АЛГОРИТМЫ И СТРУКТУРЫ ДАННЫХ

Лекции – 18

Лабораторные работы – 18

Самостоятельная работа – 108

РГР

Зачет с оценкой.

ЦЕЛИ И ЗАДАЧИ КУРСА

Цель курса– получение знаний о различных структурах данных и приемах организации обработки данных.

Основные задачи дисциплины:

- получить навыки организации структуры данных, используемых для конкретных задач;
- получить навыки программирования алгоритмов обработки данных.

ЛИТЕРАТУРА

- Конова, Е.А. Алгоритмы и программы. Язык С++ [Электронный ресурс]: учебное пособие / Е.А. Конова, Г.А. Поллак. – М.: Лань, 2017. – 384 с. – ЭБС Лань.
- Белик, А. Г. Теория и технология программирования [Текст] : учеб. пособие / А. Г. Белик, В. Н. Цыганенко ; ОмГТУ. - Омск : Изд-во ОмГТУ, 2013. - 85 с.;
- Белик, А. Г. Проектирование и архитектура программных систем [Текст] : учеб. пособие / А. Г. Белик, В. Н. Цыганенко ; ОмГТУ. - Омск : Изд-во ОмГТУ, 2016. - 94 с.;
- Цыганенко, В. Н. CALS/CASE-технологии проектирования информационных систем [Электронный ресурс] : учеб. электрон. изд. Локального распространения: конспект лекций / В. Н. Цыганенко ; ОмГТУ. - Электрон. текстовые дан. (1,23 Мб.). - Омск : Изд-во ОмГТУ, 2017. - 1 эл. опт. диск (CD-ROM);
- Запорожец, Д.Н. Алгоритмы и анализ сложности [Электронный ресурс] : учеб. пособие / Д.Н. Запорожец. - Омск : Изд-во ОмГТУ, 2014. - 1 эл. опт. диск (CD-ROM).
- Алгоритмы анализа структуры сигналов и данных [Текст] : монография / А. С. Гуменюк [и др.] ; ОмГТУ. - Омск : Изд-во ОмГТУ, 2010. - 271 с. – ЭБС АРБУЗ.

Программы и алгоритмы

Основное назначение компьютера – обработка информации, для чего необходимо выполнить определенный набор операций - **программу**.

Программа – набор инструкций, описывающих последовательность действий, приводящих к результату.

Программу можно написать на машинном языке или используя специальные языки называемые языками программирования.

Программа на языке программирования преобразуется в машинные команды, которые затем выполняются компьютером.

Программы и алгоритмы

Алгоритм – это конечная последовательность четко определенных действий, задающая обработку исходных данных с целью получения нужного результата.

Свойства алгоритмов

1. Массовость (обеспечение функций алгоритма для большой совокупности данных)
2. Дискретность (возможность представить алгоритм в виде отдельных последовательных шагов)
3. Определенность (каждый шаг алгоритма должен быть четко определен и однозначно понятен)

Свойства алгоритмов

4. Результативность (получение нужного результата)
5. Конечность (выполнение алгоритма за конечное число шагов)

Способы представления алгоритма

1. Описательная форма (на естественном языке)
2. Псевдокод (описательная форма с ограниченным числом элементов)
3. Графическая форма (схема алгоритма)
4. Табличная форма (таблицы решений)

Основные конструкции псевдокода

Псевдокод:

1. Следование

…
Действие 1
Действие 2
…

2. Ветвление

…
Если Условие
    то Действие 1
    иначе Действие 2
Все-если
…

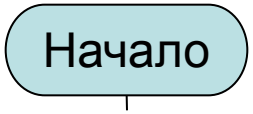
3. Цикл-пока

…
Цикл-пока Условие
    Действие
Все-цикл
…

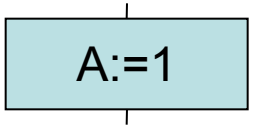
Схемы алгоритмов

Обозначения ГОСТ 19.701 – 90

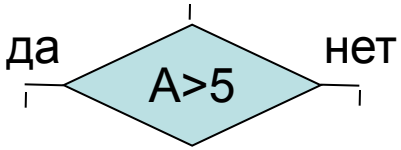
1. Терминатор (начало/конец)



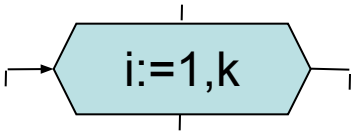
2. Процесс (вычисления)



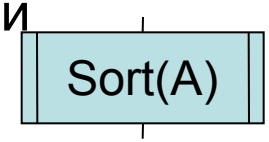
3. Анализ (проверка)



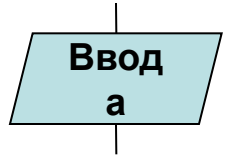
4. Модификатор (автоматическое изменение)



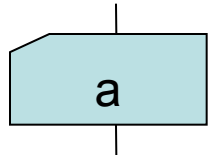
5. Предопределенный процесс (подпрограмма)



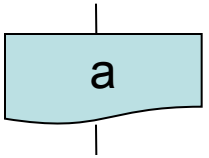
6. Ввод/вывод данных



7. Ввод с перфокарт

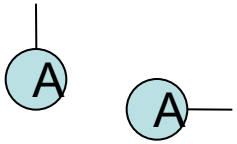


8. Вывод на принтер



9. Комментарий

10. Соединитель



Таблицы решений

Таблица составляется следующим образом.

В столбик выписываются все условия, от которых зависят дальнейшие вычисления, а по горизонтали - все случаи для вычислений.

На пересечении каждого столбца и строки ставят букву Y, если для данного решения данное условие должно выполняться, букву N, если данное условие обязательно должно не выполняться, и прочерк, если исход сравнения не важен.

Например, для алгоритма вычисления корней квадратного уравнения можно составить следующую таблицу:

Таблицы решений

	Нет корней	$x = -b / 2a$	$x=\pm(-b\sqrt{D}) / 2a$
$D < 0$	Y	N	N
$D = 0$	N	Y	N
$D > 0$	N	N	Y

Этапы создания ПО

1. **Постановка задачи** – неформальное описание задачи
2. **Анализ и уточнение требований** – формальная постановка задачи и выбор метода решения
3. **Проектирование** – разработка структуры ПО, выбор структур данных, разработка алгоритмов, определение особенностей взаимодействия с программной средой
4. **Реализация** – составление программ, тестирование и отладка
5. **Модификация** – выпуск новых версий

Пример разработки программы

1. **Постановка задачи**: Разработать программу, которая определяет наибольший общий делитель двух целых чисел.
2. **Анализ и уточнение требований:**

   1) *Функциональные требования*

   исходные данные: a, b – натуральные числа; 0 < a, b < **?** ;

   результат: x – натуральное число, такое, что

$x = \max \{y_i \,/\, i = \overline{1,n}\}$, где $((a \bmod y_i) = 0) \,\&\, (b \bmod y_i) = 0)$

   Метод решения:

   а) найти делители Y = { y_i } и определить x = max {Y};

   б) метод Евклида

Пример 1:

a	**b**
24	18
6	18
6	12
6 =	6

Пример 2:

a	**b**
3	4
3	1
2	1
1 =	1

Пример разработки программы

2) *Эксплуатационные требования*:

а) процессор – не ниже Pentium;

б) операционная система – Windows XP (консольный режим);

в) предусмотреть запрос на ввод данных с клавиатуры;

г) результаты вывести на экран дисплея.

3) *Технологические требования*:

а) язык программирования: C++;

б) среда программирования: Microsoft Visual Studio .Net 2003;

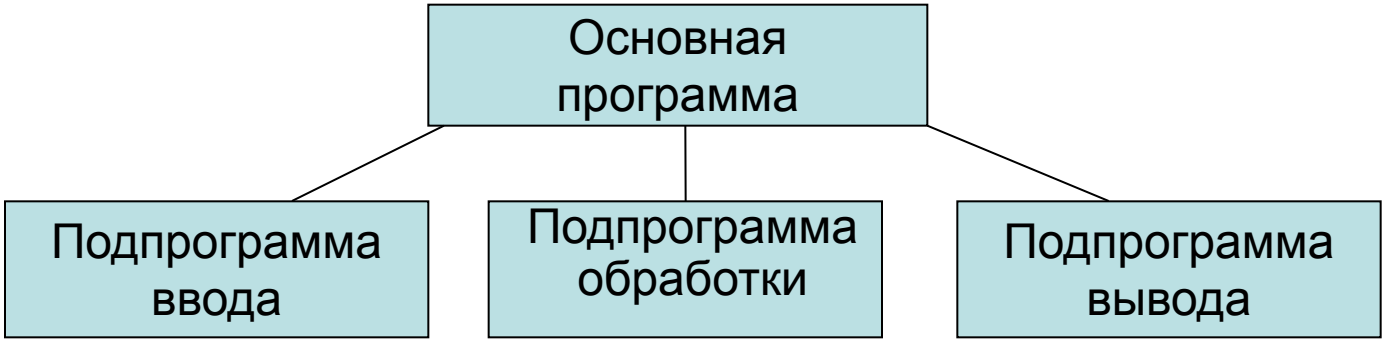
в) технология: структурный подход.

Пример разработки программы

3. Проектирование

Виды проектной документации:

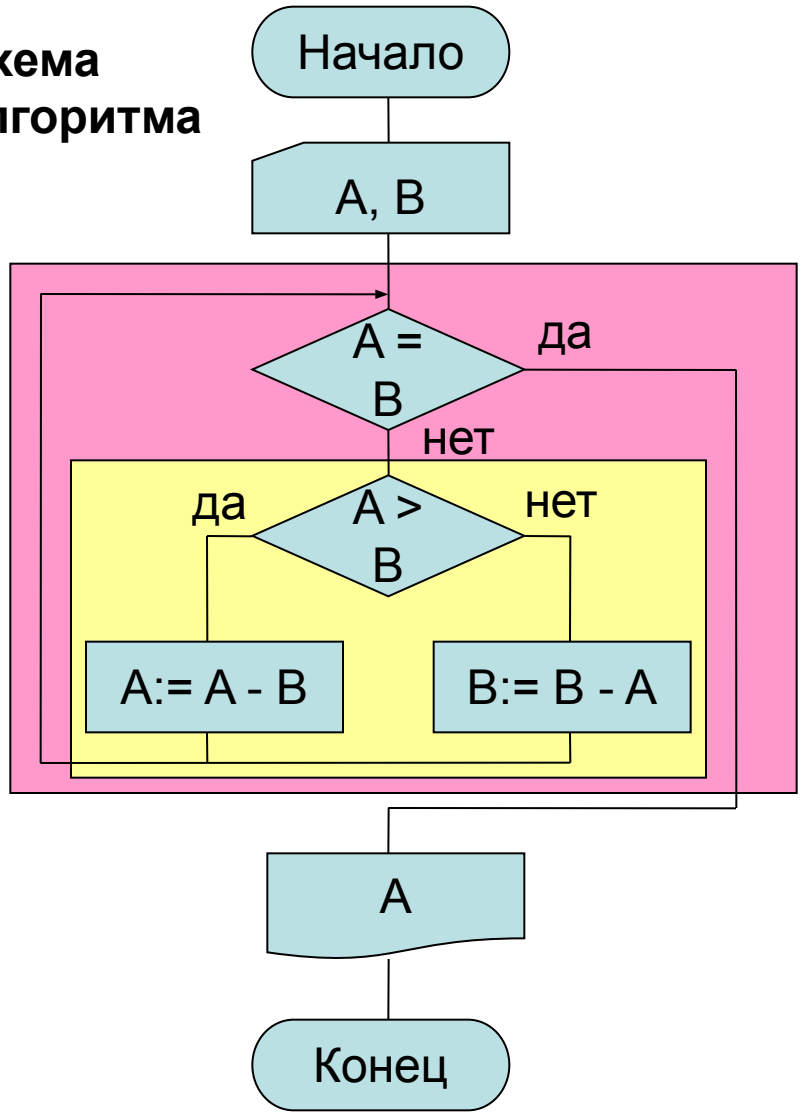
Структурная схема ПО – показывает взаимодействие по управлению основной программы и подпрограмм.



Алгоритм основной программы и подпрограмм в соответствии с выбранным способом представления

Пример разработки программы

Схема алгоритма



Алгоритм на псевдокоде

Начало

Ввести A,B

Цикл-пока A ≠ B

Если A > B

то A := A – B

иначе B := B – A

Все-если

Все-цикл

Вывести A

Конец

4. Реализация программы, ее тестирование и отладка.

Схема процесса подготовки программы

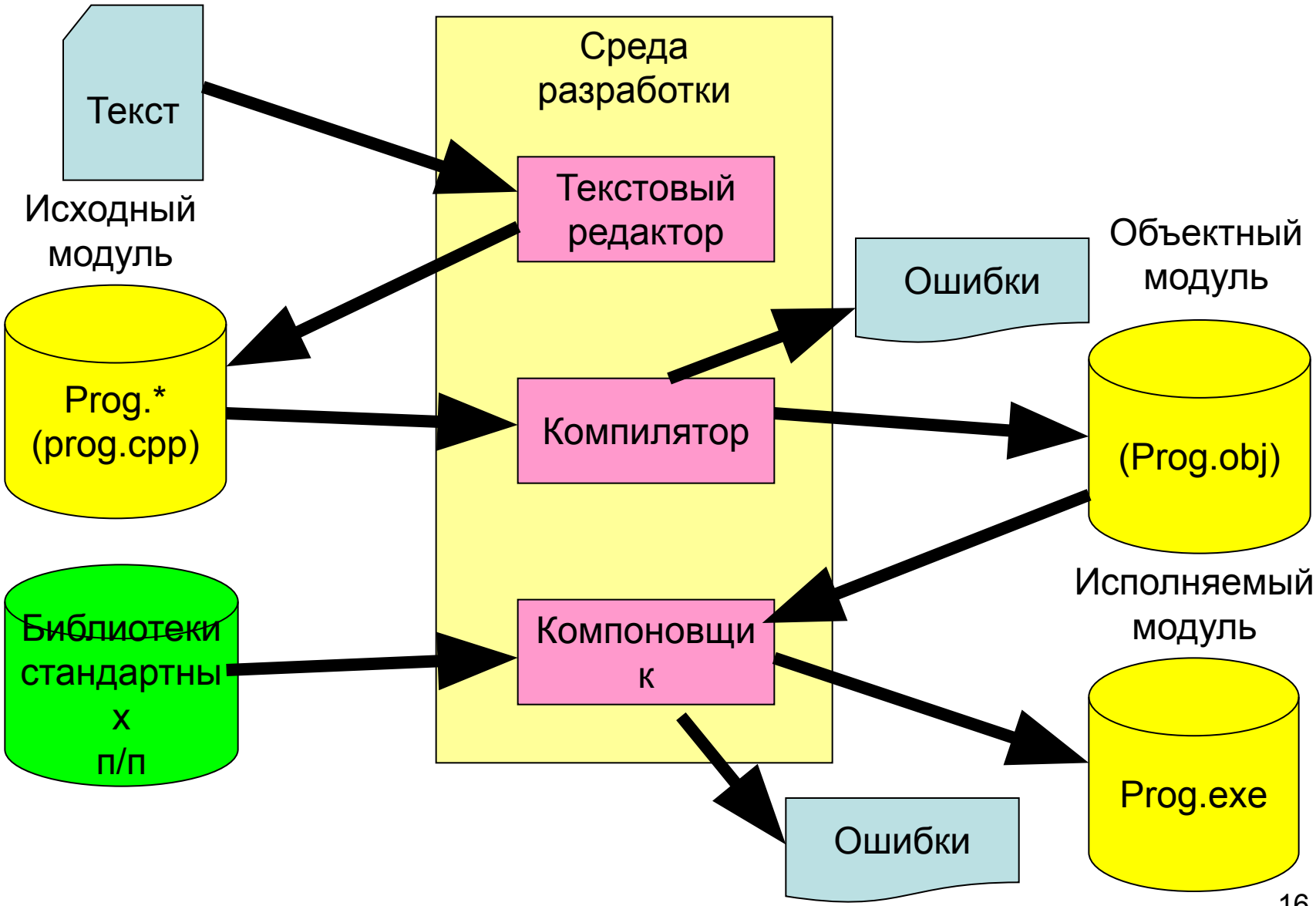
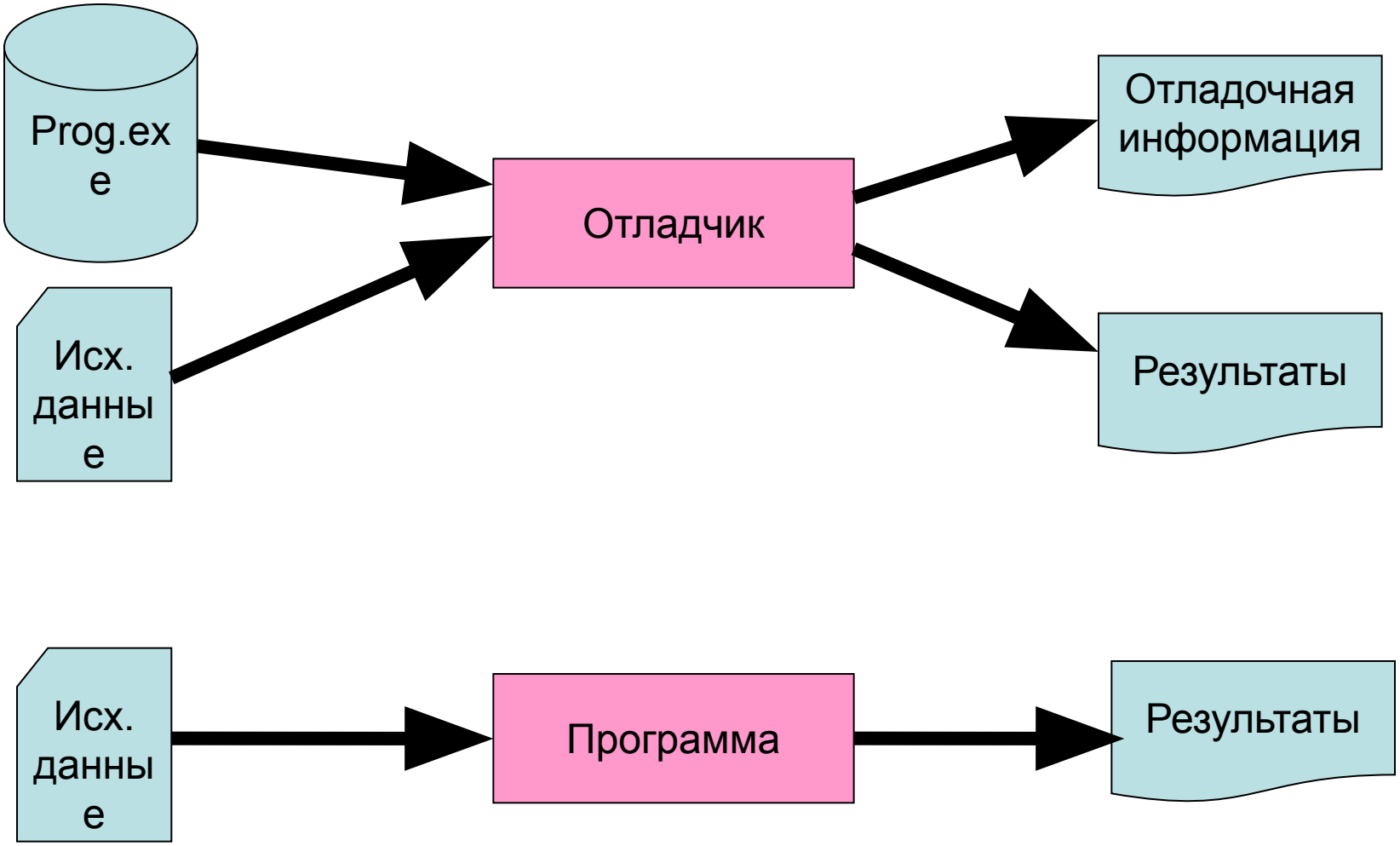


Схема процесса отладки и выполнения



Структуры данных

Структура данных – это способ хранения и организации данных, облегчающих доступ к этим данным и их модификацию.

Ни одна структура данных не является универсальной и не может подходить для всех целей, поэтому важно знать преимущества и ограничения, присущие им.

Классификация структур данных

По сложности:

простые

интегрированные

По способу представления:

физическая

логическая

По наличию связей между элементами данных:

несвязные

связные

По изменчивости:

Статические

полустатические

динамические

По характеру упорядоченности элементов в структуре:

линейные

нелинейные

По виду памяти, используемой для сохранности данных:

для оперативной

для внешней памяти

СВЯЗНЫЙ СПИСОК

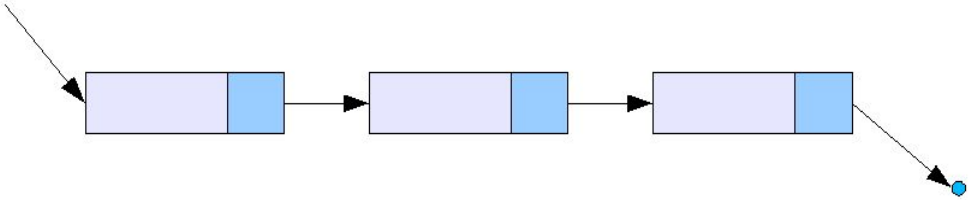
Связный список — базовая динамическая структура данных, состоящая из узлов, каждый из которых содержит как собственно данные, так и одну или две ссылки («связки») на следующий и/или предыдущий узел списка.

Принципиальным преимуществом перед массивом является структурная гибкость: порядок элементов связного списка может не совпадать с порядком расположения элементов данных в памяти компьютера, а порядок обхода списка всегда явно задаётся его внутренними связями.

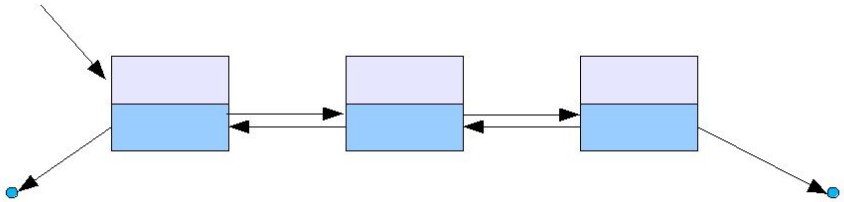
Существуют односвязные списки (одна ссылка на следующий элемент), двусвязные списки (ссылка на предыдущий и последующий элементы), кольцевые и др.

СВЯЗНЫЙ СПИСОК

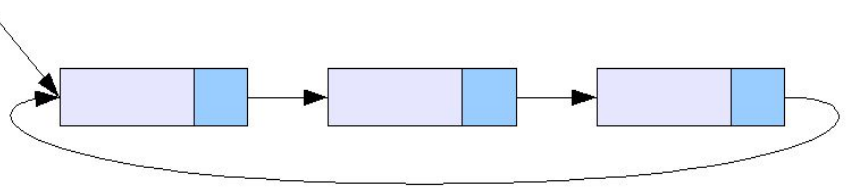
Односвязный список



Двусвязный список



Кольцевой связный список



ОПЕРАЦИИ СО СТРУКТУРАМИ ДАННЫХ

Search

Insert

Delete

Minimum

Maximum

Sort

СТРУКТУРА ДАННЫХ СТЕК

Стек (stack) — абстрактный тип данных, представляющий собой список элементов, организованных по принципу ***LIFO*** (last in — first out, «последним пришёл — первым вышел»).

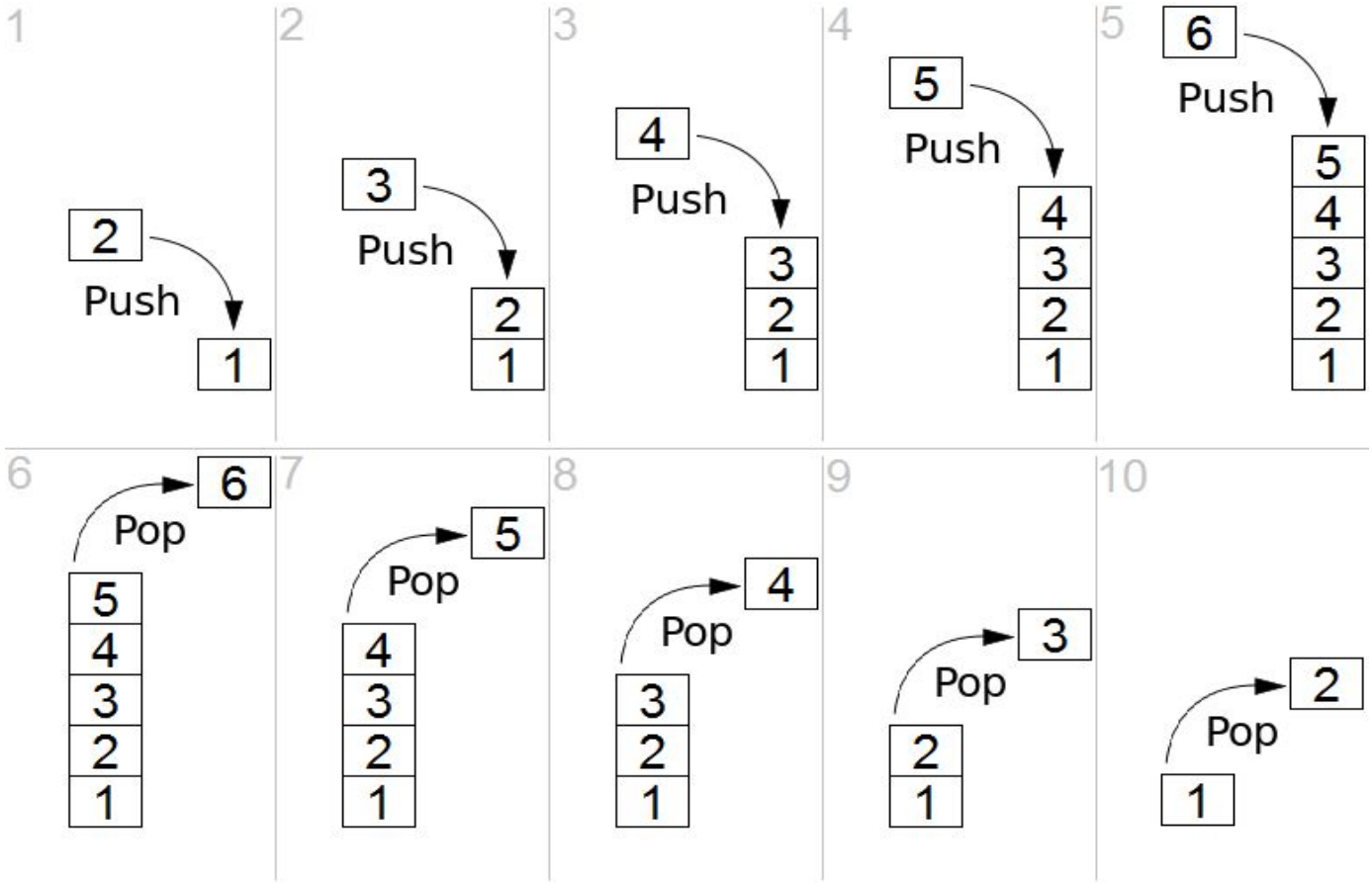
Чаще всего стек реализуется в виде однонаправленного списка (каждый элемент в списке содержит помимо хранимой информации в стеке указатель на следующий элемент стека).

Но также стек можно расположить в одномерном массиве с упорядоченными адресами. При этом отпадает необходимость хранения в элементе стека явного указателя на следующий элемент стека, что экономит память. При этом указатель стека (***Stack Pointer***) обычно является регистром процессора и указывает на адрес головы стека.

Регистр процессора — блок ячеек памяти, образующий сверхбыструю оперативную память внутри процессора; используется самим процессором и большой частью недоступен программисту: например, при выборке из памяти очередной команды она помещается в регистр команд, к которому программист обратиться не может.

СТРУКТУРА ДАННЫХ СТЕК

Возможны три операции со стеком: добавление элемента (иначе проталкивание, ***push***), удаление элемента (***pop***) и чтение головного элемента (***peek***).



СТРУКТУРА ДАННЫХ ОЧЕРЕДЬ

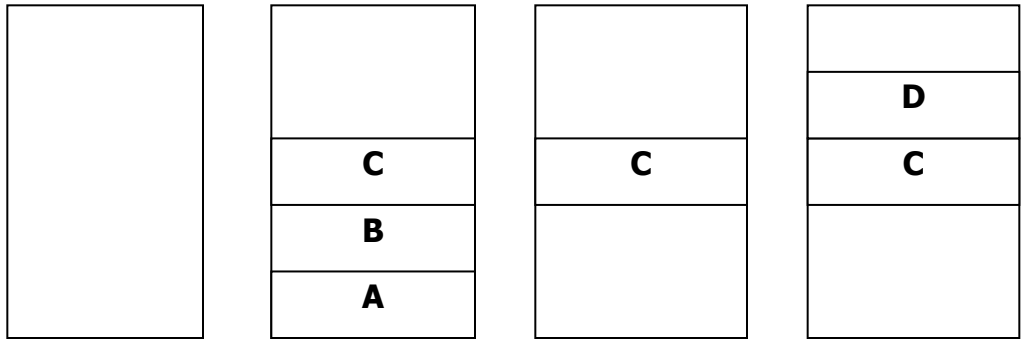
Очередью (или очередью FIFO - first in first out) называется такая последовательность элементов с переменной длиной, в которой включение элементов выполняется только с одной стороны списка (эту сторону часто называют концом или хвостом очереди), а чтение (исключение) - с другой стороны (называемой началом или головой очереди).

Стратегия "первым вошел - первым вышел"

Основные операции над очередью - те же, что и над стеком: включение, исключение, определение размера, очистка.

ЛОГИЧЕСКАЯ СТРУКТУРА ОЧЕРЕДИ

Схематичное представление логической структуры очереди



а) б) в) г)

а) пустая очередь;

б) после последовательного включения в очередь элементов A, B, C;

в) после исключения из очереди двух элементов;

г) после включения в очередь элемента D.

Дек

Дек - особый вид очереди (от англ. deq - double ended queue) - это такой последовательный список, в котором как включение, так и исключение элементов может осуществляться с любого из двух концов списка.

Операции над деком:

включение элемента справа;

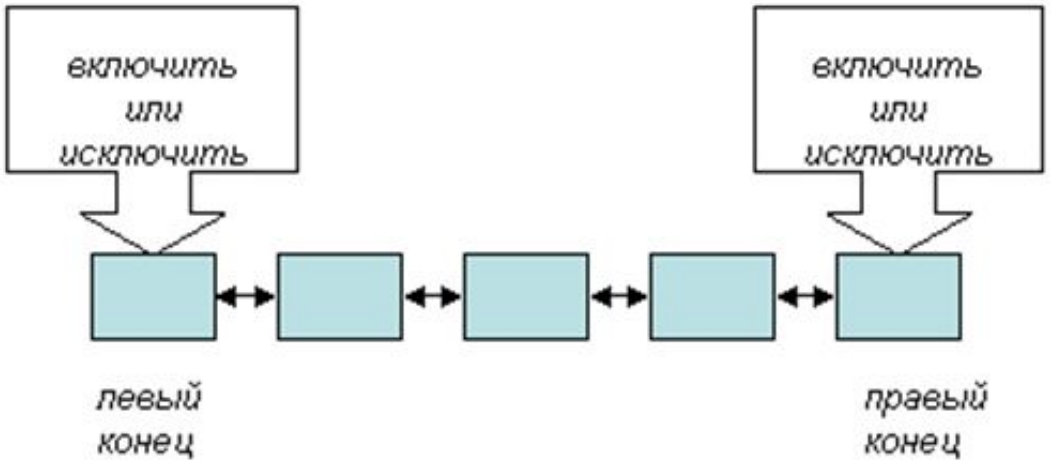
включение элемента слева;

исключение элемента справа;

исключение элемента слева;

определение размера;

очистка.



ЗАДАЧИ, РЕШАЕМЫЕ ПРИ ПОМОЩИ СТЕКА

Примеры задач, решение которых основано на использование стека:

1. Грамматика правильной скобочной последовательности.
2. Вычисления арифметических выражений.
3. Ближайший меньший слева и справа.

ЗАДАЧИ, РЕШАЕМЫЕ ПРИ ПОМОЩИ СТЕКА

Грамматика правильной скобочной последовательности.

Правильная скобочная последовательность – последовательность, состоящая из символов – «скобок», в которой каждой отрывающейся скобке соответствует закрывающаяся скобка такого же типа, что и открывающаяся скобка.
Например, **правильными** будут следующие последовательности:
[([])((([[[]]])))]{()},
()(((())))[[]].

Не будут являться правильными скобочные последовательности:
[[)) (несоответствие типа закрывающихся скобок типу открывающихся),
}{ (закрывающая скобка стоит раньше открывающейся),
[[{{}}] (не каждой открывающейся скобке соответствует закрывающаяся).

ЗАДАЧИ, РЕШАЕМЫЕ ПРИ ПОМОЩИ СТЕКА

При движении слева направо по строке в стек заносятся открывающиеся скобки. При добавлении в стек закрывающейся скобки проверяем наличие в вершине стека открывающейся скобки такого же типа. Если таковая скобка в вершине стека есть, то очередная скобка не добавляется, а имеющаяся в вершине удаляется.

Пример для правильной скобочной последовательности [([])(([]))].

Последовательность	[	(	[	]	)	(	(	[	]	)	)	]
Состояние стека	[	[(	[([	[(	[	[(	[((	[([	[((	[(	[	

В конце работы программы стек оказывается пустым – в этом случае скобочная последовательность правильная.

Задача проверки скобочной последовательности на правильность, имеет линейную сложность O(n).

ЗАДАЧИ, РЕШАЕМЫЕ ПРИ ПОМОЩИ СТЕКА

Вычисления арифметических выражений.

Арифметическое выражение состоит из чисел, знаков арифметических действий и скобок.

Например, арифметическое выражение:

(7+6)/(26-13)

Его можно записать в других формах.

Префиксная форма записи арифметического выражения представляет выражение таким образом, что знаки арифметических операций предшествуют операндам, над которыми эти операции выполняются.

А **постфиксная форма** записи (обратная польская нотация) представляет выражение таким образом, что знаки арифметических операций указываются после операндов.

ЗАДАЧИ, РЕШАЕМЫЕ ПРИ ПОМОЩИ СТЕКА

Арифметическое выражение	(7+6)/(26-13)
Префиксная форма записи	/ + 7 6 – 26 13
Постфиксная форма записи	7 6 + 26 13 - /

В постфиксной записи не нужны скобки и приоритеты операций!
Например, запись «2 3 + 4 *» обозначает (2+3)∗4,
а запись «2 3 4 * +» обозначает 2+(3∗4).

Постфиксная запись является последовательностью действий для стекового калькулятора. Числа добавляются в стек, а результат операций применяется к двум последним числам в стеке, которые из стека удаляются. Затем результат кладется в стек.
По завершении работы алгоритма в стеке оказывается один элемент – значение арифметического выражения.

ЗАДАЧИ, РЕШАЕМЫЕ ПРИ ПОМОЩИ СТЕКА

Пусть есть выражение «2 3 + 4 *» в постфиксной записи может быть вычислено так:

- Число 2 кладется в стек.
- Число 3 кладется в стек.
- Из стека извлекаются числа 3 и 2, к ним применяется операция сложения, результат (число 5) кладется в стек.
- Число 4 кладется в стек.
- Из стека удаляются числа 5 и 4, результат их умножения 20 кладется в стек.
- В итоге в стеке оказывается одно число — 20, которое и есть результат вычисления выражения.

ЗАДАЧИ, РЕШАЕМЫЕ ПРИ ПОМОЩИ СТЕКА

Пусть есть выражение «2 3 4 + *» порядок действий будет другим:

- Числа 2, 3, 4 кладутся в стек
- Из стека удаляются два последних элемента (3 и 4), результат их сложения (7) кладется в стек. В итоге в стеке — два элемента: 2 и 7
- Из стека удаляются элементы 2 и 7, результат их умножения (14) кладется в стек.
- В итоге в стеке остается одно число — 14, которое и есть результат вычисления.

Алгоритм вычисления значения арифметического выражения, записанного в постфиксной форме, имеет линейную сложность – O(n).

ЗАДАЧИ, РЕШАЕМЫЕ ПРИ ПОМОЩИ СТЕКА

Алгоритм вычисления **арифметического выражения**, использующий в неявном виде обратную польскую нотацию.
Заводится два стека.
Один – для чисел, второй для знаков арифметических операций и скобок.
Читается выражение слева направо, число, помещается в первый стек.
Если текущий символ – закрывающаяся скобка, то выполняются вычисления до тех пор, пока не встретится парная ей открывающуюся скобку.
Если текущий символ – знак арифметической операции и на вершине стека операции с таким же или большим приоритетом, то выполняются все необходимые для нее действия.
По завершении алгоритма в стеке операций могут остаться еще не выполненные операции, их надо выполнить, как было описано выше.

ЗАДАЧИ, РЕШАЕМЫЕ ПРИ ПОМОЩИ СТЕКА

Ближайший меньший слева и справа.

Дан массив чисел. Требуется вывести ближайший меньший слева и справа для данного элемента.

Например, для массива

a[9]={6 5 9 8 7 1 2 3 5} и числа 7

ближайшим меньшим слева будет 5 с индексом 2, а ближайший меньший справа будет 1 с индексом 6.

ЗАДАЧИ, РЕШАЕМЫЕ ПРИ ПОМОЩИ СТЕКА

Поиск ближайшего меньшего справа.

Последовательно берутся элементы с конца массива и записываются их индексы в стек.
На первой итерации добавляется в стек
(9) – индекс последнего элемента в массиве.
На второй итерации запоминается
(8) – индекс предпоследнего элемента в массиве.
Обращаются к вершине стека. Пока на вершине стека индексы элементов, которые больше, чем текущий или равны ему, эти элементы пропускаются и удаляются из стека.
Запоминается ответ (вершина стека) и добавляется новое число (индекс просматриваемого элемента) в стек.
Алгоритм продолжается далее, пока не будет получен ответ для последнего числа.

Таким образом, сложность алгоритма линейная – O(n).

ЗАДАЧИ, РЕШАЕМЫЕ ПРИ ПОМОЩИ СТЕКА

1 итерация. Стек	9	Добавляем в стек индекс последнего элемента массива. В ответ для него записываем -1.							
2 итерация. Стек	8		Берем из массива элемент 3 с индексом 8. В вершине стека индекс элемента 5, который больше, чем 3. Поэтому удаляем из стека индекс 9, и записываем в стек индекс 8. В ответ записываем – 1.						
3 итерация. Стек	7			Берем из массива элемент 2 с индексом 7. Просматриваем стек. На вершине стека индекс 8 (элемента 3. 3 больше 2), поэтому удаляем из стека 8, записываем 7. В ответ записываем – 1.					
4 итерация. Стек	6				Для элемента 1 с индексом 6 ответом также является -1				
5 итерация. Стек	6	5				Ответ для текущего элемента 7 с индексом 5 – индекс 6.			
6 итерация. Стек	6	5	4				Ответ для элемента 8 - индекс 5		
7 итерация. Стек	6	5	4	3				Ответ для элемента 9 с индексом 3, индекс 4.	
8 итерация. Стек	6	2							Ответ 6
9 итерация. Стек	6	2	1						2
Исходный массив	6	5	9	8	7	1	2	3	5
ans	2	6	4	5	6	-1	-1	-1	-1

По полученному ответу легко восстановить значения ближайших минимальных элементов ко всем элементам исходного массива. Индексы ближайших минимальных справа к каждому из элементов массива будут храниться в массиве ans[9]. Аналогично решается задача поиска ближайших минимальных слева элементов.

ЗАДАЧИ, РЕШАЕМЫЕ ПРИ ПОМОЩИ ОЧЕРЕДИ

Очереди также не дают доступа к произвольному элементу, но, в отличие от стека, элементы кладутся *(enqueue)* и забираются *(dequeue)* с разных концов. Такой метод называется «первый вошел, первый вышел»

Очереди часто используются в программах для реализации буфера, в который можно положить элемент для последующей обработки, сохраняя порядок поступления. Например, если база данных поддерживает только одно соединение, можно использовать очередь потоков, которые будут ждать своей очереди на доступ к БД.

ЗАДАЧИ, РЕШАЕМЫЕ ПРИ ПОМОЩИ ОЧЕРЕДИ

Постановка задачи.

Дан массив A, состоящий из n элементов. Необходимо найти минимум на всех подотрезках массива фиксированной длины K за O(n).

Например, для массива A[10]={1, 8, 4, 3, 5, 7, 4, 5, 6, 7} при K=3 ответ ans={1, 3, 3, 3, 4, 4, 4, 5}.

В основе решения идея по реализации очереди при помощи двух стеков.

Первый стек head отвечает за уход элементов из очереди, и представляет собой начало очереди.

Второй стек tail отвечает за приход элементов в очередь, и представляет собой хвост очереди.

Например, в очередь постепенно добавляются элементы: 1, 8, 4, 3, а некоторые из добавленных удаляются. Если при попытке извлечь элемент стек head оказался пустым, то переносятся все элементы из tail в head (при этом элементы будут перенесены в обратном порядке, что и нужно для извлечение первого элемента из очереди).

ЗАДАЧИ, РЕШАЕМЫЕ ПРИ ПОМОЩИ ОЧЕРЕДИ

Операции	Состояние стека tail	Состояние стека head
Добавление элемента 1 в очередь	1	Стек пустой
Добавление элемента 8 в очередь	8 (в вершине стека элемент 5) 1	Стек пустой
Добавление элемента 4 в очередь	4 8 1	Стек пустой
Удаление элемента из очереди. Удаляется первый элемент – 1. Так как стек head пустой – переносим все элементы из стека tail в стек head. Затем удаляем 1	Все элементы переносим в стек head	1 – удаляем элемент 1 8 4
Добавляем элемент 3 в очередь	3	8 4
Удаляем элемент из очереди	3	8 – удаляем элемент 8 4
Удаляем элемент из очереди	3	4 – удаляем элемент из очереди
Удаляем элемент из очереди	Элементы переместили во второй стек. Стек пустой	3 – удаляем элемент из очереди

ЗАДАЧИ, РЕШАЕМЫЕ ПРИ ПОМОЩИ ОЧЕРЕДИ

Каждый элемент один раз помещается в стек, один раз перекладывается в другой стек, и один раз удаляется из стека. Операция добавления выполняется за O(1).

Операция удаления в худшем случае выполняется за O(n).

В алгоритме реализуется идея скользящего окна – по массиву движется окно фиксированного K размера, в котором определяется минимум. В первый стек помещаем K элементов, записываем в ответ текущий минимум. Затем в первый стек добавляем (K+1) элемент. Извлекаем первый элемент очереди – для этого перекладываем все элементы из стека tail в стек head с новым минимумом для второго стека. Удаляется элемент из вершины второго стека.

На вершине второго стека будет находиться минимум для второго подотрезка – записывается ответ.

Добавляется новый элемент в очередь, выводится ответ – минимум из двух вершин стека.

Удаляется элемент из очереди. Алгоритм продолжается далее.

ЗАДАЧИ, РЕШАЕМЫЕ ПРИ ПОМОЩИ ОЧЕРЕДИ

Первый стек **tail**	Ответ **ans**	Второй стек **head**
(4,1) Стек заполнили К элементами. (8,1) **Записали** ответ из вершины стека. (1,1)	1	
(3,1) **Добавили** в стек следующий элемент. (4,1) (8,1) (1,1)	1	
Перекладываем элементы во второй стек с новым минимумом для второго стека и удалили элемент с вершины стека	1 3	(1,1) – **удалили** элемент (8,3) – **записали** ответ (4,3) (3,3)
(5,5) **добавили** элемент в очередь	1 3	(8,3) – **удаляем** элемент (4,3) (3,3)
(5,5) **Записали** ответ –минимум из двух вершин стеков	1 3 3	(4,3) (3,3)
(7,5) **Добавили** элемент в очередь (5,5)	1 3 3 3	(4,3) – **удаляем** элемент (3,3)
Продолжаем алгоритм далее.		

ЗАДАЧИ, РЕШАЕМЫЕ ПРИ ПОМОЩИ ДЕК

Дек deque (двусторонняя очередь) является контейнером позволяющим добавлять элементы в конец push_back() и начало очереди push_front() и удалять элементы в начале pop_back() и конце pop_front() очереди. deque поддерживает доступ к произвольному элементу.

ПРИМЕР:

Задача нахождения минимального на фиксированном отрезке при помощи деков.

Рассмотрим ситуацию, когда необходимо найти минимальный на одном подотрезке.

В очереди нужно хранить не все элементы, а только нужные для определения минимума.

В этом случае очередь представляет собой неубывающую последовательность чисел. deque q;// Дек для хранения неубывающей последовательности

В вершине такой очереди хранится минимум последовательности. min = q.front();

ЗАДАЧИ, РЕШАЕМЫЕ ПРИ ПОМОЩИ ДЕК

Добавление элементов в очередь происходит следующим образом: пока в конце очереди элементы, большие или равные данному, то их удаляют, и добавляют новый элемент в конец очереди.
Тем самым не нарушается порядок и не теряется текущий элемент, если он на каком-либо последующем шаге окажется минимумом.
Но при извлечении элемента из головы очереди его там, может и не оказаться.
Это происходит потому, что модифицированная очередь могла выкинуть этот элемент в процессе перестроения. При удалении элемента необходимо значение извлекаемого элемента - если элемент с этим значением находится в голове очереди, то он извлекается, а в противном случае никаких действий не производится Минимум на одном подотрезке описанным алгоритмом определяется за O(1).

ЗАДАЧИ, РЕШАЕМЫЕ ПРИ ПОМОЩИ ДЕК

Для применения этого алгоритма в случае определения минимума на всех подотрезках фиксированной длины заданного массива необходимо, как и в случае очереди на двух стеках делать следующие действия.

Сначала записать указанным способом в дек K элементов,

- записать ответ – текущий минимум на подотрезке,
- добавить новый элемент в дек,
- удалить (если это возможно) первый элемент из дека,
- записать новый ответ из начала дека,
- добавить элемент в конец дека,
- удалить (если это возможно) элемент из начала очереди и так далее.

Сложность работы всего алгоритма – O(n). Реализация задачи для нахождения минимума на фиксированном подотрезке при помощи дека проще, чем при помощи модификации очереди двумя стеками. Но для способа с деком придется хранить весь массив, а в случае с двумя стеками весь массив хранить не нужно – нужно лишь знать значение очередного i элемента.