Контрольная работа

по курсу «Алгоритмы и структуры данных»

студентки группы ПА-20-1з

Мовсисян Лауры Ростомовны

кафедра компьютерных технологий, ДНУ

2021/2022уч. Год

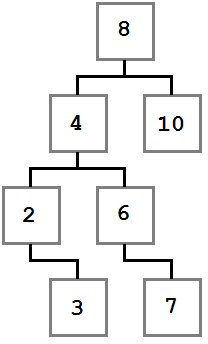
**Теоритическая часть:**

**2.Двійкрве дерево пошуку**

Двійкове дерево пошуку схоже на дерево з прикладу вище, але будується за певними правилами:

* у кожного вузла не більше двох дітей;
* будь-яке значення менше значення вузла стає лівою дитиною або дитиною лівої дитини;
* будь-яке значення більше або рівне значенню вузла стає правою дитиною або дитиною правої дитини.

Давайте подивимося на дерево, побудоване за цими правилами:



Зверніть увагу на те, як вказані обмеження впливають на структуру дерева. Кожне значення ліворуч від кореня “8” менше восьми, кожне значення праворуч – більше або дорівнює кореню. Це правило застосоване до будь-якого вузла дерева.

Враховуючи це, давайте уявимо, як можна побудувати таке дерево. Оскільки спочатку дерево було порожнім, перше додане значення – вісім – стало його коренем.

Ми не знаємо точно, в якому порядку додавалися інші значення, але можемо представити один з можливих шляхів. Вузли додаються методом Add, який набуває значення, що додається.

Розглянемо детальніше перші кроки.

У першу чергу додається 8. Це значення стає коренем дерева. Потім ми додаємо 4. Оскільки 4 менше 8, ми кладемо її в ліву дитину, згідно з правилом 2. Оскільки у вузла з вісімкою немає дітей ліворуч, 4 стає єдиною лівою дитиною.

Після цього ми додаємо 2. 2 менше 8, тому рухаємось ліворуч. Оскільки ліворуч вже є значення, порівнюємо його з тим, що вставляється. 2 менше 4, а у четвірки немає дітей ліворуч, тому 2 стає лівою дитиною 4.

Потім ми додаємо трійку. Вона розташовується ліворуч 8 і 4. Оскільки 3 більше, ніж 2, то вона стає правою дитиною 2, згідно з третім правилом.

Послідовне порівняння значення, що вставляється, з потенційним батьком триває до тих пір, поки не буде знайдено місце для вставки, і повторюється для кожного значення, що вставляється, до тих пір, поки не буде побудовано все дерево.

**Клас BinaryTreeNode**

Клас BinaryTreeNode представляє один вузол двійкового дерева. Він утримує посилання на лівому і правому піддереві (якщо піддерева немає, посилання має значення null), дані вузла і метод IComparable.CompareTo для порівняння вузлів. Він згодиться для визначення, в яке поддерево повинен йти цей вузол. Як бачите, клас BinaryTreeNode дуже простий:

#### Клас BinaryTree

Клас BinaryTree надає основні методи для маніпуляцій з даними: вставка елемента (Add), видалення (Remove), метод Contains для перевірки, чи є таке значення в дереві, декілька методів для обходу дерева різними способами, метод Count і Clear.

Крім того, в класі є посилання на кореневий вузол дерева і поле із загальною кількістю вузлів.

**Метод Add**

* **Поведінка:** додає елемент у дерево на коректну позицію.
* **Складність:** O (log n) в середньому; O (n) у гіршому разі.

Додавання вузла не становить складності. Воно стає ще простіше, якщо розв’язувати цю задачу рекурсивно. Є всього два випадки, які потрібно врахувати:

1. Дерево порожнє.
2. Дерево не порожнє.

Якщо дерево порожнє, ми просто створюємо новий вузол і додаємо його в дерево. У другому випадку ми порівнюємо передане значення зі значенням у вузлі, починаючи від кореня. Якщо значення, що додається, менше значення даного вузла, повторюємо ту саму процедуру для лівого піддерева, ящо ні, то для правого.

**Метод Remove**

* **Поведінка:** видаляє перший вузол із заданим значенням.
* **Складність:** O (log n) в середньому; O (n) у гіршому разі.

Видалення вузла з дерева – одна з тих операцій, які здаються простими, але насправді приховують у собі немало підводних каменів.

У цілому, алгоритм видалення елемента виглядає так:

* Знайти вузол, який потрібно видалити.
* Видалити його.

Перший крок простий. Ми розглянемо пошук вузла в методі Contains нижче. Після того, як ми знайшли вузол, який необхідно видалити, у нас можливі три випадки.

**Метод Contains**

* **Поведінка:** повертає true , якщо значення міститься в дереві. Якщо ні, то повертає false.
* **Складність:** O (log n) в середньому; O (n) у гіршому разі.

Метод Contains виконується за допомогою методу FindWithParent, який проходить деревом, виконуючи в кожному вузлі наступні кроки:

1. Якщо поточний вузол null, повернути null.
2. Якщо значення поточного вузла дорівнює шуканому, повернути поточний вузол.
3. Якщо шукане значення менше значення поточного вузла, встановити ліву дитину поточним вузлом і перейти до кроку 1.
4. Якщо ні, то встановити праву дитину поточним вузлом і перейти до кроку 1.

Оскільки Contains повертає нульове значення, воно визначається порівнянням результату виконання FindWithParent з null. Якщо FindWithParent повернув непорожній вузол, Contains повертає true.

**Метод Count**

* **Поведінка:** повертає кількість вузлів дерева або 0, якщо дерево порожнє.
* **Складність:** O (1)

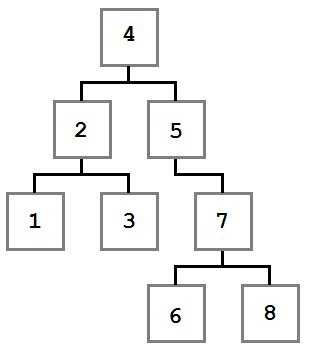
Це поле інкрементується методом Add і декрементується методом Remove.

**Метод Clear**

* **Поведінка:** видаляє всі вузли дерева.
* **Складність:** O (1)

### Обхід дерев

Обходи дерева – це сімейство алгоритмів, які дозволяють обробити кожен вузол у певному порядку. Для всіх алгоритмів обходу нижче як приклад використовуватиметься наступне дерево:

 Методи обходу в прикладах прийматимуть параметр Action, який визначає дію, виконувану з кожним вузлом.

**Метод Preorder (префіксний обхід)**

* **Поведінка:** обходить дерево в префіксному порядку, виконуючи вказану дію з кожним вузлом.
* **Складність:** O (n)
* **Порядок обходу :** 4, 2, 1, 3, 5, 7, 6, 8

За префіксного обходу алгоритм набуває значення поточного вузла перед тим, як перейти спочатку в ліве поддерево, а потім у праве. Починаючи від кореня, спочатку ми отримаємо значення 4. Потім так само обходяться ліва дитина та її діти, потім права дитина і всі її діти.

Префіксний обхід зазвичай застосовується для копіювання дерева зі збереженням його структури.

**Метод Postorder (постфіксний обхід)**

* **Поведінка:** обходить дерево в префіксному порядку, виконуючи вказану дію з кожним вузлом.
* **Складність:** O (n)
* **Порядок обходу :** 1, 3, 2, 6, 8, 7, 5, 4

За постфіксного обходу ми відвідуємо ліве піддерево, праве піддерево, а потім, після обходу всіх дітей, переходимо до самого вузла.

Постфіксний обхід часто використовується для повного видалення дерева, оскільки в деяких мовах програмування необхідно прибирати з пам’яті всі вузли явно, або для видалення піддерева. Оскільки корінь у даному випадку обробляється останнім, ми, таким чином, зменшуємо роботу, необхідну для видалення вузлів.

**Метод Inorder (інфіксний обхід)**

* **Поведінка:** обходить дерево в інфіксному порядку, виконуючи вказану дію над кожним вузлом.
* **Складність:** O (n)
* **Порядок обходу :** 1, 2, 3, 4, 5, 6, 7, 8

Інфіксний обхід використовується тоді, коли нам потрібно обійти дерево в порядку, що відповідає значенням вузлів. У прикладі вище в дереві знаходяться числові значення, тому ми обходимо їх від найменшого до найбільшого (від лівих піддерев до правих через корінь).

**Метод GetEnumerator**

* **Поведінка:** повертає ітератор для обходу дерева інфіксним способом.
* **Складність:** отримання ітератора – O (1). Обхід дерева – O (n).

Практическая часть:

Лабораторная работа №2 "Моделирование работы стека и очерди "

Вариант 8

1. Постановка задачи:

На мові С++ розробити програму, в якій:

− реалізувати лінійну структуру даних, необхідну для виконання

індивідуального завдання згідно з варіантом – стек або чергу, на основі

статичного масиву (для черги масив має бути згорнутий у кільце) з типом

елементів, які використовуються у завданні;

− реалізувати основні та додаткові операції для розробленої структури;

− реалізувати індивідуальне завдання з використанням інтерфейсу

розробленої структури даних.

***Обов’язкові вимоги.***

− Після завантаження програма пропонує користувачеві меню для вибору

дії з такими пунктами:

1. Додати елемент до *структури даних*.

2. Видалити елемент зі *структури даних*.

3. Вивести елементи *структури* на екран.

4. Вивести кількість елементів *структури*. Для черги – індекси першого

та останнього елементів (голови та хвоста).

5. Очистити структуру.

6. *Пункти щодо індивідуальних завдань згідно з варіантом*.

7. Завершення роботи.

В пунктах меню зазначати конкретну структуру даних, яка реалізується. Приклад

програмної організації текстового меню наведено у лабораторній роботі № 1.

− Організувати роботу програми таким чином, щоб перелічені вище дії можна

було б виконувати неодноразово за один сеанс роботи програми.

− При додаванні елементів передбачити обробку помилкових ситуацій, коли

структура даних вже заповнена і додавання неможливе, з друком відповідного

повідомлення.

− При видаленні елементів передбачити обробку помилкових ситуацій, коли

будуть виконуватися спроби видалення елемента із порожньої структури, з

друком відповідного повідомлення.

− Структуру даних реалізувати на базі статичного масиву із *N* елементів (*N*

задати як константу).

− Операції над структурою даних реалізувати у вигляді окремих функцій

інтерфейсу.

− Індивідуальне завдання за варіантом – окрема функції, у якій створити

екземпляр необхідної структури даних (стеку або черги) та організувати

розв’язання задачі з використанням функцій інтерфейсу цієї структури.

− У звіті навести блок-схему тільки для алгоритму індивідуального завдання.

2.Опис алгоритму індивідуального завдання

Реализовать очередь на базе статического массива. Реализовать основные методы очереди. Так же сделать возможность автозаполнения массива случайными элементами из диапазона. Среди положительных элементов массива вывести сразу чётные, потом нечётные.

1. Опис структури програми та особливостей реалізації

Разбил задачи на функции, которые вызываются с меню. Функции с задачами вызываются через Switch-caseпо соответствующему номеру задачи. Реализовал очередь с основными её методами. В задаче про положительные числа шёл по массиву, выводя чётные, а нечётные перенося в очередь. Таким образом нечётные элементы выводились после чётных.

1. Исходный текст программы

#include<iostream>

#include<ctime>

usingnamespace std;

void Menu();

structQueue

{

private:

constint n\_max = 250;

int elem[250];

public:

int n;

Queue()

{

n = 0;

}

void AddElem(intelem)

{

if (n < 250)

{

n++;

this->elem[n - 1] = elem;

}

}

int TakeElem()

{

if (n > 0)

{

int out = elem[0];

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

elem[i] = elem[i + 1];

n--;

return out;

}

cout <<"Ошибка!"<< endl;

return -1;

}

void Clear()

{

n = 0;

}

void ShowQueue()

{

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

{

int buff = TakeElem();

cout << buff << endl;

AddElem(buff);

}

}

};

Queue qu;

int mass[20];

int mass\_count = 0;

int main()

{

srand(time(0));

setlocale(LC\_ALL, "Russian");

srand(time(0));

Menu();

system("Pause");

}

void MassInit()

{

int a, b;

cout <<"Введите размер массива (0, 20]";

cin >> mass\_count;

if (mass\_count <= 0 && mass\_count > 20)

{

cout <<"Ошибкаввода"<< endl;

return;

}

cout <<"Введите диапазон случайных чисел [a, b] && (a, b) > 0"<< endl;

cout <<"Введите a - ";

cin >> a;

cout << endl;

cout <<"Введите b - ";

cin >> b;

if (a < 0 || b < 0)

{

cout <<"Ошибкаввода"<< endl;

return;

}

if (a > b) swap(a, b);

for (int i = 0; i < 20; i++)

mass[i] = rand() % (b - a) + a;

}

void Menu()

{

bool menu\_i = true;

while (menu\_i)

{

system("cls");

cout <<"1) Добавить элемент"<< endl;

cout <<"2) Видалити элемент"<< endl;

cout <<"3) Вивести элементы на экран"<< endl;

cout <<"4) Очистить структуру"<< endl;

cout <<"5) Вивести кол-во элементов, индекс первого и последнего элемента"<< endl;

cout <<"6) Сгенерировать случайный массив"<< endl;

cout <<"7) Парные - непарные"<< endl;

cout <<"8) Выход"<< endl;

cout << endl;

cout <<"Введите номер пункта, к которому хотите перейти - ";

int n\_; cin >> n\_; cout << endl;

switch (n\_)

{

case 1:

cout <<"Введитеэлемент"<< endl;

int el; cin >> el; cout << endl;

qu.AddElem(el);

system("Pause");

break;

case 2:

qu.TakeElem();

system("Pause");

break;

case 3:

if (qu.n != 0)

qu.ShowQueue();

system("Pause");

break;

case 4:

qu.Clear();

system("Pause");

break;

case 5:

cout <<"Вмассиве "<< mass\_count <<" элементов"<< endl;

if (mass\_count != 0)

cout <<"Индекспервого - 0, апоследнего - "<< mass\_count - 1 << endl;

else cout <<"Индекс первого - 0, а последнего - 0"<< endl;

system("Pause");

break;

case 6:

MassInit();

system("Pause");

break;

case 7:

{

for (int i = 0; i < mass\_count; i++)

cout << mass[i] << endl;

cout << endl;

int non\_par = 0;

for (int i = 0; i < mass\_count; i++)

{

if (mass[i] != 0)

{

if (mass[i] % 2 == 0)

cout << mass[i] << endl;

else

{

qu.AddElem(mass[i]);

non\_par++;

}

}

}

for (int i = 0; i < non\_par; i++)

cout << qu.TakeElem() << endl;

system("Pause");

cout << endl;

break;

}

case 8:

menu\_i = 0;

break;

default:

menu\_i = 0;

break;

}

}

}

1. Опис интерфейсу

При запуске программы выводится меню. Чтобы перейти к пункту меню, необходимо ввести номер пункта с клавиатуры и нажать ENTER. Дальше следовать инструкциям.

5. Тестові приклади (скріншоти ) за кожним пунктом меню

Вводил разные значения, после чего добавлял ограничения на вводимые данные. Если данные не соответсвовали требуемым, консоль очищалась, выводилось сообщение об ошибке ввода. После этого подпрограмма перезапускается.

