***Бинарное дерево поиска. Деревья, часть 2.***

Рассмотрим бинарное дерево поиска.

Бинарное корневое дерево, которое назовем просто бинарным деревом, обеспечивает *прекрасный метод организации данных*, при котором любые конкретные данные можно легко найти или установить их отсутствие. Единственным требованием к построению бинарного дерева является введение на данных некоторого *линейного порядка*. Этим порядком может быть, к примеру, алфавитный или числовой порядок.

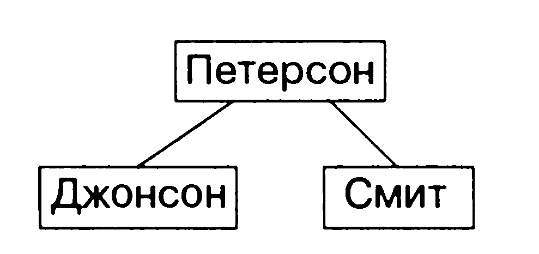
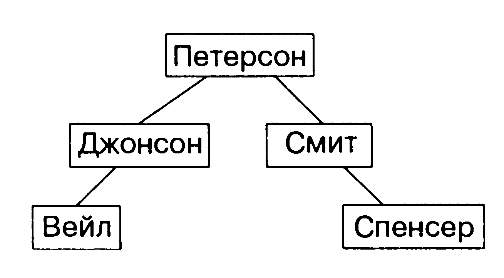
Построим бинарное дерево поиска, в каждом узле которого находится имя (или другой ключ). Для простоты примем расположение имен в алфавитном порядке.

Предположим, что нам необходимо *запомнить* имена: *Петерсон*, *Джонсон, Смит, Вейл, Спенсер, Рассел, Бауэр, Мартин, Уилсон*.

Начнем с имени *Петерсон* и поместим его в корень дерева. Поскольку следующее имя, *Джонсон*, в алфавитном порядке стоит перед именем *Петерсон*, делаем *Джонсона* левым сыном *Петерсона*. Следующее имя, *Смит*, в алфавитном порядке стоит после имени *Петерсон*, поэтому имя *Смит* становится правым сыном имени *Петерсон*, как показано на рис *8 а.*

Теперь рассмотрим имя *Вейл*. В алфавитном порядке имя Вейл стоит перед (или меньше, чем) именем *Петерсон*, спускаемся к левому сыну, *Джонсону*, и поскольку в алфавитном порядке имя *Вейл* расположено перед именем *Джонсо*н, делаем *Вейла* левым сыном *Джонсона*.

Далее рассматриваем имя *Спенсер*. Поскольку в алфавитном порядке имя Спенсер идет после имени *Петерсон*, спускаемся к правому сыну, *Смиту*. Но так как в алфавитном порядке имя *Спенсер* расположено после имени *Смита*, делаем *Спенсера* правым сыном *Смита*, как показано на рис*. 8 б.*

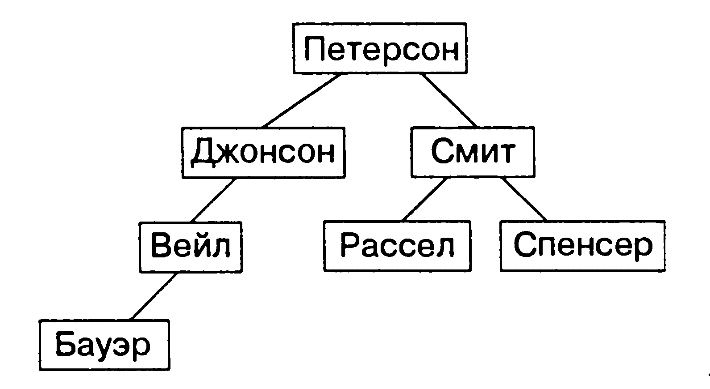
 

*Рис. 8а. Рис. 8б.*

Далее продолжаем строить дерево по тому же алгоритму. Предположим, что есть имя, которое необходимо поместить в дерево. Если в вершине дерева уже есть имя и вставляемое имя в алфавитном порядке стоит после этого имени, то следует спуститься к *правому* сыну, а если оно стоит до этого имени, то нужно спуститься к *левому* сыну. Если в вершине нет имени, то имя, которое нужно вставить, помещается в данную вершину.

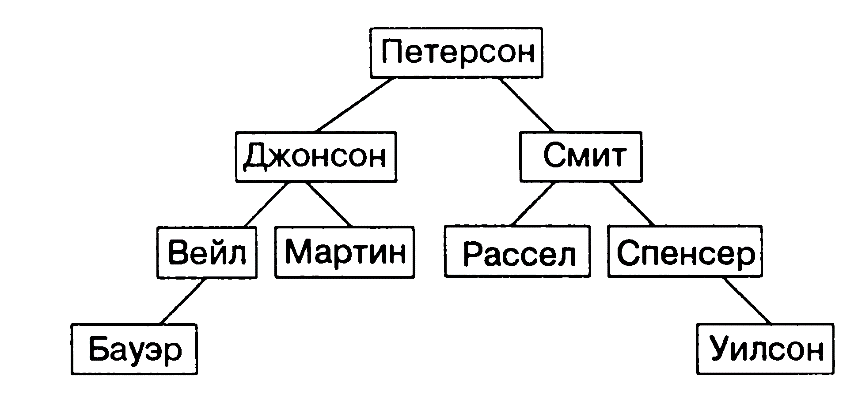
Следующее имя *Рассел*. Поскольку по алфавиту имя *Рассел* стоит после имени *Петерсон*, спускаемся к правому сыну, *Смит*у. Имя Рассел по алфавиту расположено перед именем *Смит*; делаем *Рассела* левым сыном *Смита*.

Следующее имя - *Бауэр*. По алфавиту оно стоит перед именем *Петерсон*, поэтому спускаемся к левому сыну, *Джонсону*. По алфавиту имя *Бауэр* стоит перед именем *Джонсон*, поэтому спускаемся к левому сыну, *Вейлу*. В алфавитном порядке имя *Бауэ*р располагается перед именем *Вейл*, поэтому спускаемся к левому сыну. Поскольку там нет имени, делаем *Бауэр* левым сыном *Вейла*, как показано на рис. *9а.*



*Рис. 9.*

У нас остались два имени: *Мартин* и *Уилсон*. Продолжая аналогичные рассуждения, делаем *Мартина* правым сыном *Джонсона*, а *Уилсона* – правым сыном *Спенсера*. Построенное бинарное дерево поиска приведено на *рис*. *10.*



*Рис. 10.*

Теперь приведем алгоритм *вставки* имени в дерево поиска, который реально создает дерево поиска, за исключением размещения имени в корне дерева. Используя < и >, следует иметь в виду, что сказанное применимо для любого упорядочения.

*Вставка* (элемент).

1. Начинаем с корня.

2. Если элемент < объекта в вершине, переходим к *левому* *сын*у.

3. Если элемент > объекта в вершине, переходим к *правому сыну*.

4. Повторяем шаги *2* и *3*, пока не достигнем вершины, которая не определена.

5. Если достигнутая вершина не определена, то определяем вершину и

вставляем элемент.

Рассмотрим *поиск* элемента в дереве. При поиске кроме проверки, является ли данное имя больше или меньше имени в вершине, *проверяется также совпадение этого имени с именем в вершине.* Если последнее выполняется, процесс поиска завершен. В противном случае повторяются предыдущие действия. Если достигается вершина, которая не определена, это означает, что данное имя не хранится в дереве.

Предположим, например, что в рассмотренном выше дереве разыскивается имя *Дженкинс*. Поскольку оно "меньше, чем имя Петерсон", идем к левому сыну, *Джонсону*. Но так как имя *Дженкин*с "меньше, чем имя *Джонсон*", идем к левому сыну, *Вейлу*. Поскольку данное имя "больше, чем *Вейл*" (на самом деле нет имени больше, чем *Вейл*), идем к правому сыну. Но он не определен, поэтому данное имя отсутствует в списке.

Приведем алгоритм поиска имени в дереве поиска.

*Поиск* (элемент).

1. Начинаем с корня.
2. Если элемент < объекта в вершине, идем к *левому сыну*.
3. Если элемент > объекта в вершине, идем к *правому сыну*.
4. Если элемент = объекту в вершине, то имя найдено; выполняем соответствующие действия и выходим.
5. Повторяем шаги *2, 3* и *4*, пока не достигнем вершины, которая не определена.
6. Если достигнутая вершина не определена и в дереве нет имени, то выполняем соответствующие действия и выходим.

Вполне очевидно, что *длина пути поиска* *элемента* не может превышать высоту дерева. Поэтому, если дерево полное, то в самом неблагоприятном варианте длина пути имеет порядок , где *n* - число элементов в дереве. Следовательно, если дерево сбалансированное, самый неблагоприятный случай имеет порядок  .

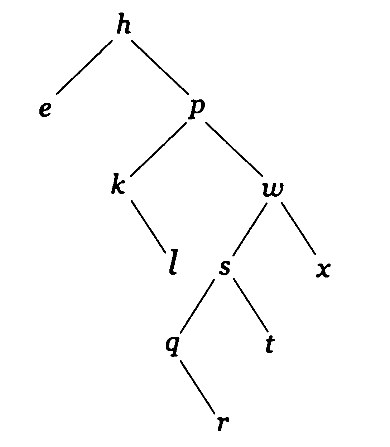
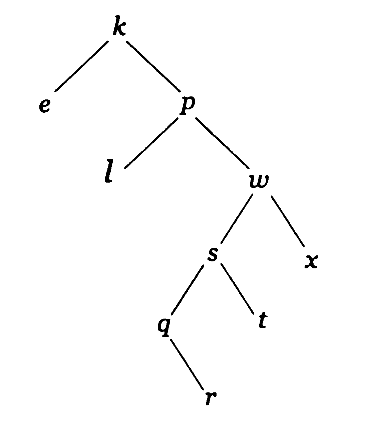
Рассмотрим *один из алгоритмов* удаления вершины из дерева поиска.

1. Если вершина  не имеет ни левого, ни правого поддерева, просто удаляем ее.
2. Если вершина  имеет одно поддерево , удаляем  и заменяем ее поддеревом (вставляем поддерево в вершину  ).
3. Если вершина  имеет два поддерева, то заменяем вершину  самым наибольшим элементом из левого поддерева или самым наименьшим злементом правого поддерева, а к освободившшемусю элементу (вершине *v*) опять применяем пункты *1-2*.

Рассмотрим пример.

Пусть задано бинарное дерево поиска, рис. *11 а*.

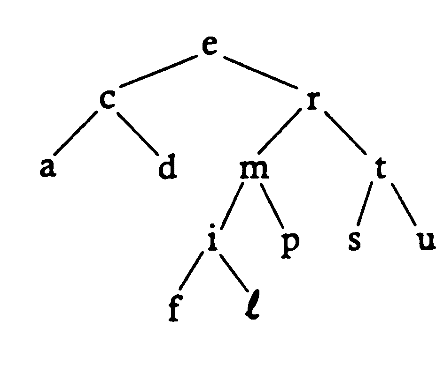
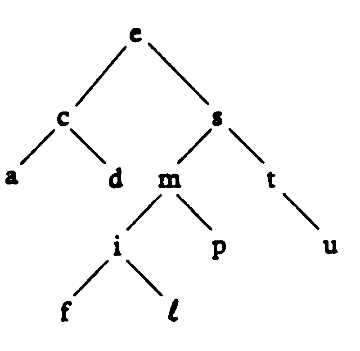
Удалим из него вершину *h*. Спускаемся из *h* к правому сыну *р*, а затем к левому сыну *k*. Поскольку *k* не имеет левого сына, это искомый элемент для замены. Итак, меняем *h* на *k*  и делаем *l* левым сыном *р*. В результате получаем дерево, изображенное на рис*. 11.б*.

*Рис. 11.а. Рис. 11.б.*

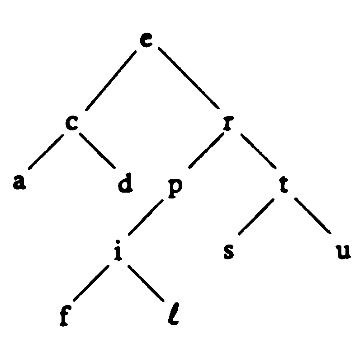
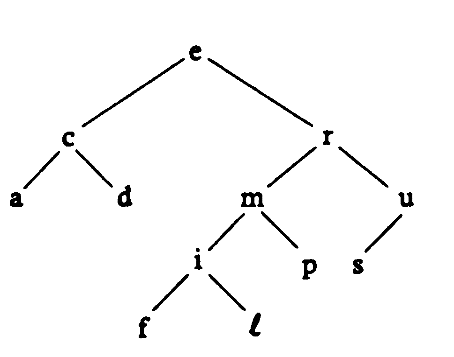
Рассмотрим дерево на рисунке *12.а* и будем удалять различные его вершины.

Удалили вершину *r*, рисунок *12 б.*

*Рис. 12.а. Рис 12.б.*

Теперь удалим вершину *m* (рисунок *13 а*) и вершину *t* (рисунок *13 а*).

*Рис. 13.а. Рис. 13.б.*