Дмитриева Марина Валерьевна

ОРГАНИЗАЦИЯ ДАННЫХ В ВИДЕ ОЧЕРЕДИ

При решении многих задач, в частности, задач моделирования, бывает полезной структура данных, называемая очередью. В статье рассматриваются способы представления очереди, реализация операций с очередью при представлении очереди списком. Приводится алгоритм сквозного обхода бинарного дерева с помощью очереди. Рассматриваются бинарные деревья поиска: построение и обходы.

Очередью будем называть последовательность элементов одного и того же типа. Добавление элементов производится в один конец очереди, а удаление происходит из другого конца.

Тот конец очереди, из которого производится удаление элементов, называется головой очереди (**head**), а другой ее конец, в который происходит добавление элементов – хвостом очереди (**tail**) (рис. 1).

Набор операций над очередью аналогичен набору операций над стеком, однако реализация этих операций должна учитывать наличие двух концов очереди. В очередь можно добавить элемент или поставить в очередь. Из очереди можно убрать элемент или исключить из очереди. Операция инициализации очереди подразумевает выполнение необходимых действий перед началом работы с очередью. Во многих случаях требуется проверка, содержит ли очередь элементы, то есть не является ли она пустой. Полезными являются операции, которые позволяют получить элемент, являющийся головой очереди, и элемент, являющийся хвостом очереди. Операция «опустошение очереди» удаляет из очереди все элементы.

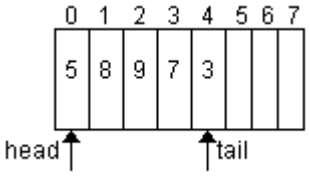

Рис. 1. Очередь с первым и последним элементом

ПРЕДСТАВЛЕНИЕ ОЧЕРЕДИ МАССИВОМ

Очередь можно представить в виде массива с двумя индексами. Один из индексов связан с первым элементом очереди, другой – с последним, так как доступным в очереди должен быть первый и последний элементы. Из очереди удаляется первый элемент, в очередь добавляется элемент вслед за последним. На рис. 2 очередь представлена массивом, показано состояние очереди после добавления и удаления некоторых элементов.

Очередью будем называть последовательность элементов одного и того же типа.

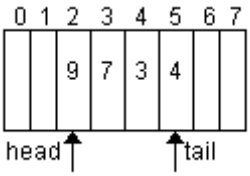

Рис. 2. Представление очереди массивом

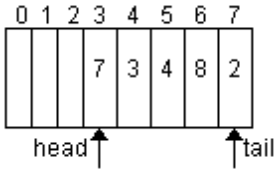

Рис. 3. Возможное состояние очереди

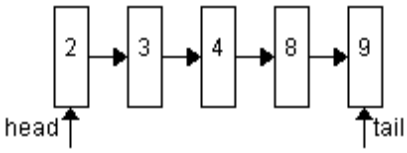

Рис. 4. Реализация очереди в виде линейного списка с указателями на первый и последний элементы

При попытке исключить элемент из очереди требуется следить за тем, чтобы очередь не была пуста, а при попытке добавления элемента в очередь требуется проверять, не переполнилась ли очередь, так как под хранение элементов массива выделяется память фиксированного размера. На рис. 3 представлена ситуация, когда хвост очереди – последний элемент массива, и хотя массив заполнен не полностью, добавить элемент в очередь нельзя.

Чтобы избежать такой ситуации, удобно считать, что очередь, представленная массивом, имеет «кольцевую» структуру. Это означает, что первый элемент массива логически следует за последним.

ПРЕДСТАВЛЕНИЕ ОЧЕРЕДИ СПИСКОМ

Элементы очереди можно связать в список. Для реализации операций над очередью требуется хранить указатели на первый и последний элементы линейного списка. На рис. 4 представлена реализация очереди в виде линейного списка с указателями на первый и последний элементы.

При такой реализации в случае добавления первого элемента в очередь требуется изменить указатель на начало очереди, а при удалении последнего (единственного) элемента из очереди требуется изменить указатель на конец (хвост) очереди.

Приведем еще один вариант реализации очереди в виде списка. Единственный указатель установлен на хвост очереди, а последний элемент в очереди имеет указатель на первый элемент. На рис. 5 представлен такой способ реализации очереди.

Опишем для последнего представления операции над очередью. Тип данных для элементов очереди представлен в листинге 1.

Кольцевой список устроен таким образом, что переменная типа queue указывает на хвост очереди, а ее голова – это непосредственно следующий за хвостом элемент. Таким образом, обеспечивается доступ как к головному, так и к хвостовому элементу.

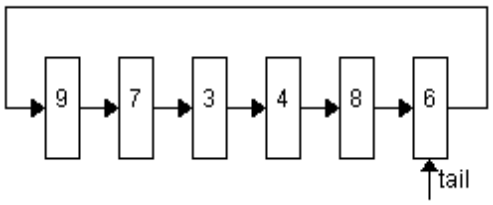

Рис. 5. Реализация очереди «кольцевым» списком

Листинг 1. Определение типа данных для элементов очереди, представленной списком

```
type elem_queue = integer;  {Тип элементов очереди}
     queue = ^elem;
     elem = record info: elem_queue;{элемент очереди}
               next: queue ;{указатель на следующий элемент очереди}
            end;
```

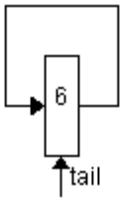

Рис. 6. Добавление в очередь первого элемента

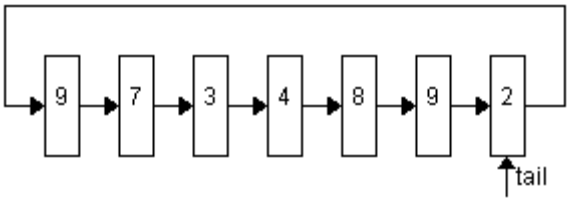

Рис. 7. Добавление в очередь нового элемента

ВСТАВКА ЭЛЕМЕНТА В ОЧЕРЕДЬ

Операция вставки элемента в очередь реализуется с помощью процедуры, имеющей два параметра. Первый параметр определяет очередь, второй – элемент, который требуется поставить в очередь.

Если в очереди не было ни одного элемента, то после добавления элемента очередь будет состоять из одного элемента, который является одновременно и головой и хвостом очереди. Так как хвост очереди должен содержать указатель на голову очереди, то единственный элемент должен содержать указатель на самого себя. Если идентификатор **q** обозначает очередь, то после добавления элемента состояние очереди будет таким, как на рис. 6.

Процедура добавления элемента в очередь представлена в листинге 2.

На рис. 7 показано состояние очереди, изображенной на рис. 5 после добавления нового элемента в очередь.

УДАЛЕНИЕ ЭЛЕМЕНТА ИЗ ОЧЕРЕДИ

Приведем описание функции удаления элемента из очереди. В качестве результата выполнения функции выдается информационное поле удаляемого элемента. Если очередь пуста, то попытка удалить из нее элемент приведет к ошибочной ситуации.

В функции используется локальная переменная **p**, которая устанавливается на удаляемый элемент, то есть на голову оче-

Листинг 2. Процедура добавления элемента в очередь

```
procedure enqueue (var q: queue; e: elem_queue);
var p : queue;
  begin new (p);
     if p = nil then error ('Нехватка памяти');
     p^.info := e;
     if q = nil
     then {добавляется первый элемент}
        p^.next := p {2}
     else
       begin p^.next := q^.next; {3}
             q^.next := p {4}
       end;
     q := p
end;
```

Листинг 3. Функция удаления элемента из очереди

```
function dequeue (var q: queue): elem_queue;
var p : queue; {Указатель на удаляемый элемент}
begin
  if q = nil then error ('Исчерпание очереди');
  p := q^.next;     {Голова очереди}
  if p = q
  then    {Единственный элемент}
       q := nil
   else
       q^.next := p^.next;
   dequeue := p^.info;
   dispose (p)
end;
```

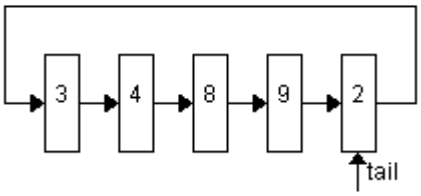

Рис. 8. Удаление из очереди одного элемента

Листинг 4. Инициализация очереди

```
procedure initqueue (var q: queue);
  begin q:= nil end;
```

Листинг 5. Опустошение очереди

```
procedure deletequeue (var q:queue);
var e: elem_queue;
  begin while q <> nil do
          e := dequeue (q)
  end;
```

реди. Если очередь содержит лишь один элемент, то значение выражения **p=q** истинно, после удаления единственного элемента очередь становится пустой.

Если очередь не пуста, то после выполнения оператора **p^.next:=q^.next** головой очереди становится следующий элемент. В качестве результата функции выдается информационное поле удаленного элемента. Вызов **dispose(p)** обеспечивает освобождение памяти, занимаемой удаляемым элементом. Описание функции, осуществляющей удаление элемента из очереди, приведено в листинге 3.

На рис. 8 представлено состояние очереди, изображенной на рис. 7, после удаления из нее двух элементов.

Операция инициализации обеспечивает начальные действия при работе с очередью. В данном случае инициализация делает очередь пустой (листинг 4).

Операция опустошения очереди состоит в обходе очереди и освобождении памяти, занимаемой элементами очереди. В листинге 5 дано описание процедуры опустошения очереди.

Операция, обеспечивающая доступ к первому элементу очереди, реализована с помощью функции **head(q)**, описанной в листинге 6.

Операция, позволяющая получить доступ к последнему элементу очереди, реа-

Листинг 6. Доступ к первому элементу очереди

```
function head (q: queue): elem_queue;
 begin
  if q = nil then error ('Исчерпание очереди');
   head := q^.next^.info
 end;
```

Операция опустошения очереди состоит в обходе очереди и освобождении памяти, занимаемой элементами очереди...

лизована с помощью функции **tail(q)**, описанной в листинге 7.

Для того чтобы проверить, является ли очередь пустой, следует воспользоваться функцией **empty(q)**, описание которой содержится в листинге 8.

Заметим, что последние три функции не изменяют состояние очереди.

Если возникает ошибочная ситуация, то осуществляется вызов функции **error(msg)**, параметром которой является строка – сообщение об ошибке. При возникновении ошибки работы с очередью выдается соответствующее сообщение об ошибке, и работа программы прекращается.

СКВОЗНОЙ ОБХОД БИНАРНОГО ДЕРЕВА С ПОМОЩЬЮ ОЧЕРЕДИ

Приведем процедуру, с помощью которой можно получить часто требующийся способ обхода дерева – сквозной обход. При сквозном обходе вершины дерева посещаются по уровням: сначала корень, затем корни его поддеревьев (вершины второго уровня), затем вершины третьего уровня и так далее.

Такой порядок обхода нельзя получить из рекурсивной процедуры обхода простой перестановкой операторов.

В рассматриваемом случае элементы очереди представляют собой указатели на вершины дерева, обход которого требуется осуществить. Описание типа элемента очереди должно быть таким:

```
type elem_queue = tree;
            {Тип элементов очереди}
```

Если дерево не пусто, то в очередь ставится указатель на вершину дерева. Далее в цикле выполняются следующие действия, пока в очереди есть элементы, ожидающие обработку:

- Выбирается из очереди первый элемент. Это указатель на корень дерева, которое требуется обойти.
- Обрабатывается корень дерева. Обработка корня представляет собой применение процедуры **p** к информационному полю вершины.
- Если дерево имеет левое поддерево, то его корень ставится в конец очереди.
- Если рассматриваемое дерево имеет правое поддерево, то его корень ставится в очередь.

Таким образом будут просмотрены все вершины дерева в требуемом порядке, то есть по уровням. На языке Turbo Pascal описание процедуры приведено в листинге 9.

Листинг 7. Доступ к последнему элементу очереди

```
function tail (q: queue): elem_queue;
  begin
   if q = nil then error ('Исчерпание очереди');
   tail := q^.info
  end;
```

Листинг 8. Проверка, не является ли очередь пустой

```
function empty (q: queue): boolean;
  begin empty := q = nil
end;
```

Листинг 9. Сквозной обход дерева с помощью очереди

```
procedure trav_tree (t: tree, p: action);
var cur: tree;
     q: queue;
begin if t = nil then exit;
      initqueue (q); {инициализация очереди}
      enqueue (q,t); {вставка в очередь ссылки на корень дерева}
      repeat cur := dequeue (q); {выбрали элемент из очереди}
             p(cur^.info); {обработали информационное поле корня}
             if cur^.left <> nil
             then enqueue (q,cur^.left);
                 {поставили в очередь корень левого поддерева}
             if cur^.right <> nil
             then enqueue (q,cur^.right)
                  {поставили в очередь корень правого поддерева}
      until empty (q)
end
```

На рис. 9 изображено бинарное дерево и состояние очереди при обходе его вершин. В каждой вершине дерева число над чертой является информационным полем корня, число под чертой определяет порядок просмотра вершины дерева при обходе.

После того как все вершины дерева будут обработаны, очередь должна стать пустой, порядок просмотра вершин указан с помощью числа, записанного под чертой (рис. 10).

...способ обхода дерева — сквозной обход...

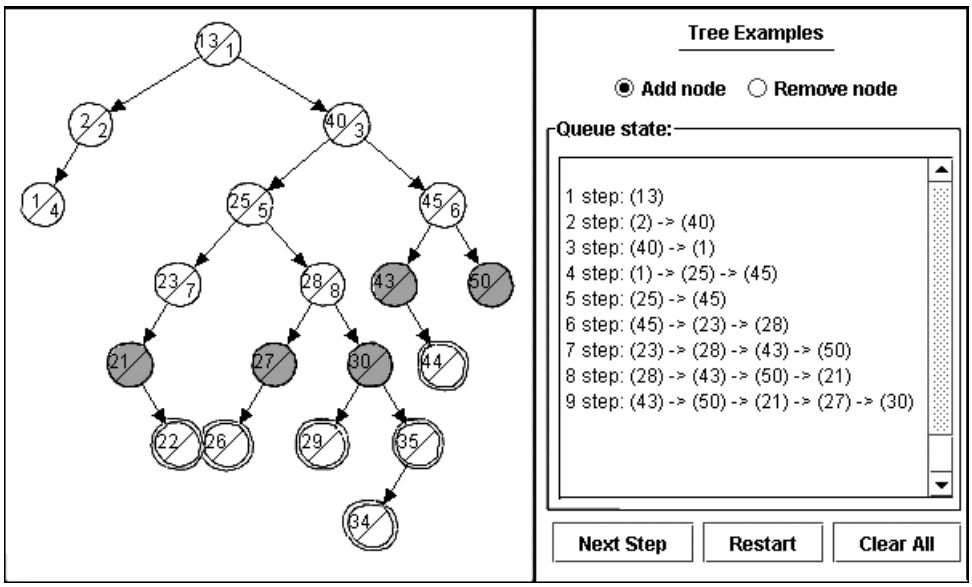

Рис. 9. Сквозной обход бинарного дерева с текущим состоянием очереди

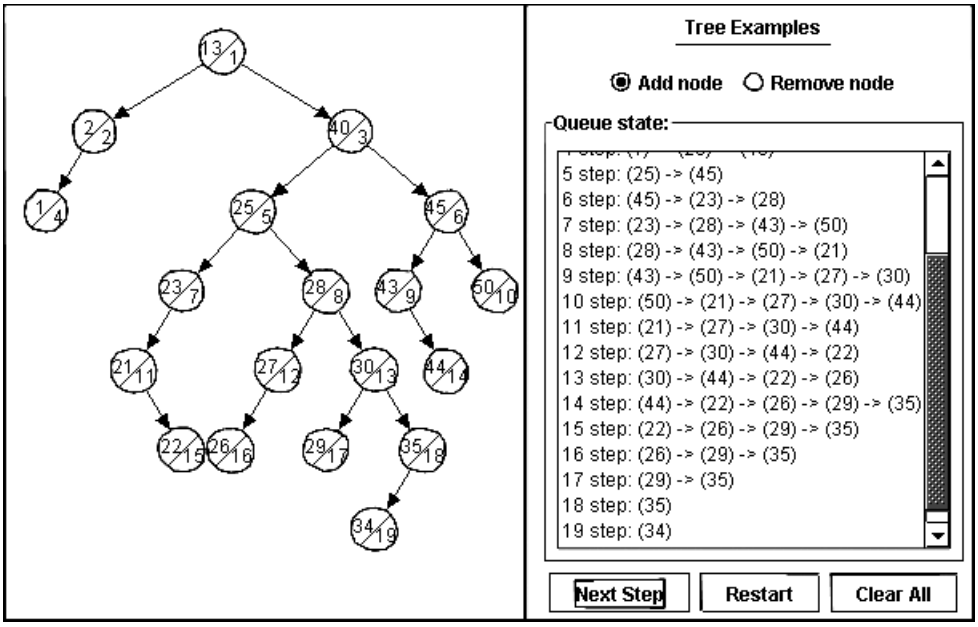

Рис. 10. Бинарное дерево после завершения обхода

БИНАРНОЕ ДЕРЕВО ПОИСКА

Бинарное дерево удобно использовать для быстрого поиска данных. Будем считать, что элементы, которые будут организованы в бинарное дерево, снабжены числовым признаком. Элементы в дереве будем размещать таким образом, чтобы левое поддерево любой вершины `L` содержало только те вершины, значение признака которых меньше, чем значение признака вершины `L`, а правое поддерево – те вершины, значение признака которых больше или равно значению признака вершины `L`. Такие деревья называют деревьями поиска или деревьями сортировки. Дерево, изображенное на рис. 10 является деревом поиска.

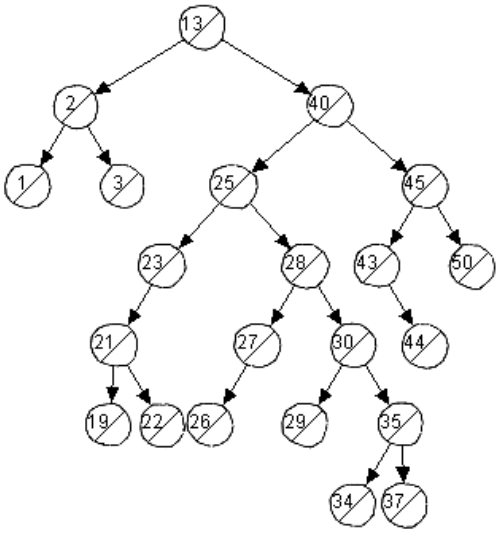

Рис. 11. Дерево поиска после добавления трех вершин

ДОБАВЛЕНИЕ ВЕРШИНЫ В ДЕРЕВО ПОИСКА

Опишем алгоритм добавления вершины с заданным значением `Z` в дерево поиска `T`. Сравнивается значение `Z` со значением, расположенным в корне дерева `T`. Если значение `Z` меньше, то новый узел следует разместить в левом поддереве дерева `T`, в противном случае – в правом. Каждый новый элемент образует лист дерева. При определении места элемента в дереве следует каждый раз просматривать вершины дерева, начиная с корня.

При добавлении элементов со значениями 3, 19, 37 в дерево поиска на рис. 10 дерево примет вид, как на рис. 11.

В листинге 10 приведено описание процедуры добавления в бинарное дерево поиска `t` элемента с информационным

Листинг 10. Процедура добавления вершины в бинарное дерево поиска

```
Procedure add_tree (var t: tree: z: elem_tree);
Var p: tree;
Begin if t=nil
      then begin new(t);
                 t^.info := z;
                 t^.left := nil;
                 t^.right := nil
           end
      else if z < t^.info
           then add_tree (t^.left,z)
           else add_tree (t^.right,z)
end
```

полем **z**. Считаем, что тип информационного поля элемента дерева – целый (**elem_tree=integer**)

АЛГОРИТМЫ ОБХОДА ДЕРЕВА ПОИСКА

Алгоритм левостороннего обхода дерева опишем следующим образом:

- Левосторонний обход левого поддерева (если оно существует).
- Обработка корня.
- Левосторонний обход правого поддерева (если оно существует).

При левостороннем обходе бинарного дерева поиска элементы дерева образуют возрастающую последовательность. Если считать, что процедура обработки вершины дерева помещает информационное поле в стандартный файл вывода, то при левостороннем обходе дерева, изображенного на рис. 11, будет получена последовательность: 1, 2, 3, 13, 19, 21, 22, 23, 25, 26, 27, 28, 29, 30, 34, 35, 37, 40, 43, 44, 45, 50.

Алгоритм правостороннего обхода дерева опишем следующим образом:

- Правосторонний обход правого поддерева (если оно существует).
- Обработка корня.
- Правосторонний обход левого поддерева (если оно существует).

При правостороннем обходе дерева поиска на рис. 11 значения будут расположены в порядке убывания.

...алгоритм добавления вершины...

Листинг 11. Правосторонний обход дерева

```
procedure trav_r (q:tree);
  begin if q<>nil
        then
         begin
          trav_r(q^.right);  {правосторонний обход правого поддерева}
          write(q^.info,' ');{обработка корня}
          trav_r(q^.left);   {правосторонний обход левого поддерева}
         end
  end;
```

Листинг 12. Программа, использующая сквозной обход дерева с помощью очереди

```
{$F+}
procedure out (var z: elem_tree);
begin write(z,' ') end;
{$F-}
begin  writeln ('Введите последовательность чисел, 0- признак конца:');
        read (z); t := nil;
        while z <> 0 do
          begin  searadd (t,z); read (z) end;
        writeln ('Последовательность в порядке убывания');
        trav_r (t);
        writeln ('Последовательность при сквозном обходе дерева');
        trav_tree(t,out)
 end.
```

Если считать, что процедура обработки вершины дерева помещает информационное поле в стандартный файл вывода, то при левостороннем обходе дерева, изображенного на рис. 11, будет получена убывающая последовательность чисел: 50, 45, 44 , ..., 19, 13, 3, 2, 1.

В листинге 12 приведено описание процедуры обработки вершины дерева и раздел операторов программы, осуществляющей построение дерева поиска по последовательности ненулевых чисел (ноль – признак конца), затем выполняющей правосторонний обход дерева и сквозной обход с помощью очереди.

Для дерева, изображенного на рис. 10, при сквозном обходе будет выведена следующая последовательность чисел: 13, 2, 40, 1, 25, 45, 23, 28, 43, 50, 21, 27, 30, 44, 22, 26, 29, 35, 34.

Обработка списков, стеков, очередей, деревьев должна стать одним из инструментов, находящихся в распоряжении программиста, позволяющих решать задачи, в которых требуется представлять и обрабатывать динамические структуры данных.

ЗАДАЧИ

1. Опишите процедуру, которая при обходе бинарного дерева на отдельной строке выходного файла указывает номер уровня дерева и затем информационные поля вершин, расположенных на рассматриваемом уровне.
2. Опишите процедуру, которая выводит вершины бинарного дерева по уровням до заданного номера уровня.
3. Опишите процедуру, которая выводит информационные поля бинарного дерева по уровням, просмотр вершин на каждом из уровней должен выполняться справа налево, выдается дополнительно количество вершин на заданном уровне.
4. Опишите процедуру, которая при сквозном обходе бинарного дерева строит бинарное дерево поиска.
5. Опишите процедуру, которая при сквозном обходе бинарного дерева поиска строит дерево поиска, не содержащее одинаковых элементов.

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