ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ
ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Αλγόριθμοι και Πολυπλοκότητα

3η Σειρά Γραπτών Ασκήσεων

Δημήτριος Γεωργούσης, 03119005

Άσκηση 1: Υπολογισιμότητα

Αν **α** διάνυσμα τότε ως |**α**| αναπαριστούμε τη διάσταση του διανύσματος. Για παράδειγμα:

|(a, b, c)| = 3.

Αναπαριστούμε την διοφαντική εξίσωση ως D(**x**, **y**), όπου **x** είναι το διάνυσμα συντελεστών και **y** το διάνυσμα μεταβλητών της εξίσωσης. Υποθέτουμε, επίσης, ότι |**x**| = n και |**y**| = m.

(α) Θέλουμε αλγόριθμο που ημιαποφασίζει το Πρόβλημα των Διοφαντικών Εξισώσεων, δηλαδή, έναν αλγόριθμο που παίρνει για είσοδο (D, **x**, **y**) και δίνει για έξοδο YES αν υπάρχει αποτίμηση του **x** τέτοια ώστε D(**x**, **y**) = 0 για κάποια αποτίμηση του **y**.

Φτιάχνουμε το διάνυσμα **z** = **x**^**y** με |**z**| = n + m. Θα αποτιμήσουμε το D(**z**) = D(**x**, **y**), ωστόσο, πρέπει να βρούμε έναν τρόπο να το κάνουμε, ώστε ο αλγόριθμός μας να καταφέρνει πάντοτε σε πεπερασμένο πλήθος βημάτων να αποκρίνεται σωστά αν η διοφαντική εξίσωση της εισόδου έχει όντως λύση.

Για τον λόγο αυτό αντιστοιχίζουμε το σύνολο NxNx…xN (n + m φορές) στο σύνολο N με 1 προς 1 αντιστοίχιση (γίνεται για κάθε καρτεσιανό γινόμενο του N με τον εαυτό του d φορές). Έστω F η συνάρτηση αντιστοίχισης τότε F(**z**) = k και G(k) = **z** η αντίστροφη συνάρτηση.

Συνεπώς, ο αλγόριθμός A μας είναι ο εξής:

```
#for (k = 0; true; ++k)
z = G(k);
x^y = z; // get x, y from z
   #if (D(z) == 0) return YES;
   #endif
#endfor
```

Αν η D(**x**, **y**) έχει όντως λύση τότε υπάρχει **zs** = (**xs**, **ys**) τέτοιο ώστε D(**zs**) = 0 τότε ο A επιστρέφει YES στο βήμα ks = F(**zs**).

(β) DE: Diophantic Equation Problem, HP: Halting Problem. Θέλουμε μια τέτοια αναγωγή.

```
             ________________________________
            | DE                             |
            |    _____________       ___     |
  D(x,y)    |   |z=R(D(x,y)) |  z   |HP |    | DE(D(x,y)) = HP(z)
----------->|->|_____________|--->|___|-->|------------------>
            |________________________________|
```

Θέλουμε μια αναγωγή όπως η παραπάνω.

Τα προβλήματα που σχετίζονται είναι τα εξής:

$$DE\big(D(x,y)\big) = \begin{cases} YES, \text{αν η } D(x,y) \text{ έχει αποτίμηση των } x, y \text{ που την λύνει} \\ NO, \text{αλλιώς} \end{cases}$$

D(x, y) μια διοφαντική εξίσωση όπως την περιγράφουμε στο (α) ερώτημα.

$$HP(P, input) = \begin{cases} YES, \text{αν } P(input) \text{ τερματίζει} \\ NO, \text{αλλιώς} \end{cases}$$

P = ένα πρόγραμμα και input είναι η είσοδος που θα δώσουμε στο πρόγραμμα.

Και ο αλγόριθμος A που βρήκαμε στο πρώτο ερώτημα λειτουργεί ως εξής:

$$A\big(D(x,y)\big) = \begin{cases} YES, \text{αν } D(x,y) \text{ έχει αποτίμηση των } x, y \text{ που την λύνει} \\ Run\ Forever, \text{αλλιώς} \end{cases}$$

Δηλαδή, ο A τερματίζει μόνο αν η διοφαντική εξίσωση έχει λύση, αλλιώς τρέχει για πάντα.

Συνεπώς, αν στο HP δώσουμε για είσοδο τον αλγόριθμο A και την διοφαντική D(x, y) τότε:

Αν HP απαντήσει YES σημαίνει ότι η DE(D(x, y)) = YES ενώ αν HP απαντήσει NO σημαίνει ότι A(D(x, y)) δεν τερματίζει, δηλαδή, DE(D(x, y)) = NO.

Η αναγωγή από το DE στο HP είναι η εξής:

```
             ________________________________
            | DE                             |
            |    _____________       ___     |
  D(x,y)    |   |z=(A,D(x,y))|  z   |HP |    | DE(D(x,y)) = HP(z)
----------->|->|_____________|--->|___|-->|------------------>
            |________________________________|
```

Όπου A ο αλγόριθμος του (α) ερωτήματος.

Άσκηση 2: Πολυπλοκότητα – Αναγωγές

(α) Π είναι C – Complete άρα για κάθε X στο C έχω X <=(R) Π άρα υπάρχει συνάρτηση R(.) τέτοια ώστε X(I) = Π(R(I)).

$$X(I) = \begin{cases} YES, \alpha\nu\ \Pi\big(R(I)\big) = YES \\ NO, \alpha\nu\ \Pi\big(R(I)\big) = NO \end{cases} (1)$$

$$X'(I) = \begin{cases} NO, \alpha\nu\ \Pi\big(R(I)\big) = YES \\ YES, \alpha\nu\ \Pi\big(R(I)\big) = NO \end{cases} (2)$$

$$\Pi'(I) = \begin{cases} NO, \alpha\nu\ \Pi(I) = YES \\ YES, \alpha\nu\ \Pi(I) = NO \end{cases} (3)$$

Τα X', Π' ανήκουν στο coC εξ ορισμού. Από (2) και (3) συμπεραίνουμε ότι

$$X'(I) = \begin{cases} NO, \alpha\nu\ \Pi'\big(R(I)\big) = NO \\ YES, \alpha\nu\ \Pi'\big(R(I)\big) = YES \end{cases}$$

Δηλαδή, X' <=(R) Π'. Το X ήταν ένα τυχαίο πρόβλημα της C, άρα κάθε πρόβλημα της coC ανάγεται μέσω της R στο Π' και επειδή Π' ανήκει στο coC έχω ότι Π' είναι coC – Complete.

(β) Βήμα 1: Έστω ότι μας δίνεται μια ανάθεση τιμών για την οποία κάποιος μας λέει ότι η Boolean έκφραση αποτιμάται false, μπορούμε να το ελέγξουμε σε πολυωνυμικό χρόνο ως προς το μέγεθος της έκφρασης αντικαθιστώντας τις τιμές που μας έδωσαν στις μεταβλητές που έχουμε.

Βήμα 2: Θα δείξουμε ότι Tautology' είναι NP – Complete τότε από (α) Tautology είναι coNP – Complete.

Tautology' ανήκει στο NP γιατί Tautology ανήκει στο coNP από Βήμα 1.

Θα δείξουμε ότι SAT <=(P) Tautology':
Έστω x είσοδος στο SAT τότε δίνουμε στο Tautology' είσοδο x' (άρνηση του x – σε πολυωνυμικό χρόνο ως προς το μέγεθος της έκφρασης x).

Αν το Tautology' επιστρέψει true τότε υπάρχει αποτίμηση της x' = false άρα για την ίδια αποτίμηση x = true άρα x ικανοποίησιμη.

Αν το Tautology' επιστρέψει false τότε για κάθε αποτίμηση της x' = true άρα x = false για κάθε δυνατή αποτίμηση των μεταβλητών της άρα x μη ικανοποίησιμη.

Συνεπώς, Tautology' είναι NP – Complete άρα Tautology coNP – Complete.

(γ) Έστω Α ένα NP – Complete και coNP πρόβλημα τότε το Α′ είναι coNP – Complete και επειδή Α ανήκει στον coNP έχω ότι Α′ ανήκει στην NP. Γενικά, Α είναι NP – Complete και Α′ είναι coNP – Complete και ανήκουν και τα δύο και στην NP και στην coNP, οπότε μπορούμε να αναγάγουμε το ένα στο άλλο σε πολυωνυμικό χρόνο. Συνεπώς, μπορούμε να αναγάγουμε κάθε πρόβλημα του NP στο Α και κάθε πρόβλημα του coNP στο Α′. Από τις δύο αυτές προτάσεις και το γεγονός ότι μπορούμε να αναγάγουμε πολυωνυμικά το Α στο Α' και το αντίστροφο προκύπτει ότι NP = coNP.

(δ)
Θα δείξουμε αυτό: 3SAT <= s3SAT <= NAE4SAT <= NAE3SAT
1η αναγωγή:
s3SAT → Σαν το 3SAT αλλά κάθε clause έχει 3 ακριβώς literals.

3SAT		Μετατροπή
(a)	→	$(\neg s \lor \neg s \lor \neg s) \land (s \lor s \lor a)$
$(a \lor b)$	→	$(\neg s \lor \neg s \lor \neg s) \land (s \lor b \lor a)$

Αν η πρόταση έχει 3 literals την αφήνουμε όπως είναι. Το s δεν θα επηρεάσει το αποτέλεσμα, γιατί αν θέλουμε να γίνει SATISFIED η πρόταση της μετατροπής πρέπει το $(\neg s \lor \neg s \lor \neg s)$ = true, δηλαδή, s = false.

2η αναγωγή:
NAE4SAT: ίδιο με το NAE3SAT αλλά με 4 literals ανά clause.

s3SAT		Μετατροπή
$(a \lor b \lor c)$	→	$(a \lor b \lor c \lor s) \land (\neg a \lor \neg b \lor \neg c \lor \neg s)$

Παρατηρούμε ότι αν βρούμε μια αποτίμηση με s = true τότε μας ικανοποιεί και μια αποτίμηση με s = false. Κρατάμε τη δεύτερη, ώστε να μην τύχει a = b = c = s = true ή a = b = c = s = false όλα ταυτόχρονα άρα να ικανοποιείται η συνθήκη του NAE4SAT.

3η αναγωγή:

NAE4SAT		Μετατροπή
$(a \lor b \lor c \lor d)$	→	$(s \lor a \lor b) \land (\neg s \lor c \lor d)$

Εδώ, αν a = b = true, μπορούμε να διαλέξουμε s = false χωρίς να αλλοιώσουμε την τιμή της έκφρασης της μετατροπής αλλά να ικανοποιούμε την συνθήκη του NAE3SAT.

Αν a = b = c = true (δεν γίνεται a = b = c = d = true, γιατί κάνουμε αναγωγή από το NAE4SAT) τότε μπορούμε να διαλέξουμε s = false.

Παρόμοια και για άλλες περιπτώσεις δείχνουμε ότι υπάρχει επιλογή του s ώστε καμία πρόταση 3 literals στη τελική μορφή να μην έχει και τα 3 αποτιμημένα σε true και επιτρέπεται να αλλάζουμε την τιμή του s ώστε να μας βολεύει χωρίς να χαλάμε την λύση που μας βρίσκει το NAE3SAT.

Όλες οι προσθήκες μεταβλητών στις αναγωγές ήταν πολυωνυμικές άρα η αναγωγή 3SAT <= NAE3SAT ήταν πολυωνυμική.

Το NAE3SAT είναι επαληθεύσιμο σε πολυωνυμικό χρόνο: Αν μας δώσουν τιμές για τις μεταβλητές τις αντικαθιστούμε και αποτιμούμε την έκφραση.

Οπότε NAE3SAT είναι NP – Complete.

(ε) U = {σύνολο κατοίκων} και τα S = {U1, U2, …, Ur} είναι υποσύνολα του U.

Κατασκευάζουμε ένα γράφημα με {u1, u2, …, ur} κορυφές και βάζουμε ακμές e = (ui, uj) αν Ui∩Uj = 0. Έστω ότι κάνουμε κάποιον νόμιμο χρωματισμό στο γράφημα τότε βλέπουμε ότι μπορούμε να χρωματίσουμε όλες τις ui που κάνουν overlap μεταξύ τους με το ίδιο χρώμα αν θέλουμε (το χρώμα αυτό σημαίνει ότι έχουν κοινό αντιπρόσωπο στο τοπικό κοινοβούλιο) ενώ δύο κορυφές που ενώνονται με ακμή έχουν διαφορετικό χρώμα (δεν είναι δυνατόν να έχουν κοινό αντιπρόσωπο άρα χρειαζόμαστε διαφορετικό αντιπρόσωπο – χρώμα για την κάθεμία).

Έτσι, λοιπόν, αν λύσουμε το k – Coloring λύνουμε και το αρχικό πρόβλημα.

Αν μας δοθεί ένα γράφημα G(V, E) με |V| = r τότε για την κορυφή us παίρνουμε το συμπλήρωμα της λίστας γειτνίασής της Us και κατασκευάζουμε τα {U1, U2, …, Ur}. Τώρα αν λύσουμε το πρόβλημα της εκφώνησης για U = {u1, u2, …, ur} και S = {U1, U2, …, Ur} λύνουμε και το k – Coloring.

Συνεπώς, τα δύο προβλήματα είναι ισοδύναμα.

Στις διαφάνειες υπάρχει ότι 3 – Coloring είναι NP – Complete, οπότε και το k – Coloring είναι NP – Complete (k – Coloring αποτελεί γενίκευση του 3 – Coloring και μπορεί να ελεγχθεί σε πολυωνυμικό χρόνο: Αν μας δώσουν έναν k – χρωματισμό τότε μετράμε τα χρώματα σε κάθε λίστα γειτνίασης κάθε κορυφής και αποφασίζουμε αν είναι όντως νόμιμος/εφικτός ή όχι).

Συνεπώς, το πρόβλημα είναι NP – Complete.

Άσκηση 3: Προσεγγιστικοί Αλγόριθμοι – Vertex Cover

1. Η συνθήκη τερματισμού του βρόγχου του αλγορίθμου μας είναι: «συνέχισε όσο το C δεν είναι Vertex Cover», δηλαδή, συνέχισε όσο υπάρχουν ακάλυπτες ακμές. Άρα το C θα καλύψει όλες τις ακμές του γραφήματος.

Ως «υπολειπόμενο» βάρος μιας κορυφής u θεωρούμε το t(u).

2. Μια κορυφή u μπαίνει στο C μόνον αν το «υπολειπόμενο» βάρος της μηδενιστεί σε κάποια επανάληψη και κάθε ακμή εξετάζεται μόνο αν καμία από τις δύο κορυφές της δεν είναι στο C. Συνεπώς, κάθε κορυφή u στο C είναι κορεσμένη και το σύνολο των ακμών μη μηδενικού κόστους που είναι προσκείμενες στην u έχει βάρος ίσο με w(u).

Συμπεριλαμβάνουμε και τις ακμές μηδενικού βάρους γιατί μας είναι αδιάφορες και έχουμε: $w(i) = \sum_{e=(i,j)} c(e)$. Άρα $w(C) = \sum_{i\epsilon V} \sum_{e=(i,j)} c(e)$. Στο άθροισμα της δεξιάς μεριάς μια ακμή e άρα και το κόστος της c(e) θα μετρηθεί το πολύ δύο φορές (αν i, j ανήκουν και τα δύο στο C) άρα $\sum_{i\epsilon V} \sum_{e=(i,j)} c(e) \leq 2 \sum_{e\in E} c(e)$ άρα, τελικά, $w(C) \leq 2 \sum_{e\in E} c(e)$.

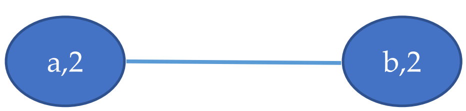
3. Από ερώτημα 2 για μια κορυφή i έχουμε είτε ότι $w(i) = \sum_{e=(i,j)} c(e)$ αν i στο C είτε $w(i) > \sum_{e=(i,j)} c(e)$, αλλιώς. Δεν γίνεται $w(i) < \sum_{e=(i,j)} c(e)$ γιατί ο αλγόριθμός βάζει την κορυφή στο C μόλις το «υπολειπόμενο» βάρος της μηδενιστεί ή μόλις το βάρος των εφαπτόμενων σε αυτήν ακμών γίνει ίσο με το δικό της. Συνεπώς, αθροίζοντας για όλες τις κορυφές του βέλτιστου C* έχω $w(C*) = \sum_{i\epsilon C*} w(i) \geq \sum_{i\epsilon C*} \sum_{e=(i,j)} c(e)$

Το C* είναι Vertex Cover άρα από (1) καλύπτει όλες τις ακμές άρα περιλαμβάνει τουλάχιστον όλα τα βάρη/κόστη c(e) των ακμών του γραφήματος. Τελικά,

$$w(C*) \geq \sum_{e\in E} c(e)$$

Από (2) και (3) έχω $w(C) \leq 2 \sum_{e\in E} c(e) \leq 2w(C*)$ και εξετάζουμε πρόβλημα ελαχιστοποίησης άρα ο αλγόριθμος είναι 2 – προσεγγιστικός.

Παράδειγμα: Αναπαράσταση κορυφών ως: (όνομα κορυφής, βάρος)



Στο γράφημα αυτό η βέλτιστη λύση είναι Copt = {a} με w(Copt) = 2 αλλά ο αλγόριθμός μας θα δώσει C = {a, b} με w(C) = 4.

Άσκηση 4: Πιθανοτικοί Αλγόριθμοι – Έλεγχος Ταξινόμησης

(α) Έστω ο πίνακας: T = [α1, α2, …, αs, b1, …, bm], s + m = n, m = ceiling(n/4) + 1 για τον οποία τα στοιχεία b είναι τα στοιχεία που χαλάνε την ταξινόμηση και ο T είναι έτσι διαμορφωμένος ώστε τα (α) να είναι ταξινομημένα μεταξύ τους και τα (b) να είναι ταξινομημένα μεταξύ τους. Συνεπώς, ο αλγόριθμος θα μας απαντήσει ότι ο T δεν είναι σχεδόν ταξινομημένος μόνο αν κάνει τη σύγκριση as με b1 και αυτή είναι η μόνη σωστή περίπτωση, γιατί έχουμε θεωρήσει ότι m > n/4.

Έστω ότι επιλέγουμε 1 στοιχείο τυχαία, τότε η πιθανότητα να επιλέξουμε κάποιο στοιχείο που θα μας δώσει σωστό συμπέρασμα είναι 2/n (μόνο η επιλογή των αs ή b1 θα δώσει σωστό συμπέρασμα). Άρα η πιθανότητα να βρούμε το λάθος είναι 1 – 2/n.

Η πιθανότητα να βρίσκουμε το λάθος σε k επιλογές είναι P = (1 – 2/n)^k

Γνωρίζουμε ότι (1 – x)^r >= 1 – rx (ανισότητα Bernoulli), αν r >= 1 και 0 <= x <= 1 και . Σε εμάς x = -2/n και r = k οπότε ισχύουν οι συνθήκες άρα P = (1 – 2/n)^k >= 1 – 2k/n και θέλουμε 0.1 > P >= 1 – 2k/n άρα 2k/n > 0.9 άρα k > 0.45n άρα:

Για να γίνει η πιθανότητα λάθους P μικρότερη από 10% χρειαζόμαστε k > 0.45n επαναλήψεις, δηλαδή, k = Ω(n).

Αποδεικνύουμε ότι ένας τέτοιος πίνακας υπάρχει:

b1 = 1, b2 = 2, …, bm = m και a1 = m + 1, a2 = m + 2, …, as = m + s

Επειδή m = ceiling(n/4) + 1 έχουμε ότι s = n – m > m άρα πρέπει να διαγράψουμε τουλάχιστον m στοιχεία (τα b1, …, bm) για να γίνει ταξινομημένος άρα ο πίνακας δεν είναι σχεδόν ταξινομημένος.

(β) Ο BINARY-SEARCH(A, input, low, up) συγκρίνει το input με το μεσαίο στοιχείο το οποίο υπολογίζει από τα low, up σε κάθε επίπεδο της αναδρομής. Αφού τα x, y έδωσαν θέσεις k, l με k < l αυτό σημαίνει ότι κάποια στιγμή, στο ίδιο επίπεδο της αναδρομής ο BINARY-SEARCH διάλεξε να ψάξει στο κάτω μισό για το x και στο πάνω μισό για το y, με άλλα λόγια σε εκείνο το επίπεδο της αναδρομής είχαμε x < A[mid] και y >= A[mid] άρα δείξαμε ότι x < y.

(γ) ---- Επισφαλές επιχείρημα ----

Στην εκτέλεση ενός BINARY-SEARCH θα συγκριθεί ένας αριθμός με το πολύ logn άλλους. Έστω ότι έχουμε ¼ αριθμούς που χαλάνε την ταξινόμηση τότε τα ¾ των επιλογών σύγκρισής μας θα δίνουν σωστό αποτέλεσμα στη τελική θέση άρα η πιθανότητα ένα BINARY-SEARCH να δώσει σωστό αποτέλεσμα είναι (3/4)^logn και αν κάνουμε k διαδοχικά BINARY-SEARCH έχουμε πιθανότητα ορθού αποτελέσματος P = (3/4)^klogn και θέλουμε P > 0.9 ➔ k < [log(0.9)/log(3/4)]*[log(2)/log(n)]. Άρα k = O(1) (φραγμένο από σταθερό αριθμό).