Επίλυση Προβλημάτων με Μετασχηματισμούς (Transformations)

Περίληψη

- Επίλυση προβλημάτων χρησιμοποιώντας μετασχηματισμούς
  - Μετασχηματισμός του προβλήματος σε πιο εύκολη έκδοση (instance simplification)
  - Μετασχηματισμός του ίδιου προβλήματος χρησιμοποιώντας νέα απεικόνιση (representation change)
  - Μετασχηματισμός του προβλήματος σε διαφορετικό πρόβλημα του οποίου η λύση είναι γνωστή (problem reduction).

Presorting

- Πολλές φορές η λύση ενός προβλήματος είναι πιο εύκολη εάν τα δεδομένα του προβλήματος είναι ταξινομημένα.
- Οπόταν, για κάποια προβλήματα, η ταξινόμηση των δεδομένων μπορεί να αποτελέσει το πρώτο βήμα.
- Παραδείγματα
  - Εύρεση των πλησιέστερων σημείων
  - convex hull
- Η ταξινόμηση μπορεί να γίνει σε χρόνο

Επίλυση Συστήματος Εξισώσεων

- Υποθέστε το σύστημα εξισώσεων

$$
\begin{array}{lllll}
a_{11}x_1 + & a_{12}x_2 & +\cdots+ & a_{1n}x_n & = b_1 \\
\vdots & \vdots & \vdots & \vdots & \vdots \\
a_{n1}x_1 + & a_{n2}x_2 & +\cdots+ & a_{nn}x_n & = b_n
\end{array}
$$

- Το οποίο γράφεται και ως

$$
Ax = b \Leftrightarrow \begin{bmatrix} a_{11} & \dots & a_{1n} \\ \vdots & \vdots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix}
$$

- Βρείτε το διάνυσμα x για το οποίο ισχύει η πιο πάνω εξίσωση.
- Υπάρχει πάντα λύση;

Gauss Elimination

- Χρησιμοποιώντας elementary operations μετασχηματίζουμε τον αρχικό πίνακα [Ab] μέχρι να γίνει «σχεδόν» τριγωνικός.

$$[Ab] = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} & b_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} & b_n \end{bmatrix}$$

- Elementary operations
  - Εναλλαγή της σειράς δύο εξισώσεων
  -
  - Αντικατάστασης μιας εξίσωσης με το άθροισμα της εξίσωσης με κάποιο πολλαπλάσιο μιας άλλης εξίσωσης
- Από τον «σχεδόν» τριγωνικό πίνακα είναι εύκολο να βρούμε τη ζητούμενη λύση στο σύστημα.

Gauss Elimination Algorithm

```
GE(A[nxn],b[n])
  for i=1 to n A[i,n+1]=b[i];
  for i=1 to n
     for j= i+1 to n
          v= A[j,i]/A[i,i];
          for k= i to n+1
              A[j,k]= A[j,k] - v*A[i,k];
```

- Απόδοση:

- Πιθανά προβλήματα;
  - Τι θα συμβεί εάν σε κάποια επανάληψη το `A[i,i]` είναι πολύ μικρό;

LU Decomposition

- **Παρατήρηση**:
  - Ο πίνακας A μπορεί να αναλυθεί στο γινόμενο δύο άλλων πινάκων

$$A = LU$$

Παράδειγμα: LU Decomposition

$$A = \begin{bmatrix} 2 & -1 & 1 \\ 4 & 1 & -1 \\ 1 & 1 & 1 \end{bmatrix}$$

$$L = \begin{bmatrix} 1 & 0 & 0 \\ & 1 & 0 \\ & & 1 \end{bmatrix} \qquad U = \begin{bmatrix} 2 & -1 & 1 \\ 0 & & \\ 0 & & \end{bmatrix}$$

$$LU = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ \frac{1}{2} & \frac{1}{2} & 1 \end{bmatrix} \begin{bmatrix} 2 & -1 & 1 \\ 0 & 3 & -3 \\ 0 & 0 & 2 \end{bmatrix} = \begin{bmatrix} 2 & -1 & 1 \\ 4 & 1 & -1 \\ 1 & 1 & 1 \end{bmatrix}$$

Gauss Elimination και LU Decomposition

- Μπορούμε να αντικαταστήσουμε $A=LU$ στο σύστημα εξισώσεων.

- Επίσης μπορούμε να ορίσουμε ένα νέο διάνυσμα $y=Ux$, οπόταν

- Έχουμε σπάσει δηλαδή το σύστημα εξισώσεων σε δύο συστήματα!
- Τα δύο αυτά συστήματα είναι πιο εύκολα να λυθούν, γιατί;
- Πια η απόδοση αυτού του αλγορίθμου;
  - Πότε είναι επιθυμητό να τον χρησιμοποιήσουμε;

Άλλες Εφαρμογές του GE

- Εύρεση του αντίστροφου ενός πίνακα

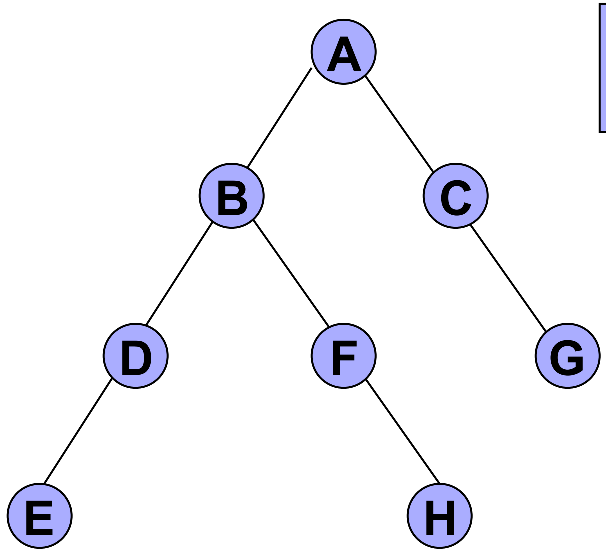
$$AA^{-1} = I$$

$$\begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \vdots & \vdots \\ a_{1n} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} x_{11} & \cdots & x_{1n} \\ \vdots & \vdots & \vdots \\ x_{1n} & \cdots & x_{nn} \end{bmatrix} = \begin{bmatrix} 1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 1 \end{bmatrix}$$

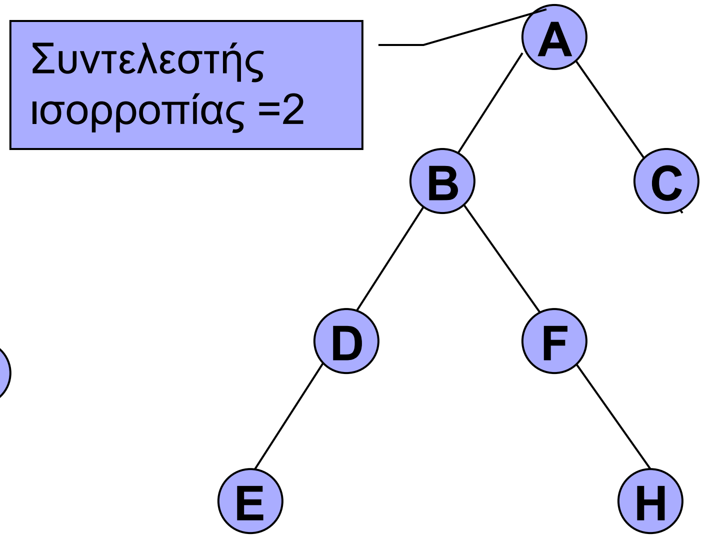
- Η εύρεση του αντίστροφου αντιστοιχεί με την επίλυση n συστημάτων εξισώσεων
- Μπορεί να χρησιμοποιηθεί το LU Decomposition για κα κάνει την διαδικασία αυτή πιο αποδοτική.

Δέντρα AVL

- Τα δέντρα AVL είναι δυαδικά δέντρα αναζήτησης των οποίων ο συντελεστής ισορροπίας (balance factor) κάθε κόμβου είναι μόνο +1, 0, -1.
  - Συντελεστής ισορροπίας: η διαφορά ύψους μεταξύ του αριστερού και δεξιού υποδέντρου



Δέντρο AVL

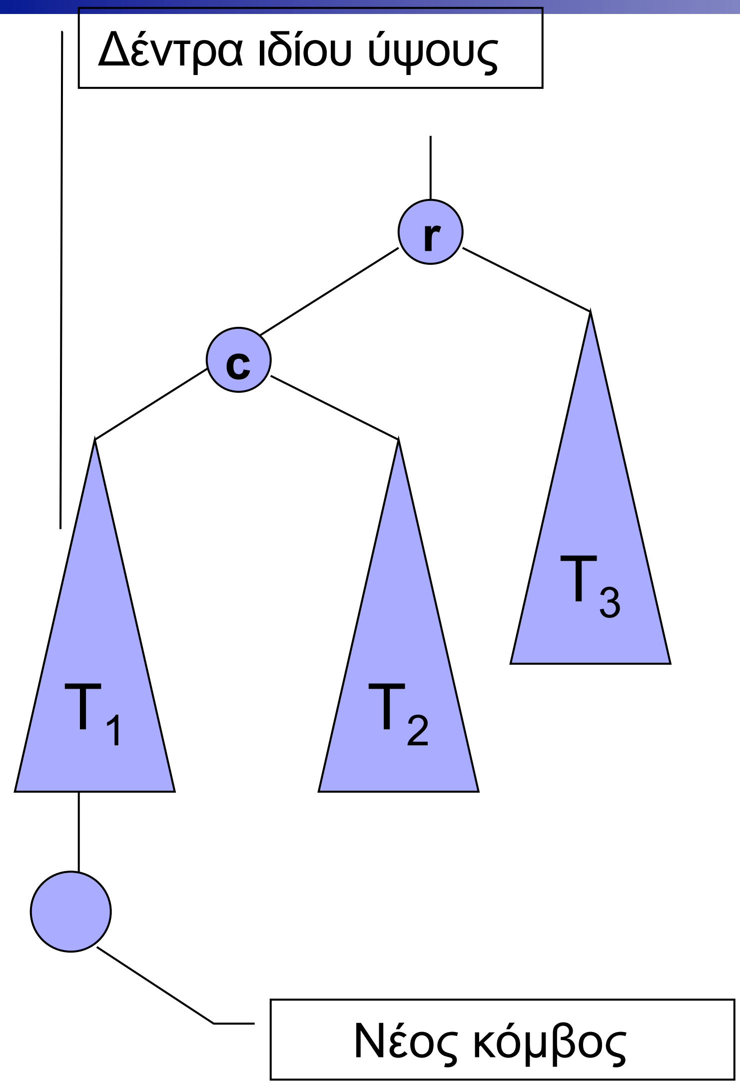


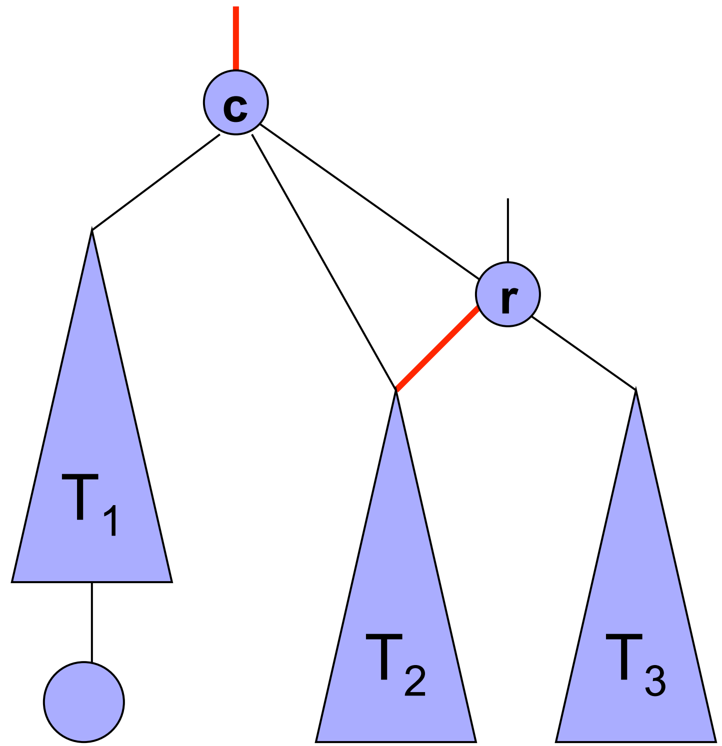
Μη Δέντρο AVL

Διατήρηση Δέντρων AVL

- Για να διατηρηθεί ένα δέντρο AVL μετά από προσθήκη ή αφαίρεση κόμβων, υπάρχουν τέσσερεις περιστροφές οι οποίες μπορούν να χρησιμοποιηθούν
  - Αριστερή περιστροφή (L-Rotation)
  - Δεξιά περιστροφή (R-Rotation)
  - Αριστερή-Δεξιά περιστροφή (LR-Rotation)
  - Δεξιά-αριστερή περιστροφή (RL-Rotation)
- Οι περιστροφές αυτές ισορροπούν το δέντρο και διατηρούν τις ιδιότητες του δυαδικού δέντρου αναζήτησης.

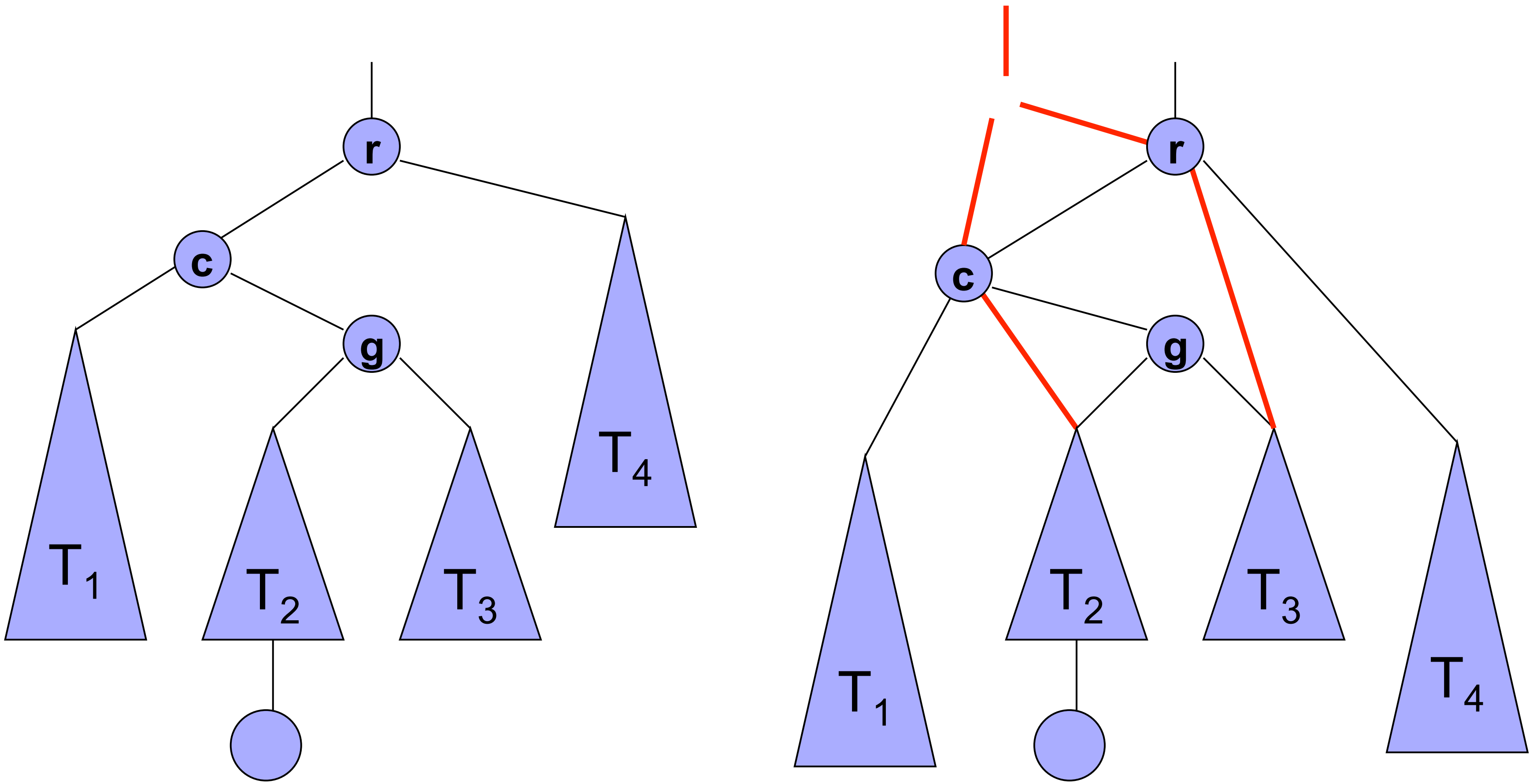
Δεξιά Περιστροφή





Η αριστερή περιστροφή είναι απλά συμμετρική.

Αριστερή-Δεξιά Περιστροφή



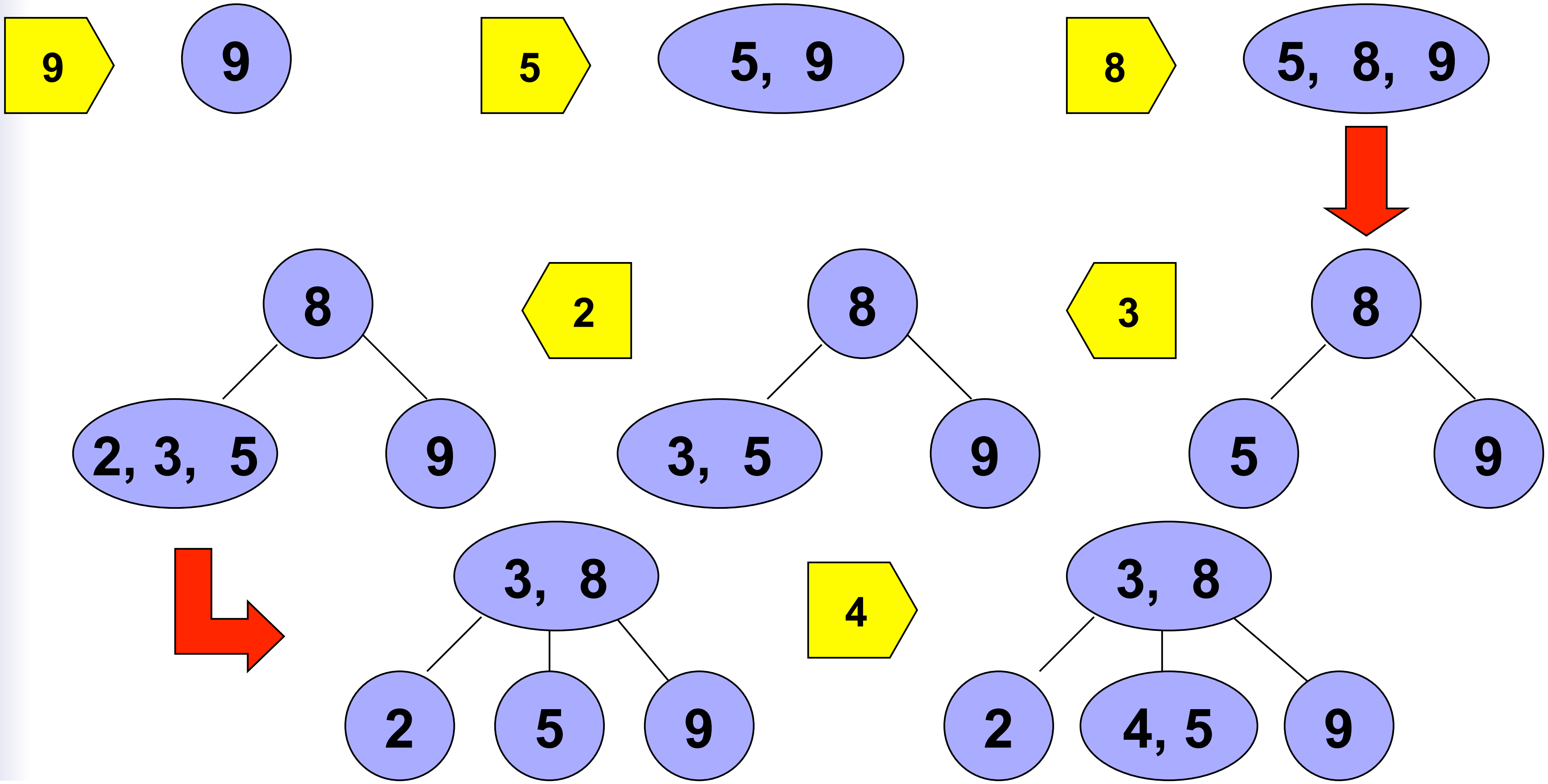
- Το αποτέλεσμα παραμένει δυαδικό δέντρο αναζήτησης;
- Η δεξιά-αριστερή περιστροφή είναι συμμετρική.

2-3 Δέντρα

- Τα 2-3 δέντρα είναι δέντρα αναζήτησης με δύο είδη κόμβων
  - Κόμβους με ένα κλειδί (δύο παιδιά) – 2-node.
    - Τα παιδιά στα αριστερά έχουν μικρότερα ή ίσα κλειδιά με τον κόμβο.
    - Τα παιδιά στα δεξιά μεγαλύτερα κλειδιά.
  - Κόμβους με δύο κλειδιά $K_1 < K_2$ (τρία παιδιά) – 3-node.
    - Τα παιδιά στα αριστερά έχουν κλειδιά μικρότερα ή ίσα με K_1.
    - Τα παιδιά στη μέση έχουν κλειδιά μεγαλύτερα από K_1 και μικρότερα ή ίσα με K_2.
    - Τα παιδιά στα δεξιά έχουν κλειδιά μεγαλύτερα από K_2.
- Τα δέντρα αυτά είναι πάντοτε απόλυτα ισορροπημένα.
- Διατήρηση ισοζυγισμού
  - Όταν ένα νέο κλειδί «φτάσει» σε ένα φύλλο τύπου 2-node τότε αυτός μετατρέπεται σε 3-node.
  - Όταν ένα νέο κλειδί «φτάσει» σε ένα φύλλο τύπου 3-node τότε αυτός σπάζει σε δύο 2-node και στέλνει το μεσαίο κλειδί στο γονέα.

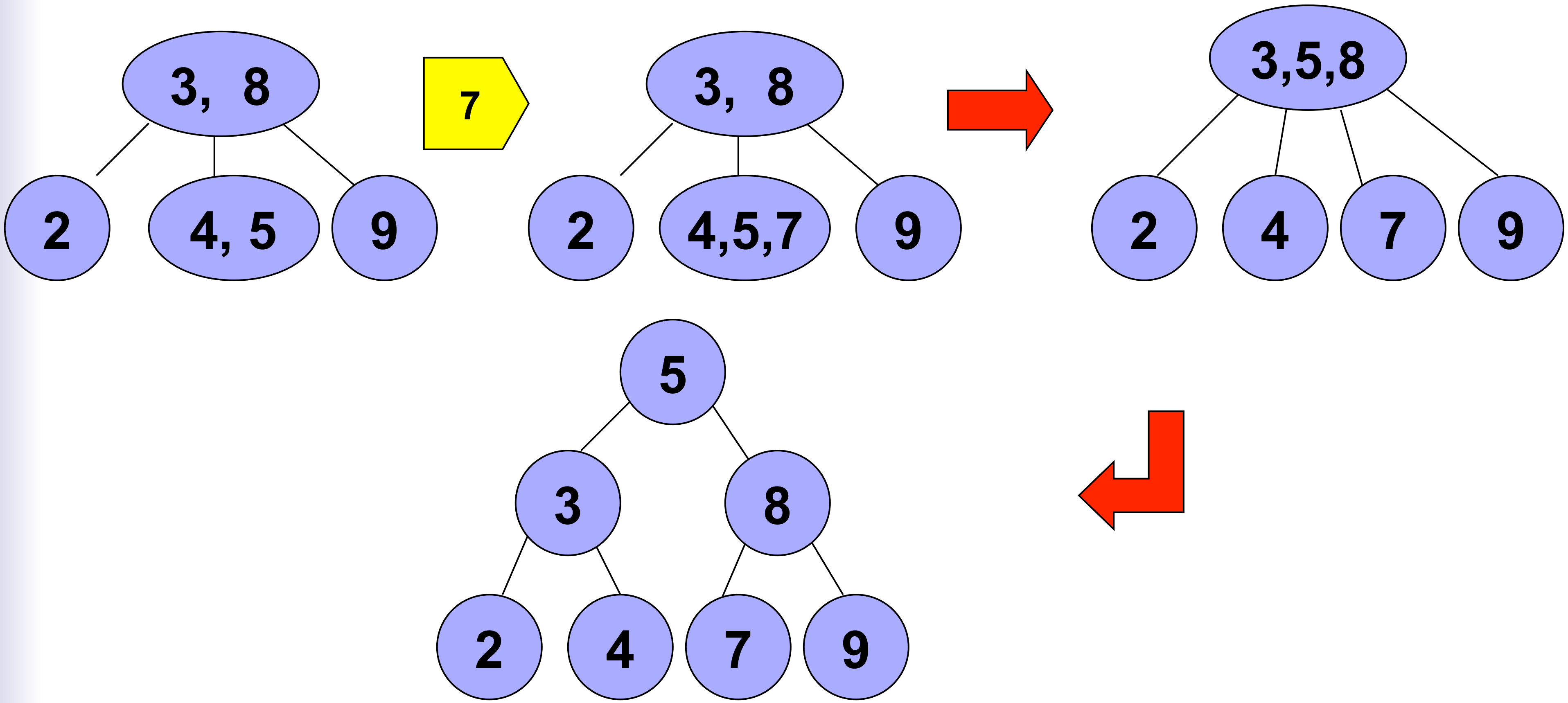
Παράδειγμα

- Υποθέστε πώς τα κλειδιά φτάνουν με την ακόλουθη σειρά: 9, 5, 8, 3, 2, 4, 7.



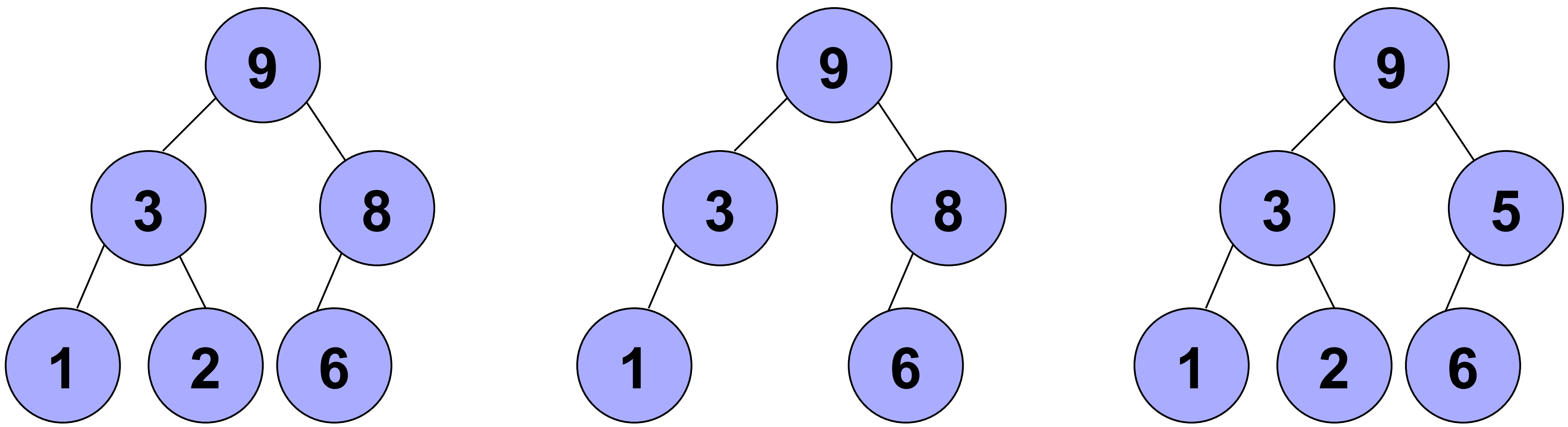
Παράδειγμα

- Υποθέστε πώς τα κλειδιά φτάνουν με την ακόλουθη σειρά: 9, 5, 8, 3, 2, 4, 7.



Heap

- Ορισμός του Heap:
  - Είναι ένα δυαδικό δέντρο (δυο παιδιά ανά κόμβο) το οποίο είναι είτε συμπληρωμένο είτε του λείπουν κάποια φύλλα από το τελευταίο επίπεδο (στα δεξιά).
  - Ο γονέας έχει πάντα μεγαλύτερο κλειδί (ή ίσο) με τα παιδιά του.



Heap

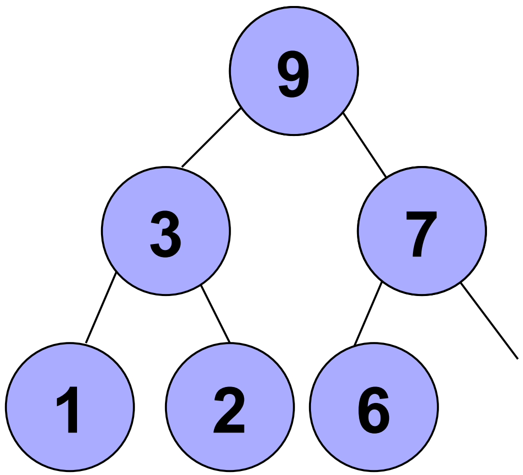
- Ιδιότητες του Heap:
  - Υπάρχει μόνο ένα συμπληρωμένο δέντρο με n κόμβους του οποίου το ύψος είναι $\lg n$
  - Η ρίζα του δέντρου είναι το **μέγιστο** κλειδί
  - Ένας κόμβος με τα παιδιά του αποτελούν πάλι heap.
  - Ένα heap υλοποιείται εύκολα σαν πίνακας
    - Τα παιδιά του κόμβου i είναι στις θέσεις $2i$, και $2i+1$.
    - Ο γονέας του κόμβου j βρίσκεται στη θέση $\left\lfloor j/2 \right\rfloor$

1	2	3	4	5	…	i	…	2*i*	2*i*+1	…	*n*

- Γιατί Heap;
  - Υλοποίηση ουράς με προτεραιότητες.
  - Βρίσκει ή διαγράφει το αντικείμενο με τη μεγαλύτερη προτεραιότητα.
  - Προσθέτει νέα αντικείμενα

Δημιουργία Heap-1

- Δημιουργία Heap (από πάνω προς τα κάτω):
  - Υποθέτουμε πως έχουμε heap με n-1 κόμβους.
  - Προσθέτουμε τον κόμβο n στην τελευταία θέση.
  - Συγκρίνουμε το κλειδί του κόμβου n με αυτό του γονέα `floor(n/2)`.
    - Εάν είναι μικρότερο, το heap είναι έτοιμο
    - Εάν όχι, ανταλλάζουμε τις θέσεις γονέα και παιδιού και επαναλαμβάνουμε



Δημιουργία Heap-2 (από κάτω προς τα πάνω)

```
Heap2(H[1…n])
  for i= floor(n/2) to 1
      k=i; v=H[k];
      heap= false;
      while !heap and 2*k<=n
           j=2*k;
           if j<n
                if H[j]<H[j+1] j=j+1;
           if v>=H[j] heap= true;
           else H[k]=H[j]; k= j;
      H[k]=v;
```

Απόδοση Αλγόριθμου Δημιουργίας Heap-2

```
Heap2(H[1…n])
   for i= floor(n/2) to 1
      k=i; v=H[k];
      heap= false;
      while !heap and 2*k<=n
           j=2*k;
           if j<n
                if H[j]<H[j+1] j=j+1;
           if v>=H[j] heap= true;
           else H[k]=H[j]; k= j;
      H[k]=v;
```

- Ένα heap έχει h=floor($\log_2 n$) επίπεδα (το ύψος του δέντρου)
- Σε κάθε επίπεδο i υπάρχουν 2^i κόμβοι
- Μετά από κάθε έλεγχο, ένας κόμβος μπορεί να βρεθεί από το επίπεδο i στο h (να γίνει δηλαδή φύλλο)
- Δύο συγκρίσεις σε κάθε βήμα

Προσθήκη και Διαγραφή κλειδιού από το Heap

- Για την προσθήκη έχουμε ήδη δει τον αλγόριθμο ο οποίος είναι τάξης $O(\log n)$. Γιατί;
- Για τη διαγραφή του μεγαλύτερου κλειδιού,
  - Ανταλλάζουμε τη ρίζα με το τελευταίο κλειδί του heap.
  - Μειώνουμε το μέγεθος του heap σε n-1.
  - Τρέχουμε τη διαδικασία δημιουργίας heap-2 (“heapify”) μόνο για i=1. (στο πρώτο for loop, `i=1 to 1`).
    - Με αυτό τον τρόπο η ρίζα πέφτει στο σωστό επίπεδο με μόνο $O(\log n)$ συγκρίσεις.
- HeapSort()
  - Τρέχουμε τη διαδικασία διαγραφής του μεγαλύτερου κλειδιού n φορές.
  - Απόδοση: $O(n\log n)$. Γιατί;

Μείωση Προβλήματος (Problem Reduction)

- Βασική Ιδέα
  - Μετασχηματίζουμε το πρόβλημα προς επίλυση σε ένα άλλο πρόβλημα το οποίο ξέρουμε πώς να επιλύσουμε!
- Παράδειγμα:
  - Θυμηθείτε τον αλγόριθμο εύρεσης του convex hull χρησιμοποιώντας αλγόριθμο τύπου divide-and-conquer. Στο αλγόριθμο αυτό έπρεπε να υπολογίσετε το σημείο που βρίσκεται πιο μακριά από μια ευθεία.
  - Το πρόβλημα αυτό «μετασχηματίστηκε» σε πρόβλημα εύρεσης του εμβαδού ενός τριγώνου.
  - Το πρόβλημα εύρεσης του εμβαδού μετασχηματίστηκε σε πρόβλημα υπολογισμού μίας ορίζουσας!

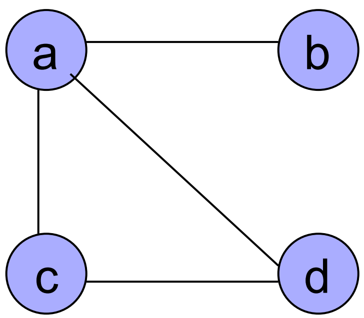
Ελάχιστο Κοινό Πολλαπλάσιο

- Πώς μπορείτε να υπολογίσετε το ελάχιστο κοινό πολλαπλάσιο δύο αριθμών m, n.
  - Ελάχιστο κοινό πολλαπλάσιο είναι ο μικρότερος αριθμός ο οποίος διαιρείται και από τους δύο χωρίς να αφήνει υπόλοιπο.

```
Algorithm1()
■ Find the prime factors of m
■ Find the prime factors of n
■ Identify all common factors that appear in both
■ Compute the product of all factors but use the
  common factors only once.
```

Αριθμός Μονοπατιών μεταξύ δύο κόμβων σε γράφο

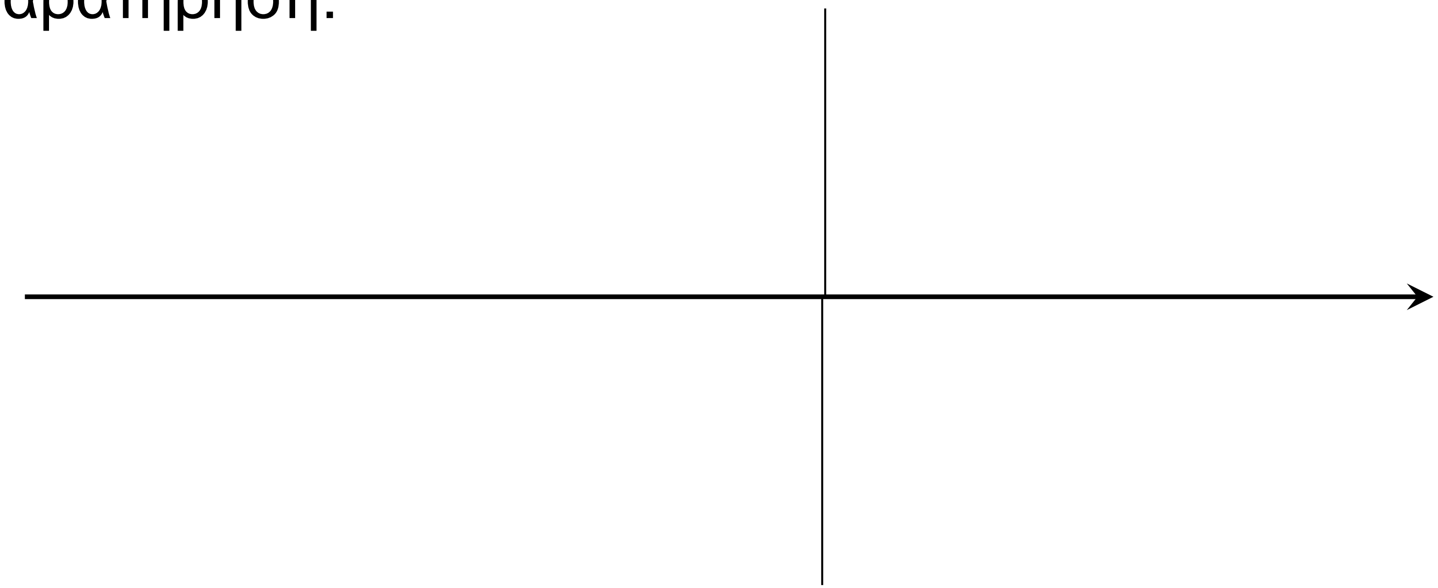
- Βρείτε των αριθμό των μονοπατιών μεταξύ των κόμβων i και j τα οποία έχουν μήκος $k>0$.
- Λύση:
  - Ο αριθμός των μονοπατιών δίνεται από το στοιχείο (i,j) του πίνακα A^k.
- Παράδειγμα:
  - Στον πιο κάτω γράφο, πόσα μονοπάτια μήκους 2 υπάρχουν από τον a στον d.



$$A = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix} \quad A^2 = \begin{bmatrix} 3 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 2 & 1 \\ 1 & 1 & 1 & 2 \end{bmatrix}$$

Ελαχιστοποίηση ή Μεγιστοποίηση Συνάρτησης

- Βρείτε τη μέγιστη τιμή της συνάρτησης $f(x)$.
- Βρείτε τη ελάχιστη τιμή της συνάρτησης $f(x)$.
- Παρατήρηση:



Γραμμικός Προγραμματισμός

- Ένας μεγάλος αριθμός προβλημάτων αποφάσεων και βελτιστοποίησης μπορεί να «μειωθεί» ή να μοντελοποιηθεί χρησιμοποιώντας **«Γραμμικό Προγραμματισμό»**
- Στη γενική του μορφή, ένα τέτοιο πρόβλημα διατυπώνεται ως ακολούθως

Συντελεστές κέρδους

Μεταβλητές απόφασης

$$\max \sum_{i=1}^{n} p_i x_i \quad \text{s.t.}$$

Συντελεστές «κατανάλωσης»

$$\sum_{i=1}^{n} a_{ji} x_i \leq b_j \quad \text{for all } j$$

Διαθέσιμοι πόροι

$$x_i \geq 0$$

- Σε προβλήματα όπου οι μεταβλητές απόφασης μπορούν να πάρουν *πραγματικές* τιμές, τότε το πρόβλημα μπορεί να λυθεί με τη μέθοδο simplex η οποία είναι γενικά αποδοτική.
- Εάν οι μεταβλητές απόφασης μπορούν να πάρουν μόνο ακέραιες τιμές, τότε το πρόβλημα γίνεται πολύ πιο πολύπλοκο.

Παράδειγμα: Γραμμικός Προγραμματισμός (ΓΠ)

- Υποθέστε πως έχετε 100 Ευρώ τα οποία θέλετε να επενδύσετε σε μπύρες για το επόμενο πάρτυ που διοργανώνεται. Στην αγορά υπάρχουν οι ακόλουθές τρεις μπύρες
  - A με τιμή p_a το λίτρο και περιεκτικότητα αλκοόλ c_a.
  - B με τιμή p_b το λίτρο και περιεκτικότητα αλκοόλ c_b.
  - C με τιμή p_c το λίτρο και περιεκτικότητα αλκοόλ c_c.
- Ξέρετε ότι οι περισσότεροι από τους φίλους σας προτιμούν την μπύρα A οπόταν θα θέλατε τουλάχιστο οι μισές μπύρες που θα αγοράσετε να είναι μάρκας A.
- Διατυπώστε ένα πρόβλημα ΓΠ που να σας βοηθήσει να αγοράσετε τις μπύρες και να μεγιστοποιήσετε το αλκοόλ που θα αγοράσετε.

$$\max c_a x_a + c_b x_b + c_c x_c$$

$$p_a x_a + p_b x_b + p_c x_c \leq 100$$

$$-x_a + x_b + x_c \leq 0$$

$$x_a, x_b, x_c \geq 0$$

Το πρόβλημα αυτό υποθέτει ότι μπορείτε να αγοράσετε οποιαδήποτε ποσότητα μπύρας.

Εάν υπήρχαν **μόνο** μπουκάλια του ενός λίτρου, τότε θα έπρεπε να μπει επιπρόσθετος περιορισμός ότι x_i πρέπει να είναι ακέραιοι.

Μείωση προβλημάτων σε γράφους

- Γιατί μας «ενδιαφέρουν» τα προβλήματα γράφων;
- Πολλά προβλήματα μπορούν να διατυπωθούν σαν προβλήματα γράφων για τα οποία υπάρχουν λύσεις!
- Παραδείγματα
  - Συντομότερο μονοπάτι
  - Ελάχιστο δέντρο επικάλυψης
  - Hamiltonian paths
  - …

Παράδειγμα μείωσης σε πρόβλημα γράφου

- Ένας βαρκάρης πρέπει να περάσει ένα λύκο, μια κατσίκα και ένα μαρούλι από την μια όχθη του ποταμού στην άλλη χρησιμοποιώντας μία βάρκα στην οποία μπορεί να χωρέσει μόνο ένα αντικείμενο!
- Λαμβάνοντας υπόψη τους περιορισμούς του προβλήματος, πως μπορεί να το πετύχει με το μικρότερο αριθμό διαδρομών;

Βλκμ**Π**:
Βαρκάρης, λύκος, κατσίκα, μαρούλι, ποτάμι, αντίστοιχα