***PROJECT 1***

Στην συγκεκριμένη άσκηση μας ζητήθηκε να δημιουργήσουμε 2 πίνακες ακέραιων αριθμών και να λύσουμε το πρόβλημα μεγίστου αθροίσματος υπακολουθίας. Στην πληροφορική το πρόβλημα μέγιστου αθροίσματος υποακολουθίας (Maximum subarray problem) είναι ένα πρόβλημα όπου έχουμε μια σειρά (πίνακα) από θετικούς και αρνητικούς αριθμούς και ψάχνουμε την συνεχή μεγαλύτερη υποσειρά (υποπίνακα) αριθμών όπου έχουμε το μεγαλύτερο άθροισμα των στοιχείων. Για την αντιμετώπιση του συγκεκριμένου προβλήματος χρησιμοποιήθηκαν 4 αλγόριθμοι διαφορετικής σύγκλισης ο καθένας.

Α) ΑΛΓΟΡΙΣΘΜΟΣ ΜΕ ΤΑΞΗ ΣΥΓΚΛΙΣΗΣ Ο(n^3): Αποτελείται από 3 loops το ένα μέσα στο άλλο και βασίζεται στην ιδέα της εύρεσης όλων των πιθανών υποπίνακων και όλων τον πιθανών αθροισμάτων. Όπως αποδείχτηκε είναι ο πιο αργός αλγόριθμος.

Β) ΑΛΓΟΡΙΣΘΜΟΣ ΜΕ ΤΑΞΗ ΣΥΓΚΛΙΣΗΣ Ο(n^2): Αποτελείται από 2 loops το ένα μέσα στο άλλο και το τρίτο το αποφεύγουμε γιατί για τον υπολογισμό του αθροίσματος από Α(i)…A(j+1) προσθέτει το A(j+1) στο άθροισμα Α(i)…A(j). Με αυτό τον τρόπο ξαναχρησιμοποιεί δεδομένα που έχει βρει.

Γ) ΑΛΓΟΡΙΣΘΜΟΣ ΜΕ ΤΑΞΗ ΣΥΓΚΛΙΣΗΣ Ο(nlogn): Αυτός ο αλγόριθμος (Διαίρει και Βασίλευε) βασίζεται στην ιδέα ότι για να λύσω ένα πρόβλημα το χωρίζω σε υποπροβλήματα. Στο συγκεκριμένο πρόβλημα αρχικά διαιρώ τον πίνακα σε δύο υποπίνακες. Τώρα ο υποπίνακας με το μεγαλύτερο άθροισμα μπορεί να βρίσκεται είτε μόνο στον αριστερό υποπίνακα, είτε μόνο στον δεξιό υποπίνακα, είτε να περιλαμβάνει στοιχεία και των δύο.

Δ) ΑΛΓΟΡΙΣΘΜΟΣ ΜΕ ΤΑΞΗ ΣΥΓΚΛΙΣΗΣ Ο(n): Αυτός ο αλγόριθμος έχει παρόμοια λογική με τον προηγούμενο, δηλαδή διαιρεί το πρόβλημα σε υποπρόβληματα. Σύμφωνα με αυτόν σαρώνουμε τις τιμές του πίνακα, και κάθε φορά υπολογίζουμε την θέση με το μέγιστο θετικό άθροισμα της υποσειράς που τερματίζεται στο συγκεκριμένο σημείο. Έχει ένα loop που εκτελεί n επαναλήψεις.

Από τα προγράμματα προέκυψαν τα εξής αποτελέσματα:

ΔΕΔΟΜΕΝΑ 1: 500 ΣΤΟΙΧΕΙΑ

ΔΕΔΟΜΕΝΑ 2: 1000 ΣΤΟΙΧΕΙΑ

|  |  |  |
| --- | --- | --- |
| ΠΟΛΥΠΛΟΚΟΤΗΤΑ | ΔΕΔΟΜΕΝΑ1 ΧΡΟΝΟΣ | ΔΕΔΟΜΕΝΑ2 ΧΡΟΝΟΣ |
| *n^3* | 4.758030499999999 | 40.9814627 |
| *n^2(χωρίς πίνακα)* | 0.03120020000000001 | 0.12480080000000002 |
| *n^2(με πίνακα)* | 0.7176046 | 5.2260335 |
| *nlogn* | 0.0 | 0.015600100000000006 |
| *n* | 0.0 | 0.0 |

***PROJECT 3***

Στην συγκεκριμένη άσκηση μας ζητήθηκε να φτιάξουμε πιστωτικές κάρτες που θα έχουν δεκαέξι (16) συγκεκριμένα σταθερά ψηφία πχ 1234567890123456 αλλά σε τέσσερις (4) από τις δεκαέξι (16) τυχαίες θέσεις έχουν τους χαρακτήρες: Α, B, C, D. (π.χ. 12D45A789012B4C6). Αυτές οι κάρτες θα κάνουν 1000000 επισκέψεις σε πολυκατάστημα που λειτουργεί από Δευτέρα ως Σάββατο για να κάνουν τις αγορές τους. Τα ποσά των πληρωμών κάθε επίσκεψης είναι μεταξύ 10 ευρώ και 100 ευρώ. Για την αντιμετώπιση της συγκεκριμένης άσκησης χρησιμοποιήθηκε η διαδικασία HASH. Η διαδικασία αυτή αντιστοιχίζει τα δεδομένα ενός κλειδιού σε συγκεκριμένη θέση ανάλογα το κλειδί έτσι ώστε να είναι συγκεντρωμένα και ευκόλως προσβάσιμα. Στην άσκηση μας τα κλειδιά αποτελούν οι κάρτες και τα δεδομένα οι μέρες και τα έξοδα. Έτσι σκοπός της άσκησης είναι κατά την δημιουργία τυχαίων δεδομένων για κάθε κάρτα αυτά να προστίθενται στο HASH TABLE έτσι ώστε μετά το πέρας 1000000 επισκέψεων να μπορούμε εύκολα να βρούμε την κάρτα με τις η κάρτα με το μεγαλύτερο συνολικό ποσό πληρωμών, την κάρτα με το μεγαλύτερο πλήθος επισκέψεων, την ημέρα με την μεγαλύτερο πλήθος επισκέψεων, το πλήθος των συγκρούσεων στον τελικό πίνακα, για 3 διαφορετικές τιμές του load factor. Βασικό πρόβλημα γενικότερα στη συγκεκριμένη διαδικασία αποτελούν οι συγκρούσεις. Σύγκρουση έχουμε όταν για διαφορετικά κλειδιά γίνεται αντιστοιχία στο ίδιο κελί του πίνακα. Υπάρχουν αρκετοί τρόποι για να αντιμετωπίσεις το πρόβλημα αυτό. Αφού η άσκηση μας λέει ότι ο HASH TABLE δεν θα γεμίσει ποτέ, χρησιμοποίησα την μέθοδο γραμμικής αναζήτησης. Με αυτή την μέθοδο, αν υπάρξει σύγκρουση σε μια θέση του πίνακα ελέγχουμε την επόμενη θέση μέχρι να μην υπάρχει σύγκρουση. Η ανάλυση του κώδικα της άσκησης βρίσκεται στην αντίστοιχη αναφορά.