Дмитриева Марина Валерьевна

АЛГОРИТМЫ НАД СПИСКАМИ

В статье рассматриваются алгоритмы, демонстрирующие различные методы работы со списками: переворот линейного вписка, построение копии списка, вставка в упорядоченный список элемента и удаление из списка всех элементов с заданным значением информационного поля. Проиллюстрированы новые возможности работы с указателями.

ПЕРЕВОРОТ ЛИНЕЙНОГО СПИСКА

Пусть задан некоторый линейный список, рассмотрим алгоритм переворота списка. Требуется перестроить элементы данного списка так, чтобы они располагались в порядке, обратном по отношению к исходному списку. Рассмотрим линейный список, приведенный на рис. 1.

...рассмотрим алгоритм переворота списка ...

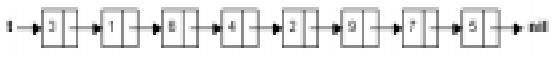


Рис. 1. Линейный цепной список

После преобразования (переворота) должен стать таким, как на рис. 2.

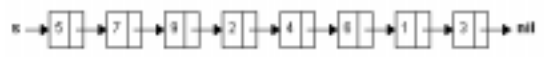


Рис. 2. Список после переворота

При описании алгоритма используем четыре переменные: указатель **t** установлен на первый элемент исходного списка, указатель **t1** установлен на текущий обрабатываемый элемент списка, указатель **s** – на начало уже сформированного списка, переменная **p** играет вспомогательную роль.

Далее необходимо установить указатель **P** на текущий элемент рассматриваемого списка {1}, указатель **t1** переместить на следующий элемент исходного списка {2}, вставить элемент, на который установлен указатель **p**, в начало списка **s** {3}, и изменить указатель на начало перевернутого списка, установив его на только что обработанный элемент {4}.

Указатель **t1** установлен на первый элемент не просмотренной части линейного списка, указатель **s** установлен на «перевернутое» начало списка. Если просмотрен весь исходный список, то переменная **s** является указателем на перевернутый список. Так как требовалось перевернуть исходный список, то переменной **t** следует присвоить значение **s**, и задача будет полностью решена. Процедура переворота линейного списка приведена в листинге 1.

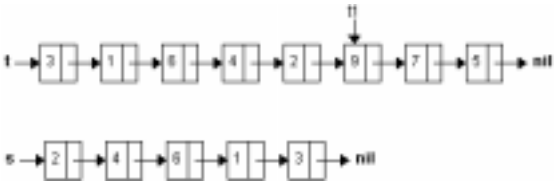


Рис. 3. Список после переворота первых элементов

На рис. 3 изображена ситуация, когда просмотрены первые элементы списка, но список еще не исчерпан.

Листинг 1. Переворот линейного списка

```
procedure rev_list  (var t: list);
  var p,t1,s: list;
begin t1 := t; s := nil;
      while t1 <> nil do
        begin
          p := t1; {1}
          t1 := t1^.next; {2}
          p^.next := s {3}
          s := p {4}
        end;
       t := s
end
```

СОЗДАНИЕ КОПИИ СПИСКА

В некоторых случаях требуется сохранить исходный список, а преобразования выполнять над его копией. Опишем процедуру, которая по заданному списку строит его копию. Пусть **t1** является указателем на текущий рассматриваемый элемент списка, переменная **p** является указателем на первый элемент списка, являющегося копией исходного, переменная **p1** указывает на последний элемент копии.

Если исходный список пуст, то и копия его считается пустой. Если список не пуст, то обрабатывается его первый элемент, то есть создается копия первого элемента. На созданный элемент установлено два указателя **p** и **p1** (первый и последний элемент списка копии). Указатель **t1** устанавливается на второй элемент списка, если этот элемент в списке есть. Остальные действия выполняются с помощью цикла. Процедура создания копии списка приведена в листинге 2.

Рассмотрим тело цикла. В точке программы {1} выполнен запрос на выделение памяти под очередной элемент копии. В {2} скопировали информационное поле рассматриваемого элемента. В {3} «подцепили» построенный элемент к последнему

Листинг 2. Процедура построения копии списка

```
procedure copy_list  (var t, p: list);
  var t1,p1,q: list;
begin if t= nil then p := nil
      else
        begin {список не пуст}
           new (p);  p1 := p;
           p^.info := t^.info;
           t1 := t^.next;
                   {построена копия для первого элемента списка}
           while t1 <> nil do
            begin
               new (q); {1}
               q^.info := t1^.info; {2}
               p1^.next := q; {3}
               p1 := q; {4}
               t1:= t1^.next {5}
            end;
           p1^.next := nil {конец списка}
end
```

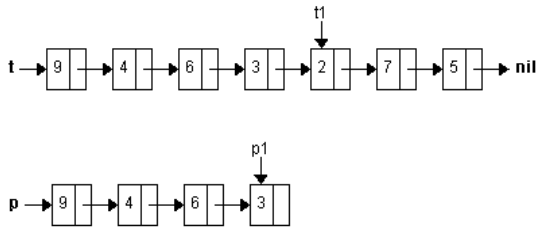


Рис. 4. Список и его копия после просмотра первых элементов

элементу списка копии. В {4} установили указатель **p1** на последний элемент копии. И, наконец, в {5} осуществлен переход к обработке следующего элемента списка.

Рис. 4 соответствует ситуации, когда просмотрены первые элементы списка, для них построена копия, указатель **p** установлен на первый элемент, указатель **p1** на последний элемент построенной копии, указатель **t1** установлен на первый элемент «хвоста» списка, который еще предстоит обработать.

ВСТАВКА И УДАЛЕНИЕ ЭЛЕМЕНТОВ СПИСКА

Гибкость динамических структур данных, в частности списков, состоит в том, что в любой момент времени можно добавить к списку новый элемент, поместив его в любое место списка, или удалить любой элемент, освободив занимаемую элементом память. Продолжим описание процедур работы со списками.

ВСТАВКА ЭЛЕМЕНТА В УПОРЯДОЧЕННЫЙ СПИСОК. *ВАРИАНТ 1*

Пусть задан список, значения информационных полей которого упорядочены по возрастанию. Требуется описать процедуру,

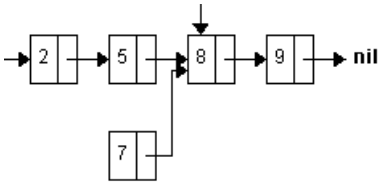
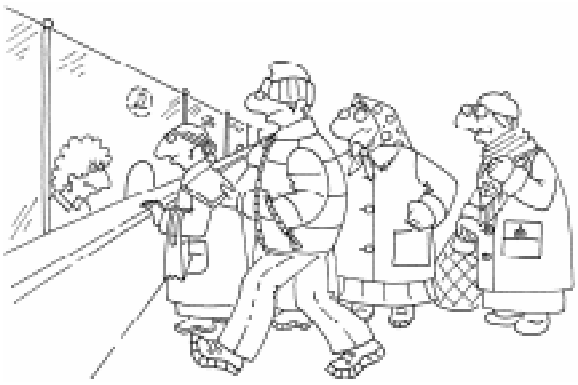


Рис. 5. Вставка элемента в упорядоченный список



...когда новый элемент требуется вставить перед первым элементом списка.

которая вставляет в упорядоченный список элемент с заданным значением информационного поля таким образом, что список остается упорядоченным, то есть вставляется элемент, не нарушая упорядоченности.

Рассмотрим следующий алгоритм решения задачи. Пусть указатель **cur** «пробегает» по списку, когда указатель установлен на какой-либо элемент списка, анализируется информационное поле элемента, непосредственно следующего за элементом с указателем **cur**. Если окажется, что значение информационного поля анализируемого элемента больше образца, то просмотр списка следует прекратить.

Нашли элемент в линейном списке, перед которым следует вставить новый элемент, при этом упорядоченность списка не будет нарушена.

Отдельно рассматриваются случаи, когда список пуст, и случай, когда новый элемент требуется вставить перед первым элементом списка. Процедура вставки элемента в упорядоченный список, представлена в листинге 3.

На рис. 5 изображена ситуация, когда найден элемент, перед которым требуется

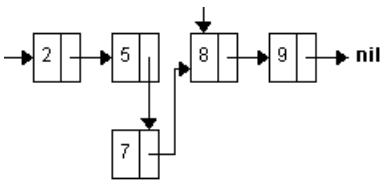


Рис. 6. Список после вставки нового элемента

Листинг 3. Процедура вставки элемента в упорядоченный список. Вариант1

```
procedure add_sort (var t: list; r: elem_list);
  var p,cur: list;
begin new (p); p^.info := r; p^.next := nil;
      if t = nil
      then t := p
      else if r < t^.info
           then begin p^.next := t; t := p end
           else
              begin  cur := t;
                     while cur^.next <> nil
                     do if cur^.next^.info > r
                        then break
                        else cur := cur^.next;
                     p^.next := cur^.next; {1}
                     cur^.next := p {2}
              end
end
```

вставить новый элемент списка. В точке процедуры {1} установлена связь между новым элементом и тем, перед которым элемент вставляется.

Список на рис. 6 соответствует точке {2}, когда изменен указатель рассматриваемого элемента, он устанавливается на новый, только что созданный элемент, тем самым элемент включен в список.

ВСТАВКА ЭЛЕМЕНТА В УПОРЯДОЧЕННЫЙ СПИСОК. *ВАРИАНТ 2*

Рассмотри еще один вариант решения задачи о вставки элемента в упорядоченный список. В языке Turbo Pascal определена операция `@`, выдающая указатель на переменную, являющуюся операндом этой операции. В следующей процедуре переменная cur описана как указатель элемента типа **`list`**, который в свою очередь является указателем на элемент типа **`elem`**. Можно сказать, что переменная cur является указателем на указатель. В точке программы {1} ситуация такая, как на рис. 7.

Процедура вставки элемента в упорядоченный список приведена в листинге 4.

Для лучшего понимания работы процедуры следует рассмотреть случаи вставки элемента перед первым элементом списка, за последним элементом списка, вставки в середину списка. Для каждого из случаев построить конкретные списки и проследить, как изменяются связи между элементами в разных точках процедуры.

Рассмотрим задачу исключения из списка элементов...

УДАЛЕНИЕ ИЗ СПИСКА ЭЛЕМЕНТОВ С ЗАДАННЫМ ИНФОРМАЦИОННЫМ ПОЛЕМ

Рассмотрим задачу исключения из списка элементов с заданным информационным полем. Опишем процедуру, которая удаля-

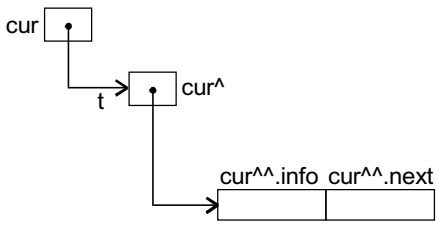


Рис. 7. Просмотр первого элемента списка

Листинг 4. Вставка элемента в упорядоченный список. *Вариант 2*

```
procedure inord (var t: list; r: elem_list);
var q: list;
    cur: ^list;
begin new(q);
      if q = nil then begin writeln ('нет памяти'); exit end
      else
       begin cur := @t; {1}
             while cur^ <> nil do
               if cur^^.info >= r
               then break
               else cur := @(cur^^.next);
             q^.info := r;
             q^.next := cur^;
             cur^ :=q
       end
end
```

ет из списка все элементы, информационные поля которых совпадает с заданным значением.

Как и в предыдущей процедуре, используется операция получения адреса объекта. При просмотре элементов списка используется указатель, значение которого также является указателем. При исключении элемента из списка, память, занимаемая элементом, возвращается системе. Текст процедуры удаления из списка элементов с заданным информационным полем приведен в листинге 5.

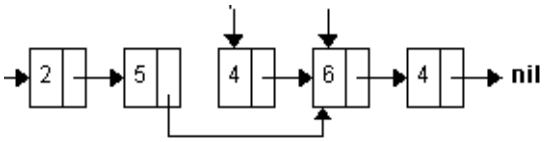


Рис. 8. Состояние списка перед удалением элемента

На рис. 8 представлен список, когда элемент, который требуется удалить, уже найден, связь между предыдущим и следующим элементом измененного списка установлена.

Следующим шагом при работе алгоритма будет возвращение памяти, занимаемой элементом списка, системе. Как и в преды-

Листинг 5. Удаление из списка элементов с заданным информационным полем

```
procedure del_list (var t:list; r:elem_list);
var d: list;
    cur: ^list;
begin  cur := @t;
       while cur^ <> nil do
         begin
      if cur^^.info = r
      then {исключаем элемент}
        begin d := cur^;
              cur :=d^.next;
              dispose (d)
        end
       else
         cur :=@(cur^^.next)
   end
end
```

дущем случае, работу процедуры легче понять, если при отслеживании значений переменных в разных точках процедуры на конкретном рисунке изменять связи в соответствии с операторами процедуры.

ЗАДАЧИ

1. Опишите процедуру, которая по исходному списку строит «перевернутый» список, оставляя исходный список без изменения.

2. Опишите процедуру, которая по двум упорядоченным спискам строит третий, который является результатом слияния первых двух. Исходные списки должны остаться без изменений.

3. Опишите функцию, которая проверяет, является ли список симметричным. В симметричном списке информационное поле первого элемента совпадает с информационным полем последнего элемента, информационное поле второго элемента с информационным полем предпоследнего элемента и т. д.

4. Опишите процедуры, реализующие теоретико-множественные операции, если каждое из множеств является списком (операции объединения, пересечения, разности).

5. Два полинома степени `n` и `m` представляются в виде списка. Информационное поле элемента списка содержит коэффициент и степень. Напишите процедуры, которые строят полином и вычисляют сумму, разность, произведение заданных полиномов. Результирующий полином также представляется списком.

Дмитриева Марина Валерьевна, доцент кафедры информатики математико-механического факультета Санкт-Петербургского государственного университета.