2ο ΣΕΤ ΑΣΚΗΣΕΩΝ
ΑΛΓΟΡΙΘΜΟΙ

ΤΖΕΝΗ ΜΠΟΛΕΝΑ 3170117

1 ΄Ασκηση 2.3

1.1 Αναδρομική εξίσωση άσκησης 2.1

Αν θεωρήσουμε f ην συνάρτηση που βρίσκει τον ελάχιστο αριθμό απο jumps που χρειάζονται για να πάω απο την αρχή μέχρι το τέλος και στην αρχή θεωρώ οτι το minimum number of jumps είναι -1 τότε η εξίσωση που δίνει την λύση είναι η εξής:

$$f(i) = \begin{cases} 0 & i = 0 \\ f(j) + 1 & 1 <= i < leaves, 0 <= j <= i-1, f(i) = -1 \\ \min\left(f(j)+1, f(i)\right) & \text{otherwise} \end{cases}$$

Απο το 0 φύλλο για να πάω στο 0 προφανώς και χρειάζομαι 0jumps. Απο το 0 για να πάω στο i χρείαζομαι όσα jumps θέλω για να πάω σε ενα φύλλο που με πάει στο i και ενα ακομα, αυτό την πρώτη φορά που βρίσκω τρόπο απο το 0 να πάω στο i, διαφορετικά η λύση θα είναι το ελάχιστο μεταξύ του νέου μονοπατιού που βρήκα και εκείνου που έχω τώρα σαν ελάχιστο. Απαραίτητη προϋπόθεση εϊναι η τροφή που έχω στο j φύλλο να επαρκεί για να πάω στο i, διαφορετικά συνέχισε και ψάξε τα υπόλοιπα φύλλα αν με πάνε στο i.

1.2 Πολυπλοκότητα άσκησης 2.1

Η πολυπλοκότητα είναι $\mathcal{O}(n^2)$
΄Εχουμε n το πλήθος των φύλλων.

Η πολυπλοκότητα είναι τόση επειδή:

$$\sum_{i=1}^{n-1}\sum_{j=0}^{i-1} 1 = \mathcal{O}(n^2)$$

2 Άσκηση 2.4

2.1 Αναδρομική εξίσωση άσκησης 2.2

Έχω την συνάσρτηση sol που μου βρίσκει την λύση, την συνάρτηση food που περιέχει τα λιπαρά και τις θερμίδες για κάθε φαγητό. Έχω επίσης μια βοηθητική μεταβλητή helpCalories η οποία δίνει πόσες θερμίδες πέρνω αν συμπεριλάβω και το συγκεκριμένω φαγητό στο μενού μου.Έχω και μια αλλη βοηθητική μεταβλητή helpFat η οποία δίνει πόσα λιπαρά πέρνω αν συμπεριλάβω και το συγκεκριμένω φαγητό στο μενού μου.

$$sol(i,j) = \begin{cases} sol(i-1,j) & j < food[i].calories \\ helpCalories, \min\left(solution[i-1][j].fat, helpFat\right) & helpCalories = solution[i-1 \\ helpCalories, helpFat & \text{otherwise} \end{cases}$$

Οπότε οι max calories που μπορώ να του δώσω μέσα στο επιτρεπτό όριο βρίσκονται στο sol[numberOfFoods -1][caloriesWanted]. Το πώς βρίσκω το συνδιασμό που μου δϊνει την καλύτερη λύση γίνεται εύκολα κατανοητό αν δείται τον κώδικα.

2.2 Πολυπλοκότητα άσκησης 2.2

Η πολυπλοκότητα είναι $\mathcal{O}(n * m)$ όπου n τό πλήθος των φαγητών και m το πλήθος των ζητούμενων calories. Επομένως έχουμε ψευδοπολυωνιμική πολυπλοκότητα(NP-δύσκολο)
Η πολυπλοκότητα είναι τόση επειδή:

$$\sum_{i=1}^{n}\sum_{j=0}^{m} 1 = \mathcal{O}(n * m)$$

Όπου n είναι τα φαγητά -1 και m είναι τα wanted calories.

3 Άσκηση 2.5

3.1 Λύση και Greedy Choice

Αρχικά πρέπει να ταξινομηθούν τα διαστήματα με βάση το σημείο που τελειώνουν(finishPoint). Αφού έγινε η ταξινόμιση είμαι πλέον έτοιμη να λύσω το πρόβλημα. Η άπληστη επιλογή είναι απο τα εναπομείναντα διαστήματα(όσα δεν έχουν καλυφθεί ήδη απο ενα σημείο) κάθε φορά να επιλέξω αυτο που τελειώνει πρώτο και να τοποθετήσω σημείο πάνω στην ευθεία σε εκείνη τη θέση όπου θα καλύπτει όσο το δυνατόν περισσότερα διαστήματα.

- Ταξινόμησε τα διαστήματα με βάση το finish point.
- Επέλεξε το διάστημα που τελειώνει πρώτο.
- Όσα διαστήματα έχουν start time πριν ή μεχρι το finish time του επιλεγμένου διαστήματος θα καλυφθούν όλα από ένα σημείο το οποίο θα τοποθετηθεί όπου τελειώνει το πρώτο διάστημα από όσα καλύπτονται απο το σημείο αυτό.
- Επέλεξε το διάστημα απο οσα δεν καλύπτονται ήδη και επανάλαβε την ίδια διαδικασία μέχρι να καλυφθούν όλα τα διαστήματα.

Algorithm 1: Βήματα

3.2 Αλγόριθμος

Έχω γράψει κώδικα σε java και βρίσκεται στα παραδοτέα στον φάκελο Greedy1 στο src. Παρέλειψα την ταξινόμιση και έβαλα τα data ήδη ταξινομημένα ως προς το finish time, άλλωστε αλγόριθμο ταξινόμισης είχε το πρώτο σετ οπότε δεν υπήρχε λόγος να ξαναγράψω.

3.3 Πολυπλοκότητα

Πολυπλοκότητα: $\mathcal{O}(nlogn)$
Η πολυπλοκότητα είναι $\mathcal{O}(nlogn)$ επειδή για να ταξινομήσω τα διαστήματα

θέλω $\mathcal{O}(nlogn)$(MergeSort), για να βρώ την λύση αρκούν n βήματα οπότε πέρνω $\max(\mathcal{O}(nlogn), \mathcal{O}(n))$.

3.4 Απόδειξη ορθότητας

Έστω ότι η λύση μου δεν είναι βέλτιστη, τότε θα μπορούσα να αφαιρέσω ένα σημείο και να καλύψω όλα τα διαστήματα, **άτοπο** αφού κάθε φορά επιλέγω σημείο που καλύπτει όσο το δυνατόν περισσότερα διαστήματα και μετά αυτά τα διαστήματα(με την υλοποίηση που έχω κάνει) δεν καλύπτονται απο κάποιο άλλο σημείο. Επομένως η λύση αυτή είναι βέλτιστη.

4 Άσκηση 2.6

4.1 Λύση και Greedy Choice

Ξεκινώντας πρέπει να ταξινομίσω τα σημεία σε αύξουσα σειρά. Στη συνέχεια ξεκινώντας από το πρώτο ακάλυπτο σημείο επιλέγω εκείνο το διάστημα το οποίο καλύπτει το σημείο αυτό και επεκτείνεται πρός τα δεξιά περισσότερο απο τα άλλα υποψήφια διαστήματα(δλδ πέρνω το μεγαλύτερο fi - σημείο). Επανλάμβάνω την διαδικασία για το αμέσως επόμενο ακάλυπτο σημείο μέχρι να καλύψω όλα τα σημεία. Οπότε η greedy επιλογή είναι για κάθε ακάλυπτο σημείο να επιλέξω εκείνο που θα καλύπτει αυτό και όσο το δυνατόν περισσότερα σημεία προς τα δεξιά.

4.2 Αλγόριθμος

Algorithm `algo()`
  while *!allPointsCovered* **do**
    `proc(`*findFirstPointNotCovered()*`)`
  end

Procedure `proc(`*pointToCover*`)`
  bestChoice = space[0]
  for $i = 1;\ i < spaces.length;\ i++$ **do**
    if *sapce[i].betterThan(bestChoice)* **then**
      bestChoice = space[i]
  end
    ▷ pointToCover is now covered probably along with some other points
  solution.add(bestChoice)

Algorithm 2: Βρές ελάχιστο πλήθος διαστημάτων που καλύπτου όλα τα σημεία πάνω στην ευθεία

4.3 Πολυπλοκότητα

Πολυπλοκότητα: $\max(\mathcal{O}(mlogm), \mathcal{O}(n*m))$
Η πολυπλοκότητα είναι $\max(\mathcal{O}(mlogm), \mathcal{O}(n*m))$ επειδή για να ταξινομήσω τα σημεία θέλω $\mathcal{O}(mlogm)$(MergeSort) και για να βρώ τα διαστήματα που θα τοποθετήσω θέλω $\mathcal{O}(n^2)$ οπότε πέρνω $\max(\mathcal{O}(mlogm), \mathcal{O}(n*m))$. Η χειρότερη περίπτωση για να καλύψω όλα τα σημεία είναι κάθε διάστημα να καλύπτει ένα μόνο σημείο οπότε θα πρέπει για m σημεία στην ευθεία να ψάξω ποιό από τα n διαστήματα το καλύπτει.

4.4 Απόδειξη ορθότητας

Έστω ότι η λύση μου δεν είναι βέλτιστη, τότε θα μπορούσα να αφαιρέσω δύο διαστήματα και να τα αντικαταστήσω με ενα άλλο, αυτό όμως δεν γίνεται αφού τα διαστήματα που εχω επιλέξει καλύπτουν όσο το δυνατόν περισσότερα σημεία, οπότε η λύση είναι βέλτιστη. Αν είχα ένα διάστημα που αντικατηστούσε τα δύο υπάρχοντα θα το είχα ήδη βρεί και θα αποτελούσε λύση του προβλήμματος.