1. Реализовать на C++ красно-чёрное дерево целых чисел. Необходимые методы: инициализация дерева (конструктор), удаление дерева (деструктор), добавление числа в дерево, удаление числа из дерева, поиск числа в дереве. Также необходимо реализовать итератор по дереву и вывод элементов дерева в консоль «по этажам»: в первой строке – корень, во второй – его потомки и т.д. Реализация дерева – с использованием «умных» указателей.
2. Реализовать на C++ АВЛ-дерево целых чисел. Необходимые методы: инициализация дерева (конструктор), удаление дерева (деструктор), добавление числа в дерево, удаление числа из дерева, поиск числа в дереве. Также необходимо реализовать итератор по дереву и вывод элементов дерева в консоль «по этажам»: в первой строке – корень, во второй – его потомки и т.д. Реализация дерева – с использованием «умных» указателей.
3. Реализовать на C++ красно-чёрное дерево строк std::string. Необходимые методы: инициализация дерева (конструктор), удаление дерева (деструктор), добавление строки в дерево, удаление строки из дерева, поиск строки в дереве. Также необходимо реализовать итератор по дереву и вывод в консоль элементов дерева нужного «этажа». Реализация дерева – с использованием «умных» указателей.
4. Реализовать на C++ АВЛ-дерево строк std::string. Необходимые методы: инициализация дерева (конструктор), удаление дерева (деструктор), добавление строки в дерево, удаление строки из дерева, поиск строки в дереве. Также необходимо реализовать итератор по дереву и вывод в консоль элементов дерева нужного «этажа». Реализация дерева – с использованием «умных» указателей.
5. Реализовать на C++ красно-чёрное дерево пар {int, char}, где int – ключ, char - значение. Необходимые методы: инициализация дерева (конструктор), удаление дерева (деструктор), добавление строки в дерево, удаление строки из дерева, поиск строки в дереве. Также необходимо реализовать итератор по дереву. Реализация дерева – с использованием «умных» указателей.
6. Реализовать на C++ АВЛ-дерево пар {int, char}, где int – ключ, char - значение. Необходимые методы: инициализация дерева (конструктор), удаление дерева (деструктор), добавление строки в дерево, удаление строки из дерева, поиск строки в дереве. Также необходимо реализовать итератор по дереву. Реализация дерева – с использованием «умных» указателей.

7. Реализовать на C++ красно-чёрное дерево пар {std::string, int}, где std::string – ключ, int - значение. Необходимые методы: инициализация дерева (конструктор), удаление дерева (деструктор), добавление строки в дерево, удаление строки из дерева, поиск строки в дереве. Также необходимо реализовать итератор по дереву. Реализация дерева – с использованием «умных» указателей.
8. Реализовать на C++ АВЛ-дерево пар {std::string, int}, где std::string – ключ, int - значение. Необходимые методы: инициализация дерева (конструктор), удаление дерева (деструктор), добавление строки в дерево, удаление строки из дерева, поиск строки в дереве. Также необходимо реализовать итератор по дереву. Реализация дерева – с использованием «умных» указателей.
9. Реализовать на C++ B-дерево целых чисел. Необходимые методы: инициализация дерева (конструктор), удаление дерева (деструктор), добавление числа в дерево, удаление числа из дерева, поиск числа в дереве. Также необходимо реализовать итератор по дереву. Реализация дерева – с использованием «умных» указателей.
10. Реализовать на C++ B-дерево строк std::string. Необходимые методы: инициализация дерева (конструктор), удаление дерева (деструктор), добавление строки в дерево, удаление строки из дерева, поиск строки в дереве. Также необходимо реализовать итератор по дереву. Реализация дерева – с использованием «умных» указателей.
11. Реализовать на C++ квадродерево для хранения двумерного облака точек. Корень – это исходный прямоугольник, охватывающий всё облако. Деление прямоугольника на четыре равные части даёт четыре вершины дерева первого уровня (потомков корня) и т.д. Деление вершин дерева продолжается, пока в прямоугольниках есть точки. Таким образом, концевые прямоугольники содержат по одной точке. Необходимые методы: инициализация дерева (конструктор), удаление дерева (деструктор), добавление точки в дерево, удаление точки из дерева, поиск точки в дереве. Также необходимо реализовать метод, позволяющий получить список точек, лежащих в данной прямоугольной окрестности данной точки. Реализация дерева – с использованием «умных» указателей.

12. Реализовать на C++ пиксельное квадродерево для хранения двумерного облака точек. Корень – это квадрат, охватывающий всё облако. Деление квадрата на четыре равные части даёт четыре вершины дерева первого уровня (потомков корня) и т.д. В конструкторе дерева задаётся высота дерева (т.е. количество делений исходного квадрата). Необходимые методы: инициализация дерева (конструктор), удаление дерева (деструктор), добавление точки в дерево, удаление точки из дерева, поиск точки в дереве. Также необходимо реализовать метод, позволяющий получить список точек, лежащих в данной прямоугольной окрестности данной точки. Реализация дерева – с использованием «умных» указателей.
13. Реализовать на C++ k-d дерево для хранения двумерного облака точек. Корень – это исходный прямоугольник, охватывающий всё облако. Деление прямоугольника пополам (линией, проходящей через середину более длинной стороны) даёт две вершины дерева первого уровня (потомков корня) и т.д. Деление вершин дерева продолжается, пока в прямоугольниках есть точки. Таким образом, концевые прямоугольники содержат по одной точке. Необходимые методы: инициализация дерева (конструктор), удаление дерева (деструктор), добавление точки в дерево, удаление точки из дерева, поиск точки в дереве. Также необходимо реализовать метод, позволяющий получить список точек, лежащих в данной прямоугольной окрестности данной точки. Реализация дерева – с использованием «умных» указателей.
14. Реализовать на C++ октодерево для хранения трёхмерного облака точек. Корень – это исходный прямоугольный параллелепипед, охватывающий всё облако. Деление параллелепипеда на восемь равных частей даёт восемь вершин дерева первого уровня (потомков корня) и т.д. Деление вершин дерева продолжается, пока в параллелепипедах есть точки. Таким образом, концевые прямоугольные параллелепипеды содержат по одной точке. Необходимые методы: инициализация дерева (конструктор), удаление дерева (деструктор), добавление точки в дерево, удаление точки из дерева, поиск точки в дереве. Реализация дерева – с использованием «умных» указателей.

15. Реализовать на C++ воксельное октодерево для хранения трёхмерного облака точек. Корень – это куб, охватывающий всё облако. Деление куба на восемь равных частей даёт восемь вершин дерева первого уровня (потомков корня) и т.д. В конструкторе дерева задаётся высота дерева (т.е. количество делений исходного куба). Необходимые методы: инициализация дерева (конструктор), удаление дерева (деструктор), добавление точки в дерево, удаление точки из дерева, поиск точки в дереве. Реализация дерева – с использованием «умных» указателей.
16. Реализовать на C++ k-d дерево для хранения трёхмерного облака точек. Корень – это исходный прямоугольный параллелепипед, охватывающий всё облако. Деление параллелепипеда пополам (линией, проходящей через середины самых длинных сторон) даёт две вершины дерева первого уровня (потомков корня) и т.д. Деление вершин дерева продолжается, пока в параллелепипедах есть точки. Таким образом, концевые прямоугольные параллелепипеды содержат по одной точке. Необходимые методы: инициализация дерева (конструктор), удаление дерева (деструктор), добавление точки в дерево, удаление точки из дерева, поиск точки в дереве. Реализация дерева – с использованием «умных» указателей.