Структуры данных

Деревья (tree)

Деревья – это графы, которые характеризуются следующими свойствами:

1. Существует единственный элемент (узел или вершина), на который не ссылается никакой другой элемент (первый элемент дерева) – который называется КОРНЕМ (root);

2. На каждый элемент, кроме корня, имеется единственная ссылка, т.е. каждый элемент адресуется единственным указателем.

3. Начиная с корня и следуя по определенной цепочке указателей, содержащихся в элементах, можно осуществить доступ к любому элементу структуры.

Элемент дерева называется вершиной или узлом (node); любой фрагмент дерева – поддерево (subtree); вершина, к которой не присоединены поддеревья – заключительный узел (terminate node) или лист (leaf). Два связанных узла – звено (link) Высота дерева (height) – это максимальное количество уровней (узлов) от корня до самого дальнего листа.

Бинарное (двоичное) дерево (binary tree)

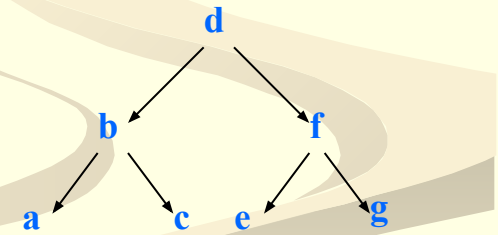
Каждый узел бинарного дерева содержит ссылки только на два последующих узла.

Основные операции над деревьями:

- **Поиск узла с заданным ключом.**
- **Добавление нового узла.**
- **Удаление узла (поддерева).**
- **Обход дерева в определенном порядке:**
  - **Нисходящий обход (d b a c f e g) – корень, левое поддерево, правое;**
  - **Восходящий обход (a c b e g f d) – левое поддерево, правое, корень;**
  - **Симметричный обход (a b c d e f g) – левое поддерево, корень, правое.**

Ориентированное дерево

В упорядоченном дереве задан порядок следования узлов.



Обход (прохождение) дерева (tree traversal) – систематический просмотр всех его узлов в определенном порядке.

Узел d называется – предок (отец), а b и f - наследники (сыновья), причем левый (b) – старший брат, а правый (f) – младший.

Число поддеревьев данного узла называется степенью этой вершины. (Узел d имеет 2 поддерева, следовательно степень вершины d равна 2).

Структуры данных

C / C++

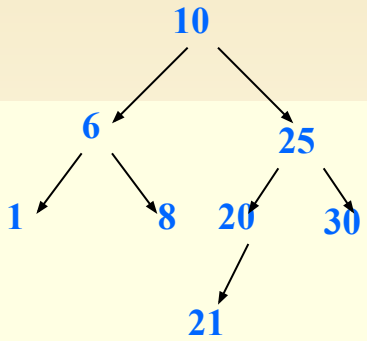
Деревья (tree)

Практическое занятие

C / C++

```
#include <iostream.h>
  typedef int bool; const true =1; const false =0;
   struct Node { int d; Node *left;   Node *right;
};

Node * firstTr (int d);
Node * search_insertTr (Node * root, int d);
void printTr (Node *root, int l);

int main()   {
int b[ ] = {10, 25, 20, 6, 21, 8, 1, 30};
    Node *root = firstTr (b[0]);
for (int i=1; i<8; i++)
    search_insertTr (root, b[i]);
printTr (root,0);
 return 0;  }

Node * firstTr (int d)  {
  Node *pv = new Node;
   pv -> d  = d;
   pv -> left  = 0;
   pv -> right = 0;
 return pv;                  }

void printTr (Node *p, int level)  {
 if (p)  {
printTr(p->left,level+1); for(int i=0; i<level; i++)
    cout << "    ";       cout << p -> d << endl;
printTr (p -> right, level +1);        } }
```

См. продолжение



продолжение:

```
Node * search_insertTr (Node *root, int d)
{ Node *pv = root,
   Node *prev;
   bool found = false;
   while (pv && !found)
    {
     prev = pv;
     if (d == pv->d)
      found = true;
     else
      if (d < pv->d)
       pv = pv->left;
      else
       pv = pv->right;
    }
   if (found)
     return pv;
   Node *pnew = new Node;
  pnew->d = d;
  pnew->left  = 0;
  pnew->right = 0;
  if (d < prev->d)
     prev->left = pnew;
  else
     prev->right = pnew;
 return  pnew;
}
```

Структуры данных

Деревья (tree)

Pascal

Pascal

Практическое занятие

```
type Pt=^node;
node=record  key: integer; Left, Right: Pt   end;
var Dtree: Pt;

Function SearchTr (var k: integer; var D,Rez: Pt): boolean;
var P: Pt;  B: boolean;
begin
    B := False;    P := d;
  if D<>nil then   repeat if P^.key=k then B:=True
                                      else if k<P^.key
                                           then P:=P^.Lev
                                           else P:=P^.Prav;
                   until B or (P=nil);
 SearchTr :=  B;    Rez := P;
end;

Procedure  AddTr (K: Integer; var D: Pt);
 var R, S: Pt;
  begin
  if not SearchTr (K, D, R)      then
   begin
    New (S);
    S^.key := K;
    S^.Left := nil;       S^.Right := nil;
    if D = nil     then     D := S
                   else
                    if K < R^.key
                    then  R^.Left := S
                    else  R^.Right := S;
   end
end;
```

```
Procedure  DelTr (var D: Pt; var K: integer);
var Q: Pt;
 Procedure DlTr (var R: Pt);
begin
  if R^.Right = nil   then
    begin
     Q^.key := R^.key;
     Q := R;
     R := R^.Left
    end
   else
    DlTr (R^.Right)
 end;

begin
 if D = nil
  then
WriteLn (' Узел с заданным ключом в дереве не найден')
  else  if K < D^.key   then  DelTr (D^.Right, K)
                        else
                          begin Q := D;
                           if Q^.Right = nil then
                             D := Q^.Left
                           else
                             if Q^.Left = nil
                              then
                                D := Q^.Right
                              else DlTr (Q^.Left)
                          end

end;
```

Структуры данных

Стеки, очереди, списки, деревья

Реализация полустатических и динамических структур с помощью массивов

Если максимальный размер данных можно определить заранее и он не изменяется в процессе работы программы, то можно использовать массивы. Связи элементов структур реализуются через вспомогательные переменные или номера элементов массивов, а не через указатели.

Стек: требуется массив (по типу данных стека) + одна переменная целого типа **i** для хранения индекса вершины стека. При помещении в стек **i++** , при выборке – **i--**.

Очередь: массив и две переменные (целые) – для хранения индексов элементов массива (начало и конец).

Линейный список: основной массив (по типу данных списка) и вспомогательный массив (целого типа), а также одна переменная.

Например:		10	25	20	6	21	8	1	30 - массив данных
1	2	3	4	5	6	7	-1		- вспомогательный массив (индекс следующего элемента)

0 – индекс первого элемента в списке, -1 – признак конца списка.

Бинарное дерево: основной массив (по типу данных списка) и 2-а вспомогательных массива (целого типа, для индексов узлов левого и правого поддерева.

Например:	10	25	20	6	21	8	1	30 - массив данных
3	2	-1	6	-1	-1	-1	-1	- левая ссылка
1	7	4	5	-1	-1	-1	-1	- правая ссылка

-1 – признак пустой ссылки

Память под такие структуры надо или задавать константой, или на этапе компиляции.

```
struct Node { Data d; int i; };
// тип данных Data надо определить заранее
Node spisok1 [1000];                 // на этапе компиляции
Node *pspisok2 = new Node[m];        // при выполнении
```

!!! Требуется контроль выхода индексов за границы массивов !!!