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-28
Гуща Дмитро Сергійович

2021

Предметна область

Варіант 4
Предметна область: Учбовий відділ
Об’єкти: Групи, Студенти
Примітка: Маємо множину учбових груп. Кожна група містить в собі множину студентів

Завдання

Реалізувати розширюване дерево

Теорія та Алгоритм

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, берігаються ближче до кореня, а до яких рідше – ближче до листків.

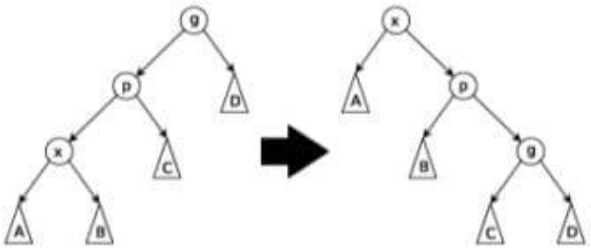
Дамо визначення операціям над розширювальним деревом **splay**,**merge** та **split**

Операція SPLAY

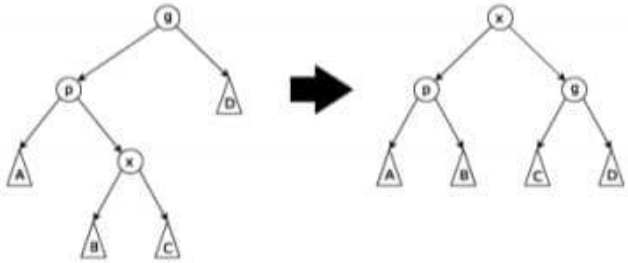
Переміщує вершину x в корінь за допомогою операцій *Zig*, *Zig-Zig* та *Zig-Zag*.

- *Zig*

- *Zig-Zig*



- *Zig-Zag*



Операція SPLAY

Переміщує вершину x в корінь за допомогою операцій: Zig, Zig-Zig та Zig-Zag.

• Zig: виконується, коли p є коренем. Дерево повертається по ребру між x і p. Існує лише для розбору крайнього випадку і виконується тільки один раз в кінці, коли початкова глибина x була непарна.

• Zig-Zig: виконується, коли і x, і p є лівими (або правими) синами. Дерево повертається по ребру між g і p, а потім - по ребру між p і x.

• Zig-Zag: виконується, коли x є правим сином, а p - лівим (або навпаки). Дерево повертається по ребру між p і x, а потім - по ребру між x і g.

1) 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.

2) 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.

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

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

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

Складність

Вартість будь-якої операції на розширюваному дереві становить O(lg n), де n – це кількість вузлів у дереві.

Мова програмування

C++

Модулі програми

student.h

```
class Student{}; //Класс опису студента
std::string getName(); // метод повертає ім’я студента
void getStudent();//метод виводить ID та ім'я студента в консоль
void setName(std::string name); //метод змінює ім'я студента
```

group.h

```
class Group {};
Group() : title("NULL"); //конструктор пустої групи
Group(std::string title); //конструктор з початковою назвою групи
Group(std::string title, Student* first_student); //конструктор з початковою назвою
групи та першим студентом
std::string getGroupTitle(); //модуль повертає назву групи
std::vector<Student*> getGroupStudents(); //модуль повертає множину студентів
void setGroupTitle(std::string title); //модуль змінює назву групи
void setGroupStudents(std::vector<Student*> students); //модуль змінює множину
студентів
void addStudent(Student* student); //додати нового студента
void printStudents(); //вивід у консоль усіх студентів групи
```

splayTree.h

```
struct Node {}; // структура вузла
class SplayTree {}; // клас Розширюваного дерева
```

```
void zig(Node* node); // операція переміщення вершини в корінь
void zig_zig(Node* node); // операція переміщення вершини в корінь
void zig_zag(Node* node); // операція переміщення вершини в корінь
void splay(Node* node); // реалізація алгоритму Splay
SplayTree(); // пустий конструктор классу розширюванного дерева
SplayTree(Node* newNode); // конструктор классу роширюваного дерева який приймає
перший вузол
Node* getRoot(); // повертає вказівник на корінь дерева
Node* find(int quantityOfStudents); // займається пошуком потрібноного вузла
void insertGroup(Group* group); // додавання нової групи
void deleteGroup(int quantityOfStudents); // видалення групи
void inOrderPrint(bool flag); // виведення в консоль всього дерева
Node* subtree_max(Node* subRoot) //вибір більшого серед двух нащадків
Node* subtree_min(Node* subRoot) // вибір меншого серед двух нащадків
void printTree(Node* root, bool brackets) // виведення піддерева в консоль
```

main.cpp

```
void manu(); //модуль відповідає за вибір подальших дій
void inputGroupAndStudents(Group& newGroup); //функція додавання нової групи та
студентів
void searchAGroup(SplayTree educationalTree); //пошук групи у дереві
void interactiveMode(SplayTree& educationalTree); //інтерактивне використання
программи
void deleteGroup(SplayTree& educationalTree) //видалення групи
int main(); //основна функція программи
```

Інтерфейс користувача

Вхідні дані вводяться в консоль користувачем і виводяться в консоль

Тестовий приклад

Групи в дереві сортуються за кількістю студентів в них.

input	**output**
Insert(group.size(16)); Insert(group.size(21)); Insert(group.size(5)); Insert (group.size(30)); Insert(group.size(28)); Delete(group.size(5));	Splay tree: 16 10 5 21 28 30 Splay tree after deleting an item with a value of 5: 10 16 21 28 30 Splay tree after adding an element with a value of 25:

Insert(group.size(25));	25 16 10 21 28 30

Висновки

Розширювальне дерево є досить ефективним оскільки він не потребує додаткових полів у вузлі, явні функції балансування відсутні, всі операції потребують часу в середньому O(lg n).

Література

- Лекція № 3
- https://ru.wikipedia.org/wiki/Splay-дерево