ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

*«*САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО»

Институт металлургии, машиностроения и транспорта

Кафедра «Мехатроника и роботостроение»

**Курсовой проект**

по дисциплине «Программирование»

**«В-дерево»**

Пояснительная записка

|  |  |  |
| --- | --- | --- |
| Выполнил  студент гр. 33335/2 | *(подпись)* | Ненахов И.Д. |
| Работу принял | *(подпись)* | Ананьевский М. С. |

Санкт-Петербург

2018 г.

Формулировка задачи, которую решает алгоритм.

В-дерево – структура данных, предназначенная для эффективного доступа к информации. При работе алгоритма реализуются добавление, удаление и поиск ключа.

Словесное описание алгоритма.

В-дерево представляет из себя совокупность связанных узлов, удовлетворяющих следующим правилам:

* Ключи в каждом узле отсортированы;
* В корне содержится от 1 до 2t-1 ключей, где t – параметр дерева;
* Во всех остальных узлах содержится от t-1 до 2t-1 ключей;
* Листья – узлы, у которых нет потомков;
* Глубина всех листьев одинакова;
* Другие узлы, содержащие n ключей, содержат n+1 потомков. При этом:
  + - Первый потомок содержит ключи из интервала (-∞,К1);
    - Каждый следующий потомок содержит ключи в интервале (Ki-2,Ki);
    - Последний потомок содержит ключи в интервале (Kn,∞).

Реализация алгоритма.

Алгоритм был реализован при помощи языка программирования C++. Для всего дерева был создан класс tree. Также создана структура node, представляющая из себя узел и содержащая несколько переменных:

1. Указатель на родителя;
2. Количество ключей;
3. Значения ключей;
4. Количество потомков;
5. Указатели на потомков.

Первоначально при создании дерева создается корневой узел root.

Также были реализованы методы для этого класса:

1. Добавление ключа:

Были созданы 2 функции: add(var) – для добавления ключа ко всему дереву и addtonode(curnode,var) – для добавления ключа к определенному узлу.

Add(var)

{

Поиск необходимого узла для ключа;

Addtonode(необходимый узел, ключ);

}

Addtonode(необходимый узел)

{

If(текущий узел – полон)

{

Расщепляем узел на 2 узла;

If(текущий узел – не корень)

{

Медианный ключ попадает в узел-родитель;

}

Else

{

Создается новый корневой узел, в который попадает медианный ключ;

}

}

Else

{

Просто добавляем ключ к узлу;

}

}

1. Удаление ключа:

Deletekey(ключ)

{

Найти узел, в котором находится ключ;

Deleteinnode(необходимый узел, ключ);

}

Deleteinnode(необходимый узел, ключ)

{

If(корень – не лист)

{

заменяем ключ на самый правый ключ из поддерева потомков

i-го потомка текущего узла или самый левый ключ из поддерева потомков i+1-го потомка текущего узла, где i – порядок удаляемого ключа в текущем узле;

}

Else

{

Удаляем ключ;

If(в узле осталось <t-1 ключей)

{

If(в следующем узле больше t-1 ключей)

{

Добавляем в текущий узел ключ-разделитель между ним и следующим узлом;

На место ключа-разделителя ставим первый ключ следующего узла;

return;

}

If(в предыдущем узле больше t-1 ключей)

{

Добавляем в текущий узел ключ-разделитель между ним и предыдущим узлом;

На место ключа-разделителя ставим последний ключ предыдущего узла;

return;

}

Else

{

Объединяем текущий узел с предыдущим или следующим узлом;

Добавляем также в текущий узел ключ-разделитель между 2-мя этими узлами;

return;

}

}

}

}

1. Поиск ключа

Search(var)

{

Searchnode() //Ищем узел, в котором содержится ключ;

Searchinnode() //Ищем положение ключа в узле;

}

Searchnode()

{

По порядку проверяем ключи, находя интервал, в котором содержится ключ;

If(искомый ключ не равен ни одному из ключей в узле)

{

Спускаемся по дереву потомков соответствующего потомка;

}

}

Searchinnode()

{

По порядку сравниваем необходимый ключ с ключами узла

}

Анализ алгоритма

1. Время работы алгоритма

Каждое действие при работе с В-деревом (поиск, удаление, добавление ключа) происходит за время О(, n – количество узлов.

Применение алгоритма

Структура B-дерева применяется для организации [индексов](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D0%B4%D0%B5%D0%BA%D1%81_(%D0%B1%D0%B0%D0%B7%D1%8B_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85)) во многих современных [системах](https://ru.wikipedia.org/wiki/%D0%A1%D0%A3%D0%91%D0%94) управления базами данных.

B-дерево может применяться для структурирования информации на жёстком диске. Время доступа к произвольному блоку на жёстком диске очень велико (порядка миллисекунд), поскольку оно определяется скоростью вращения диска и перемещения головок. Поэтому важно уменьшить количество узлов, просматриваемых при каждой операции. Использование поиска по [списку](https://ru.wikipedia.org/wiki/%D0%A1%D0%BF%D0%B8%D1%81%D0%BE%D0%BA_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) каждый раз для нахождения случайного блока могло бы привести к чрезмерному количеству обращений к диску вследствие необходимости последовательного прохода по всем его элементам, предшествующим заданному, тогда как поиск в B-дереве, благодаря свойствам сбалансированности и высокой ветвистости, позволяет значительно сократить количество таких операций.