Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Εθνικό Μετσόβιο Πολυτεχνείο
Ροή Λ, 7ο εξάμηνο

# Αλγόριθμοι και Πολυπλοκότητα

## Δεύτερη σειρά γραπτών ασκήσεων

**Σπουδαστής**
Παπασκαρλάτος Αλέξανδρος (Α.Μ.: 03111097)

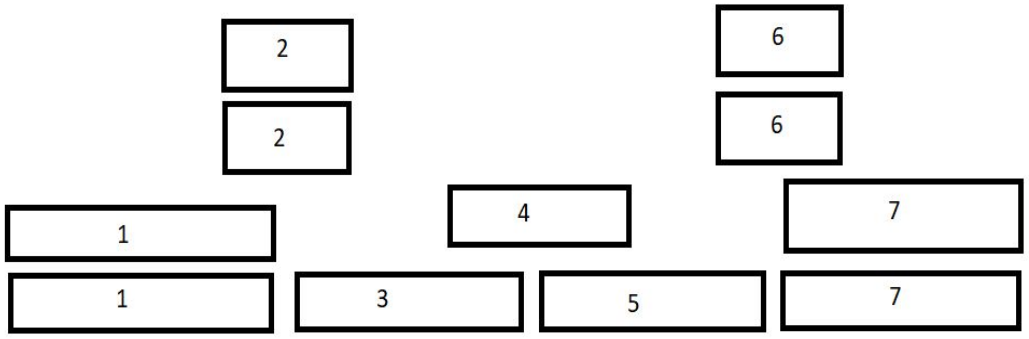
**Ημερομηνία Υποβολής:** 28 Νοεμβρίου 2018

# Άσκηση 2.1: “Επιλογή και κάλυψη μαθημάτων”

## α.1

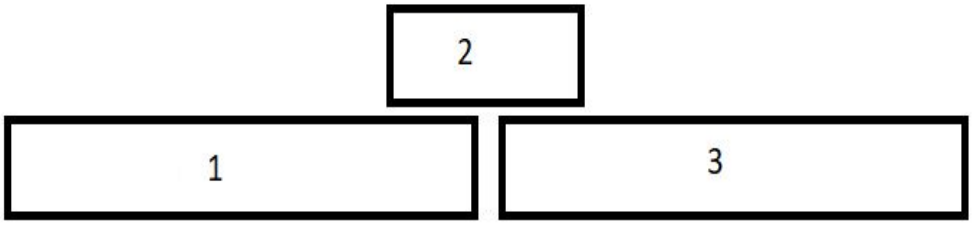
Καμία από τις προτεινόμενες μεθόδους δεν καταλήγει στη βέλτιστη λύση.
Θα το αποδείξουμε παρουσιάζοντας ένα αντιπαράδειγμα στην κάθε περίπτωση.
Τα αντιπαραδείγματα παρουσιάζονται οπτικά. Η σημασία τους είναι προφανής.
Όσα διαστήματα είναι ισοδύναμα, θα αντιστοιχούνται στον ίδιο δείκτη (δεν υπάρχει λόγος να τα ξεχωρίσουμε για τα ζητούμενα αυτής της άσκησης).

1. Λιγότερες επικαλύψεις:



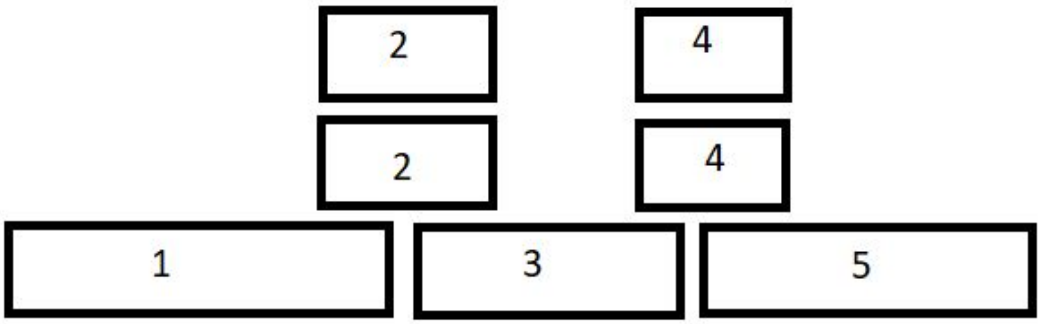

Η βέλτιστη λύση: |n| = 4, διαστήματα: 1, 3, 5, 7
Η λύση του αλγορίθμου: |n| = 3, διαστήματα: 1, 4, 6 / 1, 4, 7 / 2, 4, 6 / 2, 4, 7
(Προκύπτουν πολλές ενδεχόμενες ακολουθίες ανάλογα με το πως σπάμε τις ισοπαλίες)

2. Μεγαλύτερη διάρκεια:




Η βέλτιστη λύση: |n| = 2 διαστήματα: 1, 3
Η λύση του αλγορίθμου: |n| = 1 διαστήματα: 2

3. Περισσότερες επικαλύψεις:




Η βέλτιστη λύση: |n| = 3 διαστήματα: 1, 3, 5
Η λύση του αλγορίθμου: |n| = 2 διαστήματα: 1, 4 / 1, 5 / 2, 4 / 2, 5
(Προκύπτουν πολλές ενδεχόμενες ακολουθίες ανάλογα με το πως σπάμε τις ισοπαλίες)

**α.2**

**Ιδέα του αλγορίθμου**

Θα λύσουμε το πρόβλημα με δυναμικό προγραμματισμό.
Θα ταξινομήσουμε τα μαθήματα ως προς το χρόνο λήξης.

Για να δούμε αν συμφέρει να πάρουμε ένα μάθημα i, αρκεί να συγκρίνουμε το κέρδος που έχουμε με το να το πάρουμε, αφαιρώντας όσα μαθήματα επικαλύπτονται, με το κέρδος που έχουμε αν δεν το πάρουμε.

Παρακάτω, ο πίνακας C θα περιέχει για κάθε στοιχείο i το ακριβώς προηγούμενο μάθημα ως προς το χρόνο λήξης f με το οποίο δεν υπάρχει επικάλυψη.
Χρησιμεύει ώστε για το μάθημα i να βρίσκουμε το κέρδος αναδρομικά από το κέρδος του ακριβώς προηγούμενου μαθήματος με το οποίο δεν έχει επικάλυψη.
Τα βήματα (ι)-(ν) δημιουργούν αυτόν τον πίνακα.

Ο πίνακας D θα είναι η βάση του δυναμικού προγραμματισμού που υλοποιούμε. Από εκεί θα προκύπτει το συνολικό κέρδος σε διδακτικές μονάδες και κατ' επέκταση η βέλτιστη συλλογή μαθημάτων.

## Αλγόριθμος

**i.** Ταξινομούμε τα μαθήματα σε αύξουσα σειρά χρόνου ολοκλήρωσης σε πίνακα A. Θέτουμε ένα δείκτη a στο πρώτο στοιχείο.

**ii.** Ταξινομούμε τα μαθήματα σε αύξουσα σειρά χρόνου εκκίνησης σε πίνακα B. Θέτουμε ένα δείκτη b στο πρώτο στοιχείο.

**iii.** Δημιουργούμε ένα πίνακα C[1...n].
Κινούμε τον δείκτη b προς τα δεξιά μέχρις ότου να φτάσουμε σε μάθημα i με $s_i > f_1$ , όπου $f_1$ η πρώτη τιμή του A (στην οποία δείχνει ο a). Για όσα στοιχεία j περάσαμε με τον b, με $1 \leq j < i$, θέτουμε C[j] = -1.

**iv.** Κινούμε το δείκτη b προς τα δεξιά μέχρις ότου βρούμε είτε φτάσουμε στα όρια του πίνακα B ή βρούμε i με $s_i > f_{k+1}$ , όπου $f_k$ το στοιχείο στον A στο οποίο δείχνει ο a και $f_{k+1}$ το επόμενο. Για κάθε στοιχείο που περνάμε, θέτουμε C[j] = k.

**v.** Κινούμε το δείκτη a μία θέση δεξιά.
Αν φτάσαμε στο τελευταίο στοιχείο του πίνακα, τερματίζουμε.
Διαφορετικά, επαναλαμβάνουμε τη διαδικασία από το βήμα (iv).

**vi.** Για μάθημα i, το C[i] είναι το ακριβώς προηγούμενο μη επικαλύπτον μάθημα.
Δημιουργούμε πίνακα D[1...n].
Διατρέχουμε από αριστερά προς τα δεξιά τον πίνακα A και γεμίζουμε τον πίνακα D ως εξής:
**$D[i] = \max \{ D[i-1], w_i + D[C[i]] \}$** . Επίσης, είναι για i=1, $D[1] = w_1$ και για C[i] = -1, D[-1] = 0.

**vii.** Το D[n] θα έχει το μέγιστο αριθμό διδακτικών μονάδων που μπορούμε να πάρουμε. Προκειμένου να βρούμε τη ζητούμενη συλλογή μαθημάτων, διατρέχουμε τον D από τα δεξιά προς τα αριστερά και αν για κάποιο i, ισχύει πως D[i] > D[i-1], εισάγουμε το μάθημα i στη συλλογή μας και συνεχίζουμε στη θέση C[i].

## Πολυπλοκότητα

Έχουμε δύο ταξινομήσεις σε χρόνο O(nlogn).
Όλες οι υπόλοιπες διεργασίες γίνονται με κινήσεις δεικτών. Σε κάθε κίνηση δείκτη γίνονται O(1) διεργασίες και έχουμε συνολικά O(n) κινήσεις δεικτών.
Τελικά, προκύπτει χρονική πολυπλοκότητα O(nlogn).

**1β**

**Ιδέα του αλγορίθμου**

Θα εφαρμόσουμε έναν άπληστο αλγόριθμο πολυπλοκότητας O (nlogn).
Η πολυπλοκότητα προκύπτει από τις ταξινομήσεις για $s_i$ και $f_i$.

Η ιδέα είναι πως θέλουμε να πηγαίνουμε την τελευταία δυνατή στιγμή που μας επιτρέπεται.
Συνεπώς, κάθε φορά θα πηγαίνουμε ακριβώς πριν τελειώσει το μάθημα που τελειώνει πρώτο.
Τότε θα διαγράφουμε τόσο αυτό το μάθημα όσο και όλα τα υπόλοιπα που συμβαίνουν εκείνη τη στιγμή.
Επαναλαμβάνουμε τη διαδικασία για το νέο σύνολο μετά τις διαγραφές, μέχρι να διαγράψουμε όλα τα μαθήματα.

**Αλγόριθμος**

**i.** Ταξινομούμε τα μαθήματα ως προς το χρόνο λήξης $f_i$ σε πίνακα Α.
Θέτουμε δείκτη a που δείχνει στο πρώτο στοιχείο του πίνακα Α.

**ii.** Ταξινομούμε τα μαθήματα ως προς το χρόνο έναρξης $s_i$ σε πίνακα Β.
Θέτουμε δείκτη b που δείχνει στο πρώτο στοιχείο του πίνακα Β.

**iii.** Λαμβάνουμε την τιμή f που δείχνει ο δείκτης a.
Πηγαίνουμε στα Νέα Κτήρια λίγο πριν τη στιγμή f (στιγμή f-ε με ε>0 μια πολύ μικρή ποσότητα).

**iv.** Διαγράφουμε από το σύνολο μας όλα τα μαθήματα που διδάσκονται εκείνη τη στιγμή.
Συγκεκριμένα, μετακινούμε το δείκτη b μία μία θέση μέχρι να βρούμε μάθημα i με $s_i \geq f$.
Καθώς κινείται ο δείκτης διαγράφουμε τα μαθήματα που συναντάμε καθώς για τέτοιο μάθημα j, $s_j < f$.
Προκειμένου η “διαγραφή” να γίνει αποδοτικά, μπορούμε από μεριάς υλοποίησης, όταν “διαγράφουμε” ένα μάθημα i, να μηδενίζουμε το αντίστοιχο $f_i$ στον πίνακα F.

**v.** Μετακινούμε το δείκτη a μία θέση δεξιά.
Αν ξεπεράσαμε τα όρια του πίνακα Α, τερματίζουμε.
Αν το νέο $f_i$ ισούται με 0, δηλαδή το αντίστοιχο μάθημα i έχει ήδη διαγραφεί, επαναλαμβάνουμε το βήμα (v) (δηλαδή περνάμε στο επόμενο στοιχείο).
Διαφορετικά, επαναλαμβάνουμε τη διαδικασία από το βήμα (iii).

**Ορθότητα**

Καταρχάς, αναφέρουμε πως η λύση μας είναι έγκυρη, με την έννοια πως ικανοποιεί όλα τα μαθήματα. Είναι προφανές, καθώς σε κάθε βήμα, ελέγχουμε το σύνολο των εναπομείναντων μαθημάτων και πάμε στα Νέα Κτήρια, προτού τελειώσει το πρώτο μάθημα εξ' αυτών.

Θα αποδείξουμε πως η λύση μας είναι βέλτιστη.

Έστω μια βέλτιστη λύση OPT και η λύση του αλγορίθμου μας SOL.
Θα δείξουμε πως η λύση μας είναι (τουλάχιστον) ισοδύναμη με την OPT.
Θεωρούμε πως οποιεσδήποτε στιγμές διαγράφουν ακριβώς τα ίδια επικαλυπτόμενα μαθήματα στις δύο λύσεις είναι ισοδύναμες.

Αν οι λύσεις ταυτίζονται, εξαιρετικά, τελειώσαμε.
Διαφορετικά, χρησιμοποιούμε το επιχείρημα ανταλλαγής.
Διακρίνουμε περιπτώσεις:

**α.** Αν οι λύσεις διαφέρουν για πρώτη φορά (σε χρονική σειρά) στη στιγμή επίσκεψης $t_{OPT}$ που υπάρχει στην OPT και όχι στη SOL, έχουμε τα εξής:
Επειδή, όλες οι προηγούμενες στιγμές επίσκεψης είναι ίδιες (ή καλύτερα ισοδύναμες), μέχρι πριν την $t_{OPT}$ έχουν διαγραφεί ακριβώς τα ίδια μαθήματα.
Άρα ακριβώς πριν την $t_{OPT}$ έχουμε το ίδιο σύνολο εναπομείναντων μαθημάτων.

Έστω $t_{SOL}$ η πρώτη επόμενη στιγμή επίσκεψης στη SOL, μετά από τη στιγμή $t_{OPT}$ .
Αν υπάρχει μάθημα k που ικανοποιείται από την $t_{OPT}$ και όχι από την $t_{SOL}$ , αυτό σημαίνει πως το μάθημα τελειώνει πριν τη στιγμή $t_{SOL}$.
Όμως αυτό είναι άτοπο, καθώς η $t_{SOL}$ είναι η στιγμή ακριβώς πριν τελειώσει το μάθημα που τελειώνει πρώτο.

Αν υπάρχει μάθημα που ικανοποιείται από την $t_{SOL}$ και όχι την $t_{OPT}$ , μπορούμε απλά στην OPT να ακυρώσουμε την $t_{SOL}$ και να πάρουμε αντ' αυτού τη στιγμή $t_{SOL}$ .
Επειδή, η $t_{SOL}$ διαγράφει τα ίδια και επιπλέον τουλάχιστον 1 μάθημα, η λύση βελτιώνεται (ή τουλάχιστον παραμένει ισοδύναμη).

**β.** Αν οι λύσεις διαφέρουν για πρώτη φορά (σε χρονική σειρά) στη στιγμή επίσκεψης $t_{SOL}$ που υπάρχει στην OPT και όχι στη SOL, έχουμε τα εξής:
Επειδή, όλες οι προηγούμενες στιγμές επίσκεψης είναι ίδιες (ή καλύτερα ισοδύναμες), μέχρι πριν την $t_{SOL}$ έχουν διαγραφεί ακριβώς τα ίδια μαθήματα.
Άρα ακριβώς πριν την $t_{SOL}$ έχουμε το ίδιο σύνολο εναπομείναντων μαθημάτων.

Όμως σε αυτό το σημείο, η $t_{SOL}$ είναι η τελευταία δυνατή στιγμή πριν τελειώσει το πρώτο μάθημα, έστω k.
Αυτό σημαίνει, πως αν η OPT δεν περιέχει στιγμή που να ικανοποιεί το k πριν την $t_{SOL}$ (όπως και υποθέσαμε σε αυτήν την περίπτωση), τότε το k τελειώνει χωρίς να ικανοποιηθεί από την OPT.
Άτοπο, καθώς η OPT είναι βέλτιστη λύση, άρα είναι τουλάχιστον έγκυρη λύση.

**Πολυπλοκότητα**

Για τη δημιουργία των πινάκων A και B, χρειαζόμαστε δύο ταξινομήσεις σε χρόνο O(nlogn).

Με κάθε μετακίνηση του δείκτη b, έχουμε μια σύγκριση και μία “διαγραφή” σε χρόνο O(1). Έχουμε το πολύ n τέτοιες μετακινήσεις.

Επιπλέον, έχουμε n μετακινήσεις του δείκτη a, αλλά οι διεργασίες που επιτελούνται προσμετρήθηκαν στις διεργασίες που επιτελούνται όταν κινείται ο δείκτης b. Ο δείκτης a χρησιμεύει για να δίνει την τιμή ενάντια στην οποία συγκρίνονται οι τιμές που δείχνει ο b.

Τελικά, έχουμε πολυπλοκότητα O(nlogn + n) = **O(nlogn)**.

# Άσκηση 2.2: "Τηλεπαιχνίδι"

## Ιδέα του αλγορίθμου

Θα διαλέγουμε κάθε φορά το μικρότερο σφυρί (δηλαδή το σφυρί που διαθέτει την ελάχιστη δύναμη).
Θα παίρνουμε το σύνολο όλων των κουτιών που αυτό το σφυρί μπορεί να σπάσει και θα σπάμε αυτό που έχει το μεγαλύτερο κέρδος εξ' αυτών.
Προκειμένου να το υλοποιήσουμε αποδοτικά, θα χρησιμοποιήσουμε ταξινομήσεις και ένα σωρό (ζήτω οι σωροί!).

Σημείωση
Γνωρίζω πως υπάρχει χρονικά ισοδύναμη λύση που ταξινομεί τα κουτιά ως προς το κέρδος και μετά ψάχνει το μικρότερο σφυρί που σπάει το πιο κερδοφόρο κουτί.
Ωστόσο, εκείνη η λύση δε χρησιμοποιεί σωρό και οι σωροί είναι υπέροχοι.

## Αλγόριθμος

**i.** Ταξινομούμε τα σφυριά σε αύξουσα σειρά δύναμης $f_i$ σε έναν πίνακα A.
Ορίζουμε δείκτη a που αρχικά δείχνει στο πρώτο στοιχείο του A.

**ii.** Ταξινομούμε τα κουτιά σε αύξουσα σειρά αντίστασης $p_i$ σε έναν πίνακα B.
Ορίζουμε δείκτη b που αρχικά δείχνει στο πρώτο στοιχείο του B.

**iii.** Ορίζουμε (αρχικά κενό) σωρό μεγίστου H, στον οποίο θα αποθηκεύουμε κουτιά με βάση την αξία τους $v_i$ .

**iv.** Λαμβάνουμε το σφυρί που δείχνει ο δείκτης a. Πρόκειται για το μικρότερο που δεν έχει χρησιμοποιηθεί ήδη.

**v.** Μετακινούμε το δείκτη b προς τα δεξιά μία μία θέση μέχρις ότου βρούμε κουτί με αντίσταση μεγαλύτερη από τη δύναμη του σφυριού που εξετάζουμε (ή φτάσουμε στα όρια του πίνακα). Καθώς μετακινούμε το δείκτη b, εισάγουμε τα κουτιά από τα οποία περνάμε στο σωρό μεγίστου H με βάση την αξία τους.

**vi.** Όταν τελειώσει η κίνηση του δείκτη b, αν ο σωρός είναι άδειος πετάμε το σφυρί στα σκουπίδια. Αν ο σωρός δεν είναι άδειος, χρησιμοποιούμε το σφυρί για να σπάσουμε το κουτί που είναι στη ρίζα του σωρού. Διαγράφουμε τη ρίζα του σωρού και παίρνουμε την αξία του.

**vii.** Μετακινούμε το δείκτη a μία θέση δεξιά (ώστε να δείχνει το αμέσως επόμενο σε δύναμη σφυρί). Αν ξεπεράσαμε τα όρια του πίνακα A, τερματίζουμε. Διαφορετικά, επαναλαμβάνουμε τη διαδικασία από το βήμα (iv).

**Ορθότητα**

Καταρχάς, αναφέρουμε πως, επειδή ταξινομούμε τα σφυριά και στο σωρό εισάγουμε μονάχα όσα κουτιά μπορούν να σπάσουν από το μικρότερο αχρησιμοποίητο σφυρί, άρα και από όλα τα επόμενα, δεν κάνουμε αντικανονικές κινήσεις.
Συνεπώς, ο αλγόριθμός μας παράγει μια έγκυρη λύση.

Θα δείξουμε πως η λύση μας είναι βέλτιστη.

Έστω μια βέλτιστη λύση OPT και η λύση του αλγορίθμου μας SOL.
Θα δείξουμε πως η λύση μας είναι (τουλάχιστον) ισοδύναμη με την OPT.

Αν οι λύσεις ταυτίζονται, εξαιρετικά, τελειώσαμε.
Διαφορετικά, θα υπάρχει τουλάχιστον ένα σφυρί που χρησιμοποιείται διαφορετικά.

Έστω τα σφυριά ταξινομημένα σε αύξουσα σειρά ανά μέγεθος.
Έστω i το πρώτο σφυρί σε αυτή τη διάταξη που χρησιμοποιείται διαφορετικά στις δύο λύσεις.
Διακρίνουμε περιπτώσεις:

α) Στην OPT λύση το i σπάει κουτί k* ενώ το στη SOL το i σπάει κουτί k με $u_{k^*} > u_k$ .
Άτοπο, καθώς στη λύση μας, το σφυρί i σπάει το κουτί με το μέγιστο κέρδος, από όλα τα εναπομείναντα κουτιά που μπορεί να σπάσει. Επειδή όλα τα προηγούμενα σφυριά έχουν χρησιμοποιηθεί όμοια στις δύο λύσεις, έχουν απομείνει ακριβώς τα ίδια κουτιά για την ώρα.

β) Στην OPT λύση το i σπάει κουτί k* ενώ το στη SOL το i πετιέται αχρησιμοποίητο .
Άτοπο, αντίστοιχα με το (α), αν μπορούμε να σπάσουμε κουτί, θα σπάσουμε κουτί.

γ) Στην OPT το σφυρί i πετιέται αχρησιμοποίητο ή σπάει κουτί k*, ενώ στη SOL το ι σπάει κουτί k $u_k \geq u_{k^*}$.. Επίσης, έστω, πως το κουτί k δε σπάει καθόλου στην OPT.
Τότε, αν στην OPT χρησιμοποιήσουμε αντ' αυτού το σφυρί i στο κουτί k, αυξάνουμε (ή παίρνουμε το ίδιο) κέρδος, χωρίς να επηρεάσουμε τη χρήση κανενός άλλου σφυριού.

δ) Στην OPT το σφυρί i πετιέται αχρησιμοποίητο ή σπάει κουτί k*, ενώ στη SOL το ι σπάει κουτί k $u_k \geq u_{k^*}$. Επίσης έστω, πως το κουτί k σπάει από άλλο σφυρί j στην OPT.
Είναι $f_j \geq f_i$ , καθώς όπως είπαμε το i είναι το πρώτο (σε αύξουσα σειρά δύναμης) σφυρί που χρησιμοποιήθηκε διαφορετικά στις δύο λύσεις. Λοιπόν, και τα δύο σφυριά i, j είναι ικανά να σπάσουν τόσο το k, όσο και το k*.
Τότε, αν στην OPT χρησιμοποιήσουμε αντ' αυτού το σφυρί i στο k, και το σφυρί j στο k*, (ή αν στην αρχική OPT το i πετιόταν, πετάξουμε αντ' αυτού το j), καταλήγουμε σε ισοδύναμη λύση.

Επαναλαμβάνοντας τα (γ),(δ) όσες φορές χρειάζεται φτάνουμε από τη βέλτιστη λύση OPT στη λύση του αλγορίθμου μας SOL με (τουλάχιστον) ισοδύναμη τελική αξία.

**Πολυπλοκότητα**

Οι δύο ταξινομήσεις γίνονται σε χρόνο O(nlogn).
Με κάθε μετακίνηση του δείκτη b, εισάγουμε ένα στοιχείο σε σωρό σε χρόνο O(logn) καθώς δε γίνεται να έχουμε περισσότερα από n στοιχεία στο σωρό. Γίνονται το πολύ n τέτοιες μετακινήσεις, άρα το πολύ n εισαγωγές.
Με κάθε μετακίνηση του δείκτη a, διαγράφουμε τη ρίζα του σωρού (αν ο σωρός δεν είναι άδειος) σε χρόνο O(logn). Γίνονται n τέτοιες μετακινήσεις, άρα το πολύ n διαγραφές ρίζας.

Συνολικά, προκύπτει χρονική πολυπλοκότητα **O(logn)**.

# Άσκηση 2.3: “Αναμνηστικά”

## Ιδέα του αλγορίθμου

Το πρόβλημα μας θυμίζει το knapsack. Θα εφαρμόσουμε δυναμικό προγραμματισμό.

Ξεκινώντας από τον ελάχιστο δυνατό προϋπολογισμό C, θα εξετάζουμε ποια αντικείμενα θα δώσουν τη μεγαλύτερη αξία χωρίς να ξεπερνάμε το C. Θα αυξάνουμε μοναδιαία τον προϋπολογισμό και θα επαναλαμβάνουμε.

Επειδή, από εκφώνηση, γνωρίζουμε πως έχουμε αρκετά χρήματα για να αγοράσουμε το φθηνότερο σουβενίρ από κάθε χώρα, δηλαδή $C \geq \Sigma(\phi\theta)$, όπου $\Sigma(\phi\theta)$ το αθροιστικό κόστος των φθηνότερων αντικειμένων, θα αναδιαμορφώσουμε τα μεγέθη ως εξής:

Από εδώ και στο εξής, όπου γράφουμε “C” θα εννοούμε την τιμή “C-Σ(φθ)”.
Όπου γράφουμε “$c_{ij}$” θα εννοούμε την τιμή “$c_{ij} - c_{i\,\phi\theta}$”, όπου $c_{i\,\phi\theta}$ το φθηνότερο αντικείμενο της χώρας i.
Όπου γράφουμε “$p_{ij}$” θα εννοούμε την τιμή “$p_{ij} - p_{i\,\phi\theta}$”, όπου $p_{i\,\phi\theta}$ το φθηνότερο αντικείμενο της χώρας i.

Σημειώνουμε πως οι παραπάνω αναδιαμορφώσεις δεν είναι παραδοχές, ούτε υποθέσεις.
Πρόκειται απλά για θέμα διατύπωσης και υλοποίησης.
Ο αλγόριθμός μας θα κάνει ακριβώς αυτό που ζητάει η εκφώνηση.
Οι αναδιαμορφώσεις αυτές μπορούν να γίνουν εύκολα σε γραμμικό χρόνο.

Ουσιαστικά, λέμε πως έχουμε ως κάτω φράγμα την επιλογή όλων των φθηνότερων αντικειμένων, και ορίζουμε τα κόστη και τις αξίες ως διαφορές από αυτήν την επιλογή. Διευκολύνει λίγο κάποια τεχνικά σημεία του αλγορίθμου μας.

Περισσότερα για την επεξήγηση του αλγορίθμου στην ενότητα “Ορθότητα” παρακάτω.

**<u>Αλγόριθμος</u>**

Δημιουργούμε πίνακα μεγέθους n x C.
Θα γεμίσουμε τον πίνακα στήλη στήλη από το 1 εώς το C, όπου κάθε στήλη γεμίζεται πρώτα ανά σειρά από το 1 εώς το n, δηλαδή:
(1,1), (2,1), …, (n,1), (1,2), (2,2)...(n,2),...,(1,C), (2,C),...,(n,C)

Για κάθε χώρα $1 \leq i \leq n$, με αναμνηστικά $1 \leq j \leq k_j$,
και παρόν προϋπολογισμό c, με $0 \leq c \leq C$, έχουμε:

$$\mathbf{P(i,c) = max_{1 \leq j \leq k} \{ p_{ij} + P(i-1, c-c_{ij}) \},}$$

όπου αν $c - c_{ij} < 0$, δε λαμβάνουμε υπ' όψιν το αντίστοιχο j.

Σε κάθε τέτοιο κελί επίσης κρατάμε το στοιχείο j που επιλέγεται.

Θυμίζουμε πως με τις αναδιαμορφώσεις που κάναμε στην αρχή, (τουλάχιστον) ένα αντικείμενο (το φθηνότερο) σε κάθε χώρα θα έχει μηδενικό κόστος και αξία.

Η τελική αξία προκύπτει από το P(n,C).

Προκειμένου να βρούμε την τελική συλλογή αναμνηστικών, πρέπει να δούμε "αναδρομικά" ποιο σουβενίρ επιλέχθηκε από κάθε χώρα.
Ξεκινάμε από το κελί P(n,C), παίρνουμε το αναμνηστικό (n,j) της χώρας n (το οποίο είπαμε πως έχουμε αποθηκεύσει). Έπειτα πηγαίνουμε στο κελί $P(n-1,C-c_{nj})$, παίρνουμε το αναμνηστικό της χώρας n-1, κοκ. Δηλαδή κάθε φορά το i μειώνεται κατά 1 και το c κατά το κόστος του αναμνηστικού που επιλέξαμε.
Τελειώνουμε σε κελί της χώρας 1.

## Ορθότητα

Η ορθότητα προκύπτει από αρχή της βελτιστότητας και έγκυρη αναδρομική σχέση.

Αρκεί να δείξουμε πως η αναδρομική σχέση είναι έγκυρη και λογική.
Έστω έχουμε όλες τις πληροφορίες για όλους τους δυνατούς προϋπολογισμούς για το σύνολο χωρών [1...n-1] και τις αντίστοιχες βέλτιστες λύσεις των υποπροβλημάτων.

Για να δούμε ποιο σουβενίρ θα διαλέξουμε από τη χώρα n, θα πρέπει να εφαρμόσουμε τη σχέση $\mathbf{P(n,C) = max_{1 \le j \le k} \{ p_{nj} + P(n-1, C-c_{ij}) \},}$ ώστε να δούμε ποιο δίνει τη βέλτιστη αξία χωρίς να ξεφεύγουμε από τον προϋπολογισμό C.

Αντίστοιχα με το knapsack, εξετάζουμε ποιο εξ' αυτών θα μας δώσει καλύτερη αξία αν το "χωρέσουμε" στο μπάτζετ μας.
Όμως, σε αντίθεση με το knapsack, οφείλουμε να διαλέξουμε ακριβώς ένα εξ' αυτών. Δεν επιτρέπεται να γυρίσουμε με άδεια χέρια από τη χώρα.

Επειδή όλα τα υποπροβλήματα λύθηκαν bottom-up με λογικές αρχικοποιήσεις και έγκυρη αναδρομική, ο αλγόριθμος καταλήγει σε σωστό αποτέλεσμα.

Ειλικρινά, ίσως αυτή η εξήγηση να μην αρκεί, αλλά δεν ξέρω πως να αιτιολογήσω την παραπάνω σχέση, πέρα από το "προφανώς, έτσι έχουν τα πράγματα".

## Πολυπλοκότητα

Γεμίζουμε n*C κελιά πίνακα.
Για κάθε κελί της γραμμής i, εξετάζουμε $k_i$ στοιχεία (το καθένα σε χρόνο O(1) ) που αντιστοιχούν στα $k_i$ δυνατά σουβενίρ από τη χώρα i.
Συνεπώς, συνολικά προκύπτει πολυπλοκότητα **O(m*C)**, όπου **m =** $\sum_{i=1}^{n} k_i$ , δηλαδή m είναι το πλήθος όλων των αντικειμένων συνολικά.

Θυμίζουμε και πάλι πως ως C εννοούμε στην πραγματικότητα την ποσότητα C - Σ(φθ).

## Βελτιώσεις στον αλγόριθμο

Μια πολύ μικρή βελτίωση στον αλγόριθμο θα ήταν να μην υπολογίσουμε ολόκληρη την τελευταία γραμμή του πίνακα, παρά μόνο το τελευταίο κελί της.
Αυτό είναι εφικτό γιατί τα κελιά (n,c) με $0 \le c \le C-1$, δεν επιδρούν στην τιμή κανενός άλλου στοιχείου.

# Άσκηση 2.4: "Σοκολατάκια"

Σύμφωνα με διευκρίνηση του διδάσκονται, θεωρούμε πως όλα τα κουτιά έχουν διαφορετικό πλήθος σοκολατακίων (σοκολατακιών;).

## Ιδέα του αλγορίθμου

Θα επιλύσουμε το πρόβλημα με δυναμικό προγραμματισμό.
Η διαφορά από άλλες περιπτώσεις που έχουμε συναντήσει μέχρι τώρα είναι πως εδώ το κόστος κάθε επιλογής δεν είναι σταθερό, αλλά εξαρτάται από προηγούμενη επιλογή.

Μια άλλη διαφορά είναι πως, σε αντίθεση πχ με το knapsack, εδώ θέλουμε να φάμε τουλάχιστον Q σοκολατάκια (και όχι το πολύ).
Και πάλι θα ακολουθήσουμε τη λογική του knapsack, αλλά η πολυπλοκότητα βγαίνει λίγο περίεργη.

Θα ορίσουμε ως κόστος c[i] το κόστος σε χρόνο που μας παίρνει να φάμε όσο το δυνατόν περισσότερα σοκολατάκια στο σύνολο [1...i] (σύμφωνα με την προηγούμενη ταξινόμηση).

Θα θεωρήσουμε πως έχουμε κάθε φορά ένα σταθερό c το οποίο δεν μπορούμε να ξεπεράσουμε. Σε αντίθεση με το knapsack, όπου τερματίζουμε όταν ελέγξουμε το τελικό βάρος που μας δίνεται, εδώ, μην έχοντας δοθέν όριο στο c (είμαστε διατεθειμένοι να υπομείνουμε πολλά για να φάμε τα σοκολατάκια), θα αυξάνουμε μοναδιαία το c μέχρις ότου καταφέρουμε να φάμε Q σοκολατάκια.

## Αλγόριθμος

ι. Ταξινομούμε τα κουτιά σε αύξουσα σειρά σύμφωνα με το πόσα σοκολοτάκια περιέχουν.

ιι. Δημιουργούμε πίνακα Q[n, :] με n γραμμές και δυναμικό αριθμό στηλών.

ιιι. Γεμίζουμε τα κελιά της στήλης αξιοποιώντας την εξής σχέση:

$$Q[i,c] = \begin{cases} 0, & \text{αν } c_{ip} < c \\ \max_{1 \le j \le i} \{ Q[j, c-c_{ij}] + q_i \}, & \text{για } j: |c - c_{ij}| \ge 0 \text{ και το } j \text{ έχει διαφορετικό τύπο από το } i \\ q_i, & \text{αν } c_{ip} \ge c \text{ και δεν υπάρχει κανένα έγκυρο } j < i \end{cases}$$

όπου i είναι το κουτί προς εξέταση σε αύξουσα σειρά πλήθους σοκολατακιών,
c είναι το το παρόν όριό μας σε κούραση / χρόνο,
$q_i$ το πλήθος των σοκολατακιών που περιέχει το κουτί i,
Q[i,c] είναι το βέλτιστο συνολικό πλήθος με τελευταίο κουτί το i χωρίς να περνάμε το c,
$c_{ij}$ είναι το κόστος μετακίνησης από το i στο j και
$c_{ip}$ το κόστος μετακίνησης από το αρχικό κουτί (το οποίο ίσως να μη φάμε) στο i.

Σε κάθε κελί επίσης αποθηκεύουμε το j από το οποίο προέκυψε το max, διαφορετικά, αν η τιμή Q[i,c] προέκυψε από τις άλλες δύο συνθήκες, δεν αποθηκεύουμε τίποτα.

ιν. Αν σε κάποιο σημείο της στήλης φτάσαμε (ή ξεπεράσαμε) το Q της εκφώνησης, δηλαδή τη συνολική ποσότητα σοκολάτας που θα ικανοποιήσει την όρεξή μας, σταματάμε.
Αν ο αριθμός c (αριθμός των στηλών) έχει φτάσει το $n^2$, τερματίζουμε και αποφαινόμαστε πως το πρόβλημα δεν έχει έγκυρη λύση (βλ. παρακάτω "Πολυπλοκότητα - άνω φραγμα C").
Διαφορετικά, δημιουργούμε δυναμικά νέα στήλη και συνεχίζουμε τη διαδικασία από το βήμα (ιιι).

ν. Φτάνουμε εδώ αν για κάποιο όριο κούρασης c, καταφέραμε να φάμε Q σοκολατάκια.
Το c αυτό, δηλαδή ο αριθμός των στηλών που δημιουργήθηκαν είναι η ελάχιστη κούραση / χρόνος που χρειαστήκαμε. Το ορίζουμε c = C.

Προκειμένου να βρούμε τη συλλογή και τη σειρά των κουτιών που θέλουμε, ξεκινάμε από το κελί με αξία Q[i,C] ≥ Q, λαμβάνουμε το j που έχουμε επίσης αποθηκεύσει στο κελί στο βήμα (ιιι) και πηγαίνουμε στο κελί $Q[j, C-c_{ij}]$.
Συνεχίζουμε ούτω καθ' εξής μέχρι να φτάσουμε σε κουτί που δεν έχει αποθηκευμένο άλλο κουτί.
Εναλλακτικά, μπορούμε με την ίδια κίνηση, να κρατάμε το άθροισμα των σοκολατών από τα κουτιά από τα οποία περνάμε, μέχρι το άθροισμα να γίνει Q[i,C] (ως εναλλακτική συνθήκη τερματισμού που μας δίνει τη συλλογή, άπαξ και έχουμε βρει το Q[i,C] ).

**<u>Ορθότητα</u>**

Καταρχάς, αναφέρουμε πως η λύση μας είναι έγκυρη.

Ας επεξηγήσουμε την αναδρομική σχέση του βήματος (ιιι):

**α.** Σε κάθε κελί της γραμμής i, αρχικά εξετάζουμε αν μπορούμε να φτάσουμε σε αυτό το κελί από το αρχικό p με όριο κούρασης c.
Αν δεν μπορούμε καν να φτάσουμε από την αρχή, αυτό σημαίνει πως δεν μπορούμε να φτάσουμε καθόλου

**β.** Απαξ και δούμε πως μπορούμε να φτάσουμε, θεωρούμε πως το κουτί i είναι το τελευταίο κουτί στην ακολουθία μας και εξετάζουμε από όλα τα έγκυρα κουτιά, ποιο είναι το προτελευταίο.
Συνεπώς, παίρνουμε το κουτί j που μεγιστοποιεί το κέρδος, εφόσον j < i (ώστε να τηρηθεί η αύξουσα σειρά), $t_j \neq t_i$ ώστε να τηρηθεί η συνθήκη διαφορετικού τύπου σοκολάτας και $c_{ij} \leq c$, ώστε να μπορούμε να φτάσουμε από το j στο c χωρίς να ξεπεράσουμε το όριο χρόνου.

**γ.** Αν δεν υπάρχει κανένα τέτοιο j που να τηρεί όλες αυτές τις συνθήκες, αλλά μπορούμε να φτάσουμε στο i από το αρχικό p, απλά θεωρούμε το i ως το πρώτο και τελευταίο στοιχείο της ακολουθίας μας, που σημαίνει πως απλά τρώμε μόνο τα $q_i$ σοκολατάκια του κουτιού.

Η λύση μας είναι βέλτιστη, καθώς λύνουμε κάθε υποπρόβλημα βέλτιστα.

Σε κάθε κελί θα έχουμε την βέλτιστη αξία σε σοκολάτες που μπορούμε να πάρουμε χωρίς να ξεπερνάμε το όριο χρόνου c που έχουμε θέσει.
Αυτό από μόνο του σαν πρόβλημα μπορεί να θεωρηθεί απλή παραλλαγή του knapsack καθώς υπάρχουν έξτρα συνθήκες που συνδέουν τα αντικείμενα.

Επειδή κάθε στήλη που δημιουργείται θα περιέχει στο κάθε κελί της τη μέγιστη αξία που μπορούμε να πάρουμε τελειώνοντας στο αντίστοιχο κουτί και ελέγχουμε όλα τα κουτιά, κάθε στήλη θα έχει σε κάποιο κελί τη μέγιστη αξία σε σοκολατάκια που μπορούμε να φάμε χωρίς να ξεπεράσουμε το όριο c.

Λοιπόν, το c αυξάνεται μοναδιαία (από υλοποίηση) από τη μία στήλη στην επόμενη και σταματάμε όταν κάποιο κελί αποκτήσει αξία Q.
Αυτό σημαίνει πως στην προηγούμενη στήλη δεν είχαμε φτάσει την αξία Q (αφού δε σταματήσαμε), αλλά τώρα τη φτάσαμε.
Επομένως, αυτό το c είναι το ελάχιστο κόστος που χρειαζόμαστε για να φάμε Q σοκολατάκια.

**<u>Πολυπλοκότητα</u>**

Σε μία στήλη για το κελί i της στήλης, εξετάζουμε τα i-1 προηγούμενα κουτιά σε χρόνο O(1) το καθένα.
Συνεπώς, σε ολόκληρη τη στήλη έχουμε 1+2+3+...+n = $O(n^2)$ διεργασίες.
Έχουμε C τέτοιες στήλες.

Επομένως προκύπτει συνολική χρονική πολυπλοκότητα **$O(n^2 C)$**.
Το κακό με αυτήν την πολυπλοκότητα, είναι πως εξαρτάται από το ζητούμενο και όχι από την είσοδο. Δε μου αρέσει αλλά εντάξει.
Πάντως, το C έχει άνω φράγμα το $n^2$ καθώς τόσο χρόνο θα μας πάρει να περάσουμε από όλα τα κουτιά στη χείριστη περίπτωση διάταξης, ανάλογα με την είσοδο.
Συνεπώς, μπορούμε να πούμε πως έχουμε συνολική χρονική πολυπλοκότητα **$O(n^4)$**, αν και ούτε αυτό με ενθουσιάζει, γιατί στη μέση περίπτωση το C θα είναι μάλλον αρκετά πιο μικρό.

**<u>Σημείωση στον αλγόριθμο</u>**
Αφού ταξινομήσουμε τα κουτιά σε αύξουσα σειρά ανάλογα με το πλήθος των σοκολατακιών, μπορούμε να υπολογίσουμε το άθροισμα $\Sigma c = c_{p,1} + c_{1,2} + c_{2,3} + \ldots + c_{n-2,n-1} + c_{n-1,n}$ .
Δηλαδή το Σc, αντιστοιχεί στο χρόνο που θα μας έπαιρνε να φάμε όλα τα σοκολατάκια στη διάθεσή μας ξεκινώντας από το κουτί με το ελάχιστο πλήθος και καταλήγοντας σε αυτό με το μέγιστο (αγνοώντας τη συνθήκη για τον τύπο σοκολατών).

Αν θέλουμε να δημιουργήσουμε στατικό, αντί για δυναμικό πίνακα, θα είναι n x Σc.

# Άσκηση 2.5: "Πομποί και δέκτες"

## Ιδέα του αλγορίθμου

Καταρχάς, για κάθε κεραία i, θεωρούμε το κόστος ως το μέγεθος $\mathbf{K_i = T_i - R_i \geq 0}$ .
Στην ουσία, θεωρούμε πως για αν η κεραία i είναι δέκτης, τότε έχει "μηδενικό" κόστος, ενώ αν είναι πομπός έχει $K_i$ κόστος. Δηλαδή, το "μηδενικό" κόστος αντιστοιχεί στο κόστος που θα πρέπει να πληρώσουμε ούτως ή άλλως και το $K_i$ αντιστοιχεί στο επιπλέον κόστος που θα πρέπει να πληρώσουμε αν η κεραία είναι πομπός.

Προκύπτει πως αν έπρεπε να θέσουμε ένα πομπό και μπορούσαμε να επιλέξουμε ανάμεσα σε δύο κεραίες οι οποίες είναι "γεωγραφικά ισοδύναμες", θα μας συνέφερε να γίνει πομπός αυτή με το μικρότερο κόστος.

Σε κάθε βήμα του αλγορίθμου μας, εάν απαιτείται, θα θέτουμε ως πομπό την κεραία με το μικρότερο κόστος ανάμεσα σε όλες τις επιτρεπόμενες. Προκειμένου να το πετύχουμε αποδοτικά θα χρησιμοποιήσουμε σωρό ελαχίστου (ζήτω οι σωροί!).

Στον κάτωθι αλγόριθμο, θα θεωρούμε πως ένα στοιχείο μπορεί να τεθεί ως δέκτης ανν στα αριστερά του, υπάρχουν περισσότεροι πομποί απ' ότι δέκτες.
Αυτό συμβαίνει γιατί τότε υπάρχει τουλάχιστον ένας πομπός που δε δεσμεύεται από τους δέκτες στα αριστερά του.
Επειδή αυτό, εν τέλει θα ισχύει για όλους τους δέκτες, καταλήγουμε σε έγκυρη λύση.
(Περισσότερα στην ενότητα "Ορθότητα")

Στην πράξη, λόγω της μορφής του αλγορίθμου αυτό θα σημαίνει πως θα υπάρχει ακριβώς ένας παραπάνω πομπός στα αριστερά του.
Μάλιστα, αυτό θα συμβαίνει ανν το βήμα i είναι άρτιο, δηλαδή, όταν ο δείκτης a δείξει για πρώτη φορά στο στοιχείο 2, 4, 6, κοκ, αυτό θα τεθεί ως δέκτης (φυσικά, αυτό δεν είναι αμετάκλητο και το ίδιο στοιχείο μπορεί να τεθεί αργότερα ως πομπός).

Έπειτα, για να δημιουργήσουμε τα ζευγαρώματα αυτά καθ' αυτά, αρκεί να ζευγαρώσουμε τον πρώτο πομπό με τον πρώτο δέκτη, το δεύτερο πομπό με το δεύτερο δέκτη, κοκ.

## Αλγόριθμος

**ι.** Έστω A[1...n] ένας πίνακας. Κάθε θέση του πίνακα αντιστοιχεί σε μια κεραία με τη σειρά. Θέτουμε ένα δείκτη a στο πρώτο στοιχείο του πίνακα A.

**ιι.** Ορίζουμε τον πίνακα K[1...n], όπου για την κεραία i: $K_i = T_i - R_i \geq 0$.

**ιιι.** Ορίζουμε ένα σωρό ελαχίστου H (αρχικά άδειο) στον οποίο θα αποθηκεύουμε κεραίες με βάση το κόστος τους $K_i$.

**iv.** Έστω i το στοιχείο που δείχνει ο a.
Εάν το i μπορεί να τεθεί ως δέκτης (δηλαδή αν το i είναι άρτιο), το θέτουμε δέκτη και προσθέτουμε το i στο σωρό Η σύμφωνα με την τιμή $K_i$ .
Διαφορετικά, συγκρίνουμε την τιμή $K_i$ με τη ρίζα του σωρού, έστω το στοιχείο j με τιμή $K_j$.
Αν $K_i \leq K_j$, θέτουμε το i ως πομπό.
Διαφορετικά, θέτουμε το j ως πομπό, το i ως δέκτη και εισάγουμε το i στο σωρό.

**v.** Μετακινούμε το δείκτη a μία θέση δεξιά. Αν ξεπεράσαμε τα όρια του πίνακα, τερματίζουμε. Διαφορετικά, επαναλαμβάνουμε τη διαδικασία από το βήμα (iv).

**vi.** Σε αυτό το σημείο έχουμε ορίσει έγκυρα ποιες κεραίες είναι πομποί και ποιες δέκτες.
Θέτουμε δύο δείκτες s, r στο πρώτο στοιχείο του πίνακα A. Σημειώνουμε πως το στοιχείο αυτό είναι σίγουρα πομπός.

**vii.** Λαμβάνουμε τον πομπό που δείχνει ο δείκτης s. Μετακινούμε το r προς τα δεξιά μέχρι να βρούμε τον πρώτο δέκτη. Όταν συναντήσουμε δέκτη, ζευγαρώνουμε το πομπό που δείχνει το s με το δέκτη που δείχνει το r.

**viii.** Μετακινούμε τον r μία θέση δεξιά. Μετακινούμε τον s προς τα δεξιά μέχρι να συναντήσουμε τον επόμενο πομπό.
Αν ξεπεράσαμε τα όρια του πίνακα, τερματίζουμε.
Διαφορετικά, επαναλαμβάνουμε τη διαδικασία από το βήμα (vii).

**<u>Ορθότητα</u>**

Καταρχάς, αναφέρουμε πως ο αλγόριθμός μας θα δώσει μια έγκυρη λύση στο πρόβλημα, καθώς σε κάθε βήμα, αν μία κεραία τεθεί ως δέκτης, αυτό σημαίνει πως υπάρχει τουλάχιστον ένας παραπάνω πομπός στα αριστερά της.
Αυτό ισχύει για όλους τους δέκτες.

Λοιπόν, με τα βήματα (vii) και (viii) του αλγορίθμου μας, δημιουργούνται έγκυρα ζεύγη.
Όταν θα ζευγαρωθεί το πρώτο ζεύγος πομπού - δέκτη, μπορούμε να θεωρήσουμε πως τα βγάζουμε από το σύνολό μας.
Τότε, ο επόμενος δέκτης θα έχει και πάλι τουλάχιστον έναν παραπάνω πομπό από ότι δέκτη στα αριστερά του, καθώς διαγράψαμε έναν πομπό και έναν δέκτη στα αριστερά του, δηλαδή ίσο πλήθος.
Επαναλαμβάνοντας τη διαδικασία, δείχνουμε επαγωγικά πως όλα τα ζευγαρώματα είναι έγκυρα.

Θα αποδείξω πως η λύση μου είναι βέλτιστη.

Καταρχάς γενικεύω (λίγο) το πρόβλημα και ορίζω ως ζητούμενο για άρτιο πλήθος κεραιών ακριβώς αυτό που ορίζει η εκφώνηση και για περιττό πλήθος κεραιών το ίδιο με τη διαφορά πως επιτρέπεται να έχουμε έναν πομπό αζευγάρωτο. Θα δείξω πως ο αλγόριθμός μου βρίσκει βέλτιστη λύση σε αυτό το πρόβλημα (το οποίο περιλαμβάνει το ζητούμενο).

Θα το αποδείξω με επαγωγή.

Επαγωγική Βάση

Για n=1, ο αλγόριθμός μας βρίσκει βέλτιστη λύση τετριμμένα.

Επαγωγική υπόθεση

Έστω ότι ο αλγόριθμός μας βρίσκει βέλτιστη λύση για κάθε σύνολο κεραιών πλήθους n.

Επαγωγικό βήμα

Έστω έχουμε n+1 κεραίες.

Αγνοούμε προς στιγμήν την (n+1)-οστή κεραία και εφαρμόζουμε τον αλγόριθμο στις πρώτες n κεραίες. Από Ε.Υ. παίρνουμε λύση για αυτό το σύνολο. Και τώρα ας εισάγουμε τη (n+1)-οστή κεραία.

Διακρίνω περιπτώσεις:

α. (n+1) άρτιο.

Σύμφωνα με τον αλγόριθμό μας, η (n+1)-οστή κεραία τίθεται ως δέκτης χωρίς αλλαγές στις υπόλοιπες κεραίες.

Σε ένα οποιοδήποτε πλήρες σύνολο κεραιών άρτιου πλήθους η τελευταία πρέπει προφανώς να είναι οπωσδήποτε δέκτης.

Δηλαδή, πρόκειται για μία συνθήκη που ισχύει πάντοτε και, συνεπώς, δεν επηρεάζει τη βελτιστοποίηση του κόστους.

Ως εκ τούτου, προκύπτει πως το κόστος θα είναι ελάχιστο αν το κόστος των υπολοίπων κεραιών είναι το ελάχιστο δυνατό.

Ισχύει από Ε.Υ.

β. (n+1) περιττό.

Στο σύνολο [1...n] έχουμε ίσους πομπούς και δέκτες.

Συνεπώς, στο [1...n+1] πρέπει να έχουμε έναν παραπάνω πομπό.

Ο αλγόριθμός μας ορίζει ως πομπό την κεραία με το ελάχιστο κόστος ανάμεσα σε όλους τους δέκτες που έχουμε και στην εισαγόμενη (n+1)-οστή κεραία.

Εφόσον, από Ε.Υ. η λύση μας ήταν βέλτιστη στο σύνολο [1...n] και αυτή η λύση θα είναι βέλτιστη στο σύνολο [1...n+1], καθώς είναι $K_{n+1} \geq K_n$ και αυτή είναι η αλλαγή που αυξάνει όσο το δυνατόν λιγότερο το κόστος.

Περισσότερη ανάλυση για την περίπτωση β. (n+1) περιττό

Ας το αναλύσουμε λίγο περισσότερο αυτό το σημείο.

Θυμίζουμε πως ένας δέκτης θεωρείται πως έχει μηδενικό κόστος ενώ ένας πομπός i έχει κόστος $K_i$.

Έστω μια βέλτιστη λύση OPT σε αυτό το σημείο και η λύση μας SOL.

Αν τόσο η OPT όσο και η SOL, απαιτούν η (n+1) να τεθεί ως πομπός / δέκτης (δηλαδή το ίδιο και στις δύο λύσεις), η SOL είναι βέλτιστη, καθώς από Ε.Υ., το K[1...n] είναι βέλτιστο.

Και $K_{SOL} = K^*[1...n] + K_{n+1}$ αν (n+1) πομπός ή

$K_{SOL} = K^*[1...n] + K_j$ , αν (n+1) δέκτης με j το στοιχείο με το μικρότερο δυνατό K.

Έστω τώρα πως οι δύο λύσεις διαφέρουν ως προς το (n+1)
Διακρίνω περιπτώσεις:

a. Έστω πως ο αλγόριθμός μας απαιτεί να θέσουμε κάποια κεραία i < n+1 ως πομπό και τη (n+1)-οστή ως δέκτη, ενώ η "βέλτιστη" λύση είναι να θέσουμε τη (n+1)-οστή ως πομπό (και ίσως να αναδιαμορφώσουμε τις υπόλοιπες κεραίες).
Εφόσον, ο αλγόριθμός μας επέλεξε την κεραία έστω j να γίνει πομπός αντί για τη (n+1), συνεπάγεται πως $K_j \leq K_{n+1}$ .
Όμως, τότε το κόστος σύμφωνα με το αλγόριθμό μας είναι K*[1...n] + $K_j$ όπου j η κεραία που τέθηκε ως πομπός.
Ενώ η "βέλτιστη" έχει κόστος K'[1...n] + $K_{n+1}$ με $K_j \leq K_{n+1}$ και K*[1...n] ≤ K'[1...n].
Συνεπώς, η λύση μας είναι τουλάχιστον ισοδύναμη με τη βέλτιστη.

b. Έστω αντ' αυτού πως ο αλγόριθμός μας απαιτεί να θέσουμε τη (n+1) ως πομπό χωρίς να πειράξουμε τις υπόλοιπες, ενώ η βέλτιστη έχει τη (n+1) ως δέκτη (και άλλη διαμόρφωση στις πρώτες n).
Τότε, η λύση μας έχει κόστος K*[1...n] + $K_{n+1}$
Ενώ η βέλτιστη έχει κόστος ≥ K'[1...n] + $K_j$ όπου j μια κεραία που τέθηκε ως πομπός ενώ στη SOL είναι δέκτης.
Όμως, εφόσον ο αλγόριθμός μας επέλεξε τη (n+1) να γίνει πομπός, σημαίνει πως $K_{n+1} \leq K_j$.
Επίσης, προφανώς K*[1...n] ≤ K'[1...n] .
Συνεπώς, η λύση μας είναι τουλάχιστον ισοδύναμη με τη βέλτιστη.

**Πολυπλοκότητα**

Με κάθε κίνηση του δείκτη a, ένα στοιχείο έχουμε το πολύ μία εισαγωγή στο σωρό και το πολύ μία διαγραφή στο σωρό. Κάθε εισαγωγή / διαγραφή θέλει χρόνο O(logn) καθώς ο σωρός δε θα έχει ποτέ περισσότερα από n/2 στοιχεία μέσα του.
Ο δείκτης a κάνει n τέτοιες κινήσεις μέχρι να φτάσει τα όρια του πίνακα.

Έπειτα, με κάθε κίνηση των s, r έχουμε O(1) διεργασίες. Γίνονται n τέτοιες μετακινήσεις για τον κάθε δείκτη.

Επομένως, έχουμε συνολική πολυπλοκότητα **O(nlogn)**.