Μαρία Σχοινάκη: 3210191

ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

ΣΧΟΛΗ ΕΠΙΣΤΗΜΩΝ & ΤΕΧΝΟΛΟΓΙΑΣ ΤΗΣ ΠΛΗΡΟΦΟΡΙΑΣ SCHOOL OF INFORMATION SCIENCES & TECHNOLOGY

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ DEPARTMENT OF INFORMATICS

ΑΛΓΟΡΙΘΜΟΙ

Διδάσκουσα : Αλκμήνη Σγουρίτσα

1η Σειρά Ασκήσεων - Εαρινό Εξάμηνο 2023

Ονοματεπώνυμο:	***Μαρία Σχοινάκη***
Αριθμός Μητρώου:	***3210191***
Email:	***p3210191@aueb.gr***

Ασκήσεις

Μαρία Σχοινάκη: 3210191

Άσκηση 1

Ας υποθέσουμε αρχικά ότι στον πίνακα εισόδου όπου i = j έχουμε 0

ΠΕΡΙΓΡΑΦΗ

1) Δημιουργούμε μια λίστα καλεσμένων με τους εργαζόμενους και 2 κενές λίστες known, unknown(n).

2) Για κάθε εργαζόμενο i σε μία επανάληψη από 1 έως n, τρέχουμε την συνάρτηση **sum** και αποθηκεύουμε το αποτέλεσμα στην θέση **known[i]**, ενώ αποθηκεύουμε στο **unknown[i]**, το **n - known[i] – 1**. Η sum τρέχει σε **O(n)** άρα η διαδικασία του βήματος 2 τρέχει σε **O(n^2)** λόγω και της επανάληψης. Το **known[i]** περιέχει το πλήθος τον γνωριμιών του εργαζομένου **i**, ενώ το **unknown[i]**, το πλήθος των ξένων του **i**.

3) Αρχικοποιούμε μία μεταβλητή i από 0. Τρέχουμε μία επανάληψη **while true**(τρέχει μέχρι να την σταματήσουμε) και σε μια άλλη επανάληψη την λίστα μας(εσωτερικά). Αν είναι άδεια ή δεν υπάρχει κάποιο άτομο που να μην πληροί τις προϋποθέσεις, σταματάμε την επανάληψη και επιστρέφουμε τη λίστα καλεσμένων που περιέχει τους εργαζόμενους που πρέπει να προσκληθούν στη δεξίωση. Αν η λίστα έχει έστω και 1 άτομο που δεν πληροί τις προϋποθέσεις(Δηλαδή αν unknown[i]<4 or known[i]<4), τότε κάνουμε i = αυτό το άτομο(πρώτο που θα βρει).

4) Τρέχουμε εσωτερικά μια επανάληψη από 1 μέχρι n. Ελέγχουμε αν ο i με τον j γνωρίζονται και αν ναι μειώνουμε το known[j] κατά 1. Αν δεν γνωρίζονται και i!=j, τότε μειώνουμε το unknown[j] κατά 1. Τέλος, αφαιρούμε τον εργαζόμενο από την λίστα.

ΠΟΛΥΠΛΟΚΟΤΗΤΑ

Όλες οι αναθέσεις τιμών καθώς και το return γίνονται σε χρόνο O(1). Η δημιουργία της λίστας καλεσμένων, χρειάζεται χρόνο O(n). Το γέμισμα τον λιστών known και unknown χρειάζεται χρόνο O(n^2) καθώς η συνάρτηση sum τρέχει σε O(n). Προχωράμε στον βασικό βρόγχο. Ας ξεκινήσουμε από το εσωτερικό του. Ο έλεγχος της λίστας, αν είναι άδεια, αν περιέχει ή όχι κάποιο εσφαλμένο στοιχείο, γίνεται σε O(n), καθώς τρέχουμε την λίστα. Η συνθήκη ελέγχου και οι αναθέσεις τιμών που ακολουθούν γίνονται σε σταθερό χρόνο. Για να αφαιρέσουμε έναν εργαζόμενο από τους καλεσμένους, χρειαζόμαστε O(n) χρόνο, καθώς με δεδομένο το i, τρέχουμε μια επανάληψη η οποία ελέγχει αν κάθε j != i, γνωρίζεται με τον i. Οι πράξεις για τις 2 άλλες λίστες είναι επίσης αμελητέες. Ο εξωτερικός βρόγχος, ο οποίος τρέχει μέχρι να τον σταματήσουμε, ουσιαστικά τρέχει σε O(n), γιατί στην χειρότερη περίπτωση θα τρέξει για όλους τους εργαζόμενους. Έτσι λόγω εμφόλευσης η while τρέχει τελικά σε O(n^2), καθώς έχουμε n(n+ n) = n(2n) = 2n^2 που είναι της τάξεως του n^2.
Άρα προσεγγιστικά O(n) + O(1) + O(n^2) + O(n^2) + O(1) = **O(n^2)**

Μαρία Σχοινάκη: 3210191

ΟΡΘΟΤΗΤΑ

Για να αποδείξουμε την ορθότητα του αλγορίθμου, αρκεί να αποδείξουμε ότι η λογική επαναληπτικής αφαίρεσης μη κατάλληλων εργαζομένων από την λίστα των καλεσμένων είναι σωστή. Ο αλγόριθμός μας δίνει την ευκαιρία σε όλους τους εργαζομένους να έρθουν, έτσι ώστε να έχουμε το βέλτιστο(μέγιστο) αποτέλεσμα. Επειδή ο περιορισμός μιλάει για καλεσμένους και όχι για εργαζόμενους, αυτό σημαίνει ότι για να καλεστεί ένας εργαζόμενος, πρέπει να έχουν ήδη καλεστεί τουλάχιστον 4 που ξέρει και τουλάχιστον 4 που δεν ξέρει. Υπάρχουν 3 σενάρια. Είτε υποθετικά θα πληρούν όλοι τις προϋποθέσεις και θα επιστραφεί η λίστα άθιχτη. Είτε κανένας δεν θα τις πληροί, οπότε θα επιστραφεί μια άδεια λίστα, είτε κάποιοι θα τις πληρούν και κάποιοι όχι. Το θέμα είναι ότι αν εν τέλη δεν καλεστεί ο εργαζόμενος x, τότε ο εργαζόμενος y που τον ήξερε, πλέον πρέπει να σημειωθεί ότι ξέρει έναν λιγότερο. Και αν ο z δεν ήξερε τον y, τότε ο z έχει πλέον έναν λιγότερο ξένο. Γιαυτό σε κάθε διαγραφή κάποιου εργαζομένου από την λίστα των καλεσμένων πρέπει να μεταβάλουμε σωστά τις λίστες με τις γνωριμίες και να ξαναελέγξουμε τυχόν μη τήρηση των προϋποθέσεών. Ο αλγόριθμός μας μεταβάλει σωστά τις λίστες και ελέγχει αλλεπάλληλες φορές αν όσοι παραμένουν στην λίστα έχουν υποστεί τέτοια ζημιά, ώστε να μην μπορούν πλέον να καλεστούν. Επίσης, επειδή ξεκινάμε με όλους τους εργαζομένους και αφαιρούμε πυραμιδικά μόνο αυτούς που εξαρχής δεν πληρούν τις προϋποθέσεις, και στην συνέχεια όσους υποστούν ζημιά, μεγιστοποιούμε τους καλεσμένους. Άρα ο αλγόριθμός μας είναι ορθός, καθώς ελέγχει όλα τα πιθανά σενάρια και προσπαθεί για την βέλτιστη απόδοση.

Άσκηση 2

ΚΩΔΙΚΑΣ

```
coin_changing(coins, E)
    n = len(coins)
    DP = [[float('inf')] * (E + 1) for i = 0 to (n + 1)]
    for i = 0 to n + 1 do
            DP[i][0] = 0
    end for
    for i = 1 to (n + 1) do                    #coins
            coin = coins[i - 1]
            for j = 1 to (E + 1) do            #capacity
                    if coin <= j then
                            DP[i][j] = min(DP[i - 1 ][j], 1 + DP[i - 1][j - coin])
                    else:
                            DP[i][j] = DP[i - 1][j]
                    end if
            end for
    end for
    if DP[n][E] == 'inf' then
            return "Αδύνατο"
    end if
    i, j = n, E
    coins_used = []
```

Μαρία Σχοινάκη: 3210191

```
while i != 0 and j != 0 do
        coin = coins[i - 1]
        if DP[i][j] != DP[i - 1][j] then
                coins_used.append(coin) j -= coins[i - 1]
        end if
        i -= 1
end while
return coins_used
```

ΠΟΛΥΠΛΟΚΟΤΗΤΑ

Όπου n είναι το πλήθος των νομισμάτων και όπου E η αξία-ποσό.
Όλες οι αναθέσεις τιμών καθώς και το return γίνονται σε χρόνο O(1). Η αρχικοποίηση – δημιουργία του πίνακα DP γίνεται σε O(nE), καθώς φτιάχνουμε έναν δισδιάστατο πίνακα με στήλες τα νομίσματα και γραμμές από 0 έως E το ποσό. Το 1^{ο} for για την αρχικοποίηση της 1^{ης} στήλης γίνεται σε O(n), αφού έχουμε n στήλες(νομίσματα). Ο υπολογισμός του πίνακα DP γίνεται σε O(nE), καθώς έχουμε 2 for, το ένα εμφολευμένο. Το 1^{ο} for τρέχει σε O(n), ενώ το 2^{ο} σε O(E). Άρα nE, δηλαδή O(nE). Το if για το αν υπάρχει λύση γίνεται σε O(1), ενώ η επανάληψη για την κατασκευή της λίστας των νομισμάτων τρέχει σε O(n) προφανώς. Άρα O(nE) + O(n) + O(nE) + O(1) + O(1) + O(n) + O(1) = **O(nE)**

ΟΡΘΟΤΗΤΑ

Η αναδρομική σχέση αναθέτει την τιμή του κελιού DP[i][j] βάσει των προηγούμενων τιμών του πίνακα. Ανάλογα με την αξία του coin και το εναπομείναντα κόστος j, υπάρχουν δύο πιθανές περιπτώσεις:

1) Η περίπτωση στην οποία αναθέτουμε την τιμή του κελιού DP[i][j] με την τιμή στην ίδια στήλη, της προηγούμενης γραμμής DP[i-1][j]. Η περίπτωση αυτή στην ουσία για την τρέχουσα τιμή του ποσού(στήλη) βάζει στην θέση του νομίσματος i την τιμή του προηγούμενου νομίσματος στην ίδια στήλη. Δηλαδή το τρέχον νόμισμα δεν χρησιμοποιείται. Αυτό συμβαίνει γιατί το νόμισμά μας δεν χωράει στην τρέχουσα τιμή του ποσού.
2) Η περίπτωση στην οποία, αναθέτουμε την τιμή του κελιού DP[i][j] με την μικρότερη τιμή από τις
   α) Είτε την τιμή στην ίδια στήλη, της προηγούμενης γραμμής DP[i-1][j]
   β) Είτε στην προηγούμενη γραμμή, στην στήλη της διαφοράς της τρέχουσας τιμής του ποσού με το τρέχον νόμισμα, + 1. Ο λόγος ουσιαστικά είναι ότι αυτή η τιμή, αντιπροσωπεύει τον αριθμό των νομισμάτων που χρησιμοποιούνται συν τον αριθμό των νομισμάτων που απαιτούνται για να σχηματιστεί το υπόλοιπο της αξίας j – coin(δηλ. 1).

Δηλαδή το τρέχον νόμισμα χρησιμοποιείται. Αυτό συμβαίνει γιατί το νόμισμά μας χωράει στην τρέχουσα τιμή του ποσού.
Ξεκινώντας από τις βάσεις(1^{η} στήλη) και υπολογίζοντας σειριακά τον πίνακα DP, η αναδρομική σχέση επιλύει το πρόβλημα επαναληπτικά. Στην ουσία, σε κάθε επανάληψη, κάθε γραμμή γεμίζει με το πλήθος των νομισμάτων που συντελούν ως άθροισμα στην συμπλήρωση της τρέχοντος τιμής του ποσού(προφανώς νομίσματα που έχει διαβάσει). Η αναδρομική σχέση εξετάζει όλες τις πιθανές περιπτώσεις και μάλιστα επιλέγει την ελάχιστη δυνατή.

Μαρία Σχοινάκη: 3210191

Άσκηση 3

ΚΩΔΙΚΑΣ

```
roadtrip(a):
   n = length(a) - 1
   DP = [infinity] * (n + 1)
   DP[0] = 0

   for i = 1 to n:
      for j = 0 to i - 1:
         penalty = (40 - (a[i] - a[j])) ^ 2
         DP[i] = min(DP[i], DP[j] + penalty)

   stops = []
   i = n
   while i > 0:
      for j = i - 1 to 0 step -1:
         penalty = (40 - (a[i] - a[j]))^2
         if DP[i] == DP[j] + penalty:
            stops.append(a[i])
            i = j
            break

   reverse(stops)
   return stops, DP[n]
```

ΠΕΡΙΓΡΑΦΗ

1) Αρχικά, υπολογίζουμε το πλήθος των σταθμών και αρχικοποιούμε τον πίνακα που θα περιέχει τα ελάχιστα penalty από την στάση 0 μέχρι τον σταθμό(i).Γεμίζουμε με «άπειρο» τον πίνακα γιατί ψάχνουμε ελάχιστες τιμές και η $1^{η}$ τιμή γίνεται 0, γιατί από εκεί ξεκινάμε(σταθμός 0).

2) Αμέσως μετά, τρέχουμε έναν επαναληπτικό αλγόριθμο για να υπολογίσουμε τις βέλτιστες τιμές DP[i] για κάθε σταθμό i από το 1 έως το n. Αυτό γίνεται μέσω ενός εσωτερικού βρόγχου που εξετάζει όλες τις πιθανές προηγούμενες στάσεις j(πριν τον i) για τον τρέχοντα σταθμό i.
   Στο εσωτερικό του βρόγχου υπολογίζεται το πέναλτι με βάση την απόσταση μεταξύ των σταθμών a[i] και a[j]. Στη συνέχεια, ενημερώνεται η τιμή του DP[i] με το ελάχιστο από την τρέχουσα τιμή του DP[i] και το DP[j] + penalty.

3) Πλέον, έχοντας τον πίνακα DP έτοιμο, υπολογίζουμε την λίστα stops, που περιέχει το βέλτιστο μονοπάτι στάσεων. Αρχικοποιείται μια μεταβλητή i με την τιμή n, και στη συνέχεια επαναλαμβάνεται ένας βρόγχος μέχρι το i να γίνει μηδέν. Ο βρόχος αυτός ελέγχει για κάθε j από το i-1 μέχρι το 0 αν η τιμή του DP[i] είναι ίση με DP[j] + penalty. Αν ισχύει αυτό, προστίθεται ο σταθμός a[i] στη λίστα stops, η τιμή του i

ενημερώνεται με την τιμή του j, και ο βρόγχος διακόπτεται. Ουσιαστικά, τρέχοντας από το τέλος προς την αρχή τον πίνακα a, ψάχνουμε την διαδρομή που οδήγησε στο ελάχιστο πέναλτι της στάσης n. Όταν βρούμε ένα κατάλληλο j, που σηματοδοτεί την προηγούμενη στάση πριν την n, κάνουμε την ίδια δουλειά που κάναμε για την στάση n μέχρι να βρούμε την στάση j. Στην ουσία επειδή κάθε πέναλτι μιας στάσης πιθανώς περιέχει και πέναλτι από άλλες στάσεις(λόγω ελάχιστου πέναλτι), με αυτόν τον ανάποδα επαναληπτικό αλγόριθμο, αποσυνθέτουμε τα πέναλτι που συντελούν στην βέλτιστη διαδρομή.

4) Τέλος, η λίστα stops αντιστρέφεται, ώστε να ξεκινάμε με στάση 0 και να τελειώνουμε με n και επιστρέφεται. Αυτή η λίστα περιέχει τους σταθμούς της βέλτιστης διαδρομής από τον τελευταίο σταθμό προς τον πρώτο. Επιστρέφουμε επίσης το ελάχιστο πέναλτι.

ΠΟΛΥΠΛΟΚΟΤΗΤΑ

Όλες οι αναθέσεις τιμών καθώς και το return, γίνονται σε χρόνο O(1). Η αρχικοποίηση του πίνακα DP, γίνεται σε O(n). Ξεκινώντας την βασική επανάληψη παρατηρούμε ότι το j μιμείται το i, με αποτέλεσμα η πρώτη j να τρέξει για 0, η δεύτερη για 0, 1, κτλπ. Έτσι το συνολικό πρόβλημα πλήθους επαναλήψεων ανάγεται στο (n(n+1))/2. Το οποίο είναι τάξης προφανώς O(n^2). Όλες οι πράξεις μέσα στον εμφολευμένο βρόχο εκτελούνται σε σταθερό O(1). Έπειτα, ακολουθούν αρχικοποιήσεις που τρέχουν σε O(1). Τέλος, ακολουθεί η while, η οποία εξαρτάται από το i. Το i, αλλάζει καταλλήλως μέσα στην for και μπορεί να μην αλλάζει γραμμικά. Δηλαδή από 10 να γίνει 7. Όμως υπάρχει και το ενδεχόμενο να αλλάζει γραμμικά, που σημαίνει ότι η while θα τρέχει n φορές, όπως και η for και επειδή είναι εμφολευμένη θα είναι της τάξεως του n^2, δηλαδή O(n^2). Ο έλεγχος και οι πράξεις μέσα στην for γίνονται σε σταθερό χρόνο. Η reverse, που αναποδογυρίζει την λίστα stops, τρέχει σε O(n), καθώς πρέπει να διαβάσει και να μετακινήσει όλα τα αντικείμενα. Τέλος όπως αναφέραμε και πιο πάνω η επιστροφή, γίνεται σε O(1).
Άρα O(1) + O(n) + O(1) + O(n^2) + O(1) + O(n^2) + O(n) + O(1) = **O(n^2)**

ΟΡΘΟΤΗΤΑ

Για να αποδείξουμε την ορθότητα του αλγορίθμου αρκεί να αποδείξουμε τις 2 βασικές πτυχές του:

1) Σκοπός του αλγορίθμου είναι για κάθε σταθμό που δίνεται να ελέγχει μέχρι εκείνο τον σταθμό το ελάχιστο πέναλτι. Πράγμα το οποίο επιτυγχάνεται, καθώς η αναδρομική σχέση θέτει το κελί DP[i] ίσο με το ελάχιστο πέναλτι. Είτε το υπάρχον είτε για κάθε απόσταση μεταξύ i και j. Γιαυτό ο πίνακας αρχικά ορίζεται με άπειρες τιμές. Με την πρώτη επανάληψη η τιμή αυτή θα αλλάξει όποια και αν είναι(απόσταση 0 με i). Αν είναι η ελάχιστη, τότε όντως δεν θα ξαναλλάξει γιαυτόν τον σταθμό. Αν πάλι δεν είναι θα τρέξει με την επανάληψη j όλους τους σταθμούς μέχρι τον i, ψάχνοντας το ελάχιστο πέναλτι. Ξεκινώντας από τον 1^{ο} σταθμό μετά το 0 και ξέροντας ότι ακολουθείται η ίδια διαδικασία εύρεσης του ελαχίστου πέναλτι για κάθε σταθμό, δεν μπορούμε παρά να πούμε ότι, αφού έχουμε λάβει υπόψιν την αλληλεξάρτηση των σταθμών ως προς τις ελάχιστες τιμές πέναλτι και ελέγχουμε όλα τα πιθανά σενάρια συνδυασμών στάσεων, ο αλγόριθμος μας για την δημιουργία του πίνακα DP είναι ορθός(περιέχει ελάχιστη τιμή πέναλτι για κάθε i).

2) Σκοπός επίσης του αλγορίθμου είναι η επιστροφή της λίστας που περιέχει το σωστό μονοπάτι στάσεων. Με ακριβώς ανάλογη συμπεριφορά, τρέχουμε από το τέλος προς την αρχή τις στάσεις για να βρούμε το βέλτιστο μονοπάτι. Στην ουσία επειδή κάθε πέναλτι μιας στάσης πιθανώς περιέχει και πέναλτι από άλλες στάσεις(λόγω ελάχιστου πέναλτι), με αυτόν τον ανάποδα επαναληπτικό αλγόριθμο, αποσυνθέτουμε τα πέναλτι που συντελούν στην βέλτιστη διαδρομή. Με βάση όμως το τελικό αποτέλεσμα του πίνακα DP(το DP[n]) που είναι και το ελάχιστο πέναλτι, ξεκινάμε και βρίσκουμε τον κατάλληλο σταθμό μετά το n. Σύμφωνα με αυτόν τον σταθμό, βρίσκουμε τον επόμενο κοκ, μέχρι να φτάσουμε στον 1ο σταθμό. Είναι σαν να χτίζουμε την λύση βήμα βήμα και μετά από το τέλος προς την αρχή να την αναλύουμε. Αφού όπως είπαμε είναι η ανάποδη λογική με την δημιουργία του DP, άρα και αυτή η διαδικασία είναι ορθή.

Άσκηση 4

α)

ΚΩΔΙΚΑΣ

```
chess(C)
    n = length(C)
    DP = [[0] * n for i in range(n)]
    DP[0] = C[0]
    max_positions = [[0] * 2 for i in range(n - 1)]

    for i = 1 to n - 1:
        first_max = 0
        first_position = 0
        second_max = 0
        second_position = 0
        for j = 0 to n - 1:
            if DP[i - 1][j] > first_max:
                second_max = first_max
                second_position = first_position
                first_max = DP[i - 1][j]
                first_position = j
            else if DP[i - 1][j] > second_max:
                second_max = DP[i - 1][j]
                second_position = j
        end for
        max_positions[i - 1][0] = first_position
        max_positions[i - 1][1] = second_position
        for j = 0 to n - 1:
            if first_position == j:
                DP[i][j] = C[i][j] + second_max
            else:
                DP[i][j] = C[i][j] + first_max
```

Μαρία Σχοινάκη: 3210191

```
        end for
    end for
    max_result = 0
    selected_cells = [0] * n
    for i = 0 to n - 1:
       if DP[n - 1][i] > max_result:
          selected_cells[n - 1] = i
          max_result = DP[n - 1][i]
    end for
    for i = 1 to n - 1:
       if max_positions[n - i - 1][0] == selected_cells[n - i]:
          selected_cells[n - i - 1] = max_positions[n - i - 1][1]
      else:
          selected_cells[n - i - 1] = max_positions[n - i - 1][0]
    end for
    return selected_cells, max_result
```

ΠΕΡΙΓΡΑΦΗ

1) Αρχικά, αρχικοποιούμε τους πίνακες max_positions και DP. Θέτουμε την πρώτη γραμμή του DP ίση με την πρώτη γραμμή του πίνακα εισόδου, καθώς η πρώτη αυτή γραμμή είναι η βάση της διαδικασίας μας

2) Ξεκινάμε την βασική επανάληψη για την δημιουργία του πίνακα DP. Ο πίνακας αυτός περιέχει σε κάθε θέση το μέγιστο ποσό, σεβόμενος τον περιορισμό. Για κάθε γραμμή, βρίσκουμε τα 2 μέγιστα στοιχεία καθώς και τις θέσεις τους, τις οποίες αποθηκεύουμε στον πίνακα max_positions. Έπειτα ελέγχουμε την επόμενη γραμμή μέσα στην ίδια επανάληψη. Για κάθε στοιχείο που προσπελαύνουμε, ελέγχουμε αν βρίσκεται στην ίδια στήλη με το μέγιστο της προηγούμενης γραμμής. Αν ναι τότε στον DP αποθηκεύουμε στην θέση του στοιχείου, το άθροισμα του στοιχείου με το δεύτερο μεγαλύτερο στοιχείο της προηγούμενης γραμμής. Αν όχι τότε, στον DP αποθηκεύουμε στην θέση του στοιχείου, το άθροισμα του στοιχείου με το μεγαλύτερο στοιχείο της προηγούμενης γραμμής.

3) Έχοντας έτοιμο τον πίνακα DP, πλέον η μέγιστη αξία του πίνακα είναι το μέγιστο στοιχείο της τελευταίας γραμμής του. Με μία επανάληψη το βρίσκουμε, το αποθηκεύουμε στον πίνακα selected_cells, που μόλις έχουμε ορίσει, και αποθηκεύουμε την θέση του.

4) Στο τελευταίο βήμα μας κάνουμε μια επανάληψη από κάτω προς τα πάνω τον πίνακα DP, για να βρούμε την ακολουθία των κελιών που συντέλεσαν στο βέλτιστο αποτέλεσμα. Έτσι, για κάθε γραμμή, ελέγχουμε αν το μέγιστό της βρίσκεται στην ίδια στήλη με το μέγιστο της προηγούμενης γραμμής. Αν ναι, τότε αποθηκεύουμε και δουλεύουμε με το $2^{ο}$ μέγιστο της προηγούμενης γραμμής, αλλιώς με το πρώτο. Η διαδικασία αυτή συνεχίζεται μέχρι να φτάσουμε στην πρώτη γραμμή.

5) Τέλος, επιστρέφουμε την λίστα με τα κελιά και το βέλτιστο ποσό.

Μαρία Σχοινάκη: 3210191

β)

ΠΟΛΥΠΛΟΚΟΤΗΤΑ DP

Όλες οι αναθέσεις τιμών καθώς και το return, γίνονται σε χρόνο O(1). Η αρχικοποίηση του πίνακα DP γίνεται σε O(n^2) και του max_positions σε O(n). Η επανάληψη που δημιουργεί τον πίνακα DP γίνεται σε O(n^2). Ο εξωτερικός βρόγχος τρέχει σε O(n). Οι εσωτερικοί βρόγχοι τρέχουν σε O(n) ο καθένας και κάθε πράξη ανάθεσης και ελέγχου(if) τρέχουν σε σταθερό χρόνο O(1). Άρα στην ουσία μέσα στον εξωτερικό βρόγχο έχουμε n + n = 2n επαναλήψεις που είναι της τάξης n. Άρα συνολικά αυτό το κομμάτι του αλγορίθμου τρέχει σε O(n^2). Η αρχικοποίηση του selected_cells γίνεται σε O(n). Οι 2 επαναλήψεις που ακολουθούν, τρέχουν ξεχωριστά σε O(n), με τις πράξεις εσωτερικά να τρέχουν σε σταθερό χρόνο. Άρα O(1) + O(n^2) + O(n) + O(n^2) + O(n) + O(n) + O(n) + O(1) = **O(n^2)**

ΠΟΛΥΠΛΟΚΟΤΗΤΑ GREEDY

Στον άπληστο αλγόριθμο, επειδή ψάχνουμε κάθε φορά το max στοιχείο του πίνακα, έχουμε πολυπλοκότητα O(n^2). Η συνθήκη με τις διαδοχικές σειρές δεν επηρεάζει την πολυπλοκότητα, καθώς είναι σταθερή και για να βρούμε τα n max στοιχεία, πρέπει τουλάχιστον να διαβάσουμε όλο τον πίνακα, δηλαδή πολυπλοκότητα O(n^2).

γ)

Ο αλγόριθμος του δυναμικού προγραμματισμού δίνει την βέλτιστη λύση, ενώ του άπληστου αλγορίθμου όχι.
Ας δείξουμε ένα αντιπαράδειγμα στο οποίο ο άπληστος αλγόριθμος δεν δίνει την βέλτιστη λύση.

ΑΝΤΙΠΑΡΑΔΕΙΓΜΑ GREEDY

ΠΑΡΑΓΕΙΓΜΑ ΠΙΝΑΚΑ ΕΙΣΟΔΟΥ

1	**0**	**9**
0	**9**	**10**
1	**0**	**9**

Ο άπληστος αλγόριθμος, θα ξεκινήσει βρίσκοντας το μέγιστο, το 10. Μετά μην μπορώντας να πάρει από την ίδια γραμμή και στήλη με το 10, θα στραφεί στον άσσο της πρώτης γραμμής και μετά στον άσσο της τελευταίας, δίνοντας αξία 10 + 1 + 1 = **12**.
Ο δυναμικός αλγόριθμος θα βρει αρχικά το μέγιστο της πρώτης γραμμής το 9. Στην δεύτερη θα προσθέσει το 9 σε όλα τα στοιχεία εκτός του 10 όπου θα προσθέσει το 1. Η δεύτερη γραμμή τώρα έχει μέγιστο στην $2^η$ γραμμή το 18. Θα προσθέσει στην $3^η$ γραμμή το 18, εκτός την $2^η$ στήλη όπου θα προσθέσει 11. Πλέον το μέγιστο στοιχείο της $3^{ης}$ γραμμής και μέγιστη συνολική αξία είναι το **27**.
Ο πίνακας DP:

1	**0**	**9**
9	**18**	**11**
19	**11**	**27**

Μαρία Σχοινάκη: 3210191

GREEDY -> 12
DP -> 27

Σημείωση: Ο DP δίνει την βέλτιστη λύση

ΧΕΙΡΟΤΕΡΟ ΣΤΙΓΜΙΟΤΥΠΟ GREEDY

Η ιδέα γενικά είναι να μεγιστοποιήσουμε την διαφορά μεταξύ του **greedy** και του **DP** αλγορίθμου. Ας πάρουμε σαν **βάση** έναν **2x2** πίνακα:

0	**2**
2	**3**

Όσο τα νούμερα αυξάνονται, τόσο αυξάνεται και το κλάσμα B(I)/A(I)

Πχ ενώ έχουμε 4/3 αν αυξήσουμε τα νούμερα μόνο κατά 1 μονάδα:

0	**3**
3	**4**

Θα έχουμε **6/4 > 4/3**

Ο λόγος προφανώς είναι ότι ενώ του **DP** αυξήθηκε κατά **2** του **greedy** αυξήθηκε κατά **1** γιατί υπάρχει η αναλογία **½**

Για ζυγά n

Γενικά, η αναλογία όσο μεγαλώνουν τα νούμερα θα τείνει στο **2**
Δηλαδή **α = maxI B(I)/A(I) = 2**

Ο λόγος είναι προφανής. Για να μπορέσουμε να βρούμε ένα τόσο καλό παράδειγμα ώστε να ξεγελάσουμε τον άπληστο αλγόριθμο, πρέπει να βάλουμε ένα μέγιστο στην μέση και στην ίδια στήλη και γραμμή με αυτό το μέγιστο, τον αμέσως προηγούμενο αριθμό. Λόγω του περιορισμού, ο άπληστος αλγόριθμος δεν θα μπορέσει να χρησιμοποιήσει τα υπόλοιπα νούμερα στην ίδια γραμμή και στήλη με το μέγιστο(προηγούμενη και επόμενη σειρά) και έτσι ο δυναμικός αλγόριθμος θα εκμεταλευτεί αυτό το άθροισμα των αριθμών(χωρίς το μέγιστο στοιχείο του πίνακα), καταλήγοντας σε ένα ποσό σχεδόν διπλάσιο του μέγιστου ποσού που επέλεξε ο άπληστος αλγόριθμος. Τα υπόλοιπα στοιχεία του πίνακα μπορούμε να τα θέσουμε για ευκολία 0.

Τα περιττά n ευνοούν τον δυναμικό έναντι του άπληστου.

Μαρία Σχοινάκη: 3210191

0	9	0	0
0	10	9	0
0	9	0	0
0	0	9	0

GREEDY -> 19
DP -> 36

B(I)/A(I) = 36 / 19 = 1,9

Για περιττά n

Ο λόγος είναι απλός. Όταν ο άπληστος διαλέξει ένα μέγιστο στην μέση του πίνακα, δεν μπορεί πλέον να επιλέξει από την ίδια γραμμή ή στήλη(προηγούμενη ή επόμενη σειρά). Άρα γεμίζοντας τις απαγορευμένες θέσεις για το μέγιστο που διαλέγει ο άπληστος αλγόριθμος, με τον αμέσως μικρότερο του μεγίστου, πλέον έχουμε ένα άθροισμα σχεδόν τριπλάσιο από το μέγιστο του άπληστου.

Γενικά, η αναλογία όσο μεγαλώνουν τα νούμερα θα τείνει στο **3**
Δηλαδή ο **α = maxI B(I)/A(I) = 3**

0	9	0	0	0
0	10	9	0	0
0	9	0	0	0
0	10	9	0	0
0	9	0	0	0

GREEDY -> 20
DP -> 45

B(I)/A(I) = 45 / 20 =2,25

ΈΝΑ ΑΠΌ ΤΑ ΧΕΙΡΌΤΕΡΑ ΠΑΡΑΔΕΊΓΜΑΤΑ

9.999	0	0
10.000	9.999	0
9.999	0	0

B(I)/A(I) = 29.997 / 10000 = 2,9997

Μαρία Σχοινάκη: 3210191

ΟΡΘΟΤΗΤΑ DP

Σκοπός του αλγορίθμου μας είναι να βρίσκει το μέγιστο ποσό επιλέγοντας n κελιά, σεβόμενος του περιορισμού περί διαδοχικών σειρών. Πράγμα το οποίο επιτυγχάνεται λόγω της αναδρομικής σχέσης, όπως θα περιγράψουμε παρακάτω. Ας αναλύσουμε τις 2 βασικές πτυχές του αλγορίθμου

1) Ο αλγόριθμός μας, δημιουργεί τον πίνακα DP, ο οποίος είναι ένας πίνακας που περιέχει το μέγιστο άθροισμα κάθε θέσης, σεβόμενος τον περιορισμό. Η πρώτη γραμμή προφανώς είναι η ίδια με του πίνακα εισόδου, καθώς επιχειρούμε μια προσέγγιση στην οποία διαβάζοντας ανά γραμμή τον πίνακα εισόδου από πάνω προς τα κάτω, συμπληρώνουμε στον DP τις αλλαγές μετά την πρόσθεση των μέγιστων τιμών. Στην ουσία ξεκινάμε από την πρώτη γραμμή. Βρίσκουμε τα 2 μεγαλύτερα ποσά. Πηγαίνοντας στην δεύτερη γραμμή, για κάθε στοιχείο προσθέτουμε το μέγιστο της πρώτης γραμμής αν υπακούει τον περιορισμό, αλλιώς προσθέτουμε το δεύτερο μεγαλύτερο. Αποθηκεύουμε αυτήν την τιμή στον πίνακα DP. Ακολουθούμε αυτήν την διαδικασία για κάθε γραμμή του πίνακα. Έτσι, δημιουργούμε έναν πίνακα DP, ο οποίος λόγω της αναδρομικής σχέσης, κάθε στοιχείο του αποτελεί το στοιχείο του πίνακα εισόδου + το μέγιστο μέχρι τότε μονοπάτι γραμμής. Υπακούοντας τον περιορισμό, προσθέτουμε για κάθε στοιχείο του πίνακα εισόδου το μέγιστο ως τότε μονοπάτι γραμμής. Αυτή η διαδικασία είναι ορθή γιατί εξετάζει κάθε δυνατό συνδυασμό και καταλήγει στον βέλτιστο. Στην ουσία, η ίδια διαδικασία επαναλαμβάνεται για κάθε γραμμή του πίνακα(εκτός της πρώτης) και αφού η διαδικασία είναι ορθή για τις 2 πρώτες γραμμές σημαίνει ότι είναι και για όσες ακολουθούν. Άρα ο αλγόριθμος για την δημιουργία του πίνακα DP, είναι ορθός.

2) Για την ανάκτηση του μονοπατιού(κελιών) που ακολουθήθηκε προφανώς πρέπει να βρούμε την μέγιστη τιμή της τελευταίας γραμμής του DP, όπου είναι η μέγιστη αξία κελιών που υπακούνε τον περιορισμό του πίνακα. Κάνουμε μία διαδικασία ανάποδη. Ξεκινώντας από το μέγιστο ποσό, και έχοντας αποθηκευμένες τις θέσεις των μεγίστων κάθε γραμμής ξεκινάμε την διαδικασία. Ανά 2 γραμμές ακολουθείται η ίδια διαδικασία, οπότε αν αποδείξουμε ότι ισχύει για τις 2 τελευταίες γραμμές, τότε ισχύει για όλες. Έχοντας την θέση του μέγιστου στοιχείου της τελευταίας γραμμής του DP, ξεκινάμε την διαδικασία όπου ψάχνουμε την θέση του προηγούμενου στοιχείου(1^o ή 2^o μέγιστο). Έτσι, αν η θέση του μεγίστου της τελευταίας γραμμής, είναι ίδια με την θέση του μεγίστου της προηγούμενης γραμμής, τότε προφανώς δεν ισχύει ο περιορισμός, που σημαίνει ότι ακολουθήσαμε το 2^o μέγιστο της προηγούμενης γραμμής και όχι το πρώτο. Αφού ξεδιπλώνουμε την λύση σταδιακά και υπακούμε στον περιορισμό, η λύση μας είναι ορθή.