Київський національний університет імені Тараса Шевченка

Алгоритми та складність

Лабораторна робота 3

Виконав студент 2-го курсу Групи ПІ-22

Дідківський Володимир Вячеславович

# Завдання

Реалізувати B+ - дерево

# Теорія

B+ - дерево є підвидом звичайних B – дерев тому дамо спочатку визначення B- дереву:

* В-дерева – узагальнення бінарних дерев пошуку.
* Висока степінь розгалуження – вузли можуть мати до тисяч потомків. Дерево Т з коренем root[T] та властивостями :

1. Кожен вузол x містить поля:
   * n[x] – поточна кількість ключів вузла x;
   * впорядковано збережені ключі, так що key1[x]≤key2 [x]≤... ≤keyn[x] [x]; – логічне значення leaf[x], істинне, якщо x – лист.
2. Кожен внутрішній вузол x містить (n[x]+1) вказівник c1[x], ..., cn[x]+1[x] на дочірні вузли.
3. Ключі key i[x] розділяють піддіапазони ключів піддерев: якщо ki – ключ, що зберігається у піддереві з коренем ci [x], то k1 ≤ key[x] ≤ k1

≤key2[x] ≤ ... ≤ keyn[x][x] ≤ kn[x]+1.

1. Всі листи розташовані на одній глибині h, що дорівнює висоті дерева. (Тобто В-дерево ідеально збалансоване за висотою.)
2. Мінімальна і максимальна кількість ключів у вузлі регламентовані фіксованим цілим t≥2 (мінімальна степінь, minimum degree):

* кожен вузол крім кореня містить як мінімум (t–1) ключ, тобто матиме принаймні t синів; непорожнє дерево має в корені хоча б один ключ;
* кожен вузол містить не більше (2t–1) ключів, тобто матиме максимум 2t синів; вузол вважається повним, якщо має рівно (2t–1) ключ.

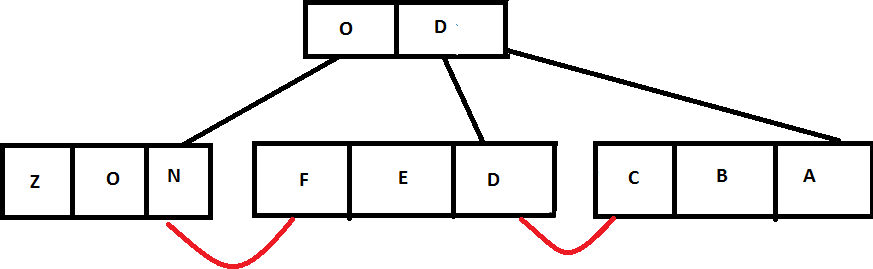
## B+ - дерева

* Істинні значення ключів містяться тільки в листках, внутрішні вузли містять лише ключі-роздільники діапазонів піддерев.
* Листки додатково зв’язані у список. Це дозволяє швидкий доступ до ключів в порядку зростання.
* Легко реалізується незалежність програми від структури інформаційної запису.
* Пошук обов'язково закінчується в листі. Видалення ключа завжди з листа.
* Вимагають більше пам'яті для представлення, порівняно з B- деревами.

# Алгоритми

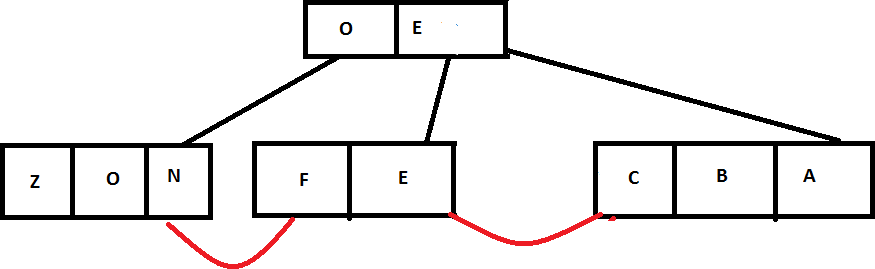
* **Delete**

Видалення ключа D у В+ - дереві починається з його видалення з списку який лежить на листі а потім його заміна у внутрішньому вузлі на таке значення K яке буде меншим за ключ I ключ який є сусідом ключа D у внутрішньому вузлі справа і більший за M (сусід зліва) K береться з того самого листа з якого ми видаляємо ключ D якщо там ключі замалі то шукаємо у сусідньому правому листі а якщо занадто великі то шукаємо у сусідньому лівому листі



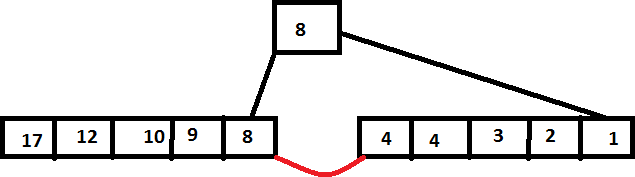
Нехай потрібно видалити елемент D , тоді видаляємо його з центрального листа. Його сусідні елементи у внутрішньому вузлі це O і null зі списку



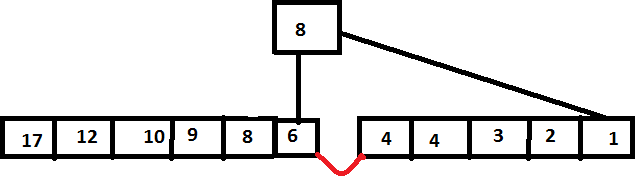
Шукаємо такий ключ K що 𝑂 > 𝐾 > 𝑛𝑢𝑙𝑙 тоді 𝐾 = 𝐸 і дерево після видалення D має вигляд

# Insert ключа K

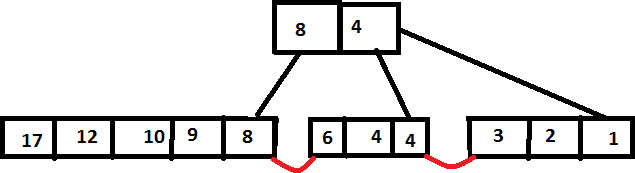
Спочатку ми просто ідем по списках листів з ліва на право і як тільки ми знайдемо ключ який більший за K то ми K вставляємо після цього ключа і далі провіряємо Ключі у батьківському внутрішньому вузлі на те чи виконуються властивості В+ -дерева тобто чи ключі зміненого листа входять в інтервал в який був до цього у батьківському внутрішньому вузлі. Якщо властивості не виконується то ми вставляємо ключ який стоїть спереду(тобто менший) К у середину того інтервалу(утвориться два нових інтервалів) і змінений лист після вставки розділяємо під ті два інтервали Наприклад:



Вставляємо 6



Так як 6 стоїть у тому листі елементи якого мають бути більші усі за 8 то вставляємо 4 у батьківський внутрішній вузол і перерозподіляєм листи



# Складність

Усі операції з деревом займають 𝑂(ℎ) де ℎ - висота дерева для якої

справедлива нерівнсть ℎ ≤ 𝑙𝑜𝑔𝑡

𝑛+1

)

(

2

де 𝑡 степінь дерева (мінімальна

кількість піддерев) то складність не перевищує 𝑂(𝑙𝑜𝑔𝑡 𝑛)

## Мова програмування

С++

## 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");

Group(std::string title);   
Group(std::string title, Student\* first\_student);

std::string getGroupTitle();   
 std::vector<Student\*> getGroupStudents();  
void setGroupTitle(std::string title);

void setGroupStudents(std::vector<Student\*> students);

void addStudent(Student\* student);

void printStudents();

bPlusTree,h

class BPlusTree;

void \_printStep(OStream& output, std::shared\_ptr<BPlusNode<DataType>> node, int level);

std::pair<Node\_ptr, unsigned>

\_subtree\_search(std::shared\_ptr<BPlusNode<DataType>> subtree\_root, const DataType& key);

void \_split\_node(std::shared\_ptr<BPlusNode<DataType>> node);

std::pair<Node\_ptr, unsigned>

\_subtree\_insert(std::shared\_ptr<BPlusNode<DataType>> subtree\_root, const DataType& key);

void \_remove\_from\_node(Node\_ptr node, unsigned index);

explicit BPlusTree(unsigned minimum\_degree = 2);  
void print(OStream& output);

void printSorted(OStream& output);

bool includes(const DataType& key);

void insert(const DataType& key);

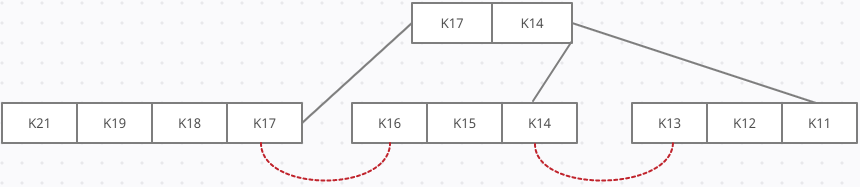
void remove(const DataType& key);

# Інтерфейс користувача

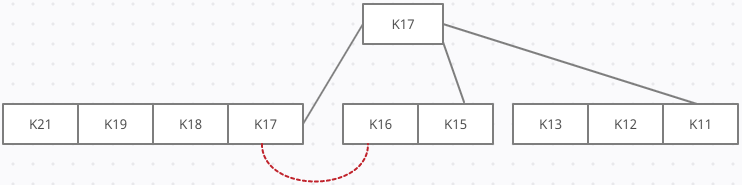
Вхідні дані вводяться з консолі користувачем і виводяться в консоль.

# Тестовий приклад

Нехай Потрібно представити групи та їх назви. Це можна зробити за допомогою B +- дерева де кожен лист це група

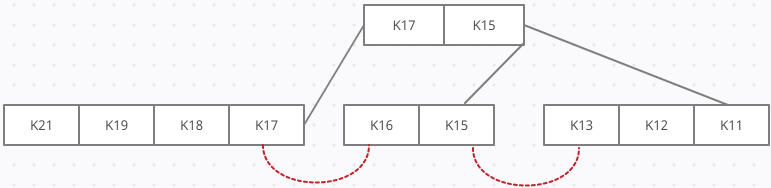


Нехай нам потрібно видалити якусь групу якого не стало у наявності наприклад К14. Спочатку за стандартним пошуком шукаєм К14 у B+ -дереві Потім видаляєм його з листа і внутрішнього вузла:



Далі зі списку вибираєм найменший такий ключ який буде задовольняти нерівність К17 < 𝐾 < 𝑛𝑢𝑙𝑙. K=К15 берем цей ключ вставляєм його на місце К14 у внутрішній вузол

Остаточний вигляд дерева:



# Висновки

Реалізували В+ - дерево, до мінусів можна віднести те що реалізація дерева потребує більше пам'яті для представлення, а до плюсів те що листки додатково зв’язані у список. Це дозволяє швидкий доступ до ключів в порядку зростання порівняно з B-деревами.

# Література

* Лекція № 4