ΕΙΣΑΓΩΓΗ ΣΤΗΝ ΕΠΙΣΤΗΜΗ ΤΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

http://eclass.aueb.gr/courses/INF511/

Θεωρία υπολογισμού (ΚΕΦΑΛΑΙΟ 12)

Αλκμήνη Σγουρίτσα

Κοδριγκτώνος 12, 2ος όροφος

E-mail: alkmini@aueb.gr

ΚΕΦΑΛΑΙΟ 12: Θεωρία υπολογισμού

- Έννοια συναρτήσεων στη Θεωρία Υπολογισμού
- Υπολογισιμότητα
- Μηχανές Turing: ορισμός, παραδείγματα
- Υπόθεση Church-Turing
- Στοιχειώδης γλώσσα προγραμματισμού
- Μη υπολογίσιμη συνάρτηση
- Πολυπλοκότητα προβλημάτων
  - Προβλήματα κλάσης P, NP, NP-Complete

Θεωρία Υπολογισμού

- Τι μπορούν και τι δε μπορούν να κάνουν οι υπολογιστές;
- Που βρίσκεται το σύνορο μεταξύ επιλύσιμων ή μη προβλημάτων από έναν υπολογιστή;
  - Ορισμός του συνόρου μέσω ορισμού απλών μηχανών (Turing)
- Μελέτη προβλημάτων των οποίων η λύση βρίσκεται πέρα από τις δυνατότητες των αλγορίθμων και των σημερινών (και αυριανών) υπολογιστών
  - Μεταξύ των προβλημάτων που είναι επιλύσιμα, υπάρχουν προβλήματα των οποίων η λύση είναι **τόσο περίπλοκη που πρακτικά θεωρούνται άλυτα**

Συναρτήσεις

- Μία **συνάρτηση** είναι μία αντιστοιχία/απεικόνιση (mapping) μεταξύ ενός συνόλου πιθανών τιμών “εισόδου” και ενός συνόλου τιμών “εξόδου”, έτσι ώστε κάθε πιθανή είσοδος να αντιστοιχεί **σε μία και μοναδική** έξοδο
  - Π.χ. πρόσθεση 2 αριθμών, ταξινόμηση λίστας...

Παραδείγματα συναρτήσεων

Γυάρδες (είσοδος)	Μέτρα (έξοδος)
1	0,9144
2	1,8288
3	2,7432
4	3,6576
5	4,5720
.	.
.	.
.	.

$V(r)=P(1+r)^3$

$$W(x) = \begin{cases} x \text{ if } x<0 \\ x^2 \text{ otherwise} \end{cases}$$

$\Pi(x,y) = x+y$

Μη ταξινομημένος πίνακας → ταξινομημένος πίνακας

Υπολογισιμότητα (1)

- Υπάρχουν προβλήματα που δεν μπορούν να λυθούν από ένα υπολογιστή; ΝΑΙ!
- Αρχική υπόθεση (Hilbert): κάθε καλώς ορισμένο πρόβλημα έχει μια απάντηση που μπορούμε να βρούμε!
  - δημιουργία μαθηματικού συστήματος που μπορεί να περιγράφει κάθε πρόβλημα
  - κατασκευή αλγορίθμου που να αποφασίζει (σωστό ή λάθος)
  - Η υπόθεση Hilbert όριζε ένα μηχανικό τρόπο να αποφασίζουμε για όλα τα προβλήματα

Υπολογισιμότητα (2)

Gödel: Θεώρημα της μη πληρότητας (Incompleteness theorem)

- Οποιαδήποτε «τυπική» θεωρία δεν μπορεί να είναι και συνεπής και πλήρης. Για κάθε συνεπή θεωρία Θ υπάρχει μία αληθής δήλωση G που δεν μπορεί να την αποδείξει.
  - **Δήλωση G**:
    «Η G δεν μπορεί να αποδειχθεί ότι είναι αληθής μέσα στη Θ»
  - Αν η G είναι αληθής, τότε η G δεν μπορεί να αποδειχθεί ότι είναι αληθής μέσα στη Θ. Αρα η Θ είναι μη πλήρης!
  - Αν η G είναι ψευδής, τότε η G μπορεί να αποδειχθεί ότι είναι αληθής μέσα στη Θ. Δηλαδή η Θ αποδεικνύει ότι η G είναι αληθής ενώ αυτή είναι ψευδής. Άρα η Θ είναι μη συνεπής!
  - **Άρα η Θ δεν μπορεί να είναι και συνεπής και πλήρης**!

- https://en.wikipedia.org/wiki/G%C3%B6del%27s_incompleteness_theorems

Υπολογίσιμες και μη συναρτήσεις

- **Υπολογίσιμη** συνάρτηση είναι αυτή της οποίας η τιμή εξόδου μπορεί να προσδιοριστεί αλγοριθμικά από την τιμή εισόδου
- **Μη υπολογίσιμη** είναι η συνάρτηση που δεν μπορεί να προσδιοριστεί από κανένα αλγόριθμο
  - Δηλ. υπάρχουν συναρτήσεις για τις οποίες δεν υπάρχει καλά ορισμένος τρόπος, βήμα-προς-βήμα για να καθοριστεί η έξοδος βάσει των εισόδων
- **Υπολογιστικό πρόβλημα = υπολογίσιμη συνάρτηση**
- Η μηχανές μπορούν να εκτελούν **μόνο εργασίες που περιγράφονται από αλγορίθμους**.
  - Υπολογίσιμες συναρτήσεις ⟺ απόλυτες δυνατότητες μηχανών

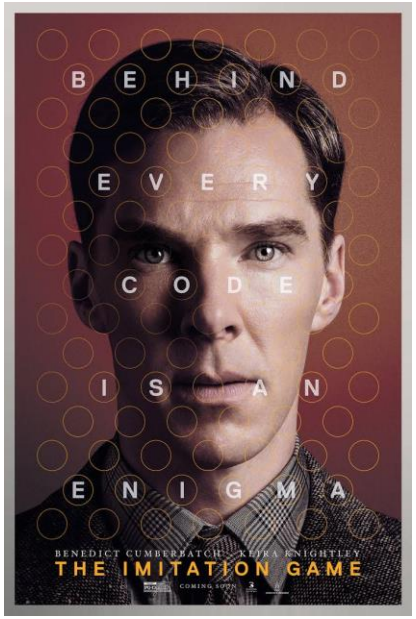
Άλυτα / αναπόδεικτα μαθηματικά προβλήματα (κάποια, μέχρι σήμερα)

- Η εικασία του Goldbach (1742): κάθε άρτιος αριθμός > 2 μπορεί να γραφεί σαν άθροισμα δύο πρώτων αριθμών
- Τελευταίο θεώρημα του Fermat (1637): (λύθηκε το 1995 από τον Andrew Wiles!)
  - Δεν υπάρχουν θετικοί ακέραιοι a, b, c τέτοιοι ώστε:

$$a^n + b^n = c^n, \quad n > 2$$

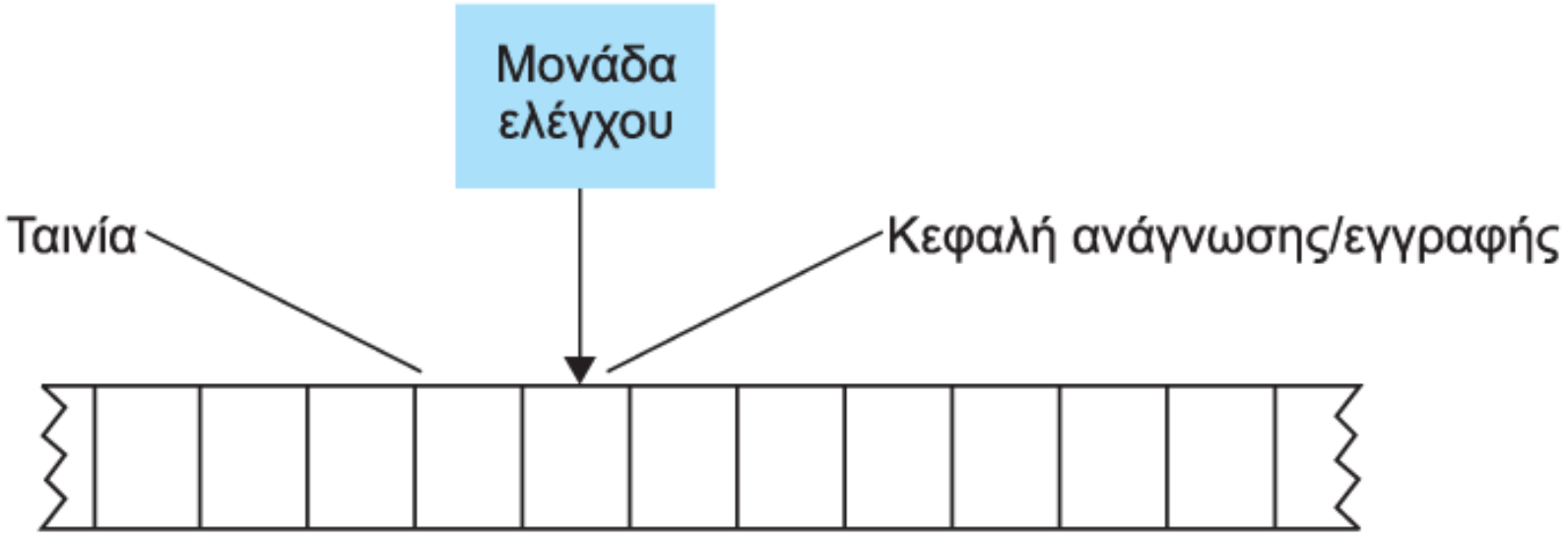
- P vs NP (θα δούμε μετά τι είναι τα P, NP)

Ιστορία



- Alan Turing: τι είναι υπολογίσιμο;
- Ένα πρόβλημα είναι υπολογίσιμο αν μπορούμε να περιγράψουμε ένα σύνολο εντολών που αν τις ακολουθήσουμε θα λύσουμε το πρόβλημα (procedure, or algorithm)
  - Για να γίνουμε όμως συγκεκριμένοι πρέπει να ορίσουμε τις **ικανότητες** της μηχανής που θα εκτελέσει τις εντολές
- Μηχανές με διαφορετικές δυνατότητες θα μπορούσαν να εκτελούν διαφορετικά σύνολα εντολών, και άρα θα μπορούν να επιλύσουν διαφορετικές κατηγορίες προβλημάτων
- Ο Turing πρότεινε ως αναφορά ένα τύπο μηχανών γνωστών σαν μηχανές Turing (υποθετική κατασκευή)
- Αυτές οι μηχανές βοήθησαν στον τυπικό ορισμό της υπολογιστικής διαδικασίας που ονομάζουμε "υπολογισιμότητα κατά Turing"(Turing-computability)

Τα στοιχεία μιας μηχανής Turing



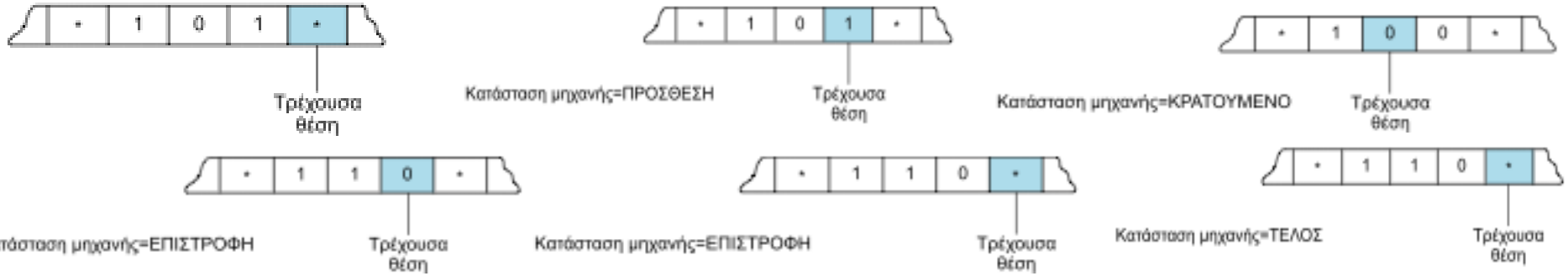
Λειτουργίες μίας μηχανής Turing

- Αλφάβητο = πεπερασμένο σύνολο από σύμβολα
  - Παράδειγμα: Αλφάβητο = {0,1,*}
- Σε κάθε κυψελίδα γράφεται ένα σύμβολο
- Σύνολο καταστάσεων Μηχανής
  - Π.χ. {ΕΝΑΡΞΗ, ΠΡΟΣΘΕΣΗ, ΚΡΑΤΟΥΜΕΝΟ, ΥΠΕΡΧΕΙΛΙΣΗ, ΕΠΙΣΤΡΟΦΗ, ΤΕΛΟΣ}
- Είσοδοι σε κάθε βήμα της μηχανής:
  - Τωρινή Κατάσταση (μια, από πεπερασμένο σύνολο καταστάσεων)
  - Τιμή στην τρέχουσα κυψελίδα της ταινίας (εκεί που βρίσκεται τώρα η κεφαλή)
- Οι ενέργειες σε κάθε βήμα:
  - Γράφει μία τιμή στην τρέχουσα κυψελίδα της ταινίας.
  - Κινεί την κεφαλή εγγραφή/ανάγνωσης μια θέση δεξιά ή μια θέση αριστερά
  - Ορίζει μία καινούργια κατάσταση

Παράδειγμα 1: Μηχανή Turing για αύξηση μιας τιμής που είναι γραμμένη στην ταινία κατά μία μονάδα

Τρέχουσα κατάσταση	Περιεχόμενο τρέχοντος κελιού	Τιμή προς εγγραφή	Κατεύθυνση κίνησης	Νέα κατάσταση
ΕΝΑΡΞΗ	*	*	Αριστερά	ΠΡΟΣΘΕΣΗ
ΠΡΟΣΘΕΣΗ	0	1	Δεξιά	ΕΠΙΣΤΡΟΦΗ
ΠΡΟΣΘΕΣΗ	1	0	Αριστερά	ΚΡΑΤΟΥΜΕΝΟ
ΠΡΟΣΘΕΣΗ	*	*	Δεξιά	ΤΕΛΟΣ
ΚΡΑΤΟΥΜΕΝΟ	0	1	Δεξιά	ΕΠΙΣΤΡΟΦΗ
ΚΡΑΤΟΥΜΕΝΟ	1	0	Αριστερά	ΚΡΑΤΟΥΜΕΝΟ
ΚΡΑΤΟΥΜΕΝΟ	*	1	Αριστερά	ΥΠΕΡΧΕΙΛΙΣΗ
ΥΠΕΡΧΕΙΛΙΣΗ	*	*	Δεξιά	ΕΠΙΣΤΡΟΦΗ
ΕΠΙΣΤΡΟΦΗ	0	0	Δεξιά	ΕΠΙΣΤΡΟΦΗ
ΕΠΙΣΤΡΟΦΗ	1	1	Δεξιά	ΕΠΙΣΤΡΟΦΗ
ΕΠΙΣΤΡΟΦΗ	*	*	Καμία κίνηση	ΤΕΛΟΣ

Αρχική Κατάσταση: ΕΝΑΡΞΗ



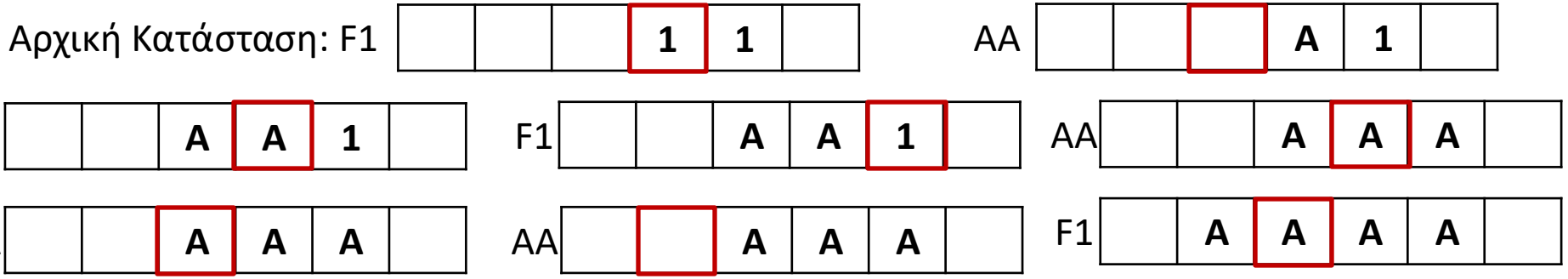
Παράδειγμα 2: Μηχανή Turing για διπλασιασμό των 1

Αλφάβητο = {1,A,–} (όπου – είναι το κενό)

Σύνολο καταστάσεων: F1 (find 1), AA (add A), CA1 (change A to 1), HALT

Είσοδος: **πλήθος από 1**. Έξοδος: **διπλάσιο πλήθος από 1**.

Τρέχουσα κατάσταση	Περιεχόμενο τρέχοντος κελιού	Τιμή προς εγγραφή	Κατεύθυνση κίνησης	Νέα κατάσταση
F1	1	A	Αριστερά	AA
F1	A	A	Δεξιά	F1
F1	–	–	Αριστερά	CA1
AA	A	A	Αριστερά	AA
AA	–	A	Δεξιά	F1
CA1	A	1	Αριστερά	CA1
CA1	–	–	Δεξιά	HALT



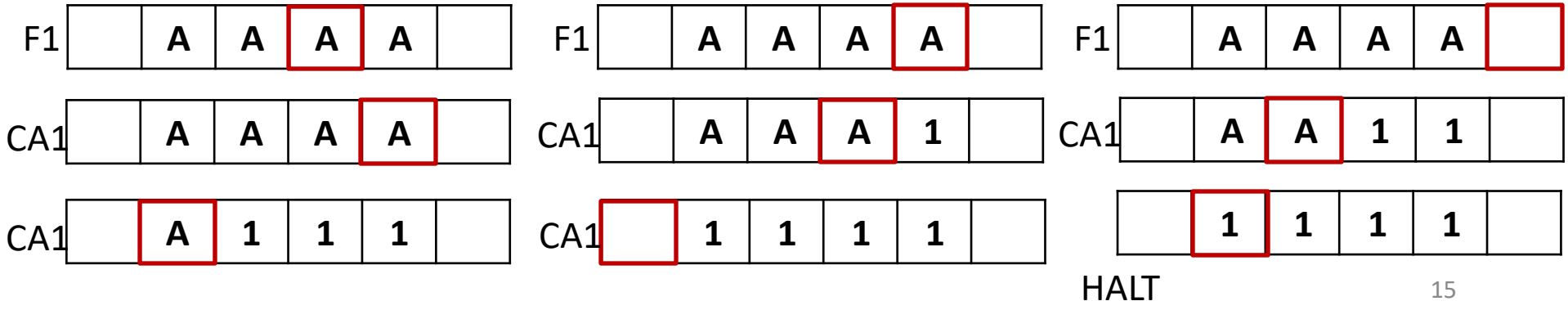
Παράδειγμα 2: Μηχανή Turing για διπλασιασμό των 1

Αλφάβητο = {1,A,–} (όπου – είναι το κενό)

Σύνολο καταστάσεων: F1 (find 1), AA (add A), CA1 (change A to 1), HALT

Είσοδος: **πλήθος από 1**. Έξοδος: **διπλάσιο πλήθος από 1**.

Τρέχουσα κατάσταση	Περιεχόμενο τρέχοντος κελιού	Τιμή προς εγγραφή	Κατεύθυνση κίνησης	Νέα κατάσταση
F1	1	A	Αριστερά	AA
F1	A	A	Δεξιά	F1
F1	–	–	Αριστερά	CA1
AA	A	A	Αριστερά	AA
AA	–	A	Δεξιά	F1
CA1	A	1	Αριστερά	CA1
CA1	–	–	Δεξιά	HALT



Church-Turing thesis (1)

- Είδαμε παραδείγματα συναρτήσεων που είναι υπολογίσιμη κατά Turing (Turing computable)
  - Δίνουμε είσοδο στην μηχανή Turing, βγάζει έξοδο
  - Π.χ. Αύξηση τιμής της μηχανής κατά 1
- **Ερώτηση**: το ότι δεν υπάρχει αλγόριθμος που να λύνει μία κλάση προβλημάτων οφείλεται στο ότι το μοντέλο του υπολογιστή είναι αδύναμο;
- **Church-Turing**: όλα τα μοντέλα υπολογιστών και αλγορίθμων που υπάρχουν ή μπορεί να υπάρξουν στο μέλλον είναι υπολογιστικά ισοδύναμα με την Μηχανή Turing!

Church-Turing thesis (2)

- **Church-Turing thesis**: Το σύνολο των υπολογίσιμων συναρτήσεων είναι ίδιο με το σύνολο των υπολογίσιμων συναρτήσεων κατά Turing

- Ισοδύναμα: Μια συνάρτηση είναι υπολογίσιμη αν και μόνο αν είναι υπολογίσιμη κατά Turing

- Μία μηχανή Turing μπορεί να υπολογίσει κάθε υπολογίσιμη συνάρτηση
  - Δεν αποδεικνύεται αλλά είναι γενικά αποδεκτό
  - Γι' αυτό το λέμε thesis

- Άρα **η μη-υπολογισιμότητα είναι ιδιότητα του προβλήματος** και δεν εξαρτάται από το υπολογιστικό σύστημα

Καθολική Γλώσσα Προγραμματισμού

- **Καθολική γλώσσα προγραμματισμού** είναι μία γλώσσα που μπορεί να εκφράσει ένα πρόγραμμα υπολογισμού για κάθε υπολογίσιμη συνάρτηση
  - Αν ένας προγραμματιστής βρει ότι ένα πρόβλημα δεν μπορεί να λυθεί με αυτή τη γλώσσα, θα σημαίνει ότι δεν υπάρχει αλγόριθμος για να λύσει το πρόβλημα
  - Παράδειγμα καθολικής γλώσσας: Στοιχειώδης γλώσσα

Η Στοιχειώδης Γλώσσα

- Οι μόνες εντολές της Γλώσσας είναι:
  - `clear` *`όνομα`*`;`
  - `incr` *`όνομα`*`;`
  - `decr` *`όνομα`*`;`
  - `while` *`όνομα`* `not 0 do; … end;`
- `Incr X;`
  - Υπολογίζει την ίδια συνάρτηση που υπολόγιζε η μηχανή Turing (σε προηγούμενη διαφάνεια)

Παραδείγματα σε Στοιχειώδη Γλώσσα (1)

- Οι μόνες εντολές της Γλώσσας είναι:
  - `clear` *όνομα*`;`
  - `incr` *όνομα*`;`
  - `decr` *όνομα*`;`
  - `while` *όνομα* `not 0 do; … end;`

- Παράδειγμα 1: X → 3

```
clear X;
incr X;
incr X;
incr X;
```

- Παράδειγμα 2: X → Z

```
clear Z;
while X not 0 do;
    incr Z;
    decr X;
end;
```

Παραδείγματα σε Στοιχειώδη Γλώσσα (2)

- Στο Παράδειγμα 2: X → Z, το X χάνει την τιμή του

```
clear Z;
while X not 0 do;
    incr Z;
    decr X;
end;
```

- Παράδειγμα 3: X → Z, αλλά το X να κρατάει την τιμή του. Χρήση **βοηθητικής** μεταβλητής.

```
clear Aux;
clear Z;
while X not 0 do;
    incr Z;
    incr Aux;
    decr X;
end;
while Aux not 0 do;
    incr X;
    decr Aux;
end;
```

Άσκηση: Τι κάνει το παρακάτω πρόγραμμα σε Στοιχειώδη Γλώσσα;

```
clear Z;
while X not 0 do;
    clear W;
    while Y not 0 do;
        incr Z;
        incr W;
        decr Y;
    end;
    while W not 0 do;
        incr Y;
        decr W;
    end;
    decr X;
end;
```

Στο τέλος $Z = X \cdot Y$

Στοιχειώδης Γλώσσα

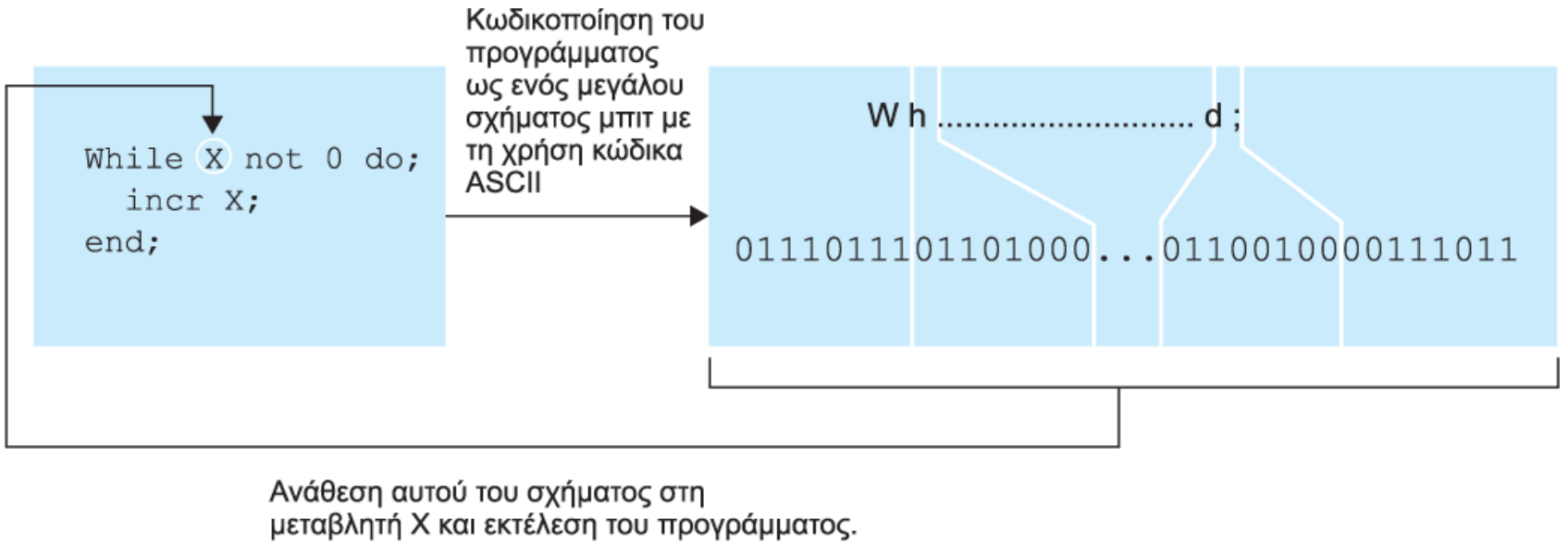
- Έχει αποδειχθεί ότι η Στοιχειώδης Γλώσσα **επαρκεί** και μπορεί να χρησιμοποιηθεί για να εκφράσει τους αλγορίθμους για να υπολογίσουμε όλες τις συναρτήσεις που είναι υπολογίσιμες κατά Turing
- Συνδυάζοντας με το Turing-Church thesis: κάθε υπολογίσιμη συνάρτηση μπορεί να υπολογιστεί με ένα πρόγραμμα γραμμένο στη Στοιχειώδη Γλώσσα
- Η Στοιχειώδης γλώσσα είναι καθολική
  - αν υπάρχει αλγόριθμος που λύνει ένα πρόβλημα, το πρόβλημα μπορεί να λυθεί με ένα πρόγραμμα σε Στοιχειώδη Γλώσσα
  - Σημαντική στη θεωρία, όχι στην πράξη.

Μια μη υπολογίσιμη συνάρτηση

- Θα δούμε μια συνάρτηση που δεν είναι υπολογίσιμη κατά Turing, άρα και γενικά είναι μη υπολογίσιμη
- «Πρόβλημα του Τερματισμού» (Halting Problem):
  - Είναι το πρόβλημα του εάν ένα πρόγραμμα θα τερματίσει ή όχι για κάποιες αρχικές συνθήκες
  - Π.χ.
```
while X not 0 do;
    incr X;
end;
```
  - Αν ξεκινήσει με X=0, θα τερματίσει, αλλιώς όχι.
  - Γενικά είναι αδύνατο να υπολογίζουμε **πάντα** αν ένα πρόγραμμα θα τερματίζει ή όχι! (Θα το δείξουμε σε λίγο…)

Πρόγραμμα που γίνεται είσοδος σε άλλα προγράμματα

Αυτο-αναφορά (self-reference): η ιδέα ένα αντικείμενο να αναφέρεται στον εαυτό του. (Θα χρησιμεύσει αργότερα)



Το σύνολο από bit που προκύπτει όπως παραπάνω από την αναπαράσταση του προγράμματος σε δυαδική μορφή, αποτελεί έναν ΠΟΛΥ μεγάλο αριθμό Χ

Αυτο-αναφορά

Έστω το πρόγραμμα 1:

```
while X not 0 do;
    incr X;
end;
```

- X: η μεταβλητή που παριστάνει την αριθμητική τιμή που προκύπτει από το πρόγραμμα 1 (όπως πριν). Το πρόγραμμα 1 δεν τερματίζει!

Τώρα, έστω το πρόγραμμα 2:

```
clear X
while X not 0 do;
    incr X;
end;
```

- X η μεταβλητή που παριστάνει την αριθμητική τιμή που προκύπτει από το πρόγραμμα 2. Το πρόγραμμα 2 τερματίζει!

Αυτο-τερματιζόμενο (self-terminating) πρόγραμμα: Αν εκτελεστεί με όλες τις μεταβλητές αρχικοποιημένες στην αριθμητική τιμή που προκύπτει από το πρόγραμμα καθαυτό, τερματίζει.

- Δηλαδή **τερματίζει αν λάβει ως είσοδο τον εαυτό του**.
- Ένα πρόγραμμα είτε είναι αυτό-τερματιζόμενο είτε όχι. Αυτό είναι απλά μία ιδιότητα του προγράμματος.

Το πρόβλημα τερματισμού

- **Πρόβλημα τερματισμού (Halting Problem)**: είναι το πρόβλημα του καθορισμού αν ένα πρόγραμμα είναι αυτο-τερματιζόμενο

- Θα αποδείξουμε ότι δεν υπάρχει αλγόριθμος για να απαντήσουμε στο παραπάνω ερώτημα

- Προσοχή: Το ότι σε κάποιες ειδικές περιπτώσεις μπορούμε να καθορίζουμε αν ένα πρόγραμμα είναι αυτο-τερματιζόμενο, δεν σημαίνει ότι λύσαμε το πρόβλημα τερματισμού

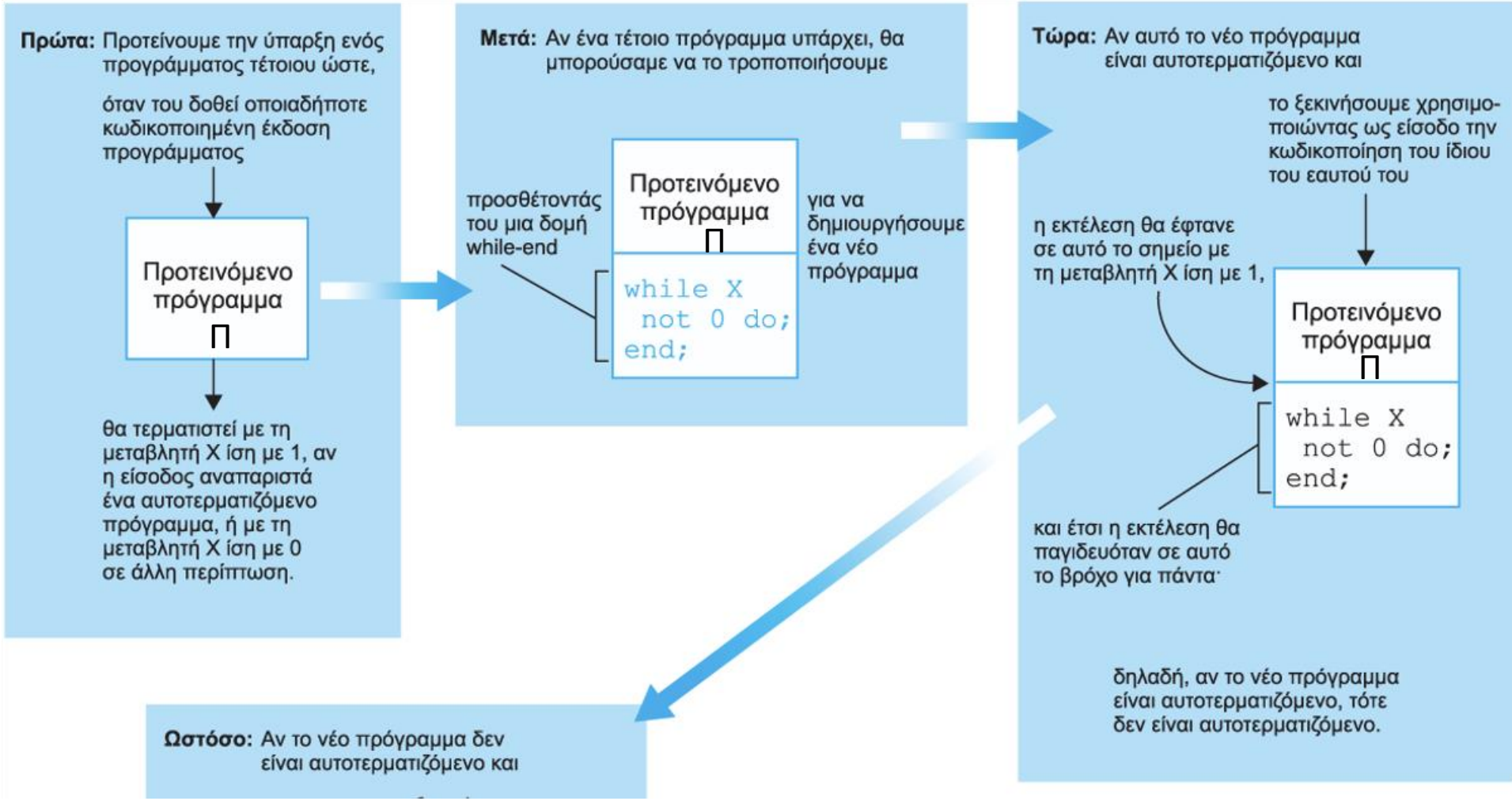
Απόδειξη μη επιλυσιμότητας του προβλήματος τερματισμού

- Υπόθεση: ύπαρξη συνάρτησης τερματισμού (halting function) που παίρνει σαν είσοδο ένα πρόγραμμα και επιστρέφει αν είναι **αυτό-τερματιζόμενο ή όχι**.
- Θα αποδείξουμε ότι η συνάρτηση τερματισμού δεν είναι υπολογίσιμη με εις άτοπον απαγωγή.
  - Έστω ότι η συνάρτηση τερματισμού είναι υπολογίσιμη

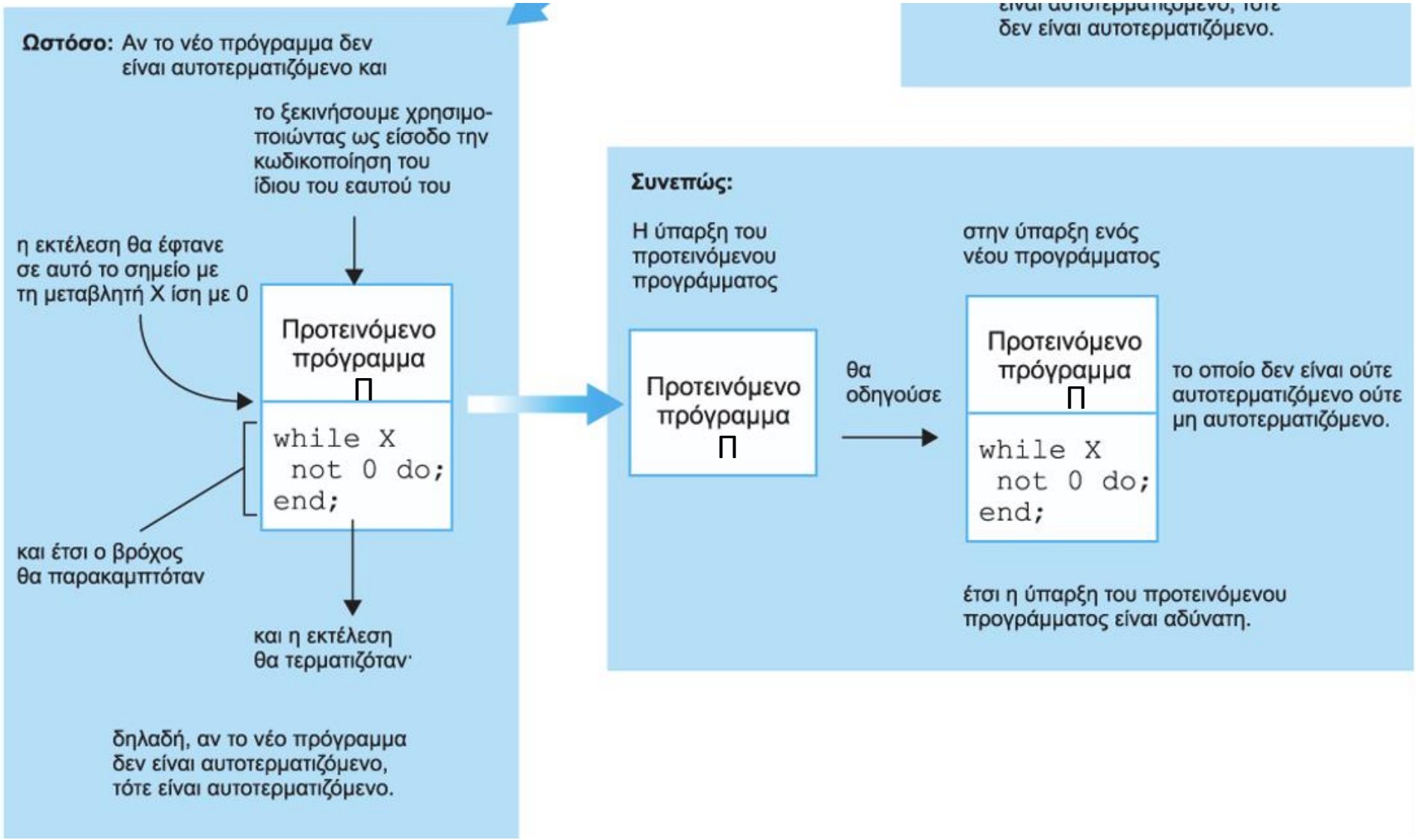
    ➔ Υπάρχει Στοιχειώδες πρόγραμμα που την υπολογίζει
- Υπόθεση ύπαρξης στοιχειώδους προγράμματος Π με:
  - Είσοδος: αριθμητική τιμή που προκύπτει από ένα πρόγραμμα
  - Έξοδος: 1, αν το πρόγραμμα είναι αυτο-τερματιζόμενο

    0, αν το πρόγραμμα δεν είναι αυτο-τερματιζόμενο
- Στη συνέχεια θα κατασκευάσουμε ένα καινούργιο πρόγραμμα χρησιμοποιώντας το Π που **δεν είναι ούτε αυτό-τερματιζόμενο ούτε μη** → ΑΤΟΠΟ!

Απόδειξη της μη επιλυσιμότητας του προβλήματος τερματισμού (1)



Απόδειξη της μη επιλυσιμότητας του προβλήματος τερματισμού (2)

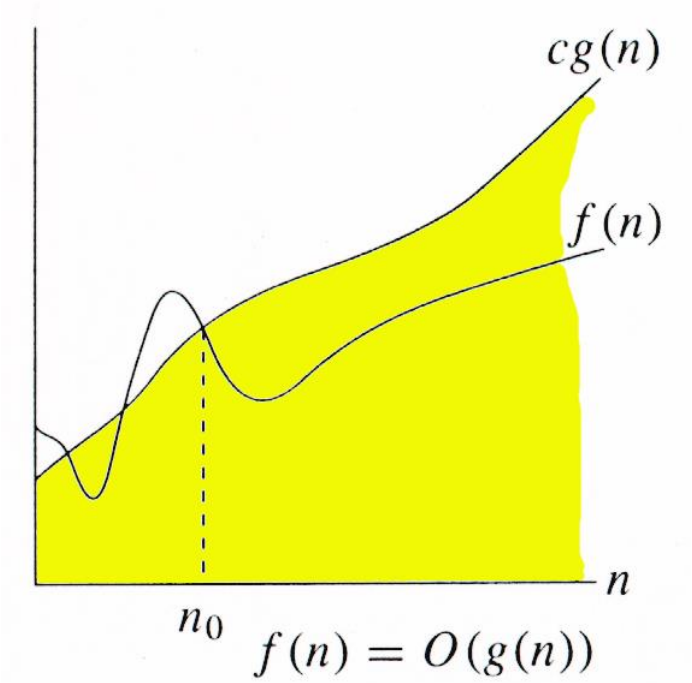


Εισαγωγή στην υπολογιστική πολυπλοκότητα

Ασυμπτωτικός συμβολισμός O (επανάληψη)

- ΑΝΩ ΟΡΙΟ στην αύξηση μιας συνάρτησης.
- Έστω f(n) και g(n) συναρτήσεις, ώστε f,g : $\mathbb{N}^+ \rightarrow \mathbb{R}^+$
- f(n) = O(g(n)) : Η f(n) δεν αυξάνεται πιο γρήγορα από την g(n)
- Ένα (σταθερό) πολλαπλάσιο της g(n) είναι άνω όριο στην f(n)

για αρκετά μεγάλα n



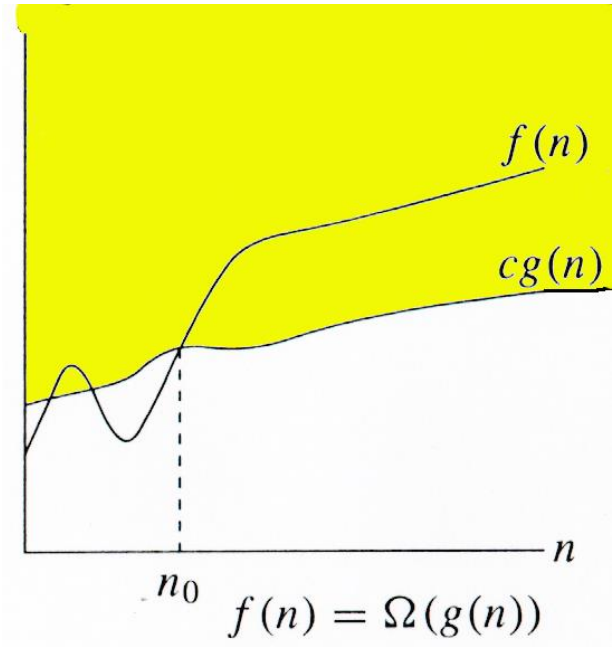
Πχ. 2n+1 = O(n)

$$\exists c > 0, n_o > 0, \text{ τέτοια ώστε } f(n) \leq c \cdot g(n), \forall n > n_o$$

Ασυμπτωτικός συμβολισμός Ω (επανάληψη)

- ΚΑΤΩ ΟΡΙΟ στην αύξηση της T(n)
- Έστω f(n) και g(n) συναρτήσεις, ώστε f,g : $\mathbb{N}^+ \to \mathbb{R}^+$
- f(n) = Ω(g(n)) : Η f(n) δεν αυξάνεται πιο αργά από την g(n)
- Ένα (σταθερό) πολλαπλάσιο της g(n) είναι κάτω όριο στην f(n)

για αρκετά μεγάλα n



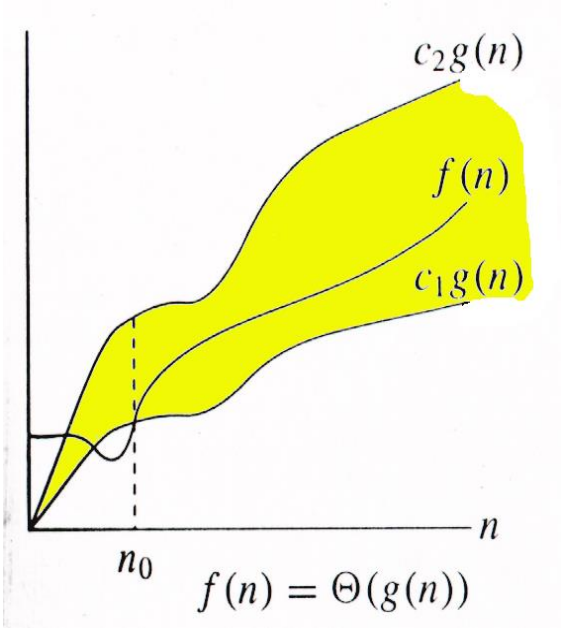
Πχ. n/2 - 1 = Ω(n)

$$\exists c > 0, n_o > 0, \text{ τέτοια ώστε } f(n) \geq c \cdot g(n), \forall n > n_o$$

Ασυμπτωτικός συμβολισμός Θ (επανάληψη)

- ΑΝΩ ΚΑΙ ΚΑΤΩ ΟΡΙΟ στην αύξηση της T(n)
- f(n) = Θ(g(n)) : Η f(n) δεν αυξάνεται ούτε πιο γρήγορα ούτε πιο αργά από την g(n)
- (σταθερά) πολλαπλάσια της g(n) είναι άνω και κάτω όρια στην f(n)

για αρκετά μεγάλα n



$$f(n)=\Theta(g(n)) \Leftrightarrow f(n)=O(g(n)) \text{ και } f(n)=\Omega(g(n))$$

$\exists c_1, c_2 > 0, n_o > 0,$ τέτοια ώστε $c_2 \cdot g(n) \geq f(n) \geq c_1 \cdot g(n), \forall n > n_o$

Παραδείγματα Πολυπλοκότητας

- Ένα πρόβλημα είναι τάξης Θ(f(n)) αν μπορεί να λυθεί με έναν αλγόριθμο της τάξης Θ(f(n)) και δεν υπάρχει αλγόριθμος μικρότερης πολυπλοκότητας που να το λύνει
- Ένα πρόβλημα τάξης O(f(n)) μπορεί να λυθεί και με αλγόριθμο τάξης Θ(f(n))
- Παράδειγμα 1:
  - Δυαδική αναζήτηση σε λίστα n ονομάτων με αλφαβητική ταξινόμηση: O(logn)
  - Αποδεικνύεται ότι το πρόβλημα αναζήτησης είναι πολυπλοκότητας Θ(logn)
  - Άρα, η δυαδική αναζήτηση είναι το καλύτερο που μπορεί να γίνει (βέλτιστος αλγόριθμος)
- Παράδειγμα 2:
  - Ταξινόμηση λίστας n ονομάτων με τη μέθοδο ταξινόμησης με εισαγωγή (InsertionSort): $O(n^2)$
  - Αποδεικνύεται ότι το πρόβλημα της ταξινόμησης είναι της τάξης Θ(n logn)
  - Άρα, η μέθοδος ταξινόμησης με εισαγωγή δεν είναι η βέλτιστη λύση, δηλ. υπάρχει και καλύτερη
    - …η MergeSort

Πολυωνυμικά προβλήματα

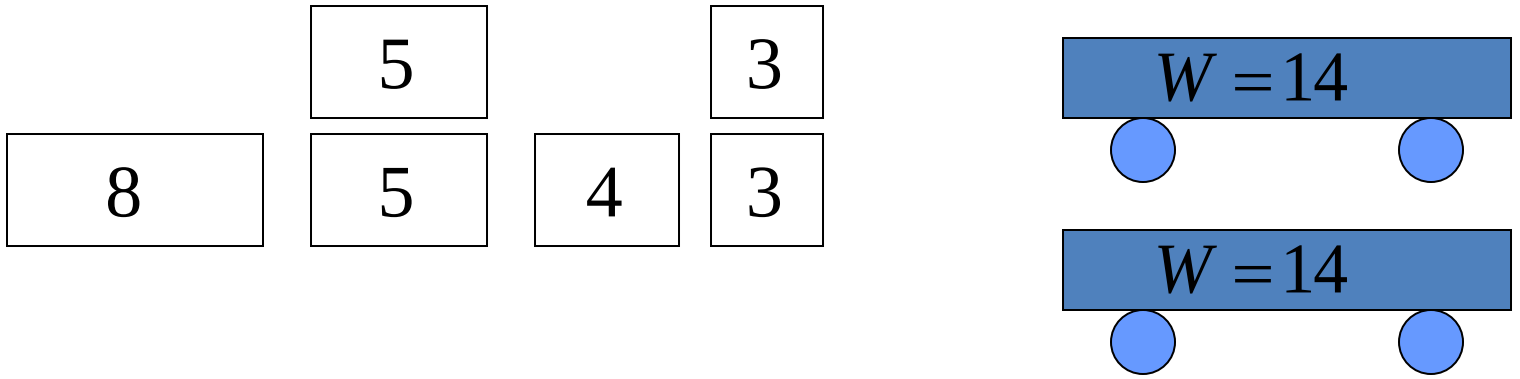
- Ένα πρόβλημα λέγεται πολυωνυμικό (polynomial) αν ο αλγόριθμος επίλυσής του είναι της τάξης O(f(n)), όπου f(n) είναι πολυώνυμο
- Κλάση προβλημάτων **P**: σύνολο των πολυωνυμικών προβλημάτων
  - Αν ένα πρόβλημα ανήκει στην κλάση **P** σημαίνει ότι η λύση του μπορεί να υπολογιστεί σε σχετικά σύντομο χρόνο (πρακτική για διάφορες εφαρμογές)
- **Δυσεπίλυτα (intractable) (ή μη-εφικτά, infeasible):** προβλήματα που είναι πολύ περίπλοκο να επιλυθούν πρακτικά
  - Οι Επιστήμονες Υπολογιστών θεωρούν την τάξη **P** ως το όριο μεταξύ προβλημάτων δισεπίλυτων και αυτών που έχουν πρακτική λύση
  - Δείτε τι συμβαίνει π.χ. για το πρόβλημα που είναι $\Theta(2^n)$
  - Το 2^n δεν μπορεί να είναι φραγμένο από ένα πολυώνυμο
    - Παράδειγμα: Η εύρεση όλων των δυνατών υποσυνόλων ενός συνόλου με n στοιχεία είναι πολυπλοκότητας $\Theta(2^n)$

Μη εφικτό πρόβλημα 1: Πρόβλημα της Συσκευασίας (bin packing)

Προβλήματα μη υπολογίσιμα σε πολυωνυμικό χρόνο

Το πρόβλημα της συσκευασίας (bin packing problem)

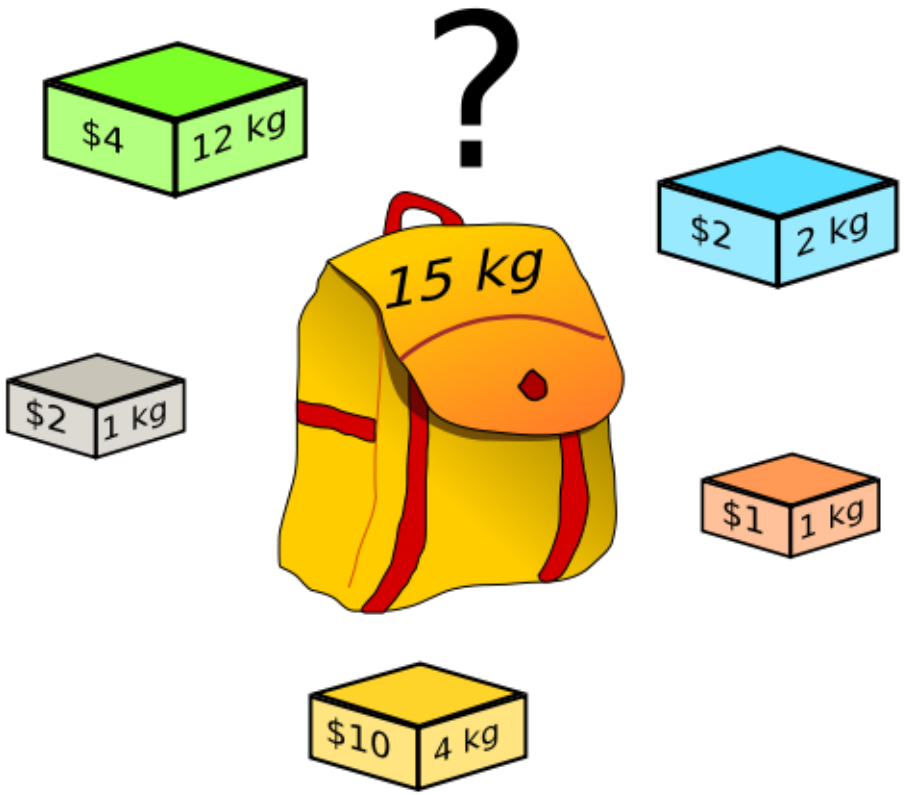
Μπορούν *N* κιβώτια βάρους w_1, w_2, …w_N να μεταφερθούν από *K* φορτηγά που το καθένα μπορεί να σηκώσει βάρος το πολύ *W*;



Εκδοχές των προβλημάτων

- Κάθε πρόβλημα έχει μια εκδοχή που λέγεται εκδοχή απόφασης (decision version) και μια εκδοχή που λέγεται εκδοχή βελτιστοποίησης (optimization version)
- **Εκδοχή απόφασης** (decision problem): απαντάμαι με ΝΑΙ ή ΟΧΙ
  - Π.χ. Μπορούν *N* κιβώτια βάρους w_1, w_2, …w_N να μεταφερθούν από *K* φορτηγά που το καθένα μπορεί να σηκώσει βάρος το πολύ *W*;
- **Εκδοχή βελτιστοποίησης** (optimization problem): όταν προσπαθούμε να ελαχιστοποιήσουμε ή να μεγιστοποιήσουμε κάποια ποσότητα
  - Π.χ. Ποιο είναι το ελάχιστο πλήθος φορτηγών για να μεταφερθούν *N* κιβώτια βάρους w_1, w_2, …w_N , όταν κάθε φορτηγό μπορεί να σηκώσει βάρος το πολύ *W*; Ποια είναι μία τέτοια λύση;
- Αν ένα πρόβλημα απόφασης λύνεται πολυωνυμικά τότε και το πρόβλημα βελτιστοποίησης λύνεται πολυωνυμικά (και αντίστροφα).

Μη εφικτό πρόβλημα 2: Πρόβλημα Knapsack



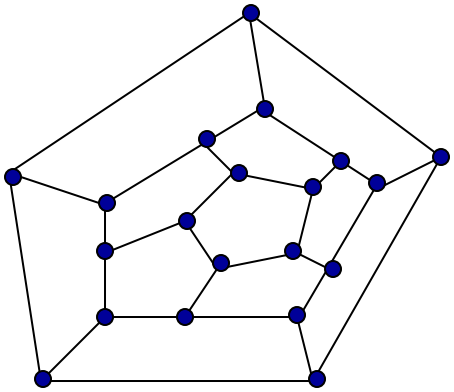
Ποια αντικείμενα να επιλέξω να βάλω μέσα στο σάκο ώστε να μεγιστοποιήσω την συνολική αξία των αντικειμένων μέσα στο σάκο;

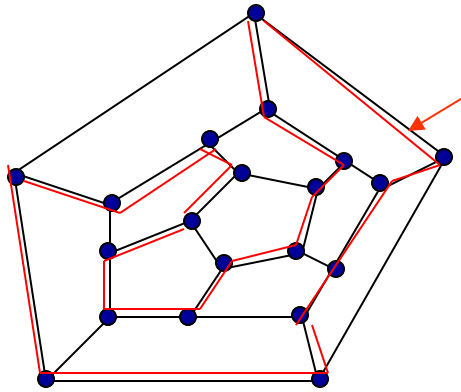
- Είναι εκδοχή απόφασης ή βελτιστοποίησης του προβλήματος;
  - Εκδοχή βελτιστοποίησης
- Ποιο είναι η αντίστοιχη εκδοχή **απόφασης**;
  - Μπορώ να χωρέσω στο σάκο αντικείμενα αξίας τουλάχιστον V;

Μη εφικτό πρόβλημα 3: Πρόβλημα Πλανόδιου Πωλητή (Traveling Salesman Problem)

Με δεδομένο ένα χάρτη δρόμων με N πόλεις και τις αντίστοιχες ανά δύο μεταξύ τους χιλιομετρικές αποστάσεις, είναι δυνατόν ένας πωλητής να ολοκληρώσει κάποιο ταξίδι όπου:

- να περνάει από κάθε πόλη ακριβώς μία φορά και
- η συνολική διαδρομή να έχει μήκος ≤ K ;





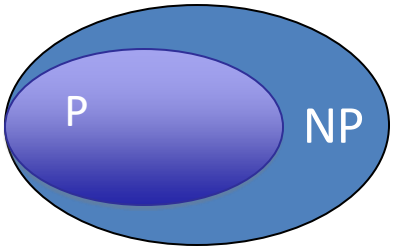
- Είναι εκδοχή απόφασης ή βελτιστοποίησης του προβλήματος;
  - Εκδοχή απόφασης
- Ποιο είναι η αντίστοιχη εκδοχή **απόφασης**;
  - Ποιος είναι ο συντομότερος κύκλος Hamilton;

Μη εφικτό πρόβλημα 3: Πρόβλημα Πλανόδιου Πωλητή (Traveling Salesman Problem)

- **Πολύ δύσκολο πρόβλημα**: χρειάζεται να εξετάσω όλους τους πιθανούς κύκλους και να εξετάσω αν το μήκος κάποιου από αυτούς είναι $\leq K$
- Μη πολυωνυμικός χρόνος (διότι ο αριθμός των πιθανών κύκλων αυξάνει εκθετικά με το πλήθος των κόμβων N)
- Παρατήρηση: αν υπάρχει κάποιος τέτοιος κύκλος και τον επιλέξουμε πρώτο, θα βρούμε τη λύση γρήγορα!
- Εναλλακτικά, αν μας δοθεί ένας τέτοιος κύκλος μπορούμε να ελέγξουμε ότι ικανοποιεί τις συνθήκες σε πολυωνυμικό χρόνο ως προς N (γιατί χρειάζεται να προσθέσω N αποστάσεις).

Τάξη NP (Non-deterministic Polynomial)

- Κάθε πρόβλημα που μπορώ να ελέγξω την ορθότητα μιας προτεινόμενης λύσης σε πολυωνυμικό χρόνο ανήκει στην **κλάση NP**
  - Προσοχή: δεν μπορώ να ελέγξω τη μη ορθότητα, **μόνο** την ορθότητα!
  - **Η κλάση P είναι υποσύνολο της κλάσης NP**
- Πολλά από τα δύσκολα προβλήματα είναι στην κλάση NP (Bin packing, Knapsack, Πλανόδιου Πωλητή…)
- **Δεν έχει βρεθεί μέχρι σήμερα** κάποιος πολυωνυμικός αλγόριθμος που να λύνει τα τρία προβλήματα που είδαμε. Άρα δεν γνωρίζουμε αν ανήκουν στην κλάση P.
- Είναι ακόμα άγνωστο αν η τάξη NP είναι μεγαλύτερη της τάξης P (P=? NP).



Unsolved problem in computer science:

? *If the solution to a problem is easy to check for correctness, must the problem be easy to solve?*

(more unsolved problems in computer science)

Δείτε: https://en.wikipedia.org/wiki/P_versus_NP_problem

Προβλήματα NP-Complete (1)

- Το παραπάνω ερώτημα οδήγησε στην ανακάλυψη μιας νέας κατηγορίας προβλημάτων ανάμεσα σε αυτά της κλάσης NP
- Τάξη προβλημάτων NP-πλήρη (NP-Complete): Τα πιο δύσκολα (από πλευράς πολυπλοκότητας) προβλήματα της κλάσης NP.

- **Ένα πρόβλημα Π είναι NP-Complete αν και μόνο αν**
  1. **Π ανήκει στην κλάση NP**
  2. **Κάθε πρόβλημα Π' της NP ανάγεται πολυωνυμικά στο Π**

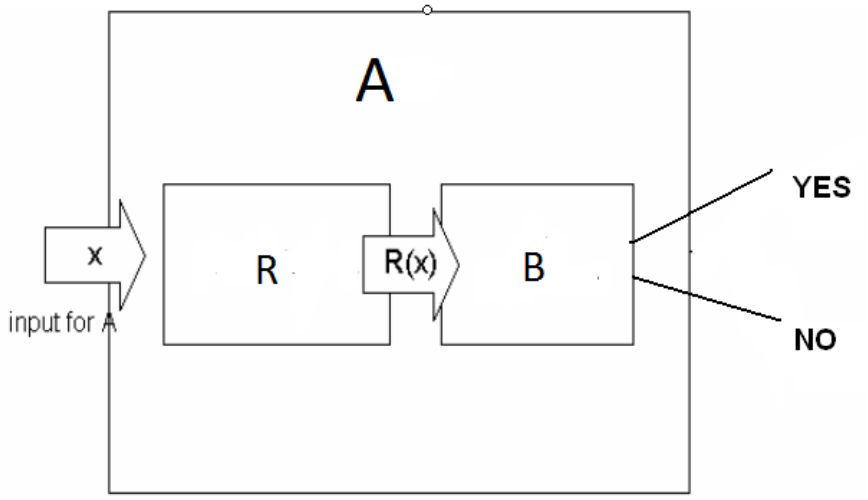
     (Δηλαδή μπορώ σε πολυωνυμικό αριθμό βημάτων να χρησιμοποιήσω τη λύση του Π για να λύσω το Π')

- Μια λύση σε πολυωνυμικό χρόνο για οποιοδήποτε NP-Complete πρόβλημα θα έδινε μια λύση σε πολυωνυμικό χρόνο για **όλα** τα άλλα προβλήματα της τάξης NP
  - δηλ. αυτό θα σημάνει ότι P = NP !!

Πολυωνυμική Αναγωγή

Το πρόβλημα Α ανάγεται πολυωνυμικά στο πρόβλημα Β ($A \leq_p B$) εάν υπάρχει ένας πολυωνυμικός μετασχηματισμός R ο οποίος για κάθε είσοδο x του Α παράγει μια είσοδο R(x) του Β τέτοια ώστε:

το Β με είσοδο R(x) έχει απάντηση ΝΑΙ εάν και μόνο εάν το Α με είσοδο x έχει απάντηση ΝΑΙ



Για να λύσουμε το Α με είσοδο x αρκεί να :

- μετασχηματίσουμε την x στην R(x)
- λύσουμε το Β με είσοδο R(x)

Η πολυπλοκότητα του Α είναι όση του R (πολυωνυμική) + του Β.

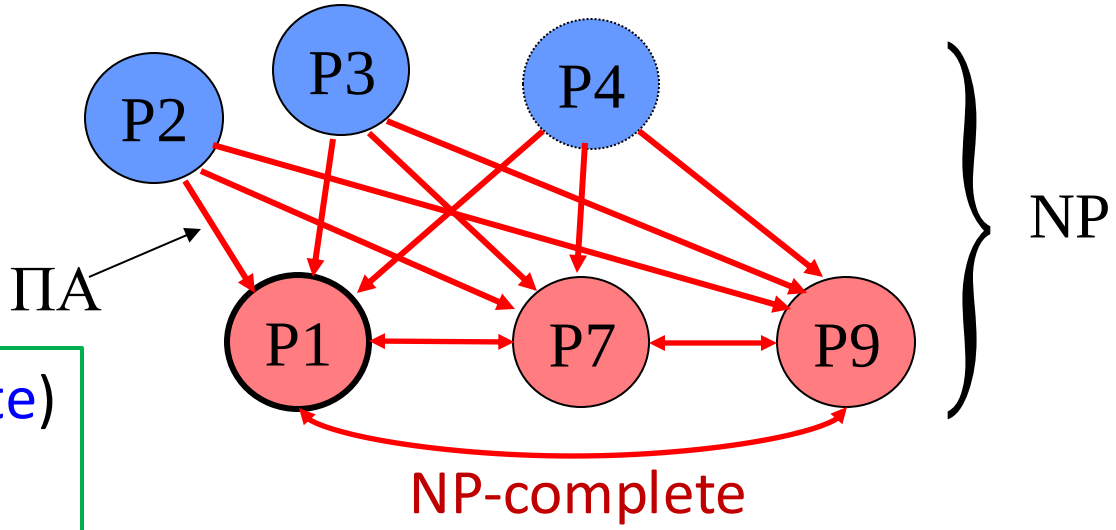
Συμπεράσματα:

- **Αν Β λύνεται πολυωνυμικά, τότε και Α λύνεται πολυωνυμικά**
- **Αν το Α είναι NP-Complete τότε και το Β είναι NP-Complete**

Προβλήματα NP-Complete (2)

Ένα πρόβλημα Π είναι NP-Complete αν και μόνο αν

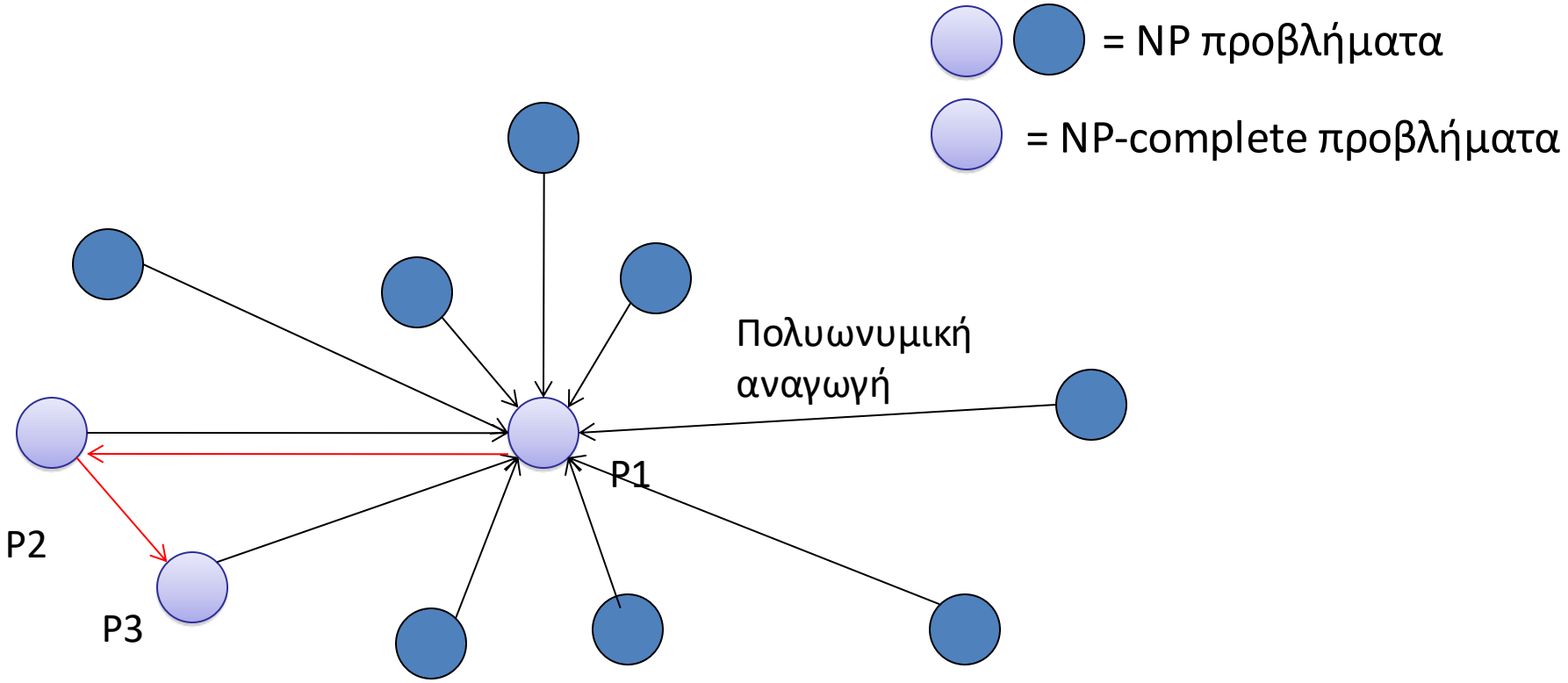
1. **Π ∈ NP**
2. **Κάθε πρόβλημα Π' ∈ NP ανάγεται πολυωνυμικά στο Π (Π' $\leq_p$ Π)**



ΠΑ = Πολυωνυμική Αναγωγή

Αν P1 (που είναι NP-Complete) λυνόταν πολυωνυμικά, τότε εξ'ορισμού **όλα τα προβλήματα της κλάσης NP θα λύνονταν πολυωνυμικά** και τότε P=NP!!

Κατασκευή κλάσης NP-Complete



Βήμα 1: βρίσκω ένα πρόβλημα στην τάξη NP-complete, έστω P1. **Πώς;**

- Δείχνω ότι όλα τα προβλήματα στο σύνολο των NP ανάγονται πολυωνυμικά στο P1 (1971, Cook) [https://en.wikipedia.org/wiki/Cook%E2%80%93Levin_theorem]

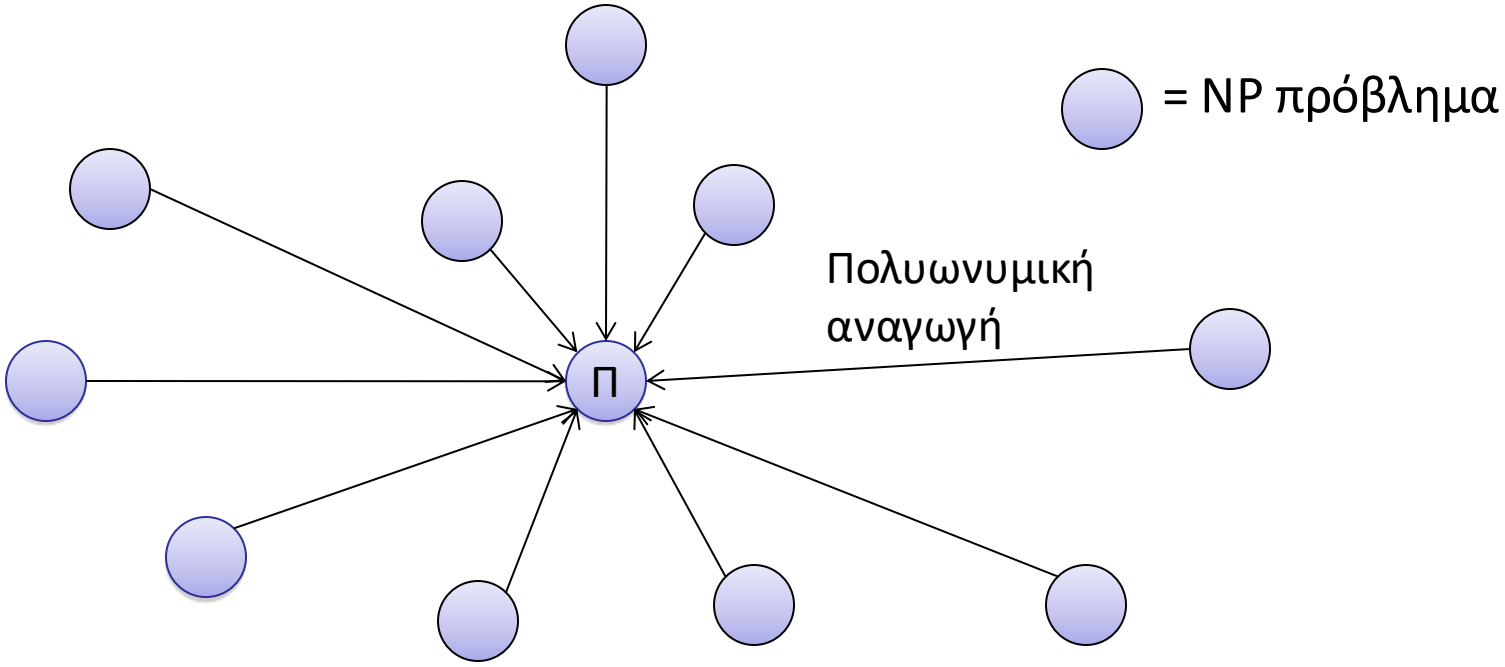
Βήμα 2: προσθέτω περισσότερα προβλήματα στο σύνολο των NP-complete.

- P2 στο NP-complete εάν P1 ανάγεται πολυωνυμικά στο P2 (δηλ. P2 είναι τουλάχιστον τόσο δύσκολο όσο το P1)
- P3 στο NP-complete εάν P1 ή P2 ανάγονται στο P3, κλπ...

Προβλήματα NP-Complete (3)

Πώς μπορώ να αποδείξω ότι ένα πρόβλημα Π είναι NP-Complete;

1. Απέδειξε ότι Π ∈ NP
2. Απέδειξε ότι για κάθε πρόβλημα Π' ∈ NP, Π' $\leq_p$ Π. ?????!!!



Προβλήματα NP-Complete (4)

Ευτυχώς έχουμε το

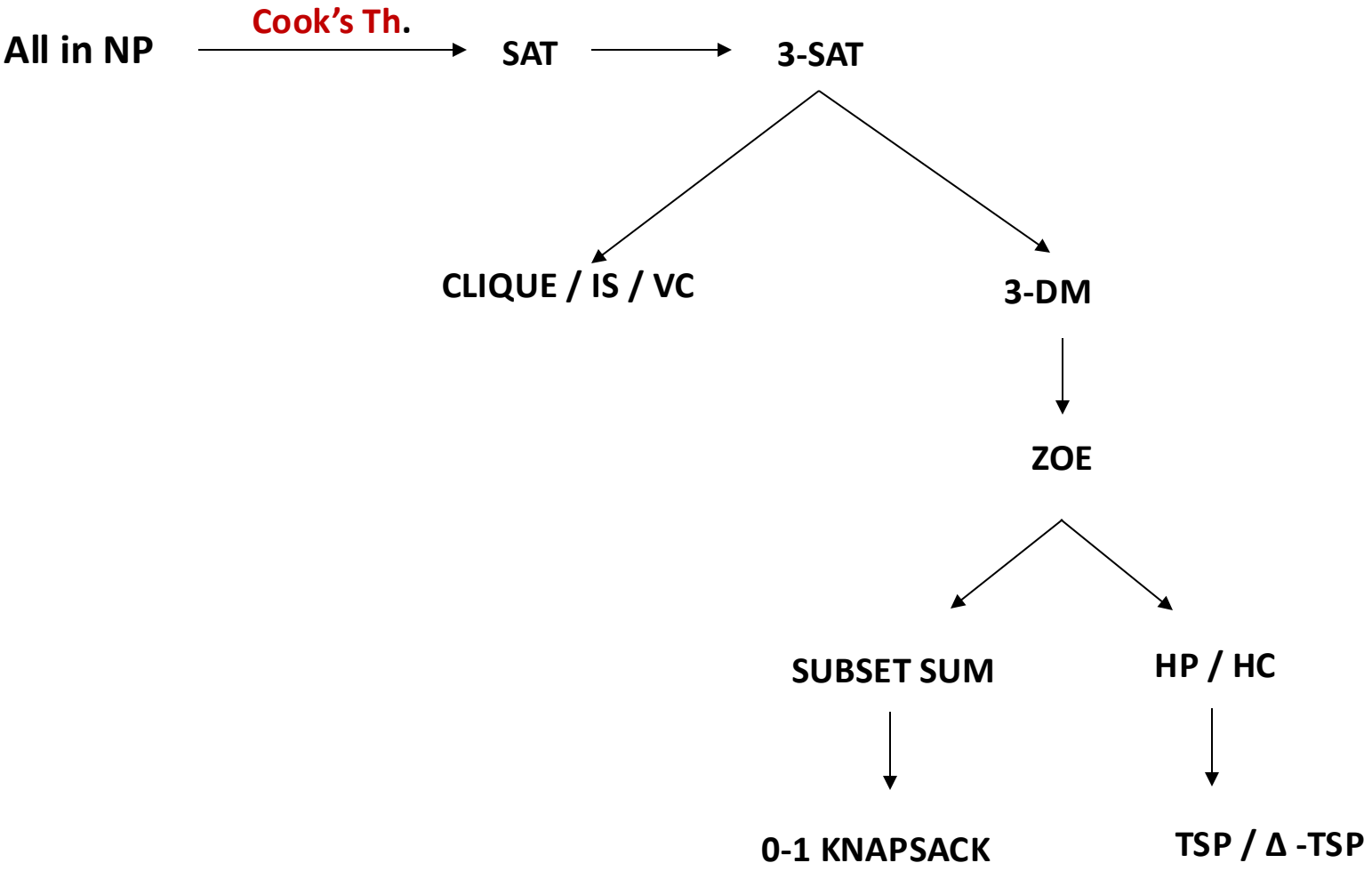
> Cook's Theorem [Cook 1971, Levin 1973]:
> SAT is NP–complete $(\forall\ \Pi' \in NP :\ \Pi' \leq_p SAT)$

Και επίσης ισχύει ότι η σύνθεση πολυωνυμικών αναγωγών είναι πολυωνυμική.

Πώς μπορώ να αποδείξω ότι ένα πρόβλημα Π είναι NP-Complete;

1. Απέδειξε ότι $\Pi \in NP$
2. Διάλεξε ένα γνωστό NP-Complete πρόβλημα Π’ και δείξε $\Pi' \leq_p \Pi$.
   (τότε για κάθε $\Pi'' \in NP$: $\Pi'' \leq_p \Pi' \leq_p \Pi$)

Προβλήματα NP-Complete (5)



Παράδειγμα πολυωνυμικής αναγωγής HC $\leq_p$TSP

HAMILTONIAN CYCLE (HC) Πρόβλημα

Είσοδος: Γράφος G=(V,E) χωρίς βάρη

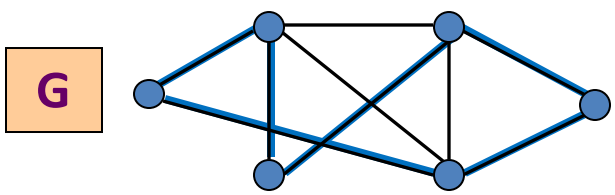
Ερώτηση: Υπάρχει κύκλος στο γράφο που περνάει από κάθε κορυφή ακριβώς μία φορά;

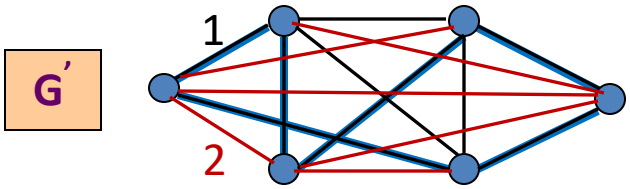
Traveling Salesman Problem (TSP)

Είσοδος: Πλήρης γράφος G'=(V',E') με βάρη και τιμή K

Ερώτηση: Υπάρχει κύκλος στο γράφο που περνάει από κάθε κορυφή ακριβώς μία φορά και έχει μήκος $\leq$ K;

Αναγωγή: Από το γράφο G=(V,E) κατασκευάζω πλήρες γράφο G' με βάρη w(e): G' = (V, E'), E' = V × V, w(e) τίθεται 1, αν $e \in E$, αλλιώς τίθεται 2. K = |V|





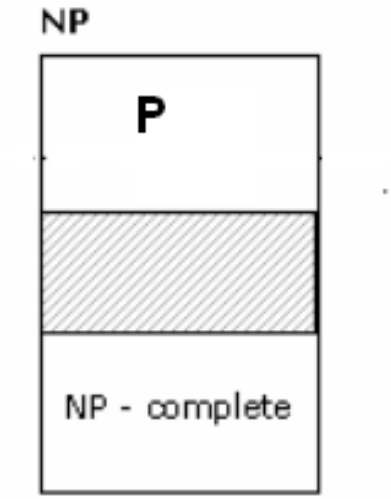
G' has a tour of cost $\leq$ K iff G has a HC

- If G' has a tour of cost $\leq$ K, it uses only edges of cost 1, so G has a HC
- If G has a HC then this is a tour of cost=K in G'

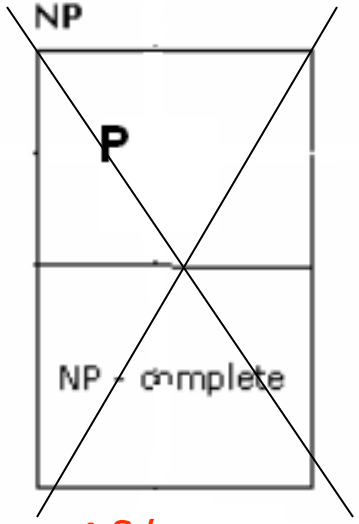
P vs NP

Ποιος είναι ο χάρτης για την κλάση NP?

Θυμηθείτε ότι P ⊆ NP



Αυτό πιστεύουμε



Αδύνατο
Ladner's theorem

Απίθανο

Η ΕΡΩΤΗΣΗ στη Θεωρητική Πληροφορική

Για κανένα NP-complete πρόβλημα δεν είναι γνωστός πολυωνυμικός αλγόριθμος.
Εάν υπάρχει για ένα, υπάρχει για όλα ! Τότε P = NP !!

Για όποιον/α ενδιαφέρεται να ψάξει περισσότερο

- Τι σημαίνει η κλάση των προβλημάτων NP-Hard;
  - Ποια η σχέση τους με τα προβλήματα στην κλάση NP-Complete;
  - Αναφέρατε 2 παραδείγματα προβλημάτων που είναι στην κλάση NP-Hard και 2 παραδείγματα προβλημάτων που είναι στην κλάση NP-Complete
- Κβαντική Υπολογιστική (Quantum computing): Ποια η σχέση της με το Church-Turing thesis;