**Київський національний університет імені Тараса Шевченка**

**Факультет комп’ютерних наук та кібернетики**

**Кафедра інформаційних систем**

**Алгоритми та складність**

**Лабораторна робота №7**

**«Біноміальна куча»**

**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 :**

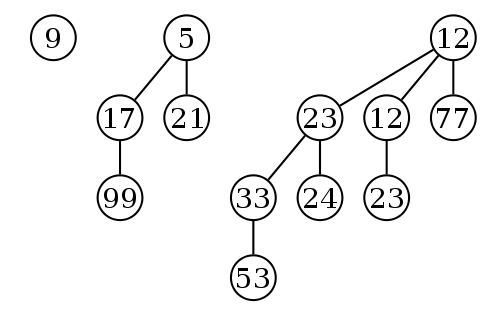
**Біноміальна купа** ([англ.](https://uk.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D1%96%D0%B9%D1%81%D1%8C%D0%BA%D0%B0_%D0%BC%D0%BE%D0%B2%D0%B0" \o "Англійська мова) *binomial heap*) — це множина біноміальних дерев, що задовольняє властивостям біноміальної купи:

1. Кожне біноміальне дерево у купі підпорядковується *властивості неспадної купи*(англ. *min-heap property*): ключ вузла не менший за ключ його батьківського вузла.
2. Для будь якого невід'ємного цілого *k* в купі існує не більше одного біноміального дерева,чий корінь має ступінь *k*.

З даних властивостей випливає, що біноміальна купа, що має *n* вузлів, складається з не більше ніж {\displaystyle \lfloor \;\log \;n\rfloor +1}біноміальних дерев.

Завдяки своїй структурі, біноміальне дерево ступеня *k* можна побудувати з двох дерев ступеня *k*−1 тривіальним приєднанням одного з них до іншого як найлівішого підпорядкованого дерева. Ця властивість є центральною для операції *злиття*біноміальних дерев, яка становить їхню основну перевагу над звичайними купами.

*Приклад біноміальної купи, що містить 13 вузлів з різними ключами.  
Купи складається з біноміальних дерев ступенів 0, 2 і 3.*



***Представлення біноміальних пірамід***

• Кожне біноміальне дерево зберігається у представленні з лівим дочірнім та правим сестринським вузлами.

• Ключ key[x], вказівник на батька p[x], вказівник на найлівішого сина child[x], вказівник на правого брата sibling[x], кількість дочірніх вузлів degree[x].

• Біноміальна піраміда представлена списком коренів її біноміальних дерев впорядкованим за зростанням степенів дерев.

• Вказівник на перший корінь біноміальної піраміди H: head[H].

• Якщо x – корінь, то sibling[x] вказує на наступний корінь у списку.

**Аналіз :**

Всі операції выполняются за час O(log n){\displaystyle O(\log n)}ЩO, де n — кількість вершин, крім вставки нового елемента О(1{\displaystyle O(1)}).

**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 product\_element {

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

string name; - відповідно ім’я.

double price; - ціна (для інформативності).

int key; - для зручності роботи з B+ деревом.

};

**vector<type \*> array\_of\_products; - масив продуктів в магазині, це основні дані, з якими працює B+ дерево.**

int check\_is\_product(string product); - перевіряє наявність вказаного продукту в магазині.

void new\_data(); - додавання нового продукту з перевіркою того чи існує вже вказаний продукт.

void delete\_data(product\_element \*name); - видаляє з каталогу вказаний продукт.

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.

Для роботи з кучею клас вершин кучі:

class Node {

public:

int key; - ключ для побудови кучі

string name; - назва продуктів

string group; - тип продуктів

double price; - ціна

unsigned int degree; - кількість дочірних

Node \*parent, \*child, \*sibling; - батько, лівий крайній син, наступна вершина

bool inf; - перевірка чи є ключ вершини нескінченною (для видалення використовується).

Node(product\_element \*val); - ініціалізація нової вершини

~Node();

void Swap(Node &node); - обмін між данною вершиною та node.

};

**2)** class Binomial\_heap {

public:

Binomial\_heap(); - ініціалізація купи

Binomial\_heap(Binomial\_heap& orig);

~Binomial\_heap(); - видалення купи

Node \*add\_key(product\_element \*key); - додавання нової вершини, повертає вказівнник на вершину

Node \*min\_key(); - поверає вказівник на мінімальний кллюч

Node \*extract\_min\_key(); - видалення вершини з кучі

void decrease\_key(Node \*x, int new\_key); - пониження значення ключа

void remove\_key(Node \*x); - видалення ключа

void operator=(Binomial\_heap &&b); - перегрузка оператора «=»

private:

Node \* head; - голова кучі

Node \*extract\_infinity\_key(); - видалення вершини, ключ якої = максимальному

void mark\_key(Node \*x); - позначити вершини як максимальну, для видалення

static void unite\_binom\_trees(Node \*T1, Node \*T2); - об’єднання 2 дерев в одне, результат зберігатиметься в Т1

static void clear\_binom\_tree(Node \*T); - видалення дерева

static Node \*copy\_binom\_tree(Node \*T); - копіювання дерева

friend Node \*merge\_heaps(Binomial\_heap &H1, Binomial\_heap &H2); -

Злити списки коренів H1 та H2 в упорядкований список (BINOMIAL\_HEAP\_MERGE).

2. Відновити властивості біноміальної піраміди H. Процедура BINOMIAL\_HEAP\_MERGE діє аналогічно етапу злиття в сортуванні злиттям, на кожному кроці переміщаючи в результуючий список дерево меншого порядку. Час її роботи Ο(lg n), де n – сумарна кількість вершин в двох пірамідах. Після злиття списків коренів відомо, що купа H містить не більше двох коренів однакової степені, і вони стоять підряд. Тому будемо зв’язувати корені однієї степені поки всі корені не отримають різні степені.

friend Binomial\_heap unite\_binom\_heaps(Binomial\_heap &, Binomial\_heap &); - Допоміжна операція зв’язування двох біноміальних дерев одного порядку в біноміальне дерево

};

**Опис алгоритму (предметна область):**

Ініціалізуємо market.h. Зчитуємо з файлу market.h :

7

candies;white bar;16.99

candies;chocoshaker;20.55

candies;cake;44.55

meat;chicken;45.99

meat;beef;60.5

fruit;orange;27.95

fruit;apple;15.55

Починаємо обробку бази даних, після завершення зберігаємо зміни до файлу і переходимо до побудови Біноміальної кучі.

Binomial\_heap::Binomial\_heap() {}; ініціалізуємо зчитані з файлу продукти в кучу. Після чого викликаємо функцію роботи з біноміальною кучею Heaps\_editing(\*this);.

Меню роботи з самим деревом :

**If file is not empty, reading data from file!**

**1)Add new data.**

**2)Delete data.**

**3)Use this data.**

**3**

**Nodes from files are ready.**

**white bar chocoshaker cake chicken beef orange apple**

**1. Add a new product.**

**2. Remove a key.**

**3. Get a minimal key.**

**4. Extract a minimal key.**

**5. Decrease key.**

**6. Exit.**

**1 –** додаємо новий елемент

**Enter a name: bzzz**

**Enter a group: drinks**

**Enter a price: 10.00**

**white bar chocoshaker cake chicken beef orange apple bzzz**

**1. Add a new product.**

**2. Remove a key.**

**3. Get a minimal key.**

**4. Extract a minimal key.**

**5. Decrease key.**

**6. Exit.**

**3 –** вивести мінімальний ключ

**Minimal key: 402**

**white bar chocoshaker cake chicken beef orange apple bzzz**

**1. Add a new product.**

**2. Remove a key.**

**3. Get a minimal key.**

**4. Extract a minimal key.**

**5. Decrease key.**

**6. Exit.**

**2 –** видалили продукт

**Enter a key for deleting: beef**

**white bar chocoshaker cake chicken orange apple bzzz**

**1. Add a new product.**

**2. Remove a key.**

**3. Get a minimal key.**

**4. Extract a minimal key.**

**5. Decrease key.**

**6. Exit.**

**6** - вийшли з програми

Кучі при завершені програми матиме в собі 7 продуктів, складається з 3 дерев – 1) 1 вершина; 2) 2 вершини; 3) 4 вершини;

**Основні модулі програми :**

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

**Alg\_ex\_6.cpp –** main частина.

Binomial\_heap.h – сама куча та методи для роботи з нею

**Висновок : Плюси: Вставка нового елементу за О(1), швидке звернення до найменшох вершини в кучі (не треба обходити всі дерева). Мінуси: Пошук рандомного елементу може зайняти більше часу, ніж в збалансованому дереві.**

**Використані джерела :**

<https://neerc.ifmo.ru/wiki/index.php?title=%D0%91%D0%B8%D0%BD%D0%BE%D0%BC%D0%B8%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D0%BA%D1%83%D1%87%D0%B0>

Лекції Оксани Степанівни (Біноміальна куча)