Ю. В. КУЛАКОВ, Е. А. БАЙБАКОВ, В. В. СЕВЕНЮК

СТРУКТУРЫ И АЛГОРИТМЫ ОБРАБОТКИ ДАННЫХ

Часть 1

Тамбов
Издательство ФГБОУ ВО «ТГТУ»
2018

Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Тамбовский государственный технический университет»

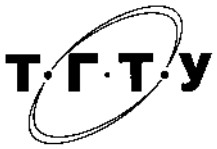
Ю. В. КУЛАКОВ, Е. А. БАЙБАКОВ, В. В. СЕВЕНЮК

СТРУКТУРЫ И АЛГОРИТМЫ ОБРАБОТКИ ДАННЫХ

Часть 1

Утверждено Учёным советом университета в качестве мультимедийного
учебного пособия для студентов направлений подготовки
09.03.02 «Информационные системы и технологии»,
27.03.03 «Системный анализ и управление» и специальности
10.05.03 «Информационная безопасность автоматизированных систем»
очной и заочной форм обучения

Учебное электронное мультимедийное издание



Тамбов
Издательство ФГБОУ ВО «ТГТУ»
2018

УДК 004.42(075.8)
ББК з973-018я73
К90

Р е ц е н з е н т ы:

Доктор технических наук, профессор,
заведующий кафедрой математического и компьютерного моделирования
и информационных технологий ФГБОУ ВО «ТГУ им. Г. Р. Державина»
А. А. Арзамасцев

Кандидат технических наук, доцент,
и.о. заведующего кафедрой «Системы автоматизированной поддержки
принятия решений» ФГБОУ ВО «ТГТУ»
И. Л. Коробова

Кулаков, Ю. В.

К90 Структуры и алгоритмы обработки данных [Электронный ресурс, мультимедиа] : учебное пособие / Ю. В. Кулаков, Е. А. Байбаков, В. В. Севенюк. – Тамбов : Изд-во ФГБОУ ВО «ТГТУ», 2018.

ISBN 978-5-8265-1908-0.

Ч. 1. – 1 электрон. опт. диск (CD-ROM). – Системные требования : ПК не ниже класса Pentium II ; CD-ROM-дисковод ; 00,0 Mb ; RAM ; Windows 95/98/XP ; мышь. – Загл. с экрана.

ISBN 978-5-8265-1909-7.

Содержит теоретический материал, лабораторные работы, раскрывающие вопросы организации линейных информационных структур и алгоритмов их обработки, и список рекомендуемой литературы.

Предназначено для студентов направлений подготовки 09.03.02 «Информационные системы и технологии», 27.03.03 «Системный анализ и управление» и специальности 10.05.03 «Информационная безопасность автоматизированных систем» очной и заочной форм обучения.

УДК 004.42(075.8)

ББК 3973-018я73

ISBN 978-5-8265-1908-0 (общ.)
ISBN 978-5-8265-1909-7 (ч. 1)

ВВЕДЕНИЕ

В первой части учебного пособия представлены лабораторные работы, которые ориентированы на изучение вопросов, связанных с представлением линейных информационных структур в памяти ЭВМ и алгоритмами их обработки.

Лабораторная работа № 1 посвящена решению задачи организации линейной информационной структуры и простейшим алгоритмам её обработки. В лабораторной работе № 2 рассматривается алгоритм перераспределения последовательных таблиц, обеспечивающий сосуществование трёх и более линейных списков в ограниченном пространстве памяти. Лабораторная работа № 3 посвящена связанному распределению памяти при хранении линейных списков и алгоритму топологической сортировки частично упорядоченного множества. В работе № 4 рассматривается связанное представление циклических списков и выполнение операции сложения многочленов, представленных циклическими списками. Работа № 5 посвящена выполнению осевого шага в разреженной матрице, представленной ортогональными связанными циклическими списками. Каждая лабораторная работа содержит цель,

задание в тридцати вариантах, основные теоретические положения и пример выполнения задания.

Новизна данного учебного пособия, в сравнении с ранее изданной литературой по структурам и алгоритмам обработки данных [1 – 5], заключается в рассмотрении примеров с подробнейшими иллюстрациями применения изучаемых алгоритмов.

Применение данного пособия будет способствовать формированию у обучающихся следующих компетенций по направлениям подготовки и специальностям.

По направлению подготовки 09.03.02 «Информационные системы и технологии»: способность разрабатывать средства реализации информационных технологий (методические, информационные, математические, алгоритмические, технические и программные) (ПК-12).

По направлению 27.03.03 «Системный анализ и управление»: готовность применять методы математики, физики, химии, системного анализа, теории управления, теории знаний, теории и технологии программирования, а также методов гуманитарных, экономических и социальных наук (ОПК-1).

По специальности 10.05.03 «Информационная безопасность автоматизированных систем»: способность корректно применять при решении профессиональных задач соответствующий математический аппарат алгебры, геометрии, дискретной математики, математического анализа, теории вероятностей, математической статистики, математической логики, теории алгоритмов, теории информации, в том числе с использованием вычислительной техники (ОПК-2).

Лабораторная работа № 1

ЛИНЕЙНЫЕ ИНФОРМАЦИОННЫЕ СТРУКТУРЫ

Цель. Знакомство с понятием линейных информационных структур и программированием простейших алгоритмов их создания и обработки.

Задание. Разработать на языке Си программу для создания с помощью алгоритма *A* линейной информационной структуры типа стек, которая представляет стопку карт, а также подсчета по алгоритму *B* количества карт в этой стопке. Выполнить графическую иллюстрацию распределения и содержания памяти компьютера в результате выполнения этой программы на ЭВМ. Информацию о картах в стопке карт взять из табл. 1.

Основные положения

Программы для ЭВМ обычно оперируют с таблицами информации, в которых присутствуют важные структурные отношения между элементами данных. В простейшем случае таблица представляет собой линейный список элементов.

Информация в списке представляется множеством записей (объектов), называемых *узлами* (*элементами*) *списка.* Каждый узел списка состоит из одного или нескольких последовательных слов в памяти машины, разделённых на именуемые части, называемые *полями.* Содержимым любого поля в узле могут быть числа, буквы, связи – всё, что только пожелает программист. В качестве примера рассмотрим случай, когда элементы списка представляют игральные карты: мы можем иметь узлы, разбитые на четыре поля – *TAG* (признак), *SUIT* (масть), *RANK* (ранг) и *NEXT* (следующий):

TAG	*SUIT*	*RANK*	*NEXT*

Пусть: *TAG* = 1 означает, что карта повёрнута лицевой стороной вниз, *TAG* = 0 – лицевой стороной вверх; *SUIT* = 1, 2, 3 и 4 – соответственно для трефовых, бубновых, червовых и пиковых карт; *RANK* = 1, 2, …, 13 – для туза, двойки, …, короля; *NEXT* – связь с картой в стопке ниже данной. *Адрес* узла

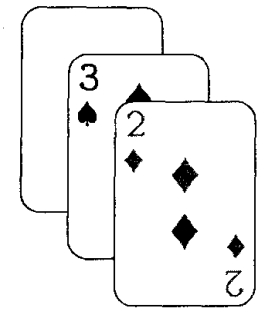
является адресом первого слова в узле. Адрес узла называют *указателем* на этот узел. Тогда некоторая колода из трёх карт может выглядеть так:

Таблица 1

№ вари-анта	*TAG*	*SUIT*	*RANK*	№ вари-анта	*TAG*	*SUIT*	*RANK*	№ вари-анта	*TAG*	*SUIT*	*RANK*
1	0	4	9	11	1	2	13	21	0	3	3
	1	2	4		1	1	13		1	3	10
	0	1	9		0	1	12		0	4	6
	0	2	7		0	3	11		1	3	7
	0	3	10		0	2	5		0	1	8
2	1	3	12	12	0	4	3	22	0	4	1
	0	3	1		1	4	7		0	1	7
	0	2	4		1	2	1		0	4	5
	0	1	11		0	4	1		0	4	8
3	0	1	2	13	0	3	8		0	2	12
	0	4	11		0	3	9	23	0	4	4
	1	1	10		0	1	10		1	3	4
	0	3	3		0	4	10		1	2	5
4	1	4	11	14	1	3	12		1	3	4
	0	2	4		0	4	8		0	1	12
	1	2	11		0	2	11		1	2	8
	0	4	9		0	3	5	24	0	3	12
	1	2	7		0	2	9		0	3	7
5	0	4	8		1	2	7		1	4	5
	0	4	2	15	1	3	13		1	3	11
	1	1	5		0	2	6		0	4	12
	1	3	1		0	1	11	25	1	2	1
	1	4	1		0	2	1		1	1	12
	1	1	3		1	2	12		0	4	11
6	1	4	13	16	1	2	5		0	4	9
	1	3	6		0	3	7	26	1	1	11
	1	1	12		1	1	3		0	3	8
	1	3	11		1	3	1		0	3	3
	1	1	4	17	1	3	10		1	1	12
	0	2	10		1	4	10	27	1	3	3

7	0	1	1		0	1	12		0	1	13
	0	4	3		1	3	2		0	3	9
	1	1	9		0	2	6		1	1	3
	1	1	4		0	1	8		0	1	7
	1	3	13	18	1	3	7	28	0	4	10
	1	3	6		0	1	7		1	2	3
8	0	1	7		1	3	8		1	3	7
	1	2	4		0	3	7		1	2	9
	0	2	6		0	3	4		1	2	3
	0	3	3	19	0	1	7	29	0	4	11
	0	1	6		0	1	8		1	4	10
9	1	1	10		0	1	11		0	3	8
	1	3	10		0	3	10		1	1	13
	1	3	4		0	3	3		0	1	4
	1	3	2		1	2	6	30	1	2	8
	0	3	7	20	1	2	13		1	1	1
10	0	2	5		0	1	3		1	4	4
	0	3	13		1	4	3		0	1	1
	0	1	3		0	1	1		0	2	12
	1	3	2		1	3	12		1	1	6

Фактические карты *Машинное представление*

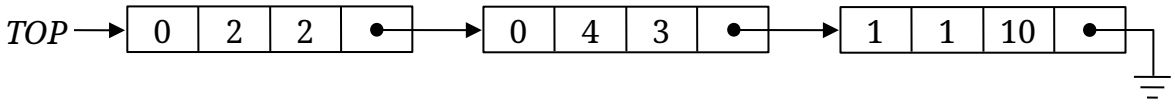


100:	1	1	10	Λ
386:	0	4	3	100
242:	0	2	2	386

В машинном представлении указаны адреса памяти 100, 386 и 242, но на их месте могли быть любые другие числа, поскольку каждая карта ссылается на следующую карту в колоде. Специальный указатель Λ (ламбда) в узле 100 обозначает *пустую связь*, т.е. связь с узлом, которого не существует (десятка треф – нижняя карта в стопке).

Введение связей с другими элементами данных является чрезвычайно важной идеей в программировании – это ключ к представлению сложных информационных структур. В машинном

представлении списка связи между узлами удобно изображать стрелками. Так, для нашего примера список приобретает следующий вид:



Фактические адреса 242, 386 и 100 (значение которых не суть важно) отсутствуют в данном представлении, а пустая связь изображена как «заземление» для электрических схем. Символ *TOP* представляет *переменную связи* или, как часто говорят, указатель, т.е. переменную, значением которой является адрес памяти. Для осуществления ссылок на поля узлов будем использовать имя поля, за которым в скобках следует связь с желаемым узлом:

$$TAG\,(TOP) = 0; \quad SUIT\,(TOP) = 2; \quad RANK\,(100) = 10;$$
$$RANK\,(NEXT\,(TOP)) = 3.$$

В качестве примера рассмотрим простой алгоритм, который помещает новую карту в колоду сверху, предполагая, что *NEWCARD* – переменная связи и её значением является связь с новой картой:

*A*1. Установить *NEXT* (*NEWCARD*) ← *TOP*. (Устанавливается соответствующая связь в узле новой карты.)

*A*2. Установить *TOP* ← *NEWCARD*. (Тем самым обеспечивается, что *TOP* по-прежнему указывает на верхнюю карту в колоде.)

*A*3. Установить *TAG* (*TOP*) ← 0. (Отмечается, что карта повёрнута лицевой стороной вверх.) ■

Другим примером является алгоритм, подсчитывающий количество карт в колоде в данный момент:

*B*1. Установить $N \leftarrow 0$, $X \leftarrow TOP$. (N – целая переменная, X – переменная связи.)

*B*2. Если $X = \Lambda$, то остановиться. (N равно числу карт в колоде.)

*B*3. Установить $N \leftarrow N + 1$, $X \leftarrow NEXT\,(X)$ и вернуться к шагу *B*2. ■

Линейный список – это множество, состоящее из $n \geq 0$ узлов $x[1]$, $x[2]$, …, $x[n]$, структурные свойства которого ограничиваются только линейным относительным положением узлов: $x[1]$ – первый узел; при $1 < k < n$ узлу $x[k]$ предшествует узел $x[k-1]$, а следует за ним узел $x[k+1]$; $x[n]$ – последний узел.

Возможные операции с линейными списками:

1) получение доступа к k-му узлу списка, чтобы проанализировать и/или изменить содержимое его полей;

2) включение нового узла в список непосредственно перед k-м узлом;

3) исключение k-го узла списка;

4) объединение двух (или более) списков в один;

5) разбиение списка на два (или более) списков;

6) копирование списка;

7) определение количества узлов в списке;

8) сортировка узлов списка по некоторым полям;

9) поиск в списке узла с заданным значением в некотором поле.

Случаи при $k = 1$ и $k = n$ в операциях 1), 2), 3) особые, поскольку в линейном списке доступ к первому и последнему узлу получить проще.

Очень часто на практике встречаются линейные списки, в которых включение, исключение и доступ к значениям производятся в первом или последнем узлах. Среди таких списков различают:

- *стек* – список, в котором все включения, все исключения и всякий доступ выполняются в одном конце этого списка;
- *очередь* – список, в котором все включения производятся на одном конце этого списка, а все исключения и всякий доступ – на другом его конце;
- *дек* – список, в котором включения, исключения и всякий доступ делаются на обоих концах этого списка.

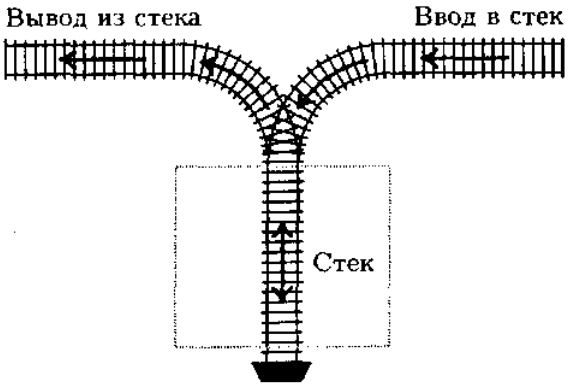


Рис. 1. Стек, представленный в виде железнодорожного разъезда

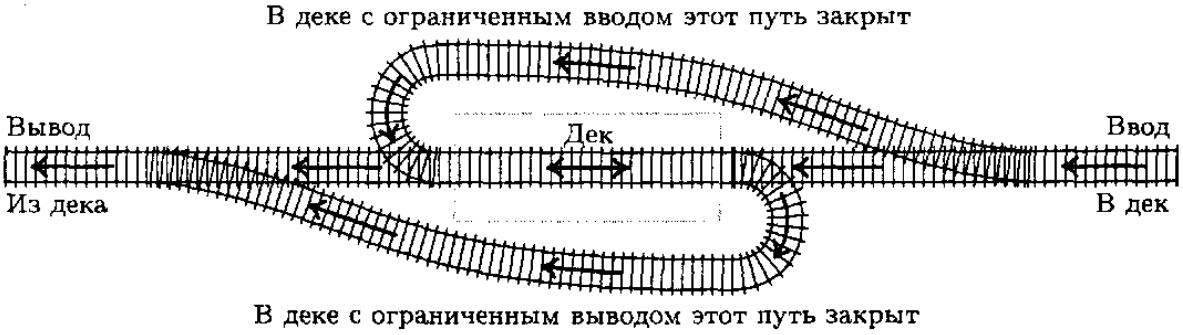


Рис. 2. Дек, представленный в виде железнодорожного разъезда

Пример выполнения задания

Задание. Разработать на языке Си программу для создания с помощью алгоритма *A* линейной информационной структуры типа стек, которая представляет стопку карт:

- верхняя карта: *TAG* = 0, *SUIT* = 2, *RANK* = 2,

- вторая сверху: *TAG* = 0, *SUIT* = 4, *RANK* = 3,

- третья сверху (нижняя): *TAG* = 1, *SUIT* = 1, *RANK* = 10,

а также подсчета по алгоритму *B* количества карт в этой стопке. Выполнить графическую иллюстрацию распределения и содержания памяти компьютера в результате выполнения этой программы на ЭВМ.

Текст программы на языке Си

```
#define WORD struct word_type
#include<stdio.h>
#include<stdlib.h>
WORD
{
      int TAG;    /* признак: 1 - карта повернута лицевой стороной вниз */
                  /*          0 - карта повернута лицевой стороной вверх */
      int SUIT;   /* масть: 1 - трефы, 2 - бубны, 3 - черви, 4 - пики */
      int RANK;   /* ранг: 1, 2, ...,13 - туз, двойка, ..., король */
      WORD *NEXT; /* следующий */
};
void main()
{
      int N;
      WORD *TOP,     /* указатель на верхний элемент стека */
           *NEWCARD, /* указатель на новый элемент стека */
           *X;       /* указатель на текущий элемент стека */
      /* Создание пустого стека  */
      TOP=NULL;

      /* Помещение нижней карты в пустую стопку  */
      NEWCARD=(WORD *)malloc(sizeof(WORD));
      printf("NEWCARD=%p\n",NEWCARD);
      /*A1*/ NEWCARD->NEXT=TOP;
      /*A2*/ TOP=NEWCARD;
      /*A3*/ TOP->TAG=1; TOP->SUIT=1; TOP->RANK=10;

      /* Добавление новой карты в стопку сверху  */
      NEWCARD=(WORD *)malloc(sizeof(WORD));
      printf("NEWCARD=%p\n",NEWCARD);
      /*A1*/ NEWCARD->NEXT=TOP;
      /*A2*/ TOP=NEWCARD;
      /*A3*/ TOP->TAG=0; TOP->SUIT=4; TOP->RANK=3;

      /* Добавление вехней карты в стопку  */
      NEWCARD=(WORD *)malloc(sizeof(WORD));
      printf("NEWCARD=%p\n",NEWCARD);
      /*A1*/ NEWCARD->NEXT=TOP;
      /*A2*/ TOP=NEWCARD;
      /*A3*/ TOP->TAG=0; TOP->SUIT=2; TOP->RANK=2;

      printf("TAG(TOP)=%d\n",TOP->TAG);
      printf("SUIT(TOP)=%d\n",TOP->SUIT);
      printf("RANK(NEXT(TOP))=%d\n",TOP->NEXT->RANK);

      /* Подсчет количества карт в стопке  */
      B1: N=0; X=TOP;
      B2: if(X==NULL) goto end;
      B3: N++; X=X->NEXT; goto B2;
        end: printf("N=%d\n",N);
```

```
}
```

Протокол работы программы,

с соответствующим распределением и содержанием памяти компьютера,

а также графическим представлением связанного списка

```
TOP=NULL
```

Фактические карты *Машинное представление*

Стопка карт пуста *TOP = NULL*

Графическое представление связанного списка

Указатель *TOP* никуда не указывает

```
/* Помещение нижней карты в пустую стопку  */
NEWCARD=(WORD *)malloc(sizeof(WORD))=07D8
/*A1*/ NEWCARD->NEXT=TOP=NULL
/*A2*/ TOP=NEWCARD=07D8
/*A3*/ TOP->TAG=1; TOP->SUIT=1; TOP->RANK=10
```

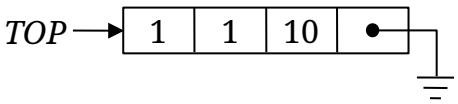
Фактические карты *Машинное представление*

TOP = 07*D*8

07*D*8:

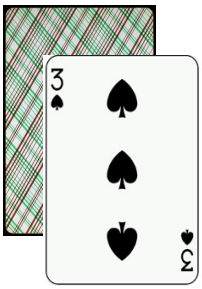
1	1	10	Λ

Графическое представление связанного списка



```
/* Добавление новой карты в стопку сверху  */
NEWCARD=(WORD *)malloc(sizeof(WORD))=07E4
/*A1*/ NEWCARD->NEXT=TOP=07D8
/*A2*/ TOP=NEWCARD=07E4
/*A3*/ TOP->TAG=0; TOP->SUIT=4; TOP->RANK=3
```

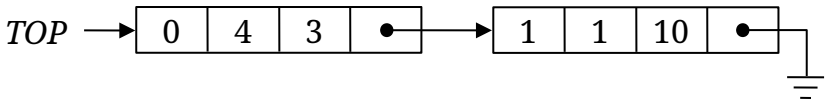
Фактические карты *Машинное представление*



TOP = 07*E*4

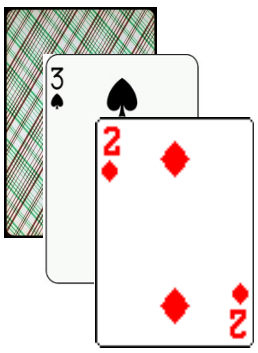
07D8:	1	1	10	Λ
07E4:	0	4	3	07D8

Графическое представление связанного списка




```
/* Добавление вехней карты в стопку  */
NEWCARD=(WORD *)malloc(sizeof(WORD))=07F0
/*A1*/ NEWCARD->NEXT=TOP=07E4
/*A2*/ TOP=NEWCARD=07F0
/*A3*/ TOP->TAG=0; TOP->SUIT=2; TOP->RANK=2
```

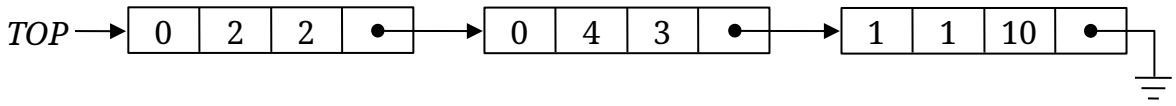
Фактические карты *Машинное представление*



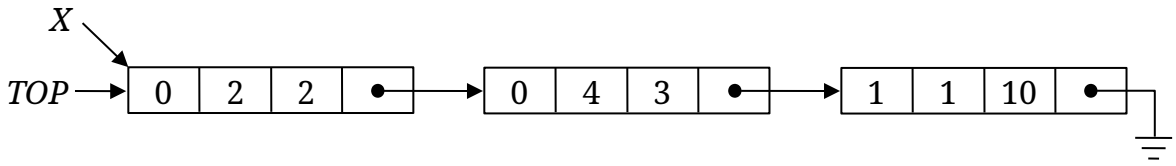
TOP = 07*F*0

07D8:	1	1	10	Λ
07E4:	0	4	3	07D8
07F0:	0	2	2	07E4

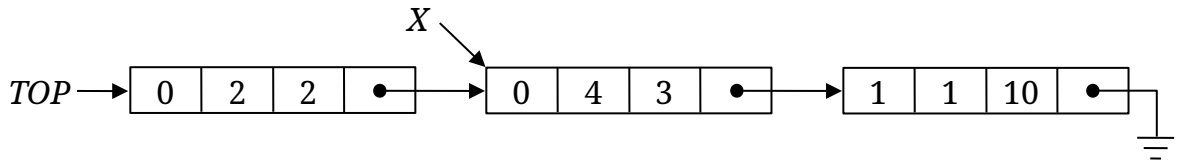
Графическое представление связанного списка



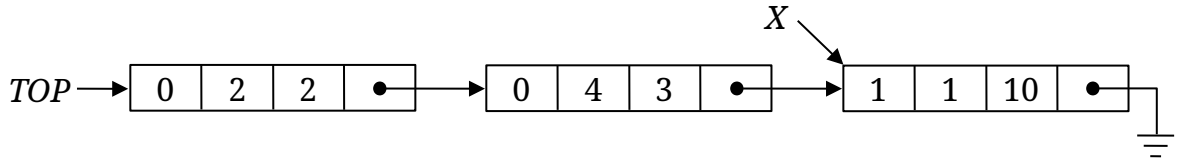

```
/* Подсчет количества карт в стопке  */
B1: N=0; X=TOP=07F0
```



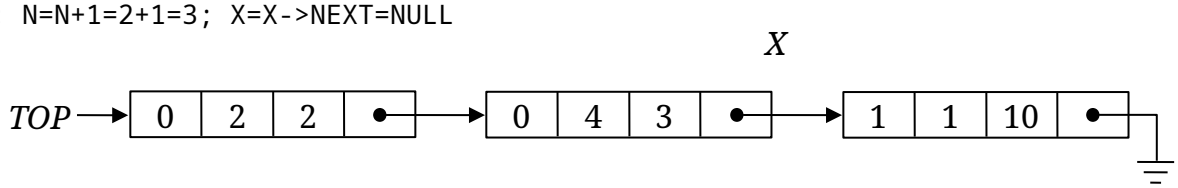

```
B2: (X==NULL)=(07F0==NULL)=false
B3: N=N+1=0+1=1; X=X->NEXT=07E4
```




```
B2: (X==NULL)=(07E4==NULL)=false
B3: N=N+1=1+1=2; X=X->NEXT=07D8
```




```
B2: (X==NULL)=(07D8==NULL)=false
B3: N=N+1=2+1=3; X=X->NEXT=NULL
```




```
B2: (X==NULL)=(NULL==NULL)=true
end: N=3
```

Лабораторная работа № 2

АЛГОРИТМ ПЕРЕРАСПРЕДЕЛЕНИЯ

ПОСЛЕДОВАТЕЛЬНЫХ ТАБЛИЦ

Цель. Знакомство с алгоритмом перераспределения последовательных таблиц, обеспечивающим сосуществование трёх и более линейных списков в ограниченном пространстве памяти.

Задание. Пусть состояние памяти, наступившее по окончании последней переупаковки четырёх последовательных стеков, задано их базовыми адресами и адресами верхних элементов. В результате выполнения заданной последовательности операций по включению новых элементов в эти списки возникает ситуация «ПЕРЕПОЛНЕНИЕ» и начинает свою работу алгоритм *G* перераспределения последовательных таблиц. Определить состояние памяти и значения указателей *BASE* [*j*] и *TOP* [*j*] ($j = 1, 2, 3, 4$) непосредственно после выполненной переупаковки памяти. Варианты конкретных исходных данных представлены в табл. 2.

Таблица 2

Вар и-ант	*BASE* [1]	*BASE* [2]	*BASE* [3]	*BASE* [4]	*TOP* [1]	*TOP* [2]	*TOP* [3]	*TOP* [4]	Последовательность операций включения
1	–1	3	6	8	0	5	7	10	*I*3*I*1*I*4*I*1*I*2*I*2
2	–1	2	7	9	–1	5	7	10	*I*1*I*1*I*2*I*4*I*4*I*4*I*4*I*3*I*2*I*3 *I*2
3	–1	1	2	5	0	1	3	8	*I*4*I*1*I*1
4	–1	3	5	9	2	5	6	9	*I*4*I*4*I*4*I*2
5	–1	1	4	5	0	1	4	6	*I*2*I*3I1*I*4*I*2*I*3
6	–1	–1	1	1	–1	1	1	2	*I*4*I*4*I*4*I*4*I*1
7	–1	2	7	11	2	4	10	14	*I*4*I*2*I*1
8	–1	1	3	9	–1	3	6	9	*I*2
9	–1	4	9	13	1	7	11	14	*I*1*I*3*I*2*I*1*I*4*I*4*I*2*I*4*I*2
10	–1	1	6	11	1	4	9	14	*I*2*I*3*I*4*I*4*I*2*I*3*I*2
11	–1	2	6	8	0	3	6	8	*I*4*I*3*I*3I3
12	–1	3	8	11	0	5	11	11	*I*1*I*4*I*3
13	–1	5	6	10	2	6	7	12	*I*1*I*3*I*3*I*4*I*3*I*3
14	–1	2	3	7	2	3	6	9	*I*3*I*2
15	–1	3	7	10	1	6	7	10	*I*3*I*3*I*2*I*4*I*2
16	–1	4	7	9	1	4	9	10	*I*2*I*1*I*2*I*2*I*3
17	–1	3	7	10	1	5	10	12	*I*4*I*1*I*1*I*4*I*4*I*4I2*I*3
18	–1	1	3	6	0	2	3	9	*I*2*I*3*I*1*I*1
19	–1	2	4	7	2	4	6	8	*I*4*I*3*I*2
20	–1	5	10	13	2	8	10	15	*I*4*I*4*I*4*I*2*I*4*I*4
21	–1	3	6	9	2	6	7	9	*I*4*I*1*I*4*I*4*I*2
22	–1	2	5	6	1	5	6	8	*I*1*I*3
23	–1	2	4	7	1	2	7	9	*I*1*I*1
24	–1	4	10	15	2	7	12	16	*I*4*I*3I2*I*2*I*3*I*4*I*1*I*4*I*3*I*4
25	–1	1	7	8	0	4	7	11	*I*1*I*4*I*2*I*4*I*3*I*2*I*4*I*1
26	–1	–1	–1	1	–1	–1	1	4	*I*4*I*2

27	–1	2	4	8	0	4	7	10	*I*2
28	–1	2	2	8	–1	2	5	9	*I*1*I*1*I*2
29	–1	2	3	7	2	3	6	7	*I*4*I*4*I*2
30	–1	3	6	10	1	3	7	10	*I*2*I*3*I*1*I*1*I*3*I*4*I*2*I*3*I*3

Основные положения

Простейший и наиболее естественный способ хранения линейного списка в памяти машины сводится к размещению элементов списка в последовательных ячейках памяти, один узел за другим. В этом случае

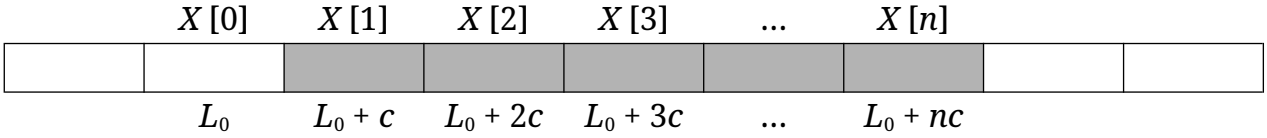
$$LOC\,(X\,[\,j+1\,]) = LOC\,(X\,[\,j\,]) + c,$$

где $LOC\,(X\,[\,j+1\,])$, $LOC\,(X\,[\,j\,])$ – адреса узлов $X\,[\,j+1\,]$ и $X\,[\,j\,]$ соответственно; c – количество слов в одном узле.

В общем случае адрес узла $X\,[\,j\,]$ определяется выражением

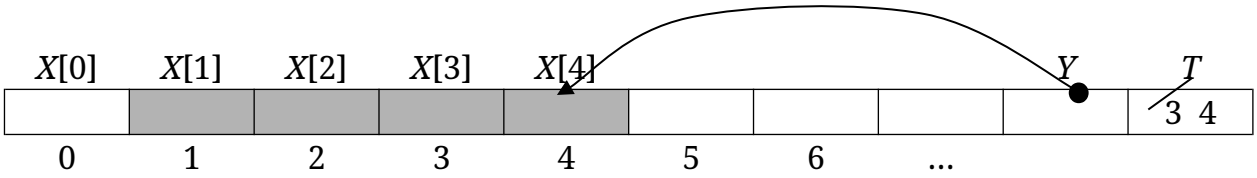
$$LOC\,(X\,[\,j\,]) = L_0 + c\,j, \qquad (1)$$

где L_0 – константа, называемая *базовым адресом* и являющаяся адресом гипотетического узла $X\,[0]$.



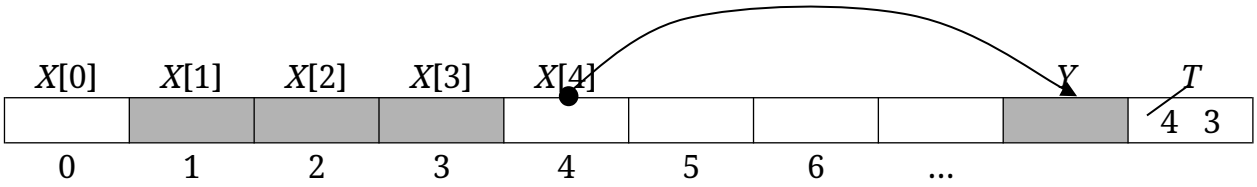
Последовательное распределение очень удобно при работе со стеком. Для этого достаточно иметь *указатель стека* T. Когда стек пуст, $T = 0$. Для того чтобы поместить новый элемент в стек, необходимо (в предположении, что $c = 1$):

$$T \leftarrow T + 1; \quad X\,[T] \leftarrow Y. \qquad (2)$$



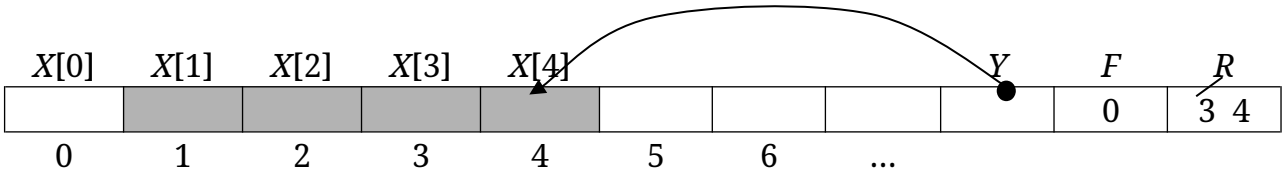
Чтобы переменной Y дать значение верхнего узла непустого стека и исключить этот узел из списка, требуется

$$Y \leftarrow X[T]\ ;\ T \leftarrow T - 1. \qquad (3)$$



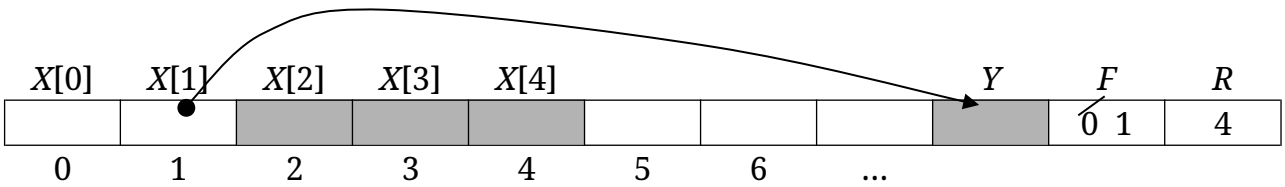
Представление очереди или, более того, дека общего вида требует некоторых ухищрений. Выберем две переменные: указатель F для начала очереди и указатель R для конца очереди. Тогда включение элемента в конец очереди осуществляется следующим образом:

$$R \leftarrow R + 1;\ \ X[R] \leftarrow Y, \qquad (4)$$



а исключение начального узла (F указывает на место, непосредственно перед началом очереди) так:

$$F \leftarrow F + 1;\ \ Y \leftarrow X[F];\ \ \text{если } F = R, \text{ то } F \leftarrow R \leftarrow 0. \qquad (5)$$



Заметим, что в такой ситуации (когда в очереди есть по крайней мере один узел) R и F растут, причём $R > F$. В результате последовательно занимаются ячейки $X[1]$, $X[2]$, … до бесконечности, что свидетельствует о чрезмерно расточительном использовании памяти. Следовательно, простой метод (4), (5) должен использоваться в такой ситуации, когда известно, что указатель F довольно регулярно догоняет указатель R (очередь опустошается).

Обойти проблему выхода очереди за пределы памяти можно зафиксировав M узлов $X[1]$, …, $X[M]$ и неявно «замкнув» их в кольцо так, что за $X[M]$ следует $X[1]$. Тогда процессы (4), (5) преобразуются к виду:

если $R = M$, то $R \leftarrow 1$; иначе $R \leftarrow R + 1$; $X[R] \leftarrow Y$. (6)

если $F = M$, то $F \leftarrow 1$; иначе $F \leftarrow F + 1$; $Y \leftarrow X[F]$. (7)

Очевидно, что в очереди при такой ситуации не может быть более M узлов.

Представим действия с включением и исключением узлов для стека (2), (3) и очереди (6), (7) с учётом того, что в результате включения узла может возникнуть выход за отведённое место в памяти (ПЕРЕПОЛНЕНИЕ) или при попытке исключения отсутствующего узла в списке (НЕХВАТКА):

X ⇐ Y (включить в стек); $\begin{cases} T \leftarrow T + 1, & \text{если } T > M, \text{ то ПЕРЕПОЛНЕ} \\ X[T] \leftarrow Y. \end{cases}$

Y ⇐ X (исключить из стека); $\begin{cases} \text{если } T = 0, \text{ то НЕХВА} \\ Y \leftarrow X[T], \quad T \leftarrow T - 1. \end{cases}$

X ⇐ Y $\begin{cases} \text{если } R = M, \text{ то } R \\ \text{если } R = F, \text{ то П} \\ X[R] \leftarrow Y. \end{cases}$

(включить в
очередь);

(начальное условие $F = R = 1$; M – 1 элементов может одновременно находиться в очереди без возникновения переполнения)

Y ⇐ X

$$\begin{cases} \text{если } R = F, \text{ то НЕХ} \\ \text{если } F = M, \text{ то } F \leftarrow \\ Y \leftarrow X[F]. \end{cases}$$

(исключить
из
очереди);

Заметим, что ПЕРЕПОЛНЕНИЕ является критической ошибкой.

Рассмотрим случаи, когда программа работает не с единственным списком. Если имеется всего два стека переменного размера, то они могут хорошо сосуществовать вместе, когда будут расти навстречу друг другу:

Начало памяти *Низ* *Верх* *Верх* *Низ* *Конец памяти*

Ситуация ПЕРЕПОЛНЕНИЕ не возникнет до тех пор, пока суммарный объём обоих списков не исчерпает всё свободное пространство в памяти. Такое распределение памяти используется очень часто.

Можно легко убедиться в том, что не существует способа, который позволяет хранить три и более стека переменного размера так, чтобы:

1) ПЕРЕПОЛНЕНИЕ возникало лишь в случае, когда суммарный размер всех списков превысит отведённую для них область памяти;

2) «нижний» элемент каждого списка имел фиксированный адрес.

Если удовлетворять первому условию, то придётся нарушить второе. Это означает, что базовый адрес L_0 в формуле (1) не является более константой и никакой указатель не может быть абсолютным адресом. Кроме того, все ссылки должны быть относительны L_0.

Предположим, что имеется n стеков, $BASE$ [i] – базовый адрес, а TOP[i] – указатель i-го стека. Тогда алгоритмы включения и исключения узлов становятся такими:

(Включение) TOP [i] ← TOP [i] + 1; если TOP [i] > $BASE$ [i + 1], то ПЕРЕПОЛНЕНИЕ;
в противном случае
$CONTENTS$ (TOP [i]) ← Y. (8)

(Исключение) если TOP [i] = $BASE$ [i], то НЕХВАТКА;
в противном случае
Y ← $CONTENTS$ (TOP [i]), TOP [i] ← TOP [i] – 1.

В данной ситуации критичность ПЕРЕПОЛНЕНИЯ можно устранить «переупаковав память». Рассмотрим простейший метод переупаковки.

Пусть n стеков располагаются в общей области памяти, состоящей из ячеек с адресами L такими, что $L_0 < L \leq L_\infty$, где L_0, L_∞ – границы области памяти, предоставленной для использования. Можно считать, что вначале все стеки пусты ($BASE$ [i] = TOP [i] = L_0 для всех i). Для правильного выполнения операции включения (8) при $i = n$ базовый адрес $BASE$ [n + 1] гипотетического (n + 1)-го стека следует считать равным L_∞. Теперь ПЕРЕПОЛНЕНИЕ будет возникать всякий раз, когда в некоторый стек, за исключением n-го, потребуется записать элементов больше, чем когда-либо прежде.

При переполнении i-го стека реализуется одна из трёх возможностей:

1) Определяется наименьшее k (если такое значение существует), для которого $i < k \leq n$ и TOP [k] < $BASE$ [k + 1] (k-й список заканчивается раньше, чем начинается (k + 1)-й). Затем осуществляется сдвиг на одну позицию *вверх*:

$$CONTENTS\,(L + 1) \leftarrow CONTENTS\,(L),$$

для $L = TOP\,[k], TOP\,[k] - 1, \ldots, BASE\,[i + 1] + 1.$

И выполняются операторы $BASE\,[j] \leftarrow BASE\,[j] + 1$, $TOP\,[j] \leftarrow TOP\,[j] + 1$ для $j = i + 1, i + 2, \ldots, k$.

2) Нельзя найти значение k, удовлетворяющее условию 1), но имеется наибольшее k, для которого $1 \le k < i$ и $TOP\,[k] < BASE\,[k + 1]$. Теперь осуществляется сдвиг на одну позицию *вниз*:

$$CONTENTS\,(L - 1) \leftarrow CONTENTS\,(L),$$

для $L = BASE\,[k + 1] + 1, BASE\,[k + 1] + 2, \ldots, TOP\,[i]$.

И выполняется $BASE\,[j] \leftarrow BASE\,[j] - 1$, $TOP\,[j] \leftarrow TOP\,[j] - 1$ для $j = k + 1, k + 2, \ldots, i$.

3) Для всех $k \neq i$ имеет место $TOP\,[k] = BASE\,[k + 1]$. Тогда невозможно найти место для нового элемента i-го стека и работу со стеками следует прекратить.

Понятно, что многих первых переполнений стеков можно избежать, если разумно выбрать начальные условия, а не отводить с самого начала всю память под n-й стек. Например, если ожидается, что стеки будут одинакового размера, то можно начать с равномерного распределения памяти:

$$BASE\,[j] = TOP\,[j] = c\left\lfloor \left(\frac{j-1}{n}\right)(L_\infty - L_0)\right\rfloor + L_0.$$

Однако каким бы хорошим ни было начальное распределение, оно позволяет сэкономить лишь фиксированное количество переполнений на ранней стадии работы программы.

Описанный простейший метод можно улучшить, если при каждой переупаковке памяти готовить место для более чем одного нового элемента. При этом выигрыш во времени достигается за счёт того, что

сдвиг таблицы на несколько позиций сразу выполняется быстрее, чем несколько сдвигов на одну позицию.

Предлагается при возникшей ситуации переполнения производить полную переупаковку памяти, основываясь на изменении размера каждого стека с момента последней переупаковки. Представленный ниже алгоритм использует дополнительный массив с именем *OLDTOP*, в котором хранятся значения, бывшие в массиве *TOP* непосредственно после предыдущего распределения памяти. Первоначально таблицы заполняются, как и прежде, причём $OLDTOP[i] = TOP[i]$, $1 \le i \le n$.

Алгоритм *G*. (*Перераспределение последовательных таблиц.*)

Предполагается, что переполнение случилось в стеке i, в соответствии с операцией включения (8). После выполнения алгоритма окажется, что память либо израсходована вся, либо будет перераспределена так, что можно выполнить $NODE(TOP[i]) \leftarrow Y$. Заметим, что $TOP[i]$ был увеличен ещё в (8), прежде чем начал работу этот алгоритм.

***G*1.** [Начальная установка.] $SUM \leftarrow L_\infty - L_0$, $INC \leftarrow 0$.

***G*2.** [Сбор статистики.] Для $j = 1, 2, \ldots, n$ выполнить $SUM \leftarrow SUM - (TOP[j] - BASE[j])$. Если $TOP[j] > OLDTOP[j]$, то $D[j] \leftarrow TOP[j] - OLDTOP[j]$ и $INC \leftarrow INC + D[j]$; в противном случае $D[j] \leftarrow 0$. (В результате значением *SUM* будет суммарное количество свободного пространства в памяти, а значением *INC* – суммарный рост размеров стеков с момента последнего распределения.)

***G*3.** [Память полна?] Если $SUM < 0$, то дальше работать нельзя.

***G*4.** [Вычисление коэффициентов распределения.] $\alpha \leftarrow 0.1 \times SUM / n$, $\beta \leftarrow 0.9 \times SUM / INC$. (Здесь α и β не целые числа, а дроби и вычислять их следует с достаточной точностью. На следующем шаге приблизительно 10% свободного пространства будут поделены поровну между всеми стеками, а остальные 90% свободной памяти – пропорционально росту размера стеков с момента предыдущего распределения.)

***G*5.** [Вычисление новых базовых адресов.] *NEWBASE* [1] ← *BASE*[1], σ ← 0. Для $j = 2, 3, \ldots, n$ $\tau \leftarrow \sigma + \alpha + D[j-1] - \beta$, *NEWBASE* [*j*] ← *NEWBASE* [*j* – 1] + *TOP* [*j* – 1] – *BASE* [*j* – 1] + $\lfloor\tau\rfloor - \lfloor\sigma\rfloor$ и σ ← τ.

***G*6.** [Переупаковка.] *TOP* [*i*] ← *TOP* [*i*] – 1. (Этим устанавливается фактический размер *i*-го стека для того, чтобы не делалось попыток перемещать информацию, выходящую за его границу.) Выполнить алгоритм *R*, приведённый ниже, и *TOP* [*i*] ← *TOP* [*i*] + 1. (Восстановить значение *TOP*[*i*].) ■

Возможно, самой интересной частью всего этого алгоритма является общий процесс переупаковки, который сейчас опишем. Переупаковка оказывается нетривиальной, поскольку одни части памяти должны сдвигаться вверх, а другие вниз. Заметим, что при этих перемещениях важно не затереть какую-либо полезную информацию.

Алгоритм *R*. (*Перемещение последовательных таблиц.*) Для $j = 1, 2, \ldots, n$ информация, специфицированная с помощью *BASE* [*j*] и *TOP* [*j*], перемещается на новые места, заданные с помощью *NEWBASE* [*j*], а значения *BASE* [*j*] и *TOP* [*j*] соответствующим образом корректируются.

***R*1.** [Начальная установка.] *j* ← 1. (Заметим, что никогда не возникает необходимости перемещать стек 1. Поэтому ради эффективности программист должен поставить наибольший стек первым, если он знает его.)

***R*2.** [Поиск начала сдвига.] (Все стеки от 1 до *j* перемещены нужным образом.) *j* ← *j* + 1. Если *NEWBASE* [*j*] < *BASE* [*j*], то перейти к шагу *R*3; иначе если *NEWBASE* [*j*] > *BASE* [*j*], то перейти к шагу *R*4; иначе если *j* > *n*, то окончить алгоритм; иначе выполнить шаг *R*2 сначала.

***R*3.** [Сдвиг вниз.] δ ← *BASE* [*j*] – *NEWBASE* [*j*]. Для *L* = *BASE* [*j*] + 1, *BASE* [*j*] + 2, ..., *TOP* [*j*] *CONTENTS* (*L* – δ) ← *CONTENTS* (*L*). (Заметим, что *BASE* [*j*]

может оказаться равным *TOP* [*j*], и в этом случае не требуется никаких действий.) *BASE* [*j*] ← *NEWBASE* [*j*], *TOP* [*j*] ← *TOP* [*j*] – δ. Перейти к *R*2.

***R*4.** [Поиск верхней границы сдвига.] Найти наименьшее $k \geq j$, для которого *NEWBASE* [k + 1] ≤ *BASE* [k + 1]. (Заметим, что *NEWBASE* [n + 1] должен быть равен *BASE* [n + 1] и поэтому такое значение k всегда существует.) Затем выполнить шаг *R*5 для $T = k, k-1, \ldots, j$; $j \leftarrow k$ и перейти к *R*2.

***R*5.** [Сдвиг вверх.] δ ← *NEWBASE* [*T*] – *BASE* [*T*]. Выполнить *CONTENTS* (*L* + δ) ← *CONTENTS* (*L*) для *L* = *TOP* [*T*], *TOP* [*T*] – 1, ..., *BASE* [*T*] + 1. (Заметим, что, как и на шаге *R*3, может не потребоваться никаких действий.) *BASE* [*T*] ← *NEWBASE* [*T*], *TOP* [*T*] ← *TOP* [*T*] + δ. ■

Описанные алгоритмы можно адаптировать и для других относительно адресуемых таблиц, в которых текущая информация располагается между указателями *BASE* [*j*] и *TOP* [*j*].

При занятой памяти только на половину алгоритм *G* работает хорошо и по крайней мере даёт правильные результаты при почти заполненной памяти. В последнем случае алгоритм *R*, осуществляющий перемещение последовательных таблиц, работает довольно долго. При большом количестве последовательных таблиц переменного размера не следует думать, что удастся использовать пространство памяти на 100% и все-таки избежать переполнения. Для исключения бессмысленных потерь времени можно остановить алгоритм *G* на шаге *G*3, если значение переменной *SUM* станет меньше величины $S_{\min}$, которую задаёт программист, избегая тем самым излишне трудоёмкие переупаковки.

Пример выполнения задания

Задание. Пусть состояние памяти, наступившее по окончании последней переупаковки четырёх последовательных стеков, задано их

базовыми адресами и адресами верхних элементов: *BASE* [1] = –1; *BASE* [2] = 2; *BASE* [3] = 5; *BASE* [4] = 9; *TOP* [1] = 1; *TOP* [2] = 2; *TOP* [3] = 8; *TOP* [4] = 11. В результате выполнения заданной последовательности операций по включению новых элементов в эти списки *I*2*I*3*I*3 возникает ПЕРЕПОЛНЕНИЕ и начинает свою работу алгоритм *G* перераспределения последовательных таблиц. Определить состояние памяти и значения указателей *BASE* [*j*] и *TOP* [*j*] (*j* = 1, 2, 3, 4) непосредственно после выполненной переупаковки памяти.

Инициализация состояния памяти

Заданные базовые адреса *BASE* [1] = –1; *BASE* [2] = 2; *BASE* [3] = 5; *BASE* [4] = 9 и адреса верхних элементов стеков *TOP* [1] = 1; *TOP* [2] = 2; *TOP* [3] = 8; *TOP* [4] = 11 определяют следующее состояние памяти, наступившее по окончании последней переупаковки этих последовательных стеков:

1_1	1_2					3_1	3_2	3_3		4_1	4_2								
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

Инициализация массива OLDTOP

В дополнительном массиве *OLDTOP* сохраним начальные значения элементов массива *TOP* (значения, бывшие в массиве *TOP* непосредственно после предыдущего распределения памяти):

OLDTOP [1] ← *TOP* [1] = 1; *OLDTOP* [2] ← *TOP* [2] = 2

OLDTOP [3] ← *TOP* [3] = 8; *OLDTOP* [4] ← *TOP* [4] = 11

Включение новых элементов в стеки

Выполним операции включения новых элементов в стеки *I*2*I*3*I*3 в соответствии с алгоритмом (8).

*I*2 (включение нового элемента во второй стек):

(*TOP* [*i*] ← *TOP* [*i*] + 1) = (*TOP* [2] ← *TOP* [2] + 1 = 2 + 1 = 3)

(*TOP* [*i*] > *BASE* [*i* + 1]) = (*TOP* [2] > *BASE* [2 + 1]) = (*TOP* [2] > *BASE* [3]) = (3 > 5) = ЛОЖЬ, следовательно, (*CONTENTS* (*TOP* [*i*]) ← *Y*) = (*CONTENTS* (*TOP* [2]) ← *Y*) = (*CONTENTS* (3) ← *Y*)

1_1	1_2		2_1			3_1	3_2	3_3		4_1	4_2								
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

BASE [1] = –1; BASE [2] = 2; BASE [3] = 5; BASE [4] = 9

TOP [1] = 1; TOP [2] = 3; TOP [3] = 8; TOP [4] = 11

*I*3 (включение нового элемента во третий стек):

(*TOP* [*i*] ← *TOP* [*i*] + 1) = (*TOP* [3] ← *TOP* [3] + 1 = 8 + 1 = 9)

(*TOP* [*i*] > *BASE* [*i* + 1]) = (*TOP* [3] > *BASE* [3 + 1]) = (*TOP* [3] > *BASE* [4]) = (9 > 9) = ЛОЖЬ, следовательно, (*CONTENTS* (*TOP* [*i*]) ← *Y*) = (*CONTENTS* (*TOP* [3]) ← *Y*) = (*CONTENTS* (9) ← *Y*)

1_1	1_2		2_1			3_1	3_2	3_3	3_4	4_1	4_2								
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

BASE [1] = –1; BASE [2] = 2; BASE [3] = 5; BASE [4] = 9

TOP [1] = 1; TOP [2] = 3; TOP [3] = 9; TOP [4] = 11

*I*3 (включение нового элемента во третий стек):

(*TOP* [*i*] ← *TOP* [*i*] + 1) = (*TOP* [3] ← *TOP* [3] + 1 = 9 + 1 = 10)

(*TOP* [*i*] > *BASE* [*i* + 1]) = (*TOP* [3] > *BASE* [3 + 1]) = (*TOP* [3] > *BASE* [4]) = (10 > 9) = ИСТИНА, следовательно, фиксируется ситуация ПЕРЕПОЛНЕНИЕ

1_1	1_2		2_1			3_1	3_2	3_3	3_4	4_1	4_2								
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

BASE [1] = –1; BASE [2] = 2; BASE [3] = 5; BASE [4] = 9

TOP [1] = 1; TOP [2] = 3; TOP [3] = 10; TOP [4] = 11

Переупаковка памяти

***G*1.** [Начальная установка.] $SUM \leftarrow L_\infty - L_0 = 19 - (-1) = 19 + 1 = 20$. $INC \leftarrow 0$.

***G*2.** [Сбор статистики.]

$j = 1$:

$SUM \leftarrow SUM - (TOP\,[\,j\,] - BASE\,[\,j\,]) = 20 - (TOP\,[\,1\,] - BASE\,[\,1\,]) =$

$= 20 - (1 - (-1)) = 20 - 2 = 18$

$(TOP\,[\,j\,] > OLDTOP\,[\,j\,]) = (TOP\,[\,1\,] > OLDTOP\,[\,1\,]) = (1 > 1) =$ ЛОЖЬ, следовательно, $(D\,[\,j\,] \leftarrow 0) = (D\,[\,1\,] \leftarrow 0)$

$j = 2$:

$SUM \leftarrow SUM - (TOP\,[\,j\,] - BASE\,[\,j\,]) = 18 - (TOP\,[\,2\,] - BASE\,[\,2\,]) =$

$= 18 - (3 - 2) = 18 - 1 = 17$

$(TOP\,[\,j\,] > OLDTOP\,[\,j\,]) = (TOP\,[\,2\,] > OLDTOP\,[\,2\,]) = (3 > 2) =$ ИСТИНА, следовательно, $(D\,[\,j\,] \leftarrow TOP\,[\,j\,] - OLDTOP\,[\,j\,]) = (D\,[\,2\,] \leftarrow TOP\,[\,2\,] - OLDTOP\,[\,2\,] = 3 - 2 = 1)$ и $INC \leftarrow INC + D\,[\,j\,] = 0 + D\,[\,2\,] = 0 + 1 = 1$

$j = 3$:

$SUM \leftarrow SUM - (TOP\,[\,j\,] - BASE\,[\,j\,]) = 17 - (TOP\,[\,3\,] - BASE\,[\,3\,]) =$

$= 17 - (10 - 5) = 17 - 5 = 12$

$(TOP\,[\,j\,] > OLDTOP\,[\,j\,]) = (TOP\,[\,3\,] > OLDTOP\,[\,3\,]) = (10 > 8) =$ ИСТИНА, следовательно, $(D\,[\,j\,] \leftarrow TOP\,[\,j\,] - OLDTOP\,[\,j\,]) = (D\,[\,3\,] \leftarrow TOP\,[\,3\,] - OLDTOP\,[\,3\,] = 10 - 8 = 2)$ и $INC \leftarrow INC + D\,[\,j\,] = 1 + D\,[\,3\,] = 1 + 2 = 3$

$j = 4$:

$SUM \leftarrow SUM - (TOP\,[\,j\,] - BASE\,[\,j\,]) = 12 - (TOP\,[\,4\,] - BASE\,[\,4\,]) =$

$= 12 - (11 - 9) = 12 - 2 = 10$

$(TOP\,[\,j\,] > OLDTOP\,[\,j\,]) = (TOP\,[\,4\,] > OLDTOP\,[\,4\,]) = (11 > 11) =$ ЛОЖЬ, следовательно, $(D\,[\,j\,] \leftarrow 0) = (D\,[\,4\,] \leftarrow 0)$

$SUM = 10$ – суммарное количество свободного пространства в памяти

$INC = 3$ – суммарный рост размеров стеков с момента последнего распределения

***G*3.** [Память полна?]

(*SUM* < 0) = (10 < 0) = ЛОЖЬ, следовательно, работу со стеками можно продолжить

***G*4.** [Вычисление коэффициентов распределения.]

$\alpha \leftarrow 0.1 \times SUM / n = 0.1 \times 10 / 4 = 0{,}25$

$\beta \leftarrow 0.9 \times SUM / INC = 0.9 \times 10 / 3 = 3{,}0$

(Здесь α – конечная десятичная дробь, а β – целое число. В общем случае α и β дроби, и вычислять их следует с достаточной точностью, например, до пяти значащих цифр после запятой).

***G*5.** [Вычисление новых базовых адресов.]

$NEWBASE[1] \leftarrow BASE[1] = -1,\ \sigma \leftarrow 0$

$j = 2$:

$\tau \leftarrow \sigma + \alpha + D[j-1]\,\beta = 0 + 0{,}25 + D[1] \times 3 = 0{,}25 + 0 \times 3 = 0{,}25$

($NEWBASE[j] \leftarrow NEWBASE[j-1] + TOP[j-1] - BASE[j-1] + \lfloor\tau\rfloor - \lfloor\sigma\rfloor$) = ($NEWBASE[2] \leftarrow NEWBASE[1] + TOP[1] - BASE[1] + \lfloor 0{,}25\rfloor - \lfloor 0\rfloor = -1 + 1 - (-1) + 0 - 0 = 1$), $\sigma \leftarrow \tau = 0{,}25$

$j = 3$:

$\tau \leftarrow \sigma + \alpha + D[j-1]\,\beta = 0{,}25 + 0{,}25 + D[2] \times 3 = 0{,}5 + 1 \times 3 = 3{,}5$

($NEWBASE[j] \leftarrow NEWBASE[j-1] + TOP[j-1] - BASE[j-1] + \lfloor\tau\rfloor - \lfloor\sigma\rfloor$) = ($NEWBASE[3] \leftarrow NEWBASE[2] + TOP[2] - BASE[2] + \lfloor 3{,}5\rfloor - \lfloor 0{,}25\rfloor = 1 + 3 - 2 + 3 - 0 = 5$), $\sigma \leftarrow \tau = 3{,}5$

$j = 4$:

$\tau \leftarrow \sigma + \alpha + D[j-1]\,\beta = 3{,}5 + 0{,}25 + D[3] \times 3 = 3{,}75 + 2 \times 3 = 9{,}75$

($NEWBASE[j] \leftarrow NEWBASE[j-1] + TOP[j-1] - BASE[j-1] + \lfloor\tau\rfloor - \lfloor\sigma\rfloor$) = ($NEWBASE[4] \leftarrow NEWBASE[3] + TOP[3] - BASE[3] + \lfloor 9{,}75\rfloor - \lfloor 3{,}5\rfloor = 5 + 10 - 5 + 9 - 3 = 16$), $\sigma \leftarrow \tau = 9{,}75$

***G*6.** [Переупаковка.]

($TOP[i] \leftarrow TOP[i] - 1$) = ($TOP[3] \leftarrow TOP[3] - 1 = 10 - 1 = 9$)

1_1	1_2		2_1			3_1	3_2	3_3	3_4	4_1	4_2								
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

BASE [1] = –1; BASE [2] = 2; BASE [3] = 5; BASE [4] = 9

TOP [1] = 1; TOP [2] = 3; TOP [3] = 9; TOP [4] = 11

Вызов алгоритма *R*

Перемещение последовательных таблиц

***R*1.** [Начальная установка.]

$j \leftarrow 1$

***R*2.** [Поиск начала сдвига.]

(Все стеки от 1 до 1 перемещены нужным образом.)

$j \leftarrow j + 1 = 1 + 1 = 2$

(*NEWBASE* [*j*] < *BASE* [*j*]) = (*NEWBASE* [2] < *BASE* [2]) = (1 < 2) = ИСТИНА, следовательно, перейти к шагу *R*3

***R*3.** [Сдвиг вниз.]

$\delta \leftarrow$ *BASE* [*j*] – *NEWBASE* [*j*] = *BASE* [2] – *NEWBASE* [2] = 2 – 1 = 1

Для *L* = *BASE* [*j*] + 1, *BASE* [*j*] + 2, ..., *TOP* [*j*] = *BASE* [2] + 1, *BASE* [2] + 2, ..., *TOP* [2] = 2 + 1, 2 + 2, ..., 3 = 3, 4, ..., 3 = 3 выполнить *CONTENTS* (*L* – δ) ← *CONTENTS* (*L*)

L = 3:

(*CONTENTS* (*L* – δ) ← *CONTENTS* (*L*)) = (*CONTENTS* (3 – 1) ← *CONTENTS* (3)) = (*CONTENTS* (2) ← *CONTENTS* (3))

1_1	1_2	2_1				3_1	3_2	3_3	3_4	4_1	4_2								
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

BASE [1] = –1; BASE [2] = 2; BASE [3] = 5; BASE [4] = 9

TOP [1] = 1; TOP [2] = 3; TOP [3] = 9; TOP [4] = 11

(*BASE* [*j*] ← *NEWBASE* [*j*]) = (*BASE* [2] ← *NEWBASE* [2] = 1)

(*TOP* [*j*] ← *TOP* [*j*] – δ) = (*TOP* [2] ← *TOP* [2] – 1 = 3 – 1 = 2)

1_1	1_2	2_1				3_1	3_2	3_3	3_4	4_1	4_2								
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

BASE [1] = –1; BASE [2] = 1; BASE [3] = 5; BASE [4] = 9

TOP [1] = 1; TOP [2] = 2; TOP [3] = 9; TOP [4] = 11

***R*2.** [Поиск начала сдвига.]

(Все стеки от 1 до 2 перемещены нужным образом.)

$j \leftarrow 2 + 1 = 3$

(*NEWBASE* [*j*] < *BASE* [*j*]) = (*NEWBASE* [3] < *BASE* [3]) = (5 < 5) = ЛОЖЬ, проверяем следующее условие

(*NEWBASE* [*j*] > *BASE* [*j*]) = (*NEWBASE* [3] > *BASE* [3]) = (5 > 5) = ЛОЖЬ, проверяем следующее условие

(*j* > *n*) = (3 > 4) = ЛОЖЬ, следовательно, выполняем шаг *R*2 сначала

***R*2.** [Поиск начала сдвига.]

(Все стеки от 1 до 3 перемещены нужным образом.)

$j \leftarrow j + 1 = 3 + 1 = 4$

(*NEWBASE* [*j*] < *BASE* [*j*]) = (*NEWBASE* [4] < *BASE* [4]) = (16 < 9) = ЛОЖЬ, проверяем следующее условие

(*NEWBASE* [*j*] > *BASE* [*j*]) = (*NEWBASE* [4] > *BASE* [4]) = (16 > 9) = ИСТИНА, следовательно, перейти к шагу *R*4

***R*4.** [Поиск верхней границы сдвига.]

$k \geq j = 4$

$k = 4$

(*NEWBASE* [k + 1] ≤ *BASE* [k + 1]) = (*NEWBASE* [4 + 1] ≤ *BASE* [4 + 1]) = (*NEWBASE* [5] ≤ *BASE* [5]) = (19 ≤ 19) = ИСТИНА, следовательно, выполним шаг *R*5 для $T = k, k - 1, \ldots, j = 4, 4 - 1, \ldots, 4 = 4, 3, \ldots, 4 = 4$

***R*5.** [Сдвиг вверх.]

$\delta \leftarrow$ *NEWBASE* [*T*] – *BASE* [*T*] = *NEWBASE* [4] – *BASE* [4] = 16 – 9 = 7

L = *TOP* [4], *TOP* [4] – 1, ..., *BASE* [4] + 1 = 11, 11 – 1, ..., 9 + 1 = 11, 10, ..., 10 = 11, 10

L = 11:

(*CONTENTS* (*L* + δ) ← *CONTENTS* (*L*)) = (*CONTENTS* (11 + 7) ← *CONTENTS* (11)) = (*CONTENTS* (18) ← *CONTENTS* (11))

1_1	1_2	2_1				3_1	3_2	3_3	3_4	4_1								4_2	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

BASE [1] = –1; BASE [2] = 1; BASE [3] = 5; BASE [4] = 9

TOP [1] = 1; TOP [2] = 2; TOP [3] = 9; TOP [4] = 11

L = 10:

(*CONTENTS* (L + δ) ← *CONTENTS* (L)) = (*CONTENTS* (10 + 7) ← *CONTENTS* (10)) = (*CONTENTS* (17) ← *CONTENTS* (10))

1_1	1_2	2_1				3_1	3_2	3_3	3_4								4_1	4_2	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

BASE [1] = –1; BASE [2] = 1; BASE [3] = 5; BASE [4] = 9

TOP [1] = 1; TOP [2] = 2; TOP [3] = 9; TOP [4] = 11

(*BASE* [T] ← *NEWBASE* [T]) = (*BASE* [4] ← *NEWBASE* [4] = 16)

(*TOP* [T] ← *TOP* [T] + δ) = (*TOP* [4] ← *TOP* [4] + 7 = 11 + 7 = 18)

1_1	1_2	2_1				3_1	3_2	3_3	3_4								4_1	4_2	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

BASE [1] = –1; BASE [2] = 1; BASE [3] = 5; BASE [4] = 16

TOP [1] = 1; TOP [2] = 2; TOP [3] = 9; TOP [4] = 18

***R*4.** [Поиск верхней границы сдвига.]

j ← k = 4

***R*2.** [Поиск начала сдвига.]

(Все стеки от 1 до 4 перемещены нужным образом.)

j ← j + 1 = 4 + 1 = 5

(*NEWBASE* [j] < *BASE* [j]) = (*NEWBASE* [5] < *BASE* [5]) = (19 < 19) = ЛОЖЬ, проверяем следующее условие

(*NEWBASE* [j] > *BASE* [j]) = (*NEWBASE* [5] > *BASE* [5]) = (19 > 19) = ЛОЖЬ, проверяем следующее условие

(j > n) = (5 > 4) = ИСТИНА, следовательно, окончить алгоритм *R*

***G*6.** [Переупаковка.]

Восстановить значение *TOP*[i]

(*TOP* [i] ← *TOP* [i] + 1) = (*TOP* [3] ← *TOP* [3] + 1 = 9 + 1 = 10)

1_1	1_2	2_1				3_1	3_2	3_3	3_4								4_1	4_2	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

BASE [1] = –1; BASE [2] = 1; BASE [3] = 5; BASE [4] = 16

TOP [1] = 1; TOP [2] = 2; TOP [3] = 10; TOP [4] = 18

(Заметим, что по указателю TOP [3] = 10 свободная ячейка готова для записи в третий стек нового элемента, который ранее не был включен в этот стек в связи с зафиксированной при выполнении операции (8) ситуацией ПЕРЕПОЛНЕНИЕ.)

Ответ:

1_1	1_2	2_1				3_1	3_2	3_3	3_4								4_1	4_2	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

BASE [1] = –1; BASE [2] = 1; BASE [3] = 5; BASE [4] = 16

TOP [1] = 1; TOP [2] = 2; TOP [3] = 10; TOP [4] = 18

Лабораторная работа № 3

ТОПОЛОГИЧЕСКАЯ СОРТИРОВКА

Цель. Знакомство с понятием связанного распределения памяти при хранении линейных списков и алгоритмом топологической сортировки частично упорядоченного множества.

Задание. Выполнить по алгоритму *T* топологическую сортировку элементов 1, 2, 3, …, 9 для заданных отношений между ними. Отношения взять из табл. 3.

Основные положения

Вместо хранения списка в последовательных ячейках памяти, можно использовать более гибкую схему, в которой каждый узел содержит связь со следующим узлом списка. Ниже представлены схемы последовательного и связанного распределения памяти при хранении линейного списка, состоящего из пяти элементов.

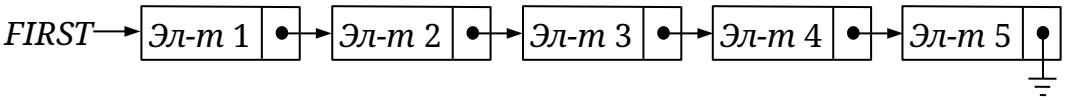
Последовательное распределение		Связанное распределение	
Адрес	Содержимое	Адре	Содержимое

		c		
$L_0 + c$:	Элемент 1	A:	Элемент 1	B
$L_0 + 2c$:	Элемент 2	B:	Элемент 2	C
$L_0 + 3c$:	Элемент 3	C:	Элемент 3	D
$L_0 + 4c$:	Элемент 4	D:	Элемент 4	E
$L_0 + 5c$:	Элемент 5	E:	Элемент 5	Λ

Здесь A, B, C, D и E – адреса произвольных ячеек в памяти, Λ – пустая связь.

В случае связанного распределения памяти в программе имеется переменная связи или константа, имеющая значение A. Отправляясь от узла по адресу A, можно найти все другие элементы списка.

Связи часто изображаются просто стрелками:



(9)

Здесь *FIRST* – указатель на первый узел в списке.

Таблица 3

Вариант	Отношения									
1	2<6	4<1	2<3	9<7	7<3	5<4	3<5	1<8	7<1	6<4
2	4<6	6<1	3<4	9<3	5<3	3<2	9<7	2<4	1<7	8<6
3	6<1	2<3	1<7	3<4	7<9	5<2	1<2	7<5	9<3	4<8
4	8<4	4<3	3<7	6<3	6<5	7<5	1<9	1<2	5<1	7<1
5	7<2	7<6	3<1	8<3	9<8	8<1	2<4	6<2	3<7	5<6
6	6<4	2<6	9<1	1<8	4<3	5<7	4<5	2<7	8<6	9<2
7	2<6	3<5	9<2	8<1	7<8	4<3	3<9	5<6	6<1	6<7
8	6<7	2<5	8<9	8<3	5<8	3<4	1<6	3<6	9<3	2<9
9	8<9	4<2	6<5	9<7	3<8	1<4	2<9	2<8	5<4	2<3
10	1<9	6<3	8<6	3<1	2<8	9<5	9<7	7<4	8<9	8<3
11	3<6	8<1	3<7	4<5	5<7	9<8	7<9	1<2	5<1	6<8
12	1<3	3<2	5<1	8<5	7<5	5<6	8<4	6<1	2<4	9<3
13	4<7	5<3	7<6	3<2	6<1	9<5	7<5	6<9	1<3	2<8
14	6<2	2<3	3<7	1<3	1<9	7<9	8<4	8<5	9<8	7<8
15	5<1	5<8	2<7	4<2	6<4	4<7	1<3	8<1	2<5	9<8
16	2<6	8<2	3<7	7<1	6<5	4<9	6<4	8<9	1<2	3<8
17	8<9	1<4	6<8	7<5	3<7	2<1	1<6	4<9	9<5	9<3
18	7<9	5<6	1<3	1<4	6<1	4<2	8<7	4<7	3<4	5<3
19	5<2	1<7	6<3	2<4	8<5	9<1	7<2	7<5	3<1	7<8
20	7<2	3<4	1<3	4<7	6<1	2<5	2<8	8<9	1<2	1<4
21	5<1	4<2	5<7	8<3	3<7	6<4	7<6	2<9	3<2	1<4
22	8<1	1<2	5<8	6<5	9<5	5<3	6<7	3<8	2<7	4<1
23	5<6	2<9	6<7	9<3	7<1	4<2	6<2	7<4	1<9	3<8
24	8<2	2<7	7<6	5<7	5<4	6<4	9<1	9<3	4<9	6<9
25	4<7	4<5	6<8	9<6	3<9	9<8	7<2	5<7	6<4	1<5
26	9<6	4<9	5<2	2<1	6<8	3<7	6<3	4<7	1<9	5<4
27	3<1	8<2	6<3	5<7	9<5	4<8	8<6	2<1	1<7	1<9

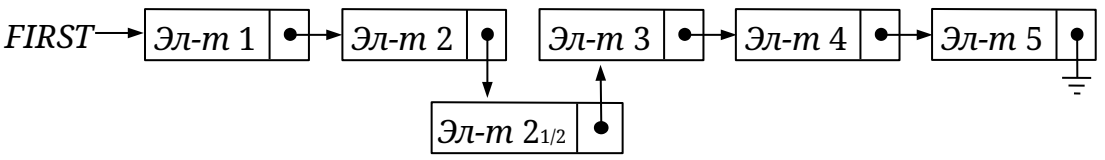
28	6<9	3<1	5<7	5<8	1<5	8<2	4<6	8<6	7<8	3<7
29	5<2	4<3	1<6	2<9	7<5	8<4	3<2	3<5	6<4	3<7
30	9<3	1<6	8<1	6<9	7<8	3<4	3<2	2<5	8<3	8<6

Сопоставим последовательное и связанное распределение:

1. Связанное распределение требует дополнительного пространства в памяти для хранения связей. Однако при использовании связанной памяти часто возникает неявный выигрыш в памяти за счёт совмещения общих частей таблиц. Кроме того, связанное распределение будет более эффективным при плотно загруженной памяти.

2. Легко исключить элемент, находящийся внутри связанного списка. Для этого необходимо только изменить связь в предшествующем элементе. При последовательном же распределении такое исключение обычно требует перемещения значительной части списка вверх на другие места памяти.

3. При связанном распределении легко включить элемент в список. Например, для включения элемента 21/2 в (9) необходимо изменить лишь две связи:



(10)

Такая операция заняла бы значительное время при работе с длинной последовательной таблицей.

4. При связанном распределении памяти значительно медленнее, чем при последовательном, выполняются обращения к произвольным частям списка. Однако в большинстве приложений продвижение по списку

осуществляется последовательно, а не произвольным образом. Кроме того, можно создать дополнительные переменные связи, указывающие на соответствующие места в списке.

5. При использовании схемы со связями упрощается задача объединения двух списков или разбиения списка на части.

6. Схема со связями годится для структур более сложных, чем линейные списки.

Рассмотрим организацию включения и исключения узлов для связанного списка.

Пусть каждый узел списка имеет два поля:

INFO	*LINK*	(11)

где *INFO* – полезная информация, *LINK* – связь со следующим узлом.

Использование связанного распределения предполагает поиск пустого пространства для нового узла. Для этого существует специальный список, назовём его *AVAIL*, который содержит все узлы, которые в данный момент не используются, и ничем не отличается от любых других списков. Переменная связи *AVAIL* ссылается на верхний элемент этого списка.

Для выделения узла из списка *AVAIL* с целью его использования необходимо поступить следующим образом:

$$X \leftarrow AVAIL, \quad AVAIL \leftarrow LINK\,(AVAIL). \tag{12}$$

В результате из стека *AVAIL* исключается верхний узел и переменная связи *X* становится указателем на только что выделенный узел. Операцию (12) для краткости будем обозначать $X \Leftarrow AVAIL$.

Если узел по адресу *X* далее не нужен, то его можно вернуть в список свободного пространства:

$$LINK\,(X) \leftarrow AVAIL, \quad AVAIL \leftarrow X. \tag{13}$$

Эту операцию будем обозначать $AVAIL \Leftarrow X$.

Для построения списка *AVAIL* в начале работы необходимо (*A*) связать вместе все узлы, которые будут использоваться, (*B*) занести в *AVAIL* адрес первого из этих узлов и (*C*) сделать связь последнего узла пустой. Множество всех узлов, которые могут быть использованы, называют *пулом памяти.*

С учётом проверки на переполнение операцию $X \Leftarrow AVAIL$ следует выполнять так:

Если $AVAIL = \Lambda$, то ПЕРЕПОЛНЕНИЕ;

в противном случае $X \leftarrow AVAIL$, $AVAIL \leftarrow LINK(AVAIL)$.

(14)

Возникшее ПЕРЕПОЛНЕНИЕ означает, что необходимо прекратить выполнение программы.

Рассмотрим несколько наиболее распространённых операций со связанными списками, если работа ведётся со стеками и очередями.

Пусть имеется стек, на верхний элемент которого указывает переменная *T.* Для включения информации из *Y* в стек используется вспомогательный указатель *P*:

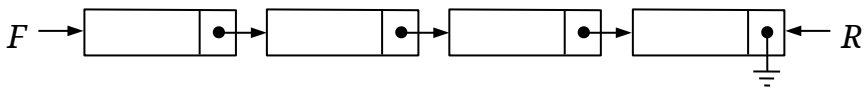
$$P \Leftarrow AVAIL,\quad INFO(P) \leftarrow Y,\quad LINK(P) \leftarrow T,\quad T \leftarrow P. \qquad (15)$$

Для считывания в *Y* информации из стека следует:

Если $T = \Lambda$, то НЕХВАТКА;

иначе $P \leftarrow T,\ T \leftarrow LINK(P),\quad Y \leftarrow INFO(P),\quad AVAIL \Leftarrow P.$ (16)

Связанное распределение особенно удобно в применении к очередям. При этом связи должны быть направлены от начального узла к последнему:



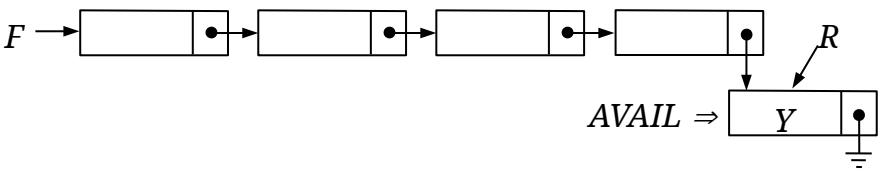
Здесь *F*, *R* – указатели на начало и конец очереди соответственно.

Включение информации из *Y* в конец очереди выполняется следующим образом:

$P \Leftarrow AVAIL$, $INFO(P) \leftarrow Y$, $LINK(P) \leftarrow \Lambda$.

Если $F = \Lambda$, то $F \leftarrow P$, иначе $LINK(R) \leftarrow P$. $R \leftarrow P$. (18)

Если (17) – ситуация перед включением, то после включения в конец очереди схема будет иметь следующий вид:



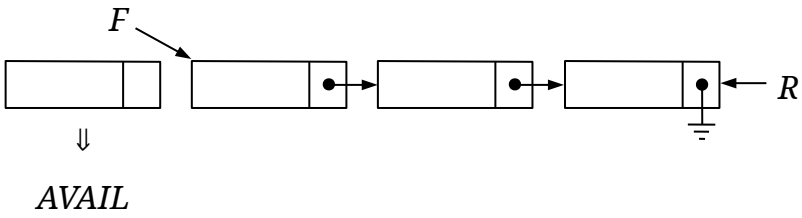
(19)

Исключение информации из начала очереди выполняется следующим образом:

Если $F = \Lambda$, то НЕХВАТКА, иначе $P \leftarrow F$, $F \leftarrow LINK(P)$,

$Y \leftarrow INFO(P)$, $AVAIL \Leftarrow P$. (20)

Если схема (17) иллюстрирует ситуацию перед исключением, то в результате этой операции она преобразуется в следующую схему:



(21)

Перейдём далее к изучению практического примера *топологической сортировки*, которая оказывается полезной всякий раз, когда мы сталкиваемся с упорядочением.

Пусть имеется следующее частично упорядоченное множество, заданное графом:

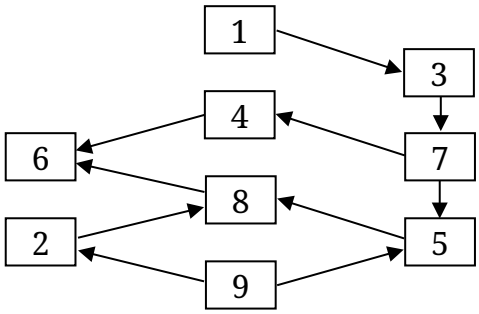


Рис. 4. Исходное частично упорядоченное множество

Здесь, например, работа 1 должна быть выполнена раньше работы 3, работа 3 – раньше работы 7, работа 7 – раньше работ 4, 5 и т.д. (рис. 4).

Требуется расположить элементы множества в линейную последовательность так, чтобы все дуги были ориентированы слева направо (рис. 5):

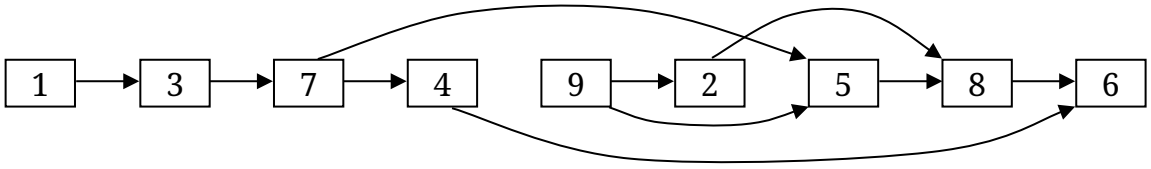


Рис. 5. Частично упорядоченное множество после топологической сортировки

Существует простой способ топологической сортировки вручную. Вначале берётся элемент, которому не предшествует никакой другой элемент. Он помещается первым в выходную последовательность и удаляется из исходного частично упорядоченного множества вместе с инцидентными ему дугами. Затем в графе выбирается ещё один элемент без предшественников, помещается вторым в выходную последовательность и исключается из него. Описанный процесс продолжается до тех пор, пока исходный граф не сократится до пустого графа. Заметим, что линейное размещение всех элементов невозможно, если в графе имеется хотя бы один контур.

Рассмотрим далее реализацию топологической сортировки на машине. При этом будем полагать, что сортируемые элементы пронумерованы числами от 1 до n в любом порядке. Информация вводится парами чисел вида (j, k) или $j < k$. Это означает, что j-й элемент множества предшествует k-му элементу.

Входной информацией применительно к нашему примеру могут быть такие пары:

$$9 < 2,\quad 3 < 7,\quad 7 < 5,\quad 5 < 8,$$
$$8 < 6,\quad 4 < 6,\quad 1 < 3, \qquad (22)$$
$$7 < 4,\quad 9 < 5,\quad 2 < 8.$$

В алгоритме будет использована последовательная таблица $X[1]$, $X[2]$, …, $X[n]$, любой k-й узел которой имеет формат:

COUNT[*k*]	*TOP*[*k*]

где *COUNT*[*k*] – *количество непосредственных предшественников* объекта k (количество встретившихся среди входной информации пар $j < k$), *TOP*[*k*] – связь с началом *списка непосредственных преемников* объекта k. Список непосредственных преемников содержит элементы формата:

SUC	*NEXT*

где *SUC* – непосредственный преемник объекта k, *NEXT* – связь со следующим элементом этого списка.

Для входной информации (22) будем иметь следующую модель памяти:

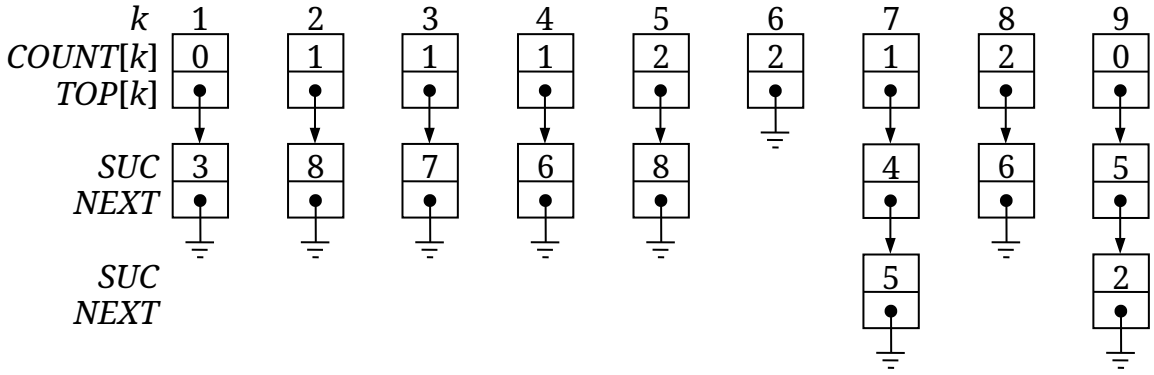


Рис. 6. Машинное представление для отношений (22)

Алгоритм будет заключаться в выводе узлов, у которых поле *COUNT* равно нулю, и последующем уменьшении на один поля *COUNT* у всех преемников этих узлов. Для избежания поиска узлов с нулевым значением поля *COUNT* организуется очередь из этих узлов. Связи для очереди размещаются в поле *COUNT*, которое к данному моменту уже выполнило

свою предыдущую функцию. Для большей наглядности в алгоритме используется обозначение $QLINK[k]$ вместо $COUNT[k]$, когда это поле уже не используется как счётчик.

Алгоритм *T*. (*Топологическая сортировка.*) Здесь вводятся пары отношений $j < k$ ($1 \le j \le n$, $1 \le k \le n$, $j \neq k$), говорящие, что объект j предшествует объекту k в некотором отношении упорядочения. Результатом является множество всех объектов, расположенных в линейном порядке. Используются внутренние таблицы $QLINK[0]$, $COUNT[1] = QLINK[1]$, $COUNT[2] = QLINK[2]$, ..., $COUNT[n] = QLINK[n]$; $TOP[1]$, $TOP[2]$, ..., $TOP[n]$; пул памяти с одним узлом (с полями *SUC* и *NEXT*) для каждой вводимой пары; P – переменная связи для ссылки на узлы в пуле памяти; F и R – целочисленные переменные, используемые для ссылок в начало и конец очереди, связи которой находятся в таблице *QLINK*; N – переменная, указывающая число невыведенных объектов.

***T*1.** [Начальная установка.] Ввести значение n. Для $k = 1, 2, \ldots, n$ $COUNT[k] \leftarrow 0$, $TOP[k] \leftarrow \Lambda$. $N \leftarrow n$.

***T*2.** [Следующее отношение.] Ввести следующее отношение $j < k$; если же входная информация исчерпана, то перейти к шагу *T*4.

***T*3.** [Регистрация отношения.] Значение $COUNT[k]$ увеличить на единицу. $P \Leftarrow AVAIL$, $SUC(P) \leftarrow k$, $NEXT(P) \leftarrow TOP[j]$, $TOP[j] \leftarrow P$. Перейти к шагу *T*2.

***T*4.** [Поиск нулей.] (К этому моменту завершена фаза ввода и входная информация преобразована в машинное представление, показанное на рис. 6. Производим теперь начальную установку очереди вывода, которая связывается по полю *QLINK*.) $R \leftarrow 0$, $QLINK[0] \leftarrow 0$. Для $k = 1, 2, \ldots, n$ просмотреть $COUNT[k]$ и, если он нулевой, $QLINK[R] \leftarrow k$, $R \leftarrow k$. После выполнения этих действий для всех k реализовать $F \leftarrow QLINK[0]$ (в нём окажется первое встреченное значение k, для которого $COUNT[k]$ был нулевым).

***T*5.** [Вывод из начала очереди.] Вывести значение F. Если F равно 0, то перейти к шагу $T8$; в противном случае $N \leftarrow N - 1$, $P \leftarrow TOP[F]$. (Поскольку таблицы *QLINK* и *COUNT* перекрываются, мы имеем $QLINK[R] = 0$; следовательно, условие $F = 0$ выполняется, когда очередь пуста.)

***T*6.** [Стирание отношения.] Если $P = \Lambda$, то перейти к шагу $T7$; в противном случае уменьшить $COUNT[SUC(P)]$ на единицу и, если при этом он стал нулевым, $QLINK(R) \leftarrow SUC(P)$ и $R \leftarrow SUC(P)$. Р $\leftarrow NEXT(P)$ и повторить данный шаг. (Мы исключаем из системы все отношения вида $F < k$ для некоторого k и помещаем новые узлы в очередь, когда все их предшественники были выведены.)

***T*7.** [Исключение из очереди.] $F \leftarrow QLINK[F]$ и вернуться к шагу $T5$.

***T*8.** [Окончание процесса.] Конец алгоритма. Если $N = 0$, то выведены все номера элементов в требуемом топологическом порядке и вслед за ними нуль. В противном случае N невыведенных номеров содержат цикл. ■

В этом алгоритме удачно сочетаются методы последовательной и связанной памяти. Для главной таблицы $X[1], X[2], \ldots, X[n]$ используется последовательная память, поскольку на шаге $T3$ алгоритм ссылается на «произвольные» части этой таблицы. Связанная память используется для таблиц «непосредственных преемников», поскольку элементы этих таблиц вводятся в произвольном порядке. Очередь узлов, ожидающих вывода, хранится внутри последовательной таблицы, причём узлы связываются в соответствии с порядком их вывода. Для связи вместо адреса используется индекс таблицы, т.е. когда начальным узлом очереди является узел $X[k]$, то мы имеем F равно k, а не F равно $LOC(X[k])$. Операции над очередью, которые использованы на шагах $T4$, $T6$ и $T7$, не идентичны операциям (18) и (20), поскольку в этой системе мы пользуемся специальными свойствами очереди и нет необходимости создавать узлы или возвращать их в свободное пространство.

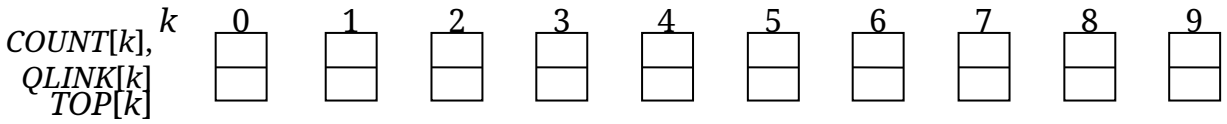
Время выполнения алгоритма оценивается значением выражения $c_1m + c_2n$, где m – количество вводимых отношений, n – количество объектов; c_1 и c_2 – константы.

Пример выполнения задания

Задание. Выполнить по алгоритму T топологическую сортировку элементов 1, 2, 3, …, 9 для заданных отношений между ними: 9 < 2, 3 < 7, 7 < 5, 5 < 8, 8 < 6, 4 < 6, 1 < 3, 7 < 4, 9 < 5, 2 < 8.

***T*1.** [Начальная установка.] Ввести значение $n = 9$.

Внутренние таблицы (*QLINK*[0], *COUNT*[1] = *QLINK*[1], *COUNT*[2] = *QLINK*[2], …, *COUNT*[*n*] = *QLINK*[*n*]) = (*QLINK*[0], *COUNT*[1] = *QLINK*[1], *COUNT*[2] = *QLINK*[2], …, *COUNT*[9] = *QLINK*[9]):



Для $k = (1, 2, …, n) = (1, 2, …, 9)$ *COUNT*[*k*] ← 0, *TOP*[*k*] ← Λ:

COUNT[1] ← 0, *TOP*[1] ← Λ

COUNT[2] ← 0, *TOP*[2] ← Λ

COUNT[3] ← 0, *TOP*[3] ← Λ

COUNT[4] ← 0, *TOP*[4] ← Λ

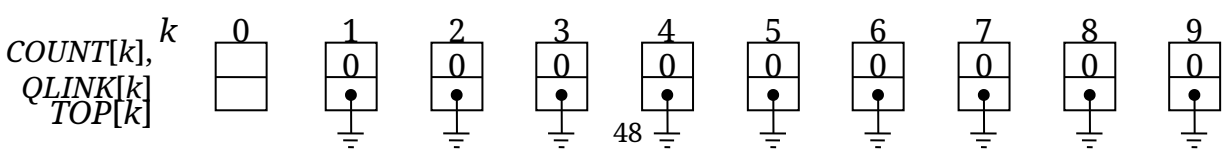
COUNT[5] ← 0, *TOP*[5] ← Λ

COUNT[6] ← 0, *TOP*[6] ← Λ

COUNT[7] ← 0, *TOP*[7] ← Λ

COUNT[8] ← 0, *TOP*[8] ← Λ

COUNT[9] ← 0, *TOP*[9] ← Λ

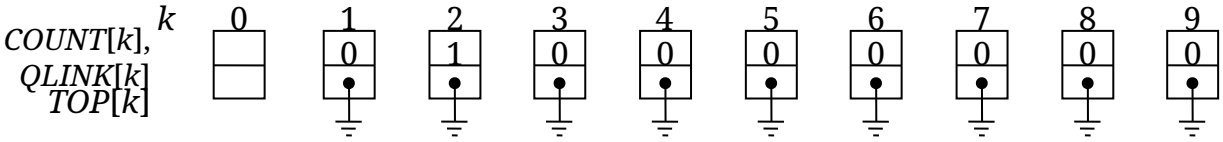


$N \leftarrow n = 9$.

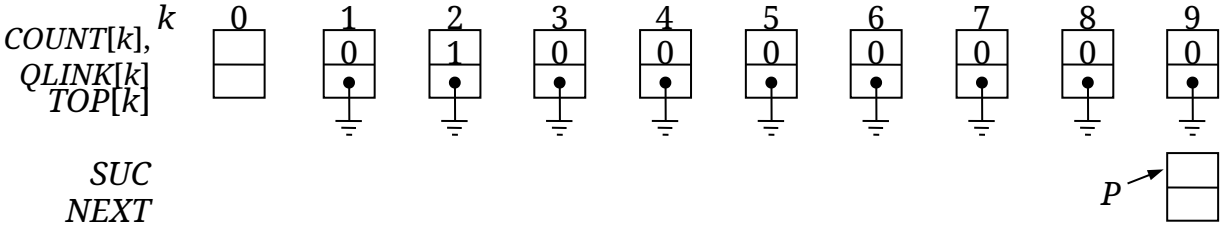
***T*2.** [Следующее отношение.] Ввести следующее отношение $(j < k) = (9 < 2)$, следовательно, $j = 9$, $k = 2$.

***T*3.** [Регистрация отношения.]

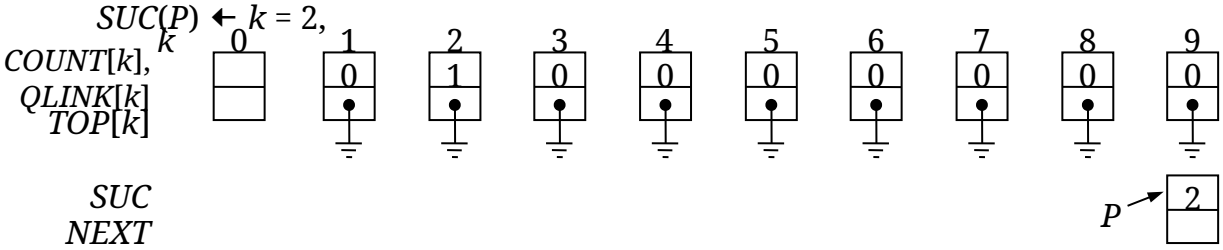
$(COUNT[k] = COUNT[k] + 1) = (COUNT[2] = COUNT[2] + 1) = 0 + 1 = 1$.



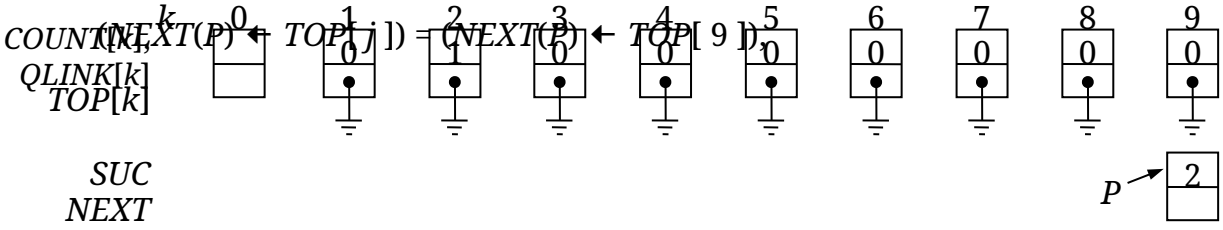
$P \Leftarrow AVAIL$,



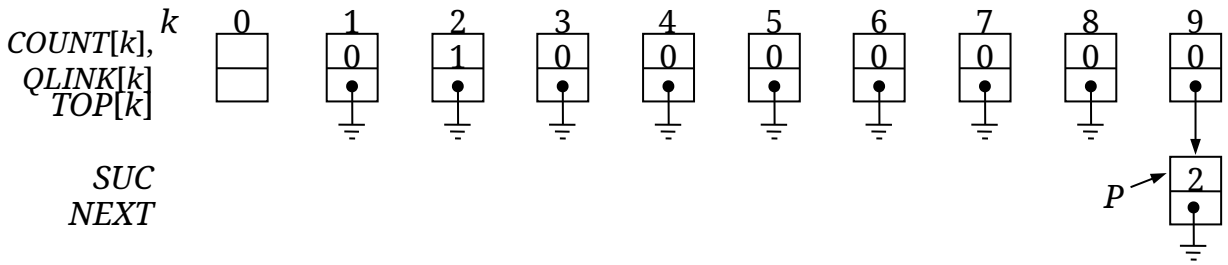
$SUC(P) \leftarrow k = 2$,



$(NEXT(P) \leftarrow TOP[\,j\,]) = (NEXT(P) \leftarrow TOP[\,9\,])$,



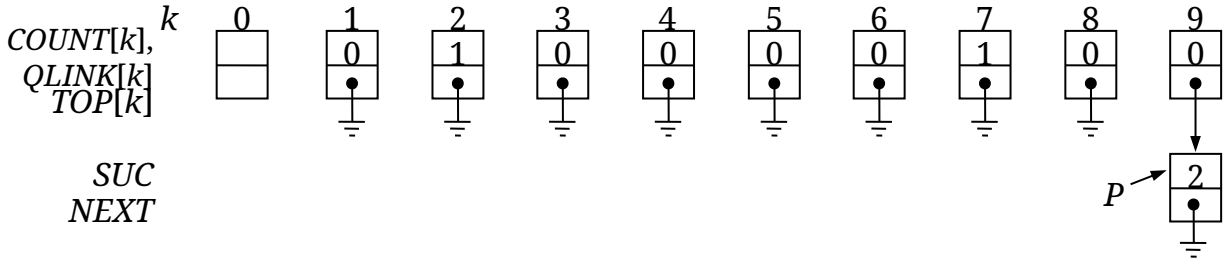
(*TOP*[*j*] ← *P*) = (*TOP*[9] ← *P*).



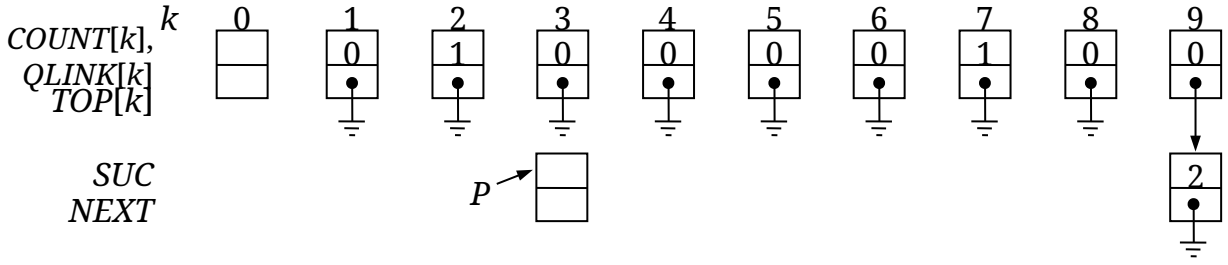
***T*2.** [Следующее отношение.] Ввести следующее отношение (*j* < *k*) = (3 < 7), следовательно, *j* = 3, *k* = 7.

***T*3.** [Регистрация отношения.]

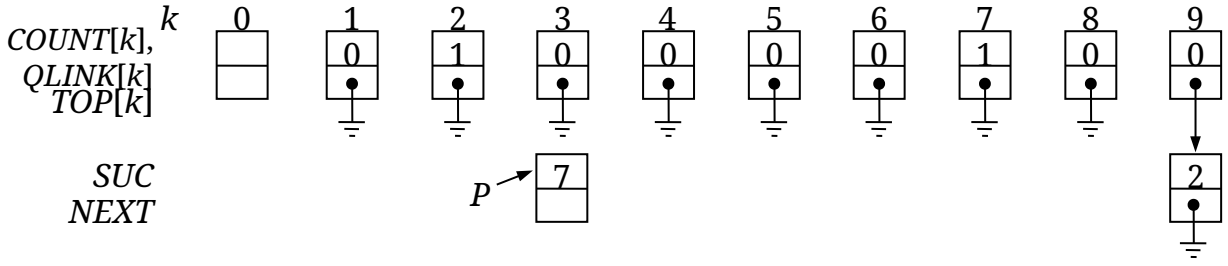
(*COUNT*[*k*] = *COUNT*[*k*] + 1) = (*COUNT*[7] = *COUNT*[7] + 1) = 0 + 1 = 1.



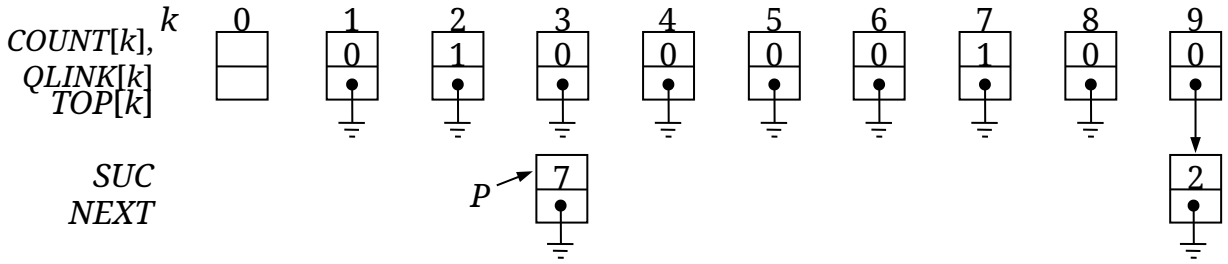
P ⇐ *AVAIL*,



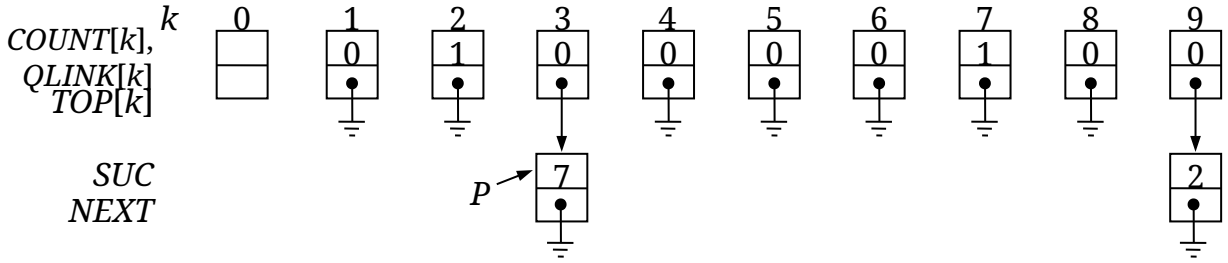
SUC(*P*) ← *k* = 7,



(*NEXT*(*P*) ← *TOP*[*j*]) = (*NEXT*(*P*) ← *TOP*[3]),



(*TOP*[*j*] ← *P*) = (*TOP*[3] ← *P*).



***T*2.** [Следующее отношение.] Ввести следующее отношение (*j* < *k*) = (7 < 5), следовательно, *j* = 7, *k* = 5.

***T*3.** [Регистрация отношения.]

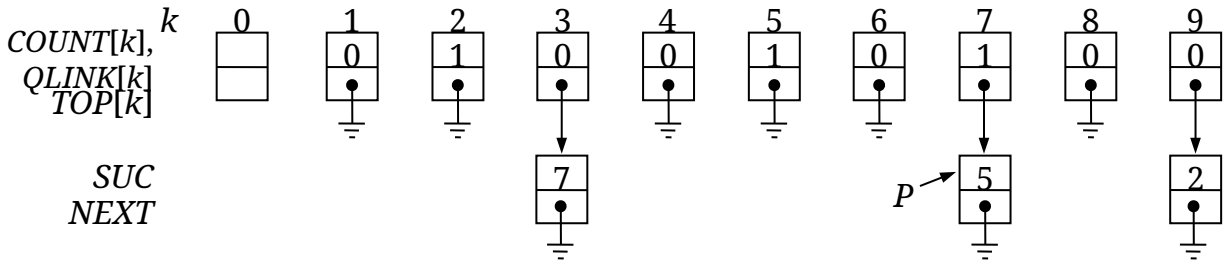
(*COUNT*[*k*] = *COUNT*[*k*] + 1) = (*COUNT*[5] = *COUNT*[5] + 1) = 0 + 1 = 1.

P ⇐ *AVAIL*,

SUC(*P*) ← *k* = 5,

(*NEXT*(*P*) ← *TOP*[*j*]) = (*NEXT*(*P*) ← *TOP*[7]),

(*TOP*[*j*] ← *P*) = (*TOP*[7] ← *P*).



***T*2.** [Следующее отношение.] Ввести следующее отношение ($j < k$) = (5 < 8), следовательно, $j = 5$, $k = 8$.

***T*3.** [Регистрация отношения.]

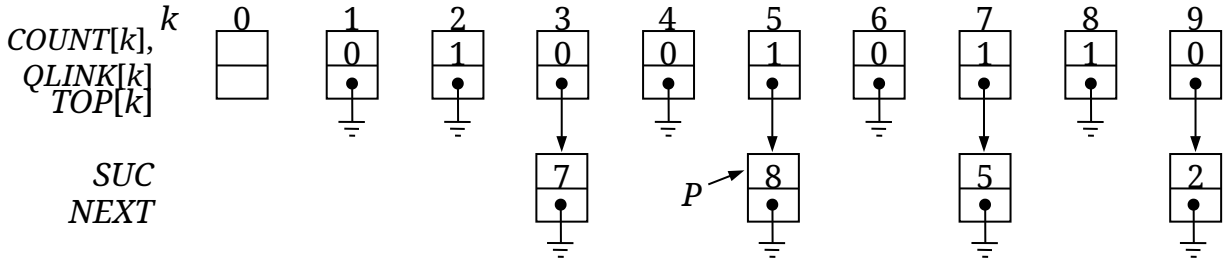
($COUNT[k] = COUNT[k] + 1$) = ($COUNT[8] = COUNT[8] + 1$) = 0 + 1 = 1.

$P \Leftarrow AVAIL$,

$SUC(P) \leftarrow k = 8$,

($NEXT(P) \leftarrow TOP[\, j\,]$) = ($NEXT(P) \leftarrow TOP[\, 5\,]$),

($TOP[\, j\,] \leftarrow P$) = ($TOP[\, 5\,] \leftarrow P$).



***T*2.** [Следующее отношение.] Ввести следующее отношение ($j < k$) = (8 < 6), следовательно, $j = 8$, $k = 6$.

***T*3.** [Регистрация отношения.]

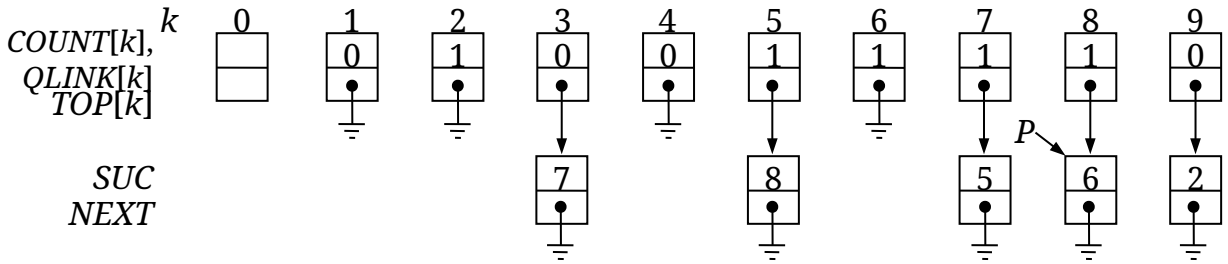
($COUNT[k] = COUNT[k] + 1$) = ($COUNT[6] = COUNT[6] + 1$) = 0 + 1 = 1.

$P \Leftarrow AVAIL$,

$SUC(P) \leftarrow k = 6$,

($NEXT(P) \leftarrow TOP[\, j\,]$) = ($NEXT(P) \leftarrow TOP[\, 8\,]$),

($TOP[\, j\,] \leftarrow P$) = ($TOP[\, 8\,] \leftarrow P$).



***T*2.** [Следующее отношение.] Ввести следующее отношение ($j < k$) = (4 < 6), следовательно, $j = 4$, $k = 6$.

***T*3.** [Регистрация отношения.]

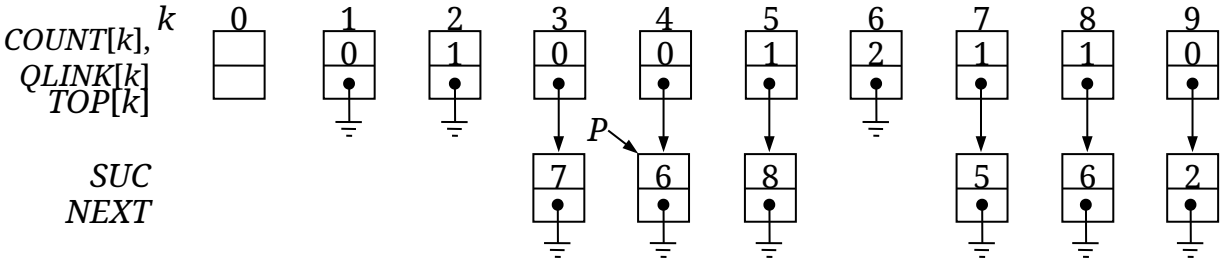
(*COUNT*[*k*] = *COUNT*[*k*] + 1) = (*COUNT*[6] = *COUNT*[6] + 1) = 1 + 1 = 2.

P ⇐ *AVAIL*,

SUC(*P*) ← *k* = 6,

(*NEXT*(*P*) ← *TOP*[*j*]) = (*NEXT*(*P*) ← *TOP*[4]),

(*TOP*[*j*] ← *P*) = (*TOP*[4] ← *P*).



***T*2.** [Следующее отношение.] Ввести следующее отношение (*j* < *k*) = (1 < 3), следовательно, *j* = 1, *k* = 3.

***T*3.** [Регистрация отношения.]

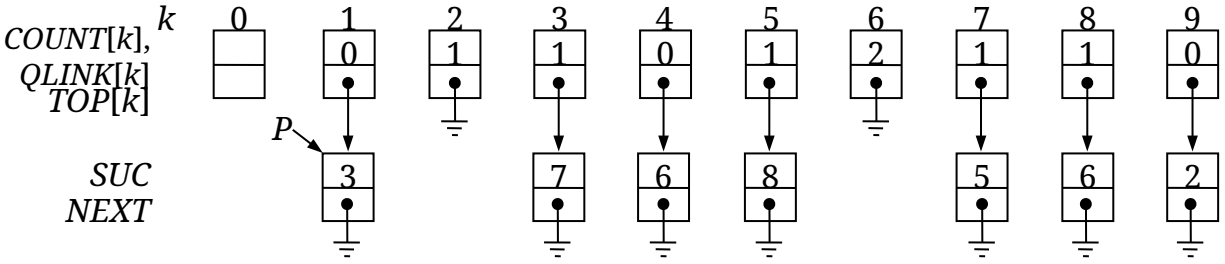
(*COUNT*[*k*] = *COUNT*[*k*] + 1) = (*COUNT*[3] = *COUNT*[3] + 1) = 0 + 1 = 1.

P ⇐ *AVAIL*,

SUC(*P*) ← *k* = 3,

(*NEXT*(*P*) ← *TOP*[*j*]) = (*NEXT*(*P*) ← *TOP*[1]),

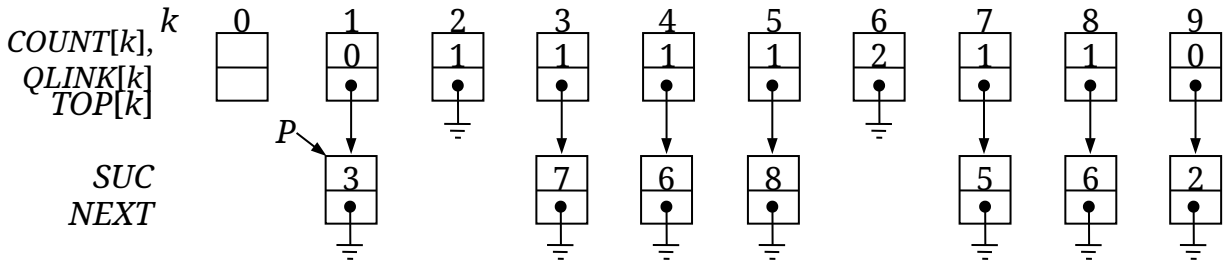
(*TOP*[*j*] ← *P*) = (*TOP*[1] ← *P*).



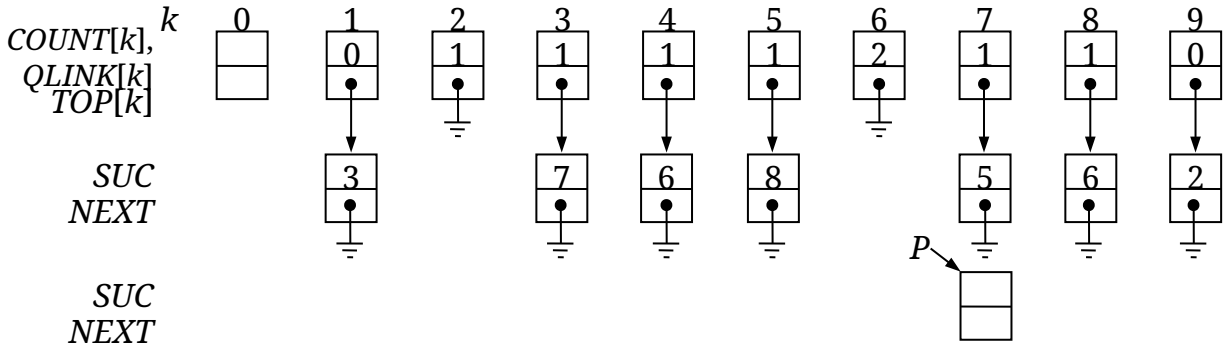
***T*2.** [Следующее отношение.] Ввести следующее отношение (*j* < *k*) = (7 < 4), следовательно, *j* = 7, *k* = 4.

***T*3.** [Регистрация отношения.]

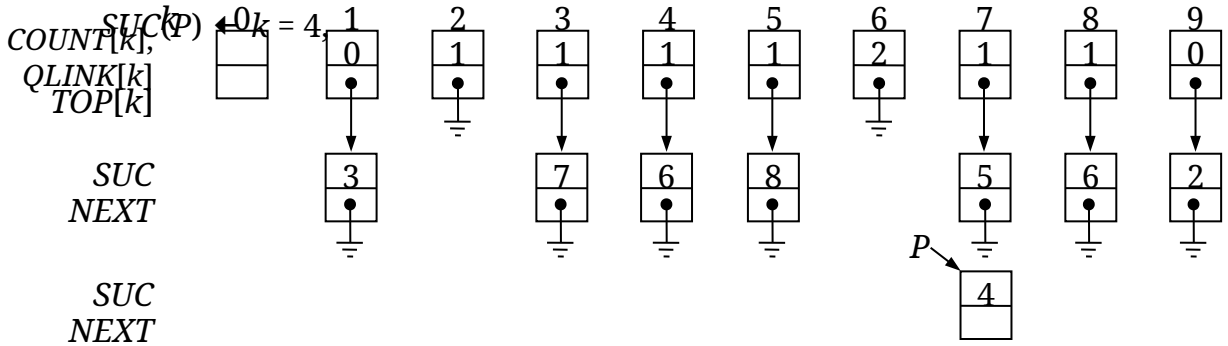
$(COUNT[k] = COUNT[k] + 1) = (COUNT[4] = COUNT[4] + 1) = 0 + 1 = 1.$



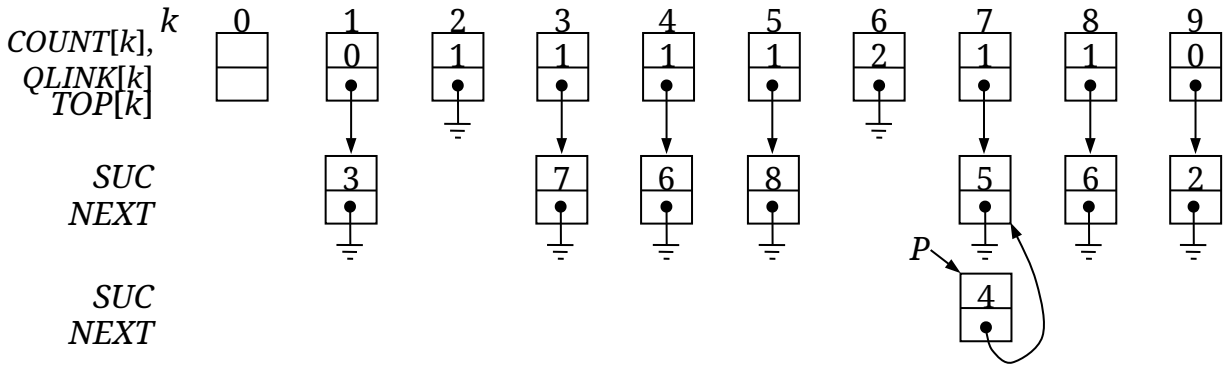
$P \Leftarrow AVAIL,$



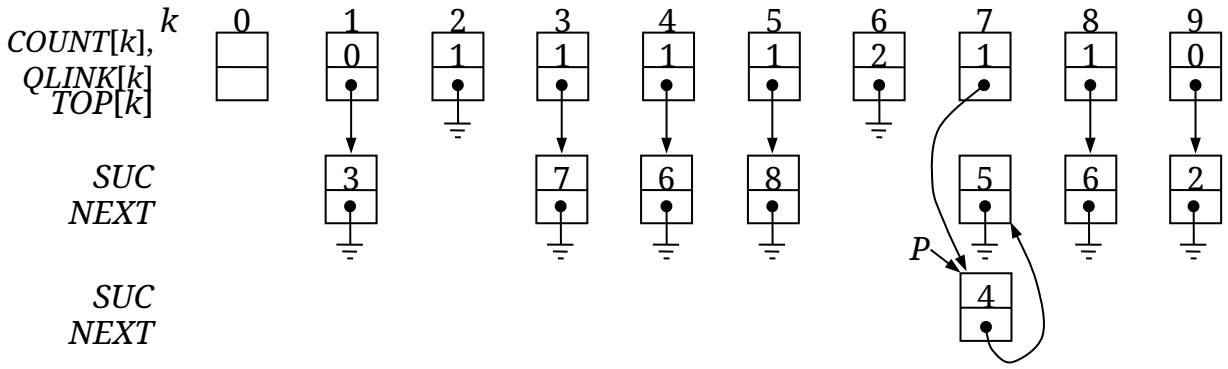
$SUC(P) \leftarrow 0, k = 4,$



$(NEXT(P) \leftarrow TOP[\ j\ ]) = (NEXT(P) \leftarrow TOP[\ 7\ ]),$



(*TOP*[*j*] ← *P*) = (*TOP*[7] ← *P*).



***T*2.** [Следующее отношение.] Ввести следующее отношение (*j* < *k*) = (9 < 5), следовательно, *j* = 9, *k* = 5.

***T*3.** [Регистрация отношения.]

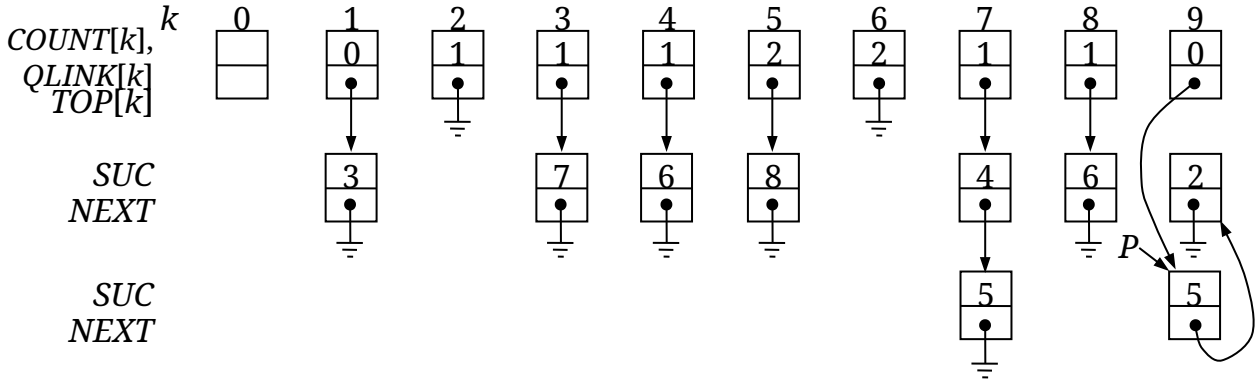
(*COUNT*[*k*] = *COUNT*[*k*] + 1) = (*COUNT*[5] = *COUNT*[5] + 1) = 1 + 1 = 2.

P ⇐ *AVAIL*,

SUC(*P*) ← *k* = 5,

(*NEXT*(*P*) ← *TOP*[*j*]) = (*NEXT*(*P*) ← *TOP*[9]),

(*TOP*[*j*] ← *P*) = (*TOP*[9] ← *P*).



***T*2.** [Следующее отношение.] Ввести следующее отношение (*j* < *k*) = (2 < 8), следовательно, *j* = 2, *k* = 8.

***T*3.** [Регистрация отношения.]

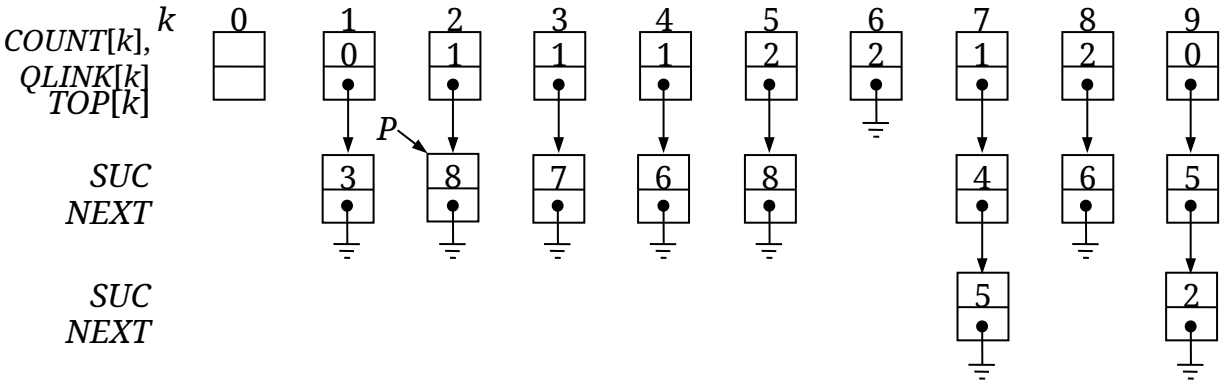
(*COUNT*[*k*] = *COUNT*[*k*] + 1) = (*COUNT*[8] = *COUNT*[8] + 1) = 1 + 1 = 2.

P ⇐ *AVAIL*,

SUC(*P*) ← *k* = 8,

(*NEXT*(*P*) ← *TOP*[*j*]) = (*NEXT*(*P*) ← *TOP*[2]),

(*TOP*[*j*] ← *P*) = (*TOP*[2] ← *P*).



***T*2.** [Следующее отношение.] Входная информация исчерпана, перейти к шагу *T*4.

***T*4.** [Поиск нулей.] (К этому моменту завершена фаза ввода и входная информация преобразована в машинное представление, показанное на рисунке выше. Производим теперь начальную установку очереди вывода, которая связывается по полю *QLINK*.)

R ← 0, *QLINK*[0] ← 0.

k ← 1:

(*COUNT*[*k*] = 0) = (*COUNT*[1] = 0) = (0 = 0) = ИСТИНА, следовательно,

(*QLINK*[*R*] ← *k*) = (*QLINK*[0] ← 1), *R* ← *k* = 1.

k ← 2:

(*COUNT*[*k*] = 0) = (*COUNT*[2] = 0) = (1 = 0) = ЛОЖЬ.

k = 3:

(*COUNT*[*k*] = 0) = (*COUNT*[3] = 0) = (1 = 0) = ЛОЖЬ.

k = 4:

(*COUNT*[k] = 0) = (*COUNT*[4] = 0) = (1 = 0) = ЛОЖЬ.

k = 5:

(*COUNT*[k] = 0) = (*COUNT*[5] = 0) = (2 = 0) = ЛОЖЬ.

k = 6:

(*COUNT*[k] = 0) = (*COUNT*[6] = 0) = (2 = 0) = ЛОЖЬ.

k = 7:

(*COUNT*[k] = 0) = (*COUNT*[7] = 0) = (1 = 0) = ЛОЖЬ.

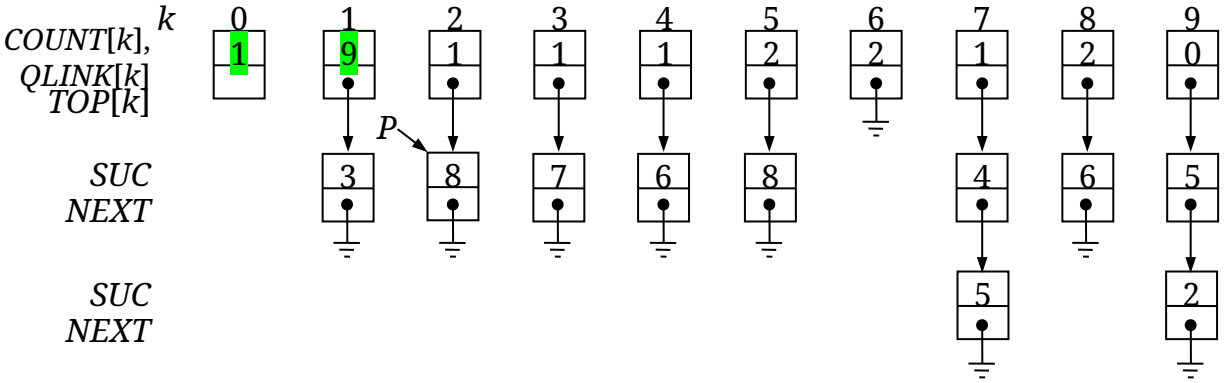
k = 8:

(*COUNT*[k] = 0) = (*COUNT*[8] = 0) = (2 = 0) = ЛОЖЬ.

k = 9:

(*COUNT*[k] = 0) = (*COUNT*[9] = 0) = (0 = 0) = ИСТИНА, следовательно,

(*QLINK*[*R*] ← k) = (*QLINK*[1] ← 9), *R* ← k = 9.



(*F* ← *QLINK*[0]) = (*F* ← *QLINK*[0] = 1).

Очередь вывода

F → 1 → 9 ← *R*

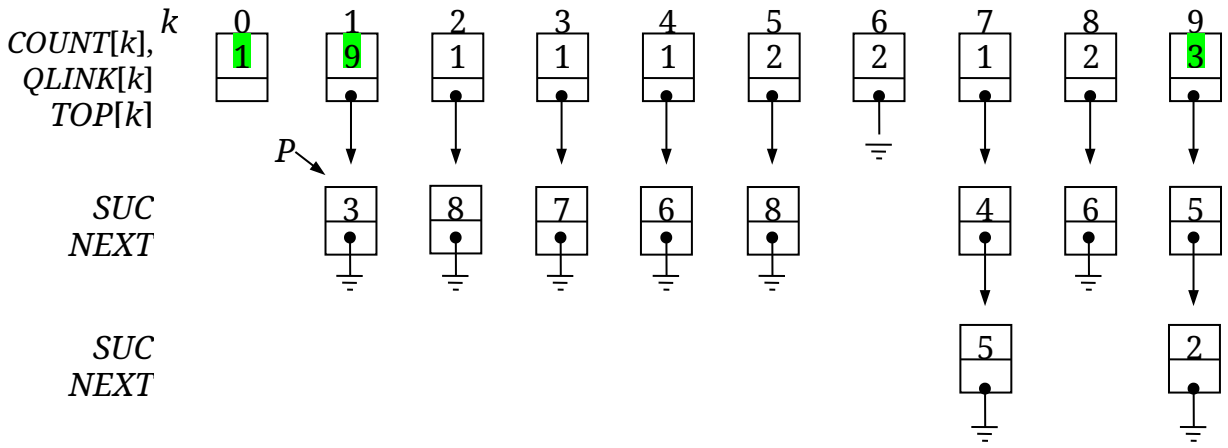
***T*5.** [Вывод из начала очереди.]

Вывести значение *F* = 1.

(*F* = 0) = (1 = 0) = ЛОЖЬ,

N ← *N* – 1 = 9 – 1 = 8,

P ← *TOP*[*F*] = *TOP*[1].



***T*6.** [Стирание отношения.]

(P = Λ) = ЛОЖЬ,

($COUNT[SUC(P)] \leftarrow COUNT[SUC(P)] - 1$) = ($COUNT[3] \leftarrow COUNT[3] - 1 = 1 - 1 = 0$)

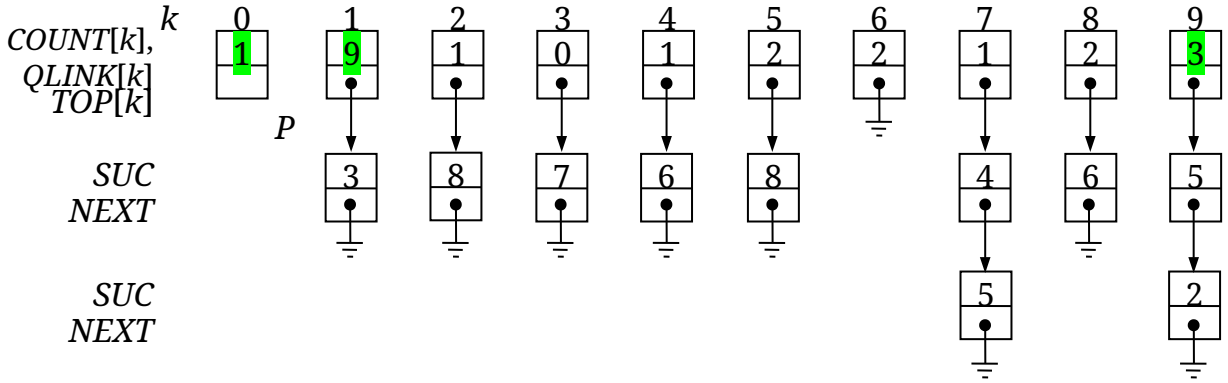
($QLINK(R) \leftarrow SUC(P) = 3$) = ($QLINK(9) \leftarrow 3$),

$R \leftarrow SUC(P) = 3$.

Очередь вывода

$F \rightarrow 1 \rightarrow 9 \rightarrow 3 \leftarrow R$

$P \leftarrow NEXT(P)$.



***T*6.** [Стирание отношения.]

(P = Λ) = ИСТИНА, перейти к шагу *T*7.

***T*7.** [Исключение из очереди.]

($F \leftarrow QLINK[F]$) = ($F \leftarrow QLINK[1] = 9$).

Очередь вывода

$F \rightarrow 9 \rightarrow 3 \leftarrow R$

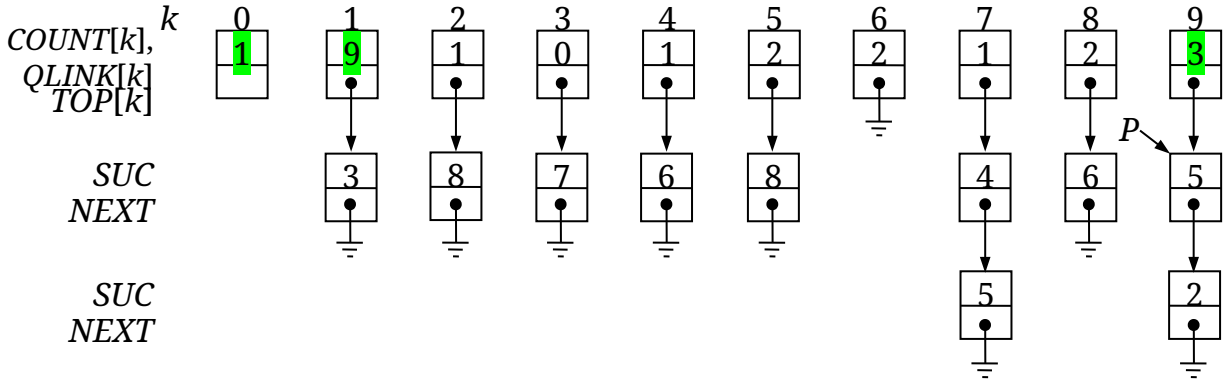
***T*5.** [Вывод из начала очереди.]

Вывести значение $F = 9$.

$(F = 0) = (9 = 0)$ = ЛОЖЬ,

$N \leftarrow N - 1 = 8 - 1 = 7$,

$P \leftarrow TOP[F] = TOP[9]$.

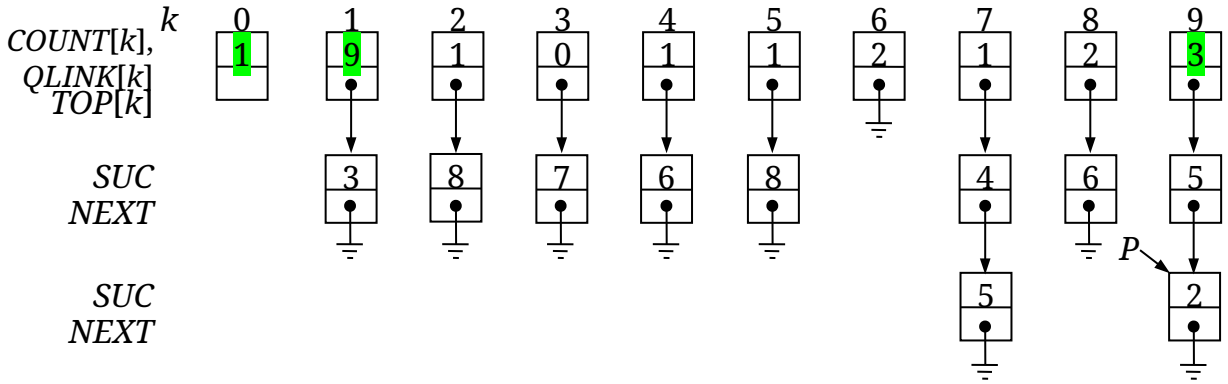


***T*6.** [Стирание отношения.]

$(P = \Lambda)$ = ЛОЖЬ,

$(COUNT[SUC(P)] \leftarrow COUNT[SUC(P)] - 1) = (COUNT[5] \leftarrow COUNT[5] - 1 = 2 - 1 = 1)$.

$P \leftarrow NEXT(P)$.



***T*6.** [Стирание отношения.]

(P = Λ) = ЛОЖЬ,

($COUNT$[SUC(P)] ← $COUNT$[SUC(P)] – 1) = ($COUNT$[2] ← $COUNT$[2] – 1 = 1 – 1 = 0).

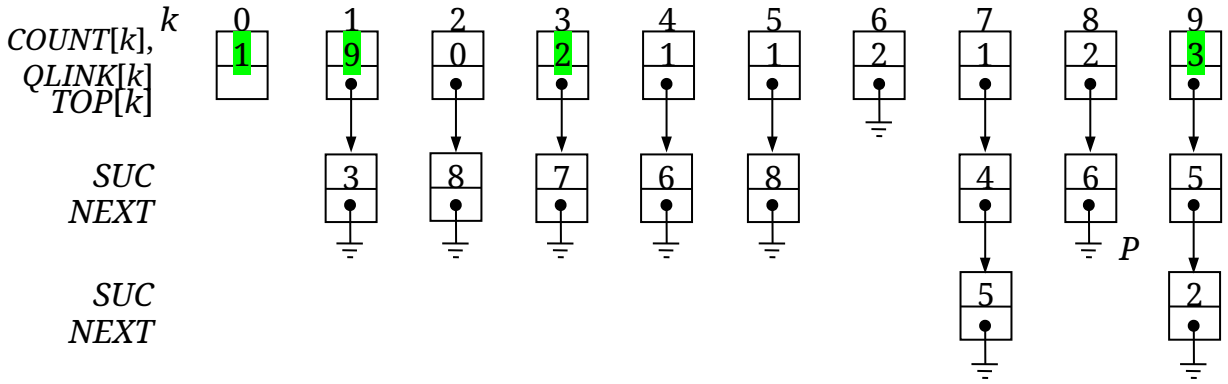
($QLINK$(R) ← SUC(P)) = ($QLINK$(3) ← 2),

R ← SUC(P) = 2.

Очередь вывода

F → 9 → 3 → 2 ← R

P ← $NEXT$(P).



***T*6.** [Стирание отношения.]

(P = Λ) = ИСТИНА, перейти к шагу *T*7.

***T*7.** [Исключение из очереди.]

(F ← $QLINK$[F]) = (F ← $QLINK$[9] = 3).

Очередь вывода

F → 3 → 2 ← R

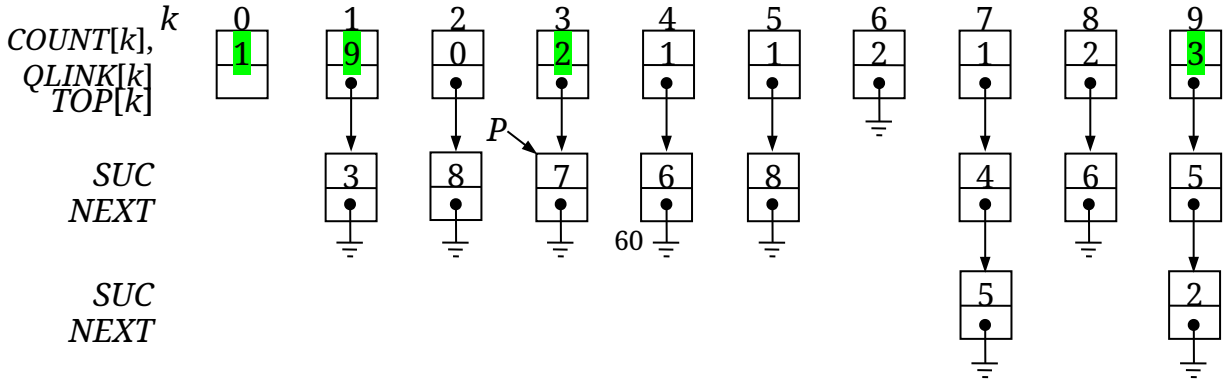
***T*5.** [Вывод из начала очереди.]

Вывести значение F = 3.

(F = 0) = (3 = 0) = ЛОЖЬ,

N ← N – 1 = 7 – 1 = 6,

P ← TOP[F] = TOP[3].



***T*6.** [Стирание отношения.]

(P = Λ) = ЛОЖЬ,

($COUNT$[SUC(P)] ← $COUNT$[SUC(P)] – 1) = ($COUNT$[7] ← $COUNT$[7] – 1 = 1 – 1 = 0).

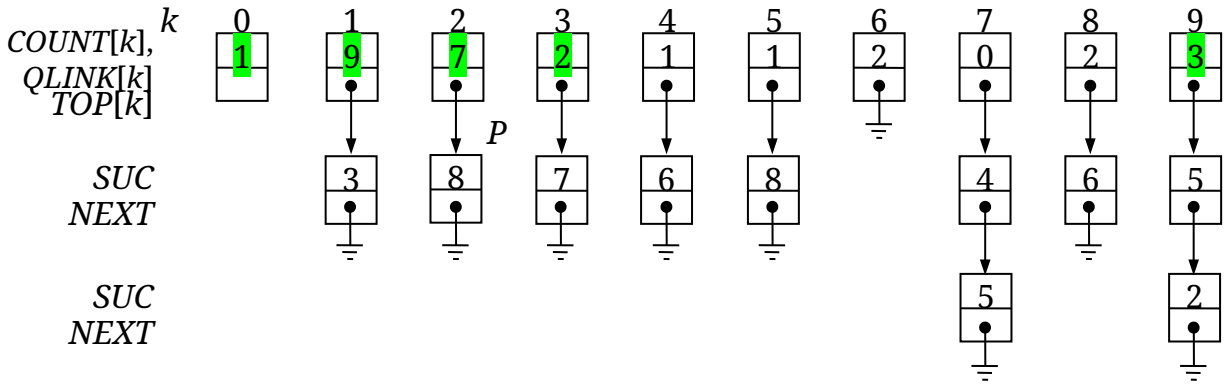
($QLINK$(R) ← SUC(P)) = ($QLINK$(2) ← 7),

R ← SUC(P) = 7.

Очередь вывода

F → 3 → 2 → 7 ← R

P ← $NEXT$(P).



***T*6.** [Стирание отношения.]

(P = Λ) = ИСТИНА, перейти к шагу *T*7.

***T*7.** [Исключение из очереди.]

(F ← $QLINK$[F]) = (F ← $QLINK$[3] = 2).

Очередь вывода

F → 2 → 7 ← R

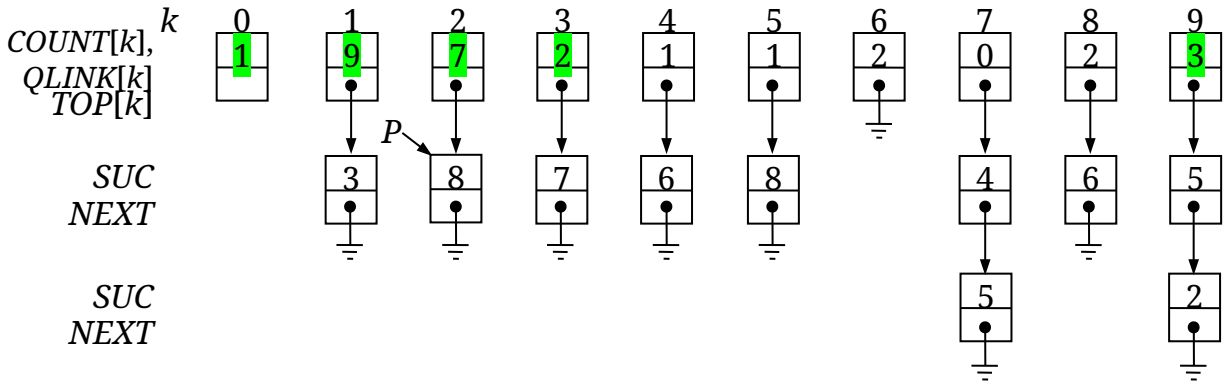
***T*5.** [Вывод из начала очереди.]

Вывести значение F = 2.

(F = 0) = (2 = 0) = ЛОЖЬ,

$N \leftarrow N - 1 = 6 - 1 = 5$,

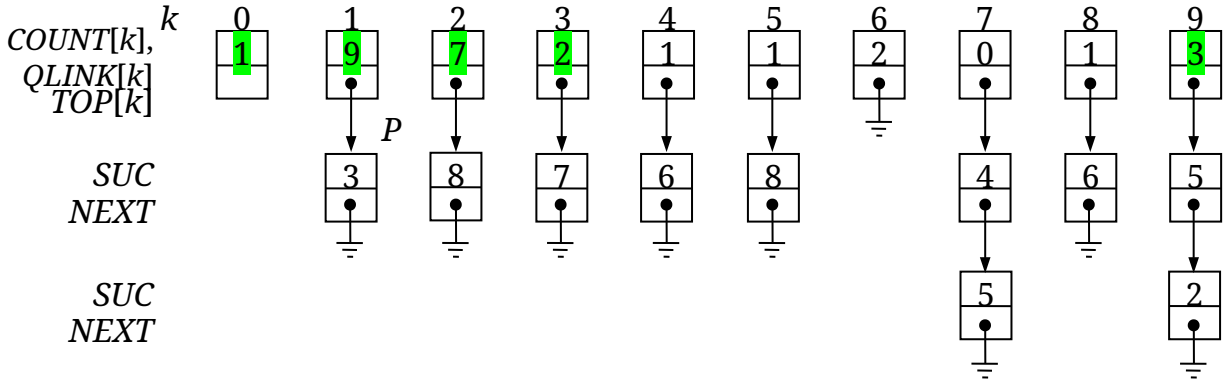
$P \leftarrow TOP[F] = TOP[2]$.



***T*6.** [Стирание отношения.]

$(P = \Lambda)$ = ЛОЖЬ,

$(COUNT[SUC(P)] \leftarrow COUNT[SUC(P)] - 1) = (COUNT[8] \leftarrow COUNT[8] - 1 = 2 - 1 = 1)$. $P \leftarrow NEXT(P)$.



***T*6.** [Стирание отношения.]

$(P = \Lambda)$ = ИСТИНА, перейти к шагу *T*7.

***T*7.** [Исключение из очереди.]

$(F \leftarrow QLINK[F]) = (F \leftarrow QLINK[2] = 7)$.

Очередь вывода

$F \rightarrow 7 \leftarrow R$

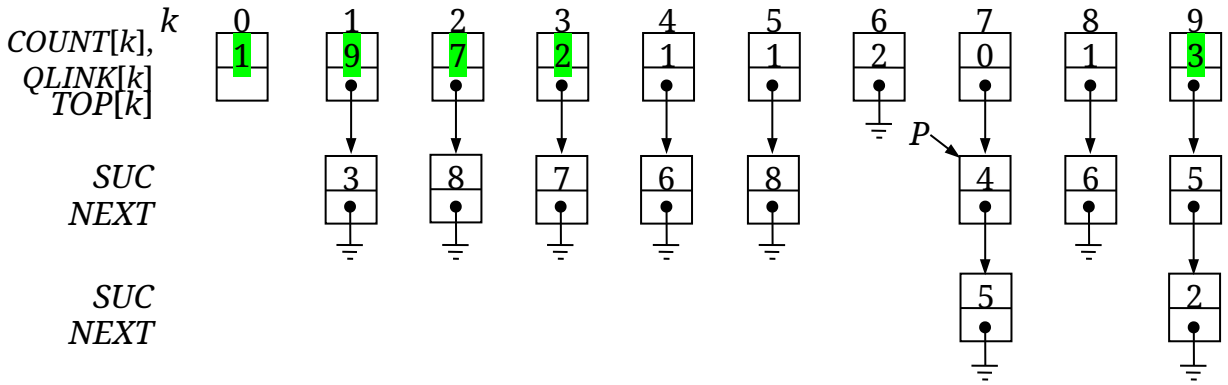
***T*5.** [Вывод из начала очереди.]

Вывести значение $F = 7$.

$(F = 0) = (7 = 0)$ = ЛОЖЬ,

$N \leftarrow N - 1 = 5 - 1 = 4$,

$P \leftarrow TOP[F] = TOP[7]$.



***T*6.** [Стирание отношения.]

$(P = \Lambda)$ = ЛОЖЬ,

$(COUNT[SUC(P)] \leftarrow COUNT[SUC(P)] - 1) = (COUNT[4] \leftarrow COUNT[4] - 1 = 1 - 1 = 0)$.

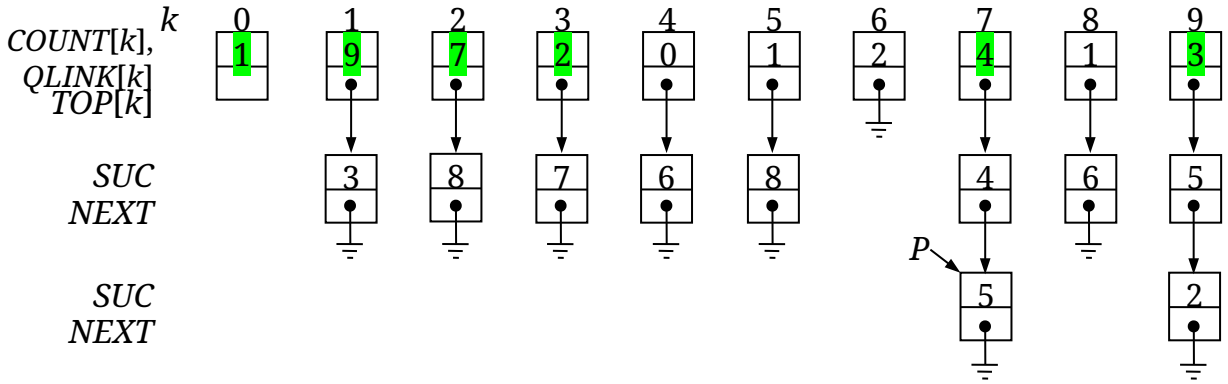
$(QLINK(R) \leftarrow SUC(P)) = (QLINK(7) \leftarrow 4)$,

$R \leftarrow SUC(P) = 4$.

Очередь вывода

$F \rightarrow 7 \rightarrow 4 \leftarrow R$

$P \leftarrow NEXT(P)$.



***T*6.** [Стирание отношения.]

$(P = \Lambda)$ = ЛОЖЬ,

$(COUNT[SUC(P)] \leftarrow COUNT[SUC(P)] - 1) = (COUNT[5] \leftarrow COUNT[5] - 1 = 1 - 1 = 0)$.

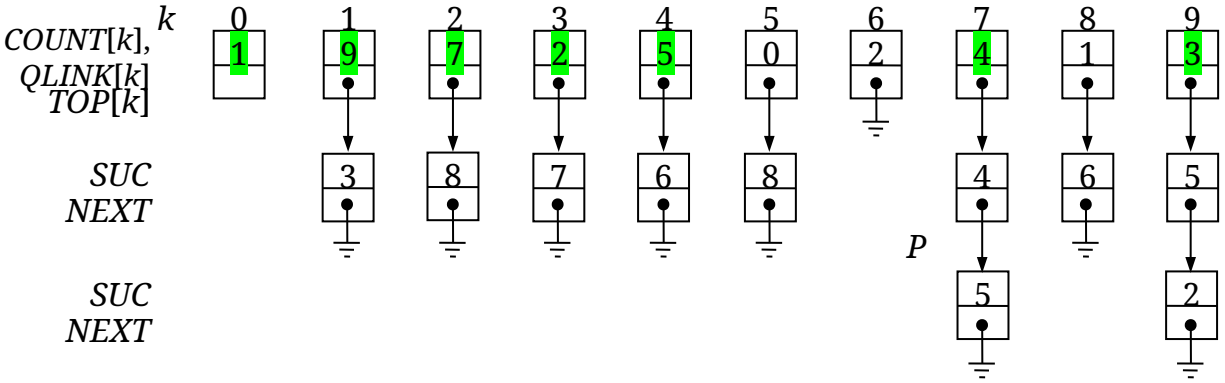
(*QLINK*(*R*) ← *SUC*(*P*)) = (*QLINK*(4) ← 5),

R ← *SUC*(*P*) = 5.

Очередь вывода

F → 7 → 4 → 5 ← *R*

P ← *NEXT*(*P*).



***T*6.** [Стирание отношения.]

(*P* = Λ) = ИСТИНА, перейти к шагу *T*7.

***T*7.** [Исключение из очереди.]

(*F* ← *QLINK*[*F*]) = (*F* ← *QLINK*[7] = 4).

Очередь вывода

F → 4 → 5 ← *R*

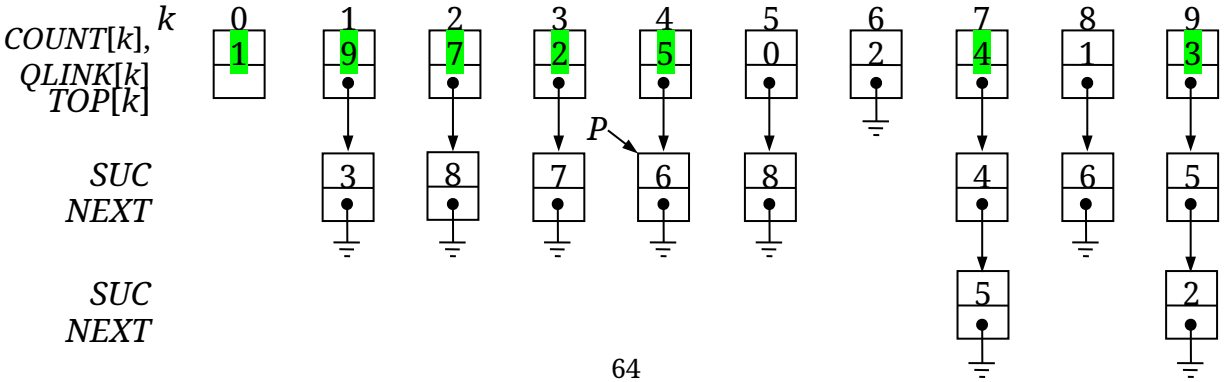
***T*5.** [Вывод из начала очереди.]

Вывести значение *F* = 4.

(*F* = 0) = (4 = 0) = ЛОЖЬ,

N ← *N* – 1 = 4 – 1 = 3,

P ← *TOP*[*F*] = *TOP*[4].

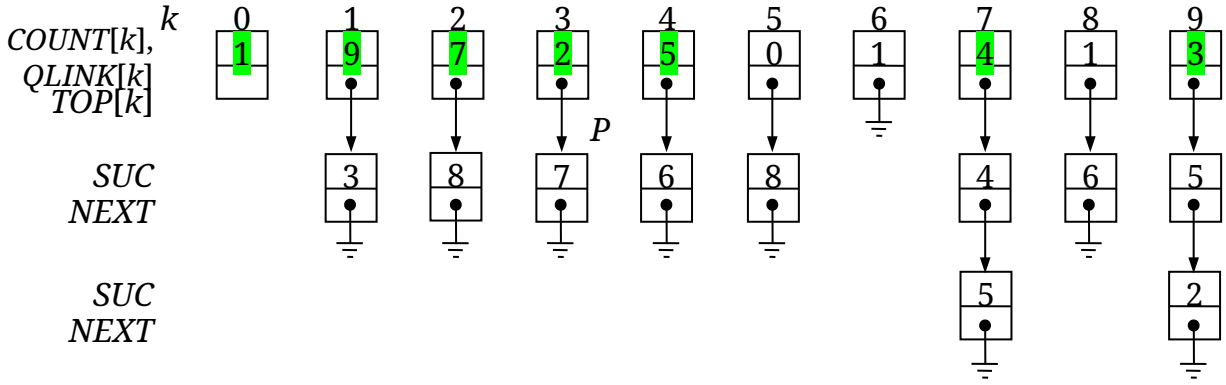


***T*6.** [Стирание отношения.]

(P = Λ) = ЛОЖЬ,

($COUNT$[SUC(P)] ← $COUNT$[SUC(P)] – 1) = ($COUNT$[6] ← $COUNT$[6] – 1 = 2 – 1 = 1).

P ← $NEXT$(P).



***T*6.** [Стирание отношения.]

(P = Λ) = ИСТИНА, перейти к шагу *T*7.

***T*7.** [Исключение из очереди.]

(F ← $QLINK$[F]) = (F ← $QLINK$[4] = 5).

Очередь вывода

F → 5 ← R

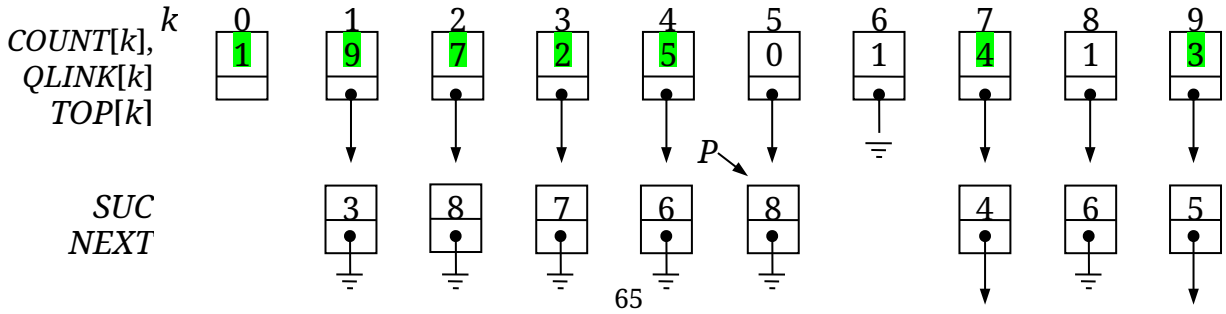
***T*5.** [Вывод из начала очереди.]

Вывести значение F = 5.

(F = 0) = (5 = 0) = ЛОЖЬ,

N ← N – 1 = 3 – 1 = 2,

P ← TOP[F] = TOP[5].



***T*6.** [Стирание отношения.]

(P = Λ) = ЛОЖЬ,

($COUNT$[SUC(P)] ← $COUNT$[SUC(P)] – 1) = ($COUNT$[8] ← $COUNT$[8] – 1 = 1 – 1 = 0).

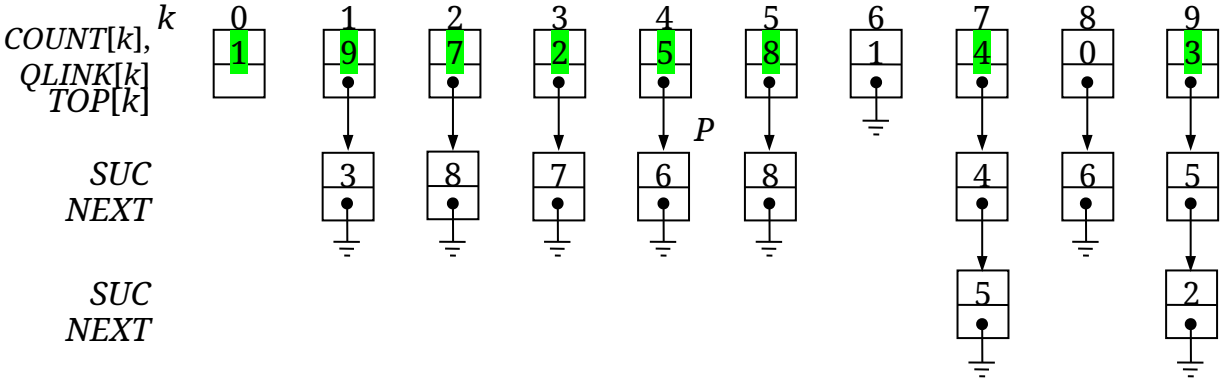
($QLINK$(R) ← SUC(P)) = ($QLINK$(5) ← 8),

R ← SUC(P) = 8.

Очередь вывода

F → 5 → 8 ← R

P ← $NEXT$(P).



***T*6.** [Стирание отношения.]

(P = Λ) = ИСТИНА, перейти к шагу *T*7.

***T*7.** [Исключение из очереди.]

(F ← $QLINK$[F]) = (F ← $QLINK$[5] = 8).

Очередь вывода

F → 8 ← R

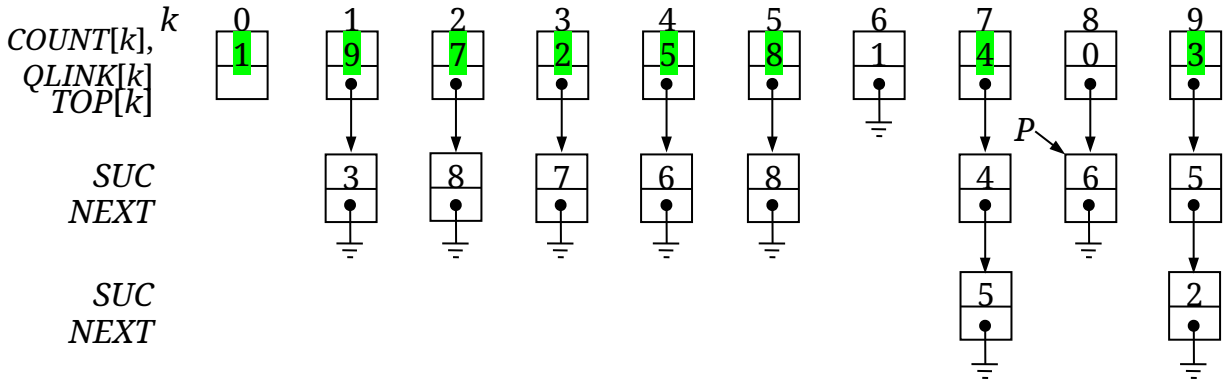
***T*5.** [Вывод из начала очереди.]

Вывести значение F = 8.

(F = 0) = (8 = 0) = ЛОЖЬ,

N ← N – 1 = 2 – 1 = 1,

$P \leftarrow TOP[F] = TOP[8]$.



***T*6.** [Стирание отношения.]

$(P = \Lambda)$ = ЛОЖЬ,

$(COUNT[SUC(P)] \leftarrow COUNT[SUC(P)] - 1) = (COUNT[6] \leftarrow COUNT[6] - 1 = 1 - 1 = 0)$.

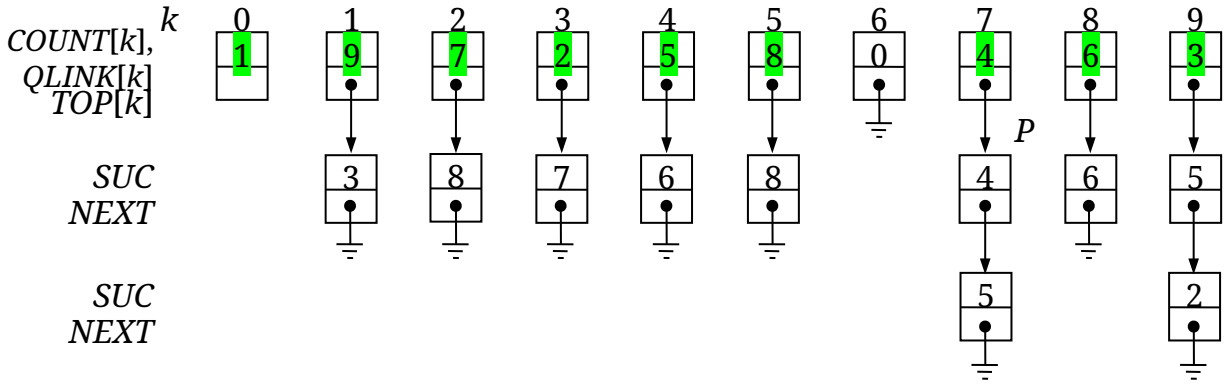
$(QLINK(R) \leftarrow SUC(P)) = (QLINK(8) \leftarrow 6)$,

$R \leftarrow SUC(P) = 6$.

Очередь вывода

$F \rightarrow 8 \rightarrow 6 \leftarrow R$

$P \leftarrow NEXT(P)$.



***T*6.** [Стирание отношения.]

$(P = \Lambda)$ = ИСТИНА, перейти к шагу *T*7.

***T*7.** [Исключение из очереди.]

$(F \leftarrow QLINK[F]) = (F \leftarrow QLINK[8] = 6)$.

Очередь вывода

$F \rightarrow 6 \leftarrow R$

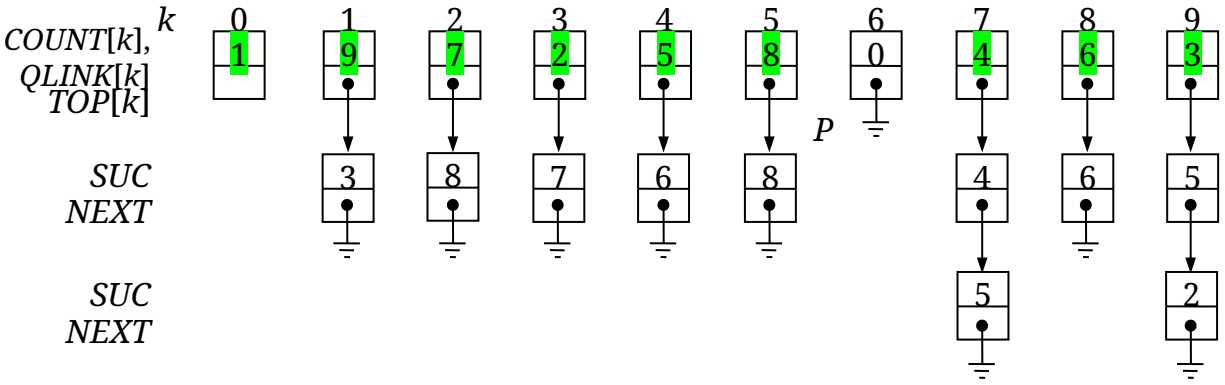
***T*5.** [Вывод из начала очереди.]

Вывести значение $F = 6$.

$(F = 0) = (6 = 0)$ = ЛОЖЬ,

$N \leftarrow N - 1 = 1 - 1 = 0$,

$P \leftarrow TOP[F] = TOP[6]$.



***T*6.** [Стирание отношения.]

$(P = \Lambda)$ = ИСТИНА, перейти к шагу *T*7.

***T*7.** [Исключение из очереди.]

$(F \leftarrow QLINK[F]) = (F \leftarrow QLINK[6] = 0)$.

Очередь вывода

$F \rightarrow 0 \quad 6 \leftarrow R$

***T*5.** [Вывод из начала очереди.]

Вывести значение $F = 0$.

$(F = 0) = (0 = 0)$ = ИСТИНА, перейти к шагу *T*8.

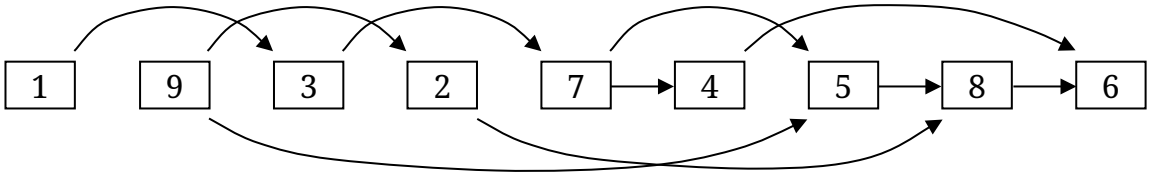
***T*8.** [Окончание процесса.]

Конец алгоритма.

$(N = 0) = (0 = 0)$ = ИСТИНА – выведены все номера элементов в требуемом топологическом порядке и вслед за ними нуль.

Ответ: 1 9 3 2 7 4 5 8 6 0

Частично упорядоченное множество после топологической сортировки:



Линейная последовательность отсортированных объектов такова, что в графе все дуги ориентированы слева-направо.

Лабораторная работа № 4

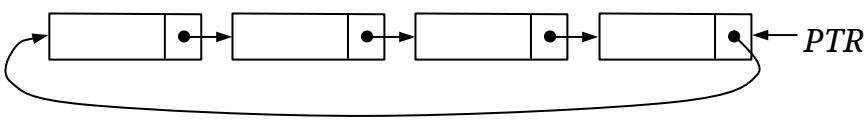
СЛОЖЕНИЕ МНОГОЧЛЕНОВ

Цель. Знакомство с понятием связанного распределения памяти при хранении циклических списков и алгоритмом сложения многочленов, представленных циклическими списками.

Задание. Выполнить по алгоритму *A* прибавление одного многочлена к другому. Определить последовательность выполнения шагов алгоритма и итоговый многочлен. Варианты конкретных прибавлений представлены в табл. 4.

Основные положения

Циклический список – связанный список такой, что связь его последнего узла не пустая, а идёт к первому узлу этого же списка:



(23)

Здесь *PTR* – указатель на самый правый узел списка; *LINK*(*PTR*) – адрес самого левого узла.

Таблица 4

Вариант	Прибавление	
	многочлена	к многочлену
1	$xy + xz - x$	$-xz + 2x - 2y$
2	$-2xy - 2xz + 1$	$xy + 2yz - 1$
3	$xy - x + 2z$	$xy - 2x - y$
4	$-2xyz - 2z - 1$	$x + 2y + 2z$
5	$2xy - 2y + 2z$	$xyz + y - 2z$
6	$2xyz + xz - yz$	$-xz - 2yz - z$
7	$2xy - y - 1$	$2xz - 2y + 2z$
8	$2xyz - 2x - z$	$-2xyz - 2z + 2$
9	$2xy - 2y + z$	$2xyz - 2xy + 2z$
10	$-2x + z + 1$	$2xyz + x - 1$
11	$2yz - z + 2$	$x + y + z$
12	$2yz + 2y - z$	$-2xy - 2yz - 1$
13	$2xz + y - 2z$	$-yz - x - y$
14	$-2xz - 2yz + y$	$xz + 2yz + 2x$
15	$xyz - xy + 2$	$-2yz - z - 1$
16	$-xyz - xy - 2z$	$xyz + xy + x$
17	$2xy + xz - x$	$x - 2y + 2$
18	$yz + y + 2z$	$xyz - y + 2z$
19	$xyz - y - z$	$2xyz + yz + z$
20	$-2yz - 2x + y$	$-2xy - y - 1$
21	$-2y + z + 1$	$-2yz - 2x - 1$
22	$-xyz + xy + 1$	$2xyz - 2xy + 2yz$
23	$-2xyz - xy - 2y$	$2xyz + 2xy + x$
24	$-y + 2z - 2$	$xyz - y + 2$
25	$2xyz + yz - 1$	$2xz - yz + 2$
26	$-2x + z + 1$	$2x - 2y + 2$
27	$-2xz + x + 2y$	$-2xy - x - y$
28	$-2yz + z - 1$	$-2xy + 2z + 1$

29	$2y + 2z + 1$	$xyz - 2y + z$
30	$xy - y - 2$	$-xz - y + 2$

Наиболее важными операциями являются:

a) Включение *Y* слева: $P \Leftarrow AVAIL$, $INFO(P) \leftarrow Y$, $LINK(P) \leftarrow LINK(PTR)$, $LINK(PTR) \leftarrow P$.

b) Включение *Y* справа: Включение *Y* слева, затем $PTR \leftarrow P$.

c) Запись в *Y* значения из левого узла и исключение этого узла из списка: $P \leftarrow LINK(PTR)$, $Y \leftarrow INFO(P)$, $LINK(PTR) \leftarrow LINK(P)$, $AVAIL \Leftarrow P$.

Описание операций (*a*), (*b*) и (*c*) не учитывает того, что циклический список может быть пустым. Условившись, что *PTR* равен Λ в случае пустого списка, уточним упомянутые операции:

a) Включение *Y* слева: $P \Leftarrow AVAIL$, $INFO(P) \leftarrow Y$. Если $PTR = \Lambda$, то $PTR \leftarrow LINK(P) \leftarrow P$; иначе $LINK(P) \leftarrow LINK(PTR)$, $LINK(PTR) \leftarrow P$.

b) Включение *Y* справа: Включение *Y* слева, затем $PTR \leftarrow P$.

c) Запись в *Y* значения из левого узла и исключение этого узла из списка: Если $PTR = \Lambda$, то НЕХВАТКА; иначе $P \leftarrow LINK(PTR)$, $Y \leftarrow INFO(P)$, $LINK(PTR) \leftarrow LINK(P)$, $AVAIL \Leftarrow P$ и если $PTR = P$, то $PTR \leftarrow \Lambda$.

Заметим, что операции (*a*), (*b*) и (*c*) являются операциями дека, ограниченного по выходу. Это значит, что циклический список можно использовать или как стек (операции (*a*) и (*c*)), или как очередь (операции (*b*) и (*c*)).

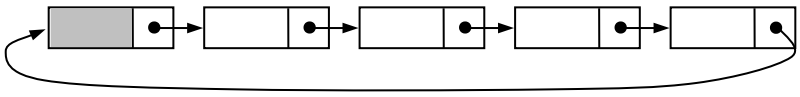
Применительно к циклическим спискам эффективны и некоторые другие важные операции:

d) Очистка списка: Если $PTR \neq \Lambda$, то $P \leftarrow AVAIL$, $AVAIL \leftarrow LINK(PTR)$, $LINK(PTR) \leftarrow P$.

e) Включение циклического списка L_2 в циклический список L_1: Если $PTR_2 \neq \Lambda$ и $PTR_1 \neq \Lambda$, то $P \leftarrow LINK(PTR_1)$, $LINK(PTR_1) \leftarrow LINK(PTR_2)$, $LINK(PTR_2) \leftarrow P$, $PTR_1 \leftarrow PTR_2$, $PTR_2 \leftarrow \Lambda$.

Расщепление одного циклического списка на два представляет ещё одну простую операцию.

Таким образом, циклические списки можно использовать не только для циклических, но и для линейных структур. Тогда возникает вопрос о конце циклического списка. Ответом на него может быть включение в каждый циклический список специального, отличимого от всех, узла, называемого *головой списка*.

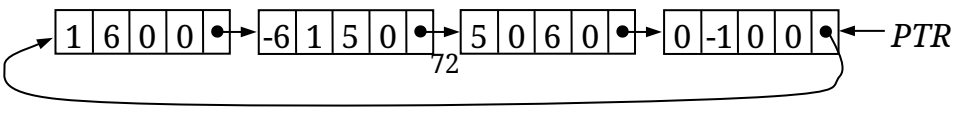


Одним из преимуществ циклического списка с головным узлом является то, что он никогда не будет пустым. При ссылке на списки такого типа вместо указателя на правый конец списка используется обычно голова списка, которая часто находится в фиксированной ячейке памяти.

В качестве примера использования циклических списков рассмотрим арифметическую операцию сложения многочленов. Предположим, что многочлен представлен в виде списка, в котором каждый узел, кроме специального, обозначает ненулевой член и имеет следующий вид:

COEF	*A*	*B*	*C*	*LINK*

Здесь *COEF* является коэффициентом при члене $x^A y^B z^C$. У специального узла *COEF* = 0 и *A* = – 1. Для полей *A*, *B* и *C* узла списка далее будет использоваться обобщённое обозначение *ABC*. Узлы списка всегда располагаются в *порядке убывания* поля *ABC*, за исключением специального узла, после которого находится узел с наибольшим значением *ABC*. Например, многочлен $x^6 - 6xy^5 + 5y^6$ будет представлен следующим образом:



Алгоритм *A*. (*Сложение многочленов.*) Этот алгоритм прибавляет многочлен, на который указывает P, к многочлену, на который указывает Q. При этом список по указателю P не изменяется, а в списке по указателю Q будет получена сумма двух многочленов. В конце алгоритма переменные P и Q принимают свои исходные значения. В алгоритме используются также вспомогательные указатели Q_1 и Q_2.

***A*1.** [Начальная установка.] $P \leftarrow LINK(P)$, $Q_1 \leftarrow Q$, $Q \leftarrow LINK(Q)$. (Теперь P и Q указывают на старшие члены многочленов. Переменная Q_1 в алгоритме почти везде будет «на один шаг отставать» от Q, т.е. будет выполняться равенство $Q = LINK(Q_1)$.)

***A*2.** [Сравнение $ABC(P)$ и $ABC(Q)$.] Если $ABC(P) < ABC(Q)$, то $Q_1 \leftarrow Q$, $Q \leftarrow LINK(Q)$ и выполнить этот шаг сначала. Если $ABC(P) = ABC(Q)$, то перейти к шагу *A*3. Если $ABC(P) > ABC(Q)$, то перейти к шагу *A*5.

***A*3.** [Сложение коэффициентов.] (Найдены два члена с равными степенями.) Если $ABC(P) < 0$, то конец алгоритма; в противном случае $COEF(Q) \leftarrow COEF(Q) + COEF(P)$. Теперь если $COEF(Q) = 0$, то перейти к шагу *A*4; в противном случае $Q_1 \leftarrow Q$, $P \leftarrow LINK(P)$, $Q \leftarrow LINK(Q)$ и вернуться к шагу *A*2.

***A*4.** [Исключение нулевого члена.] $Q_2 \leftarrow Q$, $LINK(Q_1) \leftarrow Q \leftarrow LINK(Q)$ и $AVAIL \Leftarrow Q_2$. (Нулевой член, образовавшийся на шаге *A*3, исключён из многочлена Q.) $P \leftarrow LINK(P)$ и вернуться к шагу *A*2.

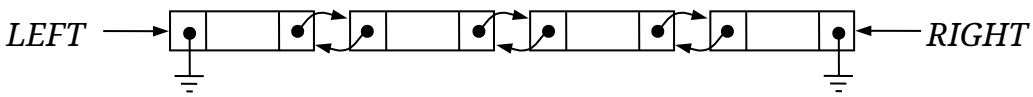
***A*5.** [Включение нового члена.] (Многочлен P содержит член, который не представлен в многочлене Q и поэтому мы включаем его в многочлен Q.) $Q_2 \Leftarrow AVAIL$, $COEF(Q_2) \leftarrow COEF(P)$, $ABC(Q_2) \leftarrow ABC(P)$, $LINK(Q_2) \leftarrow Q$, $LINK(Q_1) \leftarrow Q_2$, $Q_1 \leftarrow Q_2$, $P \leftarrow LINK(P)$ и вернуться к шагу *A*2. ■

В рассмотренном алгоритме значением отношения:

- *ABC*(*P*) < *ABC*(*Q*) следует считать значение логического выражения *A*(*P*) + *B*(*P*) + *C*(*P*) < *A*(*Q*) + *B*(*Q*) + *C*(*Q*) **или** *A*(*P*) + *B*(*P*) + *C*(*P*) = *A*(*Q*) + *B*(*Q*) + *C*(*Q*) **и** (*A*(*P*) < *A*(*Q*) **или** *A*(*P*) = *A*(*Q*) **и** *B*(*P*) < *B*(*Q*));
- *ABC*(*P*) = *ABC*(*Q*) – значение логического выражения *A*(*P*) = *A*(*Q*) **и** *B*(*P*) = *B*(*Q*) **и** *C*(*P*) = *C*(*Q*);
- *ABC*(*P*) > *ABC*(*Q*) – значение ИСТИНА, если (*ABC*(*P*) < *ABC*(*Q*)) = ЛОЖЬ и (*ABC*(*P*) = *ABC*(*Q*)) = ЛОЖЬ, а значение ЛОЖЬ в противном случае.

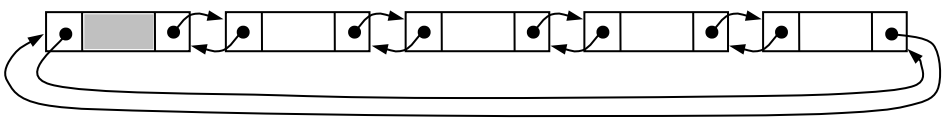
Время выполнения реализующей данный алгоритм программы для гипотетической вычислительной машины *MIX* составляет (29*m'* + 18*m"* + + 29*p'* + 8*q'* + 13) *u*, где *m'* – количество подобных и взаимно уничтожающихся членов; *m"* – количество подобных, но взаимно не уничтожающихся членов; *p'* – количество членов в многочлене *P*, которым нет подобных в многочлене *Q*; *q'* – количество членов в многочлене *Q*, которым нет подобных в многочлене *P*; *u* – время выполнения машиной одного такта. Во время выполнения алгоритма в пуле памяти необходимо иметь не менее чем (2 + *p* + *q*) и самое большое (2 + *p* + *q*+ *p'*) узлов.

Списки с двумя связями позволяют достичь ещё большей гибкости в работе с линейными списками. При этом каждый элемент списка имеет две связи *LLINK* и *RLINK*, которые указывают на элементы, находящиеся по обе стороны от данного узла:



(24)

С таким представлением линейного списка легко выполняются операции для дека общего вида. Ещё легче оперировать с двусвязным списком, если его частью является *головной узел*:



(25)

Если такой список пуст, то оба поля связи в его голове указывают на саму голову. Представление (25) полностью удовлетворяет условию

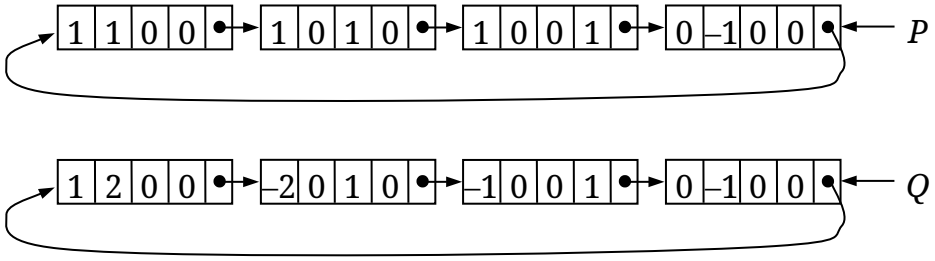
$$RLINK(LLINK(X)) = LLINK(RLINK(X)) = X,$$

где X – адрес любого узла в списке, включая голову. В этом и заключается главная причина предпочтения (25) по сравнению с (24).

Пример выполнения задания

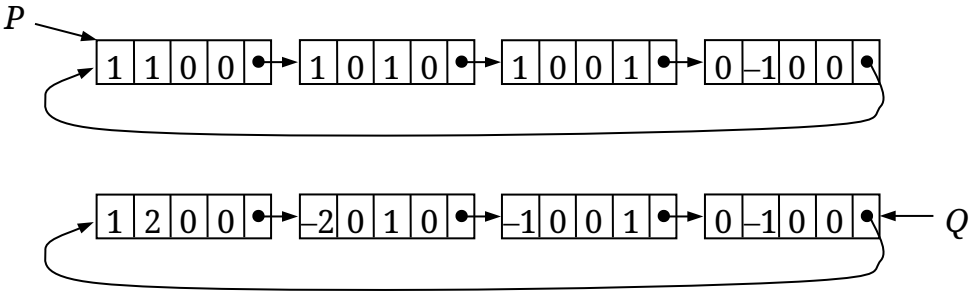
Задание. Выполнить по алгоритму A прибавление многочлена ($x + y + z$) к многочлену ($x^2 - 2y - z$). Определить последовательность выполнения шагов алгоритма и итоговый многочлен.

Представление многочленов циклическими списками

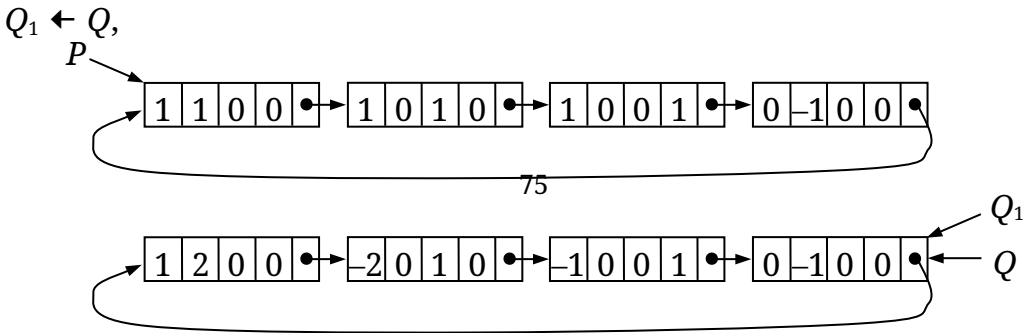


***A*1.** [Начальная установка.]

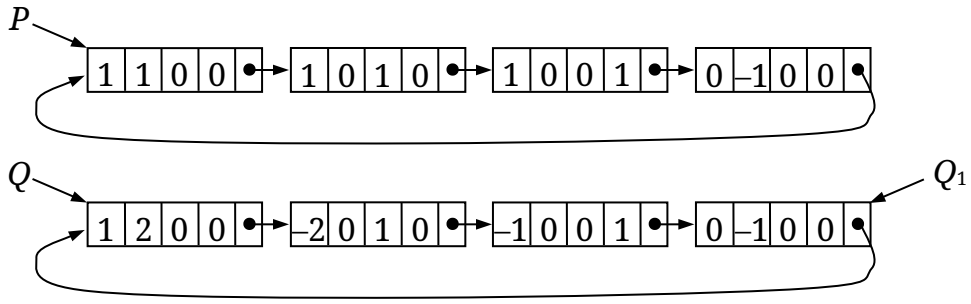
$P \leftarrow LINK(P)$,



$Q_1 \leftarrow Q$,



$Q \leftarrow LINK(Q)$.

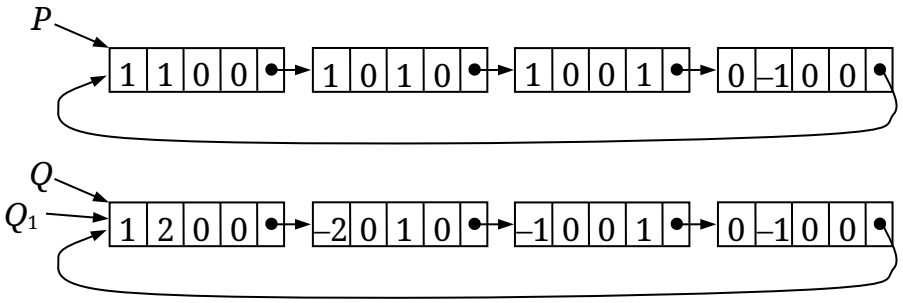


Теперь P и Q указывают на старшие члены многочленов. Переменная Q_1 в алгоритме почти везде будет «на один шаг отставать» от Q, т.е. будет выполняться равенство $Q = LINK(Q_1)$.

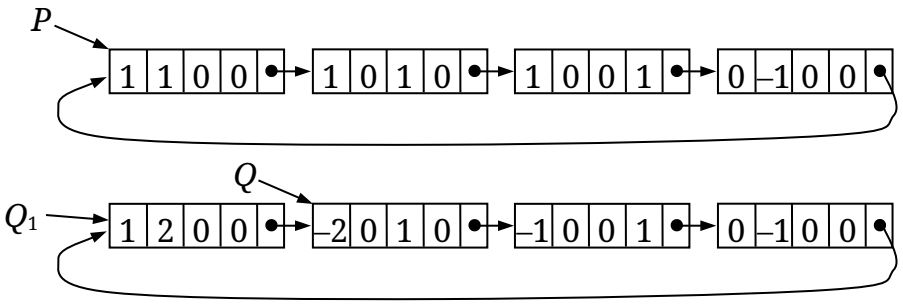
A2. [Сравнение $ABC(P)$ и $ABC(Q)$.]

($ABC(P) < ABC(Q)$) = ($A(P) + B(P) + C(P) < A(Q) + B(Q) + C(Q)$ **или** $A(P) + B(P) + C(P) = A(Q) + B(Q) + C(Q)$ **и** ($A(P) < A(Q)$ **или** $A(P) = A(Q)$ **и** $B(P) < B(Q)$)) = (1 + 0 + 0 < 2 + 0 + 0 **или** 1 + 0 + 0 = 2 + 0 + 0 **и** (1 < 2 **или** 1 = 2 **и** 0 < 0)) = (1 < 2 **или** 1 = 2 **и** (ИСТИНА **или** ЛОЖЬ **и** ЛОЖЬ)) = (ИСТИНА **или** ЛОЖЬ **и** (ИСТИНА **или** ЛОЖЬ)) = (ИСТИНА **или** ЛОЖЬ **и** ИСТИНА) = (ИСТИНА **или** ЛОЖЬ) = ИСТИНА

$Q_1 \leftarrow Q$,



$Q \leftarrow LINK(Q)$



Выполнить этот шаг сначала.

A2. [Сравнение *ABC*(*P*) и *ABC*(*Q*).]

(*ABC*(*P*) < *ABC*(*Q*)) = (*A*(*P*) + *B*(*P*) + *C*(*P*) < *A*(*Q*) + *B*(*Q*) + *C*(*Q*) **или** *A*(*P*) + *B*(*P*) + *C*(*P*) = *A*(*Q*) + *B*(*Q*) + *C*(*Q*) **и** (*A*(*P*) < *A*(*Q*) **или** *A*(*P*) = *A*(*Q*) **и** *B*(*P*) < *B*(*Q*))) = (1 + 0 + 0 < 0 + 1 + 0 **или** 1 + 0 + 0 = 0 + 1 + 0 **и** (1 < 0 **или** 1 = 0 **и** 0 < 1)) = (1 < 1 **или** 1 = 1 **и** (ЛОЖЬ **или** ЛОЖЬ **и** ИСТИНА)) = (ЛОЖЬ **или** ИСТИНА **и** (ЛОЖЬ **или** ЛОЖЬ)) = (ЛОЖЬ **или** ИСТИНА **и** ЛОЖЬ) = (ЛОЖЬ **или** ЛОЖЬ) = ЛОЖЬ

(*ABC*(*P*) = *ABC*(*Q*)) = (*A*(*P*) = *A*(*Q*) **и** *B*(*P*) = *B*(*Q*) **и** *C*(*P*) = *C*(*Q*)) = (1 = 0 **и** 0 = 1 **и** 0 = 0) = (ЛОЖЬ **и** ЛОЖЬ **и** ИСТИНА) = (ЛОЖЬ **и** ИСТИНА) = ЛОЖЬ

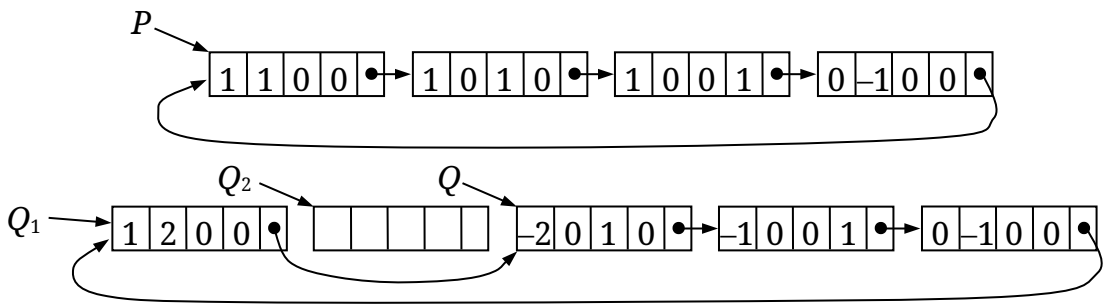
(*ABC*(*P*) < *ABC*(*Q*)) = ЛОЖЬ и (*ABC*(*P*) = *ABC*(*Q*)) = ЛОЖЬ, следовательно (*ABC*(*P*) > *ABC*(*Q*)).

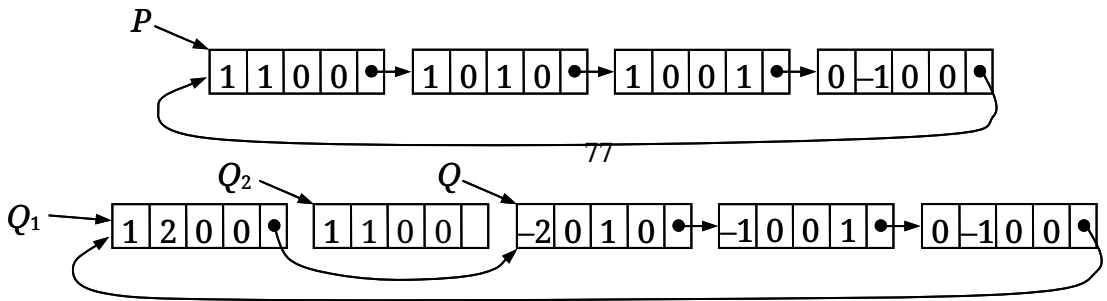
Перейти к шагу *A*5.

A5. [Включение нового члена.]

Многочлен *P* содержит член, который не представлен в многочлене *Q* и поэтому мы включаем его в многочлен *Q*.

$Q_2 \Leftarrow AVAIL$,

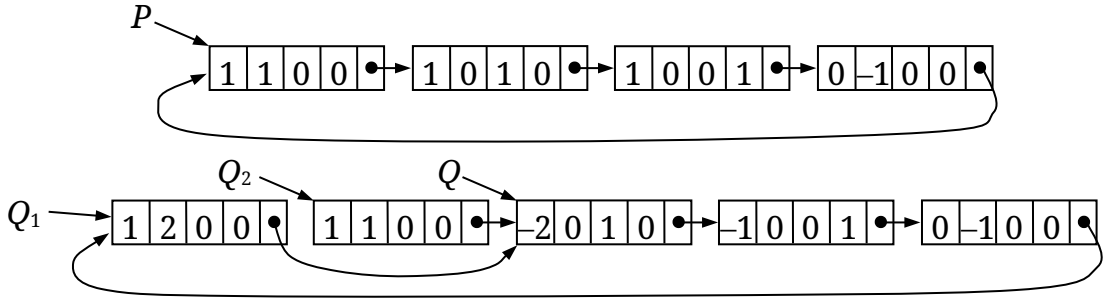




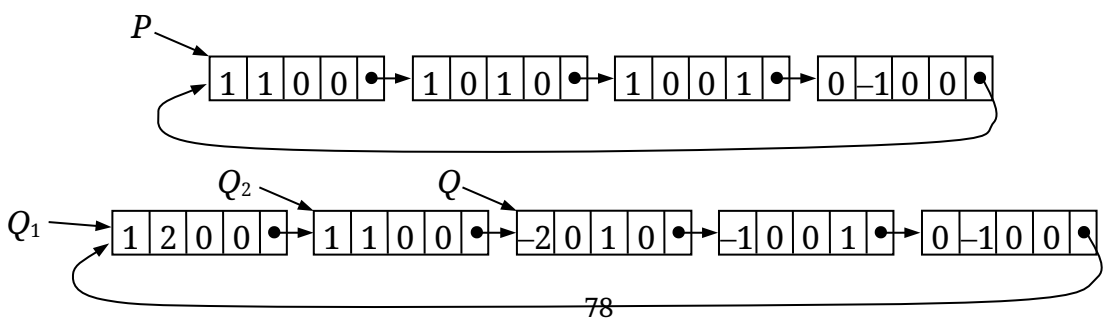
$COEF(Q_2) \leftarrow COEF(P)$,

$ABC(Q_2) \leftarrow ABC(P)$,

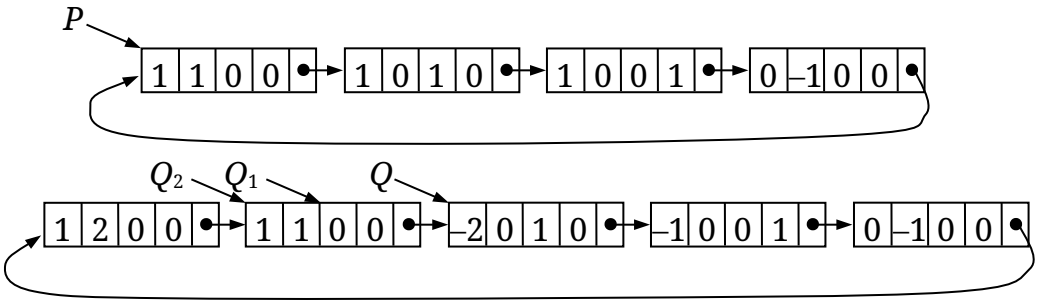
$LINK(Q_2) \leftarrow Q$,



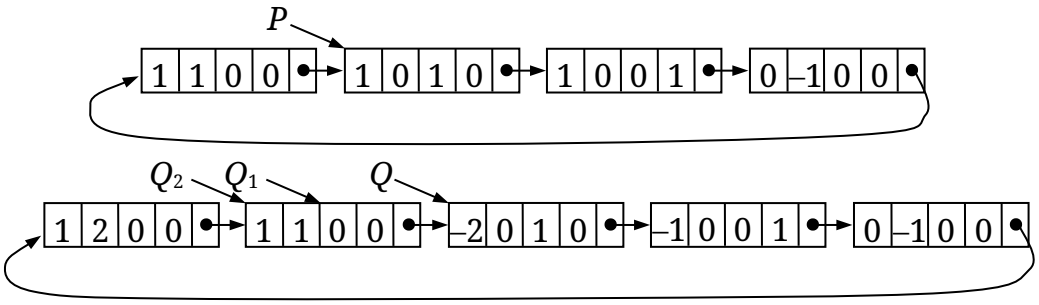
$LINK(Q_1) \leftarrow Q_2$,



$Q_1 \leftarrow Q_2$,



$P \leftarrow LINK(P)$



Вернуться к шагу *A*2.

***A*2.** [Сравнение *ABC*(*P*) и *ABC*(*Q*).]

(*ABC*(*P*) < *ABC*(*Q*)) = (*A*(*P*) + *B*(*P*) + *C*(*P*) < *A*(*Q*) + *B*(*Q*) + *C*(*Q*) **или** *A*(*P*) + *B*(*P*) + *C*(*P*) = *A*(*Q*) + *B*(*Q*) + *C*(*Q*) **и** (*A*(*P*) < *A*(*Q*) **или** *A*(*P*) = *A*(*Q*) **и** *B*(*P*) < *B*(*Q*))) = (0 + 1 + 0 < 0 + 1 + 0 **или** 0 + 1 + 0 = 0 + 1 + 0 **и** (0 < 0 **или** 0 = 0 **и** 1 < 1)) = (1 < 1 **или** 1 = 1 **и** (ЛОЖЬ **или** ИСТИНА **и** ЛОЖЬ)) = (ЛОЖЬ **или** ИСТИНА **и** (ЛОЖЬ **или** ЛОЖЬ)) = (ЛОЖЬ **или** ИСТИНА **и** ЛОЖЬ) = (ЛОЖЬ **или** ЛОЖЬ) = ЛОЖЬ

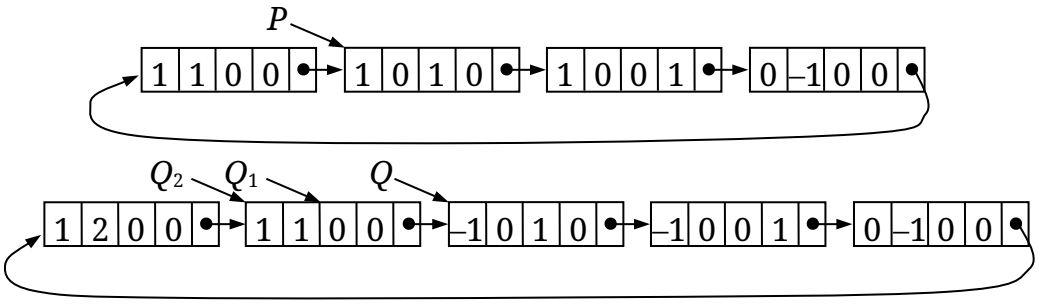
(*ABC*(*P*) = *ABC*(*Q*)) = (*A*(*P*) = *A*(*Q*) **и** *B*(*P*) = *B*(*Q*) **и** *C*(*P*) = *C*(*Q*)) = (0 = 0 **и** 1 = 1 **и** 0 = 0) = (ИСТИНА **и** ИСТИНА **и** ИСТИНА) = (ИСТИНА **и** ИСТИНА) = ИСТИНА, следовательно перейти к шагу *A*3.

A3. [Сложение коэффициентов.]

Найдены два члена с равными степенями.

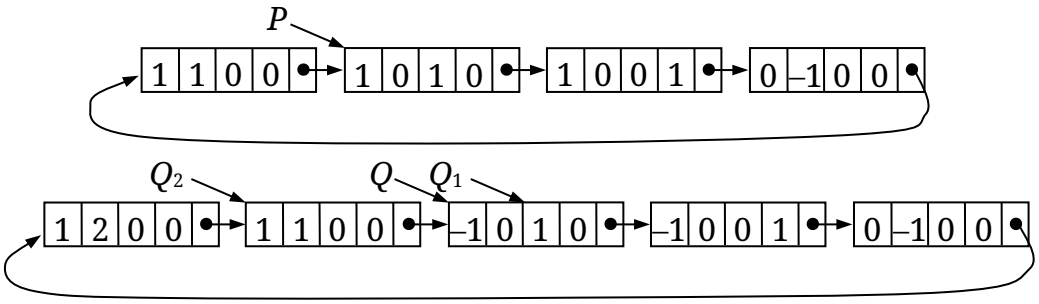
(*ABC*(*P*) < 0) = (010 < 0) = ЛОЖЬ

COEF(*Q*) ← *COEF*(*Q*) + *COEF*(*P*) = –2 + 1 = –1

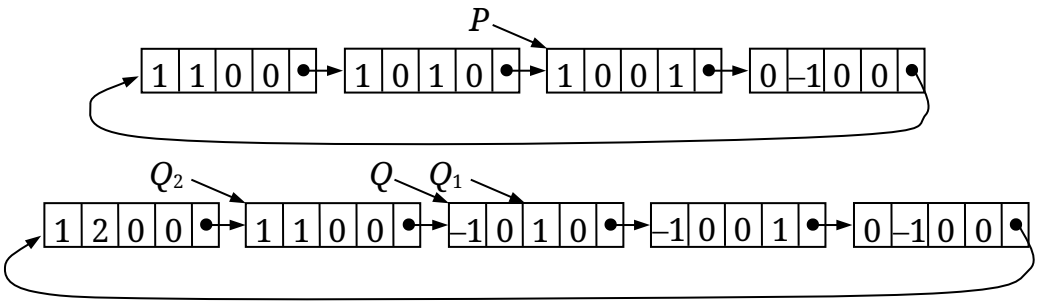


(*COEF*(*Q*) = 0) = (–1 = 0) = ЛОЖЬ

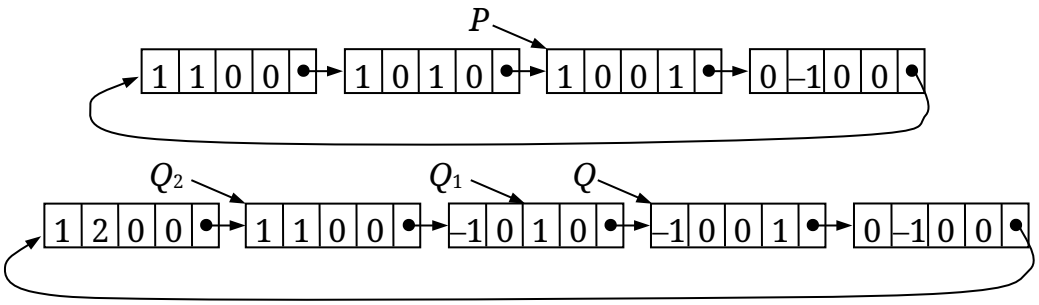
Q_1 ← *Q*,



P ← *LINK*(*P*),



Q ← *LINK*(*Q*)



Вернуться к шагу *A*2.

***A*2.** [Сравнение *ABC*(*P*) и *ABC*(*Q*).]

(*ABC*(*P*) < *ABC*(*Q*)) = (*A*(*P*) + *B*(*P*) + *C*(*P*) < *A*(*Q*) + *B*(*Q*) + *C*(*Q*) **или** *A*(*P*) + *B*(*P*) + *C*(*P*) = *A*(*Q*) + *B*(*Q*) + *C*(*Q*) **и** (*A*(*P*) < *A*(*Q*) **или** *A*(*P*) = *A*(*Q*) **и** *B*(*P*) < *B*(*Q*))) = (0 + 0 + 1 < 0 + 0 + 1 **или** 0 + 0 + 1 = 0 + 0 + 1 **и** (0 < 0 **или** 0 = 0 **и** 0 < 0)) = (1 < 1 **или** 1 = 1 **и** (ЛОЖЬ **или** ИСТИНА **и** ЛОЖЬ)) = (ЛОЖЬ **или** ИСТИНА **и** (ЛОЖЬ **или** ЛОЖЬ)) = (ЛОЖЬ **или** ИСТИНА **и** ЛОЖЬ) = (ЛОЖЬ **или** ЛОЖЬ) = ЛОЖЬ

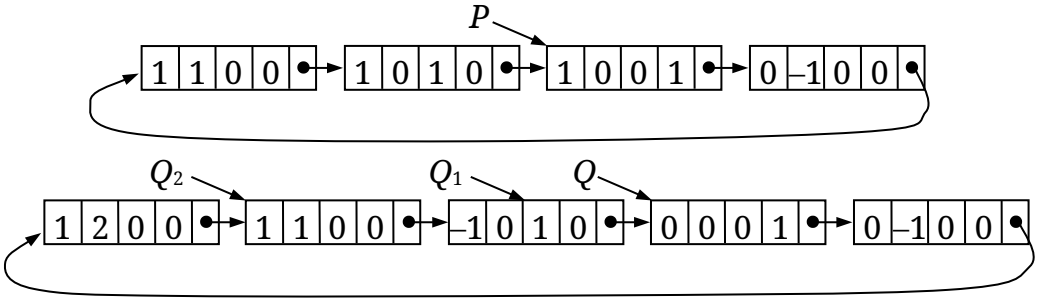
(*ABC*(*P*) = *ABC*(*Q*)) = (*A*(*P*) = *A*(*Q*) **и** *B*(*P*) = *B*(*Q*) **и** *C*(*P*) = *C*(*Q*)) = (0 = 0 **и** 0 = 0 **и** 1 = 1) = (ИСТИНА **и** ИСТИНА **и** ИСТИНА) = (ИСТИНА **и** ИСТИНА) = ИСТИНА, следовательно перейти к шагу *A*3.

***A*3.** [Сложение коэффициентов.]

Найдены два члена с равными степенями.

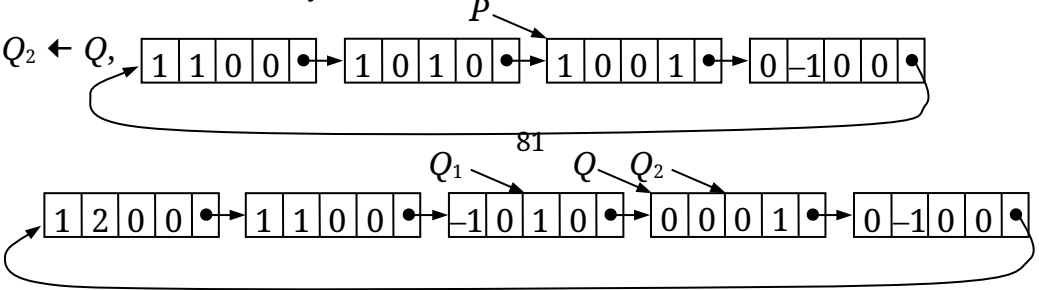
(*ABC*(*P*) < 0) = (001 < 0) = ЛОЖЬ

COEF(*Q*) ← *COEF*(*Q*) + *COEF*(*P*) = –1 + 1 = 0



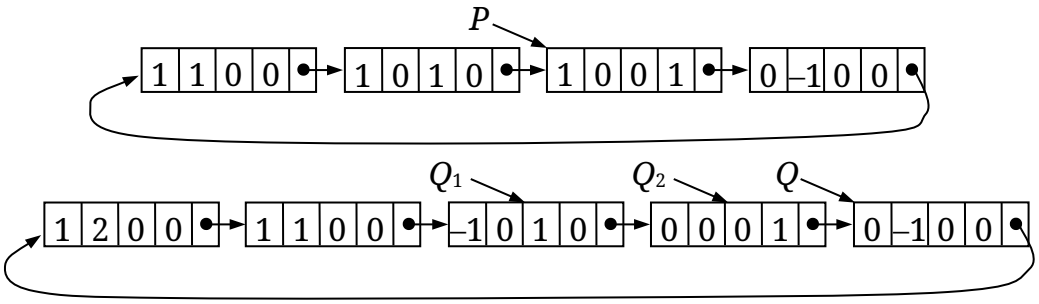
(*COEF*(*Q*) = 0) = (0 = 0) = ИСТИНА, следовательно перейти к шагу *A*4.

***A*4.** [Исключение нулевого члена.]

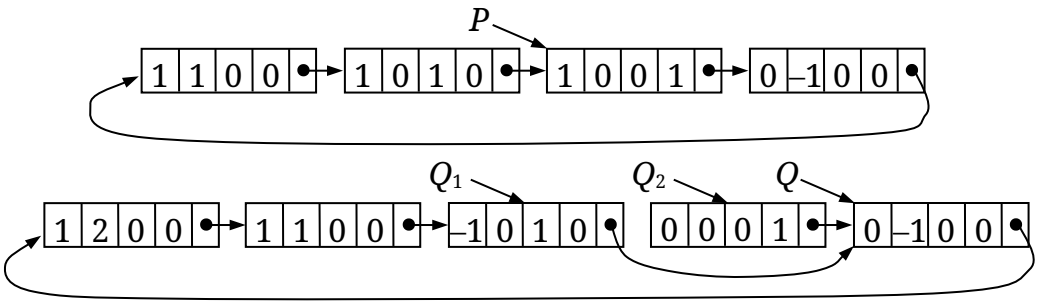


Двойное присваивание $LINK(Q_1) \leftarrow Q \leftarrow LINK(Q)$ равносильно последовательному выполнению присваиваний $Q \leftarrow LINK(Q)$ и $LINK(Q_1) \leftarrow Q$:

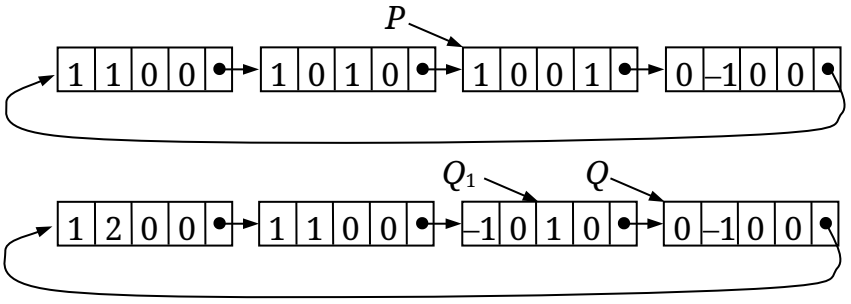
$Q \leftarrow LINK(Q)$



$LINK(Q_1) \leftarrow Q$

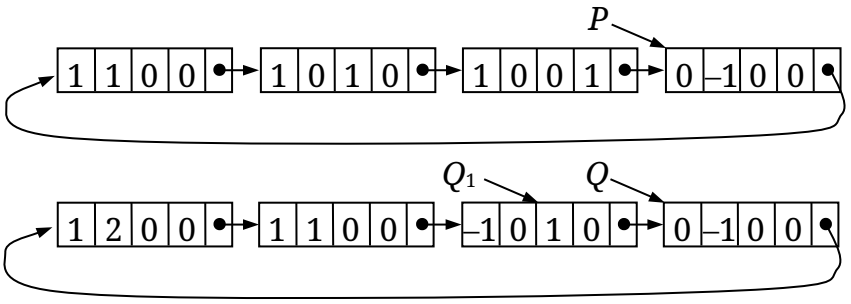


$AVAIL \Leftarrow Q_2$.



Нулевой член, образовавшийся на шаге *A*3, исключён из многочлена *Q*.

P ← *LINK*(*P*)



Вернуться к шагу *A*2.

***A*2.** [Сравнение *ABC*(*P*) и *ABC*(*Q*).]

(*ABC*(*P*) < *ABC*(*Q*)) = (*A*(*P*) + *B*(*P*) + *C*(*P*) < *A*(*Q*) + *B*(*Q*) + *C*(*Q*) **или** *A*(*P*) + *B*(*P*) + *C*(*P*) = *A*(*Q*) + *B*(*Q*) + *C*(*Q*) **и** (*A*(*P*) < *A*(*Q*) **или** *A*(*P*) = *A*(*Q*) **и** *B*(*P*) < *B*(*Q*))) = (–1 + 0 + 0 < –1 + 0 + 0 **или** –1 + 0 + 0 = –1 + 0 + 0 **и** (–1 < –1 **или** –1 = –1 **и** 0 < 0)) = (–1 < –1 **или** –1 = –1 **и** (ЛОЖЬ **или** ИСТИНА **и** ЛОЖЬ)) = (ЛОЖЬ **или** ИСТИНА **и** (ЛОЖЬ **или** ЛОЖЬ)) = (ЛОЖЬ **или** ИСТИНА **и** ЛОЖЬ) = (ЛОЖЬ **или** ЛОЖЬ) = ЛОЖЬ

(*ABC*(*P*) = *ABC*(*Q*)) = (*A*(*P*) = *A*(*Q*) **и** *B*(*P*) = *B*(*Q*) **и** *C*(*P*) = *C*(*Q*)) = (–1 = –1 **и** 0 = 0 **и** 0 = 0) = (ИСТИНА **и** ИСТИНА **и** ИСТИНА) = (ИСТИНА **и** ИСТИНА) = ИСТИНА, следовательно перейти к шагу *A*3.

***A*3.** [Сложение коэффициентов.]

Найдены два члена с равными степенями.

(*ABC*(*P*) < 0) = (–100 < 0) = ИСТИНА, следовательно конец алгоритма.

Ответ: Прибавление многочлена ($x + y + z$) к многочлену ($x^2 - 2y - z$) влечёт последовательное выполнение шагов *A*1, *A*2, *A*2, *A*5, *A*2, *A*3, *A*2, *A*3, *A*4, *A*2, *A*3 алгоритма и даёт итоговый многочлен ($x^2 + x - y$).

Лабораторная работа № 5

ОСЕВОЙ ШАГ В РАЗРЕЖЕННОЙ МАТРИЦЕ

Цель. Знакомство с понятием ортогональных связанных списков и алгоритмом выполнения осевого шага в разреженной матрице, представленной ортогональными связанными списками.

Задание. Выполнить по алгоритму *S* осевой шаг в разреженной матрице с заданными ненулевыми элементами и осевым элементом. Определить последовательность выполненных шагов алгоритма и конечную матрицу. Варианты конкретных исходных данных представлены в табл. 5.

Основные положения

Массивы (двумерные и более высокой размерности) являются одним из простейших обобщений линейных списков. Рассмотрим в качестве примера матрицу:

$$\begin{pmatrix} A[0,0] & A[0,1] & \dots & A[0,n] \\ A[1,0] & A[1,1] & \dots & A[1,n] \\ \dots & \dots & \dots & \dots \\ A[m,0] & A[m,1] & \dots & A[m,n] \end{pmatrix}.$$

Таблица 5

Вариант	Ненулевые элементы						Осевой элемент
1	$m[1][4]=5$	$m[2][4]=-60$	$m[2][2]=20$	$m[4][4]=-30$	$m[4][2]=10$	$m[4][1]=50$	$m[2][4]$
2	$m[1][4]=5$	$m[2][4]=-60$	$m[2][2]=20$	$m[4][4]=-30$	$m[4][2]=10$	$m[4][1]=50$	$m[4][2]$
3	$m[1][4]=-30$	$m[1][2]=-60$	$m[1][1]=5$	$m[3][4]=10$	$m[3][2]=20$	$m[4][4]=50$	$m[1][2]$
4	$m[1][4]=-30$	$m[1][2]=-60$	$m[1][1]=5$	$m[3][4]=10$	$m[3][2]=20$	$m[4][4]=50$	$m[3][4]$
5	$m[1][4]=50$	$m[1][3]=10$	$m[1][1]=-30$	$m[3][3]=20$	$m[3][1]=-60$	$m[4][1]=5$	$m[1][3]$
6	$m[1][4]=50$	$m[1][3]=10$	$m[1][1]=-30$	$m[3][3]=20$	$m[3][1]=-60$	$m[4][1]=5$	$m[3][3]$
7	$m[1][3]=20$	$m[1][1]=10$	$m[2][1]=50$	$m[4][4]=5$	$m[4][3]=-60$	$m[4][1]=-30$	$m[1][1]$
8	$m[1][3]=20$	$m[1][1]=10$	$m[2][1]=50$	$m[4][4]=5$	$m[4][3]=-60$	$m[4][1]=-30$	$m[4][3]$
9	$m[1][4]=5$	$m[2][4]=-60$	$m[2][1]=20$	$m[4][4]=-30$	$m[4][2]=50$	$m[4][1]=10$	$m[2][4]$
10	$m[1][4]=5$	$m[2][4]=-60$	$m[2][1]=20$	$m[4][4]=-30$	$m[4][2]=50$	$m[4][1]=10$	$m[4][1]$
11	$m[1][4]=-30$	$m[1][2]=-60$	$m[1][1]=5$	$m[3][4]=50$	$m[4][4]=10$	$m[4][2]=20$	$m[4][2]$
12	$m[1][4]=-30$	$m[1][2]=-60$	$m[1][1]=5$	$m[3][4]=50$	$m[4][4]=10$	$m[4][2]=20$	$m[4][4]$
13	$m[1][4]=10$	$m[1][3]=50$	$m[1][1]=-30$	$m[3][4]=20$	$m[3][1]=-60$	$m[4][1]=5$	$m[1][4]$
14	$m[1][4]=10$	$m[1][3]=50$	$m[1][1]=-30$	$m[3][4]=20$	$m[3][1]=-60$	$m[4][1]=5$	$m[3][1]$
15	$m[2][3]=20$	$m[2][1]=10$	$m[3][1]=50$	$m[4][4]=5$	$m[4][3]=-60$	$m[4][1]=-30$	$m[2][1]$
16	$m[2][3]=20$	$m[2][1]=10$	$m[3][1]=50$	$m[4][4]=5$	$m[4][3]=-60$	$m[4][1]=-30$	$m[4][3]$
17	$m[1][4]=5$	$m[2][4]=-60$	$m[2][2]=20$	$m[4][4]=-30$	$m[4][3]=50$	$m[4][2]=10$	$m[2][2]$
18	$m[1][4]=5$	$m[2][4]=-60$	$m[2][2]=20$	$m[4][4]=-30$	$m[4][3]=50$	$m[4][2]=10$	$m[4][2]$
19	$m[1][4]=-30$	$m[1][2]=-60$	$m[1][1]=5$	$m[2][4]=50$	$m[3][4]=10$	$m[3][2]=20$	$m[1][2]$
20	$m[1][4]=-30$	$m[1][2]=-60$	$m[1][1]=5$	$m[2][4]=50$	$m[3][4]=10$	$m[3][2]=20$	$m[3][4]$
21	$m[1][3]=10$	$m[1][2]=50$	$m[1][1]=-30$	$m[3][3]=20$	$m[3][1]=-60$	$m[4][1]=5$	$m[1][3]$

22	m[1][3]=10	m[1][2]=50	m[1] [1]=−30	m[3][3]=20	m[3] [1]=−60	m[4][1]=5	m[3][1]
23	m[1][4]=5	m[1] [3]=−60	m[1] [1]=−30	m[2][3]=20	m[2][1]=10	m[4][1]=50	m[2][1]
24	m[1][4]=5	m[1] [3]=−60	m[1] [1]=−30	m[2][3]=20	m[2][1]=10	m[4][1]=50	m[2][3]
25	m[1][1]=5	m[2][2]=20	m[2] [1]=−60	m[4][4]=50	m[4][2]=10	m[4] [1]=−30	m[2][1]
26	m[1][1]=5	m[2][2]=20	m[2] [1]=−60	m[4][4]=50	m[4][2]=10	m[4] [1]=−30	m[4][2]
27	m[1][4]=50	m[3][4]=10	m[3][2]=20	m[4] [4]=−30	m[4] [2]=−60	m[4][1]=5	m[3][4]
28	m[1][4]=50	m[3][4]=10	m[3][2]=20	m[4] [4]=−30	m[4] [2]=−60	m[4][1]=5	m[4][2]
29	m[1] [4]=−30	m[1][3]=10	m[1][1]=50	m[3] [4]=−60	m[3][3]=20	m[4][4]=5	m[1][3]
30	m[1] [4]=−30	m[1][3]=10	m[1][1]=50	m[3] [4]=−60	m[3][3]=20	m[4][4]=5	m[3][3]

В таком двумерном массиве некоторый узел A [j, k] принадлежит двум линейным спискам: списку j-й строки A [j, 0], A [j, 1], …, A [j, n] и списку k-го столбца A [0, k], A [1, k], …, A [m, k]. Эти списки ортогональных строк и столбцов, по существу, и определяют двумерную структуру матрицы.

Рассмотрим последовательное распределение памяти при хранении массивов.

Если массив хранится в последовательных ячейках памяти, то память обычно распределяется так, что:

$$LOC\,(A\,[\,j,\,k]) = A_0 + A_1 j + A_2 k, \qquad (26)$$

где A_0, A_1 и A_2 – константы. Это означает, что при любом изменении j или k легко вычислить адрес узла A [j, k].

Самым естественным и наиболее часто используемым способом распределения памяти является способ, при котором массив располагается в памяти в лексикографическом порядке индексов: A [0, 0], A [0, 1], …, A [0, n], A [1, 0], A [1, 1], …, A [1, n], …, A [m, 0], A [m, 1], …, A [m, n].

В общем случае, если задан k-мерный массив с элементами A [I_1, I_2, ..., I_k] длины c и $0 \le I_1 \le d_1$, $0 \le I_2 \le d_2$, ..., $0 \le I_k \le d_k$, то можно хранить его в памяти так, что:

$$LOC\,(A\,[I_1, I_2, \ldots, I_k]) = LOC\,(A\,[0, 0, \ldots, 0]) +$$

$$+ c\,(d_2 + 1) \ldots (d_k + 1)\,I_1 + \ldots + c\,(d_k + 1)\,I_{k-1} + c\,I_k =$$

$$= LOC\,(A\,[0, 0, \ldots, 0]) + \sum_{1 \le r \le k} a_r I_r, \qquad (27)$$

где $a_r = c \prod_{r<s\le k} (d_s + 1)$.

Например, двумерный массив для $d_1 = 3$, $d_2 = 3$ и $c = 2$ байта будет размещён в памяти следующим образом:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A[0,0]		A[0,1]		A[0,2]		A[0,3]		A[1,0]		A[1,1]		A[1,2]		A[1,3]	

16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
A[2,0]		A[2,1]		A[2,2]		A[2,3]		A[3,0]		A[3,1]		A[3,2]		A[3,3]	

В соответствии с формулой (27) адрес его некоторого элемента A [I_1, I_2] можно определить по выражению:

$$LOC\,(A[I_1, I_2]) = LOC\,(A[0, 0]) + c\,(d_2 + 1)\,I_1 + c\,I_2 =$$

$$= LOC\,(A[0, 0]) + 2\,(3 + 1)\,I_1 + 2\,I_2 =$$

$$= LOC\,(A[0, 0]) + 8\,I_1 + 2\,I_2.$$

Пусть $I_1 = 3$ и $I_2 = 1$, тогда:

$$LOC\,(A[3, 1]) = LOC\,(A[0, 0]) + 8 \cdot 3 + 2 \cdot 1 =$$

$$= LOC\,(A[0, 0]) + 26 = 0 + 26 = 26.$$

Приведённый выше метод хорош для хранения массивов, имеющих *полную* прямоугольную структуру. Однако во многих случаях приходится иметь дело с *треугольной матрицей* элементов $A\,[j, k]$ для $0 \le k \le j \le n$:

$$\begin{pmatrix} A[0,\ 0] & & & \\ A[1,\ 0] & A[1,\ 1] & & \\ \ldots & \ldots & & \\ A[n,\ 0] & A[n,\ 1] & \ldots & A[n,\ n] \end{pmatrix}.$$

Такую матрицу можно использовать, когда известно, что все другие элементы матрицы равны нулю или $A\,[j, k] = A\,[k, j]$. Поэтому необходимо и достаточно хранить в памяти немного больше половины значений: $\frac{1}{2}(n+1)(n+2)$. Если хранить эти элементы в последовательных ячейках памяти, то вместо линейного распределения (26) можно использовать распределение в форме:

$$LOC\,(A[j, k]) = A_0 + c\,f_1(j) + c\,f_2(k),$$

где $f_1(j) = \frac{j(j+1)}{2}$, $f_2(k) = k$. Следовательно, получаем довольно простую формулу:

$$LOC\,(A[j, k]) = LOC\,(A\,[0, 0]) + c\,\frac{j(j+1)}{2} + c\,k.$$

Например, для треугольной матрицы вида

$$\begin{pmatrix} A[0,\ 0] & & & \\ A[1,\ 0] & A[1,\ 1] & & \\ A[2,\ 0] & A[2,\ 1] & A[2,\ 2] & \\ A[3,\ 0] & A[3,\ 1] & A[3,\ 2] & A[3,\ 3] \end{pmatrix}$$

при c = 2 получаем такое распределение памяти:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
A[0, 0]		A[1, 0]		A[1, 1]		A[2, 0]		A[2, 1]		A[2, 2]		A[3, 0]		A[3, 1]		A[3, 2]		A[3, 3]	

и адрес элемента $A[3, 2]$:

$$LOC(A[3, 2]) = LOC(A[0, 0]) + 2\frac{3(3+1)}{2} + 2 \cdot 2 = 0 + 12 + 4 = 16.$$

В некоторых случаях, когда имеются две треугольные матрицы одинакового размера, их можно объединить в одну. Пусть треугольные матрицы $A\ [j, k]$ и $B\ [j, k]$ определены для $0 \le k \le j \le n$. Тогда их можно объединить в матрицу C так, что элемент $C[j, k]$ будет определён для $0 \le j \le n$,

$0 \le k \le n + 1$ и

$$A[j, k] = C[j, k], \quad B[j, k] = C[k, j + 1].$$

Таким образом, две треугольные матрицы плотно упаковываются вместе, занимая $(n + 1)(n + 2)$ ячеек:

$$\begin{pmatrix} C[0,0] & C[0,1] & C[0,2] & \dots & C[0,n+1] \\ C[1,0] & C[1,1] & C[1,2] & \dots & C[1,n+1] \\ \dots & \dots & \dots & \dots & \dots \\ C[n,0] & C[n,1] & C[n,2] & \dots & C[n,n+1] \end{pmatrix} \equiv \begin{pmatrix} A[0,0] & B[0,0] & B[1,0] & \dots & B[n,0] \\ A[1,0] & A[1,1] & B[1,1] & \dots & B[n,1] \\ \dots & \dots & \dots & \dots & \dots \\ A[n,0] & A[n,1] & A[n,2] & \dots & B[n,n] \end{pmatrix}$$

и может быть использована линейная адресация вида (26).

Рассмотрим связанное распределение памяти при хранении массивов.

Связанное распределение памяти вполне естественно для многомерных массивов информации. При этом в общем случае любой элемент массива должен содержать k полей связи, по одному на каждый из списков, в которые входит данный элемент. Связанная память обычно используется в тех случаях, когда массивы по своей форме не являются строго прямоугольными.

Использование связанного распределения рассмотрим подробно на примере *разреженных матриц* – матриц, в которых большинство элементов равно нулю. Идея заключается в том, что для экономии памяти не отводить в ней место под нулевые элементы.

Обсудим представление, которое состоит из циклически связанных *ортогональных* списков для каждой строки и каждого столбца матрицы. Пусть узлы списка имеют формат:

LEFT	*UP*	
RO W	*CO L*	*VA L*

где *ROW* и *COL* – индексы соответственно строки и столбца; *VAL* – значение элемента матрицы; *LEFT* и *UP* – связи со следующими ненулевыми элементами соответственно слева в строке и сверху в столбце. Для каждой *i*-й строки и *j*-го столбца имеются специальные головные узлы списков *BASEROW*[*i*] и *BASECOL*[*j*] такие, что:

$$COL(LOC(BASEROW[i])) < 0 \quad \text{и} \quad ROW(LOC(BASECOL[\,j\,])) < 0.$$

Связь *LEFT* в *BASEROW*[*i*] указывает на самый правый узел в *i*-й строке и связь *UP* в *BASECOL*[*j*] – на самый нижний узел в *j*-м столбце.

Например, матрица

$$\begin{pmatrix} 50 & 0 & 0 & 0 \\ 10 & 0 & 20 & 0 \\ 0 & 0 & 0 & 0 \\ -30 & 0 & -60 & 5 \end{pmatrix} \tag{28}$$

будет представлена так, как показано на рис. 7.

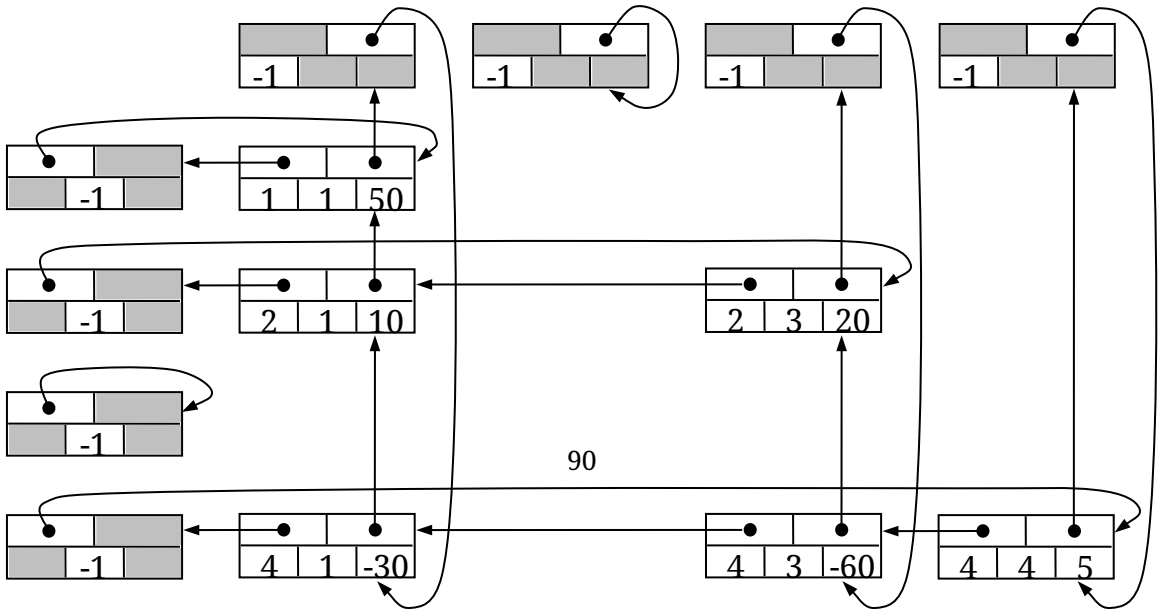


Рис. 7. Машинное представление разреженной матрицы

При последовательном распределении памяти эта матрица заняла бы больше слов, а при увеличении её размеров экономия становится ещё более существенной. При этом время, расходуемое на доступ к произвольному элементу матрицы, остаётся в разумных пределах, поскольку в каждой строке и каждом столбце встречается немного элементов. Кроме того, в большинстве матричных алгоритмов осуществляется последовательное прохождение матрицы и поэтому связанное представление приводит к незначительной потере скорости работы.

В качестве характерного примера рассмотрим операцию *осевого шага* для разреженных матриц, которая играет важную роль в алгоритмах решения систем линейных уравнений, обращения матриц и линейного программирования (симплекс-метод):

	Осевой столбец	Любой другой столбец			Осевой столбец	Любой другой столбец
Осевая строка			Осевой шаг			
Любая другая строка	a	b	$\rightarrow$		$\frac{1}{a}$	$\frac{b}{a}$
	c	d			$-\frac{c}{a}$	$d-\frac{bc}{a}$

(29)

Так, осевой шаг применительно к матрице (28) с осевым элементом 10 приводит к следующей матрице:

$$\begin{pmatrix} -5 & 0 & -100 & 0 \\ 0.1 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 \\ 3 & 0 & 0 & 5 \end{pmatrix}.$$

Отметим два случая в реализации осевого шага:

1) если в (29) $b \neq 0$, $c \neq 0$ и $d = 0$, то в представлении разреженной матрицы нет узла для образующегося ненулевого элемента и поэтому необходимо включить новый узел;

2) если $b \neq 0$, $c \neq 0$, $d \neq 0$ и $d - \frac{bc}{a} = 0$, то необходимо исключить имеющийся узел.

Алгоритм *S*. (*Осевой шаг в разреженной матрице.*)

Для эффективного выполнения операций включения и исключения узлов вводится таблица указателей *PTR*[*j*], по одному указателю на каждый столбец матрицы. Благодаря этим переменным обеспечивается возможность изменения связей узлов как в горизонтальном, так и в вертикальном измерениях. Алгоритм обрабатывает строки матрицы последовательно снизу вверх, а переменная *PIVOT* указывает на осевой элемент.

***S*1.** [Начальная установка.] $I0 \leftarrow ROW(PIVOT)$, $J0 \leftarrow COL(PIVOT)$, $ALPHA \leftarrow 1.0/VAL(PIVOT)$, $VAL(PIVOT) \leftarrow 1.0$, $P0 \leftarrow LOC(BASEROW[I0])$, $Q0 \leftarrow LOC(BASECOL[J0])$.

***S*2.** [Обработка осевой строки.] $P0 \leftarrow LEFT(P0)$, $J \leftarrow COL(P0)$. Если $J < 0$, то перейти к шагу *S*3 (осевая строка пройдена до конца); в противном случае $PTR[J] \leftarrow LOC(BASECOL[J])$, $VAL(P0) \leftarrow ALPHA \cdot VAL(P0)$ и повторить шаг *S*2.

***S*3.** [Поиск новой строки.] $Q0 \leftarrow UP(Q0)$, $I \leftarrow ROW(Q0)$. Если $I < 0$, то конец алгоритма. Если $I = I0$, то повторить шаг *S*3; в противном случае $P \leftarrow LOC(BASEROW[I])$, $P1 \leftarrow LEFT(P)$.

***S*4.** [Поиск нового столбца.] $P0 \leftarrow LEFT(P0)$, $J \leftarrow COL(P0)$. Если $J < 0$, то $VAL(Q0) \leftarrow -ALPHA \cdot VAL(Q0)$ и вернуться к *S*3. Если $J = J0$, то выполнить этот шаг сначала.

***S*5.** [Поиск элемента *I*, *J*.] Если $COL(P1) > J$, то $P \leftarrow P1$, $P1 \leftarrow LEFT(P)$ и повторить этот шаг сначала. Если $COL(P1) = J$, то перейти к *S*7; в противном случае перейти к следующему шагу.

***S*6.** [Включение элемента *I*, *J*.] Если $ROW(UP(PTR[J])) > I$, то $PTR[J] \leftarrow UP(PTR[J])$ и повторить этот шаг сначала; в противном случае $X \Leftarrow AVAIL$, $VAL(X) \leftarrow 0$, $ROW(X) \leftarrow I$, $COL(X) \leftarrow J$, $LEFT(X) \leftarrow P1$, $UP(X) \leftarrow UP(PTR[J])$, $LEFT(P) \leftarrow X$, $UP(PTR[J]) \leftarrow X$, $P1 \leftarrow X$, $PTR[J] \leftarrow X$.

***S*7.** [Осевая операция.] $VAL(P1) \leftarrow VAL(P1) - VAL(Q0) \cdot VAL(P0)$. Если $VAL(P1) = 0$, то перейти к следующему шагу; в противном случае $PTR[J] \leftarrow P1$, $P \leftarrow P1$, $P1 \leftarrow LEFT(P)$ и вернуться к *S*4.

***S*8.** [Исключение элемента *I*, *J*.] Если $UP(PTR[J]) \neq P1$, то $PTR[J] \leftarrow UP(PTR[J])$ и повторить этот шаг сначала; в противном случае $UP(PTR[J]) \leftarrow UP(P1)$, $LEFT(P) \leftarrow LEFT(P1)$, $AVAIL \Leftarrow P1$, $P1 \leftarrow LEFT(P)$. Вернуться к шагу *S*4. ■

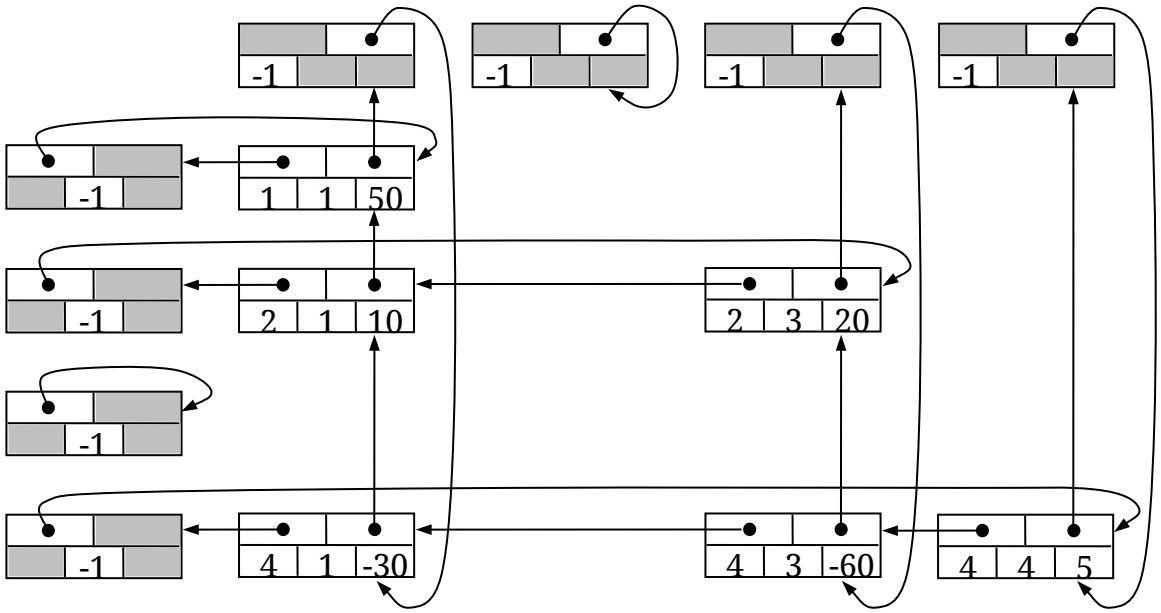
Заметим, что в узлах *BASEROW*[*I*] и *BASECOL*[*J*] используются только два поля. Поэтому для остальных полей можно не отводить место в памяти.

Время работы алгоритма *S* приблизительно пропорционально количеству элементов матрицы, которые изменяются при выполнении осевой операции.

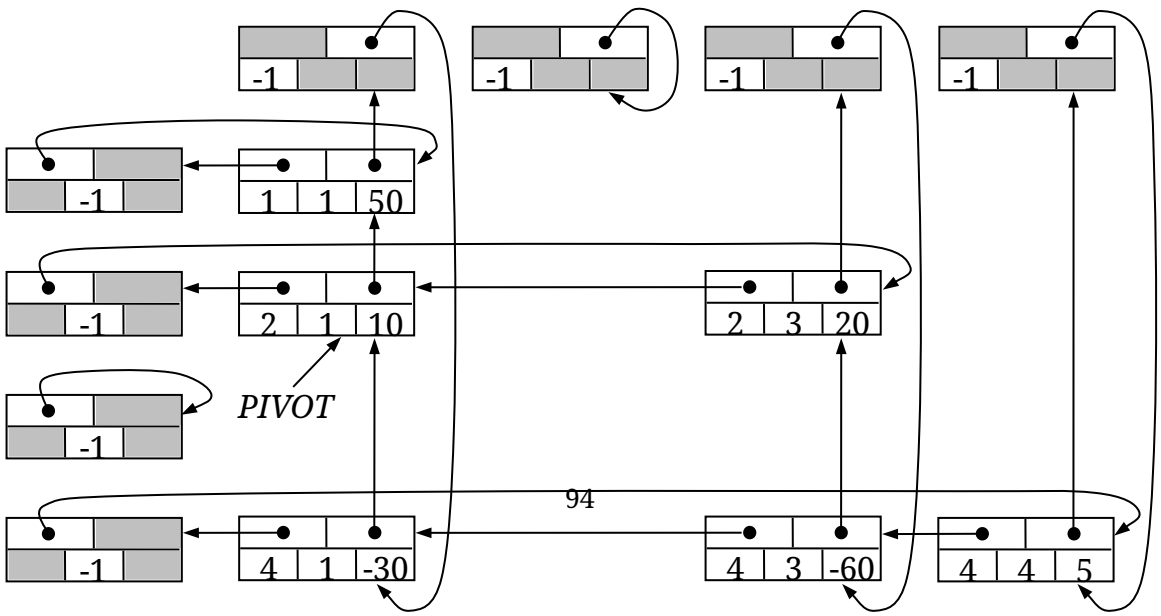
Пример выполнения задания

Задание. Выполнить по алгоритму *S* осевой шаг в разреженной матрице с заданными ненулевыми элементами $m[1][1] = 50$, $m[2][3] = 20$, $m[2][1] = 10$, $m[4][4] = 5$, $m[4][3] = -60$, $m[4][1] = -30$, если осевым элементом является $m[2][1]$. Определить последовательность выполненных шагов алгоритма и конечную матрицу.

Представление разреженной матрицы
связанными ортогональными списками



Для эффективного выполнения операций включения и исключения узлов вводится таблица указателей *PTR*[*j*], по одному указателю на каждый столбец матрицы. Благодаря этим переменным обеспечивается возможность изменения связей узлов как в горизонтальном, так и в вертикальном измерениях. Алгоритм обрабатывает строки матрицы последовательно снизу вверх, а переменная *PIVOT* указывает на осевой элемент.



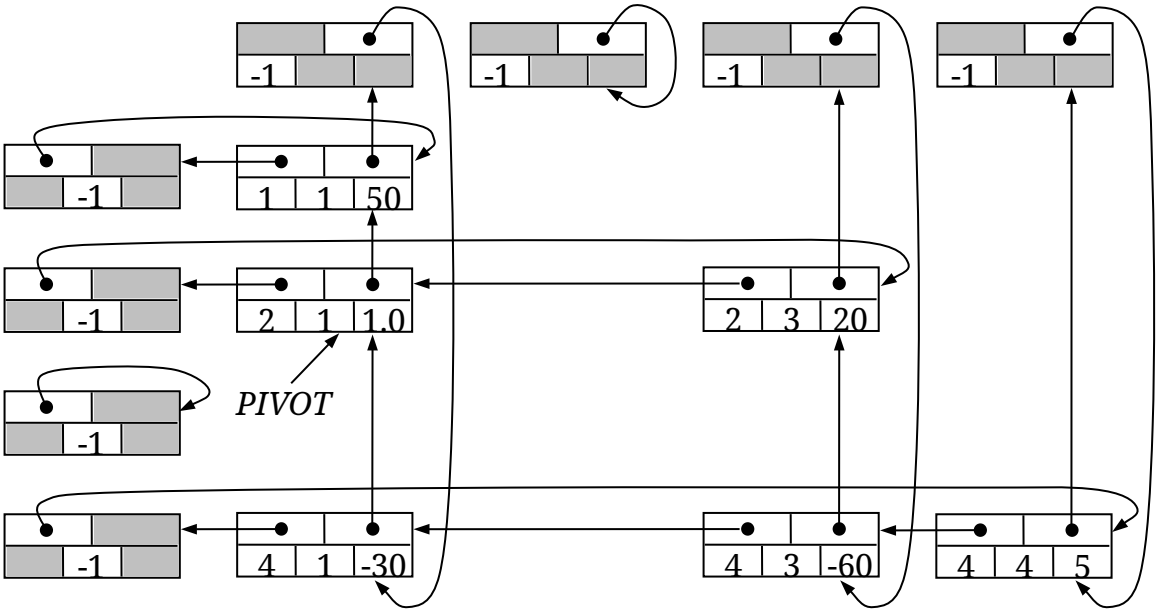
***S*1.** [Начальная установка.]

*I*0 ← *ROW*(*PIVOT*) = 2,

*J*0 ← *COL*(*PIVOT*) = 1,

ALPHA ← 1.0/*VAL*(*PIVOT*) = 1.0/10 = 0.1,

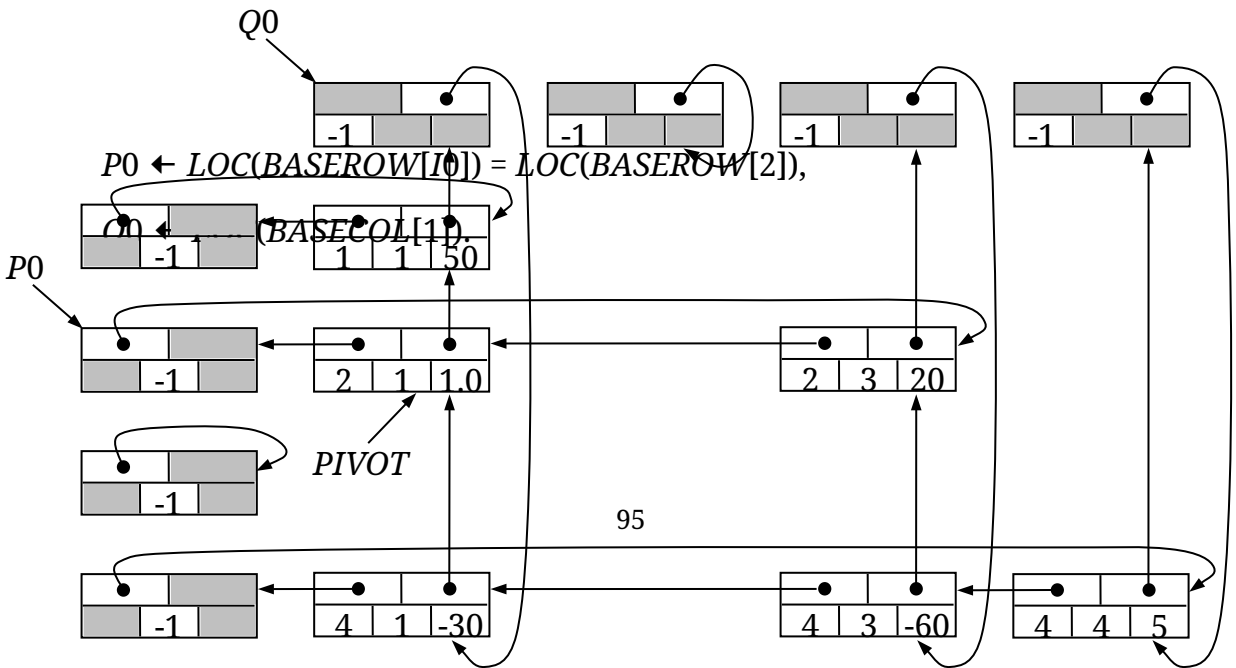
VAL(*PIVOT*) ← 1.0,



*Q*0

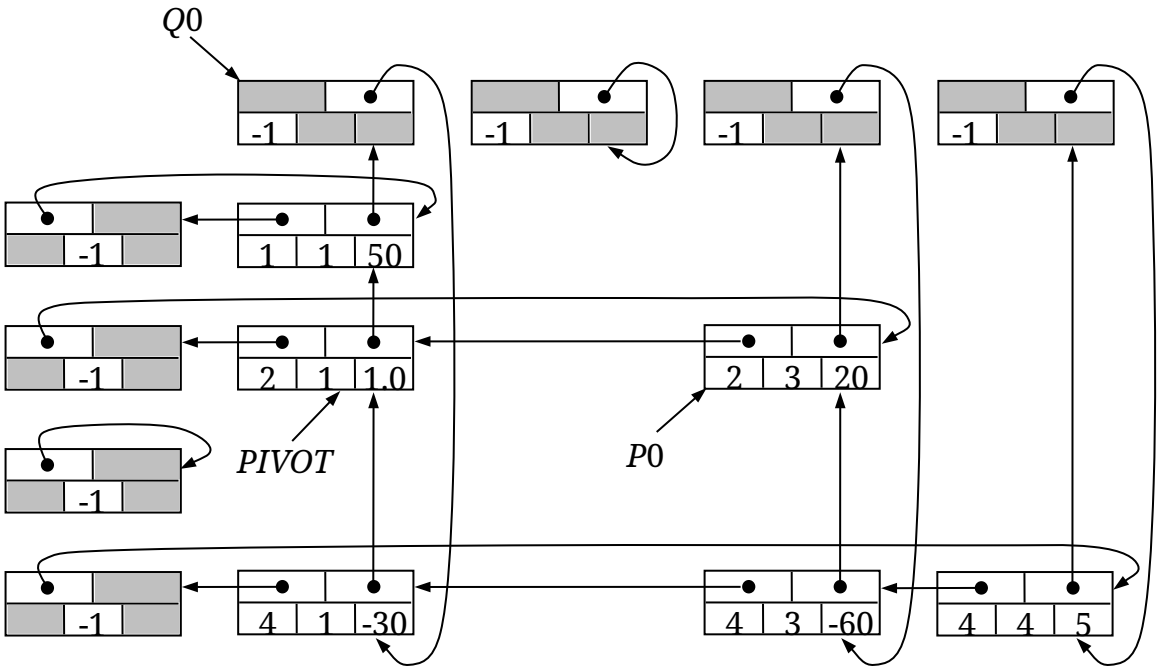
*P*0 ← *LOC*(*BASEROW*[*I*0]) = *LOC*(*BASEROW*[2]),

*Q*0 ← *LOC*(*BASECOL*[*J*0]).



S2. [Обработка осевой строки.]

$P0 \leftarrow LEFT(P0)$,

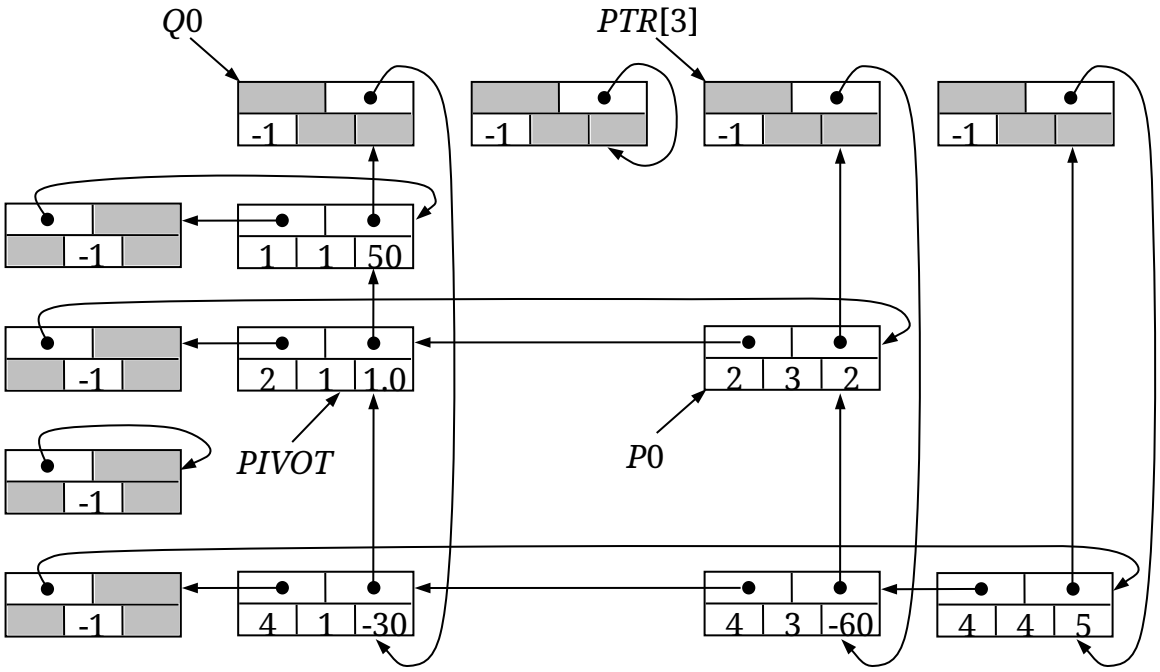


$J \leftarrow COL(P0) = 3$.

(J < 0) = (3 < 0) = ЛОЖЬ,

(PTR[J] ← LOC($BASECOL$[J])) = (PTR[3] ← LOC($BASECOL$[3])),

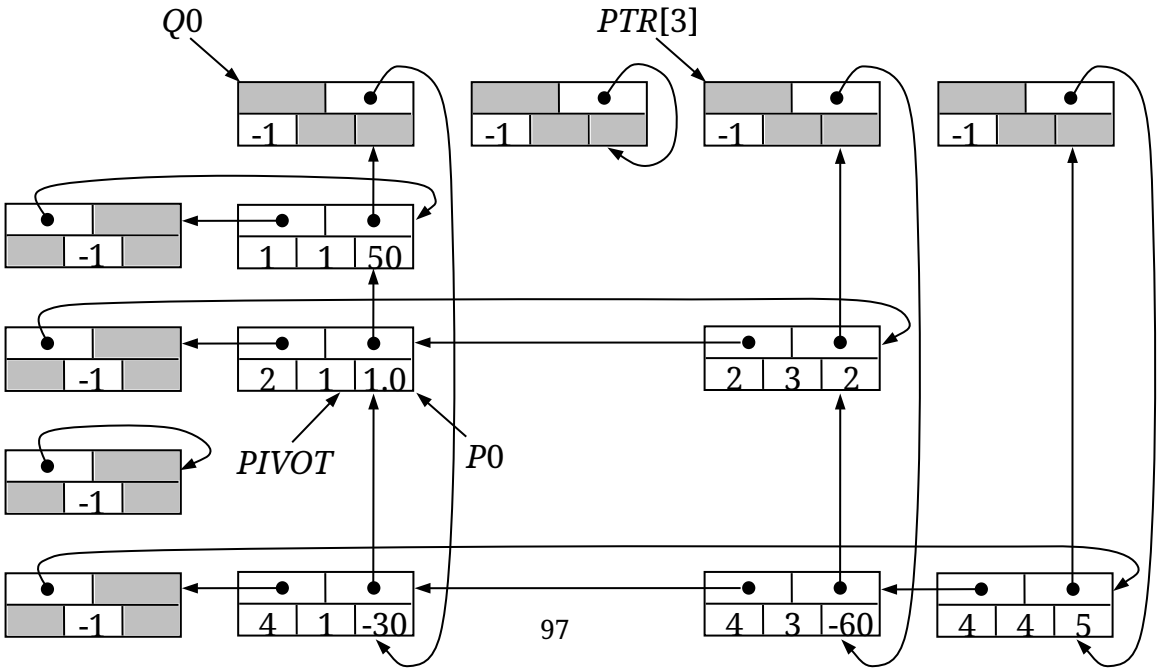
VAL(P0) ← $ALPHA$ • VAL(P0) = 0.1 • 20 = 2



Повторить шаг S2.

***S*2.** [Обработка осевой строки.]

P0 ← $LEFT$(P0),

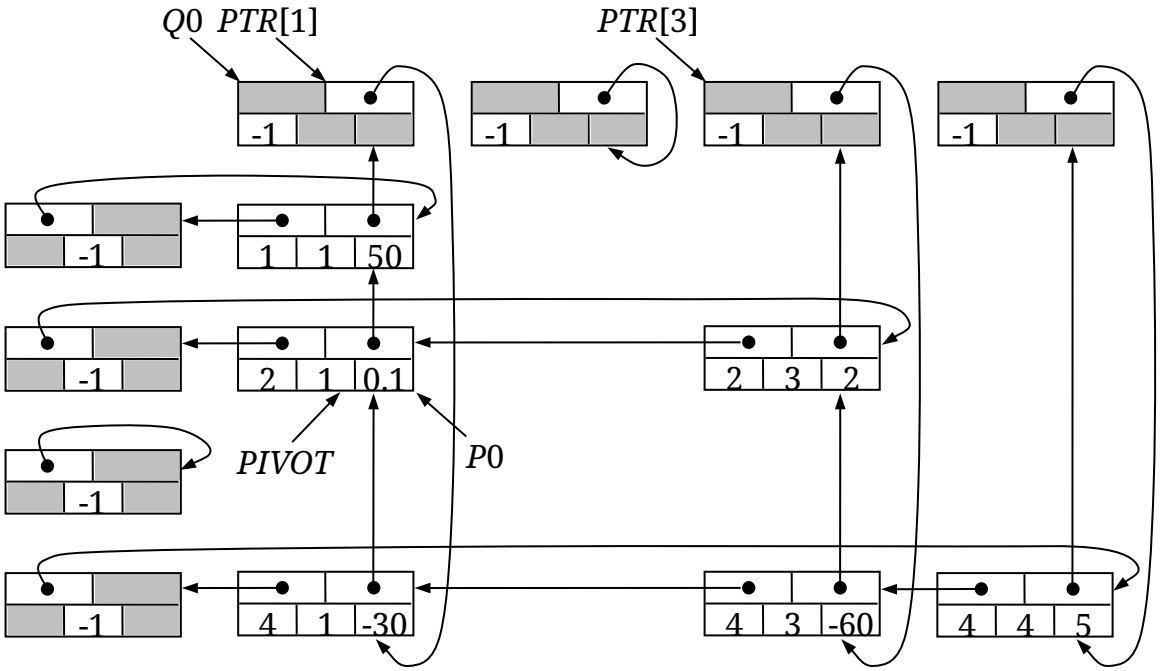


$J \leftarrow COL(P0) = 1$.

$(J < 0) = (1 < 0) =$ ЛОЖЬ,

$(PTR[J] \leftarrow LOC(BASECOL[J])) = (PTR[1] \leftarrow LOC(BASECOL[1]))$,

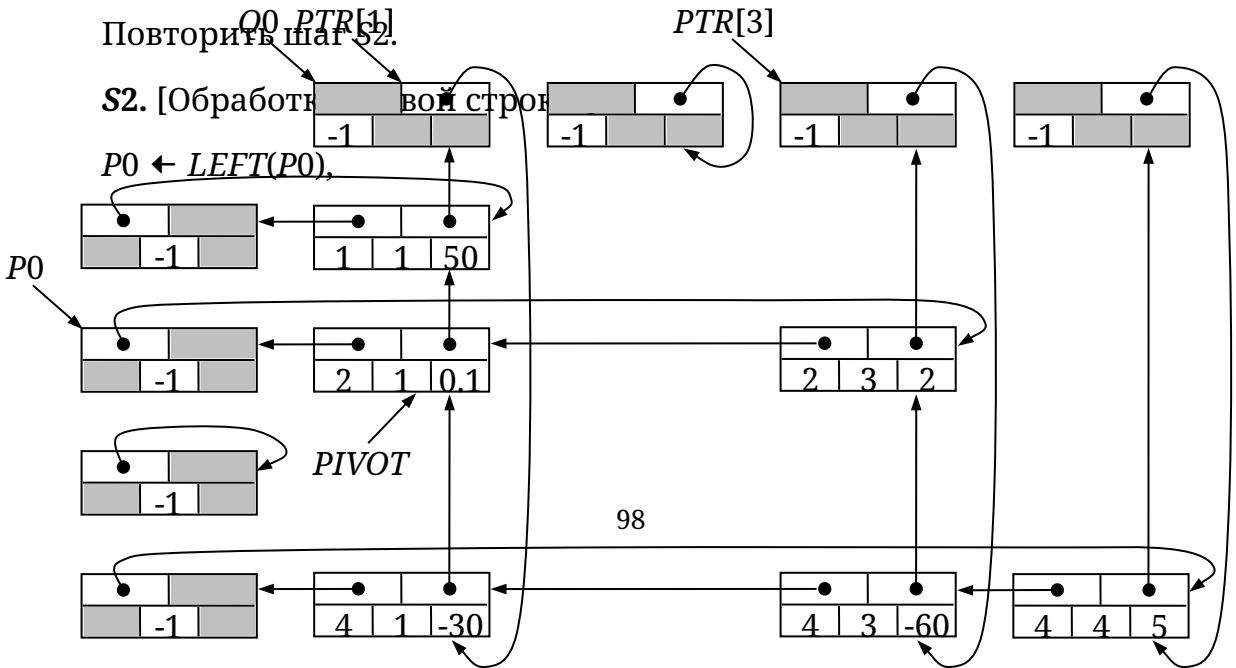
$VAL(P0) \leftarrow ALPHA \cdot VAL(P0) = 0.1 \cdot 1.0 = 0.1$



Повторить шаг S2.

S2. [Обработ вой строк

$P0 \leftarrow LEFT(P0)$,

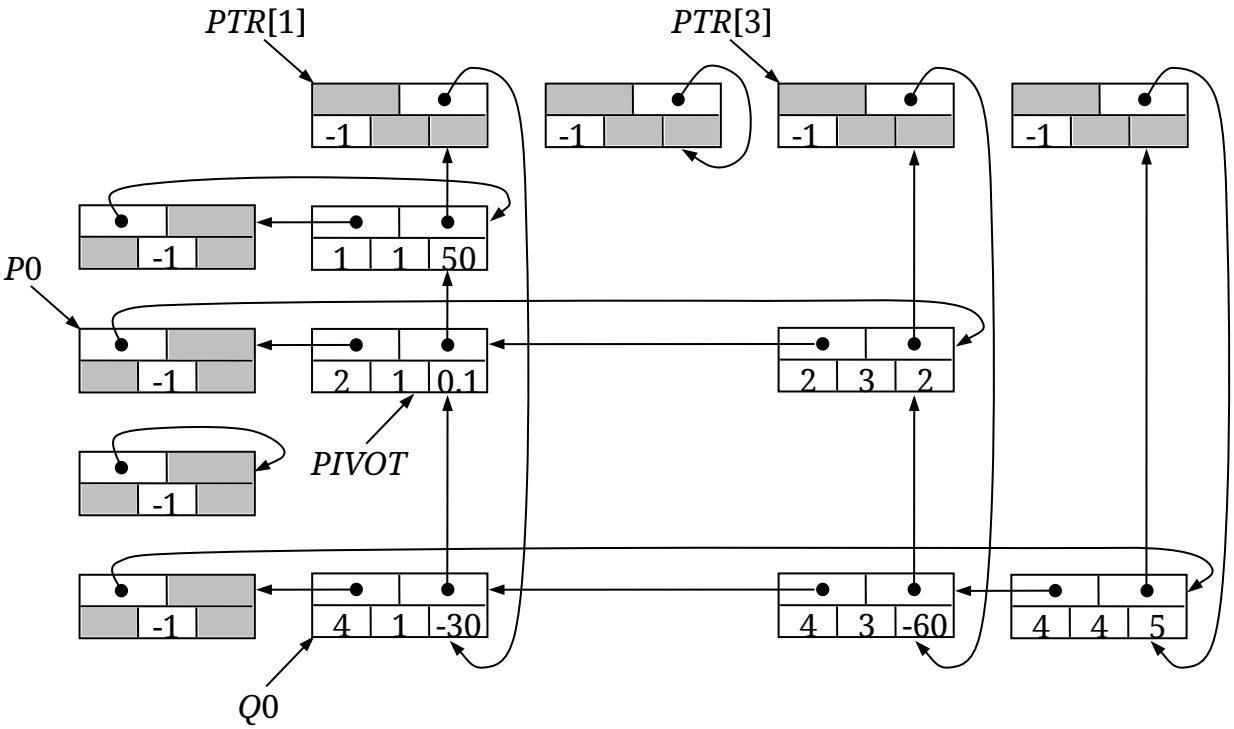


$J \leftarrow COL(P0) = -1$.

($J < 0$) = ($-1 < 0$) = ИСТИНА, следовательно перейти к шагу $S3$ (осевая строка пройдена до конца).

***S*3.** [Поиск новой строки.]

$Q0 \leftarrow UP(Q0)$,

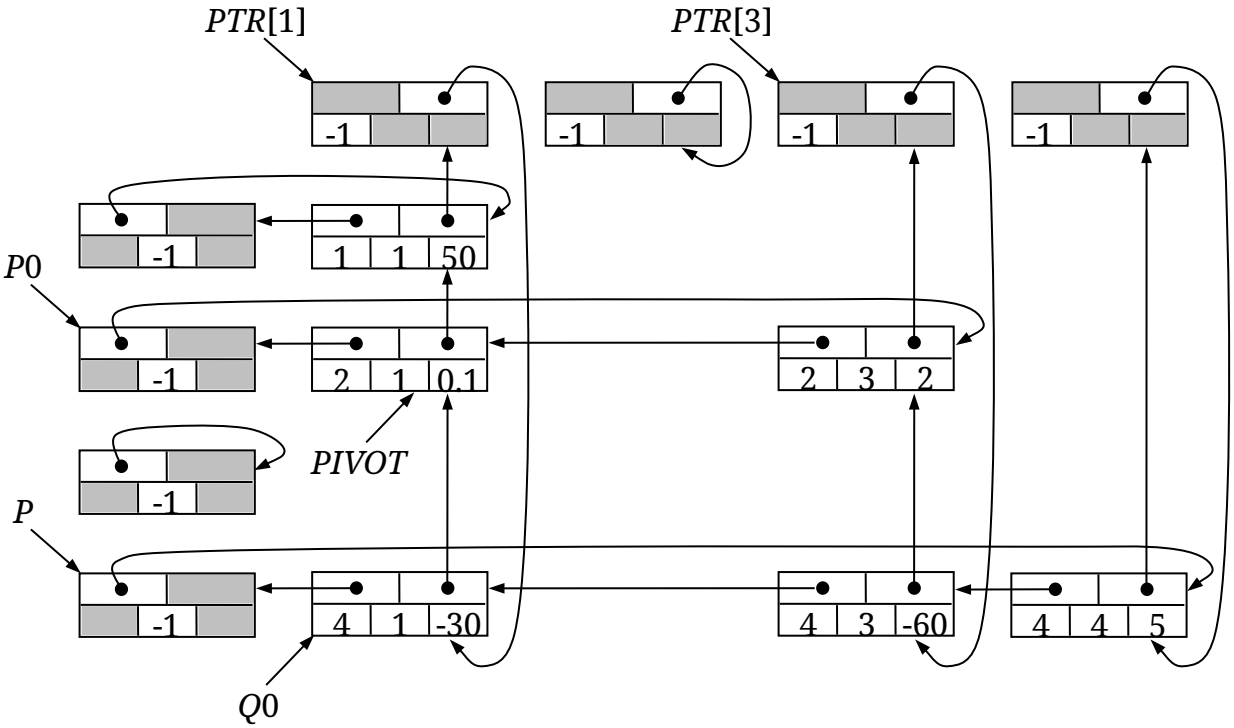


$I \leftarrow ROW(Q0) = 4$.

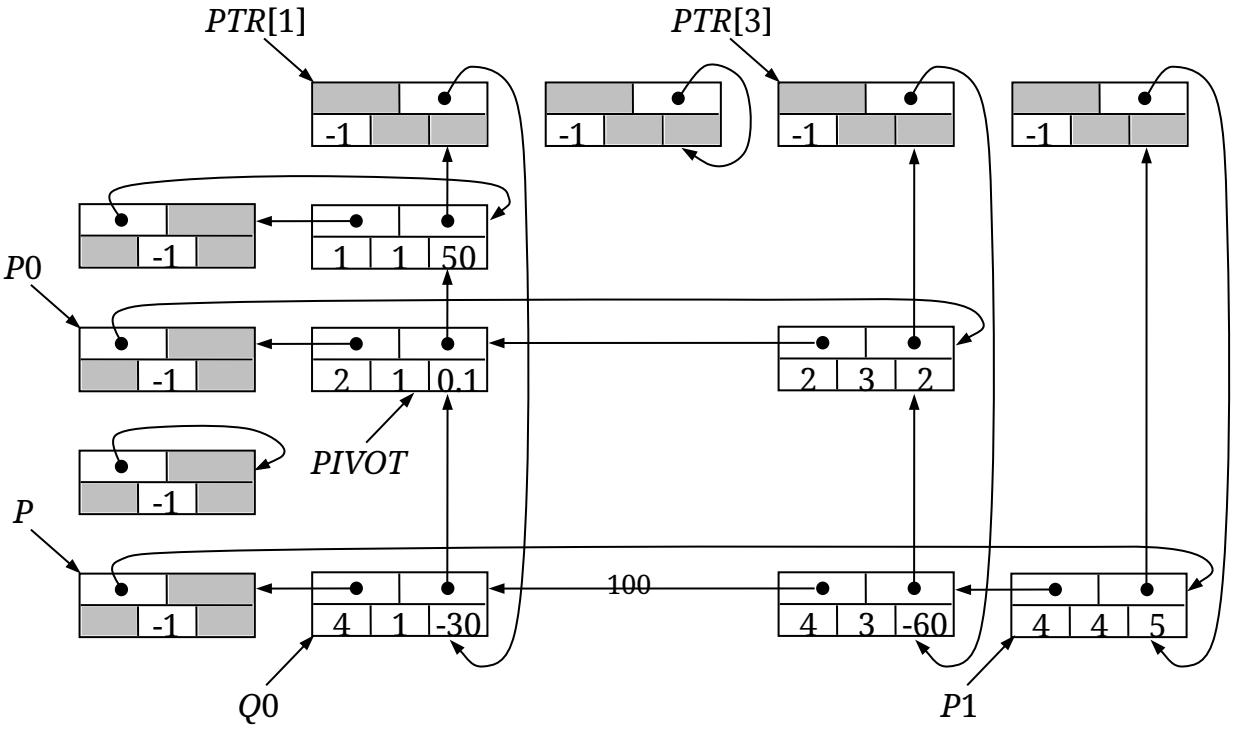
$(I < 0) = (4 < 0) =$ ЛОЖЬ.

$(I = I0) = (4 = 2) =$ ЛОЖЬ, следовательно

$(P \leftarrow LOC(BASEROW[I])) = (P \leftarrow LOC(BASEROW[4]))$,

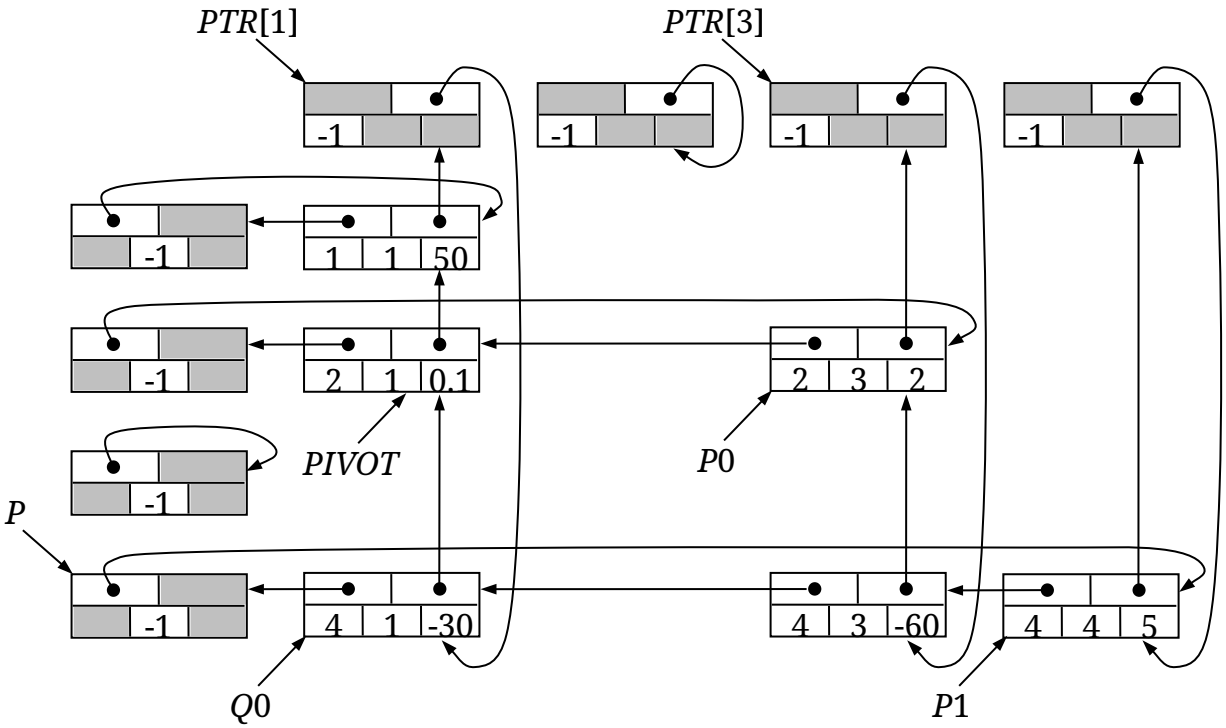


$P1 \leftarrow LEFT(P)$.



***S*4.** [Поиск нового столбца.]

$P0 \leftarrow LEFT(P0)$,



$J \leftarrow COL(P0) = 3$.

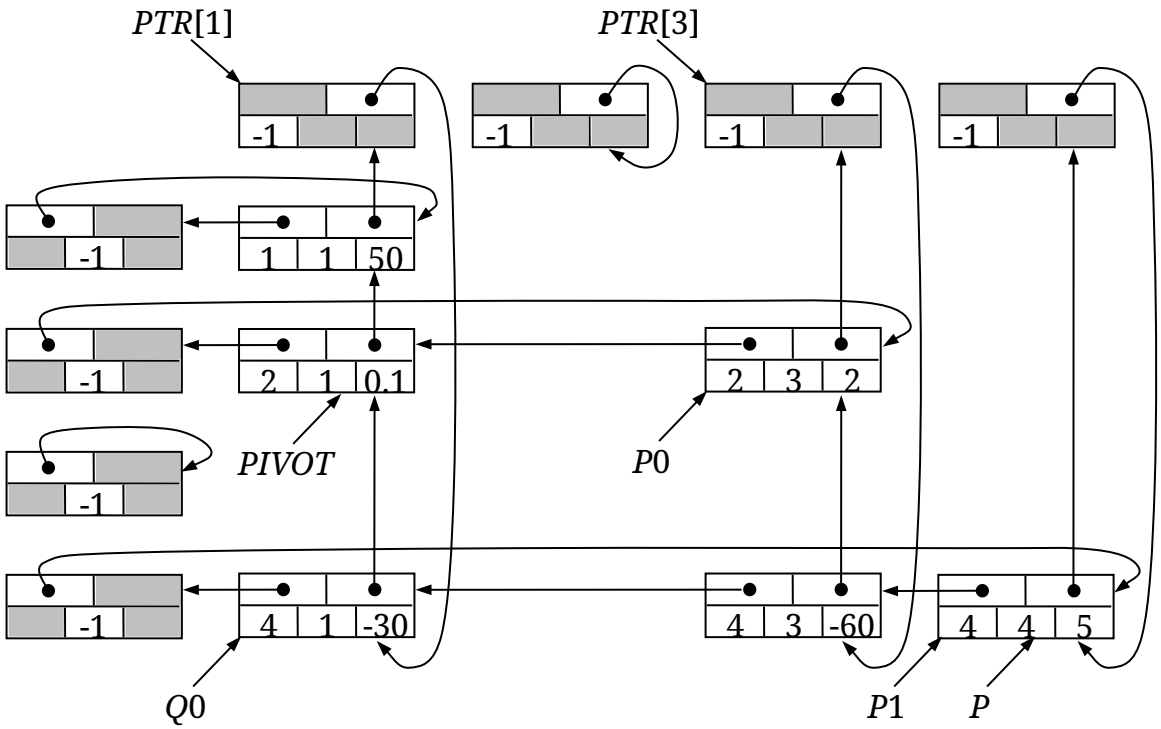
$(J < 0) = (3 < 0) =$ ЛОЖЬ.

$(J = J0) = (3 = 1) =$ ЛОЖЬ.

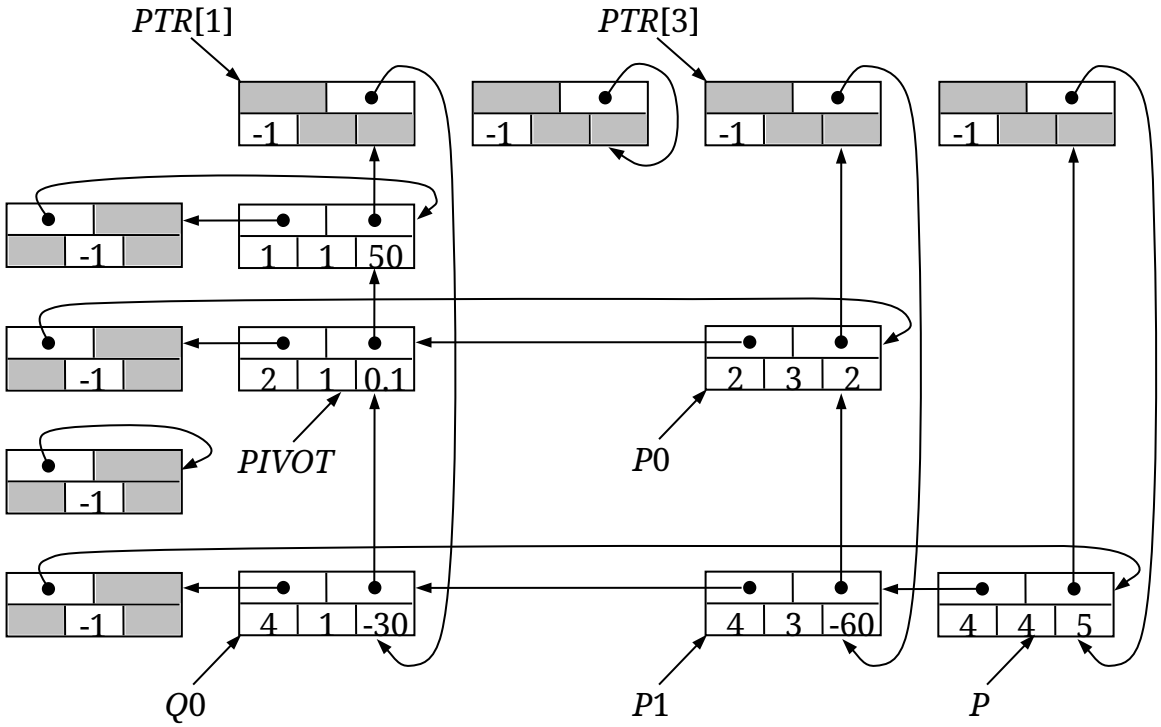
***S*5.** [Поиск элемента I, J.]

$(COL(P1) > J) = (4 > 3) =$ ИСТИНА, следовательно

$P \leftarrow P1$,



$P1 \leftarrow LEFT(P)$



и повторить этот шаг сначала.

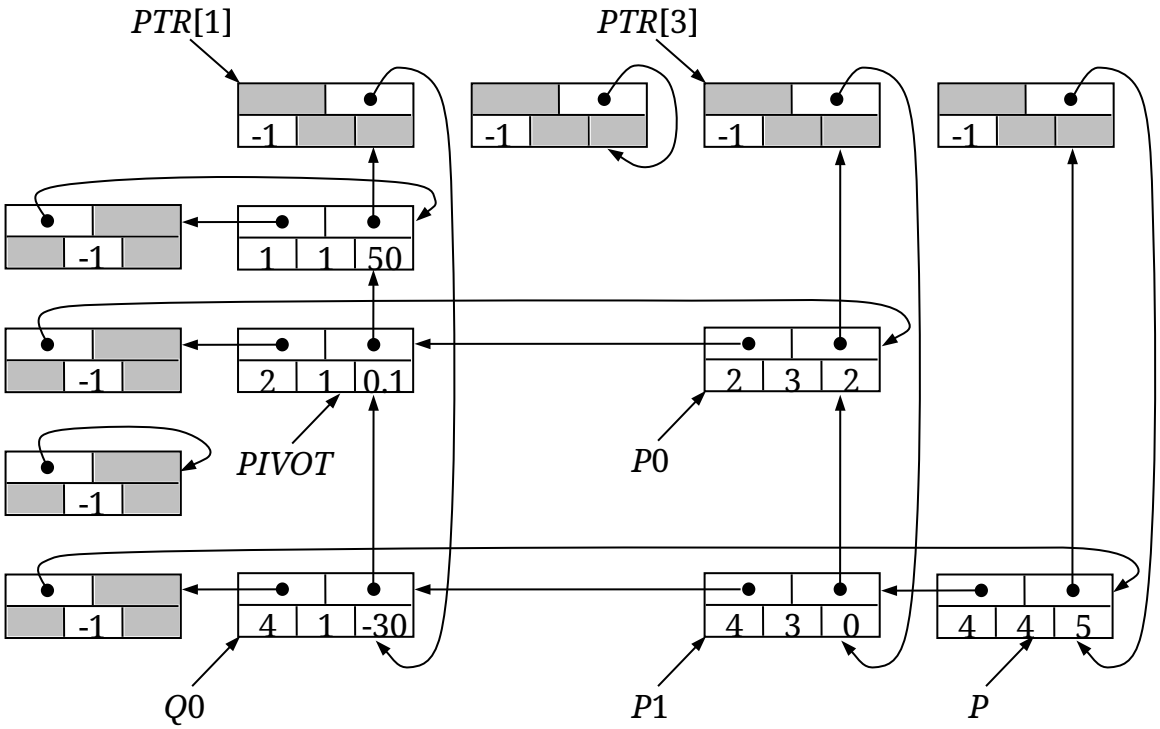
***S*5.** [Поиск элемента *I*, *J*.]

(*COL*(*P*1) > *J*) = (3 > 3) = ЛОЖЬ.

(*COL*(*P*1) = *J*) = (3 = 3) = ИСТИНА, следовательно перейти к *S*7.

***S*7.** [Осевая операция.]

VAL(*P*1) ← *VAL*(*P*1) – *VAL*(*Q*0) • *VAL*(*P*0) = –60 – (–30) • 2 = –60 + 60 = 0.

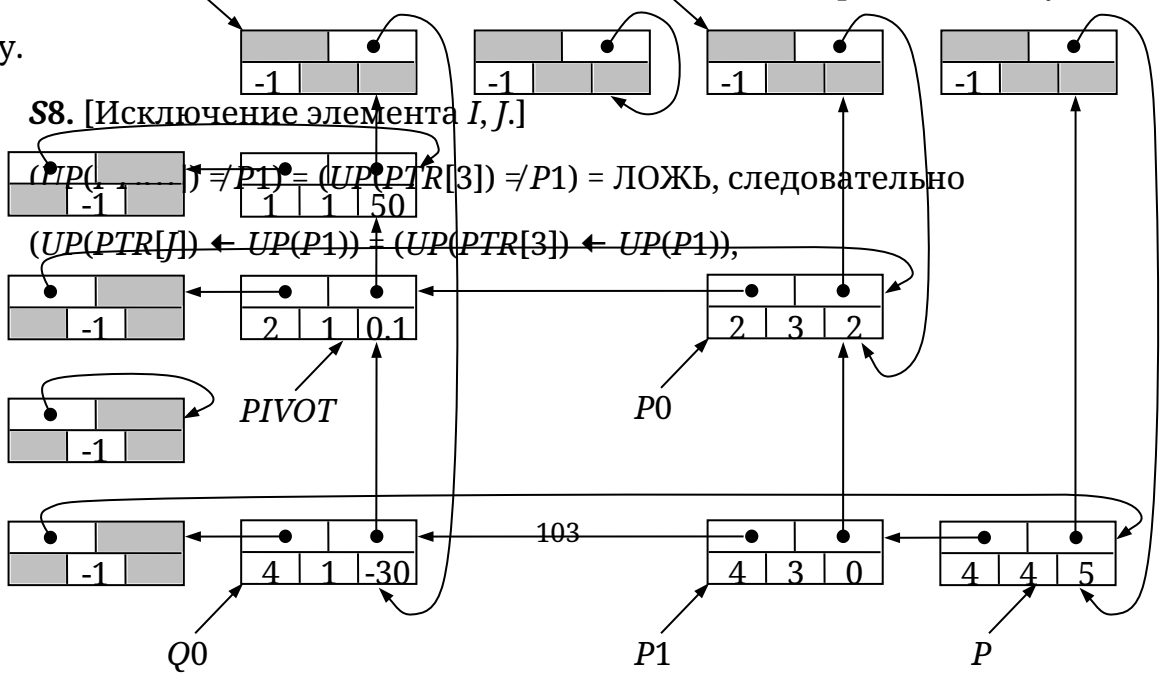


(*VAL*(*P*1) = 0) = (0 = 0) = ИСТИНА, следовательно перейти к следующему шагу.

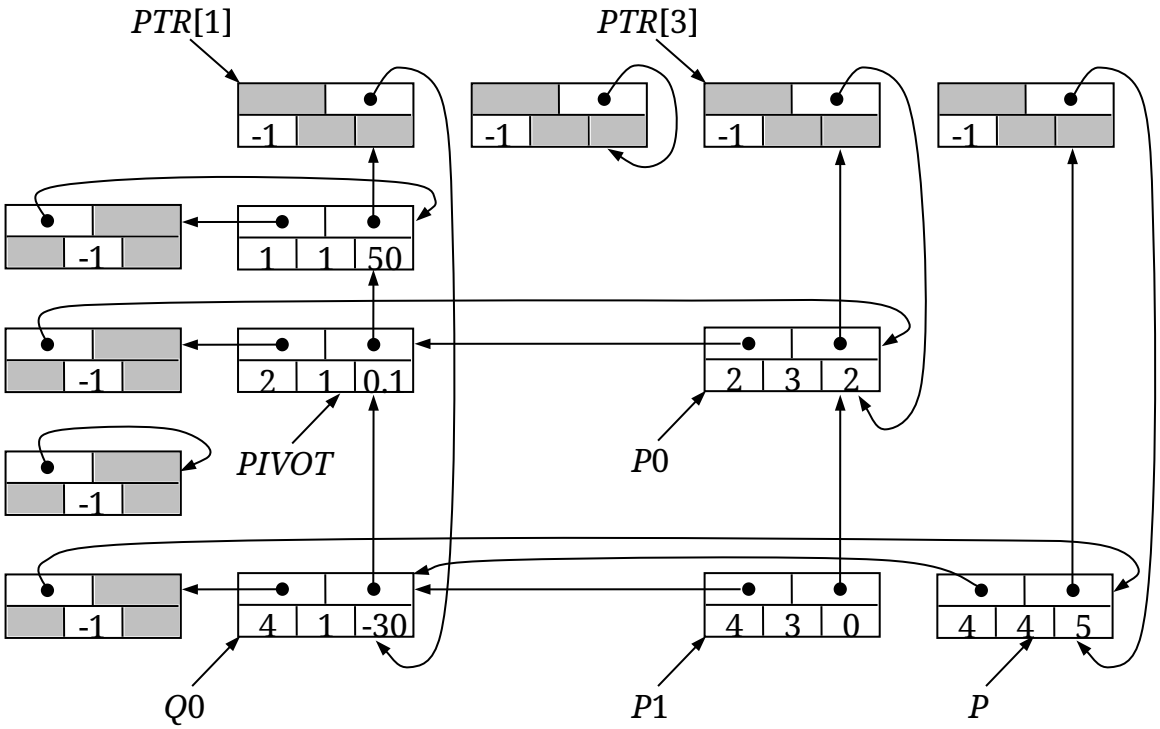
***S*8.** [Исключение элемента *I*, *J*.]

(*UP*(*P*1) ≠ *P*1) = (*UP*(*PTR*[3]) ≠ *P*1) = ЛОЖЬ, следовательно

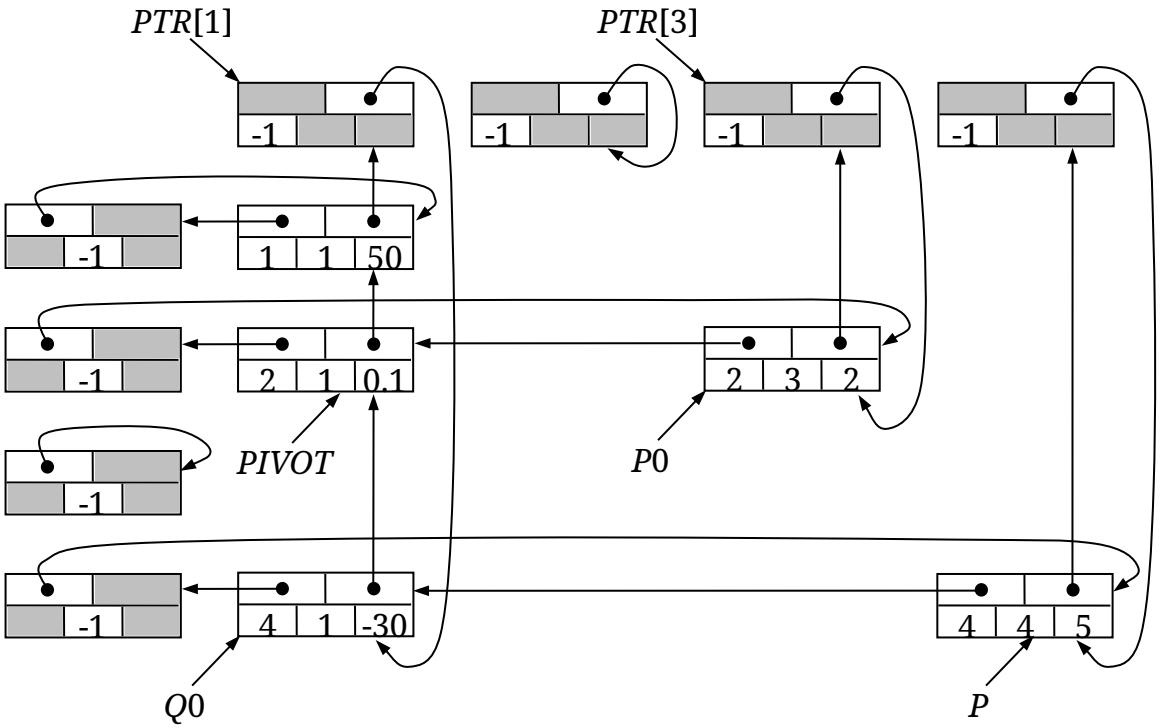
(*UP*(*PTR*[*J*]) ← *UP*(*P*1)) = (*UP*(*PTR*[3]) ← *UP*(*P*1)),



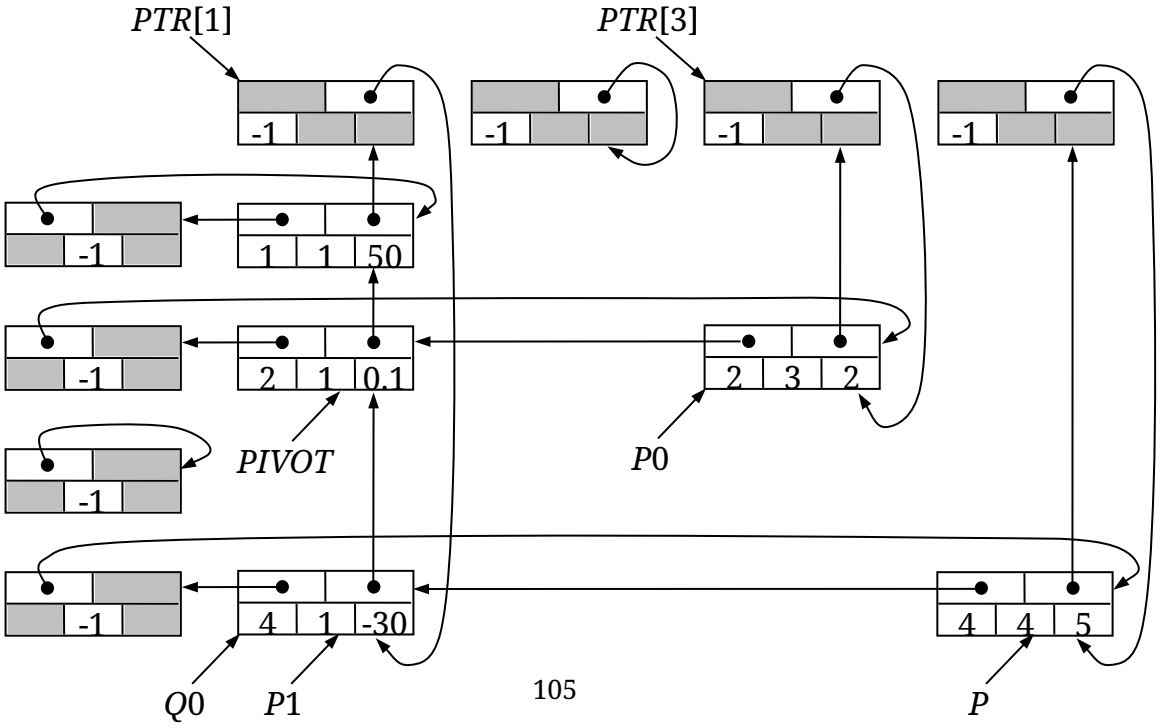
LEFT(*P*) ← *LEFT*(*P*1),



AVAIL ⇐ *P*1,



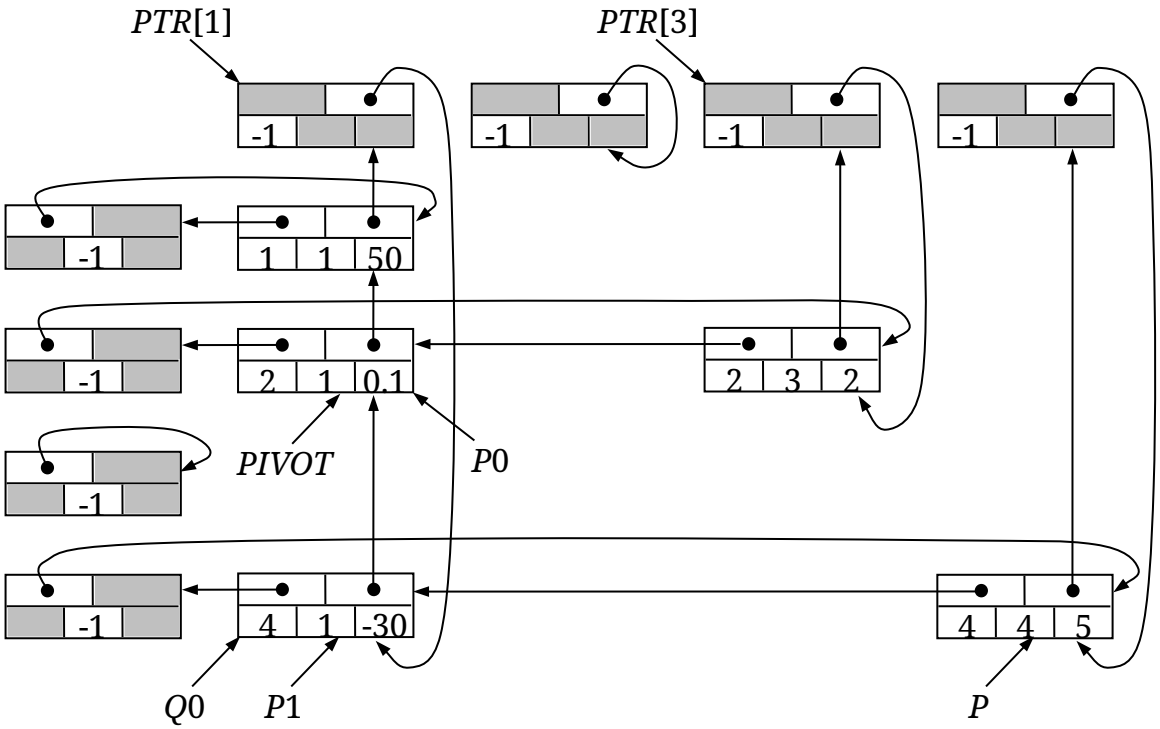
*P*1 ← *LEFT*(*P*).



Вернуться к шагу *S*4.

***S*4.** [Поиск нового столбца.]

*P*0 ← *LEFT*(*P*0),



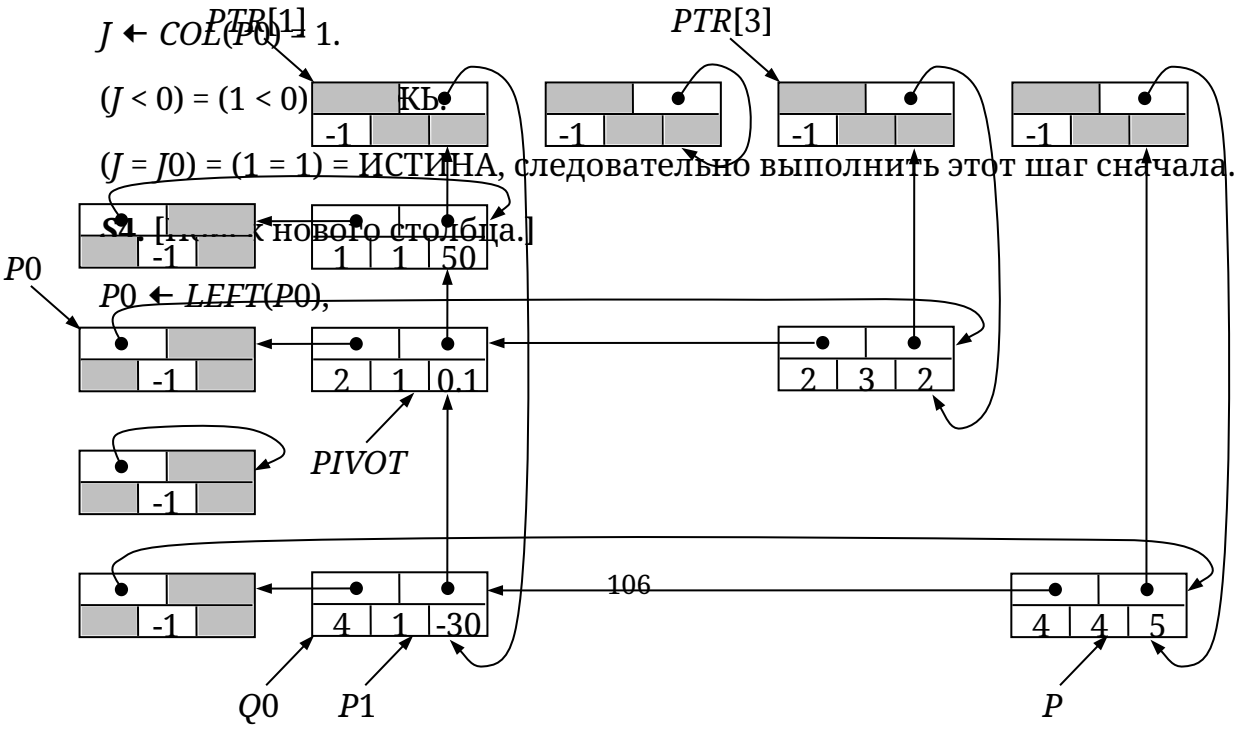
J ← *COL*(*P*0) = 1.

(*J* < 0) = (1 < 0) = ЛОЖЬ.

(*J* = *J*0) = (1 = 1) = ИСТИНА, следовательно выполнить этот шаг сначала.

***S*4.** [Поиск нового столбца.]

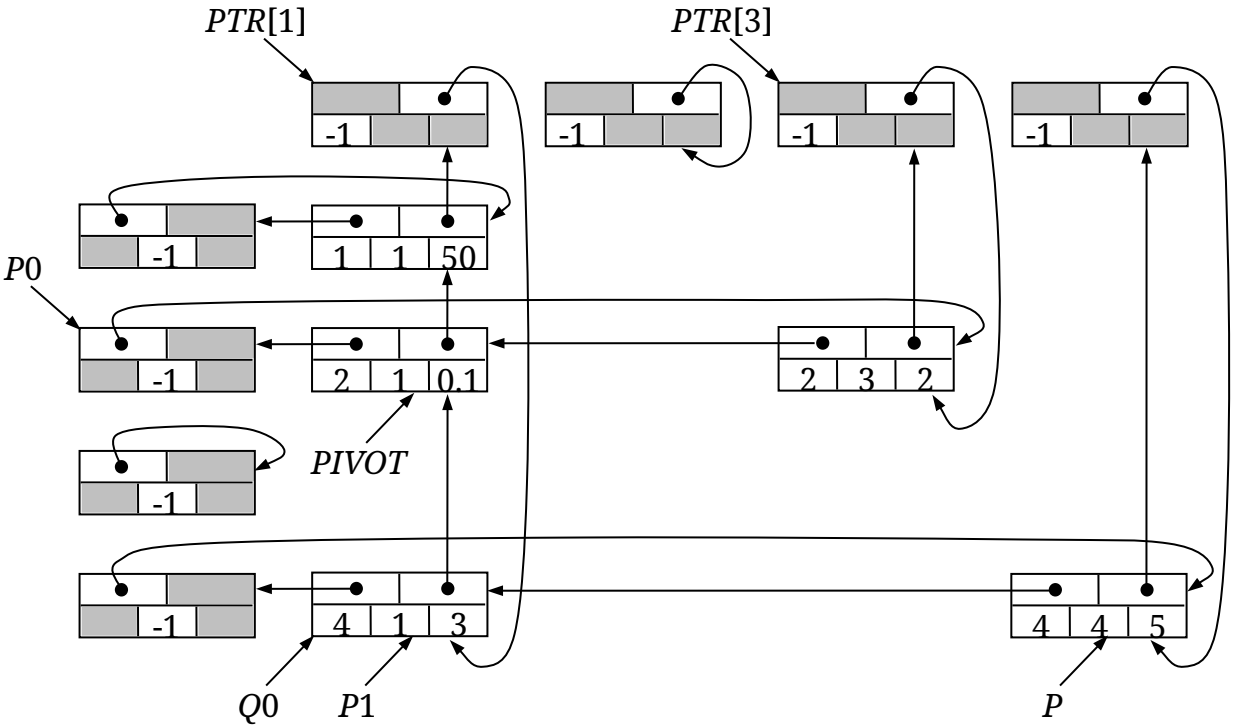
*P*0 ← *LEFT*(*P*0),



$J \leftarrow COL(P0) = -1$.

$(J < 0) = (-1 < 0)$ = ИСТИНА, следовательно

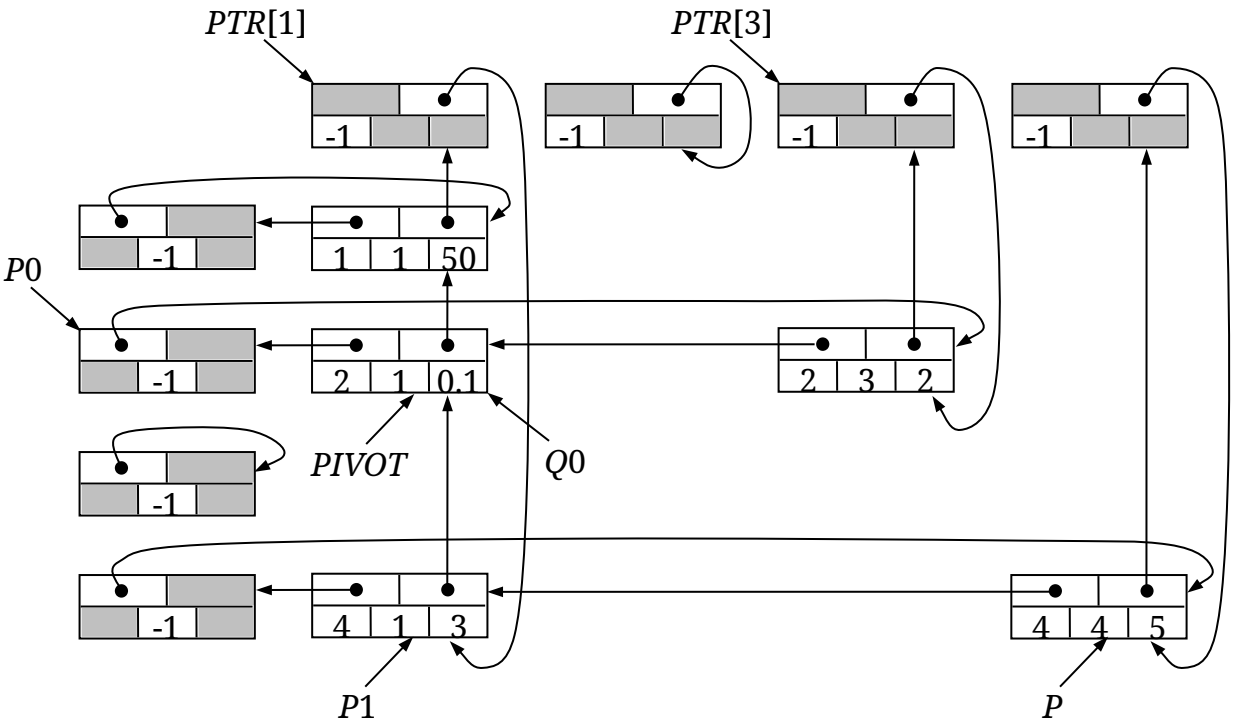
$VAL(Q0) \leftarrow -ALPHA \cdot VAL(Q0) = -0.1 \cdot -30 = 3$



и вернуться к $S3$.

***S*3.** [Поиск новой строки.]

$Q0 \leftarrow UP(Q0)$,



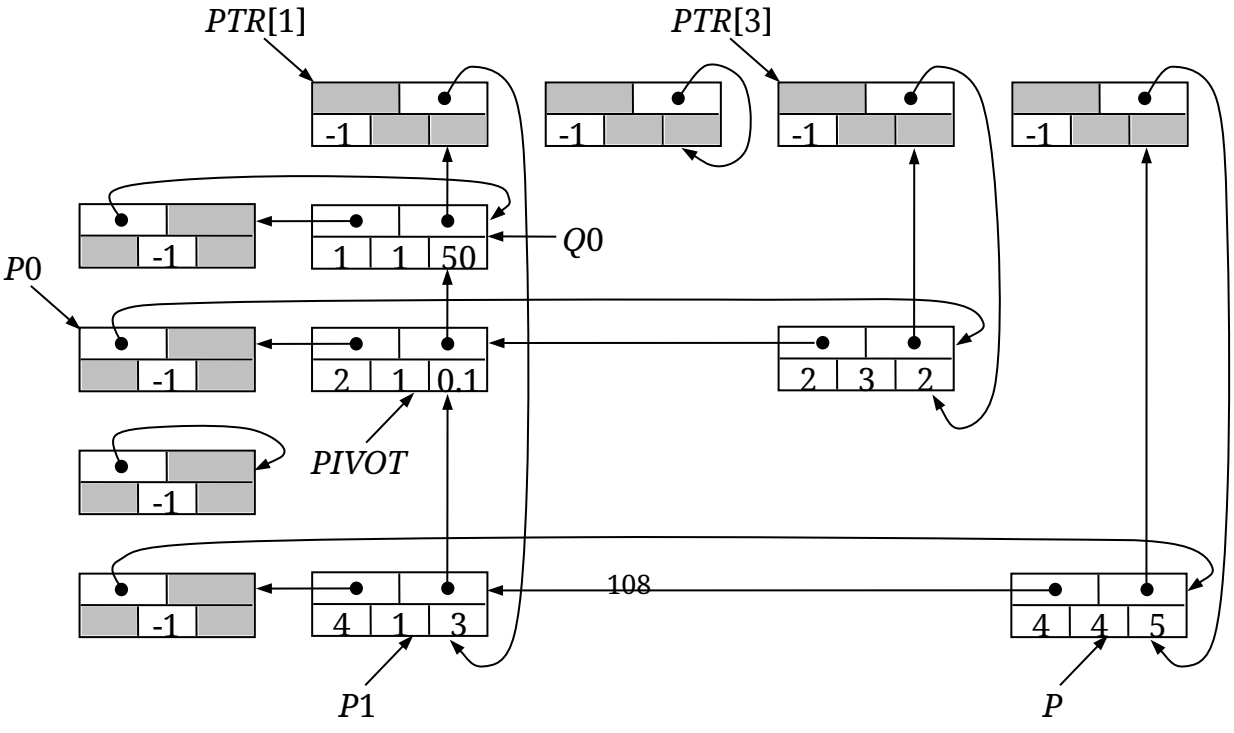
$I \leftarrow ROW(Q0) = 2$.

$(I < 0) = (2 < 0)$ = ЛОЖЬ.

$(I = I0) = (2 = 2)$ = ИСТИНА, следовательно повторить шаг *S*3.

***S*3.** [Поиск новой строки.]

$Q0 \leftarrow UP(Q0)$,

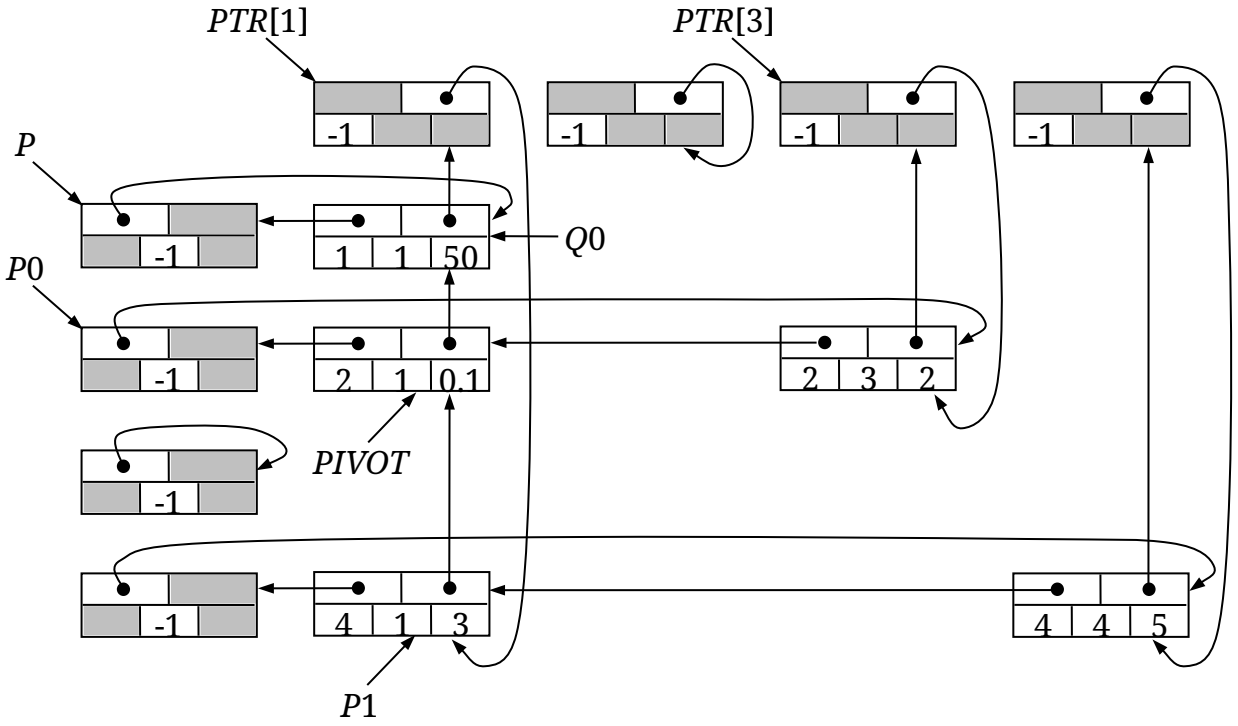


$I \leftarrow ROW(Q0) = 1$.

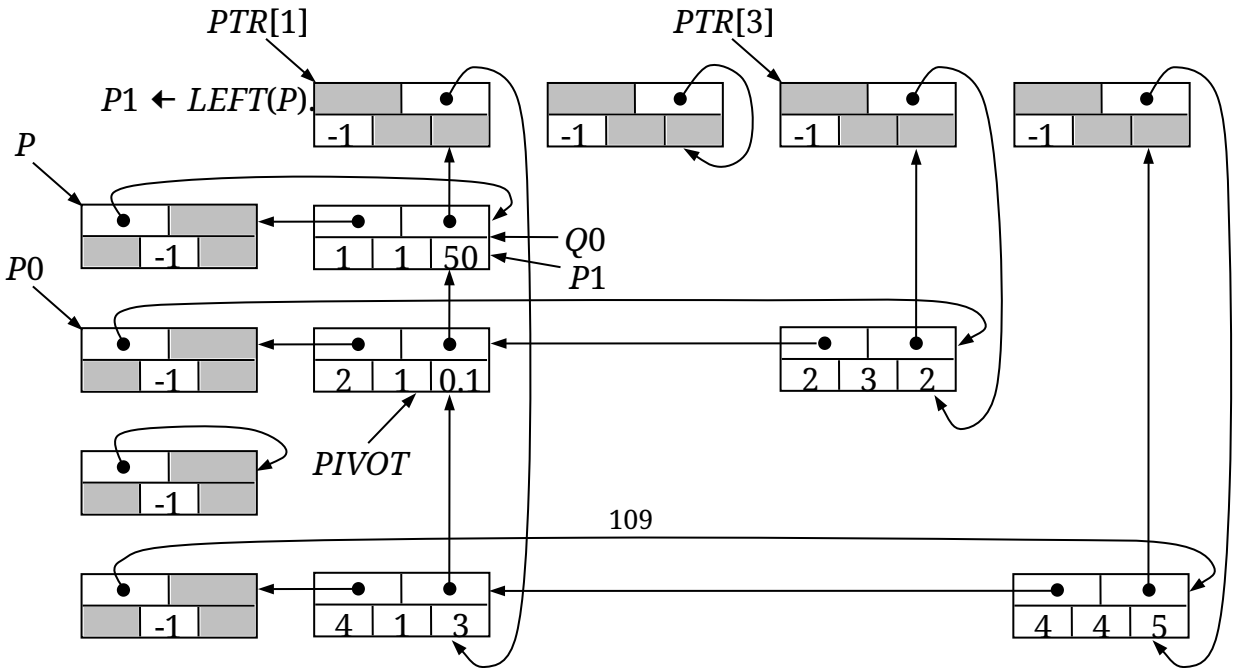
$(I < 0) = (1 < 0)$ = ЛОЖЬ.

$(I = I0) = (1 = 2)$ = ЛОЖЬ, следовательно

$P \leftarrow LOC(BASEROW[I]) = LOC(BASEROW[1])$,

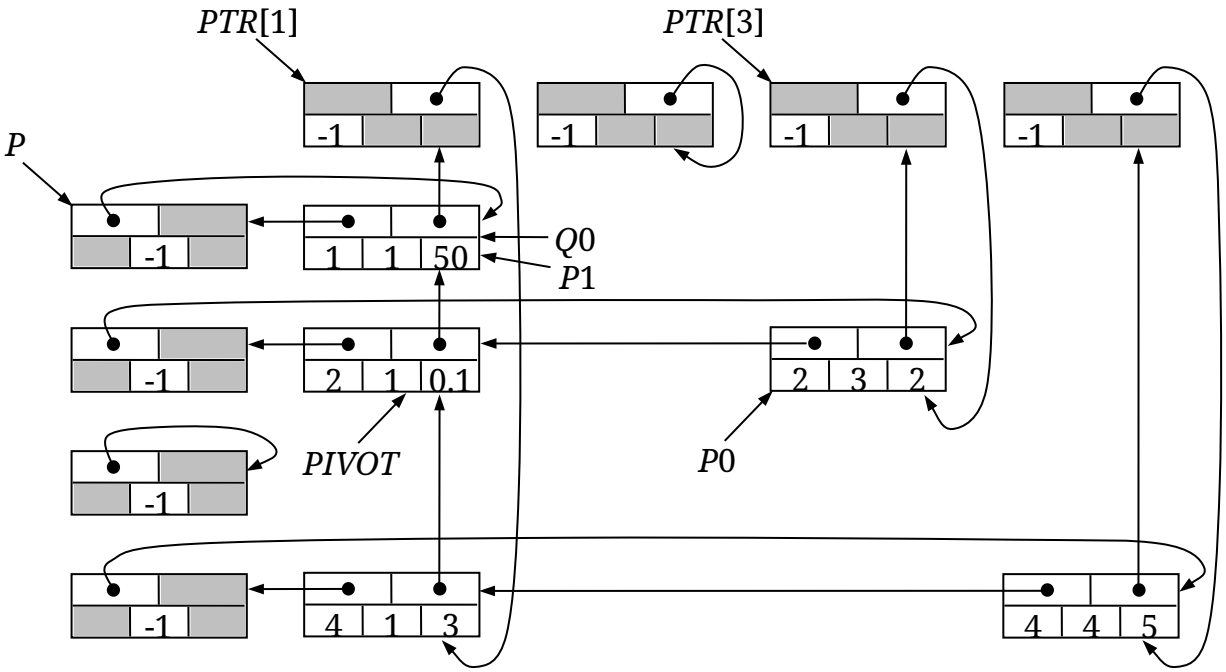


$P1 \leftarrow LEFT(P)$.



***S*4.** [Поиск нового столбца.]

$P0 \leftarrow LEFT(P0)$,



$J \leftarrow COL(P0) = 3$.

$(J < 0) = (3 < 0) =$ ЛОЖЬ.

$(J = J0) = (3 = 1) =$ ЛОЖЬ.

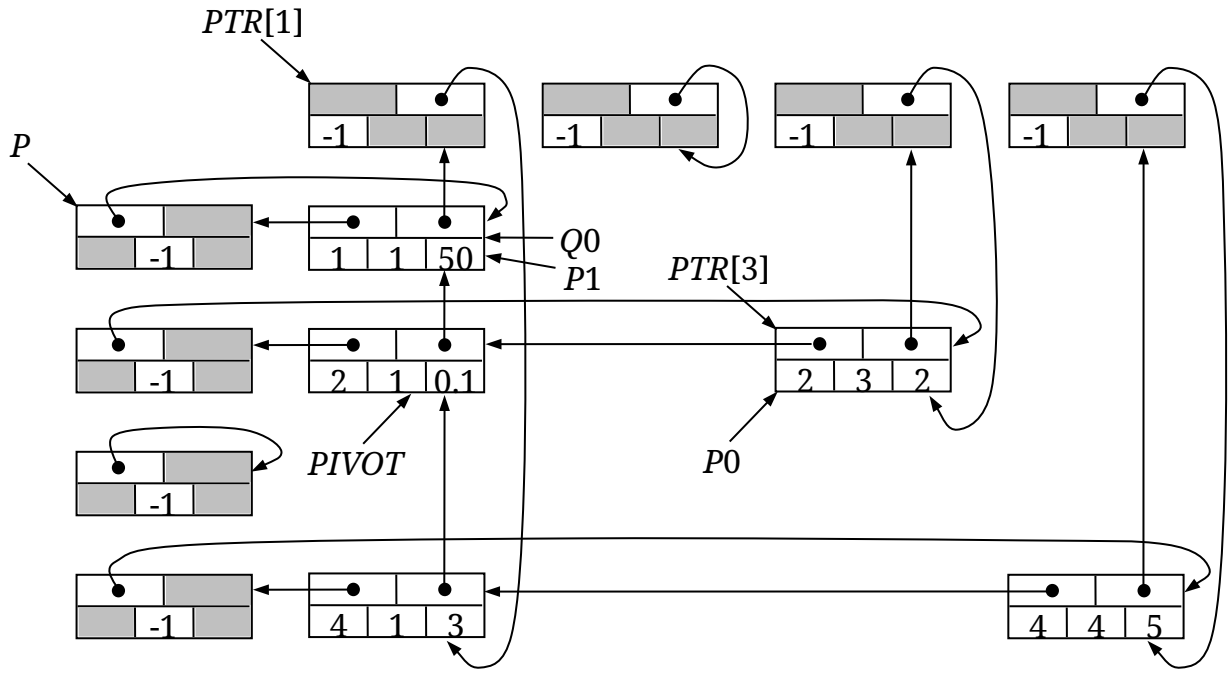
***S*5.** [Поиск элемента I, J.]

$(COL(P1) > J) = (1 > 3) =$ ЛОЖЬ.

(*COL*(*P*1) = *J*) = (1 = 3) = ЛОЖЬ, следовательно перейти к следующему шагу.

***S*6.** [Включение элемента *I*, *J*.]

(*ROW*(*UP*(*PTR*[*J*])) > *I*) = (*ROW*(*UP*(*PTR*[3])) > 1) = (2 > 1) = ИСТИНА, следовательно (*PTR*[*J*] ← *UP*(*PTR*[*J*]) = (*PTR*[3] ← *UP*(*PTR*[3])

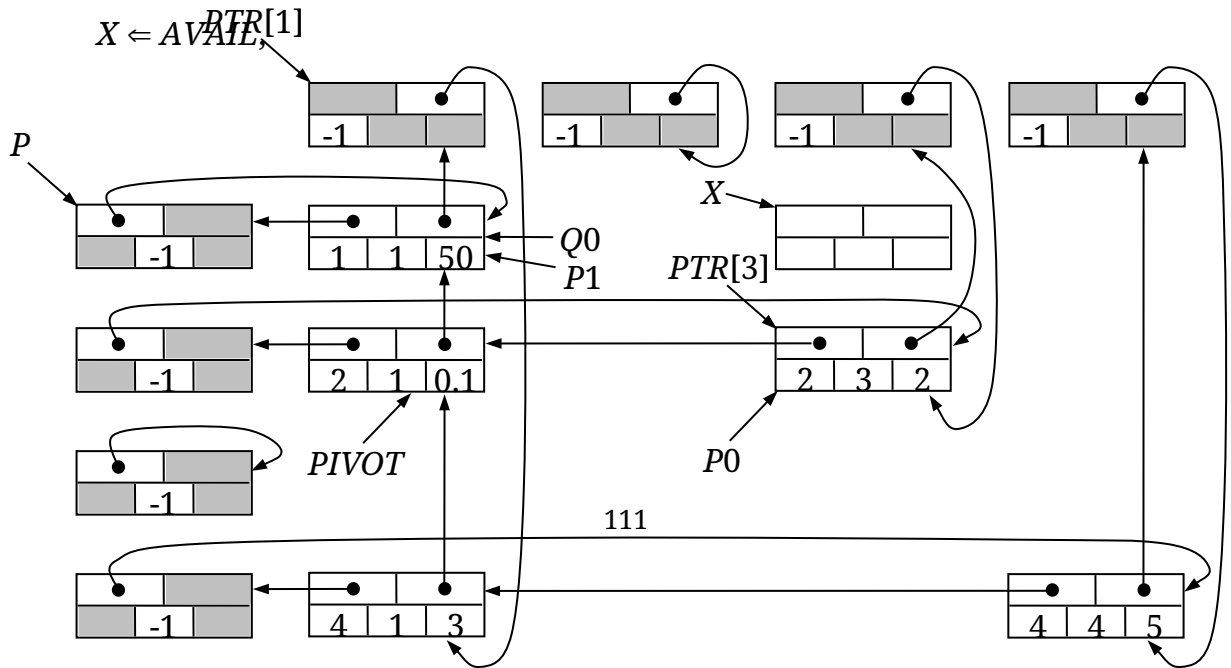


и повторить этот шаг сначала.

***S*6.** [Включение элемента *I*, *J*.]

(*ROW*(*UP*(*PTR*[*J*])) > *I*) = (*ROW*(*UP*(*PTR*[3])) > 1) = (–1 > 1) = ЛОЖЬ, следовательно

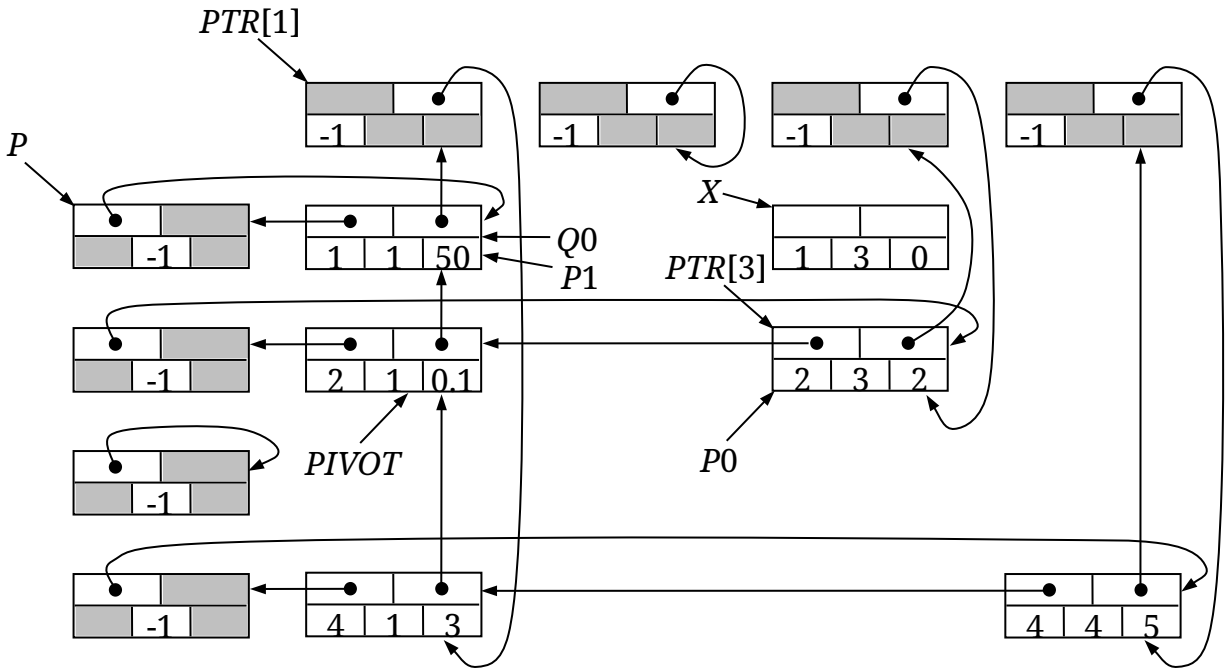
X ⇐ *AVAIL*,

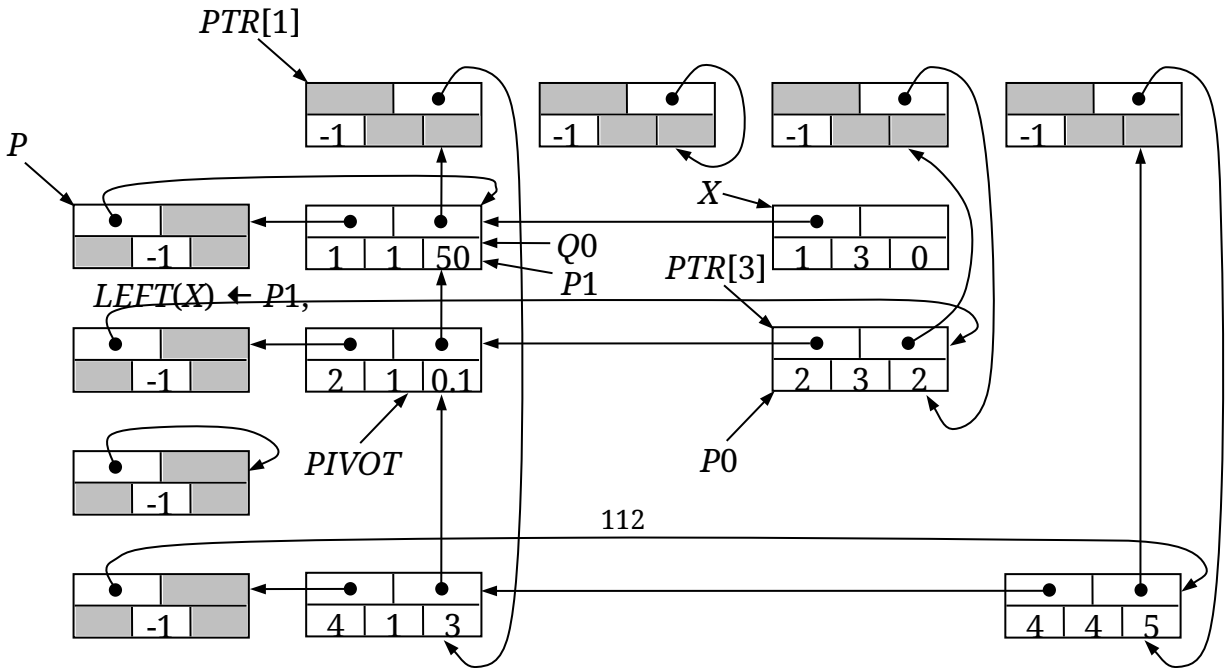


$VAL(X) \leftarrow 0$,

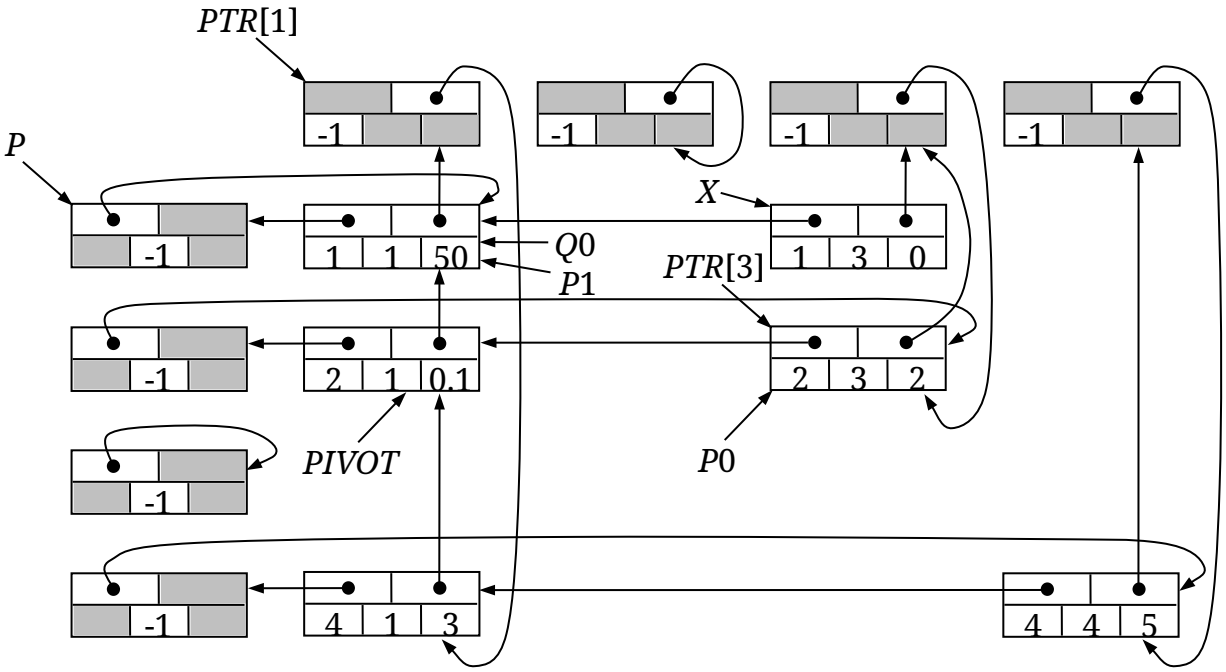
$ROW(X) \leftarrow I = 1$,

$COL(X) \leftarrow J = 3$,

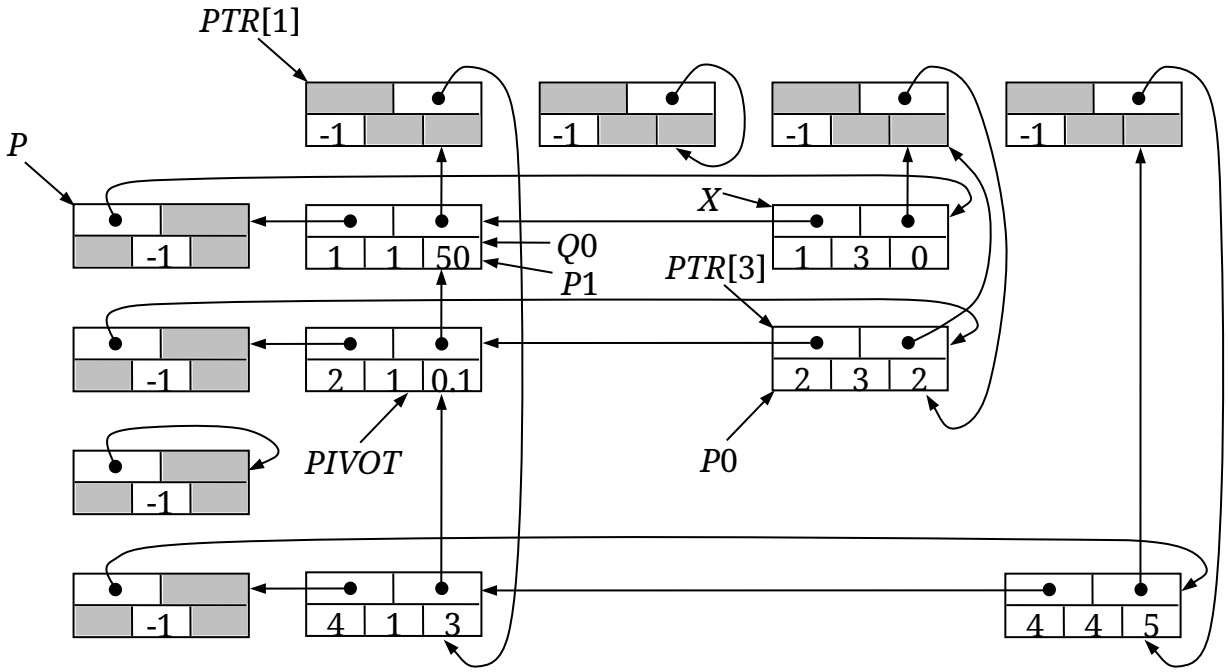




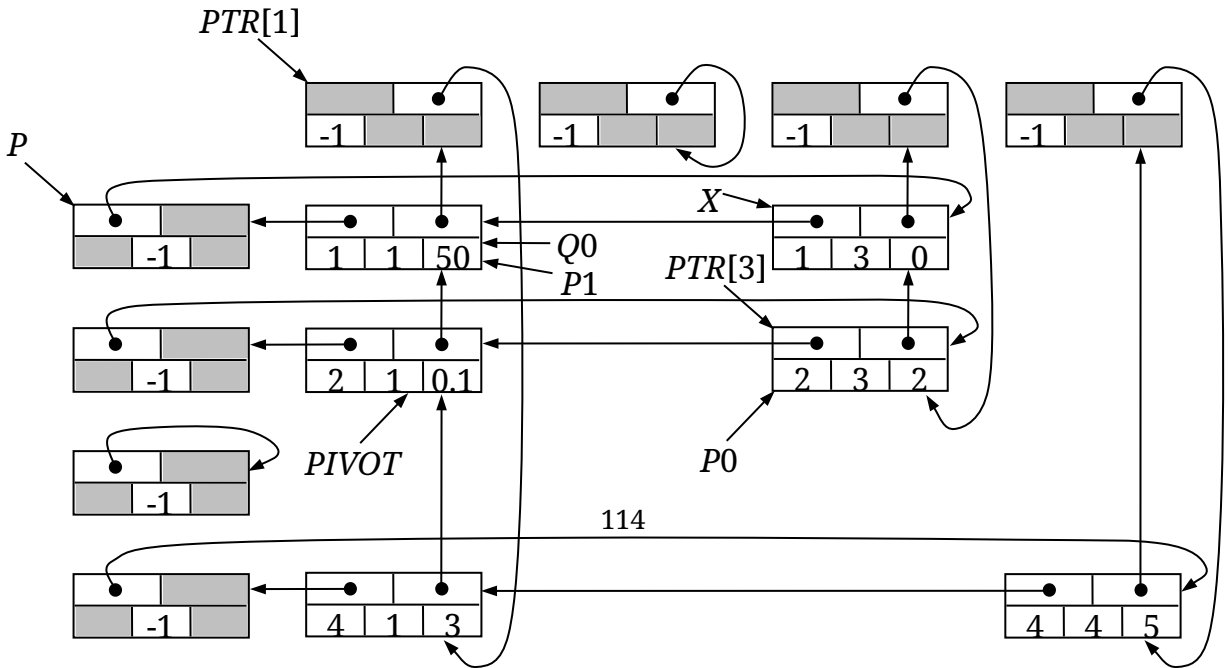
$UP(X) \leftarrow UP(PTR[J]) = UP(PTR[3])$,



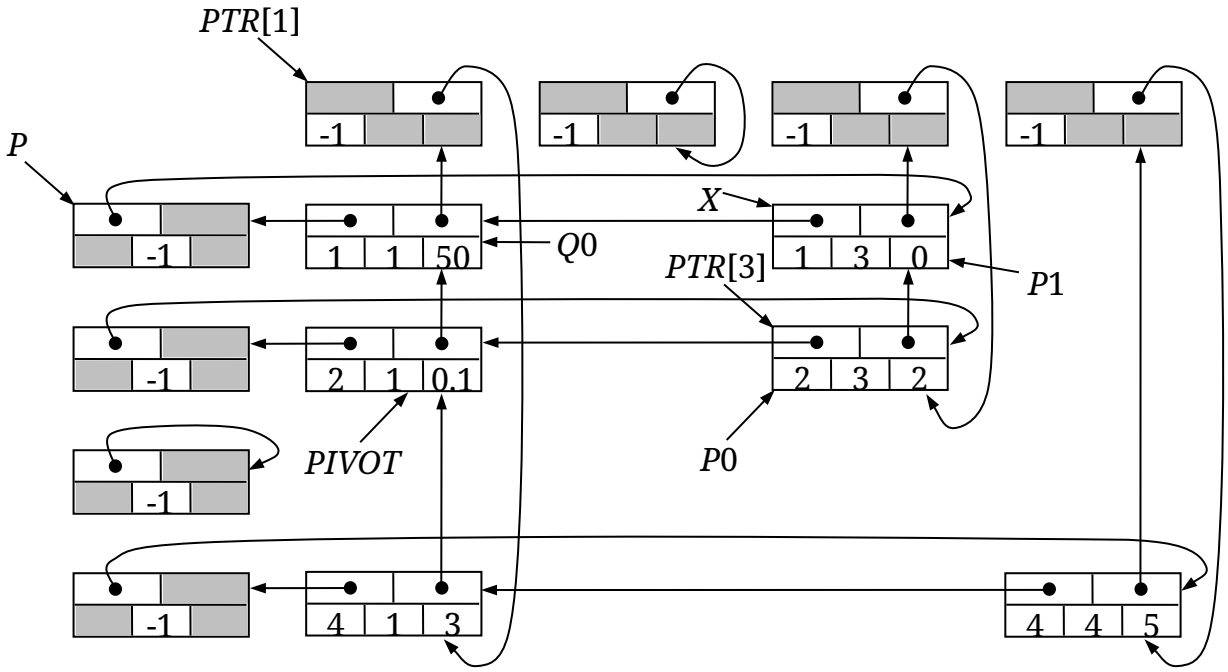
$LEFT(P) \leftarrow X$,

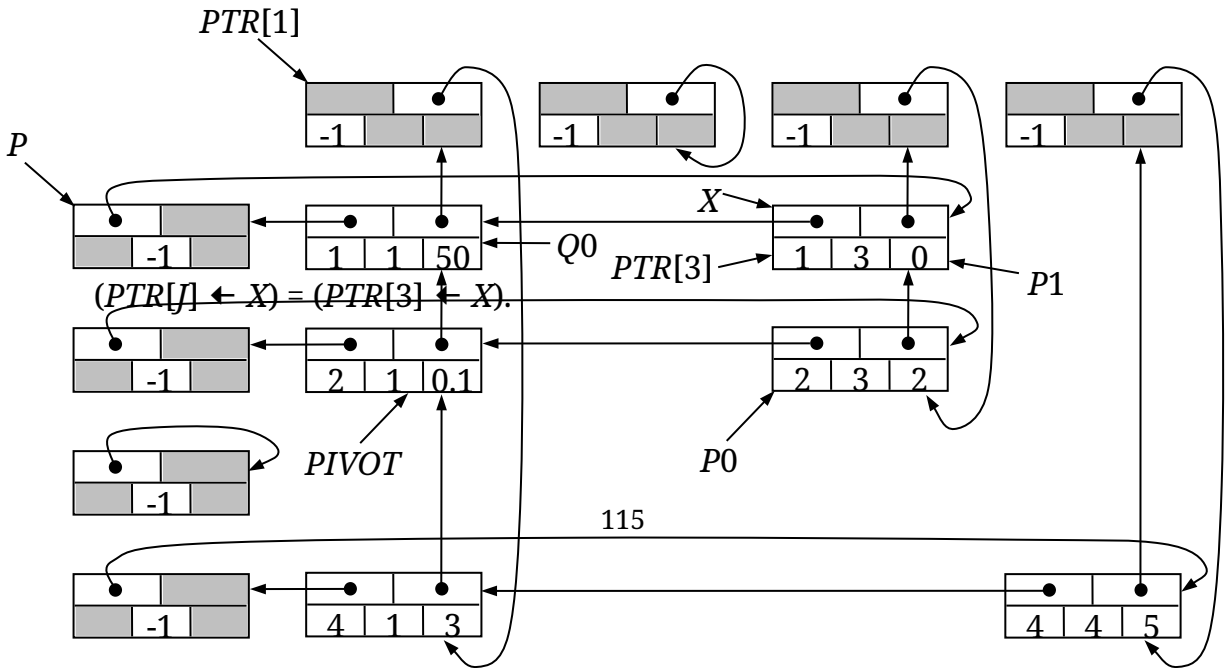


$(UP(PTR[J]) \leftarrow X) = (UP(PTR[3]) \leftarrow X)$,



$P1 \leftarrow X$,

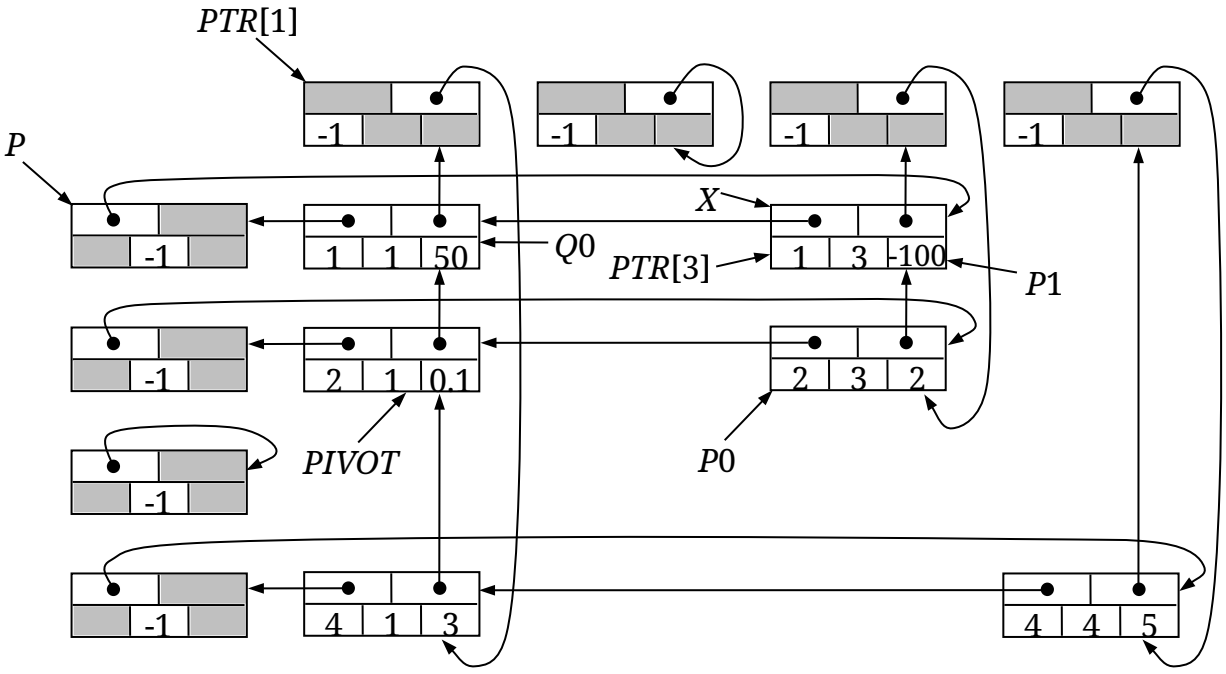




$(PTR[J] \leftarrow X) = (PTR[3] \leftarrow X)$.

***S*7.** [Осевая операция.]

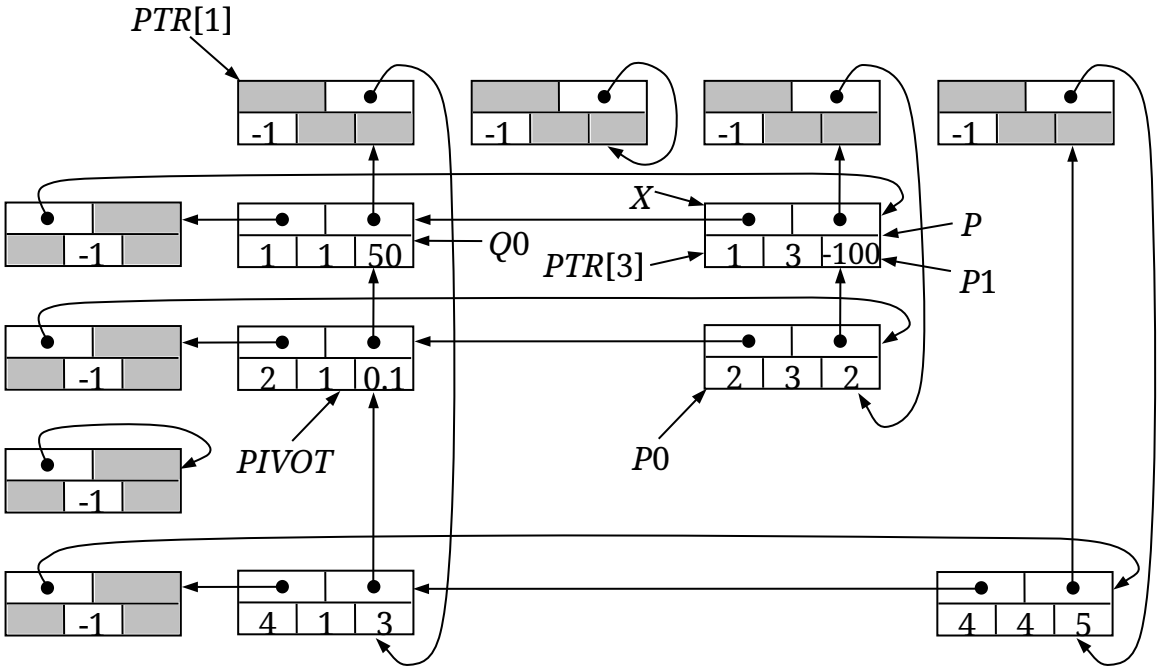
$VAL(P1) \leftarrow VAL(P1) - VAL(Q0) \cdot VAL(P0) = 0 - 50 \cdot 2 = -100.$



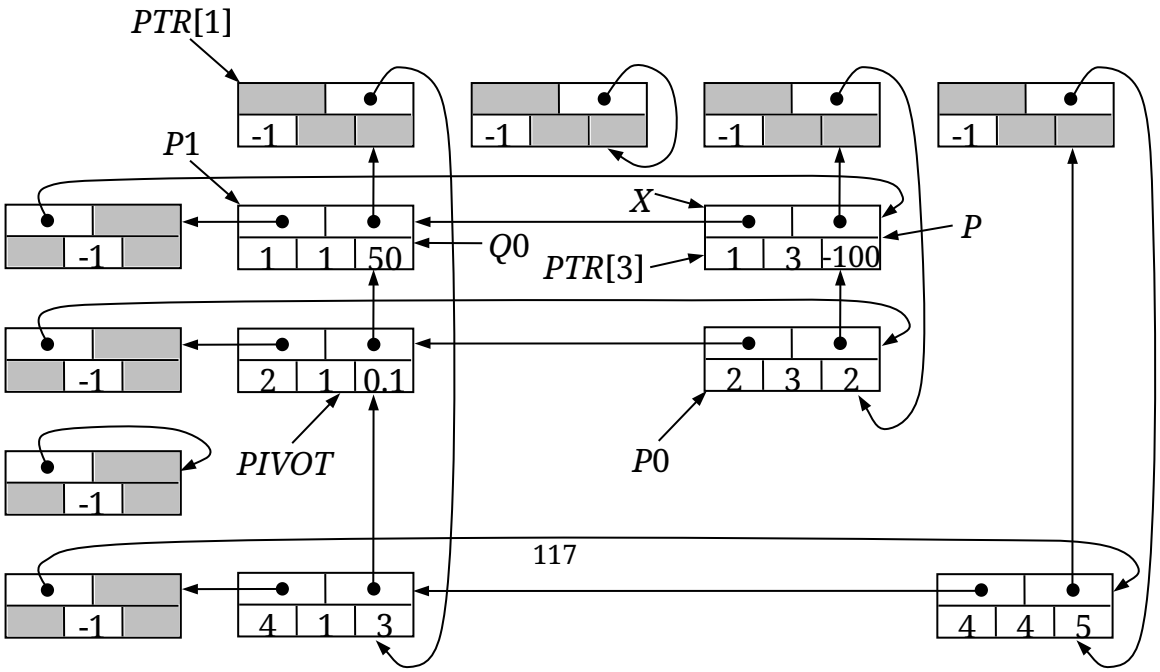
(*VAL*(*P*1) = 0) = (–100 = 0) = ЛОЖЬ, следовательно

(*PTR*[*J*] ← *P*1) = (*PTR*[3] ← *P*1),

P ← *P*1,



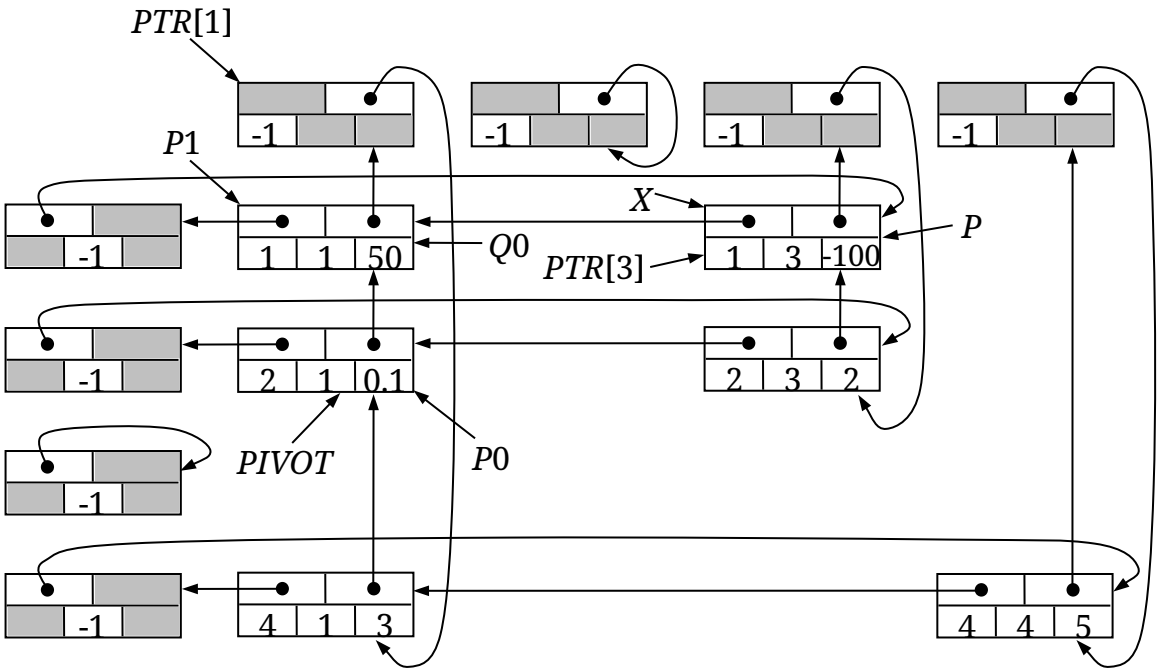
*P*1 ← *LEFT*(*P*)



и вернуться к *S*4.

***S*4.** [Поиск нового столбца.]

*P*0 ← *LEFT*(*P*0),



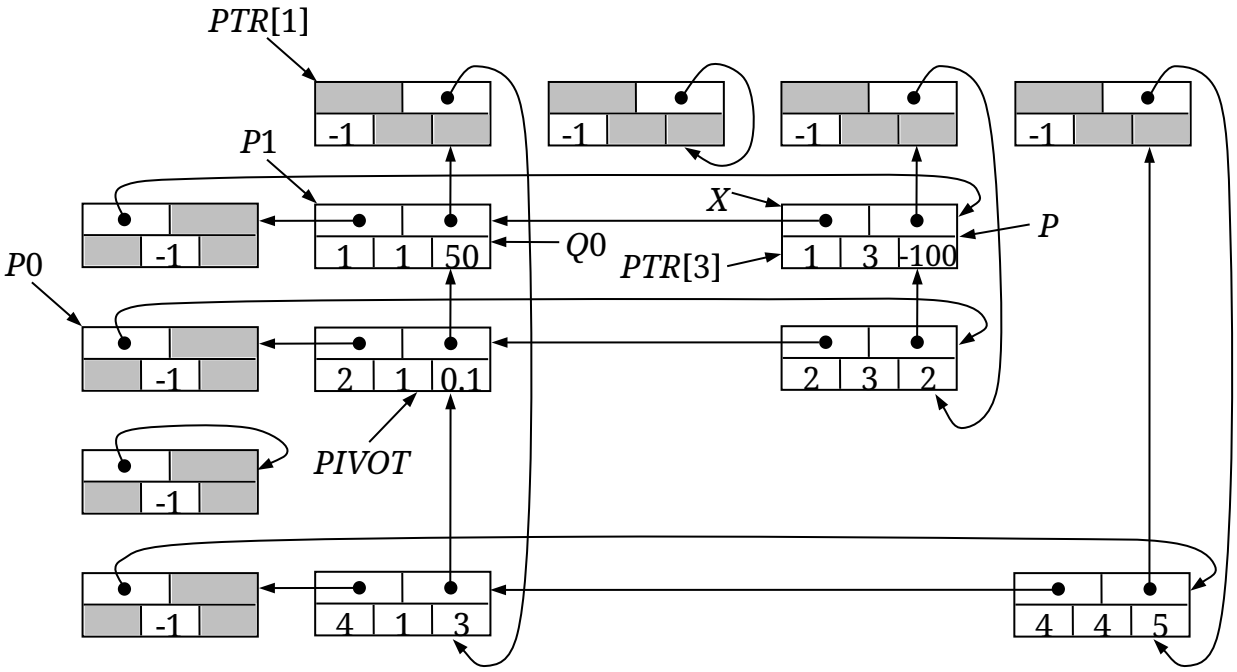
J ← *COL*(*P*0) = 1.

(*J* < 0) = (1 < 0) = ЛОЖЬ.

(*J* = *J*0) = (1 = 1) = ИСТИНА, следовательно выполнить этот шаг сначала.

***S*4.** [Поиск нового столбца.]

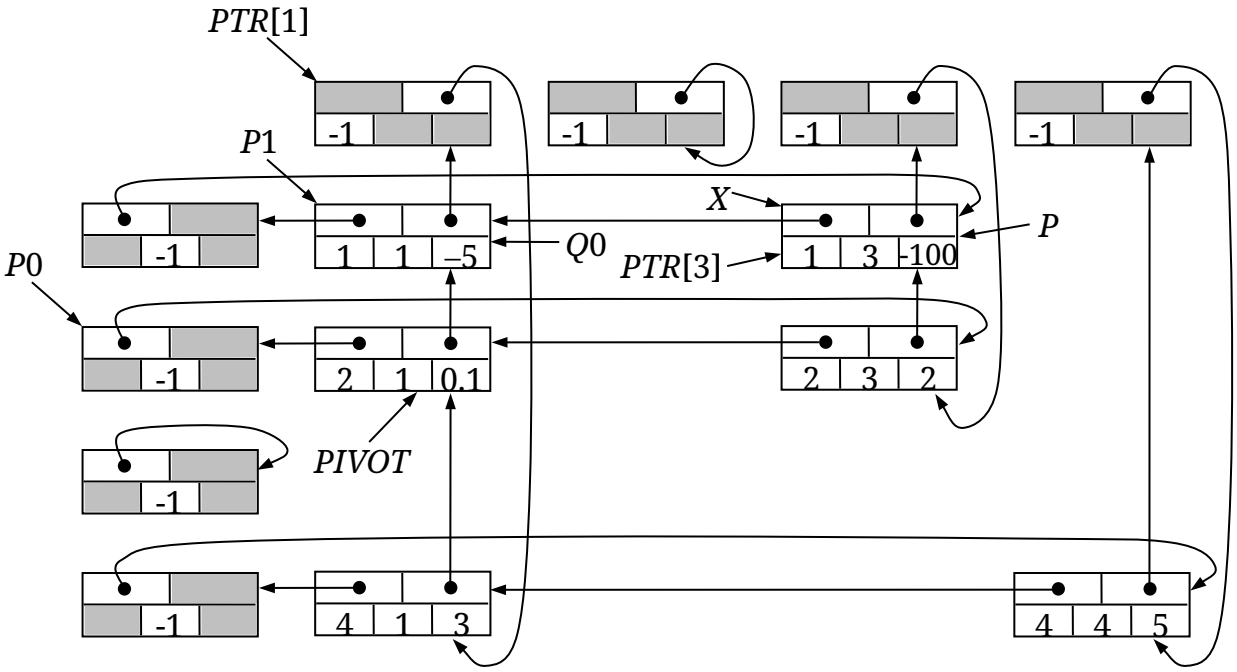
*P*0 ← *LEFT*(*P*0),



$J \leftarrow COL(P0) = -1.$

$(J < 0) = (-1 < 0) =$ ИСТИНА, следовательно

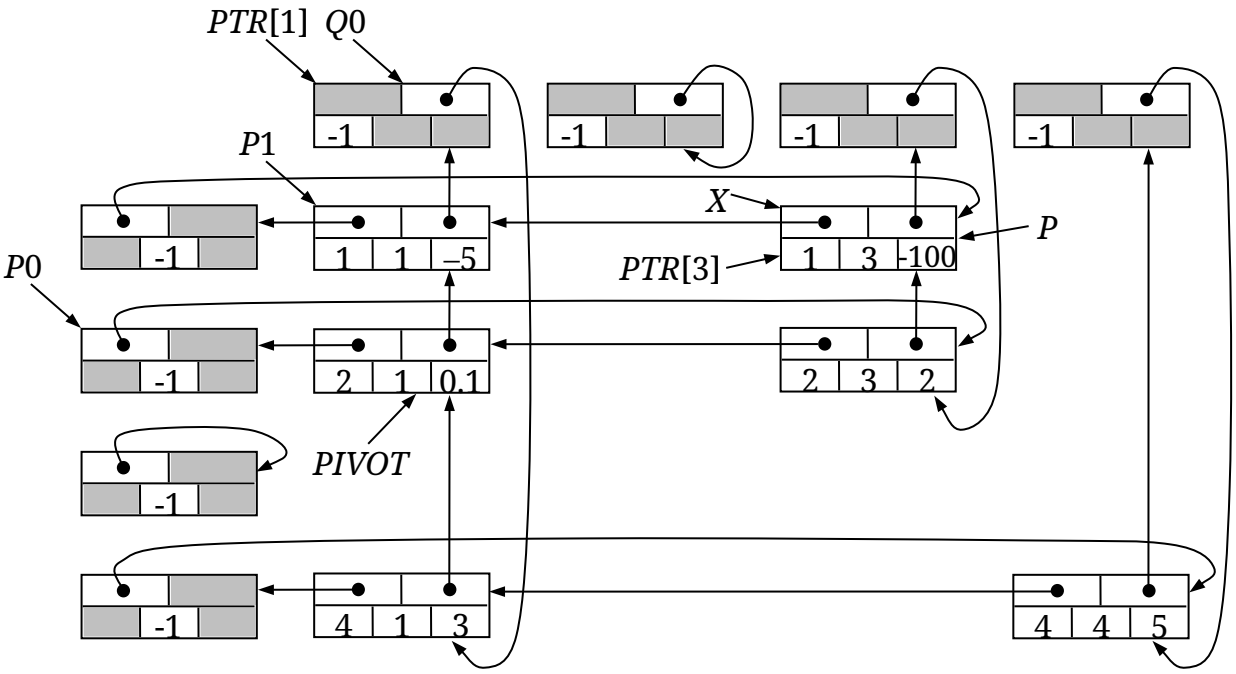
$VAL(Q0) \leftarrow -ALPHA \cdot VAL(Q0) = -0.1 \cdot 50 = -5$



и вернуться к *S*3.

***S*3.** [Поиск новой строки.]

*Q*0 ← *UP*(*Q*0),



I ← *ROW*(*Q*0) = –1.

(*I* < 0) = (–1 < 0) = ИСТИНА, следовательно конец алгоритма.

Ответ: Выполнение по алгоритму *S* осевого шага в разреженной матрице с заданными ненулевыми элементами *m*[1][1] = 50, *m*[2][3] = 20, *m*[2][1] = 10, *m*[4][4] = 5, *m*[4][3] = −60, *m*[4][1] = −30 и осевым элементом *m*[2][1] влечёт последовательное выполнение шагов алгоритма *S*1, *S*2, *S*2, *S*2, *S*3, *S*4, *S*5, *S*5, *S*7, *S*8, *S*4, *S*4, *S*3, *S*3, *S*4, *S*5, *S*6, *S*6, *S*7, *S*4, *S*4, *S*3 и конечную матрицу

$$\begin{pmatrix} -5 & 0 & -100 & 0 \\ 0.1 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 \\ 3 & 0 & 0 & 5 \end{pmatrix}.$$

ЗАКЛЮЧЕНИЕ

Первая часть учебного пособия по структурам и алгоритмам обработки данных посвящена изучению вопросов, связанных с представлением линейных информационных структур в памяти ЭВМ и алгоритмами их обработки.

Надеемся, что изучение подробных примеров выполнения лабораторных работ позволит студентам досконально разобраться с рассмотренными в пособии алгоритмами, успешно справиться с индивидуальными заданиями и, как следствие, подготовиться к программной реализации подобных алгоритмов как в процессе дальнейшего обучения в вузе, так и в профессиональной деятельности при решении различных прикладных задач.

Авторы планируют подготовку и издание других частей данного пособия, посвящённых организации и обработке древовидных структур данных, методам сортировки и поиска, алгоритмам на графах и т. д.

СПИСОК ЛИТЕРАТУРЫ

1. Кнут, Д. Искусство программирования для ЭВМ. Т. 1. Основные алгоритмы / Д. Кнут. – М. : Мир, 1976. – 736 с.

2. Макконнелл, Дж. Основы современных алгоритмов / Дж. Макконнелл. – М. : Техносфера, 2004. – 368 с.

3. Методы программирования : учебное пособие / Ю.Ю. Громов, О.Г. Иванова, Ю.В. Кулаков, Ю.В. Минин, В.Г. Однолько. – Тамбов : Изд-во ФГБОУ ВПО «ТГТУ», 2012. – 144 с.

4. Нивергельт, Ю. Машинный поход к решению математических задач / Ю. Нивергельт, Дж. Фаррар, Э. Рейнголд. – М. : Мир, 1977. – 352 с.

5. Уайс, М.А. Организация структур данных и решение задач на C++ / М.А. Уайс. – М. : ЭКОМ Паблишерз, 2008. – 896 с.

Учебное электронное мультимедийное издание

КУЛАКОВ Юрий Владимирович
БАЙБАКОВ Евгений Анатольевич
СЕВЕНЮК Валерий Васильевич

СТРУКТУРЫ И АЛГОРИТМЫ ОБРАБОТКИ ДАННЫХ

Учебное пособие

Часть 1

Редактор Е. С. Мордасова
Дизайн, структура, навигация
Обложка, упаковка, тиражирование Т. Ю. Зотовой

ISBN 978-5-8265-1909-7

Подписано к использованию 10.05.2018.
Тираж 50 шт. Заказ № 145

Издательско-полиграфический центр
ФГБОУ ВО «ТГТУ»
392000, г. Тамбов, ул. Советская, д. 106, к. 16
Телефон (4752) 63-81-08
E-mail: izdatelstvo@admin.tstu.ru