ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

*«*САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО»

Институт металлургии, машиностроения и транспорта

Кафедра «????????????????????»

**Курсовой проект**

по дисциплине «Программирование»

**«В-дерево»**

Пояснительная записка

|  |  |  |
| --- | --- | --- |
| Выполнил  студент гр. 33335/3 | *(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:

Первоначально была создана рекурсивная функция addtonode. Сначала проверяется, является ли листом текущий узел:

* Не лист:
  + добавляемый ключ меньше нулевого ключа в узле, то применяем addtonode к нулевому потомку текущего узла.
  + добавляемый ключ больше последнего ключа в узле, то применяем addtonode к последнему потомку текущего узла.
  + добавляемый ключ больше нулевого, но меньше последнего ключа узла, то ищем ключи, между которыми должен стоять добавляемый ключ, и переходим к соответствующему потомку.
* лист:
  + количество ключей не максимальное: просто добавляем в конец ключ и сортируем массив ключей.
  + Количество ключей максимальное:
    - Лист не является корнем: в родительский узел добавляем медиану ключей текущего узла и разбиваем текущий узел на 2 узла, в один из них вставляя добавляемый ключ;
    - Лист является корнем: медиана ключей текущего узла остается, а все остальные ключи переходят в 2 узла, являющихся потомками текущего. Идет вставка добавляемого ключа к одному из потомков.

1. Поиск адреса ключа.

Поиск происходит в 2 этапа:

* 1. Поиск нужного узла (метод searchnode):
     1. Если искомый ключ больше последнего ключа в текущем узле, то ищем ключ в последнем потомке текущего узла
     2. Если искомый ключ меньше нулевого ключа в текущем узле, то ищем в нулевом потомке
     3. Если искомый ключ равен одному из ключей, то возвращается указатель на текущий узел
     4. Если искомый ключ не найден, то выводится сообщение «key (значение ключа) not found».
  2. Поиск адреса ключа в узле (метод searchinnode): в цикле сравнивается значение искомого ключа с ключами в узле и возвращается указатель на необходимый ключ.

1. Удаление ключа.

Сначала была реализована функция простого удаления ключа из узла без перестройки дерева (deleteinnode).

Алгоритм: происходит проверка: корень является листком?

1. Если корень – лист, то вызывается функция deleteinnode
2. Если корень – не лист, то происходит поиск узла, в котором содержится необходимый ключ. Потом проверяется, является ли текущий узел листом:
   1. Если текущий узел – лист, то
      1. Если количество ключей в узле>t-1, то вызывается функция deleteinnode
      2. Если соседний справа лист имеет >t-1 ключей, то вызывается функция deleteinnode, затем из узла-родителя вставляется в текущий узел ключ, наиболее близкий к нулевому ключу текущего узла, на его место вставляется нулевой ключ из соседнего справа листа, а оттуда он удаляется.
      3. Если соседний лист слева имеет больше, чем t-1 ключей, происходят действия аналогичные предыдущему пункту, но теперь из соседнего листа берется последний ключ.
   2. Если текущий узел – не лист: пусть необходимый ключ является i-ым ключом в текущем узле. Тогда он заменяется последним ключом поддерева i-го потомка текущего ключа, а тот ключ удаляется.

Анализ алгоритма

1. Время работы алгоритма

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

1. Сложность алгоритма

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.