Київський національний університет імені Тараса Шевченка
Факультет комп`ютерних наук та кібернетики
Кафедра інтелектуальних інформаційних систем
Алгоритми та складність

Лабораторна робота №2_2
“Реалізація оптимального бінарного дерева пошуку”
Виконав студент 2-го курсу
Групи ІПС-22
Левицький Іван Олегович

2025

Завдання: Реалізувати оптимальне бінарне дерево пошуку(динамічне програмування) для типу даних Раціональні числа.

Предметна область: Типи даних і їх оптимальний пошук

Теорія: Оптимальне Бінарне дерево пошуку – це звичайне бінарне дерево пошуку, зі своєю модифікацією, для кожново вузла нашого дерева існує кількість його пошуків, тобто ймовірність його вибору – f(e) , а також вартість(cost(e)) – це скільки ребер потрібно пройти від кореня до нашого вузла.

І на основі цих даних, ми можемо отримати суму нашого дерева за формулою :

$f(e_1) * cost(e_1) + f(e_2) * cost(e_2) + \ldots + f(e_n) * cost(e_n)$

І головною метою оптимального BST(Біноміальне дерево пошуку) – є пошук і побудова дерева із найменшою сумою.

• e[i][j] — очікувана вартість пошуку в піддереві i..j

• w[i][j] — сумарна ймовірність ключів і хибних запитів між i..j

• rootIdx[i][j] — оптимальний корінь піддерева i..j

Алгоритм:

1) Ініціалізуємо вектори e, w, rootIdx розміра n x n.

2) Для кожного i, вставляємо базовий випадок e[i][j] = (r > i ? e[i][r - 1] : 0) + (r < j ? e[r + 1][j] : 0) + w[i][j].
3) Для всіх довжин l = 1…n
- обчислити w[i][j] = w[i][j - 1] + p[j];
- знайти r в діапазоні [i..j], що мінімізує e[i][r-1] + e[r+1][j] + w[i][j]
- записати результат у e[i][j] і root[i][j]
4) Побудувати дерево рекурсивно за таблицею root

Мова реалізацію робити: C++

Модулі програми:

class Rational – реалізовий числа типу даних «Раціональні»

struct OBSTNode – структура, яка реалізовує вузол бінарного дерева, тобто містить значення типу Rational, праве і ліве посилання на OBSTNode, а також ймовірність вибору ключа

struct KeyInfo – структура яка реалізовує ключ, дня нашого Оптимального Бінарного дерева пошуку

class AdaptiveOBST – сама реалізація Оптимального Бінарного дерева пошуку

void insert(const Rational& key, double baseWeight = 1.0) – метод, який вставляє елемент в наше дерево

void access(const Rational& key) - виконує адаптивне оновлення OBST при кожному доступі до ключа, тобто при кожному звертанні до ключа, він збільшує його ймовірність

void rebuild() – викликає перебудову, для нашого дерева через певний час (тобто метод OBSTNode* buildOBST(const std::vector<KeyInfo>& keys, const std::vector<double>& p))

OBSTNode* buildOBST(const std::vector<KeyInfo>& keys, const std::vector<double>& p) – метод, який будує оптимальне бінарне дерево пошуку, тобто головний наш метод

void printFrequencies() – метод, який виводить частоту кожного вузла

void printTree() – виводить наше дерево в консоль

void exportToDOT(const std::string& filename) const – експортує наше дерево в файл типу .dot для подальшої візуалізації дерева

void generateTreeImage(const std::string& dotFilename) – на основі .dot файлу генерує для нас картинку формату .png

void openFile(const std::string& filename) – відкриває нашу картинку через консоль

Тестовий приклад:

Ймовірності пошуку (ключі):
• p1 = 0.035714 (ключ A)
• p2 = 0.214286 (ключ B)
• P3 = 0.642857(ключ C)
• P4 = 0.107143(клюс D)

Ймовірності хибного пошуку (між/поза ключами)

q0 = 0.02
q1 = 0.02
q2 = 0.02
q3 = 0.02
q4 = 0.02

Обчислення виконується за формулою:
e[i][j] = min_{r=i..j} [e[i][r-1] + e[r+1][j] + w[i][j]] де
w[i][j] — сумарна ймовірність для піддерева від ключа i до j, а
e[i][r-1] і e[r+1][j] — вартість лівого та правого піддерева при виборі r як кореня.

▪ root[1][1]:
w[1][1] = p1 + q0 + q1 = 0.035714 + 0.02 + 0.02 = 0.075714
e[1][1] = e[1][0] + e[2][1] + w = 0.02 + 0.02 + 0.075714 = 0.115714

▪ root[2][2]:
w[2][2] = p2 + q1 + q2 = 0.214286 + 0.02 + 0.02 = 0.254286
e[2][2] = 0.02 + 0.02 + 0.254286 = 0.294286

▪ root[3][3]:
w[3][3] = p3 + q2 + q3 = 0.642857 + 0.02 + 0.02 = 0.682857
e[3][3] = 0.02 + 0.02 + 0.682857 = 0.722857

І ТД….

w[1][2] = p1 + p2 + q0 + q1 + q2 = 0.035714 + 0.214286 + 0.02 + 0.02 + 0.02 = 0.310000

Варіант r = 1:
e[1][0] + e[2][2] + w = 0.02 + 0.294286 + 0.31 = 0.624286

Варіант r = 2:
e[1][1] + e[3][2] + w = 0.115714 + 0.02 + 0.31 = 0.445714

→ мінімум = 0.445714, root[1][2] = 2

 root[1][2] = 2 — це означає:

у піддереві від ключа 1 до ключа 2 (тобто 3/10 і 1/2),
оптимально обрати ключ №2 (1/2) як корінь
— він забезпечить мінімальну очікувану вартість пошуку.

 e[1][2] = 0.445714 — означає:

Якщо побудувати піддерево з 3/10 і 1/2 оптимально,
середня вартість пошуку в ньому буде ≈ 0.4457

w[2][3] = p2 + p3 + q1 + q2 + q3 = 0.214286 + 0.642857 + 0.02 + 0.02 + 0.02 = 0.917143

r = 2:
0.02 + 0.722857 + 0.917143 = 1.66

r = 3:
0.294286 + 0.02 + 0.917143 = 1.231429

→ root[2][3] = 3

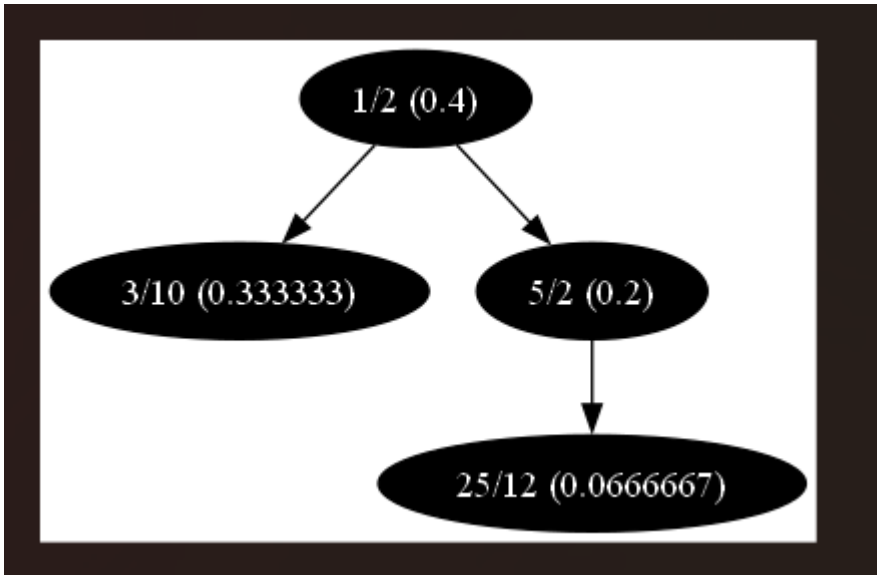
Що це означає:

- Ми розглядаємо піддерево з ключами 1/2 (ключ №2) і 5/2 (ключ №3).
- Є два варіанти вибору кореня:
  - r = 2: обираємо 1/2 як корінь
  - r = 3: обираємо 5/2 як корінь
- Мінімальна вартість досягається при r = 3
- Отже, оптимальний корінь для піддерева [2][3] — це ключ №3 (`5/2`)

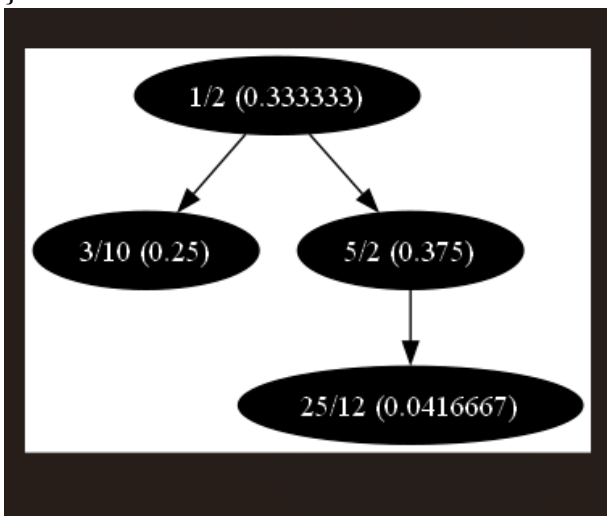
середня вартість пошуку в ньому буде ≈ = 1.231429

Тепер приклад самої програми:

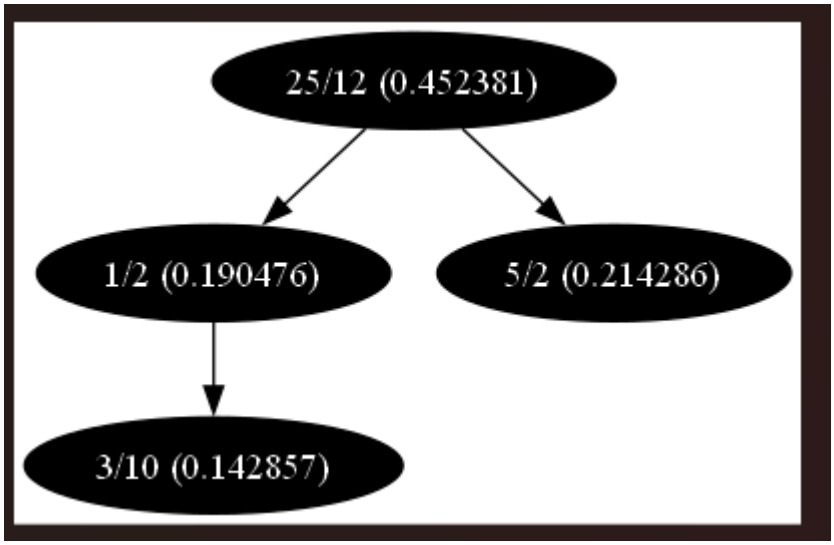
```
tree.insert(Rational(5,2),15);
tree.insert(Rational(3,10),25);
tree.insert(Rational(25,12),5);
tree.insert(Rational(1,2),30);
```




```
for (int i = 0;i < 5;i++)
{
   tree.access(Rational(3, 10));
}
for (int i = 0; i < 10; i++)
{
   tree.access(Rational(1, 2));
}
for (int i = 0; i < 30; i++)
{
   tree.access(Rational(5, 2));
}
```




```
for (int i = 0; i < 90; i++)
{
   tree.access(Rational(25, 12));
}
```



Висновки

Було реалізовано алгоритм побудови оптимального дерева пошуку з використанням динамічного програмування. Структура дерева забезпечує мінімізацію середньої вартості пошуку при відомих ймовірностях ключів.

Використані джерела:

1) Cormen T.H., Leiserson C.E., Rivest R.L., Stein C. *Introduction to Algorithms*, MIT Press, 3rd Edition.

2) https://youtu.be/vLS-zRCHo-Y?si=ZkzDXR4M15gut6XI

3) Wikipedia — Optimal Binary Search Tree