Αλγόριθμοι και Πολυπλοκότητα
2η Σειρά Γραπτών Ασκήσεων

Ελισάβετ Παπαδοπούλου,
Α.Μ.: 03120190

Άσκηση 1: Πολύχρωμος Πεζόδρομος

Προκειμένου να υπολογιστεί ο ελάχιστος αριθμός ημερών που απαιτούνται για το βάψιμο, θα χρησιμοποιήσουμε Δυναμικό Προγραμματισμό.
Ορίζουμε δύο δείκτες i, j, οι οποίοι ορίζουν το διάστημα των πλακών τις οποίες επιλέγουμε να βάψουμε κάθε φορά, και c_i το χρώμα που αντιστοιχεί στην πλάκα i. Ορίζουμε επίσης έναν δείκτη k, ο οποίος δείχνει το τέλος του υποδιαστήματος με το οποίο θα ασχοληθούμε στην συνέχεια, δηλαδή κοντινότερη στο j πλάκα εντός του διαστήματος, η οποία έχει ίδιο χρώμα με την i+1. Με το k αυτό θα γίνει η διάσπαση του διαστήματος σε δύο μικρότερα, για τα οποία θα υπολογιστεί αναδρομικά ο ελάχιστος αριθμός ημερών τους.
Η αναδρομική σχέση του αριθμού των ημερών, έστω days(i,j), υπολογίζεται ως η:

$$days(i,j) = \begin{cases} 0, i > j \\ 1, i = j \\ 1 + \min_{i \le k \le j}\{days(i+1,k) + days(k+1,j-1)\}, i < j \end{cases}$$

Για την πολυπλοκότητα του αλγορίθμου, γνωρίζουμε πως:
Ο δείκτης i θα διατρέξει όλες τις πλάκες από την αρχή μέχρι το τέλος.
Ο δείκτης j θα διατρέχει τις πλάκες από το τέλος μέχρι τον i, για κάθε διαφορετική τιμή του i.
Ο δείκτης k θα μπορεί να πάρει για κάθε έναν συνδυασμό i, j όλες τις ενδιάμεσες τιμές τους.
Έτσι, συνολικά, καταλαβαίνουμε πως ο αλγόριθμος μας έχει πολυπλοκότητα $O(n^3)$

Άσκηση 2: String Matching

1. Θα δείξουμε πως υπολογίζοντας τους πυρήνες όλων των προθεμάτων του pt, μπορούμε να ταιριάξουμε το μοτίβο p με το κείμενο t. Για κάθε ένα γράμμα του μοτίβου p που θα προσθέτουμε στο πρόθεμα, ξεκινώντας από την αρχή του, θα αναζητούμε μέσα στο pt την πρώτη του εμφάνιση. Μόλις θα έχουμε προσθέσει όλο το p, η πρώτη του εμφάνιση στην αναζήτηση του πυρήνα θα είναι το ταίριασμά του.

   Ένα παράδειγμα της διαδικασίας αυτής είναι το εξής:
   Έστω $p = abbabba$ και $t = cdabefabbabbahijk$. Τότε, η εύρεση του πυρήνα όλων των προθεμάτων θα ακολουθήσει τα παρακάτω βήματα:

Algorithm 1 FIND-PREFIX(p, t)

Έστω $string = pt$.
Έστω kernel = πίνακας μήκους του string και με μηδενικά σε κάθε κελί.
Θέσε την μεταβλητή l = 0
Όρισε έναν δείκτη pointer = 1.
while $pointer < length(pt)$ **do**
    if $string[pointer] != string[l]$ **then**
        if $l != 0$ **then**
            l = kernel[l-1]
        else
            kernel[pointer] = 0
            pointer ++
        end if
    else
        l ++
        kernel[pointer] = l
        pointer++
    end if
end while

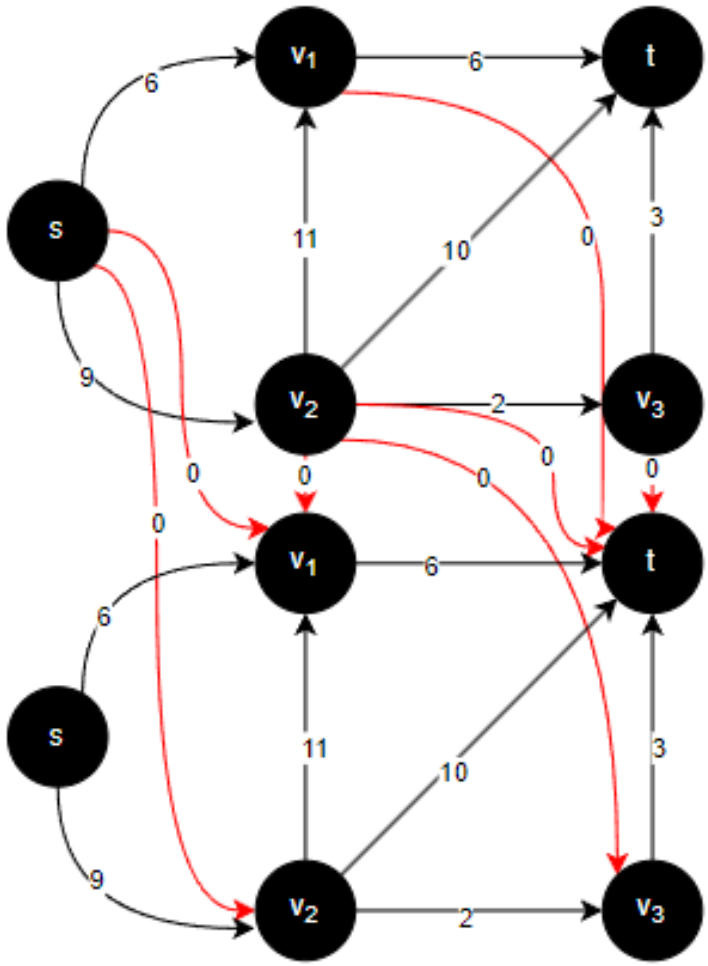
2. Ο k-πυρήνας είναι πράγματι καλά ορισμένος, καθώς:

    - Το k αποτελεί μήκος, και άρα πρόκειται για μεγαλύτερη ή ίση του μηδενός ποσότητα
    - Αφού το $u \prec v$, σημαίνει πως το v είναι της μορφής uwu, όπου w κάποια τυχαία συμβολοσειρά μήκους $|w| \geq 0$. Επομένως, ισχύει πως, σε κάθε περίπτωση, $|u| < |v|$, και άρα αφού το μήκος του u είναι το πολύ k, ισχύει πως $k \leq |v|$.

    Ο αλγόριθμος που επινοήσαμε είναι ο εξής: Για μια δεδομένη συμβολοσειρά string και ένα δοσμένο k, θα πάρουμε τους πρώτους και τους τελευταίους k χαρακτήρες του string και θα τους τοποθετήσουμε σε μια νέα συμβολοσειρά που θα ονομάζεται potential_kernel. Στην συνέχεια, θα εκτελέσουμε τον αλγόριθμο του προηγούμενου ερωτήματος, και θα βρούμε τον μεγαλύτερο δυνατό πυρήνα.

Άσκηση 3: Συντομότερα Μονοπάτια με Συντομεύσεις Ενδιάμεσων Ακμών

1. Ο υπολογισμός του συντομότερου $s - t$ μονοπατιού, δεδομένου πως μπορεί να μηδενιστεί μόνο μια ακμή του, θα γίνει ως εξής:
    Χρησιμοποιούμε έναν επιπλέον, πανομοιότυπο γράφο, με κόμβους w_i τοποθετημένο δίπλα στον κανονικό μας.
    Για κάθε κατευθυνόμενη ακμή $v_i - v_j$ που ανήκει στον αρχικό μας γράφο, δημιουργούμε μια μηδενικού βάρους ακμή $v_i - w_j$, όπου w_j ο "κλώνος" κόμβος του v_j στον παρομοιότυπο νέο γράφο. Προκύπτει έτσι το σχήμα παρακάτω:



Όπως φαίνεται, πλέον έχουν δημιουργηθεί όλα τα δυνατά μονοπάτια από τον s στον t, με μηδενισμένη μόνο μία ακμή τους. Έτσι, με μια απλή εφαρμογή του αλγορίθμου του Dijkstra μπορούμε να βρούμε το βέλτιστο δυνατό μονοπάτι. Η πολυπλοκότητα του αλγορίθμου, δεδομένου πως πλέον περιέχονται οι διπλάσιοι κόμβοι, είναι $O(2|V|log|2V|)$.

2. Για την γενίκευση του προβλήματος σε k μηδενικές ακμές, μπορούμε να εξελίξουμε τον αλγόριθμο του προηγούμενου ερωτήματος, δημιουργώντας k γράφους όπως προηγουμένος, ο καθένας με μηδενικές ακμές προς του κόμβου του επόμενού του. Εφαρμόζοντας ξανά τον αλγόριθμο του Dijktra, η πολυπλοκότητα θα είναι πλέον $O(k|V|log|kV|)$.

Άσκηση 4: Ταξίδι σε Περίοδο Ενεργειακής Κρίσης

1. Αρχικά, έχοντας δοσμένο το μονοπάτι $s-t$ με n κορυφές, απομένει μόνο να προγαρμματίσουμε το βέλτιστο δυνατό πλάνο ανεφοδιασμού, προκειμένου να φτάσουμε στον προορισμό μας με άδειο ντεπόζιτο και το μικρότερο δυνατό κόστος.
   Για την επίλυση του προβλήματος αυτού θα ακολουθήσουμε ορισμένα βήματα:

   - Αρχικά, για κάθε κόμβο i θα βρούμε τον επόμενο του ($next(v_i)$) και τον προηγούμενό του ($prev(v_i)$) ως εξής:
     Ως $prev(v_i)$ ορίζεται ο κόμβος $v_j, j \leq i$, ο οποίος έχει το μικρότερο κόστος βενζίνης ανάμεσα στις πόλεις οι οποίες απέχουν ποσότητα βενίζινης από τον v_i, η οποία δεν υπερβαίνει την χωρητικότητα του ντεπόζιτου B.
     Ως $prev(v_i)$ ορίζεται ο κόμβος $v_j, j > i$, ο οποίος έχει το μικρότερο κόστος βενζίνης ανάμεσα στις

πόλεις οι οποίες απέχουν ποσότητα βενζίνης από τον v_i, η οποία δεν υπερβαίνει την χωρητικότητα του ντεπόζιτου B.
Για κάθε πόλη v_i δηλαδή, θα διατηρούμε ένα διαφορετικό παράθυρο μήκους B, το οποίο θα συμπεριλαμβάνει τις γειτονικές μεγαλύτερες για το next και τις μικρότερες ή ίσες για το prev πόλεις της.

- Οι πόλεις των οποίων ο προηγούμενος είναι ο εαυτός τους θα είναι οι στάσεις της διαδρομής μας, καθώς δεν υπάρχει κάποια προηγούμενή τους πόλη με μικρότερο κόστος βενζίνης.
- Πλέον, το μόνο που απομένει είναι να καταγράψουμε τον ίδιο τον αλγόριθμο.

Algorithm 2 NEXT-STOP(i, k)

Έστω $x = i$.
if $d(x, k) < U$ **then**
  Γέμισε αρκετή βενζίνη για να πας στον k.
else
  Οδήγησε προς τον επόμενο.
  Θέσε $x = \text{next}(x)$.
  Πήγαινε στο βήμα 2.
end if

Ο αλγόριθμος αυτός έχει $O(n \log n)$ πολυπλοκότητα, καθώς για κάθε μία από τις n πόλεις θα γίνουν $\log n$ συγκρίσεις κόστων των γειτονικών της πόλεων.

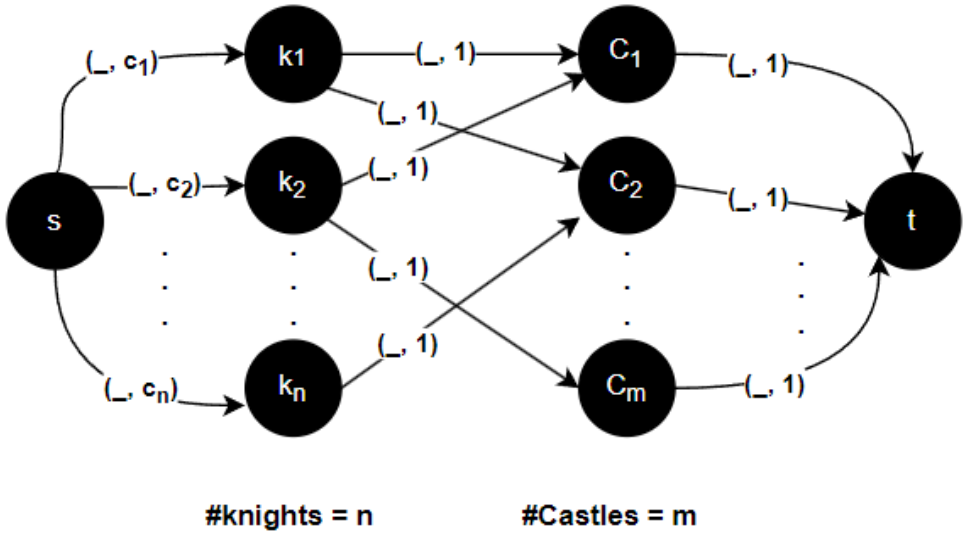
2. Για την γενίκευση του προβλήματος θα εργαστούμε πάνω σε έναν νέο γράφο G_V.
Ο γράφος αυτός θα περιλαμβάνει ως ακμές από κάθε κόμβο i όλες τις πιθανές διαδρομές από κάθε κόμβο σε κάθε άλλον απόστασης σε λίτρα το πολύ B. Το βάρος των ακμών αυτών θα είναι το κόστος c(i) επί το μήκος της διαδρομής. Έτσι, θα έχουμε πλέον έναν γράφο στον οποίο η εφαρμογή του αλγορίθμου του Dijkstra θα προσφέρει το βέλτιστο μονοπάτι s-t.
Η υλοποίηση του γράφου αυτού θα γίνει με την εκτέλεση πολλαπλών αλγορίθμων Dijkstra, οι οποίοι θα είναι μορφοποιημένοι έτσι ώστε να σταματούν με το που ξεπεραστεί το όριο της απόστασης B κατά την εύρεση ενός μονοπατιού από κάθε κόμβο προς τον t. Η πολυπλοκότητα της εκτέλεσης αυτής απαιτεί χρόνο $O(|V| \times (|E| + |V| \log |V|))$

Άσκηση 5: Παιχνίδια Εξουσίας

1. Το πρόβλημα αυτό πρόκειται για το πρόβλημα του Bipartite Matching, ανάμεσα στα σύνολα Ιπποτών και των Καστρών. Στόχος είναι η ανάθεση κάθε κάστρου υπό την εποπτεία ενός ιππότη, ο οποίος εμπεριέχεται στην λίστα των αποδεκτών για το δεδομένο κάστρο διοικητών.
Θα δημιουργήσουμε έναν γράφο, ο οποίος θα ανάγει το πρόβλημά μας σε πρόβλημα max flow:

   - Κάθε ιππότης (knight k_i) και κάθε κάστρο (Castle, C_j) θα αποτελεί έναν κόμβο.
   - Κόμβοι θα είναι επίσης μια αρχή s και ένα τέλος t.
   - Δημιουργούνται κατευθυνόμενες ακμές από το s σε κάθε κόμβο ιππότη (Knight, k_i), με $capacity = c_i$, τον μέγιστο αριθμό καστρών που μπορεί να έχει κάθε ιππότης i υπό την εποπτεία του.
   - Δημιουργούνται επίσης κατευθυνόμενες ακμές από τους ιππότες που ανήκουν σε κάθε K_j, στο κάστρο j που αναλογούν (δηλαδή συνδέονται όλοι οι ιππότες με τα κάστρα που τους δέχονται σαν υπεύθυνους), και σε κάθε μία από τις ακμές αυτές θέτουμε capacity = 1, με το flow τους να δηλώνει δηλαδή αν θα αναλάβουν την επιτήρηση του κάστρου ή όχι.
   - Τέλος, σχηματίζονται κατευθυνόμενες ακμές από όλα τα κάστρα προς το τέλος t. Το capacity των ακμών αυτών είναι παντού 1, εξασφαλίζοντας πως κάθε κάστρο μπορεί να το αναλάβει το πολύ ένας ιππότης.

Ο γράφος θα μοιάζει κάπως έτσι:



Για την επίλυση του προβλήματος, θα προσπαθήσουμε να βρούμε την μέγιστη δυνατή ροή του γράφου. Προφανώς, ο αριθμός των επιτρεπόμενων ακμών που θα συνδέεουν κάποιο κάστρο j με κάποιον ιππότη δεν μπορεί να υπερβαίνει το 1, δηλαδή ένα κάστρο δεν μπορεί να έχει παραπάνω από έναν υπεύθυνο. Αυτό διασφαλίζεται από το capacity των ακμών.
Με χρήση, στην συνέχεια, του αλγορίθμου Edmonds-Carp, επιλύουμε το πρόβλημα max flow, το οποίο τυχαίνει να συμπίπτει και με το δεδομένο μας πρόβλημα του max partite.
Ξεκινώντας δηλαδή με flow = 0, και για όσο συνεχίζει να υπάρχει κάποιο επαυξητικό μονοπάτι από το s στο t:

- Εκτελούμε τον BFS, στο υπολοιπόμενο δίκτυο για να βρούμε κάποιο επαυξητικό μονοπάτι
- Υπολογίζουμε την ελάχιστη υπολοιπόμενη χωρητικότητα μεταξύ των ακμών του μονοπατιού αυτού
- Αυξάνουμε το flow του μονοπατιού αυτού κατά αυτή την τιμή.

Στο τέλος, επιστρέφουμε το maximum flow.
Σε περίπτωση που δεν υπάρχει ανάθεση που ικανοποιεί όλους τους περιορισμούς, ο αλγόριθμος θα επιστρέψει 0 στην ακμή που συνδέει την αρχή με κάποιο κάστρο, δείχνοντας πως για καμία ανάθεση δεν μπορεί να υπάρξει υπεύθυνος για κάθε κάστρο.
Η πολυπλοκότητα του αλγορίθμου αυτού είναι $O(|V| \times |E|^2)$, με τον αριθμό των ακμών να μην υπερβαίνει τα:

$$|E| = |knights| + |Castles| + \sum_{i=0}^{|knights|} c_i = n + m + n \times m$$

Ταυτόχρονα, ο αριθμός των κόμβων θα είναι:

$$|V| = n + m + 2$$

Άρα τελικά, η πολυπλοκότητα του αλγορίθμου είναι $O(n^3m^2 + n^2m^3)$

2. Ένας απλός τρόπος να αποδείξουμε πως ο αλγόριθμος πράγματι τερματίζει με το μέγιστο flow, είναι ο εξής:

- Αρχικά, υποθέτουμε πως ο αλγόριθμος τερμάτισε με ένα flow το οποίο δεν είναι το μέγιστο.
- Στην περίπτωση που ισχύει αυτό, σημαίνει πως υπάρχει κάποιο μονοπάτι μέσα στο residual network το οποίο οδηγεί από τον s στον t, δηλαδή ένα επαυξητικό μονοπάτι.

- Αν ωστόσο ίσχυε αυτό, ο αλγόριθμος δεν θα είχε τερματίσει. Άρα η εκτέλεση του αλγορίθμου ήταν σωστή και απέδωσε την μεγαλύτερη δυνατή ροή.

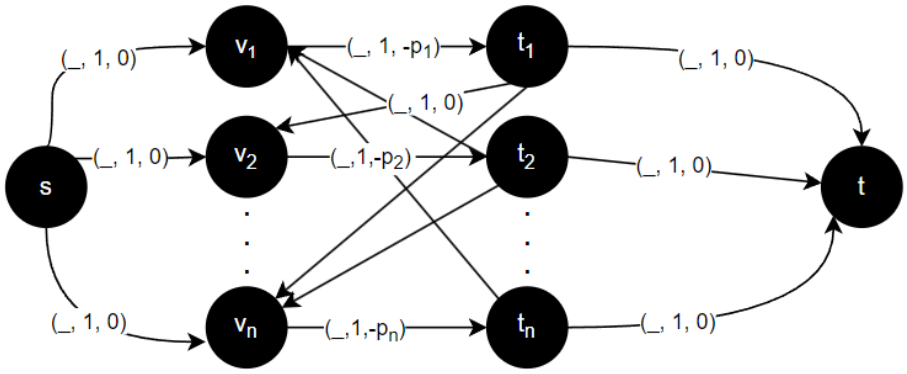
3. Προκειμένου να μπορέσει ο Βασιλιάς να εξορίσει το ελάχιστο δυνατό πλήθος ιπποτών για να εξαλειφθεί η απειλή συγκρούσεων, μπορεί αν ανάγει το πρόβλημά του στο πρόβλημα του Minimum Vertex Cover.
Ορίζουμε έναν γράφο, ο οποίος ενώνει όλα τα ζεύγη ιπποτών που συγκρούονται με ακμές. Στόχος μας είναι να βρεθεί το ελάχιστο σύνολο κόμβων, το οποίο αν αφαιρεθεί από τον γράφο τότε εκείνος θα πάψει να έχει ενωμένους κόμβους. Το πρόβλημα αυτό πρόκειται για το πρόβλημα του Minimum Vertex Cover, δηλαδή το πρόβλημα της εύρεσης ενός υποσυνόλου των κόμβων ενός γράφου, τέτοιου ώστε για κάθε ακμή (u,v) που ανήκει στονγράφο, είτε η u είτε η v να ανήκουν στο Vertex Cover.
Επειδή ο γράφος μας χωρίζεται σε δύο ομάδες ιπποτών, οι οποίες διαχωρίζουν δύο "στρατόπεδα" ιπποτών, ο γράφος αυτός είναι bipartite. Επομένως, το πρόβλημα του Minimum Vertex Cover μπορεί να λυθεί σε πολυωνυμικό χρόνο.

- Αρχικά, θα πρέπει να βρούμε το maximum matching του γράφου, είτε με τον αλγόριθμο του Dinic είτε με του Edmond-Carp, έστω M.
- Στην συνέχεια, βρίσκουμε το πλήθος των κόμβων που δεν συνδέονται με καμία ακμή και ανήκουν στους Πράσινους.
- Ορίζω ως A το σύνολο των κόμβων που είτε ανήκουν στο U είτε συνδέονται στο U με μονοπάτια που εναλλάσσονται ανάμεσα σε κόμβους που ανήκουν και όχι στο M.
- Το σύνολο $K = (\Pi\rho\alpha\sigma\iota\nu o\iota \setminus A) \cup (B\varepsilon\nu\varepsilon\tau o\iota \cap A)$ είναι το Minimum Vertex Cover.

Άσκηση 6: Ενοικίαση Αυτοκινήτων

Το πρόβλημα θα λυθεί μέσω της δημιουργίας ενός γράφου και της αναγωγής στο πρόβλημα του min cost flow. Ο γράφος θα σχηματιστεί ως εξής:

- Κάθε υπαρκτή αρχή και τέλος προσφοράς θα αποτελούν από έναν κόμβο. Θα σχηματιστούν επίσης ένας κόμβος αρχής s και ένας τέλους t.
- Οι προσφορές θα συμβολιστούν από τις κατευθυνόμενες ακμές ανάμεσα στις ανάλογες χρονικές στιγμές δέσμευσης και αποδέσμευσής τους, με τα κόστη των ακμών να αποτελούν τα αντίστοιχα αντίτιμά τους $-p_i$ και τις χωρητικότητές τους ίσες με 1.
- Ακμές θα δημιουργηθούν επίσης από τις χρονικές στιγμές αποδέσμευσης μιας προσφοράς t_i, σε όλες τις πιθανές αρχές s_j με $j > i$. κόστους 0 και χωρητικότητας 1.
- Ακμές θα δημιουργηθούν και από την αρχή s σε όλες τις πιθανές δεσμεύσεις s_i, κόστους 0 και χωρητικότητας 1.



Έχοντας διαμορφώσει τον γράφο, αναζητούμε πλέον ροή τιμής k και ελαχίστου κόστους μεταξύ των κόμβων s-t. Η χωρητικότητα των ακμών είναι 1 γιατί θεωρούμε πως η ροή μας θα διαμοιραστεί ανάμεσα στις k διαφορετικές ταυτόχρονες προσφορές ή ακολουθίες προσφορών που θα επιλέξουμε να αναλάβει η εταιρία.
Για καθένα από τα k αμάξια υπάρχει η δυνατότητα, αφού τελειώσει η προσφορά στην οποία το παρέχουμε, είτε να το αποδεσμεύσουμε και να το στείλουμε στο τέλος t, είτε να το παρέχουμε σε κάποια άλλη προσφορά. Τα κόστη είναι αρνητικές τιμές πάνω στον γράφο, και άρα το μικρότερο κόστος συνεπάγεται το μεγαλύτερο δυνατό κέρδος.
Όσο για την πολυπλοκότητα του αλγορίθμου, το γράφημα θα έχει 2n κόμβους, όπου n οι προσφορές. Οι ακμές του θα είναι το πολύ 3n, δηλαδή οι ακμές που συνδέουν τα t_i, t, τα s, s_i, και τα t_i, s_j. Όπως είναι γνωστό από Erickson, το πρόβλημα του min cost flow μπορεί να λυθεί σε πολυπλοκότητα τάξεως $O(E^2 \log^2 V)$, και επομένως, η πολυπλοκότητα του αλγορίθμου θα είναι $O(n^2 \log^2(2n))$.