```
        Node *pv = new Node;
        pv->d = d;  pv->next = 0;    pv->prev = 0;
        return pv;
        }

// Функция добавления элемента в конец списка
        void add(Node **pend, int d)
        {
        Node *pv = new Node;
        pv->d = d;  pv->next = 0;  pv->prev = *pend;
        (*pend)->next = pv;
        *pend = pv;
        }
```

Заказ №_____от «___»_____200 ____Тираж_____экз.
Изд-во СевНТУ

Министерство образования и науки Украины

Севастопольский национальный технический университет

ОБРАБОТКА СТРУКТУР ДАННЫХ

Методические указания

**к выполнению лабораторных работ
по дисциплине
«СИСТЕМНОЕ ПРОГРАММИРОВАНИЕ»
для студентов дневной и заочной форм обучения
направления 6.050101, специальности 7.091501
«Компьютерные системы и сети»**

Севастополь

2008

УДК 681.3

Обработка структур данных. Методические указания к выполнению лабораторных работ по дисциплине «Системное программирование» / А.И. Тертычный, Е.М. Шалимова. - Севастополь: Изд-во СевНТУ, 2008. -32с.

Цель методических указаний – оказание помощи студентам в выполнении лабораторных работ по дисциплине. Методические указания содержат краткое изложение основных теоретических положений, примеры обработки рассматриваемых структур данных, задания на лабораторные работы, порядок их выполнения и требования к отчетам, а также список рекомендованной литературы.

Методические указания рассмотрены и утверждены на заседании кафедры кибернетики и вычислительной техники, протокол № 9 от 28 мая 2008 года.

Рецензент:
Карлусов В.Ю. , к.т.н., доцент кафедры информационных систем.

Допущено учебно-методическим центром СевНТУ в качестве методических указаний.

Ответственный за выпуск:
заведующий кафедрой кибернетики и вычислительной техники профессор Скатков А.В.

Приложение А. Пример программы обработки двунаправленного списка на языке С++

Программа формирует двунаправленный список из 5 элементов (со значениями 1,2,3,4,5) и выводит его на экран.

Указатель на начало списка обозначен pbeg, на конец списка — pend.

```
#include <iostream.h>
struct Node
{ int d;
Node *next;
Node *prev; };

Node * first(int d);
void add(Node **pend, int d);

int main()
{
// Формирование первого элемента списка
Node *pbeg = first(1);
Node *pend = pbeg;

// Добавление в конец списка четырех элементов 2. 3. 4. и 5;
for (int i = 2; i<6; i++)   add(&pend, i);

 // Вывод списка на экран
Node * pv= pbeg;
while (pv) {
cout<< pv->d << '  ';
pv = pv->next;  }

return 0;
}
// Функция формирования первого элемента
Node * first(int d)
{
```

Библиографический список

1. Вирт Н. Алгоритмы и структуры данных/ Н.Вирт– М.: Мир, 1989. - 360 с.
2. Павловская Т.А. С/С++. Программирование на языке высокого уровня/ Т.А. Павловская -СПб.:Питер, 2002. – 464с.
3.Кнут Д. Искусство программирования для ЭВМ, т.1/ Д. Кнут – М.: Мир, 2002. – 760 с.
4. Лебедев В.Н. Введение в системы программирования/ В.Н. Лебедев – М.: Статистика, 1975. – 309 с.

Содержание

1. Общие положения, порядок выполнения лабораторных работ и требования к отчетам

В соответствии с программой дисциплины «Системное программирование» студент должен выполнить три лабораторные работы по теме «Обработка структур данных». Все лабораторные работы рассчитаны на 4 академических часа.

По результатам выполнения каждой лабораторной работы оформляется и защищается отчет, который должен содержать следующие основные элементы:

- титульный лист;
- цель работы;
- постановку задачи с указанием варианта задания;
- описание используемых структур данных;
- описание алгоритма решения задачи;
- спецификации подпрограмм;
- тестовые примеры;
- текст программы;
- результаты и выводы.

Отчёты по лабораторной работе оформляются и защищаются каждым студентом индивидуально. При защите лабораторной работы студент должен продемонстрировать работу программы.

Разработанные программы должны обеспечивать диалог с помощью меню и проверку правильности задания исходной информации; логически законченные фрагменты должны быть оформлены в виде подпрограмм, все необходимые данные которым передаются через список параметров.

Тип обхода:

1 – прямой;
2 – обратный;
3 – концевой.

5.3. Контрольные вопросы

1. Дать определение древовидных структур данных.
2. Дать определение бинарных деревьев.
3. Перечислить основные операции с деревьями.
4. Дать определение дерева двоичного поиска.
5. Как выполняется операция добавления элемента в дерево?
6. Как выполняется операция удаления элемента из дерева?
7. Перечислить способы обхода бинарного дерева.
8. Сформулировать правило обхода бинарного дерева в прямом порядке.
9. Сформулировать правило обхода бинарного дерева в обратном порядке.
10. Сформулировать правило обхода бинарного дерева в концевом порядке.

5.2. Задание на лабораторную работу

Разработать программу формирования дерева двоичного поиска и выполнения заданной операции с ним. В программе предусмотреть вывод элементов дерева в соответствии с заданным типом обхода. Варианты задания приведены в таблице 5.1.

Первый символ кода варианта задает операцию, второй - тип обхода.

Таблица 5.1–Варианты задания

№ варианта	1	2	3	4	5	6
Код	11	21	31	41	51	61
№ варианта	7	8	9	10	11	12
Код	71	81	12	22	32	42
№ варианта	13	14	15	16	17	18
Код	52	62	72	82	13	23
№ варианта	19	20	21	22	23	24
Код	33	43	53	63	73	83

Заданная операция:

1. Вычислить среднее арифметическое всех элементов.
2. Определить количество нечетных элементов.
3. Определить количество элементов, значения которых лежат в диапазоне от ***a*** до ***b***.
4. Удалить элемент с ключом ***k***.
5. Определить количество элементов, значения которых больше ***X.***
6. Вычислить среднее арифметическое минимального и максимального элементов.
7. Определить количество элементов, значения которых кратны ***X.***
8. Удалить “правого потомка” элемента с ключом ***k***.

2. Классификация структур данных

В большинстве случаев информация, обрабатываемая вычислительной машиной, представляет собой организованную совокупность данных, что означает наличие между ее элементами определенных структурных отношений.

Рассмотрим некоторые структуры данных, нашедшие широкое применение при разработке программного обеспечения, и их классификацию. Структуры данных будем полагать состоящими из элементов, каждый из которых, в свою очередь, может быть некоторой структурой. Так образуются сложные иерархические структуры. Элементарным объектом структуры данных будем считать запись. Это понятие связано с характеристикой типов или множеств значений, которые могут принимать элементы структуры. В общем случае запись может содержать несколько полей различных типов.

Примем классификацию, в соответствии с которой структуры данных подразделяются на статические и динамические [1-3]. Отличительным признаком статических структур является фиксированный размер выделяемой им памяти. При этом память выделяется на этапе компиляции. Для данных динамического типа память выделяется по мере необходимости на этапе выполнения программы, таким образом, данные динамического типа способны в процессе вычисления изменять не только свои значения, но и структуру.

Еще одним признаком классификации структур данных является наличие одномерного упорядочения их записей. В соответствии с этим признаком структуры данных принято делить на линейные и нелинейные.

При разработке программной системы на выбор той или иной структуры, как правило, определяющим образом влияет множество операций, которые будут выполняться с данными. Поэтому для каждой структуры данных будем рассматривать наиболее типичные операции, которые с этими данными выполняются.

3. ЛАБОРАТОРНАЯ РАБОТА №1. Формирование и обработка линейных списковых структур данных

Цель работы: изучение принципов организации и программирование основных операций, выполняемых с линейными списковыми структурами.

3.1. Линейные структуры данных. Основные теоретические положения

Линейные структуры, в соответствии с [1,2], – это множество, состоящее из $n \geq 0$ записей $X_1, X_2, \ldots, X_n$, структурные свойства которого определяются одномерным относительным положением записей: X_1 является первой записью; если $1 < k < n$, то k-ой записи X_k предшествует (k-1)-ая X_{k-1}, а за ней следует (k+1)-ая X_{k+1}; X_n является последней записью.

Одной из самых распространенных линейных статических структур является одномерный массив. Для хранения массива в памяти, как правило, используется ее последовательное распределение, подразумевающее размещение элементов массива в последовательных ячейках памяти. Местоположение массива в памяти определяется адресом его первого элемента.

Для хранения динамической линейной структуры используется более гибкая схема, называемая связанным распределением. Каждая запись динамической линейной структуры, называемой списком, содержит указатель на следующую запись. Такой список называется однонаправленным (односвязным). Указатель последней записи равен коду признака конца списка, а указатель на первую запись задается дополнительным параметром, называемым переменной связи. Если в каждый элемент списка добавить еще один указатель – на предыдущий элемент, то получится двунаправленный список (двусвязный). Тип списка принципиально не влияет на реализацию операций, поэтому далее ограничимся рассмотрением односвязных списков, а в Приложении А приведем пример программы обработки двунаправленного списка.

Пусть каждый элемент списка содержит два поля: INFO, содержащее значение элемента данных X_k, и LINK, содержащее адрес следующего элемента X_{k+1}. Переменную связи назовем FIRST. Введенные обозначения будем использовать в программных иллюстрациях операций над списками.

Связанное распределение требует дополнительное пространство в памяти для связей. Но при его использовании

Операция линеаризации дает линейную расстановку узлов дерева в результате применения какого-либо правила их перебора (обхода). В процессе обхода каждый узел дерева проходится один и только один раз.

Существует три способа обхода бинарных деревьев:

1) **в прямом порядке**: попасть в корень, пройти левое поддерево, пройти правое поддерево;

2) **в обратном порядке**: пройти левое поддерево, попасть в корень, пройти правое поддерево;

3) **в концевом порядке**: пройти левое поддерево, пройти правое поддерево, попасть в корень.

Выполним обход дерева, изображенного на рисунке 10, каждым из трех перечисленных способов.

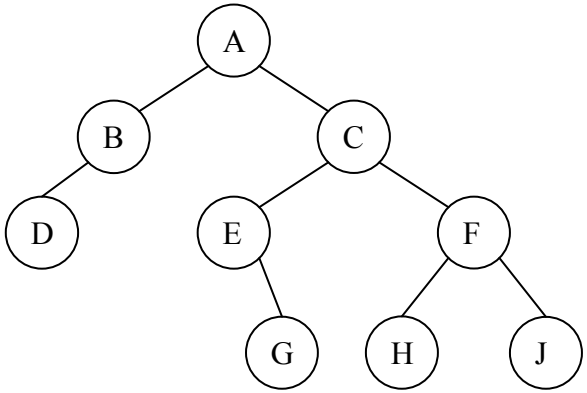
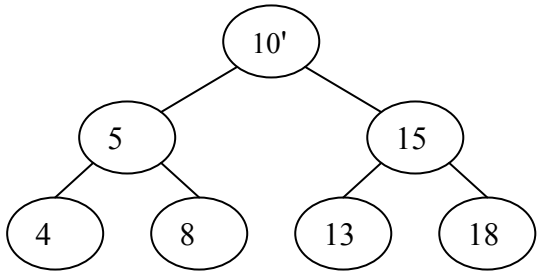
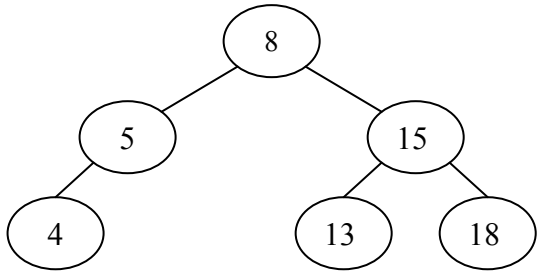


Рисунок 10 – Пример древовидной структуры

В результате получим линейные упорядоченности узлов:

A B D C E G F H J – результат обхода в прямом порядке;
D B A E G C H F J – результат обхода в обратном порядке;
D B G E H J F C A – результат обхода в концевом порядке.





или

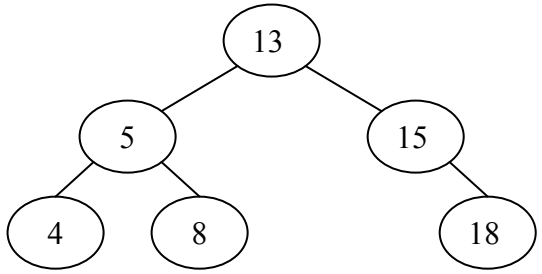
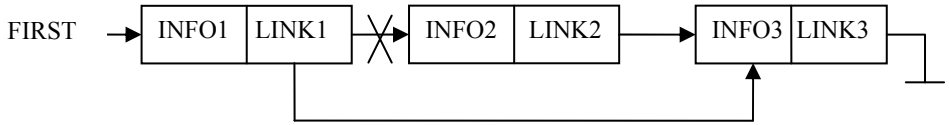
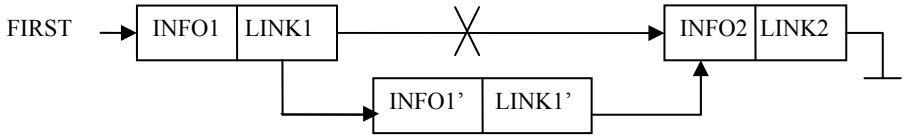


Рисунок 9 – Иллюстрация удаления из дерева вершины с двумя потомками

существенно упрощаются такие операции над линейной структурой, как удаление ее элемента и включение в нее нового элемента, которые сводятся, как показано на рисунке1, к изменению соответствующих полей LINK элементов структуры.



а)



б)

Рисунок 1 – Преобразование линейного списка
а) удаление элемента 2;
б) включение элемента 1'.

Особенно полезны списковые структуры в тех случаях, когда в памяти ЭВМ требуется разместить несколько динамически изменяющихся структур данных (массивов, строк, таблиц и т.д.) с заранее неизвестным числом элементов. Кроме того, использование списков целесообразно, когда требуется упорядочить элементы данных без их физического перемещения или связать воедино элементы данных, размещенные в разных местах памяти [1-4].

Для описания динамических структур данных удобным средством многих языков программирования являются переменные - указатели.

Приведем примеры реализации прохода списковой структуры, включения в нее нового элемента и удаления из нее элемента средствами языка Паскаль. При этом будем полагать, что список задается переменной связи FIRST и описание списковой структуры имеет вид:

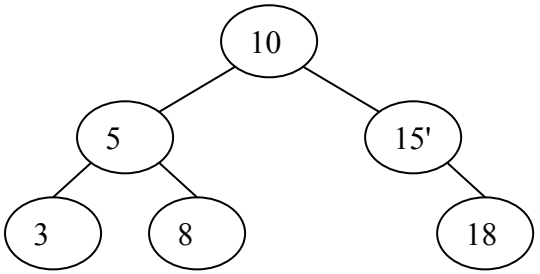
```
type
  NLINK=^LIST;
  LIST=record
        INFO:integer;
        LINK:NLINK
       end;
var FIRST:NLINK;
```

Процедура прохода списка LISTPRINT выполняет вывод всех его элементов, начиная с элемента, указанного FIRST, и заканчивая элементом, содержащим nil в поле LINK.

```
procedure LISTPRINT(FIRST:NLINK);
 var
   BASE:NLINK;
 begin
   BASE:=FIRST;
   while BASE<>nil do
    begin
     writeln(BASE^.INFO);
     BASE:=BASE^.LINK
    end;
 end;
```

Процедура включения элемента LISTADD добавляет в список новый элемент после элемента с заданным номером K, если в списке не менее K элементов. В противном случае элемент добавляется к списку в качестве последнего. Поле данных нового элемента при этом заполняется значением, заданным переменной NEWINFO.

```
procedure LISTADD(K, NEWINFO:integer; var FIRST:NLINK);
    var
      I:integer;
      NEWNODE,BASE:NLINK;
  begin
        new(NEWNODE);
        NEWNODE^.INFO:=NEWINFO;
```

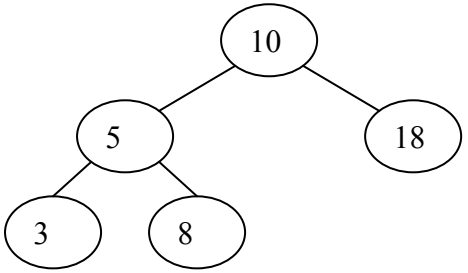
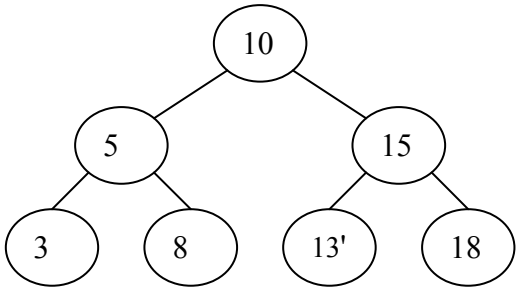


Рисунок 8 – Иллюстрация удаления из дерева вершины с одним потомком

Иллюстрация исключения элементов из дерева представлена на рисунках 7 – 9, где символом ' отмечен удаляемый элемент.



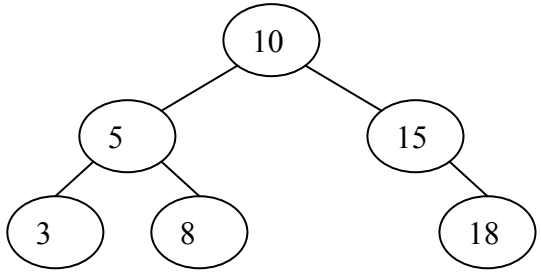


Рисунок 7 – Иллюстрация удаления из дерева терминальной вершины


```
  if FIRST<>nil then
    begin
          BASE:=FIRST;
          for I:=1 to K-1 do
          if BASE^.LINK<>nil then BASE:=BASE^.LINK;
          NEWNODE^.LINK:=BASE^.LINK;
          BASE^.LINK:=NEWNODE;
    end
  else begin FIRST:=NEWNODE;
          NEWNODE^.LINK:=nil;
       end;
end;
```

Процедура удаления элемента LISTDEL удаляет из списка элемент с заданным номером K, если в списке не менее K элементов. В противном случае процедура не изменяет структуру списка.

```
procedure LISTDEL(K:integer; var FIRST:NLINK);
 var
  I:integer;
  DELNODE,BASE:NLINK;
 begin
  if FIRST<>nil then
    begin
      BASE:=FIRST;
      for I:=1 to K-2 do
      if BASE<>nil then BASE:=BASE^.LINK;
      DELNODE:=BASE^.LINK;
      BASE^.LINK:=DELNODE^.LINK;
    end;
  end;
```

В программировании часто встречаются линейные структуры, в которых включение и исключение или доступ к значениям производятся только в первой или последней записи. Эти структуры имеют специальные названия: стек, очередь, дек [1,2].

При работе с этими структурами будем полагать, что процедура извлечения элемента предполагает и его удаление.

СТЕК – линейная структура, в которой все включения и исключения и всякий доступ делаются в одном конце структуры.

Из стека всегда извлекается "младший" элемент из имеющихся в списке, т.е. тот, который был включен позже других. Этот элемент называют вершиной стека.

Приведенная ниже процедура включения элемента в стек STACKADD полагает стек заданным указателем TOP на его вершину, а поле данных нового элемента заполняет значением, заданным переменной NEWINFO.

```
procedure STACKADD(NEWINFO:integer; var TOP:NLINK);
 var
  NEWNODE:NLINK;
 begin
  new(NEWNODE);
  NEWNODE^.INFO:=NEWINFO;
  NEWNODE^.LINK:=TOP;
  TOP:=NEWNODE;
 end;
```

Процедура STACKSEL выбирает элемент из стека, заданного указателем TOP на его вершину, и возвращает его значение через переменную X.

```
procedure STACSEL(var TOP:NLINK; var X:INTEGER);
var P:NLINK;
 begin
  if TOP<>nil then begin P:=TOP; X:=TOP^.INFO;
                     TOP:=TOP^.LINK;
                     dispose(P);
                   end;
 end;
```

ОЧЕРЕДЬ – линейная структура, в которой все включения производятся на одном конце структуры, а все исключения и доступ на другом ее конце. Очередь задается двумя указателями: на первую запись (указатель начала очереди) и на последнюю запись (указатель конца очереди). Приведем процедуру включения элемента в очередь QADD, указатель конца которой передается через параметр

Добавление элемента производится на место правого или левого пустого поддерева в зависимости от значения ключа добавляемого элемента.

Иллюстрация включения элемента с ключом 20 в бинарное дерево приведена на рисунке 6.

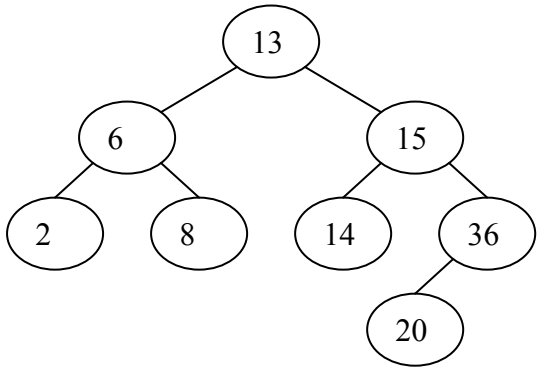


Рисунок 6 – Включение элемента в дерево.

Операция удаления из упорядоченного дерева несколько сложнее операции добавления. Здесь необходимо рассматривать три случая:

1) исключаемый элемент – терминальная вершина (лист);

2) исключаемый элемент – внутренняя вершина с одним потомком;

3) исключаемый элемент – внутренняя вершина с двумя потомками.

Если исключаемый элемент – это терминальная вершина, то соответствующему указателю предка этой вершины присваивается значение ПУСТОГО УКАЗАТЕЛЯ.

Если удаляемый элемент – вершина с одним потомком, то удаляемый элемент заменяется его непосредственным потомком.

Если удаляемый элемент – нетерминальная вершина с двумя потомками, то в этом случае удаляемый элемент заменяется либо на самый левый элемент его правого поддерева, либо на самый правый элемент его левого поддерева.

Средствами языка Паскаль такая структура может быть описана следующим образом:

```
type
    TREELINK = ^TREE;
    TREE=record
        INFO:integer;
        NLINK,RLINK:TREELINK
        end;
```

Упорядоченное дерево – это дерево, у которого ребра (ветви) исходящие из каждой вершины, упорядочены [1]. Поэтому два дерева на рисунке 5 есть разные деревья.

Рисунок 5 – Два различных двоичных дерева

Вершина Y, находящаяся непосредственно ниже вершины X, называется прямым потомком X, а вершина X – предком Y [1]. Таким образом, если X находится на уровне i, то ее прямые потомки – на уровне i+1. Вершина, имеющая только потомков, называется корнем дерева и лежит на уровне 0. Если элемент не имеет потомков, то его называют терминальной вершиной или листом. Все остальные вершины называются внутренними или нетерминальными.

На каждый узел, кроме корня, указывает только один узел верхнего уровня, поэтому имеется единственный путь от каждого узла к корню дерева (и от корня дерева к каждому узлу).

Из множества операций с бинарными деревьями рассмотрим операции линеаризации, добавления (включения) элемента и удаления (исключения) элемента из дерева. При этом будем полагать, что операции добавления и удаления выполняются для упорядоченных бинарных деревьев.

R, а значение поля данных нового элемента - через параметр NEWINFO.

```
procedure QADD(NEWINFO:integer; var R:NLINK);
 var
  NEWNODE:NLINK;
 begin
  new(NEWNODE);
  NEWNODE^.INFO:=NEWINFO;
  NEWNODE^.LINK:=nil;
  if R<>nil then R^.LINK:=NEWNODE;
  R:=NEWNODE;
 end;
```

Процедура QSEL выбирает элемент очереди, заданной указателями L начала и R конца, и возвращает его значение через переменную X.

```
procedure QSEL( var R,L:NLINK; var X:integer);
 var P:NLINK;
 begin
  if L<>R then  begin P:=L; X:=L^.INFO;
                      L:=L^.LINK;
                      dispose(P);
                     end;
   end;
```

ДЕК - это линейная структура, в которой включение и исключение элементов и доступ к элементам возможны на обоих концах структуры. Очевидно, что для реализации этих действий необходимо задавать, с какого конца требуется выполнить операцию.

3.2. Задание на лабораторную работу

Разработать программу формирования списка и выполнения заданной операции с ним. В программе предусмотреть вывод элементов исходного списка и, если необходимо, результирующего списка.

Коды вариантов задания приведены в таблице 3.1. Первая компонента кода варианта задания задает операцию, вторая – тип списка.

Таблица 3.1– Варианты задания

№ варианта	1	2	3	4	5	6
Код	1_1	2_1	3_1	4_1	5_1	6_1
№ варианта	7	8	9	10	11	12
Код	7_1	8_1	9_1	10_1	11_1	12_1
№ варианта	13	14	15	16	17	18
Код	1_2	2_2	3_2	4_2	5_2	6_2
№ варианта	19	20	21	22	23	24
Код	7_2	8_2	9_2	10_2	11_2	12_2

Заданная операция:

1. Упорядочить элементы списка по убыванию.
2. Добавить элемент после элемента с номером ***k***.
3. Вставить элемент перед элементом с номером ***k***.
4. Удалить элемент с номером ***k***.
5. Удалить элементы, информационное поле которых равно ***x***.
6. Удалить элемент, предшествующий элементу с номером ***k***.
7. Переставить ***k*** -ый и ***k +1***-ый элементы списка.
8. Удалить элемент, следующий за элементом с номером ***k*** .
9. Переставить первый и ***k***-ый элементы списка.
10. Найти сумму положительных элементов списка и записать ее последним элементом.
11. Добавить элемент после элемента, информационное поле которого равно ***x***.
12. Заменить элемент, информационное поле которого равно ***x***.

5. ЛАБОРАТОРНАЯ РАБОТА № 3. Формирование и обработка древовидных структур данных

Цель работы: изучение принципов организации и построения древовидных структур данных, а также программирование операций включения элемента, удаления элемента и обхода дерева.

5.1. Древовидные структуры. Основные теоретические положения

Одной из наиболее важных нелинейных структур, встречающихся в вычислительных алгоритмах, является дерево.

В [1] приведено следующее определение:

ДЕРЕВО - конечное множество T, состоящее из одной или более записей (узлов), таких, что:

а) имеется один специально обозначенный узел, называемый корнем дерева;

б) остальные узлы (исключая корень) содержатся в m>=0 попарно непересекающихся множествах T1,...,Tm, каждое из которых в свою очередь является деревом. Деревья T1,...,Tm называются поддеревьями данного корня.

Это рекурсивное определение отражает рекурсивность структур типа дерево, что, в свою очередь, определяет рекурсивный характер многих процедур обработки таких структур.

Из определения дерева ясно, что список есть дерево, в котором каждая вершина имеет не более одного поддерева. Поэтому список иногда называют вырожденным деревом.

Одним из наиболее важных типов дерева является бинарное дерево. Бинарное дерево - конечное множество узлов, которое или пусто, или состоит из корня и двух непересекающихся бинарных деревьев, называемых левым и правым поддеревьями данного корня.

Каждый узел бинарного дерева может быть представлен записью из трех полей: INFO, содержащим значение элемента данных, LLINK и RLINK, содержащими соответственно адреса левого и правого поддеревьев данного узла:

LLINK	INFO	RLINK

4.3. Контрольные вопросы

1. Дать определение табличных структур данных.
2. Перечислить основные типы с таблиц.
3. Какое поле называют ключевым?
4. Перечислить основные операции с таблицами.
5. Как выполняется операция выборки записи из таблицы?
6. Как выполняется операция включения записи в таблицу?
7. Дать определение ХЕШ –функции.
8. Привести примеры выбора ХЕШ –функции.
9. Указать способы разрешения конфликтов в ХЕШ–таблицах.

Тип списка:

1. – однонаправленный;
2. – двунаправленный.

3.3. Контрольные вопросы

1. Дать определение статических структур данных.
2. Дать определение динамических структур данных.
3. Какие структуры данных называются линейными?
4. Какую структуру имеют элементы однонаправленного списка?
5. Какую структуру имеют элементы двунаправленного списка?
6. Как выполняется операция удаления элемента из списка?
7. Как выполняется операция вставки элемента в список?
8. Дать определение стека.
9. Дать определение очереди.
10. Дать определение дека.

4. ЛАБОРАТОРНАЯ РАБОТА № 2. Формирование и обработка табличных структур данных

Цель работы: изучение принципов построения некоторых типов таблиц и основных операций, выполняемых с ними.

4.1. Таблицы. Основные понятия и определения

Одно из наиболее часто встречающихся в программировании действий – поиск данных, хранящихся с определенной идентификацией. Такие структуры носят названия таблиц.

ТАБЛИЦА – это набор записей, каждая из которых имеет отличительный признак, называемый ключом. Для каждой записи ключ должен иметь уникальное значение, таким образом, не может быть в таблице 2-х записей с одинаковым ключом. Помимо ключа запись таблицы имеет поле данных (или ссылки на них), несущих некоторую информацию [1,4].

Записи таблицы выбираются из нее по ключу и добавляются в нее по ключу. Задача определения по заданному ключу адреса хранения табличной записи называется задачей поиска. Эта задача решается по-разному, в зависимости от способа организации таблицы.

Основной характеристикой способа организации таблицы является среднее время поиска одной записи. Это время пропорционально средней длине поиска, под которой понимают среднее количество записей, просматриваемых для отыскания требуемой записи.

Различают таблицы неупорядоченные и упорядоченные. В упорядоченных таблицах записи упорядочены по значениям ключей. Процедура внесения записи в таблицу включает в себя процедуру поиска с последующим размещением записи в таблице, если поиск закончился неудачей, либо заключением о том, что запись с заданным ключом уже имеется в таблице, если поиск закончился успешно.

В неупорядоченных таблицах записи располагаются одна за другой, и новая запись просто добавляется в конец таблицы. Для поиска применяется последовательный просмотр, при котором записи просматриваются подряд, начиная с первой. Очевидно, неупорядоченные таблицы неэкономичны по времени поиска, но, с другой стороны, на включение новой записи в таких таблицах расходуется минимальное время.

4.2. Задание на лабораторную работу

Разработать программу, реализующую операции записи и выборки в таблице заданного вида. В программе предусмотреть вывод состояния таблицы. Варианты задания приведены в таблице 4.1.

Первый символ кода варианта задания задает тип таблицы, второй - тип данных, третий - тип ключа.

Таблица 4.1 – Варианты задания

№ варианта	1	2	3	4	5	6
Код	111	122	131	212	221	232
№ варианта	7	8	9	10	11	12
Код	311	322	331	412	421	432
№ варианта	13	14	15	16	17	18
Код	112	121	132	211	222	231
№ варианта	19	20	21	22	23	24
Код	312	321	332	411	422	431

Тип таблицы:

1 – ХЕШ–таблица с цепочками переполнения;
2 – ХЕШ–таблица с открытым перемешиванием;
3 – древовидная таблица (реализовать массивом);
4 – упорядоченная таблица.

Тип данных:

1 - символьный,
2 - целый,
3 - вещественный.

Тип ключа:

1 - символьный,
2 - целый.

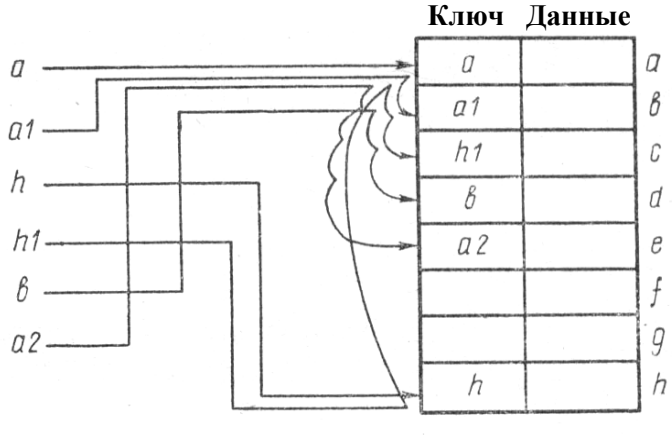


Рисунок 3 – Заполнение ХЕШ –таблицы с открытым перемешиванием заданной последовательностью ключей

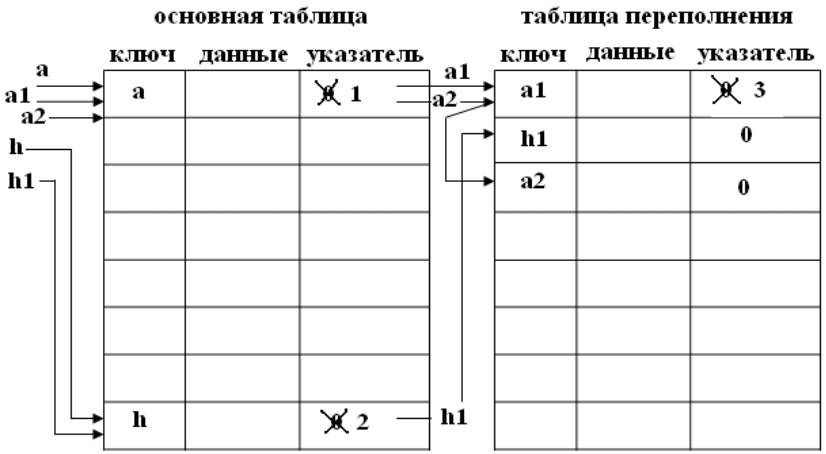


Рисунок 4 – Заполнение ХЕШ –таблицы с цепочками переполнений заданной последовательностью ключей

Очевидно, что ХЕШ–таблицы заполняются относительно просто, не требуют упорядочения записей и обеспечивают довольно быстрый поиск [4].

Рассмотрим принципы организации упорядоченных таблиц двух видов: древовидных таблиц и таблиц с вычисляемыми входами.

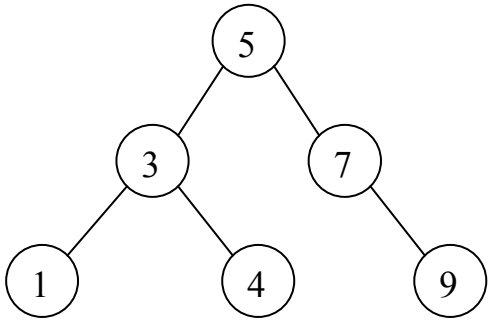
Древовидные таблицы организуются в виде двоичного дерева. Каждая запись такой таблицы сопровождается двумя указателями:

- левый – содержит адрес хранения записи с меньшим значением ключа;
- правый – содержит адрес хранения записи с большим значением ключа.

Пример древовидной таблицы и соответствующего ей дерева ключей приведены на рисунке 2. Для наглядности иллюстрации адреса записей полагаются равными их номерам.

№ записи	Запись таблицы			
	ключ	данные	левый указатель	правый указатель
1	5	· · ·	2	4
2	3	· · ·	5	3
3	4	· · ·	0	0
4	7	· · ·	0	6
5	1	· · ·	0	0
6	9	· · ·	0	0

а)



б)

Рисунок 2 – Пример древовидной таблицы:
а) последовательность записей таблицы;
б) дерево ключей.

В рассмотренном примере древовидная таблица заполнена для случая, когда записи имели следующие значения ключей (в порядке их поступления): 5,3,4,7,1 и 9.

Приведем алгоритм включения очередной записи с ключом X в древовидную таблицу, полагая, что таблица задана указателем ROOT на первую запись, расположенную в корне соответствующего дерева. Для ссылки на поля "ключ", "левый указатель" и "правый указатель" будем пользоваться обозначениями KEY, LLINK, и RLINK соответственно.

Алгоритм включения записи в древовидную таблицу.

п.1.Если ROOT=0, то поместить запись с ключом X в свободную строку таблицы.

п.2.P:=ROOT

п.3.Если X < P^.KEY, то перейти к п.4;

если X > P^.KEY, то перейти к п.5;

если X = P^.KEY, то поиск завершен, запись с ключом X уже имеется в таблице; конец алгоритма.

п.4.Если P^.LINK < > nil, то P:=P^.LLINK и перейти к п.3, иначе перейти к п.6.

п.5.Если P^.RLINK < > nil, то P:=P^.RLINK и перейти к п.3, иначе перейти к п.6.

п.6.Поместить запись с ключом X и пустыми полями LLINK и RLINK в свободную строку таблицы; если X было меньше P^.KEY, то P^.LLINK присвоить адрес этой строки; иначе P^.RLINK присвоить адрес этой строки; конец алгоритма.

Средняя длина поиска в древовидной таблице зависит от порядка поступления записей при заполнении таблицы. В худшем случае, если записи поступали в порядке возрастания или убывания значений ключа, дерево будет иметь всего одну ветвь и, следовательно, средняя длина поиска будет точно такой же, как и в неупорядоченной таблице. Во всех других случаях этот показатель будет меньше. Древовидная таблица может быть организована с помощью нелинейного списка с двумя указателями на левое и правое поддерево.

В основу построения таблицы с вычисляемыми входами положен метод, известный под названием "хеширование". В соответствии с этим методом предлагается над заданным значением X ключа произвести некоторое преобразование f(X), результат которого укажет адрес в таблице, где хранится X и ассоциированная

с ним информация (или куда следует поместить запись с ключом X) [1,4].

Пусть таблица отображается в вектор длины n. Тогда функция f(X) такая, что $1 \le f(X) \le n$ и для $i \ne j$ $f(i) \ne f(j)$, называется функцией расстановки или Хеш-функцией. Таблицы, для которых известна такая функция, называются таблицами с прямым доступом.

Обеспечить однозначность преобразования ключа в адрес практически невозможно. От этого требования обычно отказываются, что приводит к наложению записей. Ситуация, в которой значения ХЕШ-функции для различных значений ключа совпадают, называется коллизией (переполнением или конфликтом) [4]. Рассмотрим два метода устранения коллизий: метод открытого перемешивания и метод перемешивания с цепочками переполнения.

Приведем неформальное описание алгоритма, основанного на первом из указанных методов и предназначенного для выполнения выборки или включения в ХЕШ-таблицу, отображенную в вектор длины n, записи с ключом X.

Алгоритм включения и поиска записи в ХЕШ-таблице.

п.1 Вычислить i=f(X).

п.2 Если при включении в таблицу новой записи позиция i свободна, а при выборке записи эта позиция содержит заданный ключ X, то поиск завершен, иначе - перейти к п.3.

п.3 i:=i+1(mod n); перейти к п.2.

Рассмотрим пример. Пусть в таблице имен, имеющей 8 позиций, требуется разместить идентификаторы: a, a1, h, h1, a2.

Условимся, что идентификатор может начинаться с одной из восьми букв латинского алфавита: a, b, c, d, e, f, g, h, а значение функции расстановки равно порядковому номеру в алфавите первой буквы идентификатора. Схема заполнения таблицы заданными ключами изображена на рисунке 3.

Метод перемешивания с цепочками переполнения предлагает для записей переполнения завести дополнительную таблицу. При этом записи, которым соответствует одно и то же значение функции расстановки, связываются в цепочку, как в списке. Схема заполнения таких таблиц для рассмотренного выше примера приведена на рисунке 4.