**Київський національний університет імені Тараса Шевченка**

**Факультет комп’ютерних наук та кібернетики**

**Кафедра інформаційних систем**

**Алгоритми та складність**

**Лабораторна робота №4**

**«Оптимальне бінарне дерево пошуку»**

**ЗВІТ**

**Підготував студент**

**2 курсу**

**Групи К29**

**Опанюк Микита**

**Варіант 7**

**2018 рік**

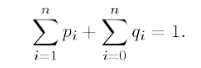
**Опис предметної області :**

*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 :**

Наведемо формальний опис задачі. Є задана послідовність К = (k1, k2, ..., kn), що складається з n різних ключів, які розташовані в відсортованому порядку (так що k1 <k2 <... <kn). З цих ключів потрібно скласти бінарне дерево пошуку. Для кожного ключа до {задана ймовірність pi пошуку цього ключа. Крім того, може виконуватися пошук значень, відсутніх в послідовності К, тому слід передбачити n + 1 фіктивних ключів (d0, d1, ..., dn), що представляють ці значення. Зокрема, d0 представляє всі значення, менші k1, a dn - все значення, що перевищують kn. Фіктивний ключ di (i = 1,2, ..., n - 1) представляє всі значення, які перебувають між ki і ki+1. Для кожного фіктивного ключа di задана відповідна їй ймовірність qi.

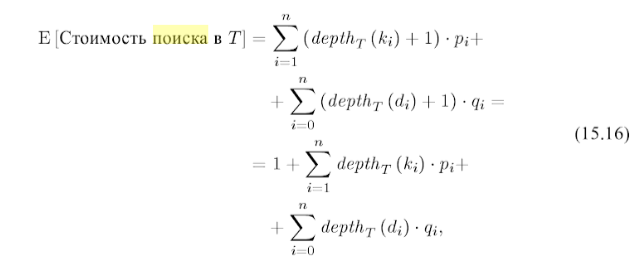
Наприклад : бінарне дерево пошуку для множини, що складається з n = 5 ключів.  
Кожен ключ ki представлений внутрішнім вузлом, а кожен фіктивний ключ di є листом. Пошук може бути або успішним (знайдений якийсь ключ ki), або невдалим (повертається якийсь фіктивний ключ di), тому справедливо співвідношення :

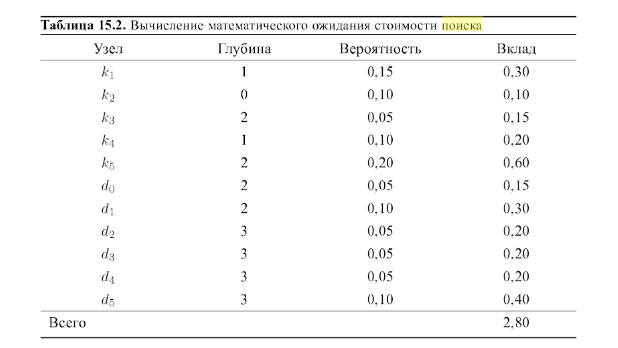


Ймовірності, що відповідають внутрішнім вузлам pi и листям qi, наведені в таблиці :

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| i | 0 | 1 | 2 | 3 | 4 | 5 |  |
| Pi |  | 0,15 | 0,10 | 0,05 | 0,10 | 0,20 |  |
| qi | 0,05 | 0,10 | 0,05 | 0,05 | 0,05 | 0,10 |  |

Оскільки ймовірність знаходження кожного звичайного і фіктивного ключа вважається відомою, можна визначити математичне очікування вартості пошуку по заданому бінарному дереву пошуку T. Припустимо, що «фактична вартість» пошуку визначається кількістю перевірених вузлів, тобто збільшеною на одиницю глибиною вузла на дереві T, в якому знаходиться шуканий ключ. Тоді математичне очікування вартості пошуку в дереві T рівне :

  
де величина depthT () позначає глибину вузла в дереві T. У таблиці показано математичне очікування вартості пошуку для вищевказаного бінарного дерева.



Побудуємо для даного набору ймовірностей бінарне дерево пошуку, математичне очікування вартості пошуку для якого буде мінімальним. Таке дерево називається оптимальним бінарним деревом пошуку.

**Аналіз :** Вартість основної операції становить в середньому O(), 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 1 типом структури :

struct product\_element {

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

string name; - відповідно ім’я.

double price; - ціна (для інформативності).

unsigned int key; - ключ, для побудови оптимального дерева

double probability; - ймовірність того, що потрапить в цю вершину

double fact\_probability; - ймовірність того, що отримаємо фіктивне значення

(в основному це листя дерева)

product\_element \*left; - лівий син

product\_element \*right; - правий син

};

unsigned int size\_of\_f; - кількість продуктів (не є фіктивними ключами)

vector<product\_element \*> main\_array\_of\_products; - вектор продуктів, зчитаних з файлу

void read\_data\_from\_file(); - зчитує данні що до продуктів з файлу. Крім цього зберігає 1 фіктивну вершину, що використовуємо для знаходження та побудови оптимального дерева.

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

**2)** class optimal\_product\_tree {

public:

vector<vector<double> > e; - e[i][j] – означає мат. очікування вартості пошуку в оптимальному бінарному дереві пошуку з ключами ki…kj

vector<vector<double> > w; - додаткова матриця, що використовується для оптимізації підрахунку e[i][j]

vector<vector<int> > root; - матриця, відносної якої будуеться бінарне дерево

product\_element \*tree\_root; - вершина бінарного дерева

void sort\_type(vector<product\_element \*> &array\_to\_sort){}; -відсортовує продукти за ключем

optimal\_product\_tree(){}; - ініціалізація матриць, що використовуються в підрахунку математичного очікування

~optimal\_product\_tree(){}; - деструктор, спустошує пам’ять

void build\_optimal\_search\_tree(int n) {}; – основна функція, що заповнює матрицю root завдяки коректному підрахунку e[i][j]. Як результат – знаходження мат очікування вартості пошуку у всьому дереві.

void cout\_matrix(){}; - виводить на екран всі три матриці

product\_element \*printOBST(int i, int j, product\_element \*elem){}; // будує оптимальне дерево

**Опис алгоритму з предметною областю:**

Внесено до файлу 8 продуктів та одне фіктивне значення :

nt;nt;0.01 0 5 – фіктивне значення

candies;boom;19.99 5 0

candies;cake;35.95 5 5

candies;chocolate bar;16.5 10 5

meat;beaf;49.95 10 10

vegetables;potato;6.99 5 5

fruits;orange;25.49 0 5

fruits;apple;19.99 10 5

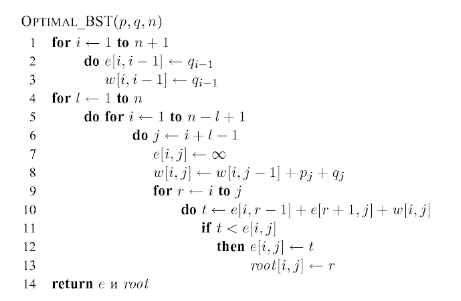
fast food;burger;25.95 5 10

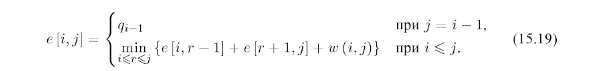
Переходимо до роботи з класом : class optimal\_product\_tree {};

Далі, після завершення роботи з файлом, ініціалізуються відповідно матриці e[][] для підрахунку мат очікування, допоміжна w[][], та для побудови дерева root[][] – всі мають розмір 9X9 (тому що 8 продуктів + 1 фіктивний елемент).

Сортуємо елементи вектора продуктів, для коректної побудови оптимального дерева. Переходимо до роботи з функцією void build\_optimal\_search\_tree(int n), де n = 8 – кількість продуктів.

Псевдокод цієї функції :



Цикл for в 1 строчці ініціалізує e[][] та w[][]. Потім в циклі for в строчках 4-13 за допомогою рекуррентных співідношень :



обчислюємо елементи матриць е[i,j] і w[i,j] для всіх індексів 1 <= i <= j <= n.

В 1 ітерації, коли l = 1, в цьому циклі обчислюємо е[i, i] і w[i, i] для i = 1,2,..., n.

В 2 ітерації, коли l = 2, обчислюємо е[i, i+1] і w[i,i+1] для i= 1,2,...,n-1 і т.д.

У внутрішньому циклі for (строки 9-13) кожний індекс r апробується на роль індекса кореневого елемента kr оптимального бінарного дерева пошуку з ключами кi,... ,kj. В цьому циклі елементу root [i,j] задається те значення індекса r, яке підходить краще всього.

Таблиці е[i, j], root[i,j], w[i,j] :

В кінці виводяться всі матриці на екран.

0 0 0 0 0 0 0 0 0

5 25 55 75 115 135 175 225 275

0 10 20 35 75 90 120 170 220

0 0 5 10 35 50 80 125 170

0 0 0 0 15 30 60 95 140

0 0 0 0 5 10 30 65 100

0 0 0 0 0 5 15 45 80

0 0 0 0 0 0 5 20 55

0 0 0 0 0 0 0 5 20

0 0 0 0 0 0 0 0 0

0 1 1 1 2 2 2 4 4

0 0 2 2 2 4 4 4 4

0 0 0 3 4 4 4 6 7

0 0 0 0 4 4 4 6 7

0 0 0 0 0 5 6 6 7

0 0 0 0 0 0 6 7 7

0 0 0 0 0 0 0 7 7

0 0 0 0 0 0 0 0 8

0 0 0 0 0 0 0 0 0

5 25 35 40 55 60 70 85 100

0 10 20 25 40 45 55 70 85

0 0 5 10 25 30 40 55 70

0 0 0 0 15 20 30 45 60

0 0 0 0 5 10 20 35 50

0 0 0 0 0 5 15 30 45

0 0 0 0 0 0 5 20 35

0 0 0 0 0 0 0 5 20

**Основні модулі програми :**

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

**Alg\_ex\_4.cpp –** основна компіляція програми.

**Висновок :** Даний тип дерева прискорює звернення до вершин, ймовірність у яких більша, намагаючись розмістити їх як можна ближче до кореня дерева. Як приклад, це дуже корисно при створені словника, де користувач може швидше звертатись до слів, які частіше зустрічаються в тексті, на основі якого побудований цей елемент. В моєму випадку : біля кореня розташовані продукті, які частіше всього «купляють продавці».

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