Λύσεις 2ης σειράς προγραμματιστικών ασκήσεων

Αλγόριθμοι και Πολυπλοκότητα

2019-2020

23 Ιανουαρίου 2020

Περιεχόμενα

Επόμενο πρόβλημα

- Το πρόβλημα στην ουσία μας ζητάει, δεδομένου ενός πίνακα N θετικών ακεραίων και ενός K να βρούμε την μεγαλύτερη υποακολουθία του πίνακα που είναι γνησίως αύξουσα με εξαίρεση K το πολύ σημεία.
- Για παράδειγμα η ακολουθία [1,2,3,2,4,5] έχει ακριβώς μια εξαίρεση.
- Το πρώτο υποπρόβλημα που θέλουμε να μελετήσουμε (και που πιάνει το 80% της βαθμολογίας) είναι η περίπτωση όπου $K = 1$.

Υποπρόβλημα με K = 1

- Αν έχουμε την δυνατότητα να το πολύ μια εξαίρεση υπάρχουν ουσιαστικά δύο περιπτώσεις. Είτε δεν θα κάνουμε καμία εξαίρεση (οπού στην ουσία απλά ψάχνουμε το μήκος της μέγιστης γνησίως αύξουσας υποακολουθίας), είτε θα κάνουμε 1 ακριβώς εξαίρεση.
- Η πρώτη περίπτωση είναι απλή και μπορούμε να την λύσουμε σε χρόνο $O(NlogN)$ με τον γνωστό αλγόριθμο. Επικεντρωνόμαστε επομένως στην δεύτερη περίπτωση. Πως θα έμοιαζε μια τέτοια υποακολουθία; Σίγουρα θα ξεκινούσε να αυξάνει μέχρι ένα σημείο, μετά θα μειωνόταν και μετά θα αύξανε συνεχώς πάλι.
- Αν με κάποιον μαγικό τρόπο γνωρίζαμε το σημείο που σπάει η μονοτονία· Το σημείο δηλαδή που πρέπει να γίνει η εξαίρεση πως θα μπορούσαμε να χρησιμοποιήσουμε αυτή την πληροφορία για να λύσουμε το πρόβλημα;

Η ΄μαγική΄ πληροφορία

- Έστω πως με κάποιο μαγικό τρόπο γνωρίζαμε πως στην βέλτιστη λύση το σημείο που πρέπει να γίνει η εξαίρεση είναι το σημείο i. Τότε σίγουρα η λύση αποτελείται από μια αύξουσα ακολουθία χρησιμοποιώντας το κομμάτι $1, 2, .., i$ και μια αύξουσα ακολουθία που χρησιμοποιεί το κομμάτι $i+1, ..., N$.
- Παρατηρούμε πως οι δύο ακολουθίες είναι εντελώς ανεξάρτητες, το πως μοιάζει η μια δεν έχει κάποια επιρροή στην άλλη.
- Εφ΄ όσον μας ενδιαφέρει να μεγιστοποιήσουμε το συνολικό μέγεθος της ακολουθίας μας πρέπει και αρκεί να πάρουμε την μέγιστη αύξουσα υποακολουθία χρησιμοποιώντας τους αριθμους $1, ..., i$ και την μέγιστη αύξουσα υποακολουθία χρησιμοποιώντας τους αριθμούς $i+1, ..., N$, η τελική απάντηση θα είναι απλά το άθροιμα των μηκών αυτών των δύο υποακολουθιών.

Κάνοντας τον προυπολογισμό

- Βάσει όσων είπαμε προηγουμένως, καταλαβαίνουμε πως πρέπει με κάποιον τρόπο να γνωρίζουμε ποιά είναι η μέγιστη αύξουσα υποακολουθία χρησιμοποιώντας τους αριθμους στις θέσεις $1, ..., i$ καθώς και στις θέσεις $i+1, ..., N$.
- Αυτό όμως είναι εύκολο με δυναμικό προγραμματισμό. Αρκεί να ορίσουμε $Left[i]$ την μέγιστη αύξουσα υποακολουθία που μπορούμε να φτιάξουμε χρησιμοποιώντας μόνο τους πρώτους i αριθμούς καθώς και $Right[i]$ την μέγιστη αύξουσα υποακολουθία που χρησιμοποιεί τους αριθμούς από $i+1, ..., N$.
- Παρατηρούμε πως το $Left[i]$ υπολογίζεται με τον κλασικό αλγόριθμο LIS ενώ το $Right[i]$ είναι η μέγιστη φθίνουσα υποακολουθία αν αντιστρέψουμε τον πίνακα.
- Επομένως μπορούμε εύκολα να τα υπολογίσουμε και τα δύο σε χρόνο $O(NlogN)$.

Φτιάχνοντας την λύση χωρίς μαγική πληροφορία

- Αν τώρα γνωρίζουμε αυτή την μαγική πληροφορία που αναφέραμε, η απάντηση είναι απλά $Left[i] + Right[i]$.
- Τι γίνεται όμως αν δεν γνωρίζουμε ποιά είναι αυτή η μαγική θέση·
- Πολύ απλά, αυτή η μαγική θέση θα είναι κάποια από τις 1,2,3,...,N-1. Αρκεί να τις δοκιμάσουμε όλες και να δούμε ποιά μας μεγιστοποιεί το άθροισμα $Left[i] + Right[i]$.
- Εν τέλει λοιπόν η απάντηση είναι:

$$max_{1 \leq i < N}(Left[i] + Right[i]) \tag{1}$$

- Η τελική πολυπλοκότητα προκύπτει εύκολα και είναι $O(NlogN)$

Ένας αλγόριθμος πολυωνυμικού χρόνου για αυθαίρετο K

- Προκειμένου να υπολογίσουμε την μεγαλύτερη υποακολουθία για $K > 1$ εξαιρέσεις θα εφαρμόσουμε δυναμικό προγραμματισμό.
- Ξεκινάμε ως γνωστόν απ' τον πιο απλό χώρο καταστάσεων που μπορούμε να σκεφτούμε. Έστω $dp[i]$ η μεγαλύτερη υποακολουθία με K εξαιρέσεις που σταματάει στο σημείο i. Εύκολα μπορούμε να δούμε ότι αύτο από μόνο του σαν πληροφορία δεν μας φτάνει αφού όπως και να πάμε να υπολογίσουμε το $dp[i]$ δεν γνωρίζουμε αν μας περισσεύουν εξαιρέσεις.
- Συνεπώς, επεκτείνουμε τον χώρο καταστάσεων με μια επιπλεόν μεταβλητή που 'θυμάται' πόσες εξαιρέσεις έχουμε κάνει μέχρι στιγμής.
- Έστω λοιπόν $dp[i, k]$ η μεγαλύτερη υποακολουθία που μπορούμε να φτιάξουμε σταματώντας στο i αν έχουμε κάνει ακριβώς k εξαιρέσεις.

Γράφοντας την αναδρομική σχέση

- Έστω πως θέλουμε να υπολογίσουμε το υποπρόβλημα $dp[i,k]$. Έχουμε δύο περιπτώσεις, είτε το i είναι ο πρώτος αριθμός που θα χρησιμοποιήσουμε μετά από κάποια εξαίρεση, είτε θα είναι μέρος κάποιας αύξουσας υποακολουθίας.
- Η πρώτη περίπτωση είναι εύκολη, εφ' όσον ο i είναι ο πρώτος αριθμός μετά από κάποια εξαίρεση αρκεί να βρούμε ποιός ήταν ο προηγούμενός του. Ο προηγούμενος του όμως μπορεί να είναι οποιοσδήποτε αριθμός πριν το i αφού κάναμε εξαίρεση(προφανώς επιλέγουμε αυτόν που μας δίνει μέγιστο μήκος), οπότε έχουμε:

$$firstcase : max_{j<i}(dp[j,k-1]+1) \tag{2}$$

Η άλλη περίπτωση

- Στην άλλη περίπτωση δεν μπορούμε να διαλέξουμε οποιονδήποτε αριθμό. Εφ' όσον δεν έχουμε κάνει κάποια εξαίρεση οι αριθμοί στις θέσεις i και j πρέπει να είναι διατεταγμένοι.
- Αν συμβολίσουμε με A τον πίνακα, αυτό μεταφράζεται πολύ φυσικά στο ότι μας νοιάζουν εκείνοι οι αριθμοί όπου $j < i$ και $A[j] < A[i]$, ακριβώς όπως και στο πρόβλημα της μέγιστης αύξουσα υποακολουθίας.
- Έχουμε επομένως:

$$secondcase : max_{j<i \wedge A[j]<A[i]}(dp[j,k]+1) \tag{3}$$

- Συνδυάζοντας τις δύο επιμέρους περιπτώσεις μαζί έχουμε την συνολική αναδρομική σχέση:

$$dp[i,k] = 1 + max(max_{j<i}(dp[j,k-1]), max_{j<i \wedge A[j]<A[i]}(dp[j,k])) \tag{4}$$

Η λύση και ανάλυση πολυπλοκότητας

- Αφού υπολογίσουμε με τον τρόπο που αναφέραμε όλα τα υποπροβλήματα θέλουμε να δώσουμε την τελική απάντηση.
- Μπορούμε να χρησιμοποιήσουμε οσεσδήποτε εξαιρέσεις θέλουμε από 1 εώς K και ο τελευταίος αριθμός που θα χρησιμοποιήσουμε μπορεί να είναι οποιοσδήποτε από τους A[1] εώς A[N].
- Συνεπώς λαμβάνουμε τελικά σαν απάντηση:

$$max_{1 \leq i \leq N}(max_{1 \leq k \leq K}(dp[i,k]) \tag{5}$$

- Τέλος, για να υπολογίσουμε την πολυπλοκότητα του αλγορίθμου μας βλέπουμε πως έχουμε N*K συνολικά υποπροβλήματα και καθένα απ΄ αυτά το υπολογίζουμε σε γραμμικό χρόνο ως προς N.
- Συνεπώς έχουμε συνολική πολυπλοκότητα $O(N^2K)$.

Ναι μεν, αλλά

- Έχουμε βρει έναν πολυωνυμικό αλγόριθμο για το πρόβλημα ο οποίος είναι αρκετός για το 100% της βαθμολογίας. Σαν μπόνους θα δούμε πως μπορούμε να τον βελτιώσουμε.
- Θα ξεκινήσουμε από βελτιώσεις οι οποίες είναι σχετικά απλές και θα καταλήξουμε σε έναν υπερβολικά αποδοτικό αλγόριθμο ο οποίος χρησιμοποιεί πολύ εξιδεικευμένες τεχνικές.
- Η πρώτη βελτίωση που θα κάνουμε θα είναι χρησιμοποιεί την μορφή της λύσης που περιγράψαμε και απλώς θα προσπαθήσει να υπολογίσει τα υποπροβλήματα πιό αποδοτικά

Υπενθύμιση

- Θυμίζουμε λίγο ποιά είναι η αναδρομική σχέση από πριν:

$$dp[i,k] = 1 + max(max_{j<i}(dp[j,k-1]), max_{j<i \wedge A[j]<A[i]}(dp[j,k])) \quad (6)$$

- Βλέπουμε πως αποτελείται από 2 μέρη. Το πρώτο μέρος απλά αναζητάει το $max_{j<i}(dp[j,k-1])$.
- Μπορούμε να το υπολογίσουμε σε καλύτερο από γραμμικό χρόνο;.
- Μπορούμε βασικά να το υπολογίσουμε σε σταθερό χρόνο. Αν σκεφτούμε τον δισδιάστατο πίνακα που έχουμε απλά ψάχνουμε για κάθε k και για κάθε i το μέγιστο στοιχείο της προηγούμενης γραμμής μέχρι την γραμμή i.
- Είναι επομένως $max_{j<i}(dp[j,k-1]) = max(max_{j<i-1}(dp[j,k-1])), dp[i-1,k-1])$.
- Διατηρώντας λοιπόν μια μεταβλητή cur_max η οποία σε κάθε βήμα της εκτέλεσης ανανεώνεται στην τωρινή μέγιστη τιμή μπορούμε σε σταθερό χρόνο κάθε φορά να λαμβάνουμε τον μέγιστο αριθμό από την προηγούμενη στήλη.

Η δεύτερη περίπτωση

- Η δεύτερη περίπτωση είναι πιο δύσκολη καθώς δεν μας ενδιαφέρουν απλά όλοι οι προηγούμενοι αριθμοί αλλά όλοι όσοι έχουν και μικρότερη τιμή από τον $A[i]$.
- Όμως, η δεύτερη περίπτωση είναι ακριβώς ταυτόσημη με την εύρεση της μέγιστης αύξουσας υποακολουθίας.
- Χωρίζουμε τον αλγόριθμο σε βήματα ως προς το k. Βρισκόμαστε στο k-οστό βήμα όταν θέλουμε να υπολογίσουμε όλες τις τιμές του πίνακα $dp[i,k], 1 \leq i \leq N$.
- Θέλουμε με κάποιον τρόπο στο k-οστό βήμα όταν θέλουμε να υπολογίσουμε το $dp[i,k]$ να βρούμε την μέγιστη τιμή $dp[j,k-1]$ για εκείνα ακριβώς τα j για τα οποία ισχύει $A[j] < A[i]$.
- Αυτό όμως μπορεί να γίνει εύκολα με την χρήση μιας δομής όπως segment tree σε χρόνο $O(logN)$.
- Περισσότερες πληροφορίες για την χρήση του segment tree στην λύση της μέγιστης αύξουσας υποακολουθίας μπορείτε να βρείτε εδώ.

Η πολυπλοκότητας της λύσης μας

- Με την χρήσης της δομής δεδομένων που αναφέραμε καταφέραμε να ρίξουμε την πολυπλοκότητα του υπολογισμού κάθε υποπροβλήματος σε χρόνο $O(logN)$ από τον γραμμικό που είχαμε πριν.
- Συνεπώς, έχουμε συνολικό χρόνο $O(NKlogN)$.
- Περισσότερες πληροφορίες για το segment tree και πιθανές εφαρμογές του μπορείτε να βρείτε και σε αυτό το άρθρο.
- Όπως υποσχεθήκαμε, η πολυπλοκότητα αναμένεται να πέσει ακόμα περισσότερο. Προς το παρόν υπάρχει μια πολλαπλασιαστική εξάρτηση μεταξύ N και K. Με την χρήση μιας ενδιαφέρουσας και αρκετά διαφορετικής τεχνικής θα προσπαθήσουμε να την σπάσουμε.
- Όλα θα βγάλουνε νόημα σύντομα.

Η παθογένεια της λύσης μας

- Στην προηγούμενη λύση που περιγράψαμε καθώς και στην αρχική πάντα υπήρχε ένας όρος NK στην πολυπλοκότητα.
- Αυτό δεν είναι απλά τυχαίο. Βασίζεται στην δομή της λύσης μας αυτή καθ' αυτή. Επειδή έχουμε NK υποπροβλήματα να υπολογίσουμε και δεν μπορούμε να αγνοήσουμε κάποιο (πρέπει δηλαδή να τα υπολογίσουμε όλα) έχουμε με αυτή την μέθοδο ένα προφανές κάτω φράγμα $\Omega(NK)$.
- Συνεπώς, χρησιμοποιώντας την μέθοδο που αναπτύξαμε στα προηγούμενα βήματα είμαστε σχεδόν βέλτιστοι ως προς το κάτω φράγμα (Έχουμε λύση $O(NKlogN)$).
- Αυτό σημαίνει πως αν θέλουμε να ρίξουμε την πολυπλοκότητα δραματικά πρέπει να βρούμε κάποια άλλη κατεύθυνση, όπου φυσικά θα δανειστούμε ιδέες από τις προηγούμενες λύσεις.
- Αυτή η ιδέα είναι οι λεγόμενοι Lagrangian Multipliers .

Μια περίεργη ιδέα

- Το πρόβλημα με την λύση μας είναι βασικά ότι έχουμε την δεύτερη διάσταση του πίνακα που ΄θυμάται΄ πόσες εξαιρέσεις έχουμε κάνει μέχρι στιγμής.
- Αυτό όμως είναι και κάπως αναγκαίο. Αν δεν την είχαμε θα μπορούσαμε να κάνουμε συνέχεια εξαιρέσεις και να καταλήξουμε με μια ακολουθία μεγέθους N η οποία φυσικά δεν είναι feasible σαν λύση.
- Τι θα γινόταν όμως αν επιβάλλαμε κάποια ΄ποινή΄ για κάθε εξαίρεση;
- Αν λέγαμε δηλαδή πως μπορούμε να κάνουμε όσες εξαιρέσεις θέλουμε, αλλά κάθε φορά που κάνουμε μια η λύση μας μειώνεται κατά μια σταθερά C.
- Σίγουρα το πόσες εξαιρέσεις θα κάνουμε έχει να κάνει άμεσα με αυτή την σταθερά, αλλά πως μοιάζει αυτή η συνάρτηση σταθεράς C-εξαιρέσεων;

Μελετώντας την εξάρτηση

- Έστω $f(C)$ το πλήθος των εξαιρέσεων που θα κάνουμε στην βέλτιστη λύση αν κάθε φορά που γίνεται μια εξαίρεση 'χρεωνόμαστε' την σταθερά C (γίνεται δηλαδή $sol = sol - C$). Θεωρούμε $C \geq 0$.
- Είναι σαφές πως $f(0) = N$ αφού αν δεν χρεωνόμαστε τίποτα μας συμφέρει να κάνουμε συνεχώς εξαιρέσεις και επιπλέον $lim_{C \to +\infty} f(C) = 0$, οπότε έχουμε στην ουσία το πρόβλημα της μέγιστης αύξουσας υποακολουθίας.
- Αν μπορούσαμε

Υποπρόβλημα 1

- Θα λύσουμε αρχικά την περίπτωση όπου K = N - 1. Σ′ αυτή την περίπτωση θέλουμε να καλύψουμε όλες τις ακμές του δέντρου διαλέγοντας το ελάχιστο πλήθος από κορυφές.
- Αποτελεί ουσιαστικά μια ειδική περίπτωση του γνωστού NP-Complete προβλήματος Vertex Cover σε δέντρα.
- Παρ′ όλο που το πρόβλημα είναι δύσκολο σε γενικούς γράφους θα δούμε πως λύνεται σε πολυωνυμικό χρόνο(και μάλιστα σε γραμμικό) σε δέντρα. Μια ενδιαφέρουσα σημείωση για το εν λόγω πρόβλημα είναι πως υπάρχουν αλγόριθμοι πολυωνυμικού χρόνου και για διμερείς γράφους.
- Για να λύσουμε το πρόβλημα θα χρησιμοποιήσουμε δυναμικό προγραμματισμό.

Προεργασία

- Αρχικά θεωρούμε χωρίς βλάβη της γενικότητας ότι το δέντρο μας έχει για ρίζα την κορυφή 1. Ο λόγος που θέλουμε να ορίσουμε μια ρίζα για το δέντρο μας είναι το ότι θέλουμε να το ΄τακτοποιήσουμε΄ ώστε να μπορούμε κάπως να ορίσουμε επι μέρους υποπροβλήματα.
- Συγκεκριμένα, αν ας πούμε το δέντρο μας ήταν απλά μια αλυσίδα, θα θέλαμε να το βάλουμε στην σειρά και να ορίζαμε υποπροβλήματα για την αλυσίδα που ορίζεται απ΄ τους πρώτους 1,2,3,...,N κόμβους.
- Φυσικά, στην γενική περίπτωση το δέντρο έχει πιο πολύπλοκη τοπολογία, ωστόσο θα δούμε πως με το να το ριζώσουμε από κάποιον κόμβο και να αφήσουμε τους υπόλοιπους κόμβους να ΄πέσουν΄ προς τα κάτω μπορούμε να ορίσουμε έναν παρόμοιο χώρο καταστάσεων.
- Είναι προφανές πως η ρίζα δεν έχει κάποια επιρροή στο ποιά θα είναι η τελική απάντηση.Η τελική απάντηση είναι ανεξάρτητη απ΄ το πως θα ΄δούμε΄ το δέντρο. Συνεπώς, επιλέγουμε για ρίζα τον κόμβο 1 μόνο για απλότητα.

Ορισμός χώρου καταστάσεων

- Πλέον είμαστε έτοιμοι να ορίσουμε τον χώρο καταστάσεων.
- Αν θυμηθούμε την απλή περίπτωση που το δέντρο μας είναι μια αλυσίδα παρατηρούμε πως όποιον κόμβο και να διαλέξουμε χωρίζει την αλυσίδα σε ένα αριστερό και ένα δεξί κομμάτι. Ακριβώς επειδή υπάρχει αυτή η συμμετρία μεταξύ αριστερού και δεξιού μέρους θα λύναμε το πρόβλημα θεωρώντας ότι γνωρίζουμε την λύση για το αριστερό ας πούμε κομμάτι και θα προσπαθούσαμε να την χρησιμοποιήσουμε για να την επεκτείνουμε ώστε να περιλαμβάνει και τον τωρινό κόμβο.
- Αντίστοιχα, έχοντας ορίσει την ρίζα μας, επιλέγοντας έναν κόμβο u βλέπουμε ότι χωρίζει το δέντρο σε ένα ΄άνω΄ και σ΄ ένα ΄κάτω΄ μέρος. Ιδανικά, θα θέλαμε να βρούμε έναν τρόπο γνωρίζοντας την λύση του προβλήματος για τα ύπόδεντρα΄ του u (δηλαδή για το κάτω μέρος) να την επεκτείνουμε ώστε να περιλαμβάνει και τον u.
- Είναι λοιπόν δελεαστικό να ορίσουμε ως χώρο καταστάσεων $dp[u]$, το βέλτιστο κάλυμα κορυφών για το υπόδεντρο που ορίζεται απ΄ τον u.

Μερικές αναγκαστικές διορθώσεις

- Πάντα είναι καλό να ξεκινάμε την λύση μας απ΄ τον πιο απλό χώρο καταστάσεων που υπάρχει και σταδιακά, αν βλέπουμε πως η πληροφορία που περιλαμβάνει δεν είναι αρκετή, να τον επεκτείνουμε με επιπλέον μεταβλητές.
- Αυτό κάναμε και εδώ, ξεκινήσαμε από έναν πολύ απλό χώρο καταστάσεων και τώρα έχει νόημα να δούμε αν ήταν καλή επιλογή.
- Αν επιχειρήσουμε να ορίσουμε μια αναδρομική σχέση για κάποιον αυθαίρετο κόμβο, έστω u, ανεξάρτητα απ΄ το πως θα μοιάζει αυτή η σχέση, θα κοιτάει τους γείτονες του u (τα παιδία του ουσιαστικά αφού έχουμε ριζώσει το δέντρο).
- Έστω v ένα από αυτά. Αν πάμε κάπως να χρησιμοποιήσουμε το $dp[v]$ προκειμένου να υπολογίσουμε το $dp[u]$ παρατηρούμε πως ενώ γνωρίζουμε ποιά είναι η βέλτιστη λύση για το υπόδεντρο του v δεν γνωρίζουμε αν αυτή η βέλτιστη λύση περιλαμβάνει ή όχι τον κόμβο v.

Μερικές αναγκαστικές διορθώσεις - μέρος 2

- Για να είμαστε σίγουροι πως η λύση μας είναι εφικτή θα έπρεπε κάθε φορά να βάζουμε τον κόμβο u για να είμαστε σίγουροι πως η ακμή (u, v) καλύπτεται. Αυτό όμως θα μας έδινε πάντα για λύση N κόμβους που φυσικά δεν είναι βέλτιστο.
- Πρέπει λοιπόν με κάποιον τρόπο να κρατάμε 2 λύσεις για κάθε κόμβο u. Μια που να μας δίνει την βέλτιστη απάντηση αν κάποιος μας ανάγκαζε να συμπεριλάβουμε τον κόμβο μέσα στο κάλυμα και μια που να μας δίνει την βέλτιστη απάντηση αν κάποιος μας ανάγκαζε να μην τον συμπεριλάβουμε.
- Έστω $dp[u, 1]$ η βέλτιστη λύση αν είμαστε αναγκασμένοι να συμπεριλάβουμε τον κόμβο u και $dp[u, 0]$ η βέλτιστη λύση αν δεν συμπεριλάβουμε τον κόμβο u.
- Αν κάπως μπορούμε να υπολογίσουμε για όλους τους κόμβους αύτη την πληροφορία. Η λύση του προβλήματος είναι απλά το $min(dp[1, 0], dp[1, 1])$ αφού η ρίζα του δέντρου είτε θα περιλαμβάνεται είτε δεν θα περιλαμβάνεται στην τελική λύση. Προφανώς επιλέγουμε το καλύτερο απ΄ τα δύο ενδεχόμενα.

Ορίζοντας τις αναδρομικές σχέσεις

- Ξεκινάμε με τον υπολογισμό του $dp[u, 0]$. Έστω πως έχουμε υπολογίσει όλες τις πληροφορίες για κάθε κόμβο στο υπόδεντρο του u και έστω επίσης $v_1, v_2, ..., v_k$ τα παιδιά του.
- Το πρώτο μας μέλημα είναι πως θα πρέπει να καλύψουμε τις ακμές $(u, v_1), (u, v_2), ..., (u, v_k)$ εφ' όσον το $dp[u, 0]$ μας αναγκάζει να μην συμπεριλάβουμε την κορυφή u στην λύση μας αυτό σημαίνει πως πρέπει αναγκαστικά να συμπεριλάβουμε όλους τους κόμβους $v_1, v_2, ..., v_k$ αφού αυτοί είναι οι μόνοι άλλοι υποψήφιοι για να καλύψουν τις εν λόγω ακμές.
- Θέλουμε όμως για κάθε υπόδεντρο που ορίζεται απ' τους κόμβους $v_i, 1 \leq k$ να το καλύψουμε βέλτιστα σιγουρεύοντας πως κάθε κόμβος v_i περιλαμβάνεται στην λύση.
- Αυτό όμως είναι εύκολο απ' τον τρόπο που ορίσαμε τα υποπροβλήματα μας, αρκεί να πάρουμε την λύση που δίνεται απ' το $dp[v_i, 1]$.

Καταλήγοντας στην πρώτη αναδρομική σχέση

- Απ΄ τις παρατηρήσεις που κάναμε παραπάνω είναι εύκολο να ορίσουμε την αναδρομική σχέση για το $dp[u, 0]$, πρέπει απλά να πάρουμε όλους τους κόμβους που δίνονται απ΄ την βέλτιστη λύση $dp[v_i, 1]$ για κάθε $v_i \in children(u)$. Έχουμε λοιπόν:

$$dp[u, 0] = \sum_{i=1}^{k} dp[v_i, 1] \tag{7}$$

- Είναι σαφές πως προκειμένου αυτό να είναι καλά ορισμένο πρέπει να υπολογίζουμε τα υποπροβλήματα ξεκινώντας απ΄ τα φύλλα του δέντρου ανεβαίνοντας προς την ρίζα. Αυτό μας το δίνει μια διάταξη που ορίζεται απ΄ την αναζήτηση κατά βάθος.

Γράφοντας και την δεύτερη αναδρομική σχέση

- Έχοντας κάνει όλα τα προηγούμενα είναι εύκολο τώρα να ορίσουμε την αναδρομική σχέση για τον υπογισμό του $dp[u,1]$.
- Εφ' όσον είμαστε αναγκασμένοι να διαλέξουμε τον κόμβο u σίγουρα θα πληρώσουμε 1 αλλά οι ακμές $(u,v_i), 1 \leq i \leq k$ είναι ήδη καλυμένες. Συνεπώς έχουμε το περιθώριο να επιλέξουμε αν θα βάλουμε στην λύση τον κάθε κόμβο v_i.
- Στην ουσία δηλαδή πρέπει να επιλέξουμε για κάθε παιδί v_i του u είτε το $dp[v_i,0]$ είτε το $dp[v_i,1]$. Λόγω της αρχής της βελτιστότητας (που παρεπιμπτόντως είναι το πρώτο και μοναδικό σημείο που την επικαλούμαστε σ' αυτό το πρόβλημα) θα διαλέξουμε εκείνο που μας δίνει την λύση με τους λιγότερους κόμβους. Το $min(dp[v_i,0], dp[v_i,1])$ δηλαδή.
- Συνδυάζοντας όλα τα παραπάνω λαμβάνουμε:

$$dp[u,1] = 1 + \sum_{i=1}^{k} min(dp[v_i,0], dp[v_i,1]) \tag{8}$$

Ανάλυση πολυπλοκότητας

- Έχουμε βρει μια λύση δυναμικού προγραμματισμού για το πρόβλημά μας η οποία είναι ορθή. Το μόνο που μένει είναι να δούμε την πολυπλοκότητα της λύσης μας.
- Έχουμε O(N) υποπροβλήματα (2 για κάθε κορυφή). Για κάθε κορυφή u κάνουμε $O(deg(u))$ για να υπολογίσουμε τα υποπροβλήματά της αφού κοιτάμε τα παιδία του μόνο και παίρνουμε σε σταθερό χρόνο την πληροφορία που μας δίνουν.
- Συνεπώς, η συνολική πολυπλοκότητα του αλγορίθμου θα είναι το άθροισμα του ΄χρόνου΄ που κάνουμε για κάθε κόμβο, οπότε:

$$\sum_{u \in V} O(deg(u)) = O(N) \tag{9}$$

Από λήμμα χειραψιών.

- Συνεπώς η συνολική πολυπλοκότητα του αλγορίθμου μας είναι γραμμική, όπως υποσχεθήκαμε και στην αρχή.

Το γενικότερο πρόβλημα

- Επιστρέφουμε στο αρχικό πρόβλημα το οποίο μας ζητάει να βρούμε το ελάχιστο πλήθος κορυφών για να καλύψουμε ακριβώς K ακμές.
- Βλέπουμε πως για K = N - 1 παίρνουμε το προηγούμενο πρόβλημα που συζητήσαμε, συνεπώς αποτελεί γενίκευση του Vertex Cover .
- Ωστόσο, θα βρούμε έναν αλγόριθμο πολυωνυμικού χρόνου ως προς το N και το K.
- Θα χρησιμοποιήσουμε στοιχεία της προηγούμενης λύσης, προσαρμόζοντάς την ώστε να είμαστε σίγουροι πως καλύπτουμε ακριβώς K ακμές.

Οδεύοντας προς τον χώρο καταστάσεων

- Αν χρησιμοποιούσαμε τον χώρο καταστάσεων του απλού προβλήματος, παρατηρούμε πως δεν γνωρίζουμε καθόλου πόσες ακμές έχουμε καλύψει στο δέντρο.
- Συνεπώς πρέπει κάπως να θυμόμαστε πόσες ακμές έχουμε καλύψει μέχρι στιγμής και ακριβώς αυτή την πληροφορία θα εισάγουμε στον χώρο καταστάσεων.
- Ορίζουμε $dp[u, k, 0], dp[u, k, 1]$ ακριβώς όπως στο προηγούμενο πρόβλημα με τον επιπλέον περιορισμό ότι έχουμε καλύψει ακριβώς k ακμές στο υπόδεντρο του u.
- Ένα βασικό πρόβλημα που αντιμετωπίζουμε τώρα είναι πως προκειμένου να λύσουμε κάποιο απ' τα υποπροβλήματα πρέπει να εργαστούμε όπως πριν. Να υποθέσουμε δηλαδή ότι γνωρίζουμε την απάντηση για τα παιδιά του u και κάπως να μοιράσουμε τις k ακμές στα υπόδεντρα.
- Όμως, υπάρχουν εκθετικά πολλοί τρόποι να μοιράσουμε τις ακμές στα υπόδεντρα και συνεπώς δεν μπορούμε να τους δοκιμάσουμε όλους. Πρέπει να δουλέψουμε βάσει βελτιστότητας.

Μια αντιστοιχία με ένα γνωστό πρόβλημα

- Στην γενική περίπτωση όπου θέλουμε να υπολογίσουμε κάποιο υποπρόβλημα $dp[u, k, 0/1]$ πρέπει με κάποιον τρόπο να ΄μοιράσουμε΄ τις k ακμές στα παιδιά του u, έστω $v_1, v_2, ..., v_m$.
- Μπορούμε να αντιμετωπίσουμε το πρόβλημα του βέλτιστου διαμοιρασμού των ακμών σαν ένα πρόβλημα σακιδίου.
- Θα δώσουμε τον τρόπο υπολογισμού του $dp[u, k, 0]$, ο τρόπος υπολογισμού του $dp[u, k, 1]$ είναι παρόμοιος.
- Σε αυτό το σημείο να αποσαφηνίσουμε κάτι. Όταν λέμε πως θέλουμε να μοιράσουμε τις k ακμές στα παιδιά του u εννοούμε πως θέλουμε να να βρούμε κατάλληλα $k_1, k_2, ..., k_m$ τέτοια ώστε $\sum_{i=1}^{m} k_i = k - m$ (επειδή πρέπει να λάβουμε υπ΄ όψην και τις ακμές (u, v_i)) και η ποσότητα $\sum_{i=1}^{m} dp[v_i, k_i, 1]$ να ελαχιστοποιείται.
- Αυτή θα είναι τελική και η λύση μας για το $dp[u, k, 0]$.

Αντιμετωπίζοντας την εκθετικότητα

- Προφανώς ένας τρόπος να βρούμε τα εν λόγω k_i είναι να δοκιμάσουμε όλους τους πιθανούς διαχωρισμούς το οποίο φυσικά θα μας δώσει εκθετική πολυπλοκότητα.
- Εδώ να κάνουμε μια παρένθεση. Αν το δέντρο ήταν δυαδικό θα μπορούσαμε να λύσουμε αυτό το πρόβλημα εξαιρετικά απλά αφού οι πιθανοί διαχωρισμοί είναι k. Πράγματι, υπάρχει τρόπος να μετασχηματίσουμε το δέντρο σε δυαδικό και να λύσουμε ένα ισοδύναμο πρόβλημα, αλλά δεν θα μιλήσουμε γι΄ αυτή την τεχνική στην παρούσα φάση.
- Αυτό που θα κάνουμε για να αντιμετωπίσουμε αυτή την εκθετική φύση του προβλήματος είναι ό,τι κάνουμε συνήθως. Θα χρησιμοποιήσουμε την βελτιστότητα και θα δουλέψουμε με δυναμικό προγραμματισμό ξανά.
- Θα ορίσουμε δηλαδή έναν δεύτερο αλγόριθμο δυναμικού προγραμματισμού που μας χωρίζεις τις K ακμές βέλτιστα και από κει θα πάρουμε την τελική λύση για το υποπρόβλημα $dp[u, k, 0]$.

Ο έσωτερικός αλγόριθμος δυναμικού προγραμματισμού

- Ορίζουμε $div[i,k]$ ως το ελάχιστο πλήθος κόμβων που πρέπει να χρησιμοποιήσουμε αν πρέπει με τα πρώτα i παιδιά του κόμβου u να καλύψουμε συνολικά k ακμές.
- Αυτό ξαφνικά μας κάνει την ζωή πολύ εύκολη αφού, δεδομένου πως ξέρουμε τις απαντήσεις για τα υποπροβλήματα $div[i',k']$ για όλα τα $i' < i$ και $k' < k$ αρκεί να διαλέξουμε πόσες ακμές θα καλύπτει το υπόδεντρο του i-οστού παιδιού.
- Πάλι χρησιμοποιούμε την μαγική πληροφορία, αν κάποιος μας ανάγκαζε το i-οστό παιδί να καλύπτει l ακμές τότε η απάντηση στο πρόβλημα θα ήταν $div[i-1,k-l]+dp[v_i,l-1,1]$. Ο λόγος που βάζουμε αυτό το $l-1$ είναι επειδή από τις k ακμές καλύπτουμε και την ακμή (u,v_i) η οποία είναι μέρος των k ακμών αφού υπολογίζουμε το υποπρόβλημα $dp[u,k,0]$.

Χάνοντας πάλι την μαγική πληροφορία

- Επειδή, όπως είπαμε και στην λύση του προηγούμενου προβλήματος, δεν μπορούμε να ξέρουμε πόσες ακμές θα καλύψει το υπόδεντρο κάθε παιδιού, δοκιμάζουμε όλους τους πιθανούς αριθμούς και εν τέλει διαλέγουμε αυτόν που μας ελαχιστοποιεί την απάντηση.
- Είναι επομένως:

$$div[i,k] = min_{l<k}(div[i-1,k-l] + dp[v_i,l]) \tag{10}$$

- Έτσι, μπορούμε να υπολογίσουμε όλο τον πίνακα div και η τελική απάντηση είναι $dp[u,k,0] = div[m,k]$.

Ανάλυση πολυπλοκότητας

- Έχουμε πολλές πληροφορίες και πολλούς αλγορίθμους οπότε πρέπει να είμαστε προσεκτικοί.
- Αρχικά για κάθε κόμβο u υπολογίζουμε K φορές έναν πίνακα div που έχει το πολύ $deg(u) * K$ υποπροβλήματα.
- Συνεπώς, αρκεί να αθροίσουμε την δουλειά που κάνουμε για κάθε κόμβο και λαμβάνουμε:

$$\sum_{u \in V} deg(u) * K * K = 2N * K^2 = O(NK^2) \tag{11}$$

- Συνεπώς, η τελική πολυπλοκότητα του αλγορίθμου μας είναι $O(NK^2)$ η οποία είναι αρκετά μεγαλύτερη απ΄ αυτή του αρχικού προβλήματος αλλά παραμένει πολυωνυμική.
- Σημείωση: Το εν λόγω πρόβλημα όπως είδαμε είναι στο P για δέντρα αλλά ακόμα και για διμερείς γράφους είναι NP-Complete . Το αν υπάρχει καλύτερος αλγόριθμος (για δέντρα) απ΄ αυτόν που περιγράψαμε είναι ενδιαφέρον ερώτημα.