Лекция 6
Абстрактные типы данных

(c) Д.Н. Лавров
2017

Определение

- Определим **абстрактный тип данных** (АТД) как математическую модель с совокупностью операторов, определенных в рамках этой модели.

- **Абстрактный тип данных (АТД)**

- это математическая модель для типов данных, где тип данных определяется поведением (семантикой) с точки зрения *пользователя* данных, а именно в терминах возможных значений, возможных операций над данными этого типа и поведения этих операций.

Обсуждение

- Простым примером АТД могут служить множества целых чисел с операторами объединения, пересечения и разности множеств.
- В модели АТД операторы могут иметь операндами не только данные, определенные АТД, но и данные других типов: стандартных типов языка программирования или определенных в других АТД.
- Результат действия оператора также может иметь тип, отличный от определенных в данной модели АТД.
- Но мы предполагаем, что по крайней мене один операнд или результат любого оператора имеет тип данных, определенный в рассматриваемой модели АТД.

Цели построения АТД

- АТД можно рассматривать как обобщение простых типов данных (целых и действительных чисел и т.д.), точно так же, как процедура является обобщением простых операторов (+, - и т.д.).
- АТД инкапсулирует типы данных в том смысле, что определение типа и все операторы, выполняемые над данными этого типа, помещаются в один раздел программы.
- Если необходимо изменить реализацию АТД, мы знаем, где найти и что изменить в одном небольшом разделе программы, и можем быть уверенными, что это не приведет к ошибкам где-либо в программе при работе с этим типом данных.

- Концепция АТД реализует принцип ***инкапсуляции***, то есть прячет конкретную реализацию типа от пользователя-программиста, предоставляя интерфейс работы со СД.
- Понятие АТД и его применение на практике появилось задолго до появления ООП и является его предтечей.
- Использовать АТД можно и в не ОО-языках.
- Набор действий (алгоритмов работы) со СД называется **интерфейсом абстрактного типа данных**. То, как реализован АТД, самим АТД *не определяется*.
- *Абстрактный тип данных* - это *интерфейс*, набор операций без реализации. Реализацию определяет сам программист.

Краткая формула

- АТД = ММ + СД + А

  ММ – Математическая модель

  СД – Структура данных

  А – Алгоритмы работы с СД

СД и АТД
!!! Не путайте АТД и СД !!!
АТД != СД

СД

- Массив
- L1-список
- L2-список
- Хэш-таблица
- Структура
- Двоичное дерево
- Сбалансированное дерево поиска
- Красно-черные деревья

АТД

- Список *
- Стэк *
- Дэк
- Очередь
- Множество
- Дерево
- Словарь *

Примеры структур данных

- Массив (для C, C++, Java, C#) –статическая структура
- Курсоры – массив специального типа
- Циклический массив –статическая структура
- L1-список – Динамическая структура
- L2-список – Динамическая структура
- Хэш-таблица – Динамическая структура
- Сбалансированное дерево поиска – Динамическая структура
- Красно-черные деревья – Динамическая структура

Примеры АТД

- Список (L1-список)
- Стек
- Очередь
- Множество
- Ассоциативный массив (Словарь)
- Дерево

Понятие массовой операции

O-нотация

$$T = O(n) \Leftrightarrow \exists\, C = const \;:\; T < C \cdot n$$

T – трудоемкость алгоритма

C – константа

n – объем входных данных

Пример. Поиск элемента в по массиву

- Трудоемкость последовательного поиска O(n)
- Трудоемкость бинарного поиска по отсортированному массиву O(log(n))

Массовые операции

- В основе реализации АТД лежит СД.
- СД определяет алгоритмы работы с ней.
- Алгоритмы определяют массовые операции (МО). МО – это операция с временем выполнения O(n) или хуже.
- **Вывод**: Разные реализации АТД имеют разные наборы массовых операций

Список

Мат.модель – последовательность, ряд.
В математике список представляет собой последовательность элементов определенного типа.

Как правило, подразумевается последовательный доступ к элементам списка.

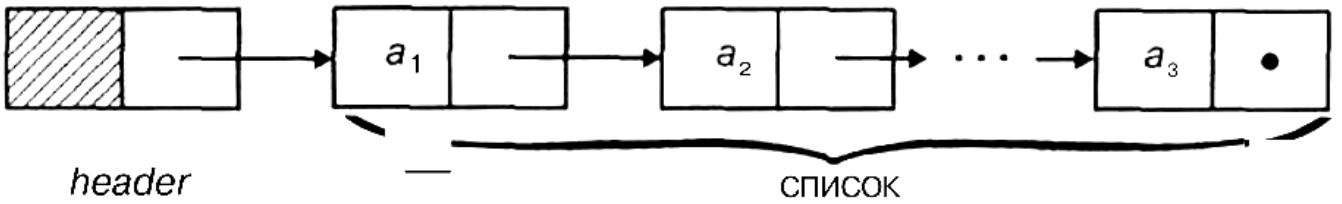
Замечание. Часто списком называется и СД и АТД. Разобраться можно только из контекста.

Список. СД для реализации

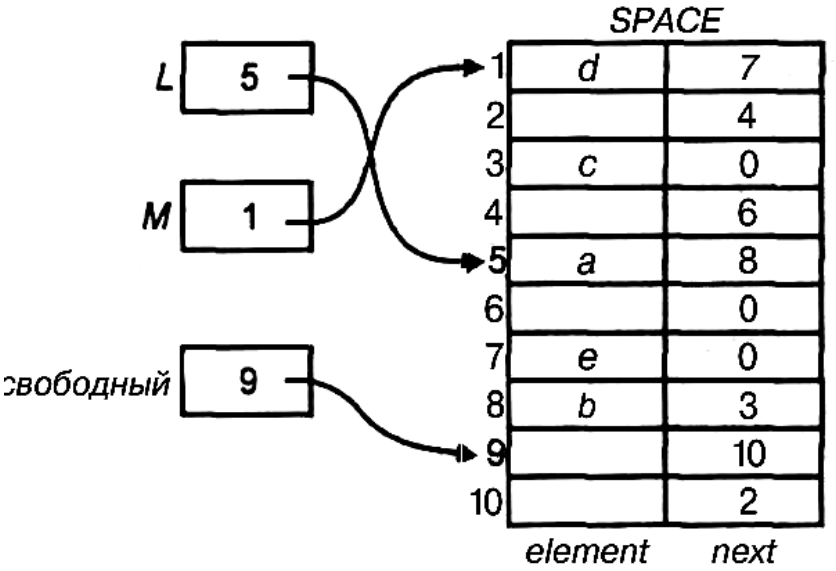
- Массив

K	0	1	2	3	4	5	6	7	8	9	10	11
A[k]	5	7	8	3	2	34	2	-6	5	8	7	12

- L1-список



- Курсоры



Операции списка по Ахо

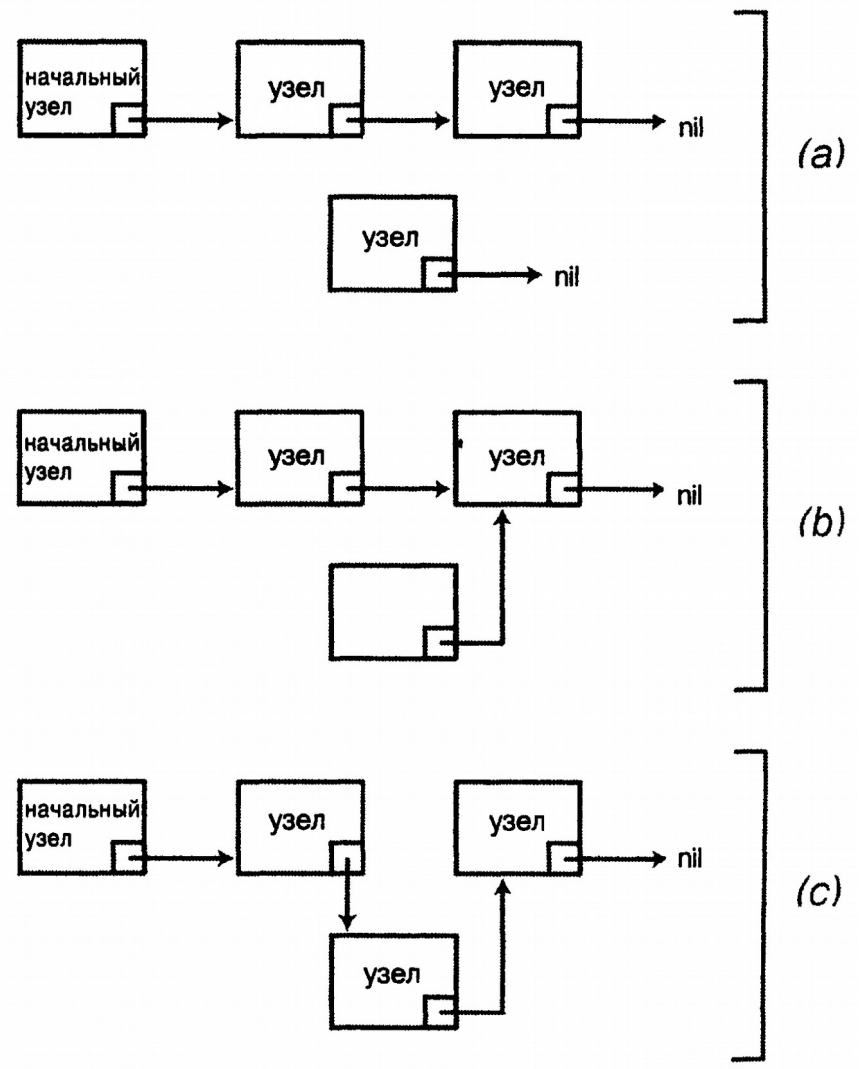
- insert(x,p,L)
- locate(x,L)
- retrieve(p,L)
- delete(p,L)
- next(p,L)
- makeNull(L)
- first(L)

Анализ операции insert

Массив – O(n)

L1-список - O(1)

Курсоры - O(1)



Сравнение реализаций для C/C++

	Массив	**L1-список**
insert(x,p,L)	O(n)	O(1)
locate(x,L)	O(n)	O(n)
retrieve(p,L)	O(1)	O(n)
delete(p,L)	O(n)	O(1)
next(p,L)	O(1)	O(1)
makeNull(L)	O(1)	O(n)
first(L)	O(1)	O(1)

Сопоставление операций списка

	По Ахо	**Python**
•	insert(x,p,L)	L.insert(p, x)
•	locate(x,L)	L.index(x)
•	retrieve(p,L)	L[p]
•	delete(p,L)	L.delete(x) – уд. пер. вх.
•	next(p,L)	L[p+1]
•	makeNull(L)	L=[ ]
•	first(L)	L[0]

Эффективность встроенной реализации

Операция	Эффективность
index []	O(1)
Присваивание по индексу	O(1)
append	O(1)
pop()	O(1)
pop(i)	O(n)
insert(i,item)	O(n)
оператор del	O(n)
итерирование	O(n)
вхождение (in)	O(n)
срез [x:y]	O(k)
удалить срез	O(n)
задать срез	O(n+k)
обратить	O(n)
конкантенация	O(k)
сортировка	O(n log n)
размножить	O(nk)

Стек

Стек — это специальный тип списка, в котором все вставки и удаления выполняются только на одном конце, называемом вершиной (top). Стеки также иногда называют "магазинами", а в англоязычной литературе для обозначения стеков еще используется аббревиатура LIFO (last-in-first-out — последний вошел — первый вышел).

Операции по Ахо

- top(s)
- makeNull(s)
- pop(s)
- push(x, s)
- isEmpty(s)

В Python реализуется доп. функциями встроенного списка: pop(), append() и др.

Получается, что один АТД построен на основе другого АТД.

Очередь

- Другой специальный тип списка — очередь (queue), где элементы вставляются с одного конца, называемого задним (rear), а удаляются с другого, переднего (front).
- Очереди также называют "списками типа FIFO" (аббревиатура FIFO асшифровывается как first-in-first-out: первым вошел — первым вышел).
- В функционал очереди в Python реализуется списком.

Очередь применение

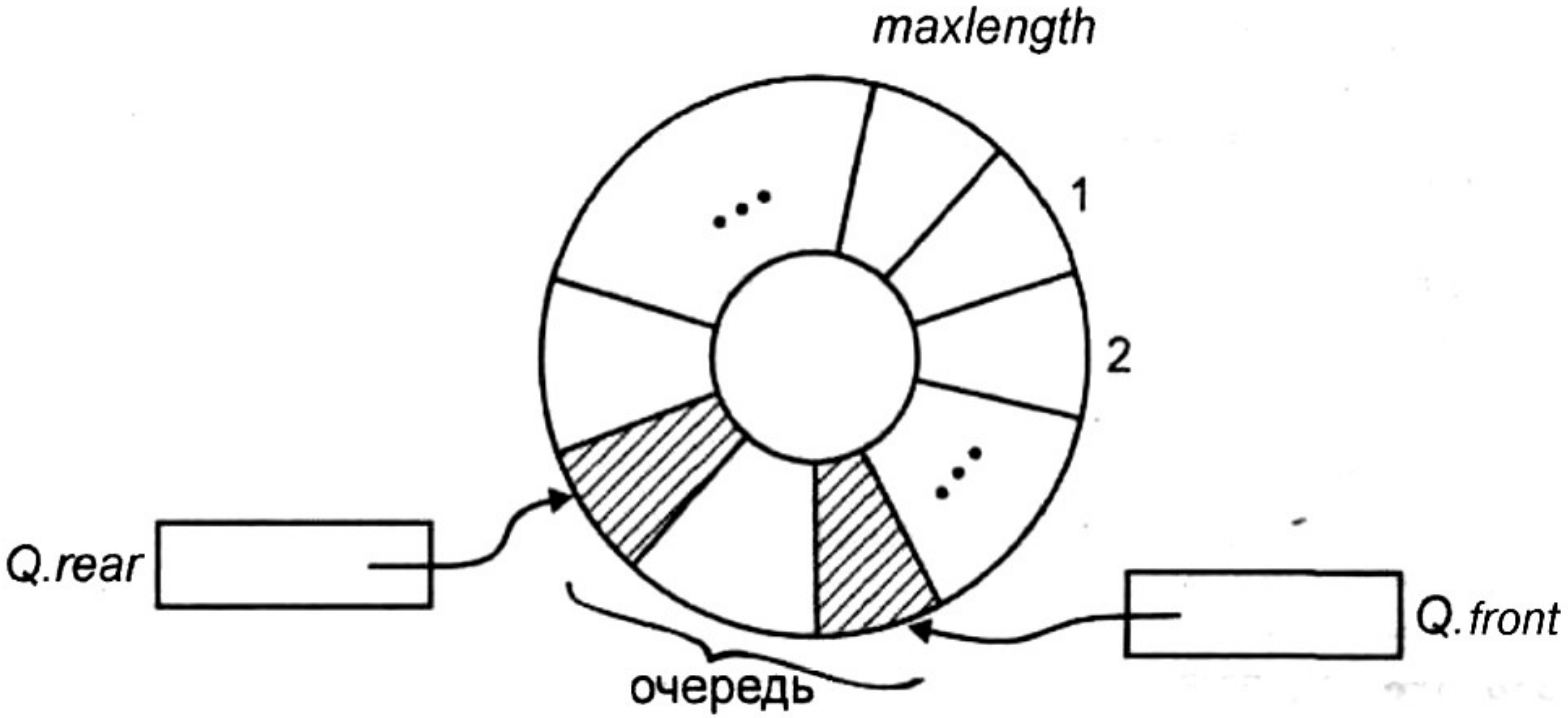
- Реализация хэш-таблиц открытого типа
- Диспетчерезация событий
- Использование в пулах обработки потоков
- Организация очередней отправки пакетов на маршрутизаторах
- и .т.п.

Операции очереди по Ахо

- makeNull(Q)
- front(Q)
- enqueued(x, Q) - вставляет элемент x в конец очереди Q.
- dequeue(Q) - удаляет первый элемент очереди Q.
- isEmpty(Q)

Реализации

- Указатели (списки)
- Массивы
- Циклические массивы



Трудоемкость реализаций очереди

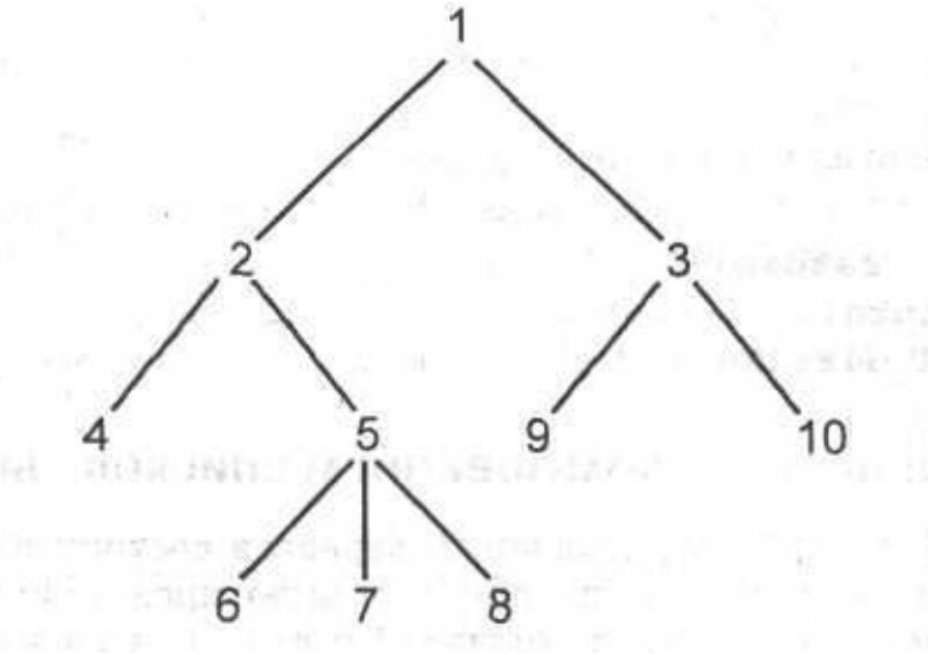
	Указатели	Массивы	ЦМ
makeNull(Q)	O(n)	O(1)	O(1)
front(Q)	O(1)	O(1)	O(1)
enqueued(x, Q)	O(1)	O(1*)	O(1)
dequeue(Q)	O(1)	O(n*)	O(1)
isEmpty(Q)	O(1)	O(1)	O(1)

Дерево

- **parent**(n, T). Эта функция возвращает родителя (parent) узла n в дереве T. Если n является корнем, то в этом случае возвращается L. Здесь L обозначает "нулевой узел" и указывает на то, что мы выходим за пределы дерева.
- **leftmostChild**(n, T). Данная функция возвращает самого левого сына узла n в дереве T. Если n является листом (и поэтому не имеет сына), то возвращается L.
- **rightSibling**(n, T). Эта функция возвращает правого брата узла n в дереве T и значение L, если такового не существует. Для этого находится родитель p узла n и все сыновья узла p, затем среди этих сыновей находится узел, расположенный непосредственно справа от узла n.
- **label**(n, T). Возвращает метку узла n дерева T. Для выполнения этой функции требуется, чтобы на узлах дерева были определены метки.
- **CreateI**(v,T1, T2 , ..., Ti) — это обширное семейство "созидающих" функций, которые для каждого i = 0, 1, 2, ... создают новый корень n с меткой v и далее для этого корня создает i сыновей, которые становятся корнями поддеревьев T1, T2 , ..., Ti . Эти функции возвращают дерево с корнем r. Если i = 0, то возвращается один узел n, который одновременно является и корнем, и листом.
- **root**(T) возвращает узел, являющимся корнем дерева T. Если T — пустое дерево, то возвращается L.
- **makeNull**(r). - Этот оператор делает дерево T пустым деревом.

Реализации деревьев

- Курсоры на родителей
- Структура (запись, класс) со ссылкой на L1-список сыновей (дочерних узлов)
- Представление левых сыновей и правых братьев



Представление через левых сыновей и правых братьев

	Leftmost child	Label	Right sibling
2		D	
5		B	11
		...	
10	5	A	
11	2	C	

Бинарное дерево поиска

- Главное отличие от обычного дерева, жесткий порядок и число потомков.
- У такого дерева два потомка: левый и правый

Возможная реализация на классах

```
class TreeNode:
    def __init__(self):
        self.data = ""
        self.left = -1
        self.right = -1

    def initNode(self, left, right):
        self.left = left
        self.right = right

    def setData(self, data):
        self.data = data
```

```
class BinaryTree:
    def __init__(self):
        self.storage = [ ]
        self.nodeNum = 0

    def addNode(self, left, right,
     data):
        node = TreeNode()
        node.initNode(left, right)
        node.setData(data)
```

Примеры

- Игра «Угадай животное»
- Субоптимальное кодирование Хаффмана

Литература и ссылки по теме

http://citforum.ru/programming/theory/adt/

http://it.mmcs.sfedu.ru/wiki/Основы_программирования_—_второй_семестр_08-09;_Михалкович_С.С.;_V_часть

Альфред В. Ахо, Джон Э. Хопкрофт, Джеффри Д. Ульман Структуры данных и алгоритмы, М.: Вильямс, 2003. 400с.

http://aliev.me/runestone/index.html