# Лабораторная работа № 5

*Тема: Структуры данных: Дерево*

***Задание:***

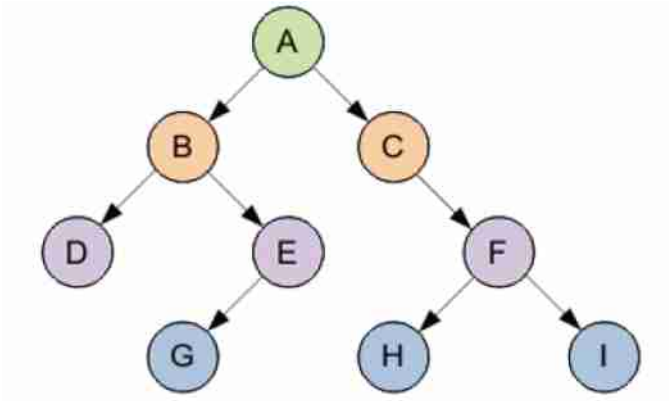
1. Вспомнить: указатели, ссылки; структуры данных (записи); динамические структуры: стек, дек; чтение данных из файла; функции: malloc, sizeof, free; new, delete.
2. Разобраться с алгоритмом кодирования Шеннона-Фано и Хеллмана.
3. Разобраться с принципами построения деревьев. Разработать подход построения бинарного дерева, который реализует соответствующий (вашему варианту) алгоритм кодирования.
4. Написать программу:
   4.1. Считать текст из файла. Это исходный текст, на основании которого будет происходить кодирование.
   4.2. Составить статистику по символам, встречающимся в тексте, в отсортированном виде:

   <Символ> <Частота>
   … …

   4.3. Реализовать необходимые типы/структуры для организации дерева. Разработать функции для работы с деревьями. Протестировать их работу, прежде чем приступать к реализации алгоритма кодирования.
   4.4. Реализовать алгоритм построения дерева.
   4.5. Отобразить (сохранить в файл) таблицу кодов для символов исходного текста.

***Деревья.***

Дерево — структура данных, представляющая собой древовидную структуру в виде набора связанных узлов. Бинарное дерево — это конечное множество элементов, которое либо пусто, либо содержит элемент (корень), связанный с двумя различными бинарными деревьями, называемыми левым и правым поддеревьями. Каждый элемент бинарного дерева называется узлом. Связи между узлами дерева называются ветвями.




Максимальный уровень какого-либо элемента дерева называется его глубиной. Если элемент не имеет потомков, он называется листом (или терминальным узлом).

Бинарное дерево применяется в тех случаях, когда в каждой точке вычислительного процесса должно быть принято одно из двух возможных решений.

На рисунке: A – корень дерева. G, H, I – листья дерева. D,E – являются потомками узла B. F – родитель узлов H и I.

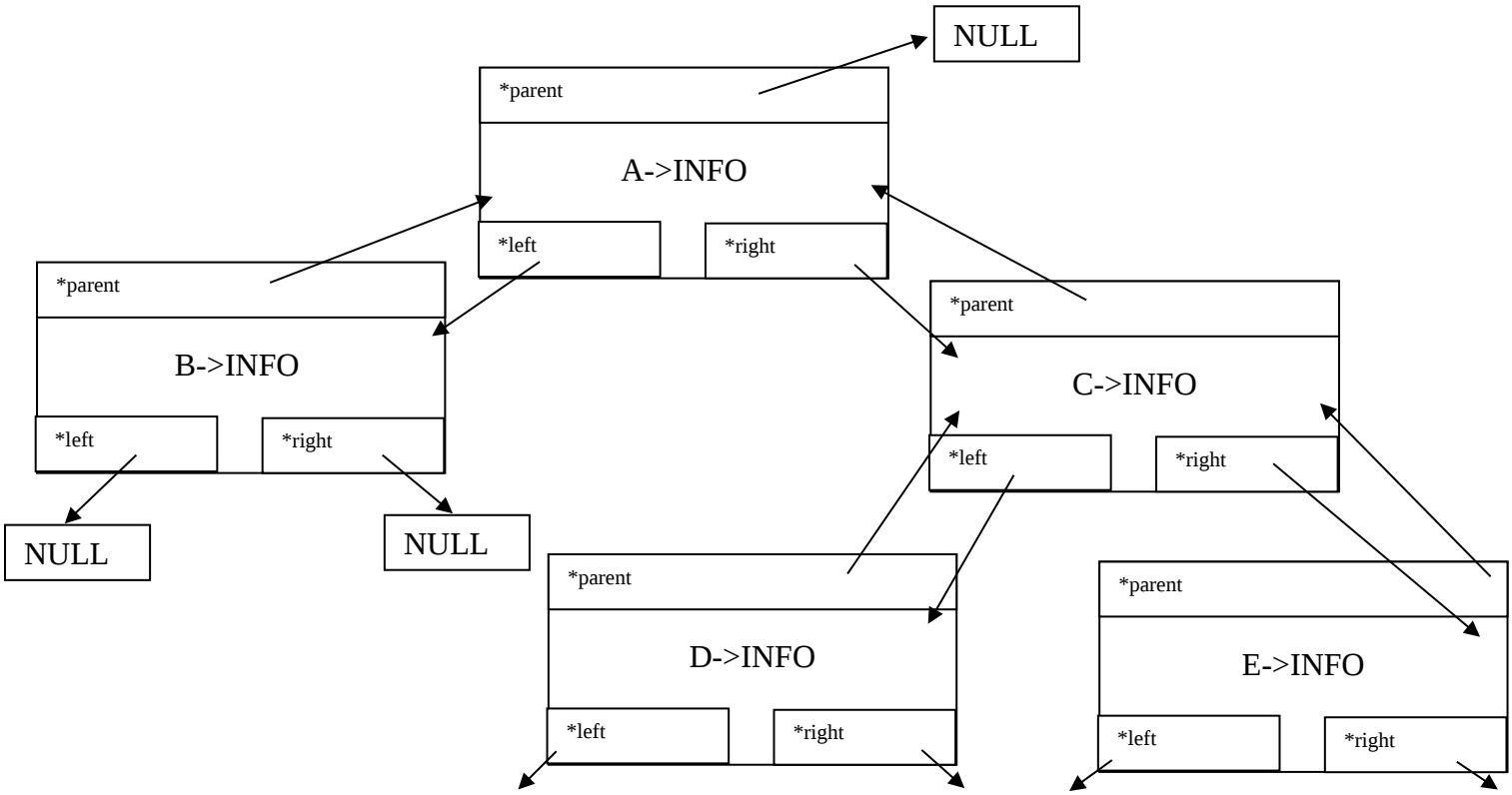
*Реализация дерева.*

Узел дерева можно описать как структуру:

```
struct tnode
{
  int field;            // поле данных, можно задать другую структуру для хранения инфо-ии
                        // полей для данных можно задать несколько
  struct tnode *left;   // указатель на левую ветку
  struct tnode *right;  // указатель на правую ветку дерева
}
```

**Примечание.** Для удобства можно в узел потомок включить ссылку на родительский узел, это облегчит обход в обратном стиле. Программист для удобства может включать много дополнительных вспомогательных элементов (например, налаживание связи по уровню на определённой глубине), однако стоит подходить аккуратно, что бы окончательно не запутаться в организованной структуре (иначе это уже будет мало похоже на дерево, а скорее на динамическую структуру, в основе которой дерево).

Пример, дерево со ссылками на родительский узел.




Пример, добавление узла потомка в левую часть для бинарного дерева:

```
struct tnode * addLeftNode(int info, tnode *parent)
{
  tnode *node = new tnode;  // выделим память под узел
  node->field = info;       // заполнили информацией
  node->left = NULL;        // ветви пока указывает на пустоту
  node->right = NULL;       // -//--
  parent->left = node;     // родительскому узлу передадим ссылку нового узла
  return (node);            // вернем адрес созданной структуры
}
```

Пример, создание родительского узла для двух элементов (обратное построение):

```
struct * tnode addParrentForTwo(int info, tnode *leftChild, tnode *rightChild)
{
  tnode *parent = new tnode; // выделим память под узел
  parent->field = info;
  parent->left = leftChild;
  parent->right = rightChild;
  return parent;             // вернём указатель на созданную структуру
}
```

Пример, обратный обход дерева (от листа к корню) и вывод информации из узлов:

```
void printInfoLeafToRoot(tnode *leaf)
{
  tnode *currNode = new tnode;  // создадим указатель на узел нашего дерева и нарекаем его текущим
  currNode = leaf;              // текущий указатель инициализируем переданной ссылкой на лист
  // цикл будет продолжаться до тех пор пока указатель на родителя не укажет в пустоту,
  // это означает что мы достигли корня, для удобства можно
  //  создать функции isRoot(tnode) (возвращает True, если узел является корнем )
  while(currNode->parent != NULL)
  {
    cout<<currNode->field<<" - "<<endl; // вывод информации из текущего узла
    currNode = currNode->parent;        //  переходим к родительскому узлу
  }
  cout<<currNode->field<<endl;          // вывод из корневого узла
}
```

***Алгоритмы оптимального кодирования.***

*1. Алгоритм Шеннона-Фано*

Один из первых алгоритмов сжатия, которые впервые сформулировали ученые Шеннон и Фано, который использует коды переменной длины: часто встречающийся символ кодируется кодом меньше длины, редко встречающийся — кодом большей длины. Коды Шеннона-Фано префиксные, то есть никакое кодовое слово не является префиксом любого другого. Это свойство позволяет однозначно декодировать любую последовательность кодовых слов.

Этапы:

1. Символы первичного алфавита выписывают по убыванию вероятностей (частоты).
2. Символы полученного алфавита делят на две части, суммарные вероятности которых максимально близки друг к другу.
3. В префиксном коде для первой части алфавита присваивается двоичная цифра «0», второй части - «1». (первая часть — может интерпретироваться как левая ветка)
4. Полученные части рекурсивно делятся и их частям назначается соответствующие двоичные цифры (до тех пор, пока количество элементов в части (в узле) не будет равно 1це).

Построение дерева для Алгоритма Шеннона-Фано начинается с корня, всё множество кодируемых элементов соответствует корню дерева. Оно разбивается на два подмножества с примерно одинаковыми суммарными вероятностями. Эти подмножества соответствуют двум вершинам второго уровня, которые соединяются с корнем. Далее каждое их этих подмножеств разбивается на два подмножества по тому же принципу. Им соответствуют вершины третьего уровня. Если подмножество содержит единственный элемент, то ему соответствует лист дерева. Таким образом выполняем алгоритм разбиения до тех пор пока не получим все концевые вершины.

**Примечание.** Формируемый код можно хранить в каждом узле, таким образом, когда узел будет содержать один символ (и останется листом), код хранящийся там будет соответствовать реальному коду этого символа по алгоритму.

*2. Алгоритм Хаффмана*

Жадный алгоритм оптимального префиксного кодирования алфавита с минимальной избыточностью.

Два основных этапа алгоритма:

- Построение оптимального кодового дерева.

- Построение отображение код — символ на основе построенного дерева.

Аналогично алгоритму Шеннона-Фано построение кодов символов по методу Хаффмана зависит от их вероятности появления и обладает свойством префиксности.

Построение Н-дерева:

1. Символы входного алфавита образуют список свободных узлов. Каждый лист имеет вес, который равен вероятности (частоте) появления символа.
2. Выбирается 2 свободных узла дерева с наименьшими весами.
3. Создается их родитель с весом равным их суммарному весу.
4. Родитель добавляется в список свободных узлов, а два его потомка удаляются из него.
5. Одной дуге, выходящей из родителя, ставится в соответствие бит 1, другой — 0.
6. Шаги, начиная со второго, повторяются, пока в списке свободных узлов не останется один свободный узел. Он и будет считаться корнем.

**Примечание.** Т.к. построение дерева начинается в обратном порядке (с листьев) необходимо иметь вспомогательную структуру, которая в алгоритме определена как «список свободных узлов».

**Примечание.** Чтобы определить код для каждого символа нужно пройти путь от листа дерева, соответствующего символу, до его корня накапливая биты при перемещении по ветвям. Полученная последовательность бит является кодом данного символа.

***Варианты.*** (<№В журнале> mod 10)

| | Алгоритм | Док-т с текстом |
|---|---|---|
| 0. | Шеннона-Фано | text1.txt |
| 1. | Хаффмана | text1.txt |
| 2. | Шеннона-Фано | text2.txt |
| 3. | Хаффмана | text2.txt |
| 4. | Шеннона-Фано | text3.txt |
| 5. | Хаффмана | text3.txt |
| 6. | Шеннона-Фано | text4.txt |
| 7. | Хаффмана | text4.txt |
| 8. | Шеннона-Фано | text5.txt |
| 9. | Хаффмана | text5.txt |

Контрольные вопросы:

1. Что такое дерево?
2. Что такое бинарное дерево?
3. Какие элементы дерева выделяют?
4. Как узнать является ли элемент дерева листом или корнем?
5. (*) Применение решающих деревьев для практических задач.
6. Что такое префиксный код?
7. Этапы алгоритма Шеннона-Фано?
8. Этапы алгоритма Хаффмана?
9. Что такое избыточность?
10. Как определить коэффициент сжатия?