Αλγόριθμοι

2η Σειρά Ασκήσεων

Ονοματεπώνυμο:	Κεχριώτη Ελένη
Αριθμός Μητρώου:	3210078
Email:	p3210078@aueb.gr

2023

Ασκήσεις

Άσκηση 1

Έχουμε μία λίστα με n πιθανούς καλεσμένους και θέλουμε να καλέσουμε όσους περισσότερους μπορούμε. Ωστόσο κάθε άτομο από τη λίστα θα πρέπει να γνωρίζει και να μην γνωρίζει τουλάχιστον 4 άτομα, για να μπορούμε να τον καλέσουμε. Επειδή, μπορεί ένα άτομο να μην πληρεί τις προϋποθέσεις και άρα να μην μπορεί να προσκεκληθεί, αλλά είναι στη λίστα των γνωστών άλλων ατόμων, θα πρέπει να ενημερώνεται η λίστα των γνωστών και άγνωστων κάθε ατόμου με κάθε αποκλεισμό.

Η ιδέα για τον αλγόριθμο είναι:

- Αντιγράφουμε το πίνακα γνωριμιών Α, έστω Γ.
- Όσο ο πίνακας Γ δεν είναι κενός, δηλαδή υπάρχουν ακόμα πιθανά άτομα για πρόσκληση, μετράμε πόσα από τα πιθανά άτομα (αυτά που παραμένουν στον Γ) γνωρίζουν και πόσα δεν γνωρίζουν. Αν πληρούν τις προϋποθέσεις, τότε συνεχίζουμε στον επόμενο. Διαφορετικά, αφαιρούμε το άτομο από το πίνακα Γ
- Όσο αφαιρούμε, έστω και ένα άτομο από το Γ τότε συνεχίζουμε την επανάληψη. Διαφορετικά, αν δηλαδή σε μια επανάληψη δεν έχουμε αφαιρέσει κάποιο άτομο τότε η επανάληψη θα τερματίσει, καθώς όλα τα άτομα στο Γ πληρούν και τις δύο προϋποθέσεις και η τελική λίστα των προσκεκλημένων είναι έτοιμη.
- Επιστρέφουμε τον αριθμό των προσκεκλημένων (μήκος του Γ)

ΟΡΘΟΤΗΤΑ

Ο πίνακας Α είναι πεπερασμένος και έτσι γνωρίζουμε ότι μετά απο τουλάχιστον N επαναλήψεις θα τερματιστεί.

Έστω ότι έχουμε μια βέλτιστη λύση Δ. Προφανώς το Δ είναι ένα υποσύνολο του Α, δηλαδή ισχύει ότι $\Delta \sqsubseteq A$ και μπορεί να κατασκευαστεί εύκολα, αφαιρώντας απλά στοιχεία απο το Α.

Θα το αποδείξουμε μέσω επαγωγής.

Έστω m τα άτομα που αφαιρέθηκαν από το Α, κατά την εκτέλεση του αλγορίθμου για n άτομα. Προφανώς $|\Delta| = n - m$.

Αν $m = 0$, τότε έχουμε ότι $|\Gamma| = n$ και άρα $|\Delta| = n$ γιατί το Δ είναι η βέλτιστη λύση Αυτό σημαίνει ότι δεν αφαιρέσαμε κανένα άτομο από τη λίστα καθώς όλοι πληρούσαν τις προυποθέσεις.

Αν $m > 0$, τότε για κάθε άτομο i (i< m) που αφαιρούμε από το Γ θα πρέπει να αφαιρείται και από το Δ, αφού το Δ είναι η βέλτιστη λύση.

Έστω ότι τρέχουμε τον αλγόριθμο με Α τέτοιο ώστε m άτομα να μην πληρούν τις προϋποθέσεις και έστω ότι ο αλγόριθμος τρέχει το ίδιο για το Γ και για το Δ μέχρι την αφαίρεση k-1 ατόμου.

Τώρα έχουμε δύο περιπτώσεις:

α) Είτε ο αλγόριθμος θα τερματίσει εδώ, δηλαδή το k-1 άτομο θα είναι το τελευταίο που θα αφαιρεθεί από τη λίστα

β) Είτε ο αλγόριθμος θα συνεχίσει και θα αφαιρεθεί και άλλο άτομο.

Για την πρώτη περίπτωση είναι προφανές ότι αν ο αλγόριθμος τερματίσει μετά την αφαίρεση του k-1 ατόμου για το Γ, τότε θα πρέπει να τερματίσει και για το Δ, αφού είναι η βέλτιστη λύση.

Για την δεύτερη περίπτωση, γνωρίζουμε ότι θα αφαιρεθεί και άλλο άτομο, το k-οστό. Αυτό σημαίνει ότι το k-οστό άτομο δεν πληρούσε την συνθήκη 1 ή 2.

Έστω ότι δεν πληρούσε την συνθήκη 1, δηλαδή γνωρίζε κάτω απο 4 άτομα στη λίστα Γ. Αυτό συνέβει γιατί αφαιρέσαμε k-1 άτομα ήδη απο το Γ (διαφορετικά θα είχε αφαιρεθεί ήδη από την πρώτη επανάληψη). Αφού όμως δεν πληρεί την συνθήκη αυτή θα πρέπει να αφαιρεθεί απο το Γ. Άρα θα αφαιρεθεί και από το Δ, δεδομένου ότι $\Delta = \Gamma$ μέχρι την k-1 ιοστή αφαίρεση, αφού είναι η βέλτιστη λύση. Αντίστοιχα και αν το k-οστό δεν πληρεί την συνθήκη 2 ή και τις δύο.

Άρα, συμπεραίνουμε ότι το Γ που προκύπτει από τον αλγόριθμο είναι η βέλτιστη λύση.

ΠΟΛΥΠΛΟΚΟΤΗΤΑ

Για τον έλεγχο αν το i άτομο γνωρίζει 4 τουλάχιστον άτομα και δεν γνωρίζει τουλάχιστον άλλα 4 άτομα θέλουμε n^2 πράξεις.

Για την επανάληψη όσο υπάρχουν άτομα στη λίστα των πιθανών καλεσμένων θέλουμε στη χειρότερη περίπτωση n επαναλήψεις. (Έστω ότι οι γνωριμίες είναι τέτοιες έτσι ώστε να αφαιρείται ένα άτομο σε κάθε επανάληψη)

Άρα έχουμε $O(n^3)$ πολυπλοκότητα.

Άσκηση 2

Έστω C ένας δισδιάστατος πίνακας (n+1) x (E+1).

Ο πίνακας C θα περιέχει ακέραιους αριθμούς και το ∞. Δηλαδή το C[i][j] θα αντιπροσωπεύει το ελάχιστο πλήθος των νομισμάτων που μπορούμε να χρησιμοποιήσουμε για να εκφράσουμε τον αριθμό E. Αν στη θέση C[i][j] βρίσκεται το ∞ τότε αυτό σημαίνει ότι δεν μπορούμε να εκφράσουμε την αξία j με τα i νομίσματα.

Διαφορετικά, θα υπάρχει ένας ακέραιος αριθμός και θα συμβολίζει το πλήθος των ελάχιστων νομισμάτων που μπορούν να χρησιμοποιηθούν.

Αρχικά, αρχικοποιούμε τον πίνακα C με ∞. Για την πρώτη στήλη του πίνακα C, θέτουμε την τιμή 0, γιατί προφανώς για 0 αξία θα χρησιμοποιήσουμε 0 νομίσματα.

Για τις υπόλοιπες θέσεις του πίνακα C, θέλουμε να αποθηκεύσουμε τον αριθμό των ελάχιστων νομισμάτων που μπορούν να χρησιμοποιηθούν.

Αν μπορώ να σχηματίσω το E, θα μπορώ να σχηματίσω τουλάχιστον ένα από τα ποσά

$E - v_1$, $E - v_2, \dots, E - v_n$.

Έτσι για κάθε κελί θα πρέπει να επιλέξουμε αν μας συμφέρει να χρησιμοποιήσουμε τα i-1 νομίσματα για να εκφράσουμε την αξία j ή αν μας συμφέρει να χρησιμοποιήσουμε το νόμισμα i, προσθέτοντας το ίδιο το νόμισμα και χρησιμοποιώντας τα i-1 νομίσματα για την αξία j – v[i]. Έτσι καταλήγω στην αναδρομική σχέση:

$$C(i,j) = min\{C(i-1,j), C(i-1,j-v_i)+1\}$$

Προσθέτουμε 1 για να εκφράσουμε την χρησιμοποίηση του νομίσματος i, επειδή μιλάμε για πλήθος νομισμάτων.

Αν δεν μπορούμε να χρησιμοποιήσουμε το νόμισμα i τότε για να εκφράσουμε την μη χρησιμοποίηση του θέτουμε στο κελί C(i,j) την τιμή του ακριβώς απο πάνω κελιού, δηλαδή του C(i-1,j).

Τελικά καταλήγουμε στην εξής αναδρομική σχέση:

$$C(i,j) = \begin{cases} 0 & , i = 0,\ \ 0 \le j \le n \\ min\{C(i-1,j), C(i-1,j-v_i)+1\} & ,\ \ i,j \ge 1 \ και\ v_i \ \le j \\ C(i-1,j) & , i,j \ge 1 \ και\ v_i \ > j \end{cases}$$

Στο τελευταίο κελί του πίνακα C, δηλαδή στο C(n,E) θα βρίσκεται η τελική απάντηση. Αν στο κελί αυτό βρίσκεται το ∞ τότε είναι αδύνατο να σχηματίσουμε το E με τα νομίσματα που δίνονται, διαφορετικά θα υπάρχει ο αριθμός των ελάχιστων νομισμάτων που μπορούμε να χρησιμοποιήσουμε.

Για την ανάκτηση των νομισμάτων που χρησιμοποιήσαμε τα βήματα είναι τα εξής:

- Ξεκινώντας από την θέση C(n,E) αν η τιμή της είναι ίδια με το C(n-1, E), τότε αυτό σημαίνει ότι δεν χρησιμοποιήσαμε το νόμισμα n-1 και προχωράμε σε αυτή τη θέση.
- Διαφορετικά, αυτό θα σημαίνει ότι έχουμε χρησιμοποιήσει το νόμισμα και αποθηκεύουμε την αξία του στη λίστα με τα νομίσματα που χρησιμοποιούμε.
- Επαναλαμβάνουμε τα βήματα μέχρι να φτάσουμε στην πρώτη γραμμή ή στήλη του πίνακα C.

Ψευδοκώδικας:

Συνάρτηση coins(v,E):

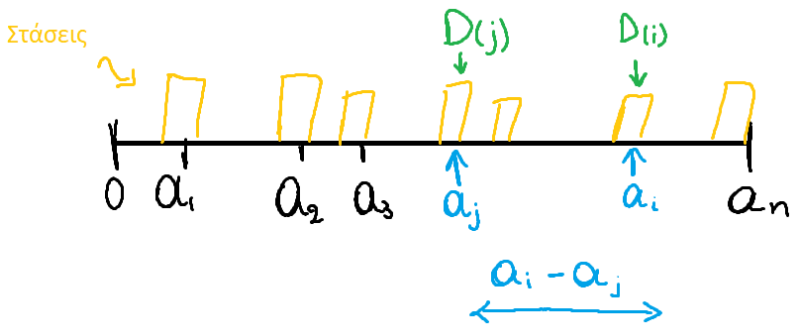
```
    n = len(v)
    c = [[float('inf')]*(E+1) for i in range(n+1)]
    for i = 0 to n+1:        c[i][0] = 0
    for i = 1 to n+1:
        for j = 0 to E+1:
            if v[i-1] <= j:
                c[i][j] = min(c[i-1][j], c[i-1][j-v[i-1]] + 1)
             else:
                c[i][j] = c[i-1][j]
     if c[n][E] == float('inf'):
        return "Αδύνατο"
    coins_used = []   ,   i = 1 , j = E
    while i > 0 and j > 0:
        if dp[i][j] == dp[i-1][j]:
            i -= 1
        else:
            selected_coins.append(coins[i-1])
            j -= coins[i-1]    ,    i -= 1
    return coins_used
```

Πολυπλοκότητα:

- Αρχικοποίηση του πίνακα C μεγέθους (n+1)x(E+1), O(n*E)
- Αρχικοποίηση της πρώτης στήλης του πίνακα C μεγέθους n+1, O(n)
- Δυο επαναλήψεις for, που τρέχουν n και E+1 φορές, O(n*E)
- Η ανάκτηση των νομισμάτων , O(n)

Άρα, ο αλγόριθμος έχει πολυπλοκότητα $O(n * E) + O(n) + O(n * E) + O(n) = O(n * E)$.

Άσκηση 3



Χ χλμ -> $(40 - x)^2$ κόστος.

Έστω ότι βρίσκομαι στη θέση a_j και θέλω να κάνω στάση στη θέση a_i, τότε η ποινή για τη στάση αυτή θα είναι $\left(40 - (a_i - a_j)\right)^2$. Για να κάνω στάση, όμως στη θέση a_i πρέπει να ξέρω αν με συμφέρει να σταθμεύσω εκεί. Επομένως, πρέπει να βρω την ελάχιστη ποινή που θα πάρω για να σταθμεύσω από την a_j στην a_i για j<i . Ωστόσο πρέπει να λάβω υπόψη και τη ποινή από την προηγούμενη στάση a_j. Έστω ο πίνακας D ο οποίος περιέχει για κάθε στάση την ελάχιστη ποινή. Για την εύρεση της ελάχιτης ποινής για την θέση a_i, όπως περιγράφηκε παραπάνω έχω

$$D(i) = \underbrace{min}_{0 \le j \le i} \left\{ D(j) + \left(40 - (a_i - a_j)\right)^2 \right\}$$

Αλγόριθμος:

- Έχω έναν πίνακα D, για τις ελάχιστες ποινές για κάθε θέση i, με την θέση 0 προφανώς να έχει ελάχιστη ποινή 0. Δηλαδή $D(0) = 0$.
- Για κάθε στάση i θα υπολογίσω την ποινή που θα έχω αν έλθω σε αυτή από κάθε στάση j, όπου $j \le i$ και θα κρατήσω την ελάχιστη ποινή από αυτές, μαζί και με τον αριθμό της στάσης που είχε την ελάχιστη ποινή.
- Τέλος, επιστρέφω την λίστα με τις στάσεις που αποθήκευσα, που έχουν την μικρότερη ποινή συνολικά.

Ψευδοκώδικας:

Συνάρτηση findStops(stop , n):

```
D[0] = 0
best_stops = []
For i = 1 to n:
    mindist = float('inf')
    minstop = -1
```

```
        For j = 0 to i:
            d = D[j] + (40 − (stop[i] − stop[j]))^2
            if d < mindist then:
                mindist = d
                minstop = j
        D[i] = mindist
        if minstop is not in best_stops:
            best_stops.append(minstop)
    return best_stops
```

Ορθότητα:

Έστω ότι έχουμε μια βέλτιστη λύση Δ. Η Δ είναι προφανώς υποσύνολο του stops, δηλαδή ισχύει ότι $\Delta \sqsubseteq stops$, εφόσον στο Δ προσθέτονται οι στάσεις που μας δίνουν την μικρότερη ποινή και ο μέγιστος αριθμός στάσεων που μπορεί να είναι στο Δ είναι n.

Θα το αποδείξω με απαγωγή σε άτοπο.

Έστω ότι ο αλγόριθμος δεν δίνει την βέλτιστη λύση. Άρα υπάρχει μια λύση καλύτερη, με στάσεις που δίνουν μικρότερη ποινή και ο αλγόριθμος μας δεν δίνει.

Όμως, ο αλγόριθμος για κάθε στάση από i = 1..n, βρίσκει και υπολογίζει κάθε ποινή, για κάθε δυνατό συνδυασμό στάσεων από την αρχή μέχρι και την στάση i και μετά από κάθε υπολογισμό ελέγχει αν η ποινή που υπολογίστηκε τώρα είναι μικρότερη από την ποινή που υπολογίστηκε για προηγούμενο ζεύγος. Ο αλγόριθμος, δηλαδή επιλέγει την στάση με τη μικρότερη ποινή μέχρι την κάθε στιγμή. Αυτό σημαίνει ότι ο αλγόριθμος προσθέτει στη λίστα best_stops την στάση που είναι η βέλτιστη για εκείνη τη στιγμή. Έτσι, ο αλγόριθμος δίνει τη βέλτιστη λύση μέχρι και την τελευταία στάση. Επομένως, δεν υπάρχει άλλη λύση με μικρότερη ποινή.

Άτοπο

Άρα ο αλγόριθμος δίνει τη βέλτιστη λύση για το πρόβλημα.

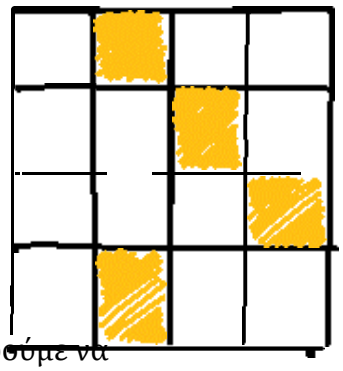
Πολυπλοκότητα:

- Η εξωτερική επανάληψη (for) τρέχει ακριβώς n-1 φορές.
- Η εσωτερική επανάληψη τρέχει κάθε φορά i φορές, την τελευταία επανάληψη n φορές
- Ο έλεγχος αν η στάση έχει ήδη βρεθεί ως βέλτιστη και έχει προστεθεί στη λίστα best_stops, στη χειρότερη περίπτωση θα τρέξει n φορές

Άρα $(n-1) \times (n+n) = 2n^2 - 2n$.

Επομένως, ο αλγόριθμος έχει $O(n^2)$ πολυπλοκότητα.

Άσκηση 4



α) Για την επίλυση αυτού του προβλήματος,

δηλαδή την εύρεση των κελιών που έχουν την

μέγιστη συνολική αξία, θα χρειαστούμε ένα πίνακα

για να αποθηκεύσουμε τις συνολικές αξίες μέχρι στιγμής.

Έστω P αυτός ο πίνακας και έστω V ο πίνακας με τις αξίες.

Το P[i][j] , δηλαδή αντιπροσωπεύει τη μέγιστη συνολική αξία που μπορούμε να συγκεντρώσουμε αν επιλέξουμε κάποιο κελί στη σειρά i και τη στήλη j, με τον περιορισμό ότι δεν μπορούμε να επιλέξουμε κελιά που ανήκουν στην ίδια στήλη με τα ακριβώς προηγούμενα επιλεγμένα κελιά.

Για την εύρεση κελιών με μέγιστη συνολική αξία:

Αναγκαστικά θα επιλέξω ένα κελί από κάθε γραμμή, αλλά δεν είναι απαραίτητο να επιλέξω ένα κελί από κάθε στήλη. Μια έγκυρη επιλογή κελιών φαίνεται στο παραπάνω πίνακα.

Θα επιλέξω ένα κελί από την πρώτη γραμμή του V, χωρίς να ξέρω εξαρχής ποιο θα μου δώσει τη μέγιστη συνολική αξία. Στον πίνακα P, περνάω σε κάθε κελί της πρώτης γραμμής τις αξίες της πρώτης γραμμής από τον πίνακα V, γιατί το μέγιστο άθροισμα, εφόσον δεν υπάρχει προηγούμενη γραμμή, αν επιλέξω το κάθε κελί, θα ισούται με το ίδιο το κελί.

Για τις υπόλοιπες γραμμές:

Για κάθε κελί, έστω ότι το υπολογίζω στο άθροισμα. Το συνολικό άθροισμα θα ισούται με αυτό το κελί συν τη μέγιστη συνολική αξία των προηγούμενων κελιών, δηλαδή το μαξ της προηγούμενης γραμμής, εφόσον έχουμε πει ότι κάθε κελί συμβολίζει την μέγιστη συνολίκη αξία που έχουμε συγκεντρώσει φτάνοντας σε αυτό το κελί. Με αυτή τη επεξήγηση, καταλαβαίνουμε ότι δεν γίνεται να επιλέξουμε την μεγιστη συνολική αξία του ακριβώς από πάνω κελιού, δίοτι έχουμε τον περιορισμό να μην επιλέξουμε γειτονικά κελιά. Για παράδειγμα, αν βρίσκομαι στο κελί P[i][j] και η μέγιστη αξία βρίσκεται στο κελί P[i-1][j] τότε δεν μπορώ να το επιλέξω.

Με αυτή τη ανάλυση, καταλήγουμε στην παρακάτω αναδρομική σχέση:

$$P(i,j) = V(i,j) + \max_{\substack{0 \le k \le n \\ k \ne j}} \{P(i-1,k)\}$$

Αλγόριθμος:

- Ορίζω ένα πίνακα P με διαστάσεις n × n και αρχικοποιώ την πρώτη γραμμή.
  - P(0,j)= V(0,j)
- Για κάθε γραμμή του πίνακα V από i = 1…n και για κάθε στήλη j από j = 0…n υπολογίζουμε το
  - $P(i,j) = V(i,j) + \max_{\substack{0 \le k \le n \\ k \ne j}} \{P(i-1,k)\}$
- Για να βρούμε τελικά ποια είναι η μέγιστη συνολική αξία, το μόνο που χρειάζεται είναι να βρούμε την μέγιστη τιμή της τελευταίας γραμμής του πίνακα P.

Ψευδοκώδικας:

```
Συνάρτηση  max_values(V, n):
p = ['']* n
for i = 0 to n:
        p[i] = ['']*len(V)
        p[0][i] = V[0][i]

for i = 1 to n:
    for j = 0 to n:
        P[i][j] = V[i][j] + max( P[i-1][k] ) , k!= j
        p[i][j] = V[i][j] + max

# Retrieve cells
    for I = 0 to n:
        if value< p[n-1][i] :
            value = p[n-1][i]       # find the position of max total value
            maxj = i
```

```
pair.insert(0,[n-1,j])                    #last cell to make max total value


value -= V[n-1][ maxj]
for i  = n-1 down to -1:
        for j = 0 to n:
            if p[i][j] == value:
                pair.insert(0, [i,j])# cell that led to max total value
                value -= V[i][j]
return pair, val
```

Ο αλγόριθμος αυτός (θα δικαιολογηθεί παρακάτω) έχει πολυπλοκότητα n^3. Ωστόσο μπορούμε θυσιάζοντας χώρο να μειώσουμε την πολυπλοκότητα του σε n^2.

Αυτό μπορεί να γίνει απλούστα, κρατώντας τα 2 πρώτα max κάθε γραμμής μαζί με τους αριθμούς των στηλών τους.

Σε έναν πίνακα π.χ maxes αποθηκεύουμε για κάθε γραμμή δύο πίνακες δύο θέσεων, όπου στη πρώτη θέση θα βρίσκεται η μέγιστη αξία και στη δεύτερη ο αριθμός της στήλης.

Σε ψευδοκώδικα η ιδέα αυτή θα είναι:

```
maxes = ['']*n
    max1 , j1 = max( V[0])
    max2, j2 = max(V[0]) για j2 ≠ j1
    maxes[i] = ([[max1, j1],[max2, j2]])
  for i = 1 το n:
    for j = 0 to n:
      if  j == maxes[i-1][0][1]:
        p[i][j] = V[i][j] + p[i-1][maxes[i-1][1][1]]
      else:
        p[i][j] = V[i][j] + p[i-1][maxes[i-1][0][1]]
      max1 , j1 = max( p[i])
      max2, j2 = max(p[i]) για j2 ≠ j1
```

maxes[i] = ([[max1, j1],[max2, j2]])

Επειδή είναι προφανές πως βρίσκουμε από ένα από κάθε γραμμή τις δύο μέγιστες αξίες δεν τις έγραψα, για μείωση χώρου και για μεγαλύτερη κατανόηση της ιδέας. Ουσιαστικά, βρίσκουμε στην αρχή τις δύο μέγιστες αξίες της πρώτης γραμμής του πίνακα V. Ύστερα, ακολουθούμε την ίδια διαδικασία με πριν, υπολογίζοντας για κάθε κελί την μέγιστη αξία που θα μπορούσε να έχει. Αν βρισκόμαστε στο κελί (i,j) και το μέγιστο στοιχείο της προηγούμενης γραμμής βρίσκεται στη στήλη j τότε θα επιλέξουμε το δεύτερο μέγιστο στοιχείο. Διαφορετικά, παίρνω το πρώτο μέγιστο στοιχείο.

Για μια γραμμή έχω δυο μέγιστες τιμές. Είναι προφανές ότι για την επόμενη γραμμή για όλα τα κελιά, πέρα από το κελί που βρίσκεται στη ίδια στήλη με την μέγιστη αξία της προηγούμενης γραμμής, η μέγιστη αξία θα είναι η πρώτη που βρήκαμε. Για το ένα, όμως κελί που βρίσκεται στην ίδια στήλη με την μέγιστη αξία της προηγούμενης γραμμής θα χρειαστεί να πάρω το δεύτερο μέγιστο στοιχείο. Για αυτό το λόγο χρειάζομαι μόνο τα δυο πρώτα max κάθε γραμμής.

Μόλις υπολογίσουμε τη συνολική μέγιστη αξία του κάθε κελιού (p[i][j]) ελέγχουμε αν έχει την μέγιστη αξία. Για την γραμμή, δηλαδή που υπολογίζουμε, τρέχουμε ταυτόχρονα και τον αλγόριθμο για την εύρεση του max, γιατί θα το χρειαστούμε στην επόμενη επανάληψη.

β) Για τον αλγόριθμο με την άπληστη τεχνική:

Βρίσκουμε το max από όλο το πίνακα, δηλαδή προσπελάζουμε όλο τον πίνακα, ο οποίος έχει n^2 στοιχεία.

Θέλουμε να συγκεντρώσουμε n κελιά από όλο τον πίνακα. Άρα, θα βρούμε n φορές το μέγιστο στοιχείο από όλο το πίνακα που να πληρεί τις προϋποθέσεις.

Η πρώτη επανάληψη, για κάθε κελί που θέλουμε να μαζέψουμε, τρέχει n φορές.

Η δεύτερη εμφωλευμένη επανάληψη, για την εύρεση του max από όλο το πίνακα, τρέχει n^2 φορές.

Ο έλεγχος για κάθε στήλη αν έχει αξία μεγαλύτερη από την προηγούμενη καθώς και ο έλεγχος αν η προηγούμενη max τιμή βρισκόταν στο ακριβώς απο πάνω κελί (αποθηκέυουμε απλά στη προηγούμενη επανάληψη τη στήλη που βρέθηκε το max) γίνεται σε σταθερό χρόνο.

Άρα, έχουμε $n \times n^2 = n^3$. Άρα, έχουμε $O(n^3)$ πολυπλοκότητα.

Για τον αλγόριθμο με την δυναμική τεχνική:

- Θέλουμε n^2 για την αρχικοποίηση και ανάθεση τιμών στον P.
- Έχουμε 3 εμφωλευμένες επαναλήψεις for, και κάθεμία τρέχει n φορές, άρα συνολικά n^3.
- Θέλουμε n για να βρούμε τελικά την μέγιστη συνολική αξία και n^2 για να εντοπίσουμε τα κελιά που έχουν την μέγιστη συνολική αξία

Άρα συνολικά $n^2 + n^3 + n^2 = O(n^3)$ *πολυπλοκότητα*.

Για την δεύτερη προσέγγιση:

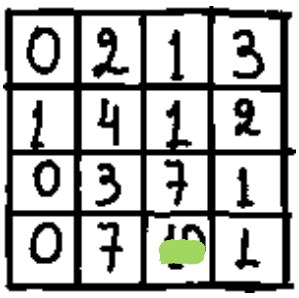
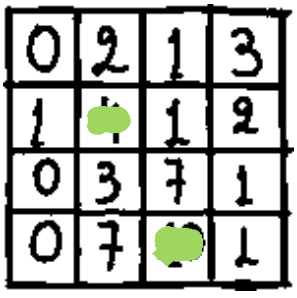
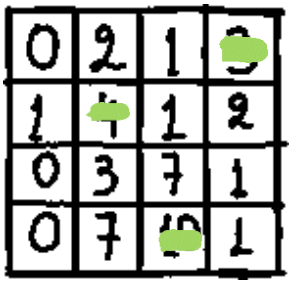
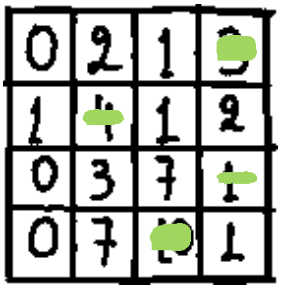
- Θέλουμε n^2 για την αρχικοποίηση και ανάθεση τιμών στον P.
- Για να βρούμε τα δύο πρώτα max της πρώτης γραμμής, θέλουμε n
- Έχουμε 2 εμφωλευμένες επαναλήψεις, για τον υπολογισμό των p και των max, άρα n^2
- Θέλουμε n για να βρούμε τελικά την μέγιστη συνολική αξία και n^2 για να εντοπίσουμε τα κελιά που έχουν την μέγιστη συνολική αξία

Άρα συνολικά $n^2 + n + n^2 + n^2 = O(n^2)$ *πολυπλοκότητα*.

γ) Ο αλγόριθμος με την άπληστη τεχνική δεν δίνει την βέλτιστη λύση. Θα το δείξω με αντιπαράδειγμα. Έστω ο πάρακάτω πίνακας V.

0	2	1	3
1	4	1	2
0	3	7	1
0	7	10	1

Στους παρακάτω πίνακες φαίνεται η επίλογη κελιών σε κάθε επανάληψη του αλγορίθμου greedy. Επιλέγουμε το μέγιστο στοιχείο από όλο τον πίνακα εφόσον πληρεί τις προϋποθέσεις.

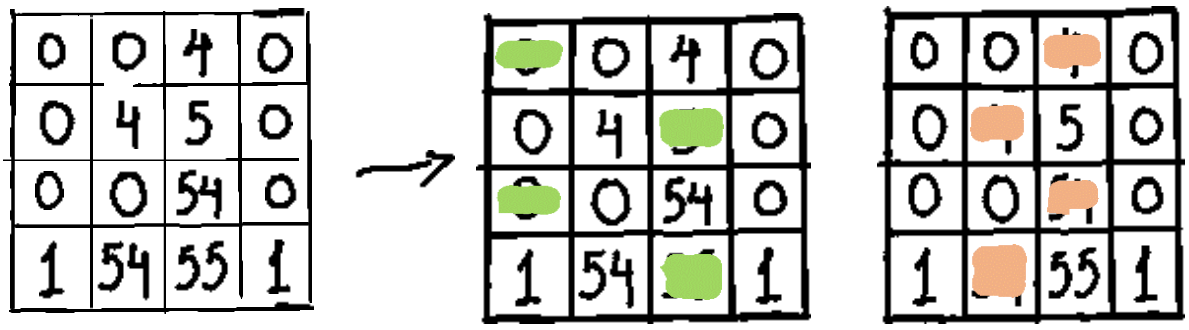





Βλέπουμε ότι επέλεξε τα κελιά (3,2) , (1,1) , (0,3) και (2,3) τα οποία έχουν αξίες 10, 4, 3, και 1 αντίστοιχα. Άρα επιστρέφει ως μέγιστη συνολική αξία $10 + 4 + 3 + 1 = 18$. Όμως, αυτή δεν είναι η μέγιστη συνολική αξία που μπορούμε να συγκεντρώσουμε. Αν επιλέξουμε τα κελιά (0,3) , (1,1) , (2,2) και (3,1) με αξίες 3, 4, 7 και 7 αντίστοιχα θα έχουμε συνολική αξία $3 + 4 + 7 + 7 = 21$. Προφανώς $21 > 18$.

Άρα ο αλγόριθμος με την άπληστη τεχνική δεν είναι βέλτιστος.

Ανάλογα με τον τρόπο υλοποίησης του αλγορίθμου έχουμε και διαφορετικό αποτέλεσμα, χωρίς αυτό να σημαίνει ότι είναι βέλτιστο. Για παράδειγμα αν υλοποιήσουμε τον αλγόριθμο να επιλέγει το πρώτο μέγιστο στοιχείο που βρήκε σε περίπτωση ισοβαθμίας ή αν επιλέγει το τελευταίο μέγιστο στοιχείο, θα επιλεχθούν διαφορετικά κελιά και μπορεί να έχουν και διαφορετική μέγιστη αξία σαν αποτέλεσμα.

Υποθέτω ότι σε περίπτωση ισοβαθμίας κρατάω το πρώτο μαξ που θα βρω. Τότε, ο μέγιστος λόγος που μπορώ να βρω μεταξύ της άπληστης και βέλτιστης λύσης δεν θα μπορεί να ξεπερνάει το 2. Τίθεται παράδειγμα



Αριστερά είναι ο αρχικός πίνακας, στη μέση με το πράσινο φαίνονται τα κελιά που επιλέγονται με την άπληστη τεχνική και έχει μέγιστη αξία 60 και δεξιά φαίνονται τα κελιά που μας δίνουν την βέλτιστη λύση και έχουν μέγιστη αξία 116.

Ο άπληστος αλγόριθμος βρίσκει την μέγιστη τιμή και την επιλέγει χωρίς να λάβει υπόψη το άθροισμα των γειτονικών κελιών του. Αν στη θέση ενός από τα 54 είχαμε τιμή ίση με την μέγιστη, δηλαδή 55 τότε ο αλγόριθμος θα επέλεγε πρώτα αυτό το κελί, γιατί αυτό προσπέλασε πρώτα και θα οδηγούσε σε μια καλύτερη λύση. Σε τέτοια παραδείγματα, όπου η μέγιστη τιμή που θα επιλέξει πρώτα ο greedy περικλείονται από κοντινές στην μέγιστη

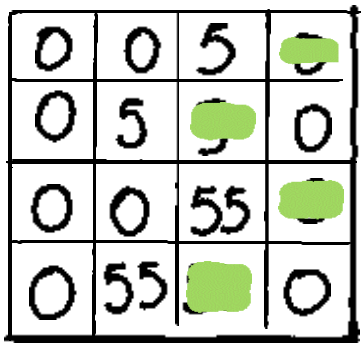
αξίες και αθροιστικά συγκεντρώνουν σχεδόν την διπλάσια αξία, τότε έχουμε το μέγιστο λόγο $max\frac{B(I)}{A(I)}$.

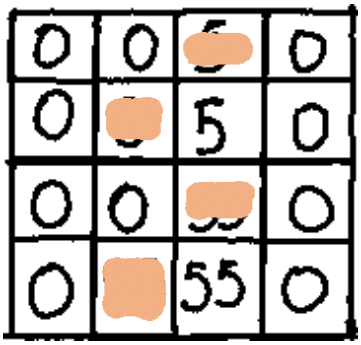
Στο παράδειγμα αυτό έχουμε $\frac{B(I)}{A(I)} = \frac{116}{60} \approx 1.93$

Σε παρόμοια λογική παραδείγματα ο λόγος θα διαφέρει αλλά θα είναι κόντα στο 2.

Και πράγματι $\frac{1}{\alpha} * 116 = \frac{1}{1.93} * 116 = 60{,}1 \approx 60$ την αξία δηλαδή που έδωσε ο greedy.

Ωστόσο αν ο αλγόριθμος ήταν υλοποιημένος έτσι ώστε σε περίπτωση ισοβαθμίας να επιλέγει την τελευταία μέγιστη τιμή, τότε ο μέγιστος λόγος που μπορούμε να πετύχουμε είναι 2. Τίθεται παράδειγμα





Δεξιά φαίνεται η άπλυστη λύση και αριστερά είναι η βέλτιστη λύση με μέγιστες συνολικές αξίες 60 και 120 αντίστοιχα.

Επειδή ο άπληστος αλγόριθμος κρατάει το τελευταίο μαξ δεν μας επηρεάζει αν θα βρει πρώτα την μέγιστη αξία γιατί θα κρατήσει την τελευταία, όπως φαίνεται στο κελί (3,2).

Στο παράδειγμα αυτό έχουμε $\frac{B(I)}{A(I)} = \frac{120}{60} = 2$.

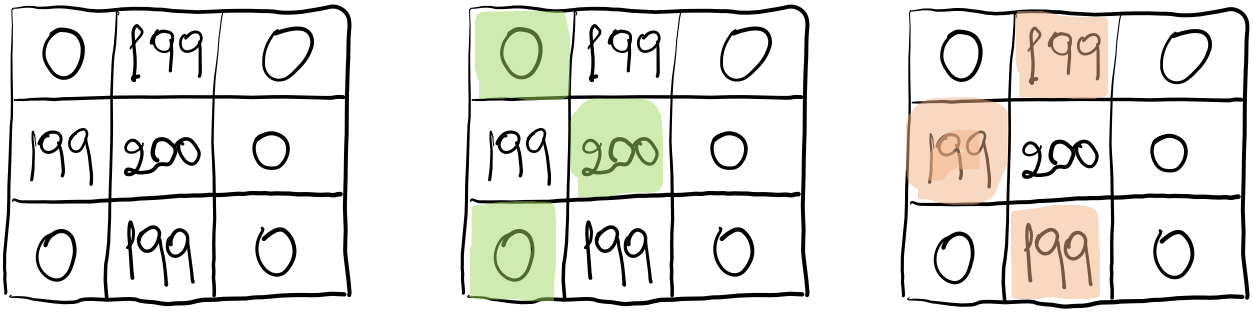
Και πράγματι $\frac{1}{\alpha} * 120 = \frac{1}{2} * 120 = 60$ την αξία δηλαδή που έδωσε ο greedy.

Άρα συμπεραίνουμε ότι ο μέγιστος λόγος $\frac{B(I)}{A(I)}$ που μπορούμε να έχουμε είναι 2.

Ωστόσο αυτά ισχύουν για την περίπτωση που το n είναι ζυγός, για παράδειγμα 4x4.

Αν μας δωθεί σαν είσοδος μια σκακιέρα μεγέθους 3x3, τότε ο λόγος σχεδόν τριπλασιάζεται.

Τίθεται παράδειγμα:



Δεξιά φαίνεται η άπλυστη λύση και αριστερά είναι η βέλτιστη λύση με μέγιστες συνολικές αξίες 200 και 597 αντίστοιχα.

Στο παράδειγμα αυτό έχουμε $\frac{B(I)}{A(I)} = \frac{597}{200} = 2{,}99$.

Άρα συμπεραίνουμε ότι ο μέγιστος προσεγγιστικός λόγος $\frac{B(I)}{A(I)}$ που μπορούμε να έχουμε είναι 3.

Σημείωση: δεν μπορούμε να έχουμε λόγο χειρότερο από 3, δηλαδή μεγαλύτερο από 3 γιατί τότε αυτό θα σήμαινε ότι υπάρχει ένα κελί με μεγαλύτερη αξία (στο παράδειγμα μεγαλύτερη από 55) και ο greedy θα το είχε βρει και θα το είχε επιλέξει πρώτο.

<u>Δυναμική προσέγγιση:</u>

Ο αλγόριθμος με την δυναμική τεχνική δίνει τη βέλτιστη λύση.

Απόδειξη:

Έστω ότι ο αλγόριθμος με την δυναμική τεχνική δεν δίνει την βέλτιστη λύση, και έστω Vmax η βέλτιστη λύση, που μεγιστοποιείται το άθροισμα των κελιών.

Έστω P η λύση που επιστρέφει ο αλγόριθμος. Ο αλγόριθμος κατασκευάζει έναν πίνακα p, στον οποίο σε κάθε κελί αποθηκεύεται η μέγιστη αξία που μπορούμε να έχουμε αν χρησιμοποιήσουμε αυτό το κελί με βάση τα προηγούμενα. Ο αλγόριθμος ξεκινάει αποθηκεύοντας στον πίνακα p στα κελιά της πρώτης γραμμής, τις αξίες τους από τον πίνακα V, καθώς δεν ύπαρχουν προηγούμενες γραμμές και κάθε κελί που επιλέγεται έχει ως μέγιστη αξία την αξία του. Για τις υπόλοιπες γραμμές, και για κάθε κελί αυτών των γραμμών, βρίσκουμε την μέγιστη αξία που έχει επιτευχθεί μέχρι στιγμής από τον πίνακα p, για την προηγούμενη γραμμή και προσθέτουμε στον p την αξία του κελιού και την μέγιστη

αξία που βρέθηκε. Προφανώς, δεν επιλέγεται κελί της ίδιας στήλης με τον έλεγχο των j (στηλών). Φτάνοντας, στο τέλος, στην τελευταία δηλαδή γραμμή, ακολουθώντας αυτόν τον αλγόριθμο, ο οποίος βρίσκει την μέγιστη συνολική αξία των κελιών, μέχρι τη δεδομένη στιγμή, υπάρχουν 2 περιπτώσεις.

- Κατά την διάρκεια του αλγορίθμου επιλέχθηκε ένα κελί διαφορετικό από την Vmax, το οποίο μεγιστοποιεί την συνολική αξία και άρα $P > Vmax$. Επομένως, η Vmax δεν είναι η βέλτιστη λύση. Άτοπο, γιατί υποθέσαμε ότι η Vmax είναι η βέλτιστη λύση, ενώ είναι η P και άρα ο αλγόριθμος δίνει την βέλτιστη λύση.
- Ο αλγόριθμος επιλέγει ακριβώς τα ίδια κελιά και άρα $P = Vmax$. Επομένως, P είναι η βέλτιστη λύση. Άτοπο, γιατί υποθέσαμε ότι ο αλγόριθμος δεν δίνει την βέλτιστη λύση.

Και τις δύο περιπτώσεις πέσαμε σε άτοπο, άρα ο αλγόριθμος δίνει την βέλτιστη λύση.

Για την δεύτερη προσέγγιση του αλγορίθμου, η λογική είναι η ίδια με μόνη διαφορά, ότι γίνεται ταυτόχρονα με τον υπολογισμό του πίνακα p, η εύρεση του max. Άρα και με την δεύτερη προσέγγιση ο αλγόριθμος είναι βέλτιστος.