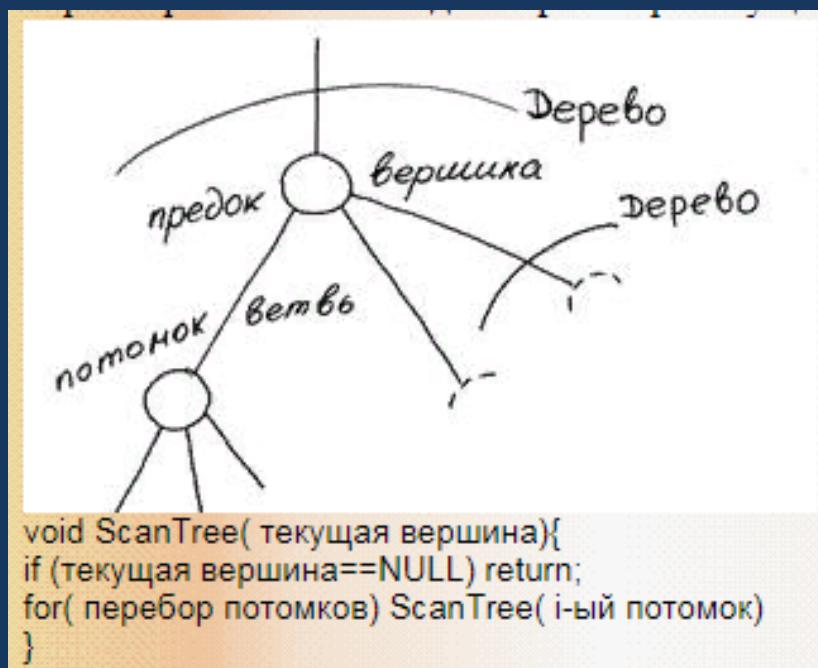




Деревья

Добрый доктор Айболит, он под деревом сидит
К.Чуковский. Айболит

- Рекурсивная структура данных: вершина + ссылки на поддеревья
- Терминология: корень, предок, потомок (отец, сын)
- Рекурсивный обход дерева: функция для вершины вызывает себя для потомков
- Жадный алгоритм (ветвление) – функция для вершины выбирает единственного потомка (цикл, линейная рекурсия)





Представление деревьев

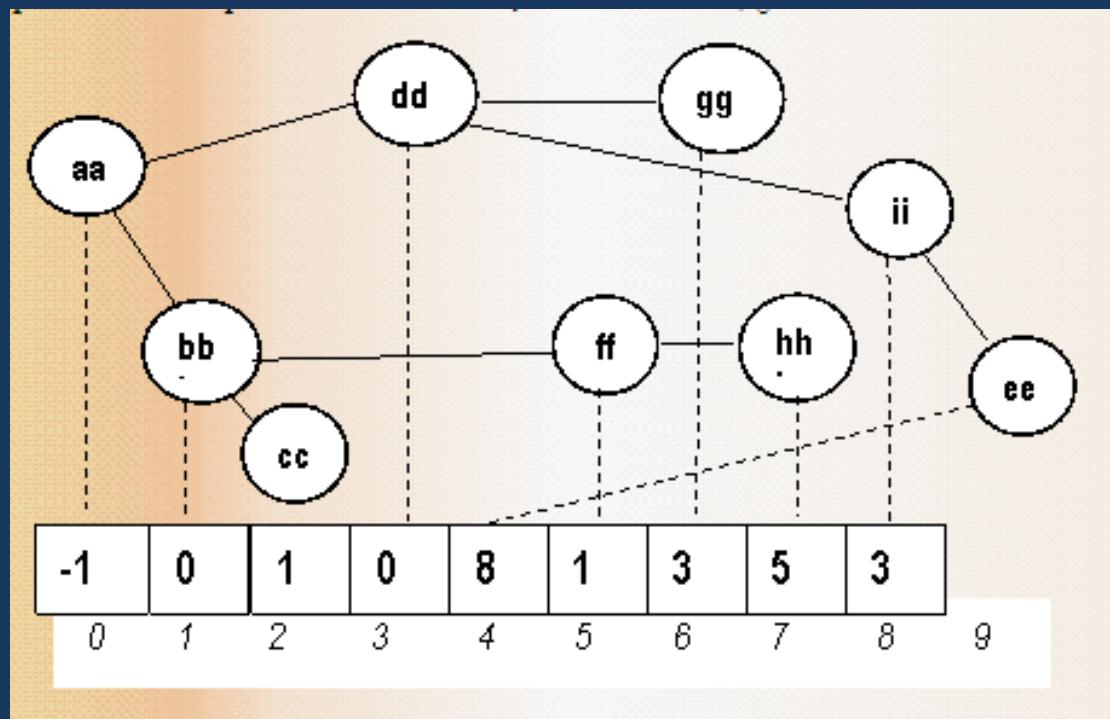
Дерево как логическая СД, формы представления – физическая СД

- если используется рекурсивный или циклический алгоритм, начинающий работать с корневой вершиной дерева, то необходимы только прямые ссылки от предка к потомкам;
- если алгоритм предполагает навигацию по дереву во всех направлениях, как вверх, так и вниз по дереву (например, в древовидной системе каталогов), то предполагается наличие как прямых, так и обратных ссылок от потомков к предкам (в системе каталогов – ссылка на родительский каталог)
- возможны алгоритмы, которые работают с деревом, начиная с терминальных вершин. Тогда кроме ссылок от потомков к предкам необходима еще структура данных, объединяющая терминальные вершины (например, массив указателей).



Представление деревьев

Дерево в массиве с индексами предков (представление в реляционной БД)



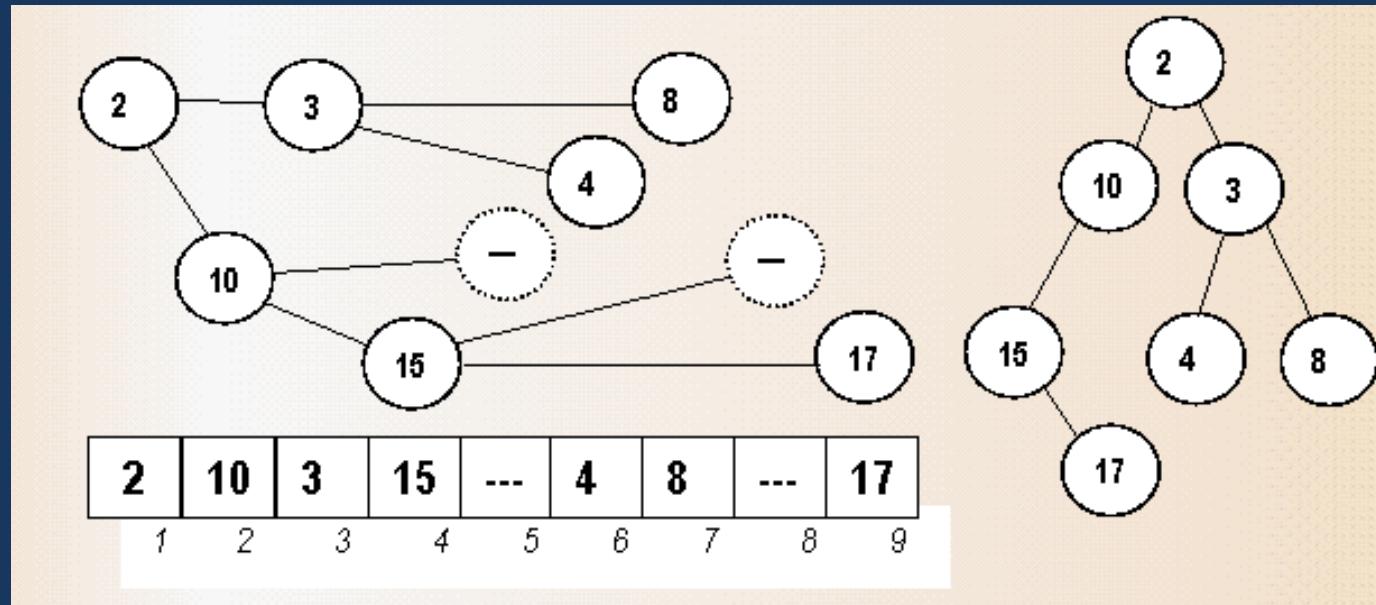
```
struct mtree{  
    char *s;  
    int parent;  
};  
mtree A1[] = {  
    {"aa",-1},  
    {"bb",0},  
    {"cc",1},  
    {"dd",0},  
    {"ee",8},  
    {"ff",1},  
    {"gg",3},  
    {"hh",5},  
    {"ii",3},  
    {"jj",7},  
    {"kk",7} };
```

```
void scan_m(mtree A[], int n, int k,int level){ // k - индекс текущей вершины  
    printf("l=%d node=%d s=%s\n",level,k,A[k].s);  
    for (int i=0;i<n;i++)  
        if (A[i].parent==k) scan_m(A,n,i,level+1);  
}
```



Представление деревьев

Пирамида: Дерево в массиве с вычислением адресов потомков
Потомки для $n \rightarrow 2n$ и $2n+1$, корень $n=1$

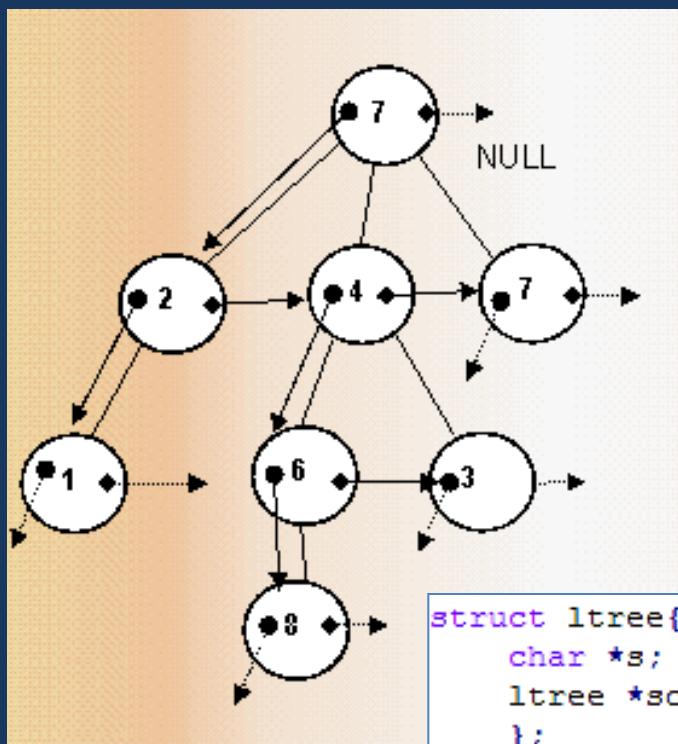


```
void scan_2(int A[], int n, int k,int level){ // k - индекс текущей вершины
    if (k>=n) return;
    if (A[k]==-1) return;
    printf("l=%d node=%d val=%d\n",level,k,A[k]);
    scan_2(A,n,2*k,level+1);
    scan_2(A,n,2*k+1,level+1);
}
//          0 1 2 3 4 5 6 7 8 9 10
int A2[]={-1,1,2,3,4,5,6,7,8,9,10};
```



Представление деревьев

Ветвящийся список: «старший сын» и «следующий брат»



```
void scan_l(ltree *p, int level){  
    if (p==NULL) return;  
    printf("l=%d val=%s\n",level,p->s);  
    for (ltree *q=p->son;q!=NULL;q=q->bro)  
        scan_l(q,level+1);  
}
```

```
struct ltree{  
    char *s;  
    ltree *son,*bro; // Указатели на старшего сына  
}; // и младшего брата
```

```
ltree A={"aa",NULL,NULL}, // Последняя в списке  
      B={"bb",NULL,&A},  
      C={"cc",NULL,&B}, // Список потомков - концевых вершин А, В, С  
      D={"dd",NULL,NULL},  
      E={"ee",&C,NULL},  
      F={"ff",&D,&E}, // Список потомков G - вершин F, Е  
      G={"gg",&F,NULL},  
      *ph = &G;
```



Представление деревьев

Массив ветвей (индекс предка-индекс потомка)

«Естественное» представление: массив указателей (ссылок) на потомков

```
#define N 4
struct tree{
    char *s;
    int n;                                // Количество
    tree *ch[N];
};

tree      H1={"aa",0},
          B1={"bb",0},
          C1={"cc",0},
          D1={"dd",0},
          E1={"ee",3,&C1,&B1,&H1},
          F1={"ff",0},
          G1={"gg",3,&F1,&E1,&D1},
          *ph1 = &G1;

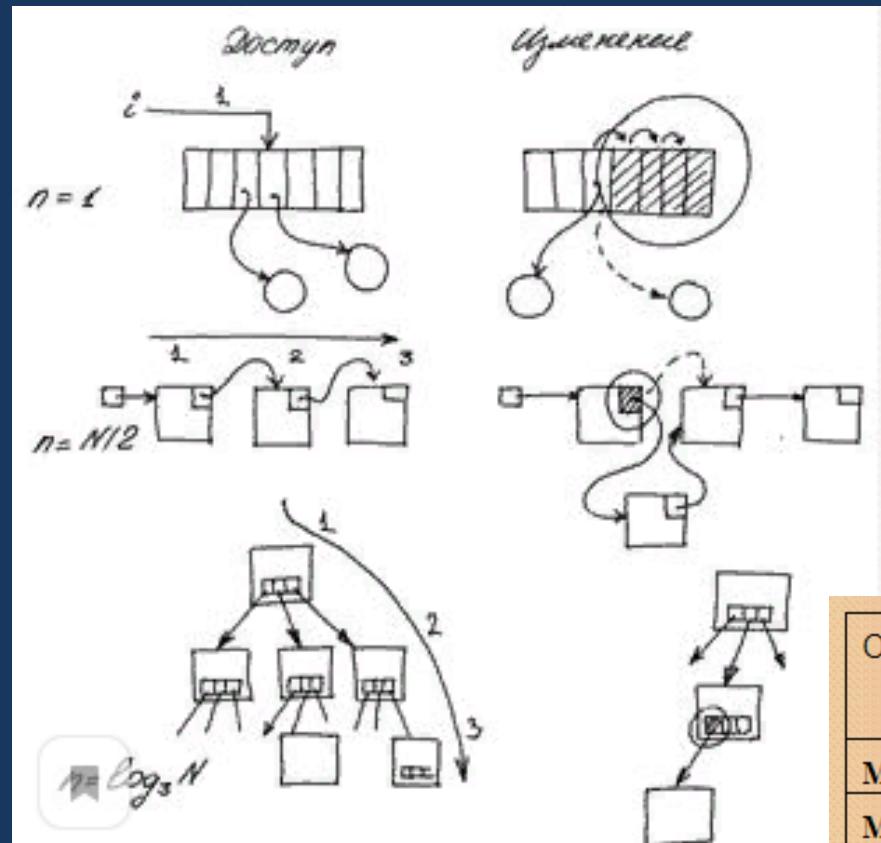
void scan(tree *p, int level){
    if (p==NULL) return;
    printf("l=%d val=%s\n",level,p->s);
    for (int i=0; i<p->n; i++)
        scan(p->ch[i],level+1);
}
```



Трудоемкость алгоритмов на деревьях

Связь между «глубиной» дерева и количеством вершин:

$$N = 1 + m + m^2 + m^3 + \dots + m^L < m^L, L < \log_m N$$



Оценка Т	Извлечение по ЛН	Вставка, удаление	Поиск по значению (упоряд.)
Массив	1	N	$\log(N)$
Массив указателей	$1(2)$	N	$\log(N)$
Список	N	$2(3)$	N
Дерево сбаланс.	$\log(N)$	$1\dots\log(N)$	$\log(N)$
Дерево несбаланс.	$\log(N)\dots N$	$1\dots\log(N)$	$\log(N)\dots N$



Рекурсивный обход дерева

Линейный поиск в дереве, параметры дерева в целом:

- Индуктивный подход: текущая вершина возвращает ???, которое формируется от полученного от потомков ??? по и известному алгоритму (сумма ЧЕГО, максимум ЧЕГО)

```
// Алгоритмы, основанные на полном обходе дерева
struct tree1{
    int val;
    int n;
    tree1 *ch[10];};
//----- Количество вершин в дереве
int F1(tree1 *p){
    int s=1;
    for (int i=0;i < p->n; i++) s+=F1(p->ch[i]);
    return s;}
//----- Сумма значений в вершине дерева
int F2(tree1 *p){
    int s=p->val;
    for (int i=0;i < p->n; i++) s+=F2(p->ch[i]);
    return s;}
```

```
//----- Максимальное значение в вершине дерева
int F3(tree1 *p){
    int s=p->val;
    for (int i=0;i < p->n; i++)
        { int vv=F3(p->ch[i]); if (vv > s) s=vv; }
    return s;}
//----- Максимальная длина ветви дерева
int F4(tree1 *p){
    int s=0;
    for (int i=0;i < p->n; i++)
        { int vv=F4(p->ch[i]); if (vv > s) s=vv; }
    return s+1;}
```

```
//----- Суммарное расстояние до корня - степень сбалансированности
int F6(tree1 *p, int L){
    int s=L;
    for (int i=0;i < p->n; i++)
        s+=F6(p->ch[i],L+1);
    return s;}
```

```
//----- Поиск первого значения, удовлетворяющего условию
tree1 *F7(tree1 *p, int vv){                                // Действие, выполняемое потомком
    if (p->val ==vv) return p;                                // Найдено в текущей вершине - вернуть
    for (int i=0;i < p->n; i++){
        tree1 *q=F7(p->ch[i]);                            // Найдено у потомка – прекратить обход
        if (q!=NULL) return q; }                                // Обработка результата предком
    return NULL;}
```



Структуры данных на деревьях

Логический (порядковый номер) – зависит от порядка обхода (рекурсивного):

- «эгоистический» - предок – потомки
- «альtruистический» - потомки – предок
- «слева направо» - двоичное дерево, левое поддерево – предок – правое поддерево

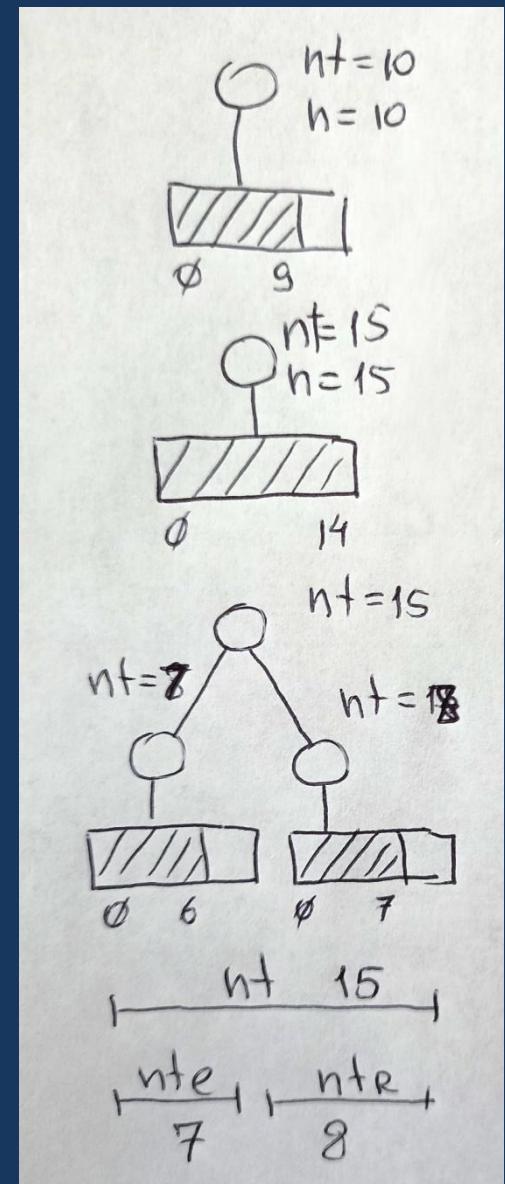
Размещение данных:

- Линейная неупорядоченная структура данных на основе нумерации вершин
- Упорядоченные данные - двоичное дерево (слева направо)
- Вертикально упорядоченное дерево – минимум в вершине дерева



Дерево с данными в конечных вершинах

```
struct node{  
    int size;           // Размерность ДМУ  
    int n;             // Количество значений в ДМУ  
    int nt;            // Общее количество в поддереве  
    node *left,*right;  
    int *data;          // ДМ данных  
};  
  
node *create(int sz0){      // Конечная вершина без данных  
    node *pp = new node();  
    pp->left=pp->right=NULL;  
    pp->size=sz0;  
    pp->data = new int[sz0];  
    pp->n = pp->nt = 0;  
}  
  
,  
void extend(node *pp){      // Проверить конечную и разделить на 2  
if (pp->n!=pp->size)  
    return;  
pp->left = create(pp->size);  
pp->right = create(pp->size);  
int dd = pp->size/2;  
for(int i=0;i<dd;i++)        // Половина в левое поддерево  
    pp->left->data[i]=pp->data[i];  
pp->left->n = pp->left->nt = dd;  
for(int i=dd;i<pp->size;i++) // Половина в правое поддерево  
    pp->right->data[i-dd]=pp->data[i];  
pp->right->n = pp->right->nt = pp->size - dd;  
}
```

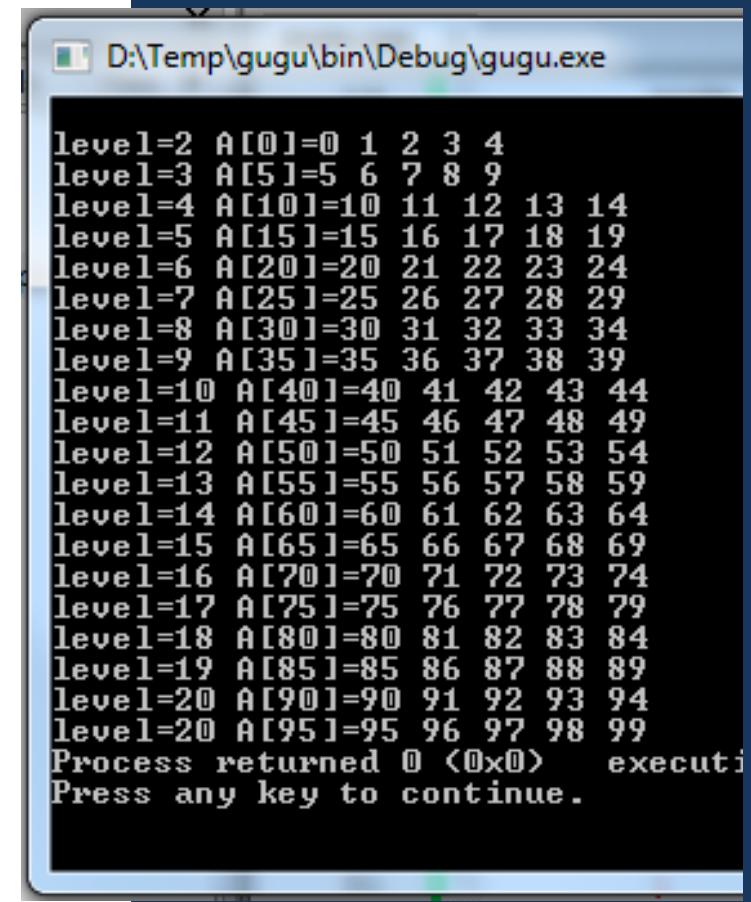




Дерево с данными в конечных вершинах

```
void scan(node *pp, int level, int idx){  
    if (pp==NULL) return;  
    if (pp->left==NULL){           // Конечная вершина  
        printf("\nlevel=%d A[%d]=",level,idx);  
        for(int i=0;i<pp->n;i++)  
            printf("%d ",pp->data[i]);  
    }  
    else{  
        scan(pp->left,level+1,idx);  
        scan(pp->right,level+1,idx+pp->left->n);  
    }  
}  
  
void add(node *pp, int v){  
    if (pp->left==NULL){  
        pp->data[pp->n++] = v;  
        pp->nt++;  
        extend(pp);  
    }  
    else{  
        add(pp->right,v);  
    }  
}  
  
int main(){  
    node *root = create(10);  
    for(int i=0;i<100;i++)  
        add(root,i);  
    scan(root,1,0);  
}
```

Нужно в обоих ветках



```
D:\Temp\gugu\bin\Debug\gugu.exe  
  
level=2 A[0]=0 1 2 3 4  
level=3 A[5]=5 6 7 8 9  
level=4 A[10]=10 11 12 13 14  
level=5 A[15]=15 16 17 18 19  
level=6 A[20]=20 21 22 23 24  
level=7 A[25]=25 26 27 28 29  
level=8 A[30]=30 31 32 33 34  
level=9 A[35]=35 36 37 38 39  
level=10 A[40]=40 41 42 43 44  
level=11 A[45]=45 46 47 48 49  
level=12 A[50]=50 51 52 53 54  
level=13 A[55]=55 56 57 58 59  
level=14 A[60]=60 61 62 63 64  
level=15 A[65]=65 66 67 68 69  
level=16 A[70]=70 71 72 73 74  
level=17 A[75]=75 76 77 78 79  
level=18 A[80]=80 81 82 83 84  
level=19 A[85]=85 86 87 88 89  
level=20 A[90]=90 91 92 93 94  
level=20 A[95]=95 96 97 98 99  
Process returned 0 (0x0)  executi  
Press any key to continue.
```



Дерево с данными в конечных вершинах

Далее – ООП нотация:

- Функции – методы класса
- Рекурсия – вызов метода через указатель на потомка
- Текущая вершина – текущий объект (контекст класса)

```
struct node{
    int size;           // Размерность ДМУ
    int n;             // Количество значений в ДМУ
    int nt;            // Общее количество в поддереве
    node *left,*right;
    int *data;          // ДМ данных
    node(int sz0){      // Конечная вершина без данных
        left=right=NULL;
        size=sz0;
        data = new int[sz0];
        n = nt = 0;
    }
}
```

```
void scan(int level, int idx){
    if (left==NULL){           // Конечная вершина
        printf("\nlevel=%d A[%d]=",level,idx);
        for(int i=0;i<n;i++)
            printf("%d ",data[i]);
    }
    else{
        left->scan(level+1,idx);
        right->scan(level+1,idx+left->n);
    }
}
```

```
void add(int v){
    nt++;
    if (left==NULL){
        data[n++] = v;
        extend();
    }
    else{
        right->add(v);
    }
}
int main(){
    node *root = new node(10);
    for(int i=0;i<100;i++)
        root->add(i);
    root->scan(1,0);
}
```



Дерево с данными в конечных вершинах

```
// Обход с линейным слиянием данных в массив
void scanForBalance(int v[], int &idx){
    if (left==NULL){
        for(int i=0;i<n;i++)
            v[idx++] = data[i];
    }
    else{
        left->scanForBalance(v, idx);
        right->scanForBalance(v, idx);
    }
}

// Заполнение вершин делением интервала пополам
void loadFrom(int v[], int a, int b, int sz){
    nt = b-a+1;
    if (nt>=sz){
        int m = (a+b+1)/2;
        left = new node(sz);
        left->loadFrom(v, a, m-1, sz);
        right = new node(sz);
        right->loadFrom(v, m, b, sz);
    }
    else{
        n = nt;
        for(int i=0;i<nt;i++)
            data[i]=v[a+i];
    }
}

node *balance(){
    int *vv = new int[nt];
    int idx=0;
    scanForBalance(vv, idx);
    node *out = new node(size);
    out->loadFrom(vv, 0, nt-1, size);
    return out;
}
```

Балансировка – выравнивание ветвей дерева

```
level=1 total=100 A[0]=
level=2 total=50 A[0]=
level=3 total=25 A[0]=
level=4 total=12 A[0]=
level=5 total=6 A[0]=0 1 2 3 4 5
level=5 total=6 A[6]=6 7 8 9 10 11
level=4 total=13 A[12]=
level=5 total=6 A[12]=12 13 14 15 16 17
level=5 total=7 A[18]=18 19 20 21 22 23 24
level=3 total=25 A[25]=
level=4 total=12 A[25]=
level=5 total=6 A[25]=25 26 27 28 29 30
level=5 total=6 A[31]=31 32 33 34 35 36
level=4 total=13 A[37]=
level=5 total=6 A[37]=37 38 39 40 41 42
level=5 total=7 A[43]=43 44 45 46 47 48 49
level=2 total=50 A[50]=
level=3 total=25 A[50]=
level=4 total=12 A[50]=
level=5 total=6 A[50]=50 51 52 53 54 55
level=5 total=6 A[56]=56 57 58 59 60 61
level=4 total=13 A[62]=
level=5 total=6 A[62]=62 63 64 65 66 67
level=5 total=7 A[68]=68 69 70 71 72 73 74
level=3 total=25 A[75]=
level=4 total=12 A[75]=
level=5 total=6 A[75]=75 76 77 78 79 80
level=5 total=6 A[81]=81 82 83 84 85 86
level=4 total=13 A[87]=
level=5 total=6 A[87]=87 88 89 90 91 92
level=5 total=7 A[93]=93 94 95 96 97 98 99
```



Дерево с данными в конечных вершинах

```
}

void insert(int idx,int v){
    if (idx>=nt){
        add(v);
        return;
    }
    if (left==NULL){
        for(int k=n-1;k>=idx;k--)
            data[k+1]=data[k];
        data[idx]=v;
        n++;
        nt++;
        extend();
        return;
    }
    nt++;
    if (idx<left->nt)
        left->insert(idx,v);
    else
        right->insert(idx-left->nt,v);
    }
}

int main(){
    node *root = new node(10);
    for(int i=0;i<100;i++)
        root->add(i);
    root->scan(1,0);
    node *two = root->balance();
    for(int k=0;k<20;k++)
        two->insert(27,k+200);
    two->scan(1,0);
}
```

```
level=1 total=120 A[0]=
level=2 total=70 A[0]=
level=3 total=25 A[0]=
level=4 total=12 A[0]=
level=5 total=6 A[0]=0 1 2 3 4 5
level=5 total=6 A[6]=6 7 8 9 10 11
level=4 total=13 A[12]=
level=5 total=6 A[12]=12 13 14 15 16 17
level=5 total=7 A[18]=18 19 20 21 22 23 24
level=3 total=45 A[25]=
level=4 total=32 A[25]=
level=5 total=26 A[25]=
level=6 total=21 A[25]=
level=7 total=16 A[25]=
level=8 total=11 A[25]=
level=9 total=6 A[25]=25 26 219 218 217 216
level=9 total=5 A[31]=215 214 213 212 211
level=8 total=5 A[36]=210 209 208 207 206
level=7 total=5 A[41]=205 204 203 202 201
level=6 total=5 A[46]=200 27 28 29 30
level=5 total=6 A[51]=31 32 33 34 35 36
level=4 total=13 A[57]=
level=5 total=6 A[57]=37 38 39 40 41 42
level=5 total=7 A[63]=43 44 45 46 47 48 49
level=2 total=50 A[70]=
level=3 total=25 A[70]=
level=4 total=12 A[70]=
level=5 total=6 A[70]=50 51 52 53 54 55
level=5 total=6 A[76]=56 57 58 59 60 61
level=4 total=13 A[82]=
level=5 total=6 A[82]=62 63 64 65 66 67
level=5 total=7 A[88]=68 69 70 71 72 73 74
level=3 total=25 A[95]=
level=4 total=12 A[95]=
level=5 total=6 A[95]=75 76 77 78 79 80
level=5 total=6 A[101]=81 82 83 84 85 86
level=4 total=13 A[107]=
level=5 total=6 A[107]=87 88 89 90 91 92
level=5 total=7 A[113]=93 94 95 96 97 98 99
```

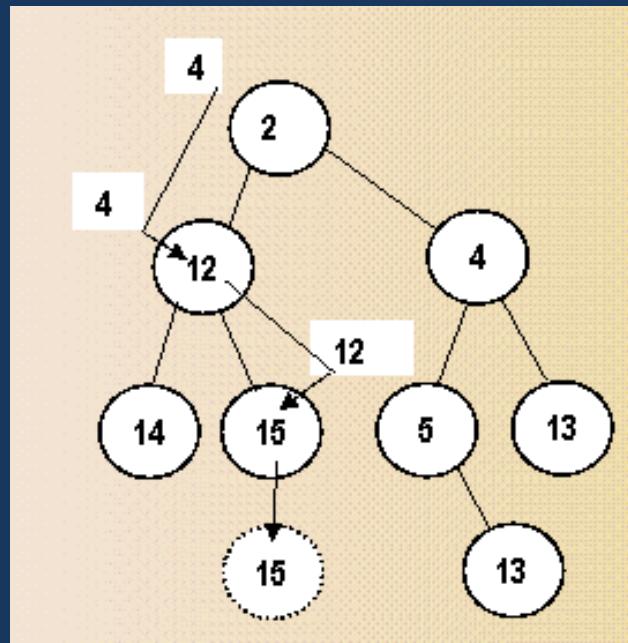
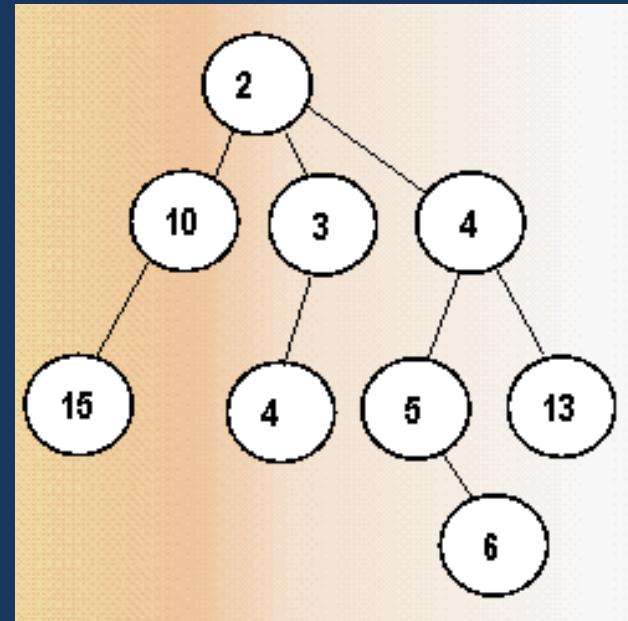


Вертикально упорядоченное дерево

Основная идея: в вершине дерева всегда минимальное значение

Построение дерева:

- алгоритм основывается на ветвлении: выполнение его для корневой вершины сопровождается выбором потомка, для которого алгоритм повторяется линейно - рекурсивно или циклически
- «замещение»: если новое значение меньше, чем значение в текущей вершине, то оно вытесняет последнее, занимая его место. Вытесненное значение участвует в последующем погружении вместо нового. Такой процесс не нарушает свойства упорядоченности дерева в глубину
- для продолжения алгоритма может быть выбран любой потомок. Если выбирать потомка с минимальным количеством вершин в поддереве, то в процессе его построения будет поддерживаться его сбалансированность





Вертикально упорядоченное дерево

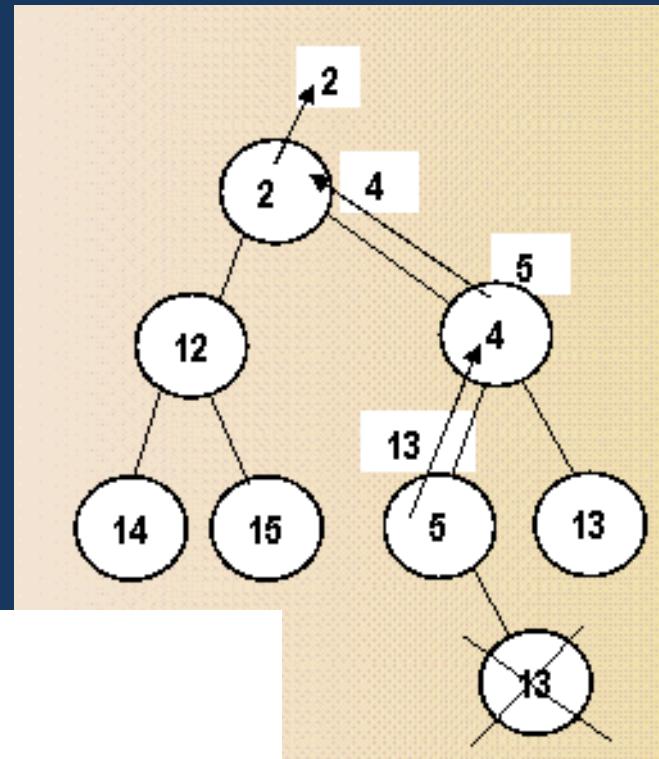
Используется ссылка на указатель: ссылка может быть NULL, тогда на нее подвешивается новая вершина

```
// Построение дерева - вытеснение в в поддерево
// с минимальным количеством вершин
void insert(vtree *&p, int v) {
    if (p==NULL){                                // Найдено свободное место
        p=new vtree;                            // Создать новую вершину
        p->l = p->r = NULL;
        p->cnt = 1;
        p->val = v;
        return;
    }
    p->cnt++;                                // Увеличить счетчик вершин
    if (v < p->val){                          // Вытеснение мешающим большего
        int c=p->val; p->val=v; v=c;          // из текущей вершины
    }
    if (p->l == NULL || p->r !=NULL && p->l->cnt < p->r->cnt)
        insert(p->l,v);                      // Выбор свободной ветви или
    else                                         // поддерева с минимумом вершин
        insert(p->r,v);
}
```



Вертикально упорядоченное дерево

Получение данных: «снятие» данных, с замещением минимального значения от прямых потомков



```
// "Сливание" дерева - замещение значения в текущей вершине
// значением минимального потомка
int shift(vtree *&p){
    if (p==NULL) return 0;
    int vv=p->val;
    if (p->l==NULL && p->r==NULL){
        delete p;
        p = NULL;
        return vv;
    }
    if (p->l==NULL || p->r!=NULL && p->r->val < p->l->val)
        p->val = shift(p->r);
    else
        p->val = shift(p->l);
    return vv;
}
```

// Возвратить значение из текущей
// Нет потомков - удалить вершину
// и возвратить ее значение

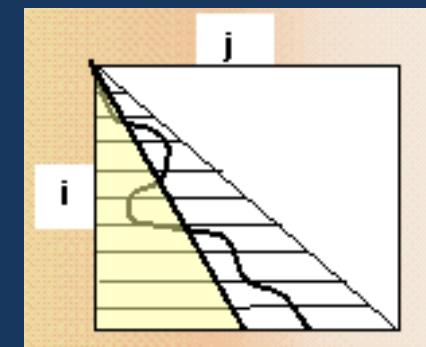
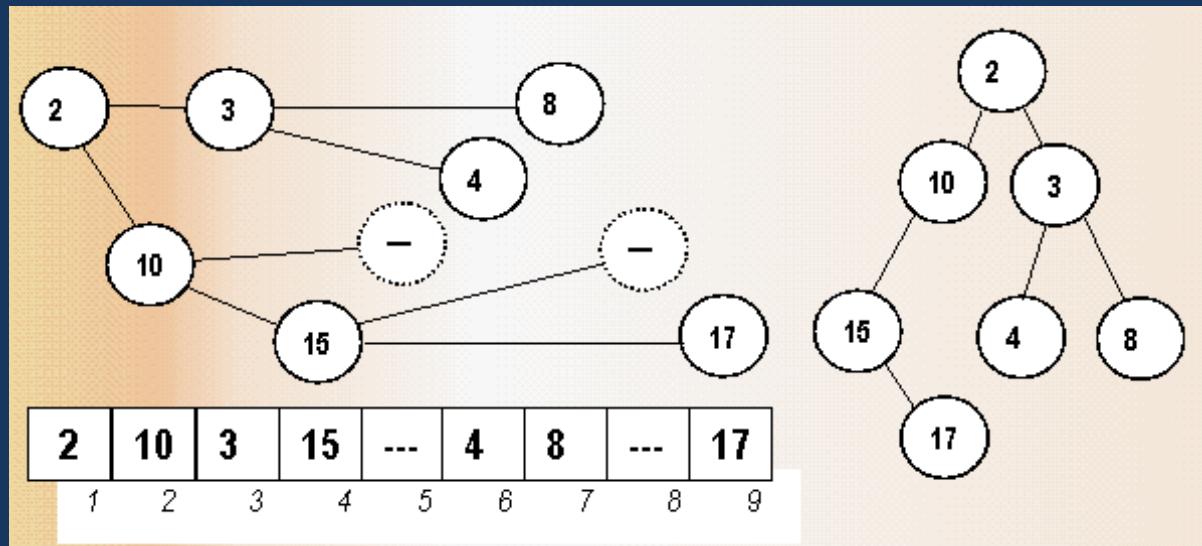
// Выбрать минимальное значение
// от потомков и заместить текущее

// вернуть значение вершины



Пирамидальная сортировка

Вертикально упорядоченное дерево с 2 потомками в массиве с вычисляемыми индексами потомков (пирамида)



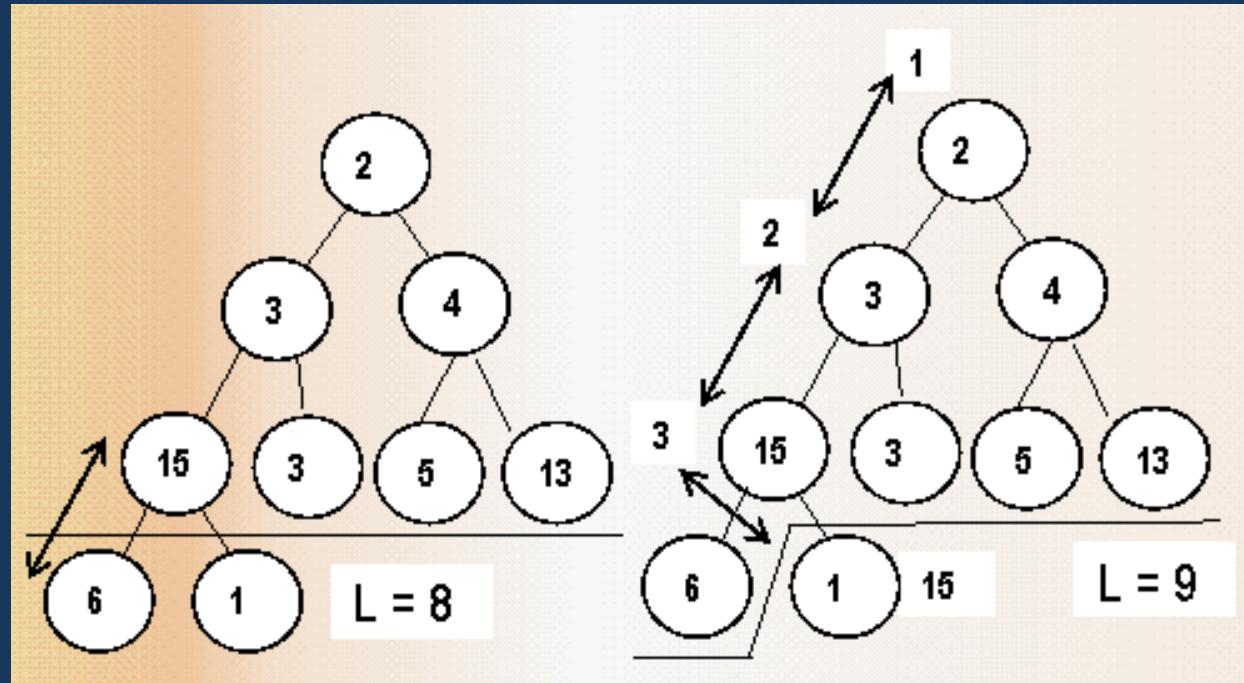
Сравнение с «погружением» - шаг =1, трудоемкость $N^2/4$ $T=O(N^2)$
Пирамида = переменный шаг погружения $1,2,4,8\dots = \log_2 N$ $T=O(N \log_2 N)$

```
//----- Вставка погружением, подозрительно похожая на обмен
void sort(int in[],int n) {
    for ( int i=1; i<n; i++) // Пока не достигли "дна" или меньшего себя
        for ( int k=i; k !=0 && in[k] < in[k-1]; k--) {
            int c=in[k]; in[k]=in[k-1]; in[k-1]=c;
        }
}
```



Пирамидальная сортировка

```
// Погружение в дерево путем обмена с предком
void set_tree(int A[],int n){
int l,i;
for (l=2;l<=n;l++) {
    for (i=l; i!=1;i=i/2) {           // 1 - индекс погружающей вершины
        if (A[i]>A[i/2]) break;      // индекс предка = i/2
        int c=A[i];
        A[i]=A[i/2];                // предок меньше потомка - выйти
        A[i/2]=c;                   // поменяться с предком
    }
}
```

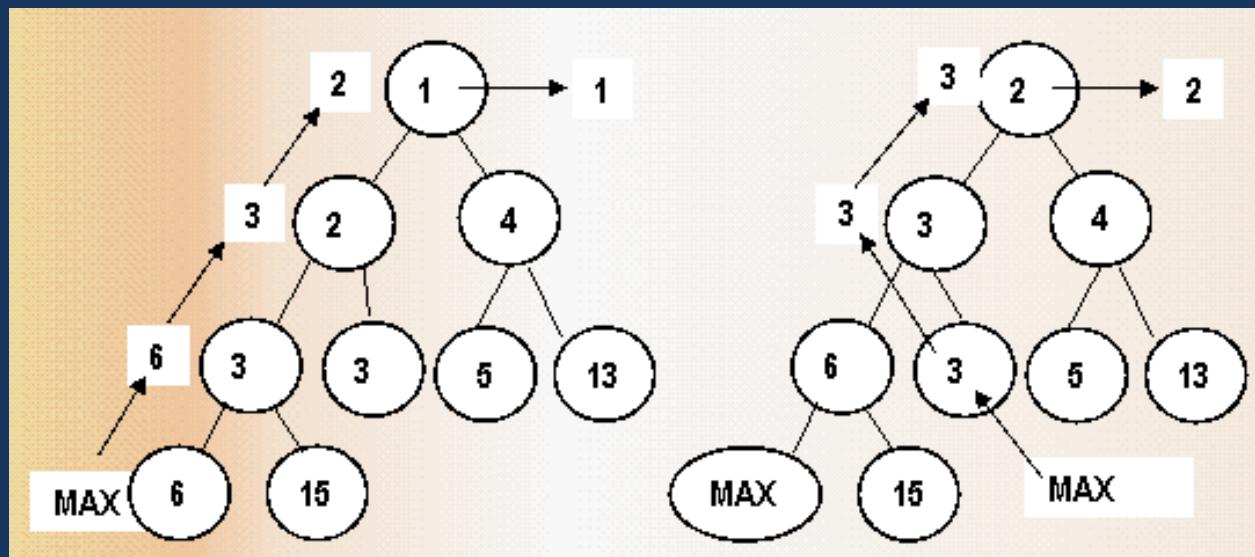




Пирамидальная сортировка

```
// --ортировка - n раз выбрать из дерева с замещением на минимального потомка
// »звлекаемые элементы в конце массива, не имеющие потомков,
// заменяются на MAXINT. «значение MAXINT обозначает отсутствие вершины
void sort(int A[],int B[], int n){
for (int i=1; i<=n; i++) {           // повторить n раз
    int j,k;
    B[i]=A[1];
    for (j=1;j<=n;j=k) {
        k=2*j;
        if (k>n) { A[j]=MAXINT; break; }
        if (k+1>n) { A[j]=A[k]; A[k]=MAXINT; break; }
        if (A[k+1]<A[k]) k++;
        A[j]=A[k];
    }
}}
```

// j=k - переход к замещающему потомку
// k - левый потомок
// нет потомков - MAXINT (затычка)
// нет второго потомка - взять первого
// иначе - выбрать минимального из них
// скопировать значение потомка в предка





Двоичное дерево

BST – Binary Search Tree

- Упорядочено «слева направо»: в левом – меньшие корневой вершины, в правом – большие
- Обход «левое-текущее-правое» - упорядочение по возрастанию
- Поиск, вставка – линейно-рекурсивные алгоритмы (цикл)
- Трудоемкость поиска/включения $\log_2 N$ (сбалансированное), N - вырожденное в список
- Топология дерева зависит от порядка следования значений при включении
- Включение упорядоченной последовательности – вырождение в список
- Балансировка (выравнивание ветвей)
 - Топологические изменения при вставке (Адельсон-Вельский)
 - «Сливание» данных в упорядоченную структуру и построение нового дерева путем деления интервала на 2 равные части с корневой вершиной посередине (аналог «сбора мусора»)



Двоичное дерево



```
// Двоичное дерево
// Обход дерева слева - направо (левое - текущая - правое)
struct btree{
    int cnt;          // Количество вершин в дереве
    char *s;
    btree *l,*r;     // Левый и правый потомки
};

// Обход дерева
void scan(btree *p, int level, int &ln){
    if (p==NULL) return;
    scan(p->l, level+1, ln);
    printf("l=%d n=%d cnt=%d :%s\n", level, ln, p->cnt, p->s);
    ln++;
    scan(p->r, level+1, ln);
}
```



Двоичное дерево

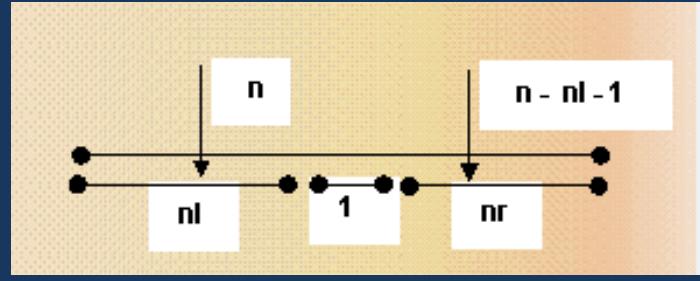
```
// Поиск вершины с заданным значением - рекурсия  
btree *search(btreet *p, char *key){
```

```
    if (p==NULL) return NULL;  
    int k=strcmp(key,p->s);  
    if (k==0) return p;  
    if (k<0) return search(p->l,key);  
    return search(p->r,key);  
}
```

```
// Поиск вершины с заданным значением - цикл
```

```
btreet *search1(btreet *p, char *key){  
    while(p!=NULL){  
        int k=strcmp(key,p->s);  
        if (k==0) return p;  
        if (k<0) p=p->l;  
        else p=p->r;  
    }  
    return NULL; }
```

```
// Включение с сохранением порядка  
void insert(btreet *&p, char *ss){  
    if (p == NULL) { p=create(ss); return; }  
    p->cnt++;  
    if (strcmp(ss,p->s)<0)  
        insert(p->l,ss);  
    else  
        insert(p->r,ss);  
}
```



```
// Создание терминальной вершины
```

```
btreet *create(char *ss){  
    btreet *q=new btreet;  
    q->l = q->r = NULL;  
    q->cnt = 1;  
    q->s = ss;  
    return q; }
```

```
// Поиск вершины по логическому номеру
```

```
btreet *get_n(btreet *p, int n){  
    if (p==NULL) return NULL;  
    if (n >= p->cnt) return NULL;  
    if (p->l!=NULL){  
        int ll=p->l->cnt;  
        if (n<ll) return get_n(p->l,n);  
        n-=ll;  
    }  
    if (n-- ==0) return p;  
    return get_n(p->r,n);  
}
```



Двоичное дерево. Балансировка

```
// Обход с сохранением вершин в порядке возрастания
void scan(btreet *qq[], btreet *p, int &ln){
    if (p==NULL) return;
    scan(qq, p->l,ln);
    qq[ln++]=p;
    scan(qq,p->r,ln);
}

void balance(btreet *&ph) {
    int sz=ph->cnt;
    btreet **pp=new btreet*[sz];
    int idx=0;
    scan(pp,ph,idx);
    ph=balance(pp,0,sz-1);
}

// Обход дерева
void scan(btreet *p, int level, int &ln) {
    if (p==NULL) return;
    scan(p->l, level+1,ln);
    printf("l=%d n=%d cnt=%d :%s\n", level, ln, p->cnt, p->s);
    ln++;
    scan(p->r, level+1,ln);
}

double sum (btreet *p, int level){
    if (p==NULL) return 0;
    return level+sum(p->l,level+1)+sum(p->r,level+1);
}

btreet *balance(btreet *pp[], int a, int b){
    if (a>b) return NULL;
    btreet *q; // Интервал =1 - концевая вершина
    if (a==b) { q=pp[a]; q->l=q->r=NULL; q->cnt=1; return q; }
    int m=(a+b)/2;
    q=pp[m]; // Вершина в середине интервала
    q->l=balance(pp,a,m-1); // Поддерево на левом интервале
    q->r=balance(pp,m+1,b); // Поддерево на правом интервале
    q->cnt=b-a+1;
    return q;
}
```



Двоичное дерево. Удаление вершины

```
// Удаление по логическому номеру
char *remove(btreet *p, int n){ // Возвращает удаляемую строку
    if (p == NULL) return NULL;
    if (n >= p->cnt) return NULL;
    p->cnt--;
    if (p->l!=NULL) {
        int ll=p->l->cnt;
        if (n<ll) // В интервале левого - рекурсия
            return remove(p->l,n);
        n-=ll;
    }
    if (n-- ==0) { // Удалить корневую
        char *ss = p->s;
        btreet *q; // p - ссылка на указатель
                    // (адрс указателя на текущую вершину)
        if (p->l==NULL && p->r==NULL) // Нет потомков
            { delete p; p=NULL; return ss; }
        if (p->l==NULL) // Только правый -
            { q=p->r; delete p; p=q; // указатель на него
              return ss; } // на место указателя на текущий
        if (p->r==NULL) // Только левый -
            { q=p->l; delete p; p=q; return ss; }
        p->s = remove(p->r,0); // на место текущей строки
                                // [0] из правого поддерева
    }
    else // В интервале правого - рекурсия
        return remove(p->r,n);
}
```



Двоичное дерево

```
int main() {
btree *ph=NULL;
int k=0;
char c[80];
c[1]=0;
for(int i=0;i<=26;i++) {
    c[0] = 'A'+i;
    insert(ph,strdup(c));
}
puts("-----");
k=0;scan(ph,1,k);
printf("mid=%lf\n",sum(ph,1)/ph->cnt);
getchar();
balance(ph);
puts("-----");
k=0;scan(ph,1,k);
printf("mid=%lf\n",sum(ph,1)/ph->cnt);
getchar();
FILE *fd=fopen("revolution.txt","r");
//-----
```

D:\Temp\fufu\bin\Debug\fufu

```
l=5 n=4 cnt=23 :E
l=6 n=5 cnt=22 :F
l=7 n=6 cnt=21 :G
l=8 n=7 cnt=20 :H
l=9 n=8 cnt=19 :I
l=10 n=9 cnt=18 :J
l=11 n=10 cnt=17 :K
l=12 n=11 cnt=16 :L
l=13 n=12 cnt=15 :M
l=14 n=13 cnt=14 :N
l=15 n=14 cnt=13 :O
l=16 n=15 cnt=12 :P
l=17 n=16 cnt=11 :Q
l=18 n=17 cnt=10 :R
l=19 n=18 cnt=9 :S
l=20 n=19 cnt=8 :T
l=21 n=20 cnt=7 :U
l=22 n=21 cnt=6 :V
l=23 n=22 cnt=5 :W
l=24 n=23 cnt=4 :X
l=25 n=24 cnt=3 :Y
l=26 n=25 cnt=2 :Z
l=27 n=26 cnt=1 :[
mid=14.000000
```

$$L = N/2$$

$$L = \log_2(N)-1$$

```
-----  
l=4 n=0 cnt=2 :A  
l=5 n=1 cnt=1 :B  
l=3 n=2 cnt=6 :C  
l=5 n=3 cnt=1 :D  
l=4 n=4 cnt=3 :E  
l=5 n=5 cnt=1 :F  
l=2 n=6 cnt=13 :G  
l=4 n=7 cnt=2 :H  
l=5 n=8 cnt=1 :I  
l=3 n=9 cnt=6 :J  
l=5 n=10 cnt=1 :K  
l=4 n=11 cnt=3 :L  
l=5 n=12 cnt=1 :M  
l=1 n=13 cnt=27 :N  
l=4 n=14 cnt=2 :O  
l=5 n=15 cnt=1 :P  
l=3 n=16 cnt=6 :Q  
l=5 n=17 cnt=1 :R  
l=4 n=18 cnt=3 :S  
l=5 n=19 cnt=1 :T  
l=2 n=20 cnt=13 :U  
l=4 n=21 cnt=2 :V  
l=5 n=22 cnt=1 :W  
l=3 n=23 cnt=6 :X  
l=5 n=24 cnt=1 :Y  
l=4 n=25 cnt=3 :Z  
l=5 n=26 cnt=1 :[  
mid=4.037037
```



Двоичное дерево

```
//  
FILE *fd=fopen("revolution.txt","r");  
ph=NULL;  
while(fgets(c,80,fd)!=NULL) {  
    c[strlen(c)-1]=0;           // Затереть \n в конце "aaaaaa\n"  
    insert(ph,strdup(c));  
}  
fclose(fd);  
puts("-----");  
k=0;scan(ph,1,k);  
printf("mid=%lf\n",sum(ph,1)/ph->cnt);  
getchar();  
balance(ph);  
puts("-----");  
k=0;scan(ph,1,k);  
printf("mid=%lf\n",sum(ph,1)/ph->cnt);  
getchar();  
insert(ph,"Aaaaa");  
insert(ph,"Bbbbb");  
insert(ph,"Ccccc");  
insert(ph,"Ddddd");  
insert(ph,"Eeeee");  
insert(ph,"Fffff");  
puts(remove(ph,14));  
puts(remove(ph,14));  
puts(remove(ph,10));  
puts(remove(ph,10));  
puts("-----"); k=0;scan(ph,1,k);  
getchar();  
}
```

```
-----  
l=5 n=0 cnt=1 :  
l=4 n=1 cnt=3 :  
l=5 n=2 cnt=1 :  
l=3 n=3 cnt=8 :  
l=5 n=4 cnt=1 :  
l=4 n=5 cnt=4 :  
l=5 n=6 cnt=2 :  
l=6 n=7 cnt=1 :All i can tell you  
l=2 n=8 cnt=18 :Alright alright  
l=5 n=9 cnt=1 :Alright alright  
l=4 n=10 cnt=4 :Alright alright  
l=5 n=11 cnt=2 :Beatles - Revoluti  
l=6 n=12 cnt=1 :But if you go carr  
l=3 n=13 cnt=9 :But when you talk  
l=5 n=14 cnt=1 :But when you want  
l=4 n=15 cnt=4 :Don't you know it'  
l=5 n=16 cnt=2 :Don't you know it'  
l=6 n=17 cnt=1 :Don't you know kno  
l=1 n=18 cnt=38 :Beatles - Revoluti  
l=6 n=19 cnt=1 :But if you go carr  
l=5 n=20 cnt=7 :But when you talk  
l=8 n=21 cnt=1 :But when you want  
l=7 n=22 cnt=4 :Don't you know it'  
l=1 n=23 cnt=38 :Beatles - Revoluti  
l=6 n=24 cnt=1 :But if you go carr  
l=3 n=25 cnt=9 :We're doing what w  
l=5 n=26 cnt=1 :Well you know  
l=4 n=27 cnt=4 :Well you know  
l=5 n=28 cnt=2 :Well you know  
l=6 n=29 cnt=1 :Well you know  
l=2 n=30 cnt=19 :Well you know  
l=5 n=31 cnt=1 :Well you know  
l=4 n=32 cnt=4 :Well you know  
l=3 n=33 cnt=9 :Well you know  
l=5 n=34 cnt=1 :Well you want a  
l=4 n=35 cnt=4 :You say you'll cha  
l=5 n=36 cnt=2 :You tell me it's t  
l=6 n=37 cnt=1 :You tell me that i  
mid=4.500000  
-----  
l=2 n=38 cnt=26 :You s:  
l=4 n=39 cnt=2 :You say you'll change the consti  
l=5 n=40 cnt=1 :You tell me it's the institution  
l=3 n=41 cnt=3 :You tell me that it's evolution  
mid=6.078947
```



Двоичное дерево

```
-----  
l=5 n=0 cnt=1 :  
l=4 n=1 cnt=3 :  
l=5 n=2 cnt=1 :  
l=3 n=3 cnt=8 :  
l=5 n=4 cnt=1 :  
l=4 n=5 cnt=4 :  
l=5 n=6 cnt=2 :  
l=6 n=7 cnt=1 :All i can tell you  
l=2 n=8 cnt=18 :Alright alright  
l=5 n=9 cnt=1 :Alright alright  
l=4 n=10 cnt=4 :Alright alright  
l=5 n=11 cnt=2 :Beatles - Revoluti  
l=6 n=12 cnt=1 :But if you go carr  
l=3 n=13 cnt=9 :But when you talk  
l=5 n=14 cnt=1 :But when you want  
l=4 n=15 cnt=4 :Don't you know it'  
l=5 n=16 cnt=2 :Don't you know it'  
l=6 n=17 cnt=1 :Don't you know kno  
l=1 n=18 cnt=38 :Don't you know yo  
l=5 n=19 cnt=1 :We all want to cha  
l=4 n=20 cnt=4 :We all want to cha  
l=5 n=21 cnt=2 :We all want to cha  
l=6 n=22 cnt=1 :We'd all love to s  
l=3 n=23 cnt=9 :We're doing what w  
l=5 n=24 cnt=1 :Well you know  
l=4 n=25 cnt=4 :Well you know  
l=5 n=26 cnt=2 :Well you know  
l=6 n=27 cnt=1 :Well you know  
l=2 n=28 cnt=19 :Well you know  
l=5 n=29 cnt=1 :Well you know  
l=4 n=30 cnt=4 :You ain't going to  
l=5 n=31 cnt=2 :You ask me for a c  
l=6 n=32 cnt=1 :You better free yo  
l=3 n=33 cnt=9 :You say you got a  
l=5 n=34 cnt=1 :You say you want a  
l=4 n=35 cnt=4 :You say you'll cha  
l=5 n=36 cnt=2 :You tell me it's t  
l=6 n=37 cnt=1 :You tell me that i  
mid=4.500000
```

```
-----  
n1right n1right  
l=5 n=0 cnt=1 :  
l=4 n=1 cnt=3 :  
l=5 n=2 cnt=1 :  
l=3 n=3 cnt=9 :  
l=5 n=4 cnt=1 :  
l=4 n=5 cnt=5 :  
l=5 n=6 cnt=3 :  
l=7 n=7 cnt=1 :Aaaaaa  
l=6 n=8 cnt=2 :All i can tell you is brother you ha  
l=2 n=9 cnt=18 :Alright alright  
l=5 n=10 cnt=1 :Bbbbb  
l=4 n=11 cnt=2 :Beatles - Revolution (lennon/mccart  
l=3 n=12 cnt=8 :But when you want money for people  
l=5 n=13 cnt=2 :Ccccc  
l=6 n=14 cnt=1 :Dddddd  
l=4 n=15 cnt=5 :Don't you know it's gonna be alrigh  
l=5 n=16 cnt=2 :Don't you know it's gonna be alrigh  
l=6 n=17 cnt=1 :Don't you know know it's gonna be a  
l=1 n=18 cnt=40 :Don't you know you can count me ou  
l=6 n=19 cnt=2 :Eeeee  
l=7 n=20 cnt=1 :Fffff  
l=5 n=21 cnt=3 :We all want to change the world  
l=4 n=22 cnt=6 :We all want to change the world  
l=5 n=23 cnt=2 :We all want to change your head  
l=6 n=24 cnt=1 :We'd all love to see the plan  
l=3 n=25 cnt=11 :We're doing what we can  
l=5 n=26 cnt=1 :Well you know  
l=4 n=27 cnt=4 :Well you know  
l=5 n=28 cnt=2 :Well you know  
l=6 n=29 cnt=1 :Well you know  
l=2 n=30 cnt=21 :Well you know  
l=5 n=31 cnt=1 :Well you know  
l=4 n=32 cnt=4 :You ain't going to make it with any  
l=5 n=33 cnt=2 :You ask me for a contribution  
l=6 n=34 cnt=1 :You better free your mind instead  
l=3 n=35 cnt=9 :You say you got a real solution  
l=5 n=36 cnt=1 :You say you want a revolution  
l=4 n=37 cnt=4 :You say you'll change the constitut  
l=5 n=38 cnt=2 :You tell me it's the institution  
l=6 n=39 cnt=1 :You tell me that it's evolution
```



Двоичное дерево в массиве

ООП-нотация

```
// Двоичное дерево в массиве
struct btree{
    char **p;
    int sz;
//--
void init(){ sz=10; p=new char*[sz];
    for (int i=0;i<sz;i++) p[i]=NULL;
}
//-- число вершин в поддереве
int size(int n){
    if (n>=sz || p[n]==NULL) return 0;
    return 1+size(2*n)+size(2*n+1); }
//-- Обход дерева
void scan(int n, int level, int &ln){
    if (n>=sz || p[n]==NULL) return;
    scan(2*n,level+1,ln);
    printf("l=%d n=%d :%s\n", level, ln, p[n]);
    ln++;
    scan(2*n+1,level+1,ln);}
//-- Поиск вершины по логическому номеру
char *get_n(int m, int n){
    if (m>=sz || m>=size(n)) return NULL;
    int ll=size(2*n);
    if (m<ll) return get_n(m,2*n);
    m-=ll;
    if (m-- ==0) return p[n];
    return get_n(m,2*n+1);
}
```

```
// Включение с сохранением порядка
void insert(int n, char *ss){
    if (n>=sz) {
        sz*=2;
        p=(char**)realloc(p,sz*sizeof(char*));
        for (int i=sz/2;i<sz;i++) p[i]=NULL;
    }
    if (p[n] == NULL) { p[n]=ss; return; }
    if (strcmp(ss,p[n])<0)
        insert(2*n, ss);
    else
        insert(2*n+1, ss); }
```



Двоичное дерево в массиве

ООП-нотация

```
...  
//--- Балансировка дерева  
void balance(char *pp[], int a, int b){  
    if (a>b) return;  
    int m=(a+b)/2;  
    insert(1,pp[m]);  
    balance(pp,a,m-1);  
    balance(pp,m+1,b);  
}  
//-- Обход дерева с сохранением в линейном массиве ук  
void set(char *pp[], int n, int &ln){  
    if (n>=sz || p[n]==NULL) return;  
    set(pp,2*n,ln);  
    pp[ln++]=p[n];  
    set(pp,2*n+1,ln);}  
  
void balance(){  
    int sz1=size(1),ln=0;  
    char **pp=new char*[sz1];  
    set(pp,1,ln);  
    delete p;  
    init();  
    balance(pp,0,sz1-1);  
}
```

```
double sum(int n, int level){  
    if (n>=sz || p[n]==NULL) return 0;  
    return sum(2*n,level+1)+level+sum(2*n+1,level+1);  
}  
int cnt(int n){  
    if (n>=sz || p[n]==NULL) return 0;  
    return cnt(2*n)+1+cnt(2*n+1);  
}
```



Двоичное дерево в массиве

```
int main(){
btree X;
X.init();
int k=0;
FILE *fd=fopen("revolution.txt","r");
char c[80];
while(fgets(c,80,fd)!=NULL) { c[strlen(c)-1]=0;
fclose(fd);
puts("-----");
k=0;X.scan(1,1,k);
printf("mid=%lf\n",X.sum(1,1)/X.cnt(1));
X.balance();
puts("-----");
k=0;X.scan(1,1,k);
printf("mid=%lf\n",X.sum(1,1)/X.cnt(1));
X.insert(1,"aaaaaa");
X.insert(1,"bbbbbb");
X.insert(1,"cccccc");
X.insert(1,"ddddd");
X.insert(1,"eeeeee");
X.insert(1,"ffffff");
puts("-----");
k=0;X.scan(1,0,k);
for (k=X.size(1)-1;k>=0;k--) puts(X.get_n());
getchar();
}
```

```
l=2 n=0 :
l=3 n=1 :
l=4 n=2 :
l=6 n=3 :
l=8 n=4 :
l=9 n=5 :
l=10 n=6 :
l=7 n=7 :All i can tell you is br
l=5 n=8 :Alright alright
l=4 n=9 :Alright alright
l=5 n=10 :Alright alright
l=6 n=11 :Beatles - Revolution <le
l=7 n=12 :But if you go carrying p
l=3 n=13 :But when you talk about
l=5 n=14 :But when you want money
l=4 n=15 :Don't you know it's gonn
l=5 n=16 :Don't you know it's gonn
l=6 n=17 :Don't you know know it's
l=1 n=18 :Don't you know you can c
l=4 n=19 :We all want to change th
l=5 n=20 :We all want to change th
l=6 n=21 :We all want to change yo
l=7 n=22 :We'd all love to see the
l=4 n=23 :We're doing what we can
l=5 n=24 :Well you know
l=3 n=25 :Well you know
l=4 n=26 :Well you know
l=5 n=27 :Well you know
l=6 n=28 :Well you know
l=9 n=29 :Well you know
l=8 n=30 :You ain't going to make
l=7 n=31 :You ask me for a contrib
l=10 n=32 :You better free your mi
l=7 n=33 :You say you got a real s
l=9 n=34 :You say you want a revol
l=8 n=35 :You say you'll change th
l=9 n=36 :You tell me it's the ins
l=10 n=37 :You tell me that it's e
mid=5.894737
```

```
l=3 n=0 :
l=4 n=1 :
l=5 n=2 :
l=6 n=3 :
l=7 n=4 :
l=8 n=5 :
l=9 n=6 :
l=10 n=7 :All i can tell you is br
l=2 n=8 :Alright alright
l=4 n=9 :Alright alright
l=5 n=10 :Alright alright
l=6 n=11 :Beatles - Revolution <le
l=7 n=12 :But if you go carrying p
l=3 n=13 :But when you talk about
l=5 n=14 :But when you want money
l=4 n=15 :Don't you know it's gonn
l=5 n=16 :Don't you know it's gonn
l=6 n=17 :Don't you know know it's
l=1 n=18 :Don't you know you can c
l=4 n=19 :We all want to change th
l=5 n=20 :We all want to change th
l=6 n=21 :We all want to change yo
l=7 n=22 :We'd all love to see the
l=3 n=23 :We're doing what we can
l=2 n=24 :Well you know
l=3 n=25 :Well you know
l=4 n=26 :Well you know
l=5 n=27 :Well you know
l=6 n=28 :Well you know
l=9 n=29 :Well you know
l=8 n=30 :You ain't going to make
l=7 n=31 :You ask me for a contrib
l=10 n=32 :You better free your mi
l=7 n=33 :You say you got a real s
l=9 n=34 :You say you want a revol
l=8 n=35 :You say you'll change th
l=9 n=36 :You tell me it's the ins
l=10 n=37 :You tell me that it's e
mid=6.078947
```

ООП-нотация