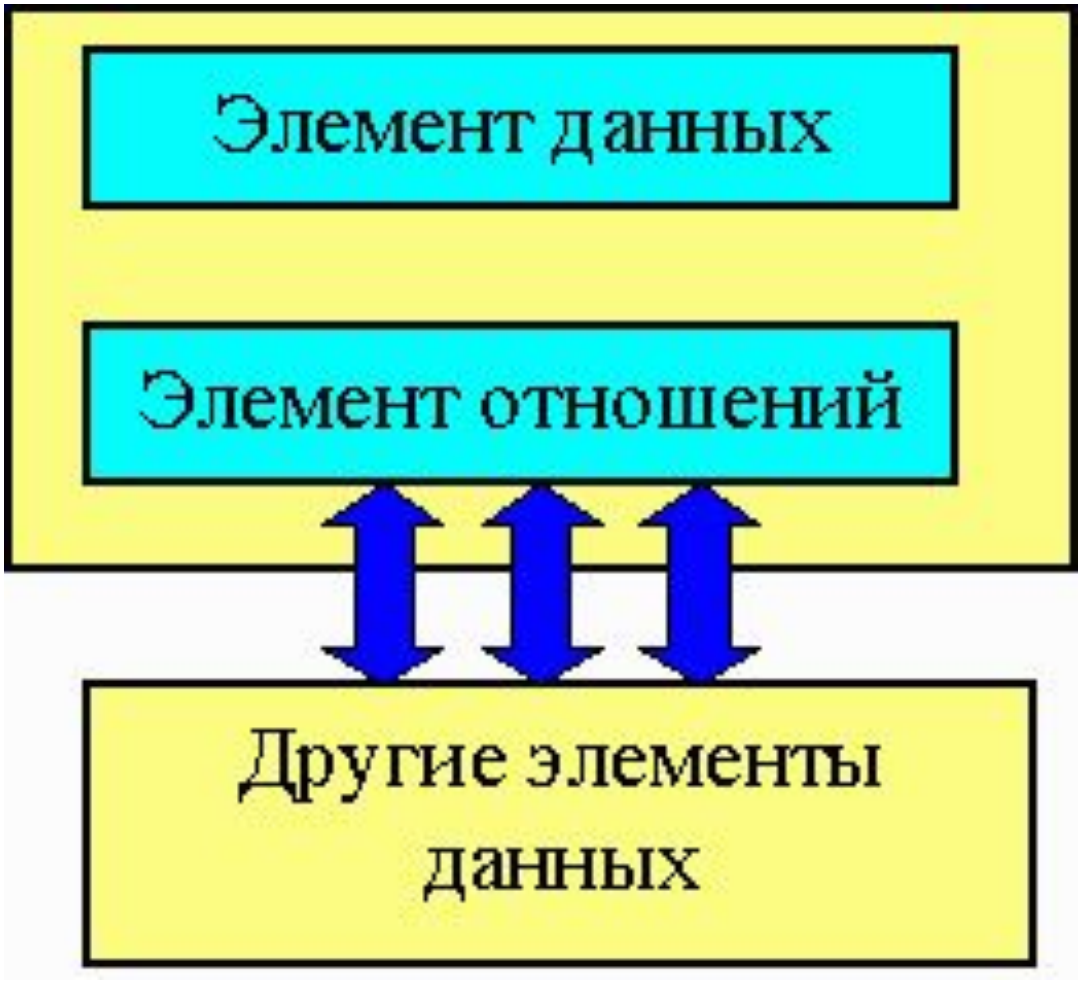
Структуры данных

Структуры данных **-** это совокупность элементов данных и отношений между ними. При этом под элементами данных может подразумеваться как простое данное, так и структура данных. Под отношениями между данными понимают функциональные связи между ними и указатели на то, где находятся эти данные.

Элемент структуры данных



$$S:=(D,R),$$

где S - структура данных, D – данные, R - отношения.


Операции над логическими структурами
Логические структуры данных
Операции над физическими структурами
Физические структуры данных

Структуры данных классифицируются:

1. По связанности данных в структуре

- если данные в структуре связаны очень слабо, то такие структуры называются несвязанными (вектор, массив, строки, стеки)

- если данные в структуре связаны, то такие структуры называются связанными (связанные списки)

2. По изменчивости структуры в процессе выполнения программы

- статические структуры - структуры, неменяющиеся до конца выполнения программы (записи, массивы, строки, векторы)

- полустатические структуры (стеки, деки, очереди)

- динамические структуры - происходит полное изменение при выполнении программы

3. По упорядоченности структуры

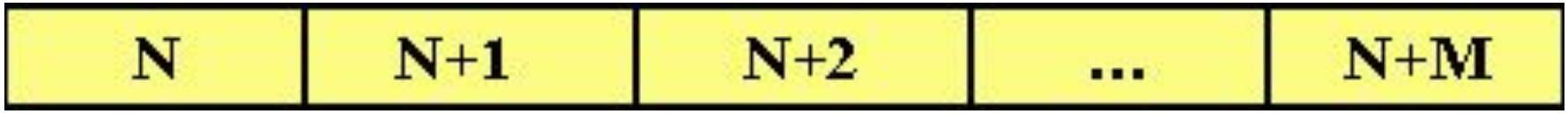
- линейные (векторы, массивы, стеки, деки, записи)

- нелинейные (многосвязные списки, древовидные структуры, графы)

Наиболее важной характеристикой является изменчивость структуры в процессе выполнения программы

СТАТИЧЕСКИЕ СТРУКТУРЫ

Вектор *(одномерный массив)* **- это чисто линейная упорядоченная структура, где отношение между ее элементами есть строго выраженная последовательность элементов структуры**



Массив – структура, в которой элементами являются векторы (элементы которого тоже являются элементами массива)

B_{11}	B_{12}	...	B_{1N}
B_{21}	B_{22}	...	B_{2N}
B_{31}	B_{32}	...	B_{3N}
B_{41}	B_{42}	...	B_{4N}

Запись представляет из себя структуру данных последовательного типа, где элементы структуры расположены один за другим как в логическом, так и в физическом представлении. Запись предполагает множество элементов разного типа.

Запись студентов

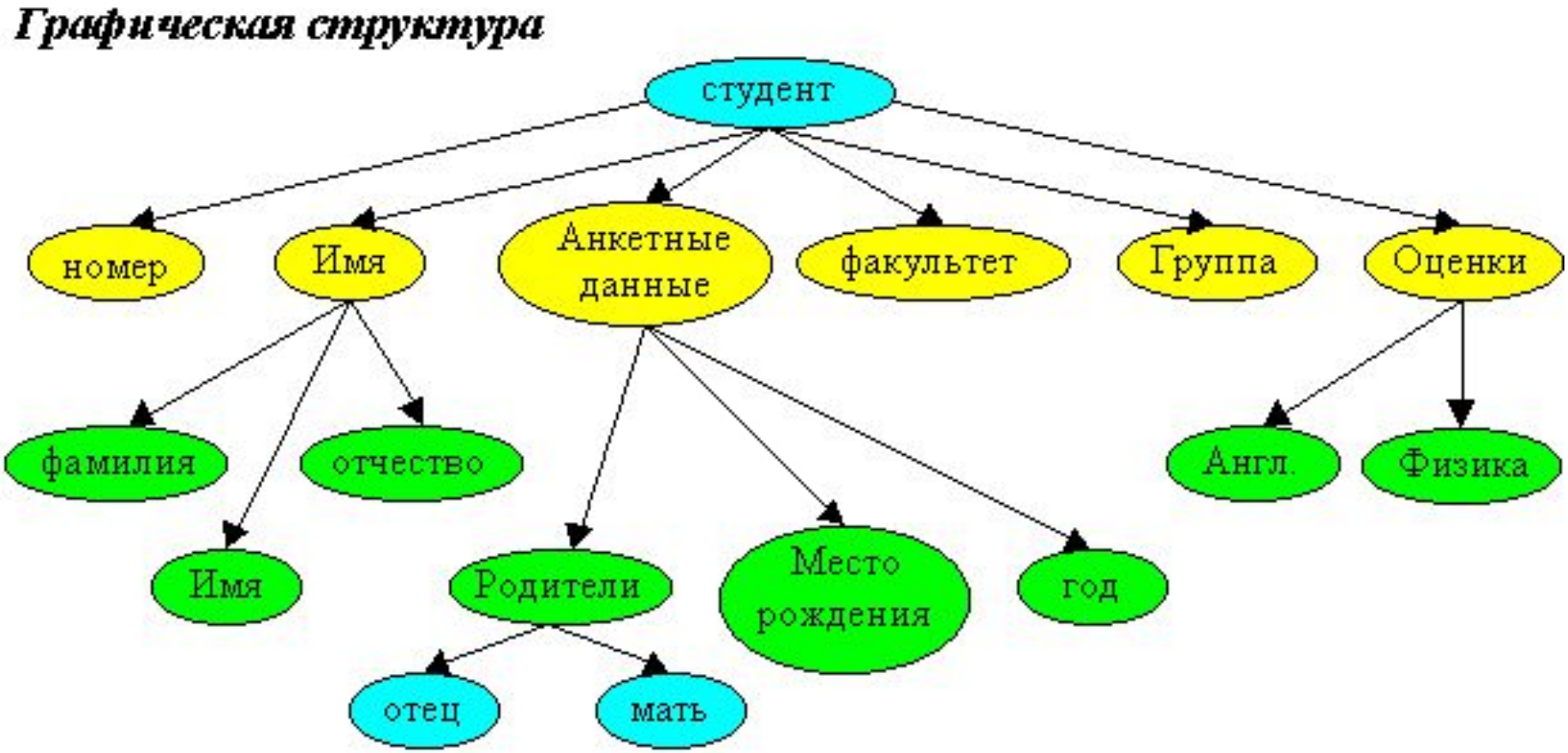
621	Иванов И.И.	ОАПП	95-ОА-21

Логическая структура

Номер	Фамилия	Факультет	Группа

Графическая структура

Элемент записи может включать в себя записи. В этом случае возникает сложная иерархическая структура данных.



Логическая структура иерархической записи

1-ый уровень Студент = запись
2-ой уровень Номер
2-ой уровень Имя = запись
3-ий уровень Фамилия
3-ий уровень Имя
3-ий уровень Отчество
2-ой уровень Анкетные данные = запись
3-ий уровень Место рождения
3-ий уровень Год рождения
3-ий уровень Родители = запись
4-ый уровень Мать
4-ый уровень Отец
2-ой уровень Факультет
2-ой уровень Группа
2-ой уровень Оценки = запись
3-ий уровень Английский
3-ий уровень Физика

Основные операции над записями:

- ✔ Прочтение содержимого поля записи.
- ✔ Занесение информации в поле записи.
- ✔ Все операции, которые разрешаются над полем записи, соответствующего типа.

Таблица - это конечный набор записей

N	ФИО	ГРУППА	ФАКУЛЬТЕТ	АНГЛ.	ФИЗИКА
1					
2					
…					
n					

Основные операции с таблицами:

- Поиск записи по заданному ключу.
- Занесение новой записи в таблицу.

Списки **-** это набор определенным образом связанных элементов данных, которые в общем случае могут быть разного типа:

$E_1, E_2, ..., E_n, ...$ $n > 1$ и не зафиксировано.

Количество элементов списка может меняться в процессе выполнения программы.

Различают 2 вида списков: ***несвязные и связные*** *(несвязанные и связанные)***.**

В ***несвязных*** списках связь между элементами данных выражена неявно (векторы, записи).

В ***связных*** списках в элемент данных заносится ***указатель*** связи с последующим или предыдущим элементом списка.

•ПОЛУСТАТИЧЕСКИЕ СТРУКТУРЫ

К ***полустатическим*** структурам данных относят, в принципе, динамические структуры (такие как **стеки**, **очереди** и **деки**), реализованные на статических векторах (одномерных массивах).

Структура вида LIFO (Last Input First Output - последним пришел, первым ушел), при которой на обработку первым выбирается тот элемент, который поступил в нее последним, называется ***стеком***.

Операции, производимые над стеками:

1. **Занесение элемента в стек.**
2. **Выборка элемента из стека.**
3. **Определение пустоты стека.**
4. **Прочтение элемента без его выборки из стека.**
5. **Определение переполнения стека (для полустатических структур)**

Очередь – это структура вида **FIFO (First In First Out** - первым пришел, первым ушел). Очередь открыта с обеих сторон, но элементы могут вставляться только в конец очереди, а удаляться только из начала очереди, т. е. удаляется первый помещаемый в очередь элемент, после чего ранее второй элемент становится первым. По этой причине очередь часто называют списком, организованным по принципу «первый размещенный первым удаляется» в противоположность принципу стековой организации — «последний размещенный первым удаляется».

Для очереди вводят **два** указателя: один - на начало очереди (***F***), второй- на ее конец (***R***).

Операции, производимые над *очередью*

Операция ***insert (q,x)*** - помещает элемент ***x*** в конец очереди ***q***.

Операция ***remove (q)*** удаляет элемент из начала очереди ***q*** и присваивает его значение внешней переменной.

Операция, ***empty (q)*** - вводится с целью предотвращения возможности выборки из пустой очереди.

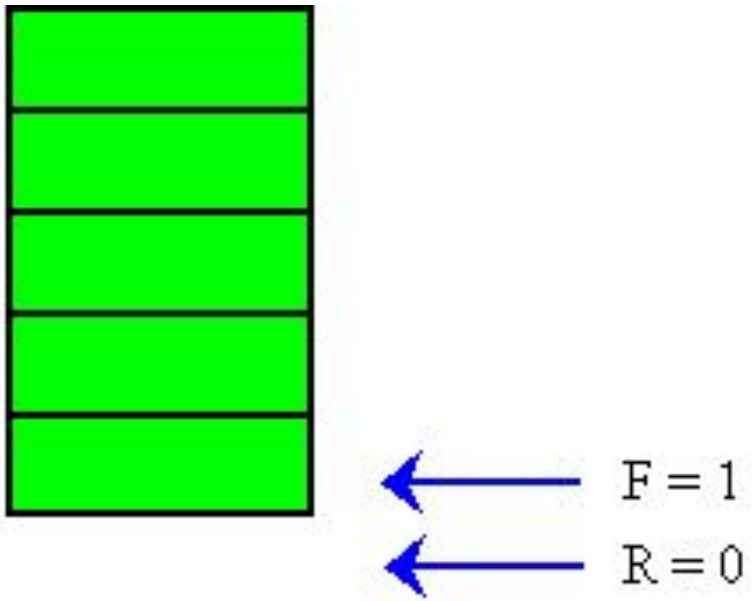
Операция ***full (q)*** - вводится с целью предотвращения возможности переполнения одномерного массива, на котором реализуется полустатическая очередь.

Пример работы с очередью при использовании стандартных процедур

$maxQ = 5$

$\boldsymbol{R = 0,\ F = 1}$

Условие ***пустоты*** очереди $\boldsymbol{R < F}$ $(0 < 1)$

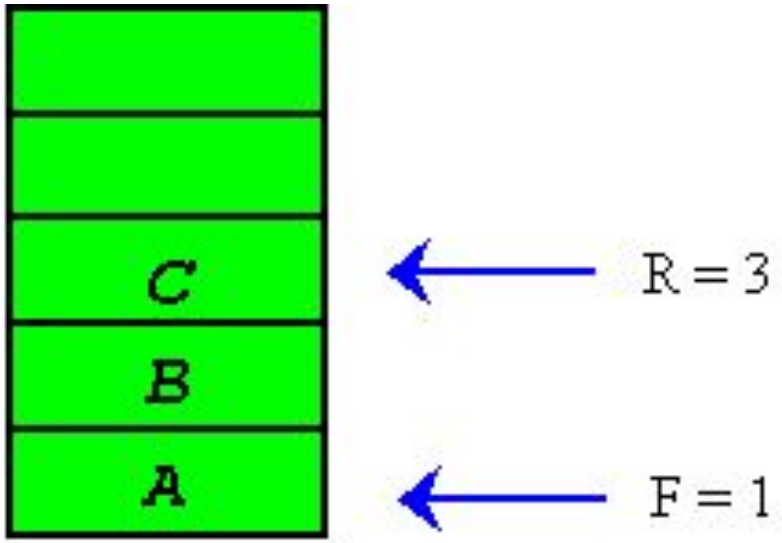


Произведем вставку элементов A, B и C в очередь:

Insert (q, A);

Insert (q, B);

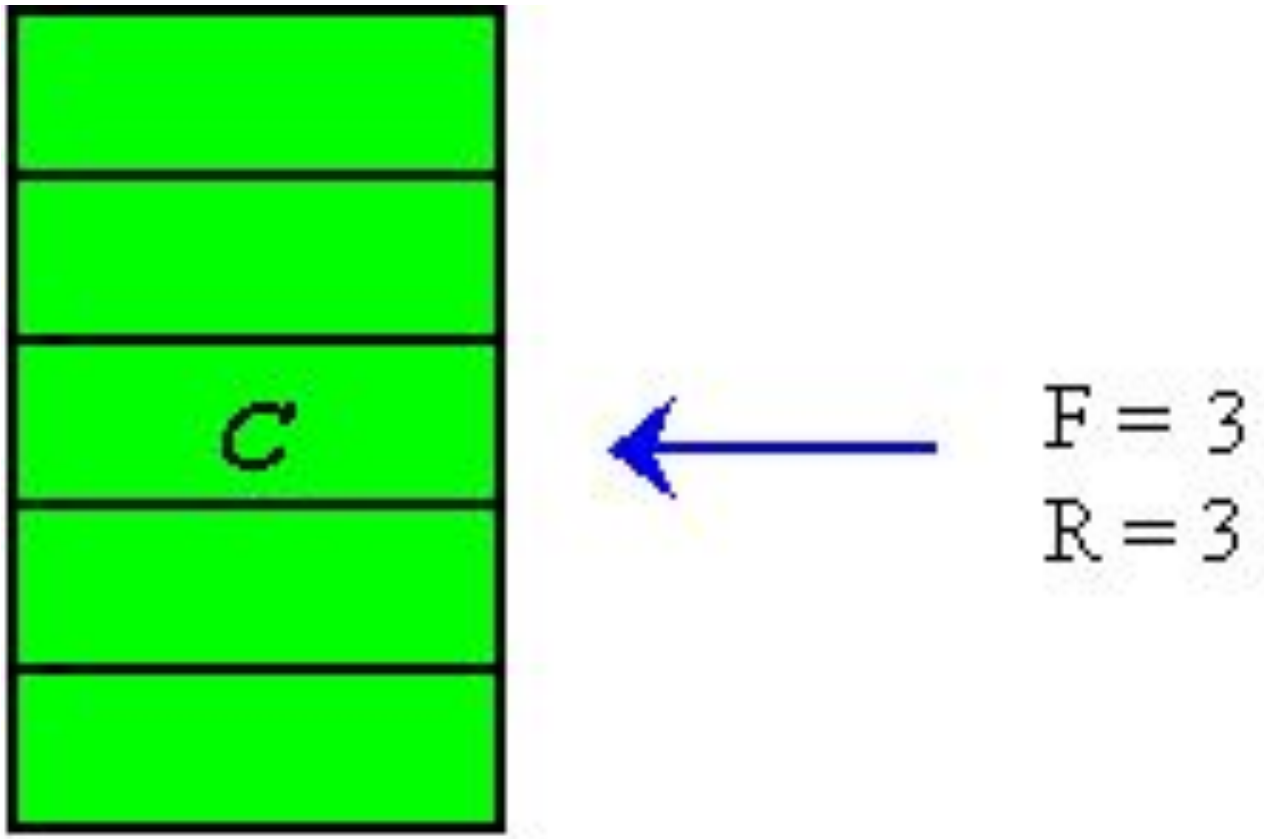
Insert (q, C);



Убираем элементы A и B из очереди:

Remove (q);

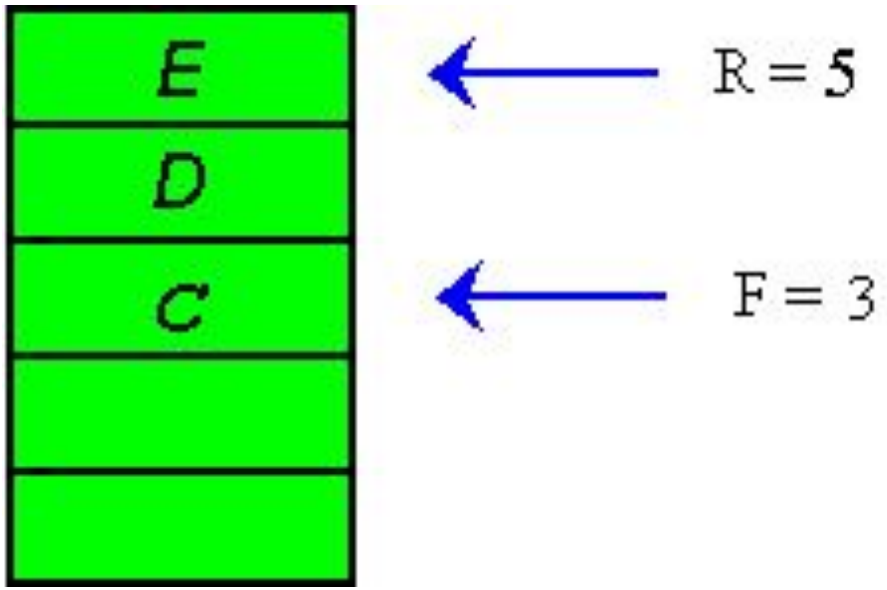
Remove (q);


C
F = 3
R = 3

Добавляем элементы ***D*** и ***E***:

Insert (q, D);

Insert (q, E);



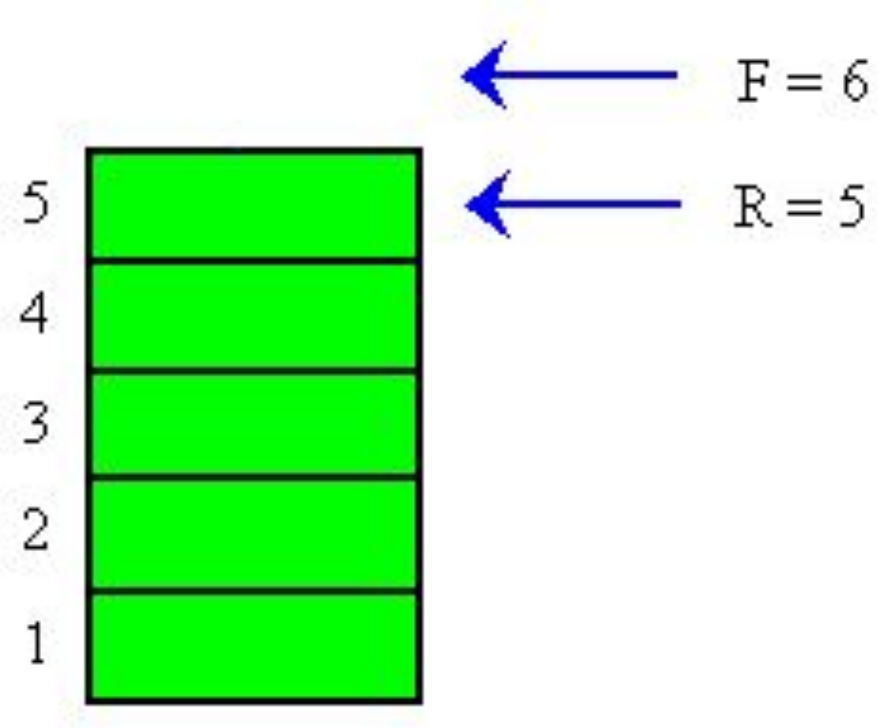
Убираем элементы ***C,D*** и ***E*** из очереди:

Remove (q);

Remove (q);

Remove (q).

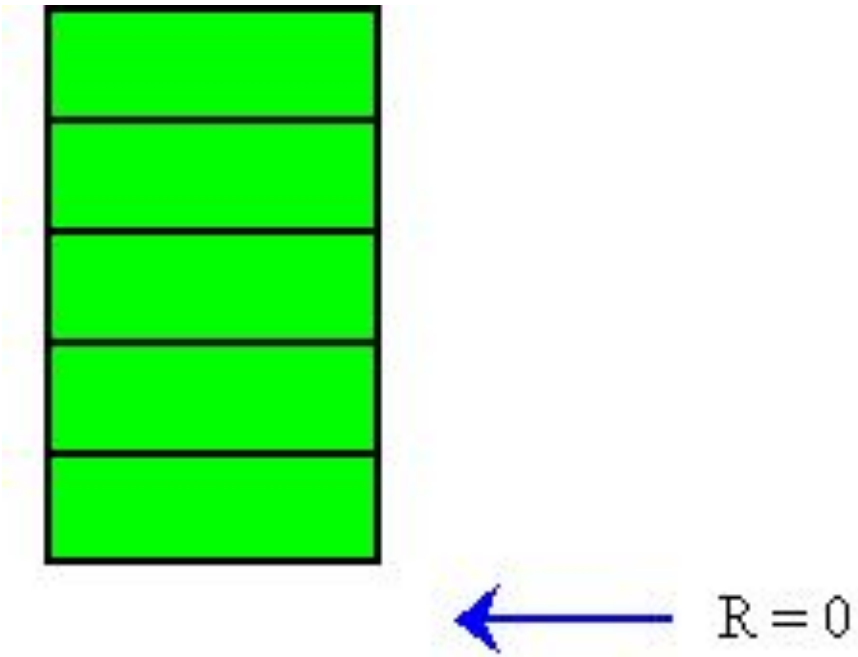
Возникла абсурдная ситуация, при которой очередь является пустой (***R < F***), однако новый элемент разместить в ней нельзя, так как ***R = maxQ***.



Одним из решений возникшей проблемы может быть модификация операции ***Remove (q)*** таким образом, что при удалении очередного элемента вся очередь смещается к началу массива.

Переменная ***F*** больше не требуется, поскольку первый элемент массива всегда является началом очереди.

Пустая очередь представлена очередью, для которой значение ***R*** равно нулю.

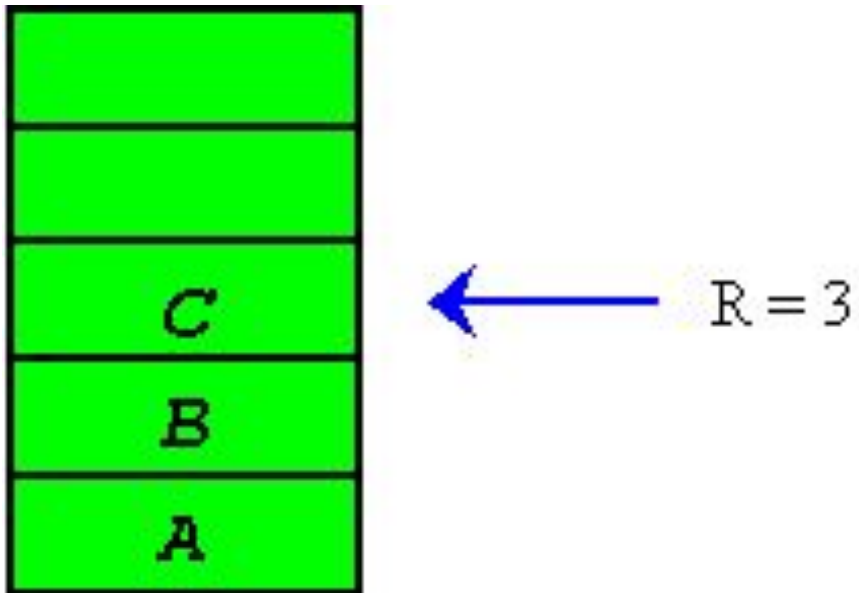


Произведем вставку элементов ***A, B*** и ***C*** в очередь:

Insert (q, A);

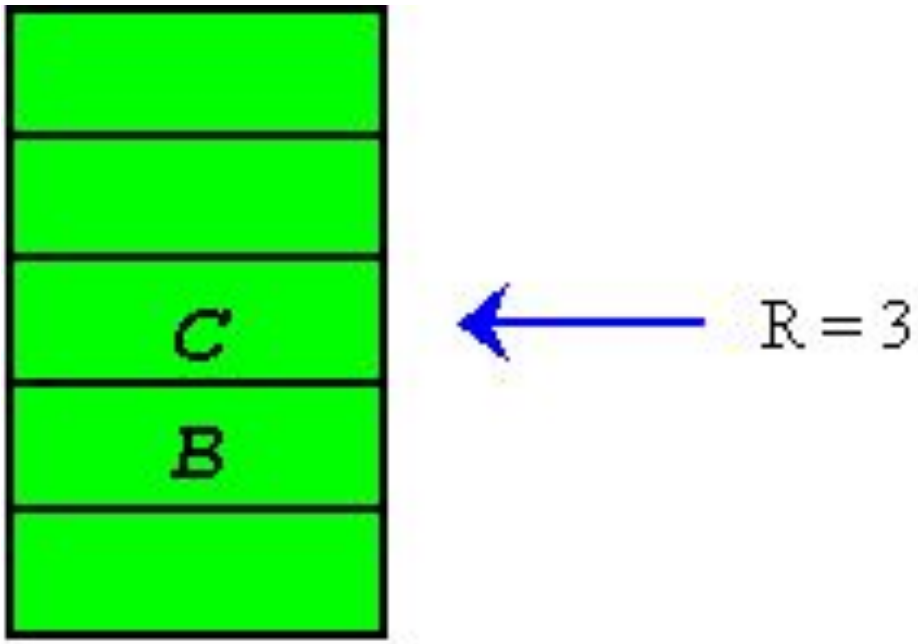
Insert (q, B);

Insert (q, C);



Убираем элемент ***A*** из очереди:

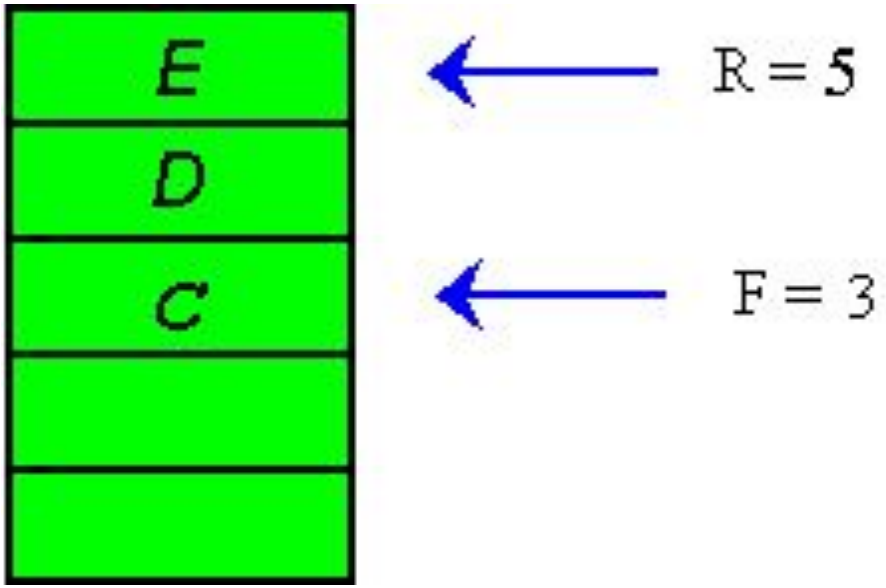
Remove (q)



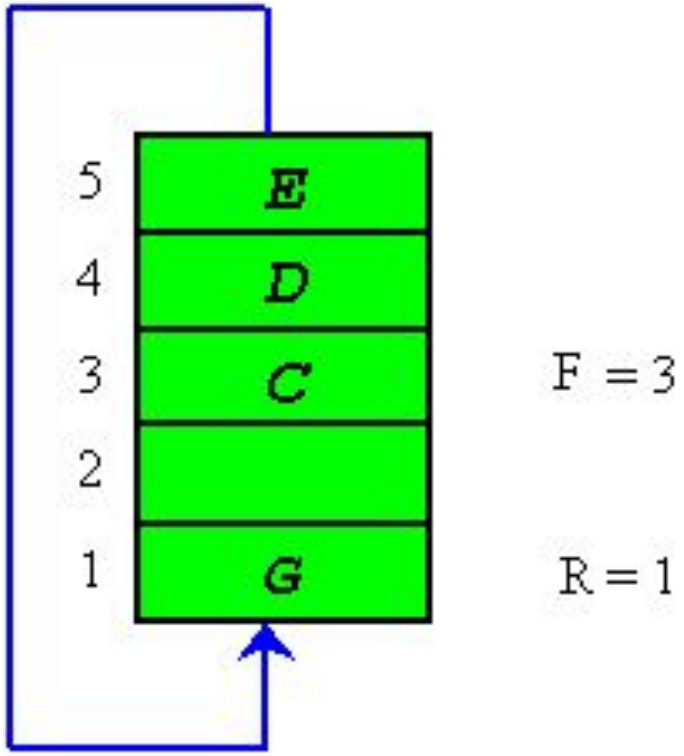
Однако этот метод весьма непроизводителен. Каждое удаление требует перемещения всех оставшихся в очереди элементов. Кроме того, операция удаления элемента из очереди логически предполагает манипулирование только с одним элементом, т. е. с тем, который расположен в начале очереди.

Другой способ предполагает рассматривать массив, который содержит очередь в виде замкнутого кольца. Это означает, что даже в том случае, если последний элемент занят, новое значение может быть размещено сразу же за ним на месте первого элемента, если этот первый элемент пуст.

Предположим, что очередь содержит три элемента - в позициях 3, 4 и 5 пятиэлементного массива. Хотя массив и не заполнен, последний элемент очереди занят.



Если теперь делается попытка поместить в очередь элемент ***G***, то он будет записан в первую позицию массива. Первый элемент очереди есть ***q***(3), за которым следуют элементы ***q*** (4), *q*(5) и ***q***(1).



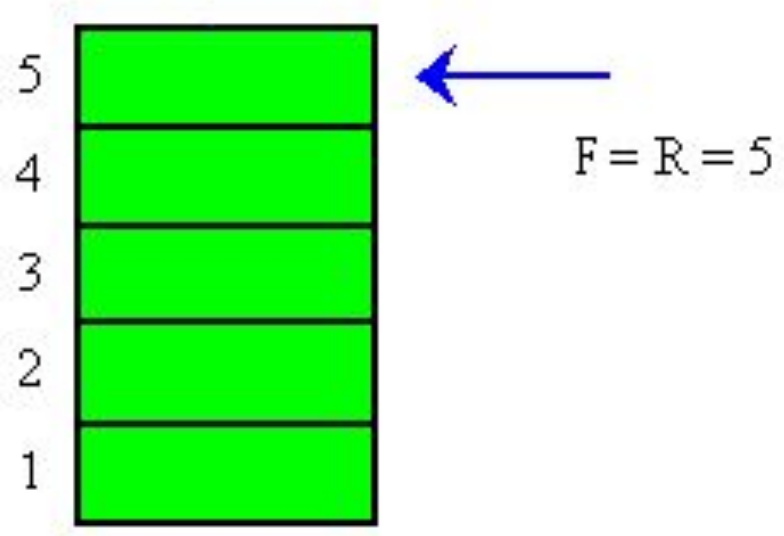
К сожалению, условие ***R<F*** больше не годится для проверки очереди на пустоту.

Одним из способов решения этой проблемы является введение соглашения, при котором значение ***F есть индекс элемента массива, немедленно предшествующего (по кольцу) первому элементу очереди***, а не индекс самого первого элемента.

В этом случае, поскольку ***R*** содержит индекс последнего элемента очереди, ***условие F = R*** подразумевает, что очередь ***пуста***.

Перед началом работы с кольцевой очередью в ***F*** и ***R*** устанавливается значение последнего индекса массива ***maxQ***, а не 1 и 0.

Поскольку ***R*** = ***F***, то очередь изначально пуста.

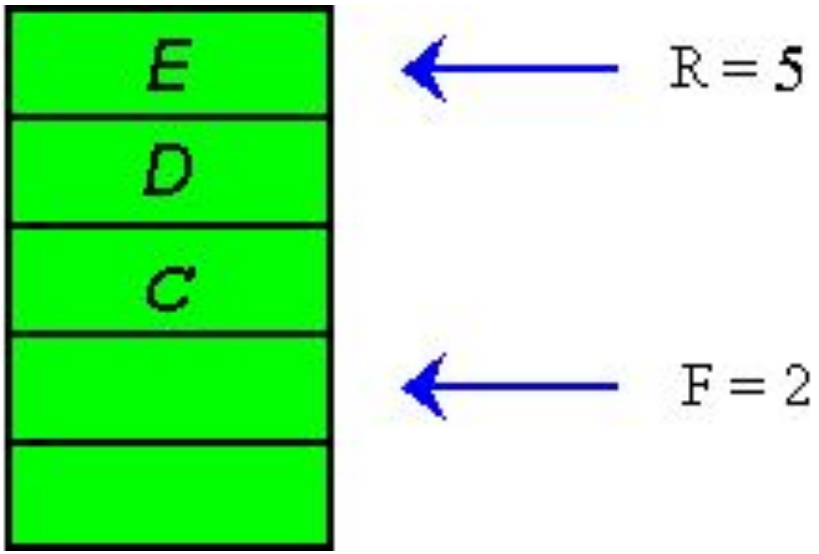


Основные операции с кольцевой очередью

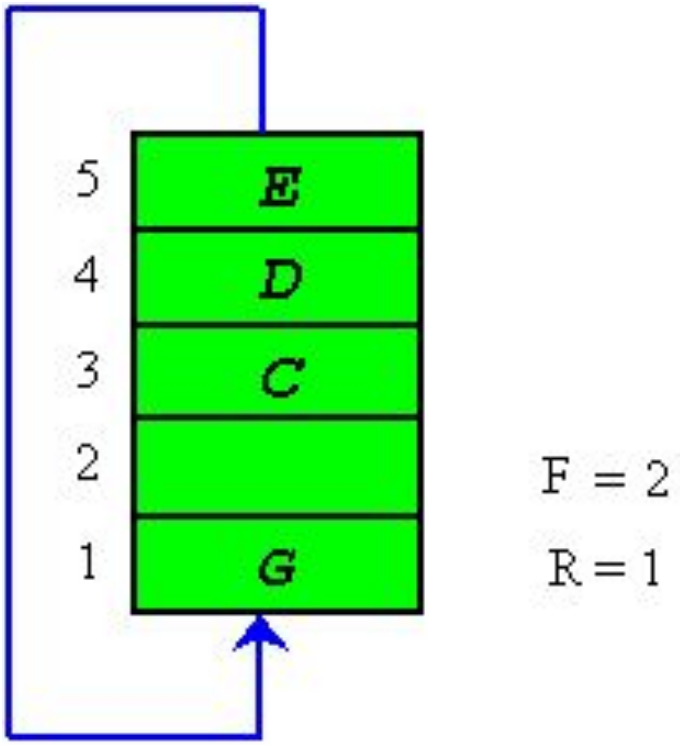
1. Вставка элемента q в очередь x
   Insert(q,x);
2. Извлечение элемента из очереди x
   Remove(q);
3. Проверка очереди на пустоту
   Empty(q);
4. Проверка очереди на переполнение
   Full(q).

Переполнение очереди происходит в том случае, если весь массив уже занят элементами очереди, и при этом делается попытка разместить в ней еще один элемент.

Исходное состояние очереди



Поместим в очередь элемент ***G***.



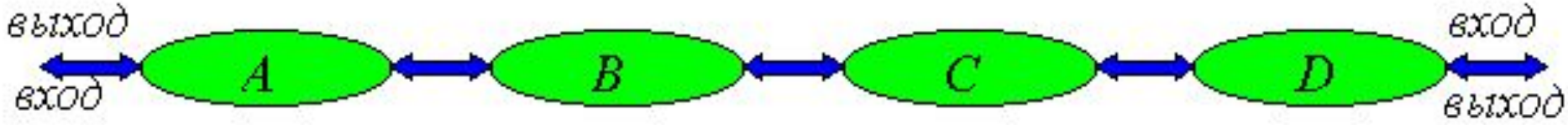
Если произвести следующую вставку, то массив становится целиком заполненным, и попытка произвести еще одну вставку приводит к переполнению. Это регистрируется тем фактом, что ***F = R***, то есть это соотношение как раз и указывает на переполнение. Очевидно, что при такой реализации нет возможности сделать различие между **пустой** и **заполненной** очередью.

Одно из решений состоит в том, чтобы пожертвовать одним элементом массива и позволить очереди расти ***до объема, на единицу меньшего максимального.*** Предыдущий рисунок иллюстрирует именно это соглашение. Попытка разместить в очереди еще один элемент приведет к переполнению.

Проверка на переполнение в подпрограмме *insert* производится *после* установления нового значения для *R*, в то время как проверка на «пустоту» в подпрограмме *remove* производится сразу же после входа в подпрограмму, *до* момента обновления значения *F*.

Дек

Происходит от английского DEQ - Double Ended Queue (очередь с двумя концами).



Дек можно рассматривать как два стека, соединенных нижними границами.

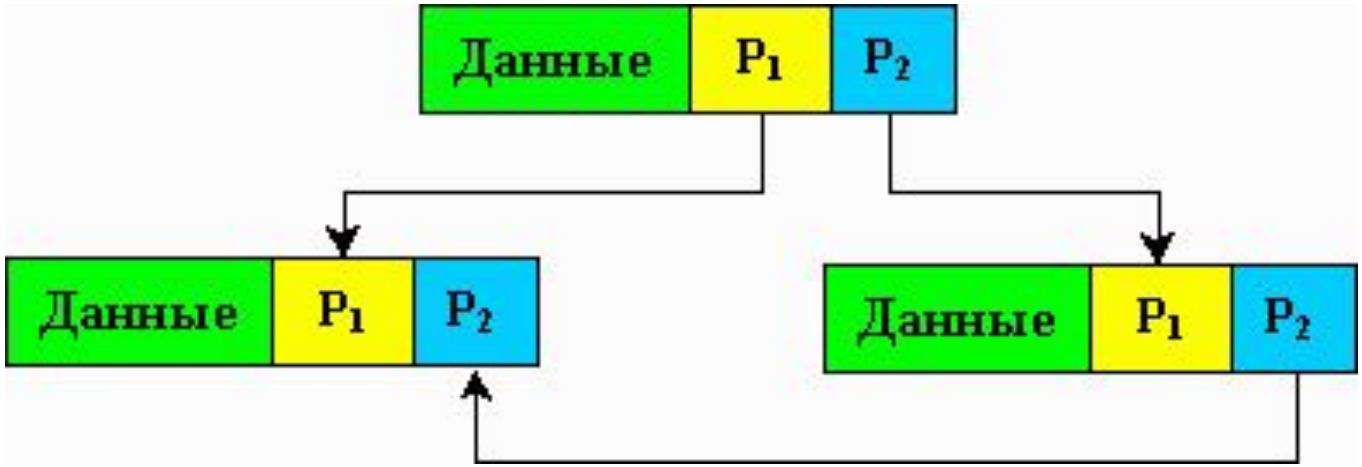
Операции над деками:

- ***Insert(d, x)*** - вставка элемента.
- ***Remove(d)*** - извлечение элемента из дека.
- ***Empty(d)*** - проверка на пустоту.
- ***Full(d)*** - проверка на переполнение.

Динамические структуры данных

Динамические структуры данных имеют две особенности:

1. Заранее не определено количество элементов в структуре.
2. Элементы динамических структур физически не имеют жесткой линейной упорядоченности. Они могут быть разбросаны по памяти.



P1 и P2 это указатели, содержащие адреса элементов, с которыми связаны соответствующие элементы структуры.

Связные списки

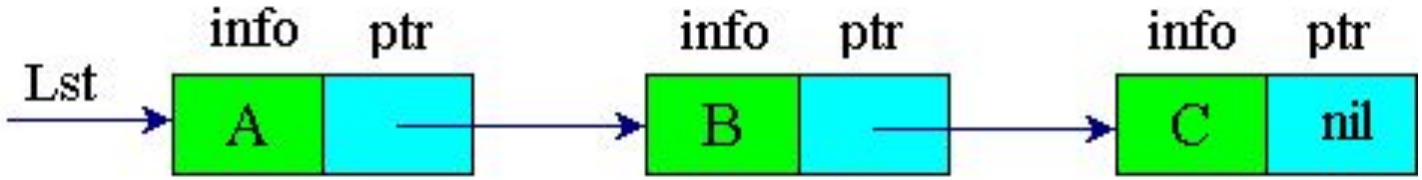
С точки зрения логического представления различают **линейные** и **нелинейные** списки.

К линейным спискам относятся **односвязные** и **двусвязные** списки. К нелинейным - многосвязные.

Элемент списка в общем случае представляет собой **информационное поле и одно или несколько полей указателей.**

Односвязные списки

Элемент односвязного списка содержит, как минимум, два поля: информационное поле (***info***) и поле указателя (***ptr***).



Особенностью **указателя** является то, что он дает только адрес **последующего** элемента списка. Поле указателя последнего элемента в списке является пустым (**NIL**). **LST** - указатель на начало списка. Список может быть пустым, тогда LST будет равен NIL.

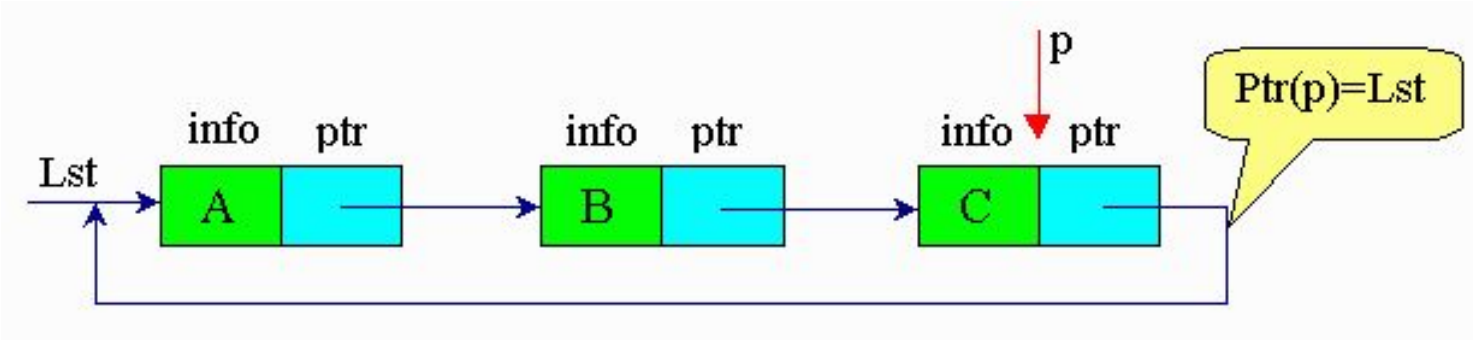
Доступ к элементу списка осуществляется только от его начала, то есть обратной связи в этом списке нет.

ТЕРМИНОЛОГИЯ

- ***p*** - указатель
- ***node(p)*** – узел, на который ссылается указатель ***p*** [при этом неважно в какое место изображения элемента (узла) списка он направлен на рисунке]
- ***ptr(p)*** – ссылка на последующий элемент узла *node(p)*
- ***ptr(ptr(p))*** – ссылка последующего для *node(p)* узла на последующий для него элемент

Кольцевой односвязный список

Кольцевой односвязный список получается из обычного односвязного списка путем присваивания указателю **последнего** элемента списка значения указателя **начала** списка.



Двусвязный список

- **Двусвязный список характеризуется тем, что у любого элемента есть два указателя.**
- **Один указывает на предыдущий (левый) элемент (L), другой указывает на последующий (правый) элемент (R).**

Фактически двусвязный список это два односвязных списка с одинаковыми элементами, записанные в противоположной последовательности.

Кольцевой двусвязный список

- Двусвязные списки получают следующим образом: в качестве значения поля **R** последнего элемента принимают ссылку на первый элемент, а в качестве значения поля **L** первого элемента - ссылку на последний элемент.

Простейшие операции над односвязными списками

- **Вставка элемента в начало односвязного списка**
- **Удаление элемента из начала односвязного списка**

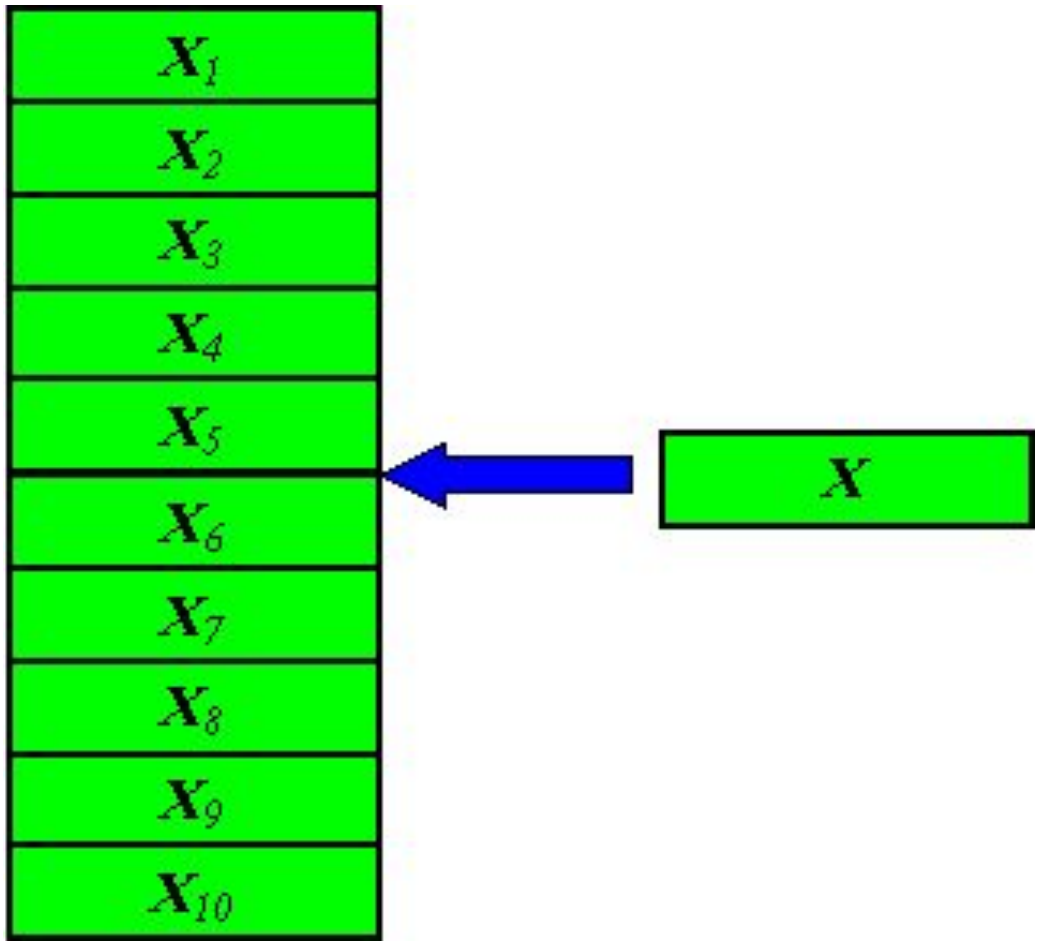
Односвязный список, как самостоятельная структура данных

Просмотр односвязного списка может производиться только последовательно, начиная с головы (с начала) списка. Если необходимо просмотреть предыдущий элемент, то надо снова возвращаться к началу списка. Это – недостаток по сравнению с массивами.

Списковая структура проявляет свои достоинства по сравнению с массивами тогда, когда число элементов списка велико, а вставку или удаление необходимо произвести внутри списка.

Пример

Необходимо вставить элемент *X* в существующий массив между 5-м и 6-м элементами.



Для проведения данной операции в массиве нужно сместить “вниз” все элементы, начиная с X6 - увеличить их индексы на единицу. В результате вставки получаем следующий массив:

Данная процедура в больших массивах может занимать значительное время.

В противоположность этому, в связанном списке операция вставки состоит в изменении значения 2-х указателей и генерации свободного элемента. Причём время, затраченное на выполнение этой операции, является постоянным и не зависит от количества элементов в списке.

Вставка и извлечение элементов из списка

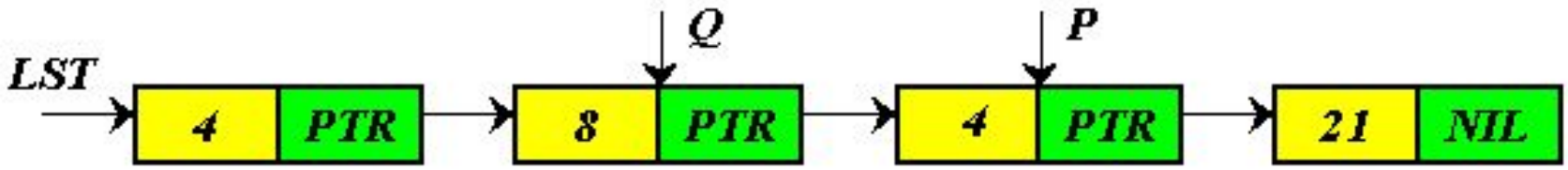
- Сначала определяем элемент, *после* которого необходимо провести операцию вставки или удаления.
- Вставка производится с помощью процедуры *InsAfter(P, x)*, а удаление - *DelAfter(P)*.
- При этом рабочий указатель *P* должен указывать на элемент, *после* которого необходимо произвести вставку или удаление.

Примеры типичных операций над списками

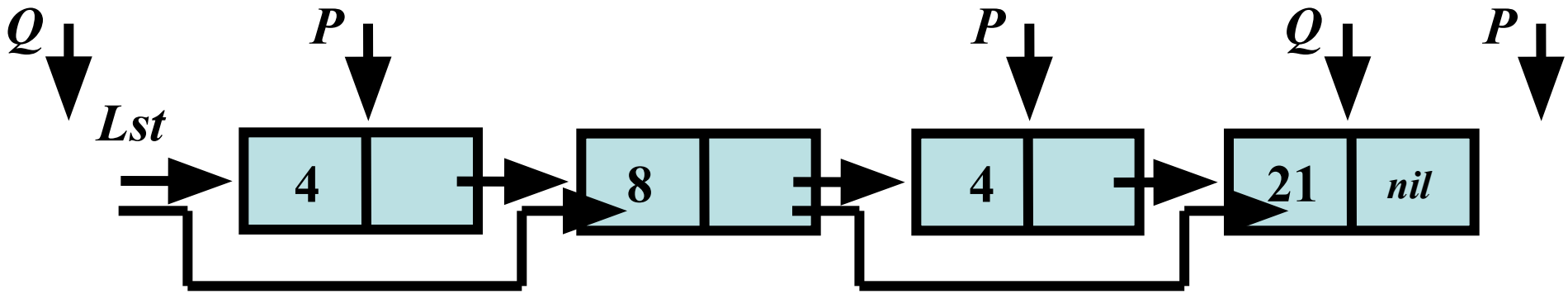
Задача 1

- Требуется просмотреть список и удалить элементы, у которых информационные поля равны **4**.

Обозначим ***P*** - рабочий указатель; в начале процедуры ***P = Lst***.
Введем также указатель ***Q***, который отстает на один элемент от ***P***.
Когда указатель ***P*** найдет заданный элемент, последний будет находиться относительно элемента с указателем ***Q*** как последующий элемент.



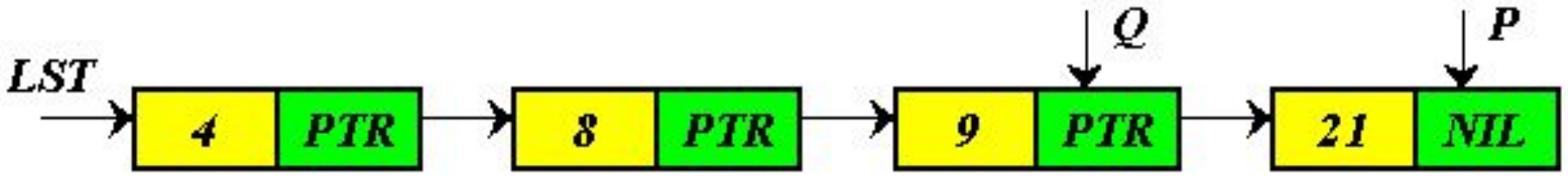
Анимация решения задачи 1



Задача 2

- Дан упорядоченный по возрастанию ***info*** - полей список. Необходимо вставить в этот список элемент со значением ***X***, не нарушив упорядоченности списка.
- Пусть ***X*** = **16**
- Начальные условия:

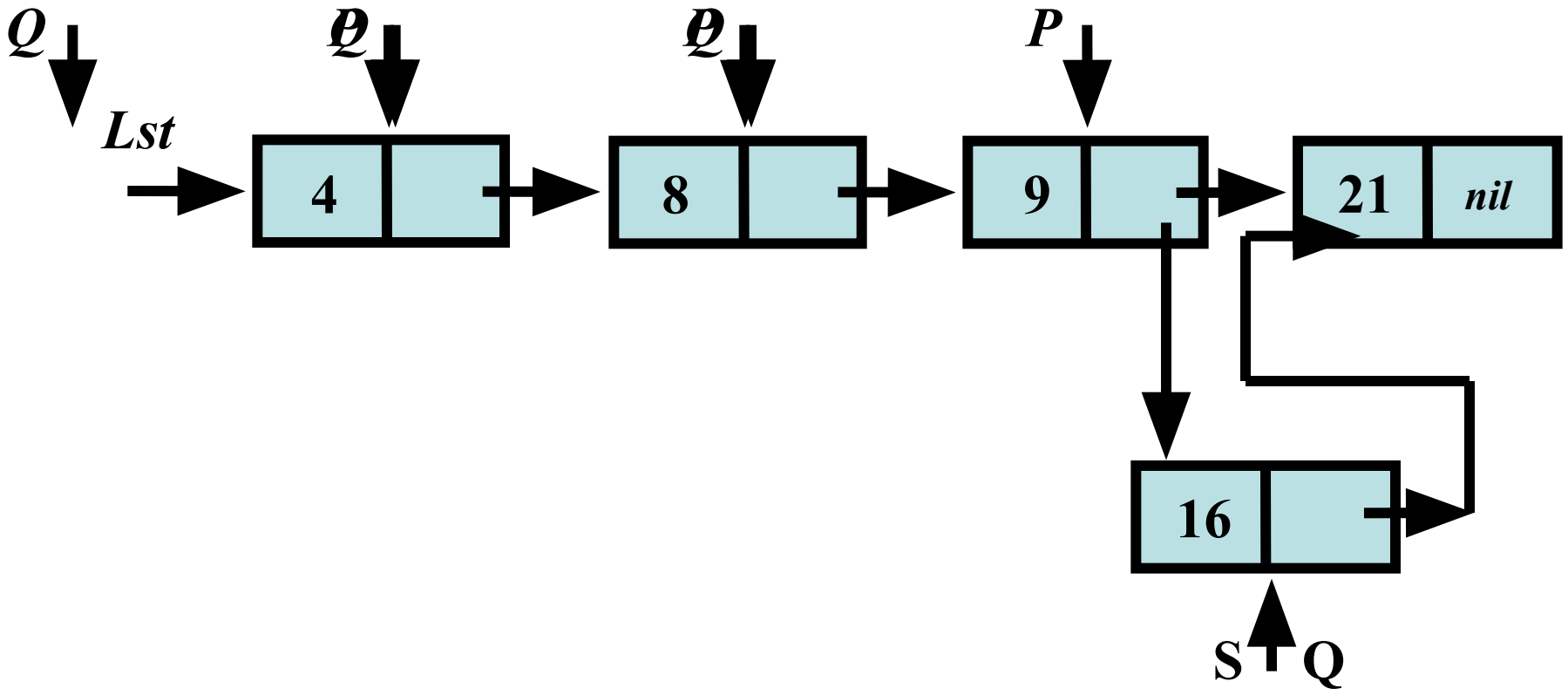
  Q = Nil, P = Lst



Анимация решения задачи 2

$X > info(P) = ?$

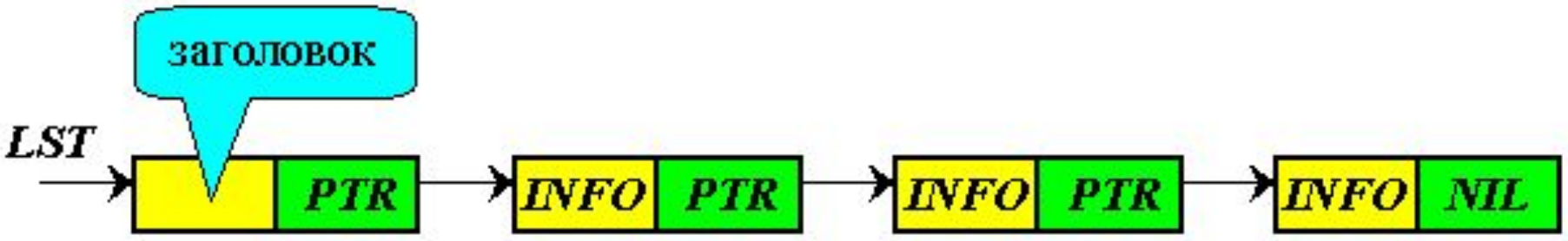
Да ! *Нет !*



Элементы заголовков в списках

- Для создания списка с заголовком в начало списка вводится дополнительный элемент, который может содержать информацию о списке.

- В заголовок списка часто помещают динамическую переменную, содержащую количество элементов в списке (не считая самого заголовка).



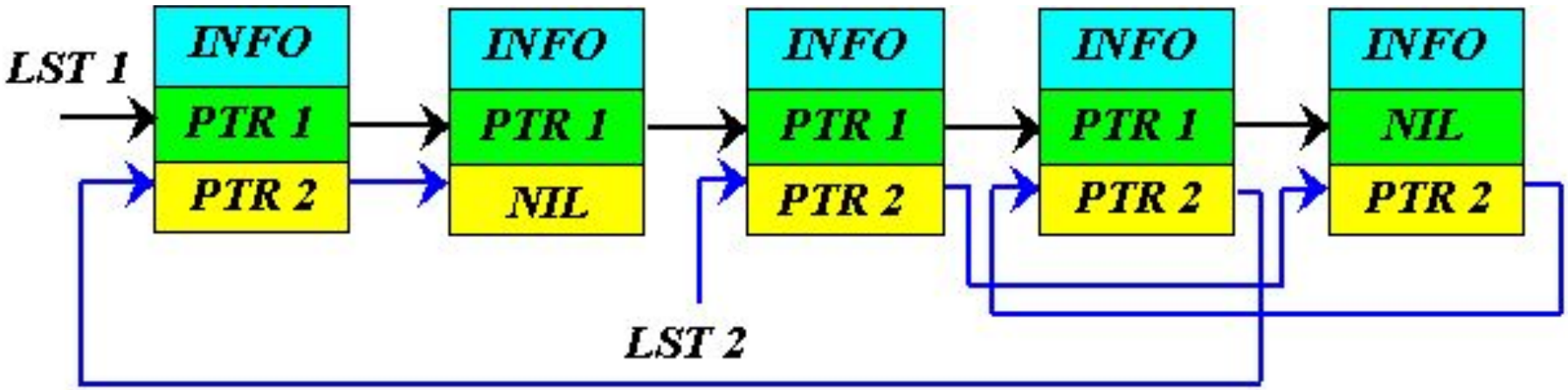
Если список пуст, то остается только заголовок списка.

Удобно занести в информационное поле заголовка значение указателя конца списка. Тогда, если список используется как очередь, то ***F = Lst***, а ***R = Info(Lst)***.

Информационное поле заголовка можно использовать для хранения рабочего указателя при просмотре списка ***P = Info(Lst).*** Другими словами, заголовок - это дескриптор (описатель) структуры данных.

Нелинейные связанные структуры

Двусвязный список может быть нелинейной структурой данных, если вторые указатели задают произвольный порядок следования элементов.

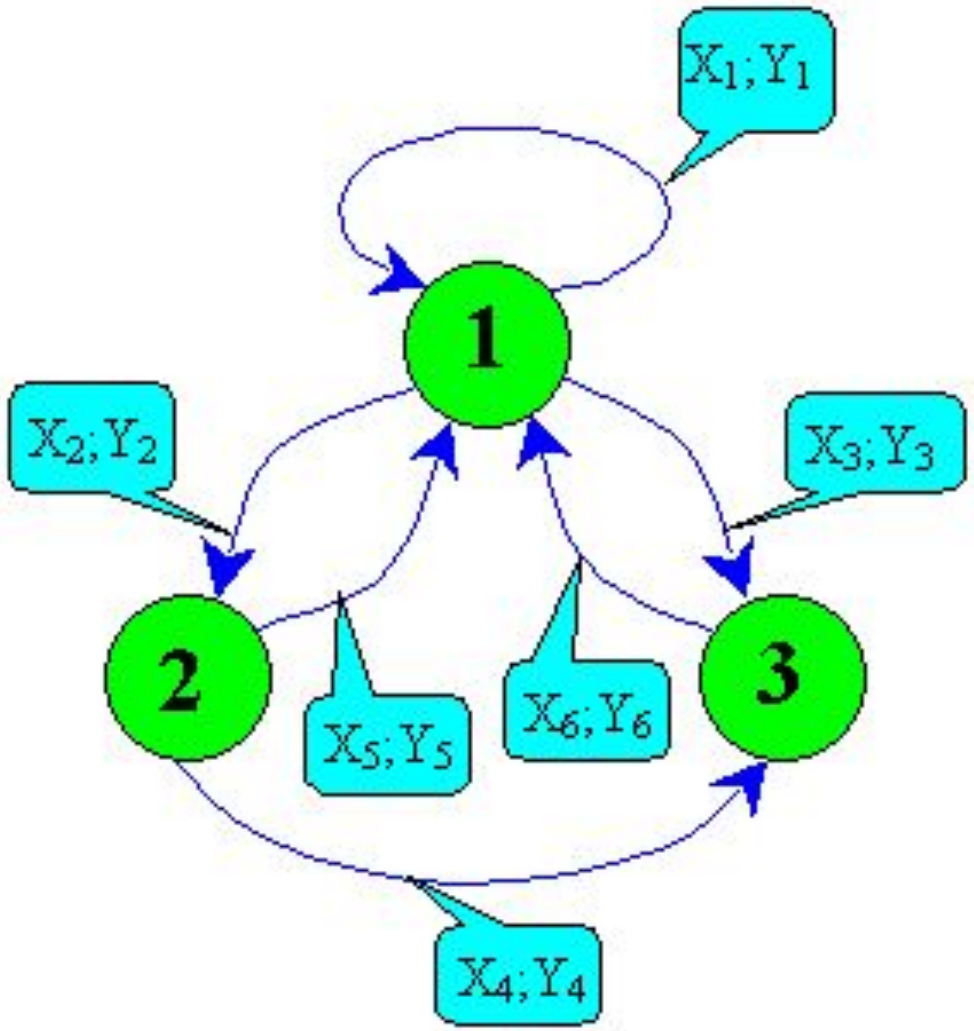


Можно выделить **три** отличительных признака **нелинейной** структуры:

- 1) Любой элемент структуры может ссылаться на любое число других элементов структуры, то есть может иметь любое число полей -указателей.
- 2) На данный элемент структуры может ссылаться любое число других элементов этой структуры.
- 3) Ссылки могут иметь вес, то есть подразумевается иерархия ссылок.

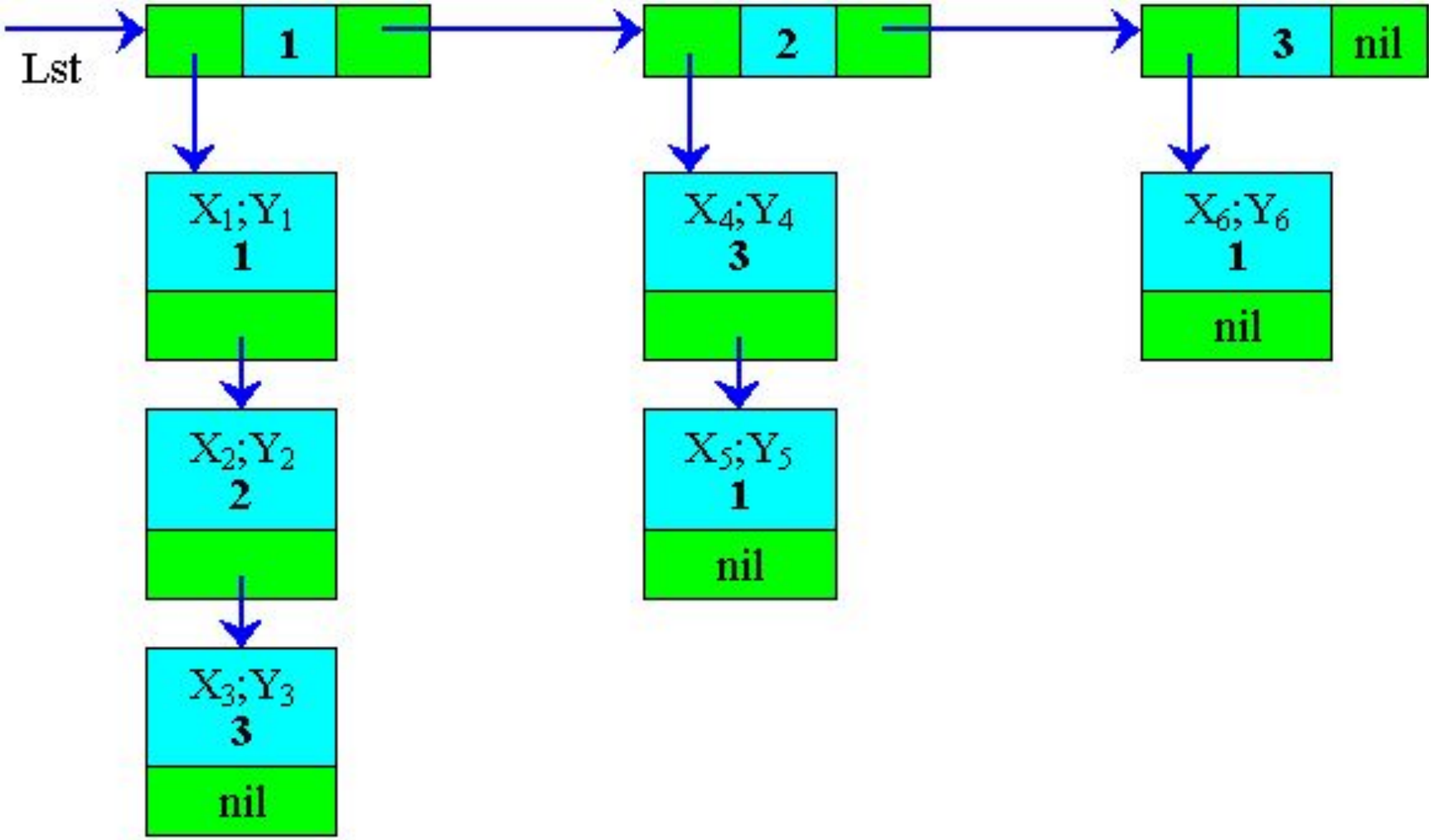
Пример моделирования с помощью нелинейного списка

- Пусть имеется дискретная система, в графе состояния которой узлы - это номера состояний, а ребра – направления переходов из состояния в состояние



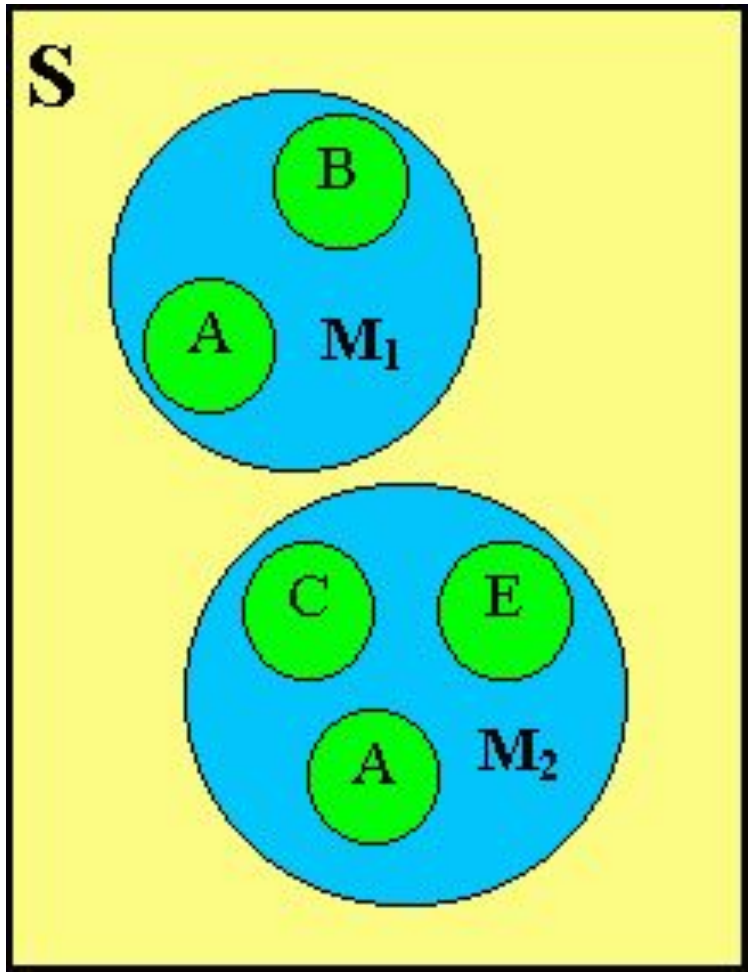
- Входной сигнал в систему это X.
- Реакцией на входной сигнал является выработка выходного сигнала Y и переход в соответствующее состояние.
- Граф состояния дискретной системы можно представит в виде комбинации одного двусвязного и трех односвязных списков, которые вместе составляют нелинейный двусвязный список. При этом в информационных полях должна записываться информация о состояниях системы и ребрах. Указатели элементов должны формировать логические ребра системы.

Реализация графа в виде нелинейного списка



Рекурсивные структуры данных

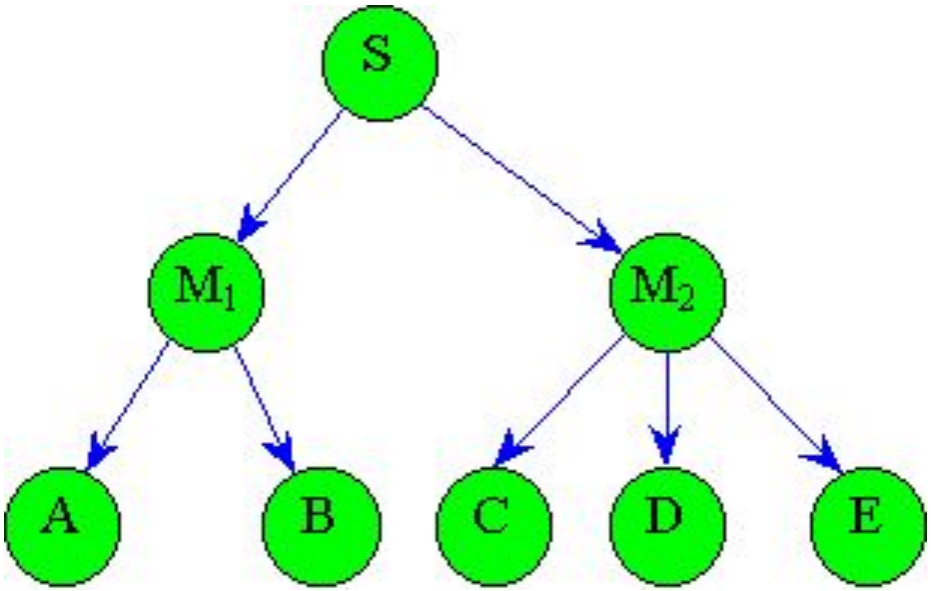
- Рекурсивная структуры данных - структура данных, элементы которой являются такими же структурами данных



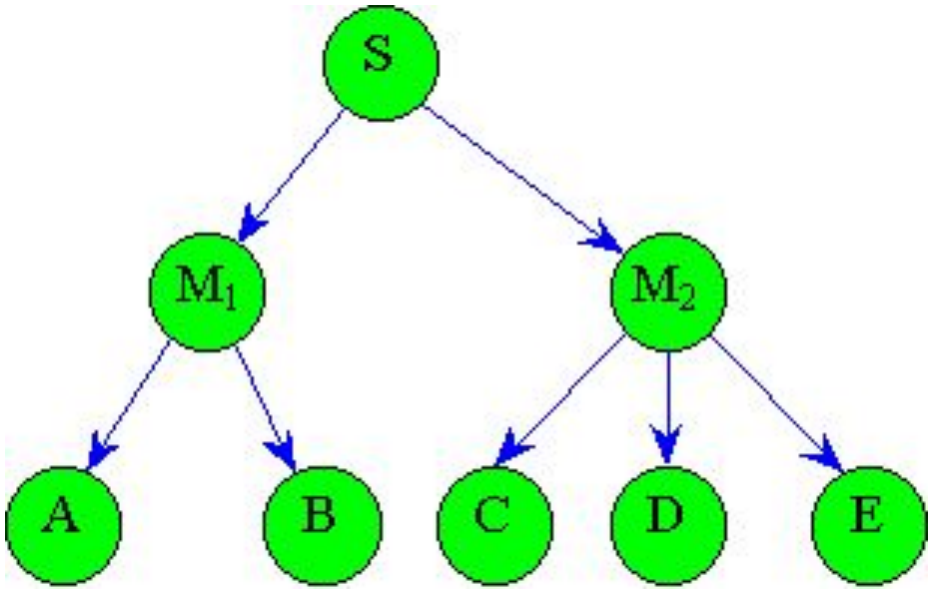
Деревья

- Дерево - нелинейная связанная структура данных.
- Дерево характеризуется следующими признаками:

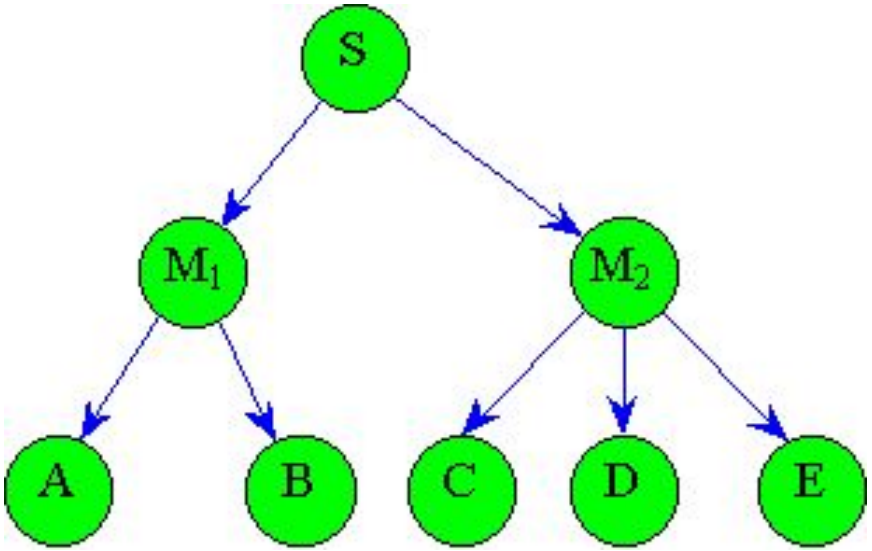
- дерево имеет один элемент, на который нет ссылок от других элементов. Этот элемент называется корнем дерева;
- в дереве можно обратиться к любому элементу путем прохождения конечного числа ссылок (указателей);
- каждый элемент дерева связан только с одним предыдущим элементом.



- Любой узел дерева может быть промежуточным либо терминальным (листом). На рисунке промежуточными являются элементы M1, M2; листьями - A, B, C, D, E. Характерной особенностью терминального узла является отсутствие ветвей.
- Высота дерева - это количество уровней дерева. У дерева на рисунке высота равна двум.



- Количество ветвей, растущих из узла дерева, называется степенью исхода узла (на рисунке для M1 степень исхода 2, для M2 - 3).
- Для описание связей между узлами дерева применяют также следующую терминологию: M1 - “отец” для элементов A и B. A и B - “сыновья” узла M1.



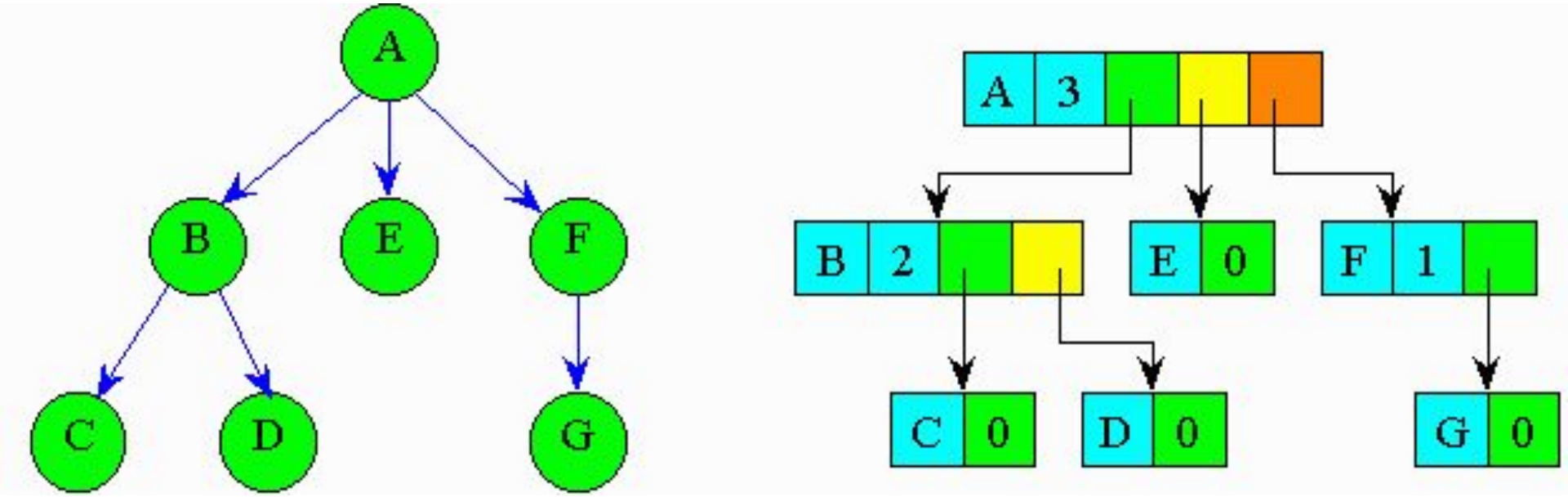
- Деревья могут классифицироваться по **степени исхода** :

1) если максимальная степень исхода равна m, то это - m-арное дерево;
2) если степень исхода равна либо 0, либо m, то это - полное m-арное дерево;
3) если максимальная степень исхода равна 2, то это - бинарное дерево;
4) если степень исхода равна либо 0, либо 2, то это - полное бинарное дерево.

Представление деревьев

- Наиболее удобно деревья представлять в памяти ЭВМ в виде связанных списков. Элемент списка должен содержать информационные поля, в которых могут содержаться значение ключа узла и другая хранимая информация, а также поля-указатели, число которых равно степени исхода.

Представление дерева в виде нелинейного списка



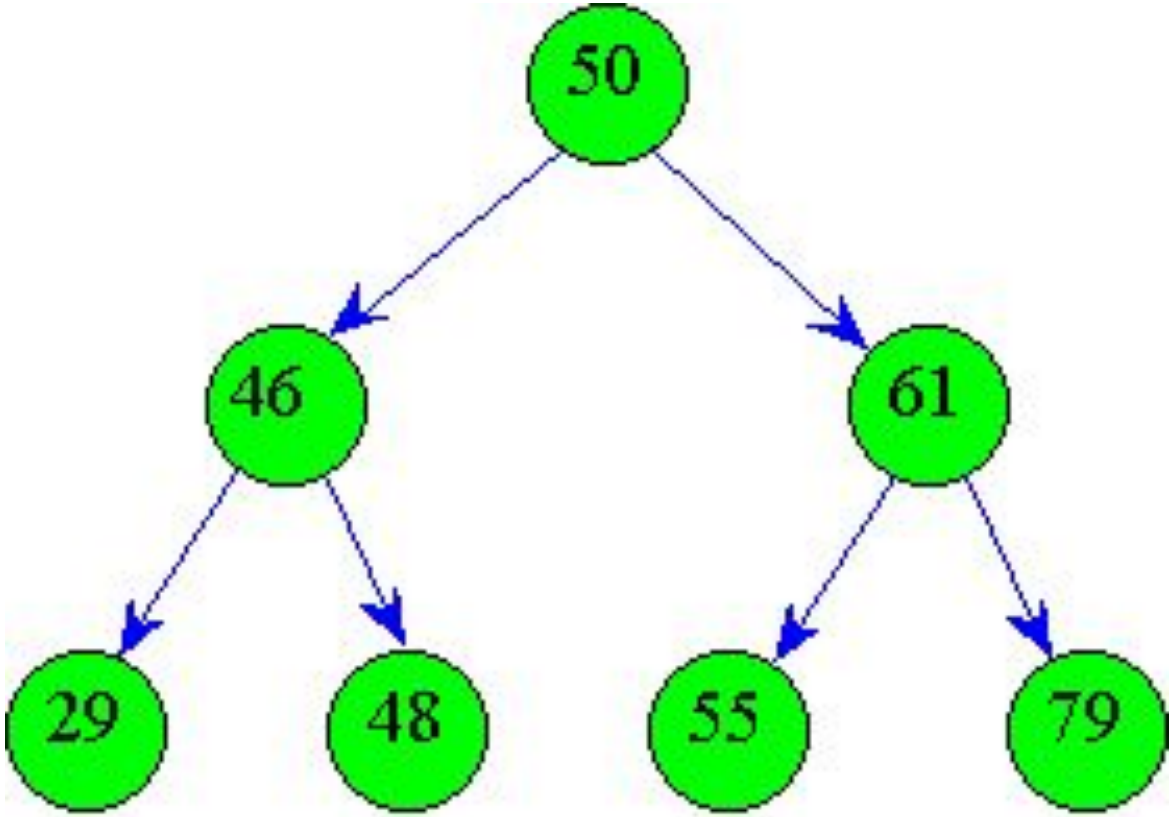
Бинарные деревья

- Согласно представлению деревьев в памяти ЭВМ каждый элемент бинарного дерева является записью, содержащей четыре поля. Значения этих полей - соответственно ключ записи, текст записи, ссылка на элемент влево – вниз и ссылка на элемент вправо – вниз.

Формат элемента бинарного дерева

Упорядоченное бинарное дерево

- В упорядоченном бинарном дереве **левый** сын имеет ключ **меньший**, чем у отца, а значение ключа **правого** сына **больше** значения ключа отца.
- Например, построим бинарное дерево из следующих элементов: **50, 46, 61, 48, 29, 55, 79.**



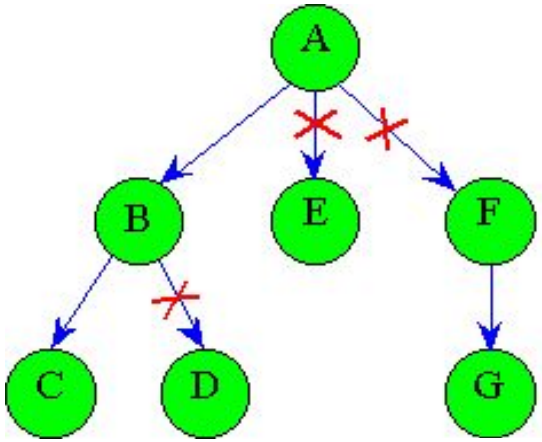
- Получено идеально сбалансированное дерево - дерево, в котором левое и правое поддеревья имеют количество уровней, отличающихся не более чем на единицу.

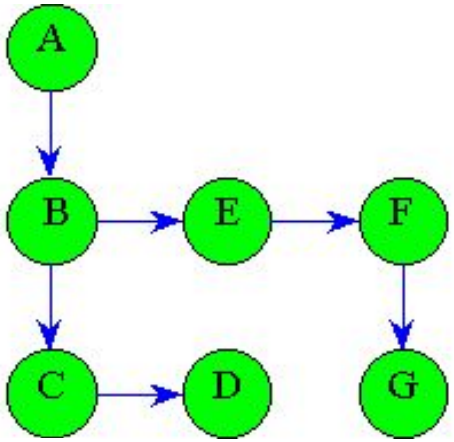
Сведение m-арного дерева к бинарному

- ***Неформальный алгоритм:***

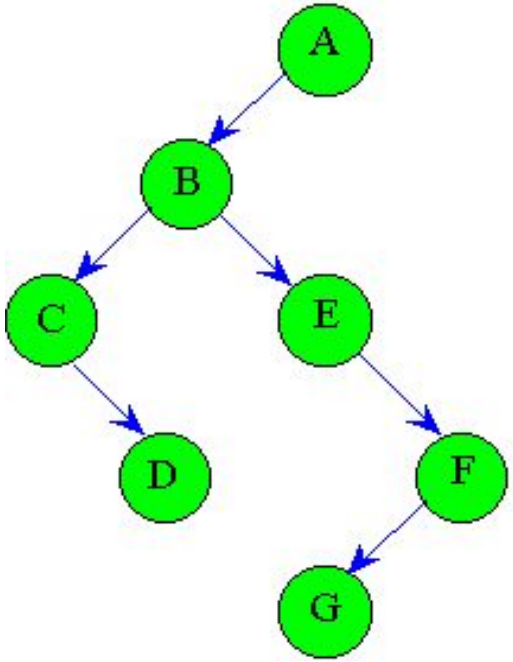
1. В любом узле дерева отсекаются все ветви, кроме крайней ***левой***, соответствующей старшим сыновьям.
2. Соединяются горизонтальными линиями все сыновья ***одного*** родителя.
3. Старшим (левым) сыном в любом узле полученной структуры будет узел, находящийся ***под*** данным узлом (если он есть).

Графическое пояснение алгоритма

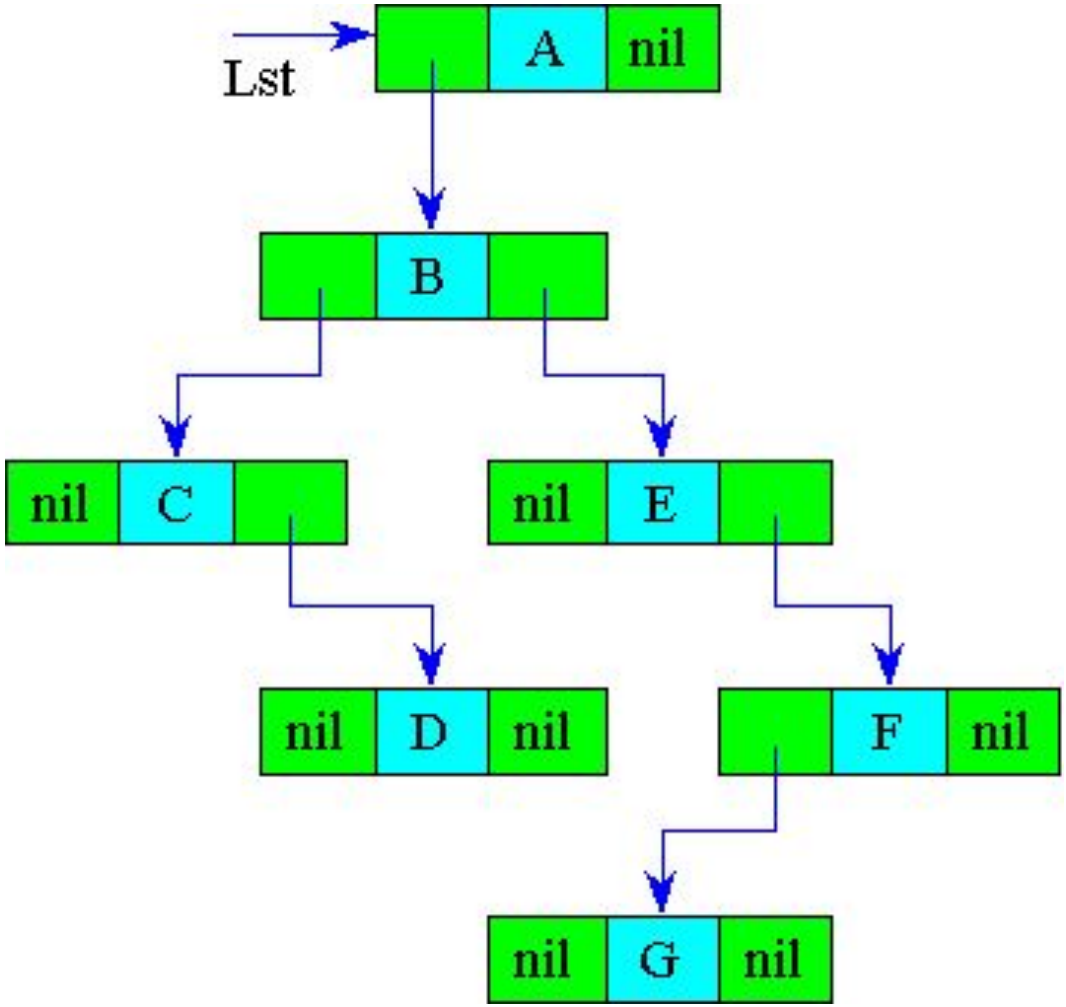




или



Реализация полученного бинарного дерева с помощью нелинейного двусвязного списка



Основные операции с деревьями

1. Обход дерева.
2. Удаление поддерева.
3. Вставка поддерева.

Для выполнения ***обхода*** дерева необходимо выполнить три процедуры:

1.Обработка корня.
2.Обработка левой ветви.
3.Обработка правой ветви.

- В зависимости от того, в какой последовательности выполняются эти три процедуры, различают три вида обхода.

1. Обход сверху вниз. Процедуры выполняются в последовательности

1- 2 - 3.

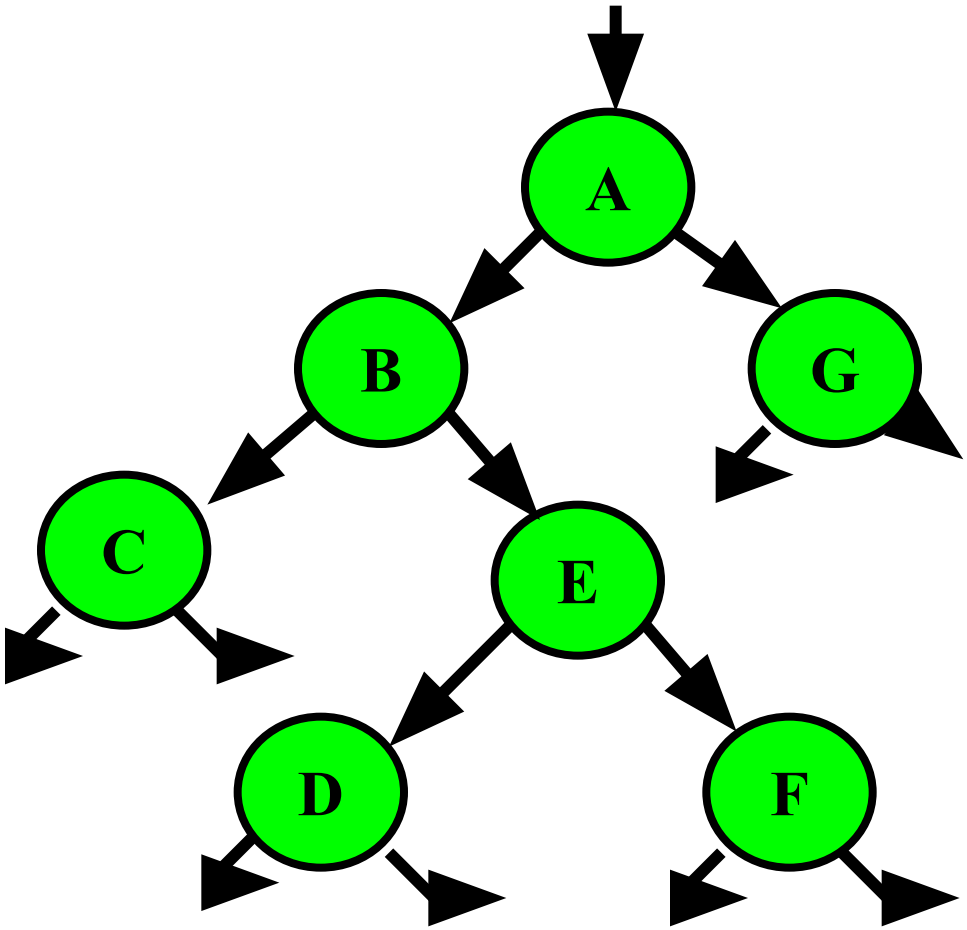
2. Обход слева направо. Процедуры выполняются в последовательности

2 - 1- 3.

3. Обход снизу вверх. Процедуры выполняются в последовательности

2 - 3 -1.

Направления обхода дерева



A-B-C-E-D-F-G – сверху вниз

C-B-D-E-F-A-G – слева направо

C-D-F-E-B-G-A – снизу вверх

- В зависимости от того, после какого по счету захода в узел он подвергается обработке, реализуется один из трех видов обхода.
- Если обработка идет после первого захода в узел, то это обход сверху вниз,
- если после второго - то слева направо,
- если после третьего - то снизу вверх.

Операция *удаления* поддерева

- Необходимо указать узел, к которому подсоединяется удаляемое поддерево и указатель корня этого поддерева.
- Исключение поддерева состоит в том, что разрывается связь с удаляемым поддеревом, т. е. указатель элемента, связанного с узлом-корнем удаляемого поддерева, устанавливается в ***nil***.

Операция *вставки* поддерева

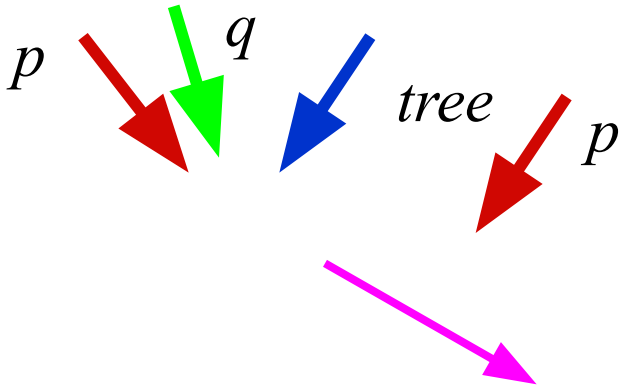
- Необходимо знать адрес корня вставляемого поддерева и узел, к которому подвешивается поддерево.
- Установить указатель этого узла на корень поддерева, а степень исхода данного узла увеличить на единицу.
- При этом, в общем случае, необходимо произвести перенумерацию сыновей узла, к которому подвешивается поддерево.

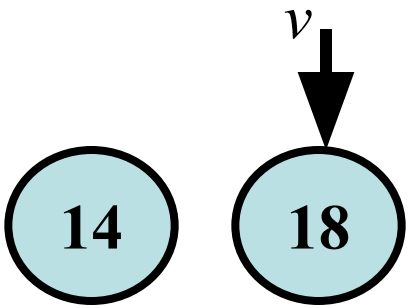
Создание дерева бинарного поиска

- Пусть заданы элементы с ключами: **14, 18, 6, 21, 1, 13, 15.**
- Построим упорядоченное бинарное дерево по этим ключам.

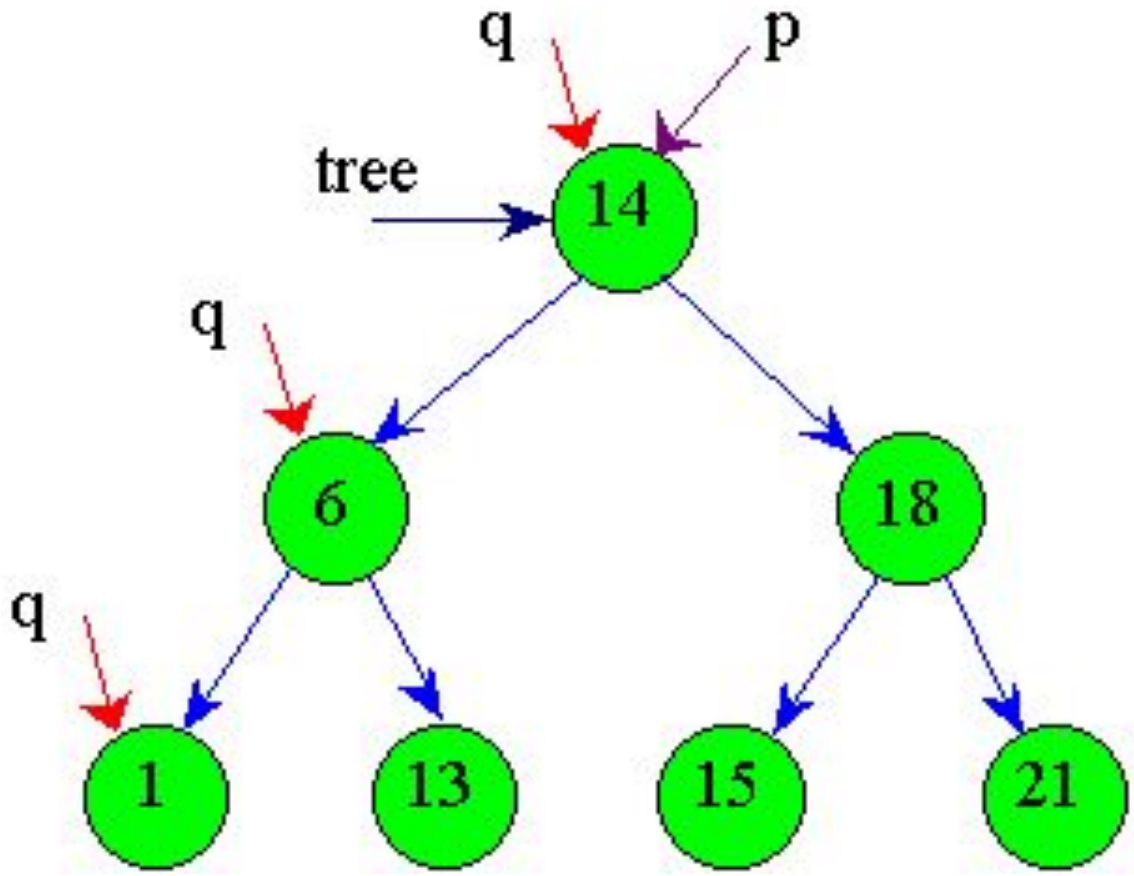
К алгоритму построения дерева

14, 18





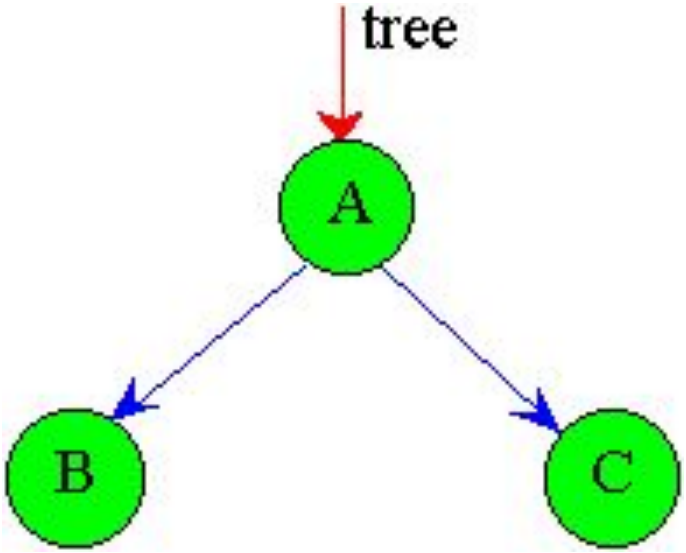
К алгоритму построения дерева



Рекурсивные алгоритмы обхода (прохождения) бинарных деревьев

1. Сверху вниз **A, B, C.**
2. Слева направо или симметричное прохождение **B, A, C.**
3. Снизу вверх **B, C, A.**

Наиболее часто применяется второй способ.



Пояснение рекурсии обхода