ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ
ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Αλγόριθμοι και Πολυπλοκότητα

2η Σειρά Γραπτών Ασκήσεων

Δημήτριος Γεωργούσης, 03119005

Άσκηση 1: Πολύχρωμος Πεζόδρομος

Μπορούμε να γράψουμε την εξής αναδρομή:

$$color(i,R) = \begin{cases} 0, αν\ i > n \\ color(i+1,R), αν\ R[i] == C[i] \\ 1 + \min\limits_{i \le j \le n} \{color(i+1,R)\}, όπου\ R[k] = C[i]\ για\ κάθε\ i \le k \le j, αλλιώς \end{cases}$$

Ως C θεωρούμε τον πίνακα με τα χρώματα που μας δίνεται και ως R θεωρούμε τον πίνακα που αρχίζει με όλες τις πλάκες λευκές (Τους αναπαριστούμε σαν πίνακες ακεραίων σαν το παράδειγμα της εκφώνησης).

Ορθότητα:

Αν ο δείκτης i είναι μεγαλύτερος του n τότε σημαίνει ότι έχουμε ολοκληρώσει τη βαφή των πλακιδίων άρα χρειαζόμαστε 0 παραπάνω ημέρες.

Όταν είμαστε στο πλακάκι i δοκιμάζουμε να το χρωματίσουμε σωστά με τους εξής τρόπους:

Αν το έχουμε χρωματίσει ήδη σωστά από το παρελθόν, τότε δεν κάνουμε τίποτα και πάμε στο επόμενο πλακάκι.

Αν δεν το έχουμε χρωματίσει τότε προσθέτουμε 1 στο αποτέλεσμα (σήμερα είναι η μέρα που θα το χρωματίσουμε) και δοκιμάζουμε να χρωματίσουμε κάθε δυνατό διάστημα που αρχίζει σε αυτό το πλακάκι και τελειώνει σε κάποιο άλλο επόμενο (ή στον εαυτό του).

Κρατάμε πάντοτε το ελάχιστο άρα μπορούμε να είμαστε σίγουροι ότι θα βρούμε τις ελάχιστες μέρες για τον χρωματισμό όλων των πλακιδίων.

Η παραπάνω αναδρομή, ακόμα και με memoisation να γίνει, επειδή χρειάζεται να κρατά ολόκληρο το state των πλακιδίων (πίνακας R) με βάση τα βαψίματα που έχουμε ήδη κάνει δεν έχει καλή πολυπλοκότητα ούτε χωρικά ούτε χρονικά.

Δεν κατάφερα να βρω καλύτερο αλγόριθμο (ή να γράψω καλύτερα τον δυναμικό προγραμματισμό…)

Άσκηση 2: String matching

Ο αλγόριθμος KMP σκοπό είχε να βρίσκει την πρώτη εμφάνιση του μοτίβου στο κείμενο (αυτοί με τα μικρότερα indexes στον πίνακα – κείμενο).

Θα τροποποιήσουμε τον αλγόριθμο αυτό ώστε να βρίσκει την πρώτη εμφάνιση του μοτίβου συμπεριλαμβάνοντας μπαλαντέρ * σε οποιαδήποτε θέση.

Οι βασικοί αλγόριθμοι είναι παρμένοι από το pdf "Σημειώσεις του Jeff Erickson" από το helios

```
def KMP(T[s..n],P[1..k]):
    if s > n
        return None
    fail[1..k] <-- computeFailure(P)
    j <-- 1
    for i <-- s to n
        while j > 0 and T[i] != P[j]
            j <-- fail[j]
        if j = k
            return (i - k + 1, i)
        j <-- j + 1
    return None
```

Θεωρούμε ότι η computeFailure είναι η συνάρτηση από το παραπάνω pdf. Η πρώτη αλλαγή που έχουμε κάνει είναι ότι επιτρέπουμε στον πίνακα T να ξεκινά από την θέση s. Δηλαδή, μπορούμε να την καλέσουμε με τρόπο που προσπερνάει τους πρώτους s – 1 χαρακτήρες το κειμένου. Επίσης, η επιστροφή είναι μία τούπλα του πρώτου και τελευταίου δείκτη του μοτίβου που βρήκαμε μέσα στο κείμενο. Αυτό θα μας βοηθήσει λίγο παρακάτω.

```
def problem2(T[1..n],P[1..m]):
    start = n + 1, end // start at big value
    if (P[1] == '*')
        start = 1 // * can be any string
    LP = P.split(*) // explaining how split is
                    // supposed to work
    // e.g. "*AB*CDE**F".split(*) returns ["AB","CDE","F"]
    temp_read = 0 // we have read 0 characters yet
    for pattern in LP: // parse through all the subpatterns
        (kmp_first,kmp_last) = KMP(T[(temp_read + 1)..n],pattern)
        if ((kmp_first,kmp_last) = None) // didn't find subpattern
            return None // won't find pattern
        start = min(start, kmp_first) // update starting position
        temp_read = kmp_last // read characters up to kmp_last
// since we exited the for loop this pattern does indeed exist
    if (P[m] == '*') end = n // we can include everything on the end
    else end = temp_read // just stop here
    return (start,end) // done
```

Αυτός είναι ο τελικός αλγόριθμος. Συμπεριφερόμαστε στα μπαλαντέρ σαν «επιτρεπόμενα κενά» μέσα στο μοτίβο. Χωρίζουμε το μοτίβο σε υπομοτίβα και τα αναζητούμε στο κείμενο με την σειρά εμφάνισής τους στο αρχικό μοτίβο. Μόλις βρούμε ένα υπομοτίβο αναζητούμε το επόμενο μόνο στο ανεξερεύνητο τμήμα του κειμένου.

Ορθότητα:

Έστω ότι υπάρχει μοτίβο για το οποίο υπάρχει λύση αλλά αλγόριθμός μας αποτυγχάνει. Από τον τρόπο που έχουμε κατασκευάσει τον αλγόριθμο φαίνεται ότι αυτό είναι δυνατό μόνο όταν σε κάποιο σημείο έχουμε προσπεράσει κάποιο από τα υπομοτίβα.

Αυτό δεν θα συμβεί γιατί 1) αναζητούμε τα υπομοτίβα με την σειρά εμφάνισής τους στο αρχικό μοτίβο και 2) πάντα βρίσκουμε την νωρίτερη εμφάνιση ενός υπομοτίβου στο υπολειπόμενο κείμενο.

Χρονική Πολυπλοκότητα:

P.split(*) → O(m)

Έστω ότι υπάρχουν p διαφορετικά υπομοτίβα με μήκη k1, k2, …, kp τότε
k1 + k2 + … + kp <= m

Όλες οι computeFailure είναι γραμμικές ως προς την είσοδο άρα

computeFailure( ) → O(k1 + k2 + … + kp) <= O(m)

Για τα for loops των KMP συναρτήσεων που εκτελούνται. Όσες φορές και να εκτελεστεί η KMP θα γίνουν συνολικά το πολύ n – 1 επαναλήψεις. Το πλήθος των φορών που μειώνεται το j δεν μπορεί να υπερβεί το πλήθος των φορών που αυτό αυξάνεται (το επιχείρημα του pdf) το οποίο είναι n – 1 άρα είναι O(n)

Συνολικά, O(n + m).

Άσκηση 3: Συντομότερα Μονοπάτια με Συντομεύσεις Ενδιάμεσων Ακμών

1. Μπορούμε να μηδενίσουμε το μήκος μιας μόνο ακμής στο s – t μονοπάτι που θα χρησιμοποιήσουμε.

Θα λύσουμε το πρόβλημα αυτό χρησιμοποιώντας ένα τροποποιημένο γράφημα H. Το γράφημα αυτό αποτελείται από το γράφημα G, ένα αντίγραφό του G′ στο οποίο όλες οι κορυφές/ακμές έχουν ίδια ονόματα με τις αντίστοιχες στο G αλλά με έναν τόνο (ʻ) για διαφοροποίηση. Για κάθε ακμή (u, v) του G βάζουμε στο H την ακμή (u, v′) με μηδενικό βάρος. Έχουμε |V(H)| = 2, |V(G)| = 2n και |E(H)| = 3, |E(G)| = 3m.

Α) Έστω x, y′ δύο κορυφές του H που ενώνονται με μονοπάτι μέσα σε αυτό. Επειδή η x είναι κορυφή του G και η y′ του G′ κάπου μέσα σε αυτό το μονοπάτι ακολουθήσαμε μια ακμή της μορφής (u, v′) και τα μονοπάτια x – u και v′ – y′ είναι μέσα στο G και το G′ αντίστοιχα, αφού υπάρχουν ακμές από το G προς το G′ αλλά όχι προς την αντίστροφη κατεύθυνση. Συνεπώς, το μονοπάτι x, y′ στο H συμπεριφέρεται σαν να ήταν το μονοπάτι x, y στο γράφημα G στο οποίο όμως μηδενίσαμε την (u, v).

Β) Όταν έχουμε έναν μηδενισμό ακμής διαθέσιμο πάντα μας ωφελεί να τον κάνουμε, γιατί πάντα θα μειώσουμε το μήκος του μονοπατιού (τα μήκη είναι θετικοί αριθμοί).

Αλγόριθμος:

1. Κατασκεύασε το γράφημα H, όπως περιγράψαμε παραπάνω.

2. Με αρχή την κορυφή s εκτέλεσε τον αλγόριθμο Dijkstra στο γράφημα H.

3. Επέστρεψε το ελάχιστο μεταξύ της τιμής που υπολόγισες για το μονοπάτι s – t και για το μονοπάτι s – t′.

Ορθότητα: Η απάντηση που επιστρέφουμε είναι σωστή γιατί με βάση την παρατήρηση Β πρέπει να κάνουμε τον μηδενισμό κάποιας ακμής στο μονοπάτι και με βάση την παρατήρηση Α το μονοπάτι s – t′ περιέχει έναν τέτοιο μηδενισμό. Επίσης, αφού τρέχουμε τον Dijkstra θα κρατήσουμε το ελάχιστο μήκος από όλα αυτά τα μονοπάτια. Η σύγκριση που κάνουμε στο βήμα 3 σκοπό έχει να συμπεριλάβει και την περίπτωση που s ταυτίζεται με t και άρα η απόσταση s – t θα είναι μηδέν.

Πολυπλοκότητα: Η κατασκευή του H είναι γραμμική ως προς τα |V| = n, |E| = m και σε χώρο καταλαμβάνει τον αντίστοιχο χώρο ανάλογα με τον τρόπο που αναπαριστούμε το γράφημα. Η υπόλοιπη πολυπλοκότητα είναι ίδια με την πολυπλοκότητα του Dijkstra αλλά αντί για (n, m) έχουμε (2n, 3m). Οι σταθερές 2 και 3 όμως δεν παίζουν κάποιον ρόλο.

2. Αφού τα μήκη των ακμών είναι θετικοί αριθμοί, μας ωφελεί να κάνουμε όλους τους μηδενισμούς ακμών που έχουμε διαθέσιμους αφού με κάθε μηδενισμό μειώνεται το συνολικό μήκος της διαδρομής.

Στην πρώτη περίπτωση αντιμετωπίσαμε τα G, G′ σαν να ήταν «στρώματα» και μεταβαίναμε στο «επόμενο» στρώμα μόνο όταν κάναμε μηδενισμό μιας ακμής. Θα λύσουμε και αυτό το πρόβλημα γενικεύοντας τον προηγούμενο αλγόριθμο και χρησιμοποιώντας ένα «πολυστρωματικό» γράφημα H.

Αλγόριθμος:

1. Κατασκευάζουμε το γράφημα H. Το H αποτελείται από τα αντίγραφα G^0, G^1, G^2, …, G^k του G (αν η κορυφή x και η ακμή (u, v) ανήκουν στο G τότε η x^j και η (u^j, v^j) ανήκουν στο G^j) και αν η ακμή (u, v) ανήκει στο G τότε οι ακμές (u^i, v^{i+1}) ανήκουν στο H για i = 0, …, k – 1 με μηδενικό βάρος.

2. Τρέχουμε τον αλγόριθμο Dijkstra εκκινώντας από την κορυφή s^0.

3. Επέστρεψε το ελάχιστο των μηκών των μονοπατιών $s^0 – t^0$, $s^0 – t^1$, …, $s^0 – t^k$.

Ορθότητα: Όπως και πριν συνεχίζει να ισχύει ότι όσο περισσότερους μηδενισμούς ακμών κάνουμε μέσα στο μονοπάτι μας τόσο μικρότερη είναι η τελική απόσταση. Το μονοπάτι $s^0 – t^j$ αντιπροσωπεύει το να πάμε από την s στην t στο γράφημα G με j μηδενισμούς ακμών μέσα στο μονοπάτι που χρησιμοποιούμε και, αφού το βρήκαμε με Dijkstra, έχει την μικρότερη δυνατή τιμή.

Ο λόγος που συγκρίνουμε τις τιμές στο βήμα 3 και δεν διαλέξαμε να επιστρέψουμε κατευθείαν το μήκος του $s^0 – t^k$ είναι για να συμπεριλάβουμε την περίπτωση που s = t όπως και στο πρώτο ερώτημα και να συμπεριλάβουμε τις περιπτώσεις που όλα τα μονοπάτια από το s έως το t έχουν μήκος μικρότερο από k.

Αν κάποιο μονοπάτι $s^0 – t^j$ δεν υπάρχει τότε θεωρούμε το μήκος του ως άπειρο (μας επιστρέφει την ένδειξη αυτή ο Dijkstra).

Χρονική Πολυπλοκότητα:

+ Κατασκευή H → O(kn + km)

+ Dijkstra → Ό,τι πολυπλοκότητα έχει η μορφή με την οποία τον χρησιμοποιούμε αλλά πρέπει να θεωρήσουμε ότι |V(H)| = O (kn) και |E(H)| = O(km)

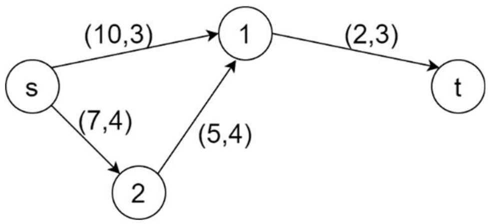
+ Επιστροφή → O(k)

Χωρική Πολυπλοκότητα:

Ο χώρος που χρειαζόμαστε εξαρτάται από τον τρόπο που έχουμε επιλέξει να αναπαραστήσουμε το γράφημα και την υλοποίηση του Dijkstra που διαλέξαμε.

Άσκηση 4: Σύντομα Μονοπάτια με Επαρκή Μεταφορική Ικανότητα

1.



(s,1,t): 12/3 = 4, (s,2,1,t): 14/3 = 4.666 άρα βέλτιστο είναι το μονοπάτι (s,1,t).

(s,1): 10/3 = 3.333, (s,2,1): 12/4 = 3 άρα το (s,2,1) είναι βέλτιστο.

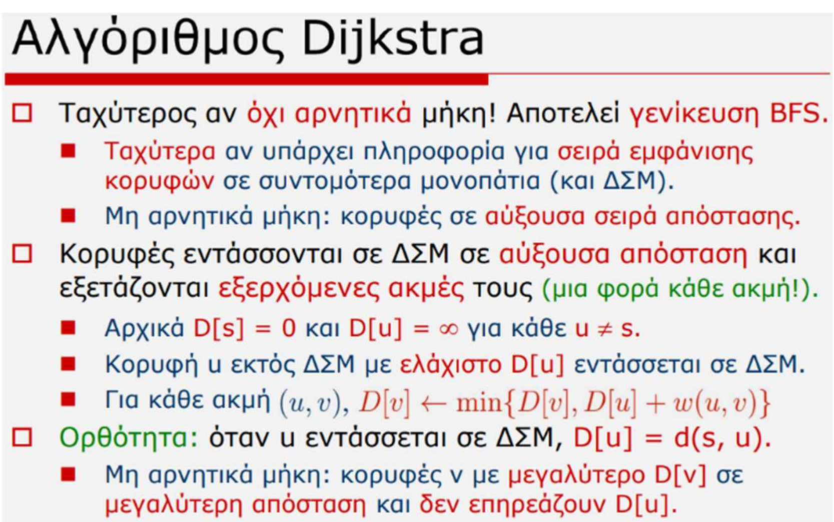
Επιβεβαιώνουμε ότι μπορεί το πρόθεμα ενός βέλτιστου s – t μονοπατιού να μην είναι βέλτιστο μονοπάτι.

2. Θεωρούμε την εξής αναδρομική σχέση:

$$dist(x,i,c,b) = \begin{cases} c/_{b}\,, \alpha\nu\ x = t \\ +\infty, \alpha\nu\ i \geq |V| \text{ ή } Neighbor(x) = \emptyset \\ \min\limits_{n \in Neighbor(x)} \{dist(n, i+1, c + c(x,n), \min\{b, b(x,n)\})\}\,, \alpha\lambda\lambda\iota\acute{\omega}\varsigma \end{cases}$$

Πραγματοποιούμε την κλήση $dist(s, 0, 0, +\infty)$ για να κάνουμε την αναδρομή να ακολουθήσει τα κατάλληλα μονοπάτια. Αν χρησιμοποιούμε memoisation τότε με backtracking μπορούμε να βρούμε το μονοπάτι στο οποίο απευθύνεται η τιμή επιστροφής αυτής της αναδρομής, η οποία είναι ο βέλτιστος λόγος κόστους προς μεταφορικής ικανότητας σε όλα τα μονοπάτια s – t.

3. Για τον αλγόριθμό μας θα χρειαστούμε μια ελαφριά τροποποίηση του Dijkstra, γιατί θέλουμε να βρίσκει το μονοπάτι μικρότερου κόστους με μέγιστη μεταφορική ικανότητα.



Στη δική μας περίπτωση το D είναι το έως τώρα εκτιμώμενο κόστος και το w(u,v) είναι το κόστος c(u,v) της ακμής.

Η τροποποίησή μας είναι η εξής:

Στο βήμα που προσθέτουμε την κορυφή χαμηλότερου κόστους σε περίπτωση που υπάρχουν περισσότερες από μία τέτοιες διαλέγουμε αυτήν που η ακμή που την ενώνει στο Δέντρο Συντομότερων Μονοπατιών (ΔΣΜ) έχει τη μεγαλύτερη μεταφορική ικανότητα.

Για να το πετύχουμε αυτό πρέπει στο βήμα που ανανεώνουμε τις εκτιμήσεις μας για τα κόστη των κορυφών να «θυμόμαστε» για κάθε κορυφή τη μέγιστη μεταφορική ικανότητα από τις ακμές που ελαχιστοποίησαν το κόστος και αν δύο κορυφές έχουν ίδιο κόστος να δίνουμε μεγαλύτερη προτεραιότητα σε αυτή με τη μεγαλύτερη μεταφορική ικανότητα (π.χ. δευτερεύον κριτήριο σε μια ουρά προτεραιότητας που συνηθίζεται να χρησιμοποιείται σε υλοποιήσεις του Dijkstra).

Επίσης, για κάθε κορυφή, όταν ανανεώνουμε το μονοπάτι που την συνδέει στο ΔΣΜ αποθηκεύουμε τη μεταφορική ικανότητα του μονοπατιού αυτού, η οποία προκύπτει από τη σύγκριση της τιμής που έχουμε αποθηκεύσει για την κορυφή του ΔΣΜ με την οποία συνδεόμαστε και της μεταφορικής ικανότητας της αντίστοιχης ακμής. Ο λόγος που αποθηκεύουμε την πληροφορία αυτή είναι για να μπορούμε να βρούμε τη χωρητική ικανότητα του μονοπατιού σε σταθερό χρόνο όταν κάνουμε συγκρίσεις στο Step2 του παρακάτω αλγόριθμου.

Οι τροποποιήσεις αυτές δεν επηρεάζουν την πολυπλοκότητα του Dijkstra (χωρικά και χρονικά).

Αλγόριθμος:

```
*****************************************************************
Start:  p* = [] // κρατάει το καλύτερο μονοπάτι που έχουμε βρει
                // στο τέλος θα είναι το βέλτιστο μονοπάτι
        r = inf // κρατάει τον λόγο c(p*)/b(p*)

    Step1:  Βρες το p, το οποίο είναι το συντομότερο s - t
            μονοπάτι με βάση τον τροποποιημένο Dijkstra

            Αν p δεν υπάρχει τότε πήγαινε στο Final

    Step2:  if c(p)/b(p) < r:   // Βρήκαμε καλύτερο μονοπάτι
                p* = p          // Ανανέωσε το βέλτιστο μονοπάτι
                r = c(p)/b(p)   // Ανανέωσε τη μεταβλητή λόγου

    Step3:  Σβήσε από το γράφημα G(V,E) όλες τις ακμές e
            με b(e) <= b(p)
            Πήγαινε στο Step1

Final:  Επίστρεψε το p*
*****************************************************************
```

Ορθότητα:

Επαναλαμβάνουμε ότι ο τροποποιημένος Dijkstra θα μας δώσει:

Από όλα τα συντομότερα s – t μονοπάτια που υπάρχουν εκείνο με τη μέγιστη μεταφορική ικανότητα, όπου η λέξη «συντομότερο» χρησιμοποιείται για να αναφερθούμε στο κόστος. Ονομάζουμε το μονοπάτι αυτό p.

Έτσι κάθε s – t μονοπάτι, έστω q, μικρότερης ή ακόμα και ίσης μεταφορικής ικανότητα στο γράφημά μας θα έχει: $b(q) <= b(p)$ και, αφού Dijkstra δίνει συντομότερο s – t μονοπάτι έχουμε ότι $c(q) >= c(p)$. Προκύπτει, λοιπόν, ότι $c(q)/b(q) >= c(p)/b(p)$.

Συνεπώς, κάθε άλλο μονοπάτι στο γράφημά μας το οποίο περιέχει έστω και μία ακμή e με $b(e) <= b(p)$ είναι σίγουρα όχι καλύτερο από το p.

Συνεπώς, σβήνοντας τις ακμές e με $b(e) <= b(p)$ δεν εξαιρούμε από το γράφημα καμία πιθανή λύση του προβλήματος, αλλά επιτρέπουμε στον Dijkstra να ψάξει εναλλακτικά μονοπάτια – πέρα από αυτά που έχουμε ήδη βρει – ώστε να βρει τη λύση.

$n = |V|$, $m = |E|$ στα παρακάτω.

Χρονική Πολυπλοκότητα:

Dijkstra (από διαφάνειες):

- Binary heap: Θ(mlogn)
- Fibonacci heap: Θ(m + nlogn)
- Ελάχιστο D[v] γραμμικά: Θ(n^2)

Σε κάθε επανάληψη: τρέχουμε έναν Dijkstra και σβήνουμε ακμές το οποίο μπορεί να γίνει με ένα πέρασμα σε όλες τις ακμές του γράφου, δηλαδή, σε O(m). Ο Dijkstra με οποιαδήποτε από τις παραπάνω υλοποιήσεις δεν έχει καλύτερη πολυπλοκότητα από O(m) άρα κάθε επανάληψη έχει ίδια πολυπλοκότητα με το να τρέξουμε Dijkstra στον γράφο που έχει απομείνει σε αυτό το βήμα.

Στη χειρότερη περίπτωση σε κάθε βήμα σβήνουμε μία μόνο ακμή άρα θα χρειαστούμε m επαναλήψεις και στην i-οστή επανάληψη τρέχουμε Dijkstra σε γράφο με n κορυφές και m – i + 1 ακμές. Επειδή (1 + 2 + … + m = Θ(m^2)) μπορούμε να πούμε ότι στη χειρότερη περίπτωση η πολυπλοκότητα του αλγορίθμου μας είναι ό,τι είναι και να τρέξουμε m φορές Dijkstra σε γράφημα G(V, E) με $|V| = n$ και $|E| = m$.

Χωρική Πολυπλοκότητα:

Ό,τι χώρο χρειάζεται το γράφημα και ο Dijkstra για να αποθηκευτούν. Επιπλέον, θέλουμε χώρο για το μονοπάτι που βρίσκουμε ή/και αποθηκεύουμε ο οποίος είναι O(min{n,m}).

Άσκηση 5: Αγορές Προϊόντων από Συγκεκριμένα Καταστήματα

1. Αρχικά, αν $|S| < n$ μπορούμε να απαντήσουμε κατευθείαν ότι δεν υπάρχει το ζητούμενο σύνολο καταστημάτων.

Για τη περίπτωση που $|S| >= n$ κάνουμε τα εξής:

Κατασκευάζουμε ένα διμερές γράφημα H(S,P,E1), όπου S, P τα σύνολα καταστημάτων και προϊόντων αντίστοιχα ως κορυφές του γραφήματος και στο E1 ανήκουν μόνο ακμές (si, pj) τέτοιες ώστε το si να ανήκει στο σύνολο καταστημάτων Sj του προϊόντος pj.

Το πρόβλημα διατυπώνεται ως εξής:

Στο H θέλουμε να βρούμε ένα P – τέλειο ταίριασμα, δηλαδή, ταίριασμα M στο οποίο όλες οι κορυφές του P είναι M – κορεσμένες. Το σύνολο κορυφών si τέτοιες ώστε η ακμή (si, pj) να ανήκει στο M για κάποιο j είναι το ζητούμενο σύνολο καταστημάτων.

Έστω ότι M ένα ταίριασμα με $|M| = k$. Κάθε ακμή του M έχει μια κορυφή στο S και μια κορυφή στο P και μία κορυφή δεν μπορεί να συμμετέχει σε δύο ακμές του M (αφού είναι ταίριασμα) άρα έχει k κορυφές από το S και k κορυφές από το P. (A)

Αν το μέγεθος ενός μέγιστου ταιριάσματος M είναι k:

Αν $k = n$ τότε είναι P – τέλειο ταίριασμα.

Αν $k < n$ τότε για κάθε πιθανό ταίριασμα N θα ισχύει $|N| <= |M| = k$ (γιατί αλλιώς το M δεν θα ήταν μέγιστο) και άρα σε κάθε ταίριασμα θα συμμετάσχουν το πολύ k κορυφές του P άρα όχι όλα τα προϊόντα άρα δεν υπάρχει το ζητούμενο σύνολο καταστημάτων.

Η περίπτωση $k > n$ είναι αδύνατη γιατί τότε θα χρειαζόταν να υπάρχουν k προϊόντα αλλά υπάρχουν μόνο n.

Βρήκαμε, λοιπόν, μια μέθοδο λύσης του προβλήματος. Την παρουσιάζουμε παρακάτω:

Αν k το μέγεθος ενός μέγιστου ταιριάσματος M τότε:

- $k < n$: Δεν υπάρχει τέτοιο σύνολο καταστημάτων
- $k = n$: Το σύνολο καταστημάτων είναι το σύνολο των si όπου (si, pj) ανήκει στο M για κάποιο j

Κατασκευάζουμε ένα γράφημα ροής G(V,E), το οποίο περιγράφουμε ως εξής:

1. Περιέχει το γράφημα H(S,P,E1) αλλά με κατευθυνόμενες ακμές ώστε να εξέρχονται από τις κορυφές του συνόλου S. Οι ακμές αυτές έχουν άπειρη χωρητικότητα.
2. Έχει μία κορυφή αφετηρία – πηγή b. Υπάρχουν οι ακμές (b, si) για κάθε i με χωρητικότητα 1.
3. Έχει μία κορυφή τερματισμού – καταβόθρα t. Υπάρχουν οι ακμές (pj, t) για κάθε j με χωρητικότητα 1.

$|V| = m + n + 2 = O(m)$, $m >= n$ και $|E| = |S1| + \ldots + |Sn| + m + n = O(m * n)$

Μία σημείωση για τις ακμές άπειρης χωρητικότητας του E1:

Έστω μια τέτοια ακμή e = (si, pj) και μια αποδεκτή b – t ροή f στο γράφημα G. Λόγω της συνθήκης ισορροπίας στις κορυφές έχουμε ότι:

Αν e συμμετέχει στην μεταφορά ροής στην f τότε μεταφέρει μοναδιαία ροή λόγω των 2. και 3. στην κατασκευή του G.

Αν e δεν συμμετέχει στην μεταφορά ροής στην f τότε έχει μηδενική τιμή.

Οι ακμές αυτές, δηλαδή, συμπεριφέρονται σαν να είχαν μοναδιαία χωρητικότητα.

Έστω f μια αποδεκτή b – t ροή τότε λόγω του 2 δεν γίνεται να υπάρχουν δύο ακμές με την ίδια αφετηρία si που μεταφέρουν ροή και λόγω του 3 δεν γίνεται να υπάρχουν δύο ακμές με το ίδιο τέρμα pj που μεταφέρουν ροή. Συνεπώς, οι ακμές του E1 που μεταφέρουν ροή στην f αποτελούν ταίριασμα στο γράφημα H.

Έστω ότι f η μέγιστη b – t ροή τότε οι ακμές του E1 που συμμετέχουν σε αυτήν αποτελούν μέγιστο ταίριασμα M στο H.

Έστω ότι αυτό δεν ισχύει. Υπάρχει ταίριασμα N στο H με $|M| = k1 < k2 = |N|$. Από (A):

Το M χρησιμοποιούσε k1 κορυφές του S και k1 κορυφές του P άρα η ροή f έχει μέγεθος k1.

Το N χρησιμοποιεί k2 κορυφές του S και k2 κορυφές του P άρα η ροή στο G που μπορούμε να φτιάξουμε αξιοποιώντας τις (b, si) με si τις k2 κορυφές του S, τις (si, pj) με pj τις κ2 κορυφές του P και τις (pj, t) ακμές έχει μέγεθος $k2 > k1$. Άτοπο.

Μπορούμε να λύσουμε το πρόβλημα με αλγόριθμο για το max flow πρόβλημα.

Αλγόριθμος:

Βήμα 0: Αν n > m δεν υπάρχει το ζητούμενο σύνολο

Βήμα 1: Κατασκεύασε το γράφημα G με την διαδικασία που περιγράψαμε παραπάνω

Βήμα 2: Βρες τη μέγιστη b – t ροή με κάποιον αλγόριθμο (π.χ. Ford – Fulkerson ή Edmonds – Karp)

Βήμα 3.1: Αν max flow < n τότε δεν υπάρχει το ζητούμενο σύνολο

Βήμα 3.2: Αλλιώς, ξέρουμε ποιες ακμές έχουν χρησιμοποιηθεί για τη μεταφορά της παραπάνω ροής (η πληροφορία συγκρατείται από τον αλγόριθμο που χρησιμοποιήσαμε για να βρούμε τη μέγιστη ροή)

Για κάθε μία από αυτές τις ακμές, αν είναι της μορφής (si, pj) για κάποια i, j βάλε στο S′ το si.

Επίστρεψε το S′.

Χρονική Πολυπλοκότητα: Αν χρησιμοποιήσουμε τον αλγόριθμο Ford – Fulkerson ή κάποια από τις βελτιώσεις σε αυτόν τότε έχουμε χρονική πολυπλοκότητα O(|E| * max_flow) αλλά max_flow <= n στο γράφημά μας και |E| = O(m * n) οπότε έχουμε χρονική πολυπλοκότητα O (m * n * n) ή, αλλιώς, O(m * n^2).

2. Η άσκηση αυτή μας ζητάει να βρούμε το μέγιστο ανεξάρτητο σύνολο σε διμερές γράφημα H(S,P,E1) το οποίο περιγράψαμε στο προηγούμενο ερώτημα.

Έχουμε το γράφημα G που παράγουμε όπως περιγράφουμε παραπάνω.

Έστω ένα πεπερασμένο κόψιμο ροής (A, B) με b να ανήκει στο A και t να ανήκει στο B. Ένα τέτοιο κόψιμο ροής πάντα υπάρχει (π.χ. το A = {b} και B = (όλες οι υπόλοιπες κορυφές)). Έστω S′ = S ∩ A και P′ = P ∩ B τότε το σύνολο S′ ∪ P′ είναι ανεξάρτητο σύνολο (οι κορυφές που περιέχει δεν ενώνονται με ακμές) γιατί, αφού θεωρήσαμε άπειρη χωρητικότητα στις ακμές της μορφής (si, pj), το παραπάνω κόψιμο δεν μπορεί να «διέσχισε» κάποια τέτοια ακμή γιατί δεν θα ήταν πεπερασμένο τότε.

Το κόψιμο έχει μέγεθος C = |S – S′| (από τις ακμές (b, si) που αφαίρεσα) + |P – P′|(από τις ακμές (pj, t) που αφαίρεσα) άρα C = |S| + |P| - | S′ ∪ P′| άρα | S′ ∪ P′| = |S| + |P| - C.

Αν το μέγεθος του κοψίματος ροής γίνει ελάχιστο τότε το | S′ ∪ P′| μεγιστοποιείται.

Έστω ότι μας δίνεται ανεξάρτητο σύνολο S′ ∪ P′ με S′ ≤ S και P′ ≤ P.

Βάζουμε στο σύνολο A το b, τις κορυφές του S′ και όλους τους γείτονες των κορυφών του S′.

Βάζουμε στο σύνολο B το t, τις P′ και όλους τους γείτονες των κορυφών του P′.

Αν μια κορυφή έχει απομείνει τότε

Αν είναι γειτονική του b βάζουμε αυτήν και τους γείτονές της στο A.

Αν είναι γειτονική του t βάζουμε αυτήν και τους γείτονές της στο B.

Αν είναι γειτονική και των δύο βάζουμε αυτήν και τους γείτονές της όπου να ναι.

Στην γειτονικότητα δεν λαμβάνουμε υπόψη την κατεύθυνση των ακμών.

Φροντίζουμε έτσι κάθε ακμή άπειρης χωρητικότητας να μην διασχίζει κορυφές των A, B άρα το κόψιμο ροής που φτιάξαμε είναι πεπερασμένο.

Συνεπώς, μπορούμε να αντιστοιχίσουμε κάθε ανεξάρτητο σύνολο κορυφών σε πεπερασμένο κόψιμο ροής στο γράφημα.

Συνεπώς, αποδείξαμε ότι:

#(μέγιστο ανεξάρτητο σύνολο κορυφών) = $|S| + |P| - |\text{min cut}|$

Αν βρούμε το min cut ή, ισοδύναμα, τη μέγιστη ροή τότε βρίσκουμε και το σύνολο καταστημάτων – προϊόντων που θέλουμε.

Αλγόριθμος:

Βήμα 1: Κατασκεύασε το γράφημα G με την διαδικασία που περιγράψαμε παραπάνω

Βήμα 2: Βρες τη μέγιστη b – t ροή με κάποιον αλγόριθμο (π.χ. Ford – Fulkerson ή Edmonds – Karp)

Βήμα 3: Στο υπολειμματικό δίκτυο (που χρησιμοποίησε ο Ford – Fulkerson για να μας δώσει απάντηση) θεώρησε ως σύνολο A την b και κάθε κορυφή στην οποία μπορούμε να φτάσουμε εκκινώντας από την b. Θεώρησε ως σύνολο B τις υπόλοιπες κορυφές. Αυτό μπορεί να γίνει με BFS ή DFS.

Έχουμε ότι το μέγιστο σύνολο προϊόντων και καταστημάτων $S' \cup P'$, $S' \leq S$, $P' \leq P$, τέτοιο ώστε κανένα από τα προϊόντα του P′ να μην πωλείται από τα καταστήματα του S′ είναι αυτό με $S' = S \cap A$ και $P' = P \cap B$.

Άσκηση 6: Ενοικίαση Αυτοκινήτων

Θα αναφερόμαστε στα [si, ti) ως διαστήματα.

Κατασκευάζουμε ένα interval Graph χρησιμοποιώντας τα διαστήματα αυτά. Δηλαδή ένα γράφημα στο οποίο κάθε κορυφή i αντιπροσωπεύει την προσφορά με διάστημα [si, ti) και υπάρχει ακμή μεταξύ δύο κορυφών αν τα αντίστοιχα διαστήματά τους τέμνονται/αλληλοκαλύπτονται.

Θέλουμε να βρούμε έναν χρωματισμό με k χρώματα σε αυτό το γράφημα τέτοιον ώστε το άθροισμα των αντίτιμων των προσφορών που αντιπροσωπεύουν οι κορυφές αυτού του χρωματισμού να είναι μέγιστο. Κάθε χρώμα αντιπροσωπεύει ένα αυτοκίνητο και το σύνολο κορυφών ίδιου χρώματος αντιπροσωπεύει το σύνολο προσφορών που εξυπηρετεί το αυτοκίνητο αυτό.

Σε ένα interval Graph μία κλίκα αντιπροσωπεύει ένα σύνολο προσφορών διένεξης (τα διαστήματά τους τέμνονται).

Στο πρόβλημά μας, αφού έχουμε k χρώματα, μπορούμε να χρωματίσουμε κλίκες με μέγεθος το πολύ k. (A)

Μετασχηματισμός προβλήματος:

Σβήσε ένα υποσύνολο των κορυφών του γραφήματος ελάχιστου συνολικού αντίτιμου των προσφορών του τέτοιο ώστε όλες οι κλίκες που μένουν στο γράφημα να έχουν μέγεθος το πολύ k. (B)

Αλγόριθμος:

1: Βρες όλες τις μεγιστοτικές κλίκες q1, …, qr του interval Graph που περιγράφουμε παραπάνω. Δεν θα αναφέρουμε τον αλγόριθμο για αυτό το πρόβλημα, ωστόσο, τις πληροφορίες σχετικά με αυτό τις πήραμε από τη βικιπαίδεια Clique problem - Wikipedia στο "Listing all maximal cliques" τμήμα.

Ταξινομούμε τις κλίκες ως προς τον χρόνο τότε μια προσφορά υπάρχει σε κλίκες που είναι όλες συνεχόμενες.

Έστω ότι αυτό δεν ισχύει τότε υπάρχει μια δουλειά i η οποία εμπεριέχεται στην κλίκα qx και στην qz και όχι στην qy με x < y < z. Όλες οι προσφορές των qx, qz τέμνονται με το διάστημα [si, ti). Υπάρχει τουλάχιστον μία προσφορά της qy με sj >= ti ή tj <= si. Αν ισχύει το πρώτο τότε οι προσφορές της qz τέμνονται με την j το οποίο είναι άτοπο γιατί qz μεγιστοτική. Αν ισχύει το δεύτερο τότε είναι άτοπο γιατί qx μεγιστοτική.

Αλγόριθμος:

Βρες τις μεγιστοτικές κλίκες και ταξινόμησέ τες όπως παραπάνω. Αν το μέγεθος της μεγαλύτερης κλίκας δεν υπερβαίνει το k τότε τα αυτοκίνητά μας αρκούν. Μπορούμε να εξυπηρετήσουμε όλες τις προσφορές.

Κατασκεύασε το κατευθυνόμενο γράφημα G ως εξής: Φτιάξε τις κορυφές u0, …, ur με ακμές (ui, ui-1) για i = 1, …, r. Οι ακμές αυτές έχουν κόστος 0 και οι χωρητικότητές τους είναι άπειρες. Κάθε τέτοια ακμή αναπαριστά μία κλίκα. Για κάθε προσφορά i που εμπεριέχεται στις κλίκες qj έως ql βάζω ακμή (uj-1,ul) κόστους pi και χωρητικότητας 1. Για κάθε κλίκα qj που δεν έχει μέγιστο μέγεθος βάζω ακμή (uj-1,uj) κόστους 0 και χωρητικότητας ίσης με (μέγιστο μέγεθος κλίκας – μέγεθος qj κλίκας) (ψευδοπροσφορές). Το u0 = s είναι η πηγή και το ur = t είναι η καταβόθρα/στόχος. Θεωρούμε ως επιθυμητό μέγεθος s – t ροής το (μέγιστο μέγεθος κλίκας – k).

Τρέχουμε min – cost – flow αλγόριθμο σε αυτό το γράφημα και οποιαδήποτε ακμή έχει μη μηδενική ροή στο αποτέλεσμα είναι μία από τις προσφορές που δεν χρειάζεται να εξυπηρετήσουμε, συνεπώς, την απορρίπτουμε και δεχόμαστε τις υπόλοιπες.

Ορθότητα:

Μπορούμε να σκεφτούμε τη ροή σε αυτό το γράφημα ως εξής: Το πλήθος των ακμών που ξεκινούν πριν μια κορυφή – κλίκα και καταλήγουν σε αυτήν ή σε επόμενή της αναπαριστά τις προσφορές που τέμνονται όλες μεταξύ τους και ανήκουν στην κλίκα αυτή. Μπορούμε να πούμε ότι η ροή που διέρχεται από αυτές τις ακμές διαπερνά την κορυφή – κλίκα. Λόγω της συμπερίληψης των ψευδοπροσφορών στο γράφημα μπορούμε να δούμε ότι κάθε κορυφή διαπερνάτε από ακμές συνολικής χωρητικότητας ίσης με το μέγιστο μέγεθος κλίκας, οι πραγματικές προσφορές έχουν χωρητικότητα 1 οπότε είτε τις επιλέγουμε είτε όχι σε μια ροή. Από αυτές τις ακμές - προσφορές πρέπει να διαλέξουμε μόνο k (θέλουμε να διαλέξουμε πραγματικές προσφορές που έχουν χωρητικότητα 1) όπως διαπιστώσαμε στο (Α). Συνεπώς, αφού με το min cost βρίσκουμε τον τρόπο προώθησης της ροής ελάχιστου οφέλους για εμάς διαλέγουμε να κάνουμε κάθε κορυφή να διαπερνάτε από (μέγιστο μέγεθος κλίκας – k) ακμές οι οποίες είναι μη ωφέλιμες προσφορές σε αντιστοιχία με το (Β). Έτσι, φαίνεται γιατί διαλέξαμε αυτή την ποσότητα για τη ροή και πώς το πρόβλημα ροής λύνει το αρχικό μας πρόβλημα.

Η ύπαρξη των ακμών άπειρης χωρητικότητας από μια κλίκα στη προηγούμενή της εξυπηρετεί στο να επιτρέπει στη ροή να γυρίσει σε προηγούμενες κορυφές ελεύθερα (αφού το κόστος των ακμών αυτών είναι 0) ώστε να ακολουθήσει από εκεί κάποια κορυφή χαμηλής/χαμηλότερης αξίας για να πετύχει την χαμηλότερη αξία συνολικά.

Το πλήθος των μεγιστοτικών κλικών είναι O(n) άρα έχουμε O(n) κορυφές. Το πλήθος των ακμών είναι O(n) επίσης.

Χρονική Πολυπλοκότητα:

Min – cost – flow έχει πολυπλοκότητα O(F * T) όπου F = size of flow και T = time required to find shortest path from source to sink. Στη δική μας περίπτωση F = O(n) προφανώς και μπορεί να χρησιμοποιηθεί μια τροποποιημένη εκδοχή του Dijkstra που χρειάζεται O(n^2) προετοιμασία και O(F * nlogn) = O(n^2 * logn).

Πηγή: [Minimum-cost flow - Algorithms for Competitive Programming (cp-algorithms.com)](https://cp-algorithms.com)

Η κατασκευή του γραφήματος και εύρεση των κλικών δεν είναι χειρότερες από O(n^2 * logn) άρα έχουμε συνολική πολυπλοκότητα O(n^2 * logn).

Χωρική πολυπλοκότητα:

Γράφημα (και το interval graph αλλά χρειάζεται το πολύ ίδια μνήμη με το τωρινό γράφημα) και να θυμόμαστε τα αντίτιμα των προσφορών. Το τελευταίο γίνεται σε γραμμικό χώρο άρα η πολυπλοκότητα εκφυλίζεται σε οτιδήποτε αναπαράσταση χρησιμοποιούμε για το γράφημα. Επίσης, ό,τι χώρο χρειάζεται το min – cost – flow.