

Δορές Αεδορένων



Σύνοδα / Sets

- Σύνοδα έχει περισσότερες από τις πιθανές τιμές που προσεται να πάρει ενα αριθμός του
- Στην θεώρη συγκαταλέγεται ότι είναι ισχυρή και ασφαλής η αριθμητική συγκέντρωση της θεώρης
- Σύνοδος διαρραγής false
- Κέντρο Set ορίζεται false

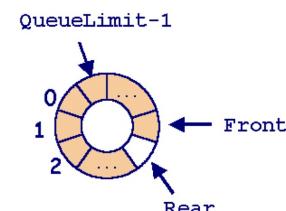
Σύνοδα - Stack

- Προσέδωση **Movo από Top**
- Κέντρο Stack, Stack.Top = -1
- Διαρραγή **POP**, λογική LIFO, δευτέρο->πρώτο
- Εισαγωγή **PUSH**, αναβάθμιση Top μετά / και συγκαταλέγεται προσεται να Item

```
void TraverseStack(StackType Stack){  
    int i;  
    printf("\nplithos sto stack %d\n", Stack.Top+1);  
    for (i=Stack.Top;i>=0;i--) {  
        printf("%d ", Stack.Element[i]);  
    }  
    printf("\n");  
}
```

Ουρά - Queue

- Έχει 2 αριθμούς **Front** για διαρραγή και **Rear** για εισαγωγή, λογική **Spiral** FIFO
- Έχει κυριότερη σημασία



- Κενό Queue : $\text{Front} = \text{Rear}$
- Γεράτο : $(\text{Rear} + 1) \% \text{QueueLimit} == \text{Front}$
- Προσέλαση $\text{Front} \neq \text{Rear}$
- Διαγραφή : $\text{Front} = (\text{Front} + 1) \% \text{Limit}$
- Εισαγωγή : $[\text{Rear}] = \text{Item}$
 $\text{Rear} = (\text{Rear} + 1) \% \text{QLimit}$

```

void TraverseQ(QueueType Queue){
    int current;
    printf("Printing Queue\n");
    if (!EmptyQ(Queue)) {
        current = Queue.Front;
        while (current != Queue.Rear) {
            printf("%d ", Queue.Element[current]);
            current = (current + 1) % QueueLimit;
        }
        printf("\n");
    } else printf("Empty Queue\n");
}

```

μπροστά αντιστρέψει

Σελίκη Ενδεξερένη Λίστα

- Κενή Λίστα : List.Size == 0
- Προσέλαση στο Τανό
- Εισαγωγή : Μεταστοιχία όλων των συστατικών από τη δεξιά, πια δεξιά δεξιά
- Διαγραφή : Μεταστοιχία όλων των συστατικών από τη δεξιά, πια δεξιά αριστερά
- Χρησιμοποιεί Nodes

Data	Next
Datatype	Ptr
- Τελευταίο συστατικό λίστας είναι Next το NilValue String, το -1
- Προσέλαση

```

void TraverseLinked(ListPointer List, NodeType Node[]){
    ListPointer CurrPtr;

    if (!EmptyList(List))
    {
        CurrPtr = List;
        while (CurrPtr != NilValue)
        {
            printf("(%d) ", Node[CurrPtr].Data);
            CurrPtr = Node[CurrPtr].Next;
        }
        printf("\n");
    } else printf("Empty List ... \n");
}

```

Η εισαγωγή έχει 2 περιπτώσεις

• Στην αρχή της λιστας

O ptr της λιστας

δείχνει το κενό πλάνο

συνέχεια και

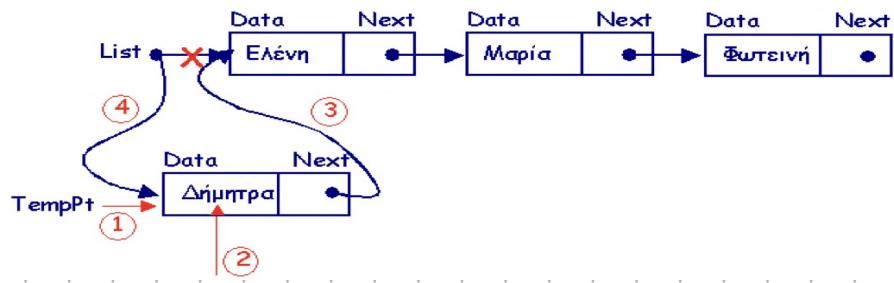
το Next του κενού

συνέχεια δείχνει

τον παλιό ptr της λιστας.

- (1) GetNode(TempP)
- (2) Data(TempP)='Δήμητρα'
- (3) Next(TempP)=List
- (4) List=TempP

PredPtr = NilValue



• Μετα από κάποιο συνέδεια

Αλλάζω το Next της προηγ.

καταχωρίζω σαν ptr της νέας

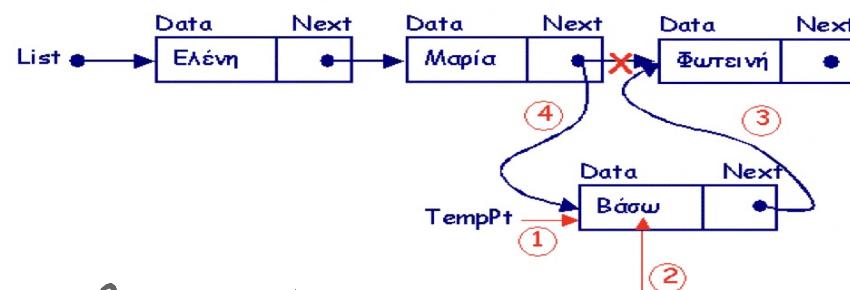
συνέδεσης και βάζω το Next

της νέας σε αυτό της εδώκε

Ο προηγούμενος

- (1) GetNode(TempP)
- (2) Data(TempP)='Βάσω'
- (3) Next(TempP)=Next(PredP)
- (4) Next(PredP)=TempP

PredPtr = Προηγ
του δεκ
-1



- Η διαγραφή αποτελείται από 2 περιπτώσεις

• Διαγραφή πρώτου συνέδεσμου

Αυτή η περιπτώση είναι η πιο εύκολη,

καθώς απλώς δίνουμε σαν ptr της

list τον Next του κορμού που διαγράφεται.

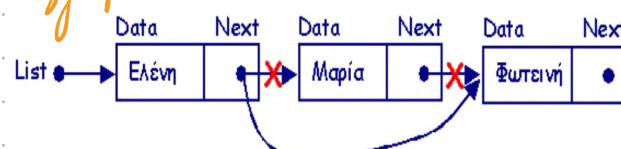
TempP=List
List=Next(List)
ReleaseNode(TempP)



• Διαγραφή συνέδεσμου που βρίσκεται μπροστά από άλλο συνέδεια

Σε αυτήν την περιπτώση το Next του προηγούμενου κορμού θα πάρει το Next αυτού του. Η διαγραφή

TempP=Next(PredP)
Next(PredP)=Next(TempP)
ReleaseNode(TempP)



Δυνατική Συνδεσμένη Λίστα

- Ισχει οτι ισχει σχηματικης συνδεσμενης λιστας αλλα μη λιστα ειναι δυνατη περιθωσ, δεν γερισει πιοτε
- Αντι για NilValue ελεγχω pe NULL
- Η προστελλομενη ολων των συστασιων

```
void LinkedTraverse(ListPointer List){  
    ListPointer CurrPtr;  
  
    if (EmptyList(List))  
        printf("EMPTY LIST\n");  
    else  
    {  
        CurrPtr = List;  
  
        while ( CurrPtr!=NULL )  
        {  
            printf("%d ",(*CurrPtr).Data);  
            CurrPtr = CurrPtr->Next;  
        }  
        printf("\n");  
    }  
}
```

Δυνατη Οργανωση - Συστασια

- Δεν υπαρχει οπιο
- Αντι για Top == -1 exw Top == NULL και αντι για Front == Rear exw Rear == NULL

Δυαδικό Δέντρο

- Αποτελειται απο Nodes (νόρβους) και Directed Arcs

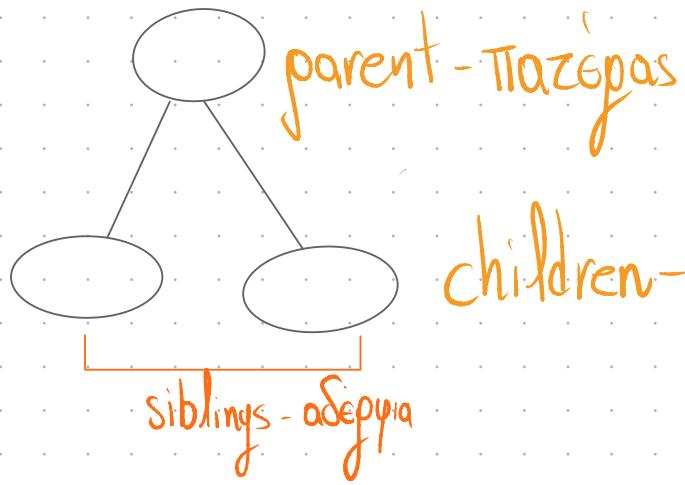
- Ειδη Nodes:

~ Root (Ριζα) → xwpis ειδερχορεια zōsa

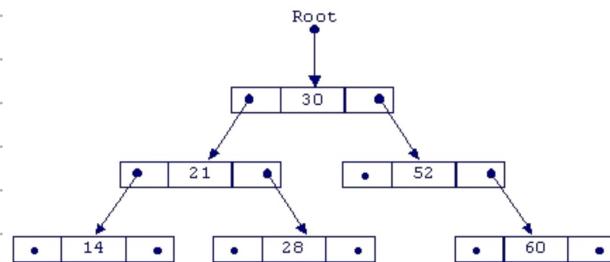
~ Leaves (γιλια) → xwpis ειδερχορεια zōsa

~ Internal Nodes (εσωτεριοι νόρβοι) → exouν και ta suo

κατεύθυνση
zōsa



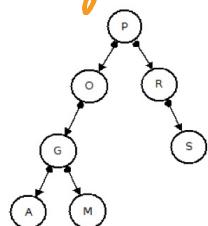
- Συνδεσμένη Δορή Δέντρου



- Αδειο Δυαδικού Δέντρου $\text{Root} == \text{NULL}$

- Προσπέλαση

Δέντρο



Αποστέλωρα

A G M O P R S

```

void InorderTraversal(BinTreePointer Root)
{
    // Προσπέλαση από αριστερά προς
    // τα δεξιά (αύξουσα σειρά)
    if (Root != NULL)
    {
        InorderTraversal(Root->LChild);
        printf("%c ", Root->Data);           // εναλλαγή των δυο
        InorderTraversal(Root->RChild);      // για ψθηνούσα
    }
}
  
```

Χαζακέρ πατιόνος

- Hash Table Length H_{Max}
- Hash Table List Length V_{Max}
- Hash Key $i \% H_{Max}$
- Τεχνικές Αποφυγής Συγκρούσεων

- ~ Αλυσιδώση Avixnevou - chaining
- ~ Ανοιξη Διεύθυνσιδόση - open addressing

Αλυσιδώση
Αρχή

HashTable	
0	-1
1	-1
2	-1
3	-1
4	-1

List		
key	Data	Link
0		1
1		2
2		3
3		4
4		5
5		6
6		7
7		8
8		9
9		10
10		11
11		12
12		13
13		14
14		-1

ΠΤΧ

$$H_{Max} = 7 \quad V_{Max} = 6$$

Tέλος

HashTable	
0	1
1	2
2	4
3	0
4	6

$$H_{Max} = 5$$

$$V_{Max} = 3$$

μεγιστα μεγιστα
υπολογισα

List		
key	Data	Link
0	23	7
1	40	5
2	71	3
3	86	-1
4	12	9
5	45	8
6	39	-1
7	68	10
8	30	-1
9	22	-1
10	3	-1
11		12
12		13
13		14
14		-1

Oι θεσεις των πρώτων εγγράφων αποθηκευονται στον HashTable
και περι λειτουργιαν σαν διορα μερι και βρει -1

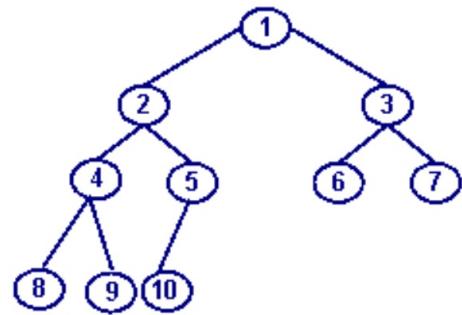
Εύπος - Heap

- Είναι ειδική περιπέτων $\Delta\Delta$

- Γεράζο $\Delta\Delta \rightarrow 2^{k+1} - 1$ nodes, k ύψος δευτερου

- Πληρες $\Delta\Delta \rightarrow$ καθε επίπεδο γεράζει (ιως ευρώς γελεγκανού)

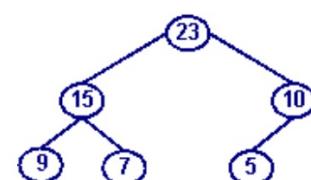
$$j = \text{νόρμα} \quad n = \text{Πλήρες νόρμων}$$



1. ο πατέρας του βρίσκεται στη θέση $j / 2$, αν $j \neq 1$.
Αν $j = 1$ τότε πρόκειται για τη ρίζα, η οποία φυσικά δεν έχει πατέρα.
2. το αριστερό παιδί του βρίσκεται στη θέσης 2^*j , αν $2^*j \leq n$.
Αν $2^*j > n$ τότε ο κόμβος j δεν έχει αριστερό παιδί.
3. το δεξί παιδί του βρίσκεται στη θέση 2^*j+1 , αν $2^*j+1 \leq n$.
Αν $2^*j+1 > n$ τότε ο κόμβος j δεν έχει δεξί παιδί.

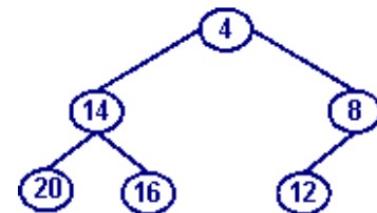
- Μεγιστο Λέντρο - Max Tree

- ~ Η ρίζα του δευτερου έχει το μεγαλύτερο συνοχείο του δευτερου
- ~ Καθε νόρμα πατέρας έχει μεγαλύτερο συνοχείο από τα παιδιά του



- Ελαχιστό Δέντρο - Min Tree

- ~ Η ρίζα του δέντρου έχει το μηδενικό συνοχείο του δέντρου
- ~ Κάθε κόρης πατέρας έχει μηδενικό συνοχείο από την πατέρα της



- Ελαχιστούς / Μεγιστούς Συνδέσμους

- ~ Είναι ένα τύπος ελαχιστού/μεγιστού ΔΔ

Κωδικοποίηση Huffman

- ~ χρησιμεύει όταν υψηλή χρήση → μηδενικούς γύκους κωδικούς

Χαρακτήρας	Βάρος
A	0.2
B	0.15
C	0.05
D	0.15
E	0.45

αυξουσα

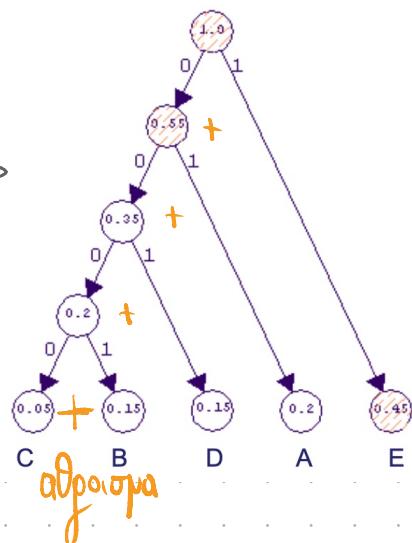
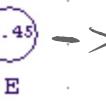


C

B

D

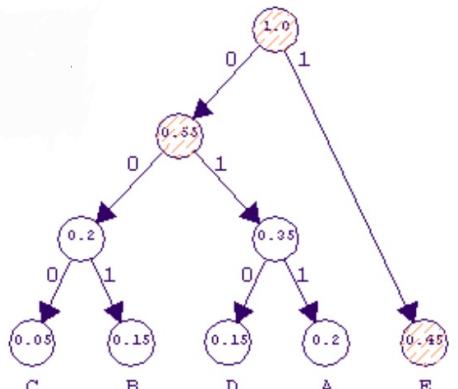
A



αρροφορα

→

Χαρακτήρας	Κωδικός Huffman
A	011
B	001
C	000
D	010
E	1

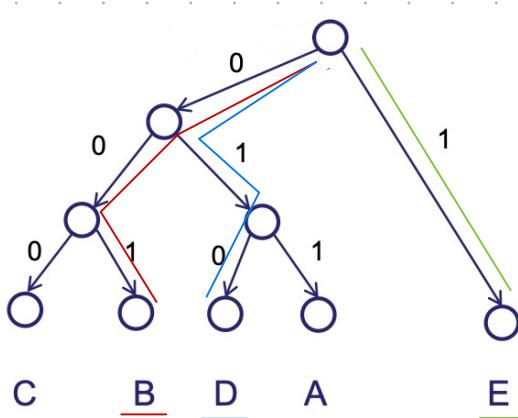


Άλλος τρόπος καζασκευής

Αποκωδικοποίηση Huffman

- Όποτε δρίσκων | συν κωδικό, πάω στο RChild. Ανα
έχει children εξετάζω το επόμενο σύριγχο, αλλιώς
πταιρώ το περιεχόμενο

- Όποτε δρίσκων 0 συν κωδικό, πάω LChild. Ανα
έχει children εξετάζω το επόμενο σύριγχο, αλλιώς
πταιρώ το περιεχόμενο.



Κωδικός

0	0	1	1	0	1	0
B	E		D			

B
E
D
B
E
D