Ενότητα 9
Ξένα Σύνολα που υποστηρίζουν τη λειτουργία της Ένωσης (Union-Find)

Ξένα Σύνολα που υποστηρίζουν τη λειτουργία της Ένωσης

Έστω ότι $S_1, \ldots, S_k$ είναι ξένα υποσύνολα ενός συνόλου **U**, δηλαδή ισχύει ότι $S_i \cap S_j = \varnothing$, για κάθε i,j με $i \neq j$ και $S_1 \cup \ldots \cup S_k = \mathbf{U}$.

Λειτουργίες

- *MakeSet(X):* επιστρέφει ένα νέο σύνολο που περιέχει μόνο το στοιχείο X.
- *Union(S,T):* επιστρέφει το σύνολο $S \cup T$, το οποίο αντικαθιστά τα S, T.
- *Find(X):* επιστρέφει το σύνολο S στο οποίο ανήκει το στοιχείο X.

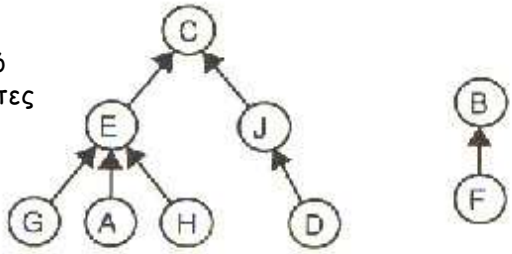
Εφαρμογές

- Το πρόβλημα της διαχειρίσεως ξένων μεταξύ τους συνόλων εμφανίζεται σε αρκετές εφαρμογές.
  - Υπάρχουν αλγόριθμοι ευρέσεως σκελετικών δένδρων (spanning trees) που ξεκινούν από n στοιχειώδη σκελετικά δένδρα με έναν κόμβο το καθένα και βήμα προς βήμα συνενώνουν ξένα μεταξύ τους σκελετικά υποδένδρα μέχρι να προκύψει τελικά ένα δένδρο που αποτελεί το προς αναζήτηση σκελετικό δένδρο.

Ξένα Σύνολα με τη λειτουργία της Ένωσης

Up-Tree

- Είναι δένδρο στο οποίο κάθε κόμβος διατηρεί μόνο ένα δείκτη προς τον πατρικό του κόμβο (έτσι όλοι οι δείκτες δείχνουν προς τα πάνω).
- Ένας κόμβος μπορεί να έχει οποιοδήποτε πλήθος παιδιών.
- Το σύνολο U υλοποιείται ως ένα δάσος από Up-trees.

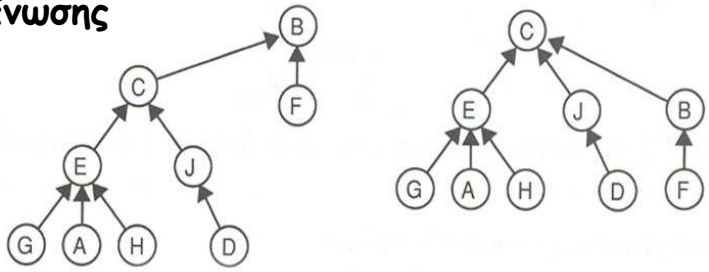


- Κάθε τέτοιο δένδρο περιέχει τα στοιχεία ενός από τα ξένα σύνολα. Το στοιχείο της ρίζας αποτελεί το αναγνωριστικό του συνόλου.

Υλοποίηση Find(X), όπου X είναι ένας δείκτης σε ένα κόμβο
Ακολούθησε τους δείκτες από τον κόμβο προς τη ρίζα.

Ποια είναι η πολυπλοκότητα της Find(); *O*(ύψος δένδρου)

Ξένα Σύνολα που υποστηρίζουν τη λειτουργία της Ένωσης



Έλεγχος αν ένα στοιχείο X ανήκει στο σύνολο S
Ελέγχουμε αν η Find(X) επιστρέφει S.

Υλοποίηση Union(S,T)
Κάνε τη ρίζα του ενός δένδρου να δείχνει στη ρίζα του άλλου.

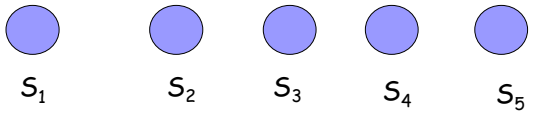
Ποια είναι η πολυπλοκότητα της Union(); *O*(1)

- Είναι επιθυμητό να κρατήσουμε το ύψος του δένδρου όσο το δυνατό μικρότερο
  - Χρειάζεται προσοχή στην επιλογή του δένδρου του οποίου η ρίζα θα δείξει στη ρίζα του άλλου!

Ξένα Σύνολα που υποστηρίζουν τη λειτουργία της Ένωσης

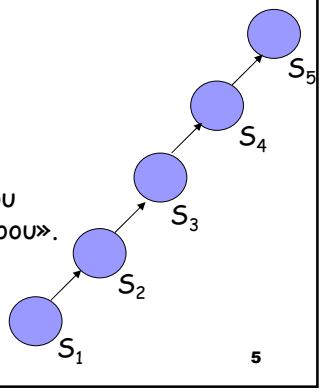
Παράδειγμα
Έστω ότι έχουμε n σύνολα με ένα στοιχείο το καθένα.

Ποιο είναι το χειρότερο ύψος δένδρου που μπορεί να προκύψει και πως πρέπει να εκτελεστεί η Union() για να προκύψει ένα τέτοιο δένδρο;



Στρατηγική για μείωση ύψους δένδρου
«Πάντα συνενώνουμε το μικρότερο δένδρο στο μεγαλύτερο, δηλαδή κάνουμε τη ρίζα του μικρότερου δένδρου να δείχνει στη ρίζα του μεγαλύτερου δένδρου».

Ένα δένδρο είναι **μεγαλύτερο** από ένα άλλο αν έχει περισσότερους κόμβους.



Ξένα Σύνολα που υποστηρίζουν τη λειτουργία της Ένωσης

Κάθε κόμβος είναι ένα struct με πεδία κάποια πληροφορία για το στοιχείο, το δείκτη **parent** στο γονικό κόμβο, και ένα μετρητή **count**, που χρησιμοποιείται μόνο αν ο κόμβος είναι η ρίζα και περιέχει το πλήθος των κόμβων στο up-tree.

```
pointer UpTreeFind(pointer P) {
// return the root of the tree containing P

  if (P == NULL) error;
  q = P;
  while (q->parent != NULL) do
       q = q->parent
  return q;
}
```

```
pointer UpTreeUnion(pointer S,T) {
// S and T are roots of up-trees */
// returns result of merging smaller into larger
   if (S == NULL OR P == NULL) return;
   if (S->count >= T->count) {
      S->count = S->count + T->count;
      T->parent = S;
       return S;
   }
   else {
       T->count = T->count + S->count;
       S->parent = T;
       return T;
   }
}
```

Ξένα Σύνολα που υποστηρίζουν τη λειτουργία της Ένωσης

Λήμμα

Έστω ότι T είναι ένα up-tree που αναπαριστά ένα σύνολο μεγέθους n, το οποίο δημιουργήθηκε με τη συνένωση n συνόλων μεγέθους 1 χρησιμοποιώντας την UPTreeUnion(). Τότε, το ύψος του T είναι το πολύ log n.

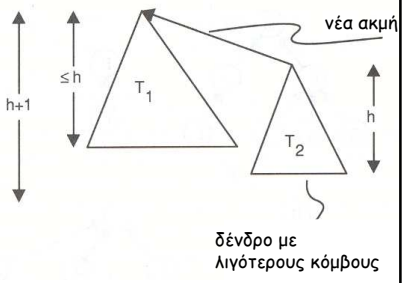
Απόδειξη: Αποδεικνύουμε ότι για κάθε h, αν το T είναι up-tree ύψους h που δημιουργήθηκε από τη συνένωση n συνόλων μεγέθους 1 (εκτελώντας επαναληπτικά την UpTreeUnion()), τότε το T έχει τουλάχιστον 2^h κόμβους, δηλαδή $|T| \geq 2^h$. Με επαγωγή στο h.

Βάση Επαγωγής (h=0): Τότε το T αποτελείται από έναν μόνο κόμβο $\Rightarrow$ $|T| = 1 \geq 2^0$.

Επαγωγική Υπόθεση: Υποθέτουμε πως για κάθε up-tree S, αν το ύψος h' του S είναι $\leq h$, τότε $|S| \geq 2^{h'}$.

Ξένα Σύνολα που υποστηρίζουν τη λειτουργία της Ένωσης

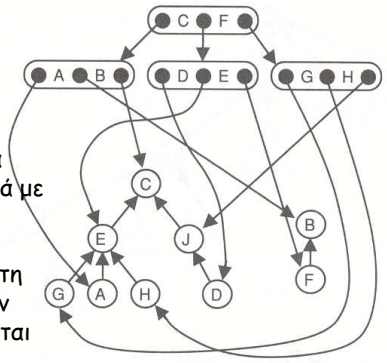
Επαγωγικό Βήμα: Συμβολίζουμε με T το **πρώτο** δένδρο που δημιουργείται με ύψος h+1 (εφαρμόζοντας τον παραπάνω αλγόριθμο).
Τότε, το T δημιουργείται με τη συνένωση ενός up-tree T_2 σε ένα up-tree T_1 για τα οποία ισχύουν τα εξής: το T_2 έχει ύψος h, ενώ το T_1 έχει ύψος το πολύ h και $|T_1| \geq |T_2|$.
Από επαγωγική υπόθεση, $|T_2| \geq 2^h$.
Επομένως,
$|T| = |T_1| + |T_2| \geq 2*|T_2|$
$\geq 2*2^h = 2^{h+1}$.
Συμπεραίνουμε επομένως ότι:
$n = |T| \geq 2^{h+1} \Rightarrow h \leq \log n - 1$



Ξένα Σύνολα που υποστηρίζουν τη λειτουργία της Ένωσης

Ποια είναι η πολυπλοκότητα της Find(X);

Υπάρχει όμως ένα λεπτό σημείο
Πως βρίσκουμε τη θέση του X στο up-tree;
Η Find(X) εμπεριέχει μια LookUp(). Πως θα υλοποιήσουμε αυτή τη LookUp;

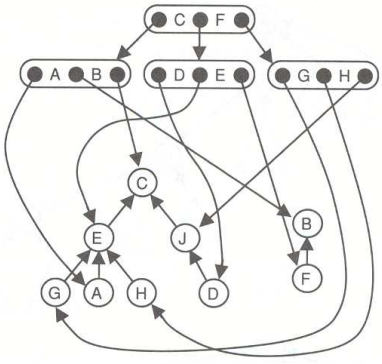


Περιπτώσεις

1. Ο χώρος των κλειδιών είναι μικρός και τα κλειδιά έχουν μικρές τιμές (π.χ. 50 κλειδιά με τιμές 1 ... 100). Τότε, είναι εφικτό να αποθηκευτούν αυτά σε έναν πίνακα 100 θέσεων του οποίου η θέση i περιέχει δείκτη προς το κλειδί i (ή NULL αν το κλειδί i δεν ανήκει στο U), οπότε η LookUp() υλοποιείται σε σταθερό χρόνο.
2. Ο χώρος των κλειδιών είναι μεγάλος. Τότε, απαιτείται μια βοηθητική δομή (π.χ., μια από τις δενδρικές δομές που υλοποιούν λεξικά) που η πληροφορία κάθε κόμβου είναι δείκτης σε ένα από τα κλειδιά του up-tree.

Ξένα Σύνολα που υποστηρίζουν τη λειτουργία της Ένωσης

- Το 2-3 δένδρο περιέχει όλα τα κλειδιά του up-tree εκτός από το μεγαλύτερο.
- Κάθε κλειδί του 2-3 δένδρου έχει συσχετισθεί με έναν δείκτη προς κάποιο κόμβο του up-tree.
- Για τα κλειδιά που είναι αποθηκευμένα σε κόμβους φύλλα, ο δείκτης αυτός είναι ο δείκτης που βρίσκεται στα αριστερά τους.
- Για τα κλειδιά που είναι αποθηκευμένα σε εσωτερικούς κόμβους, ο δείκτης αυτός είναι ο δείκτης στα δεξιά του προηγούμενού τους στην ενδοδιατεγμένη διάταξη.
- Ο δείκτης στα δεξιά του Η δείχνει στον κόμβο με το μεγαλύτερο κλειδί στο up-tree.



Ποια είναι η πολυπλοκότητα της Find() με τα νέα δεδομένα; $O(\log n)$

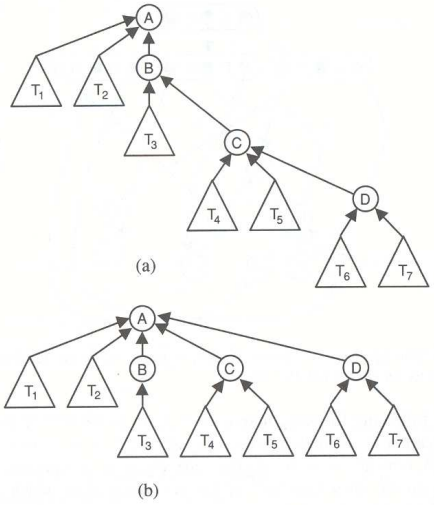
Ξένα Σύνολα που υποστηρίζουν τη λειτουργία της Ένωσης

Στρατηγική Συμπίεσης Μονοπατιού
«Κατά τη διάρκεια εκτέλεσης μιας Find(X), το parent πεδίο κάθε κόμβου στο μονοπάτι που ακολουθείται από τον κόμβο με κλειδί X στη ρίζα, αλλάζει ώστε να δείχνει στη ρίζα».

Τι αλλαγές επιφέρει η στρατηγική συμπίεσης μονοπατιού στην απόδοση?

Οι MakeEmptySet() και Union() εξακολουθούν να χρειάζονται σταθερό χρόνο.

Η Find() αρχικά εκτελείται στον ίδιο ακριβώς χρόνο όπως χωρίς να εφαρμόζεται η στρατηγική, αλλά μετά την εκτέλεση αρκετών Find(), η πολυπλοκότητά της γίνεται «σχεδόν σταθερή».



Αποτέλεσμα Find(D) όταν χρησιμοποιείται η στρατηγική συμπίεσης μονοπατιού.

Ξένα Σύνολα που υποστηρίζουν τη λειτουργία της Ένωσης

Για κάθε j, έστω F(j) η αναδρομική συνάρτηση που ορίζεται ως εξής: $F(0) = 1$ και $F(j+1) = 2^{F(j)}$, $j \geq 0$.

Οι τιμές της F(j) αυξάνουν πάρα πολύ γρήγορα καθώς αυξάνεται το j, π.χ., για j = 5, $\Phi(5) = 2^{65536} \approx 10^{19728}$. Ο αριθμός αυτός είναι πολύ μεγάλος (η διάμετρος του σύμπαντος είναι $\approx 10^{40}$ και το πλήθος μορίων σε ολόκληρο το σύμπαν είναι μικρότερο από 10^{120}).

Ορίζουμε την log*n ως εξής:

log*n = το ελάχιστο i τ.ω. $F(i) \geq n$

= το ελάχιστο i τ.ω. $\underbrace{\log \log \log \ldots \log}_{\text{i φορές}} n \leq 1$

Οι τιμές της log*n αυξάνονται πάρα πολύ αργά καθώς αυξάνει το n, π.χ., log*n είναι ≤ 5 για οποιαδήποτε «χρήσιμη» τιμή του n.

Αν η Find() εκτελείται σε χρόνο O(log*n) είναι επομένως σαν να είναι σταθερής πολυπλοκότητας.