Εθνικο Μετσοβιο Πολυτεχνειο

Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων

Δευτερη Σειρα Ασκησεων
Αλγοριθμοι και Πολυπλοκοτητα

Αναστασία Χριστίνα Λίβα
03119029

Περιεχόμενα

Άσκηση 1: Πολύχρωμος Πεζόδρομος

Βασική Ιδέα: Μια καλή εκκίνηση για την ανάπτυξη του αλγορίθμου που λύνει το πρόβλημά μου είναι ότι ψάχνω μεγάλα διαστήματα που έχουν άκρα με κοινό χρώμα και το βάφω όλο με αυτό το χρώμα. Στη συνέχεια ψάχνω αντίστοιχο υπό-διάστημα στο εσωτερικό του αρχικού διαστήματος αφού βέβαια τα άκρα του πρώτου μεγάλου διαστήματος είναι σωστά βαμμένα. Θα λύσω το πρόβλημά μου χρησιμοποιώντας δυναμικό προγραμματισμό. Ψάχνω μία αναδρομική σχέση η οποία θα βρίσκει το πλήθος των ημερών που χρειάζονται για να βαφούν όλες οι πλάκες.

Θα έχω $Days(i, j, c)$, όπου i, j είναι οι πλάκες από i μέχρι j, ενώ το c παίρνει τιμές των χρωμάτων $c_1, c_2, ... c_n$. Τις τιμές αυτές τις παίρνει όταν για παράδειγμα βρω δύο ακριανές πλάκες με ίδιο χρώμα, οπότε, όπως θα δείξω και στη συνέχεια, θα βάψω όλες τις ενδιάμεσες και θέλω να θυμάμαι το χρώμα. Αυτό με βοηθά να συμπεράνω για τις μεμονωμένες πλάκες που θα μείνουν στο τέλος αν θα χρειαστεί να τις βάψω ή όχι.

Το χρώμα μου όπως φαίνεται στην ποσότητα που θα υπολογίσω αρχικοποιείται με τιμή 0, δηλώνοντας πως για την ώρα οι πλάκες μου είναι άβαφες.

Θα χρησιμοποιήσω έναν πίνακα με διαστάσεων $n \times n \times n$, στον οποίο θα αποθηκεύω ό,τι τιμές έχω υπολογίσει για να μην τις ξαναϋπολογίσω από την αρχή. Έτσι, προκύπτει μία αναδρομική σχέση της μορφής:

$$Days[i][j][c] = \begin{cases} 1, \text{ if } i = j \text{ and } c \neq c_i \\ 0, \text{ if } i = j \text{ and } c = c_i \\ Days[i+1][j-1][c_i] + 1, \text{ if } c_i = c_j \neq c \\ Days[i+1][j-1][c_i], \text{ if } c_i = c_j = c \\ min_k\{Days[i][k][c] + Days[k+1][j][c]\} \end{cases}$$

Για να γίνει πιο κατανοητή εξηγώ τις 5 περιπτώσεις.

1. Βρίσκομαι στο τέλος της αναδρομής και η πλάκα δεν έχει το επιθυμητό χρώμα.
2. Βρίσκομαι στο τέλος της αναδρομής και η πλάκα έχει το επιθυμητό χρώμα, οπότε δε χρειάζεται να την ξαναβάψω.
3. Έχω δύο ίδιες ακριανές πλάκες οι οποίες αν δεν έχουν ήδη σωστό χρώμα θα χρειαστεί να βάψω αυτές και το διάστημα που ορίζουν.
4. Έχω δύο ίδιες ακριανές πλάκες οι οποίες έχουν ήδη σωστό χρώμα.
5. Δεν έχω τίποτα από τα παραπάνω οπότε πρέπει να κάνω κάποια διάσπαση για να συνεχίσω και να απομονώση πλάκες, κρατώνταςτο ελάχιστο, πάνω στο k, δυνατό αποτέλεσμα.

Αφού εξήγησα τα πιθανα ενδεχόμενα επίλυσης θα πρέπει να διορθώσω λίγο την παραπάνω σχέση. Πέρα από τις δύο αρχικές συνθήκες, για τις άλλες θα πρέπει να πάρω ένα ελάχιστο ώστε ο αλγόριθμός μας να είναι εξαντλητικός κατά την επίλυση. Συνεπώς:

$$Days[i][j][c] = \begin{cases} 1, \text{ if } i = j \text{ and } c \neq c_i \\ 0, \text{ if } i = j \text{ and } c = c_i \\ min \begin{cases} Days[i+1][j-1][c_i] + 1, \text{ if } c_i = c_j \neq c \\ Days[i+1][j-1][c_i], \text{ if } c_i = c_j = c \\ min_k\{Days[i][k][c] + Days[k+1][j][c]\} \end{cases} \end{cases}$$

Πολυπλοκότητα Αλγορίθμου: Η αναδρομική μου σχέση συμπληρώνει έναν τρισδιάστατο πίνακα, συνεπώς θα έχω πολυπλοκότητα της τάξης $O(n^3)$.

Ορθότητα Αλγορίθμου: Οι αρχικές συνθήκες που αφορούν μεμονομένες πλάκες είναι προφανείς. Σε κάθε άλλη περίπτωση κρατώ το ελάχιστο δυνατό πιθανό ενδεχόμενο. Αφού εξετάζω όλες τις περιπτώσεις και αποθηκεύω τον ελάχιστο αριθμό ημερών θα λάβω τον μικρότερο αριθμό ημερών, συνεπώς την βέλτιστη λύση

Άσκηση 2: String matching

Ερώτημα Πρώτο: Έστω μοτίβο p στο κείμενο t, άρα

$$t = *p*$$

όπου * οποιαδήποτε συμβολοσειρά.

Έχω: $pt = p * p*$, με πυρήνα το p ή πρόθεμα του p ή ε.

Για κάθε πρόθεμα του pt βρίσκω πάντα πυρήνα πρόθεμα του p ή του ε μέχρι να φτάσω στο p*p, όπου c(p*p) = p και συνεπώς δεν απαιτείται να συνεχιστεί η διαδικασία αφού το p ταιριάχτηκε στο t. Αν το p δεν υπάρχει στο κείμενο t τότε παραθέτοντας το p έχω pt = p*, όπου * οποιαδήποτε συμβολοσειρά εκτός του p. Καθώς κανένα πρόθεμα του pe δεν έχει τη μορφή p*p δεν θα βρω ποτέ κάποιο c(pt) = p, οπότε διασφαλίζω ότι το p δεν ταιριάζει με το κείμενο t.

Ερώτημα Δεύτερο: Ορίζω u κ-πυρήνας της $u \neq t$ αν ισχύουν τα ακόλουθα:

- $k \geq 0$
- $u < v$
- $length(u) \leq k$
- u είναι η μεγαλύτερη συμβολοσειρά με αυτή την ιδιότητα

Υποθέτοντας ότι ο κ-πυρήνας δεν είναι ορθά ορισμένος θα υπάρχουν $u_1,\ u_2$ με $u_1 \neq u_2$, για τα οποία ισχύει u_1 = κ πυρήνας της v και u_2 = κ πυρήνας της v. Ορίζω επίσης το $length(u_1) \leq k \rightarrow k_1$ και $length(u_2) \leq k \rightarrow k_2$

Το u_1 αποτελεί τη μεγαλύτερη συμβολοσειρά που διαθέτει την ιδιότητα. Το u_2 έχει επίσης την ιδιότητα και συνεπώς προκύπτει $k_1 \geq k_2$

Άρα

$$k_1 = k_2 \Longrightarrow length(u_1) = length(u_2)$$

Αφού $u_1 < v$ τότε $v = u_1 * u_1$

Αφού $u_2 < v$ τότε $v = u_2 * u_2$

Τα πρώτα k_1 σύμβολα της ίδιας συμβολοσειράς u είναι ίδια άρα $u_1 = u_2$ άτοπο, οπότε ο κ-πυρήνας είναι καλά ορισμένος.

Αλγόριθμος για Εύρεση κ-πυρήμα με Δοθέν κ: Έχοντας το κ γνωρίζω πως η συμβολοσειρά u που ζητείται δεν ξεπερνά το μήκος κ. Ορίζω τη συμβολοσειρά v σε πίνακα $p[N]$, $N = length(v)$, και έναν πίνακα $valid[k]$ ο οποίος θα έχει αληθή τιμή για κάθε i για το οποίο ισχύει $p[i] = p[N - i - 1]$. Το $length(u)$ θα ταυτίζεται με το μεγαλύτερο συνεχόμενο διάστημα αληθών τιμών στον πίνακα valid. Αφού βρούμε το $length(u)$ μπορούμε εύκολα να βρούμε και το u καθώς ταυτίζεται με τα πρώτα $length(u)$ στοιχεία του p.

Παραθέτω υποτυπώδη ψευδοκώδικα:

```
for j in range(j):
    if p[j] == p [N-j-1]:
        valid[j] = True
    else
        valid[j] = False

length = 0
for i in range(k):
    if valid[i] == True:
        length += 1
    else
        break

apotelesma = p[0]p[1]...p[length-1]
```

Άσκηση 3: Συντομότερα Μονοπάτια με Συντομεύσεις Ενδιάμεσων Ακμών

Γράφος: κατευθυνόμενος με n κόμβους και m ακμές.

Ερώτημα Πρώτο: Δημιουργώ ένα αντίγραφο του γράφου G τον οποίο ονομάζω G_1. Για κάθε ακμή του G που συνδέει έναν κόμβο u με άλλο κόμβο v δημιουργώ μια νέα ακμή μηδενικού βάρους η οποία συνδέει τον κόμβο u με τον αντίστοιχο κόμβο v_1 του γράφου G_1. Τελικά προκύπτει ένας εκτεταμένος γράφος G' ο οποίος περιλαμβάνει τον G, τον G_1 καθώς και τις νέες ακμές μηδενικού βάρους.

Στον εκτεταμένο γράφο G' εκτελώ τον αλγόριθμο του Dijkstra, ο οποίος θα μου δώσει το συντομότερο δυνατό μονοπάτι που μπορεί να προκύψει αφαιρώντας μια ακμή. Τελικά θα προκύψει ένα μονοπάτι στον εκτεταμένο γράφο το οποίο θα περιλαμβάνει μία από τις ακμές μηδενικού βάρους που πρόσθεσα. Υποθέτοντας πως η μηδενική αυτή ακμή συνδέει τον κόμβο u με τον v_1 η ακμή που τελικά θα αφαιρέσω από τον αρχικό είναι η ακμή που συνδέει τους γράφους u και v.

Ερώτημα Δεύτερο: Δημιουργώ k αντίγραφα του αρχικού μου γράφου G, οπότε πλέον έχω νέο γράφο G' με $n(k+1)$ κόμβους και $m(k+1)$ ακμές. Προσθέτω επίσης $k \cdot m$ ακμές στον γράφο G', δηλαδή για κάθε ακμή (u, v) του G και για κάθε $0 \leq i < k$ προσθέτω ακμή μηδενικού βάρους που ξεκινά από την κορυφή u του $i-$οστού αντιγράφου και καταλήγει στην κορυφη v του $(i+1)-$οστού αντιγράφου.

Είναι φυσικό επακόλουθο πως όταν βρίσκομαι στον κόμβο u του $i-$οστού από τα $k+1$ αντίγραφα του γράφου G είναι σαν να βρίσκομαι στον κόμβο u έχοντας μηδενίσει i ακμές. Άρα χρησιμοποιώντας κάποια από τις μηδενικές ακμές ουσιαστικά μηδενίζω την αντίστοιχη ακμή του αρχικού γράφου. Επίσης κάθε φορά που χρησιμοποιώ μια από αυτές τις ακμές ανεβαίνω κι ένα επίπεδο, οπότε με αυτό τον τρόπο διασφαλίζω ότι μηδενίζω το πολύ k ακμές.

Στη συνέχεια χρησιμοποιώ τον αλγόριθμο του Dijkstra για όλο το γράφο G', ο οποίος θα μου επιστρέψει το συντομότερο μονοπάτι του γράφου αυτού. Το συντομότερο μονοπάτι του G' ταυτίζεται με το συντομότερο δυνατό μονοπάτι που μπορώ να φτιάξω στον αρχικό μου γράφο μηδενίζοντας το πολύ k ακμές, αφού οι μηδενικές ακμές του συντομότερου μονοπατιού του G' ταυτίζονται με τις ακμές που επιλέγω να μηδενίσω στο γράφο G.

Βλέπω πως ο δεύτερος αλγόριθμος είναι γενίκευση του πρώτου οπότε αρκεί να αποδείξω την ορθότητα και να βρω την πολυπλοκότητα του δεύτερου αλγορίθμου για $k \geq 1$.

Ορθότητα Αλγορίθμου: Στον εκτεταμένο γράφο, ο οποίος περιλαμβάνει όλα τα επίπεδα, προσθέτω $k \cdot m$ ακμές στον γράφο G', δηλαδή για κάθε ακμή (u, v) του G και για κάθε $0 \leq i < k$ προσθέτω ακμή μηδενικού βάρους που ξεκινά από την κορυφή u του $i-$οστού αντιγράφου και καταλήγει στην κορυφη v του $(i+1)-$οστού αντιγράφου. Έτσι εξασφαλίζω πως μπορώ να πάρω το πολύ k από τις έξτρα μηδενικές ακμές που πρόσθεσα αφού ο γράφος μου είναι κατευθυνόμενος οπότε δε μπορεί να μεταβεί σε προηγούμενο επίπεδο (βλ. σημείωση). Επομένως ο αλγόριθμος Dijsktra θα τρέξει σε όλο το γράφο G' βρίσκοντας το ελάχιστο δυνατό μονοπάτι. Όπως εξηγήθηκε και νωρίτερα, κάθε μία από τις έξτρα μηδενικές ακμές που ανήκουν στο τελικό μονοπάτι που επιστρέφεται από τον αλγόριθμο Dijkstra αντιστοιχεί στο μηδενισμό του βάρους της ακμής που συνδέει τους αντίστοιχους αρχικούς κόμβους στο αρχικό γράφημα G.

Σημείωση: Κάθε μια εκ των επιπλεόν μηδενικων ακμών που πρόσθεσα είναι κατευθυνόμενη και οδηγεί απο το επίπεδο i στο επίπεδο $i+1$, άρα αφού έχω $k+1$ επίπεδα σύνολα μπορώ να χρησιμοποιήσω **το πολύ** k μηδενικές ακμές.

Πολυπλοκότητα Αλγορίθμου: Η κατασκευή των k αντιγράφων του αρχικού μου γράφου απαιτεί $O(k(n+m))$, ενώ η δημιουργία των $k \cdot m$ ακμών μηδενικού βάρους απαιτεί $O(km)$. Τέλος, είναι γνωστό πως για γράφο με V κόμβους και E ακμές ο αλγόριθμος του Dijsktra τρέχει σε χρόνο $O(V + ElogV)$, οπότε στην προκείμενη περίπτωση η εκτέλεση του αλγορίθμου για το συνολικό γράφο G' θα είναι της τάξης του $O(kn + kmlogkn)$. Τελικά προκύπτει ότι η συνολική πολυπλοκότητα του αλγορίθμου είναι $O(kn + kmlogkn)$.

Άσκηση 4: Ταξίδι σε Περίοδο Ενεργειακής Κρίσης

Ερώτημα Πρώτο:

Άπληστος Αλγόριθμος: Ορίζω πίνακα $gas[N]$, όπου $gas[i]$ η εναπομείνασα βενζίνη στο ντεπόζιτο του αυτοκινήτου ενώ βρίσκομαι στην πόλη i. Για κάθε πόλη i ψάχνω πόλη j, $j > i$ για στην οποία πωλείται φθηνότερη βενζίνη από ό,τι στην i, δηλαδή $c(i) > c(j)$, ή βρίσκχεται σε απόσταση μεγαλύτερη από B, δηλαδή $d_{ij} > B$.

- **Αν μπορώ να μεταβώ σε φθηνότερη πόλη:** Θα γεμίσω το ντεπόζιτο μου όσο ακριβώς χρειάζομαι για να καλύψω την απόσταση μεταξύ της φθηνότερης πόλης και της πόλης όπου ήδη βρίσκομαι, οπότε θα βάλω $d_{ij} - gas[i]$ λίτρα και φτάνοντας στη φθηνότερη πόλη j θα έχω $gas[j] = 0$
- **Εάν δε μπορώ να βρω φθηνότερη βενζίνη εντός του B:** Θα γεμίσω το ντεπόζιτό μου, δηλαδή θα βάλω $B - gas[i]$ λίτρα και θα επαναλάβω τη διαδικασία αυτή στον επόμενο κόμβο $i + 1$, όπου θα ισχύει $gas[i + 1] = B - d_i[i + 1]$

Αναδρομική Σχέση:

$$G(i, gas[i] = \begin{Bmatrix} G(j,0) + c(i)d_{ij} & j = min\{j > i \ \&\&d_{ij} \ \leq B\&\& \ c[j] < c[i] \\ G(i+1, B - d_{i(i+1)} + c(i)(B - gas[i]) & j = min\{j > 1 \ \&\&d_{ij} > B \end{Bmatrix}$$

Πολυπλοκότητα Αλγορίθμου: Στη χείριστη περίπτωση ο αλγόριθμος βλέπει κάθε πόλη για κάθε μια εκ των οποίων βλέπει και τις επόμενες πόλης της κλάσης του n. Άρα προκύπτει πολυπλοκότητα τάξης $O(n^2)$, όπου n είναι το πλήθος των κόμβων του γράφου.

Ερώτημα Δεύτερο:

Λύση Προβλήματος με Δυναμικό Προγραμματισμό: Θεωρώ $D(u, gas)$ το ελάχιστο κόστος μετάβασης από την πόλη u στην t με t λίτρα. Ξέρω ότι $D(t, b) = 0$ και θέλω να φτάσω στην t έχοντας τελικά καθόλου βενζίνη στο ντεπόζιτο μου, άρα $D(t, 0) = 0$. Ξεκινώντας από το t προς το s και αναζητώντας το $D(u, gas)$, όπου u η πόλη που μας κατευθύνει σε πόλεις $u_1, ..., u_i$ για τις οποίες γνωρίζω ήδη το ελάχιστο κόστος $D(u_i, gas_i)$ προκύπτει η ακόλουθη αναδρομική σχέση:

$$D(u, gas[i] = min \begin{Bmatrix} D(u,0) + c(u)(b(u,v) - gas) & c(v) < c(u) \ \&\& \ b(u,v) \geq gas \\ D(u, B - b(u,v)) + c(u)(B - gas) & c(u) < c(v) \\ \infty & \text{για τις υπόλοιπες περιπτώσεις} \end{Bmatrix}$$

Η λύση που θα προκύψει θα ισχούται με το $D(s, 0)$. Αν γνωρίζω το $D(s, 0)$ βρίσκω εύκολα το μονοπάτι.

Πολυπλοκότητα Αλγορίθμου: Για μικρές τιμές του B έχω πολυπλοκότητα $O(B|V|^2$ επειδή για κάθε κόμβο u πρέπει με ντεπόζιτο $0 \leq gas \leq B$ να δει τους γειτονικούς κόμβους που έχουν μέγιστο πλήθος Για μεγάλες τιμές του B ο αλγόριθμός μου θα έχει πολυπλοκότητα $O(|V|^3)$ καθώς το gas λαμβάνει διακριτές τιμές όσες και οι γείτονες του $|V|$.

Άσκηση 5: Παιχνίδια Εξουσίας

Ερώτημα Πρώτο: Θα ανάξω το πρόβλημα αυτό σε πρόβλημα μέγιστης ροής μέσω του ακόλουθου αλγόριθμου:

Αλγόριθμος: Έχοντας δημιουργήσει αρχικό κόμβο s και τελικό t για κάθε ιππότη $knight$ δημιουργώ έναν κόμβο k_i που συνδέεται με τον αρχικό κόμβο με ακμή με capacity c_i, η οποία ισούται με τα περισσότερα κάστρα που μπορεί να αναλάβει ο $knight$ και τον συνδέω με τον αρχικό κόμβο s. Για κάθε κάστρο $castle$ δημιουργώ κόμβο n_j τον οποίο και συνδέω με τον t με χωρητικότητα 1. Για κάθε κάστρο $castle$ και ιππότη $knight$ προσθέτω ακμή από τον k_{knight} στον n_{castle} χωρητικότητας 1. Τώρα εκτελώ τον αλγόριθμο ροής Ford - Fulkerson ξεκινώντας από τον κόμβο s και καταλήγοντας στον t. Αν στο τέλος η ροή ισούται με το πλήθος των κάστρων υπάρχει λύση.

Ορθότητα Αλγορίθμου: Αφού το πρόβλημα μου ανάγεται σε μέγιστη s-t ροή ισχύει ότι η ροή κάθε ακμής θα πρέπει να είναι μικρότερη ή ίση της χωρητικότητας, οπότε είναι σίγουρο πως κάθε ιππότης δε θα αναλάβει περισσότερα κάστρα από όσα γίνεται και σε κάθε κόμβο έχω διατήρηση ροής οπότε διασφαλίζεται ότι σε κάθε ιππότη θα ανατεθούν όσα κάστρα όσα και η εισερχόμενη ροή από την πηγή προς τον ιππότη, ενώ για τα κάστρα η εξερχόμενη ροή είναι μοναδιαία, οπότε και η εισερχόμενη θα είναι μοναδιαία και προκύπτει πως κάθε κάστρο θα ανατεθεί σε έναν ιππότη.

Πολυπλοκότητα Αλγορίθμου: Ο αλγόριθμος έχει ορθότητα $O((n+m) \cdot Emax \cdot max|c_i|)$

Ερώτημα Δεύτερο: Ο πρωθυπουργός θα μπορούσε να χρησιμοποιήσει το ακόλουθο επιχείρημα: Γνωρίζω τον αριθμό c των κάστρων που απαιτούν εποτπεία από κάποιον ιππότη και ο αλγόριθμος Ford - Fulkerson θα επιστρέψει τη μέγιστη ροή. Η μέγιστη ροή συνεπάγεται ότι το πλήθος των ακμών $n_j - t$ για τις οποίες υπάρχει ροή είναι το max της ροής που επιστρέφεται από τον Ford - Fulkerson, αφού οι χωρητικότητες κάθε ακμής που καταλήγει στην έξοδο ισούνται με τη μονάδα. Κάθε τέτοια ακμή σημαίνει ότι το κάστρο βρίσκεται στην εποπτία κάποιου ιππότη, οπότε εάν η μέγιστη ροή είναι μικρότερη από το πλήθος των κάστρων θα υπάρχουν $c-$ max της ροής κάστρα χωρίς υπεύθνο.

Ερώτημα Τρίτο: Θα κατασκευάσω έναν διμερή γράφο G. Στο αριστερό του μέρος τοποθετώ τους ιππότες B και στο αριστερο τους Π και συνδέω κάθε ζεύγος ιποτών που συγκρούονται από τις δύο ομάδες με ακμές. Στη συνέχεια θα αφαίρεσω τους ιππότες που αντιστοιχούν σε κορυφές του minimum vertex cover του γράφου και η λύση του προβλήματος αυτού θα ισοδυναμεί με τη λύση του maximum matching.

Αλγόριθμος: Έχοντας δημιουργήσει αρχικό κόμβο s και τελικό t ενώνω κάθε ιππότη B με τον s και κάθε Π με τον t με ακμές που έχουν μοναδιαία χωρητικότητα. Για κάθε σύγκρουση βάζω άπειρη χωρητικότητα στην αντίστοιχη ακμή και εκτελώ αλγόριθμο Ford - Fulkerson. Προκύπτει το μέγιστο πλήθος ακμών του γράφου τέτοιο ώστε κάθε κόμβος να συνδέεται με μια το πολύ ακμή (max cardinality matching). Θέτω Z τους κόμβους B που δεν ανήκουν στο max cardinality matching και προσθέτω στους κόμβους αυτούς τους κόμβους που μπορώ να πάω από τους κμβους Z μέσω μονοπατιών που εναλλάσσονται μεταξύ των ακμών που ανήκουν στο matching και αυτών που δεν ανήκουν. Προκύπτει το $S = (B \backslash Z) \cup (\Pi \cap Z)$ που είναι το minimum vertex cover των κόμβων. Εξορίζοντας τους ιππότες του S τα πυρά θα παύσουν.

Άσκηση 6: Ενοικίαση Αυτοκινήτων

Αναγωγή σε Min Cost Flow: Για κάθε προσφορα δημιουργώ κόμβους και ακμές με κόστη $-p_i$ και μοναδιαία χωρητικότητα, έναν αρχικό κόμβο s κι έναν τελικό t. Στη συνέχεια δημιουργώ ακμές μοναδιαίας χωρητικότητας και μηδενικού κόστους από τον αρχικό κόμβο σε κάθε s_i και από κάθε κόμβο t_i στον τελικό. Δημιουργώ επίσης ακμές $t_i \rightarrow s_j,\ \forall t_i, s_j : t_i \leq s_j$ μηδενικού κόστους και μοναδιαίας χωρητικότητας. Χρησιμοποιώ αλγόριθμο που υπολογίζει την ελάχιστη ροή για $flow = k$ και χρησιμοποιώ επαναλαμβανόμενα συντομότερα μονοπάτια αφου δε θέλω ο χρόνος εκτέλεσης να εξαρτάται από την τιμή του p_i.

Σημείωση: Εάν κάποιο s_i ή t_i υπάρχει πάνω από μια φορά θα δημιουργήσω διαφορετικούς κόμβους

Σημείωση: Αν δεν υφίσταται ροή k, οπότε μπορώ να ικανοποιήσω μόνο προσφορές με λιγότερα από k αμάξια σύνολο, θα βρώ πάλι τη βέλτιστη λύση.

Ορθότητα Αλγορίθμου: Οι ακμές $s \rightarrow s_i$ δηλώνονυ ότι ένα αμάξι ξεκινά με οποιαδήποτε προσφορά, οι $t_i \rightarrow t$ ότι μπορεί να μην ξαναχρησιμοποιηθεί μετά τη λήξη την προσφοράς και οι $t_i \rightarrow s_j$ ότι μετά το τέλος κάποιας προσφοράς μπορεί να μεταβεί σε επόμενη που ξεκινά μετά από το τέλος. Στέλνοντας ροή k από το s στο t αναθέτω k αμάξια στις προσφορές. Βρίσκοντας μια ακμή που ελαχιστοποιεί το αρνητικό κόστος μεγιστοποιώ το κέρδος. Αν έχω πολλά ίσα s_i ή t_i τα αντιμετωπίζω ξεχωριστά γιατί αποτελούν διαφορετικές προσφορές και εξυπηρετούν από ένα διαφορετικό όχημα η κάθε μια.

Σημείωση: Οι min cost flow αλγόριθμοι λειτουργούν με αρνητικές τιμές

Πολυπλοκότητα Αλγορίθμου: Για να φτιάξω το γράφο έχω πολυπλοκότητα $0(n + 2n+^2) = O(n^2)$ στη χείριστη περίπτωση, αφού έχω n^2 ακμές και $2n + 2$ κόμβους. Ο min cost flow αλγόριθμος έχει πολυπλοκότητα $O(n^3k)$. Τελικά προκύπτει πολυπλοκότητα $O(n^3k)$. Με τη βελτίωση Edmonds - Karp έχει πολυπλοκότητα $O(kn^2logn)$