2η Σειρά Γραπτών Ασκήσεων

Σωτήρου Θεόδωρος 03118209

March 12, 2023

Άσκηση 1: Πολύχρωμος Πεζόδρομος

Δοσμένων n πλακών αρχικά λευκές και την ακολουθία $(c_1, ..., c_n)$ με τα επιθυμητά χρώματα, διατυπώνουμε αλγόριθμο που να υπολογίζει το ελάχιστο πλήθος ημερών που απαιτούνται για να χρωματισούν όλες οι πλάκες δεδομένου οτι κάθε μέρα μπορούν να χρωματιστούν μόνο διαδοχικές πλάκες με το ίδιο χρώμα. Για την επίλυση του προβλήματος θα βάφουμε κάθε φορά το μεγαλύτερο διάστημα που ξεκινάει και τελειώνει με το ίδιο χρώμα, επαναλαμβάνοντας την διαδικασία στο εσωτερικό του διατήματος. Όταν πάψουν να υπάρχουν διαστήματα βάφουμε τις πλάκες που έμειναν και συνεχίζουμε στο επόμενο μεγαλύτερο διάστημα. Η πολυπλοκότητα του αλγορίθμου είναι $O(n^3)$. Αναλυτικότερα:

1. Μέχρι να χρωματιστούν σωστά όλες οι πλάκες

    1.1 Αν υπάρχουν διαστήματα λάθος χρωματισμένων πλακών που ξεκινάνε και τελειώνουν με το ίδιο χρώμα:

    - 1.1.1 Βρές το μεγαλύτερο απο αυτά και χρωμάτισέ το με αυτό το χρώμα και
    - 1.1.2 πήγαινε στο 1 με τις πλάκες στο εσωτερικό του διστήματος αυτού.

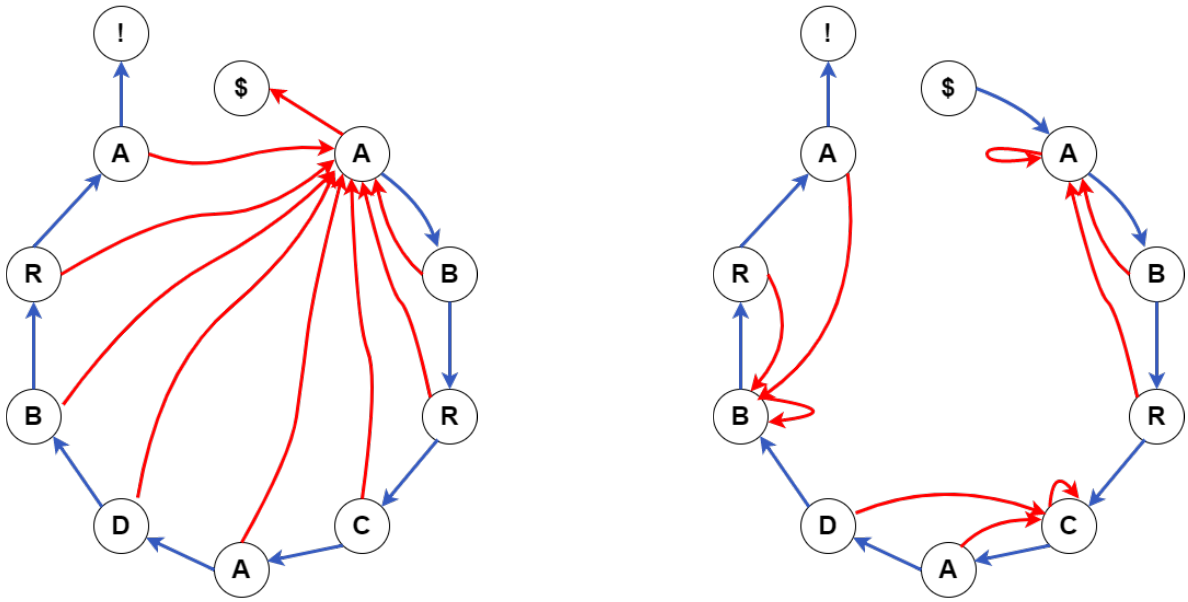
    1.1 Αν δεν υπάρχουν διαστήματα λάθος χρωματισμένων πλακών:

    - 1.1.1 Χρωμάτισε την κάθε λανθασμένα χρωματισμένη πλάκα ξεχωριστά με το σωστό χρώμα.

Άσκηση 2: String Matching

Σε αυτήν την άσκηση καλούμαστε να περιγράψουμε μια τροποποίηση του αλγορίθμου KMP (Knut Morris Pratt) στην οποία το μοτίβο μπορεί να περιέχει οποιοδήποτε αριθμό συμβόλων μπαλαντέρ *, καθένα από τα οποία ταιριάζει με μια αυθαίρετη συμβολοσειρά. Εύκολα διαπιστώνουμε οτι το σύμβολο * χωρίζει το μοτίβο μας σε άλλα μικρότερα τα οποία πρέπει να αναζητήσουμε στο κείμενο με την μόνη προϋπόθεση να εμφανίζονται με την ίδια σειρά. Παρακάτω φαίνεται ένα παράδειγμα με Finite State Machine για τις συμβολοσειρές "ABRCADBRA" και "ABR*CAD*BRA" οπου η κάθε κατάσταση συμβολίζει την αναζήτηση στο κείμενο εκείνου του συμβόλου, με μπλέ χρώμα η επιτυχία της αναζήτησης και με κόκκινο η αποτυχία.

Για την λύση του προβλήματος αρκέι να τροποποιήσουμε την failure function ώστε όταν προκύπτει κάποια αστοχία μελετάμε ξανά μόνο το "υπο-μοτίβο" που βρισκόμαστε, δηλαδή θα πρέπει να μετατοπιστούμε το πολύ μέχρι το προηγούμενο "*". Παρακάτω Φαίνονται ο αλγόριθμος ComputeFailure(P[1...m]), όπως παρουσιάζεται στις διαφάνειες του μαθήματος και ο ComputeFailure*(P[1...m]) που τροποποιεί τον KMP με βάση τις ανάγκες της άσκησης. Εισάγοντας την μεταβλητή start μπορούμε κάθε φορά που φτάνουμε στον μπαλαντέρ να αγνοούμε το προηγούμενο μέρος του pattern.



(a) "ABRCADBRA"   (b) "ABR*CAD*BRA"

Figure 1: Finite State Machine for strings:

Algorithm 1 ComputeFailure(P[1...m])

```
j ← 0
for i ← 1, m do
    fail[i] ← j
    while j > 0 and P[i] ≠ P[j]  do
        j ← fail[j]
    end while
end for
j ← j+1
```

Algorithm 2 ComputeFailure*(P[1...m])

```
start ← 0
j ← 0
for i ← 1, m do
    fail[i] ← j
    if P[i] = " * " then
        start ← i
    end if
    while j > start and P[i] ≠ P[j]  do
        j ← fail[j]
    end while
end for
j ← j+1
```

Άσκηση 3: Συντομότερα Μονοπάτια με Συντομεύσεις Ενδιάμεσων Ακμών

Σε αυτό το πρόβλημα θέλουμε να ελαχιστοποιήσουμε την απόσταση μεταξύ δύο κορυφών ενός κατευθυνόμενου γράφου G(V,E,$\vec{w}$) με n κορυφές, όταν μπορούμε να μηδενίσουμε το μήκος το πολύ k ακμών στο μονοπάτι που θα χρησιμοποιήσουμε. Αρχικά μελετάμε και διατυπόνουμε αποδοτικό αλγόριθμο όταν μπορούμε να μηδενίσουμε το μήκος μίας μόνο ακμής του μονοπατιού και έπειτα K ακμές του γράφου.

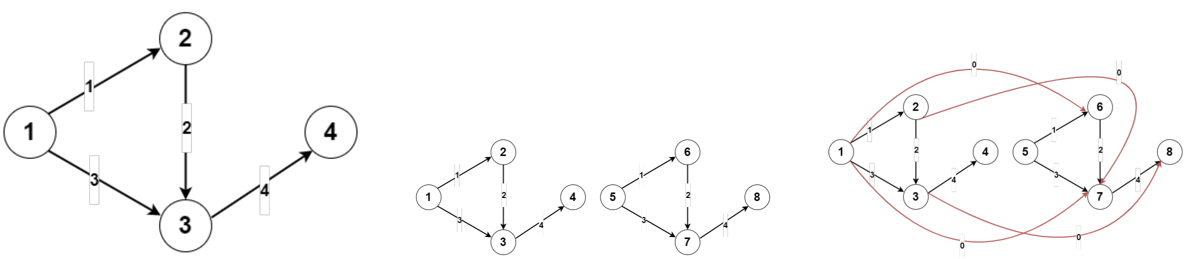
1. Ο αλγόριθμός που λύνει το πρόβλημα για K=1 φαίνεται παρακάτω. Χωρίς βλάβη της γενικότητας θεωρούμε οτι το σύνολο V περιέχει ακέραιους αριθμούς, ενώ το E περιέχει τούπλες ακεραίων (π.χ. G([0,1,2],[(0,1), (1,2)], [5,10]).

Algorithm 3 ShortestShortcut1(G(V,E,W))

$G_n(V_n, E_n, W_n) \leftarrow G(V, E, W)$
$n \leftarrow$ length(V)
for each $v \in V$ **do**
  $V_n.add(v + n)$
end for
for each $(i, (h, t)) \in enumerate(E)$ **do**
  $E_n.add((h + n, t + n))$
  $W_n.add(W[i])$
  $E_n.add((h, t + n))$
  $W_n.add(0)$
end for
return $Dijkstra(G_n(V_n, E_n, W_n))$

Διαισθητικά ο παραπάνω αλγόριθμος δημιουργεί δύο γράφους ίδιους με τον αρχικό, με ίδια βάρη ακμών, και συνδέει τον πρώτο με τον δεύτερο εισάγοντας ακμές μηδενικού βάρους για κάθε ακμή του πρώτου με αφετηρία την αφετηρία της ήδη υπάρχουσας ακμής και κατάληξη την κατάληξη της ακμής αλλά στον δεύτερο γράφο. Παρακάτω φαίνεται μια αναπαράσταση αυτής της διαδικασίας για τον γράφο G([1,2,3,4], [(1,2),(1,3),(2,3),(3,4)], [1,3,2,4]).

Τελικά στον νέο γράφο εκτελούμε τον αλγόριθμο του Dijkstra και βρίσκουμε το μήκος του συντομότερου μονοπατιού απο τον πρώτο έως τον τελευταίο κόμβο. Ο αλγόριθμος βασίζεται στο γεγονός οτι οταν επιλεγεί μια ακμή μηδενικού βάρους απο τον Dijkstra δεν μπορεί να επιλεγεί κάποια άλλη. Αφού ο Dijkstra βρίσκει το συντομότερο μονοπάτι είμαστε σίγουροι οτι ο αλγόριθμός μας θα βρεί την καλύτερη λύση.



(a) Αρχικός γράφος (b) Δημιουργία νέου γράφου (c) Τελικός γράφος

Figure 2: Δημιουργία νέου γράφου

2. Μετά την μελέτη του πρώτου ερωτήματος μπορούμε να γενικεύσουμε την λύση για $k \geq 1$. Συγκεκριμένα θα δημιουργούμε k νέα γραφήματα, όμοια με το αρχικό τα οποία θα είναι ενωμένα μεταξύ τους ανα δύο με τον τρόπο που παρουσιάστηκε στο ερώτημα 1. Κατα την εκτέλεση του Dijkstra η μετάβαση απο το ένα γράφημα στο άλλο συμβολίζει τον μηδενισμό μίας ακμής απο τις k δυνατές. Είναι σίγουρο οτι θα χρησιμοποιηθούν όλες οι k συντομεύσεις. Η πολυπλοκότητα του αλγορίθμου θα είναι $O(k(N + M)\log(N + M))$ με N και M το πλήθος των κορυφών και των ακμών του αρχικού γράφου. Ο αλγόριθμος φαίνεται παρακάτω.

Algorithm 4 ShortestShortcut(G(V,E,W), k)

$G_n(V_n, E_n, W_n) \leftarrow G(V, E, W)$
$n \leftarrow$ length(V)
for each $i \in \{1, \dots, k\}$ **do**
    for each $v \in V$ **do**
        $V_n.add(v + k * n)$
    end for
    for each $(i, (h, t)) \in enumerate(E)$ **do**
        $E_n.add((h + k * n, t + k * n))$
        $W_n.add(W[i])$
        $E_n.add((h, t + k * n))$
        $W_n.add(0)$
    end for
end for
return $Dijkstra(G_n(V_n, E_n, W_n))$

Άσκηση 4: Σύντομα Μονοπάτια με Επαρκή Μεταφορική Ικανότητα

Σε αυτό το πρόβλημα θέλουμε να υπολογίσουμε ένα μονοπάτι p ενός γράφου που ελαχιστοποιεί τον λόγο $c(p)/b(p)$, με $c(p)$ το άθροισμα του κόστους των ακμών του p και $b(p)$ η ελάχιστη μεταφορική ικανότητα του μονοπατιού. Στην ουσία αλλάζουμε την μετρική για τον υπολογισμό σύντομων μονοπατιών και τροποποιήουμε κάποιον αλγόριθμο όπως ο Bellman-Ford ή ο Dijkstra

1. Θα δείξουμε με αντιπαράδειγμα οτι το βέλτιστο μονοπάτι ώς προς τον λόγο αυτό μπορεί να μήν είναι βέλτιστο μονοπάτι ως προς το κόστος. Αναλυτικά στον γράφο του παρακάτω σχήματος με κορυφές s, v_1 και t και ακμές τις $s - v_1$, $v_1 - t$ και $s - t$ με κόστος και μεταφορική ικανότητα (10,100), (10,1000) και (10,25) αντίστοιχα για κάθε ακμή το βέλτιστο μονοπάτι ανάμεσα στους s και t με βάση τον λόγο είναι το $s - v_1 - t$ (με λόγο $\frac{c(s-v_1-t)}{b(s-v_1-t)} = \frac{10+10}{\min(100,1000)} = \frac{1}{5} < \frac{2}{5} = \frac{c(s-t)}{b(s-t)}$) ενώ το βέλτιστο μονοπάτι είναι το s-t.

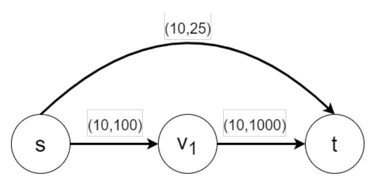


Figure 3: Αντιπαράδειγμα

2. Θα διτυπώσουμε μια αναδρομική σχέση που επιτρέπει τον υπλογισμό ενός $s - t$ μονοπατιού με βέλτιστο λόγο κόστους προς μεταφορική ικανότητα. Ορίζουμε $D(u, i)$ το μήκος συντομότερου $s - u$ μονοπατιού με το πολύ i ακμές. Αρχικά $D(s, 0) = 0$ και $D(u, 0) = \infty$ για κάθε $u \neq s$. Η αναδρομική σχέση θα είναι:

$$D(u, i) = \min\{D(u, i-1), \min_{v:(v,u)\in E}\{(D(v, i-1)b(v) + w(v, u))/b(u)\}\}$$

3. Θα χτίσουμε την αναδρομική σχέση του προηγούμενου ερωτήματος ώστε να διατυπώσουμε αποδοτικό αλγόριθμο για τον υπολογισμό του μονοπατιού. Στην εκφώνηση δεν υπάρχουν πληροφορίες για το αν ο γράφος είναι ακυκλικός ούτε αν περιέχει ακμές αρνητικού κόστους. Για αυτόν το λόγο επιλέγουμε να τροποποιήσουμε τον αλγόριθμο Bellman-Ford. Παρακάτω φαίνεται ο τροποποιημένος αλγόριθμος CappedBellmanFord, με $w(v, u)$ το κόστος και $cap(v, u)$ την μεταφορική ικανότητα της ακμής μεταξύ των κόμβων $v - u$

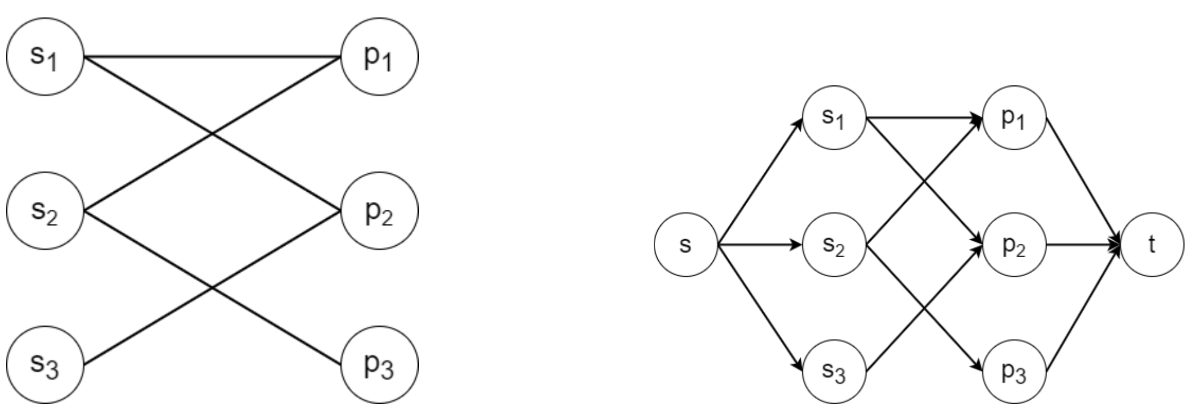
Algorithm 5 CappedBellman-Ford(G(V,E,W), s)

for each $u \in V$ **do**
  $D[u] \leftarrow \infty; p[u] \leftarrow NULL; b[u] \leftarrow \infty$
end for
$D[s] \leftarrow 0;$
for $i \leftarrow 1$ to $n-1$ **do**
  for each $(v,u) \in E$ **do**
    $mincap \leftarrow \min\{b[v], cap(v,u)\}$
    if $D[u] > (D[v]b[v] + w(v,u))/mincap$ **then**
      $D[u] \leftarrow D[v] + w(v,u);$
      $p[u] \leftarrow v;$
      $b[u] \leftarrow mincap;$
    end if
  end for
end for
for each $(v,u) \in E$ **do**
  $mincap \leftarrow \min\{b[v], cap(v,u)\}$
  if $D[u] > (D[v]b[v] + w(v,u))/mincap$ **then**
    return (NEG-CYCLE);
  end if
end for

Άσκηση 5: Αγορές Προϊόντων από Συγκεκριμένα Καταστήματα

Μας δίνεται ένα σύνολο καταστημάτων $S = s_1, ..., s_m$ και ένα σύνολο προϊόντω $P = p_1, ..., p_n$, όπου κάθε προϊόν πωλείται απο συγκεκριμένο υποσύνολο καταστημάτων.

1. Καλούμαστε να διατυπώσουμε αποδοτικό αλγόριθμο που να υπολογίζει ένα σύνολο καταστημάτων απο τα οποία μπορούμε να προμηθευτούμε όλα τα προϊόντα, αν αγοράζουμε ακριβώς ένα πορϊόν απο κάθε κατάστημα. Αναπριστόντας τα καταστήματα s_i και ταποριόντα p_i ώς κόμβους ενός γράφου με τις ακμές ανάμεσα στα καταστήματα και τα προϊόντα $s_k - p_l$ να συμβολίζουν την ύπαρξη του προϊόντως p_l στο κατάστημα s_k, καταφέρνουμε να δημιουργήσουμε ένα Διμερές γράφημα. Καλούμαστε τώρα να υπολογίσουμε τον μέγιστο αριθμό ακμών χωρίς κοινά άκρα σε αυτό το γράφημα (Maximum Matching). Θα αναγάγουμε το πρόβλημα του μέγιστου ταιριάσματος σε αυτό της μέγιστης ροής εισάγοντας στον γράφο μας δύο κόμβους s και t. Ο πρώτος κόμβος θα συνδέεται με κάθε κατάστημα με κατευθυνόμενη ακμή χωρητικότητας 1 και αντίστοιχα το κάθε προϊόν συνδέεται με τον τελικό κόμβο με κατευθυνόμενη ακμή χβρητικότητας 1. Οι ήδη υπάρχουσες ακμές μετατρέπονται σε κατευθυνόμεςνες, χωρητικότητας 1, με μροσανατολισμό απο το κατάστημα στο προϊόν. Στο παράδειγμα παρακάτω φαίνονται τα σύνολα καταστημάτων και προϊόντων $S = s_1, s_2, s_3$ και $P = p_1, p_2, p_3$ με το p_1 να πωλειται απο τα s_1, s_2 το p_2 απο τα s_1, s_3 και το p_3 μόνο απο το s_2.



(a) Διμερής γράφος   (b) Αναγωγή σε Max Flow

Figure 4: Παράδειγμα καταστημάτων-προϊόντων

Εκτελώντας τώρα οποιονδήποτε αλγόριθμο επίλυσης του Max Flow στον γράφο μας έχουμε τις εξής περιπτώσεις: Η μέγιστη ροή να είναι μικρότερη απο τον αριθμό των προϊόντων που πρέπει να αγοράσουμε και άρα να μην υπάρχει δυνατό σύνολο καταστημάτων ή η μέγιστη ροή να είναι ίση με τον αριθμό των προϊόντων που θέλουμε και το ζητούμενο σύνολο καταστημάτων θα είναι τα καταστήματα που συνδέονται με τον κόμβο s με ροή 1. Η πολυπλοκότητα διαφέρει ανάλογα με τον αλγόριθμο που θα χρησιμοποιήσουμε. Για παράδειγμα αν διαλέξουμε τον Ford-Fulkerson η πολυπλοκότητα είναι $O(n|E|)$ οπου $|E|$ το πλήθος των ακμών του γράφου και n το σύνολο των προϊόντων.

2. Τώρα αναζητούμε ενα μέγιστο σύνολο προϊόντων και καταστημάτων τέτοιο ώστε κανένα από τα προϊόντα του συνόλου να μην πωλείται απο κανένα κατάστημα του συνόλου. Δηλαδή αναζητούμε το μέγιστο ανεξάρτητο σύνολο του διμερή γράφου. Γνωρίζουμε οτι το συμπλήρωμα του μέγιστου ανεξάρτητου συνόλου ενός γράφου είναι το ελάχιστο κάλυμα (minimum vertex cover problem). Άρα χρησιμοποιώντας το θεώρημα του Kőnig[1] και τον αλγόριθμο του παραπάνω ερωτήματος μπορούμε να βρούμε το σύνολο που ζητείται.

Άσκηση 6: Ενοικίαση Αυτοκινήτων

Διαθέτοντας k ίδια αυτοκίνητα προς ενοικίαση και δεδομένου n προσφορών με κάθε προσφορά i να δεσμεύει ένα όχημα για το χρονικό διάστημα $[s_i, t_i)$ με αντίτιμο p_i θέλουμε να μεγιστοποιήσουμε το συνολικό ποσό που θα εισπράξουμε ως αντίτιμο. Η λύση του προβλήματος ανάγεται στην εύρεση μιας ροής ελαχίστου κόστους σε ένα κατάλληλο διαμορφωμένο δίκτυο G(V,E) που αναπαριστά τους περιορισμούς του προβλήματος. Το δίκτυο G κατασκευάζεται ως εξής:

1. Θεωρούμε μια αρχική κορυφή s, και μια καταληκτική κορυφή t και ένα σύνολο κορυφών v_1 που περιέχει μια κορυφή για κάθε προσφορά. Το σύνολο των κορυφών του δικτύου είναι V = n + 2.

2. Η κορυφή s συνδέεται με ακμή χωρητικότητας 1 με κάθε κορυφή $j \in V_1$.

3. Η κάθε κορυφή $j \in V_1$ συνδέεται με ακμή μοναδιαίας χωρητικότητας και κόστους $-p_j$ με την κορυφή t και με κάθε άλλη κορυφή του $i \in V_1$ ώστε να ισχύει $t_j \leq s_i$ με $i \neq j$.

4. Η κορυφή t συνδέεται με την s με ακμή χωρητικότητας k.

Δηλαδή δημιουργούμε έναν γράφο με μέγιστη ροή s-t όσα και τα αυτοκίνητα που διαθέτουμε. Άν κάποια προσφορά j τελειώνει πριν ξεκινήσει κάποια άλλη i μπορεί να χρησιμοποιηθεί το ίδιο αμάξι με την προηγούμενη. Στον γράφο αυτά τα μη επικαλυπτόμενα χρονικά διαστήματα μεταξύ των προσφορών απεικονίζονται με κατευθυνόμενη ακμή $j - i$ κόστους p_j. Ο αλγόριθμος ελαχίστου κόστους θα επιλέξει τα k φθινότερα μονοπάτια του κατευθυνόμενου ακυκλικού γράφου που δημιουργήσαμε. Η πολυπλοκότητα δημιουργίας του γράφου είναι O(nlogn)(O(nlogn) για να αποφασίσουμε ποιές προσφορές δεν έχουν επικαλυπτόμενα χρονικά διαστήματα και O(n+m) με m ¡ 3n για την κατασκευή του γράφου). Για τον αλγόριθμο Min Cost Flow η πολυπλοκότητα θα είναι $O(n^2 * m * log(n * C))$[2] οπου C είναι η τιμή του μικρότερου κόστους μονοπατιού στο γράφημα.

References

[1] Wikipedia contributors. Kőnig's theorem (graph theory) — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=K%C5%91nig's%20theorem%20(graph%20theory)&oldid=1121876491, 2022.

[2] Google. C++ reference: Min cost flow. https://developers.google.com/optimization/reference/graph/min_cost_flow, 2022.