Αλγόριθμοι Δυναμικού Προγραμματισμού

Άγγελος Μαντζαφλάρης, amantzaf@math.uoa.gr

1 Σύνοψη θεωρίας

Ο Δυναμικός προγραμματισμός είναι μία μέθοδος επίλυσης προβλημάτων βελτιστοποίησης, η οποία βασίζεται σε μια νοερή απαρίθμηση των λύσεων. Για να εφαρμοσθεί με χρησιμότητα, απαιτεί τα επεξεργαζόμενα προβλήματα να έχουν μια ειδική δομή, σειριακού τύπου.

Πολλά συνδυαστικά προβλήματα μπορούν να γραφούν υπό αυτή τη μορφή και να επιλυθούν έτσι με δυναμικό προγραμματισμό.

Η βασική αρχή του δυναμικού προγραμματισμού είναι:

- Εμφυτεύουμε το πρόβλημα μέσα σε μια οικογένεια προβλημάτων της ίδιας φύσης.
- Συνδέουμε με μια αναδρομική σχέση τις βέλτιστες λύσεις αυτών των προβλημάτων.

Η εύρεση των κατάλληλων υποπροβλημάτων και η αναδρομική σύνδεσή τους συχνά απαιτούν πειραματισμό και αρκετή εξοικείωση με το πρόβλημα που αντιμετωπίζεται. Υπάρχουν όμως μερικές συνήθεις επιλογές υποπροβλημάτων που εμφανίζονται συχνά σε διάφορα προβλήματα:

α) Η είσοδος είναι $a_1, a_2, \ldots, a_n$ και ένα υποπρόβλημα είναι έχει είσοδο $a_1, \ldots, \alpha_i$ για κάθε $i = 1 \ldots n$. Ο αριθμός των υποπροβλημάτων είναι n, δηλαδή ο αλγόριθμος που λύνει ένα τέτοιο πρόβλημα γεμίζει ένα μονοδιάστατο πίνακα και διαβάζει τη βέλτιστη λύση από το τελευταίο στοιχείο του.

β) Η είσοδος είναι $a_1, a_2, \ldots, a_n$ και $b_1, b_2, \ldots, b_m$. Τα υποπροβλήματα είναι $a_1, \ldots, \alpha_i$ και $b_1, \ldots, b_j$ για κάθε i, j. Ο αριθμός των υποπροβλημάτων είναι mn, δηλαδή ο αλγόριθμος που λύνει ένα τέτοιο πρόβλημα γεμίζει έναν πίνακα διάστασης $m \times n$ και διαβάζει τη βέλτιστη λύση από το κάτω δεξιά στοιχείο του. Ενδεχομένως να μη χρειάζεται να αποθηκεύουμε όλον αυτόν τον πίνακα, αλλά μόνο μερικές γραμμές του, αν π.χ. τα υποπροβλήματα που λύνονται στην i γραμμή χρειάζονται τις λύσεις μόνο των υποπροβλημάτων της $i-1$ γραμμής, αρκούν δυο διανύσματα μήκους n στη μνήμη για να λύσουμε το πρόβλημα.

γ) Η είσοδος είναι $a_1, a_2, \ldots, a_n$ και τα υποπροβλήματα είναι $a_i, \ldots, a_j$ όπου $1 \leq i \leq j \leq n$. Ο αριθμός των υποπροβλημάτων είναι n^2 και η βέλτιστη λύση δίνεται για $i = 1,\ j = n$.

Στο δυναμικό προγραμματισμό γράφουμε μια αναδρομική σχέση η οποία εκφράζει μεγάλα προβλήματα συναρτήσει μικρότερων υποπροβλημάτων, ούτως ώστε η επίλυση όλο και μεγαλύτερου μεγέθους υποπροβλημάτων να μας οδηγήσει τελικά στη λύση του αρχικού προβλήματος. Η αναδρομική αυτή σχέση μπορεί να ερμηνευθεί σαν ένας αναδρομικός αλγόριθμος, όμως ένας τέτοιος αλγόριθμος δεν θα ήταν αποδοτικός, καθώς το ίδιο υποπρόβλημα θα λυνόταν σε πολλές φορές σε διαφορετικές κλήσεις της αναδρομικής ρουτίνας. Σκεφτείτε ότι έχουμε ένα πρόβλημα του α) τύπου: η κλήση για το a_{i-1} λύνει αναδρομικά $i-1$ υποπροβλήματα, ενώ κατά την κλήση για το a_i λύνονται i υποπροβλήματα εκ των οποίων τα $i-1$ έχουν υπολογιστεί πριν. Αυτήν την επανάληψη των υποπροβλημάτων είναι που εκμεταλεύεται ο δυναμικός προγραμματισμός, γι αυτό και συνήθως δεν υλοποιούμε τα προγράμματα με αναδρομικές ρουτίνες, απλώς γεμίζουμε σειριακά ένα διάνυσμα(ή πίνακα) συστηματικά με τις λύσεις των υποπροβλημάτων ούτως ώστε αυτές να είναι διαθέσιμες κατά την επίλυση των επόμενων υποπροβλημάτων.

Σε μερικές περιπτώσεις είναι αποδοτικό να χρησιμοποιήσουμε μια έξυπνη αναδρομική ρουτίνα, η οποία διαβάζει κάθε φορά τις ήδη υπάρχουσες λύσεις και δεν λύνει ποτέ για δεύτερη φορά το ίδιο υποπρόβλημα. Αυτή η τακτική αποδεικνύεται αποδοτική όταν για την τελική λύση ενδέχεται να μη χρειαστούν όλα τα δυνατά αποτελέσματα των υποπροβλημάτων, παρά μόνο κάποια από αυτά. Η αναδρομή θα επιλύσει ακριβώς όποιο υποπρόβλημα είναι απαραίτητο για την εύρεση της τελικής λύσης, αντίθετα από μια σειριακή επίλυση όλων των δυνατών υποπροβλημάτων, στην οποία θα προέβαινε η συνήθης υλοποίηση ενός αλγορίθμου δυναμικού προγραμματισμού. Η τεχνική αυτή καλείται *αναδρομή με μνήμη* (memoization) και πρέπει να χρησιμοποιείται με προσοχή, καθώς όταν οι αναδρομικές κλήσεις είναι πολλές ο χρόνος εκτέλεσης είναι μεγάλος.

2 Εφαρμογές

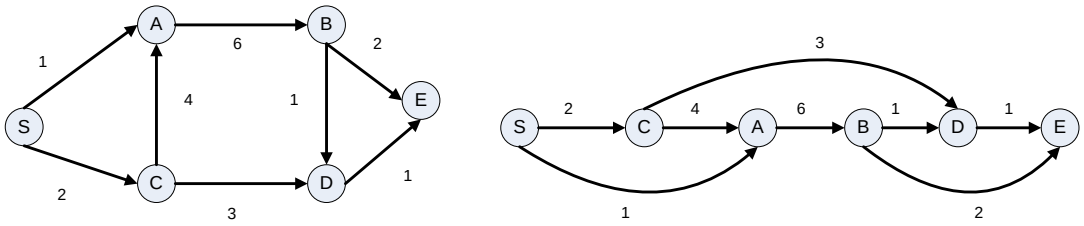
2.1 Αναζήτηση συντομότερου μονοπατιού

2.1.1 Συντομότερα μονοπάτια σε DAG

Το ειδικό χαρακτηριστικό ενός κατευθυνόμενου ακυκλικού γράφου (Directed Acyclic Graph), είναι ότι οι κόμβοι μπορούν να τοποθετηθούν σε σειρά, έτσι ώστε όλες οι ακμές να πηγαίνουν από αριστερά προς τα δεξιά. Μια τέτοια τοποθέτηση λέγεται *γραμμικοποίηση*(linearization) του γράφου. Για να δούμε πόσο βοηθάει αυτή η αλλαγή της απεικόνισης του γράφου, θα εξετάσουμε ένα παράδειγμα: Θέλουμε να βρούμε το συντομότερο μονοπάτι από τον κόμβο S έως το D. Για να φτάσουμε στο D πρέπει να περάσουμε ή από τον C ή από τον B, επομένως για να βρούμε το συντομότερο μονοπάτι π_D για το D, πρέπει να συγκρίνουμε απλώς αυτές τις δύο διαδρομές:

$$\pi_D = \min\{\pi_B + 1,\ \pi_C + 3\}$$

Μια παρόμοια σχέση μπορεί να γραφτεί για κάθε κόμβο. Αν υπολογίσουμε τις τιμές των αποστάσεων με διάταξη από αριστερά προς δεξιά, μπορούμε να είμαστε πάντα σίγουροι



Σχήμα 1: Ένας DAG και η γραμμικοποίησή του

ότι όταν φτάσουμε στον κόμβο u έχουμε όλες τις πληροφορίες για να υπολογίσουμε την π_u. Για να υπολογίσουμε όλες τις αποστάσεις σε ένα πέρασμα έχουμε τον παρακάτω αλγόριθμο:

1. Θέτουμε $\pi_u = \infty$ για όλες τις κορυφές u πλην της s, και $\pi_s = 0$.
2. Για κάθε κορυφή u σε γραμμικοποιημένη διάταξη, $\pi_u = \min\{\pi_u + l_{uv}\}$ για κάθε ακμή (u, v)

Αυτός ο αλγόριθμος λύνει ένα σύνολο από υποπροβλήματα, την εύρεση συντομότερων μονοπατιών από την s σε όλο και πιο απομακρυσμένες κορυφές. Ξεκινάμε με το μικρότερο από αυτά, το π_s, το οποίο ξέρουμε ότι είναι μηδέν. Προχωράμε σταδιακά σε μεγαλύτερα υποπροβλήματα. Θεωρούμε ένα υποπρόβλημα μεγάλο, αν πρέπει να λύσουμε πολλά υποπροβλήματα μέχρι να λύσουμε το υποπρόβλημα που θέλουμε.

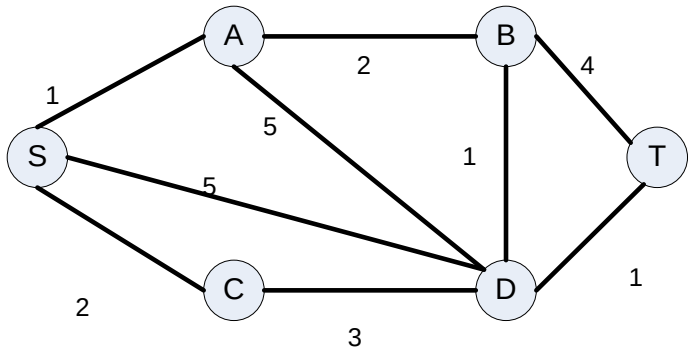
Αυτή είναι μια γενική τεχνική. Σε κάθε κόμβο υπολογίζουμε μια συνάρτηση με τιμές από τους προηγούμενους κόμβους. Στο παράδειγμά μας, η συνάρτηση είναι το ελάχιστο άθροισμα, αλλά θα μπορούσε να είναι το μέγιστο (σε περίπτωση που ψάχναμε το μεγαλύτερο μονοπάτι) ή θα μπορούσαμε να έχουμε το γινόμενο στην περίπτωση που ψάχναμε το μονοπάτι με το μικρότερο γινόμενο των μηκών των ακμών.

Ο δυναμικός προγραμματισμός είναι ένα πολύ ισχυρό αλγοριθμικό παράδειγμα όπου ένα πρόβλημα λύνεται, αναγνωρίζοντας ένα σύνολο υποπροβλημάτων και προσεγγίζοντάς τα ένα ένα αρχίζοντας από το μικρότερο, χρησιμοποιώντας τις απαντήσεις από το μικρό πρόβλημα για να βρούμε τις απαντήσεις για το μεγαλύτερο πρόβλημα, μέχρι να φτάσουμε σε αυτό που θέλουμε να λύσουμε. Στον δυναμικό προγραμματισμό υπάρχει πάντα ένας νοητός γράφος DAG. Οι κόμβοι του είναι τα υποπροβλήματα που ορίζουμε και οι ακμές του συσχετισμοί μεταξύ των υποπροβλημάτων. Αν για να λύσουμε ένα υποπρόβλημα B πρέπει να λύσουμε προηγουμένως το υποπρόβλημα A, υπάρχει μια ακμή από το A στο B. Σε αυτή την περίπτωση το A καλείται ένα υποπρόβλημα του B.

2.1.2 Συντομότερα μονοπάτια με k ακμές

Ενδεχομένως να μας ενδιαφέρουν κι άλλοι παράγοντες για την επιλογή ενός μονοπατιού, εκτός άπό το συνολικό μήκος. Σε ένα δίκτυο τηλεπικοινωνιών, για παράδειγμα, αν το μήκος της κάθε ακμής αντιστοιχεί στην καθυστέρηση μετάδοσης του σήματος,

μπορεί κάθε έξτρα ακμή να αποτελεί κίνδυνο για να χάνονται πακέτα. Σε μια τέτοια περίπτωση θα θέλαμε να αποφύγουμε κάποιο μονοπάτι με πολλές ακμές. Στην παρακάτω εικόνα, βλέπουμε ένα γράφο για τον οποίο το μικρότερο μονοπάτι από το S στο T έχει 4 ακμές, ενώ υπάρχει άλλο μονοπάτι, μεγαλύτερο αλλά με 2 ακμές.



Αν στο μονοπάτι με τις 4 ακμές έχουμε προβλήματα αξιοπιστίας της μετάδοσης, τότε θα πρέπει να κατευθύνουμε τα πακέτα στο μονοπάτι με τις 2 ακμές.

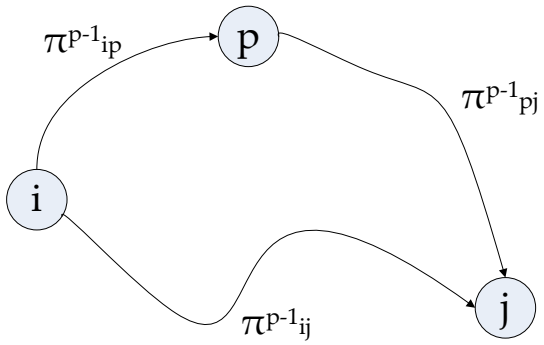
Το γενικό πρόβλημα είναι η εύρεση ενός μονοπατιού από την κορυφή s στην κορυφή t με ελάχιστο μήκος, το οποίο χρησιμοποιεί το πολύ k ακμές. Στον δυναμικό προγραμματισμό, πρέπει να διαλέγουμε τα υποπροβλήματα έτσι ώστε όλες οι απαιτούμενες πληροφορίες να είναι διαθέσιμες όταν λύνουμε μεγαλύτερα υποπροβλήματα. Σε αυτή την περίπτωση, θα ορίσουμε για κάθε κορυφή u και κάθε ακέραιο $i \leq k$, π_u^i να είναι το μήκος του μικρότερου μονοπατιού από την s στην u που χρησιμοποιεί ακριώς i ακμές. Αρχικά η τιμή π_u^0 είναι άπειρο για κάθε κορυφή, εκτός από την s, που είναι μηδέν. Η αναδρομική σχέση είναι:

$$\pi_u^i = \min_{(u,v)\in E} \left\{ \pi_v^{i-1} + l_{uu} \right\}$$

και ο αντίστοιχος αλγόριθμος είναι εύκολο να γραφεί.

2.1.3 Συντομότερα μονοπάτια όλων των ζευγών

Έστω ότι θέλουμε να βρούμε το ελάχιστο μονοπάτι, όχι μόνο μεταξύ του s και του t, αλλά μεταξύ όλων των ζευγαριών κορυφών του V. Μια ιδέα είναι η εξής: το ελάχιστο μονοπάτι $u \to w_1 \to \cdots \to w_m \to v$ μεταξύ του u και v, χρησιμοποιεί μερικούς ενδιάμεσους κόμβους, ίσως και κανέναν. Αν απαγορεύσουμε να έχουμε ενδιάμεσους κόμβους, τότε το πρόβλημα λύνεται άμεσα γιατί το ελάχιστο μονοπάτι από τον κόμβο u στο v είναι η απευθείας ακμή που τους συνδέει, αν αυτή υπάρχει. Αν αλλάζουμε σταδιακά το επιτρεπτό σύνολο των ενδιάμεσων κόμβων, προσθέτοντας έναν κόμβο σε κάθε βήμα, τότε θα φτάσουμε στο σύνολο V όλων των κόμβων και άρα στην λύση που θέλουμε. Πιο συγκεκριμένα, αριθμούμε τις κορυφές $\{1, 2, \ldots, n\}$ και ορίζουμε π_{ij}^k να είναι το μήκος του ελάχιστου μονοπατιού από τον κόμβο i στον j, στο οποίο μόνο οι κόμβοι $\{1, 2, \ldots, k\}$ μπορούν να χρησιμοποιηθούν ως ενδιάμεσοι. Αρχικά π_{ij}^0 είναι το μήκος της απευθείας ακμής μεταξύ των i και j, αν υπάρχει, αλλιώς το άπειρο.



Σχήμα 2: Η ιδέα του αλγορίθμου Floyd - Warshall

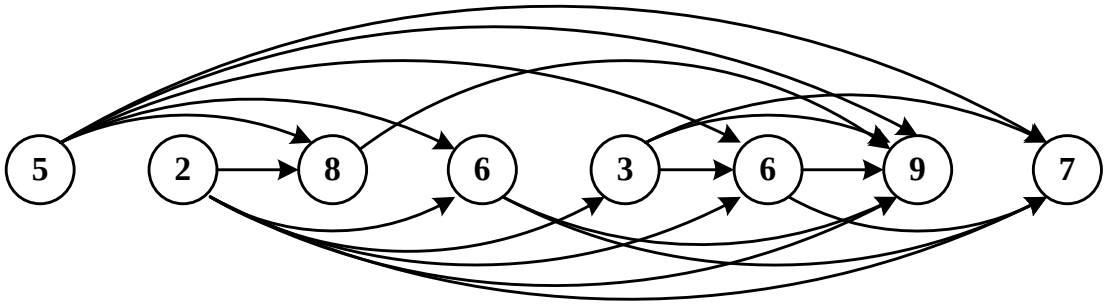
Όταν επεκτείνουμε το σύνολο των ενδιάμεσων κόμβων, με την προσθήκη ενός επιπλέον κόμβου p, πρέπει να εξετάσουμε όλα τα ζευγάρια i, j και να ελέγξουμε αν χρησιμοποιώντας το p ως ενδιάμεσο κόμβο παίρνουμε ένα μικρότερο μονοπάτι από το i στο j. Αυτό όμως είναι εύκολο, αφού το ελάχιστο μονοπάτι από το i στο j που χρησιμοποιεί το p, περνάει μόνο μια φορά από το p (υποθέτουμε ότι δεν έχουμε αρνητικούς κύκλους) και έχουμε ήδη υπολογίσει το μήκος του ελάχιστου μονοπατιού από το i στο p και από το p στο j χρησιμοποιώντας μόνο κορυφές με μικρότερη αρίθμηση. Με τον κόμβο p έχουμε μικρότερο μονοπάτι από το i στο j αν και μόνο αν $\pi_{ip}^{p-1} + \pi_{pj}^{p-1} < \pi_{i,j}^{p-1}$. Αυτός ο αλγόριθμος δυναμικού προγραμματισμού έχει πολυπλοκότητα $O(n^3)$ και είναι γνωστός ως αλγόριθμος των Floyd - Warshall:

1. Θέσε $\pi_{ij}^0 = \infty$ για όλα τα i, j
2. Για κάθε $(i, j) \in E$ θέσε $\pi_{ij}^0 = l_{ij}$
3. Για $p = 1 \ldots n$
   Για $i = 1 \ldots n$
   Για $j = 1 \ldots n$
   $\pi_{ij}^p = \min\left\{\pi_{ip}^{p-1} + \pi_{pj}^{p-1},\ \pi_{i,j}^{p-1}\right\}$

2.2 Μακρύτερη αύξουσα υπακολουθία

Μας δίνεται μια ακολουθία αριθμών $a_1, a_2, \ldots, a_n$. Μια υπακολουθία αυτής είναι μια ακολουθία $a_{i_1}, a_{i_2}, \ldots, a_{i_k}$ με $1 \leq i_1 < i_2 < \cdots < i_k \leq n$. Αναζητούμε μια (γνήσια) αύξουσα υπακολουθία με μέγιστο μήκος. Για παράδειγμα η μακρύτερη αύξουσα υπακολουθία της $5, 2, 8, 6, 9, 7$ είναι η $2, 3, 6, 9$. Για να καταλάβουμε καλύτερα το χώρο των λύσεων σχηματίζουμε το γράφο $G = (V, E), |V| = n$ όλων των δυνατών αυξουσών υπακολουθιών.

Παρατηρούμε ότι πρόκειται για κατευθυνόμενο ακυκλικό γράφο(DAG) καθώς για κάθε ακμή (i, j) έχουμε $i < j$. Επίσης οι αύξουσες υπακολουθίες αντιστοιχούν ακριβώς



Σχήμα 3: Ο γράφος των δυνατών αυξουσών υπακολουθιών

στα μονοπάτια αυτού του γράφου, άρα η εύρεση της μακρύτερης αύξουσας υπακολουθίας συνίσταται στην εύρεση του μακρύτερου μονοπατιού σε αυτόν το γράφο.

- Εμφυτεύουμε αυτό το πρόβλημα μέσα στην οικογένεια των n προβλημάτων: αναζήτηση τς μακρύτερης αύξουσας υπακολουθίας της $a_1, \ldots, a_i$ για $i = 1, 2, \ldots, n$.

- Αν L_i είναι το βέλτιστο ζητούμενο μήκος του υποπροβλήματος $a_1, \ldots, a_i$ βρίσκουμε την αναδρομική σχέση

$$L_i = 1 + \max\{L_j \ : \ (j, i) \in E\}$$

  δηλαδή το μήκος της μέγιστης υπακολουθίας ως το $i-$οστό στοιχείο είναι ο αριθμός των κόμβων του μέγιστου μήκους μονοπατιού του υπογράφου με κορυφές $a_1, \ldots, a_i$. Η αρχική συνθήκη είναι $L_1 = 1$ και κάθε επόμενο υποπρόβλημα εξαρτάται μόνο από τα προηγούμενα υποπροβλήματα που έχουν ήδη επιλυθεί.

Κατά τη διαδικασία επίλυσης των υποπροβλημάτων μπορούμε να διατηρούμε τους δείκτες j για τους οποίους επιτυγχάνεται η μέγιστη τιμή του L_i. Έτσι ακολουθώντας τους προδείκτες του L_n μπορεί να κατασκευαστεί η ζητούμενη υπακολουθία.

2.3 Μακρύτερη παλινδρομική υπακολουθία

Μια υπακολουθία είναι *παλινδρομική* αν παραμένει η ίδια όταν διαβαστεί από αριστερά προς τα δεξιά. Ο δυναμικός προγραμματισμός μας προσφέρει έναν $O(n^2)$ αλγόριθμο με είσοδο μια ακολουθία $X[1 \ldots n]$ και έξοδο το μήκος της μεγαλύτερης παλινδρομικής υπακολουθίας στην X.

Έστω $L_{ij},\ 1 \leq i \leq j \leq n$ το μήκος της μεγαλύτερης παλινδρομικής υπακολουθίας της $X[i \ldots j]$. Μια ακολουθία μήκους 1 είναι κατά τετριμμένο τρόπο παλινδρομική, άρα $L_{ii} = 1,\ 1 \leq i \leq n$. Μια ακολουθία μήκους 2 είναι παλινδρομική αν και μόνο αν οι χαρακτήρες ταυτίζονται άρα για δυο συνεχόμενους χαρακτήρες είναι $L_{i,i+1} = \delta_{i,i+1},\ 1 \leq i \leq n-1$, όπου

$$\delta_{ij} = \begin{cases} 2, & X[i] = X[j] \\ 0, & X[i] \neq X[j] \end{cases}$$

Για το L_{ij} διακρίνουμε τις εξής περιπτώσεις: Αν $X[i] = X[j]$ ($\delta_{ij} = 2$), τότε τα $X[i], X[j]$ ανήκουν στην υπακολουθία, άρα $L_{i,j} = L_{i+1,j-1} + \delta_{ij}$. Αν $X[i] \neq X[j]$ ($\delta_{ij} = 0$), τότε τουλάχιστον ένα από τα $X[i], X[j]$ δεν ανήκει στην υπακολουθία, άρα αυτή έχει μήκος το μεγαλύτερο από τα $L_{i+1,j}$ ή $L_{i,j-1}$. Τελικά

$$L_{ij} = \max\left\{L_{i+1,j},\ L_{i,j-1},\ L_{i+1,j-1} + \delta_{ij}\right\}$$

Αν $\delta_{ij} = 0$ τότε ο τρίτος όρος είναι μικρότερος ή ίσος από τους άλλους δυο, άρα η αναδρομική σχέση είναι σωστή.

Ο αλγόριθμος είναι:

1. Για $i = 1 \ldots n$ θέσε $L_{ii} = 1$.
2. Για $i = 1 \ldots n-1$ θέσε $L_{i,i+1} = \delta_{i,i+1}$.
3. Για $k = 2 \ldots n-1$
   Για $i = 1 \ldots n-k$
   θέσε $j = i + k$
   θέσε $L_{ij} = \max\left\{L_{i+1,j},\ L_{i,j-1},\ L_{i+1,j-1} + \delta_{ij}\right\}$
4. Επέστρεψε το L_{1n}.

Τα υποπροβλήματα έχουν πλήθος $O(n^2)$ και καθένα λύνεται με μια εξέταση συνθήκης(δ_{ij}) και την εύρεση του μεγίστου τριών αριθμών, τα οποία γίνονται σε σταθερό χρόνο. Άρα ο συνολικός χρόνος είναι $O(n^2)$.

2.4 Ανάθεση διαστημάτων με βάρη(weighted interval scheduling)

Στο πρόβλημα αυτό μας δίνονται n διαστήματα αριθμημένα $1, 2, \ldots, n$, πχ n χρονικά διαστήματα τα οποία διαφορετικοί πελάτες θέλουν να κλείσουν μια αίθουσα εκδηλώσεων. Κάθε διάστημα καθορίζεται από μια αρχική τιμή s_i, μια τελική τιμή f_i και ένα βάρος w_i, πχ τα χρήματα που προσφέρει ο πελάτης που ζητά την αίθουσα για αυτό το χρονικό διάστημα. Θέλουμε να επιλέξουμε ένα σύνολο $S \subseteq \{1, 2, \ldots, n\}$ διαστημάτων, τα οποία δεν επικαλύπτονται, ώστε να μεγιστοποιήσουμε το συνολικό κέρδος(βάρος) $\sum_{i \in S} w_i$.

Έστω ότι τα διαστήματα είναι ταξινομημένα κατά αύξουσα σειρά ως προς την τελική τιμή, $f_1 \leq f_2 \leq \cdots \leq f_n$. Για κάθε διάστημα i θεωρούμε $\alpha(i)$ το μεγαλύτερο δείκτη j ώστε τα διαστήματα i, j δεν επικαλύπτονται, δηλαδή το κοντινότερο διάστημα για το οποίο $f_j < s_i$. Θέτουμε $\alpha(j) = 0$ αν κανένα διάστημα $i < j$ δεν έχει πέρας μικρότερο από το s_i.

Για ένα στιγμιότυπο του προβλήματος, θεωρούμε τη βέλτιστη λύση S^*. Προφανώς το $n-$οστό διάστημα είτε ανήκει σε αυτήν, είτε όχι. Αν $n \in S^*$, τότε κανένα άλλο διάστημα i με $\alpha(n) < i < n$ δεν ανήκει σε αυτήν, καθώς, από τον ορισμό του $\alpha(n)$, γνωρίζουμε ότι τα $\alpha(n)+1, \alpha(n)+2, \ldots, n-1$ επικαλύπτουν το διάστημα n. Επίσης, το S^* θα περιέχει μια βέλτιστη λύση του υποπροβλήματος $1, 2 \ldots, \alpha(n)$, διότι διαφορετικά

θα μπορούσαμε να αντικαταστήσουμε κάποια διαστήματα του S^* με τη βέλτιστη λύση αυτού του υποπροβλήματος, χωρίς κίνδυνο να επικαλύψουμε το n, και θα είχαμε μια καλύτερη λύση. Με την ίδια λογική, αν $n \notin S^*$, τότε η βέλτιστη λύση του αρχικού προβλήματος είναι η ίδια με τη βέλτιστη λύση του συνόλου $\{1, 2, \ldots, n-1\}$. Η παρατήρηση αυτή μας οδηγεί στο δυναμικό προγραμματισμό:

- Εμφυτεύουμε το πρόβλημα μέσα στην οικογένεια των n προβλημάτων: αναζήτηση της βέλτιστης ανάθεσης στα διαστήματα $\{1, 2, \ldots, i\}$. Έστω W_i το βάρος της βέλτιστης λύσης($W_0 = 0$).

- Συνδέουμε τα υποπροβλήματα με την αναδρομική σχέση

$$W_i = \max\{w_i + W_{\alpha(i)}, W_{j-1}\}$$

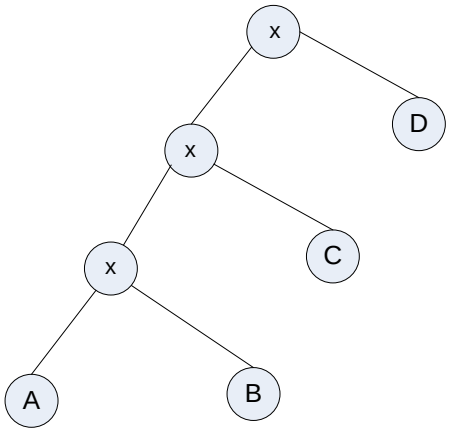
Όπως πάντα, μπορούμε εύκολα να βρούμε τα διαστήματα που επιτυγχάνουν το μέγιστο: το $i-$οστό διάστημα ανήκει στη λύση αν και μόνο αν $w_i + W_{\alpha(i)} \geq W_{j-1}$.

2.5 Πολλαπλασιασμός ακολουθίας πινάκων

Έστω ότι έχουμε να πολλαπλασιάσουμε 4 πίνακες, $A \times B \times C \times D$, διαστάσεων $50 \times 20,\ 20 \times 1,\ 1 \times 10,\ 10 \times 100$ αντίστοιχα. Αυτό σημαίνει ότι θα έχουμε επαναληπτικά, τον πολλαπλασιασμό δύο πινάκων κάθε φορά. Ο πολλαπλασιασμός πινάκων δεν είναι αντιμεταθετικός, αλλά είναι προσεταιριστικός: $A \times (B \times C) = (A \times B) \times C$. Επομένως μπορούμε να υπολογίσουμε το γινόμενο 4 πινάκων με αρκετούς διαφορετικούς τρόπους ανάλογα με τον τρόπο που τους βάζουμε στις παρενθέσεις. Για τον πολλαπλασιασμό ενός πίνακα $m \times n$ με έναν $n \times p$ χρειάζονται $m \cdot n \cdot p$ πολλαπλασιασμοί μεταξύ των στοιχείων τους. Ας συγκρίνουμε διάφορους τρόπους που βρίσκουν το $A \times B \times C \times D$, βάζοντας σε διαφορετικές θέσεις παρενθέσεις:

Παρενθέσεις	Πολλαπλασιασμοί	Τελικό κόστος
$A \times ((B \times C) \times D)$	$20 \cdot 1 \cdot 10 + 20 \cdot 10 \cdot 100 + 50 \cdot 20 \cdot 100$	120200
$(A \times (B \times C)) \times D$	$20 \cdot 1 \cdot 10 + 50 \cdot 20 \cdot 10 + 50 \cdot 10 \cdot 100$	60200
$(A \times B) \times (C \times D)$	$50 \cdot 20 \cdot 1 + 1 \cdot 10 \cdot 100 + 50 \cdot 1 \cdot 100$	7000

Η σειρά των πολλαπλασιασμών έχει μεγάλη διαφορά στον χρόνο εκτέλεσης. Πώς βρίσκουμε την καλύτερη σειρά αν θέλουμε να υπολογίσουμε το $A_1 \times A_2 \times \cdots \times A_n$ όπου A_i είναι πίνακες με διαστάσεις $m_0 \times m_1, m_1 \times m_2, \ldots, m_{n-1} \times m_n$; Το πρώτο που σκεφτόμαστε είναι ότι μια συγκεκριμένη εισαγωγή παρενθέσεων μπορεί να αναπαρασταθεί με ένα δυαδικό δέντρο όπου τα φύλλα είναι οι πίνακες, η ρίζα είναι το τελικό γινόμενο και οι εσωτερικοί κόμβοι είναι τα ενδιάμεσα γινόμενα. Οι πιθανές διατάξεις με τις οποίες κάνουμε τον πολλαπλασιασμό ανταποκρίνονται σε διάφορα πλήρη δυαδικά δέντρα. Δεν είναι σοφό να δοκιμάσουμε κάθε δέντρο, επομένως καταφεύγουμε στον δυναμικό προγραμματισμό.



Σχήμα 4: Το δέντρο του πολλαπλασιασμού $(((A \times B) \times C) \times D)$

Για να είναι ένα δέντρο βέλτιστο, πρέπει και τα υπο-δέντρα του να είναι βέλτιστα, επομένως πρέπει να βρούμε υπο-προβλήματα που σχετίζονται με τα υπο-δέντρα: Είναι τα γινόμενα της μορφής $A_i \times A_{i+1} \times \cdots \times A_j$. Για $1 \leq i \leq j \leq n$, ορίζουμε

$$C_{ij} = \text{ελάχιστο κόστος πολλαπλασιασμού των } \ A_i \times A_{i+1} \times \cdots \times A_j$$

Το μέγεθος αυτού του υπο-προβλήματος είναι ο αριθμός των πολλαπλασιασμών πινάκων που γίνονται, δηλαδή $j-i$. Το μικρότερο υποπρόβλημα είναι όταν $i = j$, όπου δεν έχουμε να πολλαπλασιάσουμε κάτι, άρα $C_{ii} = 0$. Για $j > i$ ας υποθέσουμε το βέλτιστο υπο-δέντρο C_{ij}. Ο πρώτος κλάδος σε αυτό το υπο-δέντρο, στην κορυφή, θα χωρίσει το γινόμενο σε δύο κομμάτια της μορφής $A_i \times A_{i+1} \times \cdots \times A_p$ και $A_{p+1} \times \cdots \times A_j$ για κάποιο p μεταξύ i και j. Το κόστος του υπο-δέντρου είναι το κόστος αυτών των δύο μερικών γινομένων και επιπλέον το κόστος του τελικού πολλαπλασιασμού: $C_{ip}+C_{p+1,j}+m_{i-1}m_pm_j$. Πρέπει να βρούμε το σημείο του χωρισμού p για το οποίο είναι ελάχιστο:

$$C_{ij} = \min_{i \leq p \leq j} \{C_{ik} + C_{k+1,j} + m_{i-1}m_pm_j\}$$

Έτσι καταλήγουμε στον παρακάτω αλγόριθμο(το s δηλώνει το μέγεθος του υποπροβλήματος):

1. Για $i = 1 \ldots n$, $C_{ii} = 0$

2. Για $s = 1 \ldots n-1$
   Για $i = 1 \ldots n-s$
   θέσε $j = i + s$
   $C_{ij} = \min_{i \leq p \leq j} \{C_{ik} + C_{k+1,j} + m_{i-1}m_pm_j\}$

3. Επέστρεψε το C_{1n}

2.6 Αποτέλεσμα πολλαπλασιασμού

Ορίζουμε μια διαδικασία πολλαπλασιασμού τριών στοιχείων a, b, c σύμφωνα με τον παρακάτω πίνακα. Παρατηρήστε ότι η πράξη δεν είναι προσεταιριστική, ούτε αντιμεταθετική.

	a	b	c
a	b	b	a
b	c	b	a
c	a	c	c

Θέλουμε έναν αλγόριθμο, με είσοδο μια συμβολοσειρά από a, b, c, ο οποίος να αποφασίζει αν μπορούν να μπουν παρενθέσεις έτσι ώστε το αποτέλεσμα του πολλαπλασιασμού να είναι a.

Έστω $s_1 s_2 \cdots s_n$ η συμβολοσειρά, $s_i \in \{a, b, c\}$. Θεωρούμε $R_{i,j},\ 1 \le i \le j \le n$ όλα τα δυνατά αποτελέσματα που μπορούν να προκύψουν από το κομμάτι $s_i s_{i+1} \cdots s_j$. Αν $i = j$ έχουμε ένα μόνο σύμβολο δηλαδή $R_{i,i} = s_i$.

Έστω μια συμβολοσειρά $s_i \cdots s_{i+k}$. Μπορούμε να προσδιορίσουμε το $R_{i,i+k}$ αν γνωρίζουμε τα $R_{i,j}$ και $R_{j+1,i+k}$ για $j = i \ldots i+k-1$: Το ζητούμενο είναι όλα τα δυνατά γινόμενα $uv,\ u \in R(i,j),\ v \in R_{j+1,j+k}$, διότι (επειδή η πράξη δεν είναι προσεταιριστική) αυτά είναι τα αποτελέσματα που μπορούν να προκύψουν εισάγοντας παρενθέσεις σε οποιοδήποτε σημείο: $(s_i \cdots s_j)(s_{j+1} \cdots s_{i+k})$.

Ο αλγόριθμος είναι:

1. Για $i = 1 \ldots n$ θέσε $R_{i,i} = s_i$.

2. Για $k = 1 \ldots n-1$
   Για $i = 1 \ldots n-k$
   θέσε $R_{i,i+k} = \bigcup_{i \le j < i+k} \{uv \mid u \in R_{i,j},\ v \in R_{j+1,i+k}\}$

3. Αν $a \in R_{1,n}$ απάντησε ΝΑΙ αλλιώς απάντησε ΟΧΙ.

Τα υποπροβλήματα είναι $\frac{n(n+1)}{2} = O(n^2)$ και η επίλυση κάθε υποπροβλήματος στο βήμα 2 κοστίζει $O(k) = O(n)$. Άρα ο συνολικός χρόνος είναι $O(n^3)$.

2.7 Επιλογή ξενοδοχείων

Ξεκινάτε ένα ταξίδι. Αρχίζετε στο σημείο 0 μίλια του δρόμου. Κατά μήκος του δρόμου υπάρχουν n ξενοδοχεία σε απόσταση $a_1 < a_2 < \cdots < a_n$ μίλια από εσάς. Μπορείτε να σταματήσετε μόνο σε αυτά τα ξενοδοχεία, επιλέγοντας όμως σε ποια από αυτά θα σταματήσετε. Ο προορισμός σας είναι το τελευταίο ξενοδοχείο(σε απόσταση a_n). Το ιδανικό θα ήταν να διανύετε 200 μίλια την ημέρα, όμως αυτό ενδέχεται να μην είναι εφικτό(λόγω της θέσης των ξενοδοχείων). Αν ταξιδέψετε x μίλια κάποια μέρα, το κόστος που έχετε για τη μέρα αυτή είναι $(200 - x)^2$. Ο στόχος είναι να σχεδιάσετε το ταξίδι έτσι ώστε το συνολικό κόστος να ελαχιστοποιηθεί.

Έστω $a_0 = 0$ το σημείο εκκίνησης και K_i το ελάχιστο κόστος αν ο προορισμός είναι το $i-$οστό ξενοδοχείο. Ξεκινάμε με μηδενικό κόστος άρα $K_0 = 0$. Οι αποστάσεις που επιτρέπεται να διανύσουμε είναι $x = a_i - a_j$ για $i > j$. Έστω ότι έχουμε τη βέλτιστη διαδρομή ως το K_j, $\forall j < i$. Αν επιλέξουμε να σταματήσουμε στο ξενοδοχείο i το κόστος είναι $K_j + (200 - (a_i - a_j))^2$. Άρα

$$K_i = \min_{0 \le j < i} \left\{ K_j + (200 - (a_i - a_j))^2 \right\}$$

Ο αλγόριθμος είναι:

1. $K_0 = 0$
2. Για $i = 1, \ldots n$
   $K_i = \min_{0 \le j < i} \left\{ K_j + (200 - (a_i - a_j))^2 \right\}$
   $P(i) = j_0$, αν το min είναι για $j = j_0$
3. Τύπωσε τους προδείκτες του $P(n)$

Τα υποπροβλήματα που λύνονται (βήμα 2) είναι n και καθένα λύνεται με την εύρεση ενός μεγίστου(και μερικές πράξεις που γίνονται σε σταθερό χρόνο). Ο συνολικός αριθμός των συγκρίσεων είναι $1 + 2 + \cdots + n = O(n^2)$, ο οποίος είναι και ο χρόνος του αλγορίθμου.

2.8 Το πρόβλημα του Περιοδεύοντος Πωλητή

Ένα στιγμιότυπο για το πρόβλημα του Περιοδεύοντος Πωλητή αποτελείται από ένα σύνολο $V = \{1, 2, \ldots, n\}$ σημείων και μια συνάρτηση απόστασης $d : V \times V \to \mathbb{R}_+$ που ορίζει τι αποστάσεις d_{ij} και d_{ji} για τη μετάβαση από το σημείο i στο σημείο j και από το j στο i αντίστοιχα. Το ζητούμενο είναι μια περιοδεία (δηλαδή ένας κύκλος που διέρχεται από κάθε σημείο ακριβώς μία φορά) ελαχίστου μήκους. Χωρίς βλάβη της γενικότητας, θεωρούμε ότι οι αποστάσεις είναι θετικές και ότι απόσταση κάθε σημείου απ τον εαυτό του είναι μηδενική, δηλαδή $d_{ii} = 0$.

Αφού μια περιοδεία διέρχεται από κάθε σημείο ακριβώς μία φορά, μπορούμε να θεωρήσουμε ότι κάθε περιοδεία ξεκινά από το σημείο 1. Επειδή δεν υπάρχουν άλλοι περιορισμοί, κάθε μετάθεση των σημείων που διατηρεί το σημείο 1 στην πρώτη θέση συνιστά μια περιοδεία. Με άλλα λόγια, κάθε 1-1 και επί συνάρτηση $\pi : V \to V$με $\pi(1) = 1$ αναπαριστά μια περιοδεία η οποία επισκέπτεται το σημείο $\pi(i)$ στη σειρά i. Το συνολικό κόστος της περιοδείας είναι:

$$L_\pi = d_{\pi(n)1} + \sum_{i=1}^{n-1} d_{\pi(i)\pi(i+1)}$$

Υπάρχουν $(n-1)!$ διαφορετικές περιοδείες, όσες και οι διαφορετικές μεταθέσεις των σημείων $V - \{1\}$. Το γεγονός αυτό καθιστά απαγορευτική την εξαντλητική αναζήτηση για τη βέλτιστη περιοδεία. Θα διατυπώσουμε έναν αλγόριθμο δυναμικού προγραμματισμού

για το πρόβλημα. Ο αλγόριθμος, αν και έχει εκθετικό χρόνο εκτέλεσης, είναι σημαντικά ταχύτερος από την εξαντλητική αναζήτηση.

Έστω μια βέλτιστη περιοδεία που ξεκινά από το σημείο 1 και συνεχίζει στο σημείο j. Το μήκος της περιοδείας είναι d_{1j} συν το μήκος του τμήματος μονοπατιού της από το σημείο j στο 1. Επειδή πρόκειται για περιοδεία, το τμήμα της απ το j στο 1 διέρχεται από όλα τα υπόλοιπα σημεία (δηλαδή όλα τα σημεία του συνόλου $V - \{1, j\}$). Επειδή η περιοδεία είναι βέλτιστη, το τμήμα της από το j στο 1 είναι το συντομότερο μονοπάτι που συνδέει το j με το 1 διερχόμενο από όλα τα σημεία του $V - \{1, j\}$. Με άλλα λόγια, κάθε βέλτιστη περιοδεία ακολουθεί μόνο συντομότερα μονοπάτια (που διέρχονται από όλα τα υπόλοιπα σημεία) για να μεταβεί από τα ενδιάμεσα σημεία στο αρχικό.

Με βάση αυτήν την παρατήρηση, προσπαθούμε να διατυπώσουμε μια αναδρομική σχέση που θα περιγράφει τη βέλτιστη λύση. Έστω $S \subseteq V - \{1\}$ ένα υποσύνολο σημείων και έστω $i \in V - S$ ένα σημείο διαφορετικό από το αρχικό σημείο 1, εφόσον $V - S \neq \{1\}$. Συμβολίζουμε με L_{iS} το μήκος του συντομότερου μονοπατιού που συνδέει το σημείο i με το σημείο 1 διερχόμενο απ όλα τα σημεία του S. Προφανώς, αν $S = \emptyset$ (δεν έχει μείνει κανένα σημείο ακάλυπτο), είναι $L_{i\emptyset} = d_{i1}$ για κάθε $i \in V - \{1\}$. Αν $S \neq \emptyset$, τότε το μήκος του συντομότερου μονοπατιού που ξεκινάει από το i, καταλήγει στο 1, και διέρχεται από όλα τα σημεία του S δίνεται από την αναδρομική σχέση

$$L_{iS} = \min_{j \in S} \left\{ d_{ij} + L_{i,S-\{j\}} \right\}$$

Όταν $S = V - \{1\}$, θεωρούμε σαν σημείο εκκίνησης το σημείο 1, επειδή είναι το μοναδικό σημείο στο σύνολο $V - S$. Επομένως, το μονοπάτι ξεκινάει και καταλήγει στο σημείο 1 διερχόμενο απ όλα τα υπόλοιπα σημεία και το $L_{1,V-\{1\}}$ είναι το μήκος της συντομότερης περιοδείας. Ο υπολογισμός των ενδιάμεσων μηκών γίνεται από τα μικρότερα στα μεγαλύτερα σύνολα. Συγκεκριμένα, για να υπολογίσουμε το L_{iS} για κάποιο σύνολο S με $k + 1$ σημεία, πρέπει να έχουμε ήδη υπολογίσει όλες τις τιμές $L_{j,S-\{j\}}$. Υπολογίζουμε λοιπόν τις τιμές L_{iS} ξεκινώντας από $S = \emptyset$ και συνεχίζοντας με σύνολα μεγέθους $1, 2, \ldots, n - 1$. Αυτός ο αλγόριθμος έχει χρόνο $O(n^2 2^n)$, και χρειάζεται $O(n2^n)$ θέσεις μνήμης για να υπολογίσει τη βέλτιστη περιοδεία.