Αλγόριθμοι

$2^{η}$ Σειρά Ασκήσεων

Παντελίδης Ιπποκράτης (3210150)

12-06-2023

[1]

Άσκηση 1

Το πρόβλημα αναφέρεται στην κατάλληλη επιλογή ενός υποσυνόλου από n εργαζομένους,όπου ο καθένας πληροί δύο συγκεκρίμενες συνθήκες,με σκόπο την μέγιστη δυνατή προέλευση ατόμων σε μία δεξίωση.Οι συνθήκες αναφέρουν ότι για να μπορεί κάποιος να παρευρεθεί στο πάρτι θα πρέπει να γνωρίζει τουλάχιστον άλλους 4 προσκεκλημένους,αλλά και να μην γνωρίζει τουλάχιστον άλλους 4.Για να λύσουμε το πρόβλημα θα χρησιμοποιήσουμε τον ακόλουθο αλγόριθμο.Αρχικά παρατηρούμε ότι για να επιστρέψει κάποια λύση ο αλγόριθμος θα πρέπει ο πίνακας να έχει μέγεθος τουλάχιστον 8×8,ώστε να μπορούν ικανοποιηθούν και οι δύο συνθήκες καθώς επίσης το ελάχιστο πλήθος των εργαζομένων που μπορούν να βρίσκονται στο πάρτι είναι 8.Μας δίνεται ένας πίνακας Α οι τιμές του οποίου μας δείχνουν ποιοι εργαζόμενοι γνωρίζουν ποιους και ποιοι όχι.Φτιάχνουμε ένα προσωρινό και ένα τελικό set το πλήθος του οποίου στο τέλος θα επιστρέψουμε ως το μέγιστο πλήθος εργαζομένων στο πάρτι.Για να μην συναντήσουμε πρόβλημα στην έναρξη του αλγορίθμου θα θεωρήσουμε ότι στο προσωρινό set αυτό περιέχονται αρχικά όλοι οι εργαζόμενοι και θα αφαιρούμε επαναληπτικά αυτούς για τους οποίους δεν ισχύουν οι συνθήκες σε αυτό το "τελικό" instance,ενημερώνοντας κάθε φορά τον πίνακα Α των συνδέσεων.Σε αυτό το σημείο ξεκινάμε ένα loop όπου για κάθε εργαζόμενο,δηλαδή κάθε γραμμή του πίνακα Α,αφού έχουμε ήδη βρει το άθροισμα της γραμμής(δηλαδή πληροφορίες με το πόσα άτομα γνωρίζει και πόσα όχι) ελέγχουμε αν ικανοποιεί τις συνθήκες.Αν τις ικανοποιεί τον βάζουμε στο τελικό set και συνεχίζουμε με τον επόμενο εργαζόμενο.Αν όμως όχι αφαιρούμε αυτόν τον εργαζόμενο από το προσωρινό set και ενημερώνουμε τον πίνακα Α σαν αυτός ο εργαζόμενος να μην υπάρχει,δηλαδή θεωρούμε την γραμμή και την στήλη του ανύπαρκτη,αφού πρώτα επιβεβαιώσουμε ότι οι συνθήκες των εργαζομένων που έχουν μπει στο τελικό set συνεχίζουν να ισχύουν.Αν δεν ισχύουν για κάποιον τον αφαιρούμε και αυτόν.Συνεχίζουμε τα ίδια βήματα με τους επόμενους εργαζομένους και τον ανανεωμένο πίνακα Α έως ότου να ελέγξουμε όλους τους εργαζομένους.Η ορθότητα του αλγορίθμου αποδεικνύεται από το γεγονός ότι γίνεται σωστός έλεγχος ώστε να ισχύουν πάντα και οι δύο συνθήκες στο τελικό υποσύνολο,όμως δεν επιστρέφει πάντα την βέλτιστη λύση.Η πολυπλοκότητα είναι $\mathcal{O}(n^3)$ καθώς έχουμε ένα loop που τρέχει για κάθε εργαζόμενο δηλαδή n φορές και σε κάθε επανάληψη στην χειρότερη περίπτωση θα χρειαστεί να σκανάρει όλες τις γραμμές του πίνακα Α για να τον ενημερώσει,γεγονός που χρειάζεται $\mathcal{O}(n^2)$ βήματα.Επίσης την πρώτη φορά η έυρεση του αθροίσματος κάθε γραμμής γίνεται έξω από το loop και παίρνει χρόνο πάλι $\mathcal{O}(n^2)$.Επομένως συνολικά η πολυπλοκότητα όπως αναφέραμε και πριν είναι $\mathcal{O}(n) \cdot \mathcal{O}(n^2) + \mathcal{O}(n^2) = \mathcal{O}(n^3)$.

[1]Γενική σημείωση: Η πεντάδα που αποτελεί τα βασικά στοιχεία ανάλυσης ενός αλγορίθμου δυναμικού προγραμματισμού φαίνονται υπογραμμισμένα με έντονη γραφή και στις 4 ασκήσεις,και είναι αυτά υποδεικνύουν και την ορθότητα του.

Άσκηση 2

Για να λύσουμε την συγκεκριμένη παραλλαγή του coin changing θα χρησιμοποιήσουμε την τεχνική του δυναμικού προγραμματισμού.Το πρόβλημα αναφέρεται στην ελαχιστοποίηση των νομισμάτων που μπορούμε να πάρουμε ώστε να φτάσουμε ένα ποσό E,όταν κάθε νόμισμα μπορούμε να το επιλέξουμε μόνο μία φορά.Είναι συνετό επομένως να χρησιμοποιήσουμε το συγκεκριμένο υποπρόβλημα **opt(i, j)** για να φτάσουμε στην συνολική λύση.Το υποπρόβλημα αυτό δηλώνει τον ελάχιστο αριθμό νομισμάτων που χρειάζονται για να συμπληρώσουμε το ποσό j οταν έχουμε κοιτάξει μόνο τα πρώτα i νομίσματα.Προφανώς η λύση του προβλήματος μας **sol** ως προς το ελάχιστο πλήθος νομισμάτων βρίσκεται στην θέση opt(n, E) και απο εκεί θα κάνουμε μια παρόμοια αντίστροφη διαδικασία με backtracking για να βρούμε και ποια είνα αυτά.Το χτίσιμο του πίνακα μας ξεκινάει από τις ακόλουθες βασικές περιπτώσεις **(base cases)** :

- opt(0, j) = 0, $\forall j \geq 0$ με μοναδικό τρόπο να συμπληρώνουμε ποσό j χωρίς κανένα νόμισμα όταν το j είναι ίσο με το 0.
- opt(i, 0) = 0, $\forall i$ καθώς χρειαζόμαστε 0 νομίσματα για να φτάσουμε ένα ποσό ίσο με το 0.

Για να κατασκευάσουμε την **αναδρομική σχέση** πρέπει να σκεφτούμε τι συμβαίνει κάθε φορά όταν συναντάμαι το νόμισμα i.
Έχουμε δύο περιπτώσεις σε αυτό το σημείο:

- Το νόμισμα i είναι μικρότερο ή ίσο απο το j,επομένως η βέλτιστη λύση μπορεί να το περιλαμβάνει.
  Επομένως συναντάμε και εδώ δύο περιπτώσεις:
  - Αν δεν πάρουμε το νόμισμα i στην λύση τότε το opt(i, j) παραμένει ίσο με την τιμή του προηγούμενου υποπροβλήματος, δηλαδή opt(i, j) = opt(i-1, j).
  - Αν όμως πάρουμε το νόμισμα i στην λύση τότε το opt(i, j) είναι ίσο με το υποπρόβλημα στο προηγούμενο βήμα με την διαφορά ότι αφαιρείται από το ποσό j η αξία του νομίσματος i και αυξάνεται συνολικά το πλήθος κατά ένα(λόγω του pick).Οπότε το opt(i, j) = opt(i-1, j - αξια νομισματος i) + 1

  Προκύπει λοιπόν ότι επειδή σκοπός μας είναι η ελαχιστοποίηση του πλήθους των νομισμάτων,σε κάθε βήμα να κάνουμε την επιλογή που επιτελεί αυτόν τον σκοπό.Οπότε θα παίρνουμε το ελάχιστο αυτών των δύο περιπτώσεων,δηλαδή το opt(i, j) = min{(opt(i-1, j), opt(i-1, j - αξια νομισματος i) + 1}.
- Το νόμισμα i είναι μεγαλύτερο από το ποσό j,επομένως δεν γίνεται να το πάρουμε στην λύση μας,οπότε το opt(i, j) = opt(i-1, j).

Σύμφωνα με τα παραπάνω η αναδρομική σχέση ορίζεται ως εξής:

$$opt(i,j) = \begin{cases} 0, & \text{if } i = 0 \text{ or } j = 0 \\ opt(i-1,j), & i,j \geq 1 \text{ and value of coin i } > j \\ min\{(opt(i-1,j), opt(i-1, j - \text{value of coin i}) + 1\}, & i,j \geq 1 \text{ and value of coin i } \leq j \end{cases}$$

Όπως ήδη έχουμε αναφέρει η λύση του προβλήματος μας βρίσκεται στο κελί opt(n, E), αφού κατασκευάσουμε τον πίνακα χρησιμοποιώντας την αναδρομική σχέση.Το τελευταίο κελί αποτελεί τον ελάχιστο αριθμό νομισμάτων που μπορούν να χρησιμοποιηθούν ακριβώς μία φορά και έχουν άθροισμα ίσο με το ποσό E.Αν δεν υπάρχει τρόπος να αναπαρασταθεί αυτό το ποσό με τα υπάρχοντα

νομίσματα τότε σε αυτό το κελί θα έχει μεταφερθεί η τιμή 0,υποδεικνύωντας έτσι ότι δεν υπάρχει λύση στο πρόβλημα.
Τέλος, η τεχνική του backtracking που φαίνεται στο τέλος του αλγορίθμου αναφέρεται στην μετακίνηση από το τελευταίο κελί του πίνακα προς τα πίσω στην αρχή παίρνοντας σε κάθε βήμα το νόμισμα που χρησιμοποιήθηκε στην βέλτιστη λύση.Για να το βρούμε αυτό αρκεί η συνθήκη για το αν αυτό είναι διαφορετικό απο το προηγούμενο του,που υποδηλώνει ότι σε εκείνο το βήμα πήραμε το συγκεκριμένο νόμισμα.Έτσι όταν φτάσουμε στην αρχή(μέχρι να μην ικανοποιούνται οι συνθήκες του while) θα έχουμε βρει όλα τα στοιχεία που βρίσκονται στην βέλτιστη λύση μας. Για να υπολογίσουμε την **πολυπλοκότητα** του προβλήματος πρέπει να λάβουμε υπόψη το μέγεθος του πίνακα,δηλαδή το πλήθος από κελιά,και τον χρόνο που χρειάζεται για να υπολογιστεί το κάθε κελί.Ο πίνακας μας έχει μέγεθος $n \cdot E$, δηλαδή $\mathcal{O}(n \cdot E)$, και κάθε κελί υπολογίζεται σε σταθερό χρόνο αφού έχουμε απλά το πολύ δύο προσπελάσεις και μία σύγκριση,οπότε $\mathcal{O}(1)$.Επομένως η πολυπλοκότητα συνολικά είναι $\mathcal{O}(n \cdot E)$ και δεν είναι πολυωνυμική διότι το E μπορεί να είναι πολύ μεγάλο.

Procedure 1. Coin Changing

Input: n coins with values $v_1 \geq v_2 \geq \ldots \geq v_n \geq 1$ and an amount E.
Output: Not possible or the minimum number of coins with sum E and which coins.
opt ← create 2D array of size $(len(coins) + 1) \times (amount + 1)$ ▷ Construct array
for $j = 0$ **to** E **do** ▷ Initialize first row
    $opt[0, j] \leftarrow 0$
end for
for $i = 1$ **to** n **do**
    $opt[i, 0] \leftarrow 0$ ▷ Initialize first row
    for $j = 1$ **to** E **do** ▷ Fill array
        if value of coin i $\leq j$: (coin i can be in the solution)
            opt(i, j) = min{(opt(i-1, j), opt(i-1, j - value of coin i) + 1}
        else opt(i, j) = opt(i-1, j) (if value of coin i $> j$)
    end for
end for
if opt$(n, E) = 0$ **then** ▷ Solution check
    print "No solution"
    return -1
end if
$i = n, \quad j = E, \quad S = \{\}$ ▷ Retrieve the coins of the solution
while $i \neq 0$ and $j \neq 0$ **do**
    if $opt[i, j] \neq opt[i - 1, j]$ **then**
        $S = S \cup \{i\}$
        $j = j -$ value of coin i
    end if
    $i = i - 1$
end while
return $opt(n, E), S$ ▷ Return the min #coins with sum E and a set which contains them

Άσκηση 3

Το πρόβλημα αναφέρεται στην επιλογή των κατάλληλων στάσεων σε ένα μεγάλο ταξίδι, ώστε να ελαχιστοποιήσουμε την συνολική ποινή όλων των στάσεων, αν υποθέσουμε ότι η ποινή δίνεται από την συνάρτηση $(40-x)^2$, όταν διανύουμε απόσταση x χιλιομέτρων από τη μία στάση στην άλλη. Για την απλοποίηση του συνολικού προβλήματος, μπορούμε να χρησιμοποιήσουμε τον πίνακα **opt(i)** ως υποπρόβλημα, ο οποίος συμβολίζει την ελάχιστη συνολική ποινή, δηλαδή το ελάχιστο άθροισμα των ποινών των στάσεων που έχει κάνει από την στάση 0 μέχρι την στάση i. Εύλογα προκύπτει λοιπόν, ότι η λύση του προβλήματος μας **sol** θα βρίσκεται στη θέση n του πίνακα **opt**, μιας και αυτή αναπαριστά την ελάχιστη ποινή της συνολικής διαδρομής, δηλαδή από τη θέση 0 μέχρι τη θέση n. Το γέμισμα του πίνακα ξεκινά από τη βασική περίπτωση **(base case)**: $\text{opt}(0) = 0$, καθώς έχουμε μηδενική ποινή αν δεν μετακινηθούμε καθόλου. Η **αναδρομική σχέση** προκύπτει αν σκεφτούμε ότι η απόσταση της αφετηρίας από το i θα είναι η απόσταση από την αφετηρία μέχρι μία στάση j, στην οποία θα υπάρχει η βέλτιστη προηγούμενη λύση, μαζί με την ποινή (penalty) της στάσης i από την j (δηλαδή το $(40 - \text{abs(km of j - km of i)})^2$). Για να βρούμε όμως αυτή την στάση j θα πρέπει να σαρώσουμε όλες τις στάσεις μικρότερες από την i για τις οποίες να ελαχιστοποιείται η ποσότητα που αναφέραμε, δηλαδή να ισχύει $\text{opt}(i) = \min_{j=0}^{i-1}(\text{opt}[j] + \text{penalty})$. Επομένως, όταν γεμίσει όλος ο πίνακας, τότε στην τελευταία θέση του θα βρίσκεται η λύση μας. Για να βρούμε την **πολυπλοκότητα** του αλγορίθμου, πρέπει να λάβουμε υπόψη το μέγεθος του πίνακα που γεμίζουμε, ο οποίος έχει n θέσεις, άρα $\mathcal{O}(n)$, καθώς επίσης και το χρόνο που χρειάζεται για να υπολογιστεί κάθε κελί. Στη χειρότερη περίπτωση, η οποία είναι όταν βρισκόμαστε στην τελευταία θέση, θα πρέπει να σαρώσουμε όλες τις προηγούμενες θέσεις του πίνακα για να υπολογίσουμε τη βέλτιστη τελική λύση,άρα πάλι $\mathcal{O}(n)$.Επομένως,εφόσον ο υπολογισμός κάθε κελιού γίνεται σε ένα loop από 0 μέχρι n,η συνολική πολυπλοκότητα είναι $\mathcal{O}(n) \cdot \mathcal{O}(n) = \mathcal{O}(n^2)$.

Procedure 2. Penalty minimization

```
1: Input: #stations, array with km of every station.
2: Output: Min penalty to go from station 0 to n.
3: opt(0) ← 0                                          ▷ Initialize first station
4: for i = 1 to n do
5:     opt(i) ← infinity (value greater than total distance)
6:     for j = 0 to i − 1 do
7:         penalty ← (40 − |km[j] − km[i]|)^2          ▷ Calculate penalty from j to i
8:         opt[i] ← min(opt[i], opt[j] + penalty)      ▷ Find min penalty of all previous j stations
9:     end for
10: end for
11: return opt(n)
```

[2] Άσκηση 4

Το πρόβλημα αναφέρεται στην επιλογή των κατάλληλων κελιών από μία σκακιέρα μεγέθους $n \times n$,όπου κάθε κελί διαθέτει μία αξία(βρίσκονται σε έναν άλλον $n \times n$ πίνακα,τον V),με τέτοιο τρόπο ώστε να μεγιστοποιήσουμε την συνολική αξία,δεδομένων των συνθηκών ότι μπορούμε

[2] Για ευκολία θεωρούμε ότι τα indices των πινάκων ξεκινάνε από 1 και όχι από 0 και επομένως φτάνουν μέχρι n και όχι μέχρι n-1.

να επιλέξουμε ένα κελί από κάθε γραμμή και όχι κελιά που βρίσκονται σε διαδοχικές στήλες.Έχει δοθεί μία λύση του προβλήματος με την χρήση ενός greedy αλγορίθμου,όμως εμείς θα χρησιμοποιήσουμε δυναμικό προγραμματισμό για να το λύσουμε.

(α') Για την επίλυση του συνολικού προβλήματος είναι χρήσιμο να χρησιμοποιήσουμε τον μονοδιάστατο πίνακα **opt(i)** ως υποπρόβλημα που δηλώνει την μέγιστη συνολική αξία που μπορούμε να φτάσουμε αν έχουμε κοιτάξει μέχρι την γραμμή i.Η λύση του προβλήματος μας **sol** θα βρίσκεται προφανώς στην θέση opt(n).Το χτίσιμο του πίνακα μας ξεκινάει απο την βασική περίπτωση **(base case)** σύμφωνα με την οποία η πρώτη γραμμή του πίνακα opt θα πρέπει να είναι η μέγιστη αξία όταν έχουμε μόνο κοιτάξει την 1η γραμμή,επομένως παίρνουμε το μέγιστο στοιχείο της 1ης γραμμής του πίνακα των αξιών V.Το υπόλοιπο γέμισμα του πίνακα ολοκληρώνεται χρησιμοποιώντας την ακόλουθη **αναδρομική σχέση**: opt(i) = $prevmax + currmax$ όπου prevmax είναι ουσιαστικά η μέγιστη αξία που έχουμε για όλες τις προηγούμενες γραμμές του i(δηλαδή το opt(i-1)) και το curmax είναι η τρέχουσα μέγιστη αξία της γραμμής i του πίνακα V,η οποία θα προστεθεί στο prevmax για να βρεθεί η μέγιστη συνολική αξία μέχρι και την γραμμή i,αφού πρώτα έχουμε ελέγξει μέσω της συνθήκης $j \neq prevcol$ ότι στο υπολογισμό του currmax δεν θα λαμβάνεται υπόψη η αξία της στήλης που επιλέχθηκε στο προηγούμενο υποπρόβλημα,δηλαδή στην προηγούμενη γραμμή,ικανοποιώντας έτσι την συνθήκη.Οι συγκρίσεις με το max γίνονται έναντι των αναθέσων ώστε να ενημερώσουμε τις currmax και prevmax με την μέγιστη τιμή σωστά.Επομένως,όπως αναφέραμε και πριν,όταν φτάσουμε στην θέση n του πίνακα θα έχουμε την λύση δηλαδή την μέγιστη συνολική αξία όλων των γραμμών.

Procedure 3. Value maximization

```
1: Input: 2d array with values of every cell.
2: Output: Maximum sum of values according to conditions.
3: n ← V.rows                                          ▷ Size of the array
4: prevmax ← max value of first line of V
5: prevcol ← index of max value
6: opt(1) ← prevmax                                    ▷ Base case
7: for i from 2 to n do
8:     currmax ← 0
9:     for j from 1 to n do
10:        if j ≠ prevcol then      ▷ Previous col of value selected different from current
11:            currmax ← max(currmax, V[i][j])          ▷ Find max value of line
12:            prevcol ← j                              ▷ Hold new column
13:        end if
14:    end for
15:    opt(i) ← prevmax + currmax
16:    prevmax ← max(prevmax, opt(i))                   ▷ Update prevmax
17: end for
18: return prevmax
```

(β') Για να βρούμε την **πολυπλοκότητα** του αλγορίθμου που χρησιμοποιεί την τεχνική του δυναμικού προγραμματισμού πρέπει να λάβουμε υπόψη το μέγεθος του πίνακα opt,ο οποίος έχει n θέσεις,άρα $\mathcal{O}(n)$ καθώς και τον χρόνο που χρειάζεται για να υπολογιστεί κάθε κελί του.Στην περίπτωση μας κάθε κελί προκύπτει από έναν υπολογισμό μέγιστης ποσότητας σε μία γραμμή του πίνακα V,επομένως θα σαρώσει n-1 (πλην αυτής που την απορρίπτει η συνθήκη),άρα $\mathcal{O}(n)$.Συνολικά συμπεραίνουμε ότι η πολυπλοκότητα αυτού αλγορίθμου είναι $\mathcal{O}(n) \cdot \mathcal{O}(n) = \mathcal{O}(n^2)$.Όταν χρησι-

μοποιούμε τον greedy αλγόριθμο για να υπολογίσουμε την πολυπλκότητα χρειάζεται να βρούμε το μεγαλύτερο κελί του $n \times n$ πίνακα που χρειάζεται χρόνο $\mathcal{O}(n^2)$ και αφού το βρούμε χρειαζόμαστε $\mathcal{O}(1)$ για να δούμε αν είναι σύμφωνο με τους περιορισμούς,δηλαδή να ελέγξουμε το κελί της προηγούμενης και της επόμενης στήλης.Στην χειρότερη περίπτωση θα το κάνουμε αυτό για κάθε γραμμή,δηλαδή n φορές,άρα $\mathcal{O}(n)$ και επομένως η πολυπλοκότητα συνολικά θα είναι $\mathcal{O}(n^2) \cdot \mathcal{O}(n) = \mathcal{O}(n^3)$.

(γ') Ο παραπάνω αλγόριθμος δυναμικού προγραμματισμού που αναφέραμε παραπάνω μας οδηγεί στην βέλτιστη λύση και θα το αποδείξουμε μέσω επαγωγής.Αρχικά να αναφέρουμε ότι ο αλγόριθμος λειτουργεί εξερευνόντας όλους τους πιθανούς συνδυασμούς επιλογής κελιών,χωρίς να παραβιάζει κανέναν από τους περιορισμούς.Κάθε γραμμή i εξετάζεται χωριστά και αποφεύγεται τελείως η επιλογή αξιών κελιών προηγούμενων συνεχώς και επομένως συνολικά και επόμενων γραμμών.Η χρησιμοποιήση όλων των κελιών οδηγεί στο ότι καμία από τις λύσεις δεν χάνεται.Όπως αναφέραμε όμως,για να δείξουμε ότι ο αλγόριθμος είναι βέλτιστος χρειάζεται απόδειξη:

- Επαγωγική Βάση: Όταν κοιτάμε την γραμμή i=1 το υποπρόβλημα μας παίρνει την μέγιστη τιμή της πρώτης γραμμής του πίνακα των αξιών.Προφανώς και αυτή είναι η βέλτιστη λύση καθώς δεν μπορούμε να πάρουμε μεγαλύτερη αξία με τις υπάρχουσες τιμές.

- Επαγωγική Υπόθεση: Υποθέτουμε ότι ο αλγόριθμος μας βρίσκει την βέλτιση λύση για κάθε υποπρόβλημα μέχρι και την γραμμή i-1.

- Επαγωγικό Βήμα: Για να αποδείξουμε ότι δίνει την βέλτιστη λύση συνολικά αρκεί να αποδείξουμε ότι δίνει την βέλτιστη λύση και για το υποπρόβλημα i.Ο αλγόριθμος υπολογίζει την μεγαλύτερη αξία της τρέχουσας γραμμής(currmax),δηλαδή της i,αναιρώντας την αξία του κελιού που ανήκει στην στήλη με την αξία που επιλέχθηκε στην προηγούμενη γραμμή(prevcol) και αφού την βρει την προσθέτει στην αξία των προηγούμενων βέλτιστων λύσεων.Προφανώς σε αυτό το σημείο έχουμε φτάσει στην μέγιστη αξία που μπορούμε να φτάσουμε μέχρι την γραμμή i δεδομένων των συνθηκών.

Επομένως ο αλγόριθμος βρίσκει την βέλιτστη λύση για όλα τα υποπροβλήματα μέχρι το n-οστό,και έτσι παρέχει την βέλτιστη λύση για το συνολικό πρόβλημα.

Από την άλλη πλευρά ο greedy αλγόριθμος δεν παρέχει την βέλτιστη λύση στο πρόβλημα και για το δείξουμε αυτό αρκεί να φτιάξουμε ένα στιγμυότυπο του πίνακα V σύμφωνα με το οποίο η λύση που επιστρέφει ο greedy να διαφέρει απο την βέλτιστη που μπορούμε να πετύχουμε.Ας θεωρήσουμε λοιπόν τον ακόλουθο πίνακα V1: $\begin{bmatrix} 2 & 9 & 3 \\ 1 & 8 & 1 \\ 2 & 1 & 2 \end{bmatrix}$.Στον συγκεκριμένο πίνακα αν καλέσουμε τον greedy αλγόριθμο αρχικά θα επιλέξει το μέγιστο του συνολικού πίνακα,δηλαδή το 9 από το κελί (3,2),στην συνέχεια το 8 αλλά δεν θα το συμπεριλάβει επειδή το 9 βρίσκεται στην ίδια στήλη,επομένως θα πάει στο επόμενο μέγιστο του πίνακα πλην των γραμμών που έχει χρησιμοποιήσει,δηλαδή σε ένα από τα δύο 2 στην πρώτη γραμμή.Τέλος θα πάρει έναν από τους δύο άσσους της δεύτερης γραμμής.Επομένως θα συγκεντρώσει συνολική αξία 12,ενώ αν παρατηρησουμε καλύτερα η βέλτιστη λύση είναι 13(αν πάρουμε ένα 2 από την πρώτη γραμμή,το 8 από την δεύτερη,και το 3 από την τρίτη).Στην περίπτωση μας το B(I) = 13 και το A(I) = 12 επομένως ο λόγος τους B(I)/A(I) είναι μεγαλύτερος από 1,άρα ο greedy δεν είναι βέλτιστος.Για να βρούμε το α δηλαδή τον μέγιστο λόγο θα πρέπει να βρούμε το χειρότερο στιγμυότυπο.Θεωρούμε ως το χειρότερο στιγμυότυπο το V2: $\begin{bmatrix} 0 & 0 & 10 \\ 0 & 10 & 11 \\ 0 & 0 & 10 \end{bmatrix}$.Σε αυτήν την περίπτωση η βέλτιστη λύση

B(I) = 30 και η λύση που επιστρέφει ο greedy A(I) = 11.Παρατηρούμε λοιπόν ότι ο greedy αλγόριθμος στην χειρότερη περίπτωση επιστρέφει συνολική αξία που φτάνει περίπου το 1/3 της βέλτιστης.Μπορούμε να παρατηρήσουμε ότι αν συνεχίσουμε να αυξάνουμε των θετικών κελιών κατά 1 ο λόγος B(I)/A(I) τείνει όλο και πιο κοντά στο 3.Επομένως όσο πιο μεγάλους αριθμούς θα χρησιμοποιήσουμε στο στιγμυότυπο μας τόσο πιο πολύ θα χειροτερεύει ο λόγος B(I)/A(I).Έτσι ο greedy θα παράγει μια λύση που θα είναι σχεδόν (αν όχι ακριβώς για πολύ μεγάλους αριθμούς) 3 φορές μικρότερη από την βέλτιστη λύση.