Αλγόριθμοι & Πολυπλοκότητα

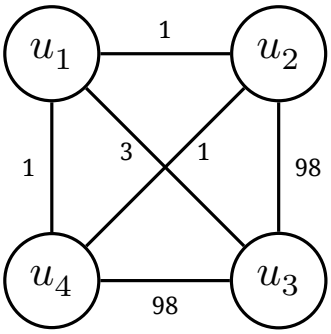
3η σειρά γραπτών ασκήσεων

---

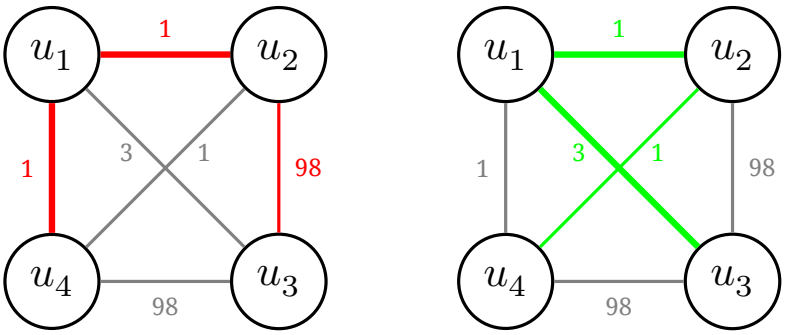
hungrydino ΣΗΜΜΥ 7ο εξάμηνο

Άσκηση 1: Ελάχιστο Συνδετικό Δέντρο με Περιορισμους

α) Έστω ότι θέλουμε να βρούμε το ΕΣΔ $T^*(u_1, 2)$ στο παρακάτω γράφημα:



Τότε στην άπληστη στρατηγική στο ΕΣΔ θα υπάρχουν οι ακμές $\{u_1, u_2\}$ και $\{u_1, u_4\}$. Στο ΕΣΔ αυτό αναγκαστικά θα υπάρχει μια εκ των $\{u_2, u_4\}$ και $\{u_3, u_4\}$, όπως φαίνεται παρακάτω στο αριστερά σχήμα. Η άπληστη στρατηγική δηλαδή παράγει ΕΣΔ με κόστος 100, ενώ μπορούμε να κατασκευάσουμε δέντρο με βαθμό 2 για τον κόμβο u_1 και συνολικό κόστος 5. Μάλιστα θα είναι και βέλτιστο (το ζητούμενο ΕΣΔ $T^*(u_1, 2)$). Αυτό φαίνεται κάτω δεξιά, επιλέγοντας την ακμή $\{u_1, u_3\}$ (η οποία αντιτίθεται στην άπληστη στρατηγική) και την $\{u_1, u_2\}$.



β) Ο αλγόριθμος για την εύρεση του $T^*(s, k)$ είναι ο εξης:

- Κατασκευάζουμε το Ελάχιστο Συνδετικό Δάσος του γράφου $G' = G \backslash \{s\}$ με μια απλή εφαρμογή του αλγόριθμου $Kruskal$, χρόνος $O(|E|log|V|)$.

- Ενώνουμε την s με τις συνεκτικές συνιστώσες (έστω d) του συνδετικού δάσους, χρησιμοποιώντας τις ελαφρότερες ακμές για την κάθε συνιστώσα. Αυτό μπορεί να γίνει σε

χρόνο $O(deg(s) \cdot (logdeg(s) + logd)) = O(|V|log|V|)$, ταξινομώντας τις ακμές του s σε φθίνουσα σειρά και ελέγχοντας κάθε φορά αν μια νέα ακμή ενώνει το s με νέα συνεκτική συνιστώσα (χρόνος $O(logd)$ με δομή $DisjointSet$).
Ισχύει: $d \leq deg(s)$. Αυτό ισχύει επειδή ο γράφος είναι συνεκτικός και ο αλγόριθμος $Kruskal$ δημιούργησε d συνεκτικές συνιστώσες, άρα για να "ενωθούν" σε μία είναι αναγκαίες οι ακμές που αγνοήσαμε: αυτές που προσπίπτουν στην κορυφή u. Επομένως αν $d = k$ τότε τελειώσαμε.

- Έστω ότι $d < k$, τότε πρέπει να προσθέσουμε $k - d$ ακμές της s. Επειδή όμως έχουμε ήδη συνδετικό δέντρο θα δημιουργηθούν κύκλοι. Επιλέγουμε κάθε φορά την ελαφρύτερη ακμή και την προσθέτουμε στο δέντρο. Με μια τροποποίηση του DFS μπορούμε να εντοπίσουμε τον κύκλο που δημιουργείται και να αφαιρέσουμε την βαρύτερη ακμή (εξαιρώντας την νεο-προστεθείσα). Αυτό γίνεται σε χρόνο $O(|V|+|E|)$. Συνολικός χρόνος για αυτό το βήμα: $O((k-d)(|V|+|E|) = \mathrm{O}(k(|V|+|E|))$.

Η πολυπλοκότητα του αλγορίθμου είναι $O((|V|+|E|)(k+log|V|))$.
Απόδειξη ορθότητας:

- Όπως αποδείξαμε, αν ο αλγόριθμος $Kruskal$ εφαρμοζόμενος στον γράφο $G\backslash\{s\}$ δημιουργήσει συνδετικό δάσος d συνιστωσών, τότε η κορυφή s θα πρέπει να έχει τουλάχιστον d ακμές οι οποίες προσπίπτουν σε κάθε συνεκτική συνιστώσα.

- Έστω ότι ο αλγόριθμος $Kruskal$ δημιουργεί $d = k$ συνεκτικές συνιστώσες. Το δάσος που θα δημιουργηθεί είναι ελάχιστο, αλλά το ίδιο και οι συνεκτικές συνιστώσες ή δέντρα. Αν κάποια συνιστώσα θα μπορούσε να περιέχει και άλλη κορυφή θα την είχε συμπεριλάβει ο αλγόριθμος, καθώς επίσης αν κάποια συνιστώσα θα μπορούσε να είχε μικρότερο βάρος θα είχε δημιουργήσει μια ίσου βάρους. Έτσι για να δημιουργήσουμε το ΕΣΔ $T^*(s,k)$ χρησιμοποιώντας $k = d$ ακμές της s αρκεί να συνδέσουμε το s με κάθε συνεκτική συνιστώσα χρησιμοποιώντας την ελαφρύτερη "συνδετική" ακμή.

- Αν $d < k$, τότε όπως και πριν δημιουργούμε το ΕΣΔ $T^*(s,d)$ αρχικά. Από εκεί και πέρα πρέπει να συμπεριλάβουμε $d - k$ ακμές της s. Αυτές θα είναι οι $d - k$ ελαφρύτερες υπολειπόμενες ακμές που προσπίπτουν σε αυτή φυσικά. Αλλά κάθε φορά που προσθέτουμε μια ακμή θα δημιουργηθεί κυκλος, το οποίο σημαίνει ότι πρέπει να αφαιρεθεί μια ακμή από το ΕΣΔ (που είναι πια γράφος). Ιδανικά θα αφαιρούσαμε την βαρύτερη, αλλά αυτό μπορεί να χαλάσει την συνεκτικότητα. Οπότε εντοπίζουμε τον (μοναδικό) απλό κύκλο που δημιουργείται. Αν αφαιρεθεί κάποια ακμή από αυτόν τον κύκλο τότε η ιδιότητα της συνεκτικότητας συνεχίζει να υφίσταται και έχουμε πια δέντρο. Αφαιρούμε την βαρύτερη ακμή (εξαιρώντας αυτήν που προσθέσαμε) ώστε να ελαχιστοποιήσουμε το βάρος του δέντρου που θα δημιουργηθεί. Επαναλαμβάνοντας $d - k - 1$ φορές έχουμε το ζητούμενο δέντρο.

Άσκηση 2: Σχεδιασμός Videogame

α) Έστω πίνακας $|V|$ θέσεων: C. Ο αλγόριθμος φαίνεται παρακάτω:

$\hookrightarrow C[s] = r$
$\triangleright$ Για κάθε $u \in V \backslash \{s\}$:

- $C[u] \leftarrow -\infty$

$\triangleright$ $|V|$ φορές:

- Για κάθε ακμή $e = (u, v) \in E$:
  - $val \leftarrow C[u] + p[v]$
  - Αν $C[v] < val$ και $val > 0$ τότε: $C[v] \leftarrow val$

Η πολυπλοκότητα του αλγορίθμου είναι προφανώς $O(|V||E|)$. Σε κάθε επανάληψη i η θέση $d[u]$ είναι η δύναμη του παίκτη στο "καλύτερο" μονοπάτι μήκους i που καταλήγει στο u εκείνη τη στιγμή (και έχει αρχή το s). Στο τέλος το $d[t]$ θα έχει την δύναμη του παίκτη όταν ακολουθήσει τη βέλτιστη διαδρομή, αν $d[t] > 0$, αλλιώς θα είναι $-\infty$ και θα υποδηλώνει ότι δεν υπάρχει η ζητούμενη διαδρομή.

β) Αρχικά εφαρμόζουμε τον παραπάνω αλγόριθμο. Αν $d[t] \geq 0$ τότε υπάρχει r-ασφαλές μονοπάτι. Αν δεν ισχύει αυτό, τότε δεν υπάρχει μονοπάτι, αλλά μπορεί να υπάρχει διαδρομή. Τότε εφαρμόζουμε άλλη μια (μόνο) επανάληψη του αλγορίθμου. Αν έχουμε αλλαγή τότε υπάρχει προσθετικός κύκλος. Αν όχι, τότε δεν υπάρχει και ο λαβύρινθος δεν είναι r-ασφαλής. Έστω ότι υπάρχει, τότε εύκολα με μικρή τροποποίηση μπορούμε να λάβουμε τις κορυφές των κύκλων.

Τώρα μένει να ελέγξουμε αν μπορεί να φτάσει στην έξοδο t από κάποια από αυτές τις κορυφές (σε αυτές τις κορυφές φτάνει από την αρχή s σίγουρα λόγω της υλοποίησης του αλγορίθμου). Αρκεί να εφαρμόσουμε τον αλγόριθμο $Kruskal$ για να κατασκευάσει ένα συνδετικό δάσος, σε χρόνο $O(|E|log|V|)$. Μπορούμε τότε να βρούμε σε ποιο συνεκτικό σύνολο κορυφών ανήκει η t και με διαδοχικές υπερωτήσεις για κάθε κορυφή των κύκλων να ελέγξουμε αν κάποια από αυτές ανήκει στο ίδιο σύνολο με την t. Αυτό επιτυγχάνεται σε χρόνο $O(|V|log|V|)$, αφού κάνουμε το πολύ $|V|$ υπερωτήσεις και καθεμία από αυτές κάνοντας χρήση της δομής $DisjointSet$ (όπως και ο $Kruskal$) χρειάζεται χρόνο $O(log|V|)$, αφού τόσα είναι το πολύ τα ξένα σύνολα. Αν λοιπόν η t ανήκει στο ίδιο σύνολο με κάποια από αυτές τις κορυφές ο λαβύρινθος είναι r-ασφαλής. Η τελική πολυπλοκότητα του αλγορίθμου είναι $O(|V||E|)$.

γ) Θα εκτελέσουμε μια μορφή δυαδικής αναζήτησης. Θεωρούμε μια πολύ "φιλόδοξη" επιλογή $R = 1$ και εκτελούμε τον προηγούμενο αλγόριθμο για να δούμε αν είναι R-ασφαλής. Αν είναι 1-ασφαλής, τότε ελέγχουμε αν είναι και 0-ασφαλής, αν η απάντηση είναι θετική το ελάχιστο r είναι 0, αλλιώς 1. Αν ο λαβύρινθος δεν ήταν 1-ασφαλής διπλασιάζουμε το R και ξαναεκτελούμε. Συνεχίζουμε αυτό τον διπλασιασμό μέχρι να έχουμε θετική απάντηση. Μόλις λάβουμε θετική απάντηση θα εκτελέσουμε δυαδική αναζήτηση με άνω άκρο το R για το οποίο βρήκαμε θετική απάντηση και κάτω άκρο το $R/2$, ώστε να βρούμε το βέλτιστο r. Μπορούμε εύκολα να δούμε ότι αυτή η διαδικασία επαναλαμβάνεται $\Theta(logr)$ φορές, όπου r το ελάχιστο που ζητείται. Άρα, η πολυπλοκότητα του αλγορίθμου είναι $O(|V||E|logr)$.

Άσκηση 3: Ταξίδι σε Περίοδο Ενεργειακής Κρίσης

α) Καταρχάς αν $\exists e \in E : b(e) > B$, δεν μπορούμε να φτάσουμε στην πόλη t. Αυτός ο έλεγχος γίνεται εύκολα σε χρόνο $O(n)$. Θεωρούμε ότι είναι δυνατή η μετάβαση στην πόλη t.

Τροποποιούμε τον αλγόριθμο εύρεσης κυρίαρχης θέσης που διατυπώσαμε στην πρώτη σειρά ασκήσεων. Εφαρμόζουμε τον παρακάτω αλγόριθμο στον πίνακα $c[1..n]$ (όπου χρησιμοποιούμε την σύντμηση $c[i] = c[u_i]$ και $u_1 := s, u_n = t$) και αποθηκεύουμε τα αποτελέσματα στον $d[1..n]$:

$ClosestRightCheaper(\ c[1..n]\ )$:
$\hookrightarrow d[1..n-1] \leftarrow 1, 2, ..., n-1, s \leftarrow stack(NULL)$

$\triangleright$ Για i από $n-1$ έως 1:

- $s.push(i+1)$
- Όσο $s \neq$ κενή:
  - Αν $c[u_i] \leq c[u_{s.top()}]$, τότε: $s.pop()$
  - Αλλιώς: $break$
- $d[i] \leftarrow s.top()$

$return\ d[1..n]$

Μετά το πέρας του αλγορίθμου/συνάρτησης ισχύει:

$$d[i] = arg \left\{ \min_{\substack{c[j]<c[i] \\ j>i}} (j-i) \right\}$$

Πιο απλά στο $d[i]$ βρίσκεται η κοντινότερη θέση j "δεξιότερα" της i ($j > i$ δηλαδή) για την οποία ισχύει $c[j] < c[i]$. Εκτός από τις θέσεις i οι οποίες "κυριαρχούν" όλων τον επόμενων,

τότε $c[i] = i$, αφού έτσι αρχικοποιήσαμε τον πίνακα (για ευκολία αργότερα). Η πολυπλοκότητα του αλγορίθμου είναι $O(n)$ και αποδείχθηκε στην πρώτη σειρά ασκήσεων (όπως και η ορθότητα του). Ο αλγόριθμος του προβλήματος παρουσιάζεται παρακάτω. Πιο συγκεκριμένα σαν μια ρουτίνα (η απάντηση είναι $MinCostToTravel(1, 0)$).

$\underline{MinCostToTravel(i,\ tank)}$:
$\hookrightarrow\ cost \leftarrow 0$
Αν μπορούμε να φθάσουμε στον προορισμό τελειώσαμε. Ο πίνακας $CostFromStart$ μπορεί να υπολογιστεί σε χρόνο $\Theta(n)$ ως ένα κινούμενο άθροισμα.
- Αν $tank \geq CostFromStart[t] - CostFromStart[u_i]$, τότε: Επέστρεψε 0

Αν δεν υπάρχουν φθηνότερα βενζινάδικα στην πορεία ή το επόμενο φθηνότερο είναι πιο μακριά από την αυτονομία μας, τότε γεμίζουμε το ντεπόζιτο και εκτελούμε τον ίδιο αλγόριθμο στην αμέσως επόμενη πόλη.
- Αν $d[i] = i$ ή $tank < CostFromStart[i] - CostFromStart[d[i]]$, τότε:

- $cost \leftarrow cost + (B - tank) \cdot c[i]$
- $tank \leftarrow B$
- Επέστρεψε $cost + MinCostToTravel(i + 1,\ \mathrm{B} - b(i,\ i + 1))$

Αλλιώς γεμίζουμε το ντεπόζιτο (αν χρειάζεται) ώστε να αρκεί ακριβώς για να φθάσουμε στην επόμενη φθηνότερη πόλη και συνεχίζουμε από αυτήν.
$\triangleright$ Αν $i < n$:

- Για j από i μέχρι $d[i]$:
  - Αν $tank > b(u_i, u_{i+1})$, τότε: $tank \leftarrow fuel - b(u_j, u_{j+1})$
  - Αλλιώς:
    - $buy \leftarrow b(u_j, u_{j+1}) - tank$
    - $tank \leftarrow 0$
    - $cost \leftarrow cost + c[i] \cdot buy$
- Επέστρεψε $cost + MinCostToTravel(d[i],\ 0)$

$\hookleftarrow$ Επέστρεψε 0.

Η πολυπλοκότητα του αλγορίθμου είναι γραμμική: $\Theta(n)$.
Η ορθότητα του βασίζεται στα εξής επιχειρήματα:

- Αν μπορούμε να φθάσουμε στον προορισμό με την υπολειπόμενη βενζίνη τελειώσαμε.
- Αν βρισκόμαστε σε μια πόλη όπου η αμέσως επόμενη φθηνότερη είναι η u_j τότε γεμίζουμε το ντεπόζιτο ακριβώς ώστε να φθάσουμε εκεί.
- Αν δεν επαρκεί το ντεπόζιτο για να φθάσουμε εκεί ή δεν έπονται φθηνότερες πόλεις

γεμίζουμε το ντεπόζιτο και επαναλαμβάνουμε την διαδικασία στην επόμενη πόλη.

Θα αποδείξουμε ότι αυτός ο άπληστος αλγόριθμος παράγει την βέλτιστη λύση. Το άπληστο κριτήριο αποτελούν τα τρία παραπάνω βήματα. Είναι προφανές ότι αν μια λύση δεν ακολουθεί το πρώτο δεν μπορεί να είναι βέλτιστη, οπότε προχωράμε στο δεύτερο. Αν ισχύει η συνθήκη του δεύτερου βήματος, τότε μια λύση που δεν το ακολουθεί θα πρέπει να αγοράζει βενζίνη σε κάποια ενδιάμεση πόλη (πριν την u_j και μετά την αρχική), ώστε να φθάσει εκει (αν δεν υπάρχουν ενδιάμεσες τότε και οι δύο λύσεις απαιτούνται να αγοράσουν όση χρειάζεται για να πάνε από την αρχική στην επόμενη). Αν σε αυτή την λύση "αφαιρέσουμε" κάποια ποσότητα βενζίνης που αγοράσθηκε στην ενδιάμεση πόλη και την αγοράσουμε στην αρχική, τότε έχουμε μικρότερο κόστος. Η ίδια τεχνική ανταλλαγής μπορεί να χρησιμοποιηθεί και και για την ορθότητα του τρίτου βήματος. Άρα ο αλγόριθμος μας παράγει την βέλτιστη λύση.

β) Για την γενική περίπτωση όπου το G είναι γράφος θα χρησιμοποιήσουμε δυναμικό προγραμματισμό. Έστω $C(u, b)$ το ελάχιστο κόστος για να μεταβούμε από την πόλη u στην t έχοντας αρχικά b λίτρα βενζίνης. Τότε:

$$C(u,b) = \min_{v \in OutAdjList(u)} \begin{cases} C(v,\ 0) + c(u)[b(v,u) - b] & c(v) < c(u) \text{ και } b \leq b(v,u) \\ C(v,\ B - b(u,v)) + c(u)(\mathrm{B} - b) & c(v) > c(u) \\ 0 & u = t \\ \infty & \text{αλλιώς} \end{cases}$$

Μπορούμε να αποδείξουμε ότι σε μια διαδρομή *βέλτιστου κόστους* όπου οι στάσεις σε βενζινάδικα γίνονται στις πόλεις $i, i+1, ..., k$ τότε πρέπει:
- Αν $c(i) \geq c(j)$ τότε "γεμίζουμε" μέχρι να φτάσουμε με ακριβώς 0 βενζίνη στην j.
- Αν ισχύει το αντίθετο, τότε γεμίζουμε το ντεπόζιτο.

Απόδειξη:
Αν "αγοράσουμε" παραπάνω στην πόλη i ενώ ισχύει $c(i) \geq c(j)$, τότε θα μπορούσαμε να αφαιρέσουμε από αυτό που αγοράσαμε "περαιτέρω" στην πρώτη πόλη και να αγοράσουμε το ίδιο ποσό βενζίνης στην επόμενη με ίσο ή μικρότερο κόστος. Παρόμοια απόδειξη και για τον δεύτερο ισχυρισμό.

Απομένει να υπολογίσουμε την πολυπλοκότητα του αλγορίθμου. Αρχικά θα αποδείξουμε ότι χρειαζόμαστε μόνο συγκεκριμένες τιμές του b. Έστω $z \neq s$ μια κορυφή όπου αγοράζουμε βενζίνη στη βέλτιστη λύση και u η προηγούμενη της. Τότε με βάση τον αλγόριθμο θα φτάσουμε στην z με βενζίνη στο σύνολο: $\{B - b(u,z) | u \in IncAdjList(z)\}\{0\}$, δηλαδή θα μπορούσε να πάρει το πολύ $|V| - 1 + 1 = |V|$ τιμές. Άρα εφόσον χρειαζόμαστε το $D(u, g)$ για κάθε πιθανό u ($|V|$ κορυφές) και πιθανό g (το πολύ $|V|$ τιμές για κάθε κορυφή), αυτό εξαρτάται από το πολύ $|V|$ γείτονες της u. Επομένως η πολυπλοκότητα είναι $O(|V|^3)$.

Άσκηση 4: Επαναφορά της Ομαλότητας στην Χώρα των Αλγορίθμων

Η ζητούμενη ελαχιστοποίηση μπορεί να μετασχηματισθεί σε μια εύρεση ελάχιστης τομής σε γράφο ροής.

- Συνδέουμε μια κορυφή "στρατόπεδου" s με κάθε κορυφή που αντιστοιχεί σε εξτρεμιστές. Η χωρητικότητα της κάθε ακμής είναι το ελάχιστο κόστος που πρέπει να δαπανήσει ο στρατός για την αιχμαλωσία της ομάδας εξτρεμιστών.

- Συνδέουμε τον κάθε εξτρεμιστή με όλες τις βάσεις από τις οποίες απέχει το πολύ d με ακμές άπειρης χωρητικότητας.

- Συνδέουμε την κάθε βάση με την κορυφή "προορισμού" t με ακμή χωρητικότητας το ελάχιστο κόστος που πρέπει να δαπανήσει ο στρατός για την καταστροφή της.

Είναι προφανές ότι μια ελάχιστη τομή $s-t$ δε θα περιέχει τις ακμές μεταξύ εξτρεμιστών και βάσεων. Η σύνδεση τους είναι απαραίτητη όμως γιατί μπορεί η επιλογή καταστροφής μιας βάσης (αποκοπή ακμής βάσης προς t) να είναι συμφέρουσα έναντι επίθεσης στην ομάδα/ομάδες εξτρεμιστών που "καλύπτει".

Για την κατασκευή του γράφου θα πρέπει αρχικά να ταξινομήσουμε τις "κορυφές" κατά απόσταση. Θα χρειαστούμε χρόνο $O(nlogn + mlogm + klogk)$. Έπειτα σε γραμμικό χρόνο μπορούμε να υπολογίσουμε τα βάρη των ελάχιστων ακμών και να κατασκευάσουμε τον γράφο.

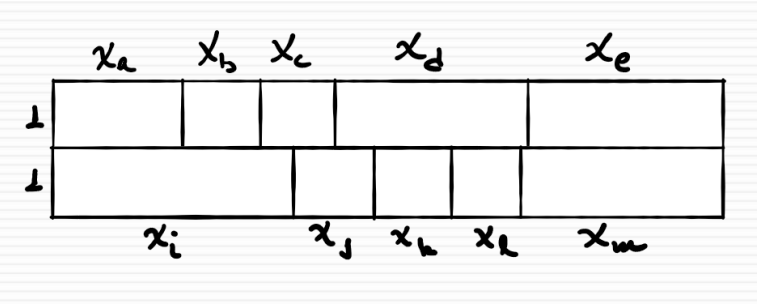
Μας μένει να βρούμε την ελάχιστη τομή, αυτό μπορεί να γίνει με τον αλγόριθμο $Edmonds-Karp$ με $push-relabel$ σε χρόνο $O(|V|^3)$ για έναν γράφο $G(V,E)$. Άρα η τελική πολυπλοκότητα του αλγορίθμου είναι: $O((m+k)^3 + nlogn + mlogm + klogk) = O((m+k)^3 + nlogn)$.

Άσκηση 5: Αναγωγές και NP-Πληρότητα

Τακτοποίηση Ορθογώνιων Παραλληλογράμμων

Καταρχάς μπορούμε σε γραμμικό χρόνο να ελέγξουμε αν μια δεδομένη τακτοποίηση είναι έγκυρη. Άρα το πρόβλημα ανήκει στο NP.

Θα δείξουμε ότι το πρόβλημα είναι γενίκευση του $2-Partition$. Έστω $B = 2 \times \left(\sum_{i=1}^{n} x_i\right)/2$. Τότε η τακτοποίηση έγκειται στην εύρεση δύο ξένων υποσυνόλων του $S = \bigcup_{i=1}^{n} \{x_i\}$ τα οποία αθροίζουν το καθένα στο $\frac{1}{2}\sum_{i=1}^{n} x_i$. Αυτό φαίνεται και σχηματικά παρακάτω:



Από τα παραπάνω το πρόβλημα είναι $NP-complete$.

Μέγιστη Τομή με Βάρη στις Κορυφές

Αρχικά η επαλήθευση μιας τομής γίνεται σε γραμμικό χρόνο ως προς τις ακμές, άρα το πρόβλημα ανήκει στο NP. Μπορούμε να ορίσουμε την παρακάτω ανισότητα για τα B στα οποία η απάντηση είναι "αληθής":

$$B \leq \sum_{u \in S} \sum_{v \in V \setminus S} w(u)w(v) \overset{\text{Πλήρες } G}{=} \sum_{u \in S} w(u) \cdot \sum_{v \in V \setminus S} w(v)$$

Έστω $K = \sum_{u \in V} w(u) =$ σταθερό και $U(S) = \sum_{u \in S} w(u)$, τότε $\sum_{v \in V \setminus S} w(v) = K - U(S)$. Άρα:

$$B \leq U(S)[K - U(S)]$$

Για τη συνάρτηση $f(x) = x(d-x), x \in R^+$ ισχύει: $f(x) = -x^2+dx = -(x-d/2)^2+d^2/4 \leq d^2/4$ και η λύση της $f(x) = d^2/4$ είναι το $x = d/2$. Άρα η μέγιστη τιμή της είναι το $d^2/4$.

Επομένως αν δοθεί είσοδος $B = K^2/4$ ζητείται μια διαμέριση $S, V \setminus S$ του συνόλου $\bigcup_{u \in V} \{w(u)\}$ ώστε $U(S) = \sum_{u \in S} w(u) = \frac{K}{2} = \frac{1}{2} \sum_{u \in V} w(u)$ και άρα $\sum_{v \in V \setminus S} w(v) = K - \frac{K}{2} = \frac{K}{2}$. Δηλαδή καταλήγουμε στο πρόβλημα $2-Partition$. Συμπεραίνουμε ότι το αρχικό πρόβλημα είναι $NP-complete$.

Αραιό Γράφημα (Sparse Subgraph)

Το πρόβλημα ανήκει στο NP αφού η επαλήθευση μιας "λύσης" γίνεται σε γραμμικό χρόνο.

Η απόδειξη πληρότητας πολύ εύκολη.

Συνάθροιση Προτιμήσεων (Preference Aggregation)

Το πρόβλημα ανήκει στο NP αφού η επαλήθευση μιας "λύσης" γίνεται σε πολυωνυμικό χρόνο.

Η απόδειξη πληρότητας πολύ εύκολη.

Συντομότερο Μονοπάτι με Περιορισμούς (Constrained Shortest Path)
Το πρόβλημα ανήκει στο NP αφού η επαλήθευση μιας "λύσης" γίνεται σε γραμμικό χρόνο.

Η απόδειξη πληρότητας πολύ εύκολη.