Министерство образования Республики Беларусь

Учреждение образования

«Белорусский государственный университет информатики и радиоэлектроники»

Филиал

«Минский радиотехнический колледж»

Учебная дисциплина «Структуры и алгоритмы обработки данных»

**Инструкция**

по выполнению лабораторной работы

«Реализация основных операций над графами и методов их обхода»

Минск

2018

**Лабораторная работа № 8**

**Тема работы: «Реализация основных операций над графами и методов их обхода»**

**1. Цель работы**

Формирование умений применения основных операций над графами и методов их обхода.

**2. Задание**

Вариант 1: Выполните обход в глубину для графа, состоящего из 10 вершин.

Вариант 2. Выполните программную реализацию обхода графа, состоящего из 10 вершин, в ширину.

**3. Оснащение работы**

ПК, Pascal.

**4. Основные теоретические сведения**

Под **обходом графов (поиском на графах)** понимается процесс систематического просмотра всех ребер или вершин графа с целью отыскания ребер или вершин, удовлетворяющих некоторому условию.

Представление графа в виде **списков смежных вершин** использует массив из **V** списков – по одному списку на вершину. Для каждой вершины список содержит в произвольном порядке все смежные с ней вершины.

*Вот как эта структура данных будет выглядеть в Паскале*

**const**

**VMax  = 100;**

**type**

**Spisok = ^List;**

**List = record**

**weight:integer;**

**next:spisok;**

**end;**

**var**

**a:array[1..VMax] of spisok;**

При использовании матрицы смежности мы нумеруем вершины графа числами **1,2,...,V** и рассматриваем матрицу **A**, в которой **a[i,j]=1** если есть ребро между вершинами i и j и **a[i,j]=0** в противном случае. В случае взвешенного графа, вместо единицы в **a[i,j]** удобно хранить вес ребра.

*В Паскале матрица будет выглядеть так:*

**const**

**VMax=100;**

**var**

**b:array[1..VMax,1..VMax] of integer;**

*Заполнение матрицы смежности из файла на Паскале*

**assign(fin,paramstr(1)); reset(fin);**

**readln(fin,n);**

**while not EOF(fin) do**

**begin**

**read(fin,i); readln(fin,j);**

**g[i,j]:=1;**

**end;**

**close(fin);**

*сначала из первой строки файла считываем количество вершин графа. Затем до конца файла в строках по два числа* ***i*** *и* ***j*** *что означает что есть ребро из i и j. Эта информация заносится в матрицу.*

К стандартным и наиболее распространенным методам обхода графов относятся:

- поиск в глубину (Depth First Search, DFS);

- поиск в ширину (Breadth First Search, BFS).

**Поиск в глубину**

Основная **идея поиска в глубину** – когда возможные пути по ребрам, выходящим из вершин, разветвляются, нужно сначала полностью исследовать одну ветку и только потом переходить к другим веткам (если они останутся нерассмотренными).

Алгоритм поиска в глубину:

**Шаг 1.** Всем вершинам графа присваивается значение не посещенная. Выбирается первая вершина и помечается как посещенная.

**Шаг 2.** Для последней помеченной как посещенная вершины выбирается смежная вершина, являющаяся первой помеченной как не посещенная, и ей присваивается значение посещенная. Если таких вершин нет, то берется предыдущая помеченная вершина.

**Шаг 3.** Повторить шаг 2 до тех пор, пока все вершины не будут помечены как посещенные.

Рассмотрим процедуры **рекурсивной** и **нерекурсивной** реализации DFS.

Граф представлен матрицей смежности C[i, j], каждый элемент которой индексирован номерами вершин и содержит значение C[i, j] = 1 для смежных вершин i и j , C[i ,j] = 0 для несмежных вершин.

Нерекурсивная реализация процедуры поиска использует структуру данных Стек. Рекурсивная реализации использует встроенные возможности системы программирования - использование стековой памяти для хранения рекурсивных вызовов и возвратов.

Для представления Стека используем одномерный массив Stek[i], обеспечивающий хранение элементов в порядке их поступления и извлечение этих же элементов в обратном порядке. Вершину Стека запомним в переменной **Uk** – указатель вершины Стека. Стек поддерживает операции добавления элемента в стек **(setStek)**, извлечения элемента из вершины стека **(getStek)**, инициализации стека **(initStek)**. Запишем программную реализацию этих процедур.

**Procedure setStek(v: integer);**

**Begin**

**Inc(Uk); Stek[Uk]:=v;**

**End;**

Procedure getStek( var v: integer);

**Begin**

**v:= Stek[Uk]; Dec(Uk);**

**End;**

Procedure initStek;

**Begin**

**Uk:=0;**

**End;**

Посещенные и добавленные в стек вершины отметим специальной меткой, в случае повторного нахождения данной отмеченной вершины как смежной с некоторой текущей вершиной поиска, отмеченную вершину не будем включать в стек повторно. Таким образом, размер стека ограничивается количеством вершин графа. Отмеченные вершины будем хранить в массиве **Visit [i]**, индекс массива соответствует номеру вершины, а элемент – метка (например, **true** – непосещенные вершины, **false** – отмечены как посещенные)

Program **Poisk\_v\_glubinu**;

**const n=9;                           {количество вершин графа}**

**C: array [0..n,0..n] of 0..1=((0,1,0,0,0,0,0,0,0,1),**

**(1,0,1,1,1,0,0,0,0,0),**

**(0,1,0,0,0,1,0,0,0,0),**

**(0,1,0,0,1,0,0,0,0,0),**

**(0,1,0,1,0,0,0,0,0,0),**

**(0,0,1,0,0,0,0,0,1,0),**

**(0,0,0,0,0,0,0,0,1,0),**

**(0,0,0,0,0,0,0,0,1,0),**

**(0,0,0,0,0,1,1,1,0,1),**

**(1,0,0,0,0,0,0,0,1,0)); {представление графа матрицей смежности}**

**Visit: array [0..n] of boolean=(true,true,true,true,true,true,true,true,true,true);**

**{вершины не посещались}**

**var v: integer;                     {вершина начала поиска}**

**{1 способ - Рекурсивная реализация}**

procedureDFS( v:integer);

**var t: integer;**

**begin**

**write(v: 3); Visit[v]:=false;**

**for t:=0 to n do**

**if (C[v,t]=1) and (Visit[t]) then**

**DFS(t);**

**end;**

**{2 способ - Нерекурсивная реализация}**

procedureDFS( v:integer);

**var Stek: array [1..(n+1)] of 0..n;   {стек представлен в виде массива}**

**Uk: integer;                             {с указателем на вершину стека}**

**t: integer; f: boolean;**

**begin**

**write(v:3);**

**{инициализация, стек пуст}**

**Uk:=0;**

**{добавить вершину в стек}**

**Uk:=Uk+1; Stek[Uk]:=v;**

**{отметить вершину как посещенную}**

**Visit[v]:=false;**

**while Uk<>0 do          {пока стек не пуст}**

**begin**

**{извлекаем элемент из стека}**

**v:=Stek[Uk];**

**{нахождение вершины t, связанной с v дугой v->t}**

**t:=0; f:=false;**

**repeat**

**{если вершина t ранее не посещалась и связана дугой с v}**

**if (C[v,t]=1) and (Visit[t]) then**

**f:= true**

**else t:=t + 1;**

**until f or (t>n);{найдена смежная и непосещенная вершина или все вершины просмотрены}**

**if f then**

**begin**

**{выводим элемент в порядке обхода в глубину}**

**write(t:3);**

**Uk:=Uk+1;**

**Stek[Uk]:=t;**

**{отмечаем как посещенную вершину}**

**Visit[t]:=false;**

**end**

**else Uk:=Uk-1;            {извлекаем вершину из стека}**

**{вершины просмотрены и смежных нет}**

**end;**

**end; {procedure}**

**begin**

**writeln(‘Поиск в глубину’);**

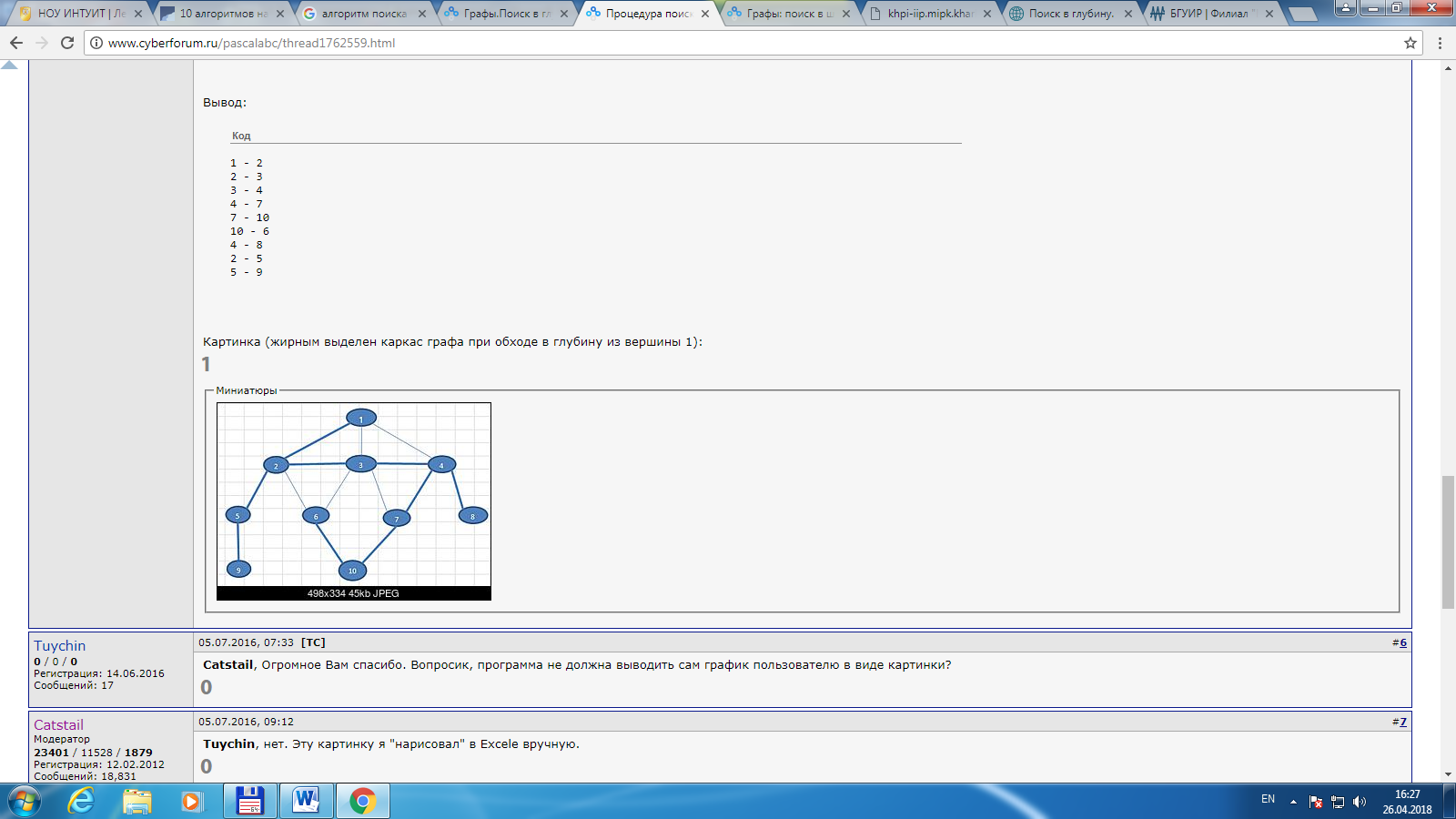
**writeln(‘Введите вершину начала поиска’); readln(v);**

**DFS(v);  { поиск в глубину от исходной вершины v}**

**end.**

*Пример:*

Процедура поиска в глубину в графе, заданном списками инцидентности. Выведите на экран номера вершин в том порядке, в котором они просматриваются процедурой.



**Program 123;**

**Const MAX\_VERT = 100;**

**Var LstS    : Array[1..MAX\_VERT,1..MAX\_VERT] of integer;**

**Var Chk     : Array[1..MAX\_VERT] of integer;**

**Var CurrN   : integer;**

**Var finp    : Text;**

**Var i,j,l,k : integer;**

**Var Stri,s,tmp : String;**

**Function isBound(i,j : integer) : Boolean;**

**Var k : integer;**

**Begin**

**for k:=1 to CurrN Do**

**if LstS[i,k]=j Then Begin**

**isBound:=True;**

**exit;**

**End;**

**For k:=1 to CurrN Do**

**if LstS[j,k]=i Then Begin**

**isBound:=True;**

**exit;**

**End;**

**isBound:=False;**

**End;**

**Procedure DFS(n : integer);**

**Var i : integer;**

**Begin**

**Chk[n]:=1;**

**for i:=1 to  CurrN Do**

**if (Chk[i]=0) And (isBound(i,n)) Then Begin**

**writeln(n,' - ',i);**

**DFS(i);**

**end;**

**End;**

**Begin**

**for i:=1 to MAX\_VERT Do Begin**

**Chk[i]:=0;**

**for j:=1 to MAX\_VERT Do**

**Lsts[i,j]:=0;**

**end;**

**assign(finp,'graph.txt');**

**reset(finp);**

**readln(finp,CurrN);**

**for i:=1 to CurrN Do Begin**

**readln(finp,Stri);**

**Stri:=Stri+',';**

**l:=length(Stri);**

**tmp:='';**

**k:=1;**

**for j:=1 to l Do Begin**

**s:=Stri[j];**

**if s <> ' ' then**

**if s = ',' then Begin**

**if tmp <> '' then Begin**

**Lsts[i,k]:=strToint(tmp);**

**tmp:='';**

**k:=k+1;**

**End;**

**End**

**Else**

**tmp:=tmp+s;**

**End;**

**End;**

**Close(finp);**

**DFS(1);**

**writeln;**

**End.**

|  |  |
| --- | --- |
| *Вывод:* | *Файл с данными:* |
| **1 – 2**  **2 - 3**  **3 - 4**  **4 - 7**  **7 - 10**  **10 - 6**  **4 - 8**  **2 - 5**  **5 - 9** | **10 //кол-во вершин**  **2,3,4 //матрица инцидентности**  **1,3,5,6**  **1,2,4,6,7**  **1,3,7,8**  **2,9**  **2,3,10**  **3,4,10**  **4**  **5**  **6,7** |

**Поиск в ширину**

Для реализации этого алгоритма необходимо запоминать посещенные смежные с **V** узлы и, рассматривая их в порядке посещения, начинать процесс посещения всех узлов, смежных с ними. Для этой цели хорошо подходит такая структура данных как очередь (действительно, посещенные узлы становятся в очередь для продолжения процесса поиска от них).

Алгоритм поиска в ширину:

**Шаг 1.** Всем вершинам графа присваивается значение не посещенная. Выбирается первая вершина и помечается как посещенная (и заносится в очередь).

**Шаг 2.** Посещается первая вершина из очереди (если она не помечена как посещенная). Все ее соседние вершины заносятся в очередь. После этого она удаляется из очереди.

**Шаг 3.** Повторяется шаг 2 до тех пор, пока очередь не пуста.

Выполним программную реализацию алгоритма поиска в ширину с учетом выбора структуры данных.

Граф представлен матрицей смежности **C[i, j]**, каждый элемент которой индексирован номерами вершин и содержит значение **C[i, j] = 1** для смежных вершин i и j, **C[i ,j] = 0** для несмежных вершин.

Для представления Очереди в Паскале используем одномерный массив **Q[i]**, обеспечивающий хранение элементов в порядке их поступления и извлечение в том же порядке. Начало и конец очереди запомним в переменных **Un** – указатель начала очереди, **Uk** – указатель конца очереди. Очередь поддерживает операции установки элемента в очередь **(setQueue)**, извлечения элемента из очереди **(getQueue)**, инициализации очереди **(initQueue)**. Запишем программную реализацию этих процедур.

Procedure**setQueue (v: integer)**;

**Begin**

**Inc(Uk); Q[Uk]:=v;**

**End;**

Procedure **getQueue (var v: integer)**;

**Begin**

**v:=Q[Un]; Inc(Un);**

**End;**

Procedure initQueue;

**Begin**

**Uk:=0; Un:=0;**

**End;**

Посещенные и добавленные в очередь вершины отметим специальной меткой, в случае повторного нахождения данной отмеченной вершины как смежной с некоторой текущей вершиной поиска, отмеченную вершину не будем включать в очередь повторно. Таким образом, размер очереди ограничивается количеством вершин графа. Отмеченные вершины будем хранить в массиве Visit [i], индекс массива соответствует номеру вершины, а элемент – метка (например, true – непосещенные вершины, false – отмечены как посещенные).

program**Poisk\_v\_shirinu**;

**const n=9;                          {количество вершин графа}**

**{представление графа матрицей смежности }**

**C: array [0..n,0..n] of 0..1=((0,1,0,0,0,0,0,0,0,1),**

**(1,0,1,1,1,0,0,0,0,0),**

**(0,1,0,0,0,1,0,0,0,0),**

**(0,1,0,0,1,0,0,0,0,0),**

**(0,1,0,1,0,0,0,0,0,0),**

**(0,0,1,0,0,0,0,0,1,0),**

**(0,0,0,0,0,0,0,0,1,0),**

**(0,0,0,0,0,0,0,0,1,0),**

**(0,0,0,0,0,1,1,1,0,1),**

**(1,0,0,0,0,0,0,0,1,0));**

**Visit: array [0..n] of boolean =(true,true,true,true,true,true,true,true,true,true);**

**{вершины не посещались}**

**var v: integer;                                 {вершина начала поиска}**

procedure BFS( v:integer);

**var Q: array [1..(n+1)] of 0..n;**

**{очередь представлена в виде массива}**

**Un,Uk: integer;**

**{с двумя указателями на начало и конец очереди}**

**j: integer;**

**begin**

**Un:=0; Uk:=0;             {инициализация, очередь пуста}**

**Uk:=Uk+1; Q[Uk]:=v; {занести вершину в очередь}**

**{отметить вершину как посещенную}**

**Visit[v]:=false;**

**while Un < Uk do                   {пока очередь не пуста}**

**begin**

**{извлекаем элемент из очереди}**

**Un:=Un+1; v:=Q[Un];**

**{выводим элемент в порядке обхода в ширину}**

**write(v:3);**

**{просмотр всех вершин, связанных с v дугой}**

**for j:=0 to n do**

**if (C[v,j]=1) and (Visit[j]) then**

**begin**

**{если вершина ранее не посещалась и связана дугой, то заносим в очередь}**

**Uk:=Uk+1; Q[Uk]:=j;**

**{отмечаем как посещенную вершину}**

**Visit[j]:=false;**

**end;**

**end; {while}**

**end; {procedure}**

**begin**

**writeln('Поиск в ширину');**

**writeln (‘Введите вершину начала поиска’); readln(v);**

**BFS(v); { поиск в ширину от исходной вершины v}**

**end.**

*Реализация на Паскале с использованием матрицы смежности*

**var**

**Q:array[1..100] of byte;   {очередь}**

**left,right:byte;**

**procedure Enqueue(x:word); {процедура добавления в очередь вершины}**

**begin**

**Q[right]:=x;**

**if right=100 then right:=1 else inc(right);**

**end;**

**procedure Dequeue;              {процедура удаления из очереди вершины}**

**begin**

**if left=100 then left:=1 else inc(left);**

**end;**

**var**

**G:array[1..100,1..100] of word;  {граф}**

**i,j,k,h:word;**

**d:array[1..100] of word;             {массив расстояний}**

**p:array[1..100] of word;             {массив предков}**

**color:array[1..100] of byte;         {массив цветов}**

**n:word;**

**fin:text;**

**beg:word;                                   {начальная вершина}**

**begin**

**k:=0;**

**{чтение графа из файла, имя которого - первый параметр командной строки}**

**assign(fin,paramstr(1)); reset(fin);**

**read(fin,n); readln(fin,beg);**

**while not EOF(fin) do**

**begin**

**read(fin,i); readln(fin,j);**

**g[i,j]:=1; g[j,i]:=1;**

**end;**

**close(fin);**

**{конец чтения из файла}**

**{инициализация массивов}**

**for i:=1 to n do**

**begin**

**color[i]:=0;**

**d[i]:=65535;**

**p[i]:=0;**

**end;**

**{конец инициализации массивов}**

**color[beg]:=1;**

**d[beg]:=0;**

**p[beg]:=0;**

**left:=1;**

**right:=1;**

**Enqueue(beg);             {добавление первой вершины в очередь}**

**while left<>right do      {пока очередь не пуста}**

**begin**

**k:=Q[left];**

**for i:=1 to n do        {для всех вершин...}**

**if g[k,i]=1 then       {...смежных с k...}**

**begin**

**if color[i] = 0 then {...если мы ее еще не обрабатывали...}**

**begin**

**color[i]:=1;       {...сделать цвет серым...}**

**d[i]:=d[k]+1;    {...указать расстояние...}**

**p[i]:=k;             {...указать предка...}**

**EnQueue(i);      {...добавить в очередь}**

**end;**

**end;**

**Dequeue;                {удалить из очереди}**

**color[k]:=2;             {сделать цвет черным: вершина полностью обработана}**

**end;**

**{вывод на экран расстояния от s до всех вершин, достигаемых из нее}**

**for i:=1 to n do**

**write(d[i],' ');**

**writeln;**

**end.**

**5. Порядок выполнения работы**

1. Матрица смежности задается произвольно.

2. Начало обхода совпадает с вершиной 8.

3. Запишите полученный список вершин.

**6. Форма отчета о работе**

*Лабораторная работа № \_\_\_*

*Номер учебной группы \_\_\_\_\_\_\_\_\_\_\_\_*

*Фамилия, инициалы учащегося \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Дата выполнения работы \_\_\_\_\_\_\_\_\_\_\_\_\_*

*Тема работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Цель работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Оснащение работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Результат выполнения работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

**7. Контрольные вопросы и задания**

1. Что представляет собой обход графа?

2. В чем заключается основная идея обхода графа в глубину?

3. В чем заключается основная идея обхода графа в ширину?

4. Перечислите способы представления графов в памяти ЭВМ?

**8. Рекомендуемая литература**

1.Ахо, А.В.Структуры данных и алгоритмы / А.В. Ахо, Дж.Хопкрофт, Дж. Д.Ульман. – пер. с англ. – М.: Издательский дом "Вильямс", 2007.-400 с.

2. Вирт,Н. Алгоритмы и структуры данных / Н. Вирт. –СПб.:Невский диалект, 2008. – 352с.

3. Вирт, Н. Алгоритмы и структуры данных с примерами на Паскале / Н. Вирт[перевод с английского Д. Б. Подшивалова]; – 2-е изд., испр., – СПб.: Невский Диалект, 2005. – 352с.

4. Гагарина, Л.Г. Алгоритмы и структуры данных / Л.Г. Гагарина, В.Д. Колдаев. – учеб.пособие – М: Финансы и статистика, 2009. – 304с.

5. Котов, В.М. Алгоритмы и структуры данных: учеб.пособие / В.М. Котов, Е.П. Соболевская, А.А. Толстиков – Минск: БГУ, 2011. – 267с.

6. Макконнелл, Дж. Основы современных алгоритмов / Дж. Макконнелл – 2-е дополненное издание – М.:Техносфера, 2006. – 368с.

7. Окулов, С.М. Программирование в алгоритмах / С.М. Окулов. – 3-е изд. – М.: БИНОМ. Лаборатория знаний, 2007. – 383с.