**Київський національний університет імені Тараса Шевченка**

**Факультет комп’ютерних наук та кібернетики**

**Кафедра інформаційних систем**

**Алгоритми та складність**

**Лабораторна робота №3**

**«Розширюване дерево»**

**ЗВІТ**

**Підготував студент**

**2 курсу**

**Групи К29**

**Опанюк Микита**

**Варіант 7**

**2018 рік**

**Опис предметної області :**

*Варіант 7*   
Предметная область : Продуктовый магазин   
Объекты : Категория продукта, Продукт   
Примечание : Продукты  в  магазине сгруппированы  по  категориям. Для  каждой  категории определено  множество продуктов.

**Теоретична частина :**

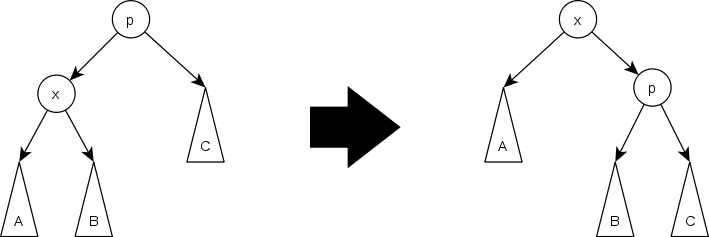
**Розширюване дерево** є двійковим деревом пошуку, у якому підтримується збалансованість. Це дерево належить до класу «саморегульованих дерев», які підтримують необхідний баланс в дереві, щоб забезпечити виконання операції пошуку, додавання і видалення за логарифмічний час від числа елементів, що зберігаються. Це реалізується без використання яких-небудь додаткових полів у вузлах дерева (як, наприклад, в Червоно-чорних деревах або АВЛ-деревах, де у вершинах зберігається, відповідно, колір вершини і глибина піддерева). Замість цього «розширювальні операції» (splay operation), частиною яких є повороти дерева, які виконуються при кожному зверненні до дерева.

**Основні операції :**

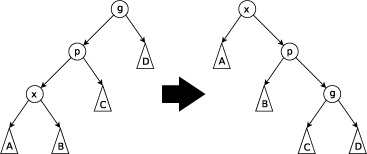
*Splay (розширення)*

Основна операція дерева. Полягає в переміщенні вершини в корінь за допомогою послідовного виконання трьох, наведених нижче, операцій: Zig, Zig-Zig і Zig-Zag. Позначимо вершину, яку хочемо перемістити в корінь за *x*, її родича — *p*, а родича *p* (якщо існує) — *g*.

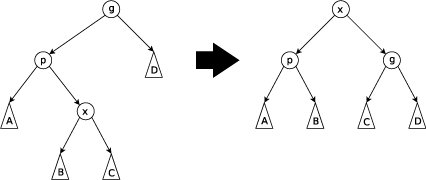
Zig: виконується, коли *p* є коренем. Дерево повертається по ребру між *x* та *p*. Існує лише для розбору крайнього випадку і виконується лише один раз в кінці, коли початкова глибина *x* була непарна.

[](https://uk.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B9%D0%BB:Splay_tree_zig.svg)

*Zig-Zig:* виконується, коли *x* і *p* є лівими (або правими) синами. Дерево повертається по ребру між *p* та *x*.

[](https://uk.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B9%D0%BB:Zigzig.gif)

*Zig-Zag:* виконується, коли *x* є правим сином, а *p* — лівим (чи навпаки). Дерево повертається по ребру між *p* та *x*, а потім — по ребру між *x* та *g*.

[](https://uk.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B9%D0%BB:Zigzag.gif)

*Search (пошук елемента)*

Пошук виконується як в звичайному двійковому дереві пошуку. При знаходженні елементу запускаємо Splay для нього.

*Insert (додавання елемента)*

Запускаємо Splay 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.

**Аналіз :** Вартість кожної операції становить в середньому O(log(n)), де n – кількість вершин.

**Реалізація алгоритму (мова) :** Алгоритм реалізований з використанням мови програмування C++ з використанням інтегрованого середовища розробки Microsoft Visual Studio.

**Інтерфейс програми :** консоль.

**Використання структур даних :**

**1)**В задачці використовується class market{}; - саме в цьому класі йде основна реалізація предметної області. А саме :

Клас працює з 2 типами структур :

struct type {

string group; - назва різновиду.

unsigned int hash\_key; - відповідно ключ, який підраховується ф-ю hash\_string.

vector<product\_element \*> array\_of\_products; - вектор вказівників на продукти вказаного типу.

}; - використовується для збереження різновидів продуктів.

struct product\_element {

string group; - назва різновиду продуктів, до якого відноситься вказаний.

string name; - відповідно ім’я.

double price; - ціна (для інформативності).

unsigned int hash\_key; - відповідно ключ, який підраховується ф-ю hash\_string.

};

vector<type \*> type\_of\_products; - масив типів продуктів в магазині, де кожен елемент масиву має вказівник на масив самих продуктів.

int check\_is\_product(int pos, string product); - перевіряє наявність вказаного продукту в магазині, де pos – це індекс елемента масива типів продуктів.

int check\_is\_type(string product); - перевіряє наявність відповідно типу продуктів в магазині.

void new\_data(); - додавання нового продукту з перевіркою на наявність типу цього продукту (якщо тип відсутній, то створюється новий), відповідно присутня перевірка того чи існує вже вказаний продукт.

void delete\_data(product\_element \*name); - видаляє з каталогу вказаний продукт (зменшує масив продуктів у відповідного типу)

void save\_changing(); - зберігає всі зміни каталогу, перезаписуючи файл, на якому зберігаються дані що до продуктів.

void read\_data\_from\_file(); - зчитує данні що до продуктів з файлу.

unsigned int hash\_string(string key); - хешує імена груп/продуктів для подальшої роботи з деревом порядкової статистики (в подальшому дане значення використовуватиметься як «ключ»).

**2)** class Splay\_tree\_type //клас для побудови основного дерева, вершинами якого є типи продуктів в магазині

{

public:

type \* root; - корінь дерева

~Splay\_tree\_type() {};

Splay\_tree\_type(market \*pr); - конструктор дерева, побудова піддерев (дерево продуктів)

type \*rotate\_left(type \*x); - поворот вліво

type \*rotate\_right(type \*x); - поворот вправо

type \*Splay\_balance(type \*x, unsigned int data); - основна операція балансування дерева

type \*insert\_node(type\_element \*type\_el); - вставка вершини

type \*delete\_node(unsigned int data); - видалення вершини

type \*is\_node(unsigned int data); - перевірка на наявність вершини

bool cout\_tree(); - виведення дерева

bool delete\_tree();

void ST\_menu();

};

**Опис алгоритму з предметною областю:**

Внесено до файлу 4 продукти : ковбаса, шоколад, цукерка «Ромашка», курятина.

Алгоритми основується на побудові дерева з піддеревами. Основне дерево :

1)Будується за алгоритмом вище, реалізований для типів продуктів.

struct type {

node\_colour colour;

string group; - назва групи

unsigned int hash\_key; - ключ для побудови дерева

unsigned int rank; - ранг

RB\_tree\_product \*new\_tree; - вказівник на піддерево продуктів.

type \*left;

type \*right;

type \*parent;

};

class RB\_tree\_group {} – основний клас, в якому описаний метод побудови дерева. Під час побудови дерева «типів» для кожного елемента будується піддерево «продукти» :

struct products {

node\_colour colour;

string name; - назва продукту

string group; - тип продуктів

double price; - ціна

unsigned int hash\_key; - ключ для побудови дерева

unsigned int rank; - ранг

products \*left;

products \*right;

products \*parent;

};

class RB\_tree\_product {} – клас піддерева «продукти», методи.

Відповідно, для користувача подається запит на пошук групи в дереві, а вже потім подається запит на пошук самого продукту в піддереві.

**Тестовий приклад :**

1. Зчитується з файлу «products.txt» інформація про продукти :

4

candies;boom;14.95 429

candies;chocolate bar;16.65 1287

candies;cake;10.59 404

meat;chicken;33.99 725

1. vector<type \*> type\_of\_products; - вноситься інформація про продукти. А саме до вектору додаються 2 нових елементи, які матимуть вказівники на відповідного типу продукти.
2. В подальшому вектор type\_of\_products використовується для побудови основного дерева **Splay\_tree\_type main\_tree(main\_data);,** де main\_data – клас, що зберігає цей вектор. main\_tree представляє із себе червоно-чорне дерево порядкової статистики, елементами якого є типи продуктів : candies та meat. Тобто дерево в даному випадку складається всього з 2 вершиню
3. Під час додавання вершини (приклад) candies починаємо побудову піддерева за вказівником **Splay\_tree\_product new\_tree;** Це дерево матиме елементи = продукти : boom, cake, chocolate bar. Відповідно, користуючись функцією **hash\_number()** отримуємо ключі відповідних імен для побудови дерева. Після побудови піддерева, відбувається виведення дерева та виклик меню редагування вказаного піддерева (додати вершину, видалити вершину). Після виходу з меню редагування повертаємось до кроку 3) (продовжуємо додавати вершини до основного дерева «груп»).
4. Після побудови основного дерева груп та всіх піддерев відбувається виведення дерева та виклик меню редагування для основного дерева (додати вершину, видалити вершину). Після завершення редагування завершується робота програми.

**Основні модулі програми :**

**Group.h –** тут знаходиться основний метод побудови дерева типів продуктів.

**market.h –** обробка файлу з інформацією.

**product.h –** основний метод побудови піддерева продуктів (відповідно для вершини «тип продуктів».

**main.h –** основна компіляція програми.

**Висновок :** Даний тип балансування дерева допомагає уникнути використання пам’яті на збереження кольору(рангу) вершини.

**Використані джерела :** Кормен “Алгоритми. Побудова та Аналіз”.