**БИОАЛГОРИТМИКА**

**11.08 Занятие 3.**

**1. Повторение**

**2. Задача склейки списков. Расширенные возможности работы со списками**

Обозначения. xA – текущий элемент списка A

Last(xA) - ДА, если текущий элемент последний, НЕТ - иначе

Next(xA) – следующий элемент списка

x.Data – данное, записанное в элементе списка x

2.1. Задача склейки списков

Дано: список A, список B

Получить: список A, состоящий из исходного списка, к которому в конце присоединен список B

Замечание. (Как всегда) в начале работы алгоритма текущий элемент списка – его заголовок.

Решение.

while NOT (IsLast(xA))

xA:=Next (xA)

end\_while

xB:=Next(xB)

while NOT IsLast(xB))

Insert (A, xB.Data) // текущий элемент – тот, который вставлен

end\_while

Insert (A, xB.Data)

Время работы: ~ length(A) +length(B)

2.2. Расширение возможностей: списки с прямым доступом к концу и указатели (pointer)

2.2.1. Списки с прямым доступом к концу

Обозначение. First(A) – заголовок списка

Last(A) - последний элемент списка

Считаем, что всегда можно объявить последний элемент списка текущим (это новая операция, разрешенная для списков)

2.2.2. Указатели.

До сих пор мы могли извлекать из ячейки списка только данное, которое там хранится (например, число). Ссылку на следующий элемент можно было использовать только для перехода к этому элементу (функция Next() )/

Разрешим использовать указатели так же, как числа – читать их значения и присваивать их. Только складывать и умножать указатели не будем ☺

Обозначения

pA – указатель на текущий элемент списка

x.Pointer – указатель, записанный в элементе списка x

Новый алгоритм сцепления списков

r:=xB.Pointer

Last(A).Pointer:=r

Более короткая запись:

Last(A).Pointer := xB.Pointer

Время работы – не зависит от длин списков. Ура!

Упражнение. Написать алгоритм вставки списка B после текущего элемента списка A

**3. Бинарные деревья поиска**.

3.1. Определение.

Замечание. Операция НАЙТИ как для массивов, так и для списков, требует просмотра всего списка (или массива). Если известно, что данные упорядочены, то это помогает, но все равно (в среднем) нужно просмотреть список (массив) до середины.

Новая структура – *сбалансированное бинарное дерево*

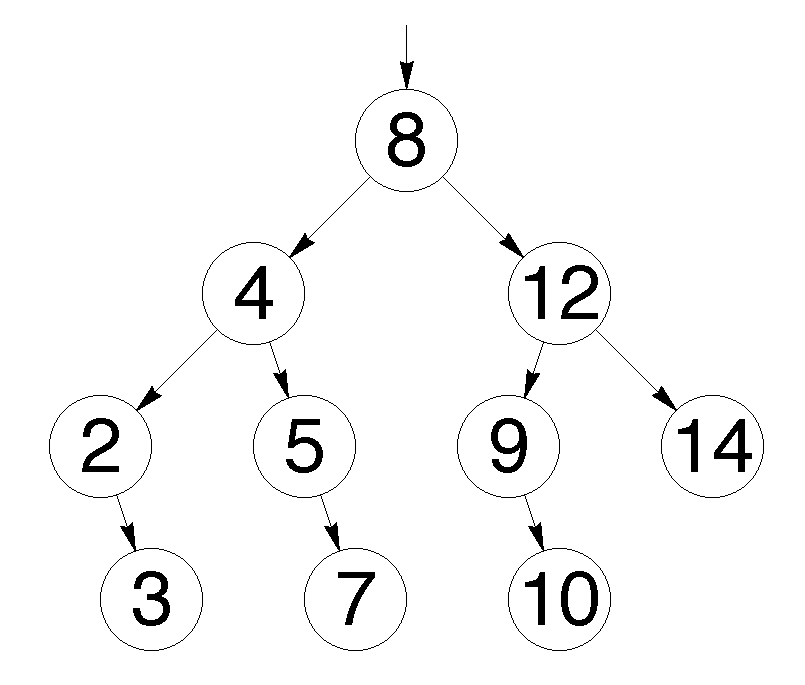
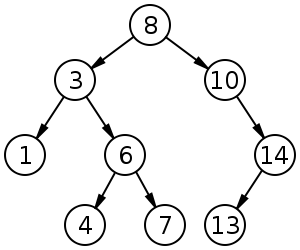


Рис.1 НЕ сбалансированное б.д. Рис.2. Сбалансированное б.д.

Сбалансированное бинарного дерева:

1. (Дерево поиска) Все числа в левом поддереве меньше числа в корне. Все числа в правом поддереве – больше.
2. (Сбалансированность) Количество узлов в левом и правом поддереве примерно одинаковое (отличаются не больше, чем на 1).

Время поиска в бинарном дереве поиска ~ высота дерева.

Высота сбалансированного бинарного дерева поиска ~logN (N – количество узлов)

3.2. Построение сбалансированного бинарного дерева.

Если все элементы известны заранее, то строить легко.

Берем средний элемент, приписываем его к корневой вершине. Если количество элементов четное – берем больший из двух средних.

Далее точно так же строим левое и правое поддерево.

Если элементы поступают по одному в произвольном порядке, то придется на ходу перестраивать уже построенное дерево – делать балансировку. Это можно сделать за время ~ N, где N – текущее количество элементов в дереве. Про это не будем.

Вместо этого разберем другую структуру – дерево отрезков

*Продолжение следует*