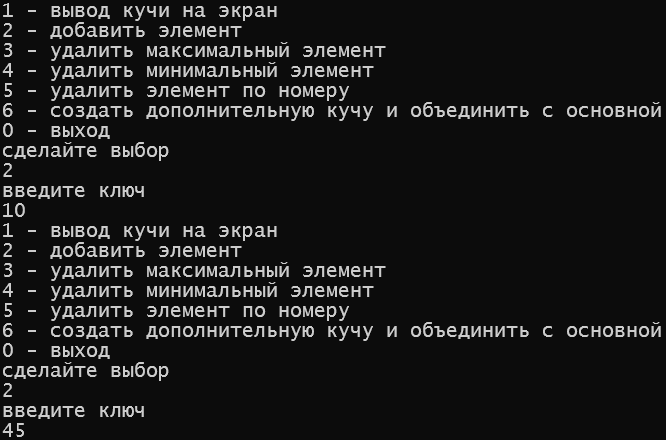
}

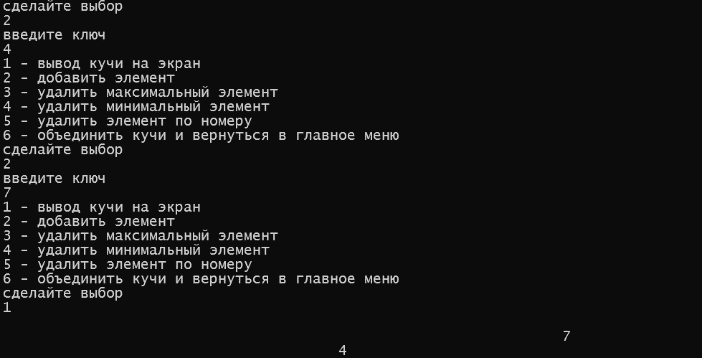
} return 0;

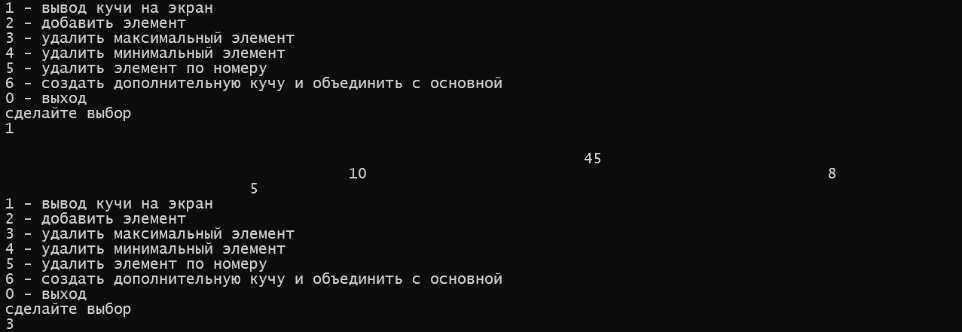
}

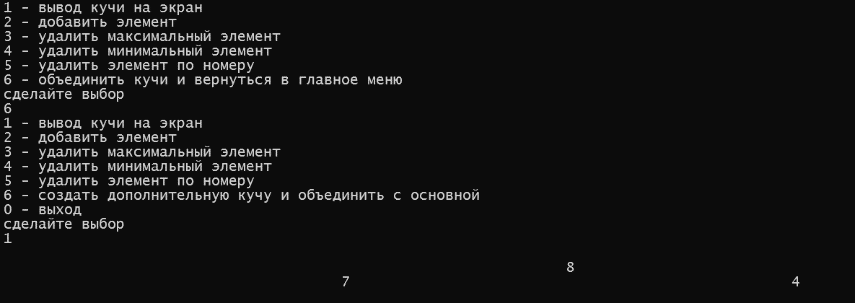
Результат выполнения:

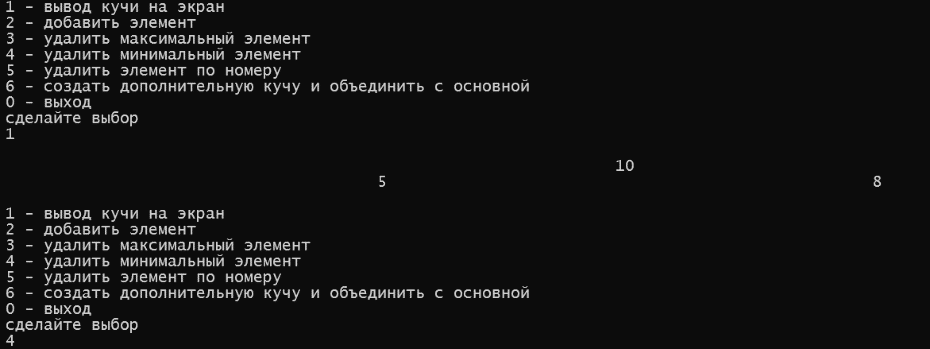












# **3. Второй способ реализации**

#include <iostream>

#include <string>

#include <regex>

#include <Windows.h>

using namespace std;

class Heap {

static const int SIZE = 100; // максимальный размер кучи

int\* h; // указатель на массив кучи

int HeapSize; // размер кучи

public:

void addelem(int); // добавление элемента кучи

void outHeap(); // вывод элементов кучи в форме кучи

int getmax(); // удаление вершины (максимального элемента)

void heapify(int); // упорядочение кучи

Heap();

};

Heap::Heap() {

h = new int[SIZE];

HeapSize = 0;

}

void Heap::addelem(int n) {

int i, parent;

i = HeapSize;

h[i] = n;

parent = (i - 1) / 2;

while (parent >= 0 && i > 0) {

if (h[i] > h[parent]) {

int temp = h[i];

h[i] = h[parent];

h[parent] = temp;

}

i = parent;

parent = (i - 1) / 2;

}

HeapSize++;

}

void Heap::outHeap(void) {

int i = 0;

int k = 1;

while (i < HeapSize) {

while ((i < k) && (i < HeapSize)) {

cout << h[i] << " ";

i++;

}

cout << endl;

k = k \* 2 + 1;

}

}

void Heap::heapify(int i) {

int left, right;

int temp;

left = 2 \* i + 1;

right = 2 \* i + 2;

if (left < HeapSize) {

if (h[i] < h[left]) {

temp = h[i];

h[i] = h[left];

h[left] = temp;

heapify(left);

}

}

if (right < HeapSize) {

if (h[i] < h[right]) {

temp = h[i];

h[i] = h[right];

h[right] = temp;

heapify(right);

}

}

}

int Heap::getmax(void) {

int x;

x = h[0];

h[0] = h[HeapSize - 1];

HeapSize--;

heapify(0);

return(x);

}

regex checknum("[\\-]?[0-9]+");

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

Heap heap;

int k, choice;

string b;

for (;;)

{

cout << "1 - вывод кучи на экран\n";

cout << "2 - добавить элемент\n";

cout << "3 - удалить максимальный элемент\n";

cout << "0 - выход\n";

cout << "сделайте выбор\n";

getline(cin, b);

while (b.length() != 1) {

cout << "Некорректный ввод\n";

getline(cin, b);

}

choice = b[0];

switch (choice)

{

case '0': exit(0);

case '1': {

heap.outHeap();

break;

}

case '2': {

cout << "введите ключ\n";

getline(cin, b);

while (!regex\_match(b, checknum)) {

cout << "Некорректный ввод\n";

getline(cin, b);

}

k = stoi(b);

heap.addelem(k);

break;

}

case '3': {

heap.getmax();

break;

}

default: cout << "\nВведена неверная команда!\n";

}

}

return 0;

}

Результат выполнения:

