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

Завдання №4
“Оптимальне бінарне дерево пошуку
(динамічне програмування)”

Виконав студент 2-го курсу
Групи К-28
Гуща Дмитро Сергійович

2021

Завдання:

Оптимальне бінарне дерево пошуку (динамічне програмування).

Предметна область:

Теорія

У двійковому дереві пошук деяких елементів може відбуватися частіше, ніж інших, тобто існують ймовірності *p(k)* пошуку k-го елемента і для різних елементів ці ймовірності не однакові. Можна припустити, що пошук в дереві в середньому буде швидшим, якщо ті елементи, які шукають частіше, будуть перебувати ближче до кореня дерева.

Нехай дано 2n+1 ймовірностей p_1 , p_2 , ..., p_n , q_0 , q_1 , ..., q_n , де p_i - ймовірність того, що аргументом пошуку є k_i-елемент; q_i -ймовірність того, що аргумент пошуку лежить між вершинами k_i і k_{i+1} ; q_0 -ймовірність того, що аргумент пошуку менше, ніж значення елемента k_1 ; q_n -ймовірність того, що аргумент пошуку більше, ніж k_n . Тоді ціна дерева пошуку C буде визначатися таким чином:

$$C = \sum_{j=1}^{n} p_j(\mathrm{levelroot}_j + 1) + \sum_{k=1}^{n} q_k(\mathrm{levellist}_k),$$

де $\mathrm{levelroot}_j$ - рівень вузла j, а $\mathrm{levellist}_k$ - рівень листа k. Дерево пошуку називається оптимальним, якщо його ціна мінімальна. Тобто оптимальне бінарне дерево пошуку – це бінарне дерево пошуку, побудоване в розрахунку на забезпечення максимальної продуктивності при заданому розподілі ймовірностей пошуку необхідних даних.

Алгоритм

У задачі динамічної оптимальності дерево може бути змінено в будь-який час, зазвичай шляхом дозволу обертання дерева . Вважається, що в дереві є курсор, що починається з кореня, який він може переміщати або використовувати для виконання змін. У цьому випадку існує деяка послідовність цих операцій з мінімальною вартістю, яка змушує курсор відвідувати кожен вузол в цільової послідовності доступу по порядку.

Алгоритм динамічного програмування Кнута

У 1971 році Кнут опублікував відносно простий алгоритм динамічного програмування , здатний побудувати статично оптимальне дерево всього за O(n^2) часу.

Основна ідея Кнута полягала в тому, що проблема статичної оптимальності демонструє оптимальну підструктуру; тобто, якщо деяке дерево є статично оптимальним для даного розподілу ймовірностей, то його ліве і праве піддерева також повинні бути статично оптимальними для своїх відповідних підмножин розподілу. Щоб побачити це, розглянемо те, що Кнут називає «зваженої довжиною шляху» дерева. Зважена довжина шляху дерева з n елементів є сумою довжин всіх можливих шляхів пошуку, зважених по їх відповідним можливостям. Дерево з мінімальною зваженої довжиною шляху по визначенню є статично оптимальним.

Нехай буде зваженою довжиною шляху статично оптимального дерева пошуку для всіх значень між a_i і a_j , нехай буде загальною вагою цього дерева і нехай буде індексом його кореня.

Алгоритм можна побудувати за такими формулами:

$$E_{i,i-1} = W_{i,i-1} = B_{i-1} \text{ for } 1 \leq i \leq n+1$$

$$W_{i,j} = W_{i,j-1} + A_j + B_j$$

$$E_{i,j} = \min_{i \leq r \leq j}(E_{i,r-1} + E_{r+1,j} + W_{i,j}) \text{ for } 1 \leq i \leq j \leq n$$

Складність

Наївна реалізація цього алгоритму фактично займає $O(n^3)$ часу, але стаття Кнута включає деякі додаткові спостереження, які можна використовувати для створення модифікованого алгоритму, що займає всього $O(n^2)$ часу.

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
```

obsTree.h

```
struct Node; //структура що описує вузол дерева
class OBSTree; //класс реалізації оптимального дерева бінарного пошуку
void generateTables(); //метод генерування таблиць
Node* constructBST(Node* node, int low, int high); //метод побудови бінарного дерева пошуку
void print(Node* node, int level, bool left) const; //вивід у консоль піддерева
void erase(Node* toErase); //видалення вузла
```

```
OBSTree(const std::vector<Group*>& groups, const std::vector<double>& probs, const std::vector<double>& fictProbs); //конструктор оптимального дерева бінарного пошуку
~OBSTree(); // деструктор оптимального дерева бінарного пошуку
```

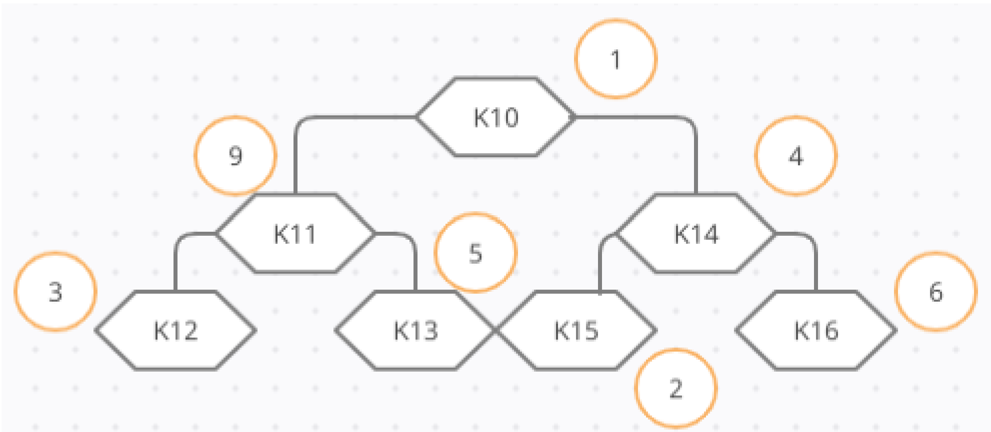
main.cpp

```
int main(); //головная фунція програми
```

Інтерфейс користувача

Вхідні дані є фіксованими і виводяться у консоль.

Тестовий приклади

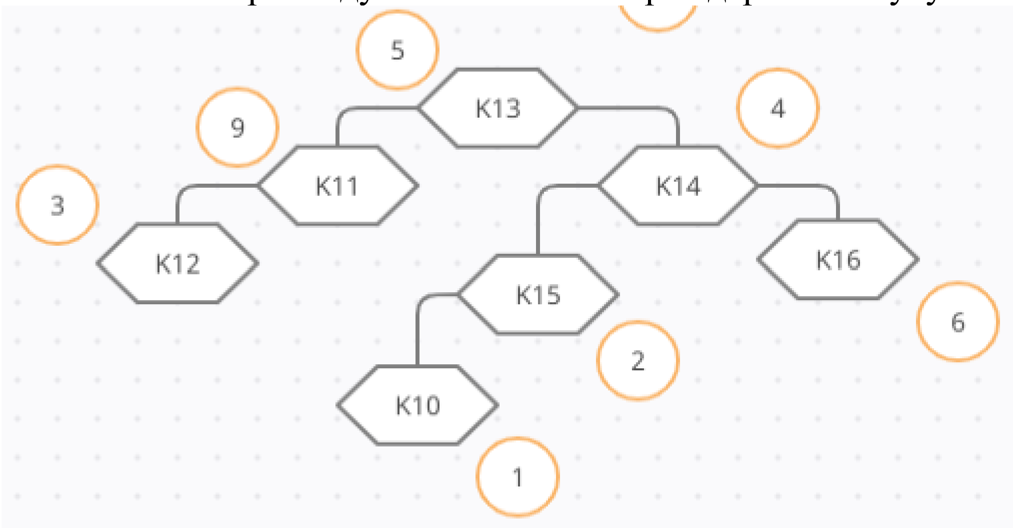


Вартість поточного Дерева Пошуку дорівнює:

$$9 + 3\times2 + 5\times2 + 4 + 2\times2 + 6\times2 = 45$$

Виникає питання: чи можна змінити структуру дерева так, щоб мінімізувати зазначену вартість?

Для зазначеного прикладу оптимальне бінарне дерево пошуку має вигляд:



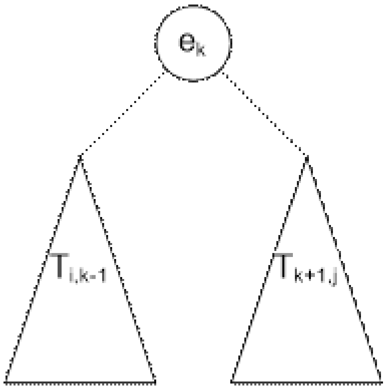
Його вартість дорівнює:

$$9 + 3\times2 + 4 + 2\times2 + 6\times2 + 1\times3 = 38$$

Нехай $T_{i,j}$ дорівнює вартості оптимального бінарного дерева пошуку, яке можна побудувати з елементів $e_i, e_{i+1}, \ldots, e_j$. Очевидно, що $T_{i,i} = 0$

(вартість дерева пошуку з однієї вершини дорівнює нулю). Для $i < j$ має місце рекурентність:

$$T_{i,j} = \min_{i \le k \le j} \left\{ \sum_{p=i}^{k-1} f(e_p) + T_{i,k-1} + \sum_{p=k+1}^{j} f(e_p) + T_{k+1,j} \right\}$$



Елемент e_k ставимо в корені. Вартість побудови лівого піддерева дорівнює $T_{i,k-1}$, правого $T_{k+1,j}$. При цьому, оскільки корінь лівого піддерева знаходиться на один рівень нижче e_k , то для обліку вартості лівого піддерева необхідно додати суму частот всіх його елементів, тобто значення $\sum_{p=i}^{k-1} f(e_p)$

Аналогічно при підрахунку вартості правого піддерева слід додати $\sum_{p=k+1}^{j} f(e_p)$.

При $i > j$ покладемо $T_{i,j} = 0$.

Відзначимо також, що рішення задачі про оптимальне бінарне дерево пошуку аналогічно рішенню завдання про оптимальне множення матриць.

Висновок

Оптимальне бінарне дерево пошуку – це бінарне дерево пошуку, побудоване в розрахунку на забезпечення максимальної продуктивності при заданому розподілі ймовірностей пошуку необхідних даних.

Існують алгоритми, які дозволяють побудувати оптимальне дерево пошуку. Однак такі алгоритми мають тимчасову складність порядку $O(n^2)$.

Таким чином, створення оптимальних дерев пошуку вимагає великих накладних витрат, що не завжди виправдовує виграш при швидкому пошуку.

Література

- https://ru.qaz.wiki/wiki/Optimal_binary_search_tree#Dynamic_optimality
- https://site.ada.edu.az/~medv/acm/Docs%20e-limp/Volume%2016/1522.htm