Εργασία 2 Αλγορίθμων

ΓΕΩΡΓΙΟΥ ΑΛΕΞΙΟΣ – ΛΑΖΑΡΟΣ

Ρ3180027

ΙΟΥΝΙΟΣ 2020

2.1)

α) Διαίρει-και-βασίλευε αλγόριθμος:

1) Ελέγχω άμα η τωρινή διαμέριση του πίνακα μπορεί να χωρέσει σε έναν κάδο (στο πρώτο βήμα για δηλαδή άμα όλος ο πίνακας χωράει σε έναν κάδο). Άμα χωράει επιστρέφω τα στοιχεία της έγκυρης τοποθέτησης.

2) Άμα δεν χωράει σε έναν κάδο τότε "σπάω" στην μέση τον πίνακα και ελέγχω για το αριστερό και το δεξί μέρος το πρώτο βήμα. Οι διαμερίσεις σε κάθε επίπεδο σταματούν όταν τα συγκεκριμένα στοιχεία της διαμέρισης χωράνε σε έναν κάδο. Τότε τα επιστρέφω ως υποσύνολο της λύσης και προσθέτω έναν κάδο στην λύση μου.

β) Η αναδρομική κλήση θα σταματάει όταν η υποδιαίρεση των στοιχείων χωράνε όλες σε έναν κάδο, δηλαδή όταν $\sum x_i \leq 5$ όπου x_iτο κάθε στοιχείο της υποδιαίρεσης.

γ) Το αποτέλεσμα του αλγόριθμου είναι ορθό και είναι προφανές. Ο αλγόριθμος επιλέγει υποσύνολα του συνόλου αντικειμένου τα οποία χωράνε σε έναν κάδο και μόνο τότε τα χρησιμοποιεί στην λύση, οπότε εξ 'ορισμού θα έχουμε μια έγκυρη λύση.

δ) Στην καλύτερη περίπτωση θα έχουμε 1 επίπεδο (το μηδενικό) αφού όλα τα στοιχεία του συνόλου θα χωράνε σε έναν κάδο οπότε δεν θα χρειαστεί να πάρουμε διαμερίσεις του συνόλου. Στην χειρότερη περίπτωση θα έχουμε n κάδους δηλαδή κάθε αντικείμενο θα μπαίνει σε έναν κάδο που σημαίνει ότι το δέντρο θα έχει $\lfloor logn \rfloor + 1$ επίπεδα αφού η διαμέριση γίνετε στο μισό και το ύψος του δυαδικού δέντρου είναι $logn$.

ε) Πολυπλοκότητα χειρότερης περίπτωσης: Η αναδρομική συνάρτηση πολυπλοκότητας είναι $T(n) = 2T\left(\frac{n}{2}\right) + n$.

Με Master theorem έχουμε $\mathrm{a} = 2, \mathrm{b} = 2, \mathrm{f(n)} = n$

$n^{\log_a b} = n^{\log_2 2} = n^1$, άρα $f(n) = \Theta(n^{\log_a b})$ και είμαστε στην δεύτερη περίπτωση του MT άρα $\boldsymbol{T(n) = \Theta(nlogn)}$

Υπολογισμός με Ανάπτυγμα δέντρου: Ο αλγόριθμος στην χειρότερη περίπτωση θα διαιρεί τα αντικείμενα και θα πάρει 1 κουβά για κάθε αντικείμενο. Σε κάθε βήμα θα πρέπει να ελέγχει το

άθροισμα των αντικειμένων του. Στο πρώτο βήμα θα πρέπει να προσθέσουμε n αντικείμενα, στο δεύτερο 2 $(\frac{n}{2})$ στο τρίτο $4(\frac{n}{4})$ κτλ. άρα θα έχουμε $logn$ φορές το n, δηλαδη $O(nlogn)$

στ) Έστω το παράδειγμα της εκφώνησης με μεγέθη {1, 2, 2, 5, 3, 1, 4, 1} και χωρητικότητα C = 5.

Ο αλγόριθμος θα δώσει ως αποτέλεσμα: [(1,2),(2),(5),(3,1),(4,1)]

2.2)

α) Ο άπληστος αλγόριθμος θα επιλέγει τα αντικείμενα με την σειρά (πρώτα το πρώτο, μετά το δεύτερο κτλ.) και θα προσθέτει στο κάδο το αντικείμενο αν χωράει ή το μέγιστο κλασματικό μέρος του αντικειμένου που μπορεί να χωρέσει(το οποίο θα το αφαιρέσω και θα επαναλάβω την διαδικασία για το υπόλοιπο του στον επόμενο κάδο). Όταν γεμίσει ο κάδος θα γεμίζουμε τον επόμενο κάδο.

β) Είναι προφανές ότι με αυτήν την τεχνική του άπληστου βήματος η λύση του αλγορίθμου θα είναι βέλτιστη, αφού γεμίζω τους κουβάδες με την σειρά και δεν αφήνω κανέναν μισογεμάτο παρά μόνο άμα είναι ο τελευταίος. Δεν υπάρχει δηλαδή μαθηματικά πιο βέλτιστος τρόπος να πάρω λιγότερους κουβάδες από ένα σύνολο αντικειμένων με συνολική χωρητικότητα x πάνω από $\frac{x}{C} + 1$ κουβάδες, όπου C η χωρητικότητα του κάθε κουβά. Δηλαδή τα αντικείμενα θα χωρέσουνε όλα σε $\frac{x}{C}$ κουβάδες και ο επόμενος θα έχει μέσα αντικείμενα χωρητικότητας $x\ mod\ C$ όπου μπορεί να είναι και 0 δηλαδή να μην χρειάζεται ο τελευταίος.

Επαγωγικά: Εφόσον σε κάθε βήμα γεμίζω ένα κουβά εξ ‘ολοκλήρου τότε το βήμα μου είναι βέλτιστο και αφού κάνω ένα βέλτιστο βήμα κάθε φορά καταλήγω σε βέλτιστο αλγόριθμο (αρχή βελτιστότητας).

γ) $O(n)$ το μόνο που χρειάζεται είναι να διατρέξουμε όλα τα αντικείμενα και να τα προσθέσουμε στον εκάστοτε κάδο.

δ) $n = 4, s_i = \{6,3,4,1\}, C = 5$

Επιλέγω το 6, δεν χωράει στον πρώτο κάδο άρα θα βάλω το μέγιστο κλασματικό μέρος του 6 που μπορεί να χωρέσει στο 5. Αρά δηλαδή θα βάλω τα 5 από τα 6 και θα γεμίσω τον πρώτο κάδο και θα έχω να μου περισσεύει 1 από το πρώτο αντικείμενο.

Προχωράω στον δεύτερο κάδο, παίρνω το 1 από το πρώτο αντικείμενο και το βάζω στον κάδο, παίρνω όλο το 3 από το δεύτερο αντικείμενο και το προσθέτω στον κάδο και τέλος παίρνω το 1 από τα 4 στο τρίτο αντικείμενο και γεμίζω τον δεύτερο κάδο. (Περισσεύουν 3 από τα 4 στο τρίτο αντικείμενο)

Προχωράω στον τρίτο κάδο, βάζω 3 από τα 4 που περίσσεψαν στον τρίτο κάδο και προσθέτω και το τελευταίο 1 στον κάδο. Τέλος επιστρέφω τι έχει ο κάθε κάδος μέσα (την έγκυρη τοποθέτηση).

2.3)

α) Ο άπληστος αλγόριθμος της προηγούμενης άσκησης δεν μπορεί να εφαρμοστεί στο συγκεκριμένο πρόβλημα αυτούσιος αφού είναι φτιαγμένος για διαιρετά αντικείμενα. Τον προσαρμόζουμε έτσι ώστε να διαλέγει με τη σειρά τα αντικείμενα και όταν δεν μπορεί να προσθέσει το αντικείμενο στον κουβά να προχωράει στον επόμενο κουβά.

$$n = 4, s = \{1,2,3,4\}, C = 5$$

Ο άπληστος αλγόριθμος θα διάλεγε, το πρώτο και το δεύτερο και θα το έβαζε στον κουβά, μετά το τρίτο δεν χωράει οπότε θα πάρουμε νέο κουβά και το τέταρτο δεν χωράει στον δεύτερο κουβά και θα πάρουμε και τρίτο κουβά για να το βάλουμε, άρα 3 κουβάδες ενώ η βέλτιστη λύση είναι 2 κουβάδες.

β) Για κάθε κουβά πρέπει να βρούμε τα βέλτιστα αντικείμενα από το σύνολο ώστε να τα βάλουμε μέσα. Δηλαδή άμα λύσουμε το πρόβλημα Knapsack 0-1 για C χωρητικότητα και στην συνέχεια να το ξαναλύσουμε για τον επόμενο κουβά με το σύνολο αντικειμένων πλην των αντικειμένων που έχουμε επιλέξει, τότε θα επιλέγουμε σε κάθε βήμα τα βέλτιστα αντικείμενα οπότε στο τέλος θα έχουμε την βέλτιστη λύση (αρχή βελτιστότητας).

Για το κάθε Knapsack 0-1 για σάκο ίσο με την χωρητικότητα του κουβά θα εφαρμόσουμε την αναδρομική σχέση του προβλήματος σε μορφή δυναμικού προγραμματισμού.

Εάν $s_i \leq j$ // Εάν μπορώ να το πάρω να δω άμα συμφέρει να το πάρω

$$A[i,j] = max\{A[i-1,j], A[i-1,j-s_i] + s_i\}$$

Αλλιώς $A[i,j] = A[i-1,j]$

Δηλαδή κάθε λύση του υποπροβλήματος $A[i,j]$ θα είναι η το μέγιστο από τη λύση του προβλήματος των προηγούμενων αντικειμένων και από την λύση των προηγούμενων αντικειμένων για χωρητικότητα $j - s_i$ συν την χωρητικότητα του αντικειμένου.

Μόλις έχουμε την λύση του εκάστοτε υποπροβλήματος knapsack 0-1 αφαιρούμε τα αντικείμενα που διαλέξαμε από το σύνολο και προσθέτουμε έναν κουβά στη λύση

γ) Η κάθε λύση των υποπροβλημάτων θα είναι ένας πίνακας δυο διαστάσεων για κάθε κουβά. Η λύση του κάθε υποπροβλήματος knapsack θα βρίσκετε στην θέση $A[n,C]$ και το τελικό αποτέλεσμα πλήθος κουβάδων είναι το άθροισμα των πινάκων που έχουμε χρησιμοποιήσει.

δ) Προφανώς ο αλγόριθμος είναι ορθός αφού εφαρμόζουμε knapsack 0-1 για κάθε πιθανό κουβά που χρειαζόμαστε και διαλέγουμε τα βέλτιστα αντικείμενα σε κάθε βήμα. Οπότε στο τέλος θα έχουμε τοποθετήσει βέλτιστα τα αντικείμενα μας με όσο τον δυνατόν λιγότερο κουβάδες γίνεται.

ε) Χειρότερη περίπτωση να κάνουμε ένα πίνακα knapsack για όλα τα αντικείμενα, δηλαδή θα έχουμε το πολύ n πινακες στο τελος με μέγιστες διαστάσεις n, C. Οι πράξεις υπολογισμού κάθε κελιού σε πίνακα γίνονται σε $\Theta(1)$ αρα η πολυπλοκοτητα θα είναι $O(n * n * C) = \boldsymbol{O(n^2 C)}$

στ)

Στιγμιότυπο: $n = 4, s = \{1,2,3,4\}, C = 5$

Ξεκινάμε με το πρώτο knapsack το οποίο θα δώσει ως βέλτιστη λύση το (1,4) για το πρώτο κουβά και θα κάνει άλλο ένα knapsack για τα υπόλοιπα δυο αντικείμενα τα οποία θα τα πάρει ως βέλτιστη λύση του δεύτερου knapsack (2,3). Άρα θα έχουμε ως λύση 2 κουβάδες με έγκυρη και βέλτιστη τοποθέτηση [(1,4),(2,3)].

2.4)

α) ΝΑΙ, το δέντρο Τ είναι δέντρο επικάλυψης ελαχίστου κόστους του G.

Εάν το δέντρο επικάλυψης δεν ήταν ελαχίστου κόστους τότε για τουλάχιστον μια ακμή του, θα υπήρχε κάποια άλλη που με μεγαλύτερο κόστος που θα συνέδεε τους κόμβους της. Όμως αυτή η ακμή θα μεγάλωνε το κόστος από έναν τουλάχιστον κόμβου της στο s. Οπότε το δέντρο δεν θα ήταν ελαχίστου κόστους, δηλαδή ΑΤΟΠΟ.

β) Ο αλγόριθμος Bellman Ford είναι σχεδιασμένος να δίνει το συντομότερο μονοπάτι μεταξύ δυο κόμβων για Γράφο με θετικά και αρνητικά βάρη με πολυπλοκότητα $O(|V| \cdot |E|)$

Θα τρέξουμε τον επαναληπτικό βρόχο του αλγόριθμου Bellman k φορές και η πολυπλοκότητα του θα είναι $O(k|E|)$.

Αν ο βρόχος τρέξει k φορές τότε θα έχουμε το συντομότερο μονοπάτι μεταξύ δυο κόμβων $s,\ t$ με βάση την λειτουργία του και το γεγονός ότι ο Γράφος έχει την ιδιότητα ότι για οποιουσδήποτε δυο κόμβους του (όπως το s, t) το συντομότερο μονοπάτι δεν θα είναι πάνω από k ακμές.

Με κάθε βρόχο i που τρεχω στον Bellman-Ford εξασφαλίζω ότι γνωρίζω το συντομότερο μονοπάτι μεταξύ δυο οποιοδήποτε κόμβων αρκεί να απέχουν το το πολύ i ακμές. Άρα αφού ξέρω ότι δεν απέχουν από το πολύ k οι δυο κόμβοι τότε δεν χρειάζεται να γνωρίζω συντομότερες αποστάσεις για παραπάνω από k ακμές απόσταση κόμβους. Άρα ουσιαστικά εκμεταλλευόμενοι αυτήν την ιδιότητα του γράφου, μειώνουμε την πολυπλοκότητα του προβλήματος.

2.5)

α) Για το πολύ b πλήθος κάδων μπορούν να τοποθετηθούν τα αντικείμενα του S σε αυτούς;

β) Πρέπει να βρούμε το ελάχιστο πλήθος κάδων και μπορούμε να “ρωτήσουμε” το αντίστοιχο decision problem για ένα ορισμένο b. Άρα θα μπορούσαμε να ρωτάμε από το 0 μέχρι κάποιο μεγάλο b μέχρι να βρούμε το πρώτο true, τότε αυτό θα είναι το ελάχιστο. Βέβαια μπορούμε να το λύσουμε και ανάλογα με δυαδική αναζήτηση πιο γρήγορα, αφού ο θεωρητικός πίνακας έχει την ιδιότητα ότι πρώτα θα έχει όλα τα False αν υπάρχουν και στην συνέχεια θα είναι μόνο True, οπότε στην ουσία ψάχνουμε την τιμή για την οποία το decision problem απαντάει True και για την προηγούμενη της False.

γ) Το D είναι NP-complete, αφού μπορώ να επαληθεύσω την λύση πολυωνυμικά. Αν έχω το σύνολο αντικειμένων των κουβάδων και την τοποθέτηση τους στους κουβάδες τότε μπορώ να επαληθεύσω πολυωνυμικά συγκρίνοντας τα δυο αθροίσματα τους αν είναι ίσα.

Partition <p Bin packing

Θα θέσουμε την συνάρτηση αναγωγής.

S = S’

b = 2 (δυο κουβάδες αφού έχουμε δυο υποσύνολα στο Partition)

C = $\frac{\sum a_i}{2}$ το ημιάθροισμα των στοιχείων του partition S’

Από τον μετασχηματισμό το partition θα έχει λύση μόνο αν το πρόβλημα bin packing έχει λύση για την μετασχηματισμένη είσοδο.

δ) Το πρόβλημα Π είναι γενίκευση του knapsack 0-1 αφού πρέπει να βάλουμε σε κάθε σάκο-κάδο τα βέλτιστα αντικείμενα κάθε φορά και μετά να ξανατρέχουμε τον αλγόριθμο για το υπόλοιπα αντικείμενα.