Αλγόριθμοι Brute-Force και Διεξοδική Αναζήτηση

Περίληψη

- Αλγόριθμοι τύπου Brute-Force
- Παραδείγματα
  - Αναζήτησης
  - Ταξινόμησης
  - Πλησιέστερα σημεία
  - Convex hull
  - Βελτιστοποίηση
  - Knapsack problem
  - Προβλήματα Ανάθεσης (Assignment)

Αλγόριθμοι Brute-Force

- Σε αυτή την κατηγορία εμπίπτουν όλοι οι αλγόριθμοι οι οποίοι πηγάζουν απευθείας από τη διατύπωση του προβλήματος και τον ορισμό των διαφόρων «εννοιών» που συνεπάγονται από τη φύση του προβλήματος.
- Πλέον Γενική Μέθοδος:
  - Τα πλείστα προβλήματα με λύση έχουν μια λύση που εμπίπτει σε αυτή την κατηγορία
- Μπορεί να είναι πολύ καλή και πρακτική μέθοδος για προβλήματα που η είσοδος είναι μικρού ή μεσαίου μεγέθους.
- Συνήθως η λύση που προκύπτει χρησιμοποιώντας αυτή τη μέθοδο **δεν είναι αποδοτική**

Προβλήματα Αναζήτησης

- Είσοδος: Λίστα $A[0,\ldots,n-1]$, Κλειδί K
- Έξοδος: Η θέση του κλειδιού στη λίστα (εάν υπάρχει)

```
Search(A[], K)
  i=0
  While i<n and A[i]!= K
     i=i+1;
   If i<n return i;
   return -1;
```

```
Search2(A[], K)
  i=0
  A[n]= K;
  While A[i]!= K
     i=i+1;
   If i<n return i;
   return -1;
```

- **Καλύτερη Περίπτωση:**
- **Χειρότερη Περίπτωση:**

Προβλήματα Ταξινόμησης

- Είσοδος: Λίστα $A[0,\ldots,n-1]$
- Έξοδος: Ταξινομημένη λίστα $A[0,\ldots,n-1]$

```
sort(A[])
  for i=0 to n-2
      min= i;
      for j=i+1 to n-1
          if A[j] < A[min]
              min= j;
      swap(A[i],A[min]);
```

Selection Sort

```
sort(A[0…n-1])
    for i=0 to n-2
        min= i;
        for j=i+1 to n-1
            if A[j] < A[min]
                min= j;
        swap(A[i],A[min]);
```

89	45	68	90	29	34	17

17	**29**	68	90	45	34	89
17	**29**	**34**	90	45	68	89
17	**29**	**34**	**45**	90	68	89
17	**29**	**34**	**45**	**68**	90	89
17	**29**	**34**	**45**	**68**	**89**	**90**

Απόδοση Selection Sort

```
sort(A[0…n-1])
   for i=0 to n-2
       min= i;
       for j=i+1 to n-1
          if A[j] < A[min]
              min= j;
       swap(A[i],A[min]);
```

- **Τι να υπολογίσουμε;**
  - **Καλύτερη, χειρότερη ή μέση περίπτωση;**

Αναζήτηση σειράς χαρακτήρων (string)

- Είσοδος: Λίστες $T[0,\ldots,n\text{-}1]$, $s[0,\ldots,k\text{-}1]$, $k<n$
- Έξοδος: Θέση στην οποία εμφανίζεται το s στην T

```
stringMatch(T[0…n-1],s[0,…,k-1])
  for i=0 to n-k
      j=0;
      while (j<k && s[j]=T[i+j]) j=j+1;
      if j=k return i;
  return -1;
```

- **Χειρότερη περίπτωση;**

Πρόβλημα εύρεσης των δύο πλησιέστερων σημείων

- Δεδομένης μιας λίστας με n σημεία $\{P_0,\ldots,P_{n-1}\}$ βρείτε ποια δύο σημεία έχουν την μικρότερη απόσταση.
- **Απόσταση** μεταξύ δύο σημείων σε χώρο D διαστάσεων:

$$d\left(P_a,P_b\right)=\sqrt{\sum_{i=1}^{D}\left(x_{a,i}-x_{b,i}\right)^2}$$

- Για παράδειγμα σε δυσδιάστατο χώρο,

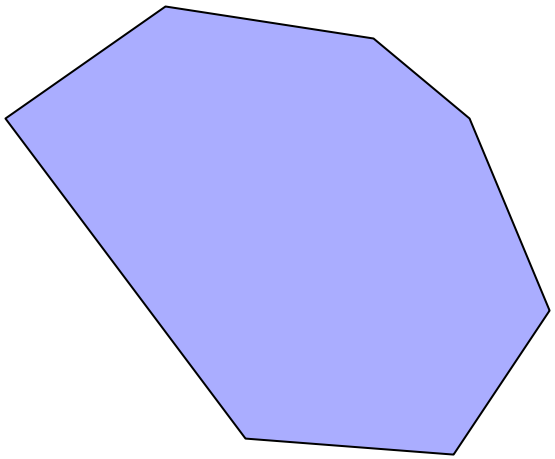
$$d\left(P_a,P_b\right)=\sqrt{\left(x_{a,1}-x_{b,1}\right)^2+\left(x_{a,2}-x_{b,2}\right)^2}=\sqrt{\left(x_a-x_b\right)^2+\left(y_a-y_b\right)^2}$$

Πλησιέστερα Σημεία

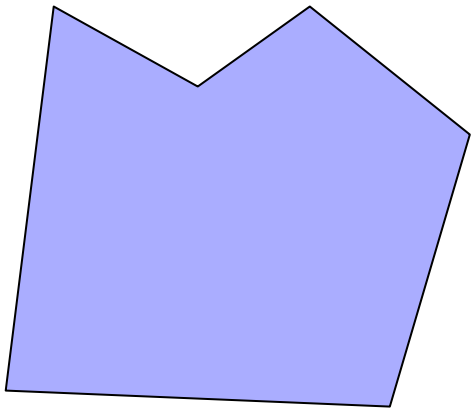
```
ClosestPoints(P[0…n-1])
  dmin= infinity;
  for i=0 to n-2
      for j=i+1 to n-1
       d= sqrt((xi-xj)2+(yi-yj)2);
       if d < dmin
              dmin= d;
              p1=i, p2=j;
  return p1, p2;
```

Πρόβλημα εύρεσης του κυρτού χώρου που εμπερικλείει ένα σύνολο από σημεία

- Κυρτός (convex) χώρος:
  - Ένας χώρος είναι κυρτός εάν για κάθε δύο σημεία που ανήκουν στον χώρο, όλα τα σημεία που βρίσκονται στο τμήμα της ευθείας που ενώνει τα δύο σημεία ανήκει επίσης στον χώρο
- Για παράδειγμα σε δυσδιάστατο χώρο,



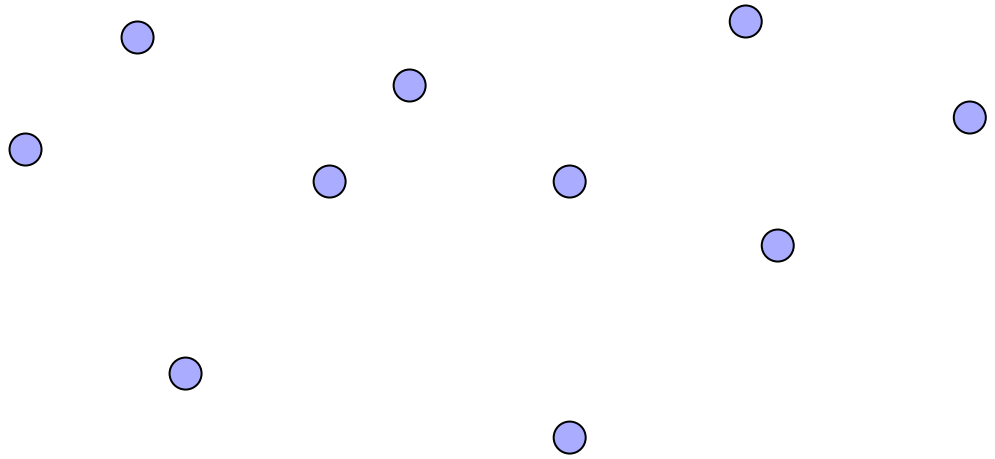
Κυρτός χώρος



Μη Κυρτός χώρος

Πρόβλημα εύρεσης του κυρτού χώρου που εμπερικλείει ένα σύνολο από σημεία

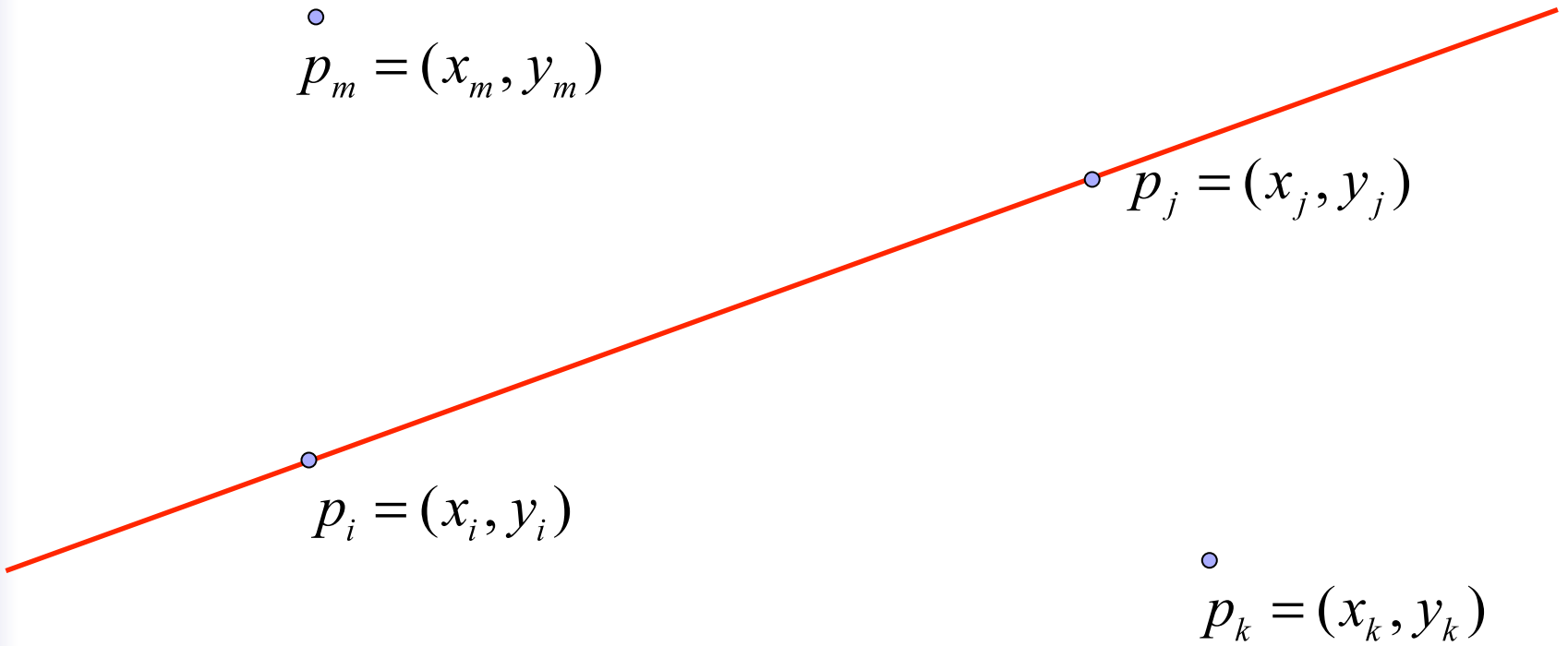
- Δεδομένης μιας λίστας με n σημεία $\{P_1,\ldots,P_n\}$ βρείτε το **μικρότερο** κυρτό χώρο που εμπερικλείει όλα τα σημεία $\{P_1,\ldots,P_n\}$ (**convex hull**).
- **Θεώρημα:** Το convex hull ενός συνόλου από σημεία $\{P_1,\ldots,P_n\}$ ($n>2$) είναι ένα πολύγωνο του οποίου οι γωνίες είναι κάποιο από τα σημεία του συνόλου $\{P_1,\ldots,P_n\}$.
- Για παράδειγμα σε δυσδιάστατο χώρο,



Convex Hull

```
ConvexHall(P[1…n])
  for i=1 to n-1
     for j=i+1 to n
       findLine(Pi,Pj);
       k=1; P=getNextPoint;
       while (k<n-2) and sameSide(P);
          k=k+1;
          P=getNextPoint;
       if k=n-2 (Pi,Pj) on boundary;
```

Γεωμετρία



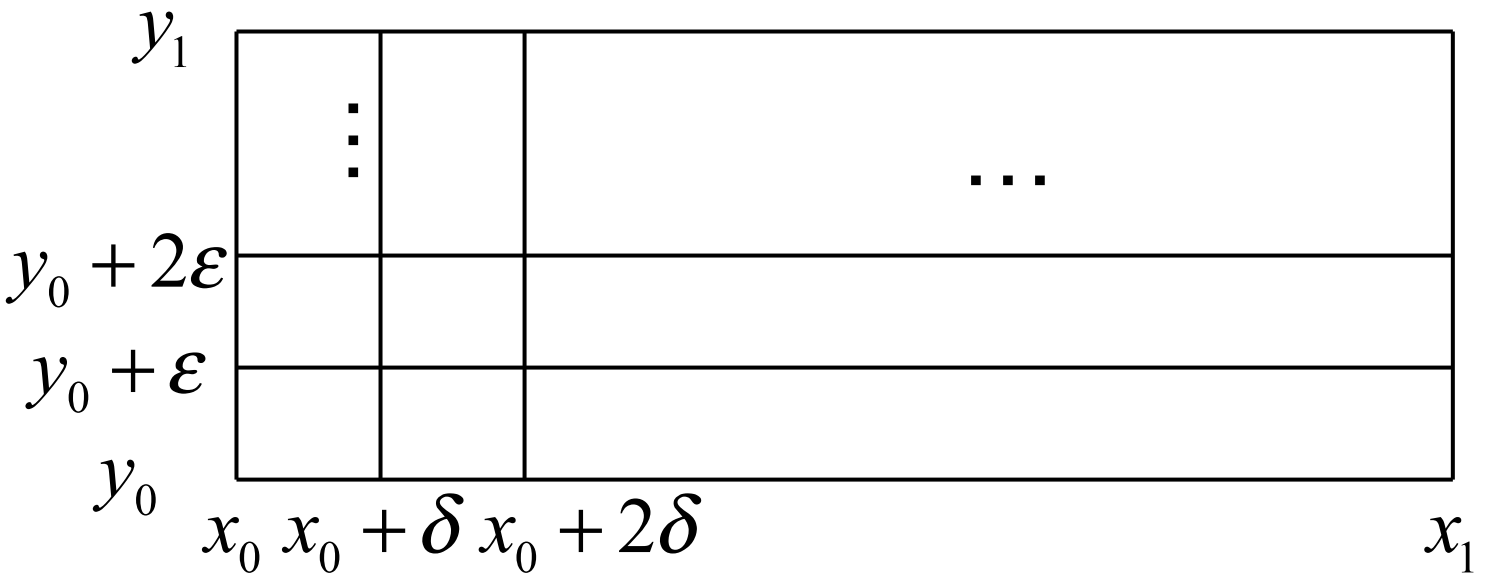
Βελτιστοποίηση

- Λύστε το πιο κάτω πρόβλημα βελτιστοποίησης

$$\min_{x,y} f(x,y)$$

$$\text{s.t.} \quad x_0 \le x \le x_1$$

$$y_0 \le y \le y_1$$

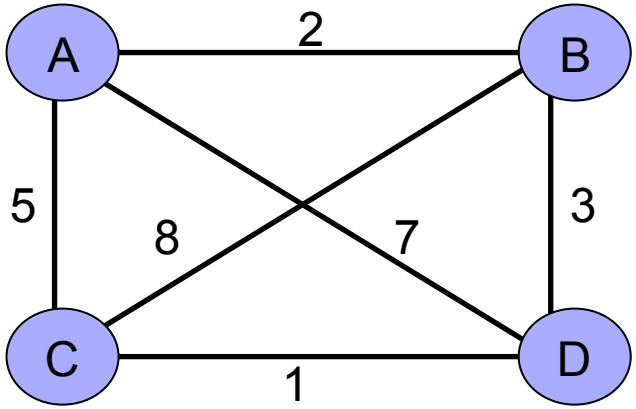


Βελτιστοποίηση

```
minimize(f,x0,x1,y0,y1,n)
  d=(x1-x0)/(n-1);
  e=(y1-y0)/(n-1);
  min= infinity;
  for i=0 to n-1
     for j=0 to n-1
       if f(x0+i*d, y0+j*e)<min;
          min= f(x0+i*d, y0+j*e);
          xm=x0+i*d; ym= y0+j*e;
  return min, xm, ym;
```

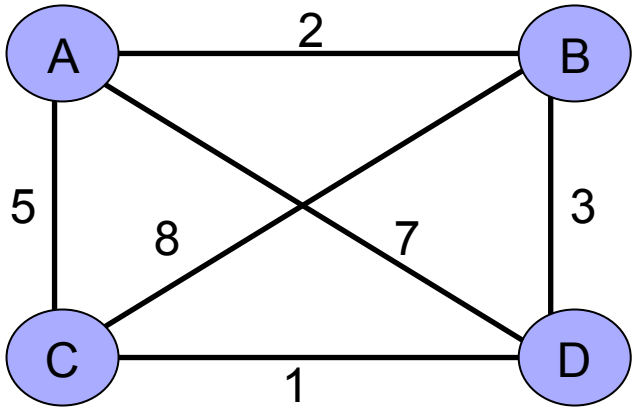
Πρόβλημα του Περιοδεύων Πωλητή (Traveling Salesman Problem (TSP))

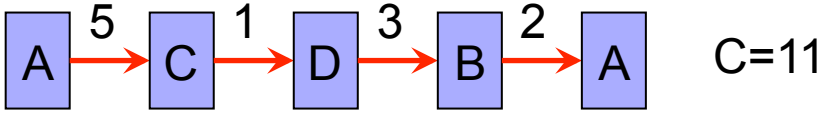
- Ένας πωλητής θέλει να επισκεφτεί n πόλεις και να επιστρέψει στο σημείο από όπου ξεκίνησε με το πιο «γρήγορο» τρόπο.
  - Δεν πρέπει να επισκεφτεί μια πόλη περισσότερες από μια φορές
- Βρείτε το πιο σύντομο **Hamiltonian Circuit.**
  - Hamiltonian Circuit: είναι ένα μονοπάτι το οποίο επισκέπτεται όλους τους κόμβους του γράφου από μία φορά και επιστρέφει στο σημείο από όπου ξεκίνησε.



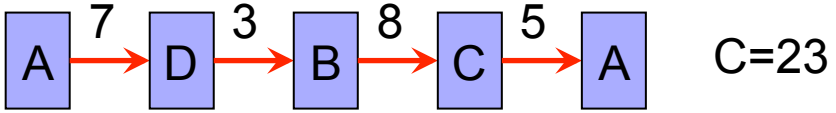
Πρόβλημα του Περιοδεύων Πωλητή (Traveling Salesman Problem (TSP))

- Βρέστε όλα τα πιθανά μονοπάτια
- Κρατήστε εκείνο με το χαμηλότερο κόστος

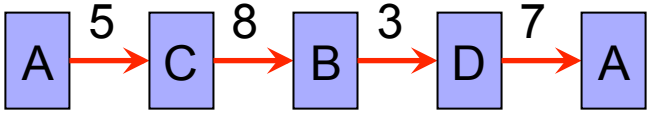




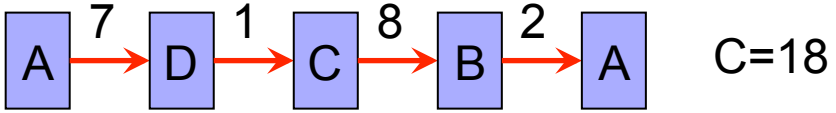
C=11



C=23



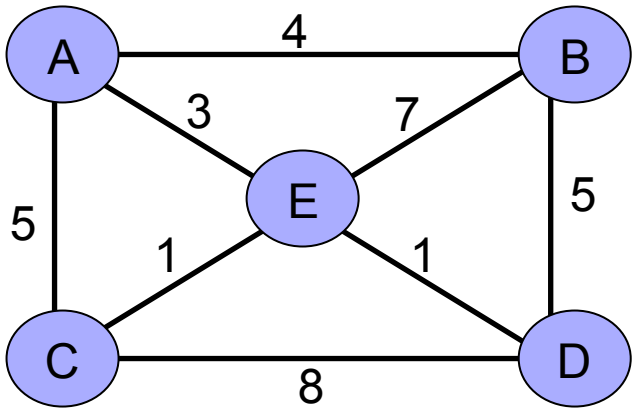
C=23



C=18

Πρόβλημα του Περιοδεύων Πωλητή (Traveling Salesman Problem (TSP))

- Πιθανή λύση: Πλεονεκτικός (greedy) αλγόριθμος
  - Σε κάθε κόμβο επιλέγω την ακμή με το μικρότερο κόστος!



- Στη γενική περίπτωση, ο αλγόριθμος **ΔΕΝ** βρίσκει τη βέλτιστη λύση.

Knapsack Problem

- Ένας ορειβάτης προγραμματίζει την επόμενη του περιπέτεια. Στο σακίδιο του μπορεί να μεταφέρει μέχρι n κιλά.
- Υπάρχουν m αντικείμενα τα οποία μπορεί να του χρησιμεύσουν
  - Κάθε αντικείμενο ζυγίζει w_i κιλά
  - Η «χρησιμότητα» του κάθε αντικειμένου δίνεται από ένα δείκτη y_i.
- Βοηθήστε τον ορειβάτη να βρει πώς να γεμίσει το σακίδιο του έτσι ώστε να μεγιστοποιήσει τη συνολική «χρησιμότητα» όλων των αντικειμένων που θα πάρει μαζί του.
- Παραδείγματα στα οποία το πιο πάνω πρόβλημα εφαρμόζει

Knapsack Problem: Παράδειγμα

- Το σακίδιο μπορεί να χωρέσει 10 κιλά
- Υπάρχουν 4 αντικείμενα
  1. Βάρος= 7, χρησιμότητα= 42
  2. Βάρος= 3, χρησιμότητα= 12
  3. Βάρος= 4, χρησιμότητα= 40
  4. Βάρος= 5, χρησιμότητα= 25
- Όλοι οι πιθανοί συνδυασμοί:

Συνδυασμός	**Βάρος**	**Τιμή**	**Συνδυασμός**	**Βάρος**	**Τιμή**
{Άδειο}	0	0	{1,4}	12	-
{1}	7	42	{2,3}	7	52
{2}	3	12	{2,4}	8	37
{3}	4	40	{3,4}	9	65
{4}	5	25	{1,2,3}, {1,2,4}		-
{1,2}	10	54	{1,3,4}, {2,3,4}		-
{1,3}	11	-	{1,2,3,4}		-

Πρόβλημα Ανάθεσης (Assignment)

- Υπάρχουν n φοιτητές και n προβλήματα. Για την καλύτερη κατανομή του φόρτου εργασίας ο κάθε φοιτητής θα λύσει **ένα** από τα προβλήματα.
- Εάν ο φοιτητής i επιχειρήσει το πρόβλημα j τότε θα πάρει b_{ij} βαθμούς
- Πως πρέπει να κατανεμηθούν τα προβλήματα έτσι ώστε να μεγιστοποιηθεί ο βαθμός ολόκληρης της τάξης;
- Άλλα παραδείγματα στα οποία εφαρμόζει αυτό το πρόβλημα

Παράδειγμα Προβλήματος Ανάθεσης

- Υποθέστε πως υπάρχουν 4 φοιτητές και 4 προβλήματα και οι βαθμοί φαίνονται πιο κάτω.

	Φοιτ. 1	Φοιτ. 2	Φοιτ. 3	Φοιτ. 4
Πρόβ.1	10	0	5	5
Πρόβ.2	9	7	3	8
Πρόβ.3	8	0	0	0
Πρόβ.4	9	10	8	0

- Το διάνυσμα $[\varphi_1, \varphi_2, \varphi_3, \varphi_4]$ αντιπροσωπεύει το φοιτητή που θα λύσει το πρόβλημα 1, 2, 3, και 4 αντίστοιχα
- Βρείτε όλες τις πιθανές αναθέσεις (permutations) και υπολογίστε το συνολικό βαθμό
- Πόσες οι πιθανές αναθέσεις υπάρχουν;