**4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ**

В данном разделе рассматриваются схемы алгоритмов, используемых в программе.

**4.1 Алгоритмы по шагам**

**4.1.1 Алгоритм записи в архив**

Для алгоритма по шагам рассмотрен метод getNextCodeBuffer класса Coder.

1. Начало.
2. Входные данные: QByteArray source, массив байтов, содержащий информацию из исходного файла, словарь кодировок QMap<char, QString> dictionary, где содержится битовая кодировка каждого встречающегося в исходном файле байта, переменная bool leftPrev, которая указывает, остался ли незаполненный байт с предыдущего вызова метода, short prevSize, которая содержит длину битовой записи в незаконченном байте, char prev, где содержится незаконченный байт, переменная int len, в которой хранится число битовый в массиве QByteArray temp, который является внутренней переменной алгоритма.
3. Выходные данные: массив QByteArray buf, в котором находится сжатая запись полученного массива source.
4. Начинается итерация по массиву source.
5. Байт массива кодируется методом QByteArray encode(char sb), где sb – байт массива source, для которого происходит текущий шаг итерации.
6. Переменная len увеличивается на длину кодировки байта в словаре dictionary.
7. Итерация по переменной len.
8. Если переменная len больше либо равна восьми, то первый байт массива temp копируется в выходной массив buf.
9. Декремент переменной len на 8
10. Конец итерации по переменной len.
11. Проверяется переменная len, если она не ноль, то оставшийся в массиве temp незаконченный байт сохраняется в переменную prev, число битов в этом байте записывается в preSize, а флаг leftPrev переходит в значение true.
12. Конец итерации по массиву source.
13. Вернуть buf.
14. Конец.

**4.1.2 Алгоритм формирования дерева байтов**

Для алгоритма по шагам рассмотрен метод formBTree() класса TreeFormer.

1. Начало.
2. Входные данные: очередь с приоритетом priority\_queue<Node<int, char>\*, vector<Node<int, char>\*>, NodeComparator> nodes, в которой узлы дерева сортируются при помощи объекта-компаратора так, что в начале очереди будет находится узел с наименьшим значением поля count.
3. Выходные данные: указатель на вершину дерева Node<int, char>\* tree.
4. Если очередь пуста, вернуть nullptr.
5. Создание нового узла с нулевым значением поля count, который будет использован для получения уникального бинарного кода конца архива. таким образом узел со значением count ноль попадет в начало очереди и для него сгенерируется максимально длинный бинарный код.
6. Итерация по очереди nodes пока размер очереди не станет равен единице.
7. Создание нового узла из двух первых в очереди, значение count нового узла является суммой полей count его наследников.
8. Помещение только что созданного узла в очередь.
9. Конец итерации по очереди nodes.
10. Значение узла tree равняется последнему узлу очереди
11. Вернуть tree.
12. Конец.