МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное
учреждение высшего образования
«СЕВАСТОПОЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ
И УПРАВЛЕНИЯ В ТЕХНИЧЕСКИХ СИСТЕМАХ

Кафедра «Информационные технологии и компьютерные системы»

ОБРАБОТКА СТРУКТУР ДАННЫХ

Методические указания
к выполнению лабораторных работ и РГР
по дисциплине
«СИСТЕМНОЕ ПРОГРАММИРОВАНИЕ»
для студентов направления подготовки
09.03.01– «Информатика и вычислительная техника»

Севастополь
СевГУ
2019

УДК 004.42
О-23

Р е ц е н з е н т:
В.Ю. Карлусов – доцент кафедры «Информационные системы»,
канд. техн. наук, доцент

Составитель: Е.М. Шалимова

О-23 Обработка структур данных: метод. указания для выполнения лабораторных работ по дисциплине «Системное программирование» для бакалавров по направлению 09.03.01 «Информатика и вычислительная техника» / Сост. Е.М. Шалимова. – Электрон. дан. – Севастополь: СевГУ, 2019. – Режим доступа: свободный после авторизации. – Загл. с экрана. – 26 с.

Целью методических указаний является оказание помощи бакалаврам при выполнении лабораторных работ по дисциплине «Системное программирование».

Методические указания содержат:

- краткие теоретические сведения;
- примеры обработки рассматриваемых структур данных;
- задания на лабораторные работы;
- порядок выполнения и требования к отчетам;
- список рекомендованной литературы.

УДК 004.42

Методические указания рассмотрены и утверждены на заседании кафедры «Информационные технологии и компьютерные системы», протокол № 6 от 05 апреля 2019 г.

Текстовое (символьное) издание (1 Мб).

Системные требования: Intel, 3,4 GHz; 150 Мб; Windows XP/Vista/7; DVD-ROM; 1 Гб свободного места на жестком диске; программа для чтения pdf-файлов: Adobe Acrobat Reader, Foxit Reader.

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 работы рассчитаны на 4 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.

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 [1-4]. 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. Основные теоретические положения

Линейные структуры, в соответствии с [1,3], – это множество, состоящее из n≥0 записей X_1, X_2, ... , 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.

Эта структура однородна, так как все элементы имеют один и тот же тип и все они равнодоступны. Массивы относятся к структурам данных со случайным доступом: для доступа к некоторому элементу к имени массива добавляется индекс (значение специального типа), который можно вычислить, поэтому элементы массива иногда называют переменными с индексами.

К базовым операциям над массивами относятся поиск и выборочное изменение отдельных элементов. Оценим трудоемкость базовых операций в предположении, что в массиве n элементов.

1. Поиск элемента x:

а) в произвольном массиве – $O(n)$;

б) в отсортированном массиве– $O(\log n)$.

2. Поиск максимального (минимального) элемента:

а) в произвольном массиве – $O(n)$;

б) в отсортированном массиве – $O(1)$.

3.Выборочное изменение элемента массива – $O(1)$, если задан индекс элемента.

Основные операции над массивами не предполагают исключения элементов или включения новых.

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 или односвязным. Указатель последней записи равен признаку 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 или двусвязный список. Таким образом, список – конечная последовательность элементов, порядок которых определяется с помощью ссылок.

Тип списка принципиально не влияет на реализацию операций, поэтому далее ограничимся рассмотрением односвязных списков (в Приложении А приведены примеры функций обработки двунаправленного списка).

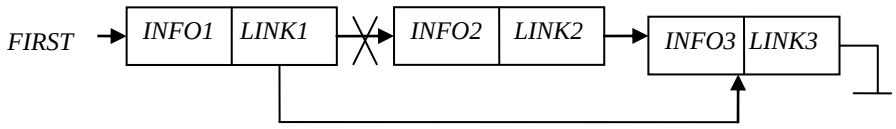
К базовым операциям для списка относятся (предполагаем, что в списке n элементов):

1) поиск элемента с ключом x (трудоемкость $O(n)$);
2) поиск максимального (минимального) элемента (трудоемкость $O(n)$);
3) включение элемента (трудоемкость $O(1)$);

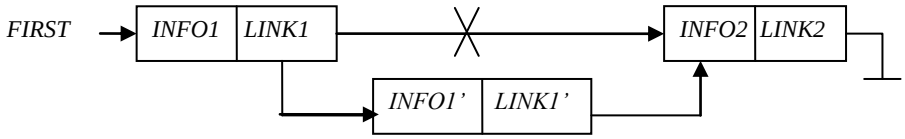
4) исключение элемента (трудоемкость *O(1)*);
5) формирование списка (трудоемкость *O(n)*);
6) просмотр списка (трудоемкость *O(n)*).

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 существенно упрощаются такие операции над линейной структурой, как удаление ее элемента и включение в нее нового элемента, которые сводятся, как показано на рисунке 1, 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;
     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;
    dispose(DELNODE)
```

```
    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 *STACKDEL* выбирает элемент из стека, заданного указателем *TOP* на его вершину, и возвращает его значение через переменную *X*.

```
procedure STACDEL(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 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 *QDEL* выбирает элемент очереди, заданной указателями *L* начала и *R* конца, и возвращает его значение через переменную *X*.

```
procedure QDEL( 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 (доступ к элементам) возможны на обоих концах структуры. Очевидно, что для реализации этих действий необходимо задавать, с какого конца требуется выполнить операцию.
Трудоемкость процедур включения и исключения элемента из стека, очереди или дека есть *O(1)*.

3.2. Задание на лабораторную работу

Разработать программу, реализующую выполнение четырех операций над заданной структурой: инициализация, запись, выборка, вывод состояния.

Коды вариантов задания приведены в таблице 3.1. Первый символ кода определяет тип структуры данных и способ ее реализации, второй – тип данных.

Таблица 3.1.–Варианты задания

№ варианта	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
код	**11**	**21**	**31**	**41**	**51**	**61**	**71**	**81**
№ варианта	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
код	**12**	**22**	**32**	**42**	**52**	**62**	**72**	**82**

Структуры данных:
1 - очередь(список),
2 - очередь(массив),
3 – список(массив),
4 - список общего вида,
5 - дек(список),
6 - дек(массив),
7 - стек(список),
8.- стек(массив).

Типы данных:
1 - символьный,
2 - целый.

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. Основные теоретические положения

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 [1-3]. Записи таблицы выбираются из нее по ключу и добавляются в нее по ключу. Задача определения по заданному ключу адреса *хранения табличной записи называется задач 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.

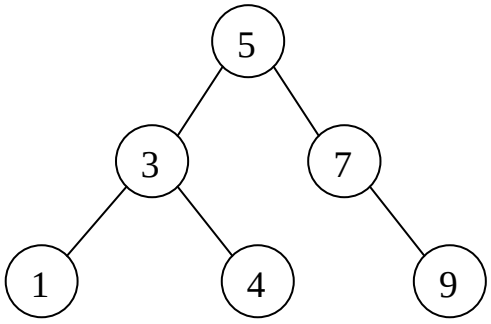
В неупорядоченных таблицах записи располагаются одна за другой, и новая запись просто добавляется в конец таблицы. Для поиска применяется последовательный просмотр, при котором записи просматриваются подряд, начиная с первой. Очевидно, неупорядоченные таблицы неэкономичны по времени поиска, но, с другой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

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

№ 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.

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.

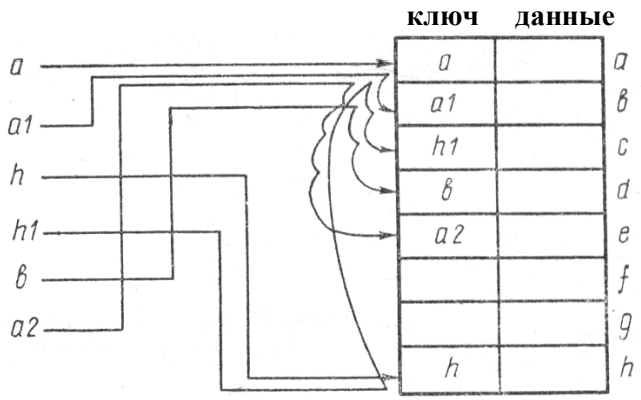
п.1. Вычислить *i=f(X).*

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.

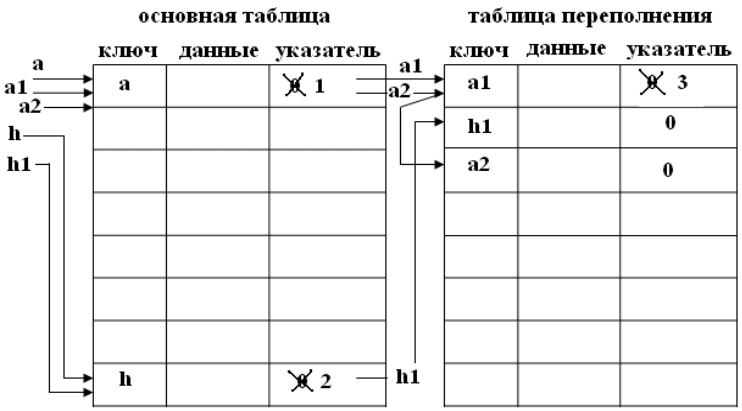
п.3. *i:=i+1(mod n);* перейти к п.2.

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,а.

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 3,б.



а)



б)

Рисунок 3 - Заполнение ХЕШ-таблицы заданной последовательностью ключей:
а) с открытым перемешиванием
б)с цепочками переполнений.

Очевидно, что ХЕШ-таблицы заполняются относительно просто, не требуют упорядочения записей и обеспечивают довольно быстрый поиск [3].

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.

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 ХЕШ –функций.
9. Указать способы разрешения конфликтов в ХЕШ–таблицах.

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. Основные теоретические положения

Одной из наиболее важных нелинейных структур, встречающихся в вычислительных алгоритмах, является дерево. В [3] 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*

Средствами языка Паскаль такая структура может быть определена 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 [2,3]. Поэтому два дерева на рисунке 4 есть разные деревья.

Рисунок 4 – 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.

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 5.

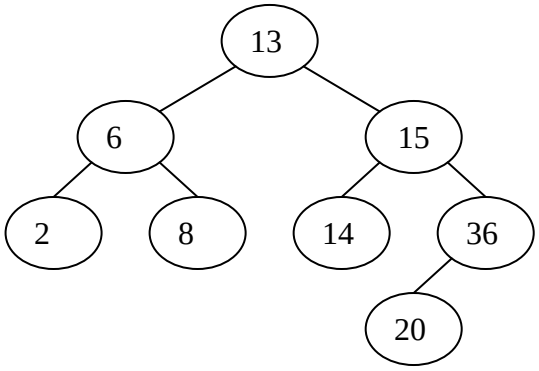


Рисунок 5 - Включение элемента в дерево.

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.

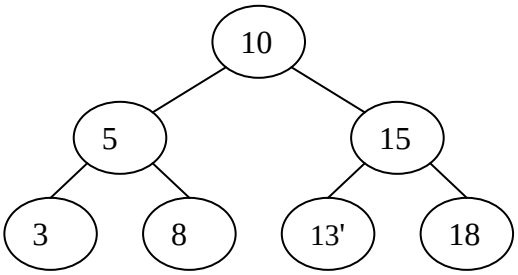
Если исключаемый элемент – это терминальная вершина, то соответствующему указателю предка этой вершины присваивается значение ПУСТОГО УКАЗАТЕЛЯ.

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

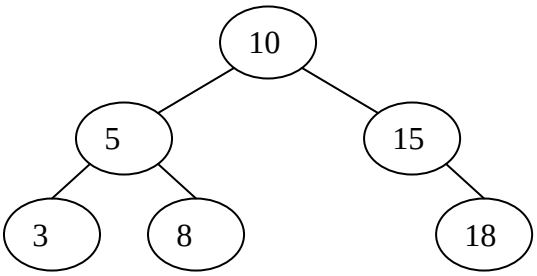
Если удаляемый элемент– нетерминальная вершина с двумя потомками, то в этом случае удаляемый элемент заменяется либо на самый левый элемент его правого поддерева, либо на самый правый элемент его левого поддерева.

Иллюстрация исключения элементов из дерева представлена на рисунке 6, где символом ' отмечен удаляемый элемент.

Исходное дерево

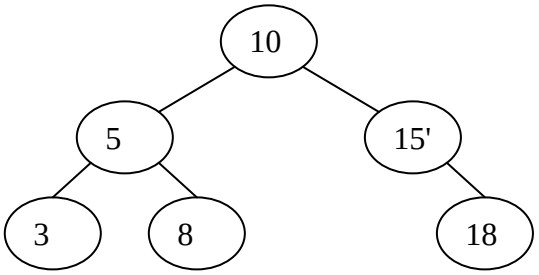


Дерево после удаления вершины

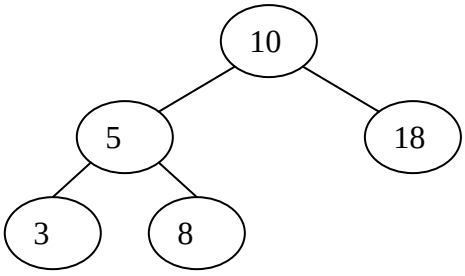


а)

Исходное дерево

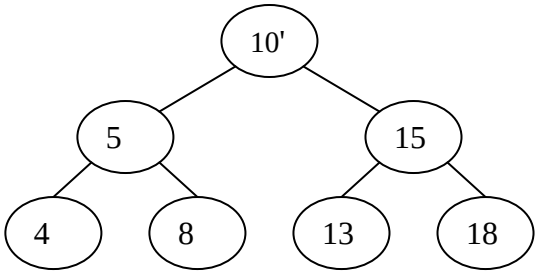


Дерево после удаления вершины

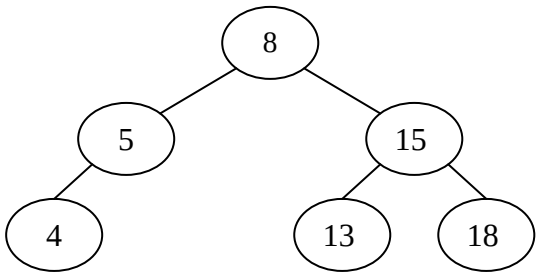


б)

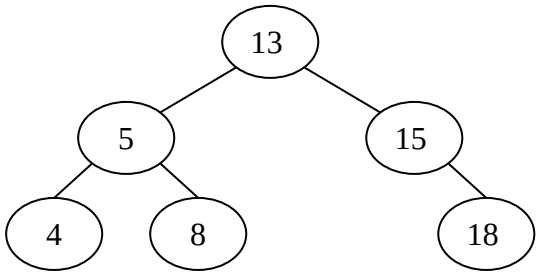
Исходное дерево



Дерево после удаления вершины



или



в)

Рисунок 6– Иллюстрация удаления элемента из дерева:
а) удаляемый элемент - терминальная вершина;
б) удаляемый элемент - вершина с одним потомком;
в) удаляемый элемент - вершина с двумя потомками.

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 7, каждым из трех перечисленных способов**.**

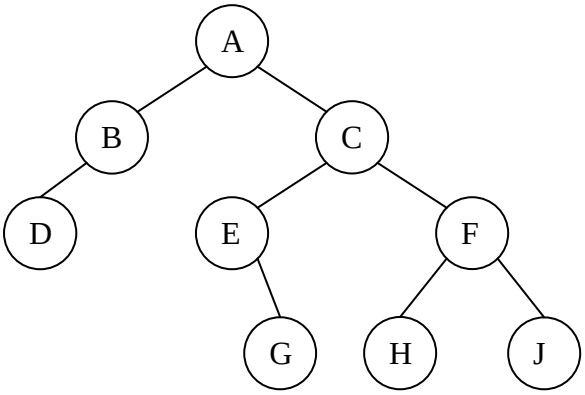


Рисунок 7 – Пример древовидной структуры

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.

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 задания 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**

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***.

Типы 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.

6. ВАРИАНТЫ ИНДИВИДУАЛЬНЫХ ЗАДАНИЙ К РГР

Задание к РГР содержит один теоретический вопрос и две задачи. Номер варианта определяется порядковым номером студента в списке группы (№ п/п в журнале преподавателя).

Варианты заданий представлены в таблицах 6.1–6.3. На теоретический вопрос следует дать развернутый ответ, привести примеры. ВНИМАНИЕ! Примеры не должны совпадать с примерами, рассмотренными на лекциях или приведенными в методических указаниях. Необходимые для выполнения заданий теоретические сведения изложены в методических указаниях к лабораторным работам по дисциплине, а также в источниках, указанных в библиографическом списке.

Решение задач должно сопровождаться подробными пояснениями и иллюстрациями. Отчет о выполнении РГР должен быть оформлен в соответствии с правилами, предъявляемыми к отчетам.

Задание1. Дать письменный ответ на теоретический вопрос.

Таблица 6.1.–Теоретические вопросы

1	Понятие вычислительной системы. Классификация программного обеспечения.
2	Структуры данных. Основные понятия. Классификация.
3	Линейные структуры данных. Массив. Операции.
4	Линейные структуры данных. Список. Операции.
5	Линейные структуры данных. Стек. Операции.
6	Линейные структуры данных. Очередь. Операции.
7	Линейные структуры данных. Дек. Операции.
8	Структуры данных. Неупорядоченная таблица. Операции.
9	Структуры данных. Упорядоченная таблица. Операции.
10	Структуры данных. Хеш- таблица с открытым перемешиванием. Операции.
11	Структуры данных. Хеш- таблица с цепочками переполнений. Операции.
12	Структуры данных. Древовидная таблица. Операции.
13	Структуры данных. Деревья. Операции.
14	Структуры данных. Двоичное дерево поиска. Операции.
15	Структуры данных. Сбалансированное дерево. Операции.
16	Структуры данных. Идеально сбалансированное дерево. Алгоритм построеия.
17	Задача поиска. Поиск в массиве.
18	Задача поиска. Поиск в таблице.
19	Задача поиска. Поиск подстроки в строке.
20	Задача идентификации. Автоматный подход.

Задание 2.

Для заданного арифметического выражения построить:

- ПОЛИЗ, используя стек с приоритетами;
- дерево операций и выполнить концевой обход его вершин.

Таблица 6.2.–Арифметические выражения

№ п/п	Выражение
1	a+b*(c-x*(y+k/(c-d/z))-f/t)
2	j-b*(c-x*(y+k*(c-d*z) -f*t))
3	a*b+(c-x*(y+k/(c-d)/z)*f-t)
4	a+b*((c-x)*(y-k/(c-d*z)-f)/t)
5	a+(b*(c-x)*(y+k/(c-d/z))-f*t)
6	a*(w+b)-(c-(x*y+k/(c-d/z))-f/t)
7	v+b-(c-x*(y+k/(c-a+d)/z))-f/t
8	h/(b-s)*(c-(x*y+k/(c-d*z))-f-t)
9	a+b*(c-x*(y+k/(c-d/z))-f/t)
10	j-b*(c-x*(y+k*(c-d*z) -f*t))
11	a+b*(c-x*(y+k-(c*d/z))-(f+s)/t)
12	j-b*(c-x-(y+ t *(c-d*z) -f* k))
13	(c-x*(y+k/(c-d)/z)*f-t)- a*b
14	a *(c-x)-(y-k/(c+d*z)-f)/t /b
15	a+(b*(c-x)*(y+k/(c-d/z))-f*t)
16	a*(w+b)-(c-(x*y+k/(c-d/z))-f/t)
17	v+b-(c-x*(y+k/(c-a+d)/z))-f/t
18	h/(b-s)+(c-(x*y+k/((c-d)*z))-f)
19	a+b*(c-x)+(y+k/(c-d/z)-f/t)
20	(j*b+c)/x*(y+k*(c-d)+z) -f*t))

Задание 3.

Для заданной последовательности ключей построить:

- двоичное дерево поиска; для полученного дерева выполнить удаление и добавление указанных вершин;
- идеально сбалансированное дерево.

Таблица 6.3.–Последовательности ключей

№ п/п	Последовательность ключей	Удаляемая вершина	Добавляемая вершина
1	30 45 15 60 80 35 25 10 12 3 70 50	35	64
2	32 45 15 65 80 35 25 10 12 5 71 54	45	39
3	30 45 15 80 35 25 60 10 12 33 70 50	45	74
4	30 45 80 35 15 25 10 12 3 60 70 50	15	28
5	50 45 15 80 13 5 25 60 10 12 3 70	50	19
6	45 15 80 35 25 30 60 12 3 70 10 50	60	11
7	45 29 80 35 25 30 60 12 36 70 110 50	25	27
8	30 45 15 80 35 12 25 60 3 70 10 50	30	53
9	30 45 15 60 80 35 25 10 2 13 50 70	15	56
10	32 45 15 65 80 35 25 10 12 5 71 54	45	26
11	25 13 45 15 60 80 35 10 30 12 70 50	13	55
12	32 45 15 54 80 35 25 10 12 65 5 71	15	24
13	30 45 15 28 35 25 9 50 10 12 70 60	15	39
14	30 45 80 5 15 25 40 12 50 3 60 70	45	48
15	50 45 15 70 13 5 52 60 10 12 3 80	70	35
16	45 15 55 35 25 30 60 12 13 70 10 50	55	26
17	45 25 80 35 25 30 60 12 36 70 110 50	30	22
18	30 45 15 80 35 12 25 50 10 70 3 60	15	44
19	25 45 15 60 80 35 21 10 12 3 73 20	25	57
20	35 45 15 65 80 32 25 10 12 5 71 54	15	19

Библиографический список

1. Алгоритмы и структуры данных : учебник / В.В. Белов, В.И. Чистякова. — М. :КУРС : НИЦ ИНФРА-М, 2017. — 240 с. — (Бакалавриат). —Режим доступа: http://znanium.com/catalog/product/766771. — Загл. с экрана.
2. Методы и алгоритмы обработки данных : учеб. пособие / А. А. Григорьев. — М. : ИНФРА-М, 2018. — 256 с. Режим доступа: http://www.znanium.com: www.dx.doi.org/10.12737/22119. — Загл. с экрана.
3. Вирт Н. Алгоритмы и структуры данных. Новая версия для Оберона. [Электронный ресурс] : учеб. пособие — Электрон. дан. — М. : ДМК Пресс, 2010. — 272 с. — Режим доступа: http://e.lanbook.com/book/1261. . — Загл. с экрана.
4. Павловская Т.А. С/С++. Программирование на языке высокого уровня/ Т.А. Павловская -СПб.:Питер, 2002.–464с.

ПРИЛОЖЕНИЕ А.
Функции обработки двунаправленного списка на языке С++

```
// Принятые обозначения:
// pbeg– указатель на начало списка,
// pend– указатель на конец списка.

// Определение списка
struct Node
{ int d;
Node *next;
Node *prev; };

Node * first(int d);

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

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

ОБРАБОТКА СТРУКТУР ДАННЫХ

Методические указания
к выполнению лабораторных работ и РГР

Составитель: **Шалимова** Елена Михайловна

Издается в авторской редакции

Изд. № 287/19. Объем 1,75 п.л.
РИИЦМ ФГАОУ ВО «Севастопольский государственный университет»