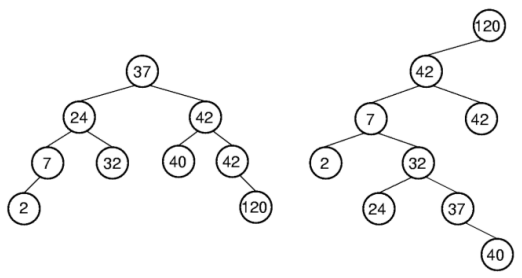
Δυαδικά Δένδρα Αναζήτησης

Ορισμός

Ένα ΔΔΑ είναι δυαδικό δένδρο με διακριτά κλειδιά (τιμές) και τις εξής ιδιότητες:

- Τα κλειδιά (αν υπάρχουν) στο αριστερό υποδένδρο της ρίζας είναι μικρότερα από το κλειδί της ρίζας
- Τα κλειδιά (αν υπάρχουν) στο δεξιό υποδένδρο της ρίζας είναι μεγαλύτερα από το κλειδί της ρίζας
- Το αριστερό και το δεξιό υποδένδρο είναι επίσης ΔΔΑ



<u>Κίνητρο</u>:
να μειώσουμε τους χρόνους ενημέρωσης και αναζήτησης σε λιγότερο από Θ(n)

ΑΤΔ BinarySearchTree

AbstractDataType *BSTree* {

instances: binary trees, each node has an element with a key field; all keys are distinct; keys in the left subtree of any node smaller than the key in the node; those in the right subtree are larger;

operations

Create (): create an empty binary search tree

Search (*k*, *e*): return in *e* the element with key *k*; return false if the operation fails, return true if it succeeds

Insert (*e*): insert element *e* into the search tree

Delete (*k*, *e*): delete the element with key *k* and return it in *e*

Ascend (): Output all elements in ascending order of key

}

Αναζήτηση στοιχείου μέσα σε ΔΔΑ

```
bool BSTree<E,K>::Search(const K& k, E &e) const
{// Search for element that matches k.
   // pointer p starts at the root and moves through
   // the tree looking for an element with key k
   BinaryTreeNode<E> *p = root;
   while (p)                    // examine p->data
      if (k < p->data) p = p->LeftChild;
      else if (k > p->data) p = p->RightChild;
           else {               // found element
                 e = p->data;
                 return true;}
   return false;
}
```

Κόστος αναζήτησης = κόστος κατάβασης = O(h)

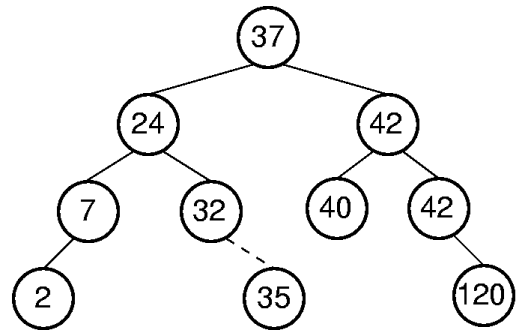
Εισαγωγή στοιχείου σε ΔΔΑ (1)

Βασική ιδιότητα ΔΔΑ:

- Η εισαγωγή γίνεται πάντα σε κάποιο (νέο) φύλλο

Διαδικασία:

- Αναζήτηση του στοιχείου (οπότε καταλήγουμε σε κόμβο-φύλλο)
- Εισαγωγή του ως παιδί εκείνου του κόμβου



Εισαγωγή στοιχείου σε ΔΔΑ (2)

```
BSTree<E,K>& BSTree<E,K>::Insert(const E& e)
{// Insert e if not duplicate.
   BinaryTreeNode<E> *p = root,  // search pointer
                     *pp = 0;    // parent of p
   // find place to insert
   while (p) {                   // examine p->data
      pp = p;
      // move p to a child
      if (e < p->data) p = p->LeftChild;
      else if (e > p->data) p = p->RightChild;
           else throw BadInput(); // duplicate
      }

   // get a node for e and attach to pp
   …
```

Εισαγωγή στοιχείου σε ΔΔΑ (3)

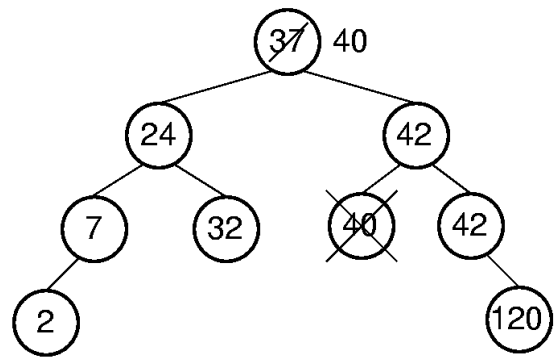
```
   …
   // get a node for e and attach to pp
   BinaryTreeNode<E> *r = new BinaryTreeNode<E> (e);
   if (root) {// tree not empty
      if (e < pp->data) pp->LeftChild = r;
      else pp->RightChild = r;}
   else // insertion into empty tree
        root = r;

   return *this;
}
```

Κόστος = Κόστος αναζήτησης + κόστος 'συγκόλλησης'
νέου κόμβου στον πατέρα-κόμβο = O(h) + O(1) = O(h)

Διαγραφή στοιχείου από ΔΔΑ (1)

3 περιπτώσεις:

- Ο κόμβος p (που περιέχει το στοιχείο) είναι φύλλο
- Το p έχει μόνο ένα μη κενό υποδένδρο
- Το p έχει ακριβώς δύο μη κενά υποδένδρα



Διαγραφή στοιχείου από ΔΔΑ (2)

```
BSTree<E,K>& BSTree<E,K>::Delete(const K& k, E& e)
{// Delete element with key k and put it in e.

   // set p to point to node with key k
   BinaryTreeNode<E> *p = root, // search pointer
                     *pp = 0;   // parent of p
   while (p && p->data != k){// move to a child of p
      pp = p;
      if (k < p->data) p = p->LeftChild;
      else p = p->RightChild;
      }
   if (!p) throw BadInput(); // no element with key k

   e = p->data;  // save element to delete
   ...
```


Διαγραφή στοιχείου από ΔΔΑ (3)

```
   ...
   // restructure tree
   // handle case when p has two children
   if (p->LeftChild && p->RightChild) {// two children
      // convert to zero or one child case
      // find largest element in left subtree of p
      BinaryTreeNode<E> *s = p->LeftChild,
                        *ps = p;  // parent of s
      while (s->RightChild) {// move to larger element
         ps = s;
         s = s->RightChild;}

      // move largest from s to p
      p->data = s->data;
       p = s;
       pp = ps;}
   ...
```

Διαγραφή στοιχείου από ΔΔΑ (4)

```
   ...
   // p has at most one child
   // save child pointer in c
   BinaryTreeNode<E> *c;
   if (p->LeftChild) c = p->LeftChild;
   else c = p->RightChild;

   // delete p
   if (p == root) root = c;
   else {// is p left or right child of pp?
         if (p == pp->LeftChild)
              pp->LeftChild = c;
         else pp->RightChild = c;}
   delete p;

   return *this;
}
```

Κόστος = ?

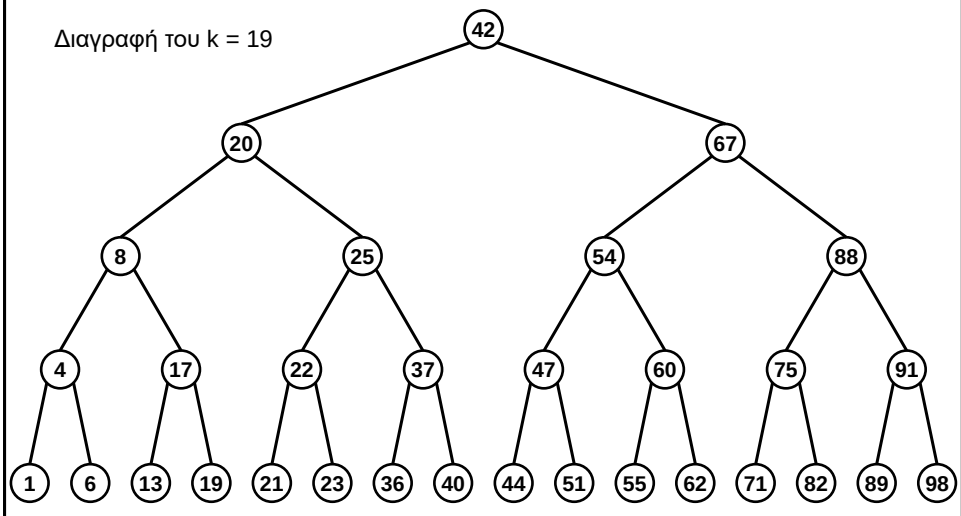
Το ΔΔΑ περιέχει μόνο 1 κόμβο (ρίζα)

Διαγραφή του k = 42


```
p = root;
pp = 0;

// move to a child of p
while (p && p->data != k){
    pp = p;
    if (k < p->data) p = p->LeftChild;
    else p = p->RightChild;
}

...

if (p->LeftChild) c = p->LeftChild;
else c = p->RightChild;

// delete p
if (p == root) root = c;
...
delete p;
```

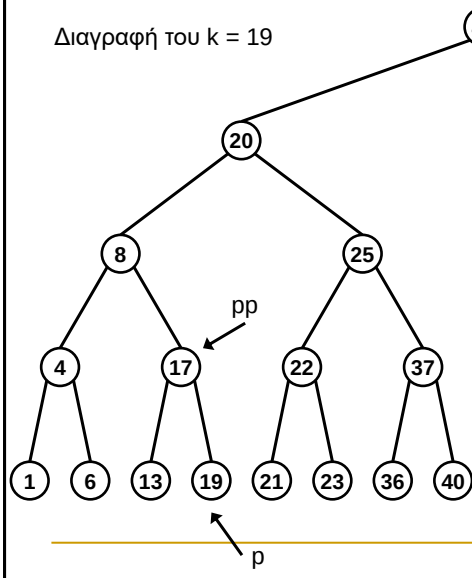
Διαγραφή φύλλου

Διαγραφή του k = 19



Διαγραφή φύλλου

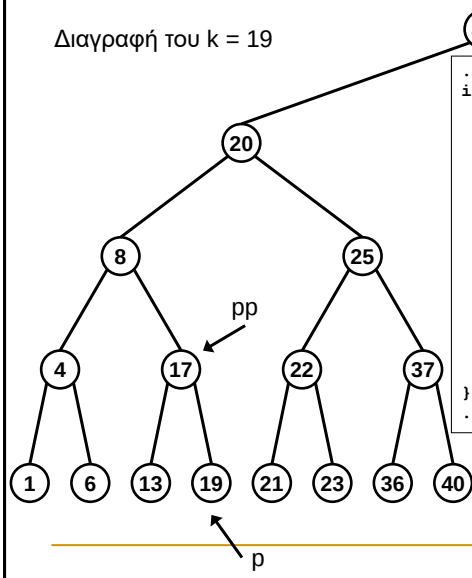
Διαγραφή του k = 19




```
p = root;
pp = 0;

// move to a child of p
while (p && p->data != k){
    pp = p;
    if (k < p->data) p = p->LeftChild;
    else p = p->RightChild;
}

...
```

Διαγραφή φύλλου

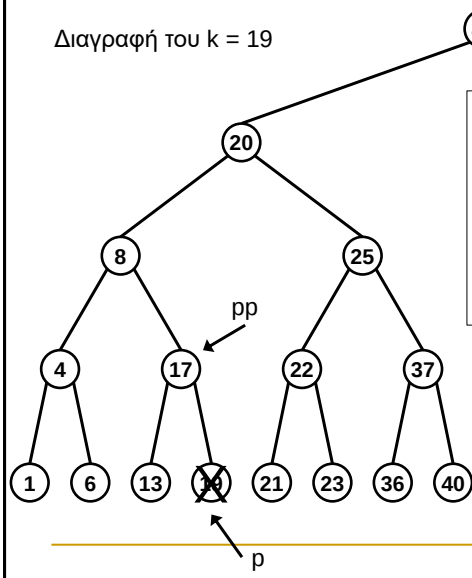
Διαγραφή του k = 19




```
...
if (p->LeftChild && p->RightChild) {
  // convert to zero or one child case
  // find largest element in
  // left subtree of p
  BinaryTreeNode<E> *s = p->LeftChild,
                    *ps = p;  // parent of s
  while (s->RightChild) {
    ps = s;
    s = s->RightChild;
  }

  // move largest from s to p
  p->data = s->data;
  p = s;
  pp = ps;
}
...
```


Διαγραφή φύλλου

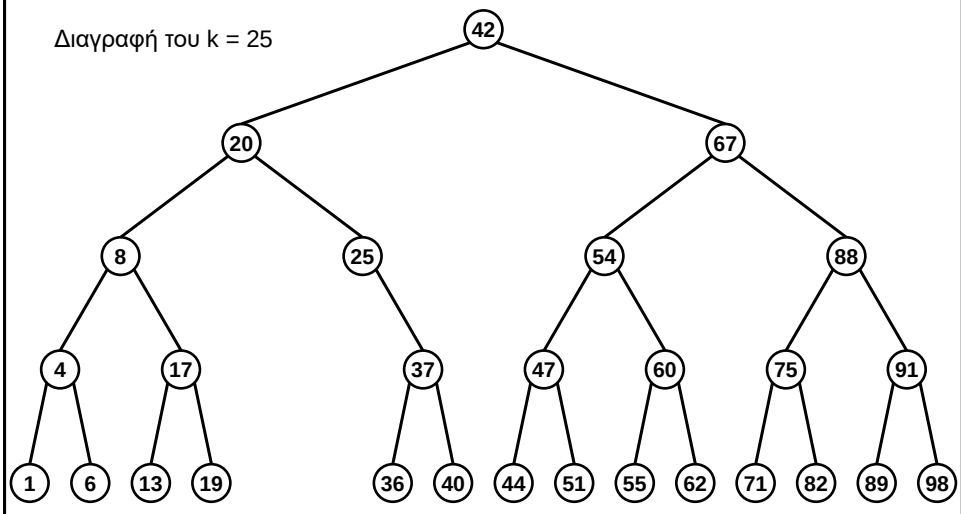
Διαγραφή του k = 19




```
BinaryTreeNode<E> *c;
if (p->LeftChild) c = p->LeftChild;
else c = p->RightChild;

// delete p
if (p == root) root = c;
else {// is p left or right child of pp?
      if (p == pp->LeftChild)
         pp->LeftChild = c;
      else pp->RightChild = c;}
delete p;
```

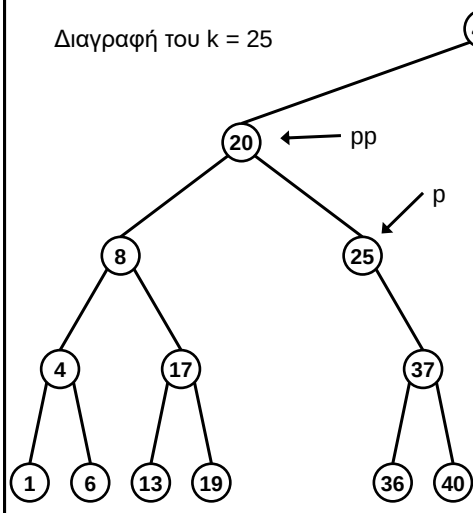
Διαγραφή κόμβου με 1 υποδέντρο

Διαγραφή του k = 25



Διαγραφή κόμβου με 1 υποδέντρο

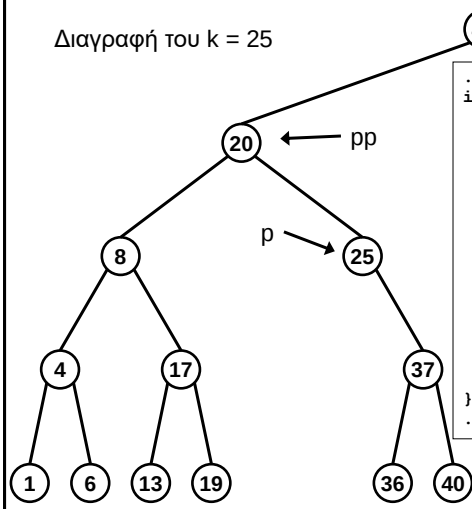
Διαγραφή του k = 25




```
p = root;
pp = 0;

while (p && p->data != k){
      pp = p;
      if (k < p->data) p = p->LeftChild;
      else p = p->RightChild;
}

...
```

Διαγραφή κόμβου με 1 υποδέντρο

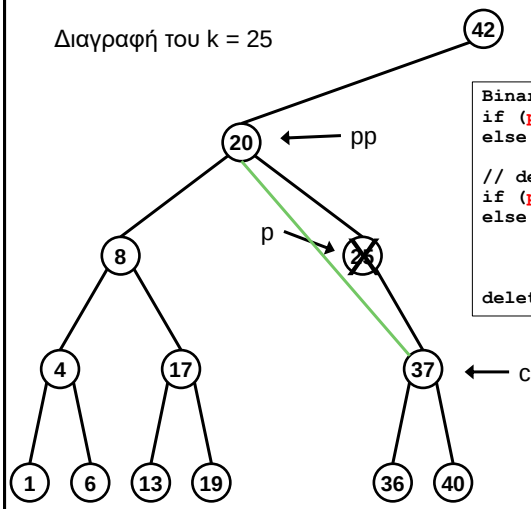
Διαγραφή του k = 25




```
...
if (p->LeftChild && p->RightChild) {
  // convert to zero or one child case
  // find largest element in
  // left subtree of p
  BinaryTreeNode<E> *s = p->LeftChild,
                    *ps = p;  // parent of s
  while (s->RightChild) {
    ps = s;
    s = s->RightChild;
  }

  // move largest from s to p
  p->data = s->data;
  p = s;
  pp = ps;
}
...
```


Διαγραφή κόμβου με 1 υποδέντρο

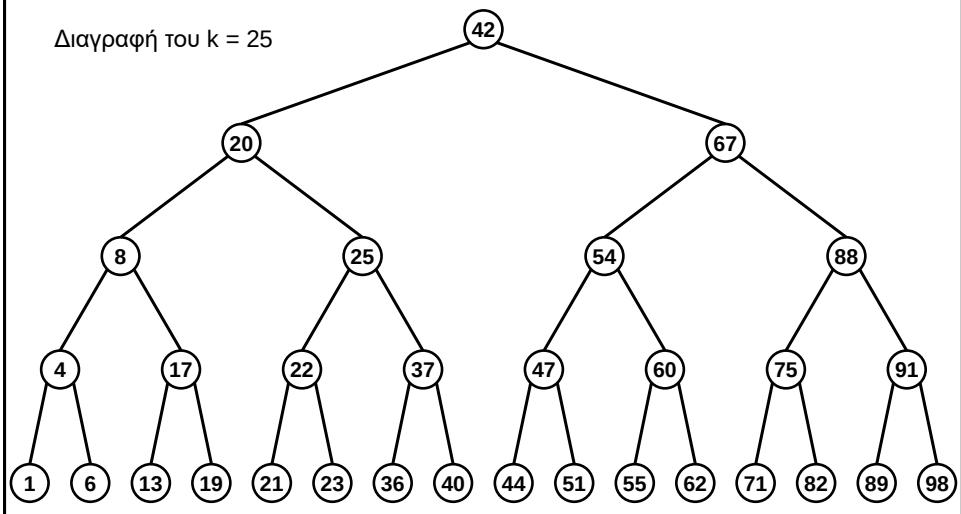
Διαγραφή του k = 25




```
BinaryTreeNode<E> *c;
if (p->LeftChild) c = p->LeftChild;
else c = p->RightChild;

// delete p
if (p == root) root = c;
else {// is p left or right child of pp?
      if (p == pp->LeftChild)
           pp->LeftChild = c;
      else pp->RightChild = c;}
delete p;
```

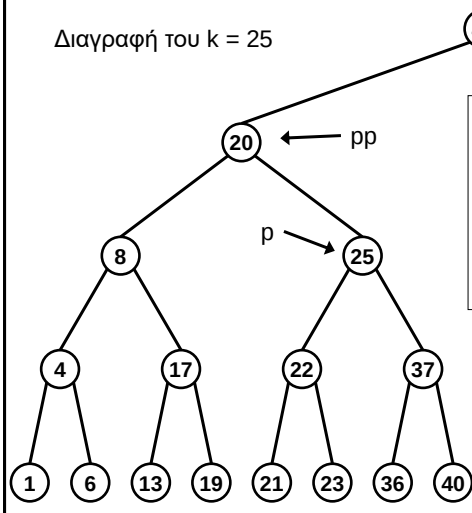
Διαγραφή κόμβου με 2 υποδέντρα

Διαγραφή του k = 25



Διαγραφή κόμβου με 2 υποδέντρα

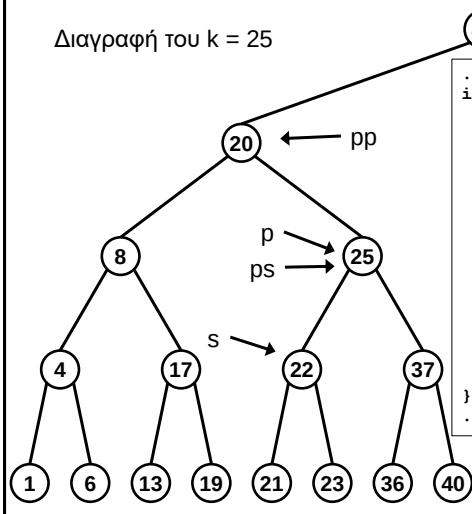
Διαγραφή του k = 25




```
p = root;
pp = 0;

while (p && p->data != k){
      pp = p;
      if (k < p->data) p = p->LeftChild;
      else p = p->RightChild;
}

...
```

Διαγραφή κόμβου με 2 υποδέντρα

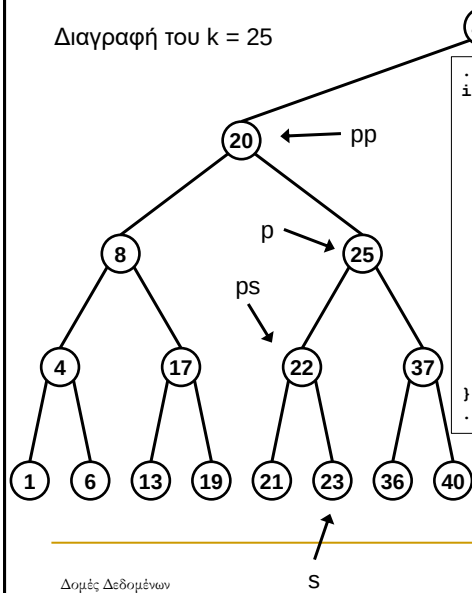
Διαγραφή του k = 25




```
...
if (p->LeftChild && p->RightChild) {
  // convert to zero or one child case
  // find largest element in
  // left subtree of p
  BinaryTreeNode<E> *s = p->LeftChild,
                    *ps = p;  // parent of s
  while (s->RightChild) {
    ps = s;
    s = s->RightChild;
  }

  // move largest from s to p
  p->data = s->data;
  p = s;
  pp = ps;
}
...
```


Διαγραφή κόμβου με 2 υποδέντρα

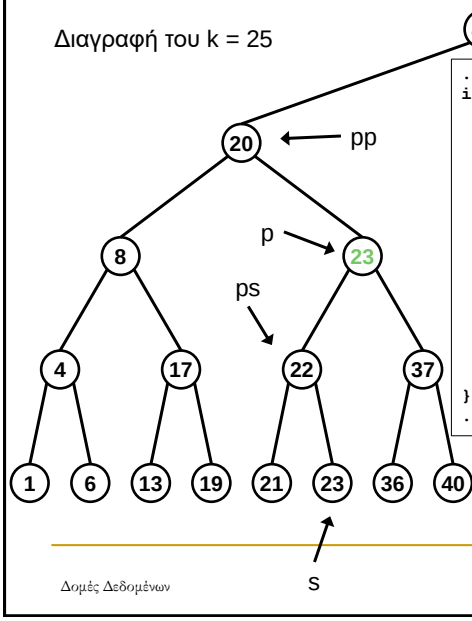
Διαγραφή του k = 25




```
...
if (p->LeftChild && p->RightChild) {
  // convert to zero or one child case
  // find largest element in
  // left subtree of p
  BinaryTreeNode<E> *s = p->LeftChild,
                    *ps = p;  // parent of s
  while (s->RightChild) {
    ps = s;
    s = s->RightChild;
  }

  // move largest from s to p
  p->data = s->data;
  p = s;
  pp = ps;
}
...
```

Διαγραφή κόμβου με 2 υποδέντρα

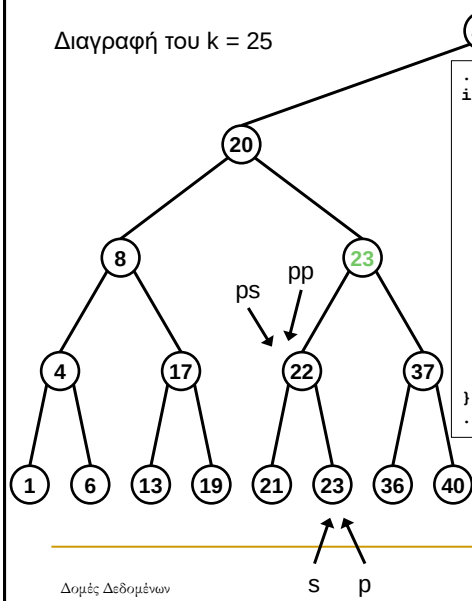
Διαγραφή του k = 25




```
...
if (p->LeftChild && p->RightChild) {
  // convert to zero or one child case
  // find largest element in
  // left subtree of p
  BinaryTreeNode<E> *s = p->LeftChild,
                     *ps = p;  // parent of s
  while (s->RightChild) {
    ps = s;
    s = s->RightChild;
  }

  // move largest from s to p
  p->data = s->data;
  p = s;
  pp = ps;
}
...
```


Διαγραφή κόμβου με 2 υποδέντρα

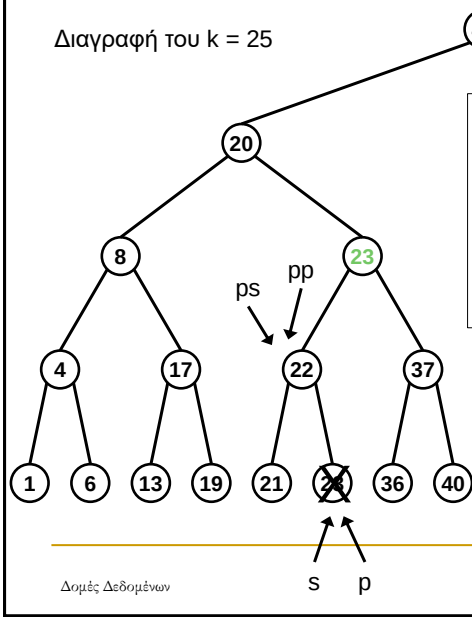
Διαγραφή του k = 25




```
...
if (p->LeftChild && p->RightChild) {
  // convert to zero or one child case
  // find largest element in
  // left subtree of p
  BinaryTreeNode<E> *s = p->LeftChild,
                     *ps = p;  // parent of s
  while (s->RightChild) {
    ps = s;
    s = s->RightChild;
  }

  // move largest from s to p
  p->data = s->data;
  p = s;
  pp = ps;
}
...
```

Διαγραφή κόμβου με 2 υποδέντρα

Διαγραφή του k = 25




```
BinaryTreeNode<E> *c;
if (p->LeftChild) c = p->LeftChild;
else c = p->RightChild;

// delete p
if (p == root) root = c;
else {// is p left or right child of pp?
      if (p == pp->LeftChild)
           pp->LeftChild = c;
      else pp->RightChild = c;}
delete p;
```


Ύψος ΔΔΑ

- Το ύψος ενός ΔΔΑ με n στοιχεία μπορεί να φτάσει μέχρι και n.
- Στη γενική περίπτωση όμως (όταν οι εισαγωγές και οι διαγραφές γίνονται τυχαία), το ύψος είναι O(log*n*) κατά μ.ό.