Списки

Односвязанный список

Односвязный список хранит набор элементов, при этом каждый элемент хранит ссылку (указатель) на следующий элемент списка.

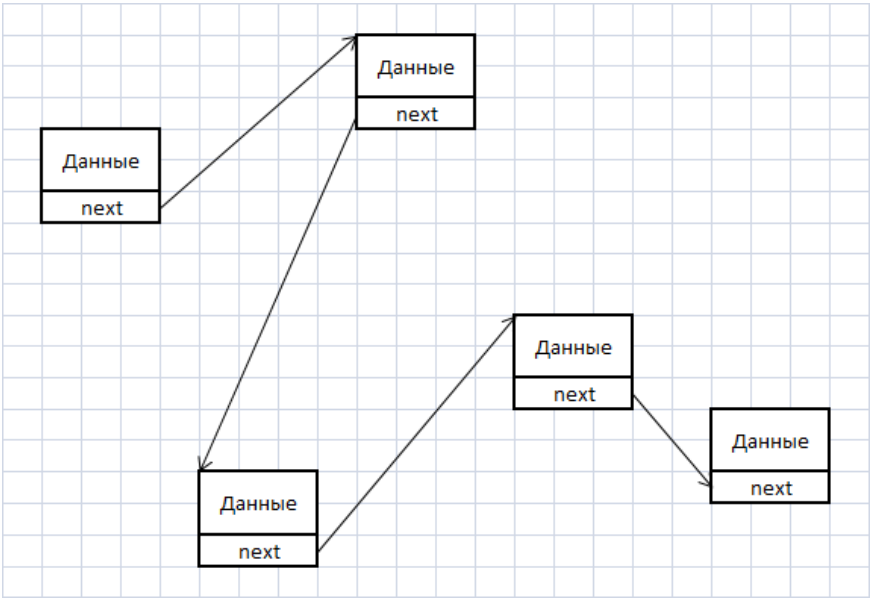

Рис. 1. Пример односвязного списка

По рисунку видно, что элементы списка занимают в памяти несмежные области. Это может быть полезным, когда суммарной свободной памяти достаточно для хранения N элементов, но нет смежной области памяти необходимого размера, что исключает использование массивов.

Элемент списка можно описать следующим кодом:

Листинг 1. Объявление элемента списка

```
type
  TPElem = ^TRElem;
  TRElem = record
    value: integer;
    next: TPElem;
  end;
```

Очень важно разобраться с этим определением, т.к. здесь используется концепция самоотносимых типов. Элемент представлен записью, одно поле которой хранит данные, а второе поле является ссылкой на другую запись такого же типа.

При работе с односвязным списком достаточно помнить только указатель на начало списка. Отталкиваясь от начала списка и двигаясь по указателям `next`, можно обойти весь список до конца. Таким образом с некоторым допущениями указатель на начало списка можно воспринимать как весь список.

Существует вариант реализации, в котором помимо указателя на начало дополнительно хранится и указатель на конец списка. Такую структуру принято называть двусторонним списком. Наличие

ссылки на последний элемент позволяет значительно ускорить процедуру вставки элемента в конец списка, что может быть полезно в некоторых случаях.

При работе с простыми списками чаще всего используются следующие процедуры:

- `InsertFirst` - добавляет элемент в начало списка;
- `DeleteFirst` - удаляет элемент из начала списка;
- `InsertAfter` - добавляет элемент в список после заданного элемента;
- `DeleteAfter` - удаляет заданный элемент из списка.

Иногда вместо того, чтобы определять частные процедуры удаления первого элемента и добавления в начало списка, используют метки. Метка - это особый элемент, который всегда находится в начале (или конце) списка и который не хранит никаких данных. При этом метку нельзя удалять. При использовании меток процедуры InsertFirst и DeleteFirst заменяются процедурами InsertAfter и DeleteAfter, только в качестве параметра в них передаются ссылки меток.

Добавление в начало односвязного списка (Insertfirst)

При добавлении элемента в начало списка необходимо выделить память под новый элемент (команда `new`), записать в полученный элемент добавляемое значение, а потом сделать этот элемент вершиной списка. Для этого в поле `next` нового элемента необходимо записать ссылку на предыдущее начало списка и переопределить эту ссылку.

Листинг 2. Добавление элемента в начало односвязного списка

```
procedure InsertFirst(newval: integer);
var
NewElem: TPElem;
begin
  new(NewElem);
  NewElem^.value:=newval;
  NewElem^.next:=List;
  List:=NewElem;
end;
```

Удаление элемента из начала односвязного списка (DeleteFirst)

При удалении первого элемента достаточно переместить указатель начала списка на второй элемент. Но тогда удалённый элемент будет по-прежнему занимать память, чтобы избежать этого, ссылку на удаляемый элемент предварительно сохраняют, а потом освобождают занимаемую элементом память.

Листинг 3. Удаление элемента из начала односвязного списка

```
procedure DeleteFirst;
var
tmp: TPElem;
begin
  tmp:=List;
  List:=List^.next;
  dispose(tmp);
end;
```

Добавление элемента в произвольную позицию односвязного списка (InsertAfter)

Добавление элемента в произвольную позицию практически ничем не отличается от добавления элемента в начало списка. Просто вместо ссылки на начало списка необходимо править ссылки элемента, после которого добавляется новый элемент.

Листинг 4. Добавление элемента в произвольную позицию односвязного списка

```
procedure InsertAfter(var elem: TPElem; newval: integer);
var
NewElem: TPElem;
begin
  new(NewElem);
  NewElem^.value:=newval;
  NewElem^.next:=Elem^.next;
  Elem^.next:=NewElem;
end;
```

Удаление элемента из произвольной позиции односвязного списка (DeleteAfter)

Как и в случае с удалением элемента из начала списка, необходимо помнить о том, что память, занятую удаляемым элементом, необходимо освобождать.

Листинг 5. Удаление элемента из произвольной позиции односвязного списка

```
procedure DeleteAfter(elem: TPElem);
var
tmp: TPElem;
begin
  tmp:=elem^.next;
  elem^.next:=tmp^.next;
  dispose(tmp);
end;
```

Другие процедуры и функции работы с односвязными списками

Проверку на пустоту списка можно выполнить, сравнив указатель на начало с NIL. В случае, если они не равны, то список не пуст и содержит не менее одного элемента.

Некоторый интерес представляет процедура поиска элемента в списке. В отличие от массива, невозможно произвольно манипулировать индексами и переходить сразу от первого элемента к пятому или десятому, для односвязных списков возможен только последовательный доступ, а значит, даже для упорядоченного списка невозможно использовать бинарный поиск. Ниже представлена сравнительная таблица сложности некоторых базовых процедур для уже изученных структур данных:

Табл. 1. Сравнительная таблица сложности базовых процедур в массивах и односвязных списках

Операция	Массив	Упорядоченный массив	Односвязный список
Вставка	$O(C)$	$O(N)$	$O(C)$
Удаление	$O(N)$	$O(N)$	$O(C)$
Поиск	$O(N)$	$O(logN)$	$O(N)$

По скорости выполнения операций вставки и удаления список превосходит обычный массив, но у списков есть существенные недостатки: дополнительный расход памяти на хранение указателей и невозможность произвольного доступа к элементам.

Двусвязные списки

При использовании односвязных списков не возникает проблем при добавлении элементов после заданного, но вставить элементы перед заданным бывает весьма затруднительно: для этого придётся пройтись по всему списку и найти элемент, который сейчас находится перед заданным, и уже для него выполнить процедуру `InsertAfter`. Аналогичные проблемы возникают и при удалении элементов.

Эффективно бороться с этими недостатками помогают двусвязные списки. В такой структуре каждый элемент хранит ссылку на предыдущий и последующий элементы. Рассматривая двусвязные списки, есть смысл более подробно рассмотреть работу с метками, о которых было упомянуто раньше. Т.к. элементы списка хранят ссылки как на предыдущие, так и на последующие элементы, необходимо использовать метки начала и конца списка, которые назовём DLF и DLL соответственно.

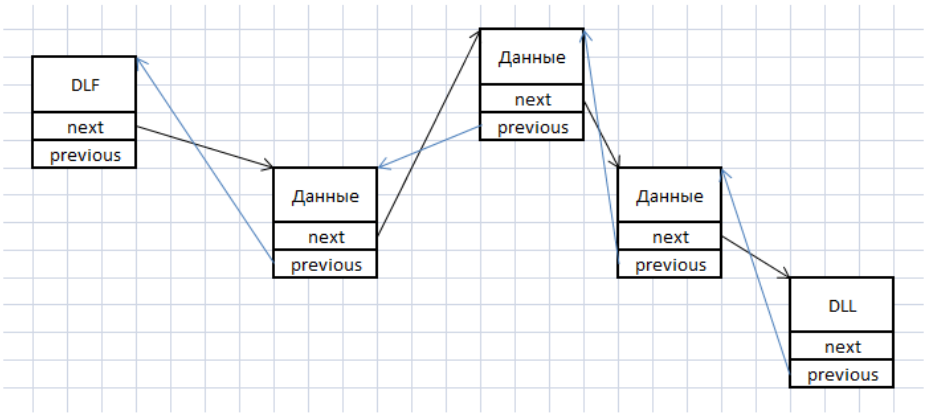

Рис. 2. Пример двусвязного списка

При такой организации на каждый элемент данных хранится уже по два указателя, которые по сути являются дополнительным расходом памяти. Но такая организация добавляет к плюсам списков возможность легко добавлять и удалять элементы в любое место.

Ниже представлены четыре процедуры:

- `InitList` -инициализация списка;
- `InsertAfter` - добавляет элемент после выбранного;
- `InsertBefore` - добавляет элемент перед выбранным;
- `Delete` - удаляет выбранные элемент.

Инициализация двусвязного списка (InitList)

Необходимость в отдельной процедуре инициализации возникает из-за использования меток. На самом деле нет необходимости устанавливать `DLL^.next` и `DLF^.prev` в `NIL`, но всё же рекомендуется это делать, чтобы в случае неправильного использования списка программа сообщала бы об ошибке обращения к пустому указателю.

Листинг 6. Инициализация двусвязного списка

```
procedure InitList;
begin
  NEW(DLF); NEW(DLL);
  DLF^.value:=0; DLF^.prev:=nil; DLF^.next:=DLL;
  DLL^.value:=0; DLL^.prev:=DLF; DLL^.next:=nil;
end;
```

Добавление элемента в двусвязный после выбранного (InsertAfter)

Эта процедура аналогична рассмотренной ранее процедуре добавления элементов в односвязный список. Главное отличие в том, что приходится править по две ссылки у каждого элемента, кроме того запрещено добавлять элементы после метки `DLL`.

Листинг 7. Добавление элемента в двусвязный список после выбранного

```
procedure InsertAfter(elem: TPDList; newval: integer);
var
NewElem, tmp: TPDList;
begin
  New(NewElem); NewElem^.value:=newval;
  tmp:=elem^.next;
  elem^.next:=NewElem;
  NewElem^.next:=tmp;
  tmp^.prev:=NewElem;
  NewElem^.prev:=elem;
end;
```

Добавление элемента в двусвязный список перед выбранным (InsertBefore)

Очевидно, что эта процедура во многом похожа на процедуру `InsertAfter`. Ограничения накладываемые на неё тоже аналогичны: нельзя добавлять элементы перед `DLF`.

Листинг 8. Добавление элемента в двусвязный список перед выбранным

```
procedure InsertBefore(elem: TPDList; newval: integer);
var
NewElem, tmp: TPDList;
begin
  New(NewElem); NewElem^.value:=newval;
  tmp:=elem^.prev;
  tmp^.next:=NewElem;
  NewElem^.next:=elem;
  elem^.prev:=NewElem;
  NewElem^.prev:=tmp;
end;
```

Процедура удаления элемента из двусвязного списка (Delete)

Процедура удаления заключается в переопределении указателей предыдущего и последующего элементов. Как и во всех процедурах удаления, важно не забыть освободить память, занимаемую удалённым элементом.

Листинг 9. Удаление элемента из двусвязного списка

```
procedure Delete(elem: TPDList);
var
```

```
before,after: TPDList;
begin
  before:=elem^.prev; after:=elem^.next;
  before^.next:=after;
  after^.prev:=before;
  Dispose(elem);
end;
```

Другие процедуры и функции работы с двусвязными списками

Как и в случае с односвязными списками, сложность процедур добавления и удаления элементов пропорциональна $O(C)$. Поиск по-прежнему пропорционален $O(N)$, воспользоваться бинарным поиском нельзя, т.к. списки занимают несмежные области памяти.