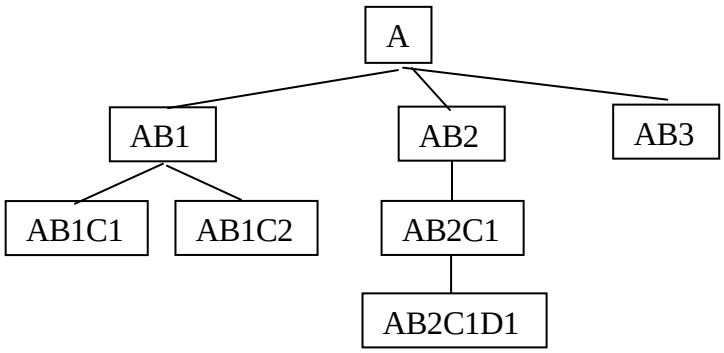
Задача 5. Деревья

В некоторых вариантах данной задачи требуется реализовать дерево с количеством потомков больше 2-х. В этом случае для хранения в узле всех его потомков следует воспользоваться списком, можно стандартной реализаций (например, ArrayList<TreeNode<T>>), а еще лучше своей собственной реализацией связного списка (MyList<TreeNode<T>>). Ввод данных (построение дерева) имеет смысл сделать в виде разбора строки, описывающей дерево в скобочной нотации (разбиралась на лекции), но реализацию разбора потребуется чуть-чуть поправить для поддержки произвольного кол-ва потомков в узле. В этих задачах, вместо рисования дерева на форме можно распечатывать дерево (также может быть на форме - см. примеры) в следующем виде (слева представление дерева, справа пояснение, что это за дерево):

```
A
  AB1
    AB1C1
    AB1C2
  AB2
    AB2C1
      AB2C1D1
  AB3
```



Для остальных задач (двоичных деревьев) самым правильным подходом будет доработка демонстрационного проекта к лекциям TreeSamples (данных пример уже содержит реализацию различных двоичных деревьев, а также поддерживает возможность ввода дерева в виде скобочной последовательности и отрисовку дерева). Для задач, в которых это возможно, допускается использование в решении уже реализованных методов обхода деревьев (реализованы в виде default-методов интерфейса DefaultBinaryTree) с применением функций обратного вызова.

Естественно, при желании вы можете выполнить свою собственную реализацию дерева и демонстрационного проекта (также допускается воспользоваться реализациями деревьев, которые вы можете найти в сети Интернет).

В любом из этих случаев в вашем проекте должно присутствовать дерево, описанное в виде класса (Tree<T> / SimpleTree<T>), а для реализации дерева нужен будет класс узла дерева (TreeNode). Последний имеет смысл описывать в виде внутреннего класса дерева (в примерах к лекциям именно так и сделано). **Непосредственно решение вашего варианта должно быть описано, как метод в класса дерева.**

Если в условии задачи что-то непонятно – попросить пояснить преподавателя.

Варианты:

1. Для двоичного дерева поиска реализовать методы поиска минимально значения (или узла, содержащего такое значение), большего или равного указанного, и максимального значения, меньшего или равного указанного. Необходимо реализовать данные методы эффективно, опираясь именно свойства двоичного дерева поиска, а не полностью обходить все дерево.

2. Для двоичного дерева реализовать метод, который будет возвращать уровень дерева, в

котором больше всего элементов. Учитывая, что может быть несколько уровней, в которых максимальное кол-во элементов, следует найти все такие уровни (вернуть в виде списка номеров уровней).

3. Реализовать функцию, которая изменить двоичное дерево следующим образом:
   Присоединит к самому правому листу в дереве в качестве правого потомка левое поддерево корня дерева (левый потомок у корня дерева при этом надо убрать).

4. Реализовать функцию, которая изменить двоичное дерево следующим образом:
   Удалит все внутренние узлы, у которых есть только один потомок (соответственно, потомки должны занять место удаляемых узлов).

5. Реализовать функцию, которая изменить двоичное дерево следующим образом:
   Удалит все листья, которые не находятся на самом нижнем уровне двоичного дерева.

6. В заданном двоичном дереве поиска для целых чисел найти уровень, на котором сумма элементов будет максимальна.

7. Найти минимум из значений, записанных в двоичном дереве для вещественных чисел.

8. Найти максимум из значений, записанных в листьях двоичного дерева для целых чисел.

9. Найти минимум из значений, которые записаны в левых потомках узлов для заданного двоичного дерева для целых чисел.

10. Реализовать метод построения случайного двоичного дерева для целых чисел. В качестве параметров задаются:
    - minValue – минимальное допустимое значение в дереве;
    - maxValue – максимальное допустимое значение в дереве (значение очередного узла берется случайным образом в промежутке от minValue до maxValue включительно);
    - leftP – вероятность наличия у узла левого потомка (значение от 0 до 1);
    - rightP – вероятность наличия у узла правого потомка (значение от 0 до 1);
    - maxHeight – максимальная глубина дерева (на случай, если построение дерева не прервется раньше).

11. Будем считать *левым относительно корня* элементом дерева такой, для которого в пути от корня дерева к данному элементу чаще надо "идти влево, чем вправо", и *правым относительно корня*, когда чаще надо "идти право, чем влево".
    Для заданного двоичного дерева подсчитать кол-во левых и правых относительно корня элементов.

12. (*) Реализовать дерево, поддерживающее произвольное кол-во потомков в узле. Реализовать для данного дерева два метода:
    - рекурсивную загрузку в данное дерево указанной директории (файлы будут листьями, директории – внутренними узлами дерева).
    - Поиск по данному дереву файлов по имени и расширению (результат – List<File>).

13. (*) Реализовать дерево, поддерживающее произвольное кол-во потомков в узле.
    Реализовать для данного дерева процедуру "переворота" дерева справа налево.
    Подсказка: для каждого узла необходимо "перевернуть" список потомков.

14. (*) Реализовать дерево, поддерживающее произвольное кол-во потомков в узле.
    Реализовать процедуру чтение дерева, состоящего из строк, из файла, где дерево представлено в виде (см. комментарии к задаче до списка вариантов):

```
A
  AB1
    AB1C1
    AB1C2
  AB2
    AB2C1
      AB2C1D1
  AB3
```

    Примечание: строки могут содержать пробелы и другие знаки препинания (кроме перевода строки).

15. (*) В заданном двоичном дереве для строк найти поддерево двоичного поиска с максимальным количеством элементов.

16. Достроить двоичное дерево следующим образом: ко всем листья добавить:
    а) если в узле значение четное, то новый левый лист со значением 1,
    б) если в узле значение нечетное, то новый правый лист со значением -1.

17. Описать метод, который строит двоичное дерево, в котором N полных уровней и на каждом уровне i располагаются узлы, информационные части которых равны i.

18. Реализовать метод, который строит двоичное дерево для заданного по счету числа Фибоначчи следующим образом:
    - В корне дерева стоит заданное число Фибоначчи, потомками которого являются два предыдущих числа Фибоначчи.
    - Любое поддерево в построенном дереве (с числом Фибоначчи) строится по такому же принципу.

    Примечание: так как количество элементов в таком дереве будет очень быстро расти, не следует указывать большие порядковые номера чисел Фибоначчи (до 20).

19. Написать метод клонирования дерева (создания дерева, в котором структура и все значения будут такими же, как в исходном дереве, но все объекты будут другими).

20. (*) В двоичном дереве для целых чисел содержатся как положительные, так и отрицательные элементы. Найти все поддеревья с максимумом суммы всех элементов поддерева. Вернуть в виде списка путей от вершины дерева до вершины каждого такого поддерева. Путь задается в виде строки из букв "L" (если на очередном шаге от узла мы идем к левому потомку) и "R" (– если к правому потомку).

21. (*) Проверить, можно ли удалением одного листа из двоичного дерева привести его к двоичному дереву поиска. Вернуть данный узел в виде пути от вершины дерева к данному узлу (как задается путь – см. предыдущую задачу).
    Само дерево в процессе поиска такого узла меняться не должно.

22. Найти значение, которое встречается в двоичном дереве наиболее часто.
    Никаких дополнительных структур данных не использовать.
    Подсказка: необходимо реализовать метод подсчета встречаемости значения в дереве. Затем в процессе обхода всех узлов дерева для каждого узла вызывать этот метод для подсчета встречаемости в дереве значения, записанного в текущем узле. Сложность будет – $O(n^2)$, но здесь ничего не поделаешь.

23. Из слов заданного текста построить двоичного дерево поиска (простое, без балансировки).

24. (*) Реализовать визуализацию пирамидальной сортировки.
    Для визуализации необходимо построить в виде двоичного дерева пирамиду (кучу), модифицировать эту пирамиду в соответствии с алгоритмом пирамидальной сортировки и визуализировать в виде дерева (воспользуйтесь готовой функцией отрисовки).

25. (*) В двоичном дереве найти все одинаковые (по значениям, но различные) пары поддеревьев. Вернуть в виде списка пар узлов.

26. Реализовать методы (прямой, обратный, симметричный) обхода дерева без использования рекурсии, с применением стека. Сигнатура методов должна соответствовать методам в примерах к лекциям (с параметром Visitor).

27. (*) Проверить, можно ли поменяв друг с другом значения всего лишь двух элементов в двоичном дереве привести его к двоичному дереву поиска. Если можно, выполнить такую замену.
    Подсказка: реализовать процедуру проверки дерева на соответствие двоичному дереву поиска. Также реализовать итератор по узлам дерева (в примерах к лекциям реализован итератор для значений в узлах, для узлов – элементарная модификация. Далее цикл в цикле с попыткой замены и проверкой, пока не получится (если получится). Оцените сложность такого алгоритма.

28. (*) Для двоичного дерева с целыми числами записать в качестве значения каждого узла высоту данного узла. Обход дерева (придется реализовать самостоятельно, модифицировав существующие методы) должен выполняться ровно 1 раз.

29. (*) «Раскрасить» двоичное дерево таким образом, что самый верхний узел был желтым, самый нижний – красным, а все промежуточные узлы – переходные от желтого к красному, в зависимости от глубины (уровня) узла (переход равномерный, т.е. второй уровень – почти желтый, середина – оранжевый, узлы на глубине – почти красные (при достаточной высоте дерева).
    Подсказка: необходимо модифицировать какое-то дерево в проекте TreeSamples так, чтобы его узлы содержали и возвращали цвет. Собственно алгоритм состоит из 2-х шагов: вначале найти высоту дерева, далее повторно обойти дерево, устанавливая для каждого узла цвет в соответствии с уровнем узла (пропорционально от желтого к красному).

30. Реализовать рекурсивный метод, который будет сравнивать эквивалентность двух двоичных деревьев.
    Примечание: пользоваться представлением дерева в виде скобочной последовательности запрещено.

31. Реализовать нерекурсивный метод, который будет сравнивать эквивалентность двух двоичных деревьев.
    Примечание: пользоваться представлением дерева в виде скобочной последовательности запрещено.
    Подсказка: вероятно, необходимо реализовать параллельный нерекурсивный обход деревьев с использованием стека или очереди.

32.

33.

34.