Algorithms
Programming exercise 2

Κωνσταντίνος Νικολούτσος
p3170122

1 Ασκηση (Frog-problem)

1.1 Αναδρομική εξίσωση

Η αναδρομική εξίσωση προκύπτει εύκολα εφόσον εχουμε κάνει τον αλγόριθμο με bottom-up approach (Θα το γράψουμε με μαθηματική μορφη).

$F_i = 1 + j$, τέτοιο ώστε, $j = \min_{j<i}(F_j) * bool$

Οπου, το bool ειναι είτε 1 είτε 0 και εξαρτάται απο το αν στο νούφαρο που εξετάζουμε ο βάτραχος μπορει να πάει στο νουφαρο που βρισκόμαστε.(Ειναι 1 οταν γινεται να παει)(Δηλαδη στην αναδρομή το επόμενο βγαινει φιλτραροντας τα προηγουμε αναλογα με το αν μπορει ο βατραχος να πηδηξει σε αυτα και μετα παιρνουμε το ελαχιστο απο αυτα και προσθετουμε 1)

1.2 Πολυπλοκότητα χρόνου

Ειναι προφανές οτι η πολυπλοκότητα χρόνου ειναι $O(n^2)$ Αφου για καθε leaf που βρισκεται στην θεση position κανουμε (αθροισμα απο ενα εως position βηματα). Η απόδειξη ειναι εύκολη αφου τα βήματα που κανει ο αλγόριθμος μας αμα τα μετρήσουμε ειναι:

$$\sum_{n=0}^{N} n := O(n^2)$$

Που αυτο γνωριζουμε οτι ειναι πολυπλοκότητας $O(n^2)$

1.3 Μια μικρή αναβάθμιση στον αλγόριθμο

Θα μπορούσαμε για καθε φύλλο αντι να ξεκινάγαμε την διαδικασία απο την αρχή, να πηγαιναμε απλα πισω οσες θεσεις ειναι η μεγιστη τροφη στο προβλημα μας. Nevertheless, αυτο θα μας εδινε καλυτερο χρονο αλλα θα ειχαμε ιδια πολυπλοκοτητα μεγαλου ομικρον(για μεγαλη εισοδο)

2 Ασκηση (Dietologist-problem)

2.1 Αναδρομική εξίσωση

$$Fi,j = \begin{cases} 0 & i=0, j=0 \\ max(Fi-1,j, cal(i-1)+Fi-1,j-weight(i-1)) & can\_be\_included \\ Fi-1,j & cannot\_be\_included \end{cases}$$

Η παραπάνω αναδρομική εξίσωση ειναι απλοποιημένη(Αν θελετε περισσοτερη λεπτομερεια δειτε τον κωδικα). Με λόγια αυτο σημαίνει οτι φτίαχνουμε

εναν πινακα [C+1][T+1] και τον γεμίζουμε κάνωντας κατα γραμμή κάνωντας τα εξείς. Ας γεμίσουμε την γραμμή i για να το καταλαβουμε. Αν το αντικειμενο i δεν χωραει να μπει(οι θερμιδες δεν επαρκουν) τοτε παιρνουμε την πανω τιμη που βρισκοταν στον δισδιαστατο πινακα. Αν ομως χωραει τοτε παιρνουμε το max μεταξυ του απο πανω και του να συμπεριλαμβαναμε το αντικειμενο μαζι με οτι αλο μπορουσαμε(πανω μειωμενο κατα θερμιδες αυτου που βαλαμε). Αυτα τα προβλήματα δεν ειναι και τοσο ευκολο να τα εξηγείς γραφοντας ενα κειμενο, συνήθως χρειάζονται ενα animation!

2.2 Πολυπλοκότητα χρόνου

Η πολυπλοκότητα χρόνου ειναι της ταξης $O(C * N)$ οπου C,N το πληθος των θερμιδων για το οποιο ζητειτε να βρουμε μενου και το πληθος των αντικειμενων που εχουμε στην διαθεση μας αντιστοιχα!

3 Ασκηση greedy 1

3.1 Λύση

- Ταξινομησε τα διαστηματα αναλογα με το finish point σε αυξουσα σειρα(mergeSort ή HeapSort)
- Παρε το διαστημα που τελειωνει πρωτο
- Βαλε ενα σημειο εκει που τελειώνει.
- Βγαλε απο το παιχνίδι τα διαστηματα που εχουν καλυφθει.(Αποθηκευσε τα σε ουρα)
- goto bullet 2 μεχρι οτου δεν υπαρχει ακαλυπτα intervals

  (*Αν για καποιο σημειο δεν υπαρχει διάστημα το οποιο το περιέχει τοτε ο αλγόριθμος μας θα επιστρέψει οτι δεν γίνεται να καλυφθουν ολα τα σημεία και θα γυρίσει την τρέχων λυση. Οπως μου απαντήσατε στο forum του eclass)

3.2 Πολυπλοκότητα χρόνου

Η πολυπλοκότητα χρόνου ειναι της ταξης $O(n * log(n))$ (χρησημοποιώντας mergesort) . Τα βηματα για να λυθει αφου εχει γινει ταξινομιση ειναι n. Αρα το πιο αργο σημειο του αλγορθμου εινια η ταξινομιση

3.3 Απόδειξη ορθότητας

Θα χρησιμοποιησουμε το επιχειρημα της ανταλλαγης.
Εστω οτι υπήρχε μια αλλη Optimum λυση και μεχρι καποιο σημειο ειχαμε επιλέξει τα ιδια σημεία. Εστω τωρα οτι σε ενα σημειο ειχαμε επιλεξει διαφορετικα, τοτε αν αντικαθιστουσε η βελτιστη λυση το σημειο το δικο μου θα εβγαζε καλυτερη λυση (αφου θα καλυπτει περισσοτερα σημειο) , το οποιο ειναι ατοπο αρα η λυση μου εινια ιδια με το Optimum.

4 Ασκηση greedy bonus

4.1 Λύση

Πριν ξεκινήσουμε να πουμε την λύση ας ορίσουμε μια μέθοδο Interval findBestInterval(Point p, Interval[] intervals) η οποια μας δινει για ενα συγκεκριμενο σημείο το interval που πρεπει να παρουμε ωστε να βγαλουμε σιγουρα optimum λυση.
Πως γινεται η επιλογή;; Η μεθοδο επιλέγη το interval το οποιο καλυπτει πρωτα το σημειο και επισης καλυπτει οσο το δυνατο δεξια απο το σημειο.
Οποτε αφου ξεκαθαρισαμε με αυτο ας πουμε τα βηματα που πρεπει να ακολουθησουμε ωστε να λυθει το προβλημα:

- Κανε ταξινόμιση στα σημεία σε αυξουσα σειρα (mergeSort ή HeapSort)
- Επιλεξε το πρωτο σημειο(το οποιο δεν εχει καλυφθει) και κανε την μεθοδο findBestInterval
- Διεγραψε το interval που χρησιμοποιηθηκε καθως και τα σημεια τα οποια καλυπτει
- Επαναλαβε, επαλαβε μεχρι οτου δεν υπαρχουν αλλα σημεια

4.2 Πολυπλοκότητα χρόνου

Η πολυπλοκότητα του συγκευκριμένου αλγορίθμου ειναι ανηκει στην πολυωνυμικη κλάση. Πιο συγκεκριμένα, η πολυπλοκότητα ειναι ιση με $O(n * M)$) οπου n και M ο αριθμος των σημειων και των διαστημάτων αντίστοιχα. Αυτο συμβαινει διότι για να λυθει αυτο θα πρεπει να υλοποιήσουμε 2 for loops τα οποια θα βρισκεται το ενα μεσα στο αλλο.

4.3 Απόδειξη ορθότητας

Εστω οτι ειχαμε την βελτιστη λυση τοτε αν διαφοροποιουταν σε ενα σημειο απο το δικο μας θα μπορουσαμε να την ανταλλασαμε και η βελτιστη να ηταν ακομα βελτιστη. Αρα αυτο σημαινει οτι εχουμε την βελτιστη λυση. Επισης εινια πολυ λογικο να εχουμε την βελτιστη λυση αφου ειμαστε biased προς τα διαστηματα που εχουν την ταση να πηγαινουν περισσοτερο δεξια στον χωρο(δηλαδη καλυπτουν οσο πιο καλα γινεται τον χωρο)