Київський національний університет імені Тараса Шевченка

Факультет комп`ютерних наук та кібернетики

Кафедра інтелектуальних програмних систем

Алгоритми та складність

Завдання №3

“Розширюване дерево ”

Виконав студент 2-го курсу

Групи К-29

Жутовський Глiб Євгенович

2021

**Предметна область**

Предметна область: Навчальний відділ

Об’єкти:  Групи, Студенти

Примітка: Існує множина навчальних груп. Кожна група включає в себе множину студентів.

**Завдання**

Реалізувати розширюване дерево. У моєму випадку елементами дерева буде інформація про групи.

**Теорія**

Розширювані дерево - це двійкове дерево пошуку з підтримкою збалансованості.

Має наступні властивості:

• Не потребує додаткових полів у вузлі.

• Явні функції балансування відсутні.

• При кожному звертанні до дерева виконується

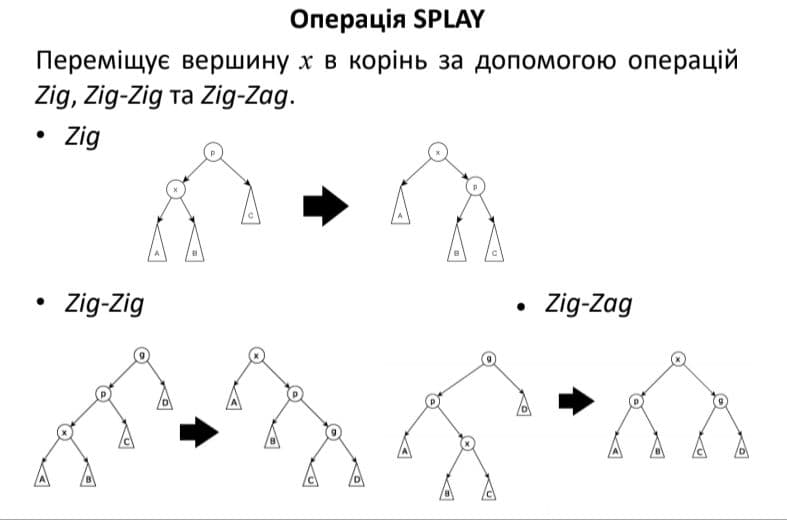
«операція розширення» (splay operation).

• В результаті вузли, до яких звертаються частіше,

зберігаються ближче до кореня, а до яких рідше –

ближче до листків.

Дамо визначення операціям над розширювальним деревом **splay**,**merge** та **split**



**Merge** (об'єднання двох дерев). Для злиття дерев T1 і T2, в яких всі ключі T1 менше ключів в T2, робимо Splay для максимального елементу T1, тоді біля кореня T1 не буде правого дочірнього елемента. Після цього робимо T2 правим дочірнім елементом T1.

**Split** (розділення дерева на дві частини). Для розділення дерева знаходиться найменший елемент, більший або рівний x і для нього робиться Splay. Після цього відрізаємо ліве піддерево у якості другого дерева.

**Алгоритми**

**• Search (пошук елемента**). Спочатку звичайний пошук. При знаходженні елементу запускаємо Splay для нього.

**• Insert (додавання елемента).** Запускаємо Split від елементу, що додається, і підвішуємо дерева, що вийшли, за нього.

**• Delete (видалення елемента).** Знаходимо елемент в дереві, робимо Splay для нього, робимо поточним деревом Merge його дітей.

**Складність**

Всі операції потребують часу в середньому О(lg n).

**Мова програмування**

С++

**Модулі програми**

//Клас вузла дерева

template<typename T>

class SplayTreeNode

{

public:

//Конструктор

SplayTreeNode(T dataArg);

SplayTreeNode\* left;

SplayTreeNode\* right;

SplayTreeNode\* parent;

T data;

};

//Розширюване дерево

template<typename T>

class SplayTree

{

private:

SplayTreeNode<T>\* root\_;

//Функція розширення

void Splay(SplayTreeNode<T>\* node) ;

//Правий оберт

void RightRotate(SplayTreeNode<T>\* node);

//Лівий оберт

void LeftRotate(SplayTreeNode<T>\* node);

//Пошук найменший елементу лівого піддерева

SplayTreeNode<T>\* LeftSubTreeMin(SplayTreeNode<T>\* localRoot) const;

//Пошук найбільшого елементу правого піддерева

SplayTreeNode<T>\* RightSubTreeMax(SplayTreeNode<T>\* localRoot) const;

//Пошук найбільшого та найменшого в обох піддеревах

SplayTreeNode<T>\* Successor(SplayTreeNode<T>\* node) const;

SplayTreeNode<T>\* Predecessor(SplayTreeNode<T>\* node) const ;

//Пошук елементу

SplayTreeNode<T>\* Search(T& key);

//Функція обміну

void Swap(SplayTreeNode<T>\* where, SplayTreeNode<T>\* what);

//Інтерактивний пошук

SplayTreeNode<T>\* ChooseNode(SplayTreeNode<T>\* current\_node) const;

//Вивід в консоль

void PrintNode(SplayTreeNode<T>\* curr);

public:

//Конструктор

SplayTree();

//Додавання елементу

void Insert(T dataArg);

//Видалення

void Remove(T& key);

//Пошук найменшого елементу у дереві

T Min() const;

//Пошук найбільшого елементу у дереві

T Max() const;

//Перевірка чи у дереві є елемент

bool At(T& key);

//Функція злиття дерев

void Merge(SplayTree<T>& tree);

//Функція відщеплення гілки дерева

SplayTree<T> split(T& key);

//Функції виводу

void print() const;

void Print();

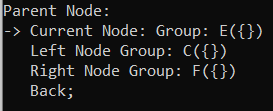
};

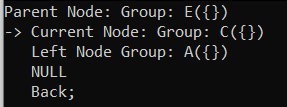
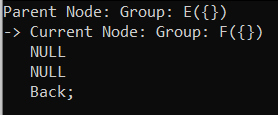
**Інтерфейс користувача**

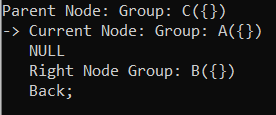
Вхідні дані вводяться з консолі користувачем і виводяться в консоль

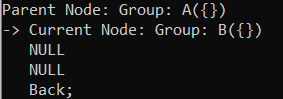
**Приклад виводу програми**

Вважаємо перше дерево яке зображене в консолі є вхідними даними користувача

****





**Тестовий приклад**

**операція insert ({D, {}})** :

Початкове дерево Після вставки

E

A

F

B

C

E

A

F

B

C

D

Зобразимо наше дерево так:

E

D

E

C

Zig-Zag

C

D

4

1

4

2

1

3

3

2

Кінцеве дерево:

D

E

A

F

B

C

**Висновки**

Розширювальне дерево є досить ефективним для бінарного пошуку оскільки при кожному звернені до його елементу він розташовується ближче до кореня, тобто елементи дерева, до яких найчастіше звертаються, буде швидше знайти в дереві і до того ж усі операції над розширювальним деревом займають час в середньому О(lg n), що досить швидко.

**Література**

* Лекція № 3
* [Splay -tree — Вікіпедія (wikipedia.org)](https://uk.wikipedia.org/wiki/%D0%A5%D0%B5%D1%88-%D1%84%D1%83%D0%BD%D0%BA%D1%86%D1%96%D1%8F)
* https://www.geeksforgeeks.org/splay-tree-set-1-insert/