Київський національний університет імені Тараса Шевченка

Факультет комп`ютерних наук та кібернетики

Кафедра інтелектуальних інформаційних систем

Алгоритми та складність

Завдання №2 (перша частина)

“ Оптимальне бінарне дерево пошуку (динамічне програмування) для комплексних чисел”

Виконав студент 2-го курсу

Групи К-28

Адамов Олексій Віталійович

2022

**Завдання**

Реалізувати оптимальне бінарне дерево пошуку за допомогою динамічного програмування. Нехай дана множина ключів , та кількість запитів пошуку для кожного елементу множини – Задача полягає в тому, об побудувати бінарне дерево з мінімальною вартістю.

**Теорія**

Глибиною вузла дерева називається довжина шляху від кореня до цієї вершини плюс один (глибина кореня дорівнює одиниці).

Вартістю вузла називається глибина вузла помножена на частоту виклику елементу, який зберігає цей вузол.

Вартістю бінарного дерева пошуку називається сума вартостей всій його вузлів. .

Наприклад:

Варіанти бінарного дерева:

1)

2)

Як бачимо меншу вартість має перше дерево, воно і є оптимальним.

**Алгоритм**

На вході маємо два масиви розмірності – відсортовані по зростанню ключі, та – кількість викликів ключа на той самій позиції в .

Скористаємося методом динамічного програмування.

Обчислимо матрицю – мінімальна вартість дерева двійкового пошуку складеного з . Тоді зрозуміло, що мінімальна вартість дерева складеного з усіх даних елементів .

Формула:

.

Таким чином ми кожний раз обираємо корінь, для якого вартість бінарного дерева з буде мінімальною, використовуючи інформацію про можливі піддерева. В вносимо суму вартість вибраного піддерева та суму по , бо додаючи корінь, збільшимо глибину всіх вузлів на одиницю. Таким чином містить мінімальна вартість дерева двійкового пошуку складеного з .

Тривіальний випадок – вартість дерева двійкового пошуку з одного елементу – його кількість викликів. Тому почнемо з обчислення

.

З елементів головної діагоналі, можемо обчислити елементи вище діагоналі на одиницю cost[i][i+1], . Потім елементи вище діагоналі на два, і так далі. В кінці дійдемо до обчислення шуканого .

Як бачимо елементи нижче головної діагоналі нам не потрібні, отже можемо їх не зберігати, а зберігати значення послідовно по колонкам () у розмірності . Тоді .

Для ефективного обчислення обчислимо масив префікс сум .

Тоді . .

Для відновлення структури дерева, будемо також зберігати обраний корінь для кожного у . Тоді рухаючись від , переходимо від до (лівого піддерева) та потім до (правого піддерева). Таким чином ми отримає прямий обхід дерева.

**Складність**

Часова складність побудови дерева – .

По пам’яті – .

**Мова програмування**

С++

**Модулі програми**

* class UpperTriangularMatrix

Клас який зберігає верхню трикутну матрицю, та дозволяє доступатися до даних використовуючи звичні індекси .

* long long optimalBST(const std::vector<long long>& freq, std::vector<std::size\_t>& path)

Функція, яка повертає мінімальну вартість дерева, а в path зберігає прямий обід даного дерева (функція працює лише з індексами елементів).

**Інтерфейс користувача**

Вхідні дані вводяться з текстового файла і виводяться також в текстовий файл.

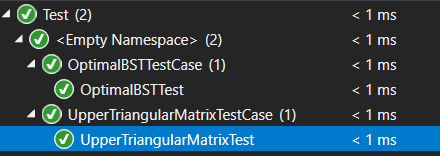
**Тестові приклади**

|  |  |
| --- | --- |
| **input.txt** | **output.txt** |
| 4  (0,1) (2,0) (2,1) (0,7)  50 70 20 30 | cost = 290  Pre-order tree traversal: (2,0) (0,1) (0,7) (2,1)  Has been done for 0.078 milliseconds |

Легко зрозуміти, що відповідь правильна. Ось побудоване дерево з прямого обходу:

**Юніт-тести**

Програма проходить юніт-тести.



**Висновки**

За наявності даних про те, які елементи частіше шукають можна побудувати найефективніше бінарне дерево пошуку на основі цих даних за .

**Література**

* <https://www.geeksforgeeks.org/optimal-binary-search-tree-dp-24/>
* https://www.ibm.com/docs/en/essl/6.2?topic=representation-upper-packed-storage-mode#am5gr\_upsm