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 (2/3)

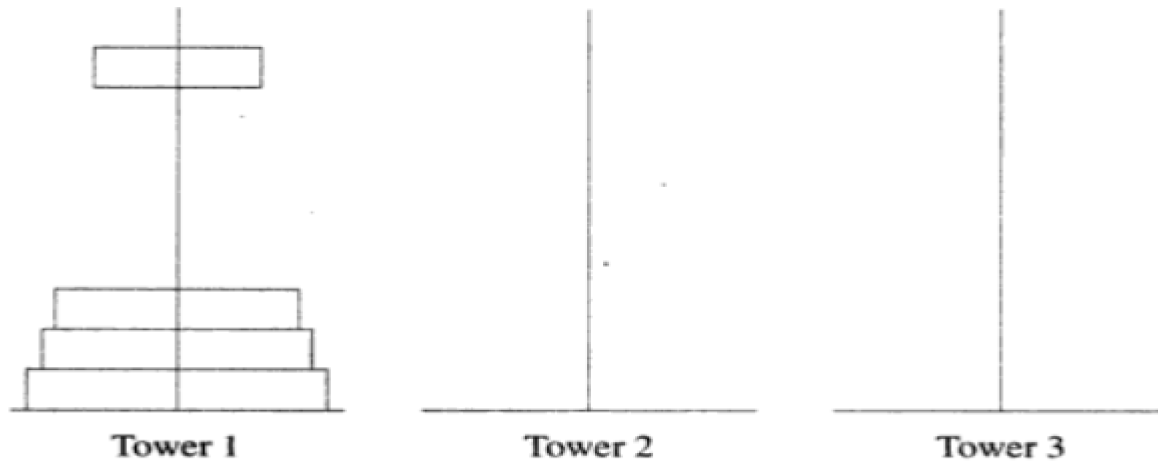
- Επίλυση του προβλήματος με χρήση στοίβας (stack)
- Η μέθοδος επίλυσης βασίζεται στη εξής παρατήρηση: σαρώνοντας την έκφραση από αριστερά προς τα δεξιά, κάθε δεξιά παρένθεση ‘)’ αντιστοιχίζεται στην τελευταία μη συσχετισμένη αριστερή παρένθεση ‘(’.
- Αλγόριθμος
  - Κάθε αριστερή παρένθεση προστίθεται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

Αντιστοίχιση Παρενθέσεων (3/3)

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

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



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 $\Theta(2^n)$

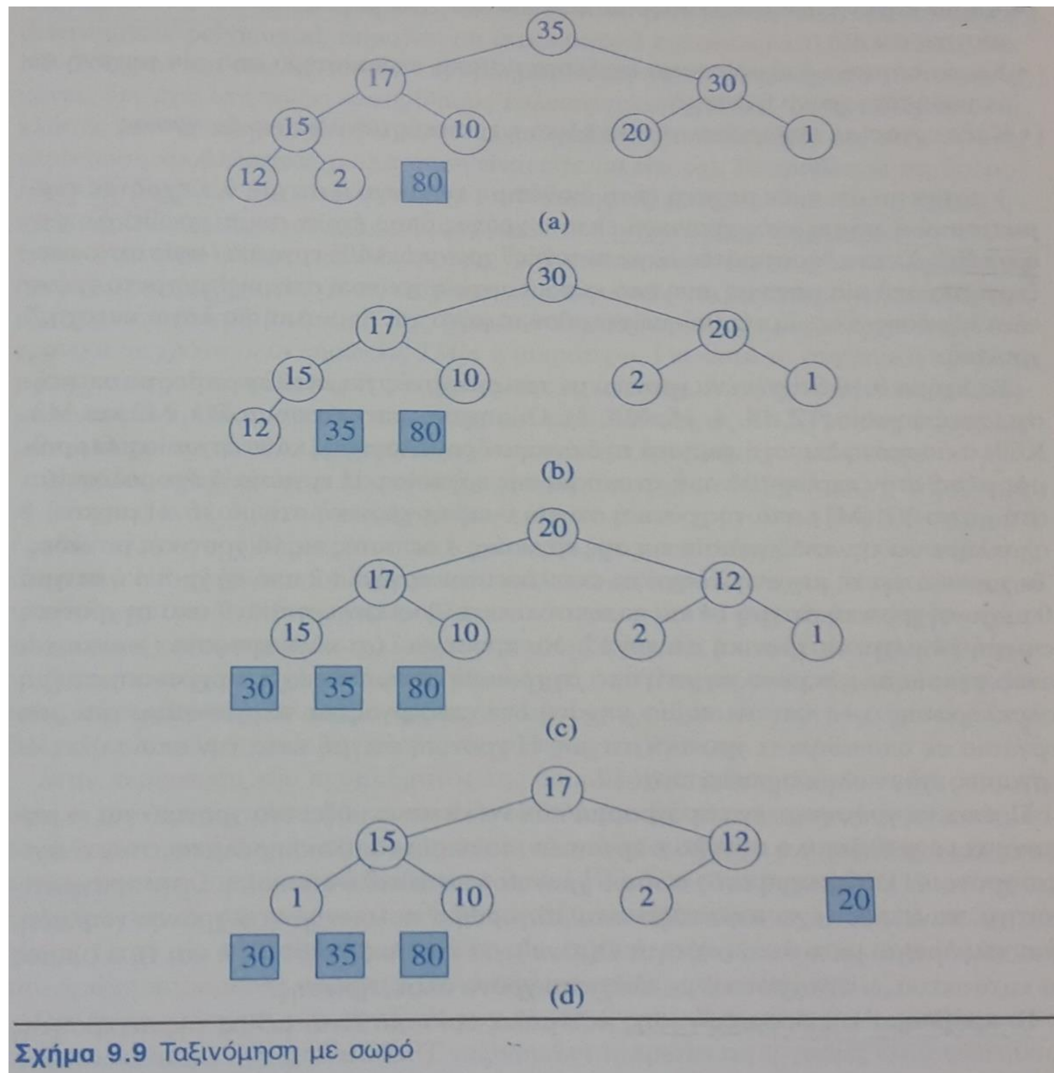
Ταξινόμηση σε Σωρό

- 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 $S = \{1, \ldots, n\}$, n διακριτών κλειδιών.

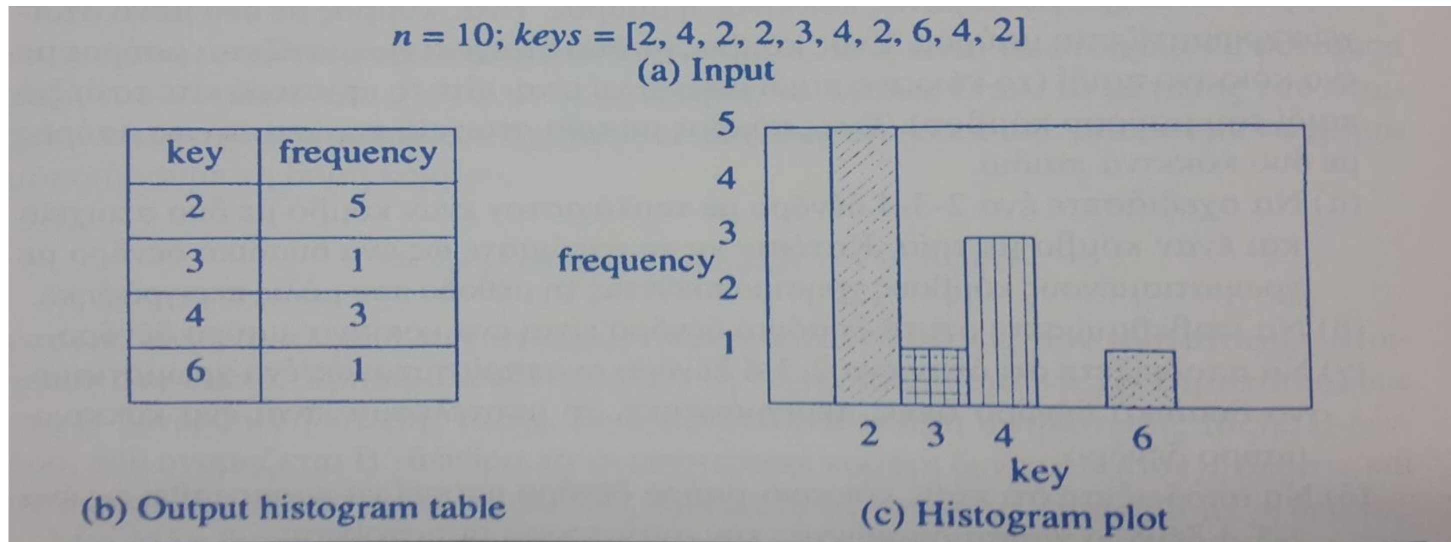
Έστω $keys = [3, 5,\ n-1,\ n, 2\ , 1, \ldots]$ πίνακας εμφανίσεων των κλειδιών

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



key	frequency
2	5
3	1
4	3
6	1

(b) Output histogram table

Σχήμα 11.30 Παράδειγμα ιστογράμματος

Πρόβλημα Ιστογράμματος

- Αναπαράστασης με Πίνακα
  - Ακέραιες τιμές κλειδιών
  - Μικρό πλήθος 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;

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<=n; 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 O$(nlogn)$

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);
     }

}
```

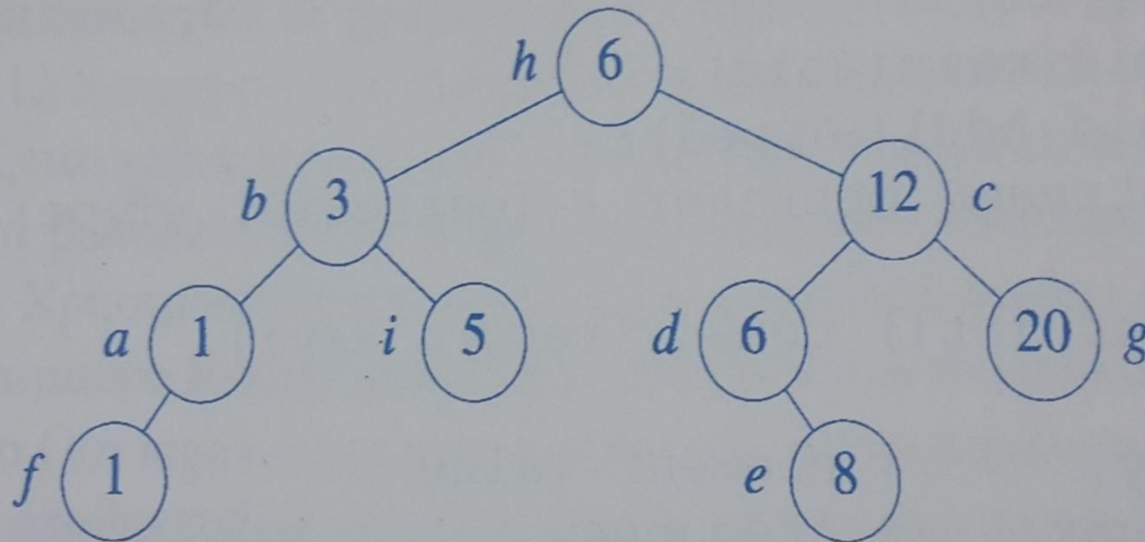
Πρόβλημα Συσκευασίας Κιβωτίων

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

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

Πρόβλημα Συσκευασίας Κιβωτίων

- Παράδειγμα



Σχήμα 11.31 Δένδρο AVL με διπλότυπα

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;
};
```

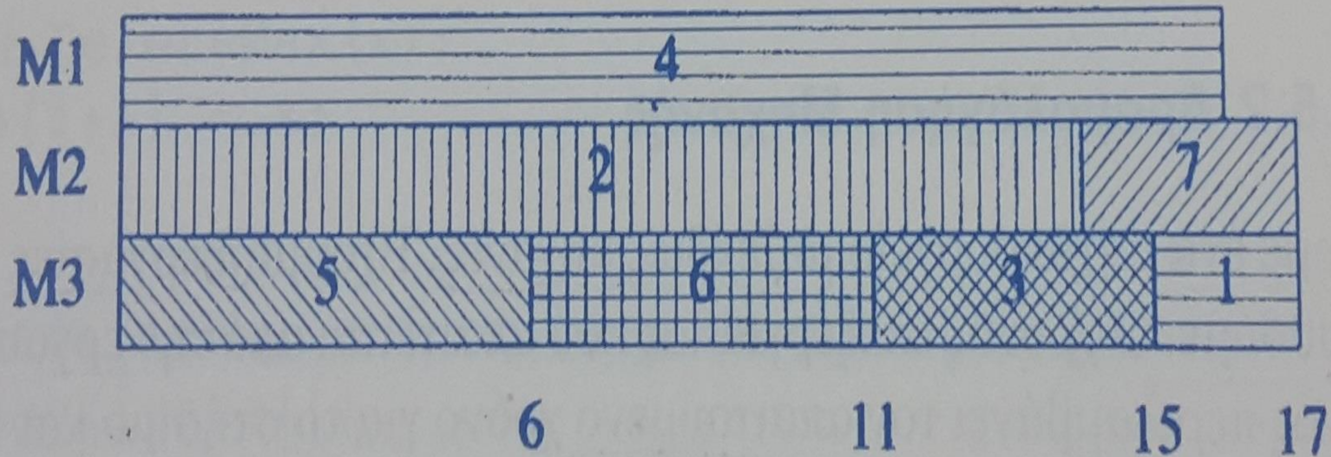
```
void BestFitPack(int s[],int n, int c){

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: $O(nlogn)$
  - Στο δεύτερο βρόχο for εκτελούνται οι πράξεις DeleteMin, Insert. Κάθε μια χρόνο $O(nlogm)$
  - Συνολικός χρόνος: $O(nlogn + nlogm) = O(nlogn)$ (δεδομένου $n > m$)