Οι διαφάνειες που ακολουθούν είναι προσαρμογή των διαφανειών από τον ιστότοπο https://www.cise.ufl.edu/~sahni/cop3530/ (σελίδα του καθ. Sartaj Sahni για το μάθήμα των Δομών Δεδομένων)

Το πρόβλημα της ένωσης-εύρεσης ανεξάρτητων συνόλων

- Δίνεται ένα σύνολο {1, 2, …, n} n στοιχείων.
- Αρχικά, κάθε στοιχείο ανήκει σε ξεχωριστό σύνολο.
  - {1}, {2}, …, {n}
- Εκτελείται μία ακολουθία λειτουργιών ένωσης και εύρεσης.
- Μία λειτουργία ένωσης συνδυάζει δύο σύνολα σε ένα.
  - Σε κάθε χρονική στιγμή, κάθε στοιχείο ανήκει σε ένα μόνο σύνολο.
- Μία λειτουργία εύρεσης εντοπίζει το σύνολο το οποίο περιέχει ένα συγκεκριμένο στοιχείο.

```
void Initialize(int n)
{// Initialize n classes with one element each.
   E = new int [n + 1];
   for (int e = 1; e <= n; e++)
      E[e] = e;
}

void Union(int i, int j)
{// Union the classes i and j.
   for (int k = 1; k <= n; k++)
      if (E[k] == j) E[k] = i;
}

int Find(int e)
{// Find the class that contains element i.
   return E[e];
}
```

Πρόγραμμα 3.46 Συναρτήσεις της online κλάσης ισοδυναμίας με χρήση πινάκων

- Συνολικός χρόνος για αρχικοποίηση, u ενώσεις και f αναζητήσεις:

$$\Theta(n+u*n+f)=\Theta(u*n+f)$$

ΔΕΥΤΕΡΗ ΛΥΣΗ

```
void Initialize(int n)
{// Initialize n classes with one element each.
   node = new EquivNode [n + 1];

   for (int e = 1; e <= n; e++) {
      node[e].E = e;
      node[e].link = 0;
      node[e].size = 1;
      }
}
```

```
void Union(int i, int j)
{// Union the classes i and j.

   // make i smaller class
   if (node[i].size > node[j].size)
      Swap(i,j);

   //  change E values of smaller class
   int k;
   for (k = i; node[k].link; k = node[k].link)
      node[k].E = j;
   node[k].E = j; // last node in chain

   // insert chain i after first node in chain j
   // and update new chain size
   node[j].size += node[i].size;
   node[k].link = node[j].link;
   node[j].link = i;
}

int Find(int e)
{// Find the class that contains element i.
   return node[e].E;
}
```

Πρόγραμμα 3.47 Συναρτήσεις της online κλάσης ισοδυναμίας με χρήση αλυσίδων

Λύση βασισμένη με πίνακες και αλυσίδες

Λήμμα: Εκκίνηση n κλάσεων 1 στοιχείου και u ενώσεις.

α) Καμιά κλάση δεν έχει περισσότερα από u+1 στοιχεία

β) Παραμένουν τουλάχιστον n-2u κλάσεις με ένα στοιχείο

γ) Ισχύει $u<n$

Απόδειξη:

α) Παρατήρηση: αν έχουν εκτελεστεί u ενώσεις, για τη δημιουργία μίας κλάσης Α χρειάστηκαν m ενώσεις όπου $m \leq u$. Θα αποδείξουμε ότι το πλήθος των στοιχείων της κλάσης Α είναι το πολύ m+1 και επομένως u+1.

Απόδειξη με επαγωγή ως προς το πλήθος m.

Αν m=1 τότε η κλάση Α προήλθε από δύο μονοσύνολα και επομένως το πλήθος των στοιχείων της Α είναι 2. Άρα ισχύει η πρόταση για m=1

Αν υπόθεση της επαγωγής ίσχύει για όλα τα $m \leq k$ θα αποδείξουμε ότι ισχύει και για m=k+1

Στην k+1-στή ένωση ενώθηκαν δύο κλάσεις Β και Γ για να δημιουργήσουν την Α. Έστω k_B, k_Γ το πλήθος των ενώσεων για τη δημιουργία των δέντρων Β και Γ αντίστοιχα.

Λύση βασισμένη με πίνακες και αλυσίδες

- Ισχύει ότι $k_B+k_\Gamma=k$. Επίσης από την υπόθεση της επαγωγής ισχύει ότι το πλήθος των στοιχείων των κλάσεων Β και Γ είναι το πολύ k_B+1 και $k_\Gamma+1$ αντίστοιχα. Επομένως το πλήθος των στοιχείων της κλάσης Α θα είναι το πολύ $k_B+k_\Gamma+2$ δηλ. m+1.
- Άρα αποδείχθηκε.

Λύση βασισμένη με πίνακες και αλυσίδες

β) Αν ξεκινήσουμε με μονοσύνολα, και ζευγαρώνουμε αν δύο τα μονοσύνολα, μετά από u ενώσεις, 2u στοιχεία θα ανήκουν στα νέα σύνολα που δημιουργήθηκαν και n-2u στοιχεία θα εξακολουθούν να είναι σε μονοσύνολα. Ο τρόπος αυτός ένωσης των συνόλων είναι και αυτός που μειώνει στο μέγιστο βαθμό τα μονοσύνολα. Άρα μετά από u οποιεσδήποτε ενώσεις συνόλων παραμένουν τουλάχιστον n-2u μονοσύνολα.

Λύση βασισμένη με πίνακες και αλυσίδες

γ) Παρατηρήστε ότι ξεκινούμε από n μονοσύνολα και μετά από κάθε ένωση το πλήθος των συνόλων μειώνονται κατά ένα αφού αφαιρούνται δύο σύνολα και προστίθεται ένα, η ένωσή τους. Άρα μετά από n-1 ενώσεις έχει απομείνει ένα μόνο σύνολο που περιέχει και τα n στοιχεία. Επομένως δεν υπάρχει λόγος να εκτελεστούν επιπλέον ενώσεις. Άρα u<n.

Λύση βασισμένη με πίνακες και αλυσίδες

- Με τη χρήση αλυσίδων και πινάκων, επιτυγχάνουμε πολυπλοκότητα

  $O(n + u \log u + f)$

όπου u και f είναι, αντίστοιχα, το πλήθος των ενώσεων και ευρέσεων που εκτελούνται.

Απόδειξη:

- Ο αλγόριθμος του Union εκτελεί ένα for-loop όπου αλλάζει το πεδίο E όλων των στοιχείων που ανήκουν στο μικρότερο από τα δύο σύνολα που ενώνονται.
- Θεωρούμε ένα στοιχείο w. Αρχικά το στοιχείο αυτό ανήκει στο μονοσύνολο {w}. Στην πρώτη ένωση που συμμετείχε αυτό το σύνολο, το w βρέθηκε μετά την ένωση σε ένα σύνολο διπλάσιου μεγέθους από αυτό που ήταν αρχικά, περιέχει δηλ. τουλάχιστον δύο στοιχεία. Αν το {w} ήταν το μικρότερο από τα δύο σύνολα που ενώθηκαν έγινε αλλαγή του πεδίου E του w.
- Αν το νέο σύνολο στο οποίο συμμετέχει το w ενώθηκε πάλι με κάποιο άλλο το οποίο είναι μεγαλύτερου μεγέθους, πάλι έγινε αλλαγή του πεδίου E του w και το νέο σύνολο μετά την ένωση έχει τουλάχιστον 4 στοιχεία.

Λύση βασισμένη με πίνακες και αλυσίδες

- Υποθέστε τώρα ότι τα σύνολα στα οποία συμμετέχει το w ενώθηκαν k φορές με μεγαλύτερα σύνολα. Τότε μετά από αυτές τις ενώσεις, το τρέχον σύνολο στο οποίο συμμετέχει το w έχει τουλάχιστον 2^k στοιχεία. Επίσης συνολικά έχει αλλάξει k φορές το πεδίο E του w.
- Όμως μετά από τις u ενώσεις που γίνονται συνολικά, από την περίπτωση α) του προηγούμενου λήμματος, δεν μπορεί να υπάρχουν περισσότερα από u+1 στοιχεία σε οποιοδήποτε σύνολο. Άρα θα πρέπει $2^k \leq u+1$ δηλ. $k \leq log_2(u+1)$.
- Από την περίπτωση β) του προηγούμενου λήμματος γνωρίζουμε ότι το πολύ 2u στοιχεία συμμετέχουν συνολικά σε σύνολα τα οποία δεν είναι μονοσύνολα. Κάποια από αυτά τα στοιχεία ανήκαν στο μικρότερο από τα δύο σύνολα όταν έγινε κάποια ένωση αλλά δεν μπορεί αυτό να συνέβη περισσότερο από $log_2(u+1)$. Κάθε φορά που συνέβη, έγινε ενημέρωση του πεδίου E του στοιχείου και αφού συνολικά αυτά τα στοιχεία είναι το πολύ 2u, σωρευτικά το πλήθος των ενημερώσεων που έγιναν στην ακολουθία των u ενώσεων είναι το πολύ $2ulog_2(u+1)$.

Λύση βασισμένη με πίνακες και αλυσίδες

- Στο αλγόριθμο Union μετά το κεντρικό for-loop, εκτελούνται επίσης και 3 εντολές, επομένως το σωρευτικό κόστος για u ενώσεις είναι 3u.
- Κάθε find λειτουργία κοστίζει O(1), άρα το συνολικό κόστος για f λειτουργίες find είναι O(f).
- Τέλος το κόστος αρχικοποίησης είναι O(n).
- Άρα συνολικό κόστος για u λειτουργίες ένωσης και f λειτουργίες σωρού είναι O(n+ulogu+f).

Λύση βασισμένη με πίνακες και αλυσίδες

Αποδείξαμε ότι με τη χρήση αλυσίδων και πινάκων, επιτυγχάνουμε πολυπλοκότητα

$$O(n + u \log u + f)$$

όπου u και f είναι, αντίστοιχα, το πλήθος των ενώσεων και ευρέσεων που εκτελούνται.

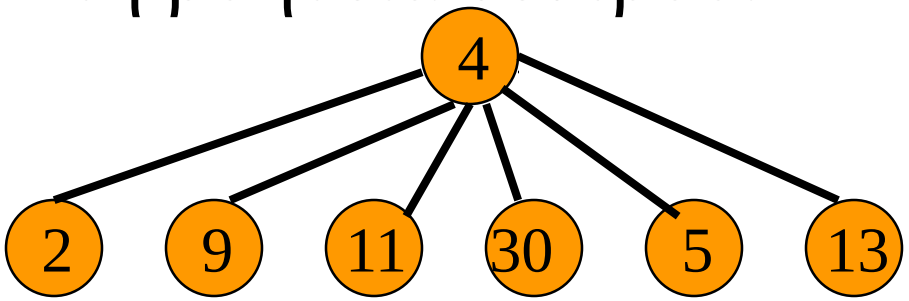
- Χρησιμοποιώντας, ένα δέντρο (όχι δυαδικό δέντρο) για την αναπαράσταση ενός συνόλου, η χρονική πολυπλοκότητα γίνεται σχεδόν

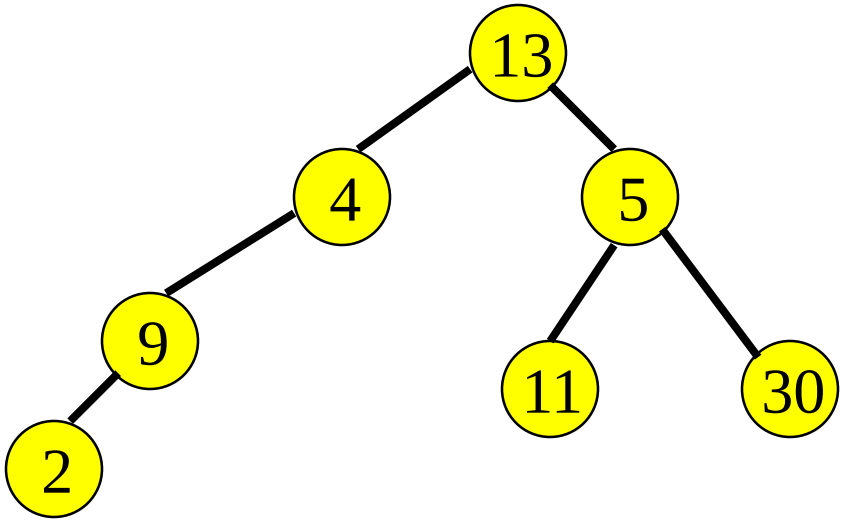
$$O(n + f)$$

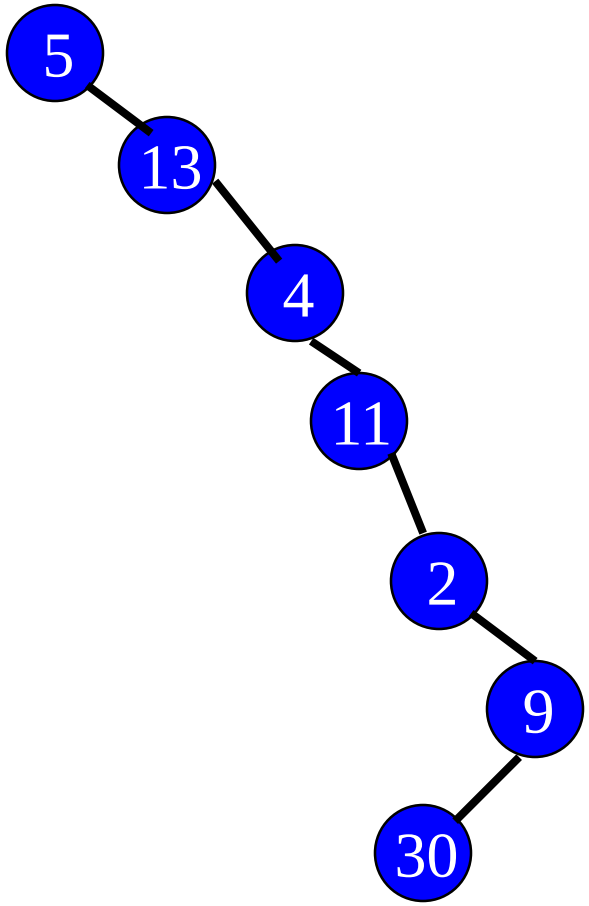
με την προϋπόθεση ότι γίνονται τουλάχιστον n/2 λειτουργίες ένωσης.

Αναπαράσταση ενός συνόλου ως Δέντρου

- S = {2, 4, 5, 9, 11, 13, 30}
- Πιθανές αναπαραστάσεις του συνόλου με τη βοήθεια δέντρου:

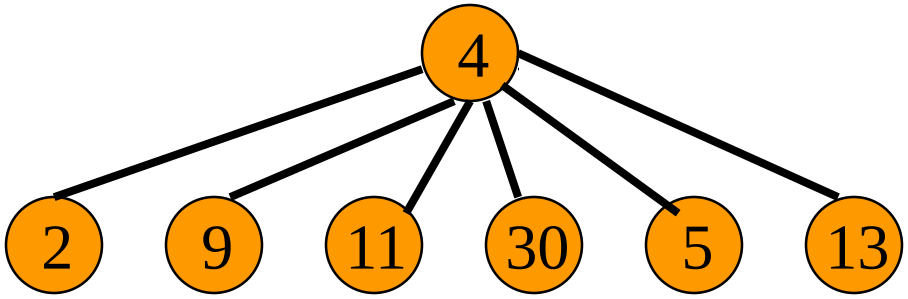






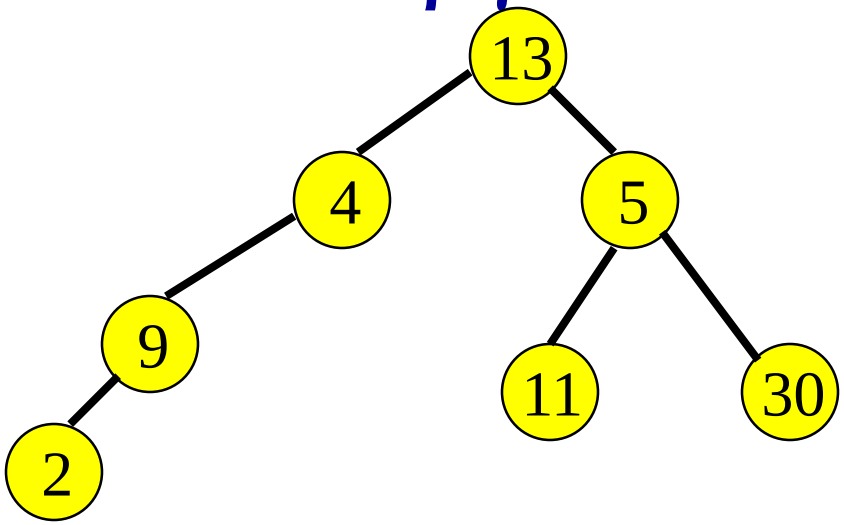
Αποτέλεσμα μία λειτουργίας Εύρεσης

- find(i): εντοπίζει το σύνολο που περιέχει το στοιχείο i.
- Στις περισσότερες εφαρμογές που βασίζονται στο πρόβλημα των ξένων συνόλων:
  - ο χρήστης δεν ενδιαφέρεται για τις συγκεκριμένες τιμές των αναγνωριστικών των συνόλων
  - αρκεί αυτά να είναι διαφορετικά.
- Η απαίτηση είναι οι λειτουργίες find(i) και find(j) να επιστρέφουν την ίδια τιμή ανν τα στοιχεία elements i και j είναι στο ίδιο σύνολο.



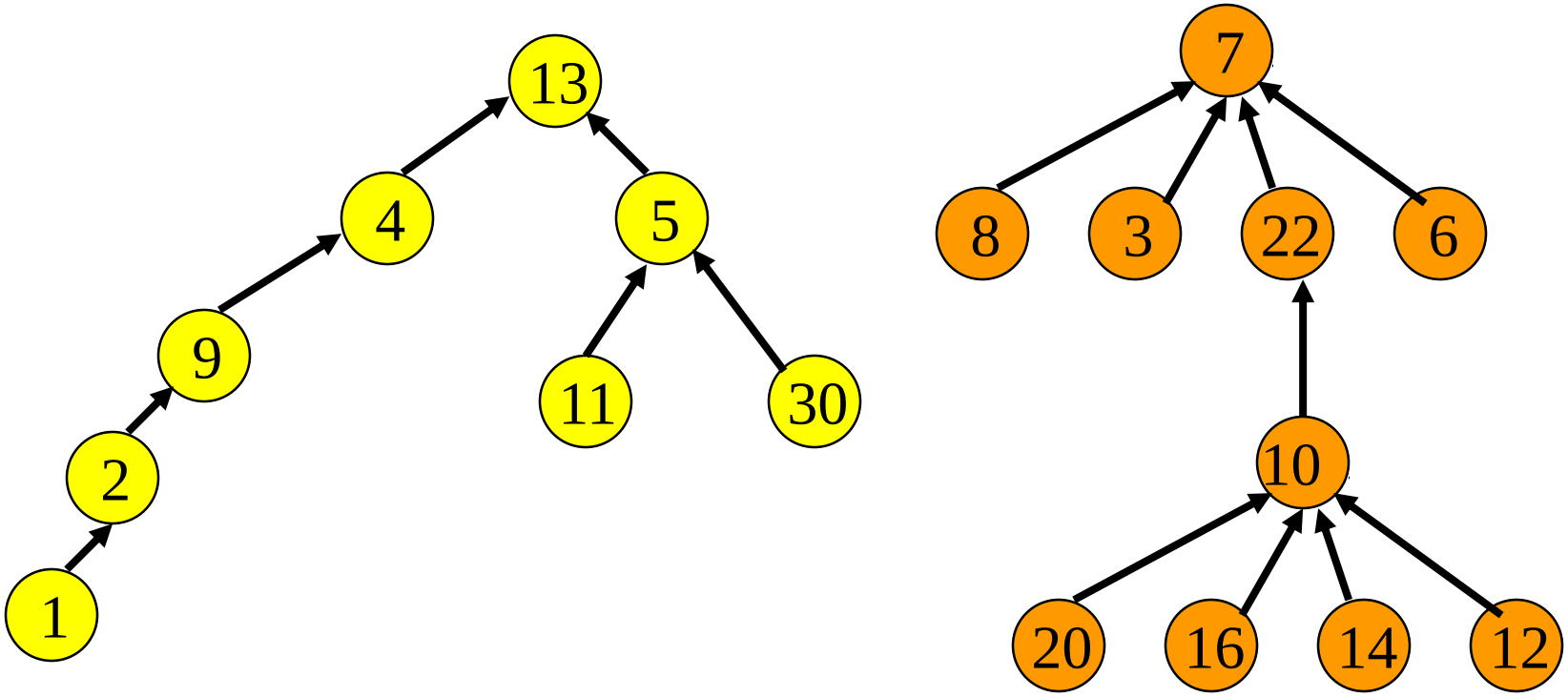
find(i): θα επιστρέψει το στοιχείο που είναι στη ρίζα του δέντρου.

Η βασική μέθοδος για τη λειτουργία find(i)



- Άρχισε από το κόμβο που αναπαριστά το στοιχείο i και ανέβα προς τα πάνω το δέντρο μέχρι να φθάσεις στη ρίζα.
- Επίστρεψε το στοιχείο της ρίζας.
- Για να είναι εφικτή η ανοδική πορεία στο δέντρο, κάθε κόμβος πρέπει να έχει ένα δείκτη προς το γονιό του.

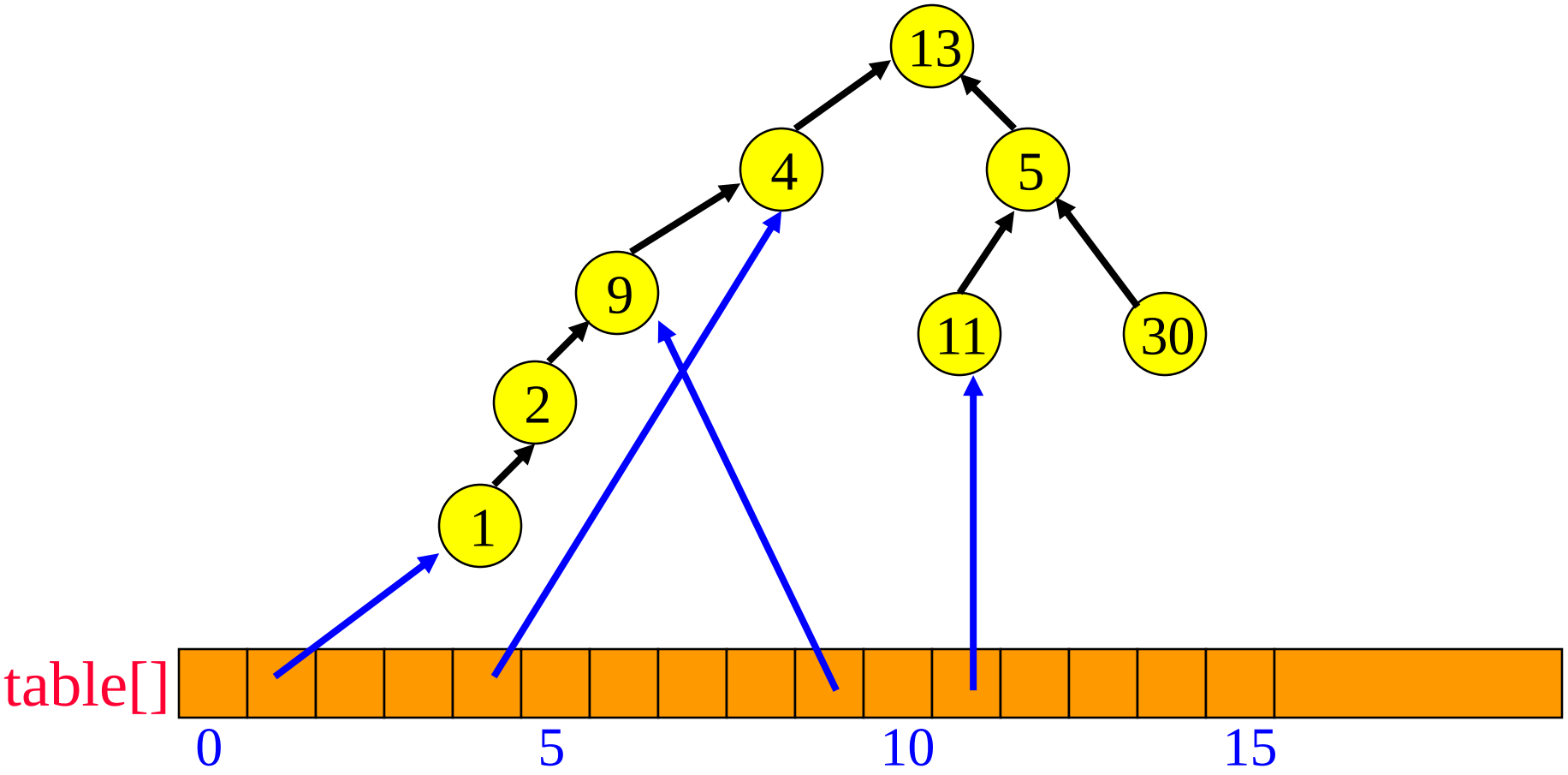
Δέντρα με δείκτες προς το γονέα



Πιθανή δομή κόμβου

- Κόμβοι με δύο πεδία : element και parent.
  - Χρήση ενός πίνακα table[] τέτοιο ώστε table[i] να είναι ο δείκτης στον κόμβο του δέντρου του οποίου το στοιχείο είναι το i.
  - Για να εκτελεστεί τη λειτουργία find(i), αρχίζουμε από τον κόμβο που δείχνει το στοιχείο table[i] και ακολουθούμε τους γονικούς δείκτες μέχρι να φθάσουμε ένα κόμβο του οποίου ο γονικός δείκτης είναι κενός.
  - Επιστρέφεται το στοιχείο που βρίσκεται στη ρίζα.

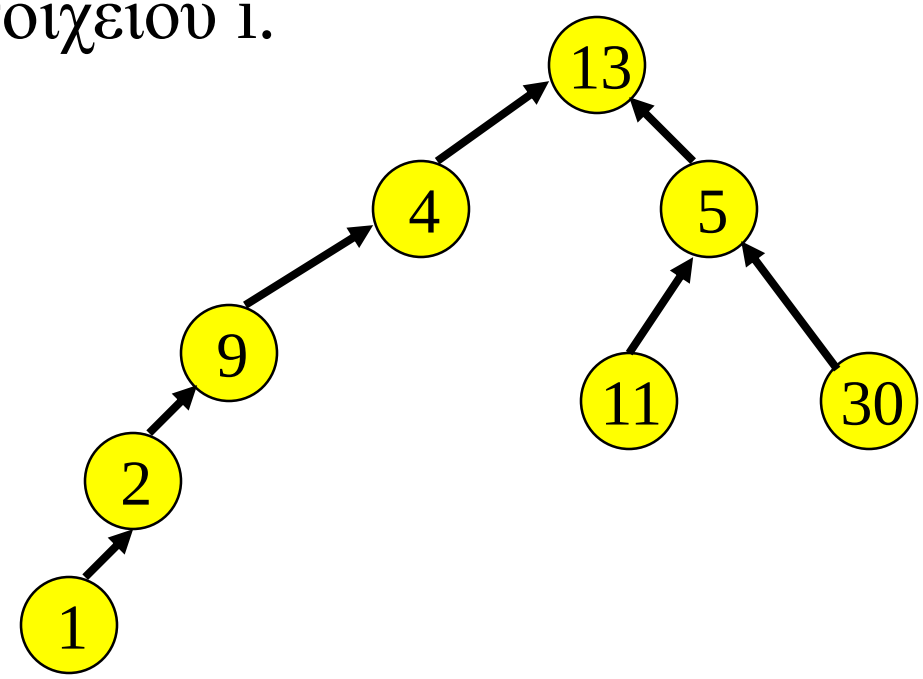
Παράδειγμα

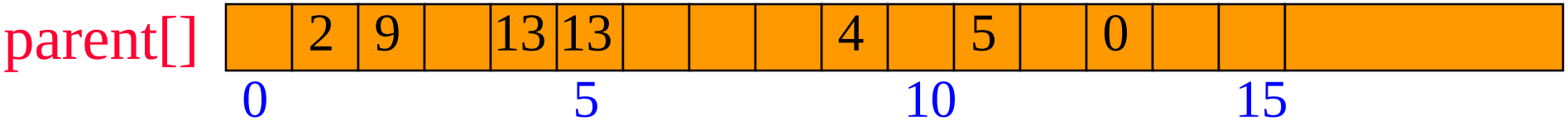


(Μόνο κάποιες θέσεις του πίνακα φαίνονται)

Καλύτερη Αναπαράσταση

- Χρήση ενός πίνακα ακεραίων parent[] τέτοιο ώστε parent[i] είναι το στοιχείο το οποίο είναι γονιός του στοιχείου i.

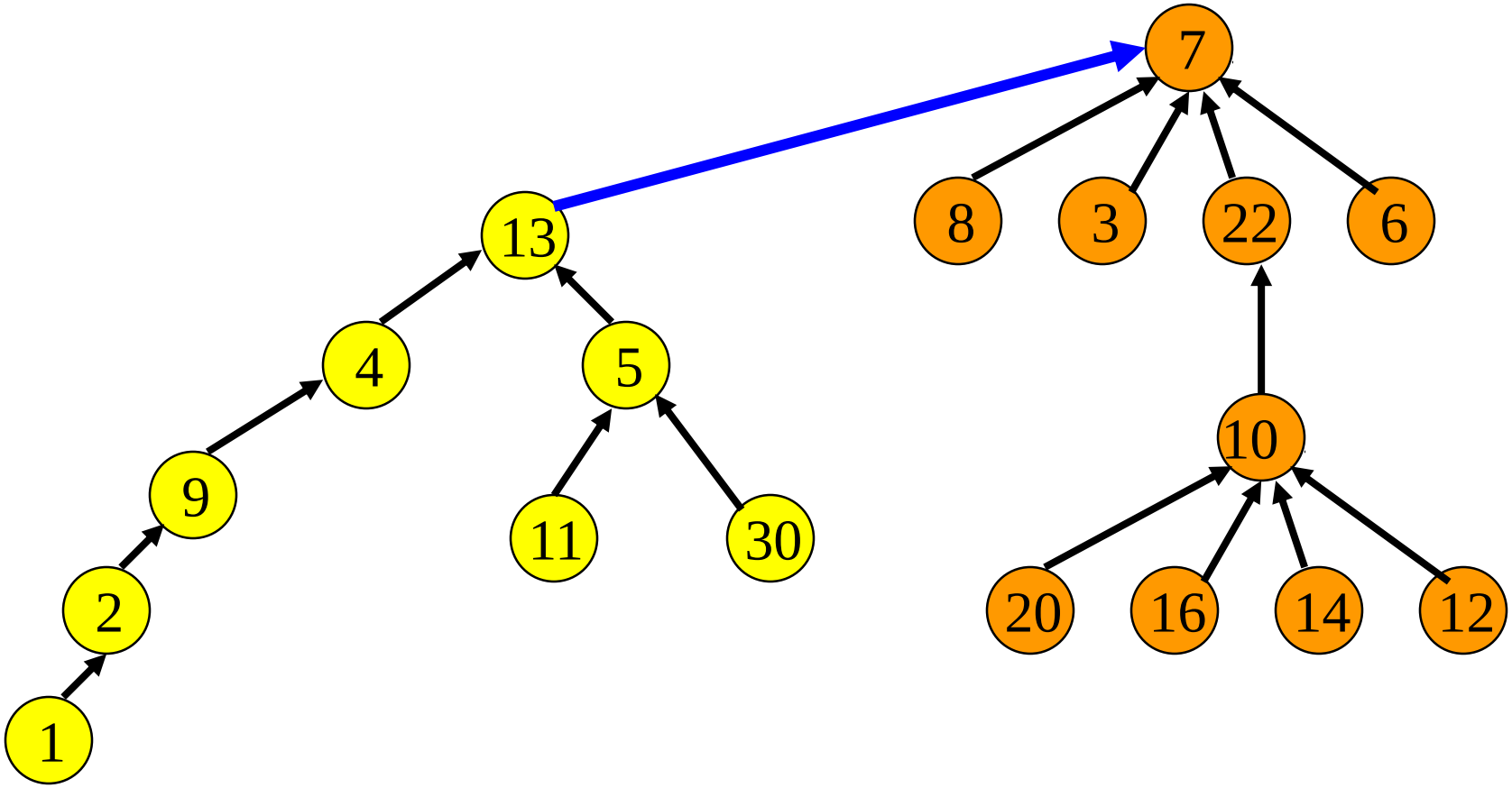




Η Λειτουργία Ένωση

- union(i,j)
  - i και j είναι οι ρίζες δύο διαφορετικών δέντρων, $i \neq j$.
- Για να ενώσεις τα δέντρα, κάνουμε το ένα δέντρο υποδέντρο του άλλου.
  - parent[j] = i

Παράδειγμα Ένωσης



- union(7,13)

Η μέθοδος Εύρεσης

```
public int find(int theElement)
{
    while (parent[theElement] != 0)
       theElement = parent[theElement];  // move up
    return theElement;
}
```

Η Μέθοδος της Ένωσης

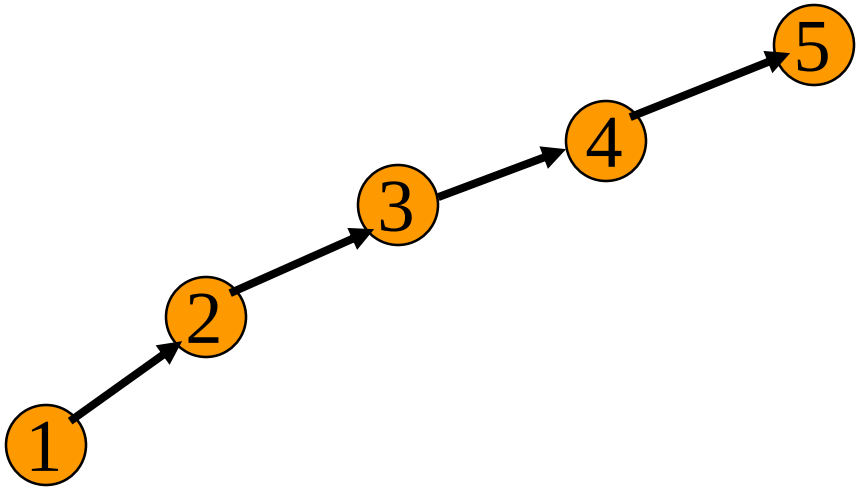
```
public void union(int rootA, int rootB)
    {parent[rootB] = rootA;}
```

Η χρονική πολυπλοκότητα της λειτουργίας της ένωσης

- O(1)

Χρονική Πολυπλοκότητα της λειτουργίας find()

- Το ύψος του δέντρου μπορεί να είναι ίσο με το πλήθος των στοιχείων του δέντρου.
  - union(2,1), union(3,2), union(4,3), union(5,4)…

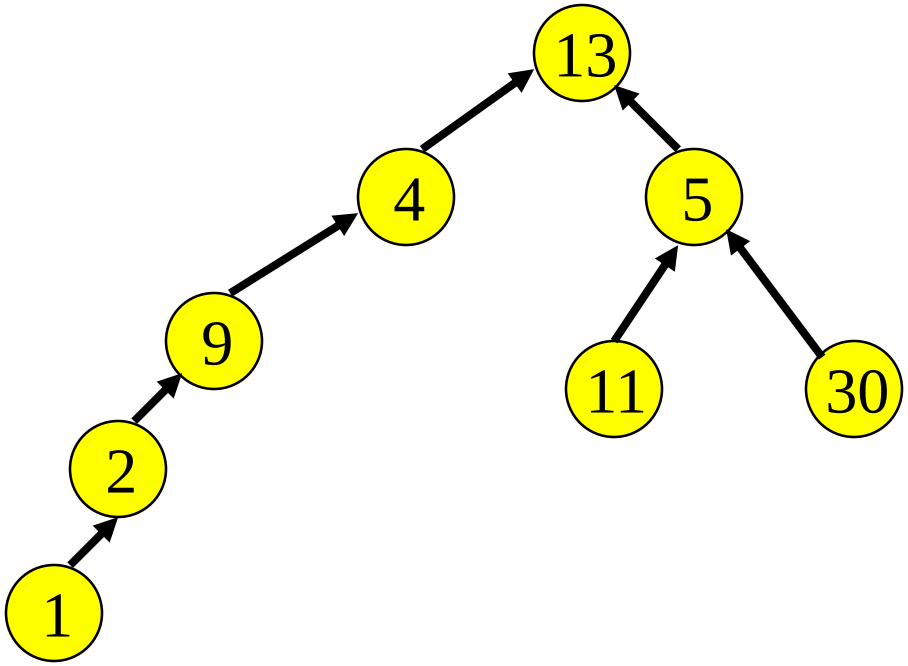


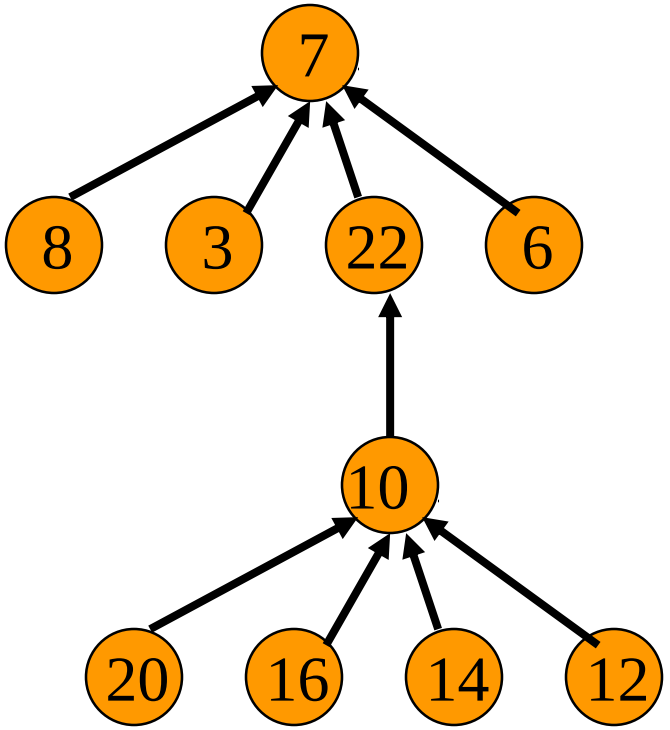
Έτσι η πολυπλοκότητα είναι O(u).

u λειτουργίες ένωσης και f λειτουργίες εύρεσης

- O(u + uf) = O(uf)
- O(n) ο χρόνος για την αρχικοποίηση
  - parent[i] = 0 για όλα τα i.
- Συνολικός χρόνος O(n + uf).
- Χειρότερος από το χρόνο που επιτυγχάνουμε με τη απλή λύση με πίνακες/λίστες.

Έξυπνες τεχνικές Ένωσης

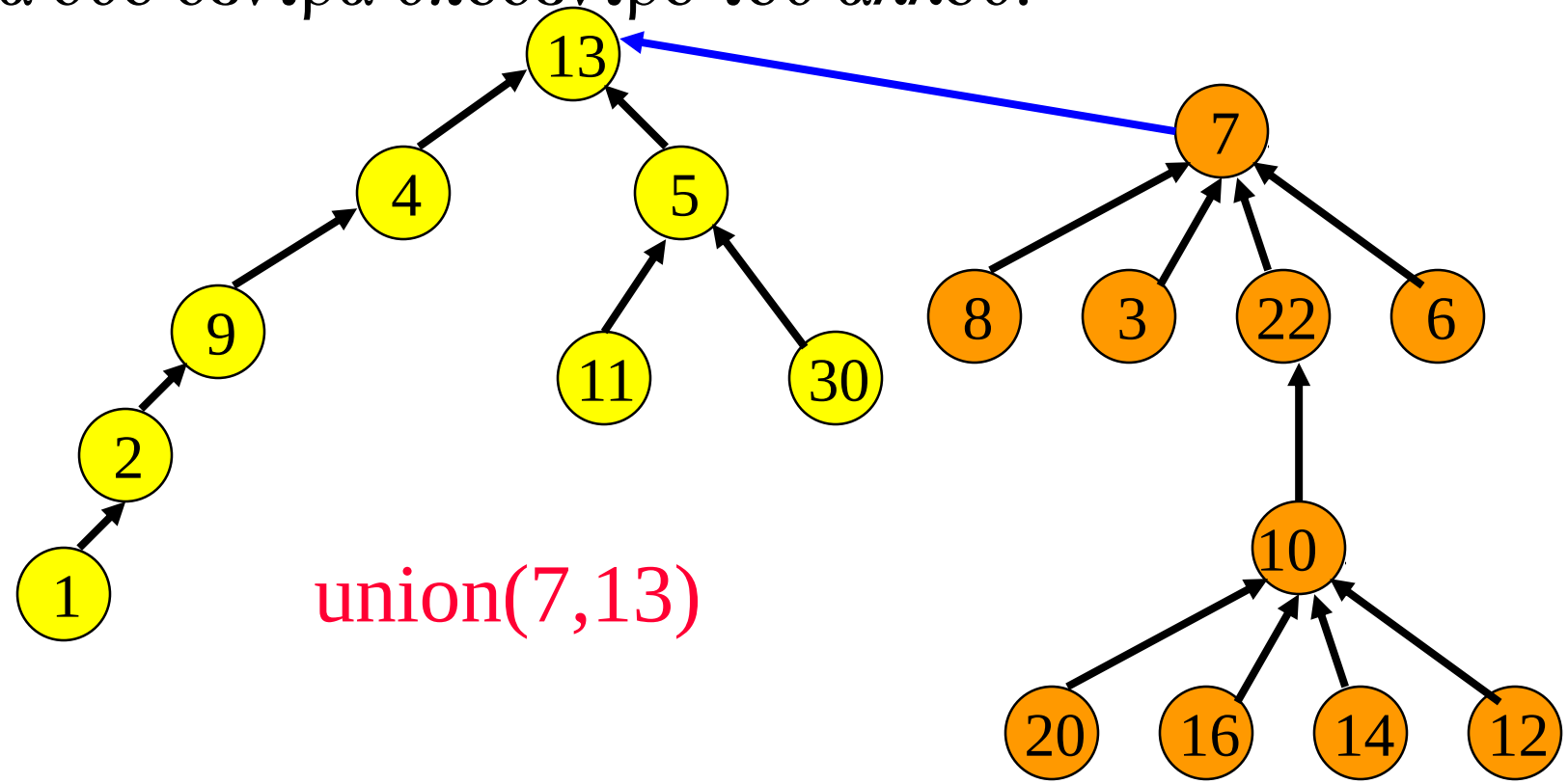




- union(7,13)
- Ποιο δέντρο πρέπει να γίνει υποδέντρο του άλλου;

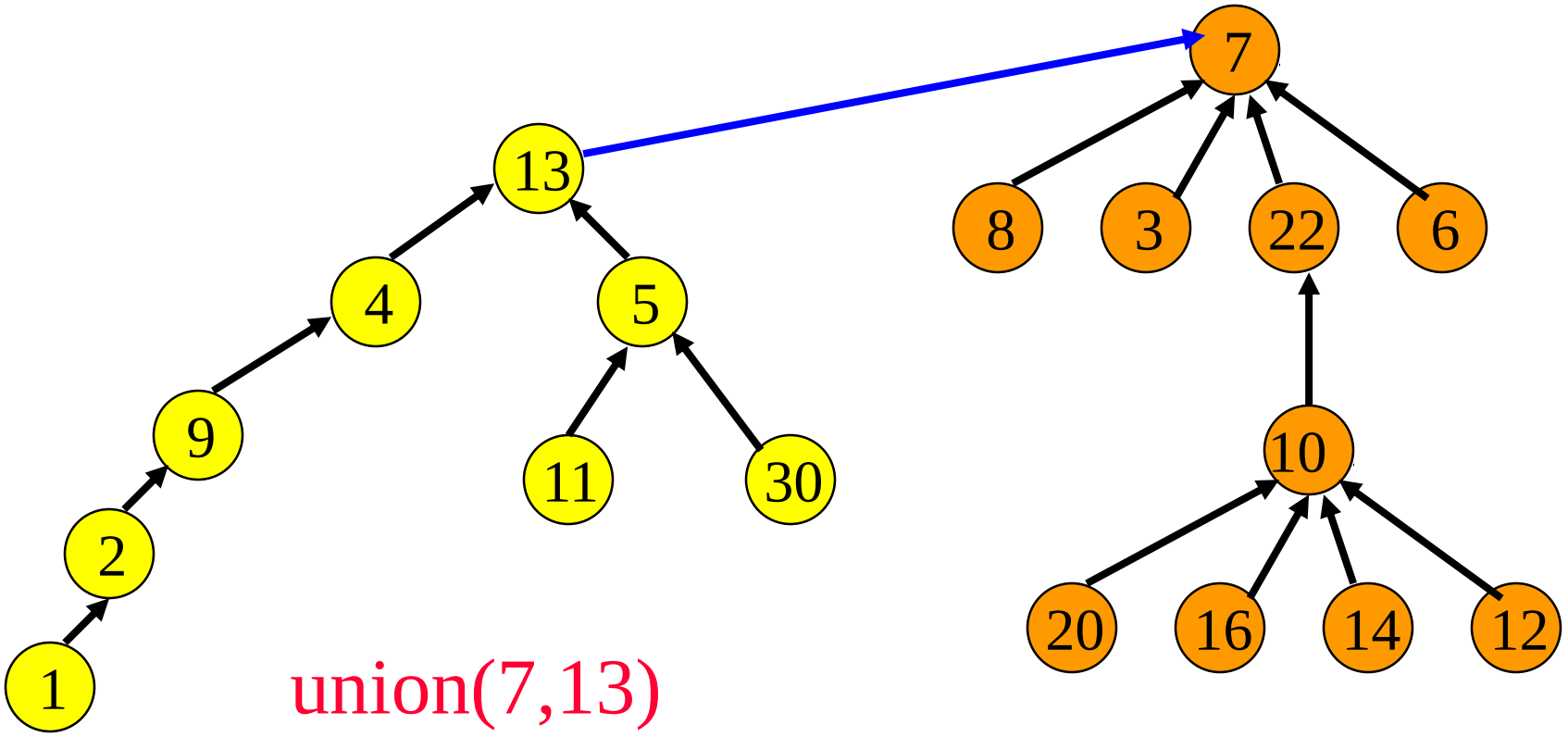
Ο κανόνας του ύψους

- Κάνε το δέντρο με το μικρότερο ύψος, υποδέντρο του άλλου δέντρου.
- Στη περίπτωση ισοϋψών δέντρων κάνε οποιοδήποτε από τα δύο δέντρα υποδέντρο του άλλου.



Ο κανόνας του βάρους

- Κάνε το δέντρο με το μικρότερο πλήθος στοιχείων υποδέντρο του άλλου
- Στη περίπτωση δέντρων με ίσο πλήθος κόμβων κάνε οποιοδήποτε από τα δύο δέντρα υποδέντρο του άλλου.



Ύψος Δέντρου

- Ας υποθέσουμε ότι αρχίζουμε με μονομελή σύνολα και εκτελούμε λειτουργίες ένωσης χρησιμοποιώντας είτε τον κανόνα του ύψους ή το κανόνα του βάρους.
- Το ύψος ενός δέντρου με p στοιχεία είναι το πολύ $\lfloor log_2 p \rfloor + 1$.
- Αποδεικνύεται με επαγωγή ως προς το p.

Απόδειξη

- Αληθές για p=1
- Έστω αληθές για όλα τα δέντρα με i κόμβους όπου $i \leq p-1$
- Έστω t το δέντρο με p κόμβους το οποίο προέκυψε από την union(k,j)

α) εφαρμόστηκε ο κανόνας του βάρους κατά την union(k,j)

- Έστω m ο αριθμός των κόμβων του δέντρου j και p-m είναι ο αριθμός των κόμβων του k
- $1 \leq m \leq p/2$
- Άρα το δέντρο j έγινε υποδέντρο της ρίζας του δέντρου k μετά την ένωση.

Απόδειξη

- Δύο περιπτώσεις:
  - το ύψος του k είναι μεγαλύτερο του ύψους του j:

    Ύψος του t = ύψος του k $\leq \lfloor log_2(p-m) \rfloor + 1 \leq \lfloor log_2 p \rfloor + 1$
  - το ύψος του k είναι μικρότερο ή ίσο του ύψους του j:

    Ύψος του t = ύψος του j $+1 \leq \lfloor log_2 m \rfloor + 2 \leq \lfloor log_2 p/2 \rfloor + 2 = \lfloor log_2 p \rfloor + 1$

Απόδειξη

β) εφαρμόστηκε ο κανόνας του ύψους κατά την union(k,j)

- Ας θεωρήσουμε ότι το δέντρο k έχει ύψος τουλάχιστον όσο το ύψος του δέντρου j.
- Επομένως το δέντρο j έγινε υποδέντρο της ρίζας του δέντρου k μετά την ένωση.
- Έστω m_k, m_j το πλήθος των στοιχείων των δέντρων k και j αντίστοιχα.
- Ισχύει ότι $m_k+m_j = p$ και $\min\{m_k, m_j\} \leq p/2$

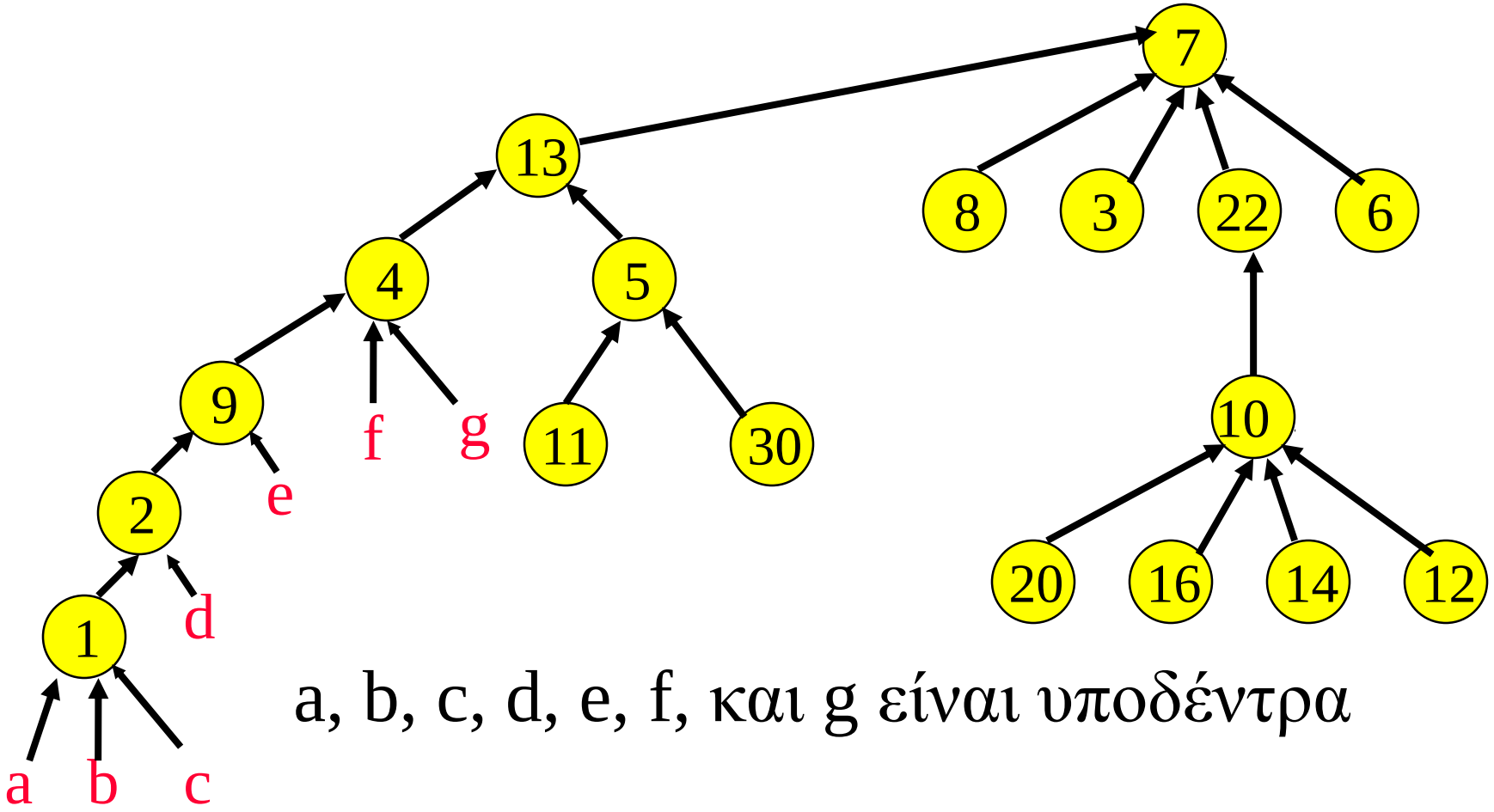
Απόδειξη

- Δύο περιπτώσεις :
  - Το ύψος του δέντρου k είναι μεγαλύτερο από το ύψος του δέντρου j:

    Ύψος του t $\leq \lfloor log_2 m_k \rfloor + 1 \leq \lfloor log_2 p \rfloor + 1$
  - Τα δύο δέντρα έχουν το ίδιο ύψος.

    Ύψος του t = 1 +ύψος του j= 1 +ύψος του k $\leq$ $1+\left\lfloor log_2 \min(m_k, m_j) \right\rfloor + 1 \leq \lfloor log_2 p/2 \rfloor + 2 \leq \lfloor log_2 p \rfloor + 1$

Τροποποίηση της Find

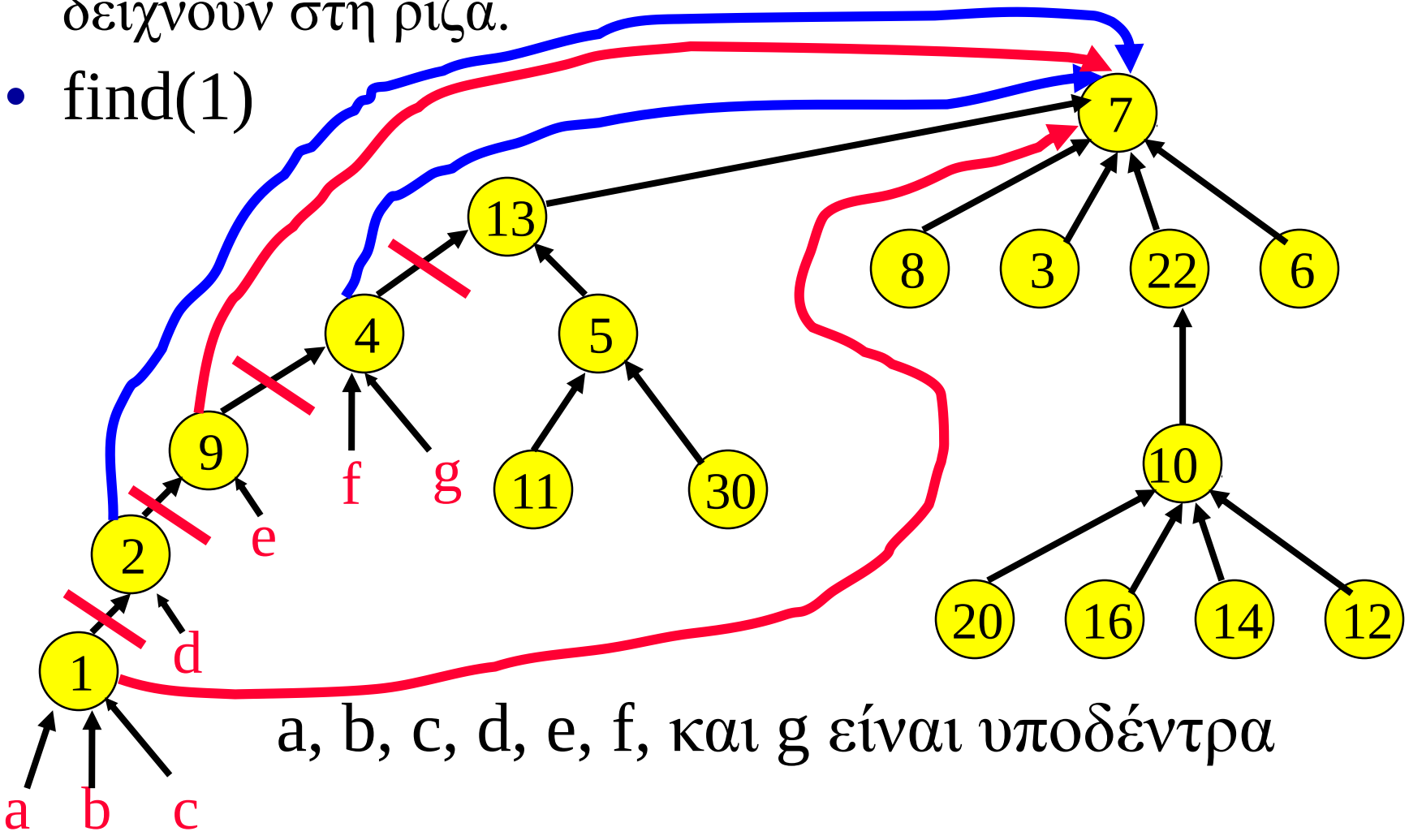


a, b, c, d, e, f, και g είναι υποδέντρα

- Find(1)
- Η ιδέα: κάνε περισσότερη δουλειά τώρα για να επιταχύνεις τις μελλοντικές Find.

Συμπίεση Μονοπατιού

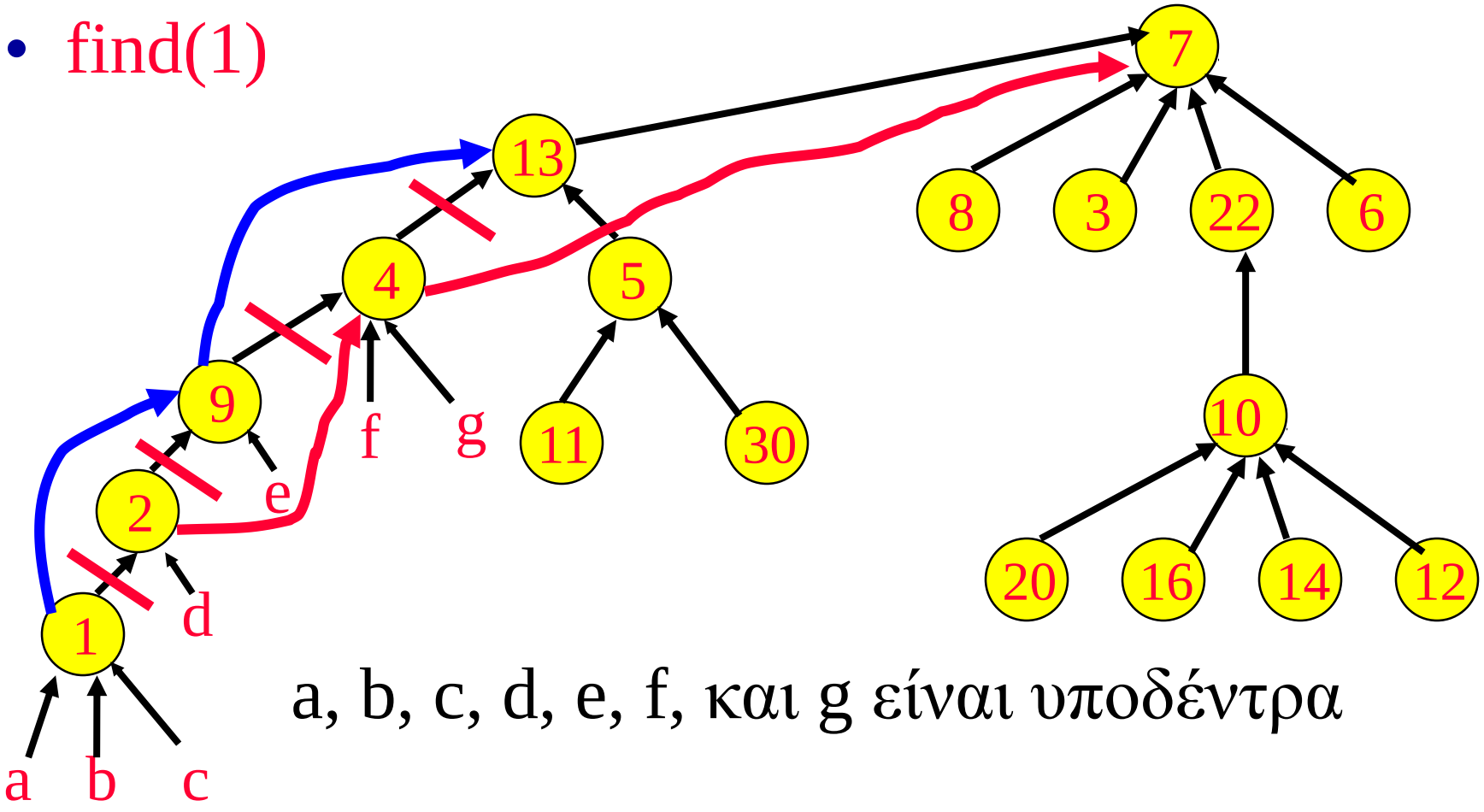
- Κάνε όλους τους κόμβους στο μονοπάτι της εύρεσης να δείχνουν στη ρίζα.
- find(1)



a, b, c, d, e, f, και g είναι υποδέντρα

Διαίρεση μονοπατιού

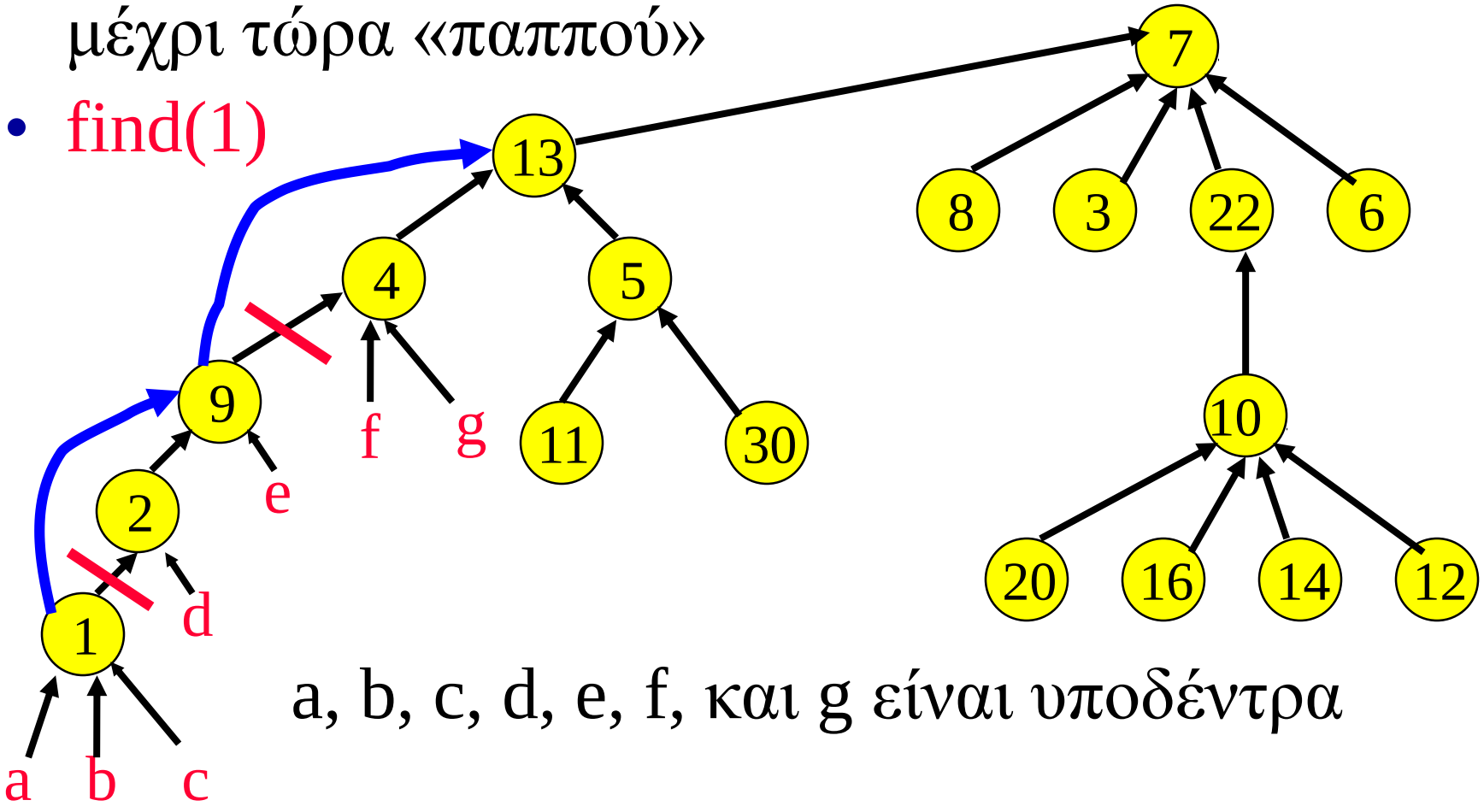
- Οι κόμβοι στο μονοπάτι εύρεσης δείχνουν στο μέχρι τώρα «παππού»
- find(1)



a, b, c, d, e, f, και g είναι υποδέντρα

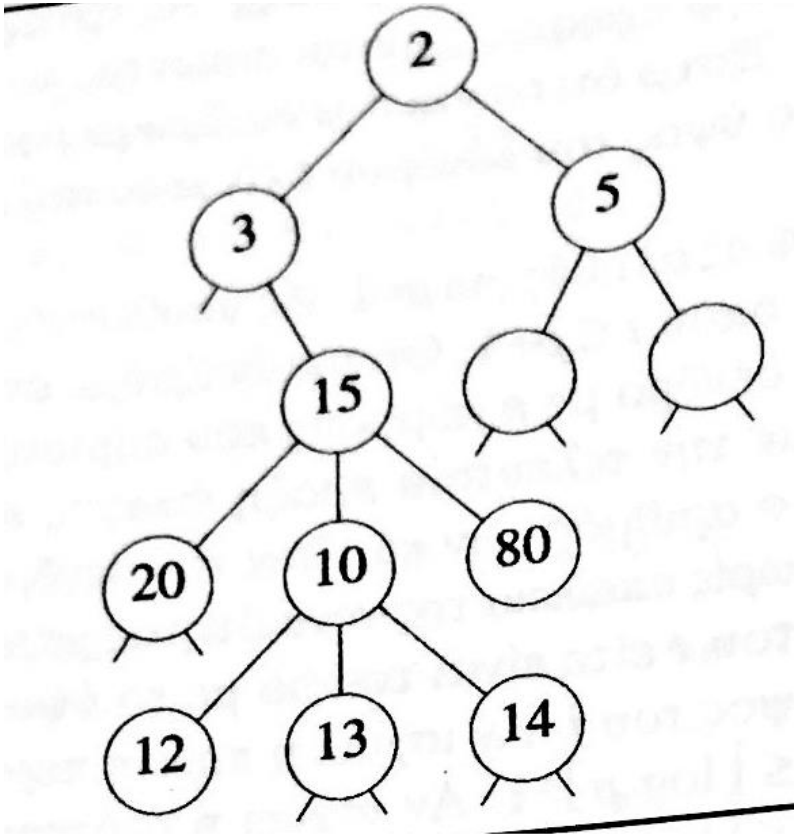
Υποδιπλασιασμός μονοπατιού

- Ο γονικός δείκτης κάθε δεύτερου κόμβου στο μονοπάτι εύρεσης αλλάζει ώστε να δείχνει στο μέχρι τώρα «παππού»
- find(1)

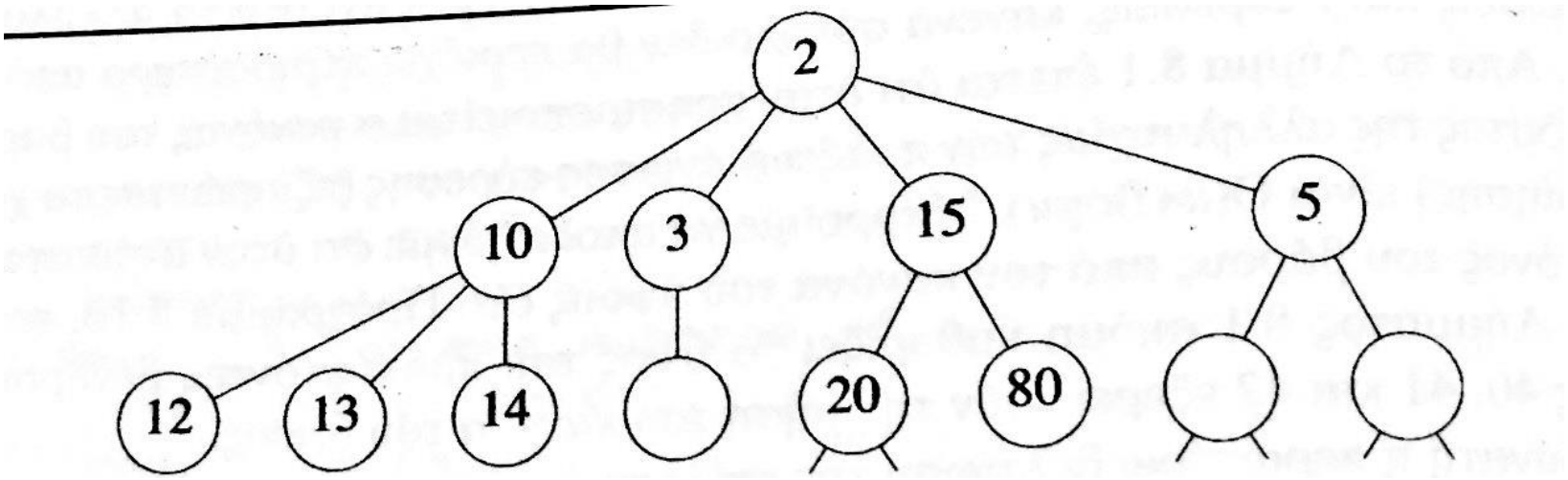


a, b, c, d, e, f, και g είναι υποδέντρα

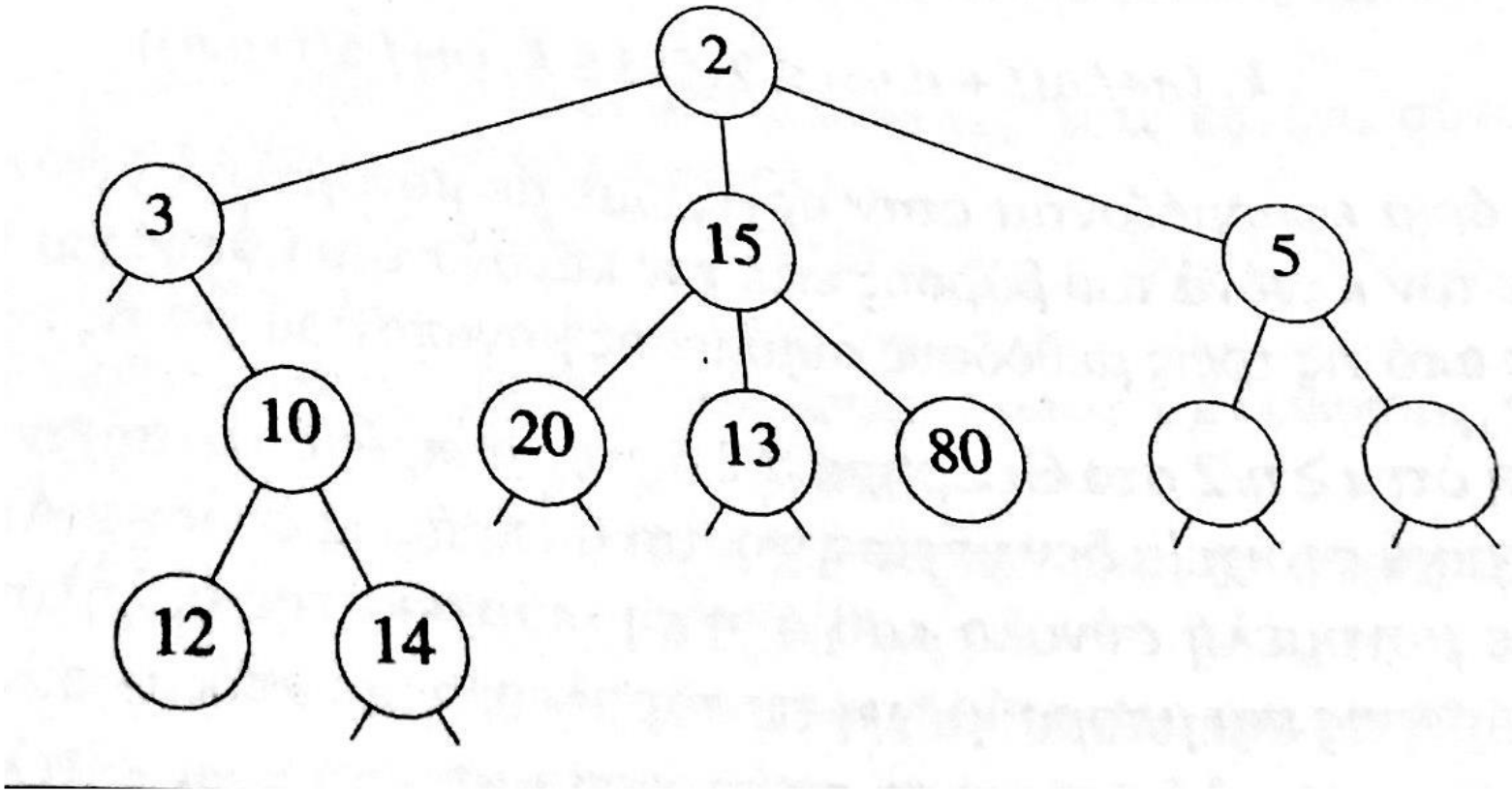
Παράδειγμα Δέντρου



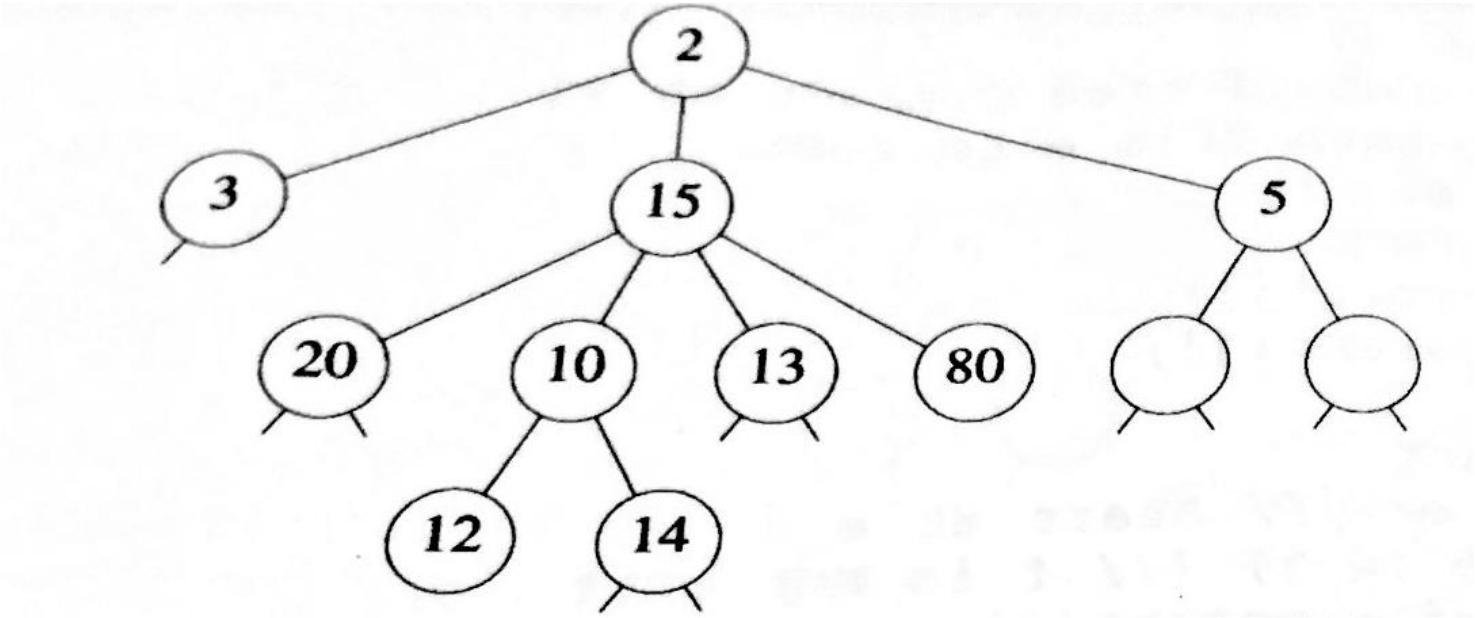
Συμπυξη μονοπατιού (Find(10))



Διάσπαση μονοπατιού (Find(13))



Υποδιπλασιασμός μονοπατιού (Find(13))



```
void Initialize(int n)
{// One element per set/class/tree.
   parent = new int[n+1];
   for (int e = 1; e <= n; e++)
      parent[e] = 0;
}

int Find(int e)
{// Return root of tree containing i.
   while (parent[e])
      e = parent[e];   // move up one level
   return e;
}

void Union(int i, int j)
{// Combine trees with roots i and j.
   parent[j] = i;
}
```

Πρόγραμμα 8.15 Απλή δενδρική λύση στο πρόβλημα της ένωσης-εύρεσης.

```
void Initialize(int n)
{// One element per set/class/tree.
   root = new bool[n+1];
   parent = new int[n+1];
   for (int e = 1; e <= n; e++) {
      parent[e] = 1;
      root[e] = true;}
}

int Find(int e)
{// Return root of tree containing e.
   while (!root[e])
      e = parent[e];  // move up one level
   return e;
}
```

```
void Union(int i, int j)
{// Combine trees with roots i and j.
 // Use weighting rule.
   if (parent[i] < parent[j]) {
      // i becomes subtree of j
      parent[j] += parent[i];
      root[i] = false;
      parent[i] = j; }
   else {// j becomes subtree of i
      parent[i] += parent[j];
      root[j] = false;
      parent[j] = i;}
}
```

Πρόγραμμα 8.16 Ενώνουμε

```
int Find(int e)
{// Return root of tree containing e.
 // Compact path from e to root.
   int j = e;
   // find root
   while (!root[j])
      j = parent[j];

   // compact
   int f = e;  // start at e
   while (f != j) {// f is not root
      int pf = parent[f];
      parent[f] = j;  // move f to level 2
      f = pf;         // f moves to old parent
      }

   return j;
}
```

Πρόγραμμα 8.17 Σύμπτυξη μονοπατιού

Πολυπλοκότητα Χρόνου

- Συνάρτηση Ackermann.
  - $A(i,j) = 2^j$, $i = 1$ and $j >= 1$
  - $A(i,j) = A(i-1,2)$, $i >= 2$ and $j = 1$
  - $A(i,j) = A(i-1,A(i,j-1))$, $i, j >= 2$
- Αντίστροφη συνάρτηση Ackermann.
  - $alpha(p,q) = \min\{z>=1 \mid A(z, floor(p/q)) > \log_2 q\}$, $p >= q >= 1$

Πολυπλοκότητα Χρόνου

- Η συνάρτηση Ackermann αυξάνεται πολύ γρήγορα καθώς τα i και j αυξάνονται.
  - $A(2,4) = 2^{65,536}$
- Η αντίστροφη συνάρτηση αυξάνει πολύ αργά.
  - alpha(p,q) < 5 μέχρι $q = 2^{A(4,1)}$
  - A(4,1) = A(2,16) >>>> A(2,4)
- Στην ανάλυση του προβλήματος ξένων συνόλων, q είναι το πλήθος n των στοιχείων; p = n + f και u >= n/2.
- alpha(p,q) < 5.

Πολυπλοκότητα Χρόνου

Theorem 12.2 [Tarjan and Van Leeuwen]

Έστω T(f,u) είναι ο μέγιστος χρόνος που απαιτείται για να επεξεργαστούμε μία οποιαδήποτε ακολουθία f λειτουργιών find and u λειτουργιών unions. Υποθέτουμε ότι u >= n/2.

a*(n + f*alpha(f+n, n)) <= T(f,u) <= b*(n + f*alpha(f+n, n))

όπου a και b είναι σταθερές.

Αυτά τα όρια ισχύουν όταν ξεκινάμε με μοναδιαία σύνολα και χρησιμοποιούμε είτε το κανόνα του ύψους ή του βάρους για τις λειτουργίες των ενώσεων και κάποια από τις μεθόδους συντόμευσης μονοπατιών για τις λειτουργίες εύρεσης.