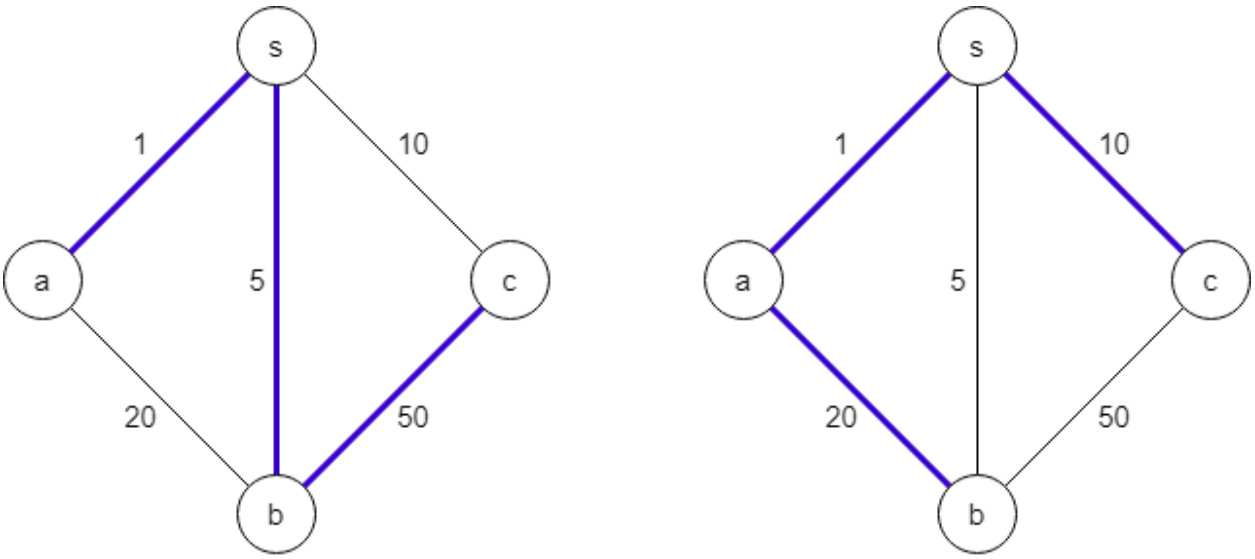
Αλγόριθμοι και Πολυπλοκότητα – 3η Σειρά Ασκήσεων
Κυριακόπουλος Γεώργιος – el18153

Άσκηση 1:

(α) Ακολουθώντας την άπληστη στρατηγική που περιγράφεται στο παρακάτω παράδειγμα θα είχαμε ως Ελάχιστο Συνδετικό Δέντρο $T^*(s,k)$ το αριστερό με συνολικός βάρος 56, ενώ το σωστό Ελάχιστο Συνδετικό Δέντρο $T^*(s,k)$ είναι το δεξί με βάρος 31.



(β) Θα χρησιμοποιήσουμε έναν αλγόριθμο όπου με χρήση της δυαδικής αναζήτησης θα ψάξουμε μία τιμή K την οποία θα προσθέτουμε κάθε φορά σε κάθε ακμή που προσπίπτει στο σχετικό κόμβο (s) για τον οποίο θέλουμε να έχει βαθμό ίσο με k. Για τις διάφορες τιμές του K είτε θα προστίθενται αυτές οι ακμές στο ΕΣΔ που ψάχνουμε, είτε θα αφαιρούνται, μέχρι να πετύχουμε το βαθμό κορυφής και το ελάχιστο δυνατό ΣΔ.

Για κάθε τέτοια τιμή του K θα ψάχνουμε το ΕΣΔ του επαυξημένου κατά K δέντρου. Για ένα αρκετά αρνητικό K το ΕΣΔ θα περιλαμβάνει όλες αυτές τις ακμές που προσπίπτουν στον κόμβο s, ενώ για ένα αρκετά θετικό K θα περιλαμβάνει τις ελάχιστες δυνατές ή μόνο μία (ώστε να καλύπτεται ο κόμβος s στο ΕΣΔ).

Ο βαθμός του κόμβου s στο ΕΣΔ αλλάζει για τιμές του K που έχουν ως αποτέλεσμα όταν μία ακμή προσπίπτουσα στον s αυξηθεί κατά K, να έχει ίσο βάρος με μία άλλη ακμή του γράφου. Επομένως, μπορούμε να περιορίσουμε το εύρος της δυναμικής αναζήτησης, ελέγχοντας τα

$K \in \{e - a: e \in E \wedge a \in A\}$, όπου Ε το σύνολο των διαφορετικών βαρών ακμών όλου του γραφήματος, ενώ Α το σύνολο των διαφορετικών βαρών ακμών που προσπίπτουν στον κόμβο s. Αυτό συμβαίνει, αφού για να αντικατασταθεί μία ακμή στο ΕΣΔ πρέπει να υπάρχει κάποια άλλη, όπου μετά την προσαύξηση κατά κάποιο Κ της πρώτης, αποκτούν ίσο βάρος. Επομένως, πρέπει να έχουν διαφορά βαρών ίσο με αυτό το Κ που ψάχνουμε.

Αρχικά, θα πρέπει να υπολογίσουμε και να ταξινομήσουμε το σύνολο των Κ που θα εξετάσουμε, το οποίο μπορεί να γίνει σε χρόνο $O(|E||A|\log(|E||A|))$. Μετά έρχεται η δυαδική αναζήτηση πάνω στο σύνολο αυτό, το οποίο μπορεί να γίνει σε χρόνο $O(\log(|E||A|)) \leq O(\log(|E||E|)) = O(2\log|E|)$. Για κάθε μία τιμή της δυαδικής αναζήτησης προσθέτουμε το συγκεκριμένο Κ στις ακμές που προσπίπτουν στον κόμβο s και βρίσκουμε το ΕΣΔ μέσω ενός αλγορίθμου όπως του Kruskal, σε $O(Elog(|E|))$. Ανάλογα με το εάν είμαστε πάνω ή κάτω από τον βαθμό στόχο μειώνουμε ή αυξάνουμε το Κ αντίστοιχα και επαναλαμβάνουμε. Τελικά, θα έχουμε πολυπλοκότητα $O(|E||A|\log(|E||A|) + |E|\log^2(|E|))$.

Άσκηση 2:

1. Έχουμε ένα Λαβύρινθο χωρίς κύκλους, επομένως με μία εκτέλεση ενός τροποποιημένου Bellman – Ford μπορούμε να βρούμε σε $O(|V||E|)$ βήματα το αποτέλεσμα.

Ουσιαστικά θα έχουμε έναν πίνακα $d[]$ με αρχικές αποστάσεις το $-\infty$ και θα κάνουμε τα βήματα μας με την παρακάτω λογική, ενώ αρχικά ο παίκτης μας έχει δύναμη r. Για κάθε ακμή που ενώνει τον κόμβο a με τον b $(a \to b)$ ελέγχουμε αν $d[a] + p[b] > 0$ και $d[a] + p[b] > d[b]$. Αυτές οι συνθήκες ουσιαστικά μας βοηθάνε να ανανεώνουμε τις αποστάσεις των κόμβων με τη λογική ότι όσο μεγαλύτερη τιμή μπορεί να λάβει η δύναμη μας τόσο το καλύτερο, ώστε να καταφέρουμε να φτάσουμε μέχρι το τέλος με θετική δύναμη και χωρίς να βρεθούμε με δύναμη μικρότερη ή ίση του 0 σε οποιαδήποτε θέση της διαδρομής του μονοπατιού. Για αυτό αρχικά χρησιμοποιούμε και την τιμή $-\infty$, καθώς ένας κλασσικός Bellman – Ford θα έψαχνε το μονοπάτι με το ελάχιστο κόστος, ενώ εμείς ψάχνουμε το μονοπάτι με το μέγιστο κόστος (τα θετικά βάρη για εμάς είναι επιθυμητά και όχι επιπλέον κόστος).

Για να πάρουμε το αποτέλεσμα πρέπει να ελέγξουμε την απόσταση που λαμβάνει στο τέλος των βημάτων η απόσταση του τελικού κόμβου, έστω t. Εάν αυτή είναι μεγαλύτερη του 0, τότε ο Λαβύρινθος αυτός είναι r – ασφαλής. Εάν, όχι, τότε δεν υπάρχει μονοπάτι που καταλήγει με θετική δύναμη στο τέλος του Λαβυρίνθου ή που να έχει κατά τη διαδρομή πάντα θετική δύναμη, δηλαδή ο Λαβύρινθος δεν είναι r – ασφαλής.

2. Αρχικά, εκτελούμε τον αλγόριθμο του βήματος 1. Έπειτα, εάν δεν έχει βρεθεί κάποια θετική λύση, δηλαδή το $d[t]$ είναι θετικό, τότε τελειώνουμε εκεί, καθώς ο Λαβύρινθος είναι r – ασφαλής. Σε διαφορετική περίπτωση, τρέχουμε άλλη μία επανάληψη των βημάτων για όλες τις ακμές (δηλαδή σύνολο V, ενώ το βήμα 1 είχε $V - 1$ επαναλήψεις). Εάν δούμε αλλαγή σε οποιαδήποτε απόσταση κόμβων, τότε σημαίνει ότι υπάρχει κύκλος προσβάσιμος από το s στο γράφημα. Πρέπει, τότε, να ελέγξουμε εάν κάποια κορυφή που ανήκει σε αυτόν τον κύκλο, έχει πρόσβαση στον τελικό κόμβο t (πχ με ένα DFS και ελέγχοντας εάν ο τελικός κόμβος t ανήκει στα visited). Εάν έχει πρόσβαση, τότε είναι και πάλι r – ασφαλής ο Λαβύρινθος. Εάν δεν έχει πρόσβαση, τότε δεν είναι r – ασφαλής ο Λαβύρινθος και η διαδικασία τελειώνει εδώ. Ο αλγόριθμος έχει και αυτός την ίδια πολυπλοκότητα $O(|V||E|)$ με τον προηγούμενο.

3. Τρέχουμε τον αλγόριθμο μας με χρήση της δυαδικής αναζήτησης για να βρούμε την ελάχιστη τιμή r για την οποία ο Λαβύρινθος G είναι r – ασφαλής. Ως αρχικό διάστημα για τη δυαδική αναζήτηση θα πάρουμε το 0 έως το άθροισμα των αρνητικών βαρών (δηλαδή της δύναμης που θα αφαιρέσουν όλα τα τέρατα στη χειρότερη περίπτωση), έστω M. Για την πολυπλοκότητα του αλγορίθμου έχουμε $O(|V||E|log(M))$.

Άσκηση 3:

1. Θα χρησιμοποιήσουμε έναν greedy αλγόριθμο για την επίλυση του προβλήματος. Αρχικά, θα ταξινομήσουμε τις πόλεις με βάση την απόσταση, ώστε να μπορούμε να τις διατρέξουμε γραμμικά μέχρι τον κόμβο στόχο (t). Έπειτα, θα χρησιμοποιήσουμε μία στοίβα για να υπολογίσουμε, για κάθε πόλη, ποια είναι η επόμενη πόλη που έχει φθηνότερη τιμή βενζίνης ανά λίτρο. Συγκεκριμένα, ξεκινάμε από το τέλος των πόλεων και ελέγχουμε εάν υπάρχει στο stack κάποια πόλη με φθηνότερη τιμή, κάνοντας pop τις ακριβότερες της στοίβας και προσθέτοντας στο τέλος και την τρέχουσα πόλη.

Έχοντας τη στοίβα με τις επόμενες φθηνότερες πόλεις θα ξεκινήσουμε ένα γραμμικό πέρασμα στις πόλεις από την αρχική (s) μέχρι την τελική (t). Κάθε φορά θα ελέγχουμε εάν μπορούμε να φτάσουμε στην επόμενη φθηνότερη πόλη με το πολύ B λίτρα βενζίνης, είτε από αυτά που έχουμε αγοράσει ήδη, είτε με αγορά ακριβώς όσων χρειάζονται για να φτάσουμε μέχρι την φθηνότερη πόλη. Σε διαφορετική περίπτωση, δηλαδή που δεν υπάρχει κάποιος τέτοιος σταθμός ή που αυτός είναι πολύ μακριά και δεν μπορούμε να τον φτάσουμε με το πολύ B λίτρα βενζίνης, τότε πρέπει να γεμίσουμε μέχρι τα B λίτρα το ντεπόζιτο, αφού η πόλη που βρισκόμαστε τώρα μας προσφέρει τη φθηνότερη δυνατή βενζίνη μέχρι να φτάσουμε σε μία άλλη με φθηνότερη βενζίνη (που να απέχει τουλάχιστον B λίτρα δρόμο από εδώ).

Οι γενικότεροι υπολογισμοί για το εάν μπορούμε να φτάσουμε με X λίτρα σε μία πόλη γίνονται με βάση τα κόστη $b(e)$ για κάθε ακμή και υπάρχει ένα ζήτημα σε σχέση με το πως δίνονται τα δεδομένα αυτά. Έχουμε θεωρήσει ότι είναι σε μορφή πίνακα και γίνονται retrieve σε $O(1)$, ενώ σε διαφορετική περίπτωση θα μπορούσαμε με χρήση κάποιου prefix array σε $O(n)$ και χωρίς επιβάρυνση στη συνολική πολυπλοκότητα μας να τα μετατρέψουμε στη μορφή που θέλουμε για να κάνουμε retrieve σε $O(1)$ για τους υπολογισμούς μας.

Για την αρχική ταξινόμηση έχουμε $O(nlogn)$, ενώ για την εύρεση των επόμενων φθηνότερων πόλεων με χρήση stack, έχουμε $O(n)$ αφού κάθε πόλη προστίθεται και αφαιρείται το πολύ 2 φορές από τη στοίβα. Τέλος, το γραμμικό πέρασμα είναι και αυτό $O(n)$, αφού για κάθε πόλη ελέγχουμε εάν φτάνει μέχρι την επόμενη φθηνότερη με βάση τον έτοιμο πίνακα που δίνει η διαδικασία με το stack και αντίστοιχα αγοράζουμε τα κατάλληλα λίτρα βενζίνης μετά από σταθερού κόστους υπολογισμούς. Άρα, συνολικά, έχουμε πολυπλοκότητα $O(nlogn)$.

2. Για αυτή την εκδοχή του προβλήματος θα χρησιμοποιήσουμε αρχικά δυναμικό προγραμματισμό και μετά έναν αλγόριθμο για εύρεση ελάχιστου μονοπατιού, όπως του Dijkstra.

Αρχικά, θα υπολογίσουμε με μία αναδρομική σχέση το ελάχιστο κόστος για να φτάσουμε από μία κορυφή u με λίτρα βενζίνης g στην κορυφή t. Δηλαδή θα υπολογίσουμε το $D(u, g)$ για όλες τις κορυφές v τις οποίες μπορώ να φτάσω με το πολύ B λίτρα βενζίνης. Η αναδρομική σχέση έχει ως εξής:

$$D(u,g) = \min_{\substack{v\ s.t.\\ d_{uv} \le B}} \begin{cases} D(v,0) + (d_{uv} - g)c(u), & c(v) \le c(u) \wedge g \le d_{uv} \\ D(v, B - d_{uv}) + (B - g)c(u), & c(v) > c(u) \end{cases}$$

Για κάθε πόλη u μπορούμε να φτάσουμε εκεί με διάφορους τρόπους από μία προηγούμενη πόλη w (είτε με ακριβώς 0 καύσιμα, σε περίπτωση που εκεί είναι φθηνότερη η βενζίνη, είτε με $B - d_{wu}$), επομένως για το g έχουμε το πολύ n τιμές.

Με τον υπολογισμό των διάφορων τιμών $D(u,g)$ έχουμε την απάντηση για το ελάχιστο κόστος για τα καύσιμα, που είναι ίσο με το $D(s,0)$, δηλαδή με το ελάχιστο κόστος για καύσιμα που χρειάζομαι για να φτάσω στην πόλη t με 0 καύσιμα, εάν ξεκινήσω από την s με 0 καύσιμα. Για να βρούμε, όμως, το μονοπάτι και το πλάνο ανεφοδιασμού, πρέπει να βρούμε το ελάχιστο μονοπάτι με βάση αυτά τα υπολογισμένα ελάχιστα κόστη.

Θα κατασκευάσουμε ένα νέο γράφημα H με κορυφές τα διάφορα ζευγάρια (u,g) για τα οποία υπολογίσαμε την αναδρομική σχέση και ακμές με αντίστοιχα θετικά βάρη w όσες τιμές προκύψουν από τον υπολογισμό του $D(u,g)$ για τις αποδεκτές v πόλεις (με $d_{uv} \le B$) κάθε φορά. Μετά, πάνω σε αυτό το νέο γράφημα θα εκτελέσουμε έναν αλγόριθμο ελάχιστου μονοπατιού, όπως ο αλγόριθμος Dijkstra και θα έχουμε τη λύση που ψάχνουμε.

Για τη συνολική πολυπλοκότητα έχουμε για τον υπολογισμό των τιμών του δυναμικού προγραμματισμού $O(n^2)$. Για τον Dijkstra, έχουμε τελικά το πολύ n^2 κορυφές και το πολύ n^3 ακμές. Άρα έχουμε πολυπλοκότητα $O(n^3 + n^2 log(n\text{^}2)) = O(n^3)$. Άρα, τελικά, για το συνολικό αλγόριθμο έχουμε $O(n^3)$.

Άσκηση 4:

Για την επίλυση του προβλήματος, θα το ανάγουμε σε ένα πρόβλημα εύρεσης ελάχιστης τομής και επομένως εύρεσης μέγιστης ροής. Για το σκοπό αυτό θα χρησιμοποιήσουμε ένα γράφημα G με $m + k + 2$ κόμβους, όπου m οι κόμβοι για τους Εξτρεμιστές, k οι κόμβοι για τις βάσεις των Εξτρεμιστών και οι 2 επιπλέον είναι οι ακραίοι κόμβοι s (source) και t (target).

Για τη δημιουργία του γραφήματος, θα συνδέσουμε, αρχικά, κάθε κόμβο από τους m των Εξτρεμιστών με όσους κόμβους από τους k των βάσεων Εξτρεμιστών απέχουν απόσταση $\leq d$, δηλαδή που μπορεί να το εξυπηρετεί για ανεφοδιασμό. Στις ακμές που ενώνουν αυτούς τους κόμβους θα θέσουμε τη χωρητικότητα ίση με το άπειρο, καθώς δεν θέλουμε να αποτελούν μέρος της λύσης της ελάχιστης τομής, αφού δεν επηρεάζουν κατά κάποιο τρόπο το αποτέλεσμα μας, παρά μόνο την επιτυχημένη επίθεση ή αιχμαλωσία της βάσης. Έπειτα, συνδέουμε κάθε κόμβο από τους m των Εξτρεμιστών με τον κόμβο s και με χωρητικότητα το ελάχιστο κόστος ώστε να επιτεθεί ο στρατός. Αυτό το ελάχιστο κόστος ουσιαστικά ισούται με το διπλάσιο της απόστασης της κοντινότερης στρατιωτικής δύναμης στη θέση του Εξτρεμιστή, δεδομένου ότι ο στρατός πρέπει να επιστρέψει και πίσω στη θέση από την οποία επιτέθηκε. Τέλος, συνδέουμε κάθε κόμβο από τους k των βάσεων Εξτρεμιστών με τον κόμβο t με χωρητικότητα το ελάχιστο κόστος ώστε να την αιχμαλωτίσει ο στρατός, το οποίο είναι με παρόμοια λογική με παραπάνω το διπλάσιο της απόστασης της κοντινότερης στρατιωτικής δύναμης.

Με βάση αυτό το γράφημα G, η εύρεση του $s - t$ Min Cut είναι το ζητούμενο κόστος, αφού περιέχει τις ακμές από το s σε κόμβους Εξτρεμιστών και από το t στους κόμβους βάσεων Εξτρεμιστών, ενώ δεν έχει επηρεαστεί με κάποιο τρόπο από τις ακμές που έχουν άπειρη χωρητικότητα, δηλαδή αυτές που συνδέουν τους Εξτρεμιστές με τις βάσεις τροφοδοσίας τους. Για να βρούμε το $s - t$ Min Cut μπορούμε να βρούμε, ισοδύναμα, το $s - t$ Max Flow με κάποιον γνωστό αλγόριθμο, όπως αυτόν του Dinic, με πολυπλοκότητα $O(V^2E)$.

Σε αυτή την πολυπλοκότητα πρέπει να προστεθεί και η πολυπλοκότητα δημιουργίας του γραφήματος G, που απαιτεί, αρχικά, την ταξινόμηση των θέσεων των στρατιωτικών δυνάμεων, των Εξτρεμιστών και των βάσεων των Εξτρεμιστών στην ευθεία γραμμή και έπειτα ένα γραμμικό πέρασμα αυτών για τη δημιουργία όλων των ακμών του γραφήματος. Αυτά έχουν αντίστοιχα πολυπλοκότητα $O((n + m + k)log(n + m + k))$ και $O(n + m + k)$. Άρα αθροιστικά θα έχουμε τελική πολυπλοκότητα $O((n + m + k)log(n + m + k) + n + m + k + (m + k + 2)^2E) = O((m + k + 2)^2E)$, με το $E \leq V^2 = (m + k + 2)^2$.

Άσκηση 5:

Τακτοποίηση Ορθογωνίων Παραλληλογράμμων

Θα κάνουμε αναγωγή από το 2-Partition πρόβλημα. Ουσιαστικά, θα διαλέξω για το μεγάλο ορθογώνιο B ένα partition που χωρίζει στη μέση το σύνολο $\{a_1, a_2, \dots, a_n\}$ σε δύο ισομεγέθη σύνολα. Έστω αυτό το partition $t = \frac{1}{2}\sum_i a_i$. Επίσης, έστω τα μικρά ορθογώνια A_i με μεγέθη $a_i \cdot \varepsilon$, με $\varepsilon \ll 1$ και κατάλληλα επιλεγμένο. Επομένως, το ορθογώνιο B θα έχει εμβαδόν $t \cdot 2\varepsilon$. Έτσι, ανάγουμε την τακτοποίηση των ορθογωνίων παραλληλογράμμων στην εύρεση του 2-partition του συνόλου των μεγεθών a_i ορθογωνίων.

Μέγιστη Τομή με Βάρη στις Κορυφές

Θα χρησιμοποιήσουμε πάλι το 2-Partition με τη λογική του να βρούμε ένα partition $(S, V\backslash S)$ των κορυφών V της κλίκας με τομή τουλάχιστον B βάρους. Τότε:

$$B \leq w(S, V\backslash S) = \sum_{u\in S, v\notin S} w_u w_v = \sum_{u\in S} w_u \sum_{v\in[n]\backslash S} w_v$$

Το πρόβλημα μας τώρα είναι ισοδύναμο με το να μεγιστοποιήσουμε το βάρος της τομής. Αν ορίσουμε $A = \sum_{i\in[n]} w_i$, έχουμε:

$$w(S)\big(A - w(S)\big) \leq \frac{A^2}{4}$$

Αυτό προκύπτει από τη μεγιστοποίηση της κοίλης συνάρτησης $f(x) = x(A - x)$, που γίνεται στη θέση $x = \frac{A}{2}$ με $f(\frac{A}{2}) = \frac{A^2}{4}$. Άρα πρέπει να χρησιμοποιήσουμε το 2-partition πρόβλημα, αφού εάν είχαμε ένα σύνολο με θετικούς ακεραίους και θέλαμε να το διαμερίσουμε σε δύο ισοβαρή υποσύνολα, τότε θα αρκούσε να μεγιστοποιήσουμε την τομή ενός γραφήματος με κορυφές ίσες με τον αριθμό των στοιχείων του 2-partition και βάρη στις κορυφές ίσα με τα στοιχεία του συνόλου που δίνεται.

Αραιό Γράφημα (Sparse Subgraph)

Θα χρησιμοποιήσουμε το πρόβλημα του Independent Set για την αναγωγή μας. Το πρόβλημα αυτό παίρνει έναν γράφο $G = (V, E)$ και ένα $g \in \mathbb{Z}$ και επιστρέφει έναν υποσύνολο $S \subset V$, όπου το $|S| \geq g$ και για κάθε $x, y \in S, (x, y)$ δεν ανήκει στο E, δηλαδή δεν υπάρχουν ακμές μεταξύ των κορυφών του S.

Για να ανάγουμε το Independent Set στο Sparse Subgraph πρέπει πρώτα να δείξουμε πως η είσοδος του πρώτου μετατρέπεται στην είσοδο του δεύτερου. Ορίζουμε το k ίσο με το g και έπειτα τρέχουμε το Sparse Subgraph με είσοδο (G, k).

Ουσιαστικά, έχουμε μια ισοδυναμία μεταξύ των δύο προβλημάτων, δηλαδή εάν έχει ένας γράφος G ένα Independent Set S με $|S| = g$, τότε ο γράφος G έχει ένα Sparse Subgraph S, όπου $|S| = k = g$. Άρα, δοσμένου ενός Independent Set S, αυτό αποτελεί και ένα Sparse Subgraph, αφού δεν υπάρχουν ακμές μεταξύ των κορυφών του S και $|S| = g = k$.

Συντομότερο Μονοπάτι με Περιορισμούς (Constrained Shortest Path)

Θα χρησιμοποιήσουμε το πρόβλημα απόφασης Knapsack με n αντικείμενα, με κόστη c_i και βάρη w_i. Για κάθε αντικείμενο του Knapsack δημιουργούμε μία κορυφή σε ένα πλήρη κατευθυνόμενο γράφο. Κάθε ακμή που φεύγει από αυτή την κορυφή έχει βάρος w_i και κόστος c_i. Έτσι, έχουμε μετατρέψει την είσοδο του Knapsack σε είσοδο του Constrained Shortest Path. Εάν υπάρχει λύση σε πολυωνυμικό χρόνο στο πρόβλημα του Constrained Shortest Path, τότε υπάρχει και λύση σε πολυωνυμικό χρόνο για το πρόβλημα απόφασης του Knapsack, γιατί ένα μονοπάτι $s - t$ περνάει μόνο μία φορά από κάθε κορυφή και ακμή. Όμως, ξέρουμε ότι το πρόβλημα απόφασης του Knapsack είναι NP-Complete, άρα αυτό δεν ισχύει. Επομένως και το Constrained Shortest Path είναι NP-Complete.