Εφαρμογές Δομών Δεδομένων

Το περιεχόμενο (εικόνες, κώδικας) της παρουσίασης βασίζεται στο βιβλίο:
Sahni, S. (1998). Data Structures, Algorithms, and Applications in C++.

Αντιστοίχιση Παρενθέσεων

- Δίνεται μια μαθηματική έκφραση της μορφής: $(a + b) * (c * (d + e))$
- Στόχος: να βρεθεί το σύνολο των ζευγών παρενθέσεων καθώς και εκείνων που δεν αντιστοιχίζονται

Αντιστοίχιση Παρενθέσεων (2/4)

- Επίλυση του προβλήματος με χρήση στοίβας (stack)
- Η μέθοδος επίλυσης βασίζεται στη εξής παρατήρηση: σαρώνοντας την έκφραση από αριστερά προς τα δεξιά, κάθε δεξιά παρένθεση ‘)’ αντιστοιχίζεται στην τελευταία μη συσχετισμένη αριστερή παρένθεση ‘(’.
- Αλγόριθμος
  - Κάθε αριστερή παρένθεση προστίθεται στην κορυφή της στοίβας (push)
  - Για κάθε δεξιά παρένθεση αφαιρείται το τελευταίο στοιχείο που προστέθηκε στη στοίβα (pop)

Αντιστοίχιση Παρενθέσεων (3/4)

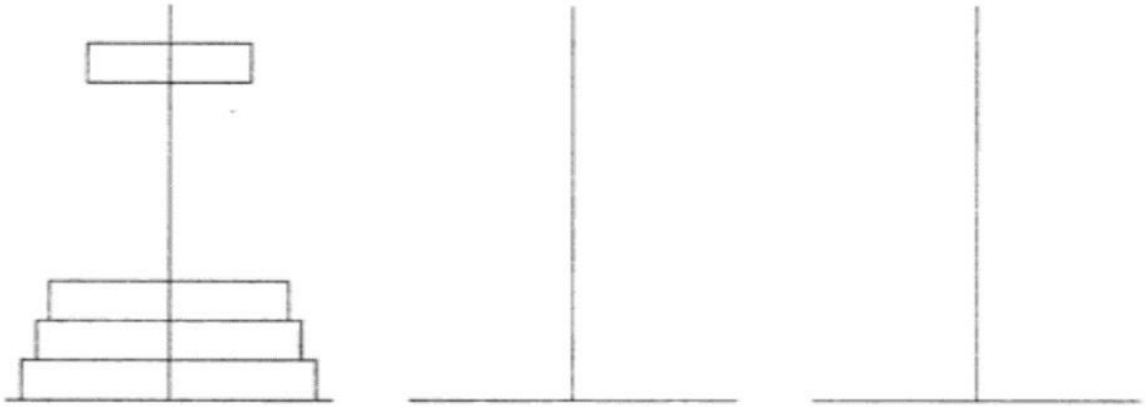
- Πολυπλοκότητα
- Η πολυπλοκότητα του αλγορίθμου είναι $\mathrm{O}(n)$, όπου n το μέγεθος της μαθηματικής έκφρασης
  - $O(n)$ push ενέργειες
  - $O(n)$ pop ενέργειες

Αντιστοίχιση Παρενθέσεων (4/4)

```
void printMatchedPairs(string expr) {// Parenthesis matching.
  arrayStack<int> s;
  int length = (int) expr.size(); // scan expression expr for ( and )
  for (int i = 0; i < length; i++)
    if (expr.at(i) == '(‘)
       s.push(i);
    else if (expr.at(i) == ')’)
       try {// remove location of matching '(' from stack
         cout << s.top() << ' ' << i << endl;
         s.pop(); // unstack match
       } catch (stackEmpty) {// stack was empty, no match exists
         // remaining '(' in stack are unmatched
         cout << "No match for right parenthesis" << " at " << i << endl; }
  while (!s.empty()) {
    cout << "No match for left parenthesis at " << s.top() << endl;
    s.pop();
  }
}
```

Πύργοι Hanoi

- Δίνονται τρεις πύργοι και *n* δίσκοι.
- Οι δίσκοι βρίσκονται στον πρώτο πύργο σε φθίνουσα σειρά (από κάτω προς τα πάνω) ως προς το μέγεθός τους



Πύργος 1 Πύργος 2 Πύργος 3

Πύργοι Hanoi

- Ζητούμενο: να μεταφερθούν οι δίσκοι στον δεύτερο πύργο
- Περιορισμοί μετακίνησης
  - Σε κάθε βήμα μόνο ένας δίσκος μπορεί να μετακινηθεί
  - Δεν πρέπει να βρεθεί μεγαλύτερος δίσκος πάνω από κάποιο μικρότερο

Πύργοι Hanoi

- Επίλυση με αναδρομή:
  - Για να μεταφέρουμε τον n-μεγαλύτερο δίσκο στον πύργο 2 πρέπει να μεταφέρουμε τους $n-1$ στον πύργο 3 και έπειτα τον n-μεγαλύτερο στον πύργο 2.
  - Στο επόμενο βήμα θα πρέπει να μεταφέρουμε τον $(n-1)$-μεγαλύτερο από τον πύργο 3 στον πύργο 2

Πύργοι Hanoi

- Πολυπλοκότητα:
- $moves(n) = \begin{cases} 0 & n = 0 \\ 2moves(n-1) + 1 & n > 0 \end{cases}$
- Πολυπλοκότητα $\Theta(2^n)$

Πύργοι Hanoi – Μέθοδος με Στοίβα

```
// global variable, tower[1:3] are the three towers
arrayStack<int> tower[4];
void moveAndShow(int, int, int, int);

void towersOfHanoi(int n) {// Preprocessor for moveAndShow.
  for (int d = n; d > 0; d--) // initialize
    tower[1].push(d); // add disk d to tower 1 // move n disks from tower 1 to 3 using 2 as //
intermediate tower
    moveAndShow(n, 1, 2, 3);
}

void moveAndShow(int n, int x, int y, int z) {// Move the top n disks from tower x to tower y showing
states. // Use tower z for intermediate storage.
  if (n > 0) {
    moveAndShow(n-1, x, z, y);
    int d = tower[x].top(); // move a disk from top of
    tower[x].pop(); // tower x to top of
    tower[y].push(d); // tower y //
    showState(); // show state of 3 towers // substitute showState code for test run
    cout << "Move disk " << d << " from tower " << x << " to top of tower " << y << endl;
    moveAndShow(n-1, z, y, x);
  }
}

void main(void) {
  cout << "Moves for a three-disk problem are" << endl;
  towersOfHanoi(3);
}
```

Πύργοι Hanoi – Με Αναδρομή

```
void towersOfHanoi(int n, int x, int y, int z) {//Move the top n
disks from tower x to tower y. Use tower z for intermediate
storage.
  if (n > 0) {
    towersOfHanoi(n-1, x, z, y);
    cout << "Move top disk from tower " << x
         << " to top of tower " << y << endl;
    towersOfHanoi(n-1, z, y, x);
  }
}

void main(void) {
  cout << "Moves for a three-disk problem are" << endl;
  towersOfHanoi(3,1,2,3);
}
```

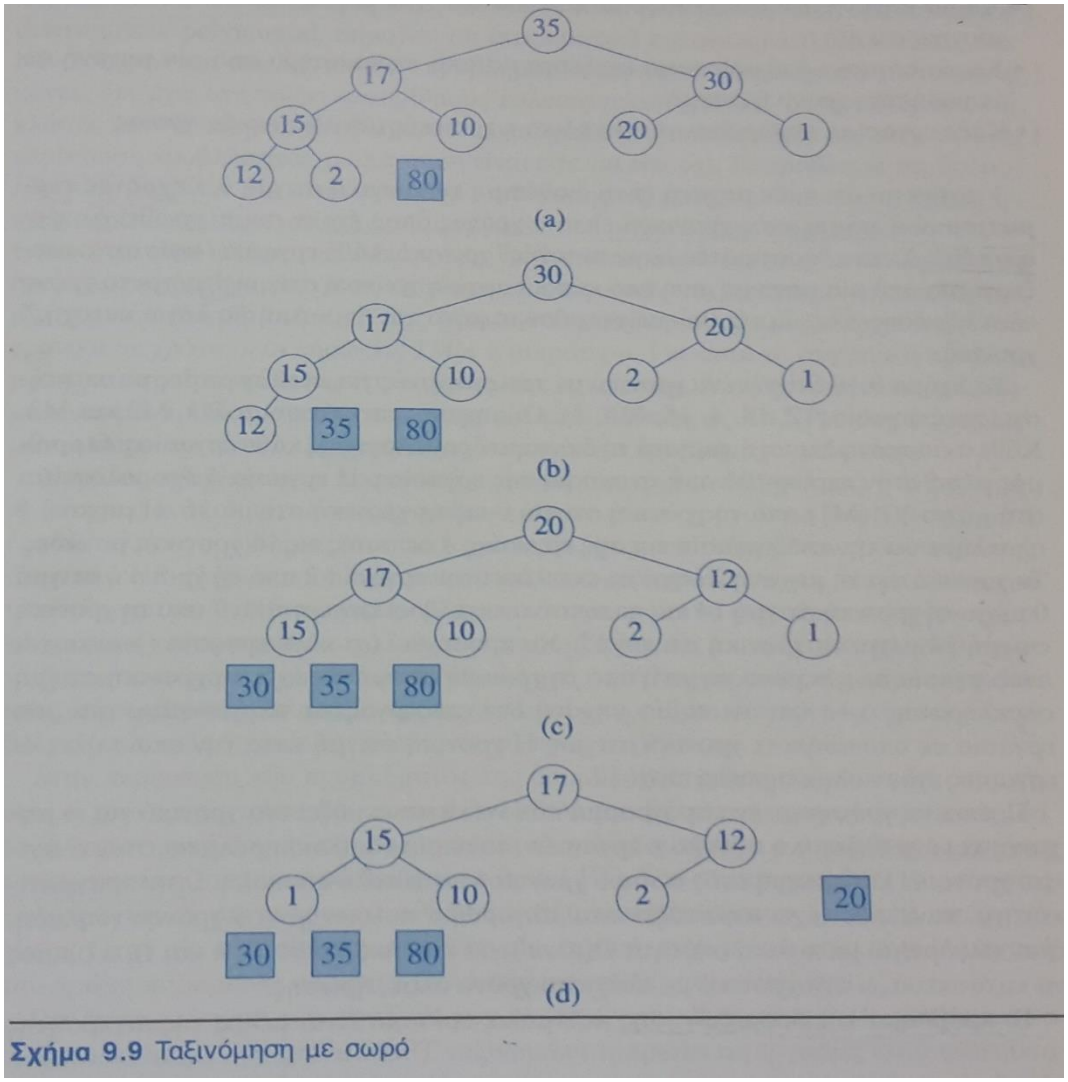
Ταξινόμηση σε Σωρό

- Ταξινόμηση n στοιχείων σε χρόνο $O(nlogn)$
- Αλγόριθμός:
  - Αρχικοποίηση των n στοιχείων σε σωρό μεγίστου (χρόνος $\Theta(n)$)
  - Εξαγωγή των στοιχείων από το σωρό (χρόνος $\mathrm{O}(logn)$)

Ταξινόμηση σε Σωρό

```
template<class T>
void HeapSort(T a[], int n){

MaxHeap<T> H(1);
H.Initialize(a,n,n);

T x;

for(int i=n-1; i>=1; i--){
        H.DeleteMax(x);
        a[i+1] = x;
}

}
```

Ταξινόμηση σε Σωρό



Σχήμα 9.9 Ταξινόμηση με σωρό

Πρόβλημα Ιστογράμματος

Έστω το σύνολο $S = \{1, \dots, n\}$, n διακριτών κλειδιών.

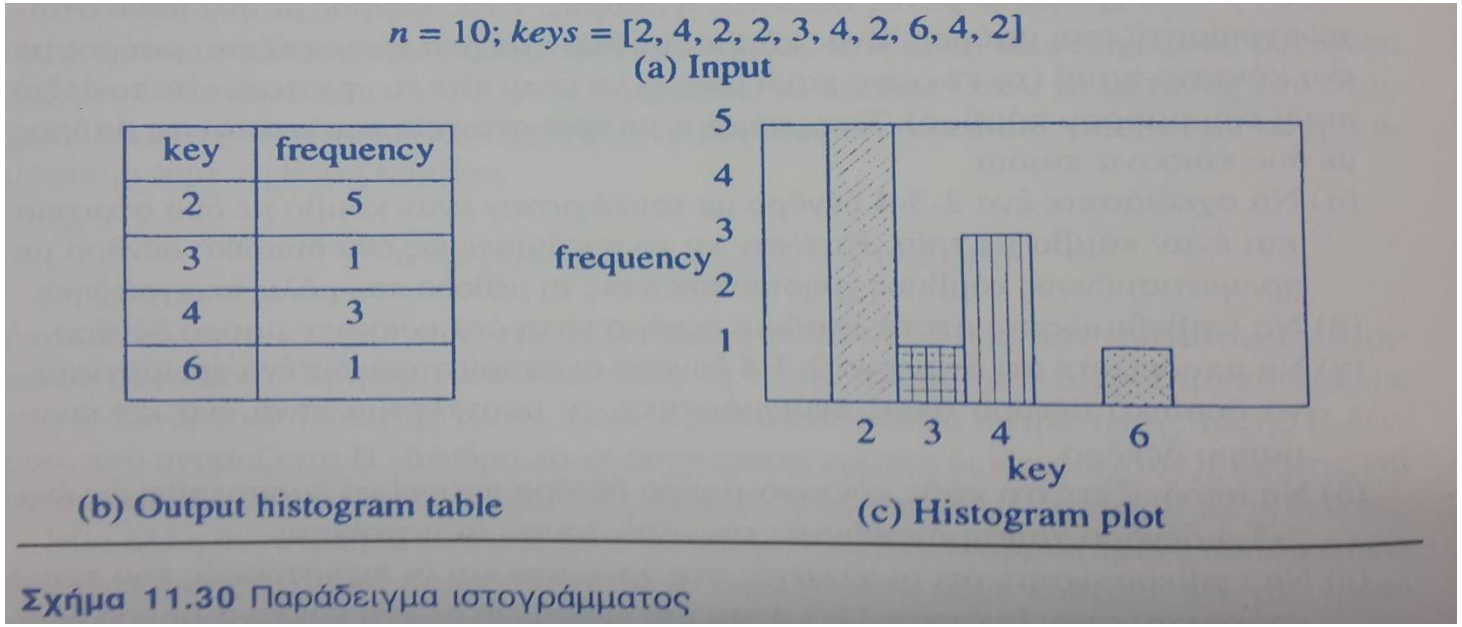
Έστω $keys = [3, 5,\ n-1,\ n, 2\ , 1, \dots]$ πίνακας εμφανίσεων των κλειδιών

Έξοδος: Μια λίστα των διακριτών κλειδιών και της συχνότητας εμφάνισής τους

Χρησιμοποιούνται συχνά για τον καθορισμό της κατανομής των δεδομένων

Πρόβλημα Ιστογράμματος

- Παραδείγματα:
  - Αποτελέσματα ενός διαγωνισμού
  - Τιμές κλίμακας του γρι σε εικόνα
  - Εγγεγραμμένα αυτοκίνητα σε μια πόλη



Σχήμα 11.30 Παράδειγμα ιστογράμματος

Πρόβλημα Ιστογράμματος

- Αναπαράσταση με Πίνακα
  - Ακέραιες τιμές κλειδιών
  - Μικρό εύρος τιμών των κλειδιών
  - Γραμμικός χρόνος κατασκευής
  - Στη θέση του πίνακα $h[i]$ αποθηκεύεται το πλήθος εμφανίσεων του κλειδιού i

Ιστόγραμμα με Array

```
void main(void){

int n,r;//n: number of elements,
        //r: values between 0 and r

cout <<"Enter number of elements
and range" <<endl;

cin >>n >>r;

int *h;

h=new int[r+1];

for(int i=0; i<=r; i++)
        h[i]=0;
```

```
for(int i=0; i<=n; i++){

int key;
cout <<"Enter element " <<i
<<endl;
cin >>key;

h[key]++
}

cout <<"Distinct elements and
frequencies are" <<endl;

for(int i=0; i<=r; i++){
        if(h[i])
                cout <<i <<" "
<<h[i] <<endl;
}
}
```

Πρόβλημα Ιστογράμματος

- Αναπαράστασης με ΔΔΑ
  - Η αναπαράσταση με πίνακα δεν είναι αποδοτική όταν το πλήθος των κλειδιών είναι πολύ μεγάλο ή όταν τα κλειδιά είναι πραγματικοί αριθμοί
  - Αποθήκευση στο δέντρο μόνο των διακριτών κλειδιών
  - Χρόνος $\mathrm{O}(n \log m)$ όπου m είναι το πλήθος των διακριτών τιμών των κλειδιών.

Ιστόγραμμα με ΔΔΑ

```
class eType{

friend void main(void);
friend void Add1(eType& );
friend ostream& operator <<(ostream&
,eType);

public:

operator int() const {return key;}

private:
int key;
int count;

};

ostream& operator<<(ostream& out, eType
x){
     out <<x.key <<" " <<x.count <<" "
;
     return out;
}

void Add1(eType& e){
     e.count++;
}
```

```
void main(void){

BSTree<eType,int> T;

int n;

cout <<"Enter number of elements"
<<endl;

cin >>n;


for(int i=1; i<=n; i++){

     eType e;

     cout <<"Enter element" <<i <<endl;

     cin >>e.key;

     e.count=1;

     T.InsertVisit(e,Add1);
     }

}
```

Πρόβλημα Συσκευασίας Κιβωτίων

Ορισμός: Έστω $S = (S_1, S_2, S_3, \dots)$ σύνολο κιβωτίων ίδιας χωρητικότητας V, και A σύνολο n αντικειμένων όγκου a_i το καθένα, όπου $i = 1, \dots, n$.

Ζητείται να βρεθεί ο ελάχιστος αριθμός κιβωτίων που χρειάζεται για να συσκευαστούν όλα τα n αντικείμενα

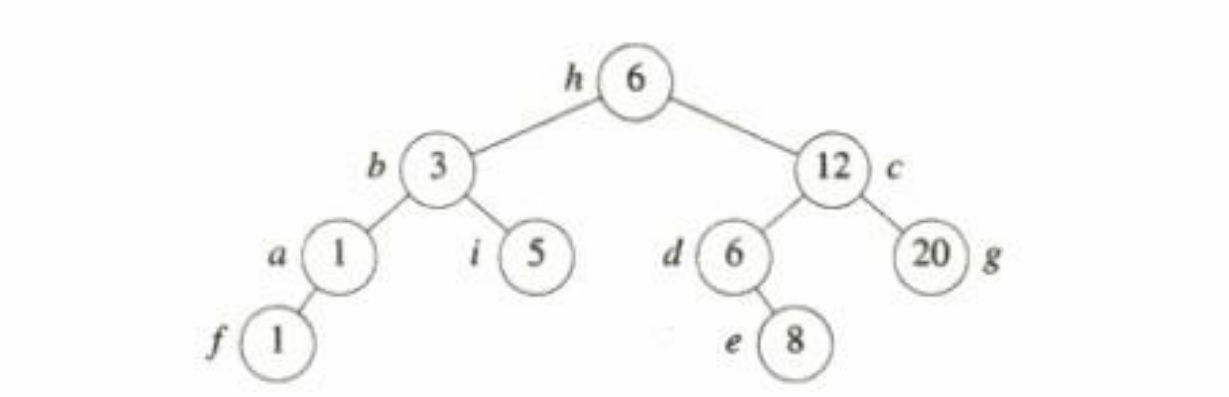
Πρόβλημα Συσκευασίας Κιβωτίων

Αλγόριθμος Best-Fit

- Για κάθε αντικείμενο i, όπου $i \in \{1, \dots, n\}$
  - Αν χωράει σε κάποιο από τα υπάρχοντα κιβώτια
    - τοποθέτησε το σε αυτό με τη **μικρότερη** εναπομείνασα χωρητικότητα και ενημέρωσε τη χωρητικότητα του κιβωτίου
  - Αν δεν υπάρχει κιβώτιο που χωράει το i
    - Τοποθέτησέ το σε ένα **νέο** κιβώτιο και ενημέρωσε τη χωρητικότητά του

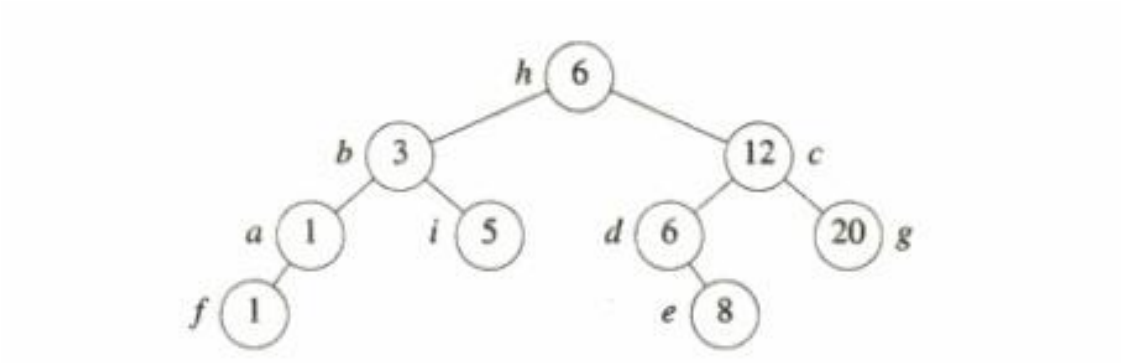
Πρόβλημα Συσκευασίας Κιβωτίων

- Υλοποίηση Best-Fit αλγορίθμου
- Με χρήση ισορροπημένου δέντρου (AVL)
- Κάθε κόμβος του δέντρου αντιστοιχεί σε κιβώτιο και έχει την πληροφορία της διαθέσιμης χωρητικότητας



Πρόβλημα Συσκευασίας Κιβωτίων

- Παράδειγμα: Έστω ότι εισάγουμε το αντικείμενο s, με $a_s = 4$
- Συγκρίνουμε το a_s με τη ρίζα του δέντρου. $h_{avail} > a_s \Rightarrow$ το κιβώτιο h είναι υποψήφιο για το s.
- Συνεχίζουμε την αναζήτηση στη ρίζα του αριστερού υποδέντρου. $b_{avail} < a_s \Rightarrow$ το b δε χωράει το s
- Ελέγχουμε τη ρίζα του δεξιού υποδέντρου. $i_{avail} > a_s$ και $i_{avail} < h_{avail} \Rightarrow$ το i γίνεται υποψήφιο για το s.
- Το i δεν έχει παιδιά, οπότε η αναζήτηση τερματίζεται



Πρόβλημα Συσκευασίας Κιβωτίων με χρήση ΔΔΑ (1/2)

```
template<class E,class K>
bool BSTree<E,K>::FindGE(const K& k, K& Kout) const{
        BinaryTreeNode<E> *p=root;
        BinaryTreeNode<E> *s=0;

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

        }
        if(!s)
                return false;

        Kout=s->data;
        return true;
}
```

Πρόβλημα Συσκευασίας Κιβωτίων με χρήση ΔΔΑ (2/2)

```
class BinNode{
friend void BestFitPack(int *,int,int);
friend ostream* operator<<(ostream&,
BinNode );

public:
operator int() const {return avail;}

private:
int ID,avail;
};
```

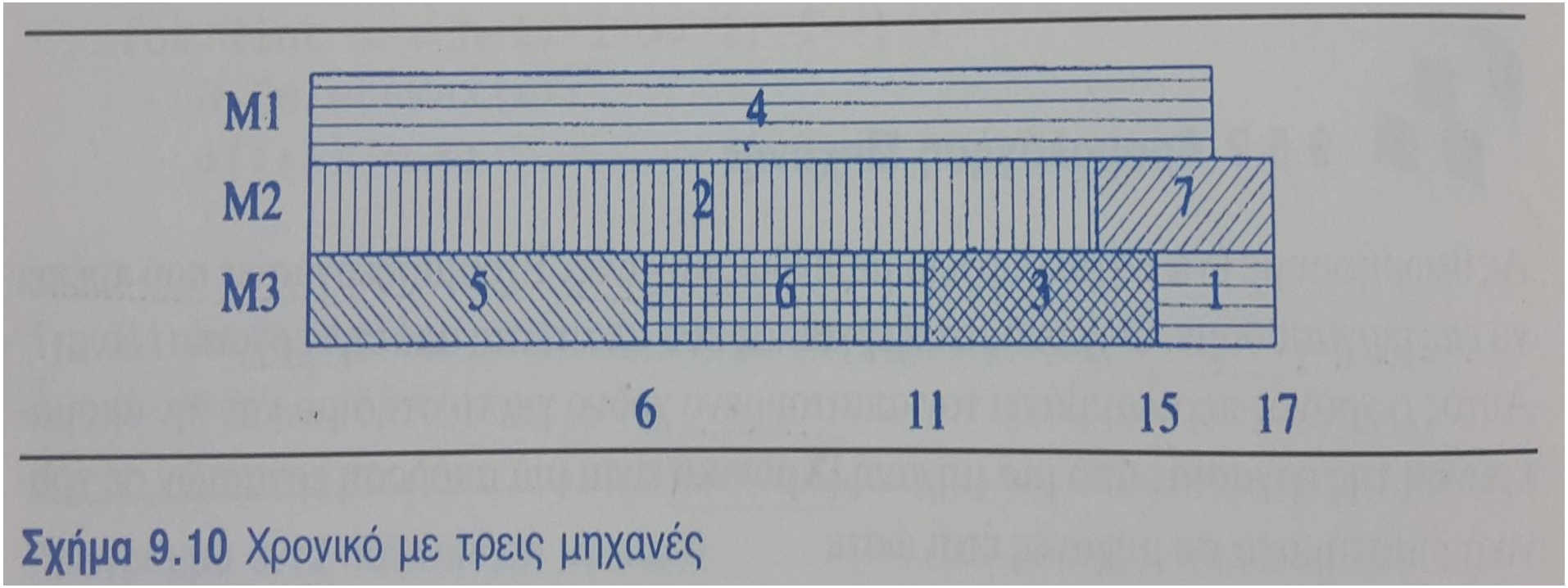
```
void BestFitPack(int s[],int n, int c){

int b=0;
BSTree<BinNode,int> T;

for(int i=1; i<=n; i++){
   int k;
   BinNode e;
   if(T.FindGE(s[i],k))
          T.Delete(k,e);
   else{
          e=*(new BinNode);
          e.ID=++b;
          e.avail=c;
   }

   e.avail-=s[i];
   if(e.avail)
          T.Insert(e);
   }

}
```

Δρομολόγηση Μηχανής

- Περιγραφή Προβλήματος:
  - m ίδιες μηχανές
  - n εργασίες
  - t_i ο χρόνος επεξεργασίας της i εργασίας
  - Στόχος: Εύρεση του σχεδίου δρομολόγησης επεξεργασίας των εργασιών έτσι ώστε να ελαχιστοποιείται ο χρόνος τερματισμού (ο χρόνος που έχουν συμπληρωθεί όλες οι εργασίες)

Δρομολόγηση Μηχανής

- Παράδειγμα



Σχήμα 9.10 Χρονικό με τρεις μηχανές

Δρομολόγηση Μηχανής

- Θεωρήσεις/Περιορισμοί
  - Καμιά μηχανή δεν επεξεργάζεται περισσότερο από μια εργασία σε οποιαδήποτε χρονική στιγμή
  - Καμιά εργασία δεν υφίσταται επεξεργασία από περισσότερο από μια μηχανή σε οποιαδήποτε χρονική στιγμή
  - Κάθε εργασία i εκχωρείται για συνολικά t_i χρονικές μονάδες επεξεργασίας

Δρομολόγηση Μηχανής - Επίλυση

- Το πρόβλημα της Δρομολόγησης Μηχανών ανήκει στα δύσκολα προς επίλυση προβλήματα (NP-hard)
- Ανάπτυξη προσεγγιστικών μεθόδων επίλυσης
- Στρατηγική Επίλυσης: Πρώτα ο Μεγαλύτερος Χρόνος Επεξεργασίας (longest processing time first, LPT)

Δρομολόγηση Μηχανής - Επίλυση

- Αλγόριθμος LPT
  - Οι εργασίες εκχωρούνται στις μηχανές κατά φθίνουσα σειρά ως προς τον απαιτούμενο χρόνο εκτέλεσης t_i
  - Όταν μια εργασία εκχωρείται σε μια μηχανή, εκχωρείται σε εκείνη την μηχανή που απελευθερώνεται πρώτη

Δρομολόγηση Μηχανής - Επίλυση

- Θεώρημα (Graham): Έστω $F^*(I)$ ο χρόνος ολοκλήρωσης ενός βέλτιστου χρονικού με m μηχανές για ένα σύνολο εργασιών I, ενώ $F(I)$ ο χρόνος ολοκλήρωσης ενός χρονικού LPT για αυτό το σύνολο εργασιών. Τότε ισχύει

$$\frac{\mathrm{F}(I)}{F^*(I)} \leq \frac{4}{3} - \frac{1}{3m}$$

Δρομολόγηση Μηχανής - Επίλυση

```
class JobNode{
        friend void
LPT(JobNode *,int ,int);

        public:
                operator int ()
const{return time;}
        private:
                int ID,time;
} ;
```

```
class MachineNode{
        friend void
LPT(JobNode *,int ,int);

        public:
                operator int ()
const{return avail;}
        private:
                int ID,avail;
};
```

Δρομολόγηση Μηχανής - Επίλυση

```
template <class T>
void LPT(T a[], int n, int m){

if(n<=m){
        cout <<"Schedule one job per machine." <<endl;
        return 0;
}

HeapSort(a,n);

MinHeap<MachineNode> H(m);
MachineNode x;

for(int i=1; i<=m; i++){
        x.avail=0;
        x.ID=i;
        H.Insert(x);
}
```

```
for(int i=n; i>=1; i--){
        H.DeleteMin(x);

        cout <<"Schedule job " <<a[i].ID <<" on machine " <<x.Id <<" from " <<x.avail <<" to " <<x.avail+a[i].time <<endl;

        x.avail+=a[i].time;
        H.Insert(x);
}

}
```

Δρομολόγηση Μηχανής - Πολυπλοκότητα

- $n \leq m$: $\Theta(1)$
- $n > m$
  - Ταξινόμηση σε σωρό: $O(n \log n)$
  - Στο δεύτερο βρόχο for εκτελούνται οι πράξεις DeleteMin, Insert. Κάθε μια χρόνο $O(n \log m)$
  - Συνολικός χρόνος: $O(n \log n + n \log m) = O(n \log n)$ (δεδομένου $n > m$)

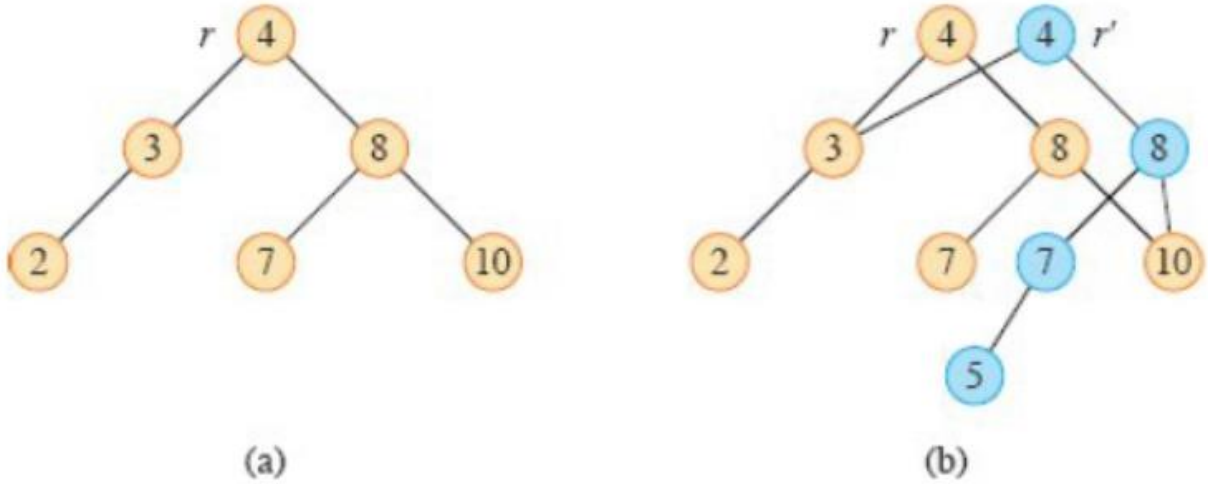
Διαχρονικά Δυναμικά Σύνολα

Περιγραφή

- Κατά τη διάρκεια εκτέλεσης κάποιων αλγορίθμων απαιτείται η διατήρηση προηγούμενων εκδοχών ενός δυναμικού συνόλου.
- Τέτοια σύνολα ονομάζονται Διαχρονικά Δυναμικά Σύνολα (Persistent Data Structures)
- Μη αποδοτική μέθοδος: Δημιουργία αντιγράφου ολόκληρου του συνόλου όταν δημιουργείται κάποια μεταβολή
  - Μειονεκτήματα: δέσμευση μεγάλου χώρου, πολυπλοκότητα αντιγραφής
- Αποδοτική μέθοδος: Διατήρηση του τμήματος που δεν επηρεάζεται από την αλλαγή και τροποποίηση μόνο του τμήματος που επηρεάζεται

Διαχρονικά Δυναμικά Σύνολα – Υλοποίηση ΔΔΑ

- Διατηρούμε μια διαφορετική ρίζα για κάθε αλλαγή του συνόλου
- Έστω εισαγωγή του κόμβου 5
  - Ο κόμβος 5 γίνεται αριστερό παιδί ενός νέου κόμβου 7
  - Ο κόμβος 7 γίνεται αριστερό παιδί ενός νέου κόμβου 8
  - Ο κόμβος 8 γίνεται δεξί παιδί ενός νέου κόμβου (ρίζα) 4
- Μετά την εισαγωγή έχει αντιγραφεί μόνο ένα τμήμα του συνόλου, αυτό που βρίσκεται στο μονοπάτι από τη ρίζα στο νέο κόμβο



Εισαγωγή σε Διαχρονικό Δυαδικό Δέντρο

Persistent_Tree_Insert(T, k)

```
1.  z = Make_NewNode(k)
2.  new_root = Copy_Node(root[T])
3.  y = null
4.  x = new_root
5.  while x! = null
6.      do y = x
7.          if key[z] < key[x]
8.                  x = Copy_Node(left[x])
9.                  left[y] = x
10.         else x = Copy_Node(right[x])
11.                 right[y] = x
12. if y == null
13.     then new_root = z
14. else if key[z] < key[y]
15.         then left[y] = z
16. else right[y] = z
17. return new_root
```

Εισαγωγή σε Διαχρονικό Δυαδικό Δέντρο με Χρήση Αναδρομής

Persistent-Tree-Insert(T, z)

1. Δημιούργησε ένα νέο Διαχρονικό ΔΔΑ T′
2. T′. root = Rec_Persistent_Tree_Insert(T. root, z)
3. return T′

Rec_Persistent_Tree_Insert(r, k)

```
1. if r == null
2.         x = Make_New_Node(k)
3. else
4.         x = Copy_Node(r)
5.         if k < key[r]
6.                 left[x] = Rec_Persistent_Tree_Insert(left[r], k)
7.         else
8.                 right[x] = Rec_Persistent_Tree_Insert(right[r], k)
9.    return x
```

Διαγραφή σε Διαχρονικό Δυαδικό Δέντρο

- Έστω ότι διαγράφεται ο κόμβος z
- Περίπτωση 1:
  - Ο z έχει το πολύ ένα παιδί ⇒ διέγραψε τον z και όπως στην εισαγωγή δημιούργησε νέους κόμβους μέχρι και τη ρίζα του δέντρου
- Περίπτωση 2:
  - Ο z έχει δύο παιδιά και ο επόμενος μεγαλύτερος του z, έστω y είναι δεξί παιδί του z ⇒ αντικατέστησε τον z με τον y και δημιούργησε νέους κόμβους μέχρι και τη ρίζα (όπως και στην εισαγωγή)
- Περίπτωση 3:
  - Ο z έχει δύο παιδιά και ο επόμενος μεγαλύτερος του z, έστω y δεν είναι δεξί παιδί του z ⇒ αντικατέστησε τον z με τον y και τον y με το δεξί του παιδί, έστω x. Αφού z, y είναι πρόγονοι του x, όλοι οι πρόγονοι του y μέχρι και τη ρίζα πρέπει να αλλάξουν (δημιουργία νέων κόμβων)

Treap

Εισαγωγή

- **Θεώρημα**: Το αναμενόμενο ύψος ενός δυαδικού δέντρου αναζήτησης που δημιουργείται με εισαγωγές n διαφορετικών κόμβων με τυχαία σειρά ως προς το κλειδί του κόμβου είναι $O(\lg n)$
- Αν έχουμε όλα τα στοιχεία εκ των προτέρων:
  - εκτελούμε μια τυχαία αντιμετάθεση και τα εισάγουμε στο δέντρο
- Τι γίνεται σε αντίθετη περίπτωση;

Δομή Δεδομένων Treap

- Μια δομή δεδομένων treap είναι ένα δυαδικό δέντρο αναζήτησης στο οποίο έχει επέλθει τροποποίηση στη διάταξη των κόμβων
- Ένας κόμβος x περιέχει τα εξής πεδία:
  - $x.key$
  - $x.priority$
  - $x.left$
  - $x.right$
- Η τιμή για το πεδίο $priority$ επιλέγεται τυχαία

Διάταξη Κόμβων Treap

- Θεωρούμε ότι τα πεδία key και $priority$ είναι διαφορετικά μεταξύ τους.
- Η διάταξη των κόμβων είναι τέτοια ώστε τα πεδία $keys$ να σχηματίζουν δυαδικό δέντρο αναζήτηση και τα πεδία $priority$ να σχηματίζουν σωρό μεγίστου.
- Αν v είναι αριστερό παιδί του u: $v.key < u.key$
- Αν v είναι δεξί παιδί του u : $v.key > u.key$
- Αν v είναι παιδί του u: $v.priority < u.priority$

Εισαγωγή Κόμβου

- Ο κόμβος εισάγεται με βάση το πεδίο key, όπως και σε ένα δυαδικό δέντρο αναζήτησης
- Γίνονται οι απαιτούμενες περιστροφές έως ότου αποκατασταθεί η ιδιότητα του σωρού μεγίστου ως προς το πεδίο $priority$.

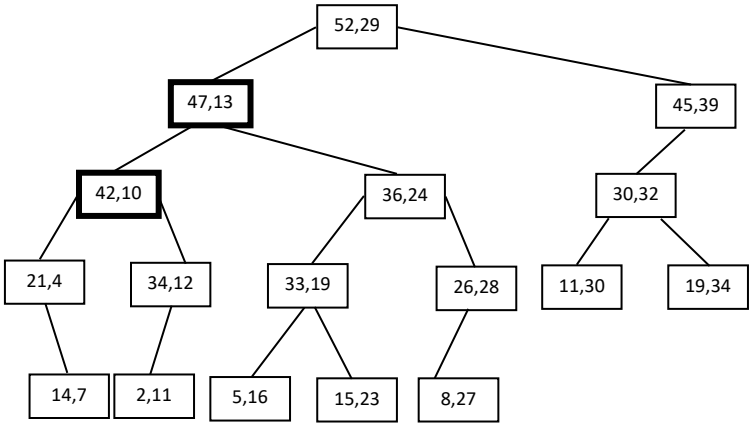
Διαγραφή Κόμβου

- Εύρεση του προς διαγραφή κόμβου.
- Αν ο κόμβος είναι φύλλο, τότε απλά διαγράφεται
- Αν δεν είναι:
  - Περιστροφές μέχρι να γίνει φύλλο.
  - Διαγραφή του φύλλου

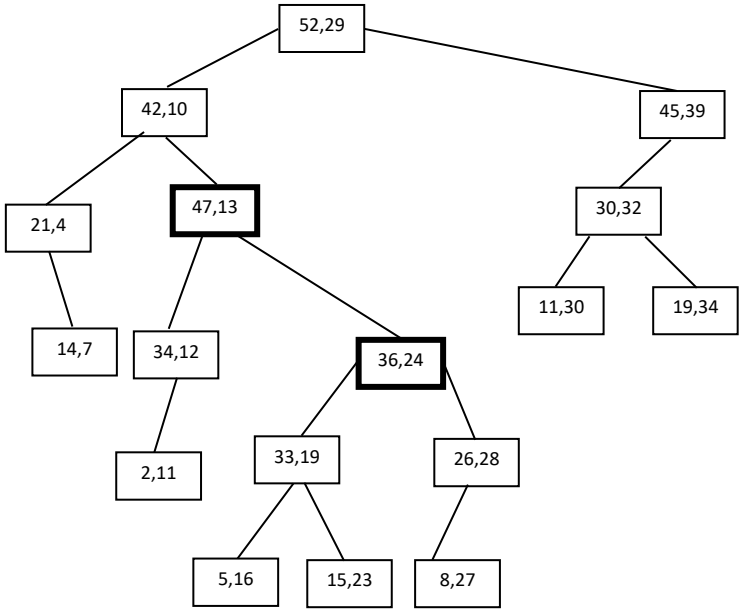
Διαγραφή - Παράδειγμα

- Στο σχήμα που ακολουθεί, διαγραφή του ζεύγους (47,13).
- Το ζεύγος υποβιβάζεται διαδοχικά στο δέντρο με μία σειρά διαδοχικών απλών περιστροφών μέχρι να αποθηκευτεί σε φύλλο του δέντρου όποτε και εύκολα διαγράφεται.
- Σε κάθε βήμα, η περιστροφή αφορά τον κόμβο στον οποίο βρίσκεται το ζεύγος (47,13) και ένα από τα δύο παιδιά του, συγκεκριμένα όποιο έχει την υψηλότερη προτεραιότητα.
- Π.χ. στο βήμα (α) ο κόμβος (47,13) έχει δύο παιδιά το (42,10) και το (36,24). Η περιστροφή γίνεται μεταξύ του κόμβου (47,13) και (42,10) δηλ. με το παιδί που έχει την υψηλότερη προτεραιότητα.
- Στο ίδιο σχήμα φαίνεται και η διαδικασία της εισαγωγής ακολουθώντας τα βήματα από το τέλος προς την αρχή.
- Συγκεκριμένα στο σχήμα (στ), γίνεται εισαγωγή του στοιχείου (47,13), λαμβάνονται υπόψη μόνο το πεδίο *key*.
- Στη συνέχεια, με μία σειρά περιστροφών το ζεύγος (47,13) ανεβαίνει ψηλότερα στο δέντρο μέχρι του σημείου που παύει να παραβιάζεται η βασική ιδιότητα του δέντρου μεγίστων (βήματα (στ)-(α)).

Διαγραφή - Παράδειγμα

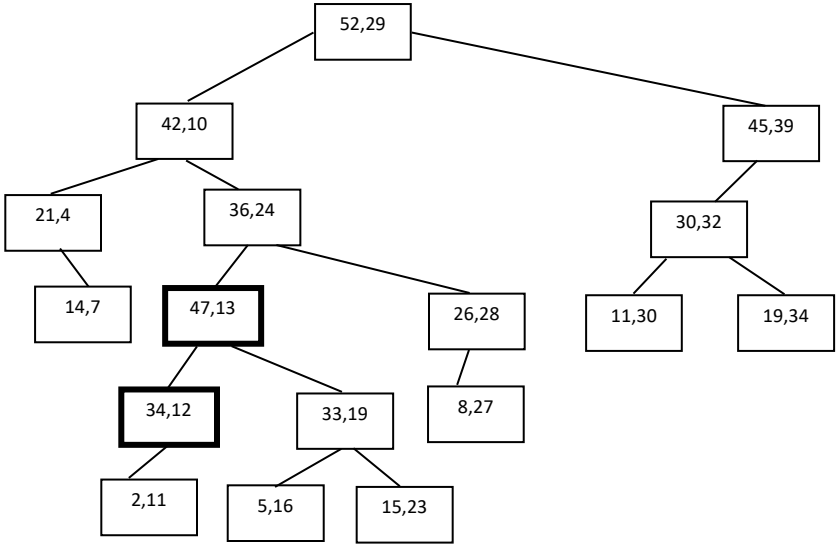


(α)

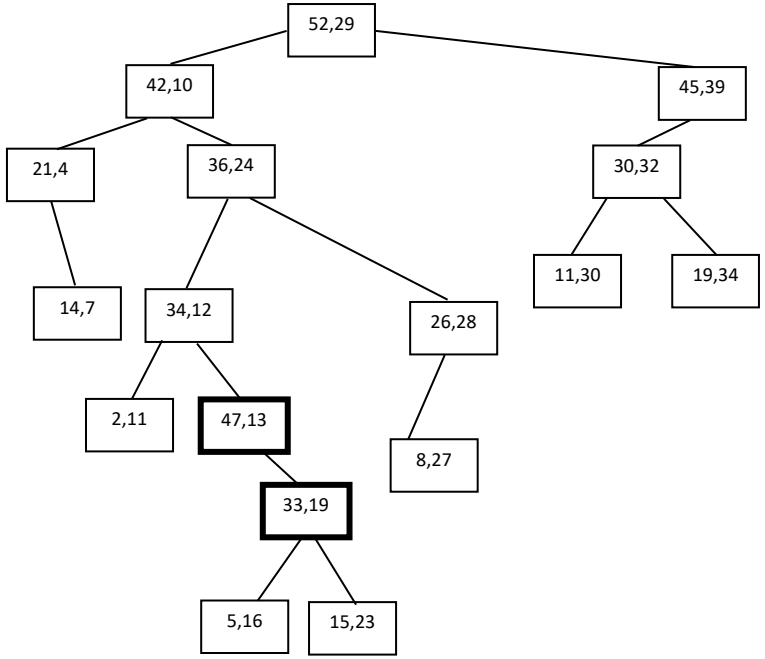


(β)

Διαγραφή – Παράδειγμα (συν.)

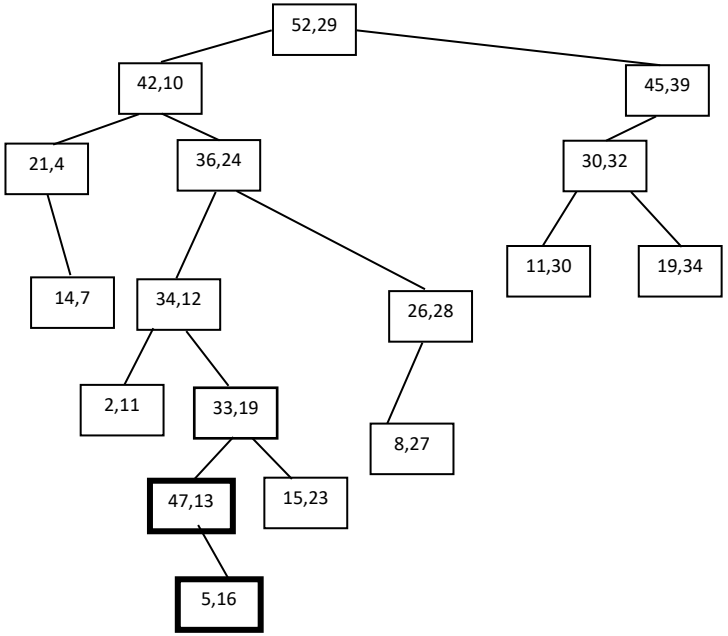


(γ)

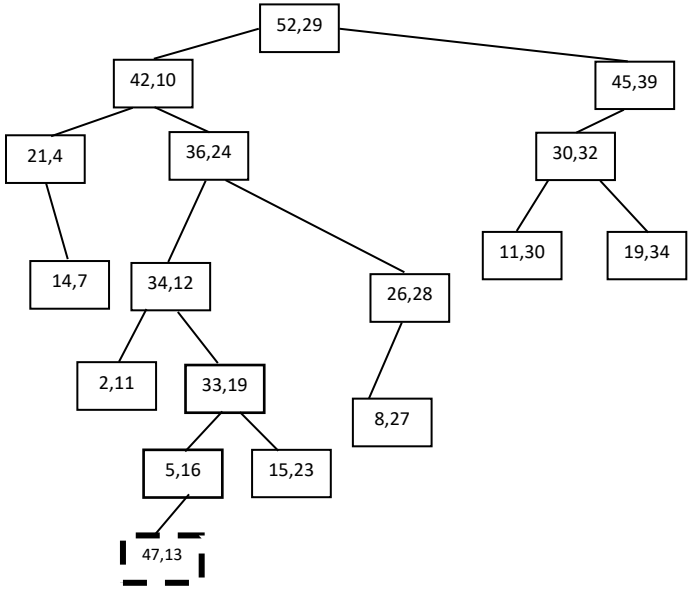


(δ)

Διαγραφή – Παράδειγμα (συν.)



(ε)



(στ)