ΤΜΗΜΑ ΕΦΑΡΜΟΣΜΕΝΗΣ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΑΝΑΛΥΣΗ ΑΛΓΟΡΙΘΜΩΝ

Εξαντλητική αναζήτηση

Κωνσταντίνος Γιαννουτάκης
Επίκουρος Καθηγητής

ΣΥΝΟΨΗ ΔΙΑΛΕΞΗΣ

- Εξαντλητική Αναζήτηση

ΕΞΑΝΤΛΗΤΙΚΗ ΑΝΑΖΗΤΗΣΗ

- *Πρόβλημα Πλανόδιου Πωλητή*
- *Πρόβλημα του Σακιδίου*
- *Πρόβλημα της Ανάθεσης*

ΕΞΑΝΤΛΗΤΙΚΗ ΑΝΑΖΗΤΗΣΗ (EXHAUSTIVE SEARCH)

Εξαντλητική Αναζήτηση

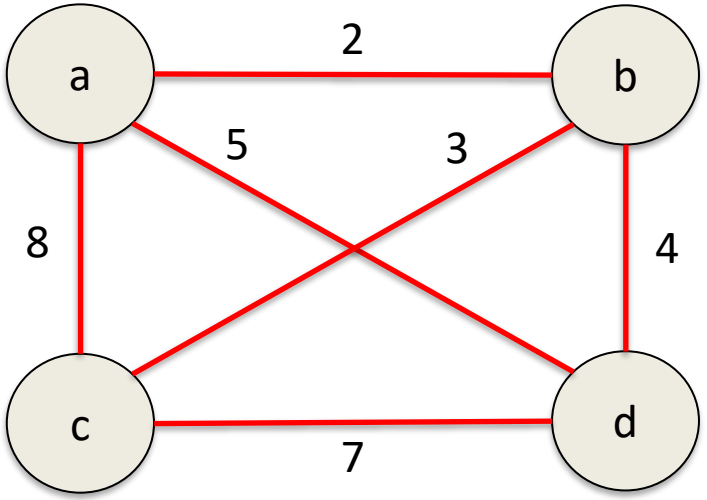
είναι μία **μέθοδος ωμής βίας** όπου **αναζητούμε ένα στοιχείο με συγκεκριμένη ιδιότητα**, συχνά μεταξύ συνδυαστικών αντικειμένων όπως συνδυασμούς, διατάξεις ή υποσύνολα συνόλου

Μεθοδολογία

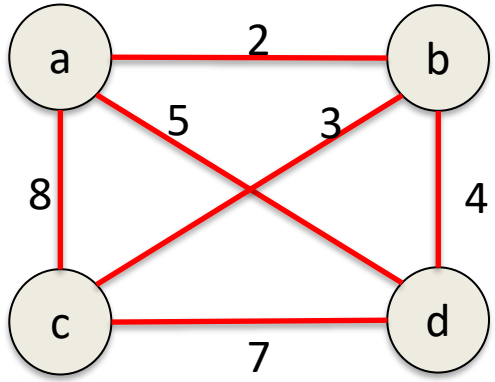
1. **Κατασκευάζουμε μία λίστα όλων των δυνατών λύσεων** στο πρόβλημα με ένα συστηματικό τρόπο
   a. Παρουσιάζονται όλες οι λύσεις
   b. Καμία λύση δεν επαναλαμβάνεται
2. **Εξετάζουμε τις λύσεις μία προς μία**, απορρίπτοντας τις μη ικανοποιητικές και κρατώντας τη μέχρι στιγμής καλύτερη
3. **Όταν τελειώσει η αναζήτηση, ανακοινώνεται ο νικητής**

ΠΡΟΒΛΗΜΑ ΠΛΑΝΟΔΙΟΥ ΠΩΛΗΤΗ (TRAVELING SALESMAN PROBLEM)

Δοσμένων ***n* πόλεων** με γνωστές μεταξύ τους αποστάσεις, να βρεθεί **η συντομότερη διαδρομή που περνά από όλες τις πόλεις ακριβώς μία φορά** πριν επιστρέψει στην αφετηρία.



ΠΡΟΒΛΗΜΑ ΠΛΑΝΟΔΙΟΥ ΠΩΛΗΤΗ



	Διαδρομή	Κόστος
1	a→b→c→d→a	2+3+7+5=**17**
2	a→b→d→c→a	2+4+7+8=21
3	a→c→b→d→a	8+3+4+5=20
4	a→c→d→b→a	8+7+4+2=21
5	a→d→b→c→a	5+4+3+8=20
6	a→d→c→b→a	5+7+3+2=**17**

Ποιο είναι το πλήθος όλων των δυνατών διαδρομών (σε σχέση με το πλήθος των κόμβων);

ΠΡΟΒΛΗΜΑ ΠΛΑΝΟΔΙΟΥ ΠΩΛΗΤΗ

1. Θεωρούμε την πόλη 1 ως το αρχικό και τελικό σημείο.
2. Δημιουργούμε όλες τις (n-1)! παραλλαγές.
3. Υπολογίζουμε το κόστος κάθε παραλλαγής και αποθηκεύουμε το ελάχιστο κόστος.
4. Επιστρέφουμε την παραλλαγή με το ελάχιστο κόστος.

Πολυπλοκότητα
$\boldsymbol{\mathcal{O}(n!)}$

ΠΡΟΒΛΗΜΑ ΤΟΥ ΣΑΚΙΔΙΟΥ (KNAPSACK PROBLEM)

Δοσμένων ***n*** **αντικειμένων** με **βάρη** $w_1, w_2, \ldots, w_n$ και **αξίες** $v_1, v_2, \ldots, v_n$ και **ενός σάκου χωρητικότητας *W***, να βρεθεί το πολυτιμότερο υποσύνολο των αντικειμένων που ταιριάζουν στο σάκο.

Χωρητικότητα *W*=16		
Είδος	Βάρος	Αξία (€)
0	2	20
1	5	30
2	10	50
3	5	10

ΠΡΟΒΛΗΜΑ ΤΟΥ ΣΑΚΙΔΙΟΥ

Χωρητικότητα *W*=16		
Είδος	Βάρος	Αξία (€)
0	2	20
1	5	30
2	10	50
3	5	10

Ποιο είναι το πλήθος όλων των δυνατών συνδυασμών αντικειμένων (σε σχέση με το πλήθος των αντικειμένων);

Υποσύνολο	Συνολικό Βάρος	Συνολική Αξία (€)
{0}	2	20
{1}	5	30
{2}	10	50
{3}	5	10
{0,1}	7	50
{0,2}	12	70
{0,3}	7	30
{1,2}	**15**	**80**
{1,3}	10	40
{2,3}	15	60
{0,1,2}	17	Μη εφικτό
{0,1,3}	12	60
{0,2,3}	17	Μη εφικτό
{1,2,3}	20	Μη εφικτό
{0,1,2,3}	22	Μη εφικτό

ΚΩΔΙΚΑΣ C

Είσοδος:

- Η χωρητικότητα του σακιδίου (W)
- Τα βάρη των αντικειμένων (πίνακας wt)
- Η αξία των αντικειμένων (πίνακας val)
- Το πλήθος των αντικειμένων (n)

Έξοδος:

- Η αξία των αντικειμένων στην καλύτερη περίπτωση

```
int knapSack(int W, int wt[], int val[], int n)
{
    // Βασική περίπτωση
    if (n == 0 || W <= 0)
        return 0;

    // Εάν το βάρος του n-οστού αντικειμένου είναι μεγαλύτερο
    // από τη χωρητικότητα W, τότε το αντικείμενο δεν μπορεί
    // να συμπεριληφθεί στην καλύτερη λύση
    if (wt[n - 1] > W)
        return knapSack(W, wt, val, n - 1);

    // Επέστρεψε το μεγαλύτερο των 2 περιπτώσεων:
    // (1) συμπεριλαμβανομένου του n-οστού αντικειμένου
    // (2) χωρίς το n-οστό αντικείμενο
    else
        return max(val[n - 1] + knapSack(W - wt[n - 1],
                                         wt, val, n - 1),
                   knapSack(W, wt, val, n - 1));
}
```

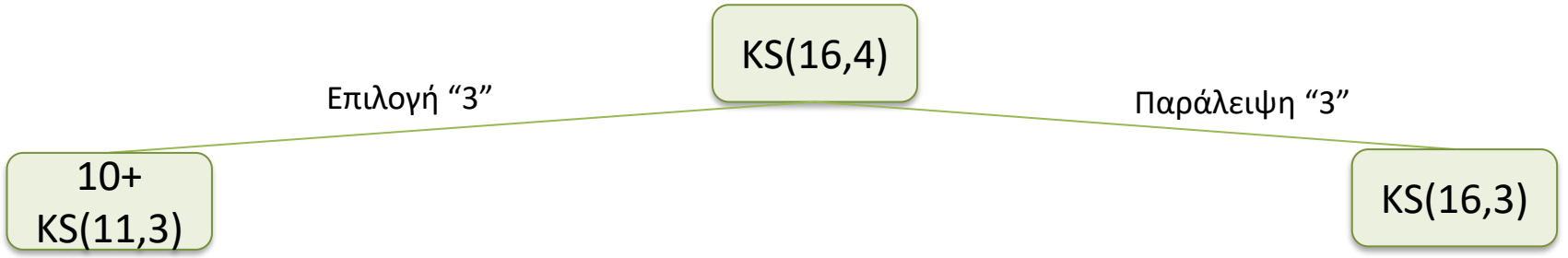
ΔΕΝΔΡΟ ΑΝΑΔΡΟΜΗΣ

KS(16,4)

W=16		
Είδος	wt	val
0	2	20
1	5	30
2	10	50
3	**5**	**10**

```
return max(val[n - 1] + knapSack(W - wt[n - 1],wt, val, n - 1), knapSack(W, wt, val, n - 1));
```

ΔΕΝΔΡΟ ΑΝΑΔΡΟΜΗΣ



W=16		
Είδος	wt	val
0	2	20
1	5	30
2	10	50
3	**5**	**10**

```
return max(val[n - 1] + knapSack(W - wt[n - 1],wt, val, n - 1), knapSack(W, wt, val, n - 1));
```

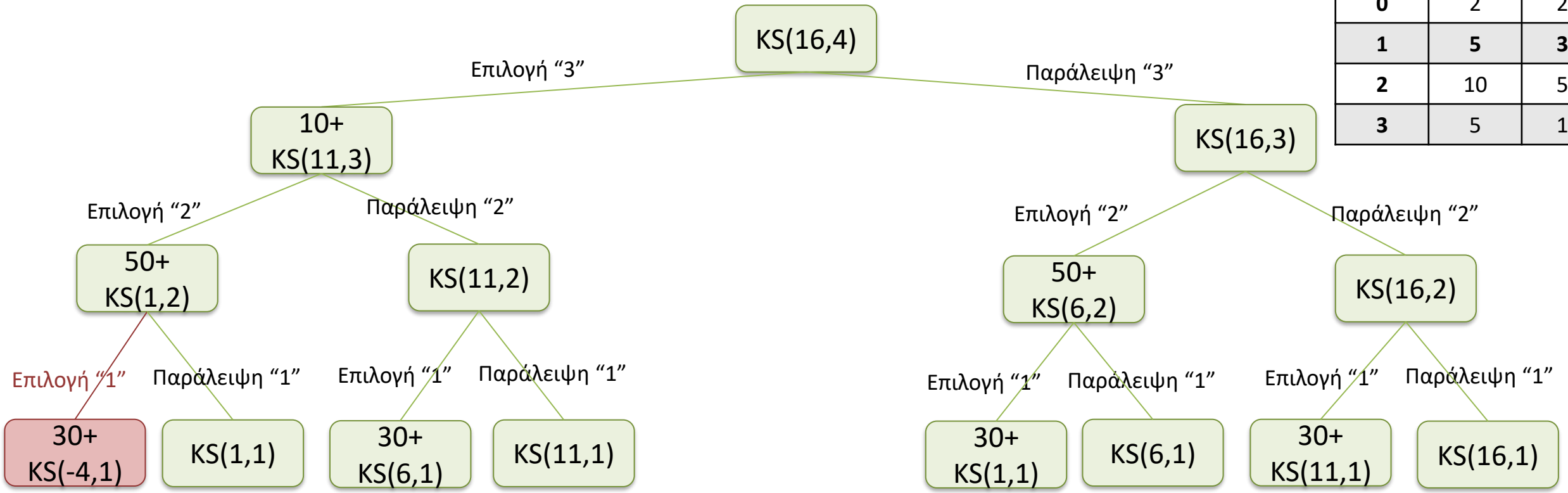
ΔΕΝΔΡΟ ΑΝΑΔΡΟΜΗΣ

W=16		
Είδος	wt	val
0	2	20
1	5	30
2	**10**	**50**
3	5	10


```
return max(val[n - 1] + knapSack(W - wt[n - 1],wt, val, n - 1), knapSack(W, wt, val, n - 1));
```

ΔΕΝΔΡΟ ΑΝΑΔΡΟΜΗΣ

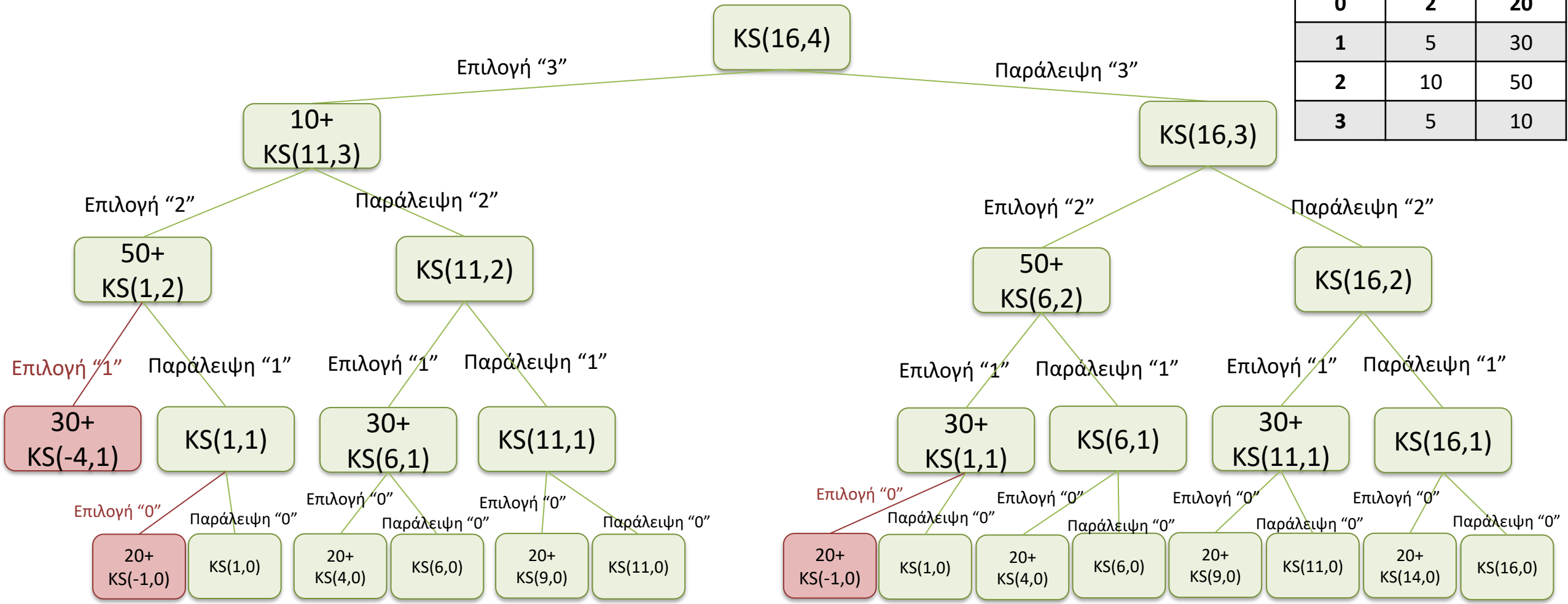
W=16		
Είδος	wt	val
0	2	20
1	**5**	**30**
2	10	50
3	5	10




```
return max(val[n - 1] + knapSack(W - wt[n - 1],wt, val, n - 1), knapSack(W, wt, val, n - 1));
```

ΔΕΝΔΡΟ ΑΝΑΔΡΟΜΗΣ

W=16		
Είδος	wt	val
0	**2**	**20**
1	5	30
2	10	50
3	5	10

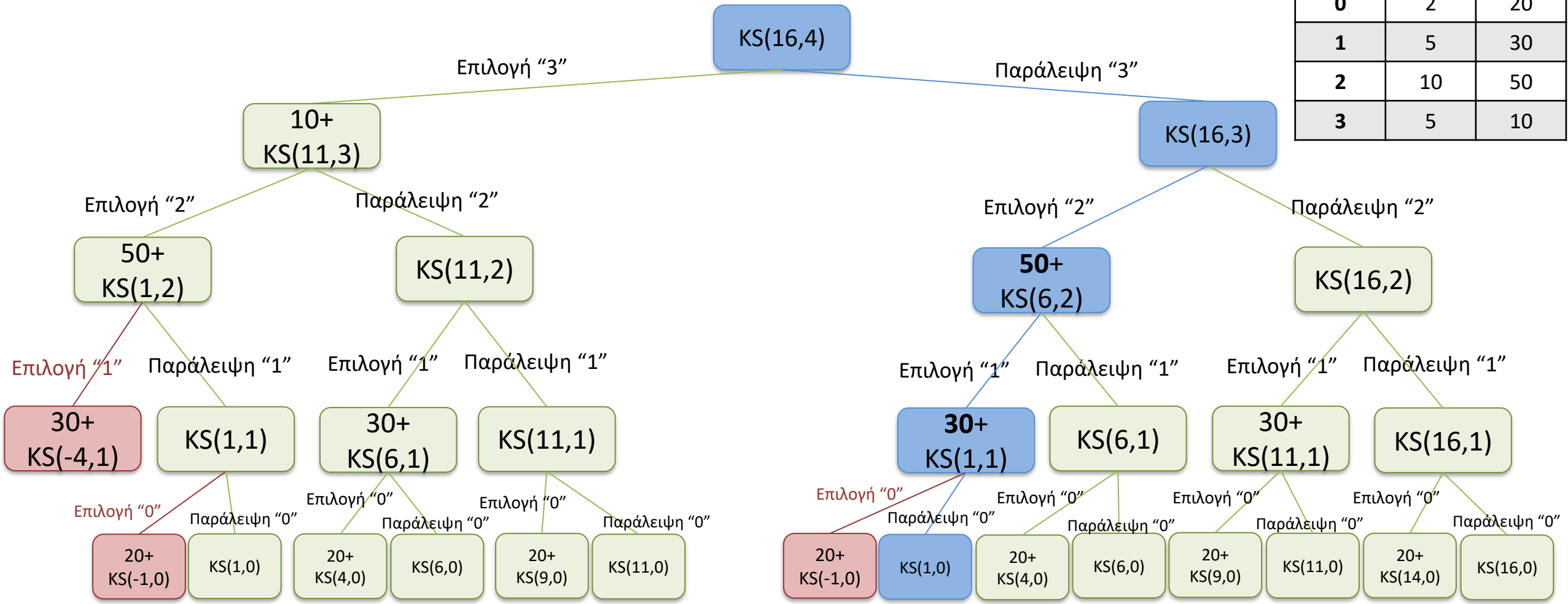



```
return max(val[n - 1] + knapSack(W - wt[n - 1],wt, val, n - 1), knapSack(W, wt, val, n - 1));
```

ΔΕΝΔΡΟ ΑΝΑΔΡΟΜΗΣ

Πολυπλοκότητα

W=16		
Είδος	wt	val
0	2	20
1	5	30
2	10	50
3	5	10

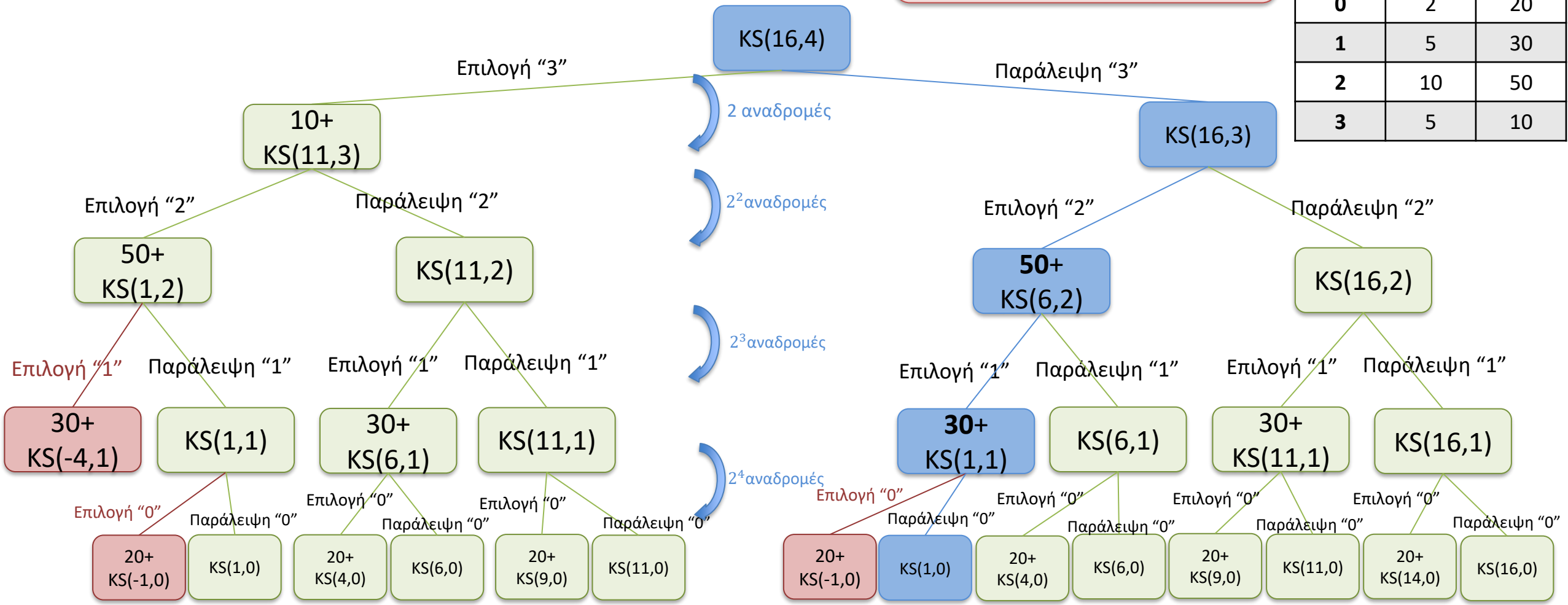



```
return max(val[n - 1] + knapSack(W - wt[n - 1],wt, val, n - 1), knapSack(W, wt, val, n - 1));
```

ΔΕΝΔΡΟ ΑΝΑΔΡΟΜΗΣ

Πολυπλοκότητα: $\mathcal{O}(2^n)$

W=16		
Είδος	wt	val
0	2	20
1	5	30
2	10	50
3	5	10




```
return max(val[n - 1] + knapSack(W - wt[n - 1],wt, val, n - 1), knapSack(W, wt, val, n - 1));
```

ΠΡΟΒΛΗΜΑ ΤΗΣ ΑΝΑΘΕΣΗΣ (ASSIGNMENT PROBLEM)

Δοσμένων ***n* ατόμων** και ***n* εργασιών**, και ***C*[*i*, *j*]** να είναι το **κόστος** αν το άτομο *i* αναλάβει την εργασία *j*, να ανατεθεί ένα άτομο σε κάθε εργασία με συνολικό ελάχιστο κόστος.

	Εργασία 1	Εργασία 2	Εργασία 3	Εργασία 4
Άτομο 1	9	2	7	8
Άτομο 2	6	4	3	7
Άτομο 3	5	8	1	8
Άτομο 4	7	6	9	4

ΠΡΟΒΛΗΜΑ ΤΗΣ ΑΝΑΘΕΣΗΣ

Ανάθεση	*Συνολικό κόστος*
<1,2,3,4>	9+4+1+4=18
<1,2,4,3>	9+4+8+9=30
<1,3,2,4>	9+3+8+4=24
<1,3,4,2>	9+3+8+6=26
<1,4,2,3>	9+7+8+9=33
<1,4,3,2>	9+7+1+6=23
.....	.....

	Εργασία 1	Εργασία 2	Εργασία 3	Εργασία 4
Άτομο 1	9	2	7	8
Άτομο 2	6	4	3	7
Άτομο 3	5	8	1	8
Άτομο 4	7	6	9	4

Πόσους δυνατούς συνδυασμούς μπορούμε να έχουμε;

ΠΡΟΒΛΗΜΑ ΤΗΣ ΑΝΑΘΕΣΗΣ

Ανάθεση	*Συνολικό κόστος*
<1,2,3,4>	9+4+1+4=18
<1,2,4,3>	9+4+8+9=30
<1,3,2,4>	9+3+8+4=24
<1,3,4,2>	9+3+8+6=26
<1,4,2,3>	9+7+8+9=33
<1,4,3,2>	9+7+1+6=23
.....	.....

	Εργασία 1	Εργασία 2	Εργασία 3	Εργασία 4
Άτομο 1	9	2	7	8
Άτομο 2	6	4	3	7
Άτομο 3	5	8	1	8
Άτομο 4	7	6	9	4

Πολυπλοκότητα:
$\mathcal{O}(n!)$

ΣΥΜΠΕΡΑΣΜΑΤΑ

- Οι αλγόριθμοι εξαντλητικής αναζήτησης **εκτελούνται σε ρεαλιστικούς χρόνους μόνο για μικρά στιγμιότυπα**
- Κάποιες φορές υπάρχουν καλύτερες εναλλακτικές λύσεις
  - Συντομότερα μονοπάτια
  - Ελάχιστα συνδετικά δένδρα
  - Πρόβλημα ανάθεσης
- **Μερικές φορές όμως η εξαντλητική αναζήτηση** (ή κάποια παραλλαγή) **είναι η μοναδική γνωστή λύση**

Κωνσταντίνος Γιαννουτάκης

Επίκ. Καθηγητής

kgiannou@uom.edu.gr